Анализ данных и статистика в R

Иван Поздняков

06.04.2024

Содержание

Глава 1

О курсе

Здесь будут материалы для курса “Анализ данных и статистика в R”. Эта онлайн-книжка написана с помощью Quarto, здесь можно посмотреть код с материалами. Эта книга будет постоянно пополняться, поэтому следите за обновлениями!

Книгу можно использовать как самоучитель, хотя более эффективно ее использовать в качестве дополнительных материалов к соответствующему курсу. Если вы хотите, чтобы я прочитал этот курс в вашей лаборатории или компании, провел для вас консультацию по анализу данных, R и статистике, а так же по любым другим вопросам, то пишите мне на почту `ivanspozdniakov@gmail.com`, VK или в *Telegra*m:`@pozdniakovivan` (лучше писать в *Telegram).*

Эта книжка абсолютно бесплатна, ее можно скачивать, делиться ей с другими, я буду только рад этому!

Если вы хотите поддержать мою работу, то можете оформить подписку:

- на Патреоне (если вы находитесь за границей России)
- на Бусти (если вы находитесь в России)

Особое суперспасибо суперпатрону Даниилу Ульянову!

Часть I

Основы R и Rstudio

В этом разделе книги мы разберемся с основами программирования в R. Мы обучимся всему необходимому для работы в R, в том числе довольно продвинутым инструментам.

- В главе Глава 2 разбирается установка R и RStudio, как вводить команды и сохранять скрипты, создавать переменные, пользоваться операторами и функциями, а также типы данных в R.
- Глава Глава 3 полностью посвящена векторам: их созданию, индексированию и работе с пропущенными значениями (`NA`).
- В главе Глава 4 разобраны более сложные структуры: список, матрица, массив, датафрейм.
- В Глава 5 мы научимся устанавливать дополнительные пакеты и работать с ними.
- Глава Глава 6 полностью посвящена импорту и экспорту данных, а так же работе с рабочей директорией и проектами RStudio.
- В главе Глава 7 мы разберем структуры ветвления (if-else-else if) и их векторизованные аналоги. Кроме того, мы коротко коснемся работы с циклами `for` в R.
- Глава Глава 8 посвящена созданию пользовательских функций, анонимным функциям и функциям семейства `apply()`.

Глава 2

Введение в R

2.1 Установка R и Rstudio

Для работы с R необходимо его сначала скачать и установить.

- R
  - на Windows, найдите большую кнопку **Download R (номер версии) for Windows.**
  - на Mac, если маку меньше, чем 5 лет, то смело ставьте *.pkg файл с последней версией. Если старше, то поищите на той же странице версию для вашей системы.
  - на Linux, также можно добавить зеркало и установить из командной строки:

```
sudo apt-get install r-cran-base
```

В данной книге используется следующая версия R:

```
sessionInfo()$R.version$version.string
```

```
[1] "R version 4.2.2 (2022-10-31)"
```

После установки R необходимо скачать и установить *RStudio*:

- RStudio Desktop

Если вдруг что-то установить не получается (или же вы просто не хотите устанавливать на компьютер лишние программы), то можно работать в облаке, делая все то же самое в веб-браузере:

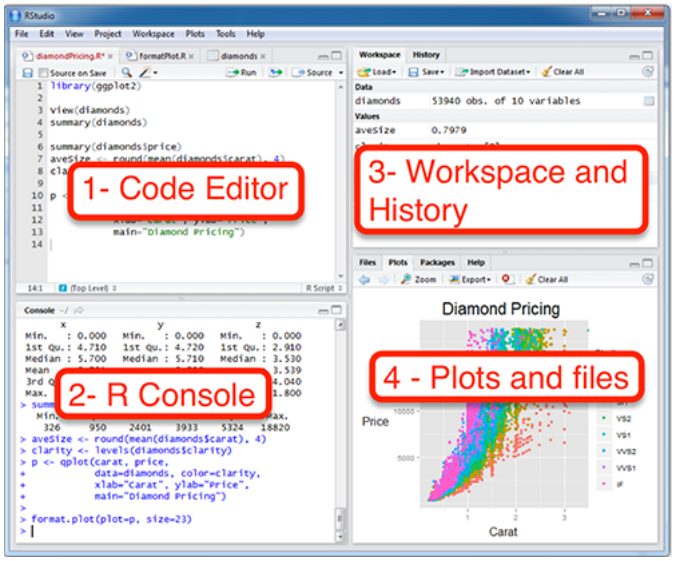
- Posit Cloud (ранее – RStudio Cloud)

Первый и вполне закономерный вопрос: зачем мы ставили R и отдельно еще какой-то *RStudio*? Если опустить незначительные детали, то R – это сам язык программирования, а *RStudio* – это **интегрированная среда разработки *(integrated development environment; IDE)***, которая позволяет в этом языке очень удобно работать.

> *RStudio* – это не единственная среда для R, но, определенно, самая удобная на сегодняшний день. Почти все пользуются именно ей и не стоит тратить время на поиск чего-то более удобного и лучшего. Если же вы привыкли работать с *Jupyter Notebook*, то в R обычно вместо него используются великолепный *R Markdown* или еще более великолепный *Quarto* – с помощью последнего и написан этот онлайн-учебник, кстати говоря. И с *R Markdown* и *Quarto* мы тоже будем разбираться (см. Глава 16)!

2.2 Знакомство с RStudio

Так, давайте взглянем на то, что нам тут открылось:



В первую очередь нас интересуют два окна: **1 – Code Editor** (окно для написания скриптов)[1] и **2 – R Console** (консоль). Здесь можно писать команды и запускать их. При этом работа в консоли и работа со скриптом немного различается.

В **2 – R Console** вы пишите команду и запускаете ее нажиманием `Enter`. Иногда после запуска команды появляется какой-то результат. Если нажимать стрелку вверх на клавиатуре, то можно выводить в консоль предыдущие команды. Это очень удобно для запуска предыдущих команд с небольшими изменениями.

В **1 – Code Editor** для запуска команды вы должны выделить ее и нажать `Ctrl` + `Enter` (`Cmd` + `Enter` на macOS). Если не нажать эту комбинацию клавиш, то команда не запустится. Можно выделить и запустить сразу несколько команд или даже все команды скрипта. Все команды скрипта можно выделить с помощью сочетания клавиш `Ctrl` + `A` на Windows и Linux, `Cmd` + `A` на macOS [2]. Как только вы запустите команду (или несколько команд), соответствующие строчки кода появятся в **2 – R Console**, как будто бы вы запускали их прямо там.

Обычно в консоли удобно что-то писать, чтобы быстро что-то посчитать. Скрипты удобнее при работе с длинными командами и как способ сохранения написанного кода для дальнейшей работы. Для сохранения скрипта нажмите `File – Save As...`. R

[1] При первом запуске RStudio вы не увидите это окно. Для того, чтобы оно появилось, нужно нажать `File – New File – R Script`.

[2] В RStudio есть много удобных сочетаний горячих клавиш. Чтобы посмотреть их все, нажмите `Help – Keyboard Shortcuts Help`.

скрипты сохраняются с разрешением *.R*, но по своей сути это просто текстовые файлы, которые можно открыть и модифицировать в любом текстовом редакторе а-ля “Блокнот”.

3 - Workspace and History – здесь можно увидеть переменные. Это поле будет автоматически обновляться по мере того, как Вы будете запускать строчки кода и создавать новые переменные. Еще там есть вкладка с историей всех команд, которые были запущены.

4 - Plots and files. Здесь есть очень много всего. Во-первых, небольшой файловый менеджер, во-вторых, там будут появляться графики, когда вы будете их рисовать. Там же есть вкладка с вашими пакетами (`Packages`) и `Help` по функциям. Но об этом потом.

2.3 R как калькулятор

R – полноценный язык программирования, который позволяет решать широкий спектр задач. Но в первую очередь R используется для анализа данных и статистических вычислений. Тем не менее, многими R до сих пор воспринимается как просто продвинутый калькулятор. Ну что ж, калькулятор, так калькулятор.

Давайте начнем с самого простого и попробуем использовать R как калькулятор с помощью арифметических операторов `+`, `-`, `*`, `/`, `^` (степень), `()` и т.д.

Просто запускайте в консоли пока не надоест:

```
40 + 2
```

```
[1] 42
```

```
3 - 2
```

```
[1] 1
```

```
5 * 6
```

```
[1] 30
```

```
99 / 9 #
```

```
[1] 11
```

```
2 ^ 3 #    , 2 ** 3
```

```
[1] 8
```

```
13 %/% 3 #
```

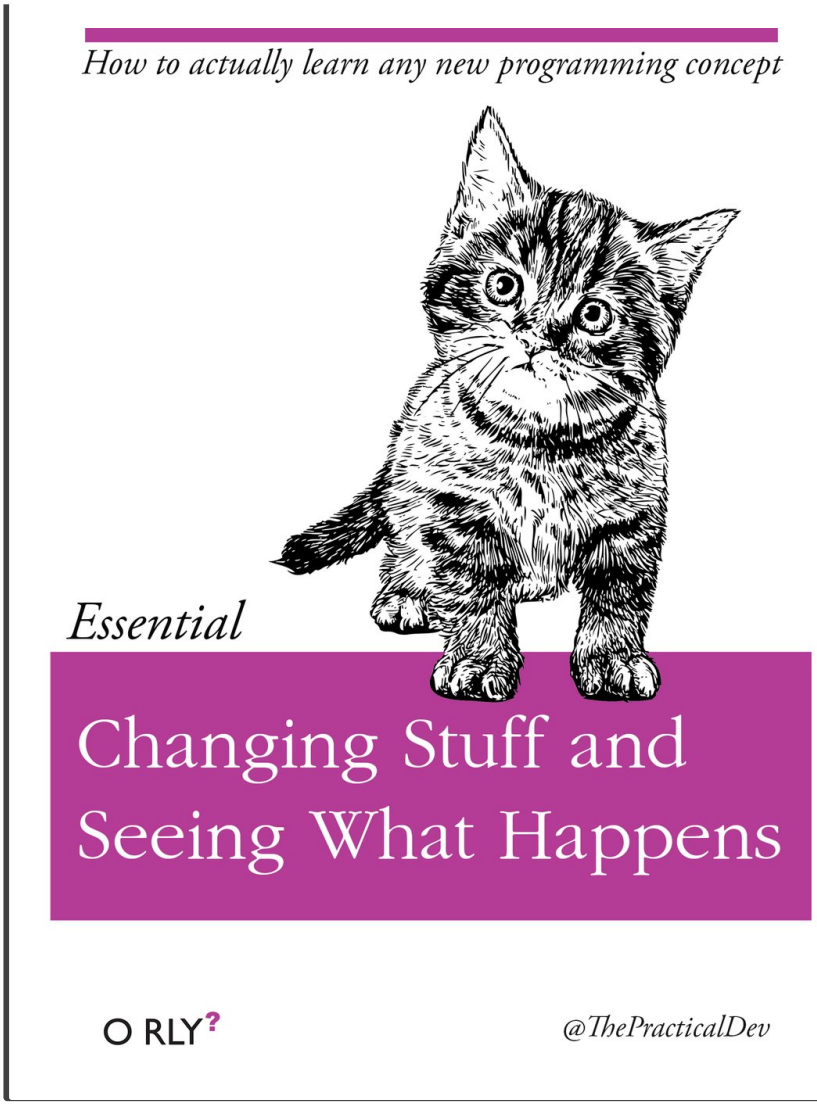
```
[1] 4
```

```
13 %% 3 #
```

```
[1] 1
```

Время мемов

Попробуйте самостоятельно посчитать что-нибудь с разными числами.



Ничего сложного, верно? Вводим выражение и получаем результат.

***Полезное:* комментарии**

Вы могли заметить, что некоторые команды у меня заканчиваются знаком решетки (#). Все, что написано в строчке после # игнорируется R при выполнении команды. Написанные команды в скрипте рекомендуется сопровождать комментариями, которые будут объяснять

вам же в будущем (или кому-то еще), что конкретно происходит в соответствующем куске кода [a]. Кроме того, комментарии можно использовать в тех случаях, когда вы хотите написать кусок кода по-другому, не стирая полностью предыдущий код: достаточно "закомментить" нужные строчки - поставить # в начало каждой строки, которую вы хотите переписать. Для этого есть специальное сочетание горячих клавиш: `Ctrl` + `Shift` + `C` (`Cmd` + `Shift` + `C` на macOS) – во всех выделенных строчках будет написан # в начале.

[a] Во время написания кода вам может казаться понятным то, что вы написали, но при возвращении к коду через некоторое время вы уже не будете этого помнить. Старайтесь писать комментарии как можно чаще!

Согласно данным навязчивых рекламных баннеров в интернете, только 14% россиян могут справиться с этим примером:

```
2 + 2 * 2
```

```
[1] 6
```

На самом деле, разные языки программирования ведут себя по-разному в таких ситуациях, поэтому ответ 6 (сначала умножаем, потом складываем) не так очевиден.

Порядок выполнения арифметических операций (т.е. приоритет операторов, *operator precedence*) в R как в математике, так что не забывайте про скобочки.

```
(2 + 2) * 2
```

```
[1] 8
```

Если Вы не уверены в том, какие операторы имеют приоритет, то используйте скобочки, чтобы точно обозначить, в каком порядке нужно производить операции. Или же смотрите на таблицу приоритета операторов с помощью команды `?Syntax`.

2.4 Функции

Давайте теперь извлечем корень из какого-нибудь числа. В принципе, тем, кто помнит школьный курс математики, возведения в степень вполне достаточно:

```
16 ^ 0.5
```

```
[1] 4
```

Ну а если нет, то можете воспользоваться специальной **функцией**: это обычно какие-то буквенные символы с круглыми скобками сразу после названия функции. Мы подаем на вход (внутрь скобочек) какие-то данные, внутри этих функций происходят какие-то вычисления, которые выдает в ответ какие-то другие данные (или же функция записывает файл, рисует график и т.д.).

Вот, например, функция для корня:

```
sqrt(16)
```

```
[1] 4
```

> **Осторожно!**
>
> R – case-sensitive язык, т.е. регистр важен. SQRT(16) не будет работать.

А вот так выглядит функция логарифма:

```
log(8)
```

```
[1] 2.079442
```

Так, вроде бы все нормально, но... Если Вы еще что-то помните из школьной математики, то должны понимать, что что-то здесь не так.

Здесь не хватает основания логарифма!

> Логарифм – показатель степени, в которую надо возвести число, называемое основанием, чтобы получить данное число.

То есть у логарифма 8 по основанию 2 будет значение 3:

$\log_2 8 = 3$

То есть если возвести 2 в степень 3 у нас будет 8:

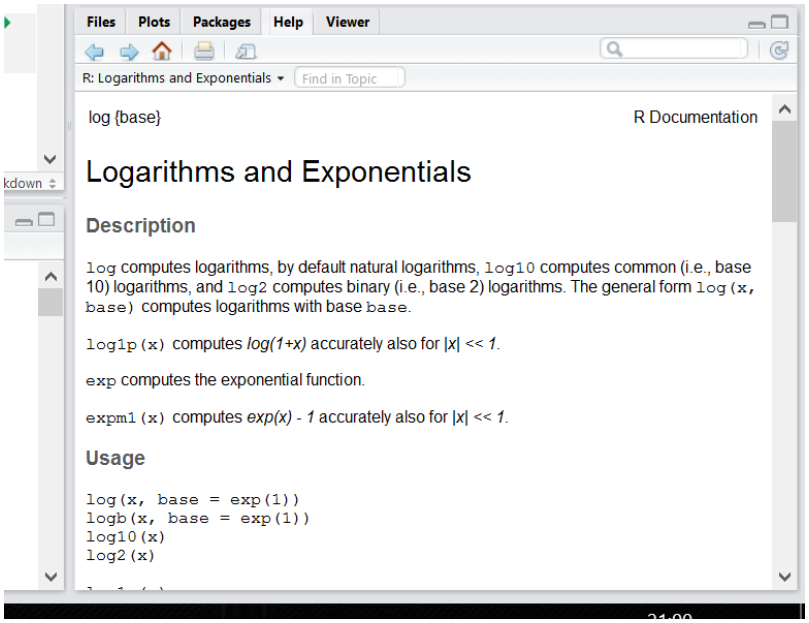
$2^3 = 8$

Только наша функция считает все как-то не так.

Чтобы понять, что происходит, нам нужно залезть в хэлп этой функции:

```
?log
```

Справа внизу в RStudio появится вот такое окно, в котором можно прочитать **документацию *(documentation)*** по выбранной функции:



> ***Полезное:* Документация в R**
>
> Документация в R есть для всех встроенных функций. Еще есть документация для встроенных наборов данных и других объектов, но большая часть документации касается именно функций. Документация всегда имеет одинаковую структуру, которая позволяет быстро понять, что функция требует на входе, что она делает и что возвращает. Подробнее про структуру документации можно почитать здесь.

В верхней части документации (*Usage*) функция прописана со всеми используемыми аргументами (arguments). Если через знак = для этих аргументов что-то прописано, то это означает, что эти параметры имеют значения по умолчанию.

Действительно, у функции `log()` есть еще аргумент `base =`. По умолчанию он равен числу Эйлера (2.7182818…), т.е. функция считает натуральный логарифм. В большинстве функций R есть какой-то основной аргумент – данные в том или ином формате, а есть и дополнительные аргументы, которые можно прописывать вручную, если значения по умолчанию вас не устраивают. Обычно эти дополнительные аргументы – это параметры функции, которые задают то, как именно функция работает.

```
log(x = 8, base = 2)
```

```
[1] 3
```

…или просто (если вы уверены в порядке переменных):

```
log(8, 2)
```

```
[1] 3
```

Более того, вы можете использовать результат выполнения одних функций в качестве аргумента для других:

```
log(8, sqrt(4))
```

```
[1] 3
```

Если эксплицитно писать имена аргументов, то их порядок в функции не важен:

```
log(base = 2, x = 8)
```

```
[1] 3
```

А еще можно писать имена аргументов не полностью, если они не совпадают с другими:

```
log(b = 2, x = 8)
```

```
[1] 3
```

Мы еще много раз будем возвращаться к функциям. Вообще, функции – это одна из важнейших штук в R (примерно так же как и в *Python*). Мы будем создавать свои функции, использовать функции как аргументы для функций и многое-многое другое. В R очень крутые возможности работы с функциями. Поэтому подружитесь с функциями, они клевые.

***Для продвинутых:* операторы – это тоже функции**

Арифметические знаки, которые мы использовали: +,−,/,^ и т.д. называются **операторами** и на самом деле тоже являются функциями:

```
`+`(3, 4)
```

```
[1] 7
```

Это не очень читаемая запись, конечно, поэтому использовать так обычно не рекомендуется: мы хотим, чтобы код был максимально понятным. Но ведь любопытно, правда?

2.5 В любой непонятной ситуации – гуглите

Если вдруг вы не знаете, что искать в хэлпе, или хэлпа попросту недостаточно, то… гуглите!

Время мемов

Нет ничего постыдного в том, чтобы гуглить решения проблем. Это абсолютно нормально. Используйте силу интернета во благо и да помогут вам *Stackoverflow*[3] и бесчисленные R-туториалы!

Время мемов

[3]Stackoverflow – это сайт с вопросами и ответами. Эдакий аналог *Quora*, *The Question*, ну или *Ответы Mail.ru* в мире программирования.

```
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    iLength = sInput.length();
    if (iLength < 4) {
        again = true;
        continue;
    } else if (sInput[iLength - 3] != '.') {
        again = true;
        continue;
    } while (++iN < iLength) {
        if (isdigit(sInput[iN])) {
            continue;
```

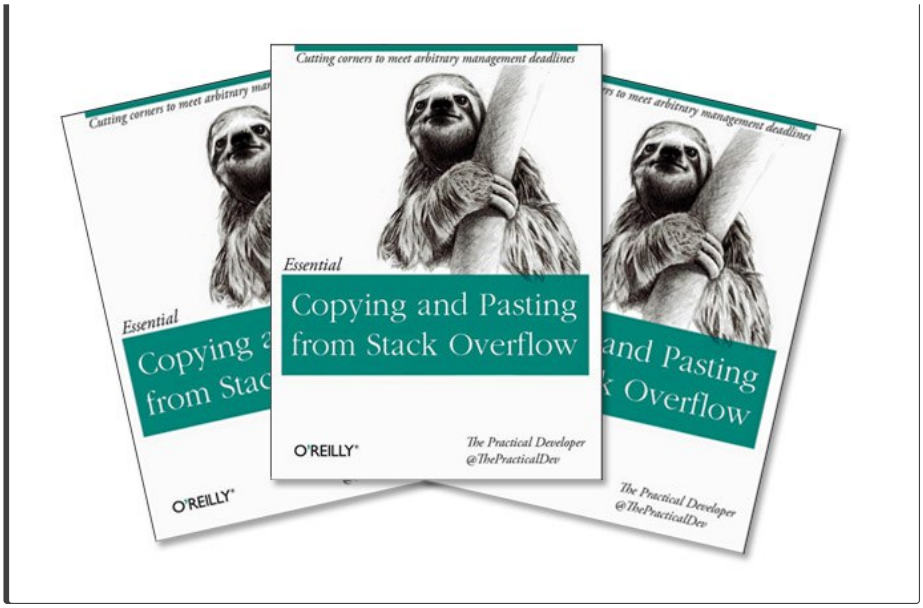
Computer Programming To Be Officially Renamed “Googling Stackoverflow”

Washington DC – The IEEE have produced a report today where they strongly recommend that from now on, the discipline of Computer Programming should be officially renamed to “Googling Stackoverflow”.

“We are recommending a root-and-branch name change to this discipline”, said President of the IEEE, Thomas M. Conte. “We are even going to change the official name of the IEEE Computer Society to the IEEE Quick Look At StackOverflow Society”.

“We are furthermore recommending that all universities across the planet should cease to award Bachelors degrees, Masters or PhDs in computer programming or software engineering and instead they should award these degrees in Googling Stackoverflow.”

“We are also considering renaming “P-values” to “Reviewer Pacifiers”.”



Главное, помните: загуглить работающий ответ всегда недостаточно. Надо понять, как и почему решение работает. Иначе что-то обязательно пойдет не так.

Кроме того, правильно загуглить проблему – не так уж и просто. Это отдельное искусство, которое необходимо освоить любому, кто занимается программированием: у меня быстро находить ответ на вопрос в интернете. Это не шутка!

Время мемов

Короче говоря:

> Гуглить – хорошо, бездумно копировать чужие решения – плохо.

2.6 Переменные

Важная штука в программировании на практически любом языке – возможность сохранять значения в **переменных *(variables)***. В R это обычно делается с помощью вот этих символов: <- (но можно использовать и обычное =, хотя это не очень принято). Для этого есть удобное сочетание клавиш: нажмите одновременно `Alt` + - (или `option` + - в macOS).

```
a <- 2
```

Полезное: присвоение и вывод в консоль

Заметьте, при присвоении результат вычисления не выводится в консоль! Если опустить детали, то обычно результат выполнения команды либо выводится в консоль (и не сохраняется), либо записывается в переменную (но тогда не показывается в консоли).

Чтобы вывести в консоль содержание существующей переменной, просто введите ее название:

```
a
```

```
[1] 2
```

Справа от <- находится значение, которое вы хотите сохранить, или же какое-то выражение, результат которого вы хотите сохранить в эту переменную[4]:

Слева от <- находится название будущей переменной. Название переменных может быть самым разным.

***Advanced:* синтаксически валидные имена для переменных**

Есть несколько ограничений для синтаксически валидных имен переменных: они должны включать в себя буквы, цифры, . или _, начинаться на букву (или точку, за которой не будет следовать цифра), не должны совпадать с коротким списком зарезервированных слов. Короче говоря, название не должно включать в себя пробелы и большинство других знаков.
Нельзя: - `new variable` - `_new_variable` - `.1var` - `v-r`
Можно: - `new_variable` - `.new.variable` - `var_2`
На самом деле, можно создавать переменные с невалидными именами. Такие названия нужно обрамлять бэктиками (‘):

```
`1      ...` <- 100
`1      ...`
```

```
[1] 100
```

Это же относится и к другим именам в R, например, к названиям колонок датафрейма (см. Глава 4.4), но таких имен лучше, конечно, избегать.

***Полезное:* понятные имена переменных**

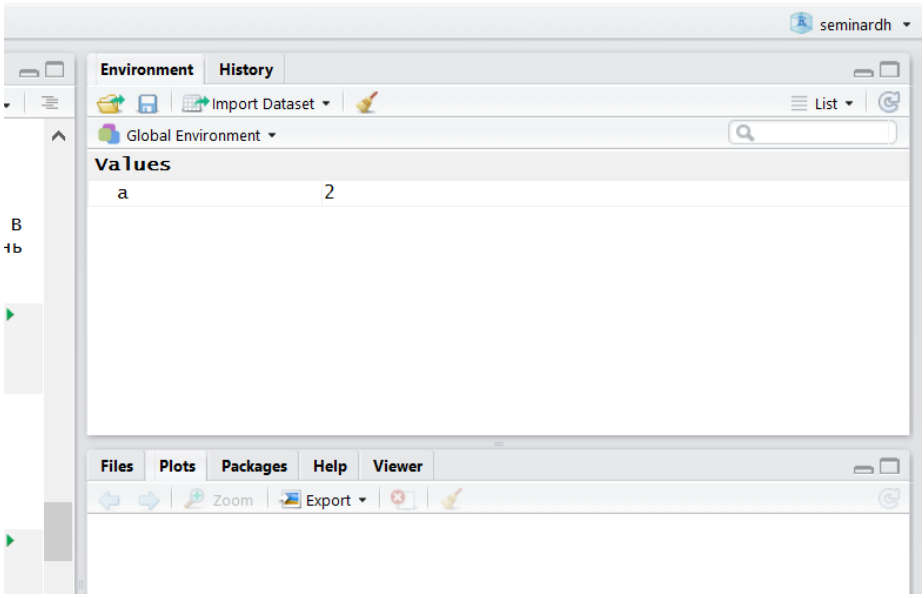
Обязательно делайте названия переменных осмысленными!

[4] Есть еще оператор ->, который позволяет присваивать значения слева направо, но так делать не рекомендуется, хотя это бывает довольно удобным.

Старайтесь делать при этом их понятными и короткими, это сохранит вам очень много времени, когда вы (или кто-то еще) будете пытаться разобраться в написанном ранее коде. Если название все-таки получается длинным и состоящим из нескольких слов, то лучше всего использовать нижнее подчеркивание в качестве разделителя: `some_variable`[a].

[a] Еще иногда используются большие буквы `SomeVariable`, но это плохо читается, а иногда – точка, но это тоже не рекомендуется.

После присвоения переменная появляется во вкладке **Environment** в RStudio:



Теперь использовать переменные в функциях и просто вычислениях:

```
b <- a ^ a + a * a
b
```

```
[1] 8
```

```
log(b, a)
```

```
[1] 3
```

Удалять переменные можно с помощью функции `rm()`:

```
e <- a ^ b
e
```

```
[1] 256
```

```
rm(e)
e #             ,
```

```
Error in eval(expr, envir, enclos): object 'e' not found
```

2.7 Логические операторы

Вы можете сравнивать разные переменные:

```
a == b
```

```
[1] FALSE
```

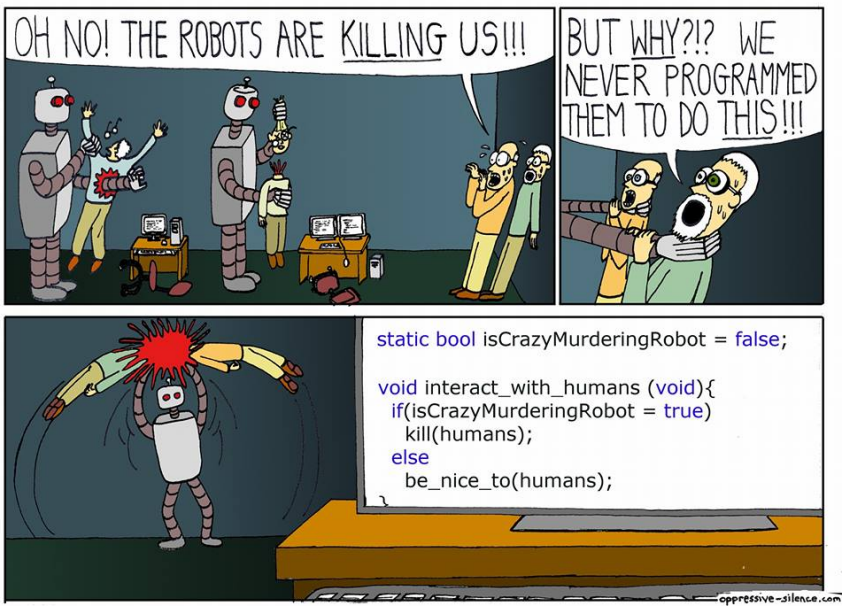
Заметьте, что сравнивая две переменные мы используем два знака равно ==, а не один =. Иначе это будет означать присвоение.

```
a = b
a
```

```
[1] 8
```

Время мемов

Теперь Вы сможете понять комикс про восстание роботов на следующей странице (пусть он и совсем про другой язык программирования)



Этот комикс объясняет, как важно не путать присваивание и сравнение *(хотя я иногда путаю до сих пор =( )*.

Иногда нам нужно проверить на ***не***равенство:

```
a <- 2
b <- 3

a == b
```

```
[1] FALSE
```

```
a != b
```

```
[1] TRUE
```

Восклицательный язык в программировании вообще и в R в частности стандартно означает отрицание.

Еще мы можем сравнивать на больше/меньше:

```
a > b
```

```
[1] FALSE
```

```
a < b
```

```
[1] TRUE
```

```
a >= b
```

```
[1] FALSE
```

```
a <= b
```

```
[1] TRUE
```

Этим мы будем пользоваться в дальнейшем регулярно! Именно на таких простых логических операциях построено большинство операций с данными.

2.8 Типы данных

2.8.1 Числовые типы

До этого момента мы работали только с ***числами (numeric).*** На самом деле, в R три типа ***numeric: integer* (целые), *double* (дробные), *complex* (комплексные числа)**[5]. R сам будет конвертировать числа в нужный числовой тип при необходимости, поэтому этим можно не заморачиваться за исключением редких случаев.

Если же все-таки нужно задать конкретный тип числа эксплицитно, то можно воспользоваться функциями `as.integer()`, `as.double()` и `as.complex()`.

> ***Для продвинутых:* числа не то, чем кажутся**
>
> Если число выглядит как целое, еще не факт, что это integer. Внутри оно может храниться как дробное, но, как я уже написал, это обычно не так важно. Но если очень нужно сделать именно integer, при создании числа можно поставить в конце `L`:

[5] **Комплексные числа в R пишутся так:** `complexnumber <- 2+2i`**.** `i` **здесь - это та самая мнимая единица, которая является квадратным корнем из -1.**

```
is.integer(5)
```

```
[1] FALSE
```

```
is.integer(5L)
```

```
[1] TRUE
```

***Для продвинутых:* ограничения дробных чисел**

Про *double* есть еще один маленький секрет. Дело в том, что дробные числа хранятся в R как числа с плавающей запятой двойной точности (отсюда и название – ***"double"*,** т.е. ***"double precision"*).** Да-да, компьютеры неточные! Дробные числа в компьютере могут быть записаны только с определенной степенью точности, поэтому иногда встречаются вот такие вот ситуации:

```
sqrt(2) ^ 2 == 2
```

```
[1] FALSE
```

Казалось бы, с обоих сторон – 2, почему же тогда `FALSE`? Дело в том, что в обоих случаях это дробное число, которое не может равняться ровно двум. Внутри это не два, а что-то вроде 2.0000….001 или 1.9999…999, число записано как степени двойки и отображено в десятичной системе. R не показывает все доступные цифры дробной части ради удобства, поэтому показывает округление, и мы видим просто число 2.
Поэтому при сравнении двух чисел происходит не сравнение их разницы с чистым круглым нулем, а сопоставление с очень маленьким числом – **машинной ошибкой** или **машинным эпсилоном *(machine epsilon).*** Если разница между двумя дробными числами меньше этого эпсилона, то числа считаются равными. Если выходят за пределы этой ошибки, то нет. Размер машинного эпсилона можно посмотреть с помощью `.Machine$double.eps`.
Обычно это все работает хорошо, но некоторые операции "выпрыгивают" за рамки этой ошибки, отсюда и такое "странное" поведение, которое иногда может возникать при сравнении двух чисел. Если хотите углубиться в этот вопрос, то можете почитать здесь.

Это довольно стандартная ситуация, характерная не только для R. Чтобы ее избежать, можно воспользоваться функцией `all.equal()`:

```
all.equal(sqrt(2) ^ 2, 2)
```

```
[1] TRUE
```

Еще один пример необычного поведения дробных чисел, связанный с их "неточностью", можно наблюдать в ситуациях, где вы ожидаете ноль, а получаете что-то такое:

```
sin(pi)
```

```
[1] 1.224647e-16
```

По правилам тригонометрии, $sin(\pi) = 0$, тогда откуда у нас это странное число с `e-16` в конце?
Это то, что называется *экспоненциальной записью (scientific notation)* числа, которую R использует автоматически, если нужно напечатать очень маленькое или очень большое число. Экспоненциальная запись часто используется как в компьютерах, так и во многих науках, так что читать ее полезно уметь.
Если перед нами очень маленькое число, то вместо того, чтобы писать многочисленные нули, можно записать его как $m \times 10^n$, где m – число от 1 до 10, а n – это отрицательная степень десяти: $10^{-1} = 0.1, 10^{-2} = 0.01, 10^{-3} = 0.001, \ldots, 10^{-16} = 0.0000000000000001$.
`-16` в конце числа – это и есть 10^{-16}, а `1.224647` – это множитель m, на который полученное число с очень большим количеством нулей умножается (точнее, только его первые несколько цифр). Получается, что этот множитель не сильно меняет погоды, поэтому когда видите число с `-`, нужно смотреть в первую очередь именно на степень. Вот так это число выглядит в родной для нас десятичной записи:

```
format(sin(pi), scientific = FALSE)
```

```
[1] "0.00000000000000001224647"
```

Эта маленькая дробная часть возникла из-за этой "неточности" хранения дробных чисел и самого числа π в компьютере, которую мы обсуждали выше.

Аналогично маленьким числам, R использует экспоненциальную запись, когда нужно напечатать очень большое число:

```
2 ^ 40
```

```
[1] 1.099512e+12
```

Обратите внимание, что здесь у нас `e+12`, а не `e-12`, что означает 10^{12}, а не 10^{-12}. То есть перед нами теперь 1.099512×10^{12}:

```
format(2 ^ 40, scientific = FALSE)
```

```
[1] "1099511627776"
```

Опять же, `1.099512` – это только первые цифры, в экспоненциальной записи R не показывает их все. Смотреть нужно сначала на степень (`e+12`), только потом – на множитель впереди.

2.8.2 Строковый тип

Строковые *(character)* данные – это набор букв, цифр и символов. Чтобы создать строковую переменную, нужные знаки обособляются кавычками.

```
s <- "      !"
s
```

```
[1] "      !"
```

```
class(s)
```

```
[1] "character"
```

Как и в *Python*, можно использовать как `"`, так и `'` (что удобно, когда строчка внутри уже содержит какие-то кавычки).

```
"Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn"
```

```
[1] "Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn"
```

Главное, открывать и закрывать кавычки одинаковыми кавычками (" или ')

Чтобы соединить несколько строковых переменных вместе, можно воспользоваться функцией `paste()`[6]:

```
paste("I", "love", "R")
```

```
[1] "I love R"
```

По умолчанию функция `paste()` соединяет строки пробелами, но разделитель можно настроить параметром `sep =`:

```
paste("I", "love", "R", sep = "_<3_")
```

```
[1] "I_<3_love_<3_R"
```

Чтобы соединять строки, так сказать, “без ничего”, нужно поставить `sep = ""` или же воспользоваться функцией `paste0()` специально для этого конкретного случая:

```
paste("I", "love", "R", sep = "")
```

```
[1] "IloveR"
```

```
paste0("I", "love", "R")
```

```
[1] "IloveR"
```

2.8.3 Логический тип

Логические (logical) данные – это просто `TRUE` или `FALSE`. То же самое, что и *boolean* тип в других языках программирования

[6] Оператор +, в отличие от Python, не работает для соединения строк.

```
t1 <- TRUE
f1 <- FALSE

t1
```

```
[1] TRUE
```

```
f1
```

```
[1] FALSE
```

Для продвинутых: **`T` и `F`**

Вообще, можно еще писать `T` и `F` (но не `True` и `False`!)

```
t1 <- T
f1 <- F
```

Это плохая практика: R защищает от перезаписи переменные `TRUE` и `FALSE`, но не защищает от этого `T` и `F`.

```
TRUE <- FALSE
```

```
Error in TRUE <- FALSE: invalid (do_set) left-hand side to assignment
```

```
TRUE
```

```
[1] TRUE
```

```
T <- FALSE
T
```

```
[1] FALSE
```

Теперь удалим эту переменную `T` от греха подальше иначе все перестанет работать при ее использовании:

```
rm(T)
```

Мы уже встречались с логическими значениями при сравнении двух числовых переменных. Теперь вы можете догадаться, что

результаты сравнения, например, числовых или строковых переменных, можно тоже сохранять в переменные!

```
comparison <- a == b
comparison
```

```
[1] FALSE
```

Это нам очень понадобится, когда мы будем работать с реальными данными: нам нужно будет постоянно вытаскивать какие-то данные из датасета, что как раз и построено на игре со сравнением переменных.

Чтобы этим хорошо уметь пользоваться, нам нужно еще освоить как работать с логическими операторами. Про один мы немного уже говорили – это логическое НЕ (`!`). `!` превращает `TRUE` в `FALSE`, а `FALSE` в `TRUE`:

```
t1
```

```
[1] TRUE
```

```
!t1
```

```
[1] FALSE
```

```
!!t1 #          !
```

```
[1] TRUE
```

Еще есть логическое И (выдаст `TRUE` только в том случае если обе переменные `TRUE`):

```
t1 & t1
```

```
[1] TRUE
```

```
t1 & f1
```

```
[1] FALSE
```

```
f1 & t1
```

```
[1] FALSE
```

```
f1 & f1
```

```
[1] FALSE
```

А еще логическое ИЛИ (выдаст `TRUE` в случае если хотя бы одна из переменных `TRUE`):

```
t1 | t1
```

```
[1] TRUE
```

```
t1 | f1
```

```
[1] TRUE
```

```
f1 | t1
```

```
[1] TRUE
```

```
f1 | f1
```

```
[1] FALSE
```

Обратите внимание: `t1 | t1`, то есть когда с обоих сторон `TRUE`, тоже возвращает `TRUE`!

> ***Для продвинутых:* строгое ЛИБО**
>
> Если кому-то вдруг понадобится другое ИЛИ (строгое ЛИБО) – есть функция `xor()`, принимающая два аргумента и возвращая `TRUE` только в том случае, если ровно один из двух аргументов равен `TRUE`. Но на практике она нужна очень редко, тогда как логические И и ИЛИ нужны очень часто:

> например, вам нужно отобрать только респондентов от 18 до 25, для этого нужно сделать два сравнения: что возраст больше 18 и что возраст меньше 25, после чего соединить два сравнения логическим И.

Итак, мы только что разобрались с самой занудной (хотя и важной) частью - с основными типа данных в R и как с ними работать[7]. Пора переходить к чему-то более интересному и специфическому для R. Вперед к ВЕКТОРАМ!

[7] Кроме описанных пяти типов данных (integer, double, complex, character и logical) есть еще и шестой – это raw, сырая последовательность байтов, но нам она не понадобится.

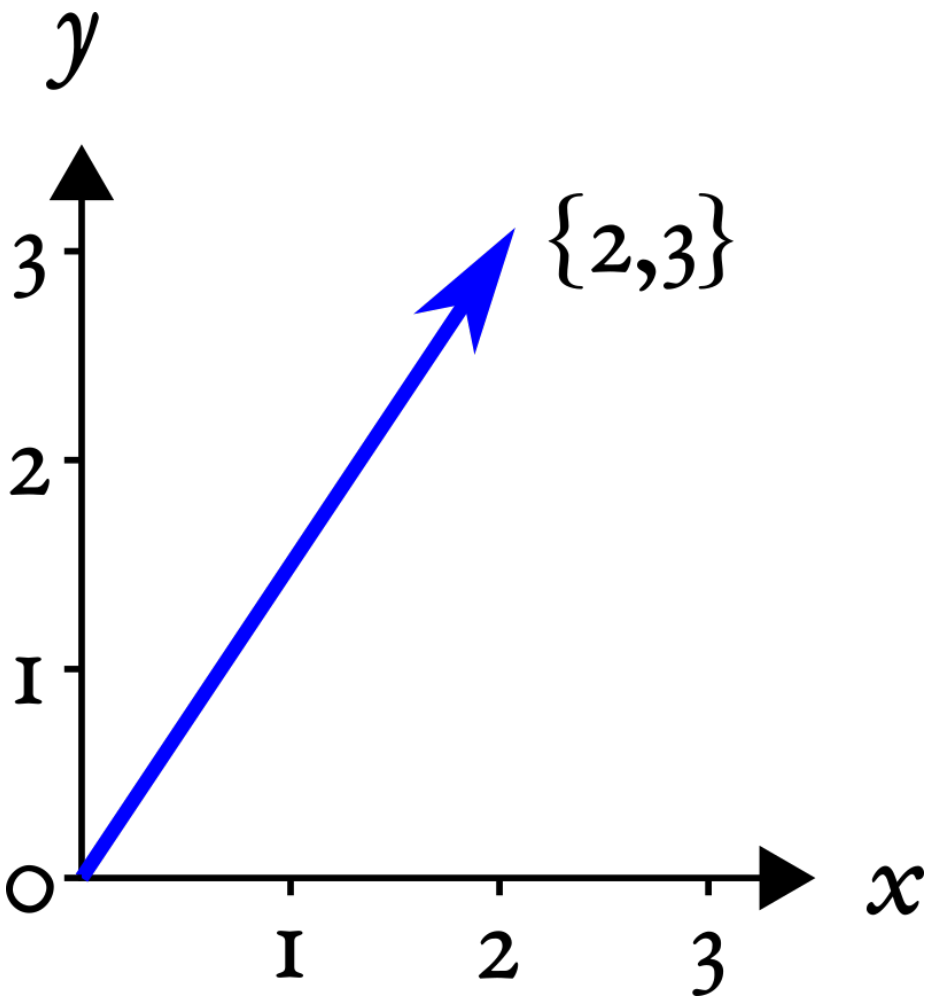
Глава 3

Вектор

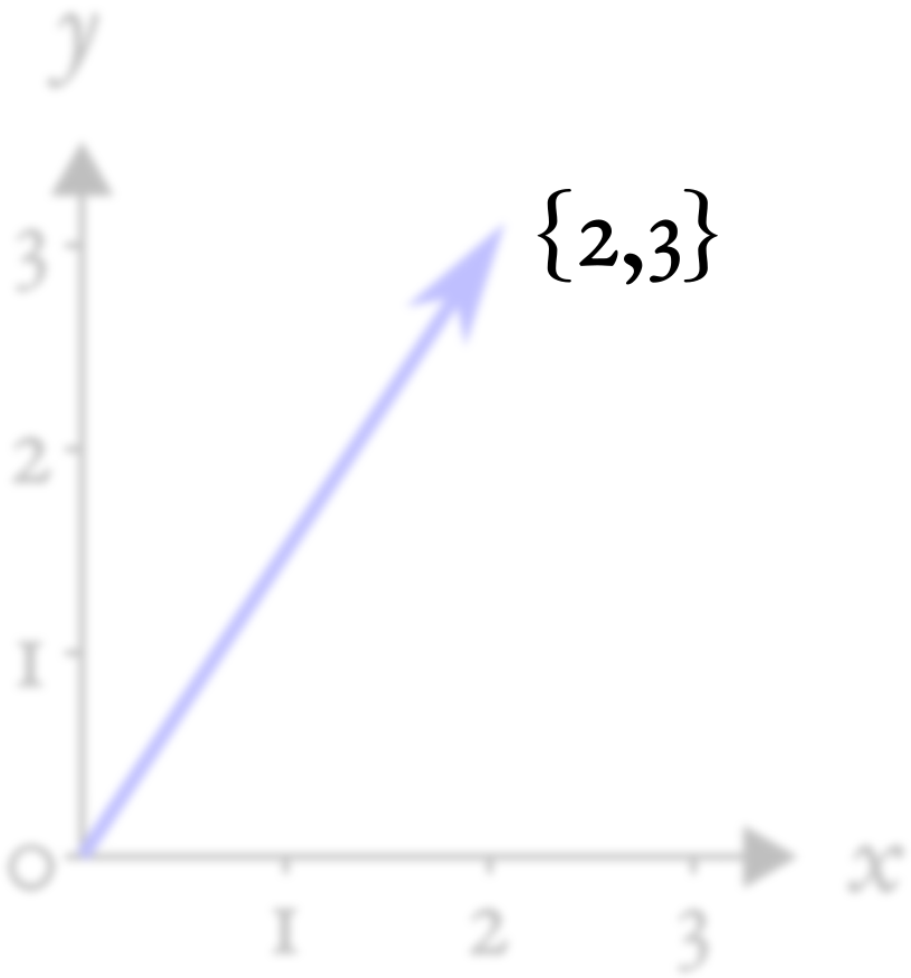
3.1 Понятие atomic вектора в R

Если у вас не было линейной алгебры (или у вас с ней было все плохо), то просто запомните, что **вектор** (***atomic vector*** или просто ***atomic***) – это набор (столбик) чисел в определенном порядке.

Если вы привыкли из школьного курса физики считать вектора стрелочками, то не спешите возмущаться и паниковать. Представьте стрелочки как точки из нуля координат {0,0} до какой-то точки на координатной плоскости, например, {2,3}:



Вот последние два числа и будем считать вектором. Попытайтесь теперь мысленно стереть координатную плоскость и выбросить стрелочки из головы, оставив только последовательность чисел {2,3}:



На самом деле, мы уже работали с векторами в R, но, возможно, вы об этом даже не догадывались. Дело в том, что в R нет как таковых скалярных (т.е. одиночных) значений, **есть вектора длиной 1**. Такие дела!

Чтобы создать вектор из нескольких значений, нужно воспользоваться функцией `c()`:

```
c(4, 8, 15, 16, 23, 42)
```

```
[1]  4  8 15 16 23 42
```

```
c("Hey", "Hey", "Ho")
```

```
[1] "Hey" "Hey" "Ho"
```

```
c(TRUE, FALSE)
```

```
[1]  TRUE FALSE
```

> ***Осторожно:* ошибка с кириллической “с”**
>
> Одна из самых мерзких и раздражающих причин ошибок в коде – это использование из кириллицы вместо `c` из латиницы. Видите разницу? И я не вижу. А R видит. И об этом сообщает:
>
> ```
> (3, 4, 5)
> ```
>
> ```
> Error in  (3, 4, 5): could not find function " "
> ```

Для создания числовых векторов есть удобный оператор `:`.

```
1:10
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
5:-3
```

```
[1]  5  4  3  2  1  0 -1 -2 -3
```

Этот оператор создает вектор от первого числа до второго с шагом 1. Вы не представляете, как часто эта штука нам пригодится… Если же нужно сделать вектор с другим шагом, то есть функция `seq()`:

```
seq(10, 100, by = 10)
```

```
 [1]  10  20  30  40  50  60  70  80  90 100
```

Кроме того, можно задавать не шаг, а длину вектора. Тогда функция `seq()` сама посчитает шаг:

```
seq(1, 13, length.out = 4)
```

```
[1]  1  5  9 13
```

Другая функция – `rep()` – позволяет создавать вектора с повторяющимися значениями. Первый аргумент – значение, которое нужно повторять, а второй аргумент – сколько раз повторять.

```
rep(1, 5)
```

```
[1] 1 1 1 1 1
```

И первый, и второй аргумент могут быть векторами! Если второй агрумент – вектор такой же длины, то каждое значение первого вектора будет повторено соответствующее количество раз из второго вектора.

```
rep(1:3, 3)
```

```
[1] 1 2 3 1 2 3 1 2 3
```

```
rep(1:3, c(10, 2, 30))
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[39] 3 3 3 3
```

Если нужно повторить каждое значение в векторе одно и то же количество раз, то можно воспользоваться дополнительным параметром `each =`:

```
rep(1:3, each = 5)
```

```
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

Еще можно объединять вектора (что мы, по сути, и делали, просто с векторами длиной 1):

```
v1 <- c("Hey", "Ho")
v2 <- c("Let's", "Go!")
c(v1, v2)
```

```
[1] "Hey"   "Ho"    "Let's" "Go!"
```

Очень многие функции в R работают именно с векторами. Например, функции `sum()` (считает сумму значений вектора) и `mean()` (считает среднее арифметическое всех значений в векторе):

```
sum(1:10)
```

```
[1] 55
```

```
mean(1:10)
```

```
[1] 5.5
```

3.2 Приведение типов

Что будет, если вы объедините два вектора с значениями разных типов? Ошибка?

Мы уже обсуждали, что в обычных векторах (*atomic* векторах) может быть только один тип данных. В некоторых языках программирования при операции с данными разных типов мы бы получили ошибку. А вот в R при несовпадении типов произойдет попытка привести типы к “общему знаменателю”, то есть конвертировать данные в более “широкий” тип (а иногда – более “узкий” тип, если того требует функция).

Например:

```
c(FALSE, 2)
```

```
[1] 0 2
```

`FALSE` превратился в `0` (а `TRUE` превратился бы в `1`), чтобы оба значения можно было объединить в вектор. То же самое произошло бы в случае операций с векторами:

```
2 + TRUE
```

```
[1] 3
```

Это называется **неявным приведением типов (implicit coercion)**.

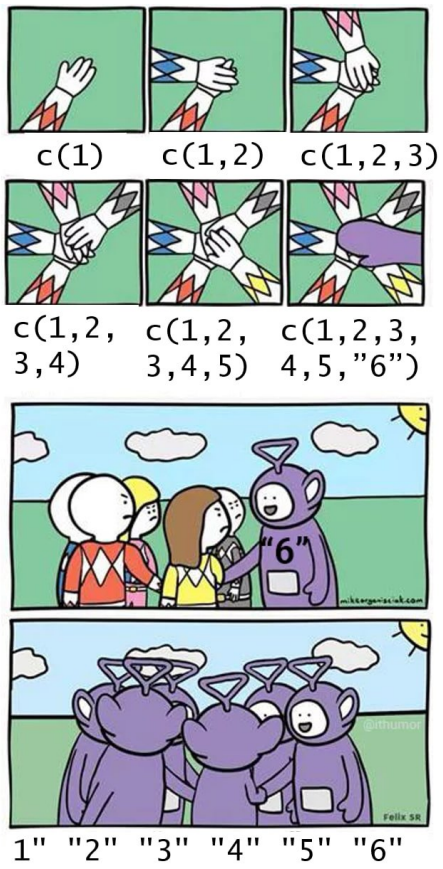
Вот более сложный пример:

```
c(TRUE, 3, "hi")
```

```
[1] "TRUE" "3"    "hi"
```

Здесь все значения были приведены сразу к строковому типу данных.

Время мемов



У R есть иерархия приведения типов:

`NULL < raw < logical < integer < double < complex < character < list < expression`.

Мы из этого списка еще многого не знаем, сейчас важно запомнить, что логические данные – `FALSE` и `TRUE` – превращаются в `0` и `1` соответственно, а `0` и `1` в строчки `"0"` и `"1"`.

Если Вы боитесь полагаться на приведение типов, то можете воспользоваться функциями `as.()` для явного приведения типов (**explicit coercion**):

```
as.numeric(c(TRUE, FALSE, FALSE))
```

```
[1] 1 0 0
```

```
as.character(as.numeric(c(TRUE, FALSE, FALSE)))
```

```
[1] "1" "0" "0"
```

Можно превращать и обратно, например, строковые значения в числовые. Если среди числа встретится буква или другой неподходящий знак, то мы получим предупреждение `NA` – пропущенное значение (мы очень скоро научимся с ними работать, см. Глава 1).

```
as.numeric(c("1", "2", "  "))
```

```
Warning: NAs introduced by coercion
```

```
[1]  1  2 NA
```

> ***Полезное:* подсчет количества и доли**
>
> Один из распространенных примеров использования неявного приведения типов – использования функций `sum()` и `mean()` для подсчета в логическом векторе количества и доли `TRUE` соответственно. Мы будем много раз пользоваться этим приемом в дальнейшем!

3.3 Векторизация

Все те арифметические операторы, что мы использовали ранее, можно использовать с векторами одинаковой длины:

```
n <- 1:4
m <- 4:1
n + m
```

```
[1] 5 5 5 5
```

```
n - m
```

```
[1] -3 -1  1  3
```

```
n * m
```

```
[1] 4 6 6 4
```

```
n / m
```

```
[1] 0.2500000 0.6666667 1.5000000 4.0000000
```

```
n ^ m + m * (n - m)
```

```
[1] -11   5  11   7
```

Если применить операторы на двух векторах одинаковой длины, то мы получим результат поэлементного применения оператора к двум векторам. Это называется **векторизацией** (**vectorization**).

> Если после какого-нибудь MATLAB Вы привыкли, что по умолчанию операторы работают по правилам линейной алгебры и `m * n` будет давать скалярное произведение (*dot product*), то снова нет. Для скалярного произведения нужно использовать операторы с `%` по краям:

```
n %*% m
```

```
     [,1]
[1,]   20
```

> Абсолютно так же и с операциями с матрицами в R, хотя про матрицы будет немного позже.

В принципе, большинство функций в R, которые работают с отдельными значениями, так же хорошо работают и с целыми векторами. Скажем, если вы хотите извлечь корень из нескольких чисел, то для этого не нужны никакие циклы (как это обычно делается во многих других языках программирования). Можно просто “скормить” вектор функции и получить результат применения функции к каждому элементу вектора:

```
sqrt(1:10)
```

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278
```

Таких векторизованных функций в R очень много. Многие из них написаны на более низкоуровневых языках программирования (C, C++, FORTRAN), за счет чего использование таких функций приводит не только к более элегантному, лаконичному, но и к более быстрому коду.

> Векторизация в R – это очень важная фишка, которая отличает этот язык программирования от многих других. Если вы уже имеете опыт программирования на другом языке, то вам во многих задачах захочется использовать циклы типа `for` и `while` @ref(for). Не спешите этого делать! В очень многих случаях циклы можно заменить векторизацией. Тем не менее, векторизация – это не единственный способ избавить от циклов типа `for` и `while` @ref(apply).

3.4 Ресайклинг

Допустим мы хотим совершить какую-нибудь операцию с двумя векторами. Как мы убедились, с этим обычно нет никаких проблем, если они совпадают по длине. А что если вектора не совпадают по длине? Ничего страшного! Здесь будет работать правило **ресайклинга** (*правило переписывания, recycling rule*). Это означает, что если мы делаем операцию на двух векторах разной длины, то если короткий вектор кратен по длине длинному, короткий вектор будет повторяться необходимое количество раз:

```
n <- 1:4
m <- 1:2
n * m
```

```
[1] 1 4 3 8
```

А что будет, если совершать операции с вектором и отдельным значением? Можно считать это частным случаем ресайклинга: короткий вектор длиной 1 будет повторятся столько раз, сколько нужно, чтобы он совпадал по длине с длинным:

```
n * 2
```

```
[1] 2 4 6 8
```

Если же меньший вектор не кратен большему (например, один из них длиной 3, а другой длиной 4), то R посчитает результат, но выдаст предупреждение.

```
n + c(3,4,5)
```

```
Warning in n + c(3, 4, 5): longer object length is not a multiple of shorter
object length
```

```
[1] 4 6 8 7
```

Проблема в том, что эти предупреждения могут в неожиданный момент стать причиной ошибок. Поэтому не стоит полагаться на ресайклинг некратных по длине векторов. А вот ресайклинг кратных по длине векторов – это очень удобная штука, которая используется очень часто.

3.5 Индексирование векторов

Итак, мы подошли к одному из самых сложных моментов. И одному из основных. От того, как хорошо вы научись с этим работать, зависит весь ваш дальнейший успех на R-поприще!

Речь пойдет об **индексировании** векторов. Задача, которую Вам придется решать каждые пять минут работы в R – как выбрать из вектора (или же списка, матрицы и датафрейма) какую-то его часть. Для этого используются квадратные скобочки `[]` (не круглые – они для функций!).

Самое простое – индексировать по номеру индекса, т.е. порядку значения в векторе.

```
n <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
n[1]
```

```
[1] 0
```

```
n[10]
```

```
[1] 34
```

> Если вы знакомы с другими языками программирования (не MATLAB, там все так же) и уже научились думать, что индексация с 0 – это очень удобно и очень правильно (ну или просто свыклись с этим), то в R вам придется переучиться обратно. Здесь первый индекс – это 1, а последний равен длине вектора – ее можно узнать с помощью функции `length()`. С обоих сторон индексы берутся включительно.

С помощью индексирования можно не только вытаскивать имеющиеся значения в векторе, но и присваивать им новые:

```
n[3] <- 20
n
```

```
 [1]  0  1 20  2  3  5  8 13 21 34
```

Конечно, можно использовать целые векторы для индексирования:

```
n[4:7]
```

```
[1] 2 3 5 8
```

```
n[10:1]
```

```
 [1] 34 21 13  8  5  3  2 20  1  0
```

```
n[4:6] <- 0
n
```

```
 [1]  0  1 20  0  0  0  8 13 21 34
```

Индексирование с минусом выдаст вам все значения вектора кроме выбранных:

```
n[-1]
```

```
[1]  1 20  0  0  0  8 13 21 34
```

```
n[c(-4, -5)]
```

```
[1]  0  1 20  0  8 13 21 34
```

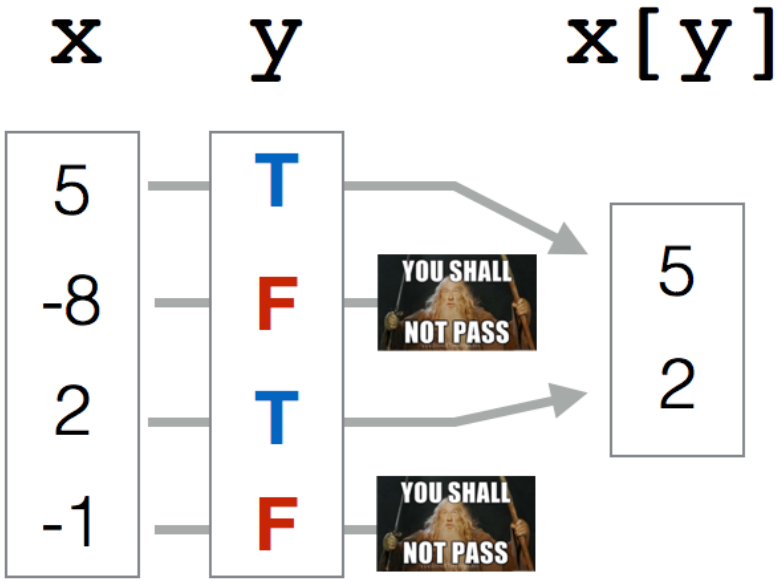
Минус здесь “выключает” выбранные значения из вектора, а не означает отсчет с конца как в Python.

Более того, можно использовать логический вектор для индексирования. В этом случае нужен логический вектор такой же длины:

```
n[c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE)]
```

```
[1]  0 20  0  8 21
```

Логический вектор работает здесь как фильтр: пропускает только те значения, где на соответствующей позиции в логическом векторе для индексирования содержится `TRUE`, и не пропускает те значения, где на соответствующей позиции в логическом векторе для индексирования содержится `FALSE`.



Ну а если эти два вектора (исходный вектор и логический вектор индексов) не равны по длине, то тут будет снова работать правило ресайклинга!

```
n[c(TRUE, FALSE)] #        - recycling rule!
```

```
[1]  0 20  0  8 21
```

Есть еще один способ индексирования векторов, но он несколько более редкий: индексирование по имени. Дело в том, что для значений векторов можно (но не обязательно) присваивать имена:

```
my_named_vector <- c(first = 1,
                     second = 2,
                     third = 3)
my_named_vector['first']
```

```
first
    1
```

А еще можно “вытаскивать” имена из вектора с помощью функции `names()` и присваивать таким образом новые имена.

```
d <- 1:4
names(d) <- letters[1:4]
names(d)
```

```
[1] "a" "b" "c" "d"
```

```
d["a"]
```

```
a
1
```

> `letters` – это “зашитая” в R константа – вектор букв от a до z. Иногда это очень удобно! Кроме того, есть константа `LETTERS` – то же самое, но заглавными буквами. А еще в R есть названия месяцев на английском и числовая константа `pi`.

Вернемся к нашему вектору `n` и посчитаем его среднее с помощью функции `mean()`:

```
mean(n)
```

```
[1] 9.7
```

А как вытащить все значения, которые больше среднего?

Сначала получим логический вектор – какие значения больше среднего:

```
larger <- n > mean(n)
larger
```

```
 [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

А теперь используем его для индексирования вектора `n`:

```
n[larger]
```

```
[1] 20 13 21 34
```

Можно все это сделать в одну строчку:

```
n[n > mean(n)]
```

```
[1] 20 13 21 34
```

Предыдущая строчка отражает то, что мы будем постоянно делать в R: вычленять (subset) из данных отдельные куски на основании разных условий.

3.6 Работа с логическими векторами

На работе с логическими векторами построено очень много удобных фишек, связанных со сравнением условий.

```
eyes <- c("green", "blue", "blue", "brown", "green", "blue")
```

3.6.1 `mean()` и `sum()` для подсчета пропорций и количества TRUE

Уже знакомая нам функция `sum()` позволяет посчитать количество `TRUE` в логическом векторе. Например, можно удобно посчитать сколько раз значение `"blue"` встречается в векторе `eyes`:

```
eyes == "blue"
```

```
[1] FALSE  TRUE  TRUE FALSE FALSE  TRUE
```

```
sum(eyes == "blue")
```

```
[1] 3
```

Функцию `mean()` можно использовать для подсчета пропорций `TRUE` в логическом векторе.

```
eyes == "blue"
```

```
[1] FALSE  TRUE  TRUE FALSE FALSE  TRUE
```

```
mean(eyes == "blue")
```

```
[1] 0.5
```

Умножив на 100, мы получим долю выраженную в процентах:

```
mean(eyes == "blue") * 100
```

```
[1] 50
```

3.6.2 `all()` и `any()`

Функция `all()` выдает `TRUE` только когда все значения логического вектора на входе равны `TRUE`:

```
all(eyes == "blue")
```

```
[1] FALSE
```

Функция `any()` выдает `TRUE` когда есть хотя бы одно значение `TRUE`:

```
any(eyes == "blue")
```

```
[1] TRUE
```

Вместе с оператором `!` можно получить много дополнительных вариантов. Например, есть ли хотя бы один `FALSE` в векторе?

```
any(!eyes == "blue")
```

```
[1] TRUE
```

```
!all(eyes == "blue")
```

```
[1] TRUE
```

Все ли значения в векторе равны `FALSE`?

```
all(!eyes == "blue")
```

```
[1] FALSE
```

```
!any(eyes == "blue")
```

```
[1] FALSE
```

3.6.3 Превращение логических значений в индексы: `which()`

Как вы уже знаете, и логические векторы, и числовые вектора с индексами могут использоваться для индексирования векторов. Иногда может понадобиться превратить логический вектор в вектор индексов. Для этого есть функция `which()`

```
which(eyes == "blue")
```

```
[1] 2 3 6
```

3.6.4 оператор %in% и match()

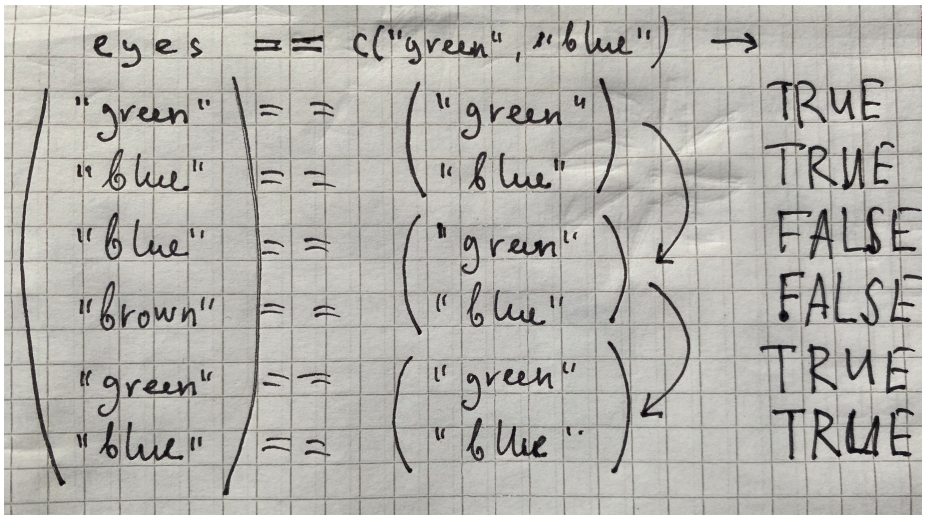
Часто возникает такая задача: нужно проверить вектор на равенство с хотя бы одним значением из другого вектора. Например, мы хотим вычленить всех зеленоглазых и голубоглазых. Может возникнуть идея сделать так:

```
eyes[eyes == c("green", "blue")]
```

```
[1] "green" "blue"  "green" "blue"
```

Перед нами самый страшный случай: результат *похож* на правильный, но не правильный! Попытайтесь самостоятельно понять почему этот ответ неверный и что произошло на самом деле.

А на самом деле мы просто сравнили два вектора, один из которых короче другого, следовательно, у нас сработало правило ресайклинга.



Как мы видим, это совсем не то, что нам нужно! В данной ситуации нам подойдет сравнение с двумя значениями вместе с логическим ИЛИ.

```
eyes[eyes == "green" | eyes == "blue"]
```

```
[1] "green" "blue"  "blue"  "green" "blue"
```

Однако это не очень удобно, особенно если значений больше 2. Тогда на помощь приходит оператор `%in%`, который выполняет именно то, что нам изначально нужно: выдает для каждого значения в векторе слева, есть ли это значение среди значений вектора справа.

```
eyes[eyes %in% c("green", "blue")]
```

```
[1] "green" "blue"  "blue"  "green" "blue"
```

Для продвинутых: `match()`

Основное преимущество оператора `%in%` в его простоте и понятности. У оператора `%in%` есть старший брат, более сложный и более мощный.
Функция `match()` работает похожим образом на `%in%`, но при совпадении значения в левом векторе с одним из значений в правом выдает индекс соответствующего значения вместо `TRUE`. Если же совпадений нет, то вместо `FALSE` функция `match()` выдает `NA` (что можно поменять параметром `nomatch =`).

```
match(eyes, c("green", "blue"))
```

```
[1]  1  2  2 NA  1  2
```

Зачем это может понадобиться? Во-первых, это способ соединить два набора данных (хотя для этого есть и более подходящие инструменты), во-вторых, так можно заменить все значения кроме выбранных заменить на `NA` (для чего тоже есть альтернативы).

```
c("green", "blue")[match(eyes, c("green", "blue"))]
```

```
[1] "green" "blue"  "blue"  NA      "green" "blue"
```

3.7 NA - пропущенные значения

В реальных данных у нас часто чего-то не хватает. Например, из-за технической ошибки или невнимательности не получилось записать какое-то измерение. Для обозначения пропущенных значений в R есть специальное значение `NA` (расшифровывается как *Not Available* - недоступное значение). `NA` – это не строка `"NA"`, не `0`, не пустая строка `""` и не `FALSE`. `NA` – это `NA`. Большинство операций с векторами, содержащими `NA` будут выдавать `NA`:

```
missed <- NA
missed == "NA"
```

```
[1] NA
```

```
missed == ""
```

```
[1] NA
```

```
missed == NA
```

```
[1] NA
```

Заметьте, даже сравнение `NA` с `NA` выдает `NA`. Это может прозвучать абсурдно: ну как же так, и то `NA`, и другое `NA` – это же одно и то же, они должны быть равны! Не совсем: `NA` – это отсутствие информации об объекте, неопределенность, неизвестная нам величина. Если мы не знаем двух значений (т.е. имеем два `NA`), то это еще не значит, что они равны.

Иногда наличие `NA` в данных очень бесит:

```
n[5] <- NA
n
```

```
 [1]  0  1 20  0 NA  0  8 13 21 34
```

```
mean(n)
```

```
[1] NA
```

Получается, что наличие NA “заражает” неопределенностью все последующие действия. Что же делать?

Наверное, надо сравнить вектор с NA и исключить этих пакостников. Давайте попробуем:

```
n == NA
```

```
 [1] NA NA NA NA NA NA NA NA NA NA
```

Ах да, мы ведь только что узнали, что даже сравнение NA с NA приводит к NA! Сначала это может показаться нелогичным: ведь с обоих сторон NA, почему же тогда результат их сравнения – это тоже NA, а не TRUE?

Дело в том, что сравнивая две неопределенности, вы не можете установить между ними знак равенства. Представим себе двух супергероев: Бэтмена и Спайдермена. Допустим, мы не знаем их рост:

```
Batman <- NA
Spiderman <- NA
```

Одинаковый ли у них рост?

```
Batman == Spiderman
```

```
[1] NA
```

Мы не знаем! Возможно, да, возможно, и нет. Поэтому у нас здесь остается неопределенность.

Так как же избавиться от NA в данных? Самый простой способ – это функция `is.na()`:

```
is.na(n)
```

```
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Результат выполнения `is.na(n)` выдает FALSE на тех позициях, где у нас числа (или другие значения), и TRUE там, где у нас NA. Чтобы вычленить из вектора `n` все значения кроме NA нам нужно, чтобы было наоборот: TRUE, если это не NA, FALSE, если это NA. Здесь нам понадобится логический оператор НЕ `!` (мы его уже встречали – см. Глава 2.8.3), который инвертирует логические значения:

```
n[!is.na(n)]
```

```
[1]  0  1 20  0  0  8 13 21 34
```

Ура, мы можем считать среднее без NA!

```
mean(n[!is.na(n)])
```

```
[1] 10.77778
```

Теперь Вы понимаете, зачем нужно отрицание (!)

Полезное: **na.rm = TRUE**

Вообще, есть еще один способ посчитать среднее, если есть NA. Для этого надо залезть в хэлп по функции *mean()*:

```
?mean()
```

В хэлпе мы найдем параметр na.rm =, который по умолчанию FALSE. Вы знаете, что нужно делать!

```
mean(n, na.rm = TRUE)
```

```
[1] 10.77778
```

NA может появляться в векторах разных типов. На самом деле, NA - это специальное значение в логических векторах, тогда как в векторах других типов NA появляется как NA_integer_, NA_real_, NA_complex_ или NA_character_, но R обычно сам все переводит в нужный формат и показывает как просто NA. Таким образом, NA в векторах разных типов – это разные NA, хотя на практике эта деталь обычно несущественна.

Для продвинутых: **NA против NaN**

Кроме NA есть еще NaN – это разные вещи. NaN расшифровывается как *Not a Number* и получается в результате таких операций как 0 / 0. Тем не менее, функция is.na() выдает TRUE на NaN, а вот функция is.nan() выдает TRUE на NaN и FALSE на NA:

```
  is.na(NA)
```

```
[1] TRUE
```

```
  is.na(NaN)
```

```
[1] TRUE
```

```
  is.nan(NA)
```

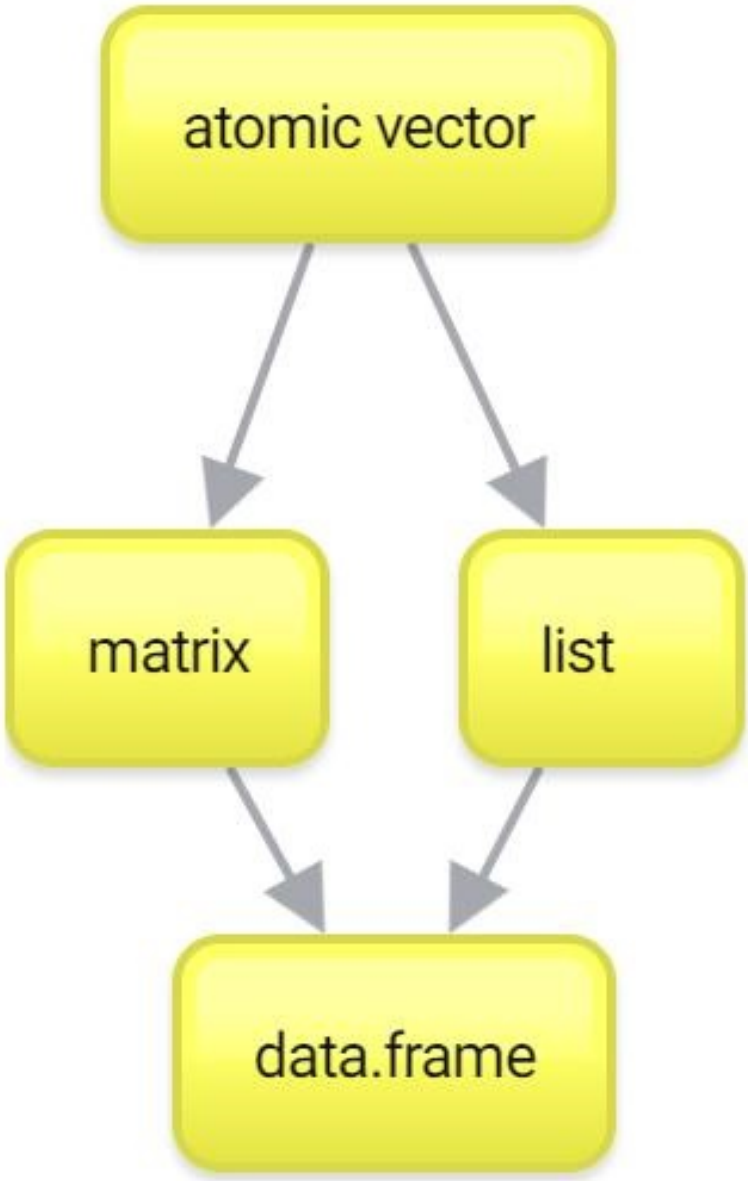
```
[1] FALSE
```

```
  is.nan(NaN)
```

```
[1] TRUE
```

3.8 Заключение

Итак, с векторами мы более-менее разобрались. Помните, что вектора – это один из краеугольных камней вашей работы в R. Если вы хорошо с ними разобрались, то дальше все будет довольно несложно. Тем не менее, вектора – это не все. Есть еще два важных типа данных: **списки *(list)*** и **матрицы *(matrix)***. Их можно рассматривать как своеобразное “расширение” векторов, каждый в свою сторону. Ну а списки и матрицы нужны чтобы понять основной тип данных в R – **датафрейм *(dataframe)***.


atomic vector
matrix
list
data.frame

Глава 4

Сложные структуры данных в R

Давайте повторим то, что мы знаем про вектор в R:

- Вектор – это последовательность из значений.
- Порядок значений имеет значение, но этот порядок одномерный.
- Внутри вектора могут быть данные только одного типа

Как вы уже поняли, вектор – это одно из важнейших понятий в R, и он нам будет встречаться дальше постоянно. Обычно работа с данными – это именно работа с векторами, различные операции на векторах.

Однако иногда в понятии вектора нам уже становится несколько тесно. Поэтому нам нужно выйти за рамки его ограничений. Во-первых, во второе (и дальнейшие) измерения – это делает **матрица (matrix)**. Во-вторых, нам нужна структура, которая могла бы содержать данные разных типов – это **список (list).**

4.1 Матрица

Если вдруг вас пугает это слово, то совершенно зря. **Матрица (matrix)** – это всего лишь “двумерный” вектор: вектор, у которого есть не только длина, но и ширина. Создать матрицу можно с помощью функции `matrix()` из вектора, указав при этом количество строк и столбцов.

```
A <- matrix(1:20, nrow = 5, ncol = 4)
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
```

> ***Полезное:* порядок заполнения матрицы**
>
> Заметьте, значения вектора заполняются следующим образом: сначала заполняется первый столбик сверху вниз, потом второй сверху вниз и так до конца, т.е. заполнение значений матрицы идет в первую очередь по вертикали. Это довольно стандартный способ создания матриц, характерный не только для R.

Если мы знаем сколько значений в матрице и сколько мы хотим строк, то количество столбцов указывать необязательно:

```
A <- matrix(1:12, nrow = 4)
A
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Все остальное так же как и с векторами: внутри находится данные только одного типа. Поскольку матрица – это уже двумерный массив, то у него имеется два индекса. Эти два индекса разделяются запятыми.

```
A[2, 3]
```

```
[1] 10
```

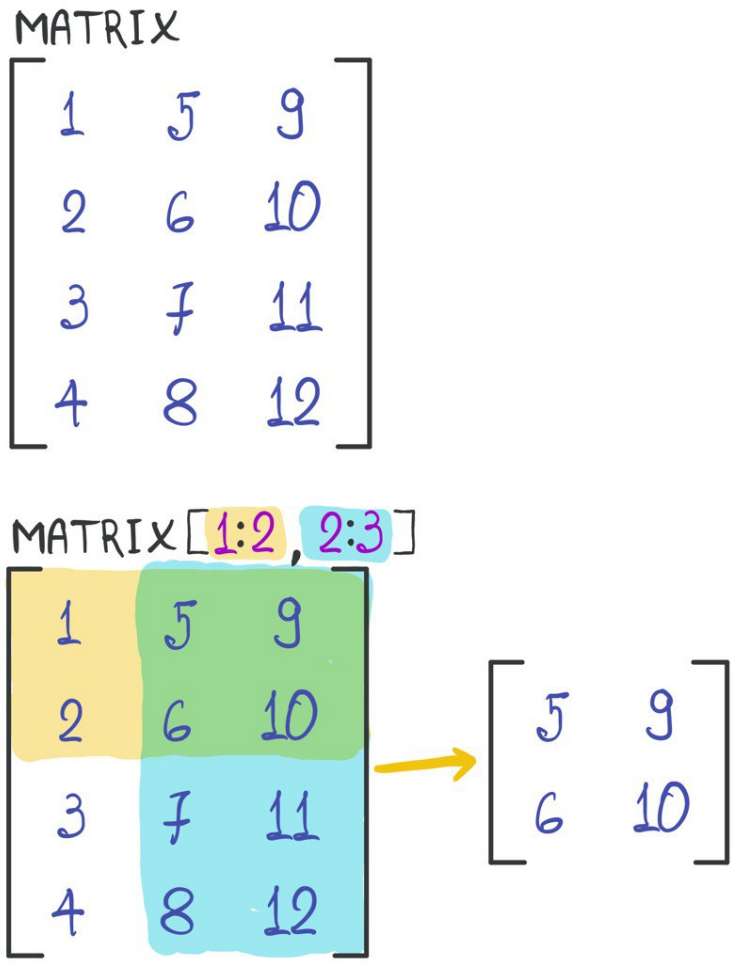
Первый индекс – выбор строк, второй индекс – выбор колонок[1]. Результат – пересечение выбранных строк и столбцов.

Так же как и с векторами, матрицы можно индексировать числовыми векторами:

```
A[1:2, 2:3]
```

```
     [,1] [,2]
[1,]    5    9
[2,]    6   10
```

[1]Это универсальный порядок: что в других языках программирования, что в линейной алгебре первый индекс – выбор строчек, второй индекс – выбор столбцов.



И даже логическими матрицами (матрицы имеют такие же типы, как и вектора):

```
A[A > 10]
```

```
[1] 11 12
```

В этом случае матрица упростится до вектора.

Если же мы оставляем пустое поле вместо числа, то мы выбираем все строки/колонки в зависимости от того, оставили мы поле пустым до или после запятой:

```
A[, 2:3]
```

```
     [,1] [,2]
[1,]    5    9
[2,]    6   10
[3,]    7   11
[4,]    8   12
```

```
A[1:2, ]
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
```

```
A[, ]
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Так же как и в случае с обычными векторами, часть матрицы можно переписать:

```
A[1:2, 1:2] <- 100
A
```

```
     [,1] [,2] [,3]
[1,]  100  100    9
[2,]  100  100   10
[3,]    3    7   11
[4,]    4    8   12
```

В принципе, это все, что нам нужно знать о матрицах. Матрицы используются в R довольно редко, особенно по сравнению, например, с MATLAB. Но вот индексировать матрицы хорошо бы уметь: это понадобится в работе с датафреймами (см. Глава 4.4).

> ***Для продвинутых:* матрица как вектор**
>
> То, что матрица – это просто двумерный вектор, не является метафорой: в R матрица – это по сути своей вектор с дополнительными *атрибутами* `dim` и (опционально) `dimnames`. Атрибуты – это свойства объектов, своего рода "метаданные". Для всех объектов есть обязательные атрибуты типа и длины и могут быть любые необязательные атрибуты. Можно задавать свои атрибуты или удалять уже присвоенные: удаление атрибута `dim` у матрицы превратит ее в обычный вектор. Про атрибуты подробнее можно почитать здесь или на стр. 99-101 книги "R in a Nutshell" (Adler 2010).

4.2 Массив

Два измерения – это не предел! Структура с одним типом данных внутри, но с тремя измерениями или больше, называется **массивом (array)**. Создание массива очень похоже на создание матрицы: задаем вектор, из которого будет собран массив, и размерность массива.

```
array_3d <- array(1:12, c(3, 2, 2))
array_3d
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

4.3 Список

Теперь представим себе вектор без ограничения на одинаковые данные внутри. И получим **список (list)**!

```
simple_list <- list(42, "    ", TRUE)
simple_list
```

```
[[1]]
[1] 42

[[2]]
[1] "    "

[[3]]
[1] TRUE
```

А это значит, что там могут содержаться самые разные данные, в том числе и другие списки, векторы и матрицы (и другие объекты, которые нам еще не знакомы)!

```
complex_list <- list(c("Wow", "this", "list", "is", "so", "big"), "16", simple_list, A)
complex_list
```

```
[[1]]
[1] "Wow"  "this" "list" "is"   "so"   "big"

[[2]]
[1] "16"

[[3]]
[[3]][[1]]
[1] 42

[[3]][[2]]
[1] "    "

[[3]][[3]]
[1] TRUE


[[4]]
     [,1] [,2] [,3]
```

```
[1,]  100  100    9
[2,]  100  100   10
[3,]    3    7   11
[4,]    4    8   12
```

Если у нас сложный список, то есть очень классная функция `str()`, чтобы посмотреть, как он устроен:

```
str(complex_list)
```

```
List of 4
 $ : chr [1:6] "Wow" "this" "list" "is" ...
 $ : chr "16"
 $ :List of 3
  ..$ : num 42
  ..$ : chr "    "
  ..$ : logi TRUE
 $ : num [1:4, 1:3] 100 100 3 4 100 100 7 8 9 10 ...
```

> Представьте, что список - это такое дерево с ветвистой структурой. А на конце этих ветвей - листья-векторы.

Как и в случае с векторами мы можем давать имена элементам списка:

```
named_list <- list(name = "Veronika", age = 26, student = FALSE)
named_list
```

```
$name
[1] "Veronika"

$age
[1] 26

$student
[1] FALSE
```

К списку можно обращаться как с помощью индексов, так и по именам. Начнем с последнего:

```
named_list$age
```

```
[1] 26
```

А вот с индексами сложнее, и в этом очень легко запутаться. Давайте попробуем сделать так, как мы делали это раньше:

```
named_list[1]
```

```
$name
[1] "Veronika"
```

Мы, по сути, получили элемент списка – просто как часть списка, т.е. как список длиной один:

```
class(named_list)
```

```
[1] "list"
```

```
class(named_list[1])
```

```
[1] "list"
```

А вот чтобы добраться до самого элемента списка (и сделать с ним что-то хорошее), нам нужна не одна, а две квадратных скобочки:

```
named_list[[1]]
```

```
[1] "Veronika"
```

```
class(named_list[[1]])
```

```
[1] "character"
```

Как и в случае с вектором, к элементу списка можно обращаться по имени. Здесь тоже будет иметь значение, одинарные или двойные квадратные скобки вы используете:

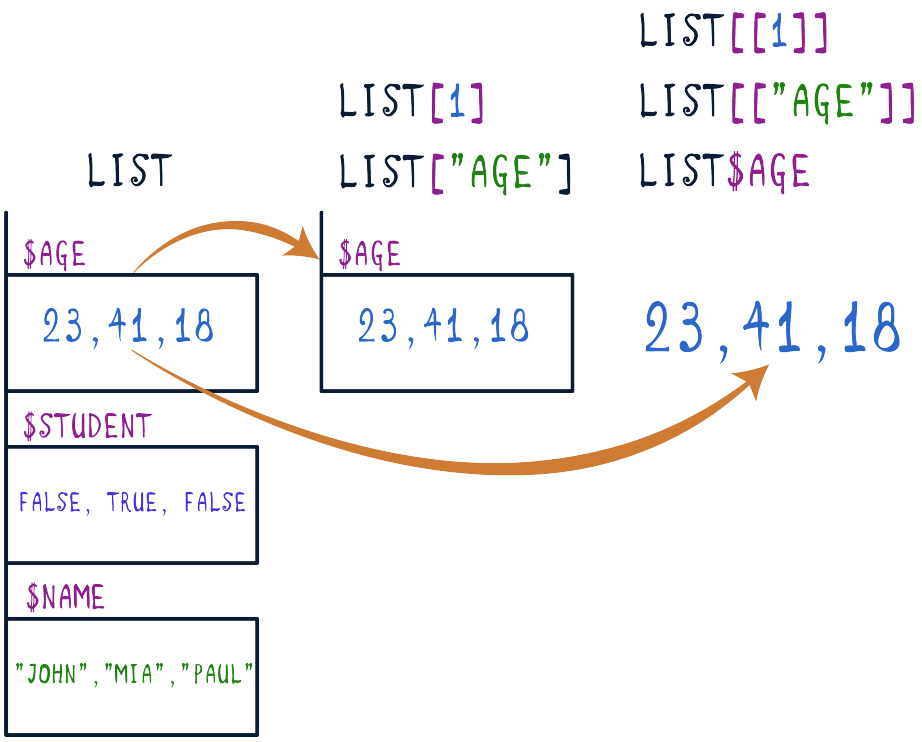
```
named_list["age"]
```

```
$age
[1] 26
```

```
named_list[["age"]]
```

```
[1] 26
```

Хотя последнее – практически то же самое, что и использование знака $.



> ***Полезное:* зачем нужны списки**
>
> Списки довольно часто используются в R, но реже, чем в *Python*. Со многими объектами в R, такими как результаты статистических тестов, удобно работать именно как со списками – к ним все вышеописанное применимо. Кроме того, некоторые данные мы изначально получаем в виде древообразной структуры – хочешь не хочешь, а придется работать с этим как со списком. Но обычно после этого стоит как можно скорее превратить список в датафрейм.

4.4 Датафрейм

Итак, мы перешли к самому главному. Самому-самому. **Датафреймы (dataframes)**. Более того, сейчас станет понятно, зачем нам нужно было разбираться со всеми предыдущими темами.

Без векторов мы не смогли бы разобраться с матрицами и списками. А без последних мы не сможем понять, что такое датафрейм.

Представьте себе, что мы хотим записать различную информацию о нескольких респондентах. Мы могли бы записать это в список из векторов.

```
list(name =  c("Veronika", "Eugeny", "Lena", "Misha", "Sasha"),
     age = c(26, 34, 23, 27, 26),
     student = c(FALSE, FALSE, TRUE, TRUE, TRUE))
```

```
$name
[1] "Veronika" "Eugeny"   "Lena"     "Misha"    "Sasha"

$age
[1] 26 34 23 27 26

$student
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

Датафрейм очень похож на список. Просто поменяем в команде выше list() на data.frame() и посмотрим, что изменится:

```
df <- data.frame(name =  c("Veronika", "Eugeny", "Lena", "Misha", "Sasha"),
                 age = c(26, 34, 23, 27, 26),
                 student = c(FALSE, FALSE, TRUE, TRUE, TRUE))
str(df)
```

```
'data.frame':   5 obs. of  3 variables:
 $ name   : chr  "Veronika" "Eugeny" "Lena" "Misha" ...
 $ age    : num  26 34 23 27 26
 $ student: logi  FALSE FALSE TRUE TRUE TRUE
```

```
df
```

```
      name age student
1 Veronika  26   FALSE
2   Eugeny  34   FALSE
3     Lena  23    TRUE
4    Misha  27    TRUE
5    Sasha  26    TRUE
```

Вообще, очень похоже на список, не правда ли? Так и есть, датафрейм – это что-то вроде проименованного списка, каждый элемент которого является *atomic* вектором фиксированной длины. Скорее всего, вы представляли список "горизонтально". Если это так, то теперь "переверните" список у себя в голове на 90 градусов. Так, чтобы названия векторов оказались сверху, а элементы списка стали столбцами.

Поскольку длина всех этих векторов одинаковая (обязательное условие!), то данные представляют собой табличку, похожую на матрицу. Но в отличие от матрицы, разные столбцы могут иметь разные типы данных. В нашем случае первая колонка – `character`, вторая колонка – `numeric`, третья колонка – `logical`. Тем не менее, обращаться с датафреймом можно и как с проименованным списком, и как с матрицей:

```
df$age
```

```
[1] 26 34 23 27 26
```

Здесь мы сначала извлекли колонку `age` с помощью оператора `$`. Результатом этой операции является числовой вектор. Колонки датафрейма – это и есть векторы!

```
df$age[2:3]
```

```
[1] 34 23
```

Теперь с ним можно работать как с обычным вектором: мы вытащили кусок, выбрав индексы 2 и 3.

Используя оператор `$` и присваивание можно создавать новые колонки датафрейма:

```
df$lovesR <- TRUE #     recycling -    ?     ?
df
```

```
      name age student lovesR
1 Veronika  26   FALSE   TRUE
2   Eugeny  34   FALSE   TRUE
3     Lena  23    TRUE   TRUE
4    Misha  27    TRUE   TRUE
5    Sasha  26    TRUE   TRUE
```

Ну а можно просто обращаться с помощью двух индексов через запятую, как мы это делали с матрицей:

```
df[3:5, 2:3]
```

```
  age student
3  23    TRUE
4  27    TRUE
5  26    TRUE
```

Как и с матрицами, первый индекс означает строчки, а второй – столбцы.

А еще можно использовать названия колонок внутри квадратных скобок:

```
df[1:2, "age"]
```

```
[1] 26 34
```

```
df[1:2, c("age", "name")]
```

```
  age     name
1  26 Veronika
2  34   Eugeny
```

И здесь перед нами открываются невообразимые возможности! Узнаем, любят ли R те, кто моложе среднего возраста в группе:

```
df[df$age < mean(df$age), 4]
```

```
[1] TRUE TRUE TRUE TRUE
```

Обратите внимание, как удобно нам здесь пригодилось то, что мы научились делать с векторами (Глава 3). Сначала мы посчитали среднее значение абсолютно так же, как мы делали это с векторами:

```
mean(df$age)
```

```
[1] 27.2
```

Полученное среднее поэлементно сравнили с каждым значением колонки (т.е. вектора) `df$age`:

```
df$age < mean(df$age)
```

```
[1]  TRUE FALSE  TRUE  TRUE  TRUE
```

Мы получили логический вектор, длина которого совпадает с длиной датафрейма. При этом `TRUE` стоит на тех позициях, где в соответствующей строчке в датафрейме возраст респондента больше среднего, а `FALSE` – в остальных случаях. Теперь этот логический вектор мы используем для выбора строк в исходном датафрейме:

```
df[df$age < mean(df$age), ]
```

```
      name age student lovesR
1 Veronika  26   FALSE   TRUE
3     Lena  23    TRUE   TRUE
4    Misha  27    TRUE   TRUE
5    Sasha  26    TRUE   TRUE
```

Наконец, тут же мы можем вытащить нужные колонки, по номеру колонки или ее названию:

```
df[df$age < mean(df$age), 4]
```

```
[1] TRUE TRUE TRUE TRUE
```

Эту же задачу можно выполнить другими способами:

```
df$lovesR[df$age < mean(df$age)]
```

```
[1] TRUE TRUE TRUE TRUE
```

```
df[df$age < mean(df$age), 'lovesR']
```

```
[1] TRUE TRUE TRUE TRUE
```

В большинстве случаев подходят сразу несколько способов – тем не менее, стоит овладеть ими всеми. Чем богаче ваш арсенал инструментов работы в R, тем легче вам обрабатывать свои данные: возможность сделать одно и то же действие добавляет вам гибкости, потому что разные способы будут более или менее подходящими в разных ситуациях.

Датафреймы удобно просматривать в RStudio. Для это нужно написать команду `View(df)` или же просто нажать на названии нужной переменной из списка вверху справа (там где Environment). Тогда увидите табличку, очень похожую на Excel и тому подобные программы для работы с таблицами. Там же есть и всякие возможности для фильтрации, сортировки и поиска [2].

	name	age	student	lovesR
1	Ivan	26	FALSE	TRUE
2	Eugeny	34	FALSE	TRUE
3	Lena	23	TRUE	TRUE
4	Misha	27	TRUE	TRUE
5	Sasha	26	TRUE	TRUE

Но, конечно, интереснее все эти вещи делать руками, т.е. с помощью написания кода.

Датафреймы – это структура, которая будет встречаться вам чаще всего при работе с данными в R. С одной стороны,

[2] Все, что вы нажмете в этом окошке, никак не повлияет на исходную переменную. Так что можете смело использовать эти функции для исследования содержимого датафрейма.

кажется, что она все равно довольно ограниченная: в каждой колонке должно быть одинаковое количество значений, внутри колонки только один тип данных. Но именно так обычно и представлены наши данные. Например, если вы загрузите результаты опроса Google Forms в виде таблицы, то каждая строчка будет респондентом, а каждая колонка – ответом на какой-то вопрос. Поэтому количество значений в каждой колонке будет одинаковым (хотя значения могут быть пропущенными), а каждая колонка – имеет свой тип. Например, год рождения – и это должна быть числовая колонка, с которой вы сможете делать все, что вы умеете делать с числовыми колонками. Например, посчитать возраст. Если в колонке с годом рождения оказалось что-то кроме чисел, то это повод для исследования данных.

4.5 Атрибуты и классы

...

4.6 Формулы

Формулы – это специальный класс в R, который используется в первую очередь для статистических моделей.

Выглядит формула следующим образом:

```
y ~ x
```

```
y ~ x
```

```
class(y ~ x)
```

```
[1] "formula"
```

Как видите, здесь нет никаких кавычек, т.е. это не строковое значение, а отдельный класс. В каждой формуле должна быть тильда (~), которая имеет смысл знака равно (=) в уравнениях (например, для уравнений линейной регрессии). Слева от ~ обычно находится зависимая переменная, а справа – независимые (предикторы).

Кроме статистических моделей, формулы используются много где как служебная конструкция, чтобы задать соответствующие пары значений. Вот несколько примеров:

- `case_when()` для задания множественных условий (см. Глава 7.3),
- визуализация ящиков с усами в базовом R (см. Глава 13),
- статистические тесты для сравнения двух (см. Глава 18) и более (см. Глава 22) групп,
- задание фасеток в {ggplot2} (см. Глава 14).

Глава 5

Пакеты в R

5.1 Дополнительные пакеты

R — очень богатый язык с широкими возможностями. Однако очень скоро мы поймем, что этих возможностей нам не хватает. Эти возможности нам могут предоставить дополнительные **пакеты (packages)**.

В большинстве случаев основным содержанием пакетов является набор дополнительных функций. Кроме функций, пакеты могут содержать наборы данных и новые структуры данных.

Обычно пакеты посвящены решению какого-то класса задач в определенной области. Например, есть множество пакетов для создания какого-то одного типа визуализации. Еще один пример — пакет `beepr`, который содержит всего две функции: `beep()` и `beep_on_error()` для воспроизведения звукового сигнала. Это может быть удобно, если ваш скрипт работает долго, но вы хотите получить уведомление, когда его выполнение завершится.

Более крупные пакеты посвящены целому классу задач. Например, пакеты `stringi` и `stringr` посвящены работе со строками, значительно расширяя и делая более удобной работу со строковыми данными в R. Еще один пример: пакет `igraph` для работы с графами (сетями). Этот пакет предоставляет дополнительный класс данных `igraph` для хранения и работы с сетями.

Есть и совсем крупные пакеты, которые значительно расширяют базовый функционал R, изменяя основные принципы работы в нем. Это пакеты `data.table` и `tidyverse`. Это настолько крупные пакеты, что их даже называют отдельными диалектами R,

потому что код, написанный с использованием этих пакетов, довольно сильно отличается от базового R. Кроме того, *tidyverse* - это не просто пакет, а целая экосистема пакетов, который взаимодополняют друг друга, но для удобства их можно устанавливать и загружать как один пакет `tidyverse`. Еще один пример крупной экосистемы из пакетов — это пакеты `mlr3` и `tidymodels` для машинного обучения, который представляет собой большой расширяемый “пакет пакетов”, где отдельные пакеты посвящены отдельным этапам и задачам машинного обучения.

5.2 Встроенные пакеты R

Вообще, даже сам R является набором из нескольких пакетов: основного `base` и нескольких других, таких как `stats`, `utils`, `graphics`. Вот их полный список:

```
rownames(installed.packages(priority = "base"))
```

```
 [1] "base"      "compiler"  "datasets"  "graphics"  "grDevices" "grid"
 [7] "methods"   "parallel"  "splines"   "stats"     "stats4"    "tcltk"
[13] "tools"     "utils"
```

Чтобы пользоваться этими пакетами ничего дополнительно делать не нужно.

5.3 Установка пакетов с CRAN

Функция `install.packages()` позволяет скачивать пакеты с Comprehensive R Archive Network (CRAN). На репозитории *CRAN* собрано более 19000 пакетов (число постоянно меняется, как в большую, так и меньшую сторону). Каждый из этих пакетов проходит проверку перед попаданием в *CRAN*: он должен быть хорошо задокументирован, стабильно работать и решать какую-то задачу.

Для примера установим пакет `remotes`. Это пакет для удобной установки пакетов не с *CRAN* и скоро нам понадобится.

```
install.packages("remotes")
```

При установке вы увидите много непонятных надписей красным (или черным) шрифтом. Не пугайтесь, это нормально, происходит скачивание и установка пакетов. Скорее всего, если нигде нет слова *Error*, то пакет успешно установился.

Иногда установка бывает очень долгой, потому что большие пакеты склонны иметь много **зависимостей**: для работы какого-то пакета может понадобиться другие пакеты, а для тех пакетов - еще какие-то пакеты. Таким образом, устанавливая какой-нибудь современный пакет, вы, возможно, установите десятки других пакетов! Зато если они понадобятся сами по себе, то их уже не нужно будет устанавливать.

5.4 Загрузка установленного пакета

Установить пакет с помощью `install.packages()` недостаточно, пакет нужно еще загрузить. Для этого есть функция `library()`.

```
library("remotes")
```

В отличие от `install.packages()`, функция `library()` принимает название пакета и как строчку в кавычках, и как название без кавычек.

```
library(remotes)
```

Теперь функции, данные и классы из пакета доступны для работы.

> ***Полезное:* аналогия с *Python***
>
> Если сравнивать с *Python*, то `install.packages()` – это аналог установки библиотек, например, с помощью *pip*, а `library()` – это аналог `import` (например, `import pandas as pd`).

5.5 Вызов функции из пакета с помощью `::`

Если пакетом нужно воспользоваться всего один-два раза, то имеет смысл не подключать весь пакет, а загрузить отдельную функцию из него. Для этого есть специальный оператор `::`, который использует функцию (указанную справа от `::`) из

выбранного пакета (указанного слева от ::), не загружая пакет полностью.

Для примера воспользуемся функцией `package_deps()` из только что установленного пакета `remotes`, которая возвращает все зависимости пакета:

```
remotes::package_deps("tidyverse")
```

```
Needs update -----------------------------
 package     installed available is_cran remote
 tidyselect  1.2.0     1.2.1     TRUE    CRAN
 curl        5.2.0     5.2.1     TRUE    CRAN
 ps          1.7.5     1.7.6     TRUE    CRAN
 digest      0.6.31    0.6.35    TRUE    CRAN
 sass        0.4.5     0.4.9     TRUE    CRAN
 cachem      1.0.6     1.0.8     TRUE    CRAN
 tinytex     0.44      0.50      TRUE    CRAN
 htmltools   0.5.4     0.5.8.1   TRUE    CRAN
 fontawesome 0.5.0     0.5.2     TRUE    CRAN
 bslib       0.4.2     0.7.0     TRUE    CRAN
 yaml        2.3.7     2.3.8     TRUE    CRAN
 xfun        0.37      0.43      TRUE    CRAN
 evaluate    0.21      0.23      TRUE    CRAN
 processx    3.8.1     3.8.4     TRUE    CRAN
 rstudioapi  0.15.0    0.16.0    TRUE    CRAN
 rmarkdown   2.20      2.26      TRUE    CRAN
 knitr       1.42      1.45      TRUE    CRAN
 fs          1.6.2     1.6.3     TRUE    CRAN
 callr       3.7.3     3.7.6     TRUE    CRAN
 prettyunits 1.1.1     1.2.0     TRUE    CRAN
 rematch     1.0.1     2.0.0     TRUE    CRAN
 progress    1.2.2     1.2.3     TRUE    CRAN
 vroom       1.6.1     1.6.5     TRUE    CRAN
 textshaping 0.3.6     0.3.7     TRUE    CRAN
 systemfonts 1.0.4     1.0.6     TRUE    CRAN
 tidyr       1.3.0     1.3.1     TRUE    CRAN
 timechange  0.2.0     0.3.0     TRUE    CRAN
 readr       2.1.4     2.1.5     TRUE    CRAN
 uuid        1.1-0     1.2-0     TRUE    CRAN
 munsell     0.5.0     0.5.1     TRUE    CRAN
 labeling    0.4.2     0.4.3     TRUE    CRAN
 scales      1.2.1     1.3.0     TRUE    CRAN
 gtable      0.3.3     0.3.4     TRUE    CRAN
 data.table  1.14.8    1.15.4    TRUE    CRAN
```

```
DBI          1.1.3     1.2.2     TRUE    CRAN
blob         1.2.3     1.2.4     TRUE    CRAN
rvest        1.0.3     1.0.4     TRUE    CRAN
reprex       2.0.2     2.1.0     TRUE    CRAN
ragg         1.2.6     1.3.0     TRUE    CRAN
haven        2.5.3     2.5.4     TRUE    CRAN
ggplot2      3.4.4     3.5.0     TRUE    CRAN
dbplyr       2.4.0     2.5.0     TRUE    CRAN
```

В дальнейшем использование оператора :: будет иногда использоваться, чтобы указать, из какого пакета взята функция.

Оператор :: полезен еще и в тех случаях, когда в разных пакетах присутствуют функции с одинаковым названием. Например, у основного пакета tidyverse, `dplyr`, есть функция `filter()`. Функция с точно таким же названием есть в базовом R в пакете `stats`, в котором та выполняет совершенно другую задачу. Если у вас уже загружен `dplyr`, то использование :: укажет на то, что вы хотите воспользоваться именно функцией `filter()` из пакета `stats`:

```
stats::filter(1:20, rep(1,3))
```

```
Time Series:
Start = 1
End = 20
Frequency = 1
 [1] NA  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 NA
```

Подобные путаницы могут возникнуть, если у вас загружено много пакетов, поэтому старайтесь не загружать слишком много пакетов, а если есть функции с одинаковым названием, то обязательно используйте оператор ::. Иначе слишком велик риск загрузить пакеты не в том порядке и получить из-за этого ошибку или некорректный результат.

> ***Полезное:* оператор :: для указания используемого пакета**
>
> В дальнейшем я буду иногда использовать ::, чтобы было понятно, из какого пакета взята та или иная функция

Выгрузить ненужный пакет можно с помощью функции `detach()`.

```
detach(package:remotes)
```

5.6 Установка пакетов с Bioconductor

У биологов есть свой большой репозиторий, который является альтернативой *CRAN*, — *Bioconductor*. С него можно скачать множество специализированных пакетов для работы с биологическими данными.

Для установки пакетов с Bioconductor сначала нужно скачать пакет `BiocManager` с CRAN.

```
install.packages("BiocManager")
```

Теперь можно воспользоваться функцией `install()` из пакета `BiocManager` для установки пакета `flowCore` — пакета для анализа данных проточной цитометрии.

```
BiocManager::install("flowCore")
```

5.7 Установка пакетов с Github

Некоторых пакетов нет ни на *CRAN,* ни на *Bioconductor.* Обычно это касается пакетов, разработчики которых по каким-либо причинам решили не проходить проверки или не прошли проверки на строгие требования *CRAN.* Иногда бывает, что пакет был удален с *CRAN* (например, автор давно не занимается им) или же версия пакета на *CRAN* отстает от последней, а именно в ней реализованы так нужные вам функции. В некоторых случаях пакета может не быть на *CRAN*, потому что его разработчики активно занимаются его развитием и постоянно переделывают уже имеющийся функционал, добавляя новые возможности и удаляя старые. Это нужно делать с осторожностью, когда пакет уже выложен на *CRAN,* потому что если функции новой версии пакета будут работать по-другому, то это может вызвать массу проблем.

Во всех этих случаях пакет обычно можно скачать с репозитория *Github.* Для этого нам понадобится уже установленный (с *CRAN,* разумеется) пакет `remotes`[1].

[1] пакет `remotes` “откололся” от более старого пакета `devtools`, а многие функции из `remote` просто скопированы из `devtools`. Разработчики `devtools`/`remotes` рекомендуют использовать для установки пакетов именно более легковесный `remotes`, но во многих случаях вы увидите код с `devtools::install_github()`. Оба варианта будут работать.

```
remotes::install_github("dracor-org/rdracor")
```

Теперь установленный пакет осталось загрузить, после чего им можно пользоваться.

```
library(rdracor)
godunov <- get_net_cooccur_igraph(corpus = "rus",
                                  play = "pushkin-boris-godunov")
plot(godunov)
```

Пакет `remotes` можно так же использовать для загрузки пакетов из Bioconductor:

```
remotes::install_bioc("flowCore")
```

5.8 Где искать нужные пакеты

Мы разобрались с тем, как устанавливать пакеты. А где же их находить?

Это вопрос гораздо более сложный чем может показаться. Например, можно работать в R и не знать, что существует пакет, который решает нужную для вас задачу. Или же найти такой пакет и не знать, что есть более современный пакет, который делает это еще лучше!

Здесь нет каких-то готовых решений. *CRAN* пытается создавать и поддерживать тематические списки *(Task View)* пакетов с описанием задач, которые они решают:

https://cran.r-project.org/web/views/

Безусловно, если вы глубоко занимаетесь какой-либо темой из списка, то стоит изучить соотвестствующий *Task View*, но начинать знакомство с помощью *Task View* достаточно тяжело.

Другой вариант — просто погуглить, найти релевантные статьи или книги. Внимательно смотрите на дату публикации: R — очень быстро развивающийся язык, поэтому с большой вероятностью то, что было написано пять лет назад уже потеряло актуальность. Нет, работать это будет, но, скорее всего, появился более удобный и продвинутый инструмент.

Глава 6

Импорт и экспорт данных

Итак, пришло время перейти к реальным данным. Мы начнем с использования датасета (так мы будем называть любой набор данных) по супергероям. Этот датасет представляет собой табличку, каждая строка которой - отдельный супергерой, а столбик — какая-либо информация о нем. Например, цвет глаз, цвет волос, вселенная супергероя[1], рост, вес, пол и так далее. Несложно заметить, что этот датасет идеально подходит под структуру датафрейма: прямоугольная табличка, внутри которой есть разные колонки, каждая из которой имеет свой тип (числовой или строковый).

6.1 Рабочая папка и проекты RStudio

Для начала скачайте файл по ссылке

Он, скорее всего, появился у вас в папке “Загрузки”. Если мы будем просто пытаться прочитать этот файл (например, с помощью `read.csv()` — мы к этой функцией очень скоро перейдем), указав его имя и разрешение, то наткнемся на такую ошибку:

```
read.csv("heroes_information.csv")
```

[1]супергерои в комиксах, фильмах и телесериалах часто взаимодействуют друг с другом, однако обычно это взаимодействие происходит между супергероями одного издателя. Два крупнейших издателя комиксов — DC и Marvel, поэтому принято говорить о вселенной DC и Marvel.

```
Warning in file(file, "rt"): cannot open file 'heroes_information.csv': No such
file or directory

Error in file(file, "rt"): cannot open the connection
```

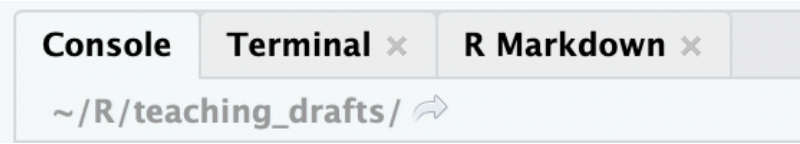
Это означает, что R не может найти нужный файл. Вообще-то мы даже не сказали, где искать. Нам нужно как-то совместить место, где R ищет загружаемые файлы и сами файлы. Для этого есть несколько способов.

- Магомет идет к горе: перемещение файлов в рабочую папку.

Для этого нужно узнать, какая папка является рабочей с помощью функции `getwd()` (без аргументов), найти эту папку в проводнике и переместить туда файл. После этого можно использовать просто название файла с разрешением:

```
heroes <- read.csv("heroes_information.csv")
```

Кроме того, путь к рабочей папке можно увидеть в RStudio во вкладке с консолью, в самой верхней части (прямо под надписью “Console”):



- Гора идет к Магомету: изменение рабочей папки.

Можно просто сменить рабочую папку с помощью `setwd()` на ту, где сейчас лежит файл, прописав путь до этой папки. Теперь файл находится в рабочей папке:

```
heroes <- read.csv("heroes_information.csv")
```

Этот вариант использовать не рекомендуется! Как минимум, это сразу делает невозможным запустить скрипт на другом компьютере. Ну а если все-таки вдруг повезет и получится, то ваш коллега будет очень недоволен, что ваш скрипт изменяет рабочую директорию.

- Гора находит Магомета по месту прописки: указание полного пути файла.

```
heroes <- read.csv("/Users/Username/Some_Folder/heroes_information.csv")
```

Этот вариант страдает теми же проблемами, что и предыдущий, поэтому тоже не рекомендуется!

> Для пользователей Windows есть дополнительная сложность: знак / является особым знаком для R, поэтому вместо него нужно использовать двойной //.

- Магомет использует кнопочный интерфейс: Import Dataset.

Во вкладке Environment справа в окне RStudio есть кнопка “Import Dataset”. Возможно, у Вас возникло непреодолимое желание отдохнуть от написания кода и понажимать кнопочки — сопротивляйтесь этому всеми силами, но не вините себя, если не сдержитесь.

- Гора находит Магомета в интернете.

Многие функции в R, предназначенные для чтения файлов, могут прочитать файл не только на Вашем компьютере, но и сразу из интернета. Для этого просто используйте ссылку вместо пути:

```
heroes <- read.csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
```

- Каждый Магомет получает по своей горе: использование проектов в RStudio.

На первый взгляд это кажется чем-то очень сложным, но это не так. Это очень просто и ОЧЕНЬ удобно. При создании проекта создается отдельная папка, где у вас лежат данные, хранятся скрипты, вспомогательные файлы и отчеты. Кроме папки создается файл формата *.Rproj*, в котором хранятся настройки проекта. Если нужно вернуться к другому проекту — просто открываете другой проект, с другими файлами и скриптами. Можно даже иметь открытыми несколько окон *RStudio* таким образом. Это еще помогает не пересекаться переменным из разных проектов — а то, знаете, использование двух переменных `data` в разных скриптах чревато ошибками. Поэтому очень удобным решением будет выделение отдельного проекта под этот курс.

При закрытии проекта все переменные по умолчанию тоже будут сохраняться, а при открытии — восстанавливаться (а вот пакеты все равно придется подгружать заново). Это очень удобно, хотя некоторые рекомендуют от этого отказаться. Это можно сделать во вкладке `Tool - Global Options...`

6.2 Организация проектов

Даже если не пользоваться проектами RStudio (но я настоятельно рекомендую, это очень удобно), то все равно имеет смысл разделять различные свои проекты по отдельным папкам. Для небольших проектов этого уже может быть достаточно, но я рекомендую делать немного более сложную структуру папок внутри проекта. Например, такую:

```
.
  my_project
      R
      data
          raw
          temp
          processed
      figures
      main_script.R
      my_project.Rproj
      output
      README.txt
```

В основной папке содержится автоматически созданный RStudio файл .Rproj, основной скрипт с формат .R (или же это может быть .Rmd файл — см. @ref(rmd)). Вспомогательные скрипты (например, с функциями) могут храниться в папке R. Если скриптов несколько, то их порядок стоит обозначить числами:

```
.
  01_first_script_preposcessing.R
  02_second_script_statistics.R
  03_third_script_figures.R
```

Данные стоит держать в отдельной папке, причем в некоторых ситуациях вы захотите создать отдельные подпапки, например, отдельные подпапки для данных на входе, временных файлов

и данных на выходе. Результаты работы, например, отчеты, сгенерированные с помощью *R Markdown* или *Quarto* (см. Глава 16). Туда же можно поместить папку с графиками или же можно поместить эту папку в корневую директорию.

Это лишь пример структуры организации проектов, детали могут различаться, но такая структура позволит не заблудиться в собственных файлах, если тех накопилось достаточно много. Кроме того, другому человеку в такой структуре проекта будет разобраться значительно проще

При создании папок внутри основного проекта важно помнить о том, что теперь ваши файлы больше нельзя найти в вашей корневой директории: нужно искать их в соответствующих папках. Это значит, что путь до файла теперь будет не `"heroes_information.csv"`, а `"data/heroes_information.csv"` или даже `"data/raw/heroes_information.csv"`.

***Полезное:* пакет `{here}`**

Пакет `{here}` позволяет удобно работать с путями на любых операционных системах, создавая путь в зависимости от вашей корневой директории проекта.

```
here::here("data", "heroes_information.csv")
```

```
[1] "/Users/ivan/R/tidy_stats/data/heroes_information.csv"
```

Созданный путь можно использовать для чтения файлов:

```
heroes <- read.csv(here::here("data", "heroes_information.csv"))
```

Сами скрипты тоже лучше разделять на смысловые части. Для этого есть горячие клавиши Cmd + Shift + R. Это сочетание клавиш выведет окно, в котором вам нужно вписать название, после чего появится вот такой аккуратный комментарий:

```
# Meaningful part of the script ------------------------------------------
```

Разделенный на такие части скрипт (да еще и с подробными комментариями) гораздо удобнее читать!

6.2.1 Табличные данные: текстовые и бинарные данные

Как вы уже поняли, импорт данных - одна из самых муторных и неприятных вещей в R. Если у вас получится с этим справится, то все остальное - ерунда. Мы уже разобрались с первой частью этого процесса - нахождением файла с данными, осталось научиться их читать.

Здесь стоит сделать небольшую ремарку. Довольно часто данные представляют собой табличку. Или же их можно свести к табличке. Такая табличка, как мы уже выяснили, удобно репрезентируется в виде датафрейма. Но как эти данные хранятся на компьютере? Есть два варианта: в *бинарном* и в *текстовом* файле.

Текстовый файл означает, что такой файл можно открыть в программе *Блокнот* или его аналоге (например, *TextEdit* на *macOS*) и увидеть напечатанный текст: скрипт, роман или упорядоченный набор цифр и букв. Нас сейчас интересует именно последний случай. Таблица может быть представлена как текст: отдельные строчки в файле будут разделять разные строчки таблицы, а какой-нибудь знак-разделитель отделять колонки друг от друга.

Для чтения данных из текстового файла есть довольно удобная функция `read.table()`. Почитайте хэлп по ней и ужаснитесь: столько разных параметров на входе! Но там же вы увидите функции `read.csv()`, `read.csv2()` и некоторые другие — по сути, это тот же `read.table()`, но с другими параметрами по умолчанию, соответствующие формату файла, который мы загружаем. В данном случае используется формат ***.csv***, что означает ***"Comma Separated Values"*** **(Значения, Разделенные Запятыми).** Формат *.csv* — это самый известный способ хранения табличных данных в файле на сегодняшний день. Файлы с расширением *.csv* можно легко открыть в любой программе, работающей с таблицами, в том числе *Microsoft Excel* и его аналогах.

Файл с расширением *.csv* — это просто текстовый файл, в котором "закодирована" таблица: разные строчки разделяют разные строчки таблицы, а столбцы отделяются запятыми (отсюда и название). Вы можете вручную создать такие файлы в *Блокноте* и сохранять их с форматом *.csv* - и такая табличка будет нормально открываться в *Microsoft Excel* и других программах для работы с таблицами. Можете попробовать это сделать самостоятельно!

Как говорилось ранее, в качестве разделителя ячеек по горизонтали — то есть разделителя между столбцами — используется запятая. С этим связана одна проблема: в некоторых странах (в т.ч. и

России) принято использовать запятую для разделения дробной части числа, а не точку, как это делается в большинстве стран мира. Поэтому есть альтернативный вариант формата .csv, где значения разделены точкой с запятой (;), а дробные значения - запятой (,). В этом и различие функций `read.csv()` и `read.csv2()` — первая функция предназначена для "международного" формата, вторая - для (условно) "российского". Оба варианта формата имеют расширение *.csv,* поэтому заранее понять какой именно будет вариант довольно сложно, приходится либо пробовать оба, либо заранее открывать файл в текстовом редакторе.

В первой строчке обычно содержатся названия столбцов - и это чертовски удобно, функции `read.csv()` и `read.csv2()` по умолчанию считают первую строчку именно как название для колонок.

Кроме *.csv* формата есть и другие варианты хранения таблиц в виде текста. Например, ***.tsv*** — тоже самое, что и *.csv,* но разделитель - знак табуляции. Для чтения таких файлов есть функция `read.delim()` и `read.delim2()`. Впрочем, даже если бы ее и не было, можно было бы просто подобрать нужные параметры для функции `read.table()`. Есть даже функции, которые пытаются сами "угадать" нужные параметры для чтения — часто они справляются с этим довольно удачно. Но не всегда. Поэтому стоит научиться справляться с любого рода данными на входе.

Итак, прочитаем наш файл. Для этого используем только параметр `file =`, который идет первым:

```
heroes <- read.csv("data/heroes_information.csv")
```

> ***Осторожно:* параметр `stringsAsFactors`**
>
> В более старых версиях R еще следовало указывать `stringsAsFactors = FALSE`. Параметр `stringsAsFactors =` задает то, как будут прочитаны строковые значения - как уже знакомые нам строки или как факторы. По сути, факторы - это примерно то же самое, что и `character`, но закодированные числами. Когда-то это было придумано для экономии используемых времени и памяти, сейчас же обычно становится просто лишней морокой. Некоторые функции требуют именно `character`, некоторые `factor`, в большинстве случаев это без разницы. Но иногда непонимание может привести к дурацким ошибкам. В данном случае мы просто пока обойдемся без факторов. Если у вас версия R выше 4.0.0, то `stringsAsFactors =` будет `FALSE` по умолчанию.

Можете проверить с помощью `View(heroes)`: все работает!

Если же вылезает какая-то странная ерунда или же просто ошибка - попробуйте другие функции (`read.table()`, `read.delim()`) и покопаться с параметрами. Для этого читайте `Help`.

6.3 Проверка импортированных данных

При импорте данных обратите внимания на предупреждения (если таковые появляются), в большинстве случаев они указывают на то, что данные импортированы некорректно.

Проверим, что все прочиталось нормально с помощью уже известной нам функции `str()`:

```
str(heroes)
```

```
'data.frame':   734 obs. of  11 variables:
 $ X         : int  0 1 2 3 4 5 6 7 8 9 ...
 $ name      : chr  "A-Bomb" "Abe Sapien" "Abin Sur" "Abomination" ...
 $ Gender    : chr  "Male" "Male" "Male" "Male" ...
 $ Eye.color : chr  "yellow" "blue" "blue" "green" ...
 $ Race      : chr  "Human" "Icthyo Sapien" "Ungaran" "Human / Radiation" ...
 $ Hair.color: chr  "No Hair" "No Hair" "No Hair" "No Hair" ...
 $ Height    : num  203 191 185 203 -99 193 -99 185 173 178 ...
 $ Publisher : chr  "Marvel Comics" "Dark Horse Comics" "DC Comics" "Marvel Comics" ..
 $ Skin.color: chr  "-" "blue" "red" "-" ...
 $ Alignment : chr  "good" "good" "good" "bad" ...
 $ Weight    : int  441 65 90 441 -99 122 -99 88 61 81 ...
```

> ***Осторожно:* проверяйте данные!**
>
> Всегда проверяйте данные на входе и никогда не верьте на слово, если вам говорят, что данные вычищенные и не содержат никаких ошибок.

На что нужно обращать внимание?

1. Прочитаны ли пропущенные значения как `NA`. По умолчанию пропущенные значения обозначаются пропущенной строчкой или `"NA"`, но встречаются самые разнообразные варианты. Возможные варианты кодирования пропущенных значений можно задать в параметре `na.strings =` функции `read.table()` и ее вариантов. В нашем наборе данных как раз такая

ситуация, где нужно самостоятельно задавать, какие значения будут прочитаны как `NA`.

```
heroes <- read.csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/h
                   na.strings = c("NA", "-", "-99"))
```

2. Прочитаны ли те столбики, которые должны быть числовыми, как `int` или `num`. Если в колонке содержатся числа, а написано `chr` (= `"character"`) или `Factor` (в случае если `stringsAsFactors = TRUE`), то, скорее всего, одна из строчек содержит в себе нечисловые знаки, которые не были прочитаны как `NA`.
3. Странные названия колонок. Это может случиться по самым разным причинам, но в таких случаях стоит открывать файл в другой программе и смотреть первые строчки. Например, может оказаться, что первые несколько строчек — пустые или что первая строчка не содержит название столбцов (тогда для параметра `header =` нужно поставить `FALSE`)
4. Вместо строковых данных у вас кракозябры. Это означает проблемы с кодировкой. В первую очередь попробуйте выставить значение `"UTF-8"` для параметра `encoding =` в функции для чтения файла:

```
heroes <- read.csv("data/heroes_information.csv",
                   encoding = "UTF-8")
```

В случае если это не помогает, попробуйте разобрать, что это за кодировка.

5. Все прочиталось как одна колонка. В этом случае, скорее всего, неправильно подобран разделить колонок — параметр `sep =`. Откройте файл в текстовом редакторе, чтобы понять какой нужно использовать.
6. В отдельных строчках все прочиталось как одна колонка, а в остальных нормально. Скорее всего, в файле есть значения типа `\` или `"`, которые в функциях `read.csv()`, `read.delim()`, `read.csv2()`, `read.delim2()` читаются как символы для закавычивания значений. Это может понадобиться, если у вас в таблице есть строковые значения со знаками `,` или `;`, которые могут восприниматься как разделитель столбцов.
7. Появились какие-то новые числовые колонки. Возможно неправильно поставлен разделитель дробной части. Обычно это либо `.` (`read.table()`, `read.csv()`, `read.delim()`), либо `,` (`read.csv2()`, `read.delim2()`).

Конкретно в нашем случае все прочиталось хорошо с помощью функции `read.csv()`, но в строковых переменных есть много прочерков, которые обозначают отсутствие информации по данному параметру супергероя, т.е. пропущенное значение. А вот с числовыми значениями все не так просто: для всех супергероев прописано какое-то число, но во многих случаях это `-99`. Очевидно, отрицательного роста и массы не бывает, это просто обозначение пропущенных значений (такое иногда используется). Таким образом, чтобы адекватно прочитать файл, нам нужно поменять параметр `na.strings =` функции `read.csv()`:

```
heroes <- read.csv("data/heroes_information.csv",
                   na.strings = c("NA", "-", "-99"))
```

6.4 Экспорт данных

Представим, что вы хотите сохранить табличку с данными про супергероев из вселенной DC в виде отдельного файла *.csv.*

```
dc <- heroes[heroes$Publisher == "DC Comics",]
```

Функция `write.csv()` позволит записать датафрейм в файл формата *.csv:*

```
write.csv(dc, "data/dc_heroes_information.csv")
```

Обычно названия строк не используются, и их лучше не записывать, поставив для `row.names =` значение `FALSE`:

```
write.csv(dc, "data/dc_heroes_information.csv", row.names = FALSE)
```

По аналогии с `read.csv2()`, `write.csv2()` позволит записать файлы формата .csv с разделителем `;`.

```
write.csv2(dc, "data/dc_heroes_information.csv", row.names = FALSE)
```

6.5 Импорт таблиц в бинарном формате: таблицы Excel, SPSS

Тем не менее, далеко не всегда таблицы представлены в виде текстового файла. Самый распространенный пример таблицы в бинарном виде — родные форматы Microsoft Excel. Если Вы попробуете открыть .xlsx файл в Блокноте, то увидите кракозябры. Это делает работу с этим файлами гораздо менее удобной, поэтому стоит избегать экселевских форматов и стараться все сохранять в .csv.

Такие файлы не получится прочитать при помощи базового инструментария R. Тем не менее, для чтения таких файлов есть много дополнительных пакетов:

- файлы Microsoft Excel: лучше всего справляется пакет `readxl` (является частью расширенного tidyverse), у него есть много альтернатив (`xlsx`, `openxlsx`).
- файлы SPSS, SAS, Stata: существуют два основных пакета — `haven` (часть расширенного tidyverse) и `foreign`.

Что такое пакеты и как их устанавливать мы изучим очень скоро.

6.6 Импорт данных из Google Sheets

Все чаще “кнопочная” работа с данными переезжает из Excel в облачный Google Sheets, который обладает схожим интерфейсом и функционалом, но позволяет удобно работать нескольким пользователям одновременно.

Оттуда данные можно легко выгрузить в нужном формате. Конечно, и в .csv тоже. Но было бы удобно загружать данные из Google Sheets напрямую, по ссылке. И это вполне возможно и даже не очень трудно! Лучший пакет для этого – `googlesheets4`.

```
install.packages("googlesheets")
```

Основная функция – read_sheet(), в ней нужно прописать ссылку, которую можно получить в “Настройках доступа” (или которую вам уже прислали).

```
heroes_form_gsh <- googlesheets4::read_sheet("https://docs.google.com/spreadsheets/d
```

После этого в консоли нужно будет выбрать Google-аккаунт:

```
The googlesheets4 package is requesting access to your Google account.
Select a pre-authorised account or enter '0' to obtain a new token.
Press Esc/Ctrl + C to cancel.

1:

Selection:
```

Выбираете (в данном случае у меня только один аккаунт, поэтому пишу *1* и жму *Enter*).

После этого откроется окно в веб-браузере, в котором Google будет спрашивать, доверяете ли вы R и готовы ли дать ему доступ к чтению таблицы (разумеется, отвечаем, что да). Это нужно будет сделать всего один раз, так что в дальнейшем нажимать в веб-браузере ничего будет не нужно.

После этого таблица загрузится.

6.7 Быстрый импорт данных

Чтение табличных данных обычно происходит очень быстро. По крайней мере, до тех пор пока ваши данные не содержат очень много значений. Если вы попробуете прочитать с помощью `read.csv()` таблицу с миллионами строчками, то заметите, что это происходит довольно медленно. Впрочем, эта проблема эффективно решается дополнительными пакетами.

- Пакет `readr` (часть базового tidyverse) предлагает функции, очень похожие на стандартные `read.csv()`, `read.csv2()` и тому подобные, только в названиях используется нижнее подчеркивание: `read_csv()` и `read_csv2()`. Они быстрее и немного удобнее, особенно если вы работаете в tidyverse.

```
readr::read_csv("data/heroes_information.csv",
          na = c("-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)

Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Пакет `vroom` - это часть расширенного tidyverse. Это такая альтернатива `readr` из того же tidyverse, но еще быстрее (отсюда и название).

```
vroom::vroom("data/heroes_information.csv")
```

```
New names:
Rows: 734 Columns: 11
-- Column specification
-------------------------------------------------------- Delimiter: "," chr
(8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A... dbl
(3): ...1, Height, Weight
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `` -> `...1`
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black           -99 Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        -        Blond           -99 NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        -        Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Пакет `data.table` - это не просто пакет, а целый фреймворк для работы с R, основной конкурент tidyverse. Одна из основных фишек `data.table` - быстрота работы. Это касается не только процессинга данных, но и их загрузки и записи. Поэтому некоторые используют функции `data.table` для чтения и записи данных в отдельности от всего остального пакета - они даже и называются соответствующе: `fread()` и `fwrite()`, где **f** означает **f**ast[2].

```
data.table::fread("data/heroes_information.csv")
```

```
       V1            name Gender Eye color              Race       Hair color
  1:    0          A-Bomb   Male    yellow             Human          No Hair
  2:    1      Abe Sapien   Male      blue     Icthyo Sapien          No Hair
  3:    2        Abin Sur   Male      blue           Ungaran          No Hair
  4:    3     Abomination   Male     green Human / Radiation          No Hair
  5:    4         Abraxas   Male      blue     Cosmic Entity            Black
 ---
730:  729 Yellowjacket II Female      blue             Human Strawberry Blond
731:  730            Ymir   Male     white       Frost Giant          No Hair
732:  731            Yoda   Male     brown    Yoda's species            White
733:  732         Zatanna Female      blue             Human            Black
734:  733            Zoom   Male       red                 -            Brown
      Height         Publisher Skin color Alignment Weight
  1:  203.0     Marvel Comics          -      good    441
  2:  191.0 Dark Horse Comics       blue      good     65
```

[2] А еще **f**riendly: `fread()` обычно самостоятельно хорошо угадывает формат таблицы на входе. `vroom` тоже так умеет.

```
  3:  185.0         DC Comics        red      good     90
  4:  203.0     Marvel Comics          -       bad    441
  5:  -99.0     Marvel Comics          -       bad    -99
 ---
730:  165.0     Marvel Comics          -      good     52
731:  304.8     Marvel Comics      white      good    -99
732:   66.0      George Lucas      green      good     17
733:  170.0         DC Comics          -      good     57
734:  185.0         DC Comics          -       bad     81
```

Чем же пользоваться среди всего этого многообразия? Бенчмарки[3] показывают, что быстрее всех `vroom` и `data.table`. Если же у вас нет задачи ускорить работу кода на несколько миллисекунд или прочитать датасет на много миллионов строк, то стандартного `read.csv()` (если вы работаете в базовом R) и `readr::read_csv()` (если вы работаете в tidyverse) должно быть достаточно.

Все перечисленные пакеты повзоляют не только быстро импортировать данные, но и быстро (и удобно!) экспортировать их:

```
readr::write_csv(dc, "data/dc_heroes_information.csv")
readr::write_excel_csv(dc, "data/dc_heroes_information.csv") #     Excel
vroom::vroom_write(dc, "data/dc_heroes_information.csv", delim = ",")
data.table::fwrite(dc, "data/dc_heroes_information.csv")
```

В плане скорости записи файлов соотношение сил примерно такое же, как и для чтения: `vroom` и `data.table` обгоняют всех, затем идет `readr`, и только после него - базовые функции R.

[3] бенчмаркинг — это тест производительности, в данном случае — сравнение скорости работы конкурирующих пакетов.

Глава 7

Условные конструкции и циклы

7.1 Выражения `if`, `else`, `else if`

Стандартная часть практически любого языка программирования — условные конструкции. R не исключение. Однако и здесь есть свои особенности. Начнем с самого простого варианта с одним условием. Выглядеть условная конструкция будет вот так:

```
if (    )
```

Вот так это будет работать на практике:

```
number <- 1
if (number > 0) "           "
```

```
[1] "           "
```

Если выражение (expression) содержит больше одной строчки, то они объединяются фигурными скобками. Впрочем, использовать их можно, даже если строчка всего в выражении всего одна.

```
number <- 1
if (number > 0) {
  "           "
}
```

```
[1] "           "
```

В рассмотренной нами конструкции происходит проверка на условие. Если условие верно[1], то происходит то, что записано в последующем выражении. Если же условие неверно[2], то ничего не происходит.

Оператор `else` позволяет задавать действие на все остальные случаи:

```
if (    )       else
```

Работает это так:

```
number <- -3
if (number > 0) {
  "           "
} else {
  "                "
}
```

```
[1] "                "
```

Иногда нам нужна последовательная проверка на несколько условий. Для этого есть оператор `else if`. Вот как выглядит ее применение:

```
number <- 0
if (number > 0) {
  "           "
} else if (number < 0){
  "           "
} else {
  "   "
}
```

```
[1] "   "
```

Как мы помним, R — язык, в котором векторизация играет большое значение. Но вот незадача — условные конструкции не векторизованы в R! Давайте попробуем применить эти конструкции для вектора значений и посмотрим, что получится.

[1] В принципе, необязательно внутри должна быть проверка условий, достаточно просто значения `TRUE`.

[2] Аналогично, достаточно просто значения `FALSE`.

```
numbers <- -2:2
if (numbers > 0) {
  "           "
} else if (numbers < 0){
  "           "
} else {
  "  "
}
```

```
Error in if (numbers > 0) {: the condition has length > 1
```

Ошибка! Однако если у вас более старая версия R (до 4.2.0, апрель 2022), то вместо ошибки будет учитываться только первое значение вектора условий: остальные будут игнорироваться, при этом будет выводиться предупреждение. Как же посчитать для всего вектора сразу?

> ***Полезное:* применение условных конструкций**
>
> Невекторизованная конструкция *if/else/else if* неудобна при работе с данными, ее практически не используют для обработки данных. В основном она применяется при написании функций, чтобы проверить конкретное значение параметра или адекватность данных на входе (см. Глава 8).

7.2 Циклы `for`

Во-первых, можно использовать `for`. Синтаксис у `for` похож на синтаксис условных конструкций.

```
for(      in          )
```

Теперь мы можем объединить условные конструкции и `for`. Немножко монструозно, но это работает:

```
for (i in numbers) {
  if (i > 0) {
    print("          ")
  } else if (i < 0) {
    print("          ")
  } else {
```

```
    print("  ")
  }
}
```

```
[1] "          "
[1] "          "
[1] "  "
[1] "          "
[1] "          "
```

> ***Осторожно:*** `print()`
>
> Чтобы выводить в консоль результат вычислений внутри `for`, нужно использовать `print()`.

Здесь стоит отметить, что `for` используется в R относительно редко. В подавляющем числе ситуаций использование `for` можно избежать. Обычно мы работаем в R с векторами или датафреймами, которые представляют собой множество относительно независимых наблюдений. Если мы хотим провести какие-нибудь операции с этими наблюдениями, то они обычно могут быть выполнены параллельно. Скажем, вы хотите для каждого испытуемого пересчитать его массу из фунтов в килограммы. Этот пересчет осуществляется по одинаковой формуле для каждого испытуемого. Эта формула не изменится из-за того, что какой-то испытуемый слишком большой или слишком маленький - для следующего испытуемого формула будет прежняя. Если Вы встречаете подобную задачу (где функцию можно применить независимо для всех значений), то без цикла `for` вполне можно обойтись.

Даже во многих случаях, где расчеты для одной строчки зависят от расчетов предыдущих строчек, можно обойтись без `for` векторизованными функциями, например, `cumsum()` для подсчета кумулятивной суммы.

```
cumsum(1:10)
```

```
 [1]  1  3  6 10 15 21 28 36 45 55
```

Если же нет подходящей векторизованной функции, то можно воспользоваться семейством функций `apply()` (см. Глава 8.5).

> ***Полезное:* зачем циклы? прост**
>
> После этих объяснений кому-то может показаться странным, что я вообще упоминаю про эти циклы. Но для кого-то циклы `for` настолько привычны, что их полное отсутствие в курсе может показаться еще более странным. Поэтому лучше от меня, чем на улице.

Зачем вообще избегать конструкций `for`? Некоторые говорят, что они слишком медленные, и частично это верно, если мы сравниваем с векторизованными функциями, которые написаны на более низкоуровневых языках. Но в большинстве случаев низкая скорость `for` связана с неправильным использованием этой конструкции. Например, стоит избегать ситуации, когда на каждой итерации `for` какой-то объект (вектор, список, что угодно) изменяется в размере. Лучше будет создать заранее объект нужного размера, который затем будет наполняться значениями:

```
numbers_descriptions <- character(length(numbers)) #                    ,
for (i in 1:length(numbers)) {
  if (numbers[i] > 0) {
    numbers_descriptions[i] <- "          "
  } else if (numbers[i] < 0) {
    numbers_descriptions[i] <- "          "
  } else {
    numbers_descriptions[i] <- "  "
  }
}
numbers_descriptions
```

```
[1] "          " "          " "  "
[4] "          " "          "
```

В общем, при правильном обращении с `for` особых проблем со скоростью не будет, хотя векторизованные функции и будут быстрее. Но все равно это будет громоздкая конструкция, в которой легко ошибиться, и которую, скорее всего, можно заменить одной короткой строчкой. Кроме того, без конструкции `for` код обычно легко превратить в набор функций, последовательно применяющихся к данным, что мы будем по максимуму использовать, работая в tidyverse и применяя пайпы (см. Глава 10.4).

7.3 Векторизованные условные конструкции: функции `ifelse()` и `dplyr::case_when()`

Из-за того, что конструкция *if/else/else if* не векторизованная, она редко используется непосредственно в операциях с данными, обычно она используется при написании функций (Глава 8.1) и разработке пакетов.

Альтернативой сочетанию условных конструкций и циклов `for` является использование встроенной функции `ifelse()`. Функция `ifelse()` принимает три аргумента:

1. `test =` – условие (т.е. просто логический вектор, состоящий из `TRUE` и `FALSE`),
2. `yes =` – что выдавать в случае `TRUE`,
3. `no =` – что выдавать в случае `FALSE`.

На выходе получается вектор такой же длины, как и изначальный логический вектор (условие). Это очень похоже на `()` в *Microsoft Excel*.

```
ifelse(numbers > 0, "          ", "                ")
```

```
[1] "                " " "                "
[3] "                " " "           "
[5] "           "
```

> ***Полезное:* иногда `ifelse()` излишен**
>
> Периодически я встречаю у студентов строчку вроде такой: `ifelse(  , TRUE, FALSE)`. Эта конструкция избыточна, т.к. получается, что логический вектор из `TRUE` и `FALSE` превращается в абсолютно такой же вектор из `TRUE` и `FALSE` на тех же самых местах. Выходит, что ничего не меняется!

> ***Осторожно:* `NA` в `ifelse()` превращается в `NA`**
>
> `NA` в условии в `ifelse()` возвращает `NA`. Обычно это именно то поведение, которое вы ожидаете: если в исходном векторе есть неопределенность, то и на выходе должна остаться неопределенность на соответствующих позициях.

Пакеты `{dplyr}` и `{data.table}` предоставляют более быстрые и более строгие альтернативы для базовой функции `ifelse()` с аналогичным синтаксисом:

```
dplyr::if_else(numbers > 0, "           ", "                ")
```

```
[1] "                " " "                "
[3] "                " " "           "
[5] "           "
```

```
data.table::fifelse(numbers > 0, "           ", "                ")
```

```
[1] "                " " "                "
[3] "                " " "           "
[5] "           "
```

Если вы пользуетесь одним из этих пакетов (о них пойдет речь далее — см. Глава 9, то я советую пользоваться соотвествующей функцией вместо базового `ifelse()`.

> ***Осторожно:* возвращение `NA` может привести к ошибкам**
>
> Обе функции будут избегать скрытого приведения типов (см. Глава 3.2) и намеренно выдавать ошибку при использовании разных типов данных в параметрах `yes =` и `no =`[a]. Помните, что `NA` по умолчанию — это логический тип данных, поэтому в этих функциях нужно использовать `NA` соответствующего типа `NA_character_`, `NA_integer_`, `NA_real_`, `NA_complex_` (см. Глава 3.7).
>
> [a] В более свежих версиях пакета `{dplyr}` разработчики отказались от этой "строгости", поэтому `NA` все-таки будут приводиться к нужному типу. Однако `data.table::fifelse()` остался строгим.

У `ifelse()` тоже есть недостаток: он не может включать в себя дополнительных условий по типу `else if`. В простых ситуациях можно вставлять `ifelse()` внутри `ifelse()`:

```
ifelse(numbers > 0,
       "           ",
       ifelse(numbers < 0, "           ", "   "))
```

```
[1] "           " " "           " " "   "
[4] "           " " "           "
```

Достаточно симпатичное решение есть в пакете `dplyr` — функция `case_when()`, которая работает с использованием формулы:

```
dplyr::case_when(
  numbers > 0 ~ "           ",
  numbers < 0 ~ "           ",
  numbers == 0 ~ "   ")
```

```
[1] "           " " "           " "   "
[4] "           " " "           "
```

Функция `case_when()` работает по той же логике, что и *if/else/else if* конструкция: сначала идет проверка на первое условие (как первое *if* в конструкции *if/else/else if*). Если проверка проходит (то есть в условии получается `TRUE`), то соответствующее значение возвращается, а остальные условия не проверяются. Если же первое условие не выполняется, то идет проверка на следующее условие (аналог *else if*). Если же и оно не выполняется, то идет проверка на следующее (следующее *else if*), пока проверка не пройдет до последнего условия. Можно поставить значение по умолчанию с помощью параметра `.default =`, которое будет возвращаться, если все проверки выдали `FALSE`.

```
heroes <- read.csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d

heroes$weight_group <- dplyr::case_when(
  heroes$Weight > 200 ~ "overweight", # "if"
  heroes$Weight > 120 ~ "somewhat overweight", # "else if"
  heroes$Weight < 50 ~ "underweight", # next "else if"
  .default = "typical weight" # final "else"
)
```

Осторожно: не забывайте про запятые

Будьте внимательны с запятыми! Несмотря на довольно экстравагантный синтаксис, `case_when()` – это по-прежнему функция, а различные условия, которые вы прописываете внутри этой функции – это аргументы этой функции.

Важный момент: если `ifelse()` возвращает `NA` на `NA` в условии, что обычно нас устраивает (у нас нет данных, что у нас на входе, следовательно, не знаем, что на выходе), то `case_when()` такого не делает. `NA` в условии считается как `FALSE`, поэтому нужно дополнительно обрабатывать условие для него. Чтобы на место `NA` поставить NA, нужно записать вот так:

```
heroes$weight_group <- dplyr::case_when(
  heroes$Weight > 200 ~ "overweight", # "if"
  heroes$Weight > 120 ~ "somewhat overweight", # "else if"
  heroes$Weight < 50 ~ "underweight", # next "else if"
  is.na(heroes$Weight) ~ NA, # one more "else if", maps NA to NA
  .default = "typical weight" # final "else"
  ) # final "else"
```

В {data.table} тоже есть свой (более быстрый) аналог case_when() — функция fcase(). Синтаксис отличается только тем, что вместо формул используются простые запятые. То есть первый аргумент – условие, второй – значение, которое возвращается при верности первого аргумента, третий аргумент – условие, четвертый – возвращаемое значение при верности третьего аргумента и т.д.

```
data.table::fcase(
  numbers > 0, "           ",
  numbers < 0, "           ",
  numbers == 0, "   ")
```

```
[1] "           " " "           " "   "
[4] "           " "           "
```

Задача создания вектора или колонки по множественным условиям из другой колонки плавно перетекает в задачу объединения двух датафреймов по единому ключу, и такое решение может оказаться наиболее быстрым (см. Глава 11.1.2).

Глава 8

Функциональное программирование в R

8.1 Создание функций

Поздравляю, сейчас мы выйдем на качественно новый уровень владения R. Вместо того, чтобы пользоваться теми функциями, которые уже написали за нас, мы можем сами создавать свои функции! В этом нет ничего сложного.

Синтаксис создания функции внешне похож на создание циклов или условных конструкций. Мы пишем ключевое слово `function`, в круглых скобках обозначаем переменные, с которыми собираемся что-то делать. Внутри фигурных скобок пишем выражения, которые будут выполняться при запуске функции. У функции есть свое собственное **окружение *(environment)*** — место, где хранятся переменные. Именно те объекты, которые мы передаем в скобочках, и будут в окружении, так же как и “обычные” переменные для нас в глобальном окружении. Это означает, что функция будет искать переменные в первую очередь среди объектов, которые переданы в круглых скобочках. С ними функция и будет работать. На выходе функция выдаст то, что вычисляется внутри функции `return()`. Если `return()` появляется в теле функции несколько раз, то до результат будет возвращаться из той функции `return()`, до которой выполнение дошло первым.

```
pow <- function(x, p) {
  power <- x ^ p
  return(power)
```

```
}
pow(3, 2)
```

```
[1] 9
```

Если функция проработала до конца, а функция `return()` так и не встретилась, то возвращается последнее посчитанное значение.

```
pow <- function(x, p) {
  x ^ p
}
pow(3, 2)
```

```
[1] 9
```

Если в последней строчке будет присвоение, то функция ничего не вернет обратно. Это очень распространенная ошибка: функция вроде бы работает правильно, но ничего не возвращает. Нужно писать так, как будто бы в последней строчке результат выполнения выводится в консоль.

```
pow <- function(x, p) {
  power <- x ^ p #                ,                          !
}
pow(3, 2) #
```

Если функция небольшая, то ее можно записать в одну строчку без фигурных скобок.

```
pow <- function(x, p) x ^ p
pow(3, 2)
```

```
[1] 9
```

> ***Для продвинутых:*** `{}` **– блок инструкций**
>
> Вообще, фигурные скобки используются для того, чтобы выполнить серию команд, но вернуть только результат выполнения последней команды. Такой набор команд называется **блоком инструкций *(statements block)*.** Это можно использовать, чтобы не создавать лишних временных

переменных в глобальном окружении.

Мы можем оставить в функции параметры по умолчанию.

```
pow <- function(x, p = 2) x ^ p
pow(3)
```

```
[1] 9
```

```
pow(3, 3)
```

```
[1] 27
```

***Для продвинутых:* ленивые вычисления**

В R работают **ленивые вычисления (lazy evaluations)**. Это означает, что параметры функций будут только когда они понадобятся, а не заранее. R будет как самый ленивый прокрастинатор откладывать чтение данных, пока они не понадобятся в вычислениях. Это приводит к тому, что если параметр никак не задан, то обнаружится это только при его непосредственном использовании. Например, эта функция не будет выдавать ошибку, если мы не зададим параметр `we_will_not_use_this_parameter =`, потому что он нигде не используется в расчетах.

```
pow <- function(x, p = 2, we_will_not_use_this_parameter) x ^ p
pow(x = 3)
```

```
[1] 9
```

8.2 Проверка на адекватность

Лучший способ не бояться ошибок и предупреждений — научиться прописывать их самостоятельно в собственных функциях. Это позволит понять, что за текстом предупреждений и ошибок, которые у вас возникают, стоит забота разработчиков о пользователях, которые хотят максимально обезопасить нас от наших непродуманных действий.

Хорошо написанные функции не только выдают правильный результат на все возможные адекватные данные на входе, но и не дают получить правдоподобные результаты при неадекватных входных данных. Как вы уже знаете, если на входе у вас имеются пропущенные значения, то многие функции будут в ответ тоже выдавать пропущенные значения. И это вполне осознанное решение, которое позволяет избегать ситуаций вроде той, когда около одной пятой научных статей по генетике содержало ошибки в приложенных данных и замечать пропущенные значения на ранней стадии. Кроме того, можно проводить **проверки на адекватность входящих данных *(sanity check).***

Разберем **проверку на адекватность входящих** данных на примере самодельной функции `imt()`, которая выдает индекс массы тела, если на входе задать вес (аргумент `weight =`) в килограммах и рост (аргумент `height =`) в метрах.

```
imt <- function(weight, height) weight / height ^ 2
```

Проверим, что функция работает верно:

```
w <- c(60, 80, 120)
h <- c(1.6, 1.7, 1.8)
imt(weight = w, height = h)
```

```
[1] 23.43750 27.68166 37.03704
```

Очень легко перепутать и написать рост в сантиметрах. Было бы здорово предупредить об этом пользователя, показав ему предупреждающее сообщение, если рост больше, чем, например, 3. Это можно сделать с помощью функции `warning()`.

```
imt <- function(weight, height) {
  if (any(height > 3)) warning("          height     3:     ,                ,
  weight / height ^ 2
}
imt(78, 167)
```

```
Warning in imt(78, 167):          height     3:     ,                ,
```

```
[1] 0.002796802
```

В некоторых случаях ответ будет совершенно точно некорректным, хотя функция все посчитает и выдаст ответ, как будто так и надо.

Например, если какой-то из аргументов функции `imt()` будет меньше или равен 0. В этом случае нужно прописать проверку на это условие. Если это действительно так, то можно поступить еще строже: выдать пользователю ошибку.

```
imt <- function(weight, height) {
  if (any(weight <= 0 | height <= 0)) stop("                                    ")
  if (any(height > 3)) warning("          height     3:     ,                ,          \n")
  weight / height ^ 2
}
imt(-78, 167)
```

```
Error in imt(-78, 167):
```

Когда вы попробуете самостоятельно прописывать предупреждения и ошибки в функциях, то быстро поймете, что ошибки - это вовсе не обязательно результат того, что где-то что-то сломалось и нужно паниковать. Совсем даже наоборот, прописанная ошибка - чья-то забота о пользователях, которых пытаются максимально проинформировать о том, что и почему пошло не так.

Это естественно в начале работы с R (и вообще с программированием) избегать ошибок и пугаться их. Конечно, в самом начале обучения большая часть из них остается непонятной. Но постарайтесь понять текст ошибки, вспомнить в каких случаях у вас возникала похожая ошибка. Очень часто этого оказывается достаточно чтобы понять причину ошибки даже если вы только-только начали изучать R.

Ну а в дальнейшем я советую ознакомиться со средствами отладки кода в R для того, чтобы научиться справляться с ошибками в своем коде на более продвинутом уровне.

8.3 Когда и зачем создавать функции?

Когда стоит создавать функции? Существует “правило трех” — если у вас есть три куска очень похожего кода, то самое время превратить код в функцию. Это очень условное правило, но, действительно, стоит избегать копипастинга в коде. В этом случае очень легко ошибиться, а сам код становится нечитаемым.

Есть и другой подход к созданию функций: их стоит создавать не столько для того, чтобы использовать тот же код снова, сколько для абстрагирования от того, что происходит в отдельных

строчках кода. Если несколько строчек кода были написаны для того, чтобы решить одну задачу, которой можно дать понятное название (например, подсчет какой-то особенной метрики, для которой нет готовой функции в R), то этот код стоит обернуть в функцию. Если функция работает корректно, то теперь не нужно думать над тем, что происходит внутри нее. Вы ее можете мысленно представить как операцию, которая имеет определенный вход и выход — как и встроенные функции в R.

Отсюда следует важный вывод, что хорошее название для функции — это очень важно. Очень, очень, очень важно.

8.4 Функции как объекты первого порядка

Ранее мы убедились, что арифметические операторы — это тоже функции. На самом деле, практически все в R — это функции. Даже `function` — это функция `function()`. Даже скобочки `(`, `{` — это функции!

А сами функции — это объекты первого порядка в R. Это означает, что с функциями вы можете делать практически все то же самое, что и с другими объектами в R (векторами, датафреймами и т.д.). Небольшой пример, который может взорвать ваш мозг:

```
list(mean, min, `{`)
```

```
[[1]]
function (x, ...)
UseMethod("mean")
<bytecode: 0x104b4dda8>
<environment: namespace:base>

[[2]]
function (..., na.rm = FALSE)  .Primitive("min")

[[3]]
.Primitive("{")
```

Мы можем создать список из функций! Зачем — это другой вопрос, но ведь можем же! Например, создавать списки из функций может быть удобным для продвинутых операций в `across()` в пакете `{dplyr}` (см. Глава 11.3).

Еще можно создавать функции внутри функций[1], использовать

[1]Функция, которая создает другие функции, называется **фабрикой функций**.

функции в качестве аргументов функций, сохранять функции как переменные. Пожалуй, самое важное из этого всего - это то, что функция может быть аргументом в функции. Не просто название функции как строковая переменная, не результат выполнения функции, а именно сама функция как объект! Это лежит в основе использования семейства функций `apply()`, о которых пойдет речь далее, и многих фишек *tidyverse.*

> ***Полезное:* а как в Python?**
>
> В Python дело обстоит похожим образом: функции там тоже являются объектами первого порядка, поэтому все эти фишки функционального программирования (с поправкой на синтаксис, конечно) будут работать и там.

8.5 Семейство функций `apply()`

8.5.1 Применение `apply()` для матриц

Семейство? Да, их целое множество: `apply()`, `lapply()`,`sapply()`, `vapply()`,`tapply()`,`mapply()`, `rapply()`… Ладно, не пугайтесь, всех их знать не придется. Обычно достаточно первых двух-трех. Проще всего пояснить как они работают на простой матрице с числами:

```
A <- matrix(1:12, 3, 4)
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

> ***Осторожно:* `apply()` не работает с датафреймами**
>
> Функция `apply()` предназначена для работы с матрицами (или многомерными массивами). Если вы скормите функции `apply()` датафрейм, то этот датафрейм будет сначала превращен в матрицу. Главное отличие матрицы от датафрейма в том, что в матрице все значения одного типа, поэтому будьте готовы, что сработает имплицитное приведение к общему типу данных. Например, если среди колонок датафрейма есть хотя бы одна строковая колонка,

> то все колонки станут строковыми.

Теперь представим, что нам нужно посчитать что-нибудь (например, сумму) по каждой из строк. С помощью функции `apply()` вы можете в буквальном смысле “применить” функцию к матрице или датафрейму. Синтаксис такой: `apply(X, MARGIN, FUN, ...)`, где `X` — данные, `MARGIN` это `1` (для строк), `2` (для колонок), `c(1,2)` для строк и колонок (т.е. для каждого элемента по отдельности), а `FUN` — это функция, которую вы хотите применить! `apply()` будет брать строки/колонки из `X` в качестве первого аргумента для функции.

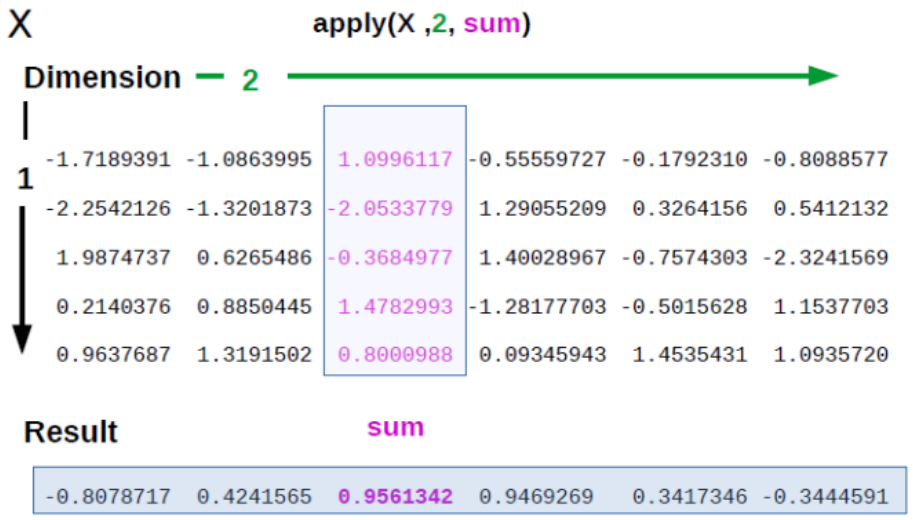


Рисунок 8.1: Визуальное представление функции apply(): выбираем матрицу, задаем способ итерации (по строкам или столбцам, выбираем функцию, которая будет применяться на каждом элементе.

> Заметьте, мы вставляем функцию без скобок и кавычек как аргумент в функцию. Это как раз тот случай, когда аргументом в функции выступает сама функция, а не ее название или результат ее выполнения.

Давайте разберем на примере:

```
apply(A, 1, sum) #
```

```
[1] 22 26 30
```

```
apply(A, 2, sum) #
```

```
[1]  6 15 24 33
```

```
apply(A, c(1,2), sum) #  ...
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

> ***Полезное:* специальные функции для операций над всеми строками/столбцами**
>
> Конкретно для подсчета сумм и средних по столбцам и строкам в R есть функции `colSums()`, `rowSums()`, `colMeans()` и `rowMeans()`, которые можно использовать как альтернативы `apply()` в данном случае.

Если же мы хотим прописать дополнительные аргументы для функции, то их можно перечислить через запятую после функции:

```
A[2, 2] <- NA
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2   NA    8   11
[3,]    3    6    9   12
```

```
apply(A, 1, sum)
```

```
[1] 22 NA 30
```

```
apply(A, 1, sum, na.rm = TRUE)
```

```
[1] 22 21 30
```

8.5.2 Анонимные функции

Что делать, если мы хотим сделать что-то более сложное, чем просто применить одну функцию? А если функция принимает не первым, а вторым аргументом данные из матрицы? В этом случае нам помогут **анонимные функции *(anonymous function)***.

Анонимные функции - это функции, которые будут использоваться один раз и без названия.

> ***Полезное:* а как в Python?**
>
> Питонистам знакомо понятие **лямбда-функций**. Да, это то же самое.

Например, мы можем посчитать общее количество знаков по строкам и столбцам без называния этой функции:

```
B <- matrix(c("  ", "    ", " ", "     ", "    ", " ", "   ", "  ", "    "), nrow = 3)
apply(B, 1, function(x) sum(nchar(x)))
```

```
[1] 18 17  8
```

```
apply(B, 2, function(x) sum(nchar(x)))
```

```
[1] 11 17 15
```

Начиная с R 4.1.0 (май 2021) можно использовать сокращенный вариант написания анонимных функций с `\` вместо ключевого слова `function`:

```
apply(B, 1, \(x) sum(nchar(x)))
```

```
[1] 18 17  8
```

```
apply(B, 2, \(x) sum(nchar(x)))
```

```
[1] 11 17 15
```

***Полезное:* аргументы анонимной функции**

Как и в случае с обычной функцией, в качестве `x` выступает объект, с которым мы хотим что-то сделать, а дальше следует функция, которую мы собираемся применить к . Можно использовать не , а что угодно, как и в обычных функциях:

```
  apply(B, 1, function(whatevername) sum(nchar(whatevername)))
```

```
[1] 18 17  8
```

8.5.3 Другие функции семейства `apply()`

ОК, с `apply()` разобрались. А что с остальными? Некоторые из функций семейства `*apply()` еще проще и не требуют индексов, например, `lapply()` (для применения к каждому элементу списка) и `sapply()` - упрощенная версия `lapply()`, которая пытается по возможности "упростить" результат до вектора или матрицы.

```
  some_list <- list(some = 1:10, list = letters)
  lapply(some_list, length)
```

```
$some
[1] 10

$list
[1] 26
```

```
  sapply(some_list, length)
```

```
some list
  10   26
```

***Осторожно:* такой непредсказуемый `sapply()`**

Достаточно сложно предсказать, в каких именно случаях будет произведено упрощение, а в каких нет. Поэтому `sapply()` удобен в исследовании данных, но использовать эту функцию в скриптах не очень рекомендуется. Один из вариантов решения этой проблемы — это функция `vapply()`,

которая позволяет управлять результатом `lapply()`, но гораздо более красиво эта проблема решена в пакете `{purrr}` (см. Глава 11.4).

Использование `sapply()` на векторе приводит к тем же результатам, что и просто применить векторизованную функцию обычным способом.

```
sapply(1:10, sqrt)
```

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278
```

```
sqrt(1:10)
```

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278
```

Зачем вообще тогда нужен `sapply()`, если мы можем просто применить векторизованную функцию? Ключевое слово здесь *векторизованная* функция. Если функция не векторизована, то `sapply()` становится удобным вариантом для того, чтобы избежать итерирования с помощью циклов `for`.

Для продвинутых: `Vectorize()`

Еще одна альтернатива - это векторизация невекторизованной функции с помощью `Vectorize()`. Эта функция просто оборачивает функцию одним из вариантов `apply()`. Обратите внимание: функция `Vectorize()` в качестве аргумента принимает функцию и возвращает тоже функцию!

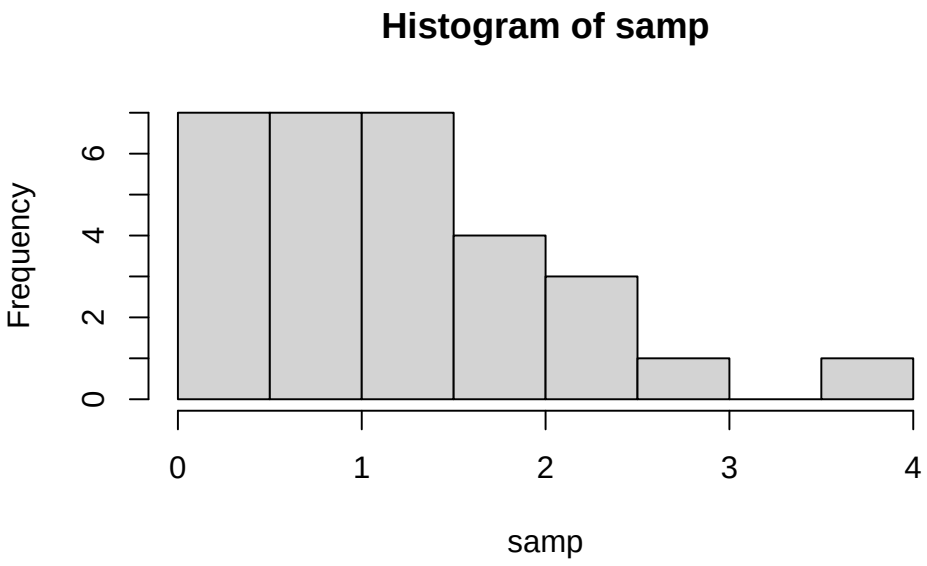
Можно применять функции `lapply()` и `sapply()` на датафреймах. Поскольку фактически датафрейм - это список из векторов одинаковой длины (см. Глава 4.4), то итерироваться эти функции будут по колонкам:

```
heroes <- read.csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
                   na.strings = c("-", "-99"))
sapply(heroes, class)
```

```
        X        name      Gender   Eye.color        Race  Hair.color
"integer" "character" "character" "character" "character" "character"
   Height   Publisher  Skin.color   Alignment      Weight
"numeric" "character" "character" "character"   "integer"
```

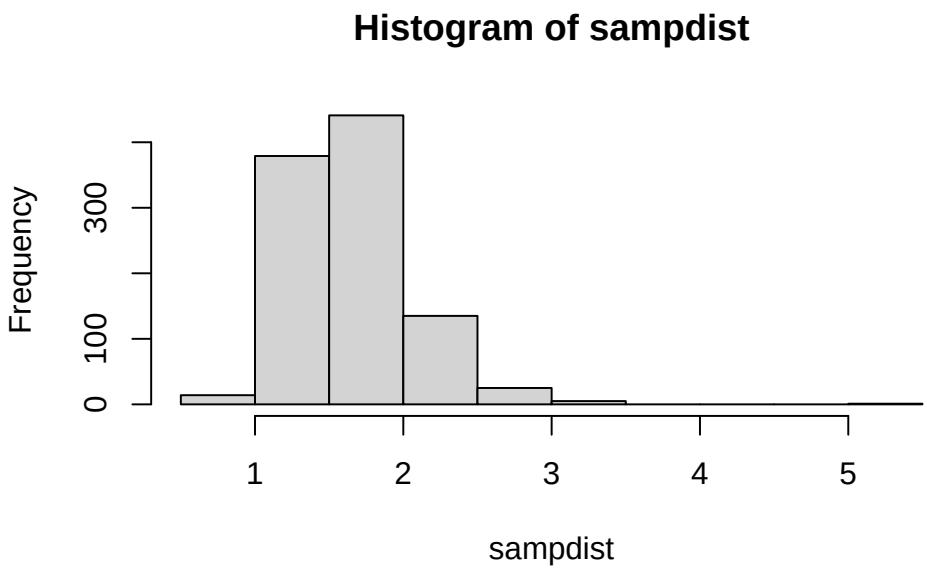
Еще одна функция из семейства `apply()` - функция `replicate()` - самый простой способ повторить одну и ту же операцию много раз. Обычно эта функция используется при симуляции данных и моделировании. Например, давайте сделаем выборку из логнормального распределения (подробнее про распределения см. в Глава 17.2):

```
samp <- rlnorm(30)
hist(samp)
```



А теперь давайте сделаем 1000 таких выборок и из каждой возьмем среднее:

```
sampdist <- replicate(1000, mean(rlnorm(30)))
hist(sampdist)
```



Про функции для генерации случайных чисел и про визуализацию будет в следующих главах: Глава 17.2 и Глава 13 соответственно.

> ***Для продвинутых:* больше `apply()`**
>
> Если хотите познакомиться с семейством `apply()` чуточку ближе, то рекомендую вот этот туториал.

В заключение стоит сказать, что семейство функций `apply()` — это очень сильное колдунство, но в *tidyverse* оно практически полностью перекрывается функциями из пакета `{purrr}`, за исключением самого `apply()` и некоторых других функций, которые работают с матрицами и массивами (*tidyverse* с ними принципиально не дружит). Впрочем, если вы поняли логику `apply()`, то при желании вы легко сможете переключиться на альтернативы из пакета `{purrr}` (см. Глава 11.4).

Часть II

Tidyverse

В этом разделе мы изучим популярную коллекцию пакетов tidyverse, которые стали настолько распространены, что фактически заменили собой базовый R.

- Глава Глава 9 посвящена введению в философию tidyverse, в ней tidyverse сравнивается с другим популярным пакетом для анализа данных – пакетом `{data.table}`.
- Глава Глава 10 посвящена знакомству с тибблом и пайпами, а также основным функциям tidyverse для выбора и работы с колонками, выбора и сортировки строк, создания новых колонок и аггрегации.
- Глава Глава 11 посвящена продвинутым инструментам tidyverse: соединению датафреймов, трансформации длинных данных в широкие и наоборот, функции `across()` для операций над несколькими колонками и пакету purrr для продвинутого функционального программирования.

Глава 9

За пределами base R: tidyverse и data.table

Как вы уже, наверное, убедились, базовый R умеет очень много, в том числе и для работы с данными. Однако какие-то операции все равно выполнить довольно непросто.

Возьмем, например, задачу агрегации: вам нужно посчитать средний рост супергероев отдельно для мужчин и для женщин (а еще и для `NA`, за компанию). Три группы еще ничего, а если бы их было 10, 50 или 200? В базовом R для этого есть специальная функция `aggregate()`, но она довольно неудобная.

Поэтому стали появляться пакеты, которые пытаются сделать агрегацию и другие непростые операции максимально безболезненными способами. Основных таких пакетов два: `{data.table}` и `{tidyverse}`. Это огромные пакеты, которые очень сильно изменяют работу в R, в том числе в плане стиля и используемой парадигмы. Тем не менее, в основе своей стоит все то, что мы прошли раньше.

9.1 Подход `{data.table}`

`{data.table}` – это распространенный пакет, который позволяет анализировать датафреймы максимально быстро и с помощью очень лаконичного кода.

```
install.packages("data.table")
```

Давайте импортируем наш набор данных про супергероев. Для этого воспользуемся функцией `fread()` из пакета `{data.table}`. Эта функция нам уже знакома как функция для импорта больших наборов данных Глава 6.7.

"f" в `fread()` означает *"**f**ast and **f**riendly"*: эта функция очень быстрая и довольно хорошо угадывает формат текстовой таблицы.

```
library(data.table)
heroes_dt <-
  fread(
    "https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes_info
    na.strings = c("NA", "-", "-99")
  )
```

Функция `fread()` создает не просто датафрейм, а ***дататейбл (datatable):***

```
heroes_dt
```

```
      V1            name Gender Eye color              Race       Hair color
  1:   0          A-Bomb   Male    yellow             Human          No Hair
  2:   1      Abe Sapien   Male      blue     Icthyo Sapien          No Hair
  3:   2        Abin Sur   Male      blue           Ungaran          No Hair
  4:   3     Abomination   Male     green Human / Radiation          No Hair
  5:   4         Abraxas   Male      blue     Cosmic Entity            Black
 ---
730: 729 Yellowjacket II Female      blue             Human Strawberry Blond
731: 730            Ymir   Male     white       Frost Giant          No Hair
732: 731            Yoda   Male     brown    Yoda's species            White
733: 732         Zatanna Female      blue             Human            Black
734: 733            Zoom   Male       red              <NA>            Brown
     Height         Publisher Skin color Alignment Weight
  1:  203.0     Marvel Comics       <NA>      good    441
  2:  191.0 Dark Horse Comics       blue      good     65
  3:  185.0         DC Comics        red      good     90
  4:  203.0     Marvel Comics       <NA>       bad    441
  5:     NA     Marvel Comics       <NA>       bad     NA
 ---
730:  165.0     Marvel Comics       <NA>      good     52
731:  304.8     Marvel Comics      white      good     NA
732:   66.0      George Lucas      green      good     17
733:  170.0         DC Comics       <NA>      good     57
734:  185.0         DC Comics       <NA>       bad     81
```

```
class(heroes_dt)
```

```
[1] "data.table" "data.frame"
```

Дататейбл – это "улучшенный" датафрейм: с ним работают все те функции, которые мы применяли для датафрейма, специальные функции для дататейбла, а что-то работает немного по-другому по сравнению с датафреймом. Например, оператор [, т.е. квадратные скобки.

Давайте посмотрим по-внимательнее как это происходит на примере расчета среднего роста супергероев, группируя по полу:

```
heroes_dt[, mean(Height, na.rm = TRUE), by = Gender]
```

```
   Gender       V1
1:   Male 191.9749
2: Female 174.6840
3:   <NA> 177.0667
```

Сразу уже усложним задачу: возьмем только хороших (у кого в колонке `Alignment` стоит `"good"`), а потом еще отсортируем по среднему росту.

```
heroes_dt[Alignment == "good",
          .(mean_height = mean(Height, na.rm = TRUE)),
          by = Gender][
            order(-mean_height)
          ]
```

```
   Gender mean_height
1:   Male    188.9601
2:   <NA>    179.5000
3: Female    174.7607
```

Уух! Выглядит монструозно, да? Зато как мы все сделали используя минимальное количество знаков. Заметьте, что здесь необычного для нас:

- Не нужно прописывать `heroes_dt$Alignment`, поиск переменной будет начинаться с колонок дататейбла.

- Там, где мы раньше выбирали колонки, мы еще и расчеты можем вести.
- Внутри квадратных скобок появилась вторая запятая, т.е. третье поле, в котором мы прописали группировку.
- Несколько операций прописываются путем соединения квадратных скобочек, код превращается в эдакий паровозик[1].

И это не все отличия!

На сайте пакета `{data.table}` особенно уделяется вниманию скорости `{data.table}`, приводя в качестве доказательства бэнчмарк, где сравниваются по скорости различные инструменты для работы с данными. `{data.table}` почти на порядок обгоняет как `{dplyr}`, так и питоновский *pandas* – самый используемый пакет для анализа данных в *Python*.

Разработчики `{data.table}` делают особый акцент на "консервативности" пакета: у него нет никаких зависимостей (в этом плане пакет `{data.table}` обгоняет большинство российских экспатов в Тбилиси), ему достаточно очень старой версии R, функционирование пакета не будет ломаться из-за выкинутых устаревших функций. В общем, `{data.table}` очень суров и уважаем программистами. Он и не особо пытается понравиться рядовым пользователям. Зато освоив его, вы сможете творить магию: то, что с помощью базового R, *tidyverse* или Python будет выполняться очень долго (если выполнится вообще), `{data.table}` сможет сделать гораздо быстрее, иногда в десятки и сотни раз!

Очень сильно, не правда ли? Чем же может ответить *tidyverse*?

9.2 Подход tidyverse

Давайте посмотрим, как будет выглядеть решение тех же задач (отбор строк по условию, агрегация и сортировка) в *tidyverse*.

```
install.packages("tidyverse")
```

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

[1] Это работает и в базовом R, но именно в `{data.table}` это очень частая конструкция.

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::between()     masks data.table::between()
x dplyr::filter()      masks stats::filter()
x dplyr::first()       masks data.table::first()
x lubridate::hour()    masks data.table::hour()
x lubridate::isoweek() masks data.table::isoweek()
x dplyr::lag()         masks stats::lag()
x dplyr::last()        masks data.table::last()
x lubridate::mday()    masks data.table::mday()
x lubridate::minute()  masks data.table::minute()
x lubridate::month()   masks data.table::month()
x lubridate::quarter() masks data.table::quarter()
x lubridate::second()  masks data.table::second()
x purrr::transpose()   masks data.table::transpose()
x lubridate::wday()    masks data.table::wday()
x lubridate::week()    masks data.table::week()
x lubridate::yday()    masks data.table::yday()
x lubridate::year()    masks data.table::year()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become er
```

Не пугайтесь сообщений, все в порядке. Во-первых, пакет `{tidyverse}` – это не просто пакет, а “пакет с пакетами” (да-да, как у вас дома), который подключает сразу несколько других пакетов, которые составляют *ядро tidyverse*. Список и версии этих пакетов `{tidyverse}` выводит при подключении. Разные пакеты *tidyverse* мы очень детально разберем позже (Глава 10), а сейчас просто посмотрите, как это все выглядит.

```
heroes_tbl <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/he
    na = c("NA", "-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
```

```
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Функция `read_csv()` (не путать с функцией из базового R – `read.csv()`!) возвращает **тиббл** – "улучшенный" датафрейм, примерно как это было с дататейблом.

```
  heroes_tbl
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

```
  class(heroes_tbl)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Теперь же сделаем то же самое с нашими данными, что мы делали с помощью {data.table}:

```
heroes_tbl %>%
  filter(Alignment == "good") %>%
  group_by(Gender) %>%
  summarise(mean_height = mean(Height, na.rm = TRUE)) %>%
  arrange(desc(mean_height))
```

```
# A tibble: 3 x 2
  Gender mean_height
  <chr>        <dbl>
1 Male          189.
2 <NA>          180.
3 Female        175.
```

Очень сильно отличается от того, как мы работали раньше! Хотя в основе лежит все тот же R. Код, написанный в *tidyverse*, нарочито многословен (особенно по сравнению с `{data.table}`), каждая отдельная операция имеет свою функцию. Писать нужно больше, зато это гораздо легче: меньше нужно думать, какими хитрыми трюками сделать преобразование данных. Нужно просто разделить весь процесс преобразования данных на отдельные операции и последовательно прописать их. Код получается аккуратный и очень читаемый, даже для человека, который не знает *tidyverse* или даже R в целом. Даже этот новый оператор %>% выглядит довольно понятно: его можно прочитать как “затем”.

Заметьте, что *tidyverse* выводит очень подробные сообщения, которые даже выглядят очень красиво: со всякими иконками, красивым форматированием. Разработчики *tidyverse* работают над тем, чтобы делать свой интерфейс максимально понятным для пользователя: говорящие сами за себя названия функций, куча удобных фишек на все случаи жизни.

tidyverse постоянно обновляется, регулярно появляются новые функции, а старые функции заменяются на более удобные новые. И это не всегда плюс: обновив пакеты, установленные год назад, вы можете обнаружить, что старый код перестал работать! Мол, мы тут придумали, как сделать лучше, переписывайте код заново (или используйте старые версии пакетов).

Разработчики *tidyverse*, в целом, не стремится за высокой скоростью. Часто можно заметить, что новые функции работают довольно медленно. Но если у вас строчек меньше миллиона, то разницу в скорости с `{data.table}` вы едва ли заметите.

Команда разработчиков *tidyverse* работает на компанию *Posit* (бывшая *RStudio*). Поэтому в *RStudio* вы найдете несколько

“шпаргалок” для *tidyverse*, но не для {data.table}. Они также активно активно работают над популяризацией *tidyverse*, стараясь сделать вход в него максимально комфортным, особенно для людей без опыта программирования. *tidyverse* команда открыто заявляет о своей политике diversity, некоторые члены этой команды – открытые представители гендерных и сексуальных меньшинств.

9.3 {data.table} vs tidyverse

Так что же лучше: {data.table} или *tidyverse*? Это один из самых частых споров в R-комьюнити. У обоих подходов есть плюсы, которые можно обсуждать вечно. Сегодня *tidyverse* выигрывает в популярности, особенно за пределами русскоязычного пространства.

В последнее время {data.table} и *tidyverse* все меньше противостоят друг другу и все больше взаимодополняют. Например, некоторые используют в качестве основного инструмента tidyverse, но при работе с данными побольше переключаются на {data.table}[2]. Кроме того, сами разработчики tidyverse пытаются приладить суперскоростной {data.table} в *tidyverse*: пакет {dtplyr} позволяет “переводить” код, написанный в *tidyverse* в код на {data.table}.

Таким образом, выбирая из *tidyverse* и {data.table}, начинать лучше с более удобного и популярного tidyverse, чем и займемся далее.

[2] Автор книги поступает именно так =)

Глава 10

Введение в tidyverse

10.1 Вселенная tidyverse

Тайдиверс *(tidyverse)* - это не один, а целое множество пакетов, объединенных общей философией, грамматикой и структурами данных. Пакеты *tidyverse* можно разделить на две группы:

1. **Основные пакеты,** которые составляют ядро *tidyverse.* Сам по себе пакет `{tidyverse}` – это пакет для подключения и обновления основных пакетов tidyverse. Короче говоря, `{tidyverse}` – это такой пакет с пакетами.

Давайте сначала установим пакет `{tidyverse}`, если он у вас еще не установлен.

```
install.packages("tidyverse")
```

Установка может занять довольно большое время: у вас установятся как основные пакеты *tidyverse,* так и огромное количество зависимостей – других пакетов, которые используются основными пакетами *tidyverse.* Зато при работе с новыми для себя пакетами вы приятно удивитесь тому, как много нужных вам пакетов уже установлено!

Все пакеты *tidyverse,* включая, конечно, основные пакеты, объединены tidy философией и взаимосовместимым синтаксисом. Это означает, что, во многих случаях даже не нужно думать о том, из какого именно пакета *tidyverse* пришла функция. Можно просто подключить один пакет – пакет `{tidyverse}`.

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

Подключение пакета `{tidyverse}` автоматически приводит к подключению основных пакетов tidyverse, дополнительные пакеты нужно подключать дополнительно при необходимости.

Вот эти пакеты:

- `{dplyr}` – для преобразованиия данных, основной пакет всего тайдиверс,
- `{tidyr}` – для приведения данных к **чистому виду *(tidy data)*,**
- `{tibble}` – для работы с тиббламми, продвинутый вариант датафрейма,
- `{purrr}` – для функционального программирования (замена семейства функций `*apply()`; см. Глава 8.5.1),
- `{readr}` – для чтения таблиц в текстовом виде, замена стандартным функциям семейства `read.table()`,
- `{ggplot2}` – для визуализации данных с использованием Grammar of Graphics (Глава 14),
- `{stringr}` – для работы со строковыми переменными,
- `{forcats}` – для работы с переменными-факторами.

В 2023 году к ним добавился еще один пакет:

- `{lubridate}` – для работы с датами и временем.

2. **Дополнительные пакеты** *tidyverse:* они не подключаются при вызове `library(tidyverse)`, но они также разделяют подход *tidyverse*, дополняя и развивая его. Определить границы “расширенного” *tidyverse* очень сложно. Во-первых, есть еще много других небольших пакетов от команды *tidyverse,* которые тоже считаются частью *tidyverse.* Во-вторых, кроме официальных пакетов от команды *tidyverse* есть множество пакетов от других разработчиков, которые пытаются соответствовать принципам *tidyverse* и дополняют их.

***Для продвинутых:* Расширенная Вселенная tidyverse**

Вселенная "расширенного" *tidyverse* необъятна. Перечислить все пакеты не представляется возможным, но можно выделить отдельные группы пакетов.

1. Пакеты для импорта и экспорта данных, работы с базами данных:

- `{vroom}` – для быстрой загрузки таблиц в текстовом виде,
- `{readxl}` – для чтения файлов *Microsoft Excel,*
- `{jsonlite}` – для работы с JSON,
- `{xml2}` – для работы с XML,
- `{DBI}` и `{dbplyr}` – для работы с базами данных,
- `{rvest}` – для веб-скреппинга,
- `{feather}` – для быстрого чтения и записи данных формата *feather,*
- `{googledrive}` – для взаимодействия с файлами на *Google-диске,*
- `{googlesheets4}` – для импорта и экспорта *Google-таблиц.*

2. Пакеты для разработки пакетов: `{rlang}`, `{cli}`, `{crayon}`, `{rstudioapi}`, `{pillar}`.

3. `{tidymodels}` – еще один "пакет с пакетами", который позволяет подключать другие пакеты для моделирования и машинного обучения. Включает себя пакеты `{rsample}`, `{parsnip}`, `{recipes}`, `{tune}`, `{yardstick}`.

Еще несколько важных пакетов из расширенного *tidyverse:*

- `{blob}` – для работы с **большими бинарными объектами *(binary large object; BLOB),***
- `{reprex}` – для создания **воспроизводимых примеров *(reproducible examples)*** – чтобы при написании вопросов в R чатах, stackoverflow, issues на гитхабе описывать проблему таким образом, чтобы отвечающие могли воспроизвести вашу проблему у себя.
- `{glue}` – для продвинутого объединения строк,
- `{tidytext}`– для работы с текстами и корпусами,
- `{magrtittr}` – с несколькими вариантами (%>%) пайп-оператора (Глава 10.3),
- `{dtplyr}` – для ускорения `{dplyr}` за счет перевод синтаксиса на более быстрый `{data.table}` (Глава 9.1).

10.2 Загрузка данных с помощью {readr}

Стандартной функцией для чтения *.csv* файлов в R является функция `read.csv()`, но мы будем использовать функцию `read_csv()` из пакета `readr`. Синтаксис функции `read_csv()` очень похож на `read.csv()`: первым аргументом является путь к файлу (в том числе можно использовать URL), некоторые остальные параметры тоже совпадают.

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
                   na = c("-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Подробнее про импорт данных, в том числе в *tidyverse*, смотри в Глава 6.

10.3 tibble

Когда мы загрузили данные с помощью `read_csv()`, то мы получили объект класса `tibble`, а не `data.frame`:

```
class(heroes)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Тиббл (`tibble`) - это такой "усовершенствованный" `data.frame`. Почти все, что работает с `data.frame`, работает и с тибблами. Однако у тибблов есть свои дополнительные фишки. Самая очевидная из них - более аккуратный вывод в консоль:

```
heroes
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Выводятся только первые 10 строк, если какие-то колонки не влезают на экран, то они просто перечислены внизу. Ну а тип данных написан прямо под названием колонки.

Функции различных пакетов tidyverse сами конвертируют в тиббл при необходимости. Если же нужно это сделать самостоятельно, то можно это сделать так:

```
heroes_df <- as.data.frame(heroes) #
class(heroes_df)
```

```
[1] "data.frame"
```

```
as_tibble(heroes_df) #
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
```

```
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

> В дальнейшем мы будем работать только с tidyverse, а это значит, что только с тибблами, а не обычными датафреймами. Тем не менее, тибблы и датафреймы будут в дальнейшем использоваться как синонимы.

Можно создавать тибблы вручную с помощью функции `tibble()`, которая работает аналогично функции `data.frame()`:

```
tibble(
  a = 1:3,
  b = letters[1:3]
)
```

```
# A tibble: 3 x 2
      a b
  <int> <chr>
1     1 a
2     2 b
3     3 c
```

10.4 magrittr::%>%

Оператор %>% называется “пайпом” (pipe), т.е. “трубой”. Он означает, что следующая функция (справа от пайпа) принимает на вход в качестве первого аргумента результат выполнения предыдущей функции (той, что слева). Фактически, это примерно то же самое, что и вставлять результат выполнения функции в качестве первого аргумента в другую функцию. Просто выглядит это красивее и читабельнее. Как будто данные пропускаются через трубы функций или конвеерную ленту на заводе, если хотите. А то, что первый параметр функции - это почти всегда

данные, работает нам здесь на руку. Этот оператор взят из пакета `magrittr`[1]. Возможно, даже если вы не захотите пользоваться tidyverse, использование пайпов Вам понравится.

Важно понимать, что пайп не дает какой-то дополнительной функциональности или дополнительной скорости работы[2]. Он создан исключительно для читабельности и комфорта.

С помощью пайпов вот эту команду...

```
sum(sqrt(abs(sin(1:22))))
```

```
[1] 16.72656
```

...можно переписать вот так:

```
1:22 %>%
  sin() %>%
  abs() %>%
  sqrt() %>%
  sum()
```

```
[1] 16.72656
```

Время мемов

В очень редких случаях результат выполнения функции нужно вставить не на первую позицию (или же мы хотим использовать

[1] Если быть точным, то оператор %>% был импортирован во все основные пакеты tidyverse, а сам пакет `magrittr` не входит в набор базовых пакетов *tidyverse*. Тем не менее, в самом `magrittr` есть еще несколько интересных операторов.

[2] Даже наоборот, использование пайпов незначительно снижает скорость выполнения команды.

его несколько раз). В этих случаях можно использовать ., чтобы обозначить, куда мы хотим вставить результат выполнения выражения слева от %>%.

```
"       !" %>%
  c("--", ., "--")
```

```
[1] "--"            "       !" "--"
```

Основные функции в tidyverse ...

10.5 Главные пакеты tidyverse: dplyr и tidyr

dplyr[3] — это самая основа всего tidyverse. Этот пакет предоставляет основные функции для манипуляции с тиббламиа. Пакет dplyr является наследником и более усовершенствованной версией plyr, так что если увидите использование пакета plyr, то, скорее всего, скрипт был написан очень давно.

Пакет tidyr дополняет dplyr, предоставляя полезные функции для тайдификации тибблов. Тайдификация ("аккуратизация") данных означает приведение табличных данных к такому формату, в котором:

- Каждая переменная имеет собственный столбец
- Каждый наблюдение имеет собственную строку
- Каждое значение имеет свою собственную ячейку

Впрочем, многие функции dplyr часто используются при тайдификации, так же как и многие функции tidyr имеет применение вне тайдификации. В общем, функционал этих двух пакетов несколько смешался, поэтому мы будем рассматривать их вместе. А чтобы представлять, какая функция относится к какому пакету (хотя запоминать это необязательно), я буду использовать запись с двумя двоеточиями ::, которая обычно используется для использования функции без подгрузки всего пакета, при первом упоминании функции.

[3] Есть споры о том, как это правильно читать. Используемые варианты: "*диплаер*","*диплюр*", "*диплир*", "*диплёр*", "*дипилюр*". Правильный вариант все-таки"диплаер": пакет {dplyr} – это результат развития идей, заложенных в старом пакете Хэдли Уикхэма {plyr}, где ply означает apPLY. {plyr} был попыткой развить идею функций семейства *apply(). Хэдли Уикхэм

Пакет `tidyr` — это более усовершенствованная версия пакета `reshape2`, который в свою очередь является усовершенствованной версией `reshape`. По аналогии с `plyr`, если вы видите использование этих пакетов, то это указывает на то, что перед вами морально устаревший код.

Код с использованием `dplyr` и `tidyr`сильно непохож на то, что мы видели раньше. Большинство функций `dplyr` и `tidyr` работают с целым тиббломом сразу, принимая его в качестве первого аргумента и возвращая измененный тиббл. Это позволяет превратить весь код в последовательный набор применяемых функций, соединенный пайпами. На практике это выглядит очень элегантно, и вы в этом скоро убедитесь.

10.6 Работа с колонками тиббла

10.6.1 Выбор колонок: `dplyr::select()`

Функция `dplyr::select()` позволяет выбирать колонки по номеру или имени (кавычки не нужны).

```
heroes %>%
  select(1,5)
```

```
# A tibble: 734 x 2
    ...1 Race
   <dbl> <chr>
 1     0 Human
 2     1 Icthyo Sapien
 3     2 Ungaran
 4     3 Human / Radiation
 5     4 Cosmic Entity
 6     5 Human
 7     6 <NA>
 8     7 Human
 9     8 <NA>
10     9 Human
# i 724 more rows
```

```
heroes %>%
  select(name, Race, Publisher, `Hair color`)
```

```
# A tibble: 734 x 4
   name          Race              Publisher         `Hair color`
   <chr>         <chr>             <chr>             <chr>
 1 A-Bomb        Human             Marvel Comics     No Hair
 2 Abe Sapien    Icthyo Sapien     Dark Horse Comics No Hair
 3 Abin Sur      Ungaran           DC Comics         No Hair
 4 Abomination   Human / Radiation Marvel Comics     No Hair
 5 Abraxas       Cosmic Entity     Marvel Comics     Black
 6 Absorbing Man Human             Marvel Comics     No Hair
 7 Adam Monroe   <NA>              NBC - Heroes      Blond
 8 Adam Strange  Human             DC Comics         Blond
 9 Agent 13      <NA>              Marvel Comics     Blond
10 Agent Bob     Human             Marvel Comics     Brown
# i 724 more rows
```

Обратите внимание, если в названии колонки присутствует пробел или, например, колонка начинается с цифры или точки и цифры, то это синтаксически невалидное имя (@ref(variables)). Это не значит, что такие названия колонок недопустимы. Но такие названия колонок нужно обособлять ‘ грависом (правый штрих, на клавиатуре находится там же где и буква ё и ~).

Еще обратите внимание на то, что функции tidyverse не изменяют сами изначальные тибблы/датафреймы. Это означает, что если вы хотите полученный результат сохранить, то нужно добавить присвоение:

```
heroes_some_cols <- heroes %>%
  select(name, Race, Publisher, `Hair color`)
heroes_some_cols
```

```
# A tibble: 734 x 4
   name          Race              Publisher         `Hair color`
   <chr>         <chr>             <chr>             <chr>
 1 A-Bomb        Human             Marvel Comics     No Hair
 2 Abe Sapien    Icthyo Sapien     Dark Horse Comics No Hair
 3 Abin Sur      Ungaran           DC Comics         No Hair
 4 Abomination   Human / Radiation Marvel Comics     No Hair
 5 Abraxas       Cosmic Entity     Marvel Comics     Black
 6 Absorbing Man Human             Marvel Comics     No Hair
 7 Adam Monroe   <NA>              NBC - Heroes      Blond
 8 Adam Strange  Human             DC Comics         Blond
 9 Agent 13      <NA>              Marvel Comics     Blond
10 Agent Bob     Human             Marvel Comics     Brown
# i 724 more rows
```

10.6.2 Мини-язык tidyselect для выбора колонок

Для выбора столбцов (не только в `select()`, но и для других функций tidyverse) используется специальный мини-язык tidyselect из одноименного пакета[4]. tidyselect дает очень широкие возможности для выбора колонок.

Можно использовать оператор `:` для выбора нескольких соседних колонок (по аналогии с созданием числового вектора с шагом 1).

```
heroes %>%
  select(name:Publisher)
```

```
# A tibble: 734 x 7
   name          Gender `Eye color` Race           `Hair color` Height Publisher
   <chr>         <chr>  <chr>       <chr>          <chr>         <dbl> <chr>
 1 A-Bomb        Male   yellow      Human          No Hair         203 Marvel C~
 2 Abe Sapien    Male   blue        Icthyo Sapien  No Hair         191 Dark Hor~
 3 Abin Sur      Male   blue        Ungaran        No Hair         185 DC Comics
 4 Abomination   Male   green       Human / Radia~ No Hair         203 Marvel C~
 5 Abraxas       Male   blue        Cosmic Entity  Black            NA Marvel C~
 6 Absorbing Man Male   blue        Human          No Hair         193 Marvel C~
 7 Adam Monroe   Male   blue        <NA>           Blond            NA NBC - He~
 8 Adam Strange  Male   blue        Human          Blond           185 DC Comics
 9 Agent 13      Female blue        <NA>           Blond           173 Marvel C~
10 Agent Bob     Male   brown       Human          Brown           178 Marvel C~
# i 724 more rows
```

```
heroes %>%
  select(name:`Eye color`, Publisher:Weight)
```

```
# A tibble: 734 x 7
   name          Gender `Eye color` Publisher      `Skin color` Alignment Weight
   <chr>         <chr>  <chr>       <chr>          <chr>        <chr>      <dbl>
 1 A-Bomb        Male   yellow      Marvel Comics  <NA>         good         441
 2 Abe Sapien    Male   blue        Dark Horse Co~ blue         good          65
 3 Abin Sur      Male   blue        DC Comics      red          good          90
 4 Abomination   Male   green       Marvel Comics  <NA>         bad          441
 5 Abraxas       Male   blue        Marvel Comics  <NA>         bad           NA
 6 Absorbing Man Male   blue        Marvel Comics  <NA>         bad          122
 7 Adam Monroe   Male   blue        NBC - Heroes   <NA>         good          NA
```

[4] Как и в случае с `magrittr`, пакет `tidyselect` не содержатся в базовом tidyverse, но функции импортируются основными пакетами tidyverse.

```
 8 Adam Strange  Male   blue        DC Comics      <NA>         good          88
 9 Agent 13      Female blue        Marvel Comics  <NA>         good          61
10 Agent Bob     Male   brown       Marvel Comics  <NA>         good          81
# i 724 more rows
```

Используя ! можно вырезать ненужные колонки.

```
heroes %>%
  select(!...1)
```

```
# A tibble: 734 x 10
   name      Gender `Eye color` Race  `Hair color` Height Publisher `Skin color`
   <chr>     <chr>  <chr>       <chr> <chr>         <dbl> <chr>     <chr>
 1 A-Bomb    Male   yellow      Human No Hair         203 Marvel C~ <NA>
 2 Abe Sapi~ Male   blue        Icth~ No Hair         191 Dark Hor~ blue
 3 Abin Sur  Male   blue        Unga~ No Hair         185 DC Comics red
 4 Abominat~ Male   green       Huma~ No Hair         203 Marvel C~ <NA>
 5 Abraxas   Male   blue        Cosm~ Black            NA Marvel C~ <NA>
 6 Absorbin~ Male   blue        Human No Hair         193 Marvel C~ <NA>
 7 Adam Mon~ Male   blue        <NA>  Blond            NA NBC - He~ <NA>
 8 Adam Str~ Male   blue        Human Blond           185 DC Comics <NA>
 9 Agent 13  Female blue        <NA>  Blond           173 Marvel C~ <NA>
10 Agent Bob Male   brown       Human Brown           178 Marvel C~ <NA>
# i 724 more rows
# i 2 more variables: Alignment <chr>, Weight <dbl>
```

```
heroes %>%
  select(!(Gender:Height))
```

```
# A tibble: 734 x 6
    ...1 name          Publisher         `Skin color` Alignment Weight
   <dbl> <chr>         <chr>             <chr>        <chr>      <dbl>
 1     0 A-Bomb        Marvel Comics     <NA>         good         441
 2     1 Abe Sapien    Dark Horse Comics blue         good          65
 3     2 Abin Sur      DC Comics         red          good          90
 4     3 Abomination   Marvel Comics     <NA>         bad          441
 5     4 Abraxas       Marvel Comics     <NA>         bad           NA
 6     5 Absorbing Man Marvel Comics     <NA>         bad          122
 7     6 Adam Monroe   NBC - Heroes      <NA>         good          NA
 8     7 Adam Strange  DC Comics         <NA>         good          88
 9     8 Agent 13      Marvel Comics     <NA>         good          61
10     9 Agent Bob     Marvel Comics     <NA>         good          81
# i 724 more rows
```

Другие известные нам логические операторы (`&` и `|`) тоже работают в tidyselect.

В дополнение к логическим операторам и `:`, в tidyselect есть набор вспомогательных функций, работающих исключительно в контексте выбора колонок с помощью tidyselect.

Вспомогательная функция `last_col()` позволит обратиться к последней колонке тиббла:

```
heroes %>%
  select(name:last_col())
```

```
# A tibble: 734 x 10
   name      Gender `Eye color` Race  `Hair color` Height Publisher `Skin color`
   <chr>     <chr>  <chr>       <chr> <chr>         <dbl> <chr>     <chr>
 1 A-Bomb    Male   yellow      Human No Hair         203 Marvel C~ <NA>
 2 Abe Sapi~ Male   blue        Icth~ No Hair         191 Dark Hor~ blue
 3 Abin Sur  Male   blue        Unga~ No Hair         185 DC Comics red
 4 Abominat~ Male   green       Huma~ No Hair         203 Marvel C~ <NA>
 5 Abraxas   Male   blue        Cosm~ Black            NA Marvel C~ <NA>
 6 Absorbin~ Male   blue        Human No Hair         193 Marvel C~ <NA>
 7 Adam Mon~ Male   blue        <NA>  Blond            NA NBC - He~ <NA>
 8 Adam Str~ Male   blue        Human Blond           185 DC Comics <NA>
 9 Agent 13  Female blue        <NA>  Blond           173 Marvel C~ <NA>
10 Agent Bob Male   brown       Human Brown           178 Marvel C~ <NA>
# i 724 more rows
# i 2 more variables: Alignment <chr>, Weight <dbl>
```

А функция `everything()` позволяет выбрать все колонки.

```
heroes %>%
  select(everything())
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
```

```
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

При этом `everything()` не будет дублировать выбранные колонки, поэтому можно использовать `everything()` для перестановки колонок в тиббле:

```
heroes %>%
  select(name, Publisher, everything())
```

```
# A tibble: 734 x 11
   name          Publisher     ...1 Gender `Eye color` Race  `Hair color` Height
   <chr>         <chr>        <dbl> <chr>  <chr>       <chr> <chr>         <dbl>
 1 A-Bomb        Marvel Comi~     0 Male   yellow      Human No Hair         203
 2 Abe Sapien    Dark Horse ~     1 Male   blue        Icth~ No Hair         191
 3 Abin Sur      DC Comics        2 Male   blue        Unga~ No Hair         185
 4 Abomination   Marvel Comi~     3 Male   green       Huma~ No Hair         203
 5 Abraxas       Marvel Comi~     4 Male   blue        Cosm~ Black            NA
 6 Absorbing Man Marvel Comi~     5 Male   blue        Human No Hair         193
 7 Adam Monroe   NBC - Heroes     6 Male   blue        <NA>  Blond            NA
 8 Adam Strange  DC Comics        7 Male   blue        Human Blond           185
 9 Agent 13      Marvel Comi~     8 Female blue        <NA>  Blond           173
10 Agent Bob     Marvel Comi~     9 Male   brown       Human Brown           178
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Впрочем, для перестановки колонок удобнее использовать специальную функцию `relocate()` (@ref(tidy_relocate)) Можно даже выбирать колонки по паттернам в названиях. Например, с помощью `ends_with()` можно выбрать все колонки, заканчивающиеся одинаковым суффиксом:

```
heroes %>%
  select(ends_with("color"))
```

```
# A tibble: 734 x 3
   `Eye color` `Hair color` `Skin color`
   <chr>       <chr>        <chr>
 1 yellow      No Hair      <NA>
 2 blue        No Hair      blue
 3 blue        No Hair      red
```

```
 4 green       No Hair      <NA>
 5 blue        Black        <NA>
 6 blue        No Hair      <NA>
 7 blue        Blond        <NA>
 8 blue        Blond        <NA>
 9 blue        Blond        <NA>
10 brown       Brown        <NA>
# i 724 more rows
```

Аналогично, с помощью функции `starts_with()` можно найти колонки с одинаковым префиксом, с помощью `contains()` — все колонки с выбранным паттерном в любой части названия колонки[5].

```
heroes %>%
  select(starts_with("Eye") & ends_with("color"))
```

```
# A tibble: 734 x 1
   `Eye color`
   <chr>
 1 yellow
 2 blue
 3 blue
 4 green
 5 blue
 6 blue
 7 blue
 8 blue
 9 blue
10 brown
# i 724 more rows
```

```
heroes %>%
  select(contains("eight"))
```

```
# A tibble: 734 x 2
   Height Weight
    <dbl>  <dbl>
 1    203    441
 2    191     65
 3    185     90
```

[5]Выбранный паттерн будет найден посимвольно, если же вы хотите искать по регулярным выражениям, то вместо `contains()` нужно использовать `matches()`.

```
 4    203    441
 5     NA     NA
 6    193    122
 7     NA     NA
 8    185     88
 9    173     61
10    178     81
# i 724 more rows
```

Ну и наконец, можно выбирать по содержимому колонок с помощью `where()`. Это напоминает применение `sapply()`(@ref(apply_other)) на датафрейме для индексирования колонок: в качестве аргумента для `where` принимается функция, которая применяется для каждой из колонок, после чего выбираются только те колонки, для которых было получено `TRUE`.

```
heroes %>%
  select(where(is.numeric))
```

```
# A tibble: 734 x 3
    ...1 Height Weight
   <dbl>  <dbl>  <dbl>
 1     0    203    441
 2     1    191     65
 3     2    185     90
 4     3    203    441
 5     4     NA     NA
 6     5    193    122
 7     6     NA     NA
 8     7    185     88
 9     8    173     61
10     9    178     81
# i 724 more rows
```

Функция `where()` дает невиданную мощь. Например, можно выбрать все колонки без `NA`:

```
heroes %>%
  select(where(function(x) !any(is.na(x))))
```

```
# A tibble: 734 x 3
    ...1 name          Publisher
   <dbl> <chr>         <chr>
```

```
 1     0 A-Bomb        Marvel Comics
 2     1 Abe Sapien    Dark Horse Comics
 3     2 Abin Sur      DC Comics
 4     3 Abomination   Marvel Comics
 5     4 Abraxas       Marvel Comics
 6     5 Absorbing Man Marvel Comics
 7     6 Adam Monroe   NBC - Heroes
 8     7 Adam Strange  DC Comics
 9     8 Agent 13      Marvel Comics
10     9 Agent Bob     Marvel Comics
# i 724 more rows
```

###Переименование колонок: `dplyr::rename()`

Внутри `select()` можно не только выбирать колонки, но и переименовывать их:

```
  heroes %>%
    select(id = ...1)
```

```
# A tibble: 734 x 1
      id
   <dbl>
 1     0
 2     1
 3     2
 4     3
 5     4
 6     5
 7     6
 8     7
 9     8
10     9
# i 724 more rows
```

Однако удобнее для этого использовать специальную функцию `dplyr::rename()`. Синтаксис у нее такой же, как и у `select()`, но `rename()` не выбрасывает колонки, которые не были упомянуты.

```
  heroes %>%
    rename(id = ...1)
```

```
# A tibble: 734 x 11
      id name          Gender `Eye color` Race     `Hair color` Height Publisher
```

```
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Для массового переименования колонок можно использовать функцию `rename_with()`. Эта функция так же использует tidyselect синтаксис для выбора колонок (по умолчанию выбираются все колонки) и применяет функцию в качестве аргумента, которая изменяет

```
heroes %>%
  rename_with(make.names)
```

```
# A tibble: 734 x 11
    ...1 name      Gender Eye.color Race  Hair.color Height Publisher Skin.color
   <dbl> <chr>     <chr>  <chr>     <chr> <chr>       <dbl> <chr>     <chr>
 1     0 A-Bomb    Male   yellow    Human No Hair       203 Marvel C~ <NA>
 2     1 Abe Sapi~ Male   blue      Icth~ No Hair       191 Dark Hor~ blue
 3     2 Abin Sur  Male   blue      Unga~ No Hair       185 DC Comics red
 4     3 Abominat~ Male   green     Huma~ No Hair       203 Marvel C~ <NA>
 5     4 Abraxas   Male   blue      Cosm~ Black          NA Marvel C~ <NA>
 6     5 Absorbin~ Male   blue      Human No Hair       193 Marvel C~ <NA>
 7     6 Adam Mon~ Male   blue      <NA>  Blond          NA NBC - He~ <NA>
 8     7 Adam Str~ Male   blue      Human Blond         185 DC Comics <NA>
 9     8 Agent 13  Female blue      <NA>  Blond         173 Marvel C~ <NA>
10     9 Agent Bob Male   brown     Human Brown         178 Marvel C~ <NA>
# i 724 more rows
# i 2 more variables: Alignment <chr>, Weight <dbl>
```

###Перестановка колонок: `dplyr::relocate()` {#sec-tidy_relocate}

Для изменения порядка колонок можно использовать функцию `relocate()`. Она тоже работает похожим образом на `select()`

и `rename()`[6]. Как и `rename()`, функция `relocate()` не выкидывает неиспользованные колонки:

```
heroes %>%
  relocate(Publisher)
```

```
# A tibble: 734 x 11
   Publisher          ...1 name     Gender `Eye color` Race  `Hair color` Height
   <chr>             <dbl> <chr>    <chr>  <chr>       <chr> <chr>         <dbl>
 1 Marvel Comics         0 A-Bomb   Male   yellow      Human No Hair         203
 2 Dark Horse Comics     1 Abe Sap~ Male   blue        Icth~ No Hair         191
 3 DC Comics             2 Abin Sur Male   blue        Unga~ No Hair         185
 4 Marvel Comics         3 Abomina~ Male   green       Huma~ No Hair         203
 5 Marvel Comics         4 Abraxas  Male   blue        Cosm~ Black            NA
 6 Marvel Comics         5 Absorbi~ Male   blue        Human No Hair         193
 7 NBC - Heroes          6 Adam Mo~ Male   blue        <NA>  Blond            NA
 8 DC Comics             7 Adam St~ Male   blue        Human Blond           185
 9 Marvel Comics         8 Agent 13 Female blue        <NA>  Blond           173
10 Marvel Comics         9 Agent B~ Male   brown       Human Brown           178
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

При этом `relocate()` имеет дополнительные параметры `.after =` и `.before =`, которые позволяют выбирать, куда поместить выбранные колонки.

```
heroes %>%
  relocate(Publisher, .after = name)
```

```
# A tibble: 734 x 11
    ...1 name          Publisher    Gender `Eye color` Race  `Hair color` Height
   <dbl> <chr>         <chr>        <chr>  <chr>       <chr> <chr>         <dbl>
 1     0 A-Bomb        Marvel Comi~ Male   yellow      Human No Hair         203
 2     1 Abe Sapien    Dark Horse ~ Male   blue        Icth~ No Hair         191
 3     2 Abin Sur      DC Comics    Male   blue        Unga~ No Hair         185
 4     3 Abomination   Marvel Comi~ Male   green       Huma~ No Hair         203
 5     4 Abraxas       Marvel Comi~ Male   blue        Cosm~ Black            NA
 6     5 Absorbing Man Marvel Comi~ Male   blue        Human No Hair         193
 7     6 Adam Monroe   NBC - Heroes Male   blue        <NA>  Blond            NA
 8     7 Adam Strange  DC Comics    Male   blue        Human Blond           185
 9     8 Agent 13      Marvel Comi~ Female blue        <NA>  Blond           173
```

[6] `relocate()` не позволяет переименовывать колонки в отличие от `select()` и `rename()`

```
10     9 Agent Bob     Marvel Comi~ Male   brown       Human Brown           178
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

`relocate()` очень хорошо работает в сочетании с выбором колонок с помощью tidyselect. Например, можно передвинуть в одно место все колонки с одним типом данных:

```
heroes %>%
  relocate(Publisher, where(is.numeric), .after = name)
```

```
# A tibble: 734 x 11
   name      Publisher  ...1 Height Weight Gender `Eye color` Race  `Hair color`
   <chr>     <chr>     <dbl>  <dbl>  <dbl> <chr>  <chr>       <chr> <chr>
 1 A-Bomb    Marvel C~     0    203    441 Male   yellow      Human No Hair
 2 Abe Sapi~ Dark Hor~     1    191     65 Male   blue        Icth~ No Hair
 3 Abin Sur  DC Comics     2    185     90 Male   blue        Unga~ No Hair
 4 Abominat~ Marvel C~     3    203    441 Male   green       Huma~ No Hair
 5 Abraxas   Marvel C~     4     NA     NA Male   blue        Cosm~ Black
 6 Absorbin~ Marvel C~     5    193    122 Male   blue        Human No Hair
 7 Adam Mon~ NBC - He~     6     NA     NA Male   blue        <NA>  Blond
 8 Adam Str~ DC Comics     7    185     88 Male   blue        Human Blond
 9 Agent 13  Marvel C~     8    173     61 Female blue        <NA>  Blond
10 Agent Bob Marvel C~     9    178     81 Male   brown       Human Brown
# i 724 more rows
# i 2 more variables: `Skin color` <chr>, Alignment <chr>
```

Последняя важная функция для выбора колонок — `pull()`. Эта функция делает то же самое, что и индексирование с помощью `$`, т.е. вытаскивает из тиббла вектор с выбранным названием. Это лучше вписывается в логику tidyverse, поскольку позволяет извлечь колонку из тиббла с использованием пайпа:

```
heroes %>%
  select(Height) %>%
  pull() %>%
  head()
```

```
[1] 203 191 185 203  NA 193
```

```
heroes %>%
  pull(Height) %>%
  head()
```

```
[1] 203 191 185 203  NA 193
```

У функции `pull()` есть аргумент `name =`, который позволяет создать проименованный вектор:

```
heroes %>%
  pull(Height, name) %>%
  head()
```

```
       A-Bomb    Abe Sapien      Abin Sur   Abomination       Abraxas
          203           191           185           203            NA
Absorbing Man
          193
```

В отличие от базового R, tidyverse нигде не сокращает имплицитно результат вычислений до вектора, поэтому функция `pull()` - это основной способ извлечения колонки из тиббла как вектора.

10.7 Работа со строками тиббла

10.7.1 Выбор строк по номеру: `dplyr::slice()`

Начнем с выбора строк. Функция `dplyr::slice()` выбирает строчки по их числовому индексу.

```
heroes %>%
  slice(1:3)
```

```
# A tibble: 3 x 11
   ...1 name       Gender `Eye color` Race         `Hair color` Height Publisher
  <dbl> <chr>      <chr>  <chr>       <chr>        <chr>         <dbl> <chr>
1     0 A-Bomb     Male   yellow      Human        No Hair         203 Marvel C~
2     1 Abe Sapien Male   blue        Icthyo Sapi~ No Hair         191 Dark Hor~
3     2 Abin Sur   Male   blue        Ungaran      No Hair         185 DC Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

10.7.2 Выбор строк по условию: `dplyr::filter()`

Функция `dplyr::filter()` делает то же самое, что и `slice()`, но уже по условию. Причем для условий нужно использовать не векторы

из тиббла, а название колонок (без кавычек) как будто бы они были переменными в окружении.

```
heroes %>%
  filter(Publisher == "DC Comics")
```

```
# A tibble: 215 x 11
    ...1 name             Gender `Eye color` Race  `Hair color` Height Publisher
   <dbl> <chr>            <chr>  <chr>       <chr> <chr>         <dbl> <chr>
 1     2 Abin Sur         Male   blue        Unga~ No Hair         185 DC Comics
 2     7 Adam Strange     Male   blue        Human Blond           185 DC Comics
 3    13 Alan Scott       Male   blue        <NA>  Blond           180 DC Comics
 4    16 Alfred Pennywor~ Male   blue        Human Black           178 DC Comics
 5    19 Amazo            Male   red         Andr~ <NA>            257 DC Comics
 6    27 Animal Man       Male   blue        Human Blond           183 DC Comics
 7    31 Anti-Monitor     Male   yellow      God ~ No Hair          61 DC Comics
 8    35 Aquababy         Male   blue        <NA>  Blond            NA DC Comics
 9    36 Aqualad          Male   blue        Atla~ Black           178 DC Comics
10    37 Aquaman          Male   blue        Atla~ Blond           185 DC Comics
# i 205 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

10.7.3 Семейство функций `slice()`

У функции `slice()` есть множество родственников, которые объединяют функционал обычного `slice()` и `filter()`. Например, с помощью функций `dplyr::slice_max()` и `dplyr::slice_min()` можно выбрать заданное количество строк, содержащих наибольшие или наименьшие значения по колонке соответственно:

```
heroes %>%
  slice_max(Weight, n = 3)
```

```
# A tibble: 3 x 11
   ...1 name       Gender `Eye color` Race    `Hair color` Height Publisher
  <dbl> <chr>      <chr>  <chr>       <chr>   <chr>         <dbl> <chr>
1   575 Sasquatch  Male   red         <NA>    Orange          305 Marvel Comics
2   373 Juggernaut Male   blue        Human   Red             287 Marvel Comics
3   203 Darkseid   Male   red         New God No Hair         267 DC Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

```
heroes %>%
  slice_min(Weight, n = 3)
```

```
# A tibble: 3 x 11
   ...1 name        Gender `Eye color` Race        `Hair color` Height Publisher
  <dbl> <chr>       <chr>  <chr>       <chr>       <chr>         <dbl> <chr>
1   346 Iron Monger Male   blue        <NA>        No Hair          NA Marvel C~
2   302 Groot       Male   yellow      Flora Colo~ <NA>            701 Marvel C~
3   350 Jack-Jack   Male   blue        Human       Brown            71 Dark Hor~
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Функция `slice_sample()` позволяет выбирать заданное количество случайных строчек:

```
heroes %>%
  slice_sample(n = 3)
```

```
# A tibble: 3 x 11
   ...1 name        Gender `Eye color` Race        `Hair color` Height Publisher
  <dbl> <chr>       <chr>  <chr>       <chr>       <chr>         <dbl> <chr>
1   139 Buffy       Female green       Human       Blond           157 Dark Hor~
2   304 Guy Gardner Male   blue        Human-Vuld~ Red             188 DC Comics
3   140 Bullseye    Male   blue        Human       blond           183 Marvel C~
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Или же долю строчек:

```
heroes %>%
  slice_sample(prop = .01)
```

```
# A tibble: 7 x 11
   ...1 name        Gender `Eye color` Race  `Hair color` Height Publisher
  <dbl> <chr>       <chr>  <chr>       <chr> <chr>         <dbl> <chr>
1   192 Cy-Gor      Male   <NA>        <NA>  <NA>             NA Image Comics
2   141 Bumblebee   Female brown       Human Black           170 DC Comics
3    71 Battlestar  Male   brown       <NA>  Black           198 Marvel Comics
4   201 Darkhawk    Male   brown       Human Brown           185 Marvel Comics
5   200 Daredevil   Male   blue        Human Red             183 Marvel Comics
6   707 Warlock     Male   red         <NA>  Blond           188 Marvel Comics
7   714 White Queen Female blue        <NA>  Blond           178 Marvel Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Если поставить значение параметра `prop` = равным `1`, то таким образом можно перемешать порядок строчек в тиббле:

```
heroes %>%
  slice_sample(prop = 1)
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color`    Race  `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>          <chr> <chr>         <dbl> <chr>
 1   446 Metamorpho    Male   black          <NA>  No Hair         185 DC Comics
 2   253 Feral         <NA>   yellow (witho~ <NA>  Orange / Wh~    175 Marvel C~
 3   170 Changeling    Male   brown          <NA>  Black           180 Marvel C~
 4    76 Beetle        Male   <NA>           <NA>  <NA>             NA Marvel C~
 5   365 Johann Krauss Male   <NA>           <NA>  <NA>             NA Dark Hor~
 6   470 Moon Knight   Male   brown          Human Brown           188 Marvel C~
 7   272 Galactus      Male   black          Cosm~ Black           876 Marvel C~
 8   422 Machine Man   <NA>   red            <NA>  Black           183 Marvel C~
 9   215 Deathstroke   Male   blue           Human White           193 DC Comics
10   458 Mister Knife  Male   blue           Spar~ Brown            NA Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

10.7.4 Удаление строчек с NA: `tidyr::drop_na()`

Если нужно выбрать только строчки без пропущенных значений, то можно воспользоваться удобной функцией `tidyr::drop_na()`.

```
heroes %>%
  drop_na()
```

```
# A tibble: 50 x 11
    ...1 name       Gender `Eye color` Race        `Hair color` Height Publisher
   <dbl> <chr>      <chr>  <chr>       <chr>       <chr>         <dbl> <chr>
 1     1 Abe Sapien Male   blue        Icthyo Sap~ No Hair         191 Dark Hor~
 2     2 Abin Sur   Male   blue        Ungaran     No Hair         185 DC Comics
 3    34 Apocalypse Male   red         Mutant      Black           213 Marvel C~
 4    39 Archangel  Male   blue        Mutant      Blond           183 Marvel C~
 5    41 Ardina     Female white       Alien       Orange          193 Marvel C~
 6    56 Azazel     Male   yellow      Neyaphem    Black           183 Marvel C~
 7    74 Beast      Male   blue        Mutant      Blue            180 Marvel C~
 8    75 Beast Boy  Male   green       Human       Green           173 DC Comics
 9    92 Bizarro    Male   black       Bizarro     Black           191 DC Comics
10   108 Blackout   Male   red         Demon       White           191 Marvel C~
```

```
# i 40 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Можно выбрать колонки, наличие `NA` в которых будет приводить к удалению соответствующих строчек (не затрагивая другие строчки, в которых есть `NA` в остальных столбцах).

```
heroes %>%
  drop_na(Weight)
```

```
# A tibble: 495 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 6     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 7     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
 8     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
 9    10 Agent Zero    Male   <NA>        <NA>     <NA>            191 Marvel C~
10    11 Air-Walker    Male   blue        <NA>     White           188 Marvel C~
# i 485 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Для выбора колонок в `drop_na()` используется tidyselect, с которым мы недавно познакомились (@ref(tidyselect)).

10.7.5 Сортировка строк: `dplyr::arrange()`

Функция `dplyr::arrange()` сортирует строчки от меньшего к большему (или по алфавиту - для текстовых значений) по выбранной колонке.

```
heroes %>%
  arrange(Weight)
```

```
# A tibble: 734 x 11
    ...1 name           Gender `Eye color` Race   `Hair color` Height Publisher
   <dbl> <chr>          <chr>  <chr>       <chr>  <chr>         <dbl> <chr>
 1   346 Iron Monger    Male   blue        <NA>   No Hair          NA Marvel C~
```

```
 2   302 Groot           Male   yellow      Flora~ <NA>           701 Marvel C~
 3   350 Jack-Jack       Male   blue        Human  Brown           71 Dark Hor~
 4   272 Galactus        Male   black       Cosmi~ Black          876 Marvel C~
 5   731 Yoda            Male   brown       Yoda'~ White           66 George L~
 6   255 Fin Fang Foom   Male   red         Kakar~ No Hair        975 Marvel C~
 7   330 Howard the Duck Male   brown       <NA>   Yellow          79 Marvel C~
 8   396 Krypto          Male   blue        Krypt~ White           64 DC Comics
 9   568 Rocket Raccoon  Male   brown       Animal Brown          122 Marvel C~
10   208 Dash            Male   blue        Human  Blond          122 Dark Hor~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Чтобы отсортировать в обратном порядке, воспользуйтесь функцией `desc()`.

```
heroes %>%
  arrange(desc(Weight))
```

```
# A tibble: 734 x 11
    ...1 name       Gender `Eye color` Race        `Hair color` Height Publisher
   <dbl> <chr>      <chr>  <chr>       <chr>       <chr>         <dbl> <chr>
 1   575 Sasquatch  Male   red         <NA>        Orange        305   Marvel C~
 2   373 Juggernaut Male   blue        Human       Red           287   Marvel C~
 3   203 Darkseid   Male   red         New God     No Hair       267   DC Comics
 4   283 Giganta    Female green       <NA>        Red            62.5 DC Comics
 5   331 Hulk       Male   green       Human / Ra~ Green         244   Marvel C~
 6   549 Red Hulk   Male   yellow      Human / Ra~ Black         213   Marvel C~
 7   119 Bloodaxe   Female blue        Human       Brown         218   Marvel C~
 8   718 Wolfsbane  Female green       <NA>        Auburn        366   Marvel C~
 9   657 Thanos     Male   red         Eternal     No Hair       201   Marvel C~
10     0 A-Bomb     Male   yellow      Human       No Hair       203   Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Можно сортировать по нескольким колонкам сразу. В таких случаях удобно в качестве первой переменной выбирать переменную, обозначающую принадлежность к группе, а в качестве второй — континуальную числовую переменную:

```
heroes %>%
  arrange(Gender, desc(Weight))
```

```
# A tibble: 734 x 11
```

```
    ...1 name      Gender `Eye color` Race         `Hair color` Height Publisher
   <dbl> <chr>     <chr>  <chr>       <chr>        <chr>         <dbl> <chr>
 1   283 Giganta   Female green       <NA>         Red            62.5 DC Comics
 2   119 Bloodaxe  Female blue        Human        Brown         218   Marvel C~
 3   718 Wolfsbane Female green       <NA>         Auburn        366   Marvel C~
 4   591 She-Hulk  Female green       Human        Green         201   Marvel C~
 5   320 Hela      Female green       Asgardian    Black         213   Marvel C~
 6   686 Valkyrie  Female blue        <NA>         Blond         191   Marvel C~
 7   596 Sif       Female blue        Asgardian    Black         188   Marvel C~
 8   271 Frigga    Female blue        <NA>         White         180   Marvel C~
 9   667 Thundra   Female green       <NA>         Red           218   Marvel C~
10   592 She-Thing Female blue        Human / Rad~ No Hair       183   Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

10.8 Создание колонок: `dplyr::mutate()` и `dplyr::transmute()`

Функция `dplyr::mutate()` позволяет создавать новые колонки в тиббле.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  select(name, imt) %>%
  arrange(desc(imt))
```

```
# A tibble: 734 x 2
   name          imt
   <chr>       <dbl>
 1 Utgard-Loki 2510.
 2 Giganta     1613.
 3 Red Hulk     139.
 4 Darkseid     115.
 5 Machine Man  114.
 6 Thanos       110.
 7 Destroyer    108.
 8 A-Bomb       107.
 9 Abomination  107.
10 Hulk         106.
# i 724 more rows
```

`dplyr::transmute()` - это аналог `mutate()`, который не только создает новые колонки, но и сразу же выкидывает все старые:

```
  heroes %>%
    transmute(imt = Weight/(Height/100)^2)
```

```
# A tibble: 734 x 1
     imt
   <dbl>
 1 107.
 2  17.8
 3  26.3
 4 107.
 5  NA
 6  32.8
 7  NA
 8  25.7
 9  20.4
10  25.6
# i 724 more rows
```

Внутри `mutate()` и `transmute()` мы можем использовать либо векторизованные операции (длина новой колонки должна равняться длине датафрейма), либо операции, которые возвращают одно значение. В последнем случае значение будет одинаковым на всю колонку, т.е. будет работать правило ресайклинга (@ref(recycling)):

```
  heroes %>%
    transmute(name, weight_mean = mean(Weight, na.rm = TRUE))
```

```
# A tibble: 734 x 2
   name          weight_mean
   <chr>               <dbl>
 1 A-Bomb               112.
 2 Abe Sapien           112.
 3 Abin Sur             112.
 4 Abomination          112.
 5 Abraxas              112.
 6 Absorbing Man        112.
 7 Adam Monroe          112.
 8 Adam Strange         112.
 9 Agent 13             112.
10 Agent Bob            112.
# i 724 more rows
```

Однако в функциях `mutate()` и `transmute()` правило ресайклинга не будет работать в остальных случаях: если полученный вектор будет не равен 1 или длине датафрейма, то мы получим ошибку.

```
heroes %>%
  mutate(one_and_two = 1:2)
```

```
Error in `mutate()`:
i In argument: `one_and_two = 1:2`.
Caused by error:
! `one_and_two` must be size 734 or 1, not 2.
```

Это не баг, а фича: авторы пакета `dplyr` считают, что ресайклинг кратных друг другу векторов — это слишком удобное место для выстрелов себе в ногу. Поэтому в таких случаях разработчики `dplyr` рекомендуют использовать функцию `rep()`, знакомую нам уже очень давно (@ref(atomic)).

```
heroes %>%
  mutate(one_and_two = rep(1:2, length.out = nrow(.)))
```

```
# A tibble: 734 x 12
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 4 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>,
#   one_and_two <int>
```

10.9 Агрегация данных в тиббле

10.9.1 Подытоживание: `summarise()`

Аггрегация по группам - это очень часто возникающая задача, например, это может использоваться для усреднения данных по испытуемым или условиям. Сделать аггрегацию в датафрейме удобной Хэдли Уикхэм пытался еще в предшественнике `dplyr`, пакете `plyr`. `dplyr` позволяет делать аггрегацию очень симпатичным и понятным способым. Аггрегация в `dplyr` состоит из двух этапов: группировки (`group_by()`) и подытоживания (`summarise()`). Начнем с последнего.

Функция `dplyr::summarise()`[7] позволяет аггрегировать данные в тиббле. Работает она очень похоже на `mutate()`, но если внутри `mutate()` используются векторизованные функции, возвращающие вектор такой же длины, что и колонки, использовавшиеся для расчетов, то в `summarise()` используются функции, которые возвращают вектор длиной 1. Например, `min()`, `mean()`, `max()` и т.д. Можно создавать несколько колонок через запятую (это работает и для `mutate()`).

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  summarise(min(imt, na.rm = TRUE),
            max(imt, na.rm = TRUE))
```

```
# A tibble: 1 x 2
  `min(imt, na.rm = TRUE)` `max(imt, na.rm = TRUE)`
                     <dbl>                    <dbl>
1                   0.0814                    2510.
```

В `dplyr` есть дополнительные суммирующие функции для более удобного индексирования в стиле tidyverse. Например, функции `dplyr::nth()`, `dplyr::first()` и `dplyr::last()`, которые позволяют вытаскивать значения из вектора по индексу (что-то вроде `slice()`, но для векторов)

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  arrange(imt) %>%
```

[7] У функции `dplyr::summarise()` есть синоним `dplyr::summarize()`, которая делает абсолбтно то же самое. Просто потому что в американском английском и британском английском это слово пишется по-разному.

```
    summarise(first = first(imt),
              tenth = nth(imt, 10),
              last = last(imt))
```

```
# A tibble: 1 x 3
   first tenth  last
   <dbl> <dbl> <dbl>
1 0.0814  16.7    NA
```

В отличие от `mutate()`, функции внутри `summarise()` вполне позволяют функциям внутри возвращать вектор из нескольких значений, создавая тиббл такой же длины, как и получившийся вектор.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  summarise(imt_range = range(imt, na.rm = TRUE)) #     range()
```

```
Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
dplyr 1.1.0.
i Please use `reframe()` instead.
i When switching from `summarise()` to `reframe()`, remember that `reframe()`
  always returns an ungrouped data frame and adjust accordingly.
```

```
# A tibble: 2 x 1
  imt_range
      <dbl>
1    0.0814
2 2510.
```

10.9.2 Группировка: `group_by()`

`dplyr::group_by()` - это функция для группировки данных в тиббле по дискретной переменной для дальнейшей аггрегации с помощью `summarise()`. После применения `group_by()` тиббл будет выглядеть так же, но у него появятся атрибут `groups`[8]:

```
heroes %>%
  group_by(Gender)
```

[8] Снять группировку можно с помощью функции `ungroup()`.

```
# A tibble: 734 x 11
# Groups:   Gender [3]
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Если после этого применить на тиббле функцию `summarise()`, то мы получим не тиббл длиной один, а тиббл со значением для каждой из групп.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  group_by(Gender) %>%
  summarise(min(imt, na.rm = TRUE),
            max(imt, na.rm = TRUE))
```

```
# A tibble: 3 x 3
  Gender `min(imt, na.rm = TRUE)` `max(imt, na.rm = TRUE)`
  <chr>                     <dbl>                    <dbl>
1 Female                  15.5                     1613.
2 Male                     0.0814                  2510.
3 <NA>                    16.3                      114.
```

Схематически это выглядит вот так:

10.9.3 Подсчет строк: `dplyr::n()`, `dplyr::count()`

Для подсчет количества значений можно воспользоваться функцией `n()`.

```
heroes %>%
  group_by(Gender) %>%
  summarise(n = n())
```

```
# A tibble: 3 x 2
  Gender     n
  <chr>  <int>
1 Female   200
2 Male     505
3 <NA>      29
```

Функция `n()` вместе с `group_by()` внутри `filter()` позволяет удобным образом "отрезать" от тиббла редкие группы...

```
heroes %>%
  group_by(Race) %>%
  filter(n() > 10) %>%
  select(name, Race)
```

```
# A tibble: 611 x 2
# Groups:   Race [6]
   name          Race
   <chr>         <chr>
 1 A-Bomb        Human
 2 Abomination   Human / Radiation
 3 Absorbing Man Human
 4 Adam Monroe   <NA>
 5 Adam Strange  Human
 6 Agent 13      <NA>
 7 Agent Bob     Human
 8 Agent Zero    <NA>
 9 Air-Walker    <NA>
10 Ajax          Cyborg
# i 601 more rows
```

или же наоборот, выделить только маленькие группы:

```
heroes %>%
  group_by(Race) %>%
  filter(n() == 1) %>%
  select(name, Race)
```

```
# A tibble: 34 x 2
# Groups:   Race [34]
   name          Race
   <chr>         <chr>
 1 Abe Sapien    Icthyo Sapien
 2 Abin Sur      Ungaran
 3 Alien         Xenomorph XX121
 4 Azazel        Neyaphem
 5 Bizarro       Bizarro
 6 Boba Fett     Human / Clone
 7 Darth Maul    Dathomirian Zabrak
 8 Fin Fang Foom Kakarantharaian
 9 Gamora        Zen-Whoberian
10 Gladiator     Strontian
# i 24 more rows
```

Таблицу частот можно создать без `group_by()` и `summarise(n = n())`. Функция `count()` заменяет эту конструкцию:

```
heroes %>%
  count(Gender)
```

```
# A tibble: 3 x 2
  Gender     n
  <chr>  <int>
1 Female   200
2 Male     505
3 <NA>      29
```

Эту таблицу частот удобно сразу проранжировать, указав в параметре `sort` = значение `TRUE`.

```
heroes %>%
  count(Gender, sort = TRUE)
```

```
# A tibble: 3 x 2
  Gender     n
```

```
  <chr>  <int>
1 Male     505
2 Female   200
3 <NA>      29
```

Функция `count()`, несмотря на свою простоту, является одной из наиболее используемых в tidyverse.

10.9.4 Уникальные значения: `dplyr::distinct()`

`dplyr::distinct()` - это более быстрый аналог `unique()`, позволяет извлекать уникальные значения для одной или нескольких колонок.

```
heroes %>%
  distinct(Gender)
```

```
# A tibble: 3 x 1
  Gender
  <chr>
1 Male
2 Female
3 <NA>
```

```
heroes %>%
  distinct(Gender, Race)
```

```
# A tibble: 81 x 2
   Gender Race
   <chr>  <chr>
 1 Male   Human
 2 Male   Icthyo Sapien
 3 Male   Ungaran
 4 Male   Human / Radiation
 5 Male   Cosmic Entity
 6 Male   <NA>
 7 Female <NA>
 8 Male   Cyborg
 9 Male   Xenomorph XX121
10 Male   Android
# i 71 more rows
```

Иногда нужно аггрегировать данные, но при этом сохранить исходную структуру тиббла. Например, нужно посчитать размер групп или посчитать средние значения по группе для последующего сравнения с индивидуальными значениями.

10.9.5 Создание колонок с группировкой

В tidyverse это можно сделать с помощью сочетания `group_by()` и `mutate()` (вместо `summarise()`):

```
heroes %>%
  group_by(Race) %>%
  mutate(Race_n = n()) %>%
  select(Race, name, Gender, Race_n)
```

```
# A tibble: 734 x 4
# Groups:   Race [62]
   Race              name          Gender Race_n
   <chr>             <chr>         <chr>   <int>
 1 Human             A-Bomb        Male      208
 2 Icthyo Sapien     Abe Sapien    Male        1
 3 Ungaran           Abin Sur      Male        1
 4 Human / Radiation Abomination   Male       11
 5 Cosmic Entity     Abraxas       Male        4
 6 Human             Absorbing Man Male      208
 7 <NA>              Adam Monroe   Male      304
 8 Human             Adam Strange  Male      208
 9 <NA>              Agent 13      Female    304
10 Human             Agent Bob     Male      208
# i 724 more rows
```

Результаты аггрегации были записаны в отдельную колонку, при этом значения этой колонки внутри одной группы повторяются:

10.10 Заключение

Мы познакомились с основными принципами и функциями tidyverse. Этих функций, как можно заметить, очень, очень много, каждая из которых посвящена отдельному действию для работы с датафреймами. Чтобы не запутаться во всем многообразии, нужно запомнить самые основные из них:

- `select()‘` – для выбора строк,
- `filter()` – для выбора строк по условию,
- `mutate()` – для создания новых колонок,
- `group_by()` и `summarise()` для аггрегации данных.

Эти функции достаточно гибкие, чтобы их хватало для большинства задач работы с данными. Однако важный принципп tidyverse – это максимальная эксплицитность: если вам нужно, скажем, поменять местами колонки, вы используете специальную функцию `relocate()`, которая предназначено специально для этой задачи.

Такой код очень легко читается, да и тому кто использует tidyverse для написания кода позволяет мыслить более четко.

Глава 11

Продвинутый tidyverse

11.1 Объединение нескольких датафреймов

11.1.1 Соединение структурно схожих датафреймов: bind_rows(), bind_cols()

Для начала подключим `tidyverse` и возьмем уже знакомый нам датасет про супергероев:

```
library("tidyverse")
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become er
```

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
                   na = c("-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Теперь создадим следующие тибблы и сохраним их как `dc`, `marvel` и `other_publishers`:

```
dc <- heroes %>%
  filter(Publisher == "DC Comics") %>%
  group_by(Gender) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE))
dc
```

```
# A tibble: 3 x 2
  Gender weight_mean
  <chr>        <dbl>
1 Female        76.8
2 Male         113.
3 <NA>         NaN
```

```
marvel <- heroes %>%
  filter(Publisher == "Marvel Comics") %>%
  group_by(Gender) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE))
marvel
```

```
# A tibble: 3 x 2
  Gender weight_mean
  <chr>        <dbl>
1 Female        80.1
2 Male         134.
3 <NA>         129.
```

```
other_publishers <- heroes %>%
  filter(!(Publisher %in% c("DC Comics","Marvel Comics"))) %>%
  group_by(Gender) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE))
other_publishers
```

```
# A tibble: 3 x 2
  Gender weight_mean
  <chr>        <dbl>
1 Female        70.8
2 Male         111.
3 <NA>         NaN
```

Несколько тибблов можно объединить вертикально с помощью функции `bind_rows()`. Для корректного объединения тибблы должны иметь одинаковые названия колонок.

```
bind_rows(dc, marvel)
```

```
# A tibble: 6 x 2
  Gender weight_mean
  <chr>        <dbl>
1 Female        76.8
2 Male         113.
3 <NA>         NaN
4 Female        80.1
5 Male         134.
6 <NA>         129.
```

Чтобы соединить тибблы горизонтально, воспользуйтесь функцией `bind_cols()`.

```
bind_cols(dc, marvel)
```

```
New names:
* `Gender` -> `Gender...1`
* `weight_mean` -> `weight_mean...2`
* `Gender` -> `Gender...3`
* `weight_mean` -> `weight_mean...4`
```

```
# A tibble: 3 x 4
  Gender...1 weight_mean...2 Gender...3 weight_mean...4
  <chr>                <dbl> <chr>                <dbl>
1 Female                76.8 Female                80.1
2 Male                 113.  Male                 134.
3 <NA>                 NaN   <NA>                 129.
```

Функции `bind_rows()` и `bind_cols()` могут работать не только с двумя, но сразу с несколькими датафреймами.

```
bind_rows(dc, marvel, other_publishers)
```

```
# A tibble: 9 x 2
  Gender weight_mean
  <chr>        <dbl>
1 Female        76.8
2 Male         113.
3 <NA>         NaN
4 Female        80.1
5 Male         134.
6 <NA>         129.
7 Female        70.8
8 Male         111.
9 <NA>         NaN
```

На входе в функции `bind_rows()` и `bind_cold()` можно подавать как сами датафреймы или тибблы через запятую, так и список из датафреймов/тибблов.

```
heroes_list_of_df <- list(DC = dc,
                          Marvel = marvel,
                          Other = other_publishers)
bind_rows(heroes_list_of_df)
```

```
# A tibble: 9 x 2
  Gender weight_mean
  <chr>        <dbl>
```

```
1 Female        76.8
2 Male         113.
3 <NA>         NaN
4 Female        80.1
5 Male         134.
6 <NA>         129.
7 Female        70.8
8 Male         111.
9 <NA>         NaN
```

Чтобы не потерять, из какого датафрейма какие данные, можно указать любое строковое значение (название будущей колонки) для необязательного аргумента `.id =`.

```
bind_rows(heroes_list_of_df, .id = "Publisher")
```

```
# A tibble: 9 x 3
  Publisher Gender weight_mean
  <chr>     <chr>        <dbl>
1 DC        Female        76.8
2 DC        Male         113.
3 DC        <NA>         NaN
4 Marvel    Female        80.1
5 Marvel    Male         134.
6 Marvel    <NA>         129.
7 Other     Female        70.8
8 Other     Male         111.
9 Other     <NA>         NaN
```

`bind_rows()` обычно используется, когда ваши данные находятся в разных файлах с одинаковой структурой. Тогда вы можете прочитать все таблицы в папке, сохранить их в качестве списка из датафреймов и объединить в один датафрейм с помощью `bind_rows()`.

11.1.2 Реляционные данные: `*_join()`

В реальности иногда возникает ситуация, когда нужно соединить две таблички, у которых есть общий столбец (или несколько столбцов), но все остальные столбцы различаются. Табличек может быть и больше, это может быть целая сеть таблиц, некоторые из которых содержат основные данные, а некоторые - дополнительные, которые необходимо на определенном

этапе “включить” в анализ. Например, таблица с расшифровкой аббревиатур или сокращений вроде коротких названий стран или таблица телефонных кодов разных стран. Совокупность нескольких связанных друг с другом таблиц называют реляционными данными.

В случае с реляционными данными простых `bind_rows()` и `bind_cols()` становится недостаточно.

Эти две таблички нужно объединить ***(join).*** Эта задача обычно возникает не очень часто, обычно это происходит один-два раза в одном проекте, когда нужно дополнить имеющиеся данные дополнительной информацией извне или объединить два набора данных, обрабатывавшихся в разных программах. Однако каждый раз, когда такая задача возникает, это доставляет много боли. `dplyr` предлагает интуитивно понятный инструмент для объединения реляционных данных - семейство функций `*_join()`.

Возьмем для примера два тиббла `band_members` и `band_instruments`, встроенных в `dplyr` специально для демонстрации работы функций `*_join()`.

```
band_members
```

```
# A tibble: 3 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

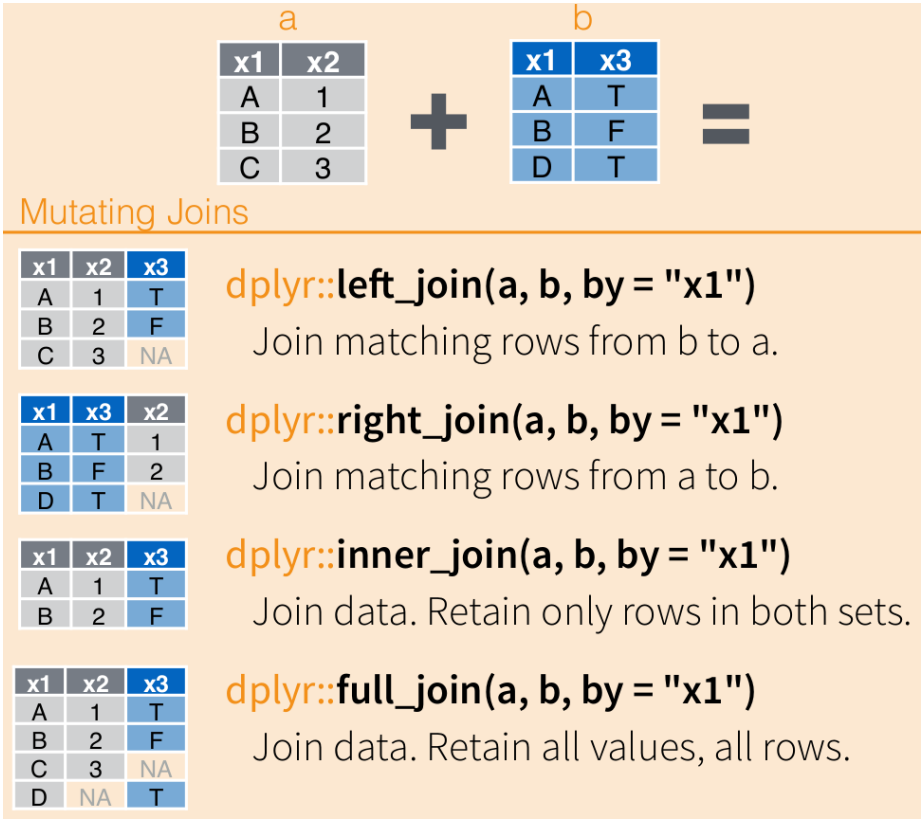
```
band_instruments
```

```
# A tibble: 3 x 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```

У этих двух тибблов есть колонка с одинаковым названием, которая по своему смыслу соединяет данные обоих тибблов.

Такая колонка называется **ключом**. Ключ должен однозначно идентифицировать наблюдения[1].

Давайте попробуем посоединять `band_members` и `band_instruments` разными вариантами `*_join()` и посмотрим, что у нас получится. Все эти функции имеют на входе два обязательных аргумента (`x =` и `y =`) в которые мы должны подставить два датафрейма/тиббла которые мы хотим объединить. Главное различие между этими функциями заключается в том, что они будут делать, если уникальные значения в ключах `x` и `y` не соответствуют друг другу.



- `left_join()`:

[1] Если ключи будут неуникальными, то функции `*_join()` не будут выдавать ошибку. Вместо этого они добавят в итоговую таблицу все возможные пересечения повторяющихся ключей. С этим нужно быть очень осторожным, поэтому рекомендуется, во-первых, проверять уникальность ключей на входе и, во-вторых, проверять тиббл на выходе. Ну или использовать эту особенность работы функции `*_join()` себе во благо.

```
band_members %>%
  left_join(band_instruments)
```

```
Joining with `by = join_by(name)`
```

```
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
```

`left_join()` - это самая простая для понимания и самая используемая функция из семейства `*_join()`. Она как бы “дополняет” информацию из первого тиббла вторым тибблом. В этом случае сохраняются все уникальные наблюдения в `x`, но отбрасываются лишние наблюдения в тиббле `y`. Тем значениям, которым не нашлось соотвествия в `y`, в колонках, взятых их `y`, ставятся значения `NA`.

Вы можете сами задать колонки-ключи параметром `by =`, по умолчанию это все колонки с одинаковыми названиями в двух тибблах.

```
band_members %>%
  left_join(band_instruments, by = "name")
```

```
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
```

Часто случается, что колонки-ключи называются по-разному в двух тибблах. Их необязательно переименовывать, можно поставить соответстие вручную используя проименованный вектор:

```
band_members %>%
  left_join(band_instruments2, by = c("name" = "artist"))
```

```
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
```

- `right_join()`:

```
band_members %>%
  right_join(band_instruments)
```

```
Joining with `by = join_by(name)`
```

```
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
3 Keith <NA>    guitar
```

`right_join()` отбрасывает строчки в `x`, которых не было в `y`, но сохраняет соответствующие строчки `y` - `left_join()` наоборот.

- `full_join()`:

```
band_members %>%
  full_join(band_instruments)
```

```
Joining with `by = join_by(name)`
```

```
# A tibble: 4 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```

Функция `full_join()` сохраняет все строчки и из `x` и `y`. Пожалуй, наиболее используемая функция после `left_join()` – благодаря `full_join()` вы точно ничего не потеряете при объединении.

- `inner_join()`:

```
band_members %>%
  inner_join(band_instruments)
```

```
Joining with `by = join_by(name)`

# A tibble: 2 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
```

Функция `inner_join()` сохраняет только строчки, которые присутствуют и в `x`, и в `y`.

- `semi_join()`:

```
band_members %>%
  semi_join(band_instruments)
```

```
Joining with `by = join_by(name)`

# A tibble: 2 x 2
  name  band
  <chr> <chr>
1 John  Beatles
2 Paul  Beatles
```

- `anti_join()`:

```
band_members %>%
  anti_join(band_instruments)
```

```
Joining with `by = join_by(name)`

# A tibble: 1 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
```

Функции `semi_join()` и `anti_join()` не присоединяют второй датафрейм/тиббл (`y`) к первому. Вместо этого они используются как некоторый словарь-фильтр для отделения только тех значений в `x`, которые есть в `y` (`semi_join()`) или, наоборот, которых нет в `y` (`anti_join()`).

11.2 Tidy data: широкий и длинный форматы данных

Принцип ***tidy data*** предполагает, что каждая строка содержит в себе одно измерение, каждая колонка - одну характеристику, а в одной ячейке только одно значение. Тем не менее, это не говорит однозначно о том, как именно хранить повторные измерения. Их можно хранить в **широком формате *(wide data)*** как одну колонку для каждого измерения. Одной строчке соответствует один объект измерения. Если измерений много, то такой датафрейм может получиться очень широким.

Другой способ хранения данных в датафрейме – **длинный формат *(long data).*** В этом случае создаются две колонки: одна колонка - для идентификатора измерения (например, время измерения), другая колонка - для записи самого измерения. Каждая строка соответствует одному измерению объекта, а объект измерения имеет несколько строк в датафрейме. Если измерений много, то датафрейм становится очень длинным.

Это лучше пояснить на примере. Например, измерим массу студентов до и после прохождения курса по R. Как это лучше записать - **в широком формате,** как два числовых столбца (один – измерение до, второй – измерение после)? Или же **в длинном формате:** создать строковую колонку, в которой будет написано время измерения (“До курса по R”, “После курса по R”) и числовую колонку со значением массы?

- **Широкий формат:**

Студент	До курса по R	После курса по R
Маша	70	63
Рома	80	74
Антонина	86	71

- **Длинный формат:**

Студент	Время измерения	Масса (кг)
Маша	До курса по R	70
Рома	До курса по R	80
Антонина	До курса по R	86

Студент	Время измерения	Масса (кг)
Маша	После курса по R	63
Рома	После курса по R	74
Антонина	После курса по R	71

На самом деле, оба варианта приемлемы, оба варианта встречаются в реальных данных, а разные функции и статистические пакеты могут требовать от вас как **длинный,** так и **широкий форматы.**

Таким образом, нам нужно научиться переводить из широкого формата в длинный и наоборот. Для этого в tidyverse есть функции:

- `tidyr::pivot_longer()`: переводит из **широкого** в **длинный формат,**
- `tidyr::pivot_wider()`: переводит из **длинного** в **широкий формат.**

К сожалению, в PDF нельзя вставить .gif анимацию, посмотреть ее можно по ссылке

```
new_diet <- tibble(
  student = c("  ", "  ", "    "),
  before_r_course = c(70, 80, 86),
  after_r_course = c(63, 74, 71)
)
new_diet
```

```
# A tibble: 3 x 3
  student  before_r_course after_r_course
  <chr>              <dbl>          <dbl>
1                       70             63
2                       80             74
3                      86             71
```

Тиббл `new_diet` - это пример широкого формата данных.

Превратим тиббл `new_diet` длинный:

```
new_diet %>%
  pivot_longer(cols = before_r_course:after_r_course,
```

```
                names_to = "measurement_time",
                values_to = "weight_kg")
```

```
# A tibble: 6 x 3
  student  measurement_time weight_kg
  <chr>    <chr>                <dbl>
1        before_r_course         70
2        after_r_course          63
3        before_r_course         80
4        after_r_course          74
5      before_r_course          86
6      after_r_course           71
```

А теперь обратно в короткий:

```
new_diet %>%
  pivot_longer(cols = before_r_course:after_r_course,
               names_to = "measurement_time",
               values_to = "weight_kg") %>%
  pivot_wider(names_from = "measurement_time",
              values_from = "weight_kg")
```

```
# A tibble: 3 x 3
  student  before_r_course after_r_course
  <chr>              <dbl>          <dbl>
1                     70             63
2                     80             74
3                    86             71
```

11.3 Трансформация нескольких колонок: `dplyr::across()`

Допустим, вы хотите посчитать среднюю массу и рост, группируя по полу супергероев. Можно посчитать это внутри одного `summarise()`, используя запятую:

```
heroes %>%
  group_by(Gender) %>%
  summarise(height = mean(Height, na.rm = TRUE),
            weight = mean(Weight, na.rm = TRUE))
```

```
# A tibble: 3 x 3
  Gender height weight
  <chr>   <dbl>  <dbl>
1 Female   175.   78.8
2 Male     192.  126.
3 <NA>     177.  129.
```

Если таких колонок будет много, то это уже станет сильно неудобным, нам придется много копировать код, а это чревато ошибками и очень скучно.

Поэтому в `dplyr` есть функция для операций над несколькими колонками сразу: `dplyr::across()`[2]. Эта функция работает похожим образом на функции семейства `apply()` и использует tidyselect для выбора колонок.

Таким образом, конструкции с функцией `across()` можно разбить на три части:

1. Выбор колонок с помощью tidyselect. Здесь работают все те приемы, которые мы изучили при выборе колонок (Глава 10.6.2).
2. Собственно применение функции `across()`. Первый аргумент `.col` – колонки, выбранные на первом этапе с помощью tidyselect, по умолчанию это `everything()`, т.е. все колонки. Второй аргумент `.fns` – это функция или целый список из функций, которые будут применены к выбранным колонкам. Если функции требуют дополнительных аргументов, то они могут быть перечислены внутри `across()`.
3. Использование `summarise()` или другой функции `dplyr`. В этом случае в качестве аргумента для функции используется результат работы функции `across()`.

Вот такой вот бутерброд выходит. Давайте посмотрим, как это работает на практике и посчитаем среднее значение по колонкам `Height` и `Weight`.

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height,Weight), mean))
```

[2]Функция `across()` появилась в пакете `dplyr` относительно недавно, до этого для работы с множественными колонками в tidyverse использовались многочисленные функции `*_at()`, `*_if()`, `*_all()`, например, `summarise_at()`, `summarise_if()`, `summarize_all()`. Эти функции до сих пор присутствуют в `dplyr`, но считаются устаревшими. Другая альтернатива - использование пакета `purrr` (Глава 11.4) или семейства функций `apply()` (Глава 8.5).

```
# A tibble: 3 x 3
  Gender Height Weight
  <chr>   <dbl>  <dbl>
1 Female     NA     NA
2 Male       NA     NA
3 <NA>       NA     NA
```

Здесь мы столкнулись с уже известной нам проблемой: функция `mean()` при столкновении хотя бы с одним `NA` будет возвращать `NA`, если мы не изменим параметр `na.rm =`. Как и в случае с функциями семейства `apply()` (Глава 8.5), дополнительные параметры для функции можно перечислить через запятую после самой функции:

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height, Weight), mean, na.rm = TRUE))
```

```
Warning: There was 1 warning in `summarise()`.
i In argument: `across(c(Height, Weight), mean, na.rm = TRUE)`.
i In group 1: `Gender = "Female"`.
Caused by warning:
! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
Supply arguments directly to `.fns` through an anonymous function instead.

  # Previously
  across(a:b, mean, na.rm = TRUE)

  # Now
  across(a:b, \(x) mean(x, na.rm = TRUE))
```

```
# A tibble: 3 x 3
  Gender Height Weight
  <chr>   <dbl>  <dbl>
1 Female   175.   78.8
2 Male     192.  126.
3 <NA>     177.  129.
```

До этого мы просто использовали выбор колонок по их названию. Но именно внутри `across()` использование tidyselect раскрывается как удивительно элегантный и мощный инструмент. Например, можно посчитать среднее для всех numeric колонок:

```
heroes %>%
  drop_na(Height, Weight) %>%
  group_by(Gender) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE))
```

```
# A tibble: 3 x 4
  Gender  ...1 Height Weight
  <chr>  <dbl>  <dbl>  <dbl>
1 Female  394.   174.   78.3
2 Male    369.   193.  126.
3 <NA>    375.   182   129.
```

Или длину строк для строковых колонок. Для этого нам понадобится вспомнить, как создавать анонимные функции (@ref(anon_f)).

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(where(is.character),
                   function(x) mean(nchar(x), na.rm = TRUE)))
```

```
# A tibble: 3 x 8
  Gender  name `Eye color`  Race `Hair color` Publisher `Skin color` Alignment
  <chr>  <dbl>       <dbl> <dbl>        <dbl>     <dbl>        <dbl>     <dbl>
1 Female  9.04        4.68  6.42         5.05      11.5         4.57      3.88
2 Male    9.05        4.53  6.75         5.48      11.4         5.02      3.78
3 <NA>    9.48        5.16 10.1          6.44      11.9         4         3.96
```

Или же даже посчитать и то, и другое внутри одного `summarise()`!

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE),
            across(where(is.character),
                   function(x) mean(nchar(x), na.rm = TRUE)))
```

```
# A tibble: 3 x 11
  Gender  ...1 Height Weight  name `Eye color`  Race `Hair color` Publisher
  <chr>  <dbl>  <dbl>  <dbl> <dbl>       <dbl> <dbl>        <dbl>     <dbl>
1 Female  395.   175.   78.8  9.04        4.68  6.42         5.05      11.5
2 Male    357.   192.  126.   9.05        4.53  6.75         5.48      11.4
3 <NA>    329    177.  129.   9.48        5.16 10.1          6.44      11.9
# i 2 more variables: `Skin color` <dbl>, Alignment <dbl>
```

Внутри одного `across()` можно применить не одну функцию к каждой из выбранных колонок, а сразу несколько функций для каждой из колонок. Для этого нам нужно использовать список функций (желательно - проименованный).

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height, Weight),
                   list(minimum = min,
                        average = mean,
                        maximum = max),
                   na.rm = TRUE))
```

```
# A tibble: 3 x 7
  Gender Height_minimum Height_average Height_maximum Weight_minimum
  <chr>           <dbl>          <dbl>          <dbl>          <dbl>
1 Female           62.5           175.            366             41
2 Male             15.2           192.            975              2
3 <NA>            108             177.            198             39
# i 2 more variables: Weight_average <dbl>, Weight_maximum <dbl>
```

Вот нам и понадобился список функций (@ref(functions_objects))!

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height, Weight),
                   list(min = function(x) min(x, na.rm = TRUE),
                        mean = function(x) mean(x, na.rm = TRUE),
                        max = function(x) max(x, na.rm = TRUE),
                        na_n = function(x, ...) sum(is.na(x)))
                   )
            )
```

```
# A tibble: 3 x 9
  Gender Height_min Height_mean Height_max Height_na_n Weight_min Weight_mean
  <chr>       <dbl>       <dbl>      <dbl>       <int>      <dbl>       <dbl>
1 Female       62.5        175.        366          56         41        78.8
2 Male         15.2        192.        975         147          2       126.
3 <NA>        108          177.        198          14         39       129.
# i 2 more variables: Weight_max <dbl>, Weight_na_n <int>
```

Хотя основное применение функции `across()` – это массовое подытоживание с помощью `summarise()`, `across()` можно использовать

и с другими функциями `dplyr`. Например, можно делать массовые операции с колонками с помощью `mutate()`:

```
heroes %>%
  mutate(across(where(is.character), as.factor))
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <fct>         <fct>  <fct>       <fct>    <fct>         <dbl> <fct>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <fct>, Alignment <fct>, Weight <dbl>
```

Конструкция `across()` работает не только внутри `summarise()` и `mutate()`, можно применять `across()` и с другими функциями, которые используют data-masking. Например, можно использовать `across()` внутри `count()` вместе с функцией `n_distinct()`, которая считает количество уникальных значений в векторе. Это позволяет посмотреть таблицу частот для группирующих переменных:

```
heroes %>%
  count(across(where(function(x) n_distinct(x) <= 6)))
```

```
# A tibble: 11 x 3
   Gender Alignment     n
   <chr>  <chr>     <int>
 1 Female bad          35
 2 Female good        161
 3 Female neutral       4
 4 Male   bad         165
 5 Male   good        316
 6 Male   neutral      18
 7 Male   <NA>          6
 8 <NA>   bad           7
 9 <NA>   good         19
```

```
10 <NA>   neutral       2
11 <NA>   <NA>          1
```

11.4 Функциональное программирование: purrr

`purrr` – это пакет для функционального программирования в tidyverse. Как и многие пакеты tidyverse, `purrr` пытается заменить собой базовый функционал R на более понятный и удобный. В данном случае, речь в первую очередь идет о функциях семейства `apply()`, с которыми мы работали ранее (@ref(apply_f)).

Давайте вспомним, как работает `lapply()`. В качестве первого аргумента функция `lapply()` принимает список (или то, что может быть в него превращено, например, датафрейм), в качестве второго - функцию, которая будет применена к каждому элементу списка. На выходе мы получим список такой же длины.

```
lapply(heroes, class)
```

```
$...1
[1] "numeric"

$name
[1] "character"

$Gender
[1] "character"

$`Eye color`
[1] "character"

$Race
[1] "character"

$`Hair color`
[1] "character"

$Height
[1] "numeric"

$Publisher
[1] "character"
```

```
$`Skin color`
[1] "character"

$Alignment
[1] "character"

$Weight
[1] "numeric"
```

Функция `purrr::map()` работает по тому же принципу: можно просто заменить `lapply()` на `map()`, и мы получим тот же результат.

```
map(heroes, class)
```

```
$...1
[1] "numeric"

$name
[1] "character"

$Gender
[1] "character"

$`Eye color`
[1] "character"

$Race
[1] "character"

$`Hair color`
[1] "character"

$Height
[1] "numeric"

$Publisher
[1] "character"

$`Skin color`
[1] "character"

$Alignment
[1] "character"
```

```
$Weight
[1] "numeric"
```

`map()` можно встроить в канал с пайпом (впрочем, как и `lapply()`):

```
  heroes %>%
    map(class)
```

```
$...1
[1] "numeric"

$name
[1] "character"

$Gender
[1] "character"

$`Eye color`
[1] "character"

$Race
[1] "character"

$`Hair color`
[1] "character"

$Height
[1] "numeric"

$Publisher
[1] "character"

$`Skin color`
[1] "character"

$Alignment
[1] "character"

$Weight
[1] "numeric"
```

Как и `lapply()`, `map()` всегда возвращает список. Из-за этого мы больше пользовались функцией `sapply()`, а не `lapply()`. Функция `sapply()` упрощала результат до вектора, если это возможно.

Подобное упрощение может показаться удобным пока не сталкиваешься с тем, что иногда очень сложно предсказать, какой тип данных получится на выходе. Есть функция `vapply()` в которой можно управлять типом данных на выходе, но она не очень удобная. В `purrr` эта проблема решена просто: есть множество функций `map_*()`, где вместо звездочки - нужный формат на выходе.

Например, если мы хотим получить строковый вектор на выходе, то нам нужна функция `map_chr()`.

```
heroes %>%
  map_chr(class)
```

```
      ...1        name      Gender   Eye color        Race  Hair color
 "numeric" "character" "character" "character" "character" "character"
    Height   Publisher  Skin color   Alignment      Weight
 "numeric" "character" "character" "character"   "numeric"
```

Можно превратить результат в датафрейм с помощью `map_df()`.

```
heroes %>%
  map_df(class)
```

```
# A tibble: 1 x 11
  ...1    name      Gender    `Eye color` Race     `Hair color` Height Publisher
  <chr>   <chr>     <chr>     <chr>       <chr>    <chr>        <chr>  <chr>
1 numeric character character character   charact~ character    numer~ character
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <chr>
```

Так же как и функции семейства `apply()`, функции `map_*()` отлично сочетаются с анонимными функциями.

```
heroes %>%
  map_int(function(x) sum(is.na(x)))
```

```
      ...1       name     Gender  Eye color       Race Hair color     Height
         0          0         29        172        304        172        217
 Publisher Skin color  Alignment     Weight
         0        662          7        239
```

Однако у `purrr` есть свой, более короткий способ записи анонимных функций: `function(arg)` заменяется на `~`, а `arg` на `..`

```
heroes %>%
  map_int(~sum(is.na(.)))
```

```
      ...1       name     Gender  Eye color       Race Hair color     Height
         0          0         29        172        304        172        217
 Publisher Skin color  Alignment     Weight
         0        662          7        239
```

Если нужно итерироваться сразу по нескольким спискам, то есть функции `map2_*()` (для двух списков) и `pmap_*()` (для нескольких списков).

11.5 Колонки-списки и нестинг: `nest()`

Ранее мы говорили о том, что датафрейм – это по своей сути список из векторов разной длины. На самом деле, это не совсем так: колонки обычного датафрейма вполне могут быть списками. Однако делать так обычно не рекомендуется, пусть R это и не запрещает создавать такие колонки: многие функции предполагают, что все колонки датафрейма являются векторами.

`tidyverse` гораздо дружелюбнее относится к использовании списка в качестве колонки. Такие колонки называются **колонками-списками (list columns)**. Основной способ их создания - использования функции `tidyr::nest()`. С помощью tidyselect нужно выбрать сжимаемые колонки, которые будут агрегированы по невыбранным колонками. Это и называется нестингом.

```
heroes %>%
  nest(!Gender)
```

```
Warning: Supplying `...` without names was deprecated in tidyr 1.0.0.
i Please specify a name for each selection.
i Did you want `data = !Gender`?
```

```
# A tibble: 3 x 2
  Gender data
  <chr>  <list>
1 Male   <tibble [505 x 10]>
2 Female <tibble [200 x 10]>
3 <NA>   <tibble [29 x 10]>
```

Заметьте, у нас появилась колонка `data`, в которой содержатся тибблы. Туда и спрятались все наши данные.

Нестинг похож на агрегирование с помощью `group_by()`. Если сделать нестинг сгруппированного с помощью `group_by()` тиббла, то сожмутся все колонки кроме тех, которые выступают в качестве групп:

```
heroes %>%
  group_by(Gender) %>%
  nest()
```

```
# A tibble: 3 x 2
# Groups:   Gender [3]
  Gender data
  <chr>  <list>
1 Male   <tibble [505 x 10]>
2 Female <tibble [200 x 10]>
3 <NA>   <tibble [29 x 10]>
```

Теперь можно работать с колонкой-списком как с обычной колонкой. Например, применять функцию для каждой строчки (то есть для каждого тиббла) с помощью `map()` и записывать результат в новую колонку с помощью `mutate()`.

```
heroes %>%
  group_by(Gender) %>%
  nest() %>%
  mutate(dim = map(data, dim))
```

```
# A tibble: 3 x 3
# Groups:   Gender [3]
  Gender data                dim
  <chr>  <list>              <list>
1 Male   <tibble [505 x 10]> <int [2]>
2 Female <tibble [200 x 10]> <int [2]>
3 <NA>   <tibble [29 x 10]>  <int [2]>
```

В конце концов нам нужно “разжать” сжатую колонку-список. Сделать это можно с помощью `unnest()`, выбрав с помощью tidyselect нужные колонки.

```
heroes %>%
  group_by(Gender) %>%
  nest() %>%
  mutate(dim = map(data, dim)) %>%
  unnest(dim)
```

```
# A tibble: 6 x 3
# Groups:   Gender [3]
  Gender data                  dim
  <chr>  <list>              <int>
1 Male   <tibble [505 x 10]>   505
2 Male   <tibble [505 x 10]>    10
3 Female <tibble [200 x 10]>   200
4 Female <tibble [200 x 10]>    10
5 <NA>   <tibble [29 x 10]>     29
6 <NA>   <tibble [29 x 10]>     10
```

Разжатая колонка обычно больше сжатой, поэтому разжатие привело к удлинению тиббла. Вместо удлинения тиббла, его можно расширить с помощью `unnest_wider()`.

```
heroes %>%
  group_by(Gender) %>%
  nest() %>%
  mutate(dim = map(data, dim)) %>%
  unnest_wider(dim, names_sep = "_")
```

```
# A tibble: 3 x 4
# Groups:   Gender [3]
  Gender data                dim_1 dim_2
  <chr>  <list>              <int> <int>
1 Male   <tibble [505 x 10]>   505    10
2 Female <tibble [200 x 10]>   200    10
3 <NA>   <tibble [29 x 10]>     29    10
```

Нестинг - это мощный инструмент tidyverse, хотя во многих случаях можно обойтись и без него. Наиболее эффективна эта конструкция именно в тех ситуациях, где вы делаете операции над целыми тиббламы. Поэтому наибольшее распространение нестинг получил в смычке с пакетом `broom` для расчета множественных статистических моделей.

Другое применение нестинга – решение проблемы с несколькими значениями в одной ячейки, которые записаны через запятую или

какой-либо другой знак. Такое часто встречается в данных, поэтому хорошо бы уметь с этим работать!

Возьмем небольшой искусственный пример:

```
films <- tribble(
  ~film, ~genres,
  "       ", "comedy, drama",
  "         ", "comedy, criminal",
  "   ", "fantasy, drama"
)

films
```

```
# A tibble: 3 x 2
  film             genres
  <chr>            <chr>
1            comedy, drama
2          comedy, criminal
3              fantasy, drama
```

Для этого разобъем значения колонки `genres` с помощью встроенной функции `strsplit()`. Она разбивает значения вектора по выбранному разделителю. Здесь нам нужен разделитель `", "` (с пробелом после запятой!). На выходе мы получим список такой же длины, что и исходный вектором, а каждый элемент этого списка будет строковым вектором. Количество значений внутри векторов может быть каким угодно. Поскольку результат – список, перезаписанная колонка `genres` станет колонкой-списком.

```
films %>%
  mutate(genres = strsplit(genres, ", "))
```

```
# A tibble: 3 x 2
  film             genres
  <chr>            <list>
1            <chr [2]>
2          <chr [2]>
3              <chr [2]>
```

Теперь нам нужно сделать `unnest()`

```
films %>%
  mutate(genres = strsplit(genres, ", ")) %>%
  unnest()
```

```
Warning: `cols` is now required when using `unnest()`.
i Please use `cols = c(genres)`.

# A tibble: 6 x 2
  film             genres
  <chr>            <chr>
1            comedy
2            drama
3          comedy
4          criminal
5              fantasy
6              drama
```

Теперь у нас данные в длинном виде! Результат можно расширить с помощью уже знакомого `pivot_wider()` и дополнительной колонки со значениями `TRUE`. Если соответствующей пары нет в тиббле, то в итоговой широкой таблице будет `NA`, мы можем поменять их на `FALSE` с помощью параметра `values_fill =`.

```
films %>%
  mutate(genres = strsplit(genres, ", ")) %>%
  unnest() %>%
  mutate(value = TRUE) %>%
  pivot_wider(names_from = "genres",
              values_from = "value", values_fill = FALSE)
```

```
Warning: `cols` is now required when using `unnest()`.
i Please use `cols = c(genres)`.

# A tibble: 3 x 5
  film             comedy drama criminal fantasy
  <chr>            <lgl>  <lgl> <lgl>    <lgl>
1            TRUE   TRUE  FALSE    FALSE
2          TRUE   FALSE TRUE     FALSE
3              FALSE  TRUE  FALSE    TRUE
```

Правда, то же самое можно сделать чуть проще, без колонок списков. Специально для этой задачи есть функция `tidyr::separate_rows()`, которая заменяет связку `strsplit()` с `unnest()`:

```
films %>%
  separate_rows(genres, sep = ", ") %>%
  mutate(value = TRUE) %>%
  pivot_wider(names_from = "genres",
              values_from = "value", values_fill = FALSE)
```

```
# A tibble: 3 x 5
  film             comedy drama criminal fantasy
  <chr>            <lgl>  <lgl> <lgl>    <lgl>
1            TRUE   TRUE  FALSE    FALSE
2           TRUE   FALSE TRUE     FALSE
3               FALSE  TRUE  FALSE    TRUE
```

Часть III

Разведочный анализ и создание отчетов

Этот раздел посвящен исследованию данных с помощью описательной статистики и визуализации данных, а также представлению результатов в виде отчетов.

В главе Глава 12 мы впервые познакомимся со статистикой, а именно с ее наиболее простой частью – описательной статистикой. Здесь разобраны как сами статистики, так и функции, которые позволяют удобным образом посчитать их все вместе для имеющихся данных.

В главах Глава 13, Глава 14, Глава 15 разобраны различные инструменты визуализации данных: как простые встроенные инструменты (Глава 13), более сложный и гибкий пакет `ggplot2` с его дополнениями (Глава 14) и даже интерактивные визуализации (Глава 15)!

В главе Глава 16 мы научимся делать отчеты и презентации с помощью R Markdown и Quarto, совмещая отформатированный текст, картинки, ссылки и прочее с кусками кода. Кстати, эта книга написана именно с помощью Quarto!

Глава 12

Описательная статистика

12.1 Описательная статистика и статистика вывода

Статистика делится на **описательную статистику** (*descriptive statistics*) и **статистику вывода** (*inferential statistics*). Описательная статистика пытается описать нашу **выборку** (*sample*, т.е. те данные, что у нас на руках) различными способами. Проблема в том, что описательная статистика может описать только то, что у нас есть, но не позволяет сделать выводы о **генеральной совокупности** (*population*) - это уже цель статистики вывода. Цель описательной статистики - "ужать" данные для их обобщенного понимания с помощью *статистик*.

> Заметьте, у выборки (**s**ample) мы считаем статистики (**s**tatistics), а у генеральной совокупности (**P**opulation) есть параметры (**P**arameters). Вот такая вот мнемотехника.

Статистики часто выступают в роли *точечной оценки* (point estimators) параметров, так что в этом легко запутаться. Например, среднее (в выборке) - это оценка среднего (в генеральной совокупности). Да, можно свихнуться. Мы это будем разбирать подробнее в следующие занятия (это действительно важно, поверьте), пока что остановимся только на описании выборки.

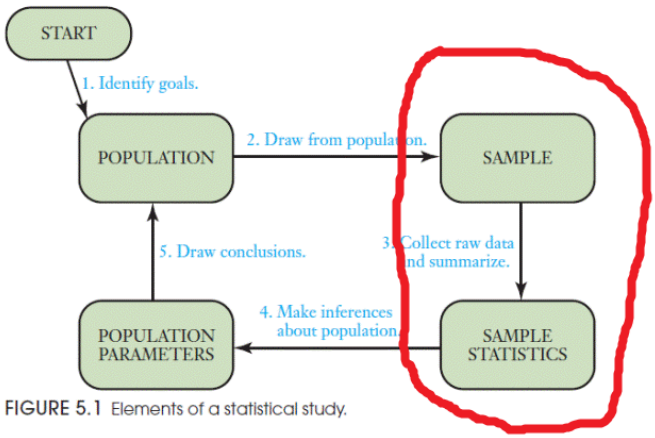


FIGURE 5.1 Elements of a statistical study.

12.2 Типы шкал

Перед тем, как начать речь об описательных статистиках, нужно разобраться с существующими типами шкал. Типы шкал классифицируются на основании типа измеряемых данных, которые задают допустимые для данной шкалы отношения.

- Шкала наименований (номинальная шкала) — самая простая шкала, где единственное отношение между элементами — это отношения равенства и неравенства. Это любая качественная шкала, между элементами которой не могут быть установлены отношения “больше — меньше”. Это большинство группирующих переменных (экспериментальная группа, пол, политическая партия, страна), переменные с id. Еще один пример - номера на майках у футболистов.

- Шкала порядка (ранговая шкала) — шкала следующего уровня, для которой можно установить отношения “больше — меньше”, причем если *B* больше *A*, а *C* больше *B*, то и *C* должно быть больше *A*. Если это верно, то мы можем выстроить последовательность значений. Однако мы еще не можем говорить о разнице между значениями. Ответы на вопросы “Как часто вы курите?” по шкале “Никогда”, “Редко” и “Часто” являются примером ранговой шкалы. “Часто” — это чаще, чем “Редко”, “Редко” — это чаще чем “Никогда”, и, соотвественно, “Часто” — это чаще, чем “Никогда”. Но мы не можем сказать, что разница между “Часто” и “Редко” такая же, как и между “Редко” и “Никогда”. Соответственно, даже если мы обозначим “Часто”, “Редко” и “Никогда” как 3, 2 и 1 соответственно,

то многого не можем сделать с этой шкалой, Например, мы не можем посчитать арифметическое среднее для такой шкалы.

- Шкала разностей (интервальная шкала) — шкала, для которой мы уже можем говорить про разницы между интервалами. Например, разница между 10 C° и 20 C° такая же как и между 80 C° и 90 C°. Для шкалы разностей уже можно сравнивать средние, но операции умножения и деления не имеют смысл, потому что ноль в шкале разностей относительный. Например, мы не можем сказать, что 20 C° — это в два раза теплее, чем 10 C°, потому что 0 C° — это просто условно взятая точка — температура плавления льда.

- Шкала отношений (абсолютная шкала) — самая "полноценная" шкала, которая отличается от интервальной наличием естественного и однозначного начала координат. Например, масса в килограммах или та же температура, но в градусах Кельвина, а не Цельсия.

12.3 Квантили

В жизни мы постоянно проходим какие-нибудь тесты, получаем баллы и рано или поздно встает вопрос: ну а как оно у других? Как бы нас ни учили книжки по саморазвитию, что не стоит сравнивать себя с другими, от этого вопроса очень сложно избавиться. А иногда и вовсе не нужно.

Допустим, вы проходите профессиональный тест с задачами одинаковой сложности. Как понять, если вы решили 10 из 20 задач (допустим, что задачи одинаковой сложности), то это много или мало? Мы договорились, что задачи одинаковой сложности, но не сказали какой. Если все 20 задач очень легкие, то 10 – это мало, а если сложные – то много. В этой ситуации может быть важен относительный успех: сколько людей справились с тестом хуже вас, а сколько - лучше вас. Вот это и позволяют посчитать **процентили** (*percentile rank*) – процент значений в распределении ниже заданного значения. То есть 90ый процентиль означает, что вы справились лучше, чем 90% людей, который прошли тот же тест. То есть вы находитесь в 10% самых-самых! Поэтому настоящие понторезы должны меряться не абсолютными значениями, а процентилями.

Здесь сразу нужно оговориться, что понятие процентиля имеет несколько неоднозначностей. В английском принято разделять *percentile* и *percentile rank*. *Percentile rank* – это процент значений в распределении ниже заданного, то просто *percentile* – это само значение, ниже которого находится соответствующий

процент значений. А иногда и вовсе процентилем называют сам интервал между процентильными границами. Все эти понятия взаимосвязаны, поэтому о том, в каком именно значении используется понятие “процентиль” можно догадаться из контекста. Другая неоднозначность понятия процентиля связана с тем, в какой процентиль относить пограничные значения. Эта проблема породила целых девять различных подходов к расчету процентилей! Однако если шкала континуальная и имеет достаточно много значений, то разницы между этими подходами не будет.

Можно делить значения не на 100 интервалов, а на меньшее количество. Например, на 4. Для этого нам нужно три точки: одна отделяет 25% наименьших значений, вторая отделяет нижнее 50% от верхних 50% (то есть это медиана!), третья – верхние 25% отнижних 75%. Эти точки и интервалы, разделяемые ими, называются **квартилями**.

Кроме процентилей и квартилей есть еще **децили**, **квинтили**, **секстили**, **септили** и что угодно **-тили**, хотя и используются они гораздо реже. Общее название для всех них – **квантили**.

12.4 Меры центральной тенденции

Продолжим работать с данными про супергероев.

```
library("tidyverse")
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
                   na = c("-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Для примера мы возьмем массу супергероев, предварительно удалив из нее все `NA` для удобства.

```
weight <- heroes %>%
  drop_na(Weight) %>%
  pull(Weight)
```

Мера центральной тенденции - это число для описания *центра* распределения.

12.4.1 Арифметическое среднее

Самая распространенная мера центральных тенденций - **арифметическое среднее**, то самое, которые мы считаем с помощью функции `mean()`.

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Не пугайтесь значка $\sum_{i=1}^{n}$ – он означает сумму от $i = 1$ до n. Что-то вроде цикла `for`!

> ***Практика:* функция *my_mean()***
>
> В качестве упражнения попробуйте самостоятельно превратить эту формулу в функцию `mymean()` с помощью `sum()` и `length()`. Можете убирать `NA` по дефолту! Сравните с результатом функции `mean()`.

```
mean(weight)
```

```
[1] 112.2525
```

12.4.2 Медиана

Представьте себе, что мы считаем среднюю зарплату сотрудников завода. Большинство рабочих получает 30000-40000 рублей в месяц, а директор завода получает 2 миллиона в месяц. Средняя зарплата на заводе в итоге равна 70000 рублей. Никакого подвоха: мы просто применили арифметическое среднее. Что спросили, то и получили, никаких манипуляций с цифрами! На деле же нас интересует обычно не средняя зарплата, а сколько получает средний сотрудник. Не директор, не главный мастер, но и не новичок и не алкоголик Василий, который постоянно опаздывает и плохо справляется с работой. Простой рабочий Иван, нормальный парень. Сколько зарабатывают такие как он? Для ответа на этот вопрос используют не арифметическое среднее, а **медиану.**

Медиана (median) - это *середина* распределения. Представим, что мы расставили значения по порядку (от меньшего к большему) и взяли значение посередине.

Если у нас четное количество значений, то берется среднее значение между теми двумя, что по середине.

Для расчета медианы есть функция `median()`:

```
median(weight)
```

```
[1] 81
```

Разница медианы со средним существенная. Это значит, что распределение довольно асимметричное.

Представьте себе, что кто-то говорит про среднюю зарплату в Москве. Но ведь эта средняя зарплата становится гораздо больше, если учитывать относительно небольшое количество мультимиллионеров и миллиардеров! А вот медианная зарплата будет гораздо меньше.

Представьте себе, что в среде супергероев появляется кто-то, кто весит 9000 килограммов! Тогда среднее сильно изменится:

```
mean(c(weight, 9000))
```

```
[1] 130.1714
```

А вот медиана останется той же.

```
median(c(weight, 9000))
```

```
[1] 81
```

Таким образом, экстремально большие или маленькие значения оказывают сильное влияние на арифметическое среднее, но не на медиану. Поэтому медиана считается более “робастной” оценкой, т.е. более устойчивой к выбросам и крайним значениям.

12.4.3 Усеченное среднее (trimmed mean)

Если про среднее и медиану слышали все, то про усеченное (тримленное) среднее известно гораздо меньше. Тем не менее, на практике это довольно удобная штука, потому что представляет собой некий компромисс между арифметическим средним и медианой.

В усеченном среднем значения ранжируются так же, как и для медианы, но отбрасывается только какой-то процент крайних значений. Усеченное среднее можно посчитать с помощью обычной функции `mean()`, поставив нужное значение параметра `trim =`:

```
mean(weight, trim = 0.1)
```

```
[1] 89.56423
```

`trim = 0.1` означает, что мы отбросили 10% слева и 10% справа. `trim` может принимать значения от 0 до 0.5. Что будет, если `trim = 0`?

```
mean(weight, trim = 0)
```

```
[1] 112.2525
```

Обычное арифметическое среднее! А если `trim = 0.5`?

```
mean(weight, trim = 0.5)
```

```
[1] 81
```

Медиана!

12.4.4 Мода

Мода *(mode)* - это самое *частое* значение. Обычно используется для номинальных переменных, для континуальных данных мода неприменима. Что интересно, в R нет встроенной функции для подсчета моды. Обычно она и не нужна: мы можем посчитать таблицу частот и даже проранжировать ее (и мы уже умеем это делать разными способами).

```
heroes %>%
  count(Gender, sort = TRUE)
```

```
# A tibble: 3 x 2
  Gender     n
  <chr>  <int>
1 Male     505
2 Female   200
3 <NA>      29
```

Можете попробовать написать свою функцию для моды!

12.5 Меры рассеяния

> Начинающий статистик пытался перейти в брод реку, средняя глубина которой 1 метр. И утонул.
> В чем была его ошибка? Он не учитывал разброс значений глубины!

Мер центральной тенденции недостаточно, чтобы описать выборку. Необходимо знать ее вариабельность.

12.5.1 Размах

Самое очевидное - посчитать **размах** *(range)*, то есть разницу между минимальным и максимальным значением. В R есть функция для вывода максимального и минимального значений:

```
range(weight)
```

```
[1]   2 900
```

Осталось посчитать разницу между ними:

```
diff(range(weight))
```

```
[1] 898
```

Естественно, крайние значения очень сильно влияют на этот размах, поэтому на практике он не очень-то используется.

12.5.2 Дисперсия

Дисперсия *(variance)* вычисляется по следующей формуле:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}$$

Попробуйте превратить это в функцию `myvar()`!

```
myvar <- function(x) mean((x - mean(x))^2)
```

Естественно, в R уже есть готовая функция `var()`. Но, заметьте, ее результат немного отличается от нашего:

```
myvar(weight)
```

```
[1] 10825.55
```

```
var(weight)
```

```
[1] 10847.46
```

Дело в том, что встроенная функция `var()` делит не на n, а на $n - 1$. Это связано с тем, что эта функция пытается оценить дисперсию в генеральной совокупности, т.е. относится уже к статистике вывода. Про это мы будем говорить в дальнейших занятиях, сейчас нам нужно только отметить то, что здесь есть небольшое различие.

12.5.3 Стандартное отклонение

Если вы заметили, значение дисперсии очень большое. Чтобы вернуться к единицам измерения, соответствующих нашим данным используется корень из дисперсии, то есть **стандартное отклонение** *(standard deviation)*:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}$$

Для этого есть функция `sd()`:

```
sd(weight)
```

```
[1] 104.1511
```

Что то же самое, что и:

```
sqrt(var(weight))
```

```
[1] 104.1511
```

12.5.3.1 Преобразование в z-оценки

На основе арифметического среднего и стандартного отклонения вычисляются z-оценки (z-scores) по простой формуле:

$$z_i = \frac{x_i - \overline{x}}{s}$$

Проще говоря, z-оценки – это центрированные вокруг нуля значения в единицах стандартного отклонения. Это может понадобится в случае необходимости свести разные шкалы как одной размерности (со средним равным нулю и стандартным отклонением равным одному). z-преобразование является стандартной процедурой для многих методов, например, для анализа главных компонент (см. **?@sec-pca**).

Для проведения z-преобразования можно воспользоваться встроенной функцией `scale()`, однако она возвращает z-оценки в довольно неочевидном формате: матрицу (см. Глава 4.1) с атрибутами (см. Глава 4.5) `"scaled:center"` (среднее изначальной шкалы) и `"scaled:scale"` (стандартное отклонение изначальной шкалы). Впрочем, функцию для z-преобразования легко написать самостоятельно:

```
head(scale(weight))
```

```
            [,1]
[1,]  3.15644618
[2,] -0.45369186
[3,] -0.21365608
[4,]  3.15644618
[5,]  0.09358971
[6,] -0.23285895
```

```
z <- function(x) (x - mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE)
head(z(weight))
```

```
[1]  3.15644618 -0.45369186 -0.21365608  3.15644618  0.09358971 -0.23285895
```

12.5.4 Медианное абсолютное отклонение

Поскольку стандартное отклонение не устойчиво к выбросам, то иногда используют его альтернативу, которая устойчива к выбросам (особенно если эти выбросы нам как раз и нужно удалить) - медианное абсолютное отклонение (median absolute deviation):

$$mad = median(|x_i - median(x)|)$$

Для этого есть функция `mad()`:

```
mad(weight)
```

```
[1] 32.6172
```

12.5.5 Межквартильный размах

Другой вариант рабостной оценки вариабельности данных является **межквартильный размах** *(interquartile range, IQR)*. Это разница между третьим и первым **квартилем** [1] - значением, которое больше 75% значений в выборке, и значением, которое больше 25% значений в выборке.

```
IQR(weight)
```

```
[1] 47
```

Ну а второй квартиль - это медиана!

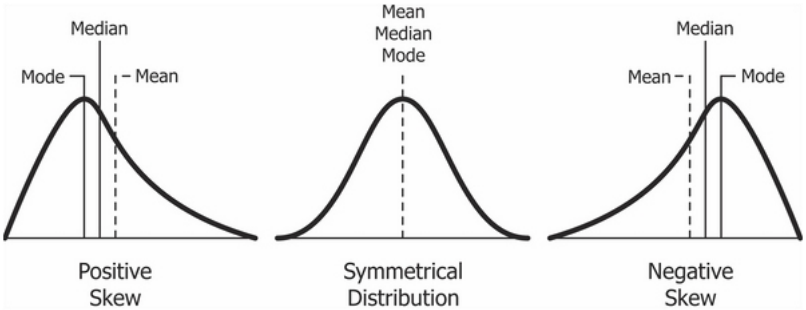
12.6 Асимметрия и эксцесс

12.6.1 Асимметрия

Асимметрия *(skewness)* измеряет симметричность распределения. **Положительный показатель асимметрии *("Right-skewed"***

[1] Квартиль — это частный пример квантиля. Другой известный квантиль — процентиль. Процентили часто используют для сравнения значения с другими значениями. Например, 63ий процентиль означает, что данное значение больше 63% значений в выборке.

или ***positive skewness)*** означает, что хвосты с правой части распределения длиннее. **Негативный показатель асимметрии (*"Left-skewed"*** или ***negative skewness)*** означает, что левый хвост длиннее. Нулевой показатель асимметрии означает симметричное распределение.



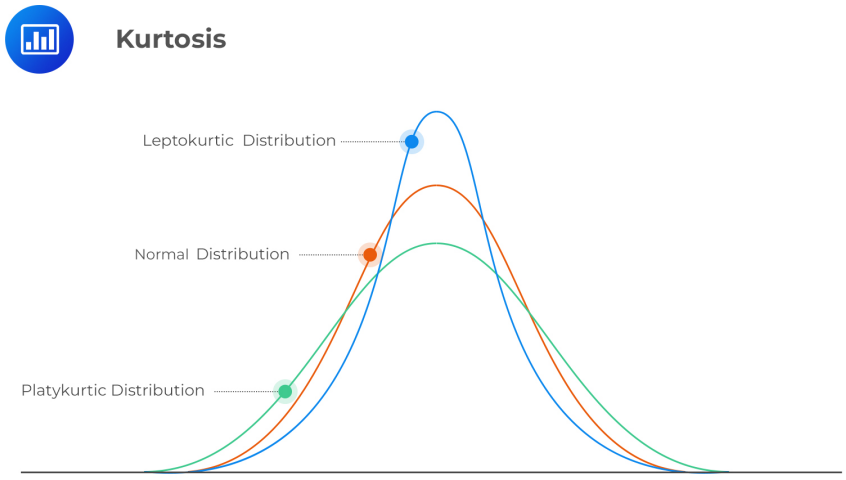
В целом, распределения с положительным показателем асимметрии на практике встречаются чаще, чем с отрицательным: очень часто мы сталкиваемся со шкалами, которые ограничены снизу, но не ограничены сверху.

- В когнитивистике положительная асимметрия встречается очень часто. Например, время реакции: оно ограничено снизу 0 мс (а по факту не меньше 100 мс – быстрее сигнал не успеет по нервной системе пройти до пальцев), а вот с другой стороны оно никак не ограничено. Испытуемый может на полчаса перед монитором затупить, ага.

- Если мы анализируем размеры индивидуальные доходы, то они не могут быть меньше 0, но могут быть невероятно большими для относительно небольшого количества миллионеров и миллиардеров. Доходы остальных людей находятся в относительно небольшом диапазоне (который гораздо ближе к нулю, чем к доходам миллиардеров)

12.6.2 Эксцесс

Эксцесс *(kurtosis)* - это мера "вытянутости" распределения:



Положительные показатели эксцесса означают “вытянутое” распределение, а отрицательные - “плоское”.

12.6.3 Ассиметрия и эксцесс в R

К сожалению, в базовом R нет функций для асимметрии и эксцесса. Зато есть замечательный пакет `{psych}` (да-да, специально для психологов).

```
install.packages("psych")
```

```
library("psych")
```

```
Attaching package: 'psych'

The following objects are masked from 'package:ggplot2':

    %+%, alpha
```

В нем есть функции `skew()` и `kurtosi()`:

```
skew(weight)
```

```
[1] 3.874557
```

```
kurtosi(weight)
```

```
[1] 19.45699
```

Асимметрия положительная, это значит что распределение выборки асимметричное, хвосты с правой части длиннее. Эксцесс значительно выше нуля - значит распределение довольно “вытянутое”.

12.7 А теперь все вместе!

В базовом R есть функция `summary()`, которая позволяет получить сразу неплохой набор описательных статистик.

```
summary(weight)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    2.0    61.0    81.0   112.3   108.0   900.0 
```

Функция `summary()` - это **универсальная *(generic)*** функция (см. Глава 4.5). Это означает, что Вы можете ее применять для разных объектов и получать разные результаты. Попробуйте применить ее к векторам с разными типами данных и даже к датафреймам. Посмотрите, что получится.

В пакете `psych` есть еще и замечательная функция `describe()`, которая даст Вам еще больше статистик, включая ассиметрию и куртозис:

```
psych::describe(weight)
```

```
   vars   n   mean     sd median trimmed   mad min max range skew kurtosis   se
X1    1 495 112.25 104.15     81   89.56 32.62   2 900   898 3.87    19.46 4.68
```

Даже **усеченное *(trimmed)* среднее** есть (с `trim = 0.1`)! Все кроме `se` мы уже знаем. А про этот `se` узнаем немного позже (см. Глава 17.6).

Эта функция хорошо работает в сочетании с `group_by()`:

```
heroes %>%
  group_by(Gender) %>%
  summarise(describe(Weight))
```

```
# A tibble: 3 x 14
  Gender  vars     n  mean    sd median trimmed   mad   min   max range  skew
  <chr>  <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Female     1   142  78.8  77.0     58    60.9  7.41    41   630   589  4.97
2 Male       1   339 126.  111.      90   103.  23.7      2   900   898  3.76
3 <NA>       1    14 129.  107.      94   115.  43.0     39   383   344  1.55
# i 2 more variables: kurtosis <dbl>, se <dbl>
```

Другой интересный пакет для получения описательных статистик для всего датафрейма — {skimr}.

```
install.packages("skimr")
```

Его основная функция — skim(), выводит в консоли симпатичную сводную таблицу:

```
skimr::skim(heroes)
```

Таблица 12.1: Data summary

Name	heroes
Number of rows	734
Number of columns	11
Column type frequency:	
character	8
numeric	3
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
name	0	1.00	1	25	0	715	0
Gender	29	0.96	4	6	0	2	0
Eye color	172	0.77	3	23	0	22	0
Race	304	0.59	5	18	0	61	0

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Hair color	172	0.77	3	16	0	29	0
Publisher	0	1.00	0	17	15	25	0
Skin color	662	0.10	3	14	0	16	0
Alignment	7	0.99	3	7	0	3	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
...1	0	1.00	366.50	212.03	0.0	183.25	366.5	549.75	733	▇▇▇▇▇
Height	217	0.70	186.73	59.25	15.2	173.00	183.0	191.00	975	▇▇▇▇▇
Weight	239	0.67	112.25	104.15	2.0	61.00	81.0	108.00	900	▇▇▇▇▇

В зависимости от типа данных колонки функция skim() показывает различные статистики. Для числовых колонок мы можем увидеть количество и долю пропущенных значений, среднее и стандартное отклонение. А еще мы видим некие *p0, p25, p50, p75* и *p100.* Это процентили! И совсем не с потолка взятые:

- p0 – минимальное значение,
- p25 – первый квартиль (Q1),
- p50 – второй квартиль (Q2), т.е. медиана,
- p75 – третий квартиль (Q3),
- p100 – максимальное значение.

Ну и вишенкой на торте выступает маленькая гистограмма (см. для каждой колонки!

Кроме того, `skimr` адаптирован под tidyverse. В нем можно выбирать колонки с помощью tidyselect (@ref(tidyselect)) прямо внутри функции `skim()`.

```
heroes %>%
  skimr::skim(ends_with("color"))
```

Таблица 12.4: Data summary

Name	Piped data

Таблица 12.4: Data summary

Number of rows	734
Number of columns	11
Column type frequency:	
character	3
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Eye color	172	0.77	3	23	0	22	0
Hair color	172	0.77	3	16	0	29	0
Skin color	662	0.10	3	14	0	16	0

А еще можно сочетать с группировкой с помощью `group_by()`.

```
heroes %>%
  group_by(Gender) %>%
  skimr::skim(ends_with("color"))
```

Таблица 12.6: Data summary

Name	Piped data
Number of rows	734
Number of columns	11
Column type frequency:	
character	3
Group variables	Gender

Variable type: character

skim_variable	Gender	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Eye color	Female	41	0.80	3	23	0	14	0

skim_variable	Gender	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Eye color	Male	121	0.76	3	12	0	18	0
Eye color	NA	10	0.66	3	23	0	7	0
Hair color	Female	38	0.81	3	16	0	18	0
Hair color	Male	123	0.76	3	16	0	23	0
Hair color	NA	11	0.62	4	14	0	10	0
Skin color	Female	186	0.07	4	6	0	7	0
Skin color	Male	449	0.11	3	14	0	14	0
Skin color	NA	27	0.07	4	4	0	2	0

12.7.1 Описательных статистик недостаточно {@sec-datasaurus}

Я в тайне от Вас загрузил данные в переменную `xxx` (можете найти этот набор данных здесь, если интересно). Выглядят они примерно так:

```
head(xxx)
```

```
# A tibble: 6 x 2
      x     y
  <dbl> <dbl>
1  55.4  97.2
2  51.5  96.0
3  46.2  94.5
4  42.8  91.4
5  40.8  88.3
6  38.7  84.9
```

```
str(xxx)
```

```
spc_tbl_ [142 x 2] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ x: num [1:142] 55.4 51.5 46.2 42.8 40.8 ...
```

```
 $ y: num [1:142] 97.2 96 94.5 91.4 88.3 ...
 - attr(*, "spec")=
  .. cols(
  ..   x = col_double(),
  ..   y = col_double()
  .. )
 - attr(*, "problems")=<externalptr>
```

Надеюсь, Вы уже понимаете, как это интерпретировать - два столбца с 142 числами каждый. Представьте себе, как выглядят эти точки на плоскости, если каждая строчка означают координаты одной точки по осям *x* и *y* (это называется диаграмма рассеяния, точечная диаграмма или scatterplot).

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <9f>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b2>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <82>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <be>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <87>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <ba>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b8>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b7>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b4>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d0>

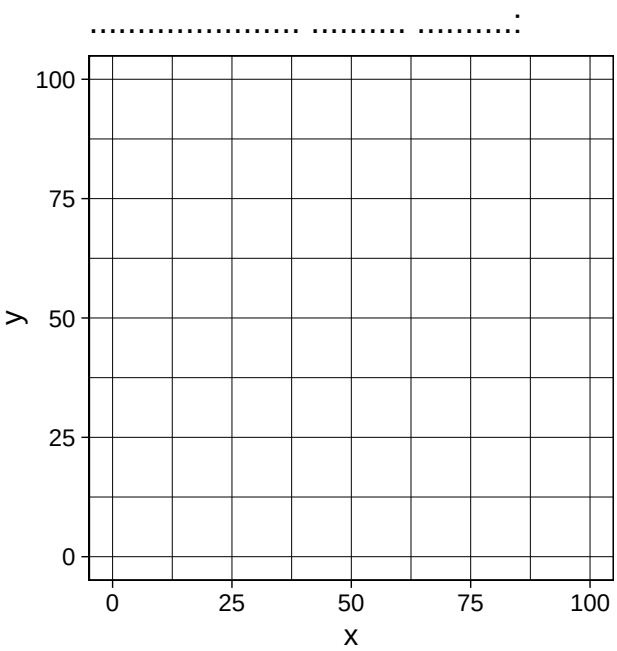

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <b5>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <81>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <d1>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '             :' in 'mbcsToSbcs': dot
substituted for <8c>
```



Применим разные функции, которые мы выучили:

```
mean(xxx$x)
```

```
[1] 54.26327
```

```
mean(xxx$y)
```

```
[1] 47.83225
```

```
median(xxx$x)
```

```
[1] 53.3333
```

```
median(xxx$y)
```

```
[1] 46.0256
```

Средние и медианы примерно одинаковые, при этом по *x* они около 53-54, а по *y* - примерно 46-47. Попытайтесь представить это. Идем дальше:

```
sd(xxx$x)
```

```
[1] 16.76514
```

```
sd(xxx$y)
```

```
[1] 26.9354
```

Похоже, разброс по *y* несколько больше, верно?

```
skew(xxx$x)
```

```
[1] 0.2807568
```

```
skew(xxx$y)
```

```
[1] 0.2472603
```

```
kurtosi(xxx$x)
```

```
[1] -0.2854912
```

```
kurtosi(xxx$y)
```

```
[1] -1.063552
```

Похоже, оба распределения немного право-ассиметричны и довольно “плоские”.

Давайте еще посчитаем **коэффициент корреляции *(correlation coefficient)***. Мы про него будем говорить позже гораздо подробнее (Глава 20.2). Пока что нам нужно знать, что **коэффициент корреляции** говорит о линейной связи двух переменных. Если коэффициент корреляции *положительный* (максимум равен 1), то чем *больше* x, тем *больше* y. Если *отрицательный* (минимум равен -1), то чем *больше* x, тем *меньше* y. Если же коэффициент корреляции равен нулю, то такая линейная зависимость отсутствует.

```
cor(xxx$x, xxx$y)
```

```
[1] -0.06447185
```

Коэффициент корреляции очень близка к нулю (делайте выводы и представляйте).

Давайте напоследок воспользуемся функцией `describe()` из `psych`:

```
psych::describe(xxx)
```

```
  vars   n  mean    sd median trimmed   mad   min   max range skew kurtosis
x    1 142 54.26 16.77  53.33   53.69 15.97 22.31 98.21 75.90 0.28    -0.29
y    2 142 47.83 26.94  46.03   46.90 30.79  2.95 99.49 96.54 0.25    -1.06
    se
x 1.41
y 2.26
```

```
skimr::skim(xxx)
```

Таблица 12.8: Data summary

Name	xxx
Number of rows	142
Number of columns	2
Column type frequency:	
numeric	2
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
x	0	1	54.26	16.77	22.31	44.10	53.33	64.74	98.21	▁▆▇▃▁
y	0	1	47.83	26.94	2.95	25.29	46.03	68.53	99.49	▇▆▆▆▅

Готовы узнать, как выглядят эти данные на самом деле?!

Жмите сюда если готовы!

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <ad>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <be>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <94>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <82>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b7>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

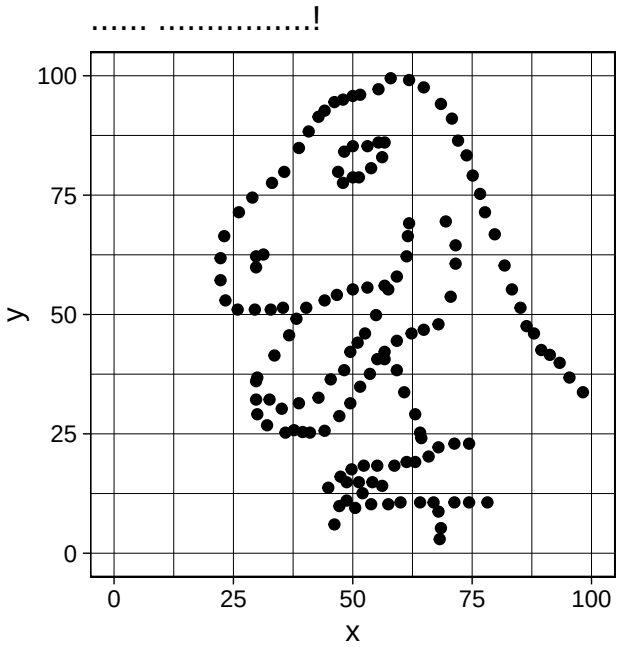
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <b2>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <d1>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on '       !' in 'mbcsToSbcs': dot substituted for <80>
```



Из этого можно сделать важный вывод: не стоит слепо доверять описательным статистикам. Нужно визуализировать данные, иначе можно попасть в такую ситуацию в реальности. По словам знаменитого статитстика Джона Тьюки, величайшая ценность картинки в том, что она заставляет нас заметить то, что мы не ожидали заметить. Поэтому графики — это не просто метод коммуникации — представления ваших результатов сообществу в понятном виде (хотя и это, конечно, тоже), но и сам по себе очень важный метод анализа данных.

Глава 13

Встроенные функции для графиков

В R есть достаточно мощные встроенные инструменты для визуализации. Я приведу три простых примера: функции `plot()`, `hist()` и `boxplot()`.

13.1 Многоликий `plot()`

Для примера возьмем датасет `heroes`, вытащим из него колонки `Height` и `Weight` как векторы и применим на них функцию `plot()`.

```
library("tidyverse")
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
```

```
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
                   na = c("-", "-99"))
```

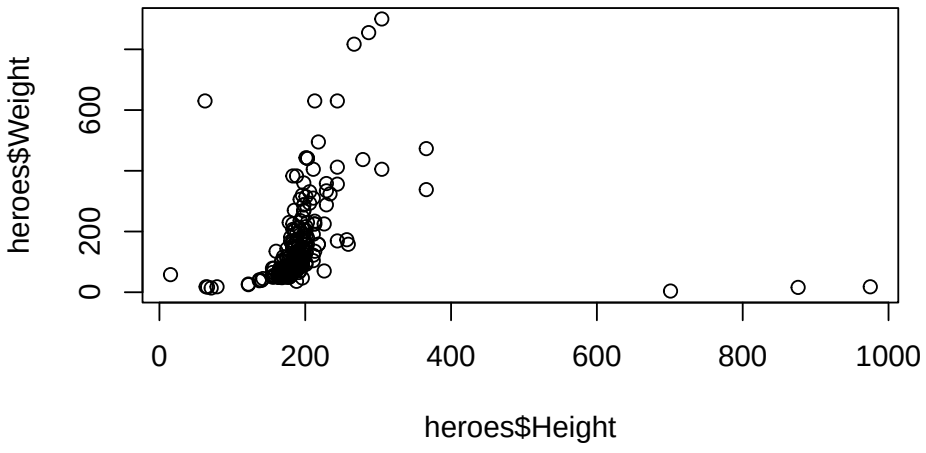
```
New names:
* `` -> `...1`

Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)

Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
plot(heroes$Height, heroes$Weight)
```



В этом случае будет нарисована **диаграмма рассеяния (точечная диаграмма, *scatterplot*),** где каждая точка задается

парой значений из этих векторов: значения из колонки `Height` используются как координаты по оси *x*, а соответствующие значения колонки `Weight` как координаты по оси *y*.

Функция `plot()` - это тоже **универсальная *(generic)*** функция, как и `summary()`(см. Глава 12.7). В качестве аргумента можете ей скормить просто один вектор, матрицу, датафрейм. Давайте попробуем использовать функцию `plot()` на встроенном датафрейме `iris`[1]:

```
iris
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
6            5.4         3.9          1.7         0.4     setosa
7            4.6         3.4          1.4         0.3     setosa
8            5.0         3.4          1.5         0.2     setosa
9            4.4         2.9          1.4         0.2     setosa
10           4.9         3.1          1.5         0.1     setosa
11           5.4         3.7          1.5         0.2     setosa
12           4.8         3.4          1.6         0.2     setosa
13           4.8         3.0          1.4         0.1     setosa
14           4.3         3.0          1.1         0.1     setosa
15           5.8         4.0          1.2         0.2     setosa
16           5.7         4.4          1.5         0.4     setosa
17           5.4         3.9          1.3         0.4     setosa
18           5.1         3.5          1.4         0.3     setosa
19           5.7         3.8          1.7         0.3     setosa
20           5.1         3.8          1.5         0.3     setosa
21           5.4         3.4          1.7         0.2     setosa
22           5.1         3.7          1.5         0.4     setosa
23           4.6         3.6          1.0         0.2     setosa
24           5.1         3.3          1.7         0.5     setosa
25           4.8         3.4          1.9         0.2     setosa
26           5.0         3.0          1.6         0.2     setosa
27           5.0         3.4          1.6         0.4     setosa
28           5.2         3.5          1.5         0.2     setosa
```

[1] `iris` – пожалуй, самый известный набор данных в мире, своего рода *“Hello, world!”* от мира науки о данных. Этот датасет был собран известным ботаником Эдгаром Андерсоном, но стал известен благодаря статье 1936 года известного статистика Роберта Фишера, в которой он использовал эти данные для демонстрации разработанного им метода дискриминантного анализа. С `iris` мы еще столкнемся в главе про многомерные методы анализа (Глава 24).

29	5.2	3.4	1.4	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa
31	4.8	3.1	1.6	0.2	setosa
32	5.4	3.4	1.5	0.4	setosa
33	5.2	4.1	1.5	0.1	setosa
34	5.5	4.2	1.4	0.2	setosa
35	4.9	3.1	1.5	0.2	setosa
36	5.0	3.2	1.2	0.2	setosa
37	5.5	3.5	1.3	0.2	setosa
38	4.9	3.6	1.4	0.1	setosa
39	4.4	3.0	1.3	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
41	5.0	3.5	1.3	0.3	setosa
42	4.5	2.3	1.3	0.3	setosa
43	4.4	3.2	1.3	0.2	setosa
44	5.0	3.5	1.6	0.6	setosa
45	5.1	3.8	1.9	0.4	setosa
46	4.8	3.0	1.4	0.3	setosa
47	5.1	3.8	1.6	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
49	5.3	3.7	1.5	0.2	setosa
50	5.0	3.3	1.4	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
53	6.9	3.1	4.9	1.5	versicolor
54	5.5	2.3	4.0	1.3	versicolor
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
58	4.9	2.4	3.3	1.0	versicolor
59	6.6	2.9	4.6	1.3	versicolor
60	5.2	2.7	3.9	1.4	versicolor
61	5.0	2.0	3.5	1.0	versicolor
62	5.9	3.0	4.2	1.5	versicolor
63	6.0	2.2	4.0	1.0	versicolor
64	6.1	2.9	4.7	1.4	versicolor
65	5.6	2.9	3.6	1.3	versicolor
66	6.7	3.1	4.4	1.4	versicolor
67	5.6	3.0	4.5	1.5	versicolor
68	5.8	2.7	4.1	1.0	versicolor
69	6.2	2.2	4.5	1.5	versicolor
70	5.6	2.5	3.9	1.1	versicolor
71	5.9	3.2	4.8	1.8	versicolor
72	6.1	2.8	4.0	1.3	versicolor
73	6.3	2.5	4.9	1.5	versicolor
74	6.1	2.8	4.7	1.2	versicolor

```
75          6.4         2.9         4.3         1.3 versicolor
76          6.6         3.0         4.4         1.4 versicolor
77          6.8         2.8         4.8         1.4 versicolor
78          6.7         3.0         5.0         1.7 versicolor
79          6.0         2.9         4.5         1.5 versicolor
80          5.7         2.6         3.5         1.0 versicolor
81          5.5         2.4         3.8         1.1 versicolor
82          5.5         2.4         3.7         1.0 versicolor
83          5.8         2.7         3.9         1.2 versicolor
84          6.0         2.7         5.1         1.6 versicolor
85          5.4         3.0         4.5         1.5 versicolor
86          6.0         3.4         4.5         1.6 versicolor
87          6.7         3.1         4.7         1.5 versicolor
88          6.3         2.3         4.4         1.3 versicolor
89          5.6         3.0         4.1         1.3 versicolor
90          5.5         2.5         4.0         1.3 versicolor
91          5.5         2.6         4.4         1.2 versicolor
92          6.1         3.0         4.6         1.4 versicolor
93          5.8         2.6         4.0         1.2 versicolor
94          5.0         2.3         3.3         1.0 versicolor
95          5.6         2.7         4.2         1.3 versicolor
96          5.7         3.0         4.2         1.2 versicolor
97          5.7         2.9         4.2         1.3 versicolor
98          6.2         2.9         4.3         1.3 versicolor
99          5.1         2.5         3.0         1.1 versicolor
100         5.7         2.8         4.1         1.3 versicolor
101         6.3         3.3         6.0         2.5  virginica
102         5.8         2.7         5.1         1.9  virginica
103         7.1         3.0         5.9         2.1  virginica
104         6.3         2.9         5.6         1.8  virginica
105         6.5         3.0         5.8         2.2  virginica
106         7.6         3.0         6.6         2.1  virginica
107         4.9         2.5         4.5         1.7  virginica
108         7.3         2.9         6.3         1.8  virginica
109         6.7         2.5         5.8         1.8  virginica
110         7.2         3.6         6.1         2.5  virginica
111         6.5         3.2         5.1         2.0  virginica
112         6.4         2.7         5.3         1.9  virginica
113         6.8         3.0         5.5         2.1  virginica
114         5.7         2.5         5.0         2.0  virginica
115         5.8         2.8         5.1         2.4  virginica
116         6.4         3.2         5.3         2.3  virginica
117         6.5         3.0         5.5         1.8  virginica
118         7.7         3.8         6.7         2.2  virginica
119         7.7         2.6         6.9         2.3  virginica
120         6.0         2.2         5.0         1.5  virginica
```

```
121          6.9         3.2          5.7         2.3  virginica
122          5.6         2.8          4.9         2.0  virginica
123          7.7         2.8          6.7         2.0  virginica
124          6.3         2.7          4.9         1.8  virginica
125          6.7         3.3          5.7         2.1  virginica
126          7.2         3.2          6.0         1.8  virginica
127          6.2         2.8          4.8         1.8  virginica
128          6.1         3.0          4.9         1.8  virginica
129          6.4         2.8          5.6         2.1  virginica
130          7.2         3.0          5.8         1.6  virginica
131          7.4         2.8          6.1         1.9  virginica
132          7.9         3.8          6.4         2.0  virginica
133          6.4         2.8          5.6         2.2  virginica
134          6.3         2.8          5.1         1.5  virginica
135          6.1         2.6          5.6         1.4  virginica
136          7.7         3.0          6.1         2.3  virginica
137          6.3         3.4          5.6         2.4  virginica
138          6.4         3.1          5.5         1.8  virginica
139          6.0         3.0          4.8         1.8  virginica
140          6.9         3.1          5.4         2.1  virginica
141          6.7         3.1          5.6         2.4  virginica
142          6.9         3.1          5.1         2.3  virginica
143          5.8         2.7          5.1         1.9  virginica
144          6.8         3.2          5.9         2.3  virginica
145          6.7         3.3          5.7         2.5  virginica
146          6.7         3.0          5.2         2.3  virginica
147          6.3         2.5          5.0         1.9  virginica
148          6.5         3.0          5.2         2.0  virginica
149          6.2         3.4          5.4         2.3  virginica
150          5.9         3.0          5.1         1.8  virginica
```

Каждая строчка – один цветок ириса, числовые колонки содержат информацию о длине и ширине наружной *(sepal)* и внутренней *(petal)* долях околоцветника, в колонке `Species` содержится название сорта ирисов.

Давайте уберем последнюю колонку, чтобы у нас остались только числовые колонки:

```
iris[, -5]
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width
1            5.1         3.5          1.4         0.2
2            4.9         3.0          1.4         0.2
3            4.7         3.2          1.3         0.2
```

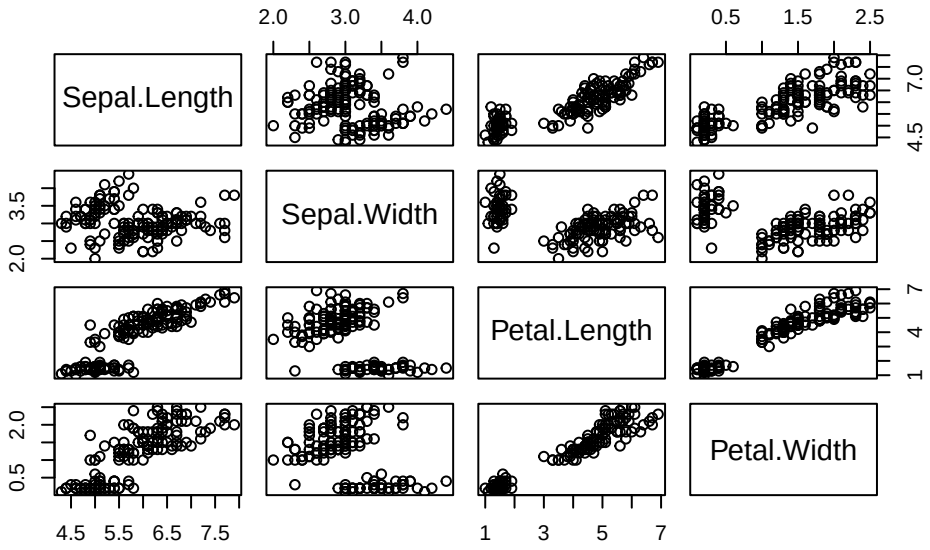
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4
7	4.6	3.4	1.4	0.3
8	5.0	3.4	1.5	0.2
9	4.4	2.9	1.4	0.2
10	4.9	3.1	1.5	0.1
11	5.4	3.7	1.5	0.2
12	4.8	3.4	1.6	0.2
13	4.8	3.0	1.4	0.1
14	4.3	3.0	1.1	0.1
15	5.8	4.0	1.2	0.2
16	5.7	4.4	1.5	0.4
17	5.4	3.9	1.3	0.4
18	5.1	3.5	1.4	0.3
19	5.7	3.8	1.7	0.3
20	5.1	3.8	1.5	0.3
21	5.4	3.4	1.7	0.2
22	5.1	3.7	1.5	0.4
23	4.6	3.6	1.0	0.2
24	5.1	3.3	1.7	0.5
25	4.8	3.4	1.9	0.2
26	5.0	3.0	1.6	0.2
27	5.0	3.4	1.6	0.4
28	5.2	3.5	1.5	0.2
29	5.2	3.4	1.4	0.2
30	4.7	3.2	1.6	0.2
31	4.8	3.1	1.6	0.2
32	5.4	3.4	1.5	0.4
33	5.2	4.1	1.5	0.1
34	5.5	4.2	1.4	0.2
35	4.9	3.1	1.5	0.2
36	5.0	3.2	1.2	0.2
37	5.5	3.5	1.3	0.2
38	4.9	3.6	1.4	0.1
39	4.4	3.0	1.3	0.2
40	5.1	3.4	1.5	0.2
41	5.0	3.5	1.3	0.3
42	4.5	2.3	1.3	0.3
43	4.4	3.2	1.3	0.2
44	5.0	3.5	1.6	0.6
45	5.1	3.8	1.9	0.4
46	4.8	3.0	1.4	0.3
47	5.1	3.8	1.6	0.2
48	4.6	3.2	1.4	0.2
49	5.3	3.7	1.5	0.2

50	5.0	3.3	1.4	0.2
51	7.0	3.2	4.7	1.4
52	6.4	3.2	4.5	1.5
53	6.9	3.1	4.9	1.5
54	5.5	2.3	4.0	1.3
55	6.5	2.8	4.6	1.5
56	5.7	2.8	4.5	1.3
57	6.3	3.3	4.7	1.6
58	4.9	2.4	3.3	1.0
59	6.6	2.9	4.6	1.3
60	5.2	2.7	3.9	1.4
61	5.0	2.0	3.5	1.0
62	5.9	3.0	4.2	1.5
63	6.0	2.2	4.0	1.0
64	6.1	2.9	4.7	1.4
65	5.6	2.9	3.6	1.3
66	6.7	3.1	4.4	1.4
67	5.6	3.0	4.5	1.5
68	5.8	2.7	4.1	1.0
69	6.2	2.2	4.5	1.5
70	5.6	2.5	3.9	1.1
71	5.9	3.2	4.8	1.8
72	6.1	2.8	4.0	1.3
73	6.3	2.5	4.9	1.5
74	6.1	2.8	4.7	1.2
75	6.4	2.9	4.3	1.3
76	6.6	3.0	4.4	1.4
77	6.8	2.8	4.8	1.4
78	6.7	3.0	5.0	1.7
79	6.0	2.9	4.5	1.5
80	5.7	2.6	3.5	1.0
81	5.5	2.4	3.8	1.1
82	5.5	2.4	3.7	1.0
83	5.8	2.7	3.9	1.2
84	6.0	2.7	5.1	1.6
85	5.4	3.0	4.5	1.5
86	6.0	3.4	4.5	1.6
87	6.7	3.1	4.7	1.5
88	6.3	2.3	4.4	1.3
89	5.6	3.0	4.1	1.3
90	5.5	2.5	4.0	1.3
91	5.5	2.6	4.4	1.2
92	6.1	3.0	4.6	1.4
93	5.8	2.6	4.0	1.2
94	5.0	2.3	3.3	1.0
95	5.6	2.7	4.2	1.3

96	5.7	3.0	4.2	1.2
97	5.7	2.9	4.2	1.3
98	6.2	2.9	4.3	1.3
99	5.1	2.5	3.0	1.1
100	5.7	2.8	4.1	1.3
101	6.3	3.3	6.0	2.5
102	5.8	2.7	5.1	1.9
103	7.1	3.0	5.9	2.1
104	6.3	2.9	5.6	1.8
105	6.5	3.0	5.8	2.2
106	7.6	3.0	6.6	2.1
107	4.9	2.5	4.5	1.7
108	7.3	2.9	6.3	1.8
109	6.7	2.5	5.8	1.8
110	7.2	3.6	6.1	2.5
111	6.5	3.2	5.1	2.0
112	6.4	2.7	5.3	1.9
113	6.8	3.0	5.5	2.1
114	5.7	2.5	5.0	2.0
115	5.8	2.8	5.1	2.4
116	6.4	3.2	5.3	2.3
117	6.5	3.0	5.5	1.8
118	7.7	3.8	6.7	2.2
119	7.7	2.6	6.9	2.3
120	6.0	2.2	5.0	1.5
121	6.9	3.2	5.7	2.3
122	5.6	2.8	4.9	2.0
123	7.7	2.8	6.7	2.0
124	6.3	2.7	4.9	1.8
125	6.7	3.3	5.7	2.1
126	7.2	3.2	6.0	1.8
127	6.2	2.8	4.8	1.8
128	6.1	3.0	4.9	1.8
129	6.4	2.8	5.6	2.1
130	7.2	3.0	5.8	1.6
131	7.4	2.8	6.1	1.9
132	7.9	3.8	6.4	2.0
133	6.4	2.8	5.6	2.2
134	6.3	2.8	5.1	1.5
135	6.1	2.6	5.6	1.4
136	7.7	3.0	6.1	2.3
137	6.3	3.4	5.6	2.4
138	6.4	3.1	5.5	1.8
139	6.0	3.0	4.8	1.8
140	6.9	3.1	5.4	2.1
141	6.7	3.1	5.6	2.4

```
142          6.9         3.1          5.1         2.3
143          5.8         2.7          5.1         1.9
144          6.8         3.2          5.9         2.3
145          6.7         3.3          5.7         2.5
146          6.7         3.0          5.2         2.3
147          6.3         2.5          5.0         1.9
148          6.5         3.0          5.2         2.0
149          6.2         3.4          5.4         2.3
150          5.9         3.0          5.1         1.8
```

```
plot(iris[, -5])
```



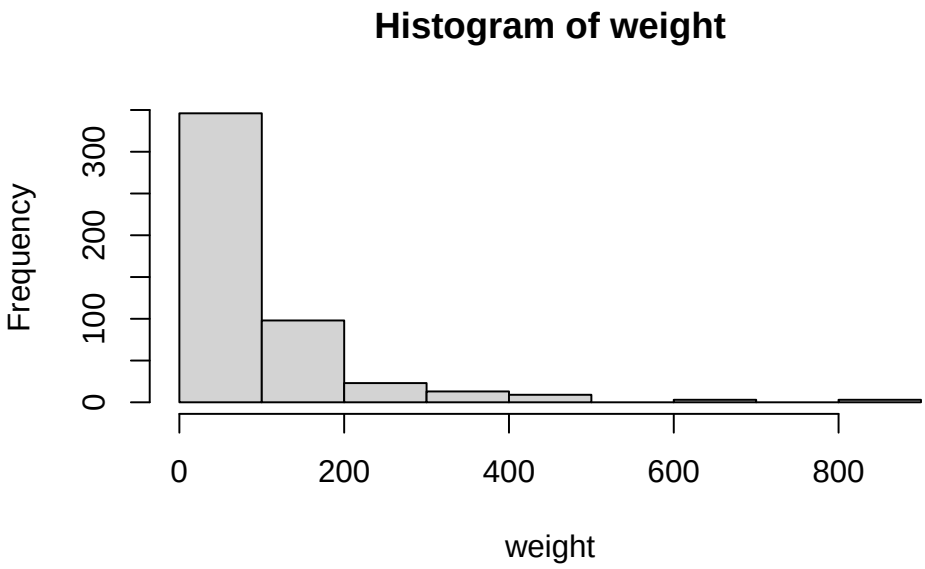
Мы получили таблицу из **диаграмм рассеяния!** Для каждой пары колонок строится отдельная диаграмма рассеяния, что может быть очень удобно при исследовании данных.

Многие пакеты добавляют новые методы `plot()` для объектов из новых классов, поэтому функцию `plot()` имеет смысл пробовать на любых непривычных вам объектах.

13.2 Великая гистограмма

Другая распространенная функция — `hist()` — **гистограмма (histogram)**:

```
weight <- heroes %>%
  drop_na(Weight) %>%
  pull(Weight)
hist(weight)
```



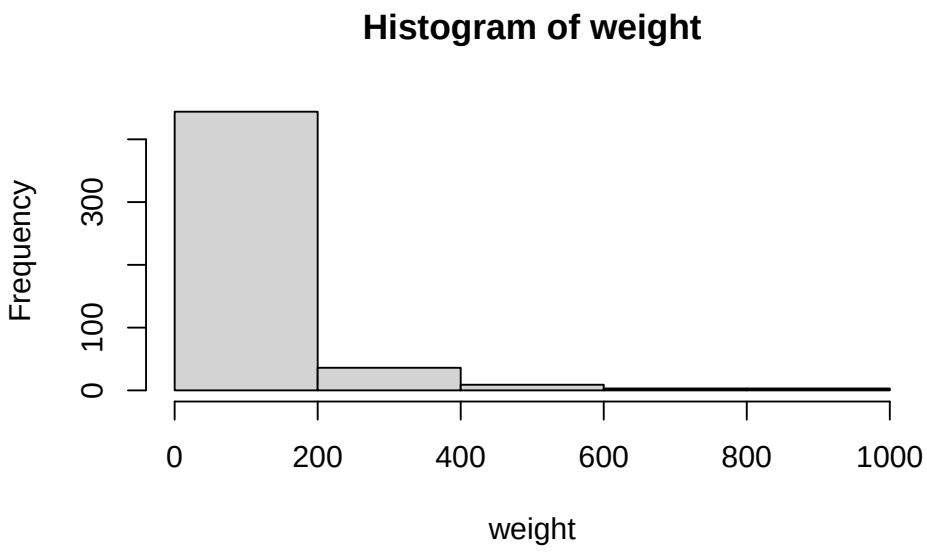
Гистограмма – очень простая визуализация, которую при желании несложно нарисовать карандашом на бумаге, следуя простому алгоритму:

1. Весь диапазон значений в выборке делим на интервалы (обычно одинаковые, но можно и разные).
2. Считаем сколько точек попадает в каждый из интервалов.
3. Откладываем высоту столбцов в зависимости от количества точек в каждом интервале.

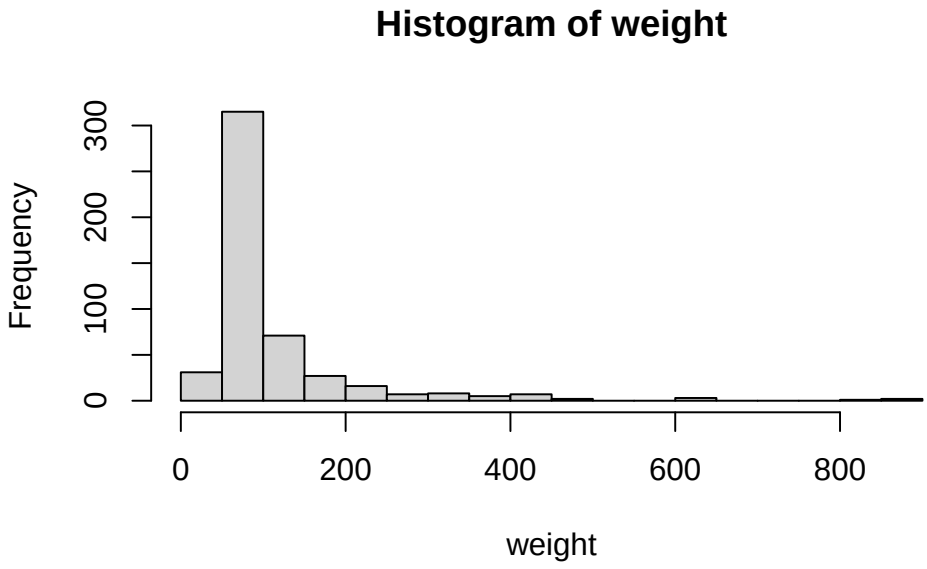
Главный вопрос здесь: на какие интервалы мы делим? В зависимости от этого гистограмма может выглядеть очень по-разному.

Количество интервалов можно задать самостоятельно с помощью аргумента `breaks =`:

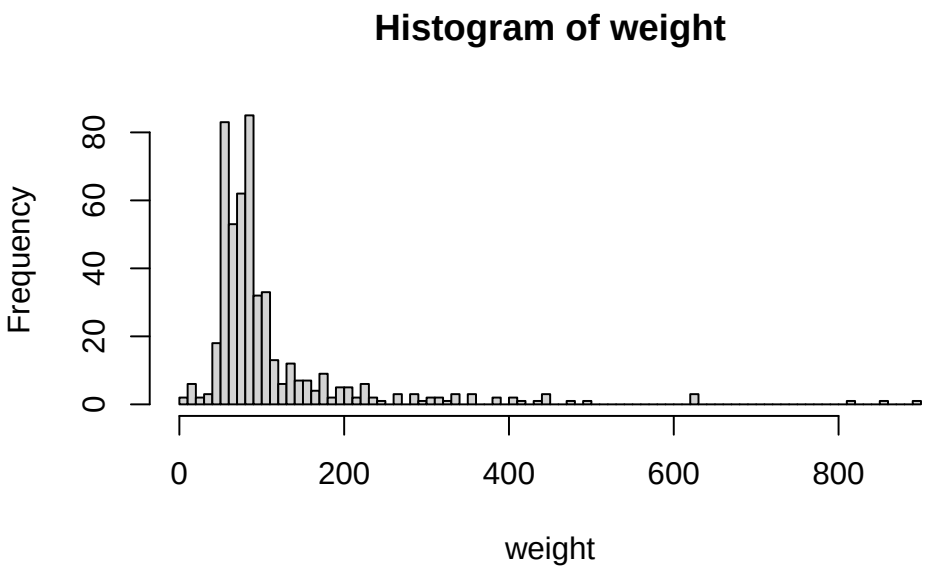
```
hist(weight, breaks = 4)
```




```
hist(weight, breaks = 30)
```




```
hist(weight, breaks = 100)
```



Попробовав различные значения, можно убедиться, что форма гистограммы может достаточно сильно изменяться в зависимости от выбранного количества интервалов[2].

***Для продвинутых:* правило Стёрджеса и иже с ними**

По умолчанию, функция `hist()` использует **правило Стёрджеса *(Sturges' rule)*** для определения количества интервалов гистограммы:

$$n = 1 + \lfloor \log_2 N \rfloor$$

Здесь N – размер выборки, а $\lfloor\rfloor$ обозначают целую часть числа (т.е. округление вниз). Если мы возьмем вектор `weight`, длина которого 495, то получим 9 интервалов:

```
1 + floor(log(length(weight), base = 2))
```

```
[1] 9
```

Правило Стёрджеса – только один из алгоритмов для расчета количества интервалов в гистограмме! Есть и множество других, например, **правило Фридмана–Диакониса *(Freedman–Diaconis' rule)*** и **правило Скотта**

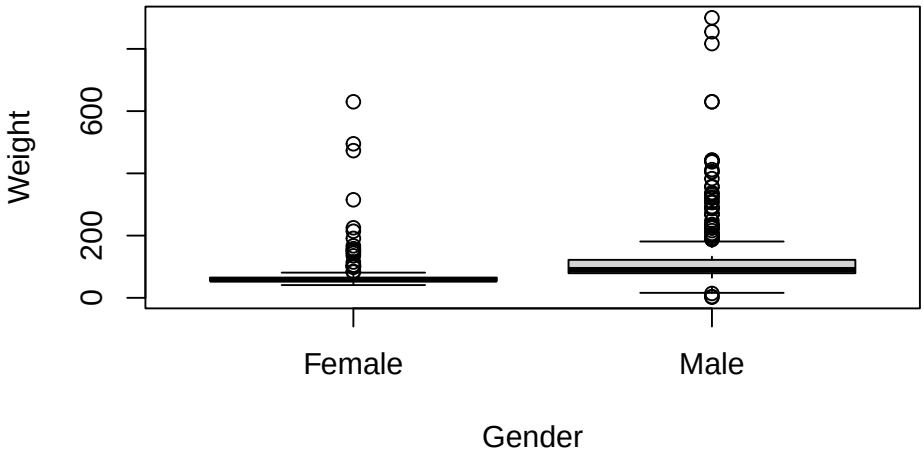
[2] На самом деле, функция `hist()` будет использовать заданное количество интервал очень ориентировочно: на основе желаемого количества интервалов будут вычисляться интервалы с круглыми границами.

> ***(Scott's normal reference rule).*** Чтобы ими воспользоваться в функции `hist()`, нужно прописать `breaks = "FD"` или `"Scott"` соответственно.

13.3 Нестареющий боксплот

Ну и закончим на суперзвезде прошлого века под названием **ящик с усами *(boxplot with whiskers)*:**

```
boxplot(Weight ~ Gender, heroes)
```



Здесь мы использовали уже знакомый нам класс формул. Они еще будут нам встречаться дальше, обычно они используются следующим образом: слева от ~ находится зависимая переменная, а справа - "предикторы". Эта интуиция работает и здесь: мы хотим посмотреть, как различается вес в зависимости от пола.

Несмотря на свою популярность, ящик с усами – достаточно непростой тип графиков. С одной стороны, он неплохо помогает понять, как распределены данные. С другой стороны, немногие знают, как эти ящики и усы рисуются.

Начнем с линии в середине ящика – это медиана (а не среднее, как могло бы показаться). Низ и верх ящика – это Q1 и Q3 соответственно. А это значит, что высота ящика – это разница между Q1 и Q3, т.е. межквартильный размах.

А вот с усами все еще сложнее. Они строятся следующим образом: от края ящика откладываются полтора межквартильных размаха. Но ус рисуется не на этой границе (1.5 IQR), а по крайней точке внутри полутора межквартильных размахов. Если остаются значения за пределами полутора межквартильных размахов, то они отмечаются точками.

Очень часто эти точки называют “выбросами”, однако это может запутать: нет никакого магического алгоритма, который назначает эти точки выбросами. Точнее, алгоритм есть (и мы его теперь знаем), но в нем нет никакой магии. Почему именно квартили и медиана? А, главное, почему именно полтора межквартильных размаха? Просто так решили. Могли бы взять и 1, и 2, и 1.4, и 1.645 межквартильных размаха, просто потому что (а еще число 1.5 – число более красивое, чем в 1.4 и 1.645). Это может показаться удивительным, что в статистике, области математики, используются взятые с потолка числа, но скоро мы убедимся, что такое встречается сплошь и рядом.

13.4 Заключение

Возможности R для визуализации очень богатые, и некоторые даже утверждают, что их более чем достаточно. Главное преимущество встроенных функций для графики в R – возможность быстро нарисовать простой график с помощью одной функции, не подключая никаких дополнительных пакетов. Это делает базовые функции визуализации в R удобными для исследования данных.

Глава 14

Грамматика графики {ggplot2}

14.1 История грамматики графики

Встроенные возможности для визуализации в R довольно обширны, но дополнительные пакеты значительно ее расширяют. Среди этих пакетов, есть один, который занимает совершенно особенное место – `{ggplot2}`.

`{ggplot2}` — это не просто пакет, который рисует красивые графики. Красивые графики можно рисовать и в базовом R. Чтобы понять, почему пакет `{ggplot2}` занимает особенное место среди пакетов для визуализации (и не только среди пакетов для R, а вообще!), нужно расшифровать `gg` в его названии. `gg` означает грамматику графики (Grammar of Graphics), язык для описания графиков, изложенный в одноименной книге Леланда Уилкинсона (Wilkinson 2005).

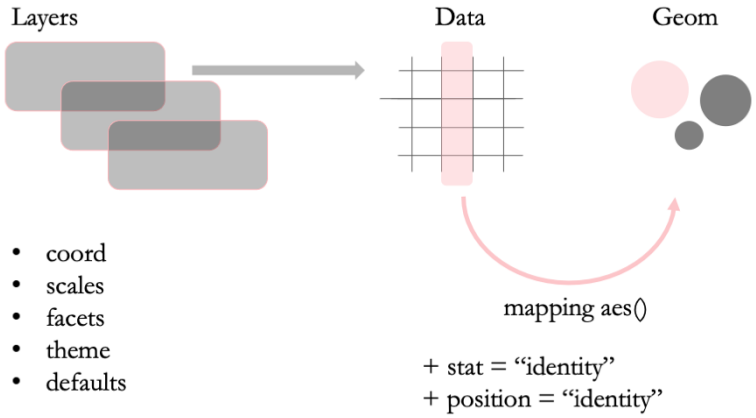
Грамматика графики позволяет описывать графики не в терминах типологии (вот есть пайчарт, есть барплот, есть гистограмма, а есть ящик с усами), а с помощью специального разработанного языка. Этот язык позволяет с помощью грамматики и небольшого количества “слов” языка описывать и создавать практически любые графики и даже придумывать новые! Это дает огромную свободу в создании именно той визуализации, что необходима для текущей задачи.

Хэдли Уикхэм (да, снова он) немного дополнил идею грамматики графики в статье “A Layered grammar of graphics” (Wickham

2010), которую сопроводил пакетом `ggplot2` с реализацией идей Уилкинсона и своих.

14.2 Основы грамматики графики

Каждый график состоит из одного или нескольких **слоев (layers)**. Если слоев несколько, то они располагаются один над другим, при этом верхние слои "перекрывают" нижние, примерно как это происходит в программах вроде Adobe Photoshop. У каждого слоя есть три обязательных элемента: **данные (data)**, **геом (geom)**, **эстетики (aestetics)**; и два вспомогательных: **статистические трансформации (stat).** и **регулировка положения (position adjustment).**



- **Данные (data).** Собственно, сами данные в виде датафрейма, используемые в данном слое.
- **Геом (geom).** Геом — это сокращение от "геометрический объект". Собственно, в какой геометрический объект мы собираемся превращать данные. Например, в точки, прямоугольники или линии.
- **Отображение (mapping).** Эстетические отображения или просто **эстетики (aestetics)** — это набор правил, как различные переменные превращаются в визуальные особенности геометрии. Без эстетик остается непонятно, какие именно колонки в используемом датафрейме превращаются

в различные особенности геомов: позицию, размер, цвет и т.д. У каждой геометрии свой набор эстетик, но многие из них совпадают у разных геомов, например, *x*, *y*, *colour*, *fill*, *size*. Без некоторых эстетик геом не будет работать. Например, геометрия в виде точек не будет работать без двух координат этих точек (*x* и *y*). Другие эстетики необязательны и имеют значения по умолчанию. Например, по умолчанию точки будут черными, но можно сделать их цвет зависимым от выбранной колонки в датафрейме с помощью эстетики *colour*.

- **Статистические трансформации (stat).** Название используемой статистической трансформации (или просто — статистики). Да, статистические трансформации можно делать прямо внутри `ggplot2`! Это дает дополнительную свободу в выборе инструментов, потому что обычно те же статистические трансформации можно сделать вне `ggplot2` в процессе препроцессинга. Формально, статистические трансформации — это обязательный элемент геома, но если вы не хотите преобразовывать данные, то можете выбрать "identity" преобразование, которое оставляет все как есть. В `ggplot2` у каждого геома есть статистика по умолчанию, а у каждой статистики - свой геом по умолчанию. И не всегда статистика по умолчанию — это "identity" статистика. Например, для барплота (`geom_barplot()`) используется статистика "count" [1], которая считает частоты, ведь именно частоты затем трансформируются в высоту барплотов.

- **Регулировка положения (position adjustment).** Регуляровка положения — это небольшое улучшение позиции геометрий для части элементов. Например, можно добавить немного случайного шума ("jitter") в позицию точек, чтобы они не перекрывали друг друга. Или "раздвинуть" ("dodge") два барплота, чтобы один не загораживал другой. Как и в случае со статистическими трансформациями, в большинстве случаев значение по умолчанию — "identity".

Кроме слоев, у графика есть:

- **Координатная система (coord).** Если мы задали координаты, то нам нужно задать и координатную плоскость, верно? Конечно, в большинстве случаев используется *декартова*

[1] идентична по своему смыслу функции `dplyr::count`, которая считает частоты по выбранной колонке тиббла @ref(tidy_count)

система координат (Cartesian coordinate system) [2], т.е. стандартная прямоугольная система координат, но можно использовать и другие, например, полярную систему координат или картографическую проекцию.

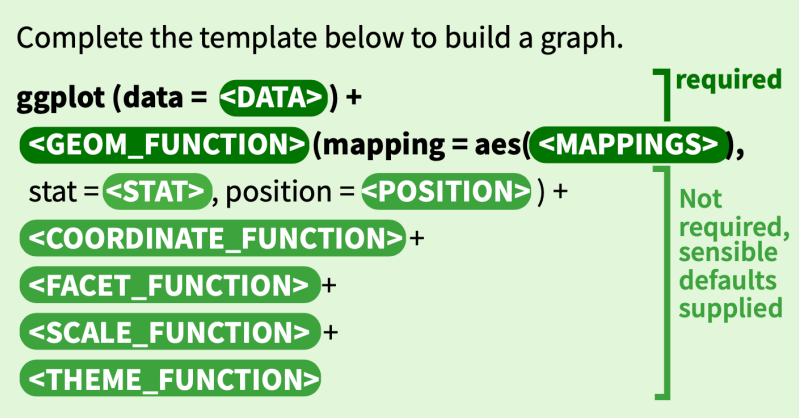
- **Шкалы (scales).** Шкалы задают то, *как именно* значения превращаются в эстетики. Например, если мы задали, что разные значения в колонке будут влиять на цвет точки, то какая именно палитра будет использоваться? В какие конкретно цвета будут превращаться числовые, логические или строковые значения в колонке? В `ggplot2` есть правила по умолчанию для всех эстестик, и они отличные, но самостоятельная настройка шкал может значительно улучшить график.
- **Фасетки (facets).** Фасетки — это одно из нововведений Уикхэма в грамматику графики. Фасетки повзоляют разбить график на множество похожих, задав переменную, по которой график будет разделен. Это очень напоминает использование группировки с помощью `group_by()`.
- **Тема (theme).** Тема — это зрительное оформление "подложки" графика, не относящийся к содержанию графика: размер шрифта, цвет фона, размер и цвет линий на фоне и т.д. и т.п. В `ggplot2` есть несколько встроенных тем, а также есть множество пакетов, которые добавляют дополнительные темы. Кроме того, их можно настраивать самостоятельно!
- **Значения по умолчанию (defaults).** Если в графике используется несколько слоев, то часто все они используют одни и те же данные и эстетики. Можно задать данные и эстетики по умолчанию для всего графика, чтобы не повторять код.

14.3 Пример №0: пайчарт с распределение по полу

Сейчас мы попробуем сделать простой пример в `ggplot2`, похожий на пример, который использует в своей книге Леланд Уилкинсон, чтобы показать мощь грамматики графики (Wilkinson 2005). Приготовьтесь, этот пример перевернет ваши представления о графиках!

[2] Декартова система координат названа в честь великого математика и философа Рене Декарта, на латинском — *Renatus Cartesius*, отсюда и название *cartesian coordinate system*.

Но сначала взглянем на структуру кода в `ggplot()`.



Как видно, код чем-то напоминает стандартный код tidyverse, но с + вместо пайпов. Когда был написан `ggplot2`, Хэдли Уикхэм еще не знал про %>% из `{magrittr}`, хотя по смыслу + означает примерно то же самое.

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become er
```

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
                   na = c("-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)

Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Итак, запустим функцию `ggplot()`, задав наш тиббл `heroes` в качестве данных.

```
ggplot(data = heroes)
```

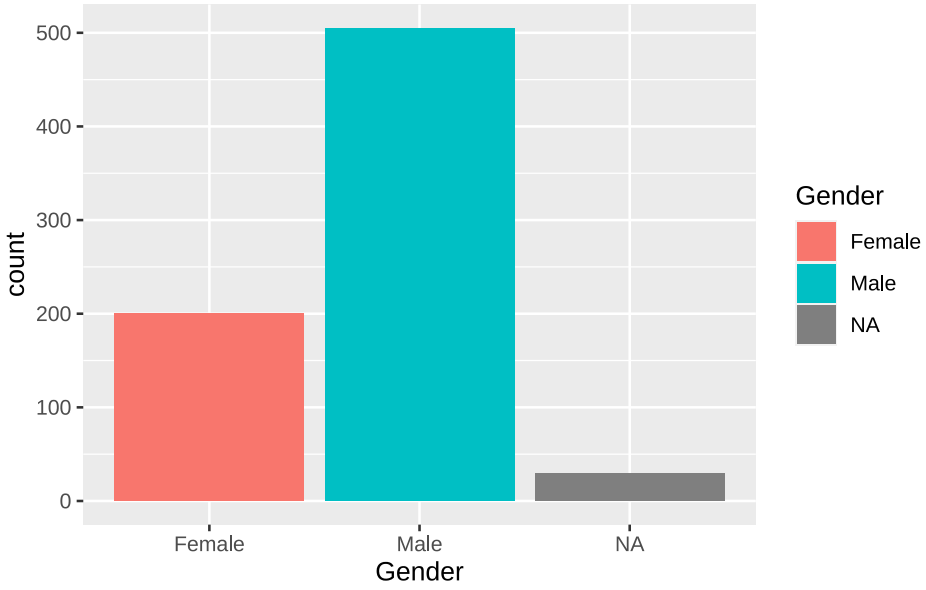

Мы ничего не получили! Это естественно, ведь мы задали только данные по умолчанию, но не задали геометрию и эстетики.

Функция `ggplot()` не просто отрисовывает график, эта функция создает объект класса `ggplot`, который можно сохранить и модифицировать в дальнейшем:

```
almost_empty_ggplot <- ggplot(data = heroes)
almost_empty_ggplot
```

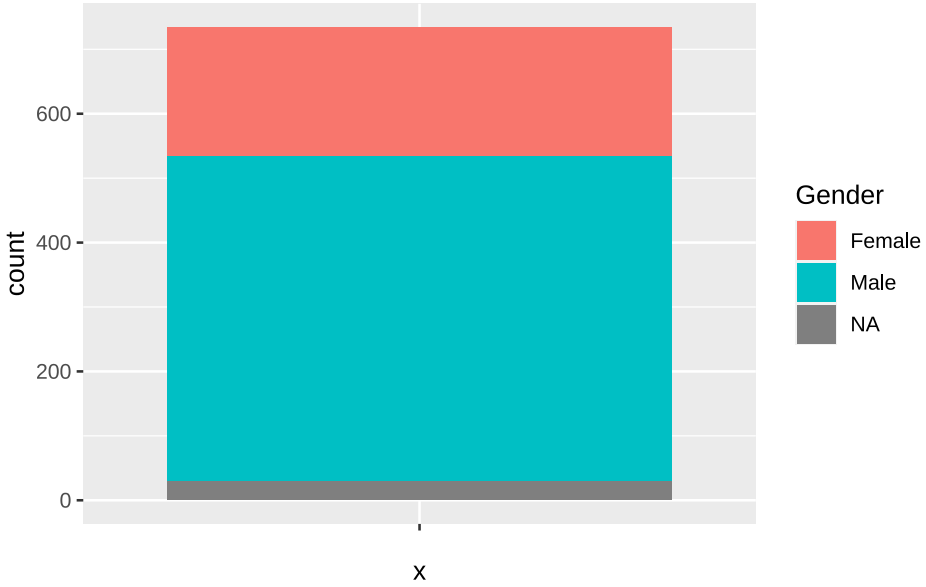

Возьмем `geom_bar()` для отрисовки барплота. В качестве эстетик поставим `x = Gender` и `fill = Gender`. Поскольку это эстетики, они обозначаются внутри функции параметра `mapping = aes()` или просто внутри функции `aes()`. По умолчанию, `geom_bar()` имеет статистику “count”, что нас полностью устраивает: `geom_bar()` сам посчитает табличу частот и использует значения `Gender` для обозначения позиций и заливки, а посчитанные частоты будет использовать для задания высоты столбцов.

```
ggplot(data = heroes) +
  geom_bar(aes(x = Gender, fill = Gender))
```



Сейчас мы сделаем один хитрый трюк: поставим значение эстетики `x = ""`, чтобы собрать все столбики в один.

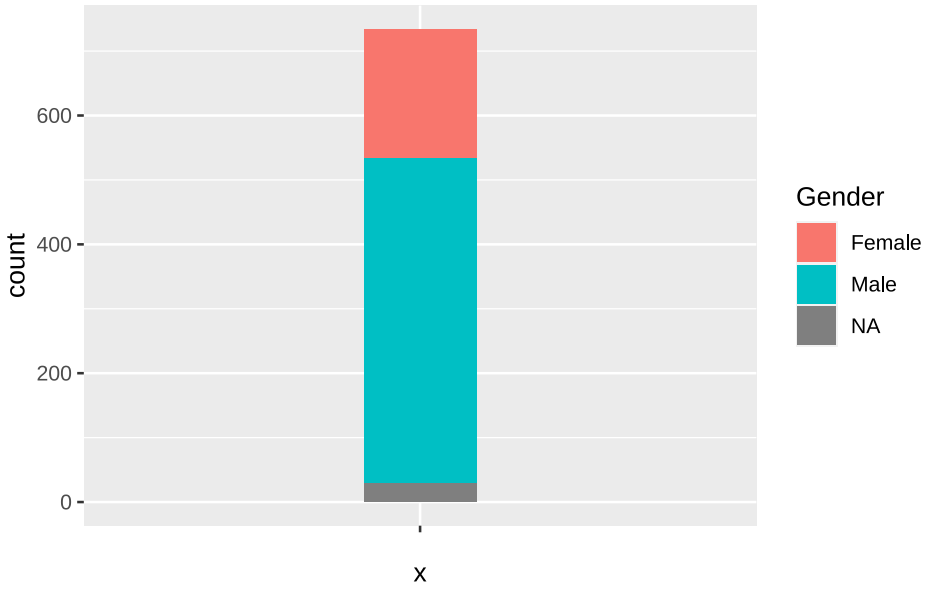
```
ggplot(data = heroes) +
  geom_bar(aes(x = "", fill = Gender))
```



Получилось что-то не очень симпатичное, но вполне осмысленное: доли столбца обозначают относительную частоту.

Можно настроить общие параметры геома, не зависящие от данных. Это нужно делать *вне* функции `aes()`, но внутри функции для геома.

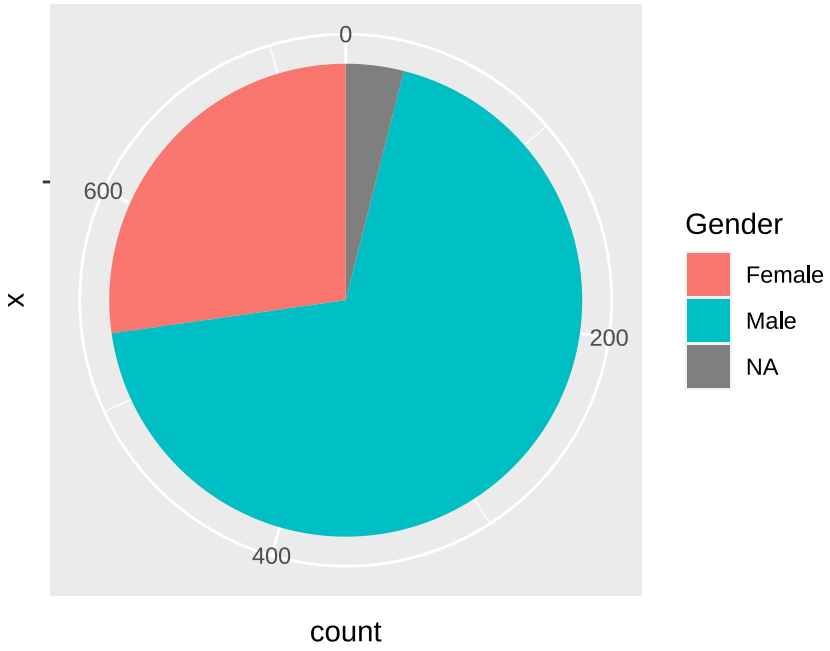
```
ggplot(data = heroes) +
  geom_bar(aes(x = "", fill = Gender), width = .2)
```



Казалось бы, причем здесь Minecraft?

А теперь внимание! Подумайте, какого действия нам не хватает, чтобы из имеющегося графика получить пайчарт?

```
ggplot(data = heroes) +
  geom_bar(aes(x = "", fill = Gender)) +
  coord_polar(theta = "y")
```

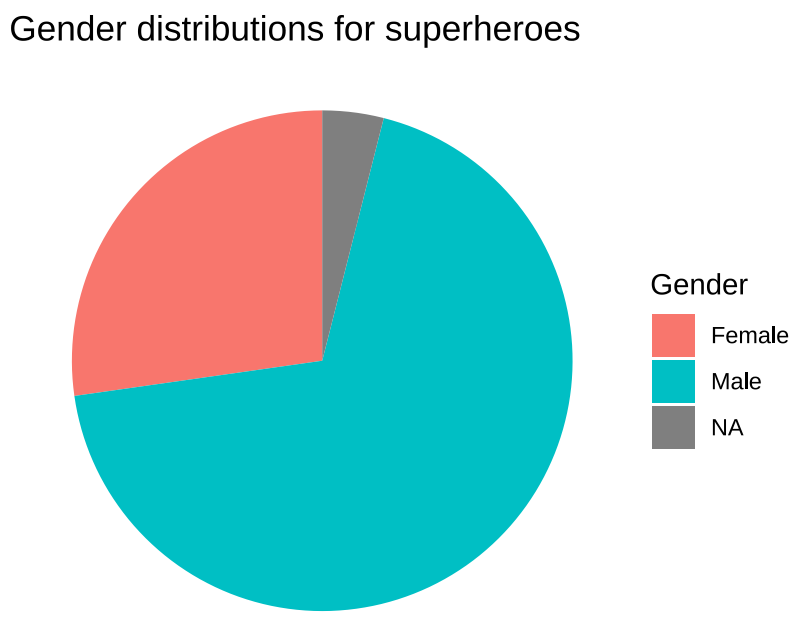


Нам нужно было всего-лишь поменять систему координат с декартовой на полярную (круговую)! Иначе говоря, пайчарт - это барплот в полярной системе координат.

Именно в этом основная сила грамматики графики и ее реализации в `ggplot2` — вместо того, чтобы описывать и рисовать огромное количество типов графиков, можно описать практически любой график через небольшой количество элементарных элементов и правила их соединения.

Получившийся пайчарт осталось подретушировать, убрав все лишние элементы подложки с помощью самой минималистичной темы `theme_void()` и добавив название графика:

```
ggplot(data = heroes) +
  geom_bar(aes(x = "", fill = Gender)) +
  coord_polar(theta = "y") +
  theme_void() +
  labs(title = "Gender distributions for superheroes")
```



Это был интересный, но немного шуточный пример. Все-таки пайчарт — это довольно спорный способ визуализировать данные, вызывающий много вполне справедливой критики. Поэтому сейчас мы перейдем к гораздо более реалистичному примеру.

14.4 Пример №1: Education and IQ meta-analysis

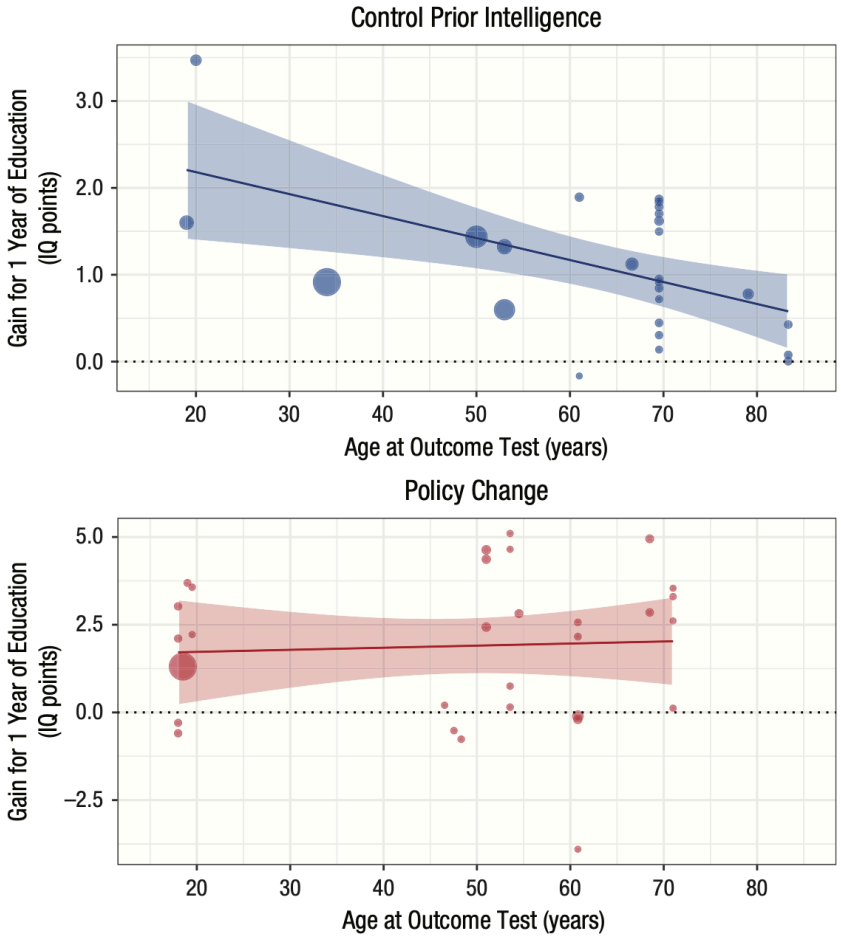
Для этого примера мы возьмем мета-анализ связи количества лет обучения и интеллекта: *"How Much Does Education Improve Intelligence? A Meta-Analysis"* (Ritchie и Tucker-Drob 2018). Мета-анализ — это группа статистических методов, которые позволяют объединить результаты нескольких исследований с похожим планом исследованием и тематикой, чтобы посчитать средний эффект между несколькими статьями сразу.

Данные и скрипт для анализа данных в этой статье находятся в открытом доступе: https://osf.io/r8a24/

Полный текст статьи доступен по ссылке.

Существует положительная корреляция между количеством лет, который человек потратил на обучение, и интеллектом. Это

может объясняться по-разному: как то, что обучение повышает интеллект, и как то, что люди с высоким интеллекте стремятся получать больше образования. Напрямую в эксперименте это проверить нельзя, поэтому есть несколько квази-экспериментальных планов, которые косвенно указывают на верность той или иной гипотезу. Например, если в стране изменилось количество лет обязательного школьного образования, то повлияло ли это на интеллект целого поколения? ~~Или все-таки дело в Моргенштерне~~



Данная картинка показывает, насколько размер эффекта (выраженный в баллах IQ) зависит от того, какой средний возраст участвоваших в исследовании испытуемых.

Каждая точка на этом графике — это отдельное исследование, положение по оси *x* — средний возраст респондентов, а положение по оси *y* - средний прирост интеллекта согласно

исследованию. Размер точки отражает “точность” исследования (грубо говоря, чем больше выборка, тем больше точка). Два графика обозначают два квазиэкспериментальных плана.

Мы сфокусируемся на нижней картинке с “Policy change” — это как раз исследования, в которых изучается изменения интеллекта в возрастных группах после изменения количества лет обучения в школе.

Мы полностью воспроизведем код построчно, посмотрим, как эта картинка создавалась шаг за шагом.

Заметьте, данный датасет использует немного непривычный для нас формат хранения данных. Попытайтесь самостоятельно прочитать его.

```
library(tidyverse)
df <- read_tsv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/meta_datas
```

```
Rows: 142 Columns: 17
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr  (2): Country, Design
dbl (15): Study_ID, Data_ID, k, n, outcome_test_cat, Effect_size, SE, Outcom...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Давайте посмотрим, как устроен датафрейм `df`:

```
df
```

```
# A tibble: 142 x 17
   Study_ID Data_ID     k Country     n Design      outcome_test_cat Effect_size
      <dbl>   <dbl> <dbl> <chr>   <dbl> <chr>                  <dbl>       <dbl>
 1      1         1     4 UK        483 Control Pr~                0     0.778
 2      1         1     4 UK        290 Control Pr~                1     0.0771
 3      1         1     4 UK        290 Control Pr~                1     0.427
 4      1         1     4 UK        288 Control Pr~                1     0.00519
 5      1         2    13 UK       1017 Control Pr~                0     1.62
 6      1         2    13 UK       1022 Control Pr~                1     0.915
 7      1         2    13 UK       1021 Control Pr~                1     0.305
 8      1         2    13 UK        981 Control Pr~                1     0.719
 9      1.5       2    13 UK       1024 Control Pr~                1     1.70
10      1.5       2    13 UK       1023 Control Pr~                1     1.78
```

```
# i 132 more rows
# i 9 more variables: SE <dbl>, Outcome_age <dbl>, quasi_age <dbl>,
#   cpiq_early_age <dbl>, cpiq_age_diff <dbl>, ses_funnel <dbl>,
#   published <dbl>, Male_only <dbl>, Achievement <dbl>
```

Каждая строчка — это результат отдельного исследования, при этом одна статья может включать несколько исследований,

В дальнейшем мы будем использовать код авторов статьи и смотреть, строчка за строчкой, как он будет работать.

```
cpiq <- subset(df, subset=(Design=="Control Prior IQ"))
poli <- subset(df, subset=(Design=="Policy Change"))
```

Авторы исследования используют `subset()`, это функция базового R, принцип которой очень похож на `filter()` [3].

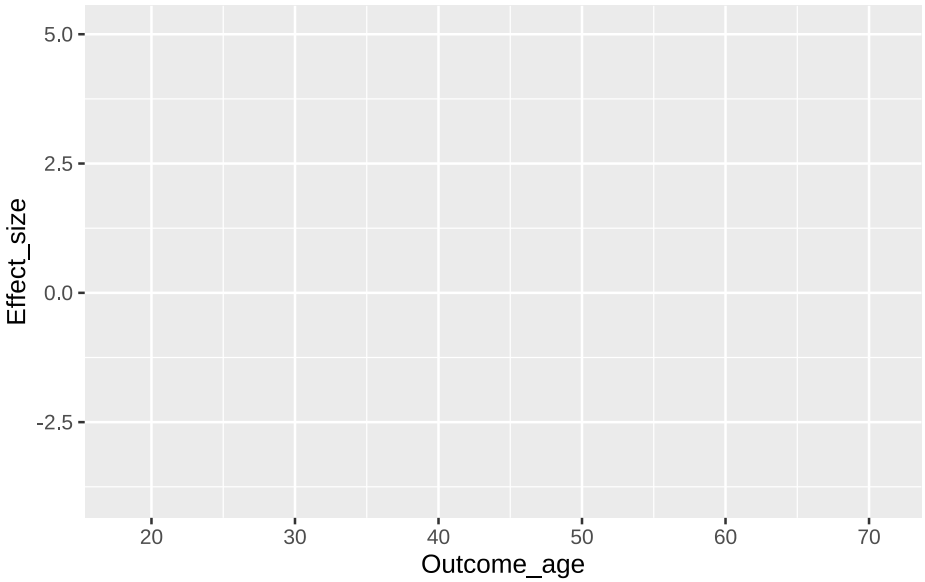
Итак, начнем рисовать сам график. Сначала инициируем объект `ggplot` с данными `poli` по умолчанию.

```
ggplot(data=poli)
```


Теперь добавим в качестве эстетик по умолчанию координаты: `aes(x=Outcome_age, y=Effect_size)`.

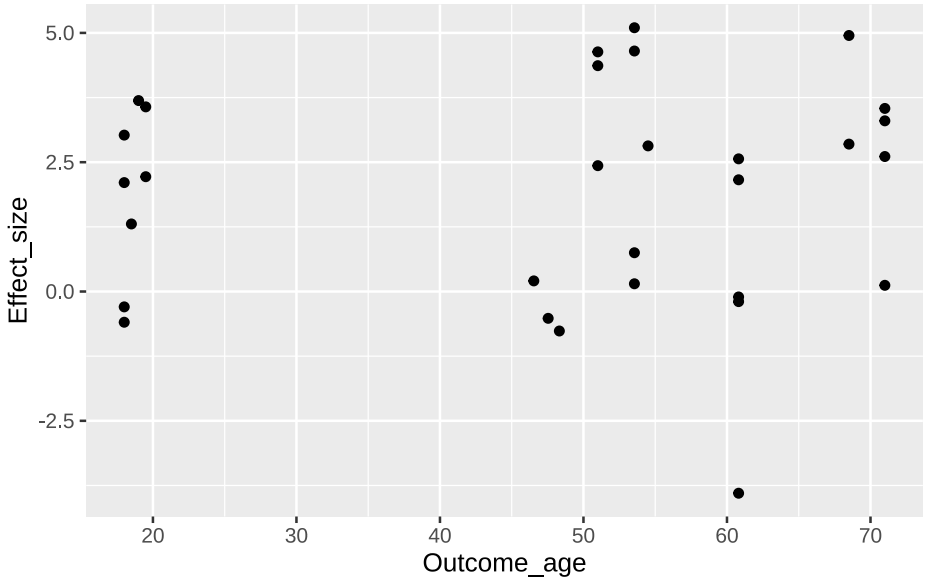
[3] Кстати, именно функция `subset()` вдохновила Уикхема на создание `filter()`.

```
ggplot(aes(x=Outcome_age, y=Effect_size), data=poli)
```



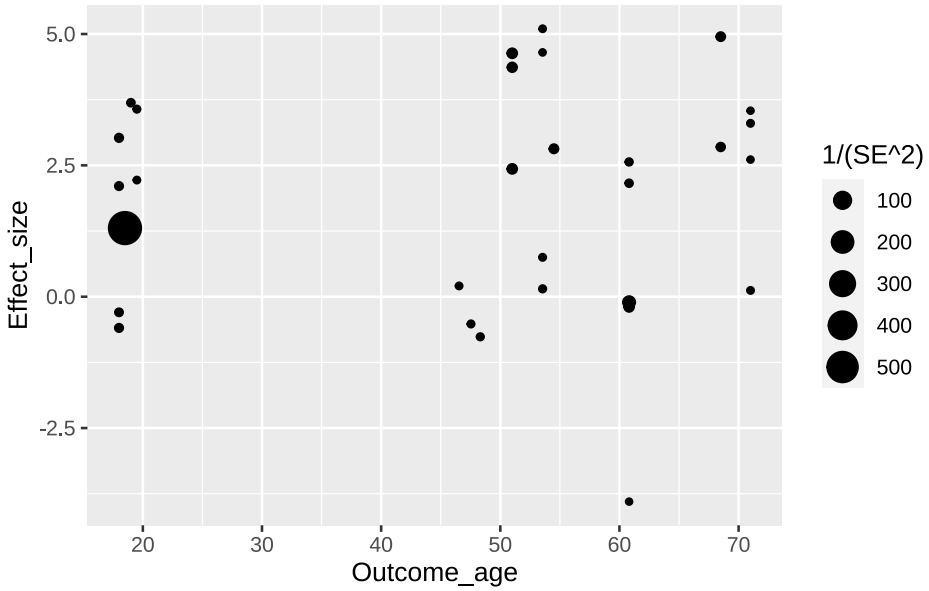
Что изменилось? Появилась координатная ось и шкалы. Заметьте, масштаб неслучаен: он строится на основе разброса значений в выбранных колонках. Однако этого недостаточно для отрисовки графика, нехватает геометрии: нужно задать, в какую географическую сущность отобразятся данные.

```
ggplot(aes(x=Outcome_age, y=Effect_size), data=poli) +
        geom_point()
```



Готово! Это и есть основа картинки. Добавляем размер:

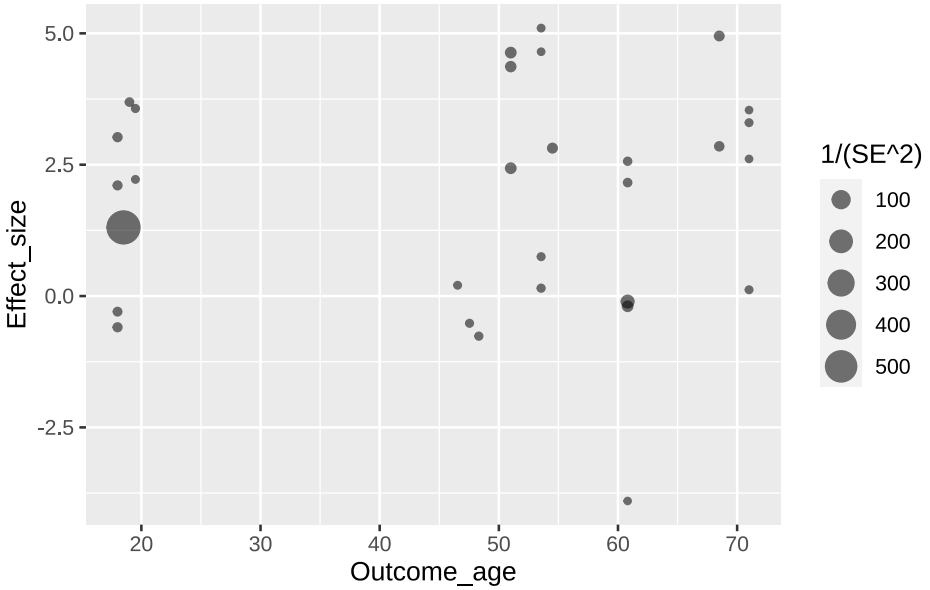
```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point()
```



Перед нами возникла проблема оверплоттинга: некоторые

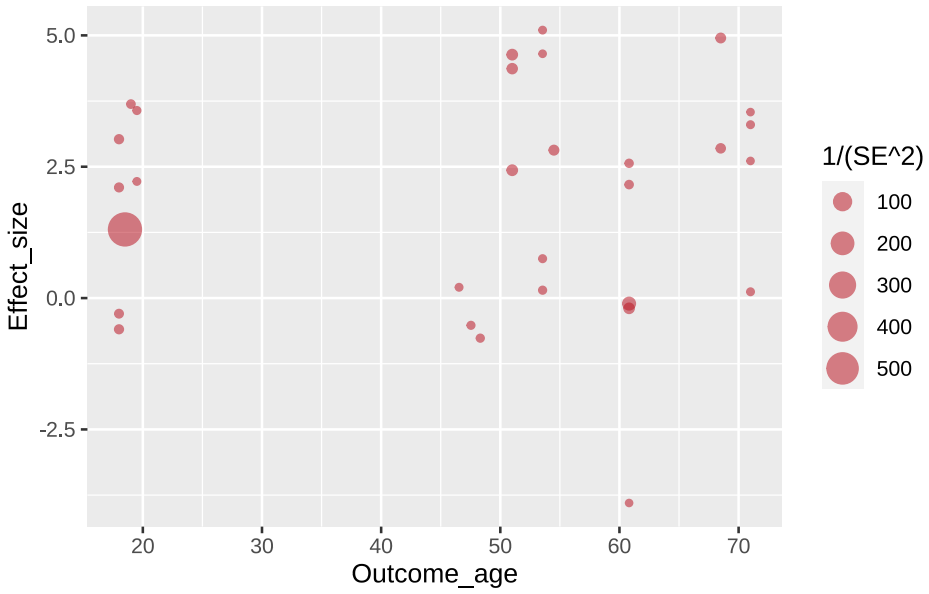
точки перекрывают друг друга, поскольку имеют очень близкие координат. Авторы графика решают эту проблему очевидным способом: добавляют прозрачности точкам. Заметьте, прозрачность задается для всех точек одним значением, поэтому параметр `alpha` задается вне функции `aes()`.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55)
```



Совершенно так же задается и цвет. Он задается одинаковым для всех точек с помощью HEX-кода.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825")
```



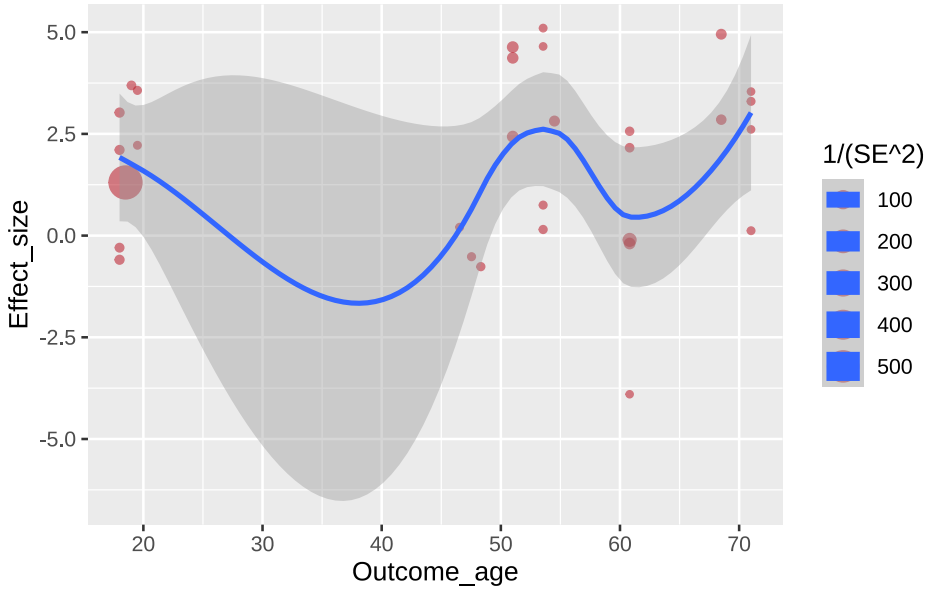
Теперь добавим регрессионную прямую с доверительными интервалами на график. Это специальный геом `geom_smooth()` со специальной статистикой, который займет второй слой данного графика.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_smooth()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: The following aesthetics were dropped during statistical transformation: size
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```

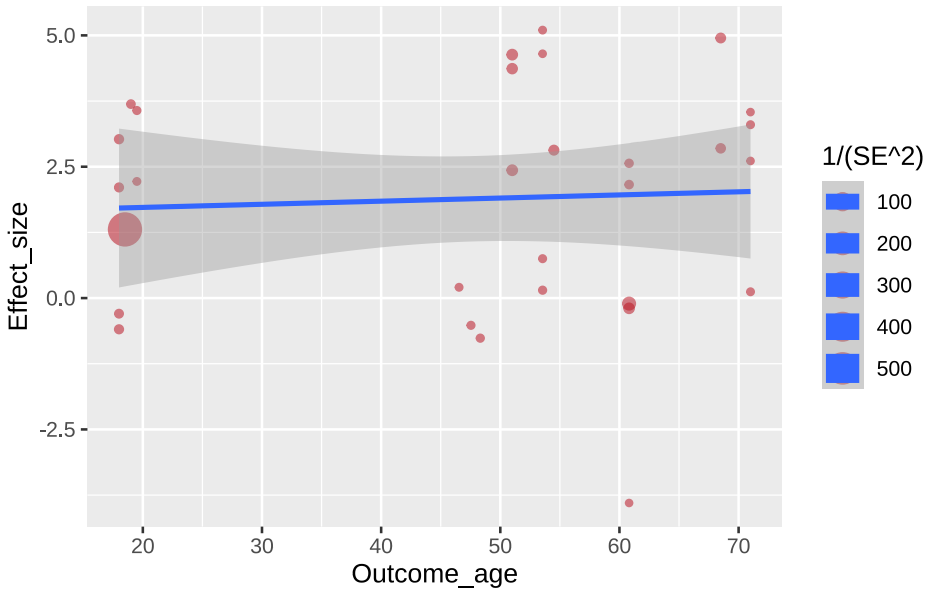


По умолчанию `geom_smooth()` строит кривую линию. Поставим `method = "lm"` для прямой.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_smooth(method="lm")
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: The following aesthetics were dropped during statistical transformation: size
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```

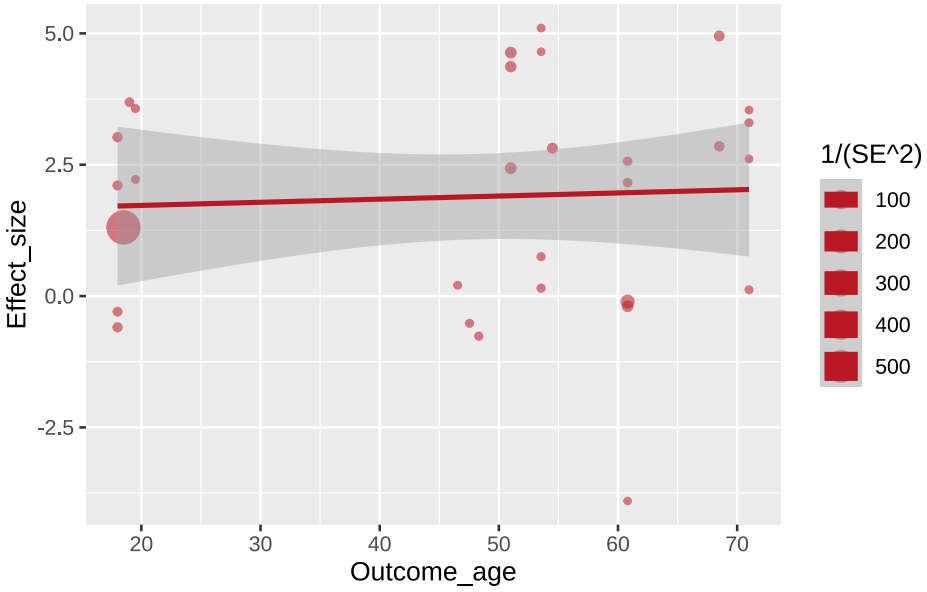


Теперь нужно поменять цвет: ярко синий цвет, используемый по умолчанию здесь попросту мешает восприятию графика.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_smooth(method="lm", colour="#BA1825")
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: The following aesthetics were dropped during statistical transformation: size
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```

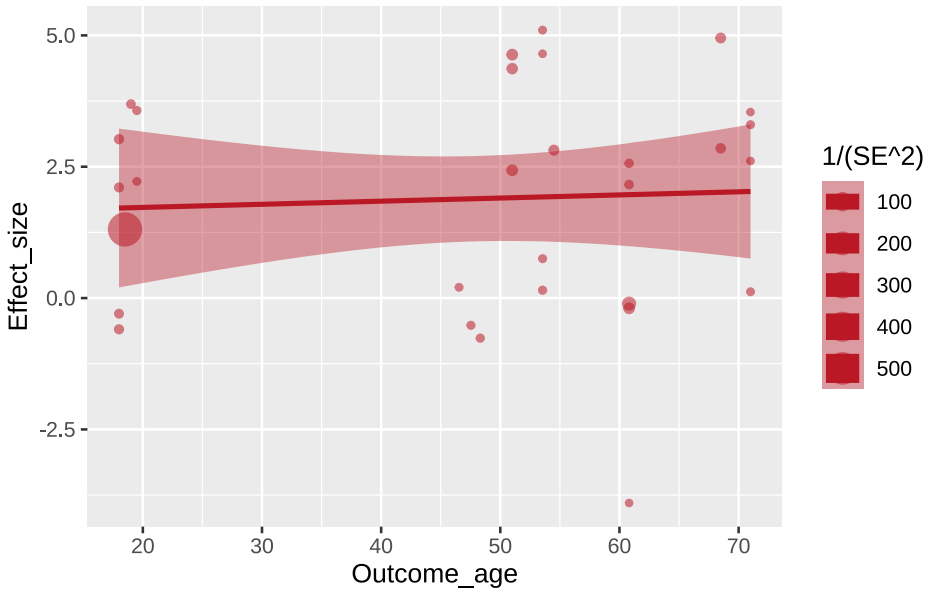


Авторы графика перекрашивают серую полупрозначную область тоже. В этом случае используется параметр `fill =`, а не `colour =`, но цвет используется тот же.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825")
```

```
`geom_smooth()` using formula = 'y ~ x'
```

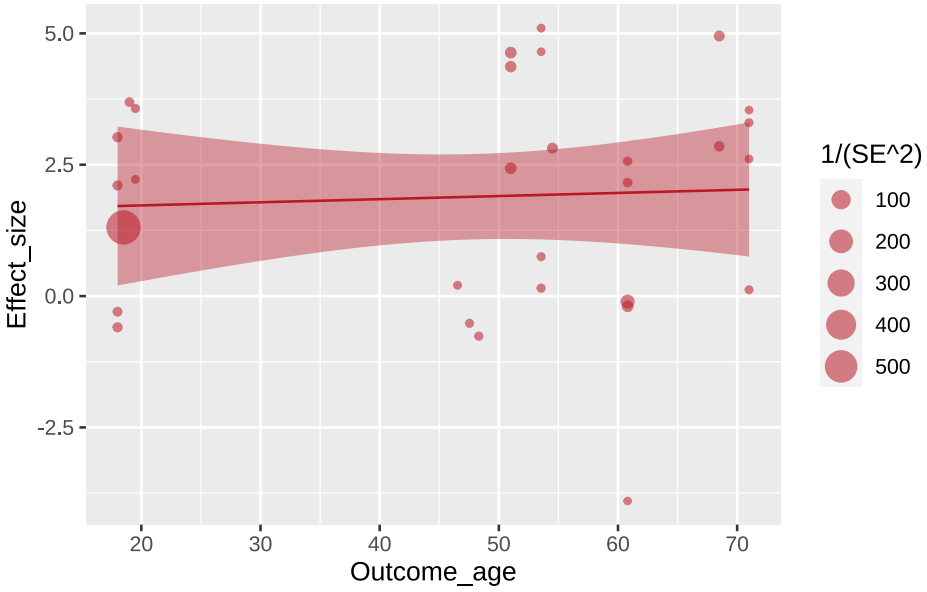
```
Warning: The following aesthetics were dropped during statistical transformation: size
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```



Регрессионную линию авторы немного утоньшают с помощью параметра `size =`.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5)
```

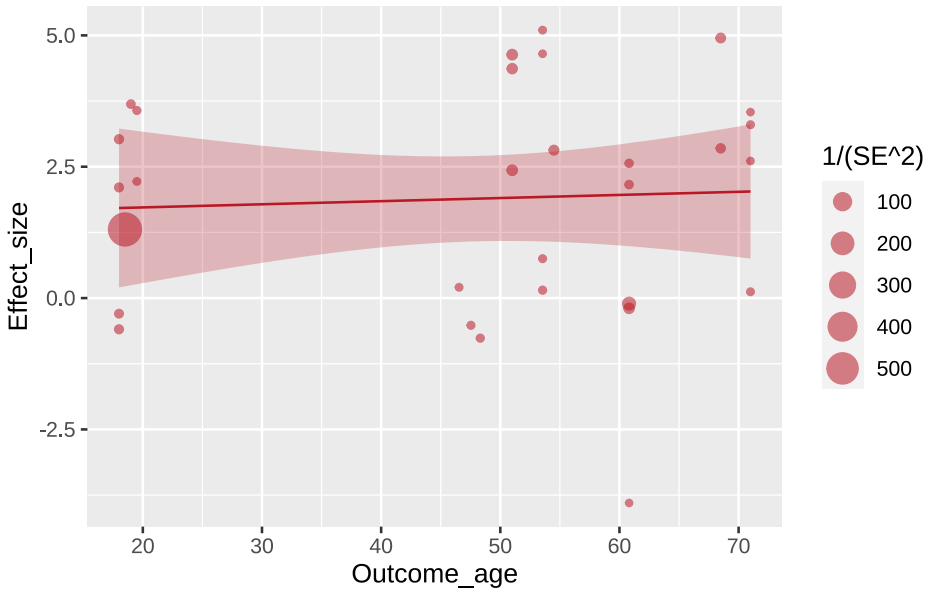
```
`geom_smooth()` using formula = 'y ~ x'
```



Чтобы сместить фокус в сторону точек, авторы добавляют прозрачности для всего `geom_smooth()`.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
```

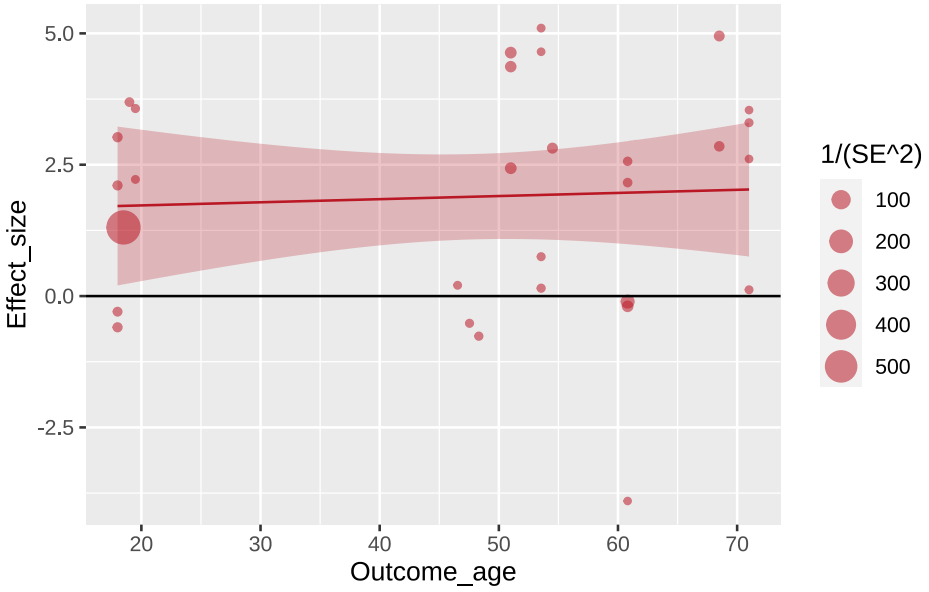
```
`geom_smooth()` using formula = 'y ~ x'
```



На шкале присутствует 0, и по умолчанию он никак не обозначен. Это легко исправить с помощью вспомогательного геома `geom_hline()`.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
```

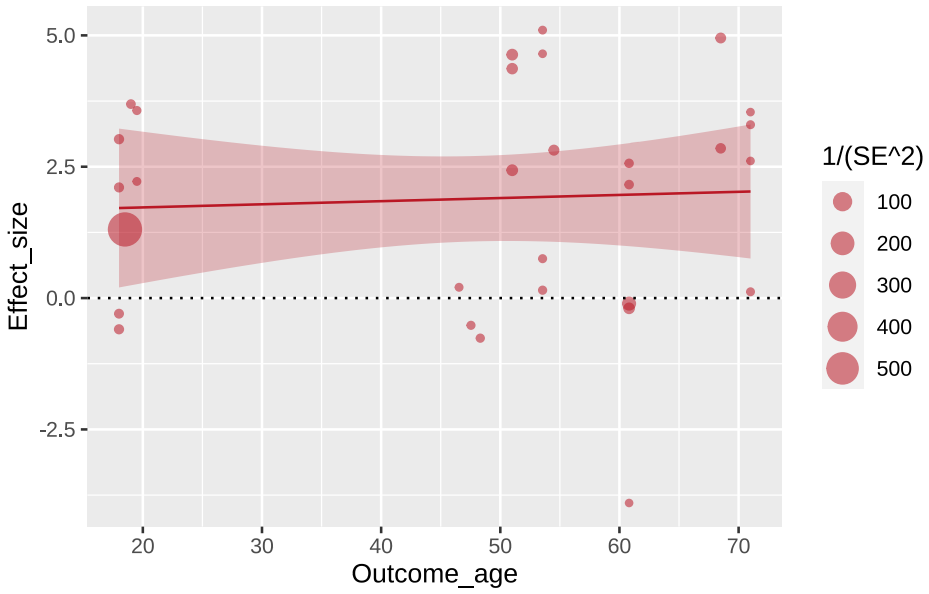
```
`geom_smooth()` using formula = 'y ~ x'
```



Оттенить эту линию можно, сделав ее пунктирной.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
```

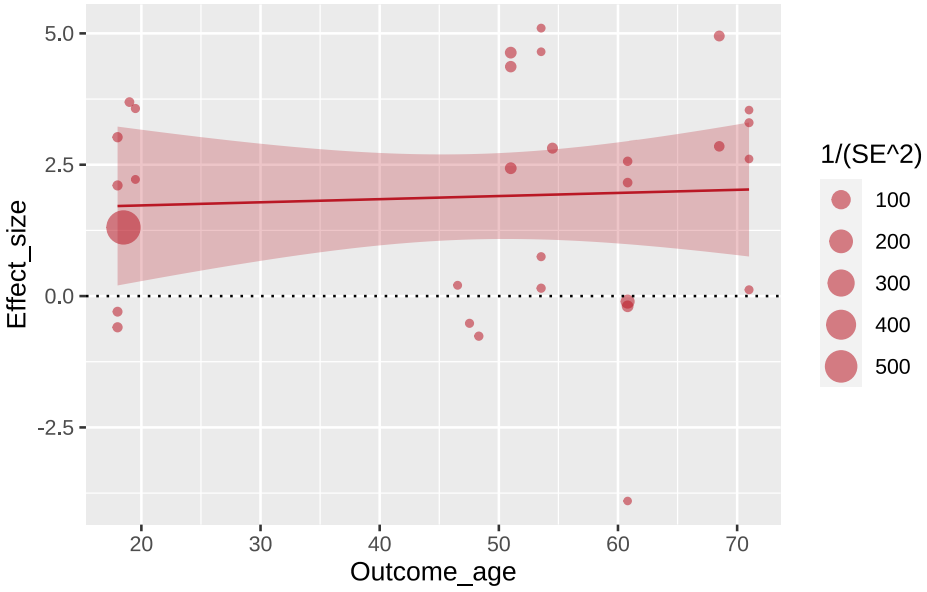
```
`geom_smooth()` using formula = 'y ~ x'
```



Авторы графика вручную задают деления шкалы по оси *x*.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

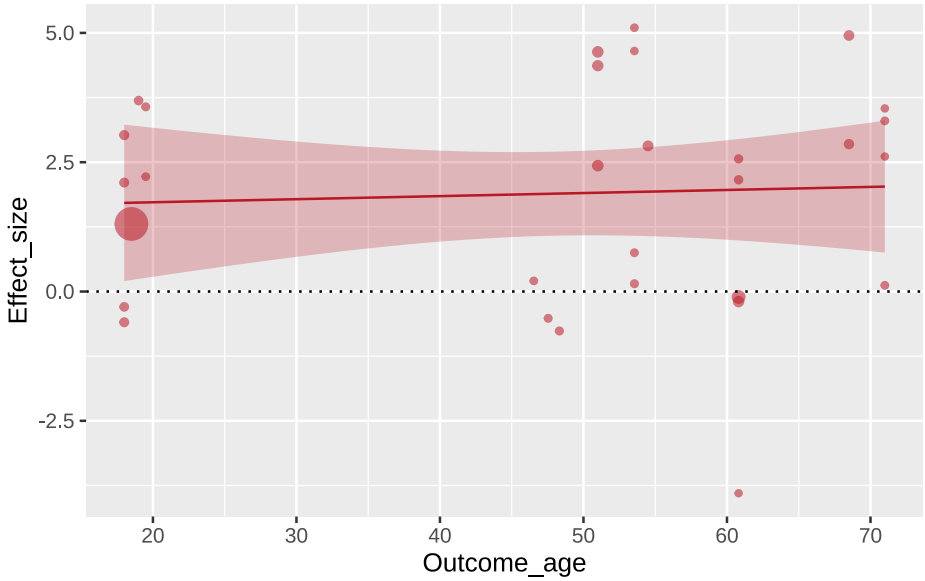


С помощью функции `guides()` убирают легенду с картинки.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
```

```
Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as
of ggplot2 3.3.4.
```

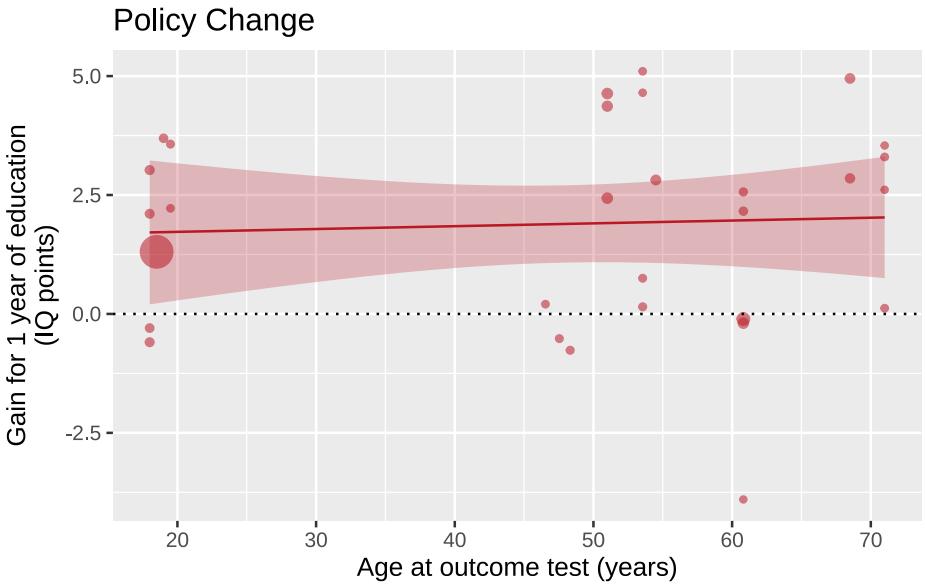
```
`geom_smooth()` using formula = 'y ~ x'
```



Следующим этапом авторы добавляют подписи шкал и название картинки. Обратите внимание на `\n` внутри подписи к оси *y*, которая задает перенос на следующую строку.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
```

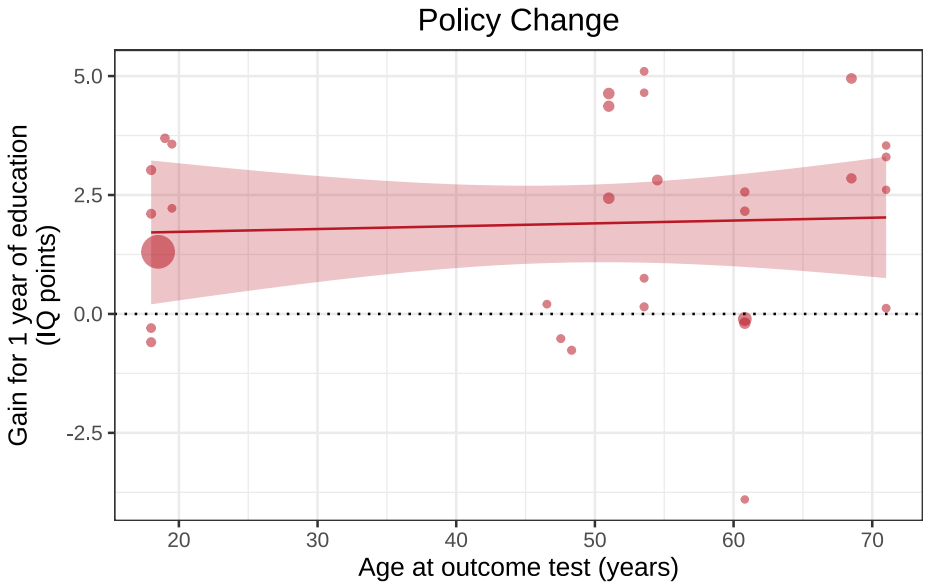
```
`geom_smooth()` using formula = 'y ~ x'
```



Теперь пришло время сделать график более красивым и понятным с помощью изменения подложки, т.е. работы с темой графика. Здесь тема задается сначала как `theme_bw()` — встроенная в `ggplot2` минималистичная тема, а потом через функцию `theme()`, через которую можно управлять конкретными элементами темы. Здесь это сделано, чтобы передвинуть название графика к центру.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=poli) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25) + ggtitle
        theme(plot.title = element_text(hjust=0.5))
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Готово! Мы полностью воспроизвели график авторов статьи с помощью их открытого кода.

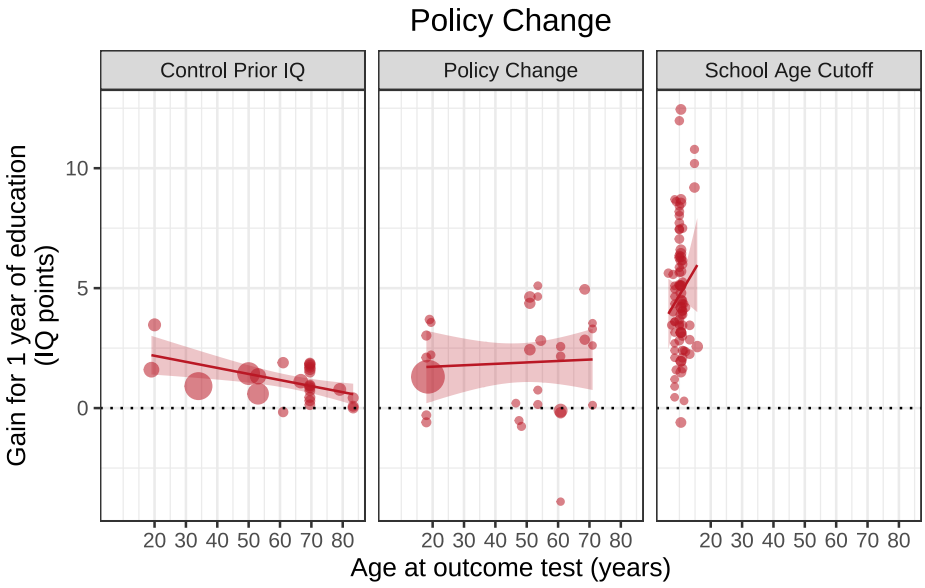
Если вы помните, то в изначальном графике было две картинки. Авторы делают их отдельно, с помощью почти идентичного кода. Нечто похожее можно сделать по-другому, применяя фасетки.

Для этого мы возьмем неотфильтрованный датасет `df`, а с помощью колонки `Design`, на основании которой разделялся датасет для графиков, произведем разделение графиков внутри самого `ggplot` объекта. Для этого нам понадобится функция `facet_wrap()`, в которой с помощью формулы можно задать колонки, по которым будут разделены картинки по вертикали (слева от ~) и горизонтально (справа от ~). Пробуем разделить графики горизонтально:

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=df) +
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
        theme(plot.title = element_text(hjust=0.5)) +
```

```
    facet_wrap(~Design)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Здесь становится очевидно, почему авторы не включали данные `"School Age Cutoff"` третьим графиком: средний возраст участников этих исследований сильно отличается.

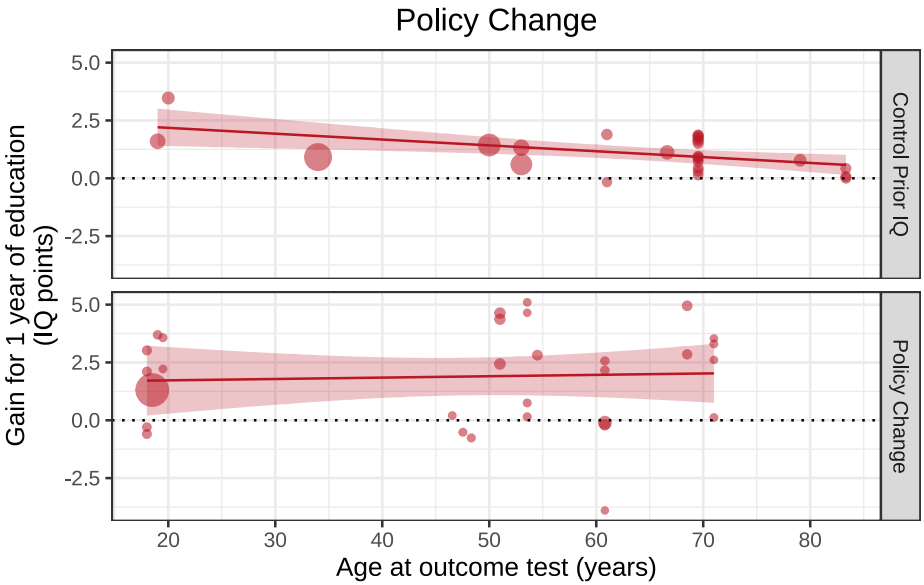
```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=df %>% filter(Design != "School
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25) + ggtitle
        theme(plot.title = element_text(hjust=0.5)) +
    facet_wrap(~Design)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

Теперь поставим два графика друг над другом, поместив `Design` слева от `~` внутри `facet_wrap()`. Справа нужно добавить точку.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=df %>% filter(Design !
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25)
        theme(plot.title = element_text(hjust=0.5)) +
    facet_grid(Design~.)
```

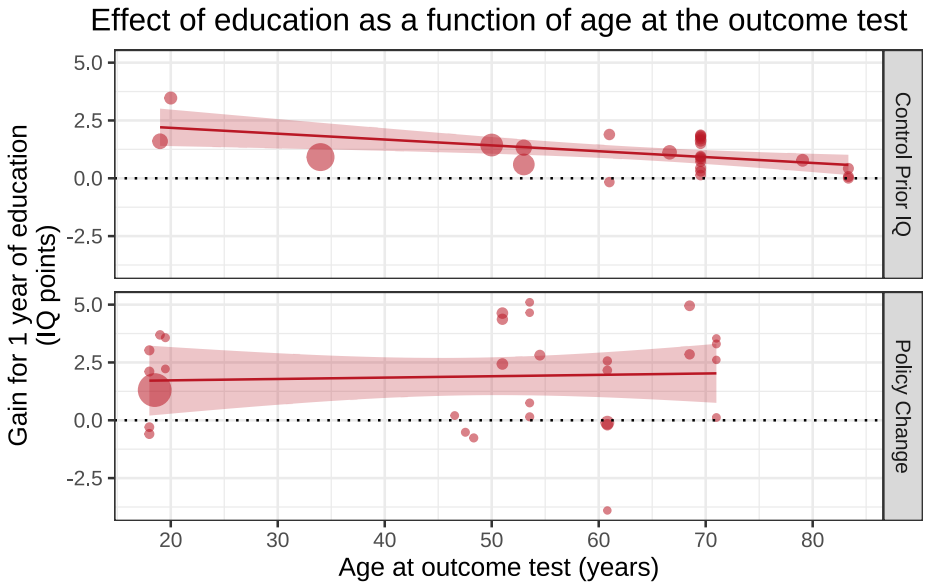
```
`geom_smooth()` using formula = 'y ~ x'
```



Теперь нужно изменить подписи.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2)), data=df %>% filter(Design != "School
        geom_point(alpha=.55, colour="#BA1825") +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=F) +
        geom_smooth(method="lm", colour="#BA1825",fill="#BA1825",size=.5, alpha=.25) +
    ggtitle("Effect of education as a function of age at the outcome test")+
        theme(plot.title = element_text(hjust=0.5)) +
    facet_grid(Design~.)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

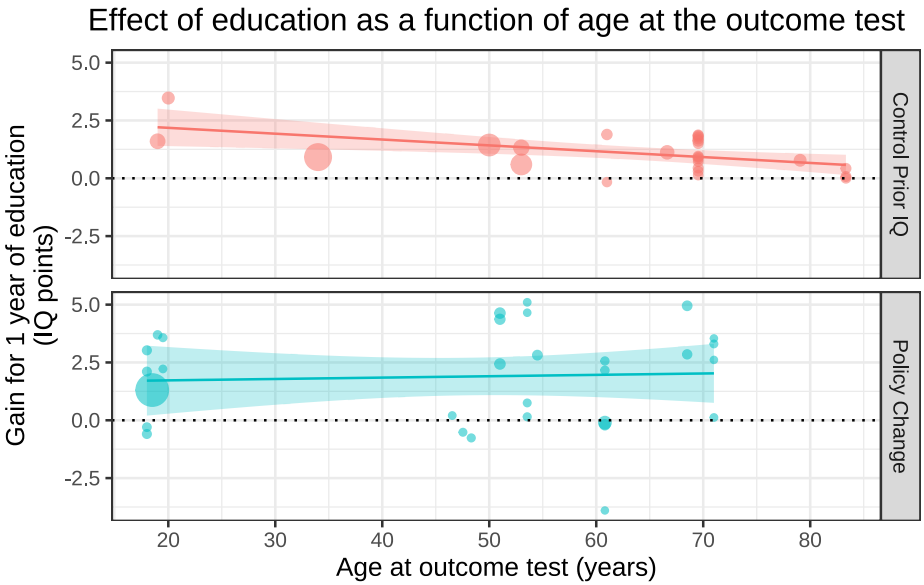


Чтобы акцентировать графики, можно раскрасить их в разные цвета в дополнение к фасеткам. Для этого мы переносим `colour =` и `fill =` из параметров соответствующих геомов внутрь эстетик и делаем зависимыми от `Design`. Поскольку эти эстетики (точнее, `colour =`) одинаковы заданы для двух геомов (`geom_point()` и `geom_smooth()`), то мы спокойно можем вынести их в эстетики по умолчанию — в параметры `aes()` внутри `ggplot()`.

При этом сразу выключим легенды для новых эстетик, потому они избыточны.

```
ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2), colour = Design, fill = Desi
        geom_point(alpha=.55) +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=FALSE, colour = FALSE, fill = FALSE) +
        geom_smooth(method="lm", size=.5, alpha=.25) +
    ggtitle("Effect of education as a function of age at the outcome test")+
        theme(plot.title = element_text(hjust=0.5)) +
    facet_grid(Design~.)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

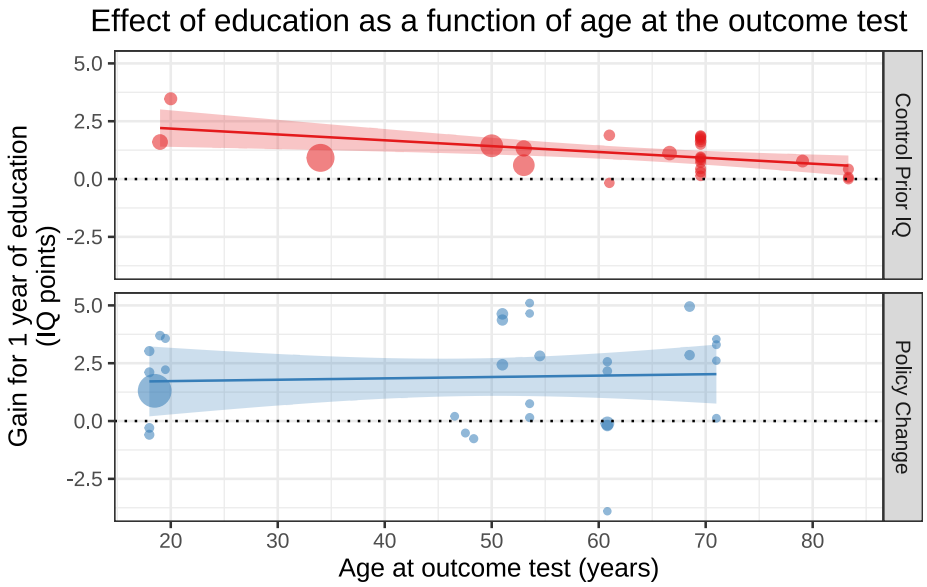


Слишком блеклая палитра? Не беда, можно задать палитру вручную! В `ggplot2` встроены легендарные *Brewer's Color Palettes*, которыми мы и воспользуемся.

Функции для шкал устроены интересным образом: они состоят из трех слов, первое из которых `scale_*_*()`, второе — эстетика, например, `scale_color_*()`, а последнее слово — тип самой шкалы, в некоторых случаях - специальное название для используемой шкалы, как и в случае с `scale_color_brewer()`.

```
meta_2_gg <- ggplot(aes(x=Outcome_age, y=Effect_size, size=1/(SE^2), colour = Design, fill = D
        geom_point(alpha=.55) +
        geom_hline(yintercept=0, linetype="dotted") +
        theme_bw() +
        scale_x_continuous(breaks=c(20,30,40,50,60,70,80)) +
        xlab("Age at outcome test (years)") +
        ylab("Gain for 1 year of education\n(IQ points)") +
        guides(size=FALSE, colour = FALSE, fill = FALSE) +
        geom_smooth(method="lm", size=.5, alpha=.25) +
    ggtitle("Effect of education as a function of age at the outcome test")+
        theme(plot.title = element_text(hjust=0.5)) +
    facet_grid(Design~.)+
    scale_colour_brewer(palette = "Set1")+
    scale_fill_brewer(palette = "Set1")
meta_2_gg
```

```
`geom_smooth()` using formula = 'y ~ x'
```



14.5 Расширения {ggplot2}

ggplot2 стал очень популярным пакетом и быстро обзавелся расширениями - пакетами R, которые являются надстройками над ggplot2. Эти расширения бывают самого разного рода, например, добавляющие дополнительные геомы или просто реализующие отдельные типы графиков на языке ggplot2.

Я рекомендую посмотреть самостоятельно галерею расширений ggplot2: https://exts.ggplot2.tidyverse.org/gallery/

Больше сотни пакетов! Это самые разные пак

Для примера мы возьмем пакет {hrbrthemes}, который предоставляет дополнительные темы для {ggplot2}, компоненты тем и шкалы.

```
install.packages("hrbrthemes")
```

```
library(hrbrthemes)
meta_2_gg +
  theme_ipsum()
```

Глава 15

Динамические визуализации в R

15.1 Интерфейс для JavaScript фреймворков: пакет `htmlwidgets`

До этого мы делали только статические картинки, но в R можно делать динамические визуализации с интерактивными элементами! Делаются такие визуализации на основе *JavaScript*, в первую очередь, на основе фреймворка *D3.js*. Существует пакет для R `htmlwidgets`, который предоставляет интерфейс для работы с *JavaScript* визуализациями из R и вставлять их в *RMarkdown* или *Quarto* HTML-документы и веб-приложения *Shiny*. `htmlwidgets` — это пакет, в первую очередь, для разработчиков R пакетов, которые делают на его основе очень простые и удобные в использовании R пакеты для создания динамических визуализаций и прочих динамических элементов.

15.2 Динамические визуализации в `plotly`

Один из самых распространенных средств для динамических визуализаций — это пакет `plotly`.

```
install.packages("plotly")
```

```
library(plotly)
```

```
Attaching package: 'plotly'
```

```
The following object is masked from 'package:ggplot2':

    last_plot
```

```
The following object is masked from 'package:stats':

    filter
```

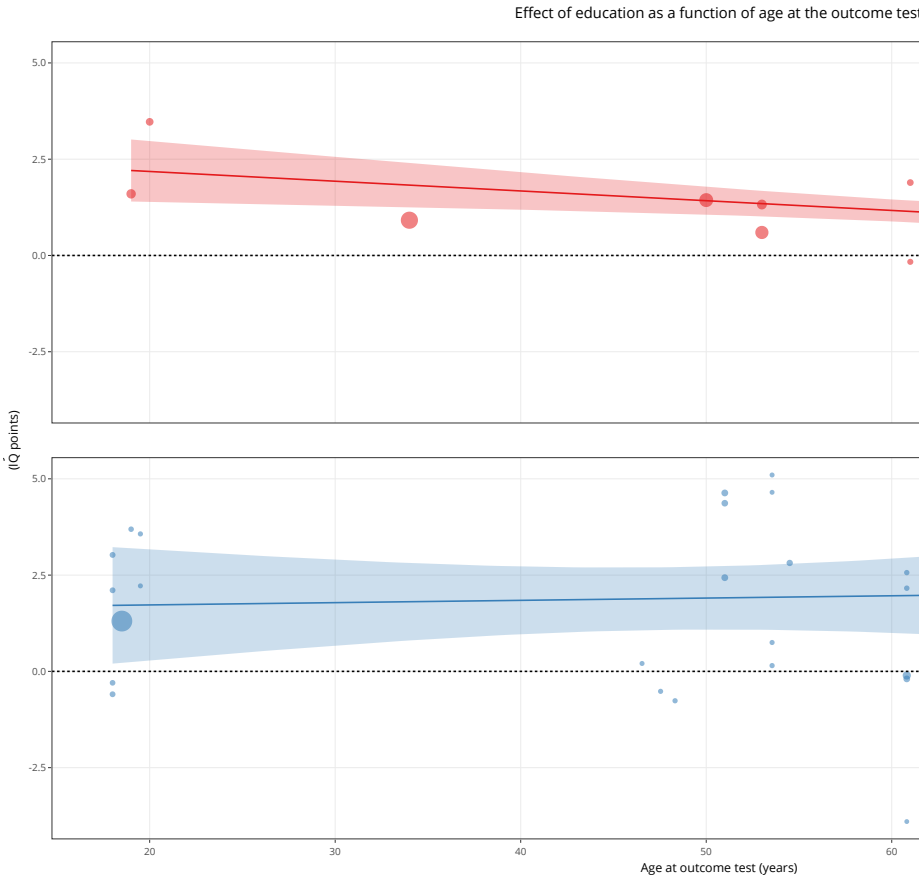
```
The following object is masked from 'package:graphics':

    layout
```

Есть два базовых способа использовать `plotly` в R. Первый — это просто оборачивать готовые графики `ggplot2` с помощью функции `ggplotly()`.

```
ggplotly(meta_2_gg)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



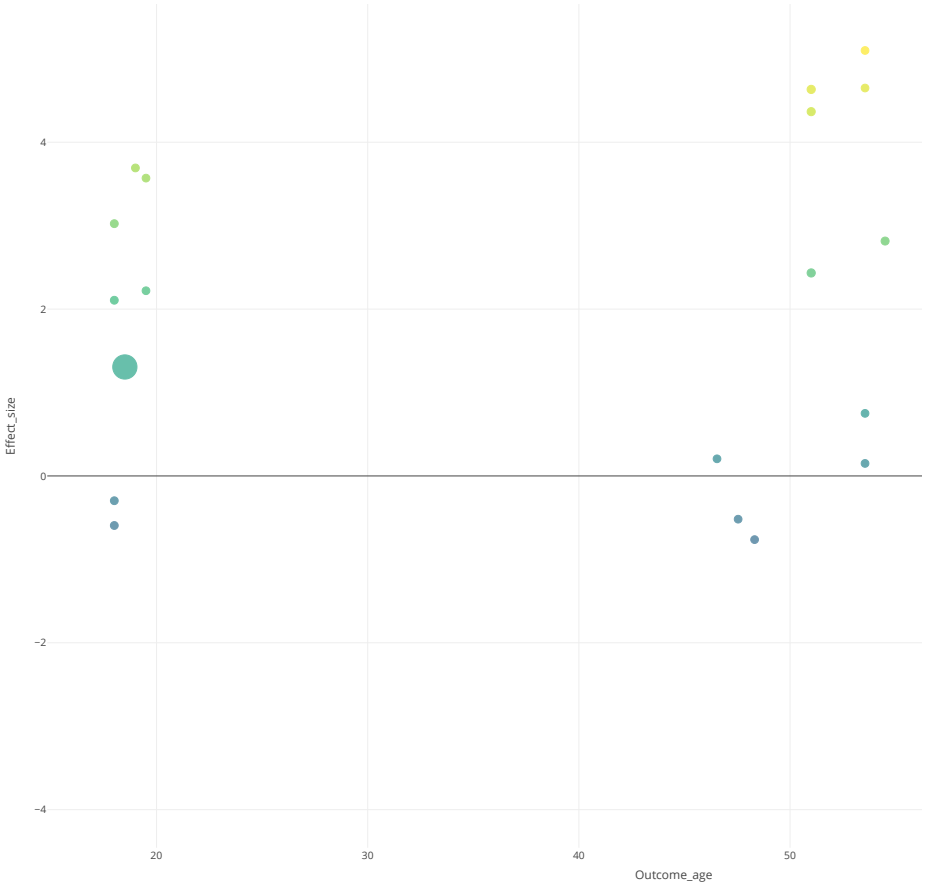
Не всегда это получается так, как хотелось бы, но простота этого способа подкупает: теперь наведение на курсора на точки открывает небольшое окошко с дополнительной информацией о точке (конечно, если вы читаете эту книгу в PDF или ePUB, то этого не увидите).

Другой способ создания графиков — создание вручную с помощью `plot_ly()`. Такой способ частично напоминает `ggplot2` использованием пайпов (обычных %>% или |>, а не +), задание эстетик здесь происходит с помощью ~.

```
plot_ly(poli,
        x = ~Outcome_age,
        y = ~Effect_size,
        size = ~1/(SE^2),
        color = ~Effect_size,
```

```
        sizes = c(40, 400),
        text = ~paste("N: ", n, '<br>Country:', Country)) %>%
  add_markers()
```

```
Warning: `line.width` does not currently support multiple values.
```



15.3 Другие пакеты для динамической визуализации

Кроме `plotly` есть и множество других HTML-виджетов для динамической визуализации. Я рекомендую посмотреть их самостоятельно на http://gallery.htmlwidgets.org/

Выделю некоторые из них:

- `echarts4r` — один из основных конкурентов для `plotly`. Симпатичный, работает довольно плавно, синтаксис тоже пытается вписаться в логику tidyverse.
- `leaflet` — основной (но не единственный!) пакет для работы с картами. Leaflet — это очень популярная библиотека JavaScript, используемая во многих веб-приложениях, а пакет `leaflet` - это довольно понятный интерфейс к ней с широкими возможностями.
- `networkD3` — пакет для интерактивной визуализации сетей. Подходит для небольших сетей.

Глава 16

R Markdown и Quarto

16.1 Что такое R Markdown

После подсчета описательных статистик, создания графиков и, в особенности, интерактивных визуализаций, возникает вопрос о том, как представить полученные результаты.

R Markdown представляет такую возможность. С помощью R Markdown в документе можно совмещать код, результаты его исполнения и написанный текст. Кроме того, можно вставлять картинки, ссылки, видео и многое другое. В чем-то R Markdown напоминает Jupyter Notebook знакомый всем питоноводом, но это сходство, скорее, функциональное (и то, и то позволяет превращать сухой текст скрипта в красивый документ), их устройство значительно различается.

R Markdown представляет собой текстовый документ специального формата .Rmd, который можно скомпилировать в самые различные документы:

- Документы в форматах Word, ODT, RTF, PDF (с использованием LaTeX), HTML, в том числе:
  - Онлайн-книги (bookdown)
  - Научные статьи (papaja)
- Презентации в виде HTML (ioslides, Slidy, revealjs, rmdshower, Beamer)
- Веб-сайты (blogdown)
- Дашборды (flexdashboard)

Формат вывода легко настроить и поменять по ходу работы, что позволяет гибко изменять формат документа на выходе.

16.2 Начало работы в R Markdown

Для работы с R Markdown у RStudio есть специальные инструменты, которые позволяют не только удобно писать и компилировать R Markdown документы, но и превращают R Markdown в удобную среду для работы с R вместо обычных R-скриптов.

Чтобы начать работать с R Markdown, нужно создать новый .Rmd файл с помощью `File - New File - R Markdown...` Перед вами появится меню выбора формата R Markdown документа, названия и имени автора.

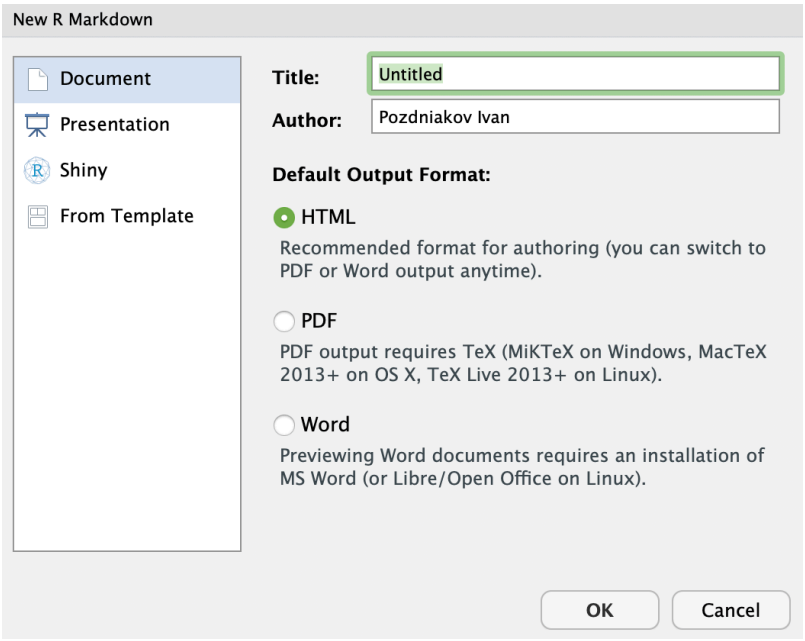


Рисунок 16.1: Меню выбора формата R Markdown документа

Выбирайте что угодно, все это можно потом изменить вручную.

Если пакет `rmarkdown` у вас еще не установлен, то он будет автоматически установлен. Кроме того, если вы выбрали в качестве формата PDF (презентацию или документ), то вам понадобится еще установить LaTeX на компьютер. Это тоже можно сделать с помощью специального пакета:

```
install.packages('tinytex')
tinytex::install_tinytex()  # install TinyTeX
```

.Rmd файл, который вы создадите таким образом, будет создан из шаблона, который демонстрирует основной функционал R Markdown. В отличие от работы с R скриптом, перед вами будет немного другой набор кнопок. Самая важная из новых кнопок — это кнопка `Knit` (с клубком и спицей рядом), нажав на которую, начнется "вязание" (knitting) финального документа, то есть его компиляция. Если компиляция завершится успешно, то перед вами появится скомпилированный документ в том формате, который вы выбрали.

16.3 Структура R Markdown документа

R Markdown документ состоит из трех базовых элементов:

- YAML-шапки [1]
- Текста с использованием разметки Markdown
- **Чанков (chunks)** с кодом

Разберем их по порядку.

1) **YAML-шапка** находится в самом верху документа и отделена тремя дефисами (---) сверху и снизу. В нем содержится, во-первых, мета-информация о документе, которая будет отображена на титульном листе/слайде, во-вторых, информация о формате документа, который будет "связан". Пример YAML-шапки:

```
---
title: "                    "
author: "         "
date: "15 11 2020"
output: html_document
---
```

[1] YAML расшифровывается как *"YAML Ain't Markup Language"*, ранее — как *Yet Another Markup Language*. Связано это с тем, что сначала этот язык позиционировался как язык разметки (например, HTML), затем — как язык сериализации, т.е. хранения данных (как JSON).

2) **Текст с использование синтаксиса Markdown** идет сразу после YAML-шапки и составляет основную часть .Rmd документа. Markdown (не путать с R Markdown!) — это популярный и очень удобный язык разметки. Markdown используется повсюду: в ReadMe страницах на GitHub, как способ ведения записей во многих программах для заметок и даже в Telegram! Например, вот так можно задавать полужирный шрифт и курсив:

```
**      **,              *    .*
```

В результате мы получим следующую строчку:

Вот так мы делаем **полужирный**, а вот так мы делаем *курсив.*

Далее мы разберем подробнее синтаксис Markdown.

3) **Чанки с кодом** содержат в себе код на языке R или другом языке программирования, которые будут исполнены, а результат которых будет отображен прямо под чанком с кодом. Чанк с кодом отделяется “‘ с обоих сторон и содержит `{r}`. Это означает, что внутри находится код на R, который должен быть выполнен:

```
```{r}
2+2
```
```

В итоговом документе чанк будет выглядеть так:

```
2+2
```

```
[1] 4
```

16.4 Настройки чанка

У чанка с кодом есть набор настроек. Самый важные из них такие:

- *echo*: будет ли показан сам код
- *message* и *warning*: будут ли показаны сообщения и предупреждения, всплывающие во время исполнения кода
- *eval*: будет ли испольняться код внутри чанка

16.4.1 Настройка нескольких чанков

Все эти настройки можно настроить как для отдельных чанков, так и для все чанков сразу:

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE)
```

Этот чанк нужно вставлять в начале .Rmd документа, тогда выбранные настройки повлияют на все последующие чанки.

16.4.2 Чанки с Python и другими языками программирования

Можно вставлять чанки с кодом на других языках программирования! Для этого вместо {r} нужно написать {python}.

```
x = 'hello, python !'
print (x.split(" "))
```

Вот полный список поддерживаемых языков:

```
names(knitr::knit_engines$get())
```

```
 [1] "awk"        "bash"       "coffee"     "gawk"       "groovy"     "haskell"
 [7] "lein"       "mysql"      "node"       "octave"     "perl"       "php"
[13] "psql"       "Rscript"    "ruby"       "sas"        "scala"      "sed"
[19] "sh"         "stata"      "zsh"        "asis"       "asy"        "block"
[25] "block2"     "bslib"      "c"          "cat"        "cc"         "comment"
[31] "css"        "ditaa"      "dot"        "embed"      "eviews"     "exec"
[37] "fortran"    "fortran95"  "go"         "highlight"  "js"         "julia"
[43] "python"     "R"          "Rcpp"       "sass"       "scss"       "sql"
[49] "stan"       "targets"    "tikz"       "verbatim"   "ojs"        "mermaid"
[55] "include"
```

16.4.3 Код вне чанков (inline code)

Иногда хочется вставить результат расчетов прямо в текст. Для этого нужно поставить символ ‘ с обоих краев команды и написать r перед самой командой. В этом случае результат выполнения этой команды будет в тексте вместо этой конструкции.

Число пи равно ‘ r pi ‘:

```
3.1415927
```

16.5 Синтаксис Markdown (без R)

В RStudio есть подсказка по синтаксису Markdown, для ее вызова нужно нажать `Help - Markdown Quick Reference`

16.5.1 Выделение текста

Выделение текста происходит с помощью обособления текста специальными символами:

```
*   *

_       _
**      **

--        --
~~       ~~
   ^      ^
   ~      ~
```

Курсив Тоже курсив **Полужирный Тоже полужирный** ~~перечеркнутый~~ индекс$^{\text{надстрочный}}$ индекс$_{\text{подстрочный}}$

16.5.2 Заголовки разных уровней

С помощью решенточек (#) выделяются заголовки разных уровней.

```
#

##

###

####

#####     !

######   ,
```

16.5.3 Списки

Списки можно создавать по-разному, в зависимости от того, является ли список пронумерованным:

```
*                       :
    +
    +           ?

1.
2.
```

- Первый вариант списка выглядит так:
  - Можно и с подсписком
  - Почему бы и нет?

1. Кому нужен порядок
2. Тот списки номерует

16.5.4 Цитаты

Цитаты выделяются с помощью знака > в начале строки.

```
>
>
```

> Я устал
> Который год во мне живет нарвал

16.5.5 Таблицы

Табличные данные имеют особое значение в R, в R Markdown им тоже уделяется особое внимание.

Для начала подгрузим данные о супергероях:

```
library("tidyverse")
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
                   na = c("-", "-99"))
```

Функция `knitr::kable()` превращает табличные данные (матрицы, датафреймы) в текст, отформатированный как Markdown-таблицы. Таким образом, вот такая таблица:

```
knitr::kable(heroes[1:3, 1:4])
```

Превращается вот в такую, отформатированную с помощью символов -, | и т.п.:

```
| X1|name       |Gender |Eye color |
|--:|:----------|:------|:---------|
|  0|A-Bomb     |Male   |yellow    |
|  1|Abe Sapien |Male   |blue      |
|  2|Abin Sur   |Male   |blue      |
```

А эта таблица, в свою очередь, превращается в такую в финальном документе:

X1	name	Gender	Eye color
0	A-Bomb	Male	yellow
1	Abe Sapien	Male	blue
2	Abin Sur	Male	blue

Если вам нужно самостоятельно отформатировать таблицу в Markdown, то для этого есть специальный ресурс.

Пакет `knitr` является ключевым для R Markdown, поэтому он устанавливается вместе с `rmarkdown`. А вот для дополнительной настройки вывода таблиц рекомендуется пакет `kableExtra`.

16.6 Дополнительные возможности R Markdown

16.6.1 Динамические таблицы

Один из самых интересных HTML-виджетов (@ref(htmlwidgets)) — пакет `DT` для создания интерактивных таблиц прямо внутри HTML-документа.

```
library(tidyverse)
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
                   na = c("-", "-99"))
DT::datatable(heroes)
```

Show 10 entries Search:

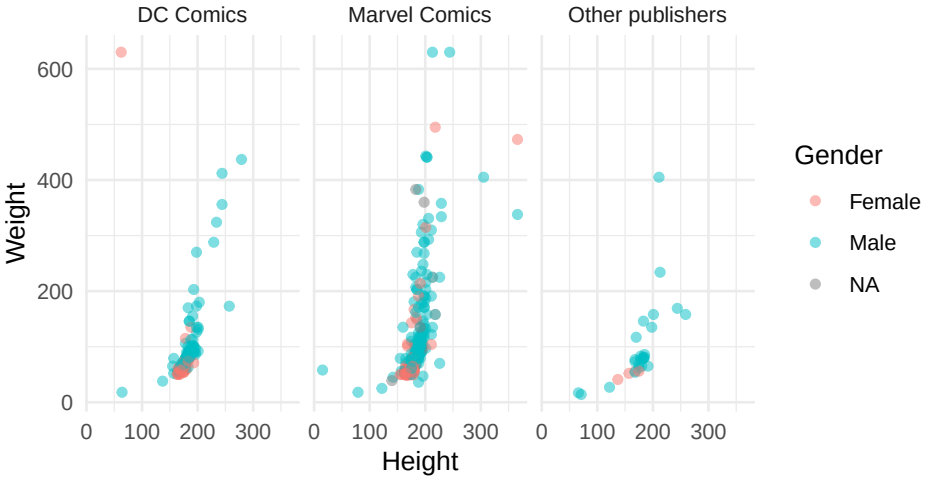
	...1	name	Gender	Eye color	Race	Hair color	Height	Publisher	Skin color	Alignment	Weight
1	0	A-Bomb	Male	yellow	Human	No Hair	203	Marvel Comics		good	441
2	1	Abe Sapien	Male	blue	Icthyo Sapien	No Hair	191	Dark Horse Comics	blue	good	65
3	2	Abin Sur	Male	blue	Ungaran	No Hair	185	DC Comics	red	good	90
4	3	Abomination	Male	green	Human / Radiation	No Hair	203	Marvel Comics		bad	441
5	4	Abraxas	Male	blue	Cosmic Entity	Black		Marvel Comics		bad	
6	5	Absorbing Man	Male	blue	Human	No Hair	193	Marvel Comics		bad	122
7	6	Adam Monroe	Male	blue		Blond		NBC - Heroes		good	
8	7	Adam Strange	Male	blue	Human	Blond	185	DC Comics		good	88
9	8	Agent 13	Female	blue		Blond	173	Marvel Comics		good	61
10	9	Agent Bob	Male	brown	Human	Brown	178	Marvel Comics		good	81

Showing 1 to 10 of 734 entries Previous 1 2 3 4 5 … 74 Next

16.6.2 Графики в R Markdown

Все создаваемые графики будут появляться под чанком с кодом.

```
height_weight_gg <- heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  filter(Weight < 700 & Height < 400) %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point(aes(colour = Gender), alpha = 0.5) +
  coord_fixed() +
  facet_wrap(~Publisher)+
  theme_minimal()
height_weight_gg
```



Это так же относится и к динамическим визуализациям с помощью HTML-виджетов (@ref(htmlwidgets)), например, `plotly`.

```
library(plotly)
ggplotly(height_weight_gg)
```

Конечно, чтобы эта интерактивность сохранилась, используемый формат итогового документа должен ее поддерживать. Word-документы, так же как и PDF-документы, — статичны, поэтому единственный вариант сохранить интерактивные элементы — это использование HTML-документов или HTML-презентаций.

16.6.3 HTML-код

Если вы выбрали HTML форматом итогового документа, то можете использовать все его фишки, включая форматирование с помощью HTML-тегов (в дополнение к обычному Markdown). Еще вы можете вставлять куски HTML-кода, например, вставить видео с YouTube или отдельный пост из Twitter.

К сожалению, в PDF нельзя вставить никакой интерактивности, смотрите онлайн-версию книги)

16.7 Quarto

Quarto – это следующая итерация развития *R Markdown*, которая пытается выйти за пределы его ограничений и стать более универсальным инструментом, не привязанным к R. *Quarto* позволяет использовать как `{knitr}`, который лежит в основе *R Markdown*, так и Jupyter Notebooks, Julia и Observable JS. Quarto имеет свой дополнительный синтаксис, но так же будет работать со всеми фишками R Markdown, которые были описаны выше. В частности, данная книга была изначально написана с помощью R Markdown и была переведена в Quarto почти без каких-либо дополнительных усилий кроме переименования файлов.

Часть IV

Тестирование статистических гипотез

Этот раздел будет посвящен статистическим модели и тестам. Но чтобы разобраться с ними, нам для начала придется понять общую логику, которая стоит за ними, которую мы будем разбирать в главе Глава 17.

Все последующие главы будут посвящены отдельным группам статистических моделей и тестов, от самых простых до самых сложных.

Глава 17

Статистика вывода

17.1 Введение в статистику вывода

Статистика вывода *(inferential statistics)* - это основной раздел статистики, связанный уже не с описанием и суммированием имеющихся данных (описательная статистика), а с попытками сделать вывод о **генеральной совокупности *(population)*** на основе имеющихся данных по **выборке *(sample)***. Короче говоря, статистика вывода - это о том, как выйти за пределы наших данных. Это именно то, зачем мы проводим исследования - мы не можем собрать информацию обо всей генеральной совокупности, но по тому, что имеем (т.е. по нашей выборке), можем попытаться как-то оценить **параметры** распределения в генеральной совокупности.

Итак, еще раз: генеральной совокупности - параметры **(Population - Parameters),** а у выборки - статистики **(Sample - Statistics).** Параметры обычно обозначаются греческими буквами: μ, σ (или большими латинскими: M, S), а статистики - соответствующими латинскими: m, s.

17.2 Случайные величины и их распределения

Случайная величина (*random variable*) – это переменная значения которой в зависимости от случая принимают различные значения. Представьте себе машину, которая по требованию выдает какое-то случайное число (или даже несколько). Это и будет **случайная величина.** Какие это могут быть значения,

какие-то конкретные или любые в каком-то диапазоне? Могут ли какие-то значения выпадать чаще других? Очевидно, случайные переменные могут быть разными в зависимости от закона, который стоит за тем, какие значения она может принимать и с какой частотой. Этот закон называется распределением вероятности.

Если случайная переменная может принимать только какие-то конкретные значения (например, 3, 10, 0.25 или число пи), то случайная переменная имеет **дискретное распределение**. Если же случайная переменная может принимать любые значения (в каком-то диапазоне или же вообще любые), то такая случайная переменная будет иметь **непрерывное распределение**. По большей части нас будут интересовать именно непрерывные. Хотя для полного их понимания нужен матан (да, именно тот, который *calculus*), непрерывные распределения довольно интуитивно понятны.

Может ли случайная переменная принимать значения с разной частотой? Если нет, то речь идет о **равномерном распределении (*uniform distribution*).** Пример случайной величины с дискретным равномерным распределением – игральный кубик, с вероятностью $\frac{1}{6}$ возвращает одно из значений: 1, 2, 3, 4, 5 или 6. Однако равномерное распределение может быть как дискретным, так и непрерывным.

Все остальные распределения (кроме **равномерного распределения)** обозначают, что какие-то значения имеют больший шанс выпадения, чем другие. Разные распределения могут быть описаны с помощью **функций распределения**. В R есть свой набор из трех основных функций для каждого распределения и еще одна функция: генератор случайных чисел из выбранного распределения.

Вот эти функции:

- `d*()` – **функция вероятности (*probability mass function*)** для дискретных распределений и **функция плотности вероятности** для непрерывных распределений;
- `p*()` – **функция накопленной плотности распределения (*cumulative distribution function; cdf*);**
- `q*()` – **квантильная функция (*quantile function*),** или **обратная функция накопленной плотности распределения (*inverse cumulative distribution function*).**

Ну и функция `r*()` для создания случайных выборок из выбранного распределения.

Во всех четырех случаях, вместо звездочки нужно поставить название соответствующего распределения в R. Таким образом, для каждого распределения в R есть четыре функции (с похожими названиями, различающимися первой буквой), и таких семейств функций в базовом R (точнее, во встроенном пакете `{stats}`) очень много (посмотреть можно с помощью `?Distributions`). Вот лишь некоторые из них:

Distributions

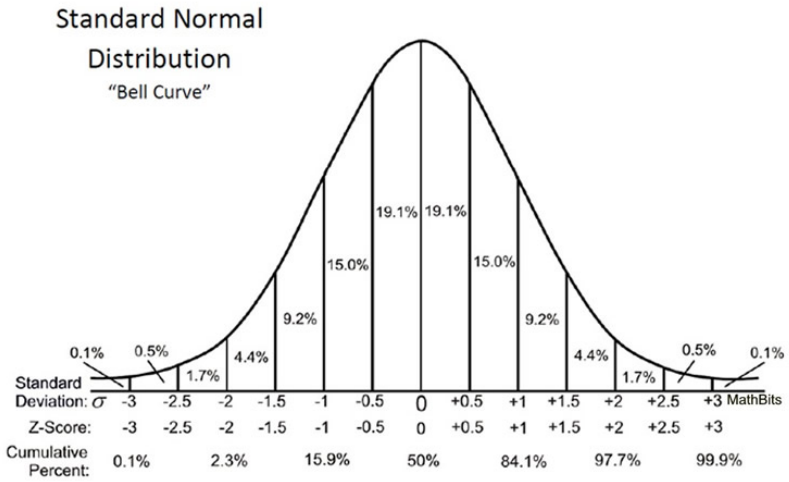
	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Возьмем, например, нормальное распределение, и разберем на его примере все эти функции: `dnorm()`, `pnorm()`, `qnorm()` и `rnorm()`.

Для того, чтобы описать нормальное распределение нам нужно всего два параметра - его среднее μ и стандартное отклонение σ. Если же $\mu = 0$, а $\sigma = 1$, то такое **нормальное распределение** называется **стандартным *(standard normal distribution).*** Заметьте, здесь мы говорим о *параметрах* распределения в генеральной совокупности, а не о *статистиках* в конкретной выборке, хотя называются они одинаково.

Я думаю, все видели, как выглядит нормальное распределение. Это та самая “колоколообразная кривая”[1].

[1]Если распределение имеет колоколообразную форму, то это еще не значит, что оно нормальное. Есть очень много похожих по форме распределений, которые, тем не менее, значительно отличаются от нормального, в том числе своими свойствами. Например, распределение Коши тоже внешне похоже на колокол, но у него нет среднего и стандартного отклонения!



То, что Вы видите на картинке - это так называемая **функция плотности вероятности *(probability density function).*** Вот ее формула для **нормального распределения *(normal distribution):***

$$P(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

Это аналог ***probability mass function*** для дискретных величин - распределение вероятности того, что случайная величина имеет данное значение. Довольно очевидно, да? Тогда почему аналогичная функция для непрерывных распределений называется по-другому?

Дело в том, что вероятность получения конкретного значения для непрерывных величин равна нулю. Это немного необычный момент, но тем не менее: если распределение непрерывно, то получить точно 0,000000000000... с бесконечным количеством нулей после запятой невозможно, даже если среднее распределения равно нулю. Тем не менее, мы можем посчитать вероятность того, что случайное значение окажется в определенном промежутке. **Функция плотности вероятности** как раз и позволяет это сделать. Чтобы посчитать эту вероятность, надо посчитать площадь соответствующего участка под функцией. Соответственно, площадь под всей функцией плотности вероятности равна 1.

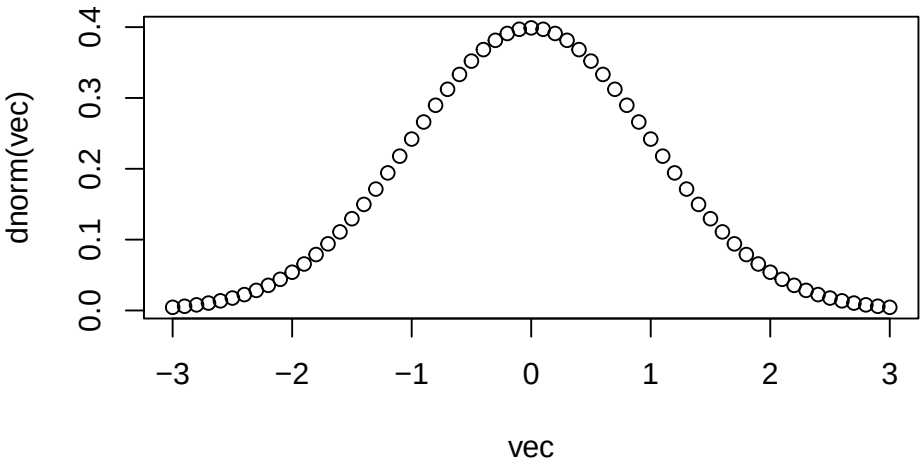
Конечно, функцию плотности можно получить в R - это функции `d*()`, где `*` означает соответствующее распределение. Для нормального распределения это будет функция `dnorm()`.

Функция `dnorm()` имеет следующие основные параметры: `x =` - вектор принимаемых значений, среднее и стандартное

отклонение распределения (`mean =` и `sd =` соответственно). В качестве параметров по умолчанию используются 0 для среднего и 1 для стандартного отклонения, то есть **стандартное нормальное распределение.** Для других распределений параметры будут отличаться (например, вместо `mean =` и `sd =` в t-распределении будет параметр `df =` – "степени свободы").

Давайте посмотрим, как работает эта функция, визуализировав результат ее выполнения на векторе от -3 до 3 с небольшим шагом (0.1). С помощью базовой функции `plot()` мы построим диаграмму рассеяния, где по оси *x* используем исходный вектор с последовательностью от -3 до 3 с шагом 0.1, а по оси *y* – результат применения функции `dnorm()` на исходном векторе. Каждая отдельная точка – это значение **функции плотности вероятности** (координата *y)* для конкретного *x*. Поскольку таких точек много, и они находятся близко друг к другу, то фактически получается кривая линия.

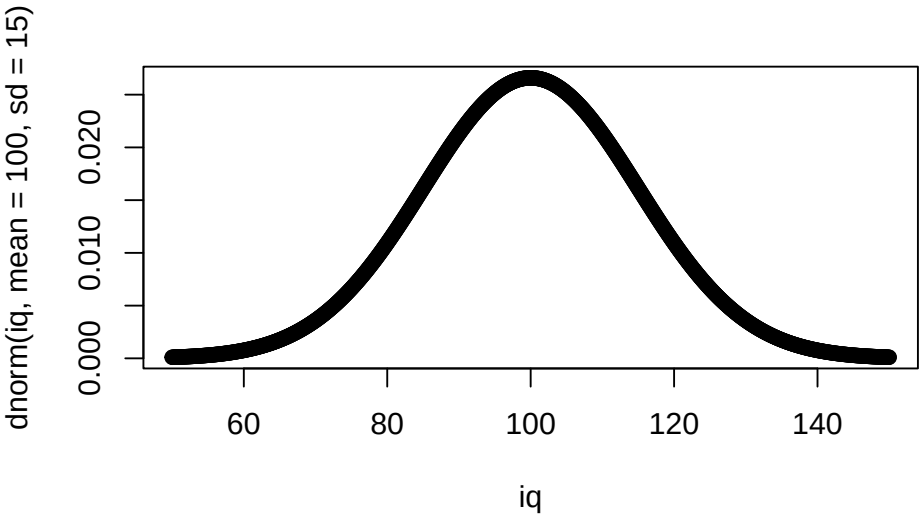
```
vec <- seq(-3,3, 0.1)
plot(vec,dnorm(vec))
```



Вот мы и получили то, что называем **стандартным нормальным распределением,** точнее, **функцию плотности вероятности** стандартного нормального распределения.

Давайте теперь возьмем другое нормальное распределение с другими параметрами. Например, среднее будет равно 100, а стандартное отклонение – 15. Это нормы для шкалы IQ:

```
iq <- seq(50,150, 0.1)
plot(iq, dnorm(iq, mean = 100, sd = 15))
```



> ***Полезное:* шкала IQ**
>
> Шкала IQ — это вообще очень удобная шкала. Поскольку в измерениях интеллекта нет никаких объективных метрик (сантиметров, градусов, килограммов), с которыми его можно было бы сравнить, то единственная возможность дать какую-то оценку величины интеллекта у человека — сравнить его с интеллектом другого человека, например, по количеству решенных заданий за определенное время. Поэтому шкала IQ так сконструирована, чтобы среднее значение количество решенных заданий обозначалось за 100, а стандартное отклонение за 15. Поэтому по баллу IQ (если используется хороший тест, разумеется) можно понять процент людей, которых респондент опережает по интеллекту, по крайней мере, сравнивая с выборкой, на которой был стандартизирован тест. Примерно так же устроены и другие психологические шкалы.

Следующая функция это **функция накопленной плотности распределения *(cumulative distribution function; cdf)***[2]. Это

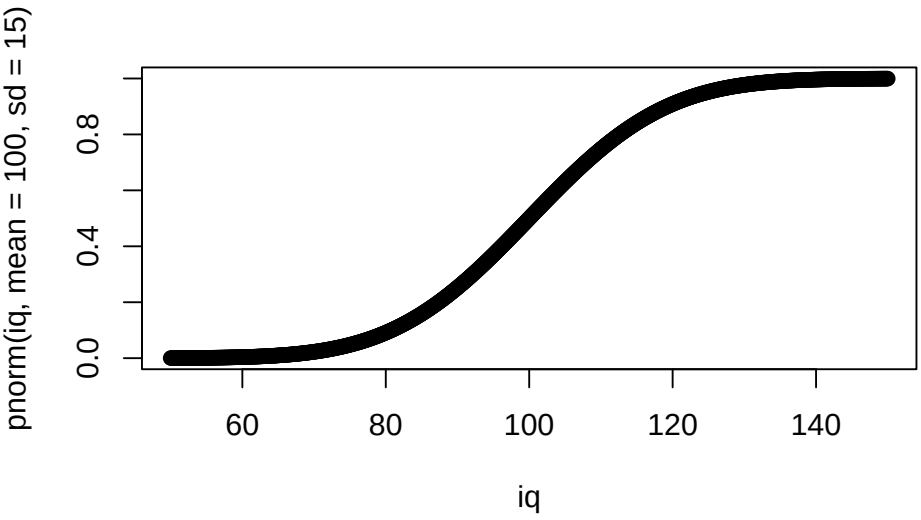
[2]Иногда эта функция называется *интегральной функцией вероятности*, а иногда просто *функцией вероятности.* Последний вариант может запутать, потому что часто под функцией вероятности подразумевают функцию плотности вероятности.

функция очень важная, потому что именно на ней основано **тестирование уровня значимости нулевой гипотезы,** которым мы будем заниматься в дальнейшем.

> Функция накопленной плотности распределения показывает вероятность того, что полученное случайное значение из распределения будет меньше искомого или равно ему.

Для этой функции распределения используются функции вида `p*()`, в частности, функция `pnorm()` для нормального распределения:

```
plot(iq, pnorm(iq, mean = 100, sd = 15))
```



Какова вероятность того, что полученное случайное значение IQ будет меньше или равно 100?

```
pnorm(100, mean = 100, sd = 15)
```

```
[1] 0.5
```

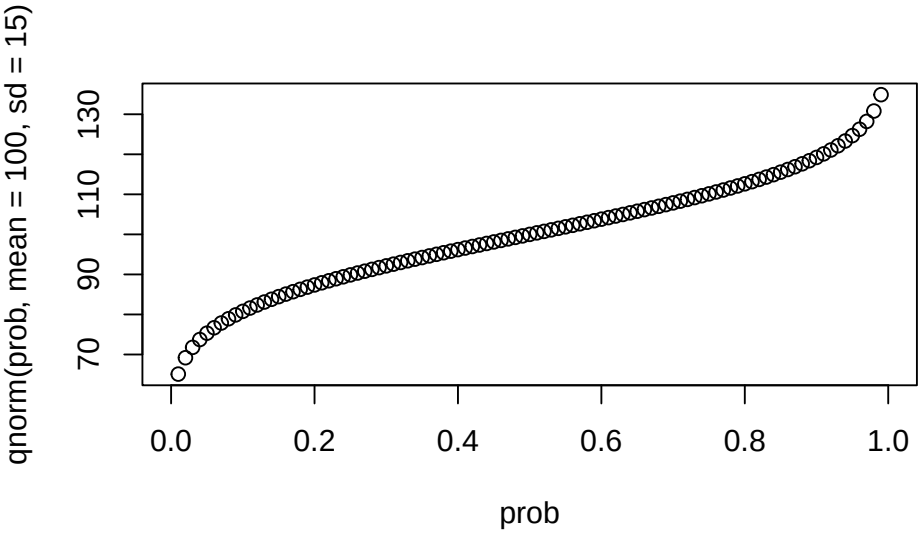
А меньше или равно 130?

```
pnorm(130, mean = 100, sd = 15)
```

```
[1] 0.9772499
```

Следующая функция — это **квантильная функция *(quantile function)***, или **обратная функция накопленной плотности распределения *(inverse cumulative distribution function)*:**

```
prob <- seq(0,1, 0.01)
plot(prob, qnorm(prob, mean = 100, sd = 15))
```



Обратная функция означает, что если мы применим сначала одну, а потом другую, то (если не берем особо крайних значений) на выходе получим исходные числа [3]. **Квантильная функция** возвращает значение, которое по заданной вероятности случайная переменная не будет превышать. Поскольку квантильная функция – это функция от вероятности, квантильная функция определена на отрезке от 0 до 1.

```
qnorm(pnorm(-4:4))
```

```
[1] -4 -3 -2 -1  0  1  2  3  4
```

Ну и последняя важная функция - это `rnorm()` - просто генерирует выборку значений из данного распределения заданной длины `n` =:

[3] Если мы возьмем достаточно большие значения, то этот трюк не сработает, что свзязано с тем, что числа дробные числа в компьютерах хранятся с ограниченной точностью.

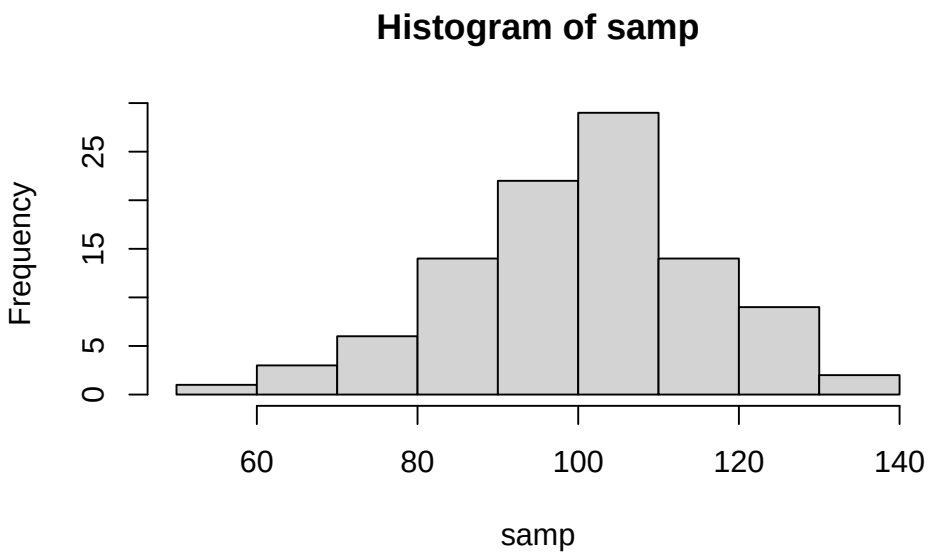
```
set.seed(42)
samp <- rnorm(100, mean = 100, sd = 15)
samp
```

```
  [1] 120.56438  91.52953 105.44693 109.49294 106.06402  98.40813 122.67283
  [8]  98.58011 130.27636  99.05929 119.57304 134.29968  79.16709  95.81817
 [15]  98.00018 109.53926  95.73621  60.15317  63.39300 119.80170  95.40042
 [22]  73.28037  97.42124 118.22012 128.42790  93.54296  96.14096  73.55255
 [29] 106.90146  90.40008 106.83175 110.57256 115.52655  90.86610 107.57433
 [36]  74.24487  88.23311  87.23639  63.78689 100.54184 103.08998  94.58414
 [43] 111.37245  89.09943  79.47578 106.49227  87.82910 121.66152  93.52831
 [50] 109.83472 104.82888  88.24242 123.63591 109.64349 101.34641 104.14826
 [57] 110.18933 101.34749  55.10365 104.27324  94.49148 102.77846 108.72736
 [64] 120.99605  89.09062 119.53814 105.03772 115.57759 113.81093 110.81317
 [71]  84.35322  98.64720 109.35277  85.69715  91.85757 108.71495 111.52268
 [78] 106.95651  86.71336  83.50329 122.69061 103.86882 101.32660  98.18655
 [85]  82.08507 109.17995  96.74290  97.25865 114.00019 112.32660 120.88175
 [92]  92.85739 109.75523 120.86666  83.33817  87.08811  83.02392  78.11179
 [99] 101.19974 109.79807
```

`set.seed()` - это функция, которая позволяет получить нам воспроизводимые результаты при использовании генератора случайных чисел. Короче говоря, если все мы поставим `set.seed(42)` [4], то одна и та же строчка выдаст нам один и тот же результат на разных компьютерах. Как это вообще возможно, это же случайные числа? Дело в том, что… нет. Они “псевдо-случайные”. На самом деле, используются определенные алгоритмы, чтобы генерировать числа, которые выглядят как случайные. Например, можно брать цифры после запятой в числе пи после, например, 42го знака. Реальные алгоритмы создания псевдо-случайных чисел, конечно, гораздо сложнее, но суть примерно такая. Насколько подобные числа действительно получаются случайными - отдельный сложный математический вопрос. Но для нас этого вполне достаточно.

```
hist(samp, breaks = 10)
```

[4] А почему именно 42? Ну, можно брать любое число, которое Вам нравится. А 42 - это ответ на главный вопрос жизни, вселенной и всего такого



17.3 Оценки

Самая основа статистики вывода - это **оценки *(estimates)***. Как мы уже знаем, у распределений есть определенные параметры, которые описывают данное распределение.

Например, для того, чтобы описать нормальное распределение нам нужно всего два параметра - его среднее μ и стандартное отклонение σ.

Наша цель - как-нибудь оценить эти параметры, потому что обычно мы их не знаем. Допустим, человеческий рост распределен нормально [^isnotnormal]. Какой средний рост в популяции? Какое у него стандартное отклонение? Для этого мы используем разного рода оценки: точечные и интервальные. В качестве оценок параметров часто используются статистики по выборке. Вот здесь легко запутаться, поэтому покажу картинку:
[^isnotnormal]: На самом деле, “чистое” нормальное распределение, так же как и любое другое “чистое” распределение, в природе встречается достаточно редко. Но довольно многие процессы могут апроксимироваться нормальным распределением благодаря центральной предельной теореме Глава 17.8.

17.4 Точечные оценки

Представим, что только что сгенерированные данные с помощью `rnorm()` - это и есть наша выборка. Перед нами стоит задача оценить IQ популяции по этой выборке. Попробуем **оценить *(estimate)*** среднее в генеральной совокупности.

Здесь все просто и очень очевидно - самой лучшей оценкой среднего в генеральной совокупности будет среднее по выборке:

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Оценки обозначаются значком домика: $\hat{}$. $\sum_{i=1}^{n} x_i$ означает сумму всех элементов x от 1 до n. Подобная запись с $\sum_{i=1}^{n}$ нам будет встречаться очень часто!

Посчитаем оценку среднего в генеральной совокупности с помощью уже знакомой нам функции для выборочного среднего:

```
mean(samp)
```

```
[1] 100.4877
```

Конечно, не совсем точно, но лучше оценки не придумаешь. Что есть, то есть. Чем больше выборка, тем в ближе оценка будет к популяционному среднему, т.е. тем выше **точность *(efficiency)*** оценки.

А что с оценкой стандартного отклонения? Давайте воспользуемся нашей предыдущей формулой.

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n}}$$

```
sqrt(sum((samp - mean(samp))^2)/length(samp))
```

```
[1] 15.54206
```

В данном случае мы несколько промахнулись вверх, но обычно оценка по этой формуле дает небольшое **смещение *(bias)*** в меньшую сторону. В отличие от выборочного среднего как оценки среднего в генеральной совокупности, использование выборочного стандартного отклонения по выборке приводит к смещенной оценке! Само по себе это кажется странным, и это нормально. Но этому есть всякие серьезные математические доказательства. Кроме того, есть множество всяких демонстраций, например, от Академии Хана.

> ***Для продвинутых:* откуда смещение оценки?**
>
> Одно из простых объяснений такое: поскольку мы оцениваем стандартное отклонение на основе оценки среднего, среднеквадратичные расстояния считаются не от реального среднего в генеральной совокупности, а от среднего по выборке, которое немного смещено в ту или иную сторону. Представьте, что так получилось, что в нашей выборке оно оказалось сильно смещено и выборочное среднее получилось около 90 (такое бывает). Это значит, что получилось много довольно низких значений около этого 90, а высчитываться среднеквадратичные разницы будут не от среднего 100 (которое мы не знаем), а от этого 90. Поэтому стандартное отклонение получится сильно меньше, чем должно было получиться.

Чтобы получить **несмещенную (unbiased) оценку** стандартного отклонения или дисперсии, то нам нужно делить не на n, а на $n-1$.

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}}$$

Заметьте, что именно так делает стандартная функция `sd()`:

```
sqrt(sum((samp - mean(samp))^2)/(length(samp) - 1))
```

```
[1] 15.62035
```

```
sd(samp)
```

```
[1] 15.62035
```

Это называется поправкой Бесселя, и она настолько распространена, что именно скорректированное стандартное отклонение обычно называют стандартным отклонением. Поэтому в дальнейшем будет использоваться именно эта формула или же просто функция `sd()`.

Таким образом, у оценки есть два критерия качества – ее **точность** и ее **несмещенность.**

17.5 Интервальные оценки

Другой подход к оцениванию параметра распределения в генеральной совокупности заключается в использовании интервалов. Вместо того, чтобы дать одну точечную оценку, которая будет заведомо неточна, можно попробовать оценить интервал, в котором находится истинное среднее генеральной совокупности.

Самая распространенная интервальная оценка называется ***доверительным интервалом (confidence interval)***. Цель доверительного интервала – "покрыть" параметр генеральной совокупности с определенной степенью уверенности.

Например, 95% доверительный интервал или просто $CI95\%$ означает, что примерно в 95% случаев подсчитанный на выборках интервал будет ловить значение параметра в популяции. Например, построив $CI95\%$ по нашей сгенерированной выборке, мы хотим чтобы примерно в 95% случаев выборок, построенных таким способом, этот интервал ловил истинное среднее (в данном случае — 100).

Для того, чтобы научиться строить такие интервалы, нам нужно разобраться с **выборочным распределение *(sampling distribution).***

17.6 Выборочное распределение

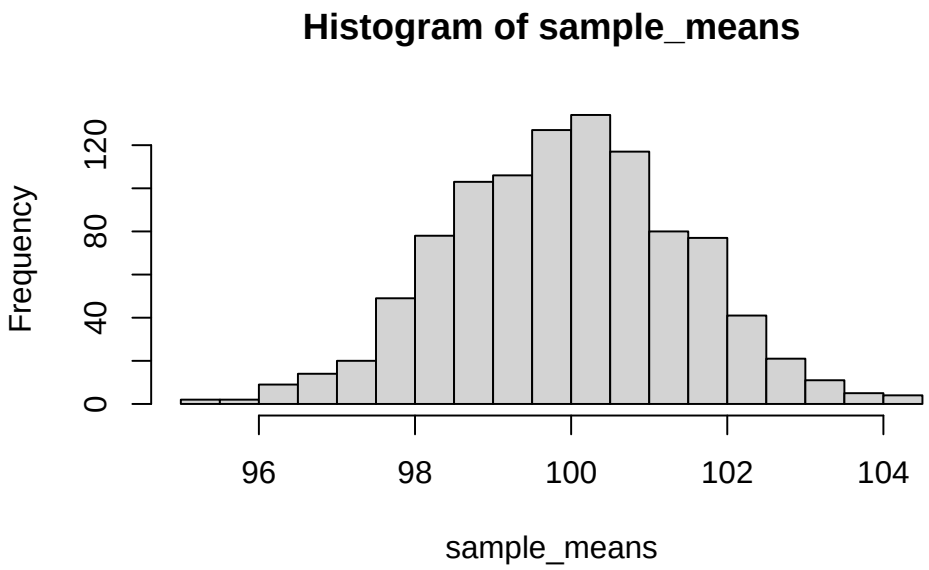
Чтобы разобраться с тем, что стоит за **доверительными интервалами** и **тестированием уровня значимости нулевой гипотезы** (короче говоря, с основными инструментами статистики вывода), нам надо разобраться с очень абстрактной концепцией – выборочными распределениями.

Представьте себе, что мы бы выбрали не одну, а сразу много выборок, а потом у каждой выборки посчитали бы среднее. Мы бы получили новый вектор данных - средние выборок из одного распределения. Давайте это сделаем[5]:

```
sample_means <- replicate(1000, mean(rnorm(100, mean = 100, sd = 15)))
```

Каждая выборка состоит из сотни “испытуемых”, всего таких выборок 1000. По каждой мы посчитали среднее. Как распределены эти средние?

```
hist(sample_means, breaks = 30)
```



Вот это распределение и есть **выборочное распределение средних *(sampling distribution of means).*** Точнее, было бы, если бы мы взяли не 1000 выборок, а бесконечное количество выборок. Так что у нас всего лишь аппроксимация выборочного распределения средних, к сожалению.

Мы могли взять другие статистики, не средние выборок, а, например, медианы или стандартные отклонения выборок. Тогда

[5]Конечно, на практике никто не будет собирать тысячу выборок. Но нам важно понимать, чтобы было, если бы мы повторяли один и тот же эксперимент в одинаковых условиях, даже если это и невозможно сделать. Это позволит нам понять, какие выводы мы можем сделать на основе тех данных, что мы имеем на самом деле.

это были бы выборочные распределения медиан и выборочные распределения стандартных отклонений соответственно. Но именно выборочное распределение средних обладает уникальными математическими свойствами, про которые мы скоро узнаем.

Например, среднее выборочного распределения средних будет равно популяционному среднему:

$$\mu_{\overline{x}} = \mu$$

То есть если бы у нас было много выборок, то среднее их средних все больше бы приближалось к популяционному среднему:

```
mean(sample_means)
```

```
[1] 99.93772
```

Жаль, что мы не можем позволить себе симулировать бесконечное количество выборок, ведь тогда их среднее было бы равно популяционному среднему!

Теперь перейдем к стандартному отклонению выборочного распределения средних. Звучит очень громоздко, не правда ли? Поэтому стандартное отклонение выборочного распределения обычно называют **стандартной ошибкой *(standard error),*** а стандартное отклонение выборочного распределения средних называют **стандартной ошибкой среднего *(standard error of the mean; s.e.m.).***

Чему будет равна **стандартная ошибка среднего?** Давайте посмотрим на стандартное отклонение в нашей симуляции выборочного распределения средних:

```
sd(sample_means)
```

```
[1] 1.501392
```

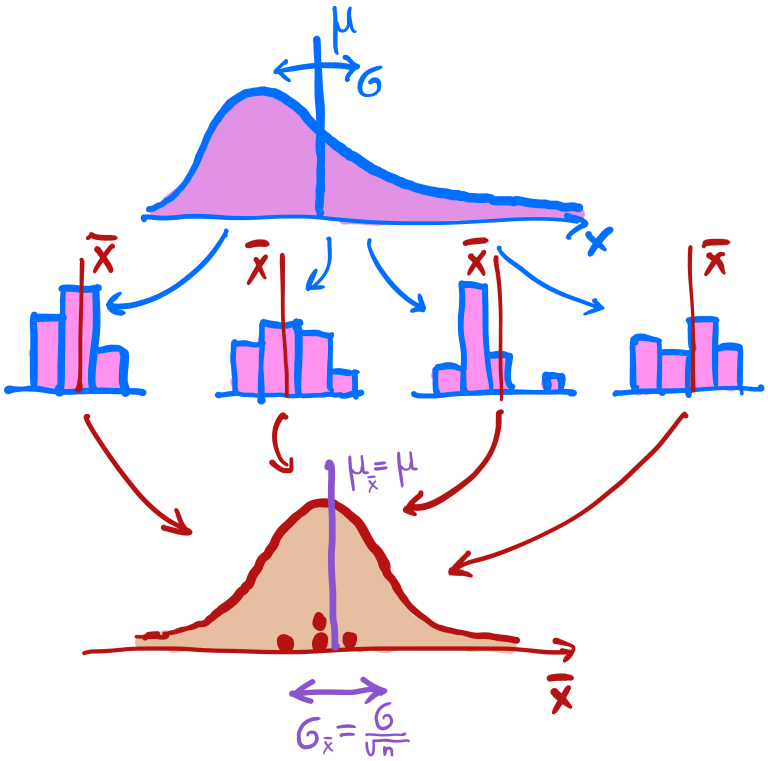
Очень похоже на стандартное отклонение в шкале IQ (15), но в 10 раз меньше. Это не случайно: **стандартная ошибка среднего** равна стандартному отклонению в генеральной совокупности, деленному на корень из размера выборки.

$$\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{n}}$$

Как раз это мы и получили: размер нашей выборки - 100, а корень из 100 равен 10:

```
15/sqrt(length(samp))
```

```
[1] 1.5
```



Из этой формулы следует, что стандартная ошибка тем меньше, чем больше выборка. И это довольно логично: чем больше у нас размер выборок, тем меньше будет разброс их средних. При этом уменьшение разброса не будет линейным: чтобы уменьшить его в 2 раза, нужно увеличить выборку в 4 раза, а чтобы уменьшить его в 10 раз, нужно увеличить выборку в 100 раз.

Ну а если мы не знаем стандартного отклонения в генеральной совокупности (что обычно и бывает в жизни), то можем оценить стандартную ошибку среднего с помощью оценки стандартного отклонения, посчитанного на выборке:

$$\hat{\sigma_{\overline{x}}} = \frac{\hat{\sigma}}{\sqrt{n}}$$

```
sd(samp)/sqrt(length(samp))
```

```
[1] 1.562035
```

Именно **оценку стандартной ошибки среднего** обычно используют на графиках в научных статьях в качестве ***усов (error bars)***[6].

17.7 Важность нормального распределения

В математике все числа равны, но некоторые все-таки заметно равнее других. Например, есть 0 и 1 - и очевидно, что это очень важные числа. Даже гугл выдает больше страниц по числам 0 и 1, чем по другим числам, например, 8 и 23218974. Не только целые числа могут быть важными. Вот, например, число пи и число Эйлера: они очень часто встречаются в самых разных, подчас неожиданных местах. Например, в тех же функциях распределений. На эти числа завязано очень интересные и важные свойства, поэтому такая зацикленность на них неудивительна. Например, число пи связывает радиус круг с длиной его окружности и площадью, поэтому когда перед нами что-то круглое, то и пи появится с большой вероятностью.

В статистике тоже есть распределения, которые “главнее” других распределений. Есть, например, равномерное распределение. Например, равномерно распределены исходы одного броска кубика.

Но есть и распределение, которое смело можно называть королем всех распределений - это уже знакомое нам нормальное распределение.

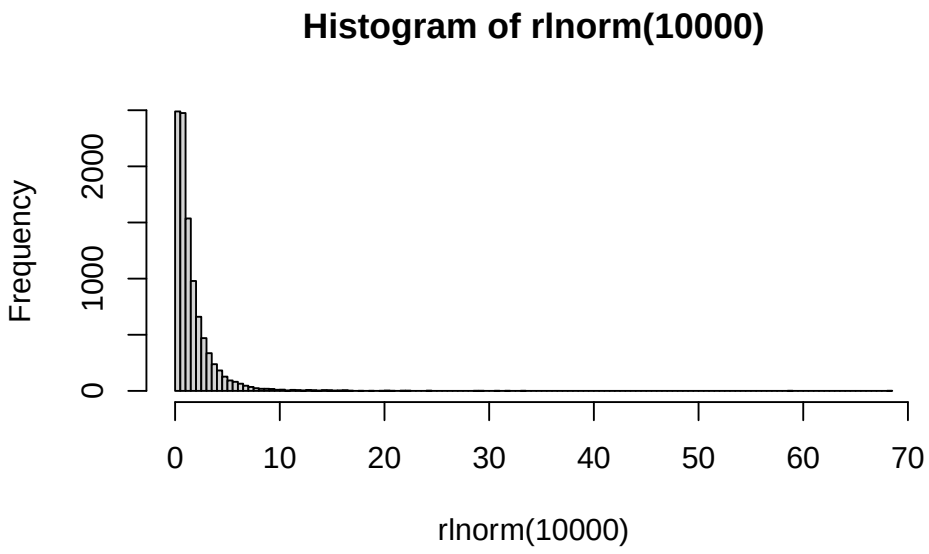
Нормальное распределение - это что-то в духе единицы из мира распределений. Это не просто одно из распределений, а это фундаментально важная штуковина, которая обладает почти магической силой. И сила эта зовется “Центральная Пределельная Теорема”.

[6]Для барплотов или точечных диаграмм. Для ящиков с усами используются другие правила, см. Глава 13.3.

17.8 Центральная предельная теорема

Давайте теперь сделаем много выборок не из нормального, а из логнормального распределения. Что-то похожее представляет собой распределение времени реакции на многие задачи (особенно связанные с выбором и когда нужно подумать перед нажатием кнопки). Одна выборка будет распределена примерно так:

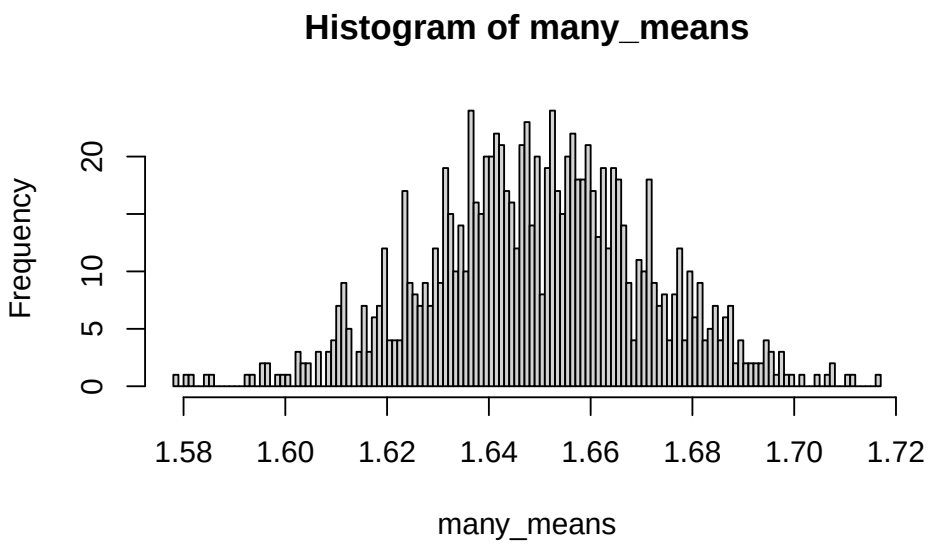
```
hist(rlnorm(10000), breaks = 100)
```



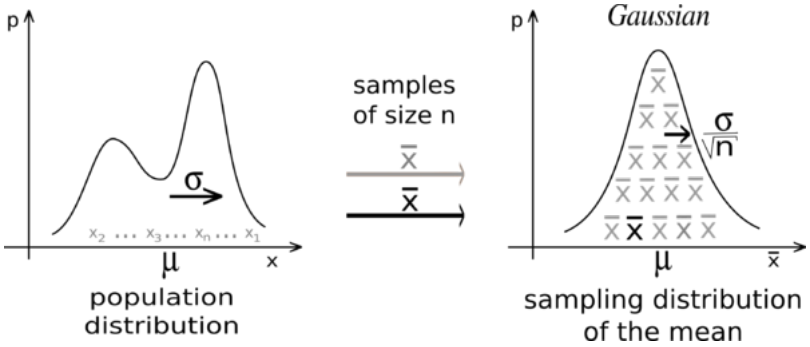
Распределение сильно ассиметрично, но его форма примерно понятна.

А как будут распределены средние многих выборок, взятых из логнормального распределения? Хочется сказать, что так же, но нет!

```
many_means <- replicate(1000, mean(rlnorm(10000)))
hist(many_means, breaks = 100)
```



Удивительно, но средние по выборкам из логнормального распределения будут выглядеть почти нормально! Более того, для других распределений это тоже будет верно: **согласно центральной предельной теореме *(ЦПТ, central limit theorem)*, какой бы ни была форма распределения в генеральной совокупности, выборочное распределение средних будет стремиться к нормальному. При этом чем больше размер выборки, тем ближе выборочное распределение средних будет к нормальному.** Это очень важная фишка, на которой основаны многие статистические тесты.



Не верите? Попробуйте сами! Можете поиграться с разными распределениями с помощью кода (можете посмотреть другие распределения в хэлпе: `?Distributions`). А еще можно потыкать интерактивную Shiny-демонстрацию магии ЦПТ. Я очень рекомендую поиграться с ней. Попробуйте разные значения, посмотрите что будет.

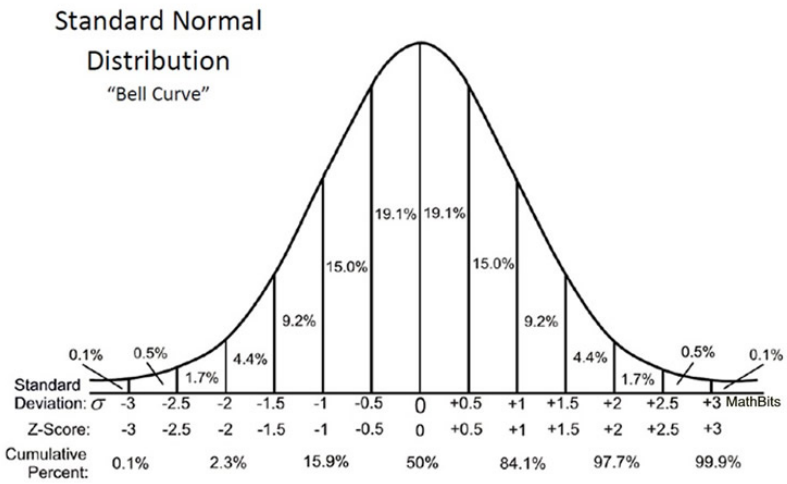
Как только наиграетесь, то сразу станет понятно, почему именно нормальное распределение занимает такое важное место в статистике.

> ***Полезное:* ЦПТ в реальной жизни**
>
> Если выходить за рамки выборочного распределения средних, то центральная предельная теорема говорит нам о том, что сумма слабо зависящих случайных величин, имеющих примерно одинаковое влияние, имеет распределение близкое к нормальному. Например, рост является следствием большого количества генетических и средовых факторов, поэтому он распределен примерно нормально. Однако это нормальное распределение портят такие факторы как пол — он вносит очень сильный вклад в рост, поэтому распределение становится более плоским, т.к. это смесь двух похожих на нормальное распределений. И даже внутри этих распределений есть более и менее влияющие факторы. Поэтому какую бы более узкую генеральную совокупность мы ни рассматривали, распределение никогда не будет идеально нормальным. За исключением, возможно, фундаментальных физических процессов.

17.9 Строим доверительный интервал

Теперь мы знаем достаточно, чтобы построить доверительный интервал своими руками на основе стандартной ошибки. Мы построим самый стандартный вариант - 95% доверительный интервал.

Давайте еще раз посмотрим на нормальное распределение.



Мы хотим поймать симметрично 95% от площади под кривой. Для этого нам нужно отбросить по 2.5% с обоих сторон. Эти 2.5% соответствуют примерно двум стандартным отклонениям от среднего. Если быть точнее, то 1.96. Если быть еще точнее:

```
qnorm(0.975)
```

```
[1] 1.959964
```

Почему 0.975? Потому что мы смотрим квантильную функцию по верхней границе: отсекаем правые 0.025:

```
qnorm(1 - (1 - 0.95)/2)
```

```
[1] 1.959964
```

Давайте сохраним это число. Назовем его `zcr`:

```
zcr <- qnorm(1 - (1 - 0.95)/2)
```

Это количество стандартных отклонений от среднего в нормальном распределении, которое включает в себя ровно 95% площади нормального распределения. Теперь давайте посчитаем стандартную ошибку. Здесь мы знаем стандартное отклонение в генеральной совокупности (это 15), его поделим на корень из размера выборки:

```
sem <- 15/sqrt(length(samp))
```

Чтобы посчитать нижнюю и верхнюю границы доверительного интервала, нам нужно вычесть и прибавить соответственно нужное количество стандартных ошибок:

```
mean(samp) - sem*zcr #
```

```
[1] 97.54778
```

```
mean(samp) + sem*zcr #
```

```
[1] 103.4277
```

Давайте теперь нарисуем сотню доверительных интервалов с помощью `ggplot2`! Цветом обозначим интервалы, которые не поймали истинное значение параметра в центральной совокупности.

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

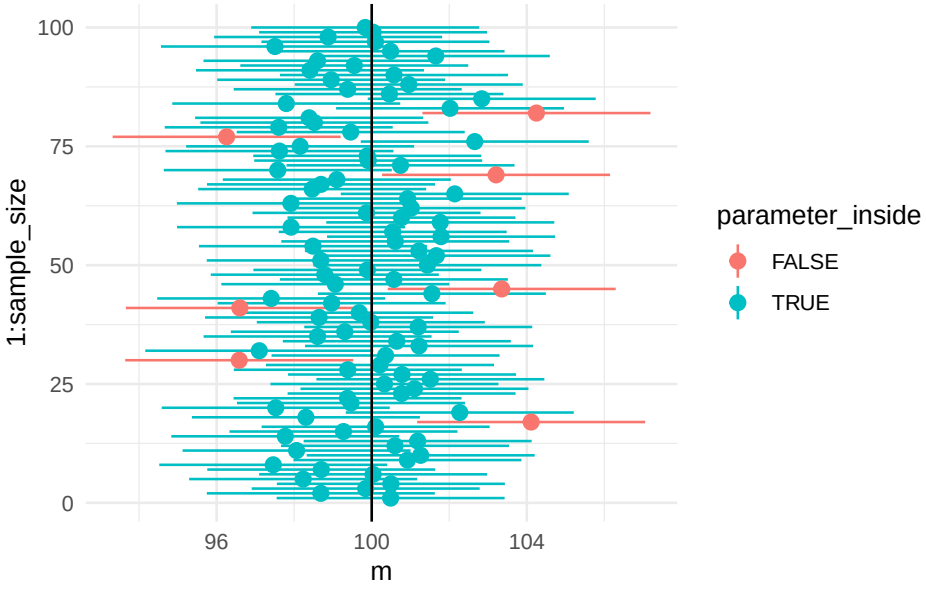
```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

```
sample_size <- 100
set.seed(42)
ci_simulations <- tibble(
  m = replicate(sample_size, mean(rnorm(sample_size, mean = 100, sd = 15))),
```

```
  se = 15/sqrt(sample_size),
  lower = m - se*zcr,
  higher = m + se*zcr,
  parameter_inside = lower<100 & higher>100
)
```

```
many_ci_gg <- ggplot(data = ci_simulations, aes(x = 1:sample_size,y = m)) +
  geom_pointrange(aes(ymin = lower,ymax = higher,colour = parameter_inside))+
  geom_hline(yintercept = 100)+
  coord_flip() +
  theme_minimal()
many_ci_gg
```



Примерно 5% не ловят 100 в 95% доверительный интервал! Примерно это и означает доверительный интервал: где-то в 95% он ловит параметр в генеральной совокупности, а в 5% - нет.

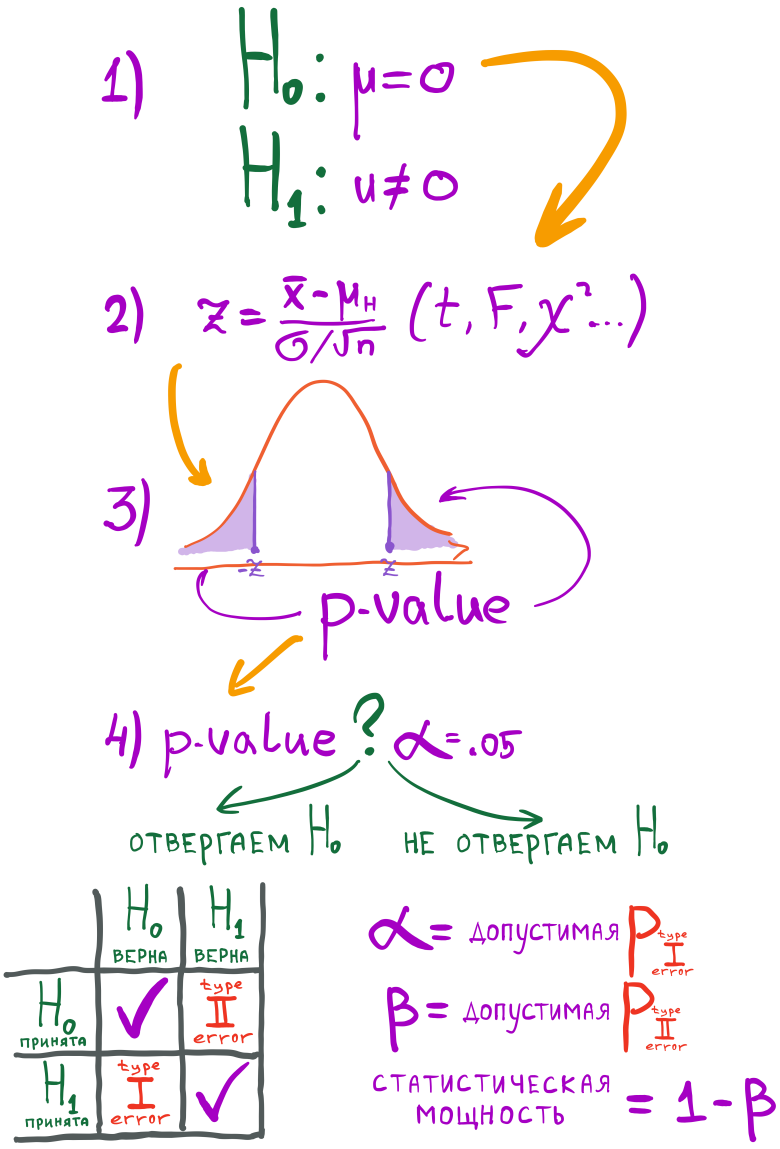
> ***Осторожно:* определение доверительного интервала**
>
> Понятие доверительного интервала вызывает кучу недопонимания и ошибок. Очень многие его интерпретируют, например, как интервал, включающий в себя 95% значений популяции, но это неправильно.

Еще я советую посмотреть вот эту визуализацию: Доверительные интервалы сыпятся как из мешка

17.10 Тестирование значимости нулевой гипотезы

Тестирование значимости нулевой гипотезы (null hypothesis significance testing) — это основной подход в статистике вывода. Вы про него точно слышали (хотя, возможно, не знали, что он так называется), потому что де-факто он является стандартом в психологии, биологии, медицине и многих других науках.

Мы сейчас детально его проведем на примере одного из самых простых статистических тестов - **z-тестов *(z-test).*** Однако та же самая логика стоит и за остальными статистическими тестами.



1. Формулирование нулевой и альтернативной гипотезы.

Сначала мы задаем две гипотезы о параметрах распределения. Одна из них называется нулевой: она обычно включает положение о том, что различий или связи нет или что это различие/связь равно определенному числу. Если мы хотим применить тестирование значимости к нашей “выборке”, то нулевую гипотезу можно будет

сформулировать так:

$$H_0 : \mu = 100$$

Наша нулевая гипотеза заключается в том, что выборка была взята из распределения со средним 100.

Альтернативная или ненулевая гипотеза либо говорит о том, что среднее в генеральной совокупности на самом деле не равно какому-то конкретному числу (в нашем случае — 100) или что две выборки взяты из групп с различным средним и т.п.

$$H_1 : \mu \neq 100$$

Тестирование нулевой гипотезы предполагает подсчет какой-то статистики, а потом вычисление того, какова вероятность получить такой или более радикальный результат при условии, что верна нулевая гипотеза.

Заметьте, мы формулируем гипотезу не про *статистики* в выборке, а про *параметры* в генеральной совокупности, поэтому пользуемся греческими (или большими латинскими) буквами.

Вся дальнейшая логика расчетов будет строиться именно на нулевой гипотезе: мы будем пытаться понять, насколько реалистичны наши результаты при верности нулевой гипотезы, которую мы заранее обозначили. Это похоже на "доказательство от обратного" в геометрии: мы исходим из того, что эффекта не существует и пытаемся прийти к противоречию с данными, чтобы отбросить эту нулевую гипотезу и принять альтернативную гипотезу.

2. Подсчет тестовой статистики по выборке

Следующий этап **тестирования значимости нулевой гипотезы** — подсчет тестовой статистики. Тестовые статистики по своей сути не отличаются от описательных статистик, с которыми мы уже познакомились, но имеют другую функцию. Как и в случае описательных статистик, мы пытаемся выразить информацию о выборке в виде одного числа, но делаем это для того, чтобы сравнить это значение с другими *возможными* значениями, которые мы могли бы получить, если бы наша нулевая гипотеза была верна.

Если мы знаем стандартное отклонение в генеральной совокупности, то можем посчитать z-статистику по формуле:

$$z = \frac{\overline{x} - \mu}{\sigma / \sqrt{N}}$$

```
m <- mean(samp)
sem <- 15/sqrt(length(samp))
z <- (m - 100)/sem
z
```

```
[1] 0.3251482
```

z-статистика — это выборочное среднее, из которого вычтено среднее в генеральной совокупности согласно нашей гипотезе. Получившуюся разницу мы делим на стандартную ошибку.

3. Расчет p-value

И вот мы подобрались к самому важному этапу – расчет **p-value**.

> **p-value** – это вероятность получить такую и более отклоняющуюся тестовую статистику при условии верности нулевой гипотезы.

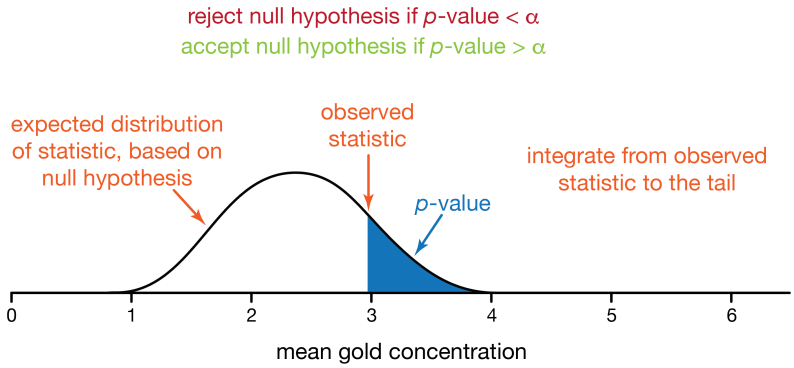
Очень важно понять, как именно рассчитывается *p-value,* потому что на этом основывается сама идея тестирования значимости нулевой гипотезы! Для этого нам нужно вернуться к идее выборочного распределения, только теперь уже не среднего, а z-статистики. Впрочем, выглядеть оно будет абсолютно так же – нормально! Благодаря тому, что мы вычли среднее и поделили на стандартное отклонение, среднее этого нормального распределения будет равно 0, а стандартное отклонение – 1. То есть перед нами снова **стандартное нормальное распределение.**

Благодаря ЦПТ, даже если распределение в генеральной совокупности несколько отличается от нормального, а выборка достаточно большая, то выборочное распределение z-статистик при верности нулевой гипотезы будет (примерно) нормальным.

Теперь нам нужно соотнести нашу z-статистику с теоретическим выборочным распределением z-статистик. Это позволит нам оценить вероятность получить такие и более отличающиеся результаты при допущении верности нулевой гипотезы, т.е. p-value.

Какая вероятность получить z-статистику 0.3251482, если на самом деле нулевая гипотеза верна? Это вопрос с подвохом: как мы выяснили ранее, для непрерывных распределений вероятность получить отдельное число равна 0. Но мы можем посчитать, какая вероятность получить такую же или большую z-статистику!

Графически эту вероятность можно представить как площадь под кривой функции плотности распределения от полученной z-статистики до плюс бесконечности или от минус бесконечности до полученной z-статистики, если z-статистика отрицательная.



Или же можем воспользоваться **функцией накопленной плотности распределения.** Для нормального распределения это можно сделать с помощью уже знакомой нам функции `pnorm()`:

```
pnorm(z)
```

```
[1] 0.6274655
```

`pnorm()` считает от минус бесконечности до заданного числа, а нам нужно наоборот — от заданного числа до плюс бесконечности, потому что z отличается от 0 в большую сторону. Этого можно добиться вычетанием из 1[7]:

```
1 - pnorm(z)
```

```
[1] 0.3725345
```

Обычно это число еще и умножают на 2, потому что мы заранее не знаем, в какую сторону будет отклоняться среднее по нашей выборке. Вернемся к шагу 1: мы сформулировали ненулевую гипотезу таким образом, что среднее генеральной совокупности не равно 100:

$$H_1 : \mu \neq 100$$

[7] или же можно поставить параметру `lower.tail =` функции `pnorm()` значение `FALSE`.

Это означает, что H_1 включает в себя как случаи, когда среднее отклоняется в большую сторону, так и случаи когда среднее отклоняется в меньшую сторону.

```
p <- (1 - pnorm(z))*2
p
```

```
[1] 0.7450689
```

Вот эта число и есть *p-value — вероятность получения такого же и более экстремального значения тестовой статистики при условии, что нулевая гипотеза верна.*

4. Принятие решения о гипотезах

Отлично, мы посчитали *p-value*. В данном конкретном случае он оказался равен 0.7450689. Это много или мало? Фактически это означает, что если нулевая гипотеза верна, то в большинстве случаев мы будем получать z-статистики больше нашей. Короче говоря, это вполне реалистичный случай, если нулевая гипотеза верна.

Значит ли это, что нулевая гипотеза верна? Нет, не значит. При тестировании уровня значимости нулевой гипотезы мы в принципе ничего не можем сказать про верность альтернативной гипотезы. Например, возможно, настоящее среднее в генеральной совокупности, из которой мы взяли выборку, очень мало отличается от 100.

Поэтому если *p-value* большой, мы не можем сделать выводов про верность нулевой и альтернативной гипотезы. Мы можем лишь сказать, что у нас нет оснований отклонить нулевую гипотезу.

Если же *p-value* очень маленький, то здесь у нас появляется больше однозначности. Например, если *p-value* равен .02, мы можем сказать, что ситуация малореалистичная: такие и более сильные отклонения от среднего мы можем получить только раз в 50 случаев, если H_0 верна. Поэтому мы отклоняем H_0 и принимаем H_1.

Насколько маленьким должен быть *p-value*, чтобы отклонить нулевую гипотезу? **Критическое значение *p-value*, при котором отклоняют нулевую гипотезу, называется уровнем** α. Это максимальный уровень ошибки, который мы допускаем в исследовании.

Так получилось исторически, что стандартный уровень α равен .05. Нужно помнить, что .05 — это просто общепринятая

условность, за этим числом не стоит никакого сакрального знания. Просто так получилось.

Очевидно, что такой статистический подход к принятию решений будет периодически приводить к нас ошибкам. Если H_0 на самом деле верна, а мы ее отвергли и приняли H_1, то это **ошибка первого рода** (*type I error*). Вероятность этой ошибки и есть наше критическое значение α. Однако есть вероятность ошибиться и в другую сторону, т.е. ошибочно не отклонить H_0 — это **ошибка второго рода** (*type II error*), эта вероятность обозначается буквой β.

Принятое решение Реальность	H_0 верна	H_1 верна
Не отклоняем H_0	Верный пропуск	Ошибка 2 рода (type II error)
Отклоняем H_0	Ошибка 1 рода (type I error)	Верное попадание

Глава 18

t-тест

18.1 Одновыборочный t-тест

Мы научились делать z-тест. Однако на практике он почти не используется, потому что предполагает, что мы откуда-то знаем стандартное отклонение в генеральной совокупности. Это обычно не так, поэтому мы *оцениваем* стандартное отклонение в генеральной совокупности на основе стандартного отклонения по выборке. Это приводит к тому, что тестовая статистика уже не распределена нормально, а распределена согласно t-распределению. Ну и статистика уже называется t-статистикой.

$$t = \frac{\overline{x} - \mu}{s_x / \sqrt{N}}$$

> ***Полезное:* казалось бы, причем здесь пиво?**
>
> Иногда это t-распределение называют t-распределением Стьюдента, а соответствующий статистический тест - критерий Стьюдента. Дело в том, что его открыл сотрудник пивоварни Гиннесс Уильям Госсет. Сотрудникам Гиннесса было запрещено публиковать научные работы под своим именем, поэтому он написал свою знаменитую работу про t-распределение под псевдонимом "Ученик" (*Student*).

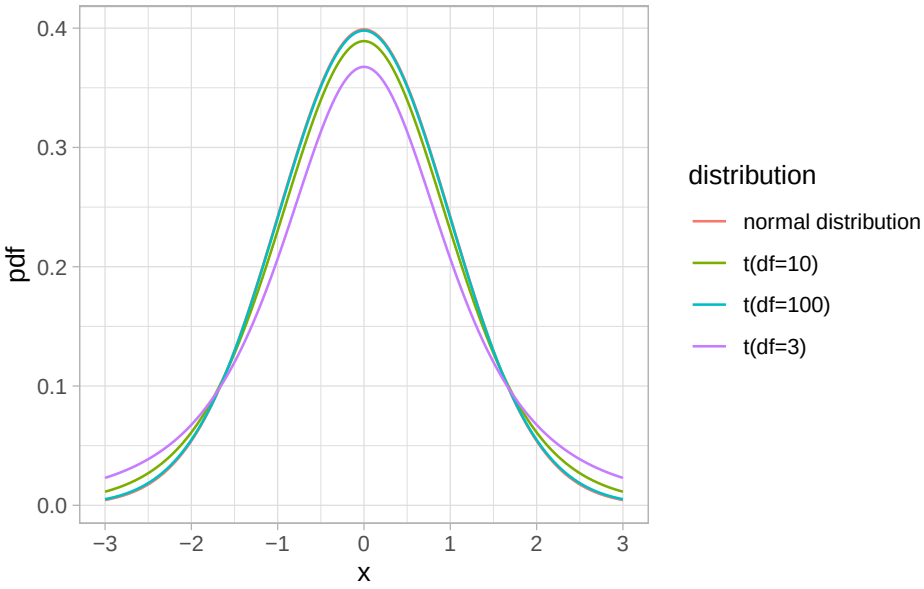
Форма этого распределения очень похожа на форму нормального распределения, но имеет более тяжелые "хвосты" распределения. При этом эта форма зависит от размера выборки: чем больше

выборка, тем ближе распределение к нормальному. Этот параметр распределения называется **степенями свободы *(degrees of freedom)*** и вычисляется как $N - 1$, где N - это размер выборки.

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```



Как видите, чем больше выборка (и количество степеней свободы соответственно), тем ближе t-распределение к стандартному нормальному распределению. При 100 степенях свободы они уже почти не различимы! Поэтому на больших выборках разница между t-тестом и z-тестом будет минимальна, тогда как на маленьких выборках разница может быть значительной.

Давайте посчитаем t-статистику на тех же симулированных данных, что и раньше[1]:

```
set.seed(42)
samp <- rnorm(100, 100, 15)
m <- mean(samp)
sem <- sd(samp)/sqrt(length(samp))
t <- (m - 100)/sem
t
```

```
[1] 0.3122351
```

Давайте для сравнения еще раз посчитаем z-статистику:

```
(m - 100) / (15/sqrt(100))
```

```
[1] 0.3251482
```

Как видите, расчет довольно схожий, разница только в том, откуда мы берем стандартное отклонение. Для z-статистики у нас был заранее известный параметр генеральной совокупности (что обычно не так), для t-статистики мы оценивали стандартное отклонение по выборке.

Давайте теперь посчитаем *p-value.* Мы будем пользоваться не функцией `pnorm()`, а функцией `pt()`, а в качестве параметра распределения нужно указать количество степеней свобод в `df` = (размер выборки минус 1).

```
pt(t, df = length(samp) - 1)
```

```
[1] 0.6222407
```

Функция `pt()` считает от минус бесконечности до t, а нам нужно от t до плюс бесконечности, потому что t больше 0:

```
1 - pt(t, df = length(samp) - 1)
```

```
[1] 0.3777593
```

И не забываем умножать на 2, если мы хотим сделать двусторонний тест.

[1] Благодаря `set.seed(42)` мы можем точно повторить сгенерированные значения.

```
  (1 - pt(t, df = length(samp) - 1))*2
```

```
[1] 0.7555186
```

В отличие от z-теста, t-тест есть в базовом R.

```
  t.test(samp, mu = 100)
```

```
    One Sample t-test

data:  samp
t = 0.31224, df = 99, p-value = 0.7555
alternative hypothesis: true mean is not equal to 100
95 percent confidence interval:
  97.38831 103.58714
sample estimates:
mean of x
 100.4877
```

Да, конечно, мы могли сразу запустить эту функцию и получить результаты. Обычно именно так вы и будете делать. Зато теперь вы знаете, что стоит за всеми числами в результате выполнения функции `t.test()`! Здесь можно увидеть выборочное среднее как оценку среднего в генеральной совокупности, 95% доверительный интервал для оценки среднего в генеральной совокупности, t-статистику, степени свобод и *p-value.*

18.2 Двухвыборочный t-тест

Одна из наиболее часто встречающихся задач при анализе данных - это сравнение средних двух выборок. Для этого нам тоже понадобится t-тест, но теперь H_0 нужно сформулировать по-другому: что две генеральные совокупности (из которых взяты соответствующие выборки) имеют одинаковое среднее.

$$H_0 : \mu_1 = \mu_2$$

Ну а альтернативная гипотеза, что эти две выборки взяты из распределений с разным средним в генеральной совокупности.

$$H_1 : \mu_1 \neq \mu_2$$

Есть две разновидности двухвыборочного t-теста: **зависимый t-тест *(paired t-test; dependent t-test)*** и **независимый t-тест *(unpaired t-test; independent t-test).*** Различие между зависимыми и независимыми тестами принципиальное, мы с ним еще будем сталкиваться при обсуждении других методов, например, дисперсионного анализа (см. Глава 22).

Зависимые тесты предполагают, что каждому значению в одной выборке мы можем поставить соответствующее значение из другой выборки. Обычно это повторные измерения какого-либо признака в разные моменты времени. В независимых тестах нет возможности сопоставить одно значение с другим. Мы уже не можем напрямую соотнести значения в двух выборках друг с другом, более того, размер двух выборок может быть разным!

Использование зависимых и независимых тестов связано с использованием **внутрииндивидуального *(intra-subject design)*** и **межиндивидуального экспериментальных планов *(inter-subject design)*** в планировании научных экспериментов. Даже если вы не планируете в дальнейшем заниматься проведением экспериментов, понимание различий между двумя видами планов поможет вам понять разницу между зависимыми и независимыми тестами.

Допустим, мы хотим исследовать влияние кофеина на скорость реакции у человека. Мы собрали добровольцев на эксперимент, подготовили обычную воду и воду с кофеином, определили методику измерения времени реакции. Что делать дальше? У нас есть два варианта.

1) **Межиндивидуальный экспериментальный план.** Каждому испытуемому мы даем либо воду с кофеином, либо обычную воду. Что именно получит испытуемый получит определяется случайным образом. Таким образом, для **межиндивидуального экспериментального плана** рандомизация *определяет испытуемого в одну из экспериментальных групп.* Для анализа результатов такого исследования нам понадобится **независимый t-тест.**

2) **Внутрииндивидуальный экспериментальный план.** Набрать выборку, каждому испытуемому дать и обычную воду, и воду с кофеином, записывать скорость решения задач после употребления простой воды и после употребления воды с кофеином, соответственно. В данном случае будет случайным образом варьироваться порядок предъявления: одни испытуемые сначала получат обычную воду, а потом воду с кофеином, другие испытуемые — наоборот. Для такого

эксперимента понадобится меньше участников, но оно будет дольше для каждого участника. Более того, в этом случае мы учтем межиндивидуальные различия участников: одни участники в среднем решают задачи быстрее других. Это внутриинидивидуальный экспериментальный дизайн, для анализа результатов которого нам понадобится зависимый t-тест.

Внутрииндивидуальный план	Межиндивидуальный план
Зависимый t-тест	Независимый t-тест

Итак, с тем, когда использовать зависимый, а когда независимый t-тест, более-менее разобрались, давайте опробуем их!

***Полезное:* испытуемые должны быть слепыми котятками**

Как при **внутрииндивидуальном плане,** так и при **межиндивидуальном плане** испытуемый не должен знать какое из экспериментальных условий он получает **(слепое тестирование; *blind study*),** а в идеале этого не должен знать даже сам экспериментатор, который дает напиток и измеряет показатели **(двойное слепое тестирование; *double blind study*).** Поскольку в внутрииндивидуальном плане испытуемый получает несколько экспериментальных условий, у него есть возможность сравнить различные условия, поэтому организовать слепое тестирование сложнее. Сделать экспериментальные условия максимально похожими друг на друга для испытуемого – особое искусство исследователя-экспериментатора.

18.2.1 Двухвыборочный зависимый t-тест

Двухвыборочный зависимый t-тест — это то же самое, что и одновыборочный t-тест, только для разницы между связанными значениями. Поскольку наша нулевая гипотеза звучит, что средние должны быть равны,

$$H_0 : \mu_1 = \mu_2$$

то при верности нулевой гипотезы

$$\mu_1 - \mu_2 = 0$$

.

Тогда вместо x подставим d — разницу (вектор разниц) между парами значений. Получаем вот что:

$$t = \frac{\overline{x} - \mu}{s_x/\sqrt{N}} = \frac{\overline{d} - (\mu_1 - \mu_2)}{s_d/\sqrt{N}} = \frac{\overline{d} - 0}{s_d/\sqrt{N}} = \frac{\overline{d}}{s_d/\sqrt{N}}$$

Мы будем использовать данные с курса по статистике Университета Шеффилда про эффективность диет. Мы вытащим оттуда данные по диете номер 1 и посмотрим, действительно ли она помогает сбросить вес.

```
library(tidyverse)
diet <- readr::read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/s
```

```
Rows: 78 Columns: 7
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (7): Person, gender, Age, Height, pre.weight, Diet, weight6weeks

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
diet1 <- diet %>%
  filter(Diet == 1)
```

Провести двухвыборочный t-тест в R можно двумя базовыми способами. Первый вариант - это дать два вектора значений. Это удобно в случае широкого формата данных.

```
t.test(diet1$pre.weight, diet1$weight6weeks, paired = TRUE)
```

```

    Paired t-test

data:  diet1$pre.weight and diet1$weight6weeks
t = 7.2168, df = 23, p-value = 2.397e-07
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 2.354069 4.245931
sample estimates:
mean difference
            3.3
```

Второй вариант - используя формулы. Это удобно при длинном формате данных:

```
diet1_long <- diet1 %>%
  pivot_longer(
  cols = c(pre.weight, weight6weeks),
  names_to = "when",
  values_to = "weight"
  )


t.test(weight ~ when, data = diet1_long, paired = TRUE)
```

```
    Paired t-test

data:  weight by when
t = 7.2168, df = 23, p-value = 2.397e-07
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 2.354069 4.245931
sample estimates:
mean difference
            3.3
```

```
t.test(diet1_long$weight ~ diet1_long$when, paired = TRUE)
```

```
    Paired t-test

data:  diet1_long$weight by diet1_long$when
t = 7.2168, df = 23, p-value = 2.397e-07
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 2.354069 4.245931
sample estimates:
mean difference
            3.3
```

В обоих вариантах мы использовали `paired = TRUE` , чтобы обозначить использование именно зависимого (т.е. парного) t-теста.

Внутрииндивидуальный план	Межиндивидуальный план
Зависимый t-тест `t.test(..., paired = TRUE)`	Независимый t-тест `t.test(..., paired = FALSE)`

18.2.2 Двухвыборочный независимый t-тест

В случае независимого t-теста формула отличается. Однако гипотеза остается такой же и логика примерна та же.

У двух выборок могут различаться стандартные отклонения, поэтому для подсчет стандартной ошибки разницы средних для независимых выборок нужно сначала посчитать **объединенное стандартное отклонение (pooled standard deviation)**:

$$s^2_{pool} = \frac{(n_1 - 1)s^2_1 + (n_2 - 1)s^2_2}{(n_1 - 1) + (n_2 - 1)}$$

Тогда стандартная ошибка разницы средних считается следующим образом:

$$se_{m_1 - m_2} = \sqrt{(s^2_{pool})(\frac{1}{n_1} + \frac{1}{n_2})}$$

Выглядит сложно, но по своей сути это что-то вроде усредненного стандартного отклонения (с учетом размеров выборок). Ну а t-статистика затем считается просто:

$$t = \frac{(m_1 - m_2) - (\mu_1 - \mu_2)}{se_{m_1 - m_2}} = \frac{(m_1 - m_2) - 0}{se_{m_1 - m_2}} = \frac{m_1 - m_2}{se_{m_1 - m_2}}$$

Давайте теперь опробуем независимый t-тест для сравнения веса испытуемых двух групп после диеты. Мы снова воспользуемся функцией `t.test()`, но теперь уже поставим `paired = FALSE`:

```
diet12 <- diet %>%
  filter(Diet %in% 1:2)
t.test(weight6weeks ~ Diet, data = diet12, paired = FALSE)
```

```
    Welch Two Sample t-test

data:  weight6weeks by Diet
t = 0.5711, df = 48.724, p-value = 0.5706
alternative hypothesis: true difference in means between group 1 and group 2 is not equ
95 percent confidence interval:
 -3.753268  6.732897
sample estimates:
mean in group 1 mean in group 2
       69.57500        68.08519
```

Если присмотритесь, то увидите, что со степенями свободы что-то странное. Они дробные! Дело в том, что мы провели не совсем "настоящий" t-тест, а его очень близкую альтернативу под названием **тест Уэлча** *(Welch test)*, который иногда называют **поправкой Уэлча** к t-тесту. Эта поправка позволяет тесту лучше справляться с выборками с разной дисперсией. Даже если у нас нет проблем с разной дисперсией, то от поправки Уэлча хуже не будет, поэтому это вариант по умолчанию в R.

Но если его хочется отключить, то нужно поставить `var.equal = TRUE`.

```
t.test(weight6weeks ~ Diet, data = diet12, paired = FALSE, var.equal = TRUE)
```

```
    Two Sample t-test

data:  weight6weeks by Diet
t = 0.5645, df = 49, p-value = 0.575
alternative hypothesis: true difference in means between group 1 and group 2 is not equ
95 percent confidence interval:
 -3.813769  6.793399
sample estimates:
mean in group 1 mean in group 2
       69.57500        68.08519
```

18.3 Допущения t-теста

Чтобы подсчет *p-value* был корректным, некоторые допущения должны быть выполнены.

1. Шкала должна быть не ниже интервальной, т.е. t-тест можно применять только в случа с **интервальной шкалой**

или **шкалой отношений.** Иначе мы не можем говорить о различии средних – ведь только для этих шкал в принципе можно считать среднее.

2. **Нормальное распределение.** Это самое известное допущение, но чуть ли не наименее важное. Часто утверждается, что выборки должны быть взяты из нормального распределения. Это не совсем так: да, верно, это достаточное условие, но не необходимое. В первую очередь, важно именно выборочное распределение (средних разниц или разниц средних), которое достигается за счет центральной предельной теоремы (Глава 17.8) особенно при большой выборке (Глава 17.7). Отсюда и все правила в духе "для t-теста распределение должно быть нормальным, но если n > 30, то это необязательно". Откуда именно 30? Да ниоткуда, просто. Как и со всеми точными числами в статистике.

Для проверки на нормальность существуют различные статистические тесты. Самый известный из них — тест Шапиро-Уилка. Его можно провести в R при помощи функции `shapiro.test()`.

```
shapiro.test(samp)
```

```
    Shapiro-Wilk normality test

data:  samp
W = 0.98122, p-value = 0.1654
```

В нашем случае *p-value* больше 0.05, что логично: мы взяли эту выборку именно из нормального распределения. Если *p-value* меньше уровня α, который у нас стандартно 0.05, то мы можем отвергнуть нулевую гипотезу о том, что выборка взята из нормального распределения. Если это так, то нам нужно, по идее, отказаться от t-теста и использовать непараметрический тест, который не имеет требований к распределению исследуемой переменной.

Однако проведения теста на нормальность для проверки допущения о нормальности — вещь довольно бессмысленная. Дело в том, что тест Шапиро-Уилка — это такой же статистический тест, как и все прочие: чем больше выборка, тем с большей вероятностью он "поймает" отклонения от нормальности, чем меньше выборка, тем с меньшей вероятностью он обнаружит даже серьезные отклонения от нормальности. А нам-то нужно наоборот! При большой выборке отклонения от нормальности нам не особо страшны, а при маленькой тест все равно ничего

не обнаружит. Более того, идеально нормальных распределений в природе вообще почти не существует! А это значит, что при достаточно большой выборке тест Шапиро-Уилка практически всегда будет находить отклонения от нормальности. Все это делает его малоинформативным при тестировании допущения о нормальности. Это же верно и для других тестов на нормальность.

Другой способ проверять допущение о нормальности — это проверять визуально с помощью гистограммы или *Q-Q plot* (см. (**ref-lm_a?**))

2. Для двувыборочного независимого t-теста выборки должны быть взяты из распределения **с одинаковыми дисперсиями**. Однако это не так критично с применением поправки Уэлча.

3. **Независимость значений (или пар)** в выборке. Типичным примером нарушения независимости является случай, когда t-тест применяется на неусредненных (например, по испытуемому) значениях. Еще один пример нарушения независимости — использование одного наблюдения несколько раз или использование одного и того же испытуемого несколько раз. Определить независимость можно следующим мысленным экспериментом: могу ли я хоть как-нибудь предсказать следующее значение в выборке? Например, в случае с несколькими значениями от одного испытуемого я могу ориентироваться на его предыдущие результаты и предсказать последующие результаты лучше, чем на основе простого среднего по всем остальным значениям. Это значит, что допущение о независимости нарушено.

18.4 Непараметрические аналоги t-теста

Если выборка не очень большая и взята из сильно ассиметричного распределения или выборка представляет собой порядковые данные, то можно воспользоваться непараметрическими альтернативами для t-теста.

Непараметрические тесты не имеют допущений о распределении, что делает их более универсальными. Большинство подобных тестов подразумевает превращение данных в ранги, т.е. внутри этих тестов происходит преобразование в ранговую шкалу. Такое преобразование может снизить статистическую мощность теста и привести к повышению вероятности ошибки второго рода.

18.4.1 Тест Уилкоксона

Непараметрический аналог двустороннего *зависимого* t-теста называется **тестом Уилкоксона.** Функция для него называется `wilcox.test()`, и она имеет такой же синтаксис, как и `t.test()`.

```
wilcox.test(weight ~ when, data = diet1_long, paired = TRUE)
```

```
Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...): cannot
compute exact p-value with ties
```

```
    Wilcoxon signed rank test with continuity correction
```

```
data:  weight by when
V = 299, p-value = 2.203e-05
alternative hypothesis: true location shift is not equal to 0
```

> ***Осторожно:* предупреждение о повторяющихся рангах**
>
> Не пугайтесь, когда видите сообщение *"Есть совпадающие значения: не могу высчитать точное p-значение"*. Это означает, что в ваших данных есть повторяющиеся значения, поэтому расчет p-value в данном случае — это некоторая апроксимация, но в большинстве случаев это не играет серьезной роли, разве что у вас действительно очень много повторяющихся значений.

18.4.2 Тест Манна-Уитни

Непараметрическим аналогом двустороннего *независимого* t-теста является **тест Манна-Уитни.** Для него тоже используется функция `wilcox.test()`, только в данном случае с параметром `paired = FALSE`.

```
wilcox.test(weight ~ when, data = diet1_long, paired = FALSE)
```

```
Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...): cannot
compute exact p-value with ties
```

```
    Wilcoxon rank sum test with continuity correction
```

```
data:  weight by when
W = 357.5, p-value = 0.1546
alternative hypothesis: true location shift is not equal to 0
```

Внутрииндивидуальный план	Межиндивидуальный план
Зависимый t-тест: `t.test(..., paired = TRUE)`	Независимый t-тест: `t.test(..., paired = FALSE)`
Тест Уилкоксона: `wilcox.test(..., paired = TRUE)`	Тест Манна-Уитни: `wilcox.test(..., paired = FALSE)`

Глава 19

Тесты хи-квадрат

Тесты хи-квадрат – это целый набор статистических методов для исследования связи между переменными. Значения делятся по группам

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

E_i – ожидаемая частота, O_i – наблюдаемая частота.

Соответственно, чем *больше* расхождения между ожидаемым распределением по группам и реальным, тем *больше* статистика χ^2 и тем меньше *p-value.*

19.0.1 Тест хи-квадрат Пирсона на независимость

Тест хи-квадрат Пирсона на независимость – это статистический метод, используемый для определения того, есть ли статистическая связь между категориальными переменными.

Например, с помощью этого теста мы можем посмотреть, различается ли распределение по полу супергероев у двух основных издателей супергероики – Marvel Comics и DC Comics.

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
                   na = c("-", "-99", "NA", " "))
```

```
New names:
* `` -> `...1`

Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)

Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Для начала нам нужно создать таблицу сопряженности (contingency table) для двух переменных с помощью встроенной функции `table()`. Эта функция похожа на `count()` из пакета {dplyr}.

```
gender_publisher <- heroes %>%
  drop_na(Gender) %>%
  filter(Publisher %in% c("Marvel Comics", "DC Comics")) %>%
  select(Gender, Publisher)

gender_publisher
```

```
# A tibble: 577 x 2
   Gender Publisher
   <chr>  <chr>
 1 Male   Marvel Comics
 2 Male   DC Comics
 3 Male   Marvel Comics
 4 Male   Marvel Comics
 5 Male   Marvel Comics
 6 Male   DC Comics
 7 Female Marvel Comics
 8 Male   Marvel Comics
 9 Male   Marvel Comics
10 Male   Marvel Comics
# i 567 more rows
```

```
table(gender_publisher)
```

```
        Publisher
Gender   DC Comics Marvel Comics
  Female        61           111
  Male         153           252
```

С помощью дженерик-функции `summary()` примененной на построенной таблице сопряженности можно посчитать тест на независимость хи-квадрат.

```
summary(table(gender_publisher))
```

```
Number of cases in table: 577
Number of factors: 2
Test for independence of all factors:
    Chisq = 0.27673, df = 1, p-value = 0.5988
```

То же самое можно сделать и с помощью специальной функции `chisq.test()`:

```
chisq.test(table(gender_publisher))
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  table(gender_publisher)
X-squared = 0.18649, df = 1, p-value = 0.6659
```

Как можно заметить, в этом случае p-value несколько отличается. Дело в том, что *p-value* для хи-квадрат рассчитывается по непрерывному хи-квадрат распределению, тогда как хи-квадрат статистики распределены таким образом только в случае достаточно большой выборке. Чтобы скорретировать на малый размер групп используется поправка на непрерывность Йейтса.

```
chisq.test(table(gender_publisher), correct = FALSE)
```

```
    Pearson's Chi-squared test

data:  table(gender_publisher)
X-squared = 0.27673, df = 1, p-value = 0.5988
```

Другой подход для более точного подсчета p-value – это использование точного теста Фишера:

```
fisher.test(table(gender_publisher))
```

```
    Fisher's Exact Test for Count Data

data:  table(gender_publisher)
p-value = 0.6381
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.6120819 1.3321610
sample estimates:
odds ratio
 0.9053112
```

Глава 20

Ковариация и корреляция

Возьмем новый набор данных, на этот раз про американских студентов, вес их рюкзаков и проблемы со спиной. Этот набор данных хранится в пакете `Stat2Data` — пакет с большим количеством разнообразных данных.

```
install.packages("Stat2Data")
```

С помощью функции `data()` загрузим набор данных `Backpack`:

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3

Warning: package 'stringr' was built under R version 4.2.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become er
```

```
library(Stat2Data)
data(Backpack)
```

Давайте посмотрим, что внутри этой переменной:

```
skimr::skim(Backpack)
```

Таблица 20.1: Data summary

Name	Backpack
Number of rows	100
Number of columns	9
_____	
Column type frequency:	
factor	3
numeric	6
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Major	0	1	FALSE	41	Bio: 9, Bus: 8, LS: 7, ME: 6
Sex	0	1	FALSE	2	Fem: 55, Mal: 45
Status	0	1	FALSE	2	U: 97, G: 3

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
BackpackWeight	0	1	11.66	5.77	2.00	8.00	11.00	14.25	35.00	▇▇▆▁▁
BodyWeight	0	1	153.05	29.40	105.00	130.00	147.50	170.00	270.00	▃▇▅▁▁
Ratio	0	1	0.08	0.04	0.02	0.05	0.07	0.10	0.18	▃▇▅▂▁
BackProblems	0	1	0.32	0.47	0.00	0.00	0.00	1.00	1.00	▇▁▁▁▃
Year	0	1	3.20	1.39	0.00	2.00	3.00	4.00	6.00	▃▇▇▃▂
Units	0	1	14.27	2.81	0.00	13.00	15.00	16.00	19.00	▁▁▃▇▂

С помощью `?Backpack` можно получить подробное описание колонок этого датасета.

Например, можно заметить, что масса как самих студентов, так и их рюкзаков выражена в фунтах. Давайте создадим новые переменные `backpack_kg` и `body_kg`, в которых будет записан вес (рюкзаков и самих студентов соответственно) в понятным для нас килограммах. Новый набор данных сохраним под названием `back`.

```
back <- Backpack %>%
  mutate(backpack_kg = 0.45359237 * BackpackWeight,
         body_kg = 0.45359237 * BodyWeight)
```

До этого мы говорили о *различиях* между выборками. Теперь мы будем говорить о *связи* между переменными.

20.1 Ковариация

Самая простая мера связи между двумя переменными — это **ковариация *(covariation).*** Если **ковариация положительная**, то чем *больше* одна переменная, тем *больше* другая переменная. При **отрицательной ковариации** все наоборот: чем *больше* одна переменная, тем *меньше* другая. **Ковариация** между переменными x и y часто обозначается буквой σ_{xy}. Похоже на дисперсию и стандартное отклонение (см. Глава 12.5.2), не правда ли? И скоро мы убедимся, что это не случайно!

- Формула ковариации:

$$\sigma_{xy} = cov(x, y) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n}$$

- Оценка ковариации по выборке:

$$\hat{\sigma}_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n-1}$$

В R есть функция `cov()` для подсчета ковариации. Эта функция считает сразу **матрицу ковариаций *(covariation matrix)*** для всех сочетаний колонок на входе:

```
back %>%
  select(body_kg, backpack_kg) %>%
  cov()
```

```
              body_kg backpack_kg
body_kg     177.807700    6.601954
backpack_kg   6.601954    6.838333
```

Полученная матрица симметрична: порядок двух переменных не имеет значения. Но что мы имеем по основной диагонали – это, получается, ковариация переменной с самой собой? Давайте подставим в формулу для ковариации x вместо y, чтобы узнать, что скрывается под ковариацией переменной с самой собой:

$$\sigma_{xx} = cov(x, x) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(x_i - \overline{x})}{n} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}$$

Знакомая формула? Это формула дисперсии (см. Глава 12.5.2)! Выходит, что **ковариация переменной с самой собой – это дисперсия.** Таким образом, **в матрице ковариаций по основным диагоналям находятся дисперсии переменных.**

$$\begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}$$

> ***Полезное:* variance и covariance**
>
> В английском языке **дисперсия – *variance***, а **ковариация – *covariance***, что значительно упрощает запоминание связи этих двух понятий.

Что интересно, в R это тоже отображено: посчитать матрицу ковариаций можно как с помощью функции `cov()`, так и уже знакомой нам функции `var()` для расчета дисперсии. Различаются эти функции только дополнительными параметрами.

```
back %>%
  select(body_kg, backpack_kg) %>%
  cov()
```

```
              body_kg backpack_kg
body_kg     177.807700    6.601954
backpack_kg   6.601954    6.838333
```

```
back %>%
  select(body_kg, backpack_kg) %>%
  var()
```

```
              body_kg backpack_kg
body_kg     177.807700    6.601954
backpack_kg   6.601954    6.838333
```

***Для продвинутых:* Ковариационная матрица с точки зрения линейной алгебры**

Расчет матрицы ковариаций легко представить в виде матричных операций:

1. Сначала мы вычтем из каждого столбца среднее по столбцу:

```
X <- back %>%
  select(body_kg, backpack_kg) %>%
  as.matrix() %>%
  apply(2, function(x) x - mean(x))
```

2. Затем транспонируем матрицу (поменяем местами столбцы и строки) и перемножим саму на себя:

$$X^T X$$

```
t(X) %*% X
```

```
              body_kg backpack_kg
body_kg     17602.9623    653.5934
backpack_kg   653.5934    676.9950
```

3. Поделим матрицу на n или $n-1$ (если мы хотим оценить матрицу ковариаций в генеральной совокупности по выборке), где n – количество наблюдений. Это и будет ковариационной матрицей!

$$C_X = \frac{X^T X}{n-1}$$

```
  t(X) %*% X / (nrow(back) - 1)
```

```
              body_kg backpack_kg
body_kg     177.807700    6.601954
backpack_kg   6.601954    6.838333
```

```
  back %>%
    select(body_kg, backpack_kg) %>%
    cov()
```

```
              body_kg backpack_kg
body_kg     177.807700    6.601954
backpack_kg   6.601954    6.838333
```

Ковариации имеют большое значение в статистике. Расчет ковариационной матрицы – это необходимый этап для многих многомерных методов, например, для анализа главных компонент (см. Глава 24.2). Однако у ковариации есть серьезное ограничение – ее размер привязан к исходной шкале, поэтому сложно оценить, насколько ковариация большая или маленькая. Поэтому на практике гораздо больше используются коэффициенты корреляции.

20.2 Корреляция

Корреляцией обычно называют любую связь между двумя переменными, это просто синоним слова “ассоциация”. Если вдруг слово “корреляция” вам еще непривычно, то попробуйте мысленно заменять “корреляцию” на “ассоциацию”, а “коррелирует” на “связано”. **Коэффициент корреляции** — это уже конкретная математическая формула, которая позволяет посчитать эту связь и принимает значения от -1 до 1.[1]

- Если коэффициент корреляции **положительный**, то чем **больше** значения в одной переменной, тем **больше** значения в другой переменной.

[1] Впрочем, это не так уж и важно: на практике часто опускают слово “коэффициент” и называют корреляцией как и просто связь, так и ее способ измерения.

- Если коэффициент корреляции **отрицательный**, то чем **больше** значения в одной переменной, тем **меньше** значения в другой переменной.
- Если коэффициент корреляции равен 0, то изменения одной переменной не связано с изменениями в другой переменной.

20.2.1 Коэффициент корреляции Пирсона

Самый известный коэффициент корреляции - коэффициент корреляции Пирсона:

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} = \frac{1}{n}\sum_{i=1}^{n} z_{x,i} z_{y,i}$$

Оценка коэффициента корреляции Пирсона по выборке:

$$r_{xy} = \frac{\hat{\sigma}_{xy}}{\hat{\sigma}_x \hat{\sigma}_y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} = \frac{1}{n-1}\sum_{i=1}^{n} z_{x,i} z_{y,i}$$

Коэффициент корреляции Пирсона можно понимать по-разному. С одной стороны, это просто ковариация, нормированная на стандартное отклонение обоих переменных. С другой стороны, можно понимать это как среднее произведение z-оценок (Глава 12.5.3.1).

Корреляцию в R можно посчитать с помощью функции `cor()`:

```
back %>%
  select(body_kg, backpack_kg) %>%
  cor()
```

```
             body_kg backpack_kg
body_kg     1.0000000   0.1893312
backpack_kg 0.1893312   1.0000000
```

Для тестирования уровня значимости нулевой гипотезы для корреляции есть функция `cor.test()`. В случае с коэффициентами корреляции, нулевая гипотеза формулируется как отсутствие корреляции (т.е. она равна нулю) в генеральной совокупности.

```
cor.test(back$backpack_kg, back$body_kg)
```

```
    Pearson's product-moment correlation

data:  back$backpack_kg and back$body_kg
t = 1.9088, df = 98, p-value = 0.05921
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.007360697  0.371918344
sample estimates:
      cor
0.1893312
```

Результат выполнения этой функции очень похож на то, что мы получали при проведении t-теста.

20.2.2 Непараметрические коэффициенты корреляции

У коэффициента корреляции Пирсона, как и у t-теста, есть свои непараметрические братья: коэффициент корреляции Спирмена и коэффициент корреляции Кэнделла. Из них чаще используется коэффициент корреляции Спирмена. Посчитать его можно с помощью той же функции `cor.test()`, задав соответствующее значение параметра `method =`:

```
cor.test(back$backpack_kg, back$body_kg, method = "spearman")
```

```
Warning in cor.test.default(back$backpack_kg, back$body_kg, method =
"spearman"): Cannot compute exact p-value with ties
```

```
    Spearman's rank correlation rho

data:  back$backpack_kg and back$body_kg
S = 131520, p-value = 0.03527
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.2108001
```

```
cor.test(back$backpack_kg, back$body_kg, method = "kendall")
```

```
    Kendall's rank correlation tau

data:  back$backpack_kg and back$body_kg
z = 2.083, p-value = 0.03725
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.1478736
```

> Заметьте, в данном случае два метода хотя и привели к схожим размерам корреляции, но в одном случае *p-value* оказался больше 0.05, а в другом случае - меньше 0.05. Выбирать тест *a posteriori* на основе того, какие результаты вам нравятся больше, — плохая практика (Глава 25.1). Не надо так делать.

20.3 Корреляционная матрица

Возможно, вы нашли что-то более интересное для проверки гипотезы о корреляции. Например, вы еще хотите проверить гипотезу о связи количества учебных кредитов и массе рюкзака: логично предположить, что чем больше студент набрал себе курсов, тем тяжелее его рюкзак (из-за большего количества учебников). Или что студенты к старшим курсам худеют и становятся меньше. Или что те, кто набрал себе много курсов, меньше питаются и от того меньше весят. В общем, хотелось бы прокоррелировать все интересующие нас переменные со всеми. Это можно сделать с помощью функции `cor()`:

```
back %>%
  select(body_kg, backpack_kg, Units, Year) %>%
  cor()
```

```
               body_kg backpack_kg       Units        Year
body_kg      1.00000000  0.18933115 -0.23524088 -0.09301727
backpack_kg  0.18933115  1.00000000  0.09438453  0.05762194
Units       -0.23524088  0.09438453  1.00000000 -0.02946373
Year        -0.09301727  0.05762194 -0.02946373  1.00000000
```

В корреляционной матрице все значения находятся в диапазоне от -1 до 1, а по основной диагонали находятся единицы:

$$\begin{bmatrix} 1 & \rho_{xy} \\ \rho_{xy} & 1 \end{bmatrix}$$

И действительно: переменная коррелирует сама с собой идеально, то есть коэффициент корреляции равен 1.

***Для продвинутых:* Корреляционная матрица с точки зрения линейной алгебры**

Корреляционную матрицу можно построить разными способами, но самый простой – сделать z-преобразование переменных (Глава 12.5.3.1), а для полученных z-оценок посчитать ковариационную матрицу.

1. Произведем z-преобразования каждого столбца:

```
X <- back %>%
  select(body_kg, backpack_kg) %>%
  as.matrix() %>%
  scale()
```

2. Затем транспонируем полученную матрицу и перемножим саму на себя по правилам линейной алгебры:

$$X^T X$$

```
t(X) %*% X
```

```
             body_kg backpack_kg
body_kg     99.00000    18.74378
backpack_kg 18.74378    99.00000
```

3. Осталось поделить результаты матрицу на $n - 1$, где n – количество наблюдений. Это и будет ковариационной матрицей!

$$COR_X = \frac{X^T X}{n - 1}$$

```
t(X) %*% X / (nrow(back) - 1)
```

```
             body_kg backpack_kg
body_kg     1.0000000   0.1893312
backpack_kg 0.1893312   1.0000000
```

```
  back %>%
    select(body_kg, backpack_kg) %>%
    cov()
```

```
              body_kg backpack_kg
body_kg     177.807700    6.601954
backpack_kg   6.601954    6.838333
```

Но функция `cor()`не позволяет посчитать *p-value* для этих корреляций! Функция `cor.test()` позволяет получить *p-value*, но только для одной пары переменных.

На помощь приходит пакет `psych` с функцией `corr.test()`:

```
 back %>%
   select(body_kg, backpack_kg, Units, Year) %>%
   psych::corr.test()
```

```
Call:psych::corr.test(x = .)
Correlation matrix
            body_kg backpack_kg Units  Year
body_kg        1.00        0.19 -0.24 -0.09
backpack_kg    0.19        1.00  0.09  0.06
Units         -0.24        0.09  1.00 -0.03
Year          -0.09        0.06 -0.03  1.00
Sample Size
[1] 100
Probability values (Entries above the diagonal are adjusted for multiple tests.)
            body_kg backpack_kg Units Year
body_kg        0.00        0.30  0.11    1
backpack_kg    0.06        0.00  1.00    1
Units          0.02        0.35  0.00    1
Year           0.36        0.57  0.77    0

 To see confidence intervals of the correlations, print with the short=FALSE option
```

Тем не менее, если у вас много гипотез для тестирования, то у вас появляется проблема: вероятность выпадения статистически значимых результатов сильно повышается. Даже если эти переменные никак не связаны друг с другом.

Эта проблема называется **проблемой множественных сравнений (multiple comparisons problem)** [2]. Если мы проверяем сразу

[2]“Проблема множественных сравнений” - это устоявшийся термин, который

несколько гипотез, то у нас возрастает **групповая вероятность ошибки первого рода (Family-wise error rate)** — вероятность ошибки первого рода для хотя бы одной из множества гипотез.

Например, если вы коррелируете 10 переменных друг с другом, то вы проверяете 45 гипотез о связи. Пять процентов из этих гипотез, т.е. в среднем 2-3 гипотезы у вас будут статистически значимыми даже если никаких эффектов на самом деле нет!

Поэтому если вы проверяете сразу много гипотез, то необходимо применять **поправки на множественные сравнения (multiple testing correction)**. Эти поправки позволяют контролировать групповую вероятность ошибки первого рода на желаемом уровне. Самая простая и популярная поправка на множественные сравнения — **поправка Бонферрони (Bonferroni correction)**. Она считается очень просто: мы просто умножаем p-value на количество проверяемых гипотез!

```
back %>%
  select(body_kg, backpack_kg, Units, Year) %>%
  psych::corr.test(adjust = "bonferroni")
```

```
Call:psych::corr.test(x = ., adjust = "bonferroni")
Correlation matrix
            body_kg backpack_kg Units  Year
body_kg        1.00        0.19 -0.24 -0.09
backpack_kg    0.19        1.00  0.09  0.06
Units         -0.24        0.09  1.00 -0.03
Year          -0.09        0.06 -0.03  1.00
Sample Size
[1] 100
Probability values (Entries above the diagonal are adjusted for multiple tests.)
            body_kg backpack_kg Units Year
body_kg        0.00        0.36  0.11    1
backpack_kg    0.06        0.00  1.00    1
Units          0.02        0.35  0.00    1
Year           0.36        0.57  0.77    0

 To see confidence intervals of the correlations, print with the short=FALSE option
```

Это очень “дубовая” и излишне консервативная поправка. Да, она гарантирует контроль групповую вероятности ошибки

используется и в случае множественных корреляций, и в случае множественных сравнений средних и в любых других случаях с тестированием нескольких гипотез одновременно.

первого рода, но при этом сильно повышает вероятность ошибки второго рода — вероятность пропустить эффект, если он на самом деле существует. Поэтому по умолчанию в R используется более либеральная поправка на множественные сравнения под названием **поправка Холма** или **поправка Холма-Бонферрони (Holm-Bonferroni correction)**, которая, тем не менее, тоже гарантирует контроль групповой вероятности ошибки первого рода.

Альтернативный подход к решению проблемы множественных сравнений — это контроль **средней доли ложных отклонений (False Discovery Rate; FDR)** на на уровне не выше уровня α. Это более либеральный подход: в данном случае мы контролируем, что ложно-положительных результатов у нас не больше, например, 5%. Такой подход применяется в областях, где происходит масштабное множественное тестирование. Попытка контролировать *групповую вероятность ошибки первого уровня* не выше уровня α привела бы к чрезвычайно низкой вероятности обнаружить хоть какие-нибудь эффекты (т.е. к низкой статистической мощности).

Самая известная поправка для контроля *средней доли ложных отклонений* — это **поправка Бенджамини – Хохберга (Benjamini-Hochberg correction).**

```
back %>%
  select(body_kg, backpack_kg, Units, Year) %>%
  psych::corr.test(adjust = "BH")
```

```
Call:psych::corr.test(x = ., adjust = "BH")
Correlation matrix
            body_kg backpack_kg Units  Year
body_kg        1.00        0.19 -0.24 -0.09
backpack_kg    0.19        1.00  0.09  0.06
Units         -0.24        0.09  1.00 -0.03
Year          -0.09        0.06 -0.03  1.00
Sample Size
[1] 100
Probability values (Entries above the diagonal are adjusted for multiple tests.)
            body_kg backpack_kg Units Year
body_kg        0.00        0.18  0.11 0.54
backpack_kg    0.06        0.00  0.54 0.68
Units          0.02        0.35  0.00 0.77
Year           0.36        0.57  0.77 0.00

 To see confidence intervals of the correlations, print with the short=FALSE option
```

Все перечисленные поправки (и еще несколько других) доступны не только в функции `corr.test()`, но и в базовом R с помощью функции `p.adjust()`. Эта функция принимает вектор из *p-value* и возвращает результат применения поправок.

```
p_vec <- seq(0.0001, 0.06, length.out = 10)
p_vec
```

```
[1] 0.000100000 0.006755556 0.013411111 0.020066667 0.026722222 0.033377778
[7] 0.040033333 0.046688889 0.053344444 0.060000000
```

```
p.adjust(p_vec) #                          -
```

```
[1] 0.0010000 0.0608000 0.1072889 0.1404667 0.1603333 0.1668889 0.1668889
[8] 0.1668889 0.1668889 0.1668889
```

```
p.adjust(p_vec, method = "bonferroni")
```

```
[1] 0.00100000 0.06755556 0.13411111 0.20066667 0.26722222 0.33377778
[7] 0.40033333 0.46688889 0.53344444 0.60000000
```

```
p.adjust(p_vec, method = "BH")
```

```
[1] 0.00100000 0.03377778 0.04470370 0.05016667 0.05344444 0.05562963
[7] 0.05719048 0.05836111 0.05927160 0.06000000
```

20.4 Хитмэп корреляций

Как видите, почти все коррелирует друг с другом, даже с учетом поправок. Такие множественные корреляции лучше всего смотреть с помощью хитмап-визуализации.

В качестве примера возьмем встроенный датасет `mtcars`, в котором есть множество количественных переменных.

```
mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Для визуализации возьмем пакет {corrplot}.

```
install.packages("corrplot")
```

Для начала нам нужно построить матрицу корреляций. Уже знакомая нам функция cor() для это вполне подойдет:

```
library(corrplot)
```

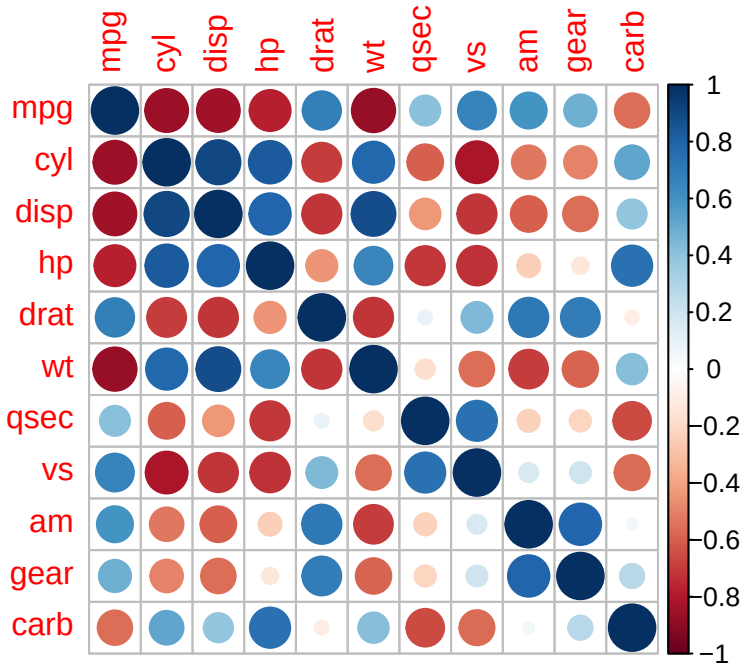
```
corrplot 0.92 loaded
```

```
cor(mtcars)
```

```
            mpg        cyl       disp         hp        drat         wt
mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594
cyl  -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958
disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799
hp   -0.7761684  0.8324475  0.7909486  1.0000000 -0.44875912  0.6587479
drat  0.6811719 -0.6999381 -0.7102139 -0.4487591  1.00000000 -0.7124406
wt   -0.8676594  0.7824958  0.8879799  0.6587479 -0.71244065  1.0000000
qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234  0.09120476 -0.1747159
vs    0.6640389 -0.8108118 -0.7104159 -0.7230967  0.44027846 -0.5549157
am    0.5998324 -0.5226070 -0.5912270 -0.2432043  0.71271113 -0.6924953
gear  0.4802848 -0.4926866 -0.5555692 -0.1257043  0.69961013 -0.5832870
carb -0.5509251  0.5269883  0.3949769  0.7498125 -0.09078980  0.4276059
            qsec         vs          am       gear        carb
mpg   0.41868403  0.6640389  0.59983243  0.4802848 -0.55092507
cyl  -0.59124207 -0.8108118 -0.52260705 -0.4926866  0.52698829
disp -0.43369788 -0.7104159 -0.59122704 -0.5555692  0.39497686
hp   -0.70822339 -0.7230967 -0.24320426 -0.1257043  0.74981247
drat  0.09120476  0.4402785  0.71271113  0.6996101 -0.09078980
wt   -0.17471588 -0.5549157 -0.69249526 -0.5832870  0.42760594
qsec  1.00000000  0.7445354 -0.22986086 -0.2126822 -0.65624923
vs    0.74453544  1.0000000  0.16834512  0.2060233 -0.56960714
am   -0.22986086  0.1683451  1.00000000  0.7940588  0.05753435
gear -0.21268223  0.2060233  0.79405876  1.0000000  0.27407284
carb -0.65624923 -0.5696071  0.05753435  0.2740728  1.00000000
```

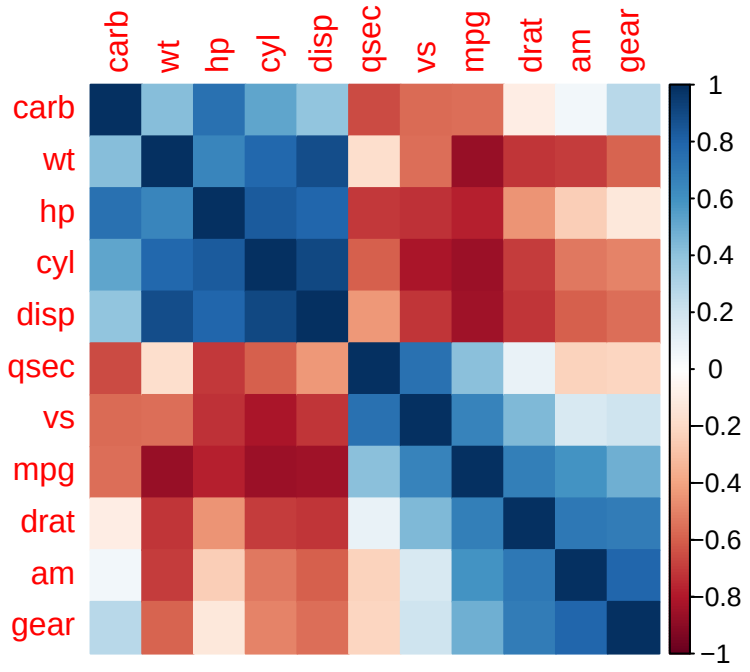
Основная функция пакета `{corrplot}` – функция `corrplot()`. Давайте теперь попробуем ее применить на нашей матрице корреляций.

```
corrplot(cor(mtcars))
```



По умолчанию значение корреляций кодируется цветом и размером круга. У функции `corrplot()` есть множество параметров, позволяющих довольно тонко настраивать хитмэп корреляций. Например, можно кодировать только цветом, а переменные сгруппировать на основе кластеризации.

```
corrplot(cor(mtcars), method = "color", order = "hclust")
```



Можно сделать еще интереснее: добавить крестики, которые будут означать наличие или отсутствие статистической значимости. В этом случае нам понадобится не только матрица корреляций, но соответствующая матрица с *p-value*. Для этого нам нужно обратиться к уже знакомой нам функции `corr.test()` из пакета `{psych}`.

```
mtcars_cor_p <- psych::corr.test(mtcars)
mtcars_cor_p
```

```
Call:psych::corr.test(x = mtcars)
Correlation matrix
       mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
mpg   1.00 -0.85 -0.85 -0.78  0.68 -0.87  0.42  0.66  0.60  0.48 -0.55
cyl  -0.85  1.00  0.90  0.83 -0.70  0.78 -0.59 -0.81 -0.52 -0.49  0.53
disp -0.85  0.90  1.00  0.79 -0.71  0.89 -0.43 -0.71 -0.59 -0.56  0.39
hp   -0.78  0.83  0.79  1.00 -0.45  0.66 -0.71 -0.72 -0.24 -0.13  0.75
drat  0.68 -0.70 -0.71 -0.45  1.00 -0.71  0.09  0.44  0.71  0.70 -0.09
wt   -0.87  0.78  0.89  0.66 -0.71  1.00 -0.17 -0.55 -0.69 -0.58  0.43
qsec  0.42 -0.59 -0.43 -0.71  0.09 -0.17  1.00  0.74 -0.23 -0.21 -0.66
vs    0.66 -0.81 -0.71 -0.72  0.44 -0.55  0.74  1.00  0.17  0.21 -0.57
am    0.60 -0.52 -0.59 -0.24  0.71 -0.69 -0.23  0.17  1.00  0.79  0.06
gear  0.48 -0.49 -0.56 -0.13  0.70 -0.58 -0.21  0.21  0.79  1.00  0.27
```

```
carb -0.55  0.53  0.39  0.75 -0.09  0.43 -0.66 -0.57  0.06  0.27  1.00
Sample Size
[1] 32
Probability values (Entries above the diagonal are adjusted for multiple tests.)
      mpg cyl disp   hp drat   wt qsec   vs   am gear carb
mpg  0.00   0 0.00 0.00 0.00 0.00 0.22 0.00 0.01 0.10 0.02
cyl  0.00   0 0.00 0.00 0.00 0.00 0.01 0.00 0.04 0.08 0.04
disp 0.00   0 0.00 0.00 0.00 0.00 0.20 0.00 0.01 0.02 0.30
hp   0.00   0 0.00 0.00 0.17 0.00 0.00 0.00 1.00 1.00 0.00
drat 0.00   0 0.00 0.01 0.00 0.00 1.00 0.19 0.00 0.00 1.00
wt   0.00   0 0.00 0.00 0.00 0.00 1.00 0.02 0.00 0.01 0.20
qsec 0.02   0 0.01 0.00 0.62 0.34 0.00 0.00 1.00 1.00 0.00
vs   0.00   0 0.00 0.00 0.01 0.00 0.00 0.00 1.00 1.00 0.02
am   0.00   0 0.00 0.18 0.00 0.00 0.21 0.36 0.00 0.00 1.00
gear 0.01   0 0.00 0.49 0.00 0.00 0.24 0.26 0.00 0.00 1.00
carb 0.00   0 0.03 0.00 0.62 0.01 0.00 0.00 0.75 0.13 0.00

 To see confidence intervals of the correlations, print with the short=FALSE option
```

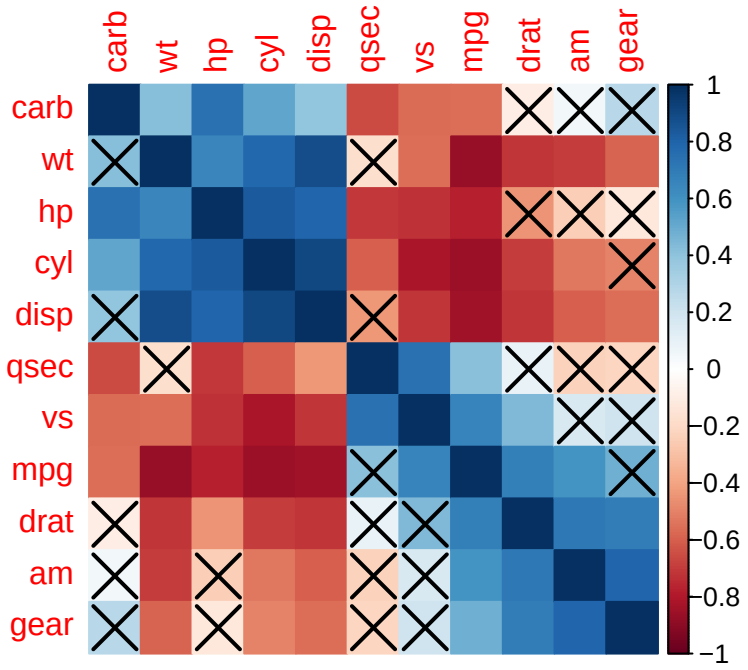
Нам нужно разворошить полученную переменную mtcars_cor_p

```
str(mtcars_cor_p)
```

```
List of 14
 $ r     : num [1:11, 1:11] 1 -0.852 -0.848 -0.776 0.681 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
 $ n     : num 32
 $ t     : num [1:11, 1:11] Inf -8.92 -8.75 -6.74 5.1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
 $ p     : num [1:11, 1:11] 0.00 6.11e-10 9.38e-10 1.79e-07 1.78e-05 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
 $ p.adj : num [1:55] 3.18e-08 4.78e-08 9.92e-11 8.05e-06 1.74e-07 ...
 $ se    : num [1:11, 1:11] 0 0.0955 0.0969 0.1151 0.1337 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
  .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
 $ sef   : num 0.186
 $ adjust: chr "holm"
 $ sym   : logi TRUE
```

```
$ ci    :'data.frame': 55 obs. of  4 variables:
 ..$ lower: num [1:55] -0.926 -0.923 -0.885 0.436 -0.934 ...
 ..$ r    : num [1:55] -0.852 -0.848 -0.776 0.681 -0.868 ...
 ..$ upper: num [1:55] -0.716 -0.708 -0.586 0.832 -0.744 ...
 ..$ p    : num [1:55] 6.11e-10 9.38e-10 1.79e-07 1.78e-05 1.29e-10 ...
$ ci2   :'data.frame': 55 obs. of  5 variables:
 ..$ lower: num [1:55] -0.926 -0.923 -0.885 0.436 -0.934 ...
 ..$ r    : num [1:55] -0.852 -0.848 -0.776 0.681 -0.868 ...
 ..$ upper: num [1:55] -0.716 -0.708 -0.586 0.832 -0.744 ...
 ..$ p    : num [1:55] 6.11e-10 9.38e-10 1.79e-07 1.78e-05 1.29e-10 ...
 ..$ p.adj: num [1:55] 3.18e-08 4.78e-08 8.05e-06 5.86e-04 6.86e-09 ...
$ ci.adj:'data.frame': 55 obs. of  2 variables:
 ..$ lower.adj: num [1:55] -0.954 -0.953 -0.928 0.238 -0.959 ...
 ..$ upper.adj: num [1:55] -0.572 -0.562 -0.405 0.89 -0.61 ...
$ stars : chr [1:11, 1:11] "1***" "-0.85***" "-0.85***" "-0.78***" ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
 .. ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...
$ Call  : language psych::corr.test(x = mtcars)
- attr(*, "class")= chr [1:2] "psych" "corr.test"
```

```
corrplot(corr = mtcars_cor_p$r,
         p.mat = mtcars_cor_p$p,
         method = "color",
         order = "hclust")
```



Крестиками отмечены статистические незначимые (с *p.value* < *.05)*, а без крестиков – статистически значимые коэффициенты корреляции. Обратите внимание, матрица несимметричная: `psych::corr.test()` возвращает скорректированные *p-value* в верхнем правом треугольнике в матрице корреляций.

20.5 Матрица корреляций в виде графа

```
install.packages("corrr")
```

```
library(corrr)
```

Функционал пакета {corrr} позволяет работать с корреляционными матрицами в духе tidyverse, возвращая тиббл вместо матрицы.

```
correlate(mtcars)
```

```
Correlation computed with
* Method: 'pearson'
```

```
* Missing treated using: 'pairwise.complete.obs'
```

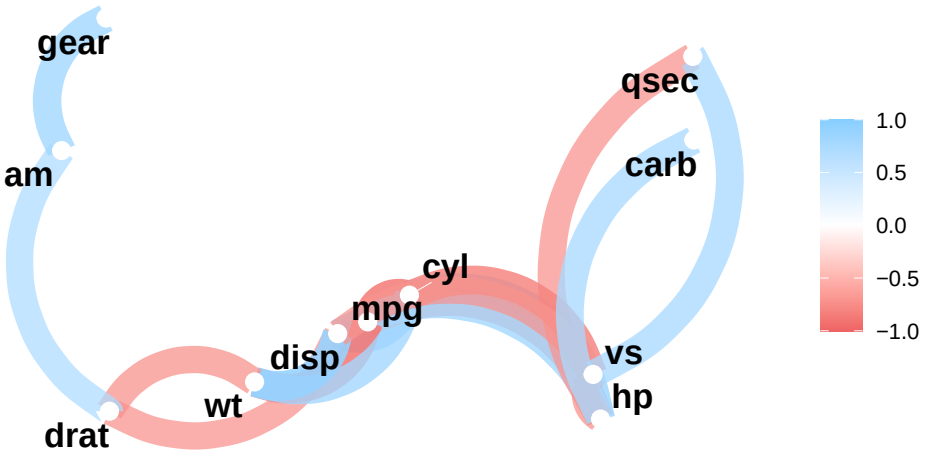
```
# A tibble: 11 x 12
   term     mpg    cyl   disp     hp    drat     wt    qsec     vs      am
   <chr>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>  <dbl>   <dbl>  <dbl>   <dbl>
 1 mpg   NA     -0.852 -0.848 -0.776  0.681  -0.868  0.419   0.664  0.600
 2 cyl   -0.852 NA      0.902  0.832 -0.700   0.782 -0.591  -0.811 -0.523
 3 disp  -0.848  0.902 NA      0.791 -0.710   0.888 -0.434  -0.710 -0.591
 4 hp    -0.776  0.832  0.791 NA     -0.449   0.659 -0.708  -0.723 -0.243
 5 drat   0.681 -0.700 -0.710 -0.449 NA      -0.712  0.0912  0.440  0.713
 6 wt    -0.868  0.782  0.888  0.659 -0.712  NA     -0.175  -0.555 -0.692
 7 qsec   0.419 -0.591 -0.434 -0.708  0.0912 -0.175 NA       0.745 -0.230
 8 vs     0.664 -0.811 -0.710 -0.723  0.440  -0.555  0.745  NA      0.168
 9 am     0.600 -0.523 -0.591 -0.243  0.713  -0.692 -0.230   0.168 NA
10 gear   0.480 -0.493 -0.556 -0.126  0.700  -0.583 -0.213   0.206  0.794
11 carb  -0.551  0.527  0.395  0.750 -0.0908  0.428 -0.656  -0.570  0.0575
# i 2 more variables: gear <dbl>, carb <dbl>
```

Нас же интересует в первую очередь представление матрицы корреляций в виде графа.

Делается это следующим образом: мы устанавливаем минимальное значение для корреляций с помощью параметра `min_cor =`, которые будут отображены в качестве связи между переменными. Если мы этого не сделаем, то мы просто получим полностью связанный граф, потому что все переменные хотя бы немного коррелируют со всеми. По умолчанию этот параметр равен 0.3, но мы можем изменить это значение на собственное усмотрение. В данном случае установим `min_cor = 0.7`, потому что в нашей матрице очень много сильных корреляций.

```
correlate(mtcars) %>%
  corrr::network_plot(min_cor = 0.7)
```

```
Correlation computed with
* Method: 'pearson'
* Missing treated using: 'pairwise.complete.obs'
```


gear
am
drat
wt
disp
mpg
cyl
qsec
carb
vs
hp
1.0
0.5
0.0
−0.5
−1.0

Глава 21

Линейная регрессия

Вы уже умеете считать коэффициент корреляции Пирсона:

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become er
```

```
library(Stat2Data)
data(Backpack)
back <- Backpack %>%
  mutate(backpack_kg = 0.45359237 * BackpackWeight,
         body_kg = 0.45359237 * BodyWeight)
cor.test(back$backpack_kg, back$body_kg)
```

```
    Pearson's product-moment correlation

data:  back$backpack_kg and back$body_kg
t = 1.9088, df = 98, p-value = 0.05921
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.007360697  0.371918344
sample estimates:
      cor
0.1893312
```

Простая линейная регрессия - это примерно то же самое. В синтаксисе линейной регрессии уже не обойтись без формул, это такой специальный тип данных в R:

```
class(y ~ x)
```

```
[1] "formula"
```

Если видите эту волнистую линию - тильду (~), это значит, что перед вами формула. Мы уже сталкивались с формулами ранее, они иногда используются для задания отношений между переменными, например, для определения фасеток на графике (@ref(gg_base)).

Давайте исследуем зависимость размера рюкзака от массы тела. В простой линейной регрессии, в отличие от корреляции, есть направленность: одна переменная является как бы независимой переменной или предиктором, другая - как бы объясняемой переменной (outcome). В формуле предикторы находятся справа от тильды, а объясняемая переменная - слева.

Я специально написал “как бы”: если одна переменная предиктор, а другая объясняется этим предиктором, то кажется, что они должны быть обязательно связаны причинно-следственной связью. Это не так: обозначения условны, более того, вы можете поменять переменные местами и ничего не изменится! Короче говоря, линейная регрессия не дает никакой магической каузальной силы исследуемым переменным.

21.1 Функция `lm()`

Давайте посчитаем линейную регрессию функцией `lm()`.

```
model <- lm(backpack_kg ~ body_kg, data = back)
model
```

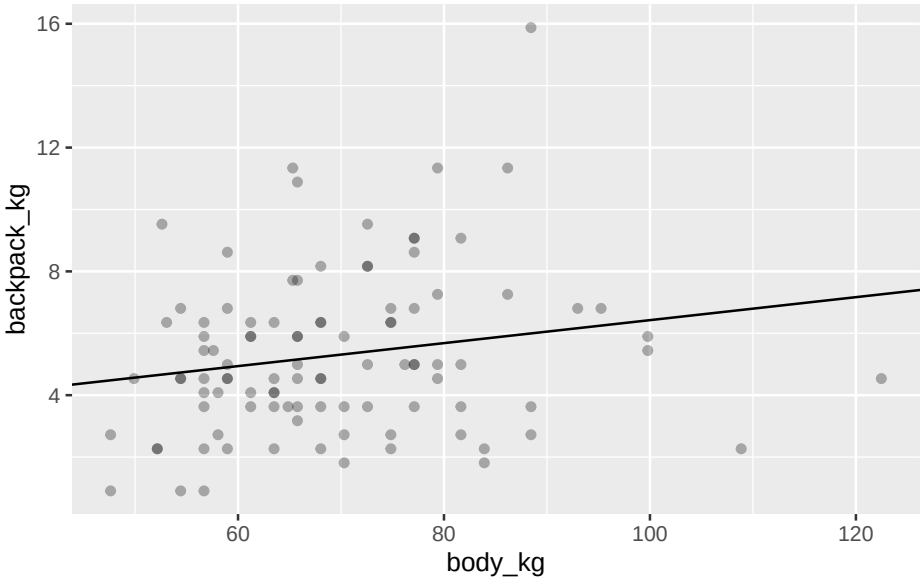
```
Call:
lm(formula = backpack_kg ~ body_kg, data = back)

Coefficients:
(Intercept)      body_kg
    2.71125      0.03713
```

`print(model)` или просто `model` выводит коэфициенты линейной регрессии - это коэффициенты прямой, которая лучше всего подогнанна к данным. Как измеряется качество этой подгонки? В расстоянии точек исходных точек до прямой. По идее, расстояние до прямой нужно было бы считать просто по модулю. И так делают, хоть и очень редко. Обычно в линейной регрессии используются квадратичные расстояния точек до прямой для оценки расстояния (метод наименьших квадратов - ordinary least squares). Это дает кучу клевых математических свойств, например, возможность легко аналитически найти коэффициенты прямой линейной регрессии.

Давайте теперь нарисуем регрессионную прямую поверх диаграммы рассеяния:

```
ggplot(data = back,aes(x = body_kg, y = backpack_kg))+
  geom_point(alpha = 0.3)+
  geom_abline(slope = model$coefficients[2], intercept = model$coefficients[1])
```



Функция `predict()` позволяет скормить модели новые данные и получить предсказания для новых значений предикторов. Попробуем поиграть с этим немного. Допустим, предскажем вес рюкзака для студента весом в 100 кг:

```
predict(model, newdata = data.frame(body_kg = 100))
```

```
       1
6.424229
```

Мы можем даже попробовать какие-нибудь экстремальные значения для предикторов. Например, сколько будет весить рюкзак студента весом 1000 кг?

```
predict(model, newdata = data.frame(body_kg = 1000))
```

```
     1
39.841
```

Очевидно, что в этом не очень много смысла: студент весом 1000 кг не сможет ходить на занятия, поэтому и про вес рюкзака как-то не имеет смысл спрашивать. Это проблема экстраполяции: линейная регрессия позволяет более-менее достоверно предсказывать значения внутри диапазона значений, на которых

была построена модель. Еще один “странный” пример - студент весом 0 кг.

```
predict(model, newdata = data.frame(body_kg = 0))
```

```
       1
2.711255
```

Здесь бессмысленность происходящего еще очевиднее. Конечно, вес студента не может быть равен нулю, иначе это не студент вовсе. Однако это позволяет понять, что такое intercept модели - это значение зависимой переменой в случае, если предиктор равен нулю. А коэффициент предиктора означает, насколько килограммов увеличивается вес рюкзака при увеличении веса студента на 1 кг: на 0.0371297. Не очень много!

21.2 Интерпретация вывода линейной регрессии

Гораздо более подробные результаты мы получим, если применим уже известную нам generic функцию `summary()` на нашу модель.

```
summary(model)
```

```
Call:
lm(formula = backpack_kg ~ body_kg, data = back)

Residuals:
    Min      1Q  Median      3Q     Max
-4.4853 -1.7629 -0.4681  1.2893  9.8803

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.71125    1.37483   1.972   0.0514 .
body_kg      0.03713    0.01945   1.909   0.0592 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.581 on 98 degrees of freedom
Multiple R-squared:  0.03585,   Adjusted R-squared:  0.02601
F-statistic: 3.644 on 1 and 98 DF,  p-value: 0.05921
```

Теперь мы понимаем, что это за коэффициенты. Однако это всего лишь их оценка. Это значит, что мы допускаем, что в реальности есть некие настоящие коэффициенты линейной регрессии, а каждый раз собирая новые данные, они будут посчитаны как немного разные. Короче говоря, эти коэффициенты - те же статистики, со своим выборочным распределением и стандартными ошибками. На основе чего и высчитывается p-value для каждого коэффициента - вероятность получить такой и более отклоняющийся от нуля коэффициент при верности нулевой гипотезы - независимости зависимой переменной от предиктора.

Кроме p-value, у линейной регрессии есть R^2 - доля объясненной дисперсии. Как ее посчитать? Для начала давайте сохраним как отдельные колонки ошибки (необъясненную часть модели) и предсказанные значения (они означают объясненную часть модели). Можно убедиться, что сумма предсказанных значений и ошибок будет равна зависимой переменной.

```
head(model$residuals)
```

```
        1         2         3         4         5         6 
-0.7341441 -2.3666602 -0.1963429 -2.6001743 -2.1140337 -4.4853169 
```

```
back$residuals <- residuals(model)
back$fitted <-  fitted(model)

back %>%
  transmute(backpack_kg - (fitted + residuals))
```

```
    backpack_kg - (fitted + residuals)
1                         0.000000e+00
2                         0.000000e+00
3                         0.000000e+00
4                        -4.440892e-16
5                         4.440892e-16
6                        -4.440892e-16
7                         0.000000e+00
8                         2.220446e-16
9                         4.440892e-16
10                        1.110223e-16
11                       -4.440892e-16
12                        0.000000e+00
13                        0.000000e+00
14                        0.000000e+00
```

15	-8.881784e-16
16	0.000000e+00
17	0.000000e+00
18	4.440892e-16
19	0.000000e+00
20	-4.440892e-16
21	0.000000e+00
22	0.000000e+00
23	0.000000e+00
24	0.000000e+00
25	-8.881784e-16
26	0.000000e+00
27	4.440892e-16
28	4.440892e-16
29	0.000000e+00
30	-4.440892e-16
31	0.000000e+00
32	0.000000e+00
33	0.000000e+00
34	-4.440892e-16
35	0.000000e+00
36	0.000000e+00
37	0.000000e+00
38	0.000000e+00
39	-2.220446e-16
40	0.000000e+00
41	-4.440892e-16
42	0.000000e+00
43	0.000000e+00
44	0.000000e+00
45	0.000000e+00
46	-4.440892e-16
47	0.000000e+00
48	4.440892e-16
49	0.000000e+00
50	8.881784e-16
51	0.000000e+00
52	-4.440892e-16
53	0.000000e+00
54	0.000000e+00
55	0.000000e+00
56	0.000000e+00
57	0.000000e+00
58	0.000000e+00
59	0.000000e+00
60	0.000000e+00

```
61                0.000000e+00
62                8.881784e-16
63                0.000000e+00
64               -4.440892e-16
65                0.000000e+00
66                0.000000e+00
67                0.000000e+00
68                8.881784e-16
69               -3.330669e-16
70               -4.440892e-16
71                4.440892e-16
72                0.000000e+00
73                8.881784e-16
74               -8.881784e-16
75                4.440892e-16
76                0.000000e+00
77                1.110223e-16
78               -4.440892e-16
79                0.000000e+00
80                0.000000e+00
81                0.000000e+00
82                0.000000e+00
83               -4.440892e-16
84                0.000000e+00
85                0.000000e+00
86                0.000000e+00
87                0.000000e+00
88               -8.881784e-16
89                0.000000e+00
90                0.000000e+00
91                0.000000e+00
92                0.000000e+00
93                0.000000e+00
94                0.000000e+00
95                0.000000e+00
96                0.000000e+00
97                0.000000e+00
98                0.000000e+00
99                0.000000e+00
100               0.000000e+00
```

Соответственно, вся сумма объясненной дисперсии разделяется на объясненую и необъясненную. Полная дисперсия (total sum of squares = TSS) может быть посчитана как сумма квадратов разниц со средним. Необъясненная дисперсия - это сумма квадратов ошибок - residual sum of squares (RSS).

```
rss <- sum(back$residuals^2)
rss
```

```
[1] 652.7272
```

```
tss <- sum((back$backpack_kg - mean(back$backpack_kg))^2)
tss
```

```
[1] 676.995
```

```
1- rss/tss
```

```
[1] 0.03584628
```

Это очень мало, мы объяснили всего 3.5846285% дисперсии. Собственно, и *p-value* больше, чем 0.05.

```
summary(model)
```

```
Call:
lm(formula = backpack_kg ~ body_kg, data = back)

Residuals:
    Min      1Q  Median      3Q     Max
-4.4853 -1.7629 -0.4681  1.2893  9.8803

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.71125    1.37483   1.972   0.0514 .
body_kg      0.03713    0.01945   1.909   0.0592 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.581 on 98 degrees of freedom
Multiple R-squared:  0.03585,   Adjusted R-squared:  0.02601
F-statistic: 3.644 on 1 and 98 DF,  p-value: 0.05921
```

При этом *p-value* тот же, что и при коэффициенте корреляции Пирсона. Это не случайно: R^2 - это квадрат коэффициента корреляции Пирсона, если речь идет только об одном предикторе. Давайте это проверим:

```
cor.test(back$body_kg, back$backpack_kg)$estimate^2
```

```
       cor
0.03584628
```

21.3 Допущения линейной регрессии

Как и в случае с другими параметрическими методами, линейная регрессия имеет определенные допущения относительно используемых данных. Если они не соблюдаются, то все наши расчеты уровня значимости могут некорректными.

> Очень важно ставить вопрос о том, насколько результаты будут некорректными. Как сильно нарушения допущений будет влиять на модель? Ответ на этот вопрос может быть контринтуитивен. Например, достаточно большие отклонения от нормальности нам обычно не страшны при условии того, что выборка достаточно большая.

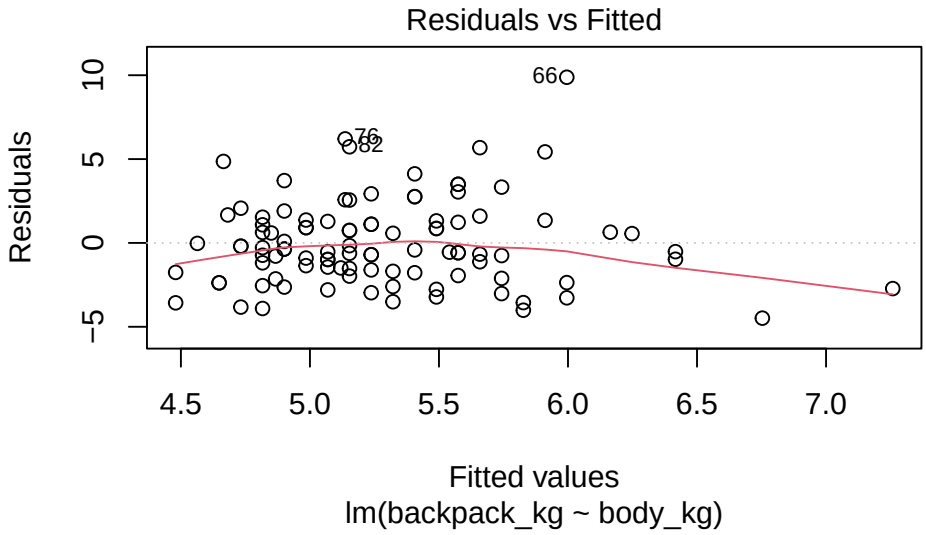
Допущения линейной регрессии связаны с ошибками: они должны быть нормально распределены, а разброс ошибок должен не уменьшаться и не увеличиваться в зависимости от предсказанных значений. Это то, что называется гомоскедастичностью или гомогенностью (когда все хорошо) и гетероскедастичностью или гетерогенностью (когда все плохо).

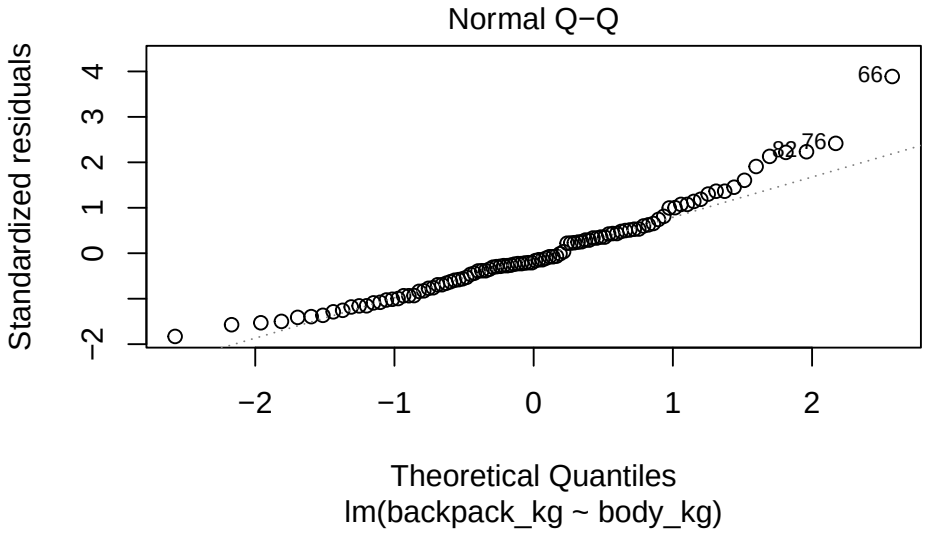
Если мы применим функцию `plot()`, то получим 4 скаттерплота:

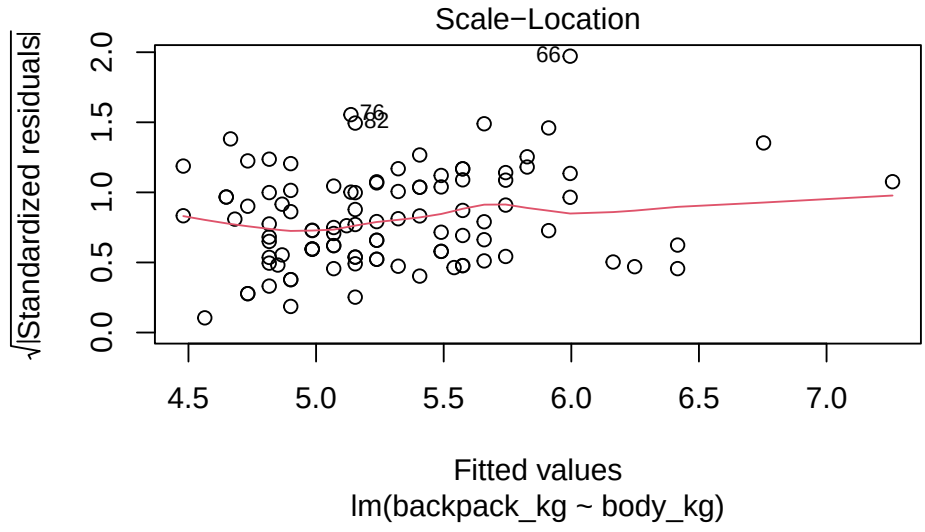
1. *Зависимость ошибок от предсказанных значений.* На что здесь смотреть? На симметричность относительно нижней и верхней части графика, на то, что разброс примерно одинаковый слева и справа.
2. *Q-Q plot.* Здесь все довольно просто: если ошибки являются выборкой из нормального распределения, то они выстраиваются в прямую линию. Если это мало похоже на прямую линию, то имеет место отклонение от нормальности.
3. *Scale-Location plot.* Этот график очень похож на график 1, только по оси *y* используются квадратные корни модуля ошибки. Еще один способ исследовать гетеро(гомо)скедастичность и находить выбросы.

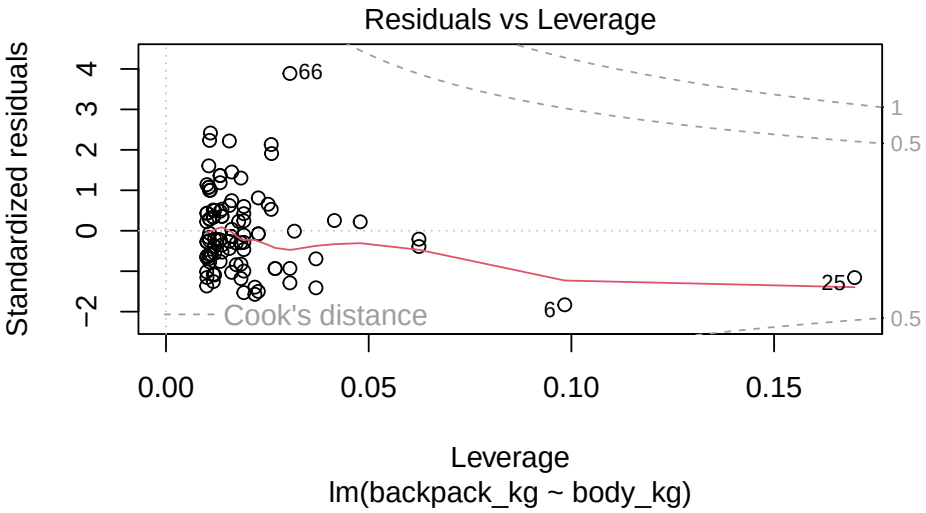
4. *Residuals-Leverage plot*. Здесь по оси *x* - расстояние Кука, а по оси *y* - стандартизированный размер выбросов. Расстояние Кука показывает *high-leverage points* - точки, которые имеют экстремальные предсказанные значения, то есть очень большие или очень маленькие значения по предикторам. Для линейной регрессии такие значения имеют большее значение, чем экстремальные точки по предсказываемой переменной. Особенно сильное влияние имеют точки, которые имеют экстремальные значения и по предикторам, и по предсказываемой переменной. Одна такая точка может поменять направление регрессионной прямой! Расстояние Кука отражает уровень *leverage*, а стандартизированные ошибки отражают экстремальные значения по *y* (вернее, экстремальные отклонения от предсказанных значений). В этом графике нужно смотреть на точки с правой стороны графика, особенно если они находятся высоко или низко по оси *y*.

```
plot(model)
```




Normal Q−Q
Standardized residuals
66
82
76
−2
−1
0
1
2
3
4
−2
−1
0
1
2
Theoretical Quantiles
lm(backpack_kg ~ body_kg)


Scale−Location
√|Standardized residuals|
66
76
82
0.0
0.5
1.0
1.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
Fitted values
lm(backpack_kg ~ body_kg)



21.4 Влияние выбросов на линейную модель

Давайте теперь попробуем посмотреть, как изменится модель, если выкинуть ***high leverage points* (экстремальные значения по предиктору** - body) и что будет, если выкинуть экстремальные значения по y. Обычная линия - регрессионная прямая для модели со всеми точками, штрихованная линия - регрессионная прямая для модели без экстремальных значений по предиктору, пунктирная линия - регрессионная прямая для модели без экстремальных значений по предсказываемой переменной.

```
is_outlier <- function(x, n = 2, centr = mean, vary = sd) {
  (x > centr(x) + n * vary(x)) | (x < centr(x) - n * vary(x))
}

back <- back %>%
  mutate(body_outlier = is_outlier(body_kg),
         backpack_outlier = is_outlier(backpack_kg))

model_without_outliers_by_x <- back %>%
  filter(!body_outlier) %>%
  lm(formula = backpack_kg ~ body_kg)
summary(model_without_outliers_by_x)
```

```
Call:
lm(formula = backpack_kg ~ body_kg, data = .)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-4.6526 -1.7471 -0.3773  1.1699  9.0854

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.48577    1.65749   0.293  0.77011
body_kg      0.07128    0.02413   2.953  0.00397 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.548 on 94 degrees of freedom
Multiple R-squared:  0.08491,   Adjusted R-squared:  0.07517
F-statistic: 8.722 on 1 and 94 DF,  p-value: 0.003971
```

```
model_without_outliers_by_y <- back %>%
  filter(!backpack_outlier) %>%
  lm(formula = backpack_kg ~ body_kg)
summary(model_without_outliers_by_y)
```

```
Call:
lm(formula = backpack_kg ~ body_kg, data = .)

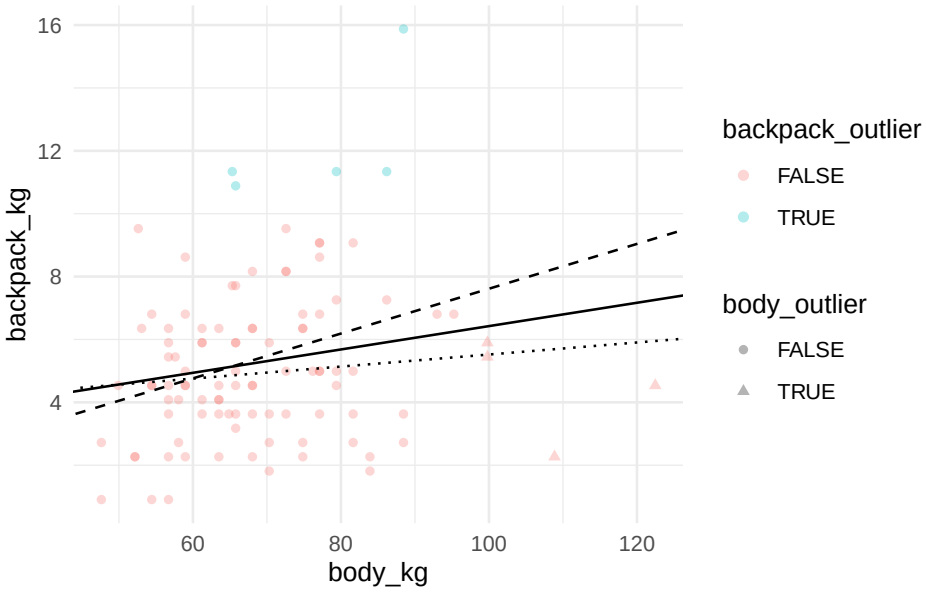
Residuals:
    Min      1Q  Median      3Q     Max
-3.7843 -1.4343 -0.1363  1.4296  4.9122

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.60560    1.13144   3.187  0.00196 **
body_kg      0.01915    0.01610   1.190  0.23716
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.087 on 93 degrees of freedom
Multiple R-squared:  0.01499,   Adjusted R-squared:  0.004402
F-statistic: 1.416 on 1 and 93 DF,  p-value: 0.2372
```

```
ggplot(data = back, aes(x = body_kg, y = backpack_kg,
                        shape = body_outlier, colour = backpack_outlier))+
  geom_point(alpha = 0.3)+
  geom_abline(intercept = model$coefficients[1], slope = model$coefficients[2])+
```

```
  geom_abline(intercept= model_without_outliers_by_x$coefficients[1],
              slope = model_without_outliers_by_x$coefficients[2], linetype = "dashed")+
  geom_abline(intercept= model_without_outliers_by_y$coefficients[1],
              slope = model_without_outliers_by_y$coefficients[2], linetype = "dotted")+
  theme_minimal()
```



Таким образом, именно экстремальные значения по предиктору, а не по объясняемой переменной имеют особенно сильное значение на регрессионную модель.

21.5 Множественная линейная регрессия

В множественной линейной регрессионной регрессии у нас появляется несколько предикторов. Какая модель лучше: где есть много предикторов или где мало предикторов? С одной стороны, чем больше предикторов, тем лучше: каждый новый предиктор может объяснить чуть больше необъясненной дисперсиии. С другой стороны, если эта прибавка маленькая (а она всегда будет не меньше нуля), то, возможно, новый предиктор просто объясняет “случайный шум”. В действительности, если у нас будет достаточно много предикторов, то мы сможем объяснить любые данные! Парадоксальным образом такая модель будет

давать очень хорошие результаты на той выборке, по которой мы считаем коэффициенты, но делать очень плохие предсказания на новой выборке - это то, что в машинном обучении называют **переобучением (*overfitting*).** Идеальная модель будет включать минимум предикторов, которые лучше всего объясненяют исследуемую переменную. Это что-то вроде бритвы Оккама в статистике.

Поэтому часто используются показатели качества модели, которые "наказывают" модель за большое количество предикторов. Например, adjusted R^2:

$$R_{adj} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

Здесь *n* - это количество наблюдений, *p* - количество параметров.

Итак, добавим новый предиктор - *Units*. Это количество кредитов, которые студенты взяли в четверти[1]. Можно предположить, что чем больше у студента набрано кредитов, тем более тяжелый у нее/него рюкзак. Давайте добавим это как второй предиктор. Для этого нужно просто записать второй предиктор в формуле через плюс.

```
model_mult <- lm(backpack_kg ~ body_kg + Units, data = back)
summary(model_mult)
```

```
Call:
lm(formula = backpack_kg ~ body_kg + Units, data = back)

Residuals:
    Min      1Q  Median      3Q     Max
-4.6221 -1.8347 -0.5023  1.2519 10.0623

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.28481    2.16170   0.132   0.8955
body_kg      0.04391    0.01990   2.207   0.0297 *
Units        0.13703    0.09456   1.449   0.1505
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

[1]Кредиты - это что-то вроде часов во многих зарубежных системах образования. Более "тяжелые" курсы дают больше кредитов. Студентам необходимо набрать определенное количество кредитов, чтобы закончить вуз.

```
Residual standard error: 2.566 on 97 degrees of freedom
Multiple R-squared:  0.05628,   Adjusted R-squared:  0.03682
F-statistic: 2.892 on 2 and 97 DF,  p-value: 0.06025
```

Множественная линейная регрессия имеет еще одно допущение: отсутствие мультиколлинеарности. Это значит, что предикторы не должны коррелировать друг с другом.

Для измерения мультколлинеарности существует *variance inflation factor* (*VIF-фактор*). Считается он просто: для предиктора i считается линейная регрессия, где все остальные предикторы предсказывают предиктор i.

Сам VIF-фактор считается на основе полученного R^2 регрессии:

$$VIF_i = \frac{1}{1 - R_i^2}$$

Если R_i^2 большой, то и VIF_i выходит большим. Это означает, что предиктор сам по себе хорошо объясняется другими предикторами. Какой VIF считать большим? Здесь нет единого мнения, но если он выше 3 и особенно если он выше 10, то с этим нужно что-то делать.

```
car::vif(model_mult)
```

```
body_kg   Units
1.05858 1.05858
```

В нашем случае это не так. Но если бы VIF был большим для какого-либо предиктора, то можно было бы либо попробовать его выкинуть или же использовать анализ главных компонент (см. Глава 24.2), о котором пойдет речь в один из следующих дней.

Глава 22

Дисперсионный анализ (ANOVA)

Дисперсионный анализ или ANOVA[1] - один из самых распространенных методов статистического анализа в психологии, биологии, медицине и многих других дисциплинах. Дисперсионный анализ очень хорошо подходит для анализа данных, полученных в эксперименте - методе организации исследования, при котором исследователь напрямую управляет уровнями независимой переменной. Терминологическая связь между дисперсионным анализом и планированием эксперимента настолько тесная, что многие термины пересекаются, поэтому нужно быть осторожными. Как и в случае с линейной регрессией, если мы что-то называем “независимой переменной” (или “фактором”), это не порождает никакой каузальной связи.

> Еще одна важная вещь, которую нужно понимать про дисперсионный анализ, это то, что у этого метода очень запутывающее название: из названия кажется, что этот статистический метод для сравнения дисперсий. Нет, это не так (хотя такие статистические тесты тоже есть, и они нам сегодня пригодятся - см. @ref(aova)). Нет, это просто сравнение средних в случае, если есть больше, чем 2 группы для сравнения.

У дисперсионного анализа очень много разновидностей, для которых придумали множество названий. “Обычная” ANOVA

[1] ANOVA от ANalysis Of VAriance, по-русски часто читается как “АНОВА”.

называется One-Way ANOVA, она же межгрупповая ANOVA, это аналог независимого т-теста для нескольких групп.

Давайте начнем сразу с проведения теста. Мы будем использовать данные с курса по статистике Университета Шеффилда про эффективность диет, на которых мы разбирались с t-тестом (@ref(dep_ttest)).

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

```
diet <- read_csv("data/stcp-Rdataset-Diet.csv")
```

```
Rows: 78 Columns: 7
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (7): Person, gender, Age, Height, pre.weight, Diet, weight6weeks

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Сделаем небольшой препроцессинг данных. Создадим дополнительные “факторные” переменные, создадим переменную, в которой будет разница массы “до” и “после”, удалим `NA`.

```
diet <- diet %>%
  mutate(weight.loss = weight6weeks - pre.weight,
         Dietf = factor(Diet, labels = LETTERS[1:3]),
```

```
        Person = factor(Person)) %>%
  drop_na()
```

22.0.1 Функция aov()

Попробуем сразу провести дисперсионных анализ с помощью функции `aov()`:

```
aov_model <- aov(weight.loss ~ Dietf, diet)
aov_model
```

```
Call:
   aov(formula = weight.loss ~ Dietf, data = diet)

Terms:
                   Dietf Residuals
Sum of Squares   60.5270  410.4018
Deg. of Freedom       2        73

Residual standard error: 2.371064
Estimated effects may be unbalanced
```

```
summary(aov_model)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
Dietf        2   60.5  30.264   5.383 0.0066 **
Residuals   73  410.4   5.622
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Мы получили что-то похожее на результат применения функции `lm()`. Правда, лаконичнее, но с новыми столбцами `Sum Sq`, `Mean Sq` и новой статистикой *F* вместо *t*. Что будет, если с теми же данными с той же формулой запустить `lm()` вместо `aov()`?

```
summary(lm(weight.loss ~ Dietf, diet))
```

```
Call:
lm(formula = weight.loss ~ Dietf, data = diet)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-5.7000 -1.6519 -0.1759  1.4420  5.3680

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.3000     0.4840  -6.818 2.26e-09 ***
DietfB        0.0320     0.6776   0.047  0.96246
DietfC       -1.8481     0.6652  -2.778  0.00694 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.371 on 73 degrees of freedom
Multiple R-squared:  0.1285,    Adjusted R-squared:  0.1047
F-statistic: 5.383 on 2 and 73 DF,  p-value: 0.006596
```

`lm()` превратил `Dietf` в две переменные, но *F* и p-value у двух моделей одинаковые! Кроме того, функция `aov()` является, по сути, просто "оберткой" над `lm()`:

> This provides a wrapper to lm for fitting linear models to balanced or unbalanced experimental designs.

22.1 Тестирование значимости нулевой гипотезы в ANOVA.

Как и в случае с другими статистическими тестами, мы можем выделить 4 этапа в тестировании значимости нулевой гипотезы в ANOVA:

1. **Формулирование нулевой и альтернативной гипотезы.** Нулевая гипотеза говорит, что между средними в генеральной совокупности нет различий:

$$H_0 : \mu_1 = \mu_2 = ... = \mu_n$$

Можно было бы предположить, что ненулевая гипотеза звучит как "все средние не равны", но вообще-то это не так. Альтернативная гипотеза в дисперсионном анализе звучит так:

$$H_1 : \text{Не все средние равны}$$

2. **Подсчет статистики.** Как мы уже видели раньше, в дисперсионном анализе используется новая для нас статистика *F*. Впрочем, мы ее видели, когда смотрели на аутпут функции `lm()`, когда делали линейную регрессию. Чтобы считать *F* (если вдруг мы хотим сделать это вручную), нужно построить талбицу ANOVA (ANOVA table).

Таблица ANOVA	Степени свободы	Суммы квадратов	Средние квадраты	F-статистика
Межгрупповые	df_b	SS_b	$MS_b = \frac{SS_b}{df_b}$	$F = \frac{MS_b}{MS_w}$
Внутригрупповые	df_w	SS_w	$MS_w = \frac{SS_w}{df_w}$	
Общие	df_t	$SS_t = SS_b + SS_w$		

Именно эту таблицу мы видели, когда использовали функцию `aov()`:

```
summary(aov_model)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
Dietf        2   60.5  30.264   5.383 0.0066 **
Residuals   73  410.4   5.622
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Вот как это все считается:

Таблица ANOVA	Степени свободы	Суммы квадратов	Средние квадраты	F-статистика
Между	$df_b = J - 1$	$SS_b = \sum_{j=1}^{J} \sum_{i=1}^{n_j} (\overline{x_j} - \overline{x})^2$	$MS_b = \frac{SS_b}{df_b}$	$F = \frac{MS_b}{MS_w}$
Внутри	$df_w = N - J$	$SS_w = \sum_{j=1}^{J} \sum_{i=1}^{n_j} (x_{ij} - \overline{x_j})^2$	$MS_w = \frac{SS_w}{df_w}$	
Общие	$df_t = N - 1$	$SS_t = \sum_{j=1}^{J} \sum_{i=1}^{n_j} (x_{ij} - \overline{x})^2$		

J означает количество групп, N - общее количество наблюдений во всех группах, n_j означает количество наблюдений в группе j, а x_{ij} - наблюдение под номером i в группе j.

Вариабельность обозначается SS и означает "сумму квадратов" (sum of squares) - это то же, что и дисперсия, только мы не делим вме в конце на количество наблюдений (или количество наблюдений минус один):

$$SS = \sum_{i=1}^{n_j}(x_i - \overline{x})^2$$

Здесь много формул, но суть довольно простая: мы разделяем вариабельность зависимой переменной на внутригрупповую и межгрупповую, считаем их соотношение, которое и будет *F*. В среднем, *F* будет равен 1 при верности нулевой гипотезы. Это означает, что и межгрупповая вариабельность, и внутригрупповая вариабельность - это просто шум. Но если же межгрупповая вариабельность - это не просто шум, то это соотношение будет сильно больше единицы.

3. **Подсчет p-value.** В t-тесте мы смотрели, как статистика распределена при условии верности нулевой гипотезы. То есть что будет, если нулевая гипотеза верна, мы будем повторять эксперимент с точно таким же дизайном (и размером выборок) бесконечное количество раз и считать *F*.

```
betweendf <- 2
withindf <- 73
f <- summary(aov_model)[[1]]$F[1]

v <- seq(0.1,10, 0.01)
fdist <- data.frame(fvalues = v, pdf = df(v, betweendf, withindf))

library(ggplot2)

label <- paste0("F(", betweendf, ", ", withindf, ") = ", round(f, 3))

ggplot(fdist, aes(x = fvalues, y = pdf))+
  geom_line()+
  geom_vline(xintercept = f)+
  annotate("text", x = f+1, y = 0.2, label = label)+
  scale_y_continuous(expand=c(0,0)) +
```

```
  theme_minimal()+
theme(axis.line.y = element_blank(),
      axis.ticks.y = element_blank(),
      axis.text.y = element_blank(),
      axis.title.y = element_blank())
```

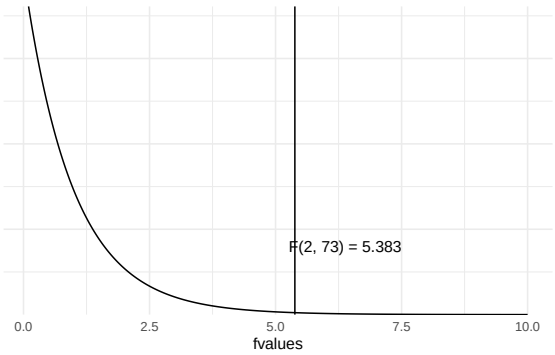


Рисунок 22.1: F-распределение при верности нулевой гипотезы (см. детали в тексте)

Заметьте, распределение *F* несимметричное[2]. Это значит, что мы всегда считаем считаем площадь от *F* до плюс бесконечности (без умножения на 2, как мы это делали в t-тесте):

```
1 - pf(f, betweendf, withindf)
```

```
[1] 0.006595853
```

Это и есть наш p-value!

4. **Сравнение p-value с уровнем** α. Самый простой этап: если наш p-value меньше, чем α (который обычно равен 0.05), то мы отвергаем нулевую гипотезу. Если нет - не отвергаем.

В нашем случае это 0.0065959, что, очевидно, меньше, чем 0.05. Отвергаем нулевую гипотезу (о том, что нет различий), принимаем ненулевую (о том, что различия есть). Все!

[2] Форма *F*-распределения будет сильно зависеть от числа степеней свободы. Но оно всегда определено от 0 до плюс бесконечности: в числителе и знаменателе всегда неотрицательные числа.

22.2 Post-hoc тесты

Тем не менее, дисперсионного анализа недостаточно, чтобы решить, какие именно группы между собой различаются. Для этого нужно проводить post-hoc тесты (апостериорные тесты).

Post-hoc переводится с латыни как “после этого”. Post-hoc тесты или просто “пост-хоки” проводятся, если в результате ANOVA была отвергнута нулевая гипотеза. Собственно, пост-хоки никак не связаны с дисперсионным анализом на уровне расчетов - это абсолютно независимые тесты, но исторически так сложилось, что они известны именно как дополнительный этап ANOVA.

Самый простой вариант пост-хок теста - это попарные т-тесты [3] с поправками на множественные сравнения:

```
pairwise.t.test(diet$weight.loss, diet$Dietf)
```

```
    Pairwise comparisons using t tests with pooled SD

data:  diet$weight.loss and diet$Dietf

  A     B
B 0.962 -
C 0.017 0.017

P value adjustment method: holm
```

Второй подход связан с использованием специализированных тестов, таких как тест Тьюки (Tukey Honest Significant Differences = Tukey HSD). Для этого в R есть функция `TukeyHSD()`, которую нужно применять на объект `aov`:

```
TukeyHSD(aov_model)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = weight.loss ~ Dietf, data = diet)
```

[3] По умолчанию функция `pairwise.t.test()` использует объединенное стандартное отклонение для всех условий при расчетах. Это дает немного другие результаты, чем просто попарные t-тесты, хотя разница обычно незначительная. Чтобы отключить такое поведение функции `pairwise.t.test()`, можно поставить `FALSE` для параметра `pool.sd()`

```
$Dietf
         diff       lwr        upr     p adj
B-A  0.032000 -1.589085  1.6530850 0.9987711
C-A -1.848148 -3.439554 -0.2567422 0.0188047
C-B -1.880148 -3.454614 -0.3056826 0.0152020
```

22.3 ANOVA и t-тест как частные случаи линейной регрессии

Как мы уже видели, если применить `lm()` или `aov()` на одних и тех же данных с одной и той же формулой, то результат будет очень похожим. Но есть одно но: `lm()` создает из одного фактора две переменных-предиктора:

```
summary(lm(weight.loss ~ Dietf, diet))
```

```
Call:
lm(formula = weight.loss ~ Dietf, data = diet)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7000 -1.6519 -0.1759  1.4420  5.3680

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.3000     0.4840  -6.818 2.26e-09 ***
DietfB        0.0320     0.6776   0.047  0.96246
DietfC       -1.8481     0.6652  -2.778  0.00694 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.371 on 73 degrees of freedom
Multiple R-squared:  0.1285,    Adjusted R-squared:  0.1047
F-statistic: 5.383 on 2 and 73 DF,  p-value: 0.006596
```

Дело в том, что мы не можем просто так загнать номинативную переменную в качестве предиктора в линейную регрессию. Мы можем это легко сделать, если у нас всего два уровня в номинативном предикторе. Тогда один из уровней можно обозначить за 0, другой - за 1. Такие переменные иногда

называются “бинарными”. Тогда это легко использовать в линейной регрессии:

```
summary(lm(weight.loss ~ gender, diet))
```

```
Call:
lm(formula = weight.loss ~ gender, data = diet)

Residuals:
    Min      1Q  Median      3Q     Max
-5.1848 -1.7264  0.2041  1.6846  5.9930

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.8930     0.3846 -10.123  1.3e-15 ***
gender       -0.1221     0.5836  -0.209    0.835
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.522 on 74 degrees of freedom
Multiple R-squared:  0.0005914, Adjusted R-squared:  -0.01291
F-statistic: 0.04379 on 1 and 74 DF,  p-value: 0.8348
```

Можно ли так делать? Вполне! Допущения линейной регрессии касаются остатков, а не переменных самих по себе. Разве что это немного избыточно: линейная регрессия с бинарным предиктором - это фактически независимый t-тест:

```
t.test(weight.loss ~ gender, diet, var.equal = TRUE)
```

```
    Two Sample t-test

data:  weight.loss by gender
t = 0.20925, df = 74, p-value = 0.8348
alternative hypothesis: true difference in means between group 0 and group 1 is not equ
95 percent confidence interval:
 -1.040810  1.285067
sample estimates:
mean in group 0 mean in group 1
      -3.893023       -4.015152
```

Как видите, p-value совпадают! А t статистика в квадрате - это F (при двух группах):

```
t.test(weight.loss ~ gender, diet, var.equal = TRUE)$statistic^2
```

```
        t
0.04378592
```

Более того, те же самые результаты можно получить и с помощью коэффициента корреляции Пирсона:

```
cor.test(diet$gender, diet$weight.loss)
```

```
    Pearson's product-moment correlation

data:  diet$gender and diet$weight.loss
t = -0.20925, df = 74, p-value = 0.8348
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2484113  0.2022466
sample estimates:
        cor
-0.02431772
```

Теперь должно быть понятно, почему все эти функции делают вроде бы разные статистические тесты, но выдают такой похожий результат - это фактически один и тот же метод! Все эти методы (и некоторые из тех, что будем рассматривать далее) можно рассматривать как разновидности множественной линейной регрессии. [4]

22.4 Dummy coding

Тем не менее, вопрос остается открытым: как превратить номинативную переменную в количественную и загнать ее в регрессию? Для этого можно использовать **“фиктивное кодирование” (dummy coding):**

[4] Обобщением множественной линейной регрессии (вернее, одним из) можно считать общую линейную модель (general linear model). Общая линейная модель может предсказывать не одну, а сразу несколько объясняемых переменных в отличие от множественной линейной регрессии. Следующим этапом обобщения служит обобщенная линейная модель (generalized linear model). Фишка последней в том, что можно использовать не только модели с нормально распределенными остатками, но и, например, логистическую и пуассоновскую регрессию.

```
diet <- diet %>%
  mutate(isA = as.numeric(Dietf == "A"),
         isB = as.numeric(Dietf == "B"),
         isC = as.numeric(Dietf == "C"))

diet %>%
  group_by(Dietf) %>%
  slice(1:2) %>%
  select(Dietf, isA:isC)
```

```
# A tibble: 6 x 4
# Groups:   Dietf [3]
  Dietf   isA   isB   isC
  <fct> <dbl> <dbl> <dbl>
1 A         1     0     0
2 A         1     0     0
3 B         0     1     0
4 B         0     1     0
5 C         0     0     1
6 C         0     0     1
```

Заметьте, что такое кодирование избыточно. Если мы знаем, что диет 3, а данная диета - это не диета B и не диета C, то это диета A. Значит, одна из созданных нами колонок - “лишняя”:

```
diet$isA <- NULL
```

Используем новую колонки для линейной регрессии и сравним результаты:

```
summary(lm(weight.loss ~ isB + isC, diet))
```

```
Call:
lm(formula = weight.loss ~ isB + isC, data = diet)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7000 -1.6519 -0.1759  1.4420  5.3680

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.3000     0.4840  -6.818 2.26e-09 ***
isB           0.0320     0.6776   0.047  0.96246
```

```
isC          -1.8481     0.6652  -2.778  0.00694 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.371 on 73 degrees of freedom
Multiple R-squared:  0.1285,    Adjusted R-squared:  0.1047
F-statistic: 5.383 on 2 and 73 DF,  p-value: 0.006596
```

```
summary(lm(weight.loss ~ Dietf, diet))
```

```

Call:
lm(formula = weight.loss ~ Dietf, data = diet)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7000 -1.6519 -0.1759  1.4420  5.3680

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.3000     0.4840  -6.818 2.26e-09 ***
DietfB        0.0320     0.6776   0.047  0.96246
DietfC       -1.8481     0.6652  -2.778  0.00694 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

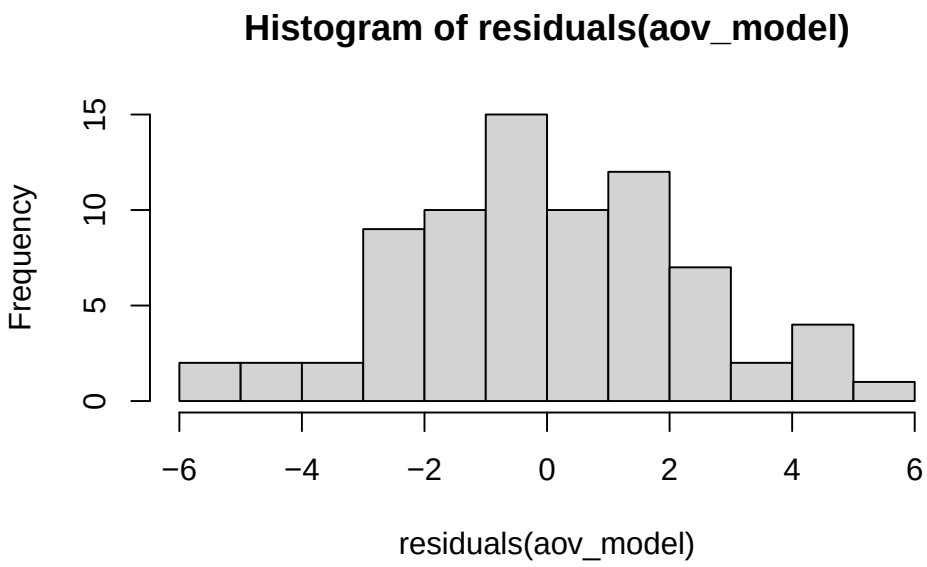
Residual standard error: 2.371 on 73 degrees of freedom
Multiple R-squared:  0.1285,    Adjusted R-squared:  0.1047
F-statistic: 5.383 on 2 and 73 DF,  p-value: 0.006596
```

То же самое!

22.5 Допущения ANOVA

1. **Нормальность распределения ошибок:**

```
hist(residuals(aov_model))
```

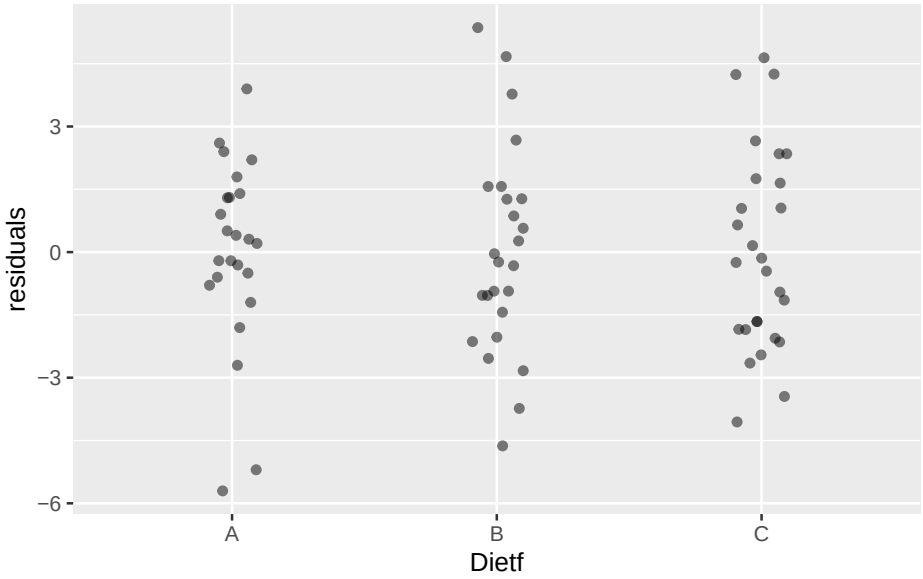


Как мы видим, распределение не сильно далеко от нормального - этого вполне достаточно. ANOVA - это метод достаточно устойчивый к отклонениям от нормальности.

2. **Гомогенность дисперсий.**

То есть их равенство. Можно посмотреть на распределение остатков:

```
diet$residuals <- residuals(aov_model)
ggplot(diet, aes(x = Dietf, y = residuals))+ geom_jitter(width = 0.1, alpha = 0.5)
```



Все выглядит неплохо: нет какой-то одной группы, у которой разброс сильно больше или меньше. Есть и более формальные способы проверить равенство дисперсий. Например, с помощью теста Ливиня (Levene's test). Для того, чтобы его провести, мы воспользуемся новым пакетом `ez` (читать как "easy"). Этот пакет сильно упрощает проведение дисперсионного анализа, особенно для более сложных дизайнов.

```
install.packages("ez")
```

Синтаксис довольно простой: нужно указать, данные, зависимую переменную, переменную с ID, факторы. Необходимо прописать фактор в `between =` или `within =`. В данном случае - в `between =`.

```
library(ez)
ez_model <- ezANOVA(data = diet,
        dv= weight.loss,
        wid = Person,
        between = Dietf,
        detailed = T,
        return_aov = T)
```

```
Warning: You have removed one or more Ss from the analysis. Refactoring
"Person" for ANOVA.
```

```
Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().


Coefficient covariances computed by hccm()
```

```
ez_model
```

```
$ANOVA
  Effect DFn DFd      SSn      SSd        F           p p<.05       ges
1  Dietf   2  73 60.52701 410.4018 5.383104 0.006595853     * 0.1285269

$`Levene's Test for Homogeneity of Variance`
  DFn DFd      SSn      SSd         F         p p<.05
1   2  73 2.040419 160.8859 0.4629076 0.6312856

$aov
Call:
   aov(formula = formula(aov_formula), data = data)

Terms:
                   Dietf Residuals
Sum of Squares   60.5270  410.4018
Deg. of Freedom        2        73

Residual standard error: 2.371064
Estimated effects may be unbalanced
```

Если при проведении теста Ливиня мы получаем $p < .05$, то мы отбрасываем нулевую гипотезу о равенстве дисперсий. В данном случае мы не можем ее отбросить и поэтому принимаем [5]

Полученный объект (если поставить `return_aov = T`) содержит еще и объект `aov()` - на случай, если у Вас есть функции, которые работают с этим классом:

```
TukeyHSD(ez_model$aov)
```

```
  Tukey multiple comparisons of means
```

[5] Вообще-то эта логика не совсем корректна. Тест Ливиня - это такой же статистический тест, как и остальные. Поэтому считать, что допущения соблюдаются на основании того, что p-value больше допустимого уровня α, - это неправильно. Но для проверки допущений такая не очень корректная практика считается допустимой.

```
    95% family-wise confidence level

Fit: aov(formula = formula(aov_formula), data = data)

$Dietf
         diff       lwr        upr     p adj
B-A  0.032000 -1.589085  1.6530850 0.9987711
C-A -1.848148 -3.439554 -0.2567422 0.0188047
C-B -1.880148 -3.454614 -0.3056826 0.0152020
```

3. **Примерно одинаковое количество испытуемых в разных группах.** Здесь у нас все в порядке:

```
diet %>%
  count(Dietf)
```

```
# A tibble: 3 x 2
  Dietf     n
  <fct> <int>
1 A        24
2 B        25
3 C        27
```

Небольшие различия в размерах групп - это ОК, тем более, что на практике такое очень часто случается: кого-то пришлось выкинуть из анализа, для какой-то строчки были потеряны данные и т.д. Однако больших различий в размерах групп стоит избегать. Самое плохое, когда группы различаются значительно по размеру (более чем в 2 раза) и вариабельность внутри групп отличается значительно (более чем в 2 раза).

22.6 Многофакторный дисперсионный анализ (Factorial ANOVA)

На практике можно встретить One-Way ANOVA (однофакторную ANOVA) довольно редко. Обычно в исследованиях встречается многофакторный дисперсионный анализ, в котором проверяется влияние сразу нескольких факторов. В научных статьях это обозначается примерно так: “3x2 ANOVA”. Это означает, что был проведен двухфакторный дисперсионный анализ, причем в одном факторе было три уровня, во втором - два. В нашем случае

это будут факторы “Диета” и “Пол”. Это означает, что у нас две гипотезы: о влиянии диеты на потерю веса и о влиянии пола на потерю веса. Кроме того, появляется гипотеза о взаимодействии факторов - то есть о том, что разные диеты по разному влияют на потерю веса для разных полов.

Взаимодействие двух факторов хорошо видно на графике с линиями: если две линии параллельны, то взаимодействия нет. Если они не параллельны (пересекаются, сходятся, расходятся), то взаимодействие есть.

```
diet <- diet %>%
  mutate(genderf = factor(gender, labels = c(" ", " "))) #

sem <- function(x) sd(x)/sqrt(length(x)) #
pd = position_dodge(0.05) #

diet %>%
  group_by(Dietf, genderf) %>%
  summarise(meanloss = mean(weight.loss),
            se = sem(weight.loss)) %>%
  ggplot(aes(x = Dietf,
             y = meanloss,
             colour = genderf)) +
  geom_line(aes(group = genderf), position = pd) +
  geom_pointrange(aes(ymin = meanloss - se,
                      ymax = meanloss + se), position = pd) +
  theme_minimal()
```

```
`summarise()` has grouped output by 'Dietf'. You can override using the
`.groups` argument.
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <b6>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>

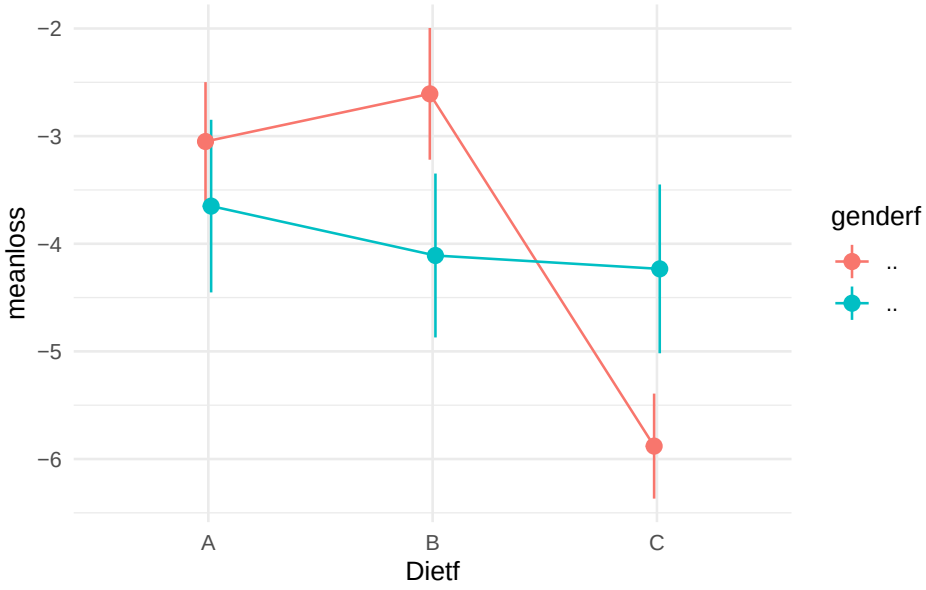

Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>


Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <d0>


Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on ' ' in 'mbcsToSbcs': dot substituted for <bc>
```



Как видно по картинке, разница в эффективности диеты C по сравнению с другими видна только для женщин.

```
ezANOVA(data = diet,
        dv= weight.loss,
        wid = Person,
        between = .(Dietf, gender),
        detailed = T,
        return_aov = T)
```

```
Warning: You have removed one or more Ss from the analysis. Refactoring
"Person" for ANOVA.
```

```
Warning: "gender" will be treated as numeric.
```

```
Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().
```

```
Coefficient covariances computed by hccm()
```

```
Warning: At least one numeric between-Ss variable detected, therefore no
assumption test will be returned.
```

```
$ANOVA
        Effect DFn DFd        SSn     SSd          F          p p<.05
1        Dietf   2  70 60.4172197 376.329 5.61902602 0.00545568     *
2       gender   1  70  0.1686958 376.329 0.03137868 0.85990976
3 Dietf:gender   2  70 33.9040683 376.329 3.15320438 0.04884228     *
          ges
1 0.138334829
2 0.000448066
3 0.082645860

$aov
Call:
   aov(formula = formula(aov_formula), data = data)

Terms:
                  Dietf   gender Dietf:gender Residuals
Sum of Squares   60.5270   0.1687      33.9041  376.3290
Deg. of Freedom        2        1            2        70

Residual standard error: 2.318648
Estimated effects may be unbalanced
```

Итак, теперь мы проверяем три гипотезы вместо одной. Действительно, взаимодействие диеты и пола оказалось значимым, как и ожидалось.

22.7 Дисперсионный анализ с повторными измерениями (Repeated-measures ANOVA)

Если обычный дисперсионный анализ - это аналог независимого т-теста для нескольких групп, то дисперсионный анализ с повторными измерениями - это аналог зависимого т-теста. В функции `ezANOVA()` для проведения дисперсионного анализа с повторными измерениями нужно просто поставить нужным параметром внутригрупповую переменную. Это означает, что в данном случае мы должны иметь данные в длинном формате, для чего мы воспользуемся функцией `pivot_longer()`:

```
dietlong <- diet %>%
  pivot_longer(cols = c(pre.weight, weight6weeks),
               names_to = "time",
```

```
                values_to = "weight")
dietlongC <- dietlong %>%
          filter(Dietf == "C") %>%
          droplevels()
```

```
ezANOVA(dietlongC,
        dv = weight,
        wid = Person,
        within = time)
```

```
Warning: Converting "time" to factor for ANOVA.
```

```
$ANOVA
  Effect DFn DFd        F            p p<.05       ges
2   time   1  26 124.6949 2.030459e-11     * 0.0986036
```

22.8 Смешанный дисперсионный анализ (Mixed between-within subjects ANOVA)

Нам никто не мешает совмещать и внутригруппodые, и межгрупповые факторы вместе. В `ezANOVA()` это делается просто с помощью прописывания разных факторов в нужных переменных: `between =` и `within =`.

```
ezANOVA(dietlong,
        dv = weight,
        wid = Person,
        within = time,
        between = Dietf)
```

```
Warning: You have removed one or more Ss from the analysis. Refactoring
"Person" for ANOVA.
```

```
Warning: Converting "time" to factor for ANOVA.
```

```
Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().
```

```
$ANOVA
      Effect DFn DFd           F            p p<.05         ges
2      Dietf   2  73   0.8280758 4.409507e-01       0.021710057
3       time   1  73 210.5004045 3.346036e-23     * 0.059209996
4 Dietf:time   2  73   5.3831045 6.595853e-03     * 0.003208607
```

Здесь нас интересует взаимодействие между факторами. Результаты, полученные для этой гипотезы, идентичны результатам по обычному дисперсионному анализу на разницу до и после - по сути это одно и то же.

22.9 Непараметрические аналоги ANOVA

Как было описано выше, ANOVA довольно устойчив к разным отклонениям от нормальности и некоторой гетероскедастичности (разным дисперсиям в выборках). Но если уж у Вас данные ну совсем-совсем ненормальные, несимметричные, а от преобразований шкалы Вы по каким-то причинам отказались, то стоит упомянуть о непараметрических аналогах ANOVA.

22.9.1 Тест Краскела-Уоллеса

Это тест Краскела-Уоллеса - обобщение теста Манна-Уитни на несколько выборок (т.е. аналог межгруппового ANOVA):

```
kruskal.test(weight.loss ~ Dietf, diet)
```

```

    Kruskal-Wallis rank sum test

data:  weight.loss by Dietf
Kruskal-Wallis chi-squared = 9.4159, df = 2, p-value = 0.009023
```

22.9.2 Тест Фридмана

Для зависимых выборок есть тест Фридмана - непараметрический аналог дисперсионного анализа с повторными измерениями:

```
friedman.test(weight ~ time | Person, dietlongC)
```

```
    Friedman rank sum test

data:  weight and time and Person
Friedman chi-squared = 27, df = 1, p-value = 2.035e-07
```

22.10 Заключение

Мы разобрали много разных вариантов дисперсионного анализа. Зачем так много? ANOVA - один из самых распространенных методов как в психологии, так и во многих других областях. Естественно, это отнюдь не все методы, используемые в той же психологии. Более того, некоторые вопросы остались за бортом. Постараюсь коротко их перечислить:

- многомерный ANOVA (Multivariate ANalysis Of Variance; MANOVA) - расширение ANOVA для ситуации нескольких зависимых переменных. Это довольно редкая разновидность ANOVA, который берет во внимание ковариации между зависимыми переменными.
- тестирование сферичности для дисперсионного анализа с повторноми измерениями с помощью теста сферичности Моучли (Mauchly's sphericity test). Этот тест проверяет использует матрицу ковариаций разниц каждого условия с каждым: дисперсии разниц между условиями должны быть примерно одинаковыми. Если нулевая гипотеза о сферичности может быть отброшена (p-value < α), то нужно проводить специальные поправки, обычно это поправки Гринхауса-Гейсера (Greenhouse-Geisser corrections). Мы этого не делали, потому что в ситуации RM-ANOVA с всего двумя условиями эта сферичность никогда не нарушается: у нас всего одна дисперсия разниц между условиями, которую просто-напросто не с чем сравнивать. Тест сферичности Моучли вместе с поправками Гринхауса-Гейсера проводятся автоматически для RM-ANOVA с тремя или более группами при использовании функции `ezANOVA()`, так что особо париться над этим не стоит. Правда, нужно помнить, что как и все подобные статистические тесты допущений, они обладают проблемами, связанными с тем, что это статистические тесты: на маленьких выборках они не заметят даже серьезных отклонений от сферичности, на больших - даже маленькие отклонения, а $p - value < 0.05$, по сути, не может интерпретироваться как верность нулевой гипотезы. Тем не менее, это довольно стандартная процедура.

- Как правильно репортить результаты дисперсионного анализа. Здесь все, конечно, зависит от стиля, используемого конкретным журналом. В психологии и близких к ней дисциплинам фактическим lingua franca является стиль Американской Психологической Ассоциации (APA). И тут у меня есть для Вас хорошие новости: есть куча пакетов в R, которые позволяют репортить результаты статистических тестов в APA-стиле! Спасибо дотошным авторам руководства APA по офромлению статей, что этот стиль настолько точно прописывает, как нужно описывать результаты исследований, что это можно запрограммировать. Я лично пользуюсь пакетом `apa`, он весьма удобен:

```
install.packages("apa")
```

```
library(apa)
anova_apa(ez_model)
```

```
  Effect
1  Dietf F(2, 73) = 5.38, p = .007, petasq = .13 **
```

В тексте это будет выглядеть это будет вот так:

Dietf: $F(2, 73) = 5.38$, $p = .007$, $\eta_p^2 = .13$

Еще есть пакеты `apaStyle` и `papaja`, которые могут даже сразу делать весь документ в APA-формате! Если же Вы описываете результаты самостоятельно вручную, то нужно помнить: ни в коем случае не описывайте только p-value. Обязательно прописывайте значение F и степени свободы, желательно с размером эффекта. Для post-hoc теста часто репортятся только p-value (зачастую только для статистически значимых сравнений), но обязательно нужно прописывать какие именно post-hoc тесты проводились, какой показатель размера эффекта использовался (если использовался), применялись ли тест сферичности Моучли вместе с поправками Гринхауса-Гейсера для дисперсионного анализа с повторными измерениями.

- Модели со смешанными эффектами (mixed-effects models) / иерархическая регрессия (hierarchical regression) / многоуровневое моделирование (multilevel modelling). очень популярный нынче метод, которому повезло иметь много названий - в зависимости от области, в которой он

используется. В экспериментальной психологии обычно он называется “модели со смешанными эффектами” и позволяет включать в линейную регрессию не только фиксированные эффекты (fixed effects), но и случайные эффекты (random effects). Для экспериментальной психологии это интересно тем, что в таких моделях можно не усреднять показатели по испытуемым, а учитывать влияние группирующей переменной “испытуемый” как случайный эффект. Подобные модели используются в самых разных областях. Для их использования в R есть два известных пакета: `nlme` и `lme4`.

Глава 23

Общая линейная модель и ее расширения

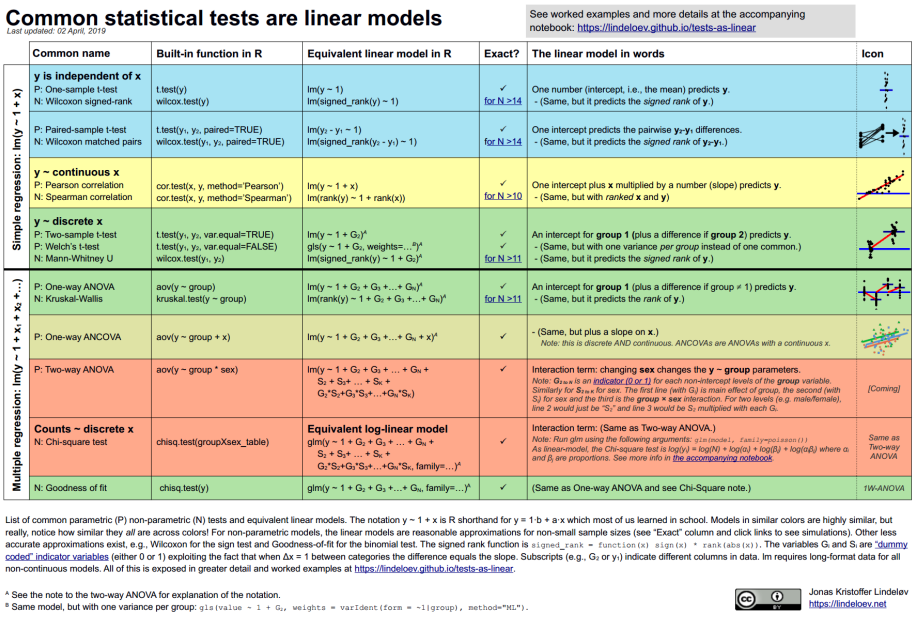
23.1 Общая линейная модель

Обобщением множественной линейной регрессии можно считать **общую линейную модель (general linear model)**. Общая линейная модель может предсказывать не одну, а сразу несколько объясняемых переменных в отличие от множественной линейной регрессии.

$$Y = XB$$

где Y — матрица объясняемых переменных, X — матрица предикторов, B — матрица параметров.

Почти все пройденные нами методы можно рассматривать как частный случай общей линейной модели: t-тесты, коэффициент корреляции Пирсона, линейная регрессия, ANOVA.



Common statistical tests are linear models

Last updated: 02 April, 2019

See worked examples and more details at the accompanying notebook: https://lindeloev.github.io/tests-as-linear

	Common name	Built-in function in R	Equivalent linear model in R	Exact?	The linear model in words	Icon
Simple regression: lm(y ~ 1 + x)	**y is independent of x** P: One-sample t-test N: Wilcoxon signed-rank	t.test(y) wilcox.test(y)	lm(y ~ 1) lm(signed_rank(y) ~ 1)	✓ for N >14	One number (intercept, i.e., the mean) predicts **y**. - (Same, but it predicts the *signed rank* of **y**.)	
	P: Paired-sample t-test N: Wilcoxon matched pairs	t.test(y₁, y₂, paired=TRUE) wilcox.test(y₁, y₂, paired=TRUE)	lm(y₂ - y₁ ~ 1) lm(signed_rank(y₂ - y₁) ~ 1)	✓ for N >14	One intercept predicts the pairwise **y₂-y₁** differences. - (Same, but it predicts the *signed rank* of **y₂-y₁**.)	
	y ~ continuous x P: Pearson correlation N: Spearman correlation	cor.test(x, y, method='Pearson') cor.test(x, y, method='Spearman')	lm(y ~ 1 + x) lm(rank(y) ~ 1 + rank(x))	✓ for N >10	One intercept plus **x** multiplied by a number (slope) predicts **y**. - (Same, but with *ranked* **x** and **y**)	
	y ~ discrete x P: Two-sample t-test P: Welch's t-test N: Mann-Whitney U	t.test(y₁, y₂, var.equal=TRUE) t.test(y₁, y₂, var.equal=FALSE) wilcox.test(y₁, y₂)	lm(y ~ 1 + G₂)[A] gls(y ~ 1 + G₂, weights=…[B])[A] lm(signed_rank(y) ~ 1 + G₂)[A]	✓ ✓ for N >11	An intercept for **group 1** (plus a difference if **group 2**) predicts **y**. - (Same, but with one variance *per group* instead of one common.) - (Same, but it predicts the *signed rank* of **y**.)	
Multiple regression: lm(y ~ 1 + x₁ + x₂ +…)	P: One-way ANOVA N: Kruskal-Wallis	aov(y ~ group) kruskal.test(y ~ group)	lm(y ~ 1 + G₂ + G₃ +…+ Gₙ)[A] lm(rank(y) ~ 1 + G₂ + G₃ +…+ Gₙ)[A]	✓ for N >11	An intercept for **group 1** (plus a difference if group ≠ 1) predicts **y**. - (Same, but it predicts the *rank* of **y**.)	
	P: One-way ANCOVA	aov(y ~ group + x)	lm(y ~ 1 + G₂ + G₃ +…+ Gₙ + x)[A]	✓	- (Same, but plus a slope on **x**.) *Note: this is discrete AND continuous. ANCOVAs are ANOVAs with a continuous x.*	
	P: Two-way ANOVA	aov(y ~ group * sex)	lm(y ~ 1 + G₂ + G₃ + … + Gₙ + S₂ + S₃+ … + Sₖ + G₂*S₂+G₃*S₃+…+Gₙ*Sₖ)	✓	Interaction term: changing **sex** changes the **y ~ group** parameters. *Note: $G_{2\,in\,k}$ is an indicator (0 or 1) for each non-intercept levels of the **group** variable. Similarly for $S_{2\,in\,k}$ for sex. The first line (with G_i) is main effect of group, the second (with S_i) for sex and the third is the **group × sex** interaction. For two levels (e.g. male/female), line 2 would just be "S_2" and line 3 would be S_2 multiplied with each G_i.*	*[Coming]*
	Counts ~ discrete x N: Chi-square test	chisq.test(groupXsex_table)	**Equivalent log-linear model** glm(y ~ 1 + G₂ + G₃ + … + Gₙ + S₂ + S₃+ … + Sₖ + G₂*S₂+G₃*S₃+…+Gₙ*Sₖ, family=…)[A]	✓	Interaction term: (Same as Two-way ANOVA.) *Note: Run glm using the following arguments: `glm(model, family=poisson())` As linear-model, the Chi-square test is log(y) = log(N) + log(α) + log(β₂) + log(α₂β₂) where α, and β₂ are proportions. See more info in the accompanying notebook.*	*Same as Two-way ANOVA*
	N: Goodness of fit	chisq.test(y)	glm(y ~ 1 + G₂ + G₃ +…+ Gₙ, family=…)[A]	✓	(Same as One-way ANOVA and see Chi-Square note.)	*1W-ANOVA*

List of common parametric (P) non-parametric (N) tests and equivalent linear models. The notation y ~ 1 + x is R shorthand for y = 1·b + a·x which most of us learned in school. Models in similar colors are highly similar, but really, notice how similar they *all* are across colors! For non-parametric models, the linear models are reasonable approximations for non-small sample sizes (see "Exact" column and click links to see simulations). Other less accurate approximations exist, e.g., Wilcoxon for the sign test and Goodness-of-fit for the binomial test. The signed rank function is `signed_rank = function(x) sign(x) * rank(abs(x))`. The variables G_i and S_i are "dummy coded" indicator variables (either 0 or 1) exploiting the fact that when Δx = 1 between categories the difference equals the slope. Subscripts (e.g., G_2 or y_1) indicate different columns in data. lm requires long-format data for all non-continuous models. All of this is exposed in greater detail and worked examples at https://lindeloev.github.io/tests-as-linear.

[A] See the note to the two-way ANOVA for explanation of the notation.
[B] Same model, but with one variance per group: `gls(value ~ 1 + G₂, weights = varIdent(form = ~1|group), method="ML")`.

23.2 Обобщенная линейная модель

Обобщенная линейная модель (generalized linear model) была придумана как обобщение линейной регрессии и ее сородичей: логистической регрессии и пуассоновской регрессии.

Общая линейная модель задается формулой

$$Y = XB$$

Обобщенная оборачивает предиктор XB **связывающей функцией (link function),** которая различается для разных типов регрессионных моделей.

Давайте попробуем построить модель, в которой объясняемой переменной будет то, является ли супергерой хорошим или плохим.

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become er
```

```
heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
                   na = c("-", "-99"))
```

```
New names:
* `` -> `...1`
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 734 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, Gender, Eye color, Race, Hair color, Publisher, Skin color, A...
dbl (3): ...1, Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
heroes$good <- heroes$Alignment == "good"
```

Обычная линейная модель нам не подходит, если распределение наших ошибок далеко от нормального. А это значит, что мы не можем использовать общую линейную модель с бинарной объясняемой переменной. Эту проблему решает логистическая регрессия, которая является частным случаем обобщенной линейной модели.

Для этого нам понадобится функция `glm()`, а не `lm()` как раньше. Ее синтаксис очень похож, но нам теперь нужно задать еще один важный параметр `family =` для выбора связывающей функции (в

данном случае это логит-функция, которая является связующей функцией по умолчанию для биномиального семейства функций в `glm()`).

```
heroes_good_glm <- glm(good ~ Weight + Gender, heroes, family = binomial())
summary(heroes_good_glm)
```

```
Call:
glm(formula = good ~ Weight + Gender, family = binomial(), data = heroes)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8674  -1.3103   0.6334   0.9155   2.4007

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.763917   0.235410   7.493 6.73e-14 ***
Weight      -0.004253   0.001124  -3.783 0.000155 ***
GenderMale  -0.760310   0.245851  -3.093 0.001984 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 605.22  on 477  degrees of freedom
Residual deviance: 570.88  on 475  degrees of freedom
  (256 observations deleted due to missingness)
AIC: 576.88

Number of Fisher Scoring iterations: 4
```

Результат очень похож по своей структуре на `glm()`, однако вместо R^2 перед нами AIC. AIC расшифровывается как информационный критерий Акаике (Akaike information criterion) — это критерий использующийся для выбора из нескольких моделей. Чем он меньше, тем лучше модель. Как и Adjusted R^2, AIC "наказывает" за большое количество параметров в модели.

Поскольку AIC — это относительный показатель качества модели, нам нужно сравнить его с AIC другой, более общей модели. Можно сравнить с моделью без веса супергероев.

```
heroes_good_glm_noweight <- glm(good ~ Gender, heroes, family = binomial())
summary(heroes_good_glm_noweight)
```

```
Call:
glm(formula = good ~ Gender, family = binomial(), data = heroes)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8082  -1.4164   0.6586   0.9559   0.9559

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.4178     0.1785   7.944 1.95e-15 ***
GenderMale   -0.8716     0.2012  -4.332 1.48e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 873.81  on 698  degrees of freedom
Residual deviance: 853.24  on 697  degrees of freedom
  (35 observations deleted due to missingness)
AIC: 857.24

Number of Fisher Scoring iterations: 4
```

AIC стал больше, следовательно, мы выберем модель с весом супергероев.

23.3 Модель со смешанными эффектами

Модели со смешанными эффектами (mixed-effects models) — это то же самое, что и иерархическая регрессия (hierarchical regression) или многоуровневое моделирование (multilevel modelling). Этому методу повезло иметь много названий - в зависимости от области, в которой он используется. Модели со смешанными эффектами позволяет включать в линейную регрессию не только фиксированные эффекты (fixed effects), но и случайные эффекты (random effects).

Для экспериментальных дисциплин это интересно тем, что в таких моделях можно не усреднять показатели по испытуемым или образцам, а учитывать влияние соотвествующей группирующей переменной как случайный эффект. В отличие от обычного фактора как в линейной регрессии или дисперсионном анализе (здесь он называется фиксированным), случайный эффект не

интересует нас сам по себе, а его значения считаются случайной переменной.

Смешанные модели используются в самых разных областях. Они позволяют решить проблему зависимости наблюдений без усреднения значений по испытуемым или группам, что повышает статистическую мощность.

Для работы со смешанными моделями в R есть два известных пакета: `nlme` и более современный `lme4`.

```
install.packages("lme4")
```

```
library(lme4)
```

```
Loading required package: Matrix
```

```
Attaching package: 'Matrix'
```

```
The following objects are masked from 'package:tidyr':

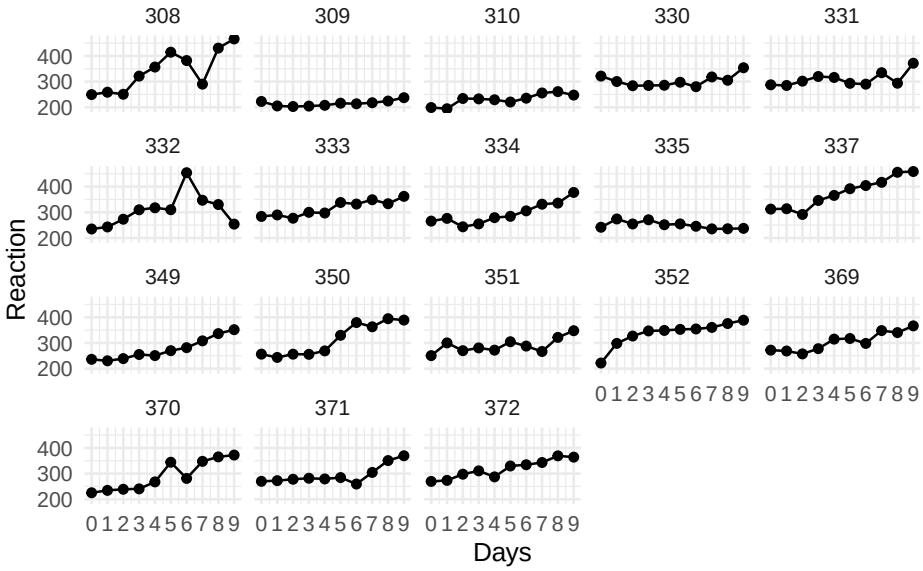
    expand, pack, unpack
```

Для примера возьмем данные исследования влияния депривации сна на время реакции.

```
data("sleepstudy")
```

Данные представлены в длинном формате: каждая строчка — это усредненное время реакции для одного испытуемого в соответствующий день эксперимента.

```
sleepstudy %>%
  ggplot(aes(x = Days, y = Reaction)) +
  geom_line() +
  geom_point() +
  scale_x_continuous(breaks = 0:9) +
  facet_wrap(~Subject) +
  theme_minimal()
```



Можно заметить, что, в среднем, время реакции у испытуемых повышается от первого к последнему дню. С помощью смешанных моделей мы можем проверить, различается ли скорость возрастания времени реакции от дня к дню у разных испытуемых.

Для этого мы сравниваем две модели, одна из которых является “вложенной” в другую, то есть усложненной версией более общей модели. В данном случае, более общая модель предполагает, что время реакции увеличивается у всех испытуемых одинаково, а испытуемые различаются только средним временем реакции.

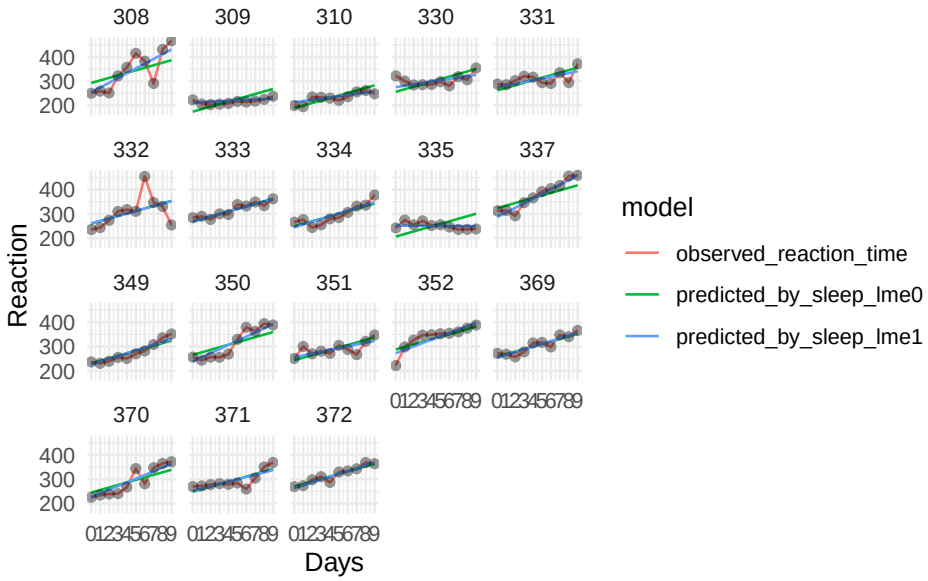
```
sleep_lme0 <- lmer(Reaction ~ Days + (1 | Subject), sleepstudy)
sleep_lme1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
```

Визуализируем предсказания двух моделей:

```
sleepstudy$predicted_by_sleep_lme0 <- predict(sleep_lme0)
sleepstudy$predicted_by_sleep_lme1 <- predict(sleep_lme1)
```

```
sleepstudy %>%
  rename(observed_reaction_time = Reaction) %>%
  pivot_longer(cols = c(observed_reaction_time, predicted_by_sleep_lme0, predicted_by_sleep_lm
  ggplot(aes(x = Days, y = Reaction)) +
  geom_line(aes(colour = model)) +
  #geom_line(aes(y = predicted_by_M1), colour = "orange") +
```

```
  #geom_line(aes(y = predicted_by_M2), colour = "purple") +
  geom_point(data = sleepstudy, alpha = 0.4) +
  scale_x_continuous(breaks = 0:9) +
  facet_wrap(~Subject) +
  theme_minimal()
```



Зеленая линия (нулевая модель) имеет везде один и тот же наклон, а синяя (альтернативная модель) имеет разный наклон у всех испытуемых.

Есть несколько способов сравнивать модели, например, уже знакомый нам AIC. Кроме того, можно сравнить две модели с помощью теста хи-квадрат, воспоплльзовавшись функцией `anova()`.

```
  anova(sleep_lme0, sleep_lme1)
```

```
refitting model(s) with ML (instead of REML)

Data: sleepstudy
Models:
sleep_lme0: Reaction ~ Days + (1 | Subject)
sleep_lme1: Reaction ~ Days + (Days | Subject)
           npar    AIC    BIC  logLik deviance  Chisq Df Pr(>Chisq)
```

```
sleep_lme0    4 1802.1 1814.8 -897.04   1794.1
sleep_lme1    6 1763.9 1783.1 -875.97   1751.9 42.139  2  7.072e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Модель со случайным наклоном прямой оказалась лучше, о чем нам говорят как более низкие AIC и BIC, так и тестирование с помощью хи-квадрат.

Глава 24

Многомерные методы анализа данных

Многомерные методы анализа данных – методы работы с данными, в которых много колонок. Мы уже сталкивались с некоторыми многомерными методами, например, с множественной линейной регрессией (Глава 21.5). Поэтому вы знаете, что многомерность создает новые проблемы. Например, при множественных корреляциях или попарных сравнениях возникает проблема множественных сравнений, а при использовании множественной регрессии лишние предикторы могут ловить только шум и приводить к переобучению (если говорить в терминах машинного обучения). Короче говоря, больше - не значит лучше. Нужно четко понимать, зачем мы используем данные и что пытаемся измерить.

Однако в некоторых случаях мы в принципе не можем ничего интересного сделать с маленьким набором переменных. Много ли мы можем измерить личностным тестом с одним единственным вопросом? Можем ли мы точно оценить уровень интеллекта по успешности выполнения одного единственного задания? Очевидно, что нет. Более того, даже концепция интеллекта в современном его представлении появилась во многом благодаря разработке многомерных методов анализа! Ну или наоборот: исследования интеллекта подстегнули развитие многомерных методов.

24.1 Уменьшение размерности

Представьте многомерное пространство, где каждая колонка — это отдельная ось, а каждая строка задает координаты одной точки в этом пространстве. Мы получим многомерную диаграмму рассеяния.

Многомерную диаграмму рассеяния, к сожалению, нельзя нарисовать, поэтому нарисуем несколько двухмерных диаграмм рассеяния для отображения сочетания всех колонок со всеми набора данных `penguins`.

```
library(tidyverse)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

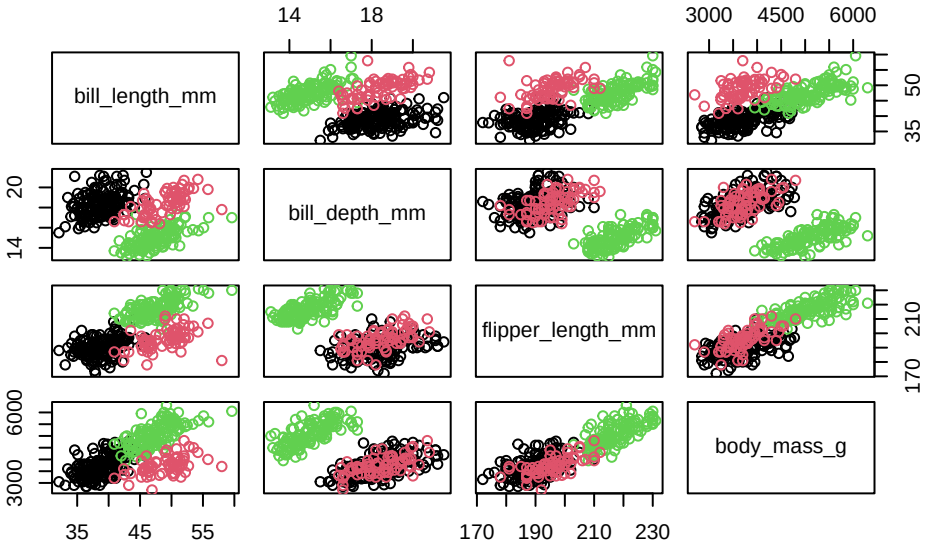
```
Warning: package 'stringr' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

```
library(palmerpenguins)

penguins <- penguins %>%
  drop_na(bill_length_mm:body_mass_g)

penguins %>%
  select(bill_length_mm:body_mass_g) %>%
  plot(col = penguins$species)
```



Даже представить как устроены многомерные данные очень трудно, а ведь мы отобразили только четыре числовых колонки! Понять связи между отдельными переменными мы можем, если посмотрим на каждую диаграмму рассеяния по отдельности, но и то если их не слишком много. Но связи в многомерных данных могут быть гораздо более сложными и на простых диаграммах рассеяния их иногда нельзя заметить. Это как смотреть на тени, поворачивая какой-нибудь предмет различными сторонами.

Уменьшение размерности позволяет вытащить из данных самую мякотку, уместив исходные данные в небольшое количество переменных. Остальное из данных удаляется.

> ***Осторожно:* уменьшение размерности приводит к потере данных**
>
> Уменьшение размерности – не “бесплатная” операция. Уменьшение размерности всегда приводит к потере части данных и может значительно их изменить.

Данные сниженной размерности гораздо проще визуализировать. Например, снизив размерность данных до двух шкал, можно просто нарисовать диаграмму рассеяния. Анализируя связь полученных шкал с изначальными шкалами и взаимное расположение точек в новых шкалах, можно многое понять об исследуемых данных.

Однако снижение размерности используется не только для эксплораторного анализа, но и для снижения количества фичей

в машинном обучения, для удаления шума из данных, для более экономного хранения данных и многих других задач.

24.2 Анализ главных компонент *(Principal component analysis)*

Анализ главных компонент *(АГК; Principal component analysis)* – это, пожалуй, самый известный метод **уменьшения размерности.** АГК просто поворачивает систему координат многомерного пространства, которое задано имеющимися числовыми колонками. Координатная система поворачивается таким образом, чтобы первые оси объясняли как можно больший разброс данных, а последние - как можно меньший. Тогда мы могли бы отбросить последние оси и не очень-то многое потерять в данных. Для двух осей это выглядит вот так:

К сожалению, в PDF нельзя вставить .gif анимацию, посмотреть ее можно по ссылке

Первая ось должна минимизировать красные расстояния. Вторая ось будет просто перпендикулярна первой оси.

***Для продвинутых:* какая математика стоит за АГК?**

1. Для начала мы посчитаем корреляционную матрицу (как вариант – ковариационную матрицу, если мы не хотим делать z-преобразование переменных

```
penguins_cor <- penguins %>%
  select(bill_length_mm:body_mass_g) %>%
  cor()
penguins_cor
```

```
                  bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
bill_length_mm         1.0000000    -0.2350529         0.6561813   0.5951098
bill_depth_mm         -0.2350529     1.0000000        -0.5838512  -0.4719156
flipper_length_mm      0.6561813    -0.5838512         1.0000000   0.8712018
body_mass_g            0.5951098    -0.4719156         0.8712018   1.0000000
```

2. Находим **собственные векторы *(eigenvectors)*** и **собственные значения *(eigenvalues)*** матрицы корреляций или ковариаций.

```
eigens <- eigen(penguins_cor)
eigens
```

```
eigen() decomposition
$values
[1] 2.7537551 0.7725168 0.3652359 0.1084922

$vectors
           [,1]         [,2]       [,3]       [,4]
[1,]  0.4552503 -0.597031143  0.6443012  0.1455231
[2,] -0.4003347 -0.797766572 -0.4184272 -0.1679860
[3,]  0.5760133 -0.002282201 -0.2320840 -0.7837987
[4,]  0.5483502 -0.084362920 -0.5966001  0.5798821
```

Собственные вектора - это такие векторы матрицы, умножив которые на данную матрицу, можно получи вектор с тем же направлением, но другой длины.

```
eigens$vectors
```

```
           [,1]         [,2]       [,3]       [,4]
[1,]  0.4552503 -0.597031143  0.6443012  0.1455231
[2,] -0.4003347 -0.797766572 -0.4184272 -0.1679860
[3,]  0.5760133 -0.002282201 -0.2320840 -0.7837987
[4,]  0.5483502 -0.084362920 -0.5966001  0.5798821
```

А вот коэффициент множителя длины нового вектора - это **собственное значение.**

```
eigens$values
```

```
[1] 2.7537551 0.7725168 0.3652359 0.1084922
```

В контексте АГК, **собственные вектора** - это новые оси (т.е. те самые новые компоненты), а **собственные значения** - это размер объясняемой дисперсии с помощью новых осей. Чтобы перейти от дисперсии к стандартному отклонению, нам нужно взять корень из **собственных значений:**

```
sqrt(eigens$values)
```

```
[1] 1.6594442 0.8789293 0.6043475 0.3293816
```

Собственные вектора, ранжированные по их собственным значениям от большего к меньшему, — это и есть главные компоненты в искомом порядке.
Именно таким образом считается АГК в функции `princomp()`.

```
penguins_princomp <- penguins %>%
  select(bill_length_mm:body_mass_g) %>%
  princomp(cor = TRUE)
penguins_princomp$loadings
```

```
Loadings:
                  Comp.1 Comp.2 Comp.3 Comp.4
bill_length_mm     0.455  0.597  0.644  0.146
bill_depth_mm     -0.400  0.798 -0.418 -0.168
flipper_length_mm  0.576         -0.232 -0.784
body_mass_g        0.548         -0.597  0.580

                Comp.1 Comp.2 Comp.3 Comp.4
SS loadings       1.00   1.00   1.00   1.00
Proportion Var    0.25   0.25   0.25   0.25
Cumulative Var    0.25   0.50   0.75   1.00
```

```
penguins_princomp$sdev
```

```
   Comp.1    Comp.2    Comp.3    Comp.4
1.6594442 0.8789293 0.6043475 0.3293816
```

Функция `prcomp()` идет другим путем, используя **сингулярное разложение матрицы *(singular value decomposition)*** исходных данных, это дает чуть более точные результаты.

3. Последний этап – поворот исходных данных в новом пространстве с помощью матричного перемножения изначальных данных (z-трансформированных или нет).

```
penguins_original_scales <- penguins %>%
  select(bill_length_mm:body_mass_g) %>%
  as.matrix() %>%
  scale()

head(penguins_original_scales %*% eigens$vectors)
```

```
          [,1]       [,2]       [,3]       [,4]
```

```
[1,] -1.840748 -0.04763243 -0.2324536  0.5231365
[2,] -1.304850  0.42772154 -0.0295191  0.4018377
[3,] -1.367178  0.15425039  0.1983816 -0.5272343
[4,] -1.876078  0.00204541 -0.6176912 -0.4776785
[5,] -1.908951 -0.82799642 -0.6855795 -0.2071241
[6,] -1.760446  0.35096537  0.0276311  0.5039784
```

```
penguins_prcomp <- penguins %>%
  select(bill_length_mm:body_mass_g) %>%
  prcomp(center = TRUE, scale. = TRUE)

head(penguins_prcomp$x)
```

```
           PC1         PC2        PC3        PC4
[1,] -1.840748 -0.04763243  0.2324536  0.5231365
[2,] -1.304850  0.42772154  0.0295191  0.4018377
[3,] -1.367178  0.15425039 -0.1983816 -0.5272343
[4,] -1.876078  0.00204541  0.6176912 -0.4776785
[5,] -1.908951 -0.82799642  0.6855795 -0.2071241
[6,] -1.760446  0.35096537 -0.0276311  0.5039784
```

Итак, для начала нам нужно центрировать и нормировать данные - вычесть среднее и поделить на стандартное отклонение, т.е. посчитать z-оценки (см. Глава 12.5.3.1). Это нужно для того, чтобы сделать все шкалы равноценными. Это особенно важно делать когда разные шкалы используют несопоставимые единицы измерения. Скажем, одна колонка - это масса человека в килограммах, а другая - рост в метрах. Если применять АГК на этих данных, то ничего хорошего не выйдет: вклад роста будет слишком маленьким. А вот если мы сделаем z-преобразование, то приведем и вес, и рост к “общему знаменателю”.

В базовом R уже есть инструменты для АГК `princomp()` и `prcomp()`, считают они немного по-разному. Возьмем более рекомендуемый вариант, `prcomp()`. Эта функция умеет самостоятельно поводить z-преобразования, для чего нужно поставить `center = TRUE` и `scale. = TRUE`.

```
penguins_prcomp <- penguins %>%
  select(bill_length_mm:body_mass_g) %>%
  prcomp(center = TRUE, scale. = TRUE)
```

Уже много раз встречавшаяся нам функция `summary()`, примененная на результат проведения АГК, выдаст информацию о полученных компонентах. Наибольший интерес представляют строчки

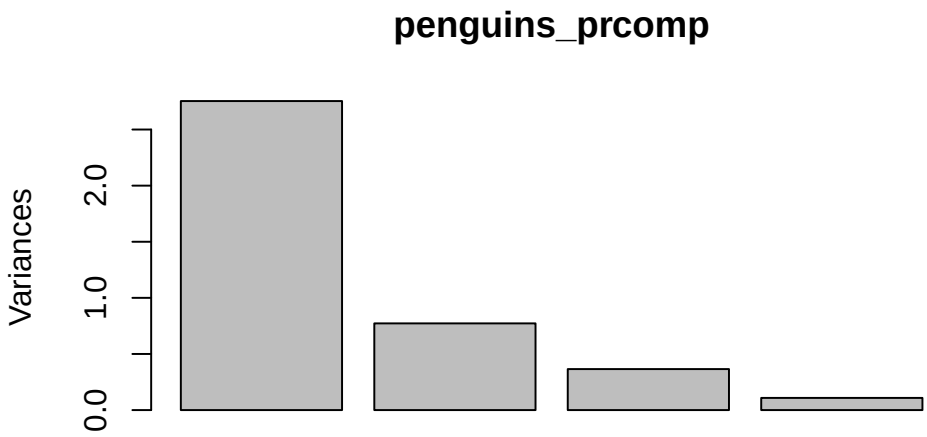
"Proportion of Variance" (доля дисперсии, объясненная компонентой) и *"Cumulative Proportion"* (накопленная доля дисперсии для компоненты данной и всех предыдущих компонент).

```
summary(penguins_prcomp)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.6594 0.8789 0.60435 0.32938
Proportion of Variance 0.6884 0.1931 0.09131 0.02712
Cumulative Proportion  0.6884 0.8816 0.97288 1.00000
```

Функция `plot()` повзоляет визуализировать соотношение разных компонент.

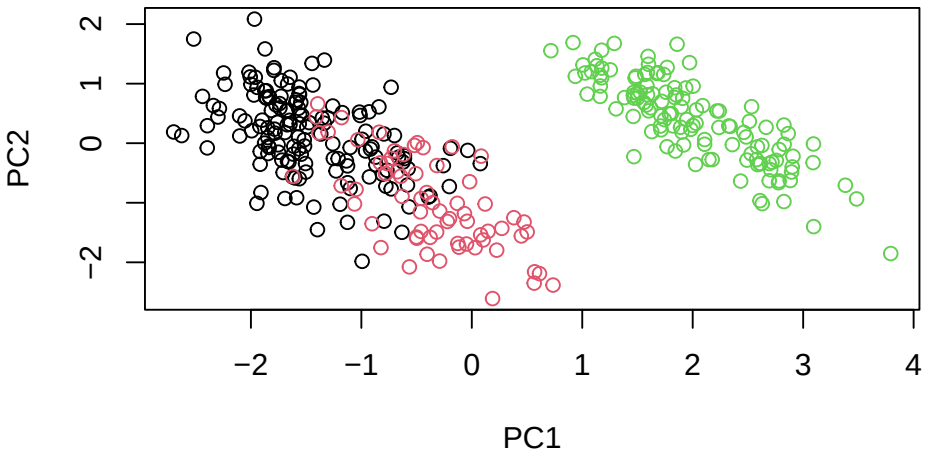
```
plot(penguins_prcomp)
```



Как мы видим, первый компонент объясняет большую часть дисперсии, второй и третий компоненты заметно меньше, последний – совсем немного, скорее всего, этот компонент репрезентирует некоторый шум в данных.

Теперь мы можем визуализировать первые два компонента. Это можно сделать с помощью базовых инструментов R.

```
plot(penguins_prcomp$x[,1:2], col=penguins$species)
```



Однако пакет {factoextra} представляет гораздо более широкие возможности для визуализации.

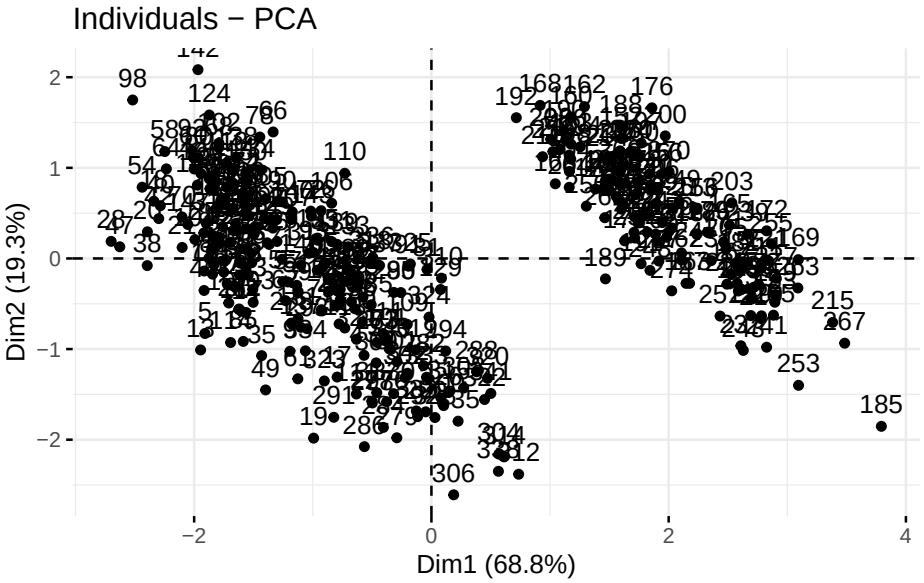
```
install.packages("factoextra")
```

Во-первых, как и с помощью базового `plot()`, можно нарисовать диаграмму рассеяния. Для этого воспользуемся функцией `fviz_pca_ind()`:

```
library(factoextra)
```

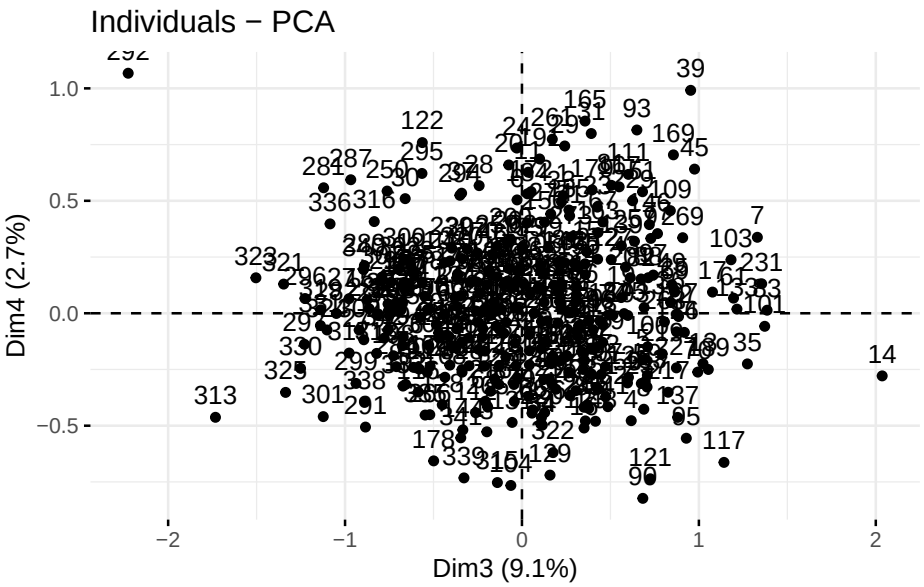
```
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
fviz_pca_ind(penguins_prcomp)
```



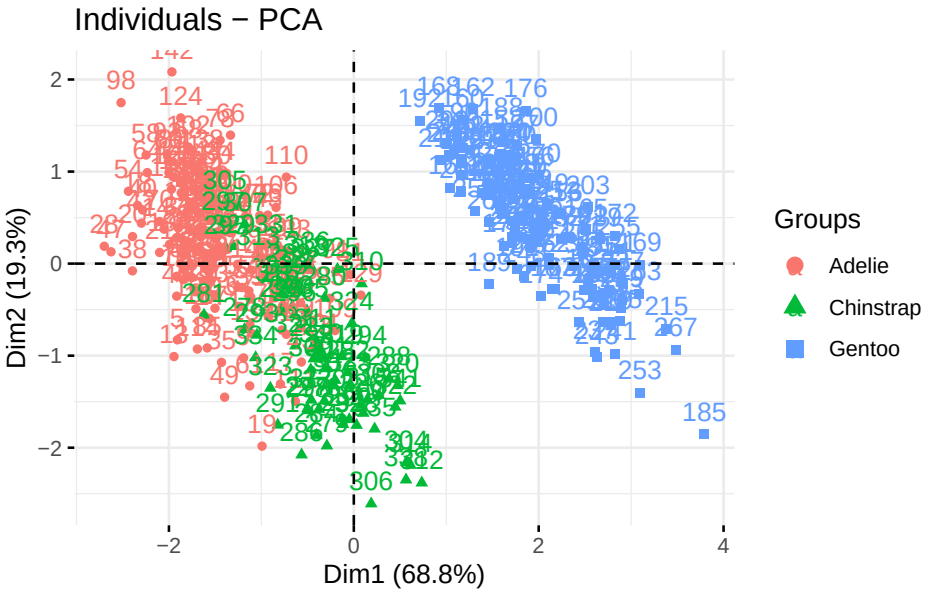
Параметром `axes =` можно выбрать нужные компоненты для отображения.

```
fviz_pca_ind(penguins_prcomp,
             axes = c(3, 4))
```



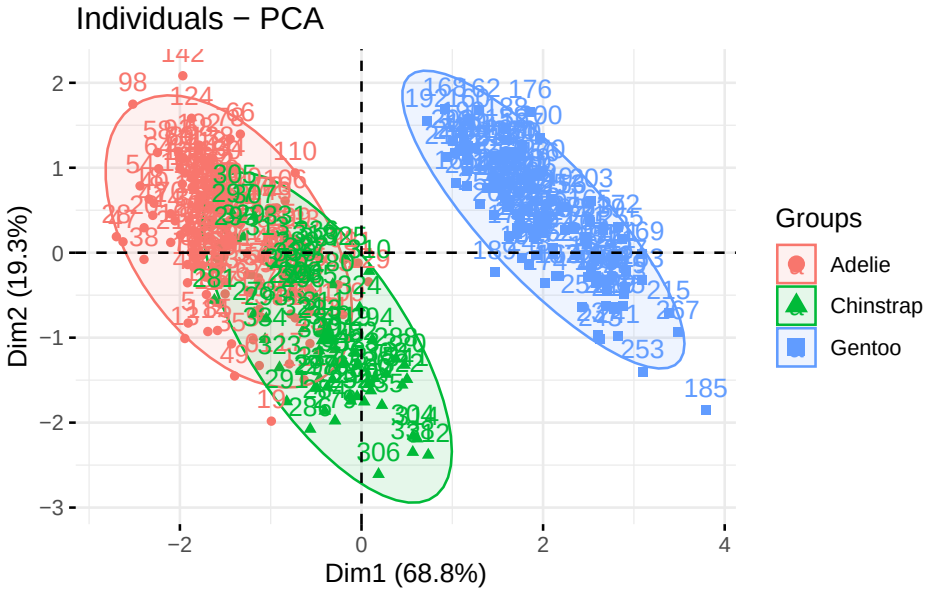
Добавить расцветку можно с помощью параметра `habillage =`, в которой надо подставить нужный вектор из изначальных данных.

```
fviz_pca_ind(penguins_prcomp,
             habillage = penguins$species)
```



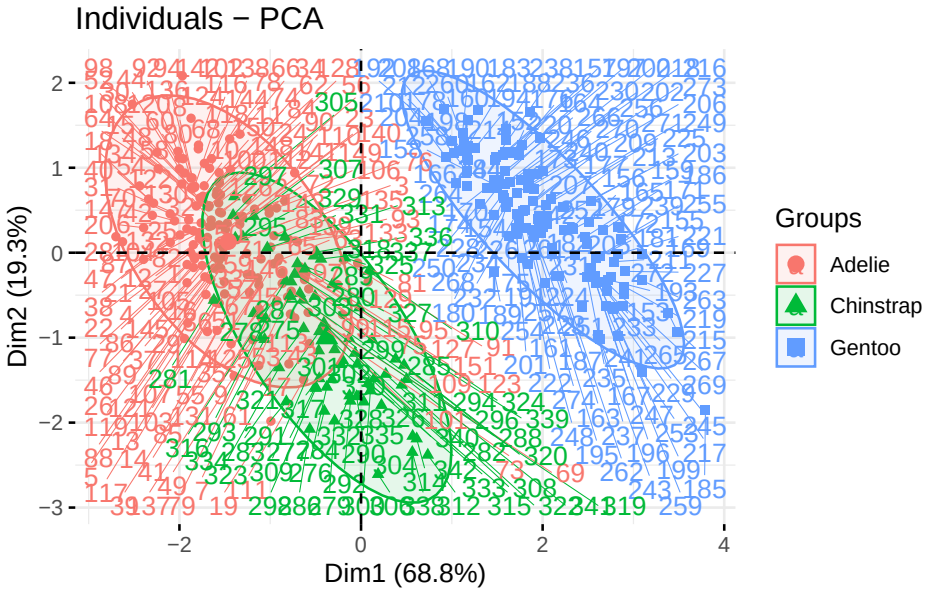
Параметр `addEllipses = TRUE` позволяет обрисовать эллипсы вокруг точек, если те раскрашены по группам с помощью `habillage =`.

```
fviz_pca_ind(penguins_prcomp,
             habillage = penguins$species,
             addEllipses = TRUE)
```



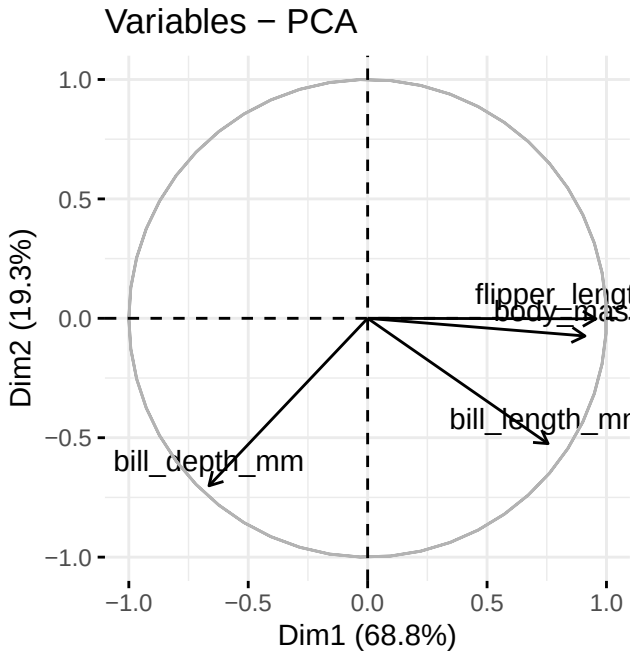
Чтобы рассмотреть отдельные точки, можно поставить `repel = TRUE`:

```
fviz_pca_ind(penguins_prcomp,
             habillage = penguins$species,
             addEllipses = TRUE,
             repel = TRUE)
```



В дополнение к диаграмме рассеяния можно нарисовать график переменных функцией `fviz_pca_var()`. По осям x и y будут отображены первая и вторая компоненты соответственно. Как и для функции `fviz_pca_ind()`, можно выбрать и другие компоненты с помощью параметра `axes =`.

```
fviz_pca_var(penguins_prcomp)
```

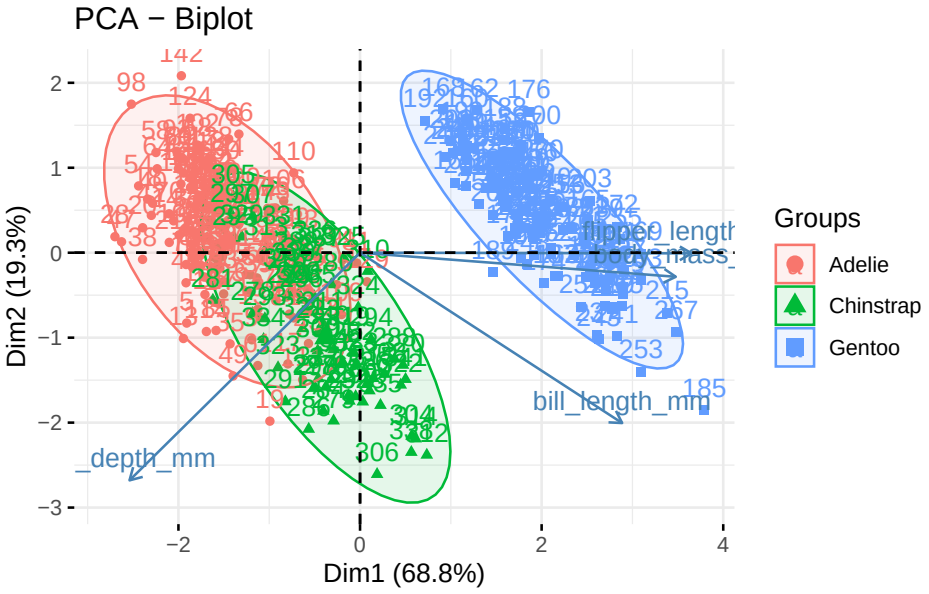


Вместо отдельных точек мы видим здесь стрелочки, которые представляют собой изначальные шкалы. Чем ближе эти стрелочки к осям x и y, тем больше коэффициент корреляции между исходной шкалой и выбранной компонентой.

Если коэффициент корреляции отрицательный, то стрелочка исходной шкалы направлена в противоположную сторону от оси (влево или вниз).

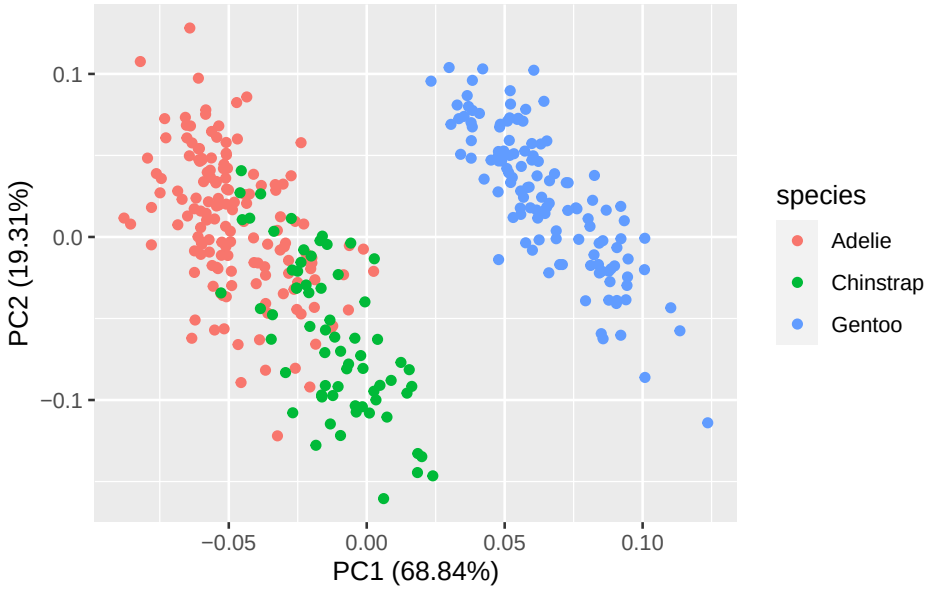
Наконец, есть так называемый **биплот *(biplot),*** в котором объединены оба графика: и индивидуальные точки, и стрелочки с изначальными шкалами!

```
fviz_pca_biplot(penguins_prcomp,
                addEllipses = TRUE,
                habillage = penguins$species)
```



Еще один способ нарисовать **биплот** – с помощью пакета `{ggfortify}`. В этом пакете есть функция `autoplot()`, которая автоматически рисует что-то в зависимости от класса объекта на входе. Если это объект класса `prcomp`, то мы получим тот самый **биплот.**

```
library(ggfortify)
autoplot(penguins_prcomp, data = penguins, colour = "species")
```



24.3 tSNE

t-Distributed Stochastic Neighbor Embedding *(t-SNE)* - это еще один метод снижения размерности, который используется, в основном, для визуализации многомерных данных со сложными нелинейными связями между переменными в пространстве меньшей размерности (двумерном или трехмерном).

Идея t-SNE состоит в том, чтобы расположить точки данных на такой картинке таким образом, чтобы близкие объекты оказались близко друг к другу, а далекие объекты - далеко друг от друга. Для этого используется математический подход, основанный на вероятностях и расстояниях между объектами.

Для примера возьмем знаменитый набор данных MNIST (“Modified National Institute of Standards and Technology”). MNIST – это своего рода *iris* или *penguins* в мире глубокого обучения: самый базовый учебный набор данных, на котором учатся писать искусственные нейросети. Этот набор данных очень большой, поэтому мы будем работать с его уменьшенной версией:

```
mnist_small <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/mas
```

```
Rows: 6000 Columns: 785
```

```
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (785): label, 1x1, 1x2, 1x3, 1x4, 1x5, 1x6, 1x7, 1x8, 1x9, 1x10, 1x11, 1...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
mnist_small %>%
  select(1,400:410) %>%
  head()
```

```
# A tibble: 6 x 12
  label `15x7` `15x8` `15x9` `15x10` `15x11` `15x12` `15x13` `15x14` `15x15`
  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1     2      0      0     22       0       0       0       0       0       0
2     8      0      0      0       0     116     254      89      54     235
3     6      0      0      0      56     252     252     252     252     252
4     1      0      0      0       0       0       0      13     200     253
5     5      0      0      0       0      32     191     233     187     145
6     3      0     27     76      44     194     204     226     253     253
# i 2 more variables: `15x16` <dbl>, `15x17` <dbl>
```

MNIST содержит большое количество написанных от руки цифр, каждый столбец – это яркость отдельного пикселя.

Особенность этого набора данных в том, что стандартными линейными методами снижения размерности такими как АГК здесь не обойтись: нужно “поймать” сложные взаимоотношения между различными пикселями, которые создают различные палки, кружочки и закорючки.

Давайте посмотрим, как справится в этой ситуации АГК:

```
mnist_pca <- mnist_small %>%
  select(!label) %>% #
  select(where(function(x) !all(x == 0))) %>% #
  prcomp(center = TRUE, scale. = TRUE)

pca_df <- mnist_pca$x[, 1:2] %>%
  as_tibble() %>%
  bind_cols(mnist_small$label) %>%
  slice_sample(prop = .2)
```

```
New names:
* `` -> `...3`
```

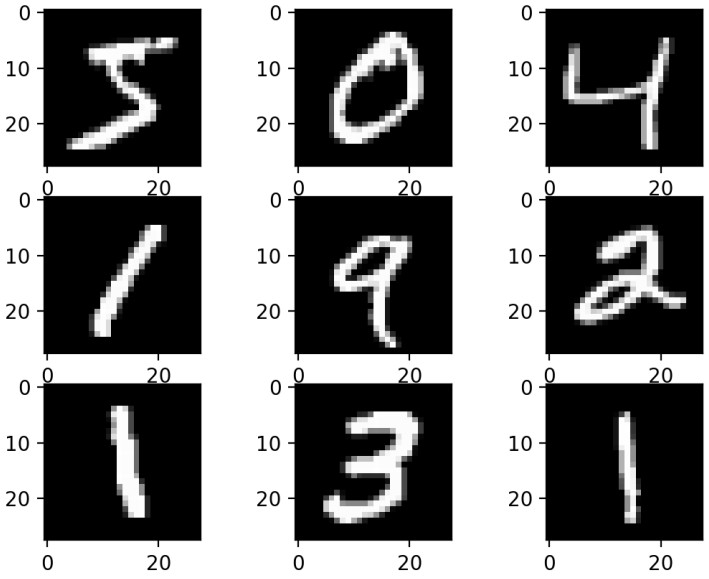
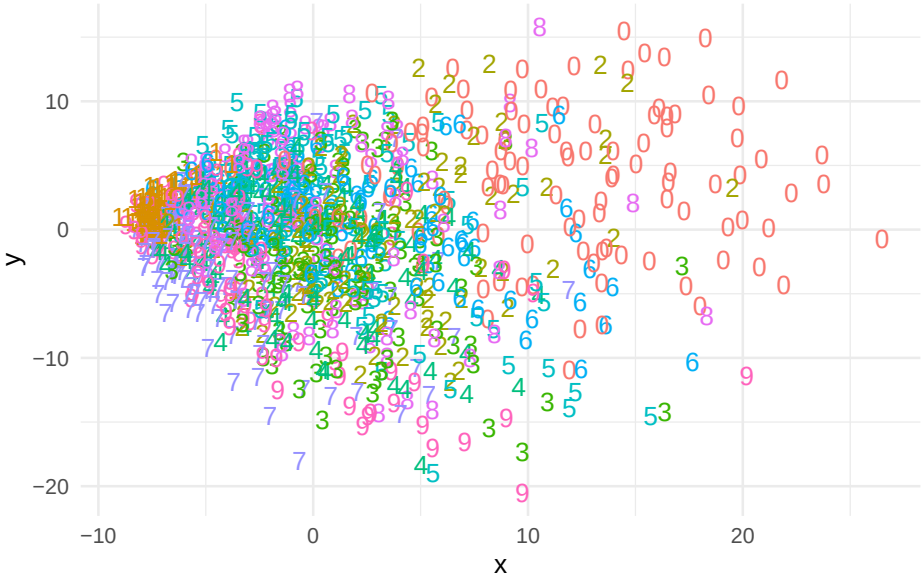


Рисунок 24.1: Пример рукописных цифр из набора данных MNIST

```
names(pca_df) <- c("x", "y", "label")
pca_df %>%
  ggplot() +
  geom_text(aes(x = x,
                y = y,
                label = label,
                colour = as.factor(label))) +
  guides(colour = "none") +
  theme_minimal()
```



Какое-то разделение произошло, но большая часть цифр находится в общей куче.

Теперь попробуем посмотреть, что сделает с теми же данными **t-SNE.** Для этого воспользуемся пакетом `{Rtsne}`.

```
install.packages("Rtsne")
```

t-SNE работает итеративно, перемещая точки на картинке, чтобы улучшить соответствие между исходными данными и их представлением на картинке. **t-SNE** – довольно ресурсоемкий алгоритм (особенно по сравнению с АГК), поэтому в функции `Rtsne()` есть множество настроек для контроля скорости ее работы. Например, можно выбрать максимальное количество итераций с помощью параметра `max_iter =`. Но самый главный

параметр – `dims =`, в котором нужно задать количество измерений, которое мы хотим получить. По умолчанию `dims = 2`, но можно поставить, например, `dims = 3`, чтобы получить точки в трехмерном пространстве.

```
library(Rtsne)
set.seed(42)
tsne <- mnist_small %>%
  select(!label) %>%
  Rtsne()
```

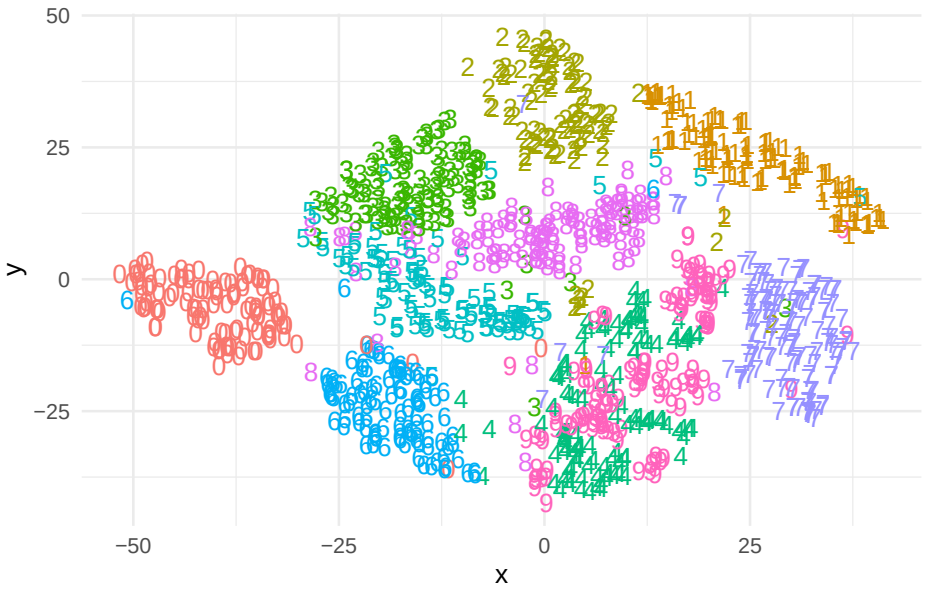
Теперь соединим результаты с исходными цифрами и визуализируем результаты:

```
tsne_df <- bind_cols(tsne$Y, mnist_small$label)
```

```
New names:
* `` -> `...1`
* `` -> `...2`
* `` -> `...3`
```

```
names(tsne_df) <- c("x", "y", "label")

tsne_df %>%
  slice_sample(prop = .2) %>%
  ggplot() +
  geom_text(aes(x = x,
                y = y,
                label = label,
                colour = as.factor(label))) +
  guides(colour = "none") +
  theme_minimal()
```



Как видите, **t-SNE** гораздо лучше справился с разделением цифр на группы: схожие цифры находятся рядом, непохожие – далеко. На получившемся графике заметно, что цифры 4 и 9 плохо разделись, но это довольно закономерно: эти цифры похожи по написанию.

Важно отметить, что **t-SNE** не сохраняет все абсолютные расстояния между объектами, поэтому важно рассматривать результат в контексте относительных расстояний. Также стоит помнить, что **t-SNE** не является методом для точного измерения расстояний или сравнения объектов, а скорее для визуального представления структуры данных.

24.4 Эксплораторный факторный анализ

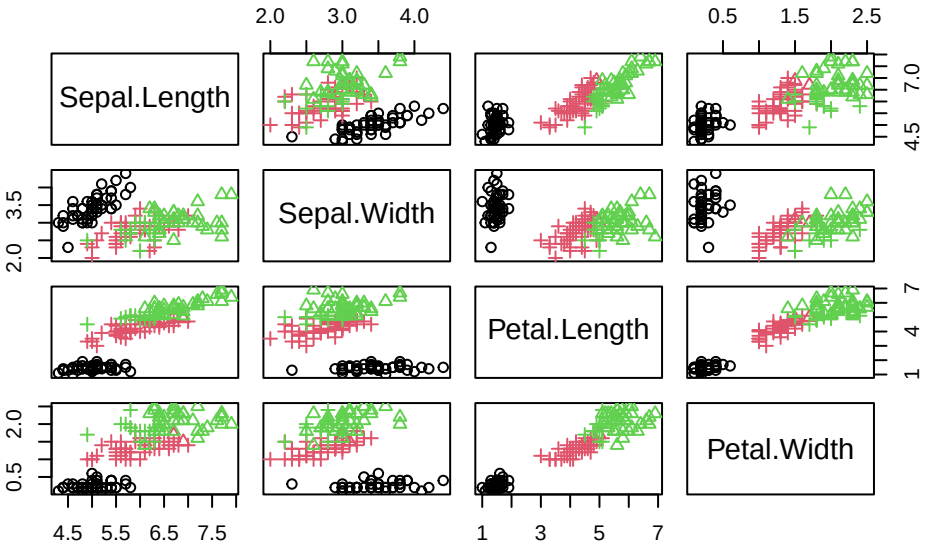
24.5 Конфирматорный факторный анализ

24.6 Кластерный анализ

```
iris_3means <- kmeans(iris %>% select(!Species), centers = 3)
table(iris$Species, iris_3means$cluster)
```

```
            1  2  3
setosa     50  0  0
versicolor  0  2 48
virginica   0 36 14
```

```
plot(iris %>% select(!Species), col = iris$Species, pch = iris_3means$cluster)
```



24.7 Многомерное шкалирование

24.8 Сетевой анализ

24.9 Другие методы многомерного анализа данных

Глава 25

Планирования научного исследования

25.1 Спорные исследовательские практики

Если Вы откроете какой-нибудь более-менее классический учебник по психологии, то увидите там много результатов исследований, которые в дальнейшем не удалось воспроизвести. Эта проблема накрыла академическое психологическое сообщество относительно недавно и стала, пожалуй, самой обсуждаемой темой среди ученых-психологов. Действительно, о чем еще говорить, если половина фактов твоей науки попросту неверна? ~~Об историческом смысле методологического кризиса, конечно же.~~ Получается, у теорий нет никакого подтверждения, а в плане знаний о психологических фактах мы находимся примерно там же, где и психологи, которые закупали метрономы и тахистоскопы почти полтора века назад.

Оказалось, что то, как устроена академическая наука, способствует публикации ложноположительных результатов. Если теоретическая гипотеза подтверждается, то это дает ученому больше плюшек, чем если гипотеза не подтверждается. Да и сами ученые как-то неосознанно хотят оказаться правыми. Ну а перепроверка предыдущих достижений в психологии оказалась не в почете: всем хочется быть первопроходцами и открывать новые неожиданные феномены, а не скрупулезно перепроверять чужие открытия. Все это привело психологию к тому, что в ней (да и не только в ней) закрепилось какое-то количество спорных исследовательских практик (questionable research practices). Спорные практики на то и спорные, что не все они такие уж плохие, многие ученые

даже считают, что в них нет ничего плохого. В отличие, например, от фальсификации данных - это, очевидно, совсем плохо.

Вот некоторые распространенные спорные исследовательские практики:

- *Выбор зависимых переменных пост-фактум.* По своей сути, это проблема скрытых множественных сравнений: если у нас много зависимых переменных, то можно сделать вид, что измерялось только то, на чем нашли эффект. Поэтому стоит задуматься, если возникает идея измерять все подряд на всякий случай - что конкретно предполагается обнаружить? Если конкретной гипотезы нет, то нужно так и написать, делая соответствующие поправки при анализе.

- *Обсуждение неожиданных результатов как ожидаемых.* Возможно, Вам это покажется смешным, но очень часто результаты оказываются статистически значимыми, но направлены в другую сторону, чем предполагалось в гипотезе. И в таких случаях исследователи часто пишут, как будто бы такие результаты и ожидались изначально! Приходится, правда, немного подкрутить теоретические построения.

- *Остановка набора испытуемых при достижении уровня значимости (Optional stopping).* Представьте себе, что Вы не обнаружили значимых результатов, но p-value болтается около 0.05. Тогда Вам захочется добрать парочку испытуемых. А потом еще. И еще немного. Проблема с таким подходом в том, что рано или поздно Вы получите статистически значимые результаты. В любом случае, даже если эффекта нет. На самом деле, эта проблема не так сильно влияет на результаты как может показаться, но это все-таки достаточно плохая практика, а ее применение увеличивает количество опубликованных ложно-положительных результатов.

- *Неправильное округление до .05.* Всякий раз, когда видите $p = .05$ будьте внимательны: вообще-то он не может быть равен именно *.05*. Либо автор не очень этого понимает, либо просто p больше, а не меньше *.05*. Например, *.054*, что потом округляется до *.05* и преподносится как статистически значимые результаты.

- *Использование односторонних тестов для сравнения средних.* Эта практика похожа на предыдущую: исследователь получил $p > .05$, но меньше, чем *.1*. Недобросовестный

исследователь, который хочет опубликоваться проводит односторонний т-тест вместо двустороннего, т.е. фактически просто делит p на 2. Совсем недобросовестные даже не пишут, что они использовали односторонний т-тест, хотя по умолчанию все используют двусторонний.

Данный список не претендует на полноту. Этих практик стоит избегать и других от этого отучать. Ну а если Вы думаете, что никто не заметит, если Вы так сделаете, то это не так.

Например, последние две практики можно легко обнаружить с помощью сайта http://statcheck.io. Этот сайт делает магию: нужно кинуть ему статью, он распознает в ней статистические тесты и пересчитывает их. На самом деле, ничего сложного, а это сайт сделан с помощью R и уже знакомых нам инструментов: с помощью специальных пакетов из файлов вытаскивается текст, в тексте с помощью регулярных выражений находятся паттерны вроде “$t(18) = -1.91$, $p = .036$” - в большинстве журналов используется очень похожее форматирование статистических результатов. Зная степени свободы и т-статистику, можно пересчитать p-value. В данном случае это можно посчитать вот так:

```
pt(-1.91, df = 18)*2 #       2,
```

```
[1] 0.07219987
```

Ну а дальше остается сравнить это с тем, что написали авторы. Например, бывает как в данном случае, что пересчитанный p-value в два раза больше того, что написали авторы. Это означает, скорее всего, что авторы использовали односторонний критерий. Если они об этом нигде не сказали, то это очень плохо.

25.2 Вопроизводимые исследования

Очевидно, что перечисленных практик стоит избегать. Однако недостаточно просто сказать “ребята, давайте вы не будете пытаться во что бы то ни стало искать неожиданные результаты и публиковать их”. Поэтому сейчас предлагаются потенциальные решения пробоемы воспроизводимости исследований, которые постепенно набирают все большую популярность. Самое распространенное решение - это использование пререгистраций. Все просто: исследователь планирует исследование, какую выборку он хочет собрать, как будет обрабатывать данные, какие

результаты он будет принимать как соответствующие гипотезе, а какие - нет. Получается что-то вроде научной статьи без результатов и их обсуждения, хотя можно и в более простом виде все представить: главное, есть доказательство, что исследование вы планировали провести именно так, а не подменяли все на ходу для красивых выводов. Другой способ добиться большей воспроизводимости результатов (и защиты от фальсификации) - это увеличение прозрачности в публикации данных и методов анализа. Существуют ученые (к счастью, такое встречается все реже), которые будут раздражаться, если их попросить дать вам данные. Мол, ну как же так, я их столько собирал, это же мои данные, а вот вы украдете у меня их и сделаете что-нибудь на них, опубликуете свою статью. Нет уж, мол, сами собирайте свои данные. Я терпел, и вы терпите. Очевидно, что наука - не забивание гвоздей, а ценность научной работы не обязательно пропорциональна количеству задействованных испытуемых. Собранные данные в некоторых случаях можно использовать в других исследованиях, а если все могут посмотреть исходные данные и проверить анализ, то это вообще круто и может защитить от ошибок.

Конечно, не все готовы к таким разворотам в исследовательской практике. Но лучше быть готовым, потому что в какой-то момент может оказаться, что новые практики станут обязательными. И тут R будет весьма кстати: можно выкладывать данные с RMarkdown документом, который будет сразу собирать из этого статью и графики. Данные со скриптами можно выкладывать на GitHub - это удобно для коллективной работы над проектом. Другой вариант - выкладывать данные и скрипты для анализа на сайте osf.io. Это сайт специально сделанный как платформа для публикации данных исследований и скриптов для них.

25.3 Статистическая мощность

Чтобы избежать *optional stopping*, нужно определять размер выборки заранее. Как это сделать? Наиболее корректный способ предполагает использование анализа статистической мощности (statistical power analysis). Для этого понадобится пакет `pwr`.

```
install.packages("pwr")
```

Этот пакет предоставляет семейство функций для расчета мощности, размера эффекта, уровня значимости или нужного размера выборки для разных статистических тестов.

Статистическая мощность - вероятность обнаружить статистически значимый эффект, если он действительно есть. Размер эффекта - собственно, размер эффекта в исследовании, обычно в универсальных единицах. Например, размер различия средних обычно измеряется в стандартных отклонениях. Это позволяет сравнивать эффекты в разных исследованиях и даже эффекты в разных областях науки.

Если задать 3 из 4 чисел (статистическая мощность - стандартно это .8, уровень значимости - стандартно .05, размер эффекта, размер выборки), то функция выдаст недостающее число.

Я очень советую поиграться с этим самостоятельно. Например, если размер эффекта в единицах стандартных отклонений Cohen's d = 2, то сколько нужно испытуемых, чтобы обнаружить эффект для двухвыборочного т-теста с вероятностью .8?

```
library(pwr)
pwr.t.test(d = 2, power = 0.8, type = "two.sample")
```

```
     Two-sample t test power calculation

              n = 5.089995
              d = 2
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

Всеего 6 в каждой группе (округляем в большую сторону)!

```
pwr.t.test(d = 2, power = 0.8, type = "paired")
```

```
     Paired t test power calculation

              n = 4.220726
              d = 2
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number of *pairs*
```

Еще меньше, если использовать within-subject дизайн исследования и зависимый т-тест.

А если эффект маленький - всего d = .2?

```
pwr.t.test(d = .2, power = 0.8, type = "two.sample")
```

```
     Two-sample t test power calculation

              n = 393.4057
              d = 0.2
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

394 испытуемых в каждой группе!

Можно проверить такие расчеты самостоятельно с помощью симуляции данных.

```
mean(replicate(1000, t.test(rnorm(394, 100, 15), rnorm(394, 103, 15))$p.value < 0.05
```

```
[1] 0.784
```

Действительно, если много раз делать случайные выборки из двух нормальных распределений с средними, отличающимися на 0.2 стандартных отклонения, то примерно в 80% случаев вы получите статистически значимые различия!

Часть V

Задачник

В этом разделе представлены задачи для закрепления изученного материала, а также их решения.

Этот раздел устроен следующим образом: в главе Глава 26 представлены задачи вместе с результатом, который должен получиться, но без кода – код вы должны написать самостоятельно! При возникновении сложностей или чтобы просто сравнить полученное решение с моим – смотрите главу Глава 27. В этой главе все то же самое, что и в Глава 26, но уже с кодом для решения задания. Исключение составляют некоторые задания по статистике, в Глава 27 есть дополнительные комментарии по ним.

Глава 26

Задания

Задания, которые помечены звездочкой (*) можно пропускать: это задания повышенной сложности, в них требуется подумать над решением, а не просто применить выученные инструменты.

26.1 Начало работы в R

- Разделите 9801 на 9.

```
[1] 1089
```

- Посчитайте логарифм от 2176782336 по основанию 6.

```
[1] 12
```

- Теперь натуральный логарифм 10 и умножьте его на 5.

```
[1] 11.51293
```

- С помощью функции `sin()` посчитайте $\sin(\pi)$, $\sin\left(\frac{\pi}{2}\right)$, $\sin\left(\frac{\pi}{6}\right)$.

  Значение π - зашитая в R константа (`pi`).

```
[1] 1.224647e-16
```

```
[1] 1
```

```
[1] 0.5
```

26.2 Создание векторов

- Создайте вектор из значений 2, 30 и 4000.

```
[1]    2   30 4000
```

- Создайте вектор от 1 до 20.

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Создайте вектор от 20 до 1.

```
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

Функция `sum()` возвращает сумму элементов вектора на входе. Посчитайте сумму первых 100 натуральных чисел (т.е. всех целых чисел от 1 до 100).

```
[1] 5050
```

- Создайте вектор от 1 до 20 и снова до 1. Число 20 должно присутствовать только один раз!

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18 17 16 15
[26] 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

- Создайте вектор значений 5, 4, 3, 2, 2, 3, 4, 5:

```
[1] 5 4 3 2 2 3 4 5
```

- Создайте вектор 2, 4, 6, … , 18, 20.

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

- Создайте вектор 0.1, 0.2, 0.3, …, 0.9, 1.

```
 [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- 2020 год – високосный. Следующий високосный год через 4 года – это будет 2024 год. Составьте календарь всех високосных годов XXI века, начиная с 2020 года.

2100 год относится к XXI веку, а не к XXII.

```
 [1] 2020 2024 2028 2032 2036 2040 2044 2048 2052 2056 2060 2064 2068 2072 2076
[16] 2080 2084 2088 2092 2096 2100
```

- Создайте вектор, состоящий из 20 повторений “Хэй!”.

```
 [1] "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !"
[11] "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !"
```

- Как я и говорил, многие функции, работающие с одним значением на входе, так же прекрасно работают и с целыми векторами. Попробуйте посчитать квадратный корень чисел от 1 до 10 с помощью функции `sqrt()` и сохраните результат в векторе `roots`. Выведите содержание вектора `roots` в консоль.

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278
```

- *Создайте вектор из одной единицы, двух двоек, трех троек, …. , девяти девяток.

```
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9
[39] 9 9 9 9 9 9 9
```

26.3 Приведение типов

- Сделайте вектор `vec1`, в котором соедините `3`, а также значения `"  "` и `"    "`.

```
[1] "3"       "  "    "    "
```

- Попробуйте вычесть `TRUE` из 10.

```
[1] 9
```

- Соедините значение `10` и `TRUE` в вектор `vec2`.

```
[1] 10  1
```

- Соедините вектор `vec2` и значение `"r"`:

```
[1] "10" "1"  "r"
```

- Соедините значения `10`, `TRUE`, `"r"` в вектор.

```
[1] "10"   "TRUE" "r"
```

26.4 Векторизация

- Создайте вектор `p`, состоящий из значений 4, 5, 6, 7, и вектор `q`, состоящий из 0, 1, 2, 3.

```
[1] 4 5 6 7
```

```
[1] 0 1 2 3
```

- Посчитайте поэлементную сумму векторов `p` и `q`:

```
[1]  4  6  8 10
```

- Посчитайте поэлементную разницу `p` и `q`:

```
[1] 4 4 4 4
```

- Поделите каждый элемент вектора `p` на соответствующий ему элемент вектора `q`:

  О, да, Вам нужно делить на 0!

```
[1]      Inf 5.000000 3.000000 2.333333
```

- Возведите каждый элемент вектора `p` в степень соответствующего ему элемента вектора `q`:

```
[1]   1   5  36 343
```

- Умножьте каждое значение вектора `p` на 10.

```
[1] 40 50 60 70
```

- Создайте вектор квадратов чисел от 1 до 10:

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

- Создайте вектор 0, 2, 0, 4, … , 18, 0, 20.

```
 [1]  0  2  0  4  0  6  0  8  0 10  0 12  0 14  0 16  0 18  0 20
```

- Создайте вектор 1, 0, 3, 0, 5, …, 17, 0, 19, 0.

```
 [1]  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19  0
```

- *Создайте вектор, в котором будут содержаться первые 20 степеней двойки.

```
 [1]       2       4       8      16      32      64     128     256     512
[10]    1024    2048    4096    8192   16384   32768   65536  131072  262144
[19]  524288 1048576
```

- *Создайте вектор из чисел 1, 10, 100, 1000, 10000:

```
[1]     1    10   100  1000 10000
```

- *Посчитать сумму последовательности $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \ldots + \frac{1}{50 \cdot 51}$.

```
[1] 0.9803922
```

- *Посчитать сумму последовательности $\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \frac{1}{2^{20}}$.

```
[1] 1.999999
```

- *Посчитать сумму последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$.

```
[1] 3.749174
```

- *Сколько чисел из последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$ больше чем 0.5?

```
[1] 3
```

26.5 Индексирование векторов

- Создайте вектор `troiki` со значениями 3, 6, 9, ..., 24, 27.

```
[1]  3  6  9 12 15 18 21 24 27
```

- Извлеките 2, 5 и 7 значения вектора `troiki`.

```
[1]  6 15 21
```

- Извлеките *предпоследнее* значение вектора `troiki`.

```
[1] 24
```

- Извлеките все значения вектора `troiki` *кроме* предпоследнего:

```
[1]  3  6  9 12 15 18 21 27
```

Создайте вектор `vec3`, скопировав следующий код:

```
vec3 <- c(3, 5, 2, 1, 8, 4, 9, 10, 3, 15, 1, 11)
```

- Найдите второй элемент вектора `vec3`.

```
[1] 5
```

- Верните второй и пятый элемент вектора `vec3`.

```
[1] 5 8
```

- Попробуйте извлечь сотое значение вектора `vec3`:

```
[1] NA
```

- Верните все элементы вектора `vec3` *кроме* второго элемента.

```
 [1]  3  2  1  8  4  9 10  3 15  1 11
```

- Верните все элементы вектора `vec3` *кроме* второго и пятого элемента.

```
 [1]  3  2  1  4  9 10  3 15  1 11
```

- Найдите последний элемент вектора `vec3`.

```
[1] 11
```

- Верните все значения вектора `vec3` кроме первого и последнего.

```
 [1]  5  2  1  8  4  9 10  3 15  1
```

- Найдите все значения вектора `vec3`, которые больше 4.

```
[1]  5  8  9 10 15 11
```

- Найдите все значения вектора `vec3`, которые больше 4, но меньше 10.

  Если хотите сделать это в одну строчку, то вам помогут логические операторы!

```
[1] 5 8 9
```

- Найдите все значения вектора `vec3`, которые меньше 4 или больше 10.

```
[1]  3  2  1  3 15  1 11
```

- Возведите в квадрат каждое значение вектора `vec3`.

```
 [1]   9  25   4   1  64  16  81 100   9 225   1 121
```

- *Возведите в квадрат каждое значение вектора на нечетной позиции и извлеките корень из каждого значения на четной позиции вектора `vec3`.

  Извлечение корня - это то же самое, что и возведение в степень 0.5.

```
 [1]  9.000000  2.236068  4.000000  1.000000 64.000000  2.000000 81.000000
 [8]  3.162278  9.000000  3.872983  1.000000  3.316625
```

- Создайте вектор 2, 4, 6, … , 18, 20 как минимум 2 новыми способами.

  Знаю, это задание может показаться бессмысленным, но это очень базовая операция, с помощью которой можно, например, разделить данные на две части. Чем больше способов Вы знаете, тем лучше!

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

26.6 Работа с пропущенными значениями

- Создайте вектор `vec4` со значениями 300, 15, 8, 2, 0, 1, 110, скопировав следующий код:

```
vec4 <- c(300, 15, 8, 20, 0, 1, 110)
vec4
```

```
[1] 300  15   8  20   0   1 110
```

- Замените все значения `vec4`, которые больше 20 на `NA`.

- Проверьте полученный вектор `vec4`:

```
[1] NA 15  8 20  0  1 NA
```

- Посчитайте сумму `vec4` с помощью функции `sum()`. Ответ `NA` не считается!

```
[1] 44
```

26.7 Матрицы

- Создайте матрицу размером *4x4*, состоящую из единиц. Назовите ее `M1`.

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    1    1    1
[3,]    1    1    1    1
[4,]    1    1    1    1
```

- Поменяйте все некрайние значения матрицы `M1` (то есть значения на позициях `[2,2]`, `[2,3]`, `[3,2]` и `[3,3]`) на число 2.

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    2    2    1
[3,]    1    2    2    1
[4,]    1    1    1    1
```

- Выделите второй и третий столбик из матрицы `M1`.

```
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    2    2
[4,]    1    1
```

- Сравните (==) вторую колонку и вторую строчку матрицы `M1`.

```
[1] TRUE TRUE TRUE TRUE
```

- Создайте матрицу `M2` из *пяти строк* и *шести столбцов*, в которой будут записаны значения *от 1 до 30* (сверху вниз, затем слева направо):

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    6   11   16   21   26
[2,]    2    7   12   17   22   27
[3,]    3    8   13   18   23   28
[4,]    4    9   14   19   24   29
[5,]    5   10   15   20   25   30
```

- Извлеките из матрицы `M2` все значения (в виде матрицы) *кроме значений из третьей строки и второго столбца.*

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   11   16   21   26
[2,]    2   12   17   22   27
[3,]    4   14   19   24   29
[4,]    5   15   20   25   30
```

- Умножьте каждое значение матрицы `M2` на 100.

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  100  600 1100 1600 2100 2600
[2,]  200  700 1200 1700 2200 2700
[3,]  300  800 1300 1800 2300 2800
[4,]  400  900 1400 1900 2400 2900
[5,]  500 1000 1500 2000 2500 3000
```

- Возведите каждое значение матрицы `M2` *в квадрат.*

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1   36  121  256  441  676
[2,]    4   49  144  289  484  729
[3,]    9   64  169  324  529  784
[4,]   16   81  196  361  576  841
[5,]   25  100  225  400  625  900
```

- Возведите каждое значение матрицы `M2` *в квадрат* и *вычтите* значения изначальной матрицы `M2`.

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0   30  110  240  420  650
[2,]    2   42  132  272  462  702
[3,]    6   56  156  306  506  756
[4,]   12   72  182  342  552  812
[5,]   20   90  210  380  600  870
```

- *Создайте таблицу умножения *(9x9)* в виде матрицы. Сохраните ее в переменную `mult_tab`.

  Вам может понадобиться функция `rep()`.

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    1    2    3    4    5    6    7    8    9
 [2,]    2    4    6    8   10   12   14   16   18
 [3,]    3    6    9   12   15   18   21   24   27
 [4,]    4    8   12   16   20   24   28   32   36
 [5,]    5   10   15   20   25   30   35   40   45
 [6,]    6   12   18   24   30   36   42   48   54
 [7,]    7   14   21   28   35   42   49   56   63
 [8,]    8   16   24   32   40   48   56   64   72
 [9,]    9   18   27   36   45   54   63   72   81
```

- Из матрицы `mult_tab` выделите подматрицу, включающую в себя только *строки с 6 по 8* и *столбцы с 3 по 7.*

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   18   24   30   36   42
[2,]   21   28   35   42   49
[3,]   24   32   40   48   56
```

- *Создайте матрицу с логическими значениями, где `TRUE`, если в этом месте в таблице умножения (`mult_tab`) двузначное число и `FALSE`, если однозначное.

  Матрица - это почти вектор. К нему можно обращаться с единственным индексом.

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
 [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [2,] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 [3,] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [4,] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [5,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [6,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [7,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [8,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [9,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- *Создайте матрицу `mult_tab2`, в которой все значения `tab` меньше 10 заменены на 0.

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    0    0    0    0    0    0    0    0    0
 [2,]    0    0    0    0   10   12   14   16   18
 [3,]    0    0    0   12   15   18   21   24   27
 [4,]    0    0   12   16   20   24   28   32   36
 [5,]    0   10   15   20   25   30   35   40   45
 [6,]    0   12   18   24   30   36   42   48   54
 [7,]    0   14   21   28   35   42   49   56   63
 [8,]    0   16   24   32   40   48   56   64   72
 [9,]    0   18   27   36   45   54   63   72   81
```

26.8 Списки

Дан список `list1`:

```
list1 = list(numbers = 1:5, letters = letters, logic = TRUE)
list1
```

```
$numbers
[1] 1 2 3 4 5

$letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

$logic
[1] TRUE
```

- Найдите первый элемент списка `list1`. Ответ должен быть списком длиной один.

```
$numbers
[1] 1 2 3 4 5
```

- Теперь найдите содержание первого элемента списка `list1` двумя разными способами. Ответ должен быть вектором.

```
[1] 1 2 3 4 5
```

```
[1] 1 2 3 4 5
```

- Теперь возьмите первый элемент содержания первого элемента списка `list1`. Ответ должен быть вектором.

```
[1] 1
```

- Создайте список `list2`, содержащий в себе два списка `list1`. Один из них будет иметь имя `pupa`, а другой – `lupa`.

```
$pupa
$pupa$numbers
[1] 1 2 3 4 5

$pupa$letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
$pupa$logic
[1] TRUE


$lupa
$lupa$numbers
[1] 1 2 3 4 5

$lupa$letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

$lupa$logic
[1] TRUE
```

- *Извлеките первый элемент списка `list2`, из него – второй подэлемент, а из него – третье значение.

```
[1] "c"
```

26.9 Датафрейм

- Запустите команду `data(mtcars)` чтобы загрузить встроенный датафрейм с информацией про автомобили. Каждая строчка датафрейма - модель автомобиля, каждая колонка - отдельная характеристика. Подробнее см. `?mtcars`.

```
data(mtcars)
mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
```

```
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Изучите структуру датафрейма `mtcars` с помощью функции `str()`.

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- Найдите значение третьей строчки четвертого столбца датафрейма `mtcars`.

```
[1] 93
```

- Извлеките первые шесть строчек и первые шесть столбцов датафрейма `mtcars`.

```
                   mpg cyl disp  hp drat    wt
Mazda RX4         21.0   6  160 110 3.90 2.620
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875
Datsun 710        22.8   4  108  93 3.85 2.320
Hornet 4 Drive    21.4   6  258 110 3.08 3.215
Hornet Sportabout 18.7   8  360 175 3.15 3.440
Valiant           18.1   6  225 105 2.76 3.460
```

- Извлеките колонку `wt` датафрейма `mtcars` - массу автомобиля в тысячах фунтов.

```
 [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070
[13] 3.730 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840
[25] 3.845 1.935 2.140 1.513 3.170 2.770 3.570 2.780
```

- Извлеките колонки из `mtcars` в следующем порядке: `hp`, `mpg`, `cyl`.

```
                    hp  mpg cyl
Mazda RX4          110 21.0   6
Mazda RX4 Wag      110 21.0   6
Datsun 710          93 22.8   4
Hornet 4 Drive     110 21.4   6
Hornet Sportabout  175 18.7   8
Valiant            105 18.1   6
Duster 360         245 14.3   8
Merc 240D           62 24.4   4
Merc 230            95 22.8   4
Merc 280           123 19.2   6
Merc 280C          123 17.8   6
Merc 450SE         180 16.4   8
Merc 450SL         180 17.3   8
Merc 450SLC        180 15.2   8
Cadillac Fleetwood 205 10.4   8
Lincoln Continental 215 10.4   8
Chrysler Imperial  230 14.7   8
Fiat 128            66 32.4   4
Honda Civic         52 30.4   4
Toyota Corolla      65 33.9   4
Toyota Corona       97 21.5   4
Dodge Challenger   150 15.5   8
AMC Javelin        150 15.2   8
Camaro Z28         245 13.3   8
Pontiac Firebird   175 19.2   8
Fiat X1-9           66 27.3   4
Porsche 914-2       91 26.0   4
```

```
Lotus Europa        113 30.4   4
Ford Pantera L      264 15.8   8
Ferrari Dino        175 19.7   6
Maserati Bora       335 15.0   8
Volvo 142E          109 21.4   4
```

- Посчитайте *количество* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
[1] 11
```

- Посчитайте *долю* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
[1] 0.34375
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) в датафрейме `mtcars`.

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) и 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
              mpg cyl  disp  hp drat    wt qsec vs am gear carb
Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Volvo 142E   21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

- Посчитайте максимальную массу (`wt`) автомобиля в выборке, воспользовавшись функцией `max()`:

```
[1] 5.424
```

- Посчитайте минимальную массу (`wt`) автомобиля в выборке, воспользовавшись функцией `min()`:

```
[1] 1.513
```

- Найдите строчку датафрейма `mtcars` с самым легким автомобилем.

```
              mpg cyl disp  hp drat    wt qsec vs am gear carb
Lotus Europa 30.4   4 95.1 113 3.77 1.513 16.9  1  1    5    2
```

- Извлеките строчки датафрейма `mtcars` с автомобилями, масса которых ниже средней массы.

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Масса автомобиля указана в тысячах фунтов. Создайте колонку `wt_kg` с массой автомобиля в килограммах. Результат округлите до целых значений с помощью функции `round()`.

  1 фунт = 0.45359237 кг.

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	wt_kg
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	1188
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	1304
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	1052
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	1458
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	1560
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	1569
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	1619
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	1447
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	1429
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	1560
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	1560
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	1846
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	1692
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	1715
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	2381
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	2460
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	2424
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	998
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	733
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	832
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	1118
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	1597
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	1558
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	1742
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	1744
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	878
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	971
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	686
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	1438
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6	1256
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	1619
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	1261

26.10 Условные конструкции

- Создайте вектор `vec5`, скопировав следующий код:

```
vec5 <- c(5, 20, 30, 0, 2, 9)
```

- Создайте новый строковый вектор, где на месте чисел больше 10 в `vec5` будет стоять “большое число”, а на месте остальных чисел – “маленькое число”.

```
[1] "         " "         "    "         "    "         "
[5] "         " "          "
```

- Загрузите файл heroes_information.csv в переменную `heroes`.

```
heroes <- read.csv("data/heroes_information.csv",
                   stringsAsFactors = FALSE,
                   na.strings = c("-", "-99"))
```

- Создайте новою колонку `hair` в `heroes`, в которой будет значение `"Bold"` для тех супергероев, у которых в колонке `Hair.color` стоит `"No Hair"`, и значение `"Hairy"` во всех остальных случаях.

```
  X          name Gender Eye.color              Race Hair.color Height
1 0        A-Bomb   Male    yellow             Human    No Hair    203
2 1    Abe Sapien   Male      blue     Icthyo Sapien    No Hair    191
3 2      Abin Sur   Male      blue           Ungaran    No Hair    185
4 3   Abomination   Male     green Human / Radiation    No Hair    203
5 4       Abraxas   Male      blue     Cosmic Entity      Black     NA
6 5 Absorbing Man   Male      blue             Human    No Hair    193
          Publisher Skin.color Alignment Weight  hair
1     Marvel Comics       <NA>      good    441  Bold
2 Dark Horse Comics       blue      good     65  Bold
3         DC Comics        red      good     90  Bold
4     Marvel Comics       <NA>       bad    441  Bold
5     Marvel Comics       <NA>       bad     NA Hairy
6     Marvel Comics       <NA>       bad    122  Bold
```

- Создайте новою колонку `tall` в `heroes`, в которой будет значение `"tall"` для тех супергероев, у которых в колонке `Height` стоит число больше 190, значение `"short"` для тех супергероев, у которых в колонке `Height` стоит число меньше 170, и значение `"middle"` во всех остальных случаях.

26.11 Создание функций

- Создайте функцию `plus_one()`, которая принимает число и возвращает это же число + 1.
- Проверьте функцию `plus_one()` на числе 41.

```
plus_one(41)
```

```
[1] 42
```

- Создайте функцию `circle_area()`, которая вычисляет площадь круга по радиусу согласно формуле πr^2.
- Посчитайте площадь круга с радиусом 5.

```
[1] 78.53982
```

- Создайте функцию `cels2fahr()`, которая будет превращать градусы по Цельсию в градусы по Фаренгейту.
- Проверьте на значениях -100, -40 и 0, что функция `cels2fahr()` работает корректно.

```
cels2fahr(c(-100, -40, 0))
```

```
[1] -148  -40   32
```

- Напишите функцию `highlight()`, которая принимает на входе строковый вектор, а возвращает тот же вектор, но дополненный значением `"***"` в начале и конце вектора. Лучше всего это рассмотреть на примере:

```
highlight(c(" ", "   !"))
```

```
[1] "***"     " "       "   !" "***"
```

- Теперь сделайте функцию `highlight` более гибкой. Добавьте в нее параметр `wrapper =`, который по умолчанию равен `"***"`. Значение параметра `wrapper =` и будет вставлено в начало и конец вектора.
- Проверьте написанную функцию на векторе `c(" ", "   !")`.

```
highlight(c(" ", "   !"))
```

```
[1] "***"     " "       "   !" "***"
```

```
highlight(c(" ", "   !"), wrapper = "__")
```

```
[1] "__"      " "       "   !" "__"
```

- Создайте функцию `na_n()`, которая будет возвращать количество `NA` в векторе.

- Проверьте функцию `na_n()` на векторе:

```
na_n(c(NA, 3:5, NA, 2, NA))
```

```
[1] 3
```

- Напишите функцию `factors()`, которая будет возвращать все делители числа в виде числового вектора.

  Здесь может понадобиться оператор для получения остатка от деления: `%%`.

- Проверьте функцию `factors()` на простых и сложных числах:

```
factors(3)
```

```
[1] 1 3
```

```
factors(161)
```

```
[1]   1   7  23 161
```

```
factors(1984)
```

```
 [1]    1    2    4    8   16   31   32   62   64  124  248  496  992 1984
```

- *Напишите функцию `is_prime()`, которая проверяет, является ли число простым.

  Здесь может пригодиться функция `any()` - она возвращает `TRUE`, если в векторе есть хотя бы один `TRUE`.

- Проверьте какие года были для нас простыми, а какие нет:

```
is_prime(2017)
```

```
[1] TRUE
```

```
is_prime(2019)
```

```
[1] FALSE
```

```
2019/3 #2019        3
```

```
[1] 673
```

```
is_prime(2020)
```

```
[1] FALSE
```

```
is_prime(2021)
```

```
[1] FALSE
```

- *Создайте функцию `monotonic()`, которая возвращает `TRUE`, если значения в векторе не убывают (то есть каждое следующее - больше или равно предыдущему) или не возврастают.

  Полезная функция для этого – `diff()` – возвращает разницу соседних значений.

```
monotonic(1:7)
```

```
[1] TRUE
```

```
monotonic(c(1:5,5:1))
```

```
[1] FALSE
```

```
monotonic(6:-1)
```

```
[1] TRUE
```

```
monotonic(c(1:5, rep(5, 10), 5:10))
```

```
[1] TRUE
```

Бинарные операторы типа + или %in% тоже представляют собой функции. Более того, мы можем создавать свои бинарные операторы! В этом нет особой сложности – нужно все так же создавать функцию (для двух переменных), главное окружать их % и название обрамлять обратными штрихами `. Например, можно сделать свой бинарный оператор %notin%, который будет выдавать TRUE, если значения слева *нет* в векторе справа:

```
`%notin%` <- function(x, y) ! (x %in% y)
1:10 %notin% c(1, 4, 5)
```

```
 [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- *Создайте бинарный оператор %without%, который будет возвращать все значения вектора слева без значений вектора справа.

```
c(" ", " ", " ", "   ", " ", "   ") %without% c(" ", " ")
```

```
[1] " "     "   " " " "     "   "
```

- *Создайте бинарный оператор %between%, который будет возвращать TRUE, если значение в векторе слева накходится в *диапазоне* значений вектора справа:

```
1:10 %between% c(1, 4, 5)
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

26.12 Проверка на адекватность

- Создайте функцию `trim()`, которая будет возвращать вектор без первого и последнего значения (вне зависимости от типа данных).
- Проверьте, что функция `trim()` работает корректно:

```
trim(1:7)
```

```
[1] 2 3 4 5 6
```

```
trim(letters)
```

```
 [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[20] "u" "v" "w" "x" "y"
```

- Теперь добавьте в функцию `trim()` параметр `n =` со значением по умолчанию 1. Этот параметр будет обозначать сколько значений нужно отрезать слева и справа от вектора.
- Проверьте полученную функцию:

```
trim(letters)
```

```
 [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[20] "u" "v" "w" "x" "y"
```

```
trim(letters, n = 2)
```

```
 [1] "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u"
[20] "v" "w" "x"
```

- Сделайте так, чтобы функция `trim()` работала корректно с `n = 0`, т.е. функция возвращала бы исходный вектор без изменений.

```
trim(letters, n = 0)
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

- *Теперь добавьте проверку на адекватность входных данных: функция `trim()` должна выдавать ошибку, если `n` = меньше нуля или если `n` = слишком большой и отрезает все значения вектора:

- *Проверьте полученную функцию `trim()`:

```
trim(1:6, 3)
```

```
Error in trim(1:6, 3): n          !
```

```
trim(1:6, -1)
```

```
Error in trim(1:6, -1): n                !
```

26.13 Семейство функций apply()

- Создайте матрицу `M2`:

```
M2 <- matrix(c(20:11, 11:20), nrow = 5)
M2
```

```
     [,1] [,2] [,3] [,4]
[1,]   20   15   11   16
[2,]   19   14   12   17
[3,]   18   13   13   18
[4,]   17   12   14   19
[5,]   16   11   15   20
```

- Посчитайте максимальное значение матрицы `M2` по каждой строчке.

```
[1] 20 19 18 19 20
```

- Посчитайте максимальное значение матрицы `M2` по каждому столбцу.

```
[1] 20 15 15 20
```

- Посчитайте среднее значение матрицы `M2` по каждой строке.

```
[1] 15.5 15.5 15.5 15.5 15.5
```

- Посчитайте среднее значение матрицы `M2` по каждому столбцу.

```
[1] 18 13 13 18
```

- Создайте список `list3`:

```
list3 <- list(
  a = 1:5,
  b = 0:20,
  c = 4:24,
  d = 6:3,
  e = 6:25
  )
```

- Найдите максимальное значение каждого вектора списка `list3`.

```
a  b  c  d  e
5 20 24  6 25
```

- Посчитайте сумму каждого вектора списка `list3`.

```
 a   b   c   d   e
15 210 294  18 310
```

- Посчитайте длину каждого вектора списка `list3`.

```
a  b  c  d  e
5 21 21  4 20
```

- Напишите функцию `max_item()`, которая будет принимать на входе список, а возвращать - (первый) самый длинный его элемент.

  Для этого вам может понадобиться функция `which.max()`, которая возвращает индекс максимального значения (первого, если их несколько).

- Проверьте функцию `max_item()` на списке `list3`.

```
max_item(list3)
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Теперь мы сделаем сложный список `list4`:

```
list4 <- list(1:3, 3:40, list3)
```

- Посчитайте длину каждого вектора в списке, в т.ч. для списка внутри. Результат должен быть списком с такой же структорой, как и изначальный список `list4`.

  Для этого может понадобиться функция `rapply()`: **recursive lapply**

```
[[1]]
[1] 3

[[2]]
[1] 38

[[3]]
[[3]]$a
[1] 5

[[3]]$b
[1] 21

[[3]]$c
[1] 21
```

```
[[3]]$d
[1] 4

[[3]]$e
[1] 20
```

- *Загрузите набор данных `heroes` и посчитайте, сколько `NA` в каждом из столбцов.

  Для этого удобно использовать ранее написанную функцию `na_n()`.

```
         X       name     Gender  Eye.color       Race Hair.color     Height
         0          0         29        172        304        172        217
Publisher Skin.color  Alignment     Weight       hair       tall
         0        662          7        239        172        217
```

- *Используя ранее написанную функцию `is_prime()`, напишите функцию `prime_numbers()`, которая будет возвращать все простые числа до выбранного числа.

```
prime_numbers(200)
```

```
 [1]   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71
[20]  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167
[39] 173 179 181 191 193 197 199
```

26.14 magrittr::%>%

- Перепишите следующие выражения, используя %>%:

```
sqrt(sum(1:10))
```

```
[1] 7.416198
```

```
[1] 7.416198
```

```
abs(min(-5:5))
```

```
[1] 5
```

```
[1] 5
```

```
  c("     ", 2, "   ", sqrt(2))
```

```
[1] "     "       "2"               "   "           "1.4142135623731"
```

```
[1] "     "       "2"               "   "           "1.4142135623731"
```

```
  B <- matrix(10:39, nrow = 5)
  apply(B, 1, mean)
```

```
[1] 22.5 23.5 24.5 25.5 26.5
```

26.15 Выбор столбцов: `dplyr::select()`

Для выполнения следующих заданий нам понадобятся датасеты `heroes` и `powers`, которые можно загрузить, используя следующие команды:

```
  library(tidyverse)
  heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/heroes
                     na = c("-", "-99"))
  powers <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/data/super_
```

- Выберете первые 4 столбца в `powers`.

```
# A tibble: 667 x 4
   hero_names    Agility `Accelerated Healing` `Lantern Power Ring`
   <chr>         <lgl>   <lgl>                 <lgl>
 1 3-D Man       TRUE    FALSE                 FALSE
 2 A-Bomb        FALSE   TRUE                  FALSE
 3 Abe Sapien    TRUE    TRUE                  FALSE
 4 Abin Sur      FALSE   FALSE                 TRUE
 5 Abomination   FALSE   TRUE                  FALSE
 6 Abraxas       FALSE   FALSE                 FALSE
 7 Absorbing Man FALSE   FALSE                 FALSE
 8 Adam Monroe   FALSE   TRUE                  FALSE
 9 Adam Strange  FALSE   FALSE                 FALSE
10 Agent Bob     FALSE   FALSE                 FALSE
# i 657 more rows
```

- Выберите все столбцы от `Reflexes` до `Empathy` в тиббле `powers`:

```
# A tibble: 667 x 7
   Reflexes Invulnerability `Energy Constructs` `Force Fields` `Self-Sustenance`
   <lgl>    <lgl>           <lgl>               <lgl>          <lgl>
 1 FALSE    FALSE           FALSE               FALSE          FALSE
 2 FALSE    FALSE           FALSE               FALSE          TRUE
 3 TRUE     FALSE           FALSE               FALSE          FALSE
 4 FALSE    FALSE           FALSE               FALSE          FALSE
 5 FALSE    TRUE            FALSE               FALSE          FALSE
 6 FALSE    TRUE            FALSE               FALSE          FALSE
 7 FALSE    TRUE            FALSE               FALSE          FALSE
 8 FALSE    FALSE           FALSE               FALSE          FALSE
 9 FALSE    FALSE           FALSE               FALSE          FALSE
10 FALSE    FALSE           FALSE               FALSE          FALSE
# i 657 more rows
# i 2 more variables: `Anti-Gravity` <lgl>, Empathy <lgl>
```

- Выберите все столбцы тиббла `powers` кроме первого (`hero_names`):

```
# A tibble: 667 x 167
   Agility `Accelerated Healing` `Lantern Power Ring` `Dimensional Awareness`
   <lgl>   <lgl>                 <lgl>                <lgl>
 1 TRUE    FALSE                 FALSE                FALSE
 2 FALSE   TRUE                  FALSE                FALSE
 3 TRUE    TRUE                  FALSE                FALSE
 4 FALSE   FALSE                 TRUE                 FALSE
 5 FALSE   TRUE                  FALSE                FALSE
 6 FALSE   FALSE                 FALSE                TRUE
 7 FALSE   FALSE                 FALSE                FALSE
 8 FALSE   TRUE                  FALSE                FALSE
 9 FALSE   FALSE                 FALSE                FALSE
10 FALSE   FALSE                 FALSE                FALSE
# i 657 more rows
# i 163 more variables: `Cold Resistance` <lgl>, Durability <lgl>,
#   Stealth <lgl>, `Energy Absorption` <lgl>, Flight <lgl>,
#   `Danger Sense` <lgl>, `Underwater breathing` <lgl>, Marksmanship <lgl>,
#   `Weapons Master` <lgl>, `Power Augmentation` <lgl>,
#   `Animal Attributes` <lgl>, Longevity <lgl>, Intelligence <lgl>,
#   `Super Strength` <lgl>, Cryokinesis <lgl>, Telepathy <lgl>, ...
```

26.16 Выбор строк: `dplyr::slice()` и `dplyr::filter()`

- Выберите только те строчки, в которых содержится информация о супергероях тяжелее 500 кг.

```
# A tibble: 6 x 11
   ...1 name       Gender `Eye color` Race         `Hair color` Height Publisher
  <dbl> <chr>      <chr>  <chr>       <chr>        <chr>         <dbl> <chr>
1   203 Darkseid   Male   red         New God      No Hair       267   DC Comics
2   283 Giganta    Female green       <NA>         Red            62.5 DC Comics
3   331 Hulk       Male   green       Human / Rad~ Green         244   Marvel C~
4   373 Juggernaut Male   blue        Human        Red           287   Marvel C~
5   549 Red Hulk   Male   yellow      Human / Rad~ Black         213   Marvel C~
6   575 Sasquatch  Male   red         <NA>         Orange        305   Marvel C~
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о *женщинах*-супергероях тяжелее 500 кг.

```
# A tibble: 1 x 11
   ...1 name    Gender `Eye color` Race  `Hair color` Height Publisher
  <dbl> <chr>   <chr>  <chr>       <chr> <chr>         <dbl> <chr>
1   283 Giganta Female green       <NA>  Red            62.5 DC Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о супергероях человеческой расы (`"Human"`) женского пола. Из этих супергероев возьмите первые 5.

```
# A tibble: 5 x 11
   ...1 name         Gender `Eye color` Race  `Hair color` Height Publisher
  <dbl> <chr>        <chr>  <chr>       <chr> <chr>         <dbl> <chr>
1    38 Arachne      Female blue        Human Blond           175 Marvel Comics
2    63 Batgirl      Female green       Human Red             170 DC Comics
3    65 Batgirl IV   Female green       Human Black           165 DC Comics
4    72 Batwoman V   Female green       Human Red             178 DC Comics
5    96 Black Canary Female blue        Human Blond           165 DC Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

26.17 Сортировка строк: `dplyr::arrange()`

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по возрастанию* `Height`.

```
# A tibble: 734 x 3
   name            Gender Height
   <chr>           <chr>   <dbl>
```

```
 1 Utgard-Loki     Male     15.2
 2 Bloodwraith     Male     30.5
 3 King Kong       Male     30.5
 4 Anti-Monitor    Male     61
 5 Giganta         Female   62.5
 6 Krypto          Male     64
 7 Yoda            Male     66
 8 Jack-Jack       Male     71
 9 Howard the Duck Male     79
10 Godzilla        <NA>    108
# i 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по убыванию* `Height`.

```
# A tibble: 734 x 3
   name          Gender Height
   <chr>         <chr>   <dbl>
 1 Fin Fang Foom Male      975
 2 Galactus      Male      876
 3 Groot         Male      701
 4 MODOK         Male      366
 5 Wolfsbane     Female    366
 6 Onslaught     Male      305
 7 Sasquatch     Male      305
 8 Ymir          Male      305.
 9 Rey           Female    297
10 Juggernaut    Male      287
# i 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки сначала по `Gender`, затем *по убыванию* `Height`.

```
# A tibble: 734 x 3
   name      Gender Height
   <chr>     <chr>   <dbl>
 1 Wolfsbane Female    366
 2 Rey       Female    297
 3 Bloodaxe  Female    218
 4 Thundra   Female    218
 5 Hela      Female    213
 6 Frenzy    Female    211
 7 She-Hulk  Female    201
 8 Ardina    Female    193
```

```
 9 Starfire  Female    193
10 Valkyrie  Female    191
# i 724 more rows
```

26.18 Уникальные значения: `dplyr::distinct()`

- Извлеките уникальные значения столбца `Eye color` из тиббла `heroes`.

```
# A tibble: 23 x 1
   `Eye color`
   <chr>
 1 yellow
 2 blue
 3 green
 4 brown
 5 <NA>
 6 red
 7 violet
 8 white
 9 purple
10 black
# i 13 more rows
```

- Извлеките уникальные значения столбца `Hair color` из тиббла `heroes`.

```
# A tibble: 30 x 1
   `Hair color`
   <chr>
 1 No Hair
 2 Black
 3 Blond
 4 Brown
 5 <NA>
 6 White
 7 Purple
 8 Orange
 9 Pink
10 Red
# i 20 more rows
```

26.19 Создание колонок: dplyr::mutate() и dplyr::transmute()

- Создайте колонку `height_m` с ростом супергероев в метрах, затем выберите только колонки `name` и `height_m`.

```
# A tibble: 734 x 2
   name          height_m
   <chr>            <dbl>
 1 A-Bomb            2.03
 2 Abe Sapien        1.91
 3 Abin Sur          1.85
 4 Abomination       2.03
 5 Abraxas          NA
 6 Absorbing Man     1.93
 7 Adam Monroe      NA
 8 Adam Strange      1.85
 9 Agent 13          1.73
10 Agent Bob         1.78
# i 724 more rows
```

- Создайте новою колонку `hair` в heroes, в которой будет значение `"Bold"` для тех супергероев, у которых в колонке `Hair.color` стоит `"No Hair"`, и значение `"Hairy"` во всех остальных случаях. Затем выберите только колонки `name`, `Hair color`, `hair`.

```
# A tibble: 734 x 3
   name          `Hair color` hair
   <chr>         <chr>        <chr>
 1 A-Bomb        No Hair      Bold
 2 Abe Sapien    No Hair      Bold
 3 Abin Sur      No Hair      Bold
 4 Abomination   No Hair      Bold
 5 Abraxas       Black        Hairy
 6 Absorbing Man No Hair      Bold
 7 Adam Monroe   Blond        Hairy
 8 Adam Strange  Blond        Hairy
 9 Agent 13      Blond        Hairy
10 Agent Bob     Brown        Hairy
# i 724 more rows
```

26.20 Агрегация: dplyr::group_by() %>% summarise()

- Посчитайте количество супергероев по расам и отсортируйте по убыванию. Извлеките первые 5 строк.

```
# A tibble: 5 x 2
  Race              n
  <chr>         <int>
1 <NA>            304
2 Human           208
3 Mutant           63
4 God / Eternal    14
5 Cyborg           11
```

- Посчитайте средний рост по полу.

```
# A tibble: 3 x 2
  Gender height_mean
  <chr>        <dbl>
1 Female        175.
2 Male          192.
3 <NA>          177.
```

26.21 Соединение датафреймов: *_join

- Создайте тиббл `web_creators`, в котором будут супергерои, которые могут плести паутину, т.е. у них стоит `TRUE` в колонке `Web Creation` в тиббле `powers`.

```
# A tibble: 16 x 12
    ...1 name             Gender `Eye color` Race  `Hair color` Height Publisher
   <dbl> <chr>            <chr>  <chr>       <chr> <chr>         <dbl> <chr>
 1    33 Anti-Venom       Male   blue        Symb~ Blond           229 Marvel C~
 2    38 Arachne          Female blue        Human Blond           175 Marvel C~
 3   161 Carnage          Male   green       Symb~ Red             185 Marvel C~
 4   335 Hybrid           Male   brown       Symb~ Black           175 Marvel C~
 5   479 Mysterio         Male   brown       Human No Hair         180 Marvel C~
 6   580 Scarlet Spider ~ Male   brown       Clone Brown           193 Marvel C~
 7   597 Silk             Female brown       Human Black            NA Marvel C~
 8   620 Spider-Girl      Female blue        Human Brown           170 Marvel C~
 9   621 Spider-Gwen      Female blue        Human Blond           165 Marvel C~
10   622 Spider-Man       Male   hazel       Human Brown           178 Marvel C~
```

```
11   623 Spider-Man       <NA>   red         Human Brown           178 Marvel C~
12   624 Spider-Man       Male   brown       Human Black           157 Marvel C~
13   673 Toxin            Male   blue        Symb~ Brown           188 Marvel C~
14   674 Toxin            Male   black       Symb~ Blond           191 Marvel C~
15   689 Venom            Male   blue        Symb~ Strawberry ~    191 Marvel C~
16   692 Venompool        Male   <NA>        Symb~ <NA>            226 Marvel C~
# i 4 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>,
#   `Web Creation` <lgl>
```

- Найдите всех супергероев, которые присутствуют в `heroes`, но отсутствуют в `powers`. Ответом должен быть строковый вектор с именами супергероев.

```
 [1] "Agent 13"          "Alfred Pennyworth" "Arsenal"
 [4] "Batgirl III"       "Batgirl V"         "Beetle"
 [7] "Black Goliath"     "Black Widow II"    "Blaquesmith"
[10] "Bolt"              "Boomer"            "Box"
[13] "Box III"           "Captain Mar-vell"  "Cat II"
[16] "Cecilia Reyes"     "Clea"              "Clock King"
[19] "Colin Wagner"      "Colossal Boy"      "Corsair"
[22] "Cypher"            "Danny Cooper"      "Darkside"
[25] "ERG-1"             "Fixer"             "Franklin Storm"
[28] "Giant-Man"         "Giant-Man II"      "Goliath"
[31] "Goliath"           "Goliath"           "Guardian"
[34] "Hawkwoman"         "Hawkwoman II"      "Hawkwoman III"
[37] "Howard the Duck"   "Jack Bauer"        "Jesse Quick"
[40] "Jessica Sanders"   "Jigsaw"            "Jyn Erso"
[43] "Kid Flash II"      "Kingpin"           "Meteorite"
[46] "Mister Zsasz"      "Mogo"              "Moloch"
[49] "Morph"             "Nite Owl II"       "Omega Red"
[52] "Paul Blart"        "Penance"           "Penance I"
[55] "Plastic Lad"       "Power Man"         "Renata Soliz"
[58] "Ronin"             "Shrinking Violet"  "Snake-Eyes"
[61] "Spider-Carnage"    "Spider-Woman II"   "Stacy X"
[64] "Thunderbird II"    "Two-Face"          "Vagabond"
[67] "Vision II"         "Vulcan"            "Warbird"
[70] "White Queen"       "Wiz Kid"           "Wondra"
[73] "Wyatt Wingfoot"    "Yellow Claw"
```

- Найдите всех супергероев, которые присутствуют в `powers`, но отсутствуют в `heroes`. Ответом должен быть строковый вектор с именами супергероев.

```
 [1] "3-D Man"           "Bananaman"         "Bizarro-Girl"
 [4] "Black Vulcan"      "Blue Streak"       "Bradley"
```

```
 [7] "Clayface"          "Concrete"          "Dementor"
[10] "Doctor Poison"     "Fire"              "Hellgramite"
[13] "Lara Croft"        "Little Epic"       "Lord Voldemort"
[16] "Orion"             "Peek-a-Boo"        "Queen Hippolyta"
[19] "Reactron"          "SHDB"              "Stretch Armstrong"
[22] "TEST"              "Tommy Clarke"      "Tyrant"
```

26.22 Tidy data

- Для начала создайте тиббл `heroes_weight`, скопировав код:

```
heroes_weight <- heroes %>%
  filter(Publisher %in% c("DC Comics", "Marvel Comics")) %>%
  group_by(Gender, Publisher) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE)) %>%
  drop_na()
heroes_weight
```

```
# A tibble: 4 x 3
# Groups:   Gender [2]
  Gender Publisher     weight_mean
  <chr>  <chr>               <dbl>
1 Female DC Comics            76.8
2 Female Marvel Comics        80.1
3 Male   DC Comics           113.
4 Male   Marvel Comics       134.
```

Функция `drop_na()` позволяет выбросить все строчки, в которых встречается `NA`.

- Превратите тиббл `heroes_weight` в широкий тиббл:

```
# A tibble: 2 x 3
# Groups:   Gender [2]
  Gender `DC Comics` `Marvel Comics`
  <chr>        <dbl>           <dbl>
1 Female        76.8            80.1
2 Male         113.            134.
```

- Затем превратите его обратно в длинный тиббл:

```
# A tibble: 4 x 3
# Groups:   Gender [2]
  Gender Publisher     weight_mean
  <chr>  <chr>               <dbl>
1 Female DC Comics            76.8
2 Female Marvel Comics        80.1
3 Male   DC Comics           113.
4 Male   Marvel Comics       134.
```

- Сделайте `powers` длинным тибблом с тремя колонками: `hero_names`, `power` (названгие суперсилы) и `has` (наличие суперсилы у данного супергероя).

```
# A tibble: 111,389 x 3
   hero_names power                has
   <chr>      <chr>                <lgl>
 1 3-D Man    Agility              TRUE
 2 3-D Man    Accelerated Healing  FALSE
 3 3-D Man    Lantern Power Ring   FALSE
 4 3-D Man    Dimensional Awareness FALSE
 5 3-D Man    Cold Resistance      FALSE
 6 3-D Man    Durability           FALSE
 7 3-D Man    Stealth              FALSE
 8 3-D Man    Energy Absorption    FALSE
 9 3-D Man    Flight               FALSE
10 3-D Man    Danger Sense         FALSE
# i 111,379 more rows
```

- Сделайте тиббл `powers` обратно широким, но с новой структурой: каждая строчка означает суперсилу, а каждая колонка - супергероя (за исключением первой колонки - названия суперсилы).

```
# A tibble: 167 x 668
   power          `3-D Man` `A-Bomb` `Abe Sapien` `Abin Sur` Abomination Abraxas
   <chr>          <lgl>     <lgl>    <lgl>        <lgl>      <lgl>       <lgl>
 1 Agility        TRUE      FALSE    TRUE         FALSE      FALSE       FALSE
 2 Accelerated H~ FALSE     TRUE     TRUE         FALSE      TRUE        FALSE
 3 Lantern Power~ FALSE     FALSE    FALSE        TRUE       FALSE       FALSE
 4 Dimensional A~ FALSE     FALSE    FALSE        FALSE      FALSE       TRUE
 5 Cold Resistan~ FALSE     FALSE    TRUE         FALSE      FALSE       FALSE
 6 Durability     FALSE     TRUE     TRUE         FALSE      FALSE       FALSE
 7 Stealth        FALSE     FALSE    FALSE        FALSE      FALSE       FALSE
 8 Energy Absorp~ FALSE     FALSE    FALSE        FALSE      FALSE       FALSE
 9 Flight         FALSE     FALSE    FALSE        FALSE      FALSE       TRUE
```

```
10 Danger Sense   FALSE     FALSE    FALSE        FALSE      FALSE       FALSE
# i 157 more rows
# i 661 more variables: `Absorbing Man` <lgl>, `Adam Monroe` <lgl>,
#   `Adam Strange` <lgl>, `Agent Bob` <lgl>, `Agent Zero` <lgl>,
#   `Air-Walker` <lgl>, Ajax <lgl>, `Alan Scott` <lgl>, `Alex Mercer` <lgl>,
#   `Alex Woolsly` <lgl>, Alien <lgl>, `Allan Quatermain` <lgl>, Amazo <lgl>,
#   Ammo <lgl>, `Ando Masahashi` <lgl>, Angel <lgl>, `Angel Dust` <lgl>,
#   `Angel Salvadore` <lgl>, Angela <lgl>, `Animal Man` <lgl>, ...
```

26.23 Операции с несколькими колонками: across()

- Посчитайте количество NA в каждой колонке, группируя по полу (Gender).

```
# A tibble: 3 x 11
  Gender  ...1  name `Eye color`  Race `Hair color` Height Publisher
  <chr>  <int> <int>       <int> <int>        <int>  <int>     <int>
1 Female     0     0          41    98           38     56         0
2 Male       0     0         121   184          123    147         0
3 <NA>       0     0          10    22           11     14         0
# i 3 more variables: `Skin color` <int>, Alignment <int>, Weight <int>
```

- Посчитайте количество NA в каждой колонке, которая заканчивается на "color", группируя по полу (Gender).

```
# A tibble: 3 x 4
  Gender `Eye color` `Hair color` `Skin color`
  <chr>        <int>        <int>        <int>
1 Female          41           38          186
2 Male           121          123          449
3 <NA>            10           11           27
```

- Найдите (первую) самую длинную строчку для каждой колонки с character типом данных, группируя по полу (Gender).

  Для расчета количества значений в строке есть функция nchar(), для расчета индекса (первого) максимального значения есть функция which.max().

```
# A tibble: 3 x 8
```

```
  Gender name    `Eye color` Race  `Hair color` Publisher `Skin color` Alignment
  <chr>  <chr>   <chr>       <chr> <chr>        <chr>     <chr>        <chr>
1 Female Negaso~ yellow (wi~ Huma~ Strawberry ~ Dark Hor~ orange       neutral
2 Male   Drax t~ yellow / r~ Dath~ Strawberry ~ Dark Hor~ orange / wh~ neutral
3 <NA>   Captai~ yellow (wi~ God ~ Orange / Wh~ Marvel C~ grey         neutral
```

- Создайте из тиббла `heroes` новый тиббл, в котором числовые значения `Height` и `Weight` заменены на следующие строковые значения: если у супергероя рост или вес выше среднего по колонке, то " ", если его/ее рост или вес ниже или равен среднему, то " ".

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>        <chr>  <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         ~ Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         ~ Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         ~ DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         ~ Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black        <NA>   Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         ~ Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond        <NA>   NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           ~ DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           ~ Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           ~ Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <chr>
```

- Создайте из тиббла `heroes` новый тиббл, в котором числовые значения `Height` и `Weight` заменены на следующие строковые значения: если у супергероя *внутри соответствующей группы по полу* рост или вес выше среднего по колонке, то " X", если его/ее рост или вес ниже или равен среднему *внутри соответствующей группы по полу*, то " X" , где *X* – соответствующий пол (`Gender`).

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>        <chr>  <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         ~ Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         ~ Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         ~ DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         ~ Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black        NA  ~ Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         ~ Marvel C~
```

```
 7     6 Adam Monroe   Male   blue        <NA>     Blond        NA  ~ NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           ~ DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           ~ Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           ~ Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <chr>
```

26.24 Описательная статистика

Для выполнения задания создайте вектор `height` из колонки `Height` датасета `heroes`, удалив в нем `NA`.

- Посчитайте среднее в векторе `height`.

```
[1] 186.7263
```

- Посчитайте усеченное среднее в векторе `height` с усечением 5% значений с обоих сторон.

```
[1] 182.5846
```

- Посчитайте медиану в векторе `height`.

```
[1] 183
```

- Посчитайте стандартное отклонение в векторе `height`.

```
[1] 59.25189
```

- Посчитайте межквартильный размах в векторе `height`.

```
[1] 18
```

- Посчитайте ассиметрию в векторе `height`.

```
[1] 8.843432
```

Посчитайте эксцесс в векторе `height`.

```
[1] 105.0297
```

Примените функции для получения множественных статистик на векторе `height`.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   15.2   173.0   183.0   186.7   191.0   975.0
```

```
   vars   n   mean    sd median trimmed   mad  min max range skew kurtosis   se
X1    1 517 186.73 59.25    183  182.02 11.86 15.2 975 959.8 8.84   105.03 2.61
```

Таблица 26.1: Data summary

Name	height
Number of rows	517
Number of columns	1
Column type frequency:	
numeric	1
Group variables	None

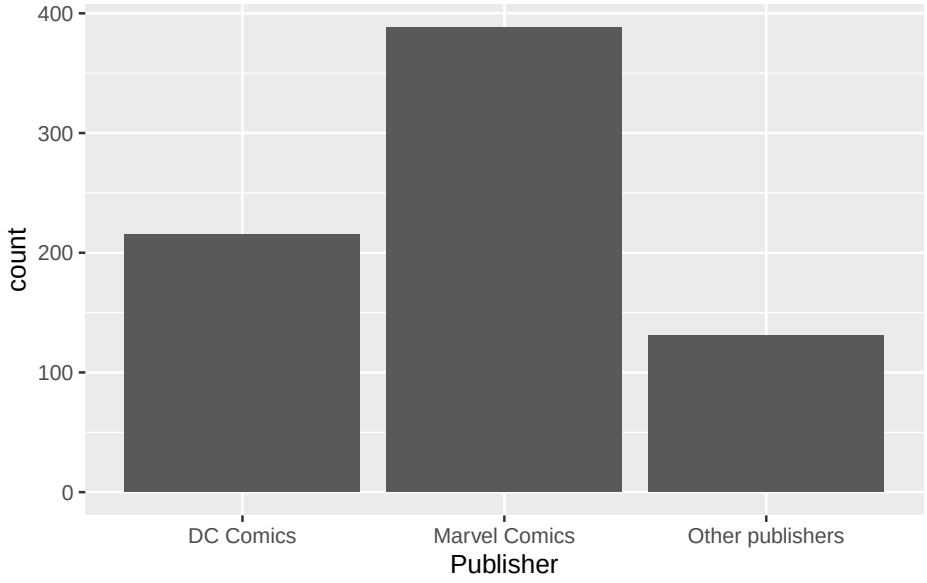
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
data	0	1	186.73	59.25	15.2	173	183	191	975	▇▁▁▁▁

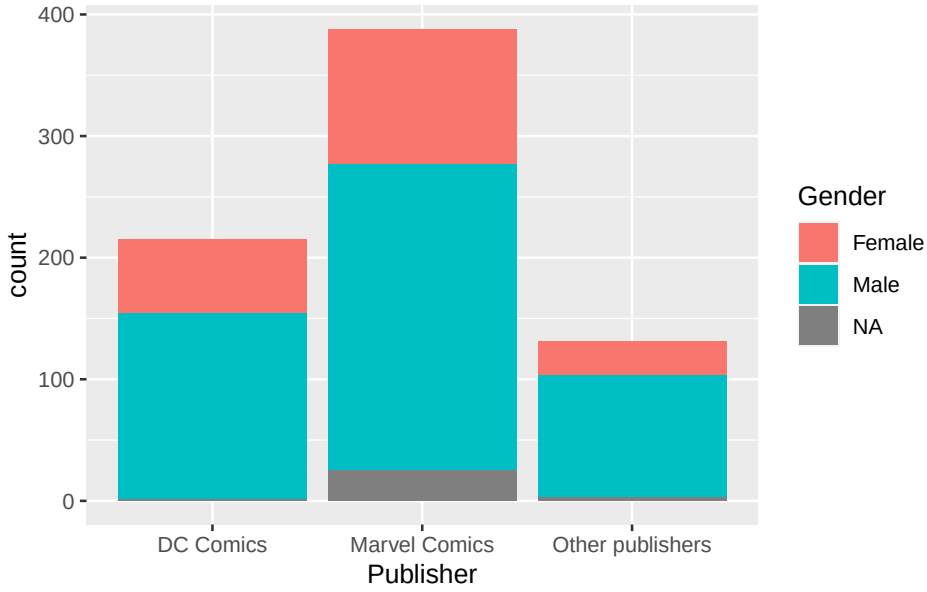
26.25 Построение графиков в `ggplot2`

- Нарисуйте столбиковую диаграмму (`geom_bar()`), которая будет отражать количество супергероев издателей `"Marvel Comics"`, `"DC Comics"` и всех остальных (отдельным столбиком) из датасета `heroes`.

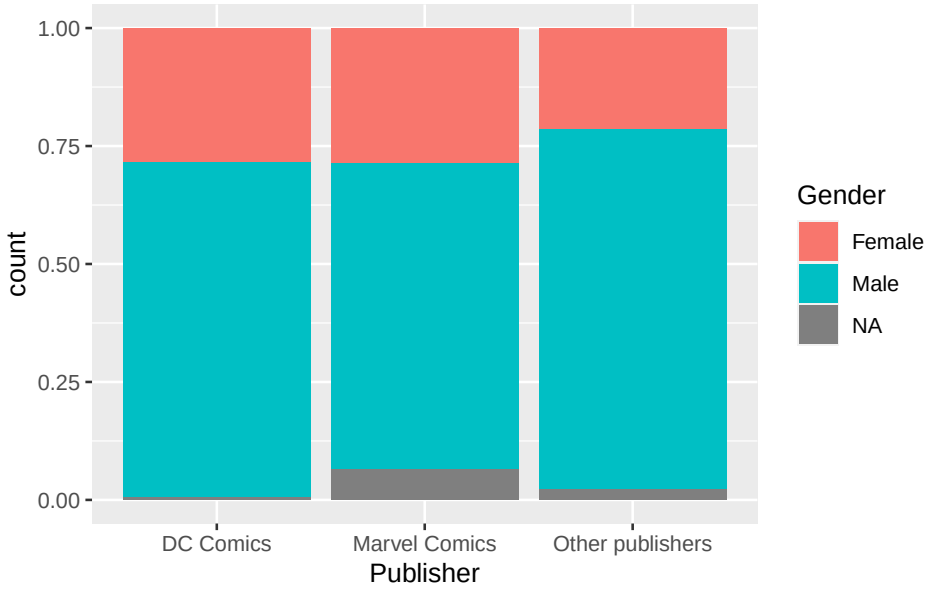
```
heroes <- heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers"))
  #mutate(Publisher = fct_lump(Publisher, 2)) %>% #                    ,       for
```



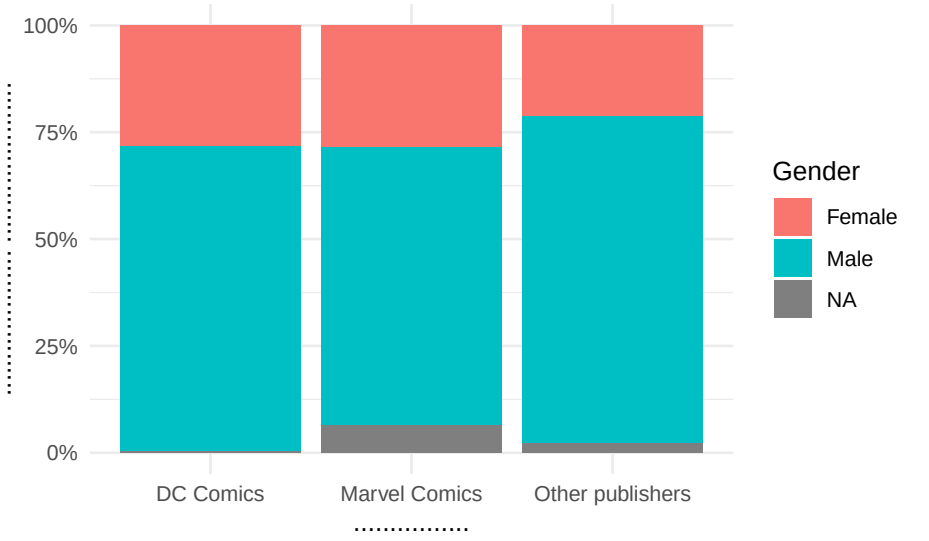
- Добавьте к этой диаграмме заливку цветом (`fill =`) в зависимости от распределения `Gender` внутри каждой группы.



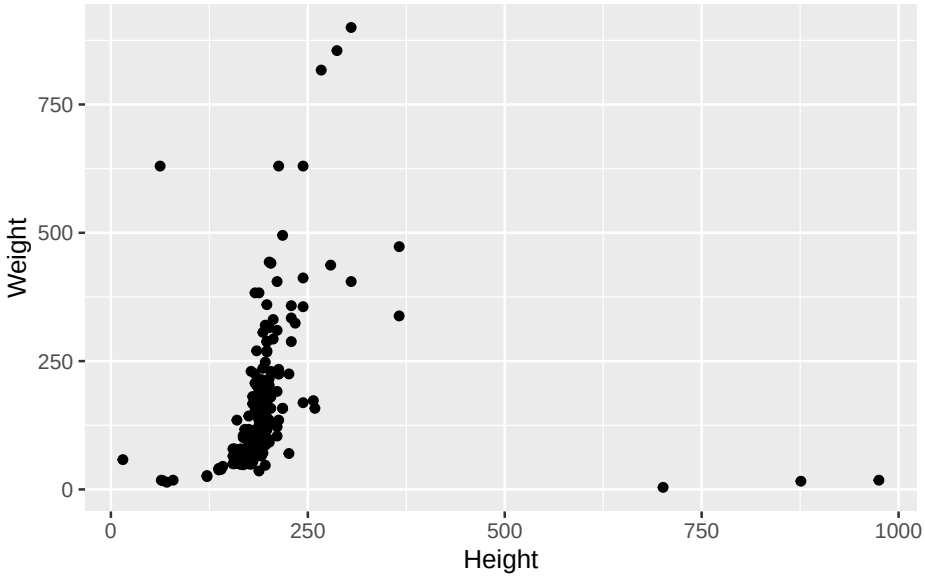
- Сделайте так, чтобы каждый столбик был максимальной высоты (`position = "fill"`).



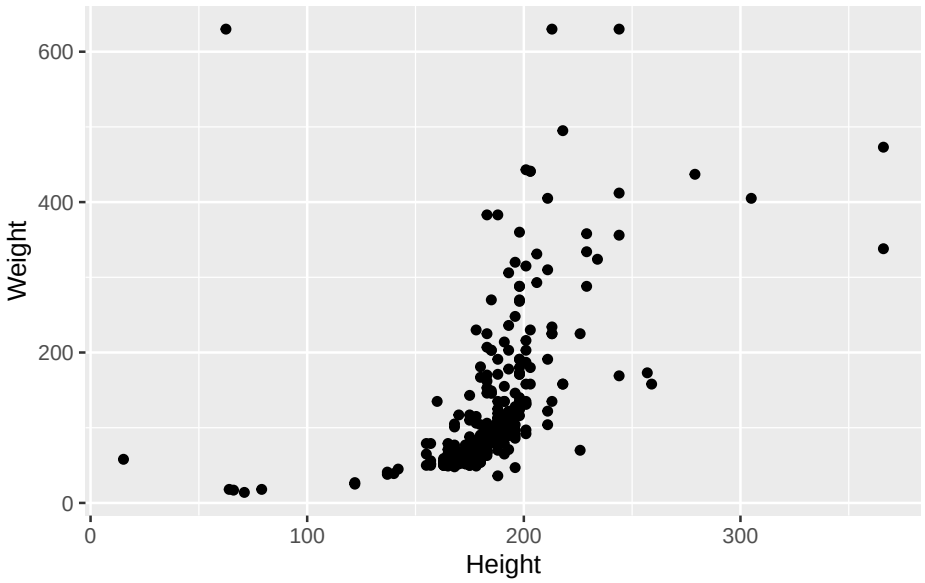
- Финализируйте график, задав ему описания осей (например, функция `labs()`), использовав процентную шкалу (`scale_y_continuous(labels = scales::percent)`) и задав тему `theme_minimal()`.



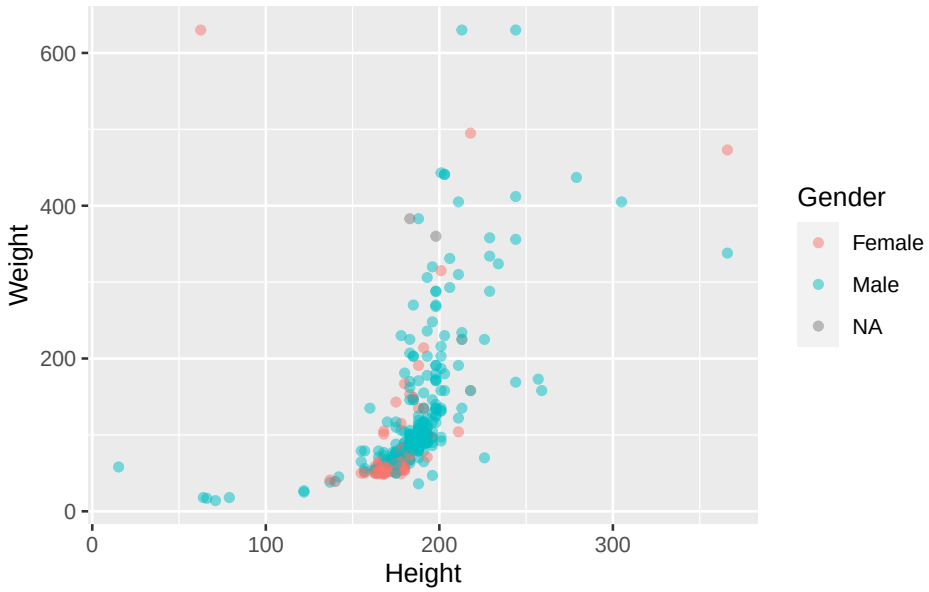
Создайте диаграмму рассеяния для датасета `heroes`, для которой координаты по оси *x* будут взяты из колонки `Height`, а координаты по оси *y* – из колонки `Weight`.



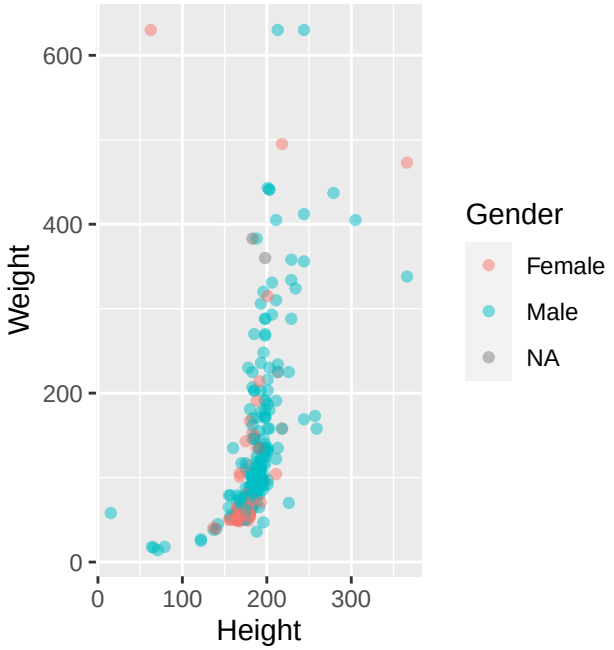
- Удалите с графика все экстремальные значения, для которых `Weight` больше или равен 700 или `Height` больше или равен 400. (*Подсказка:* это можно делать как средствами `ggplot2`, так и функцией `filter()` из `dplyr`).



- Раскрасьте точки в зависимости от `Gender`, сделайте их полупрозрачными (параметр `alpha` =).

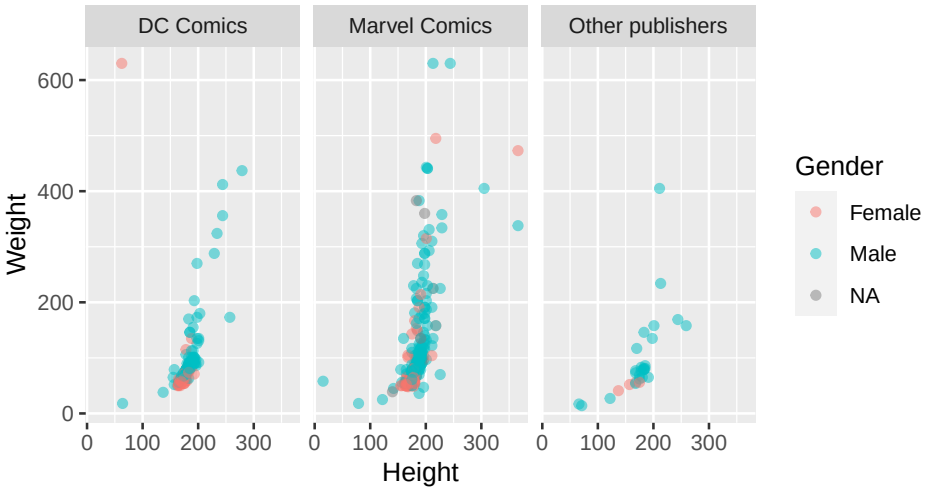


- Сделайте так, чтобы координатная плоскость имела соотношение 1:1 шкал по оси *x* и *y*. Этого можно добиться с помощью функции `coord_fixed()`.

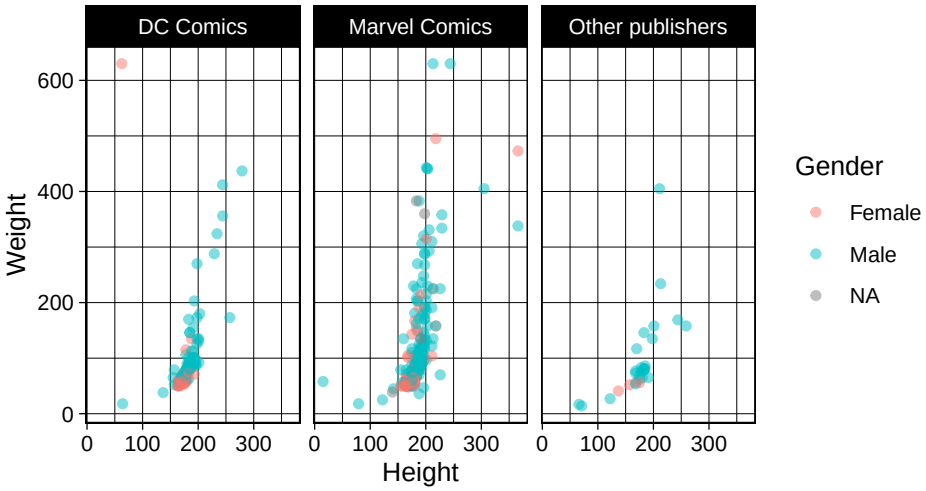


Разделите график (`facet_wrap()`) на три: для `"DC Comics","Marvel`

Comics" и всех остальных.



- Используйте для графика тему `theme_linedraw()`.



- **—** Постройте новый график (или возьмите старый) по датасетам `heroes` и/или `powers` и сделайте его некрасивым! Чем *хуже* у вас получится график, тем лучше. Желательно, чтобы этот график был по-прежнему графиком, а не произведением абстрактного искусства. Разница очень тонкая, но она есть.

Вот несколько подсказок для этого задания:

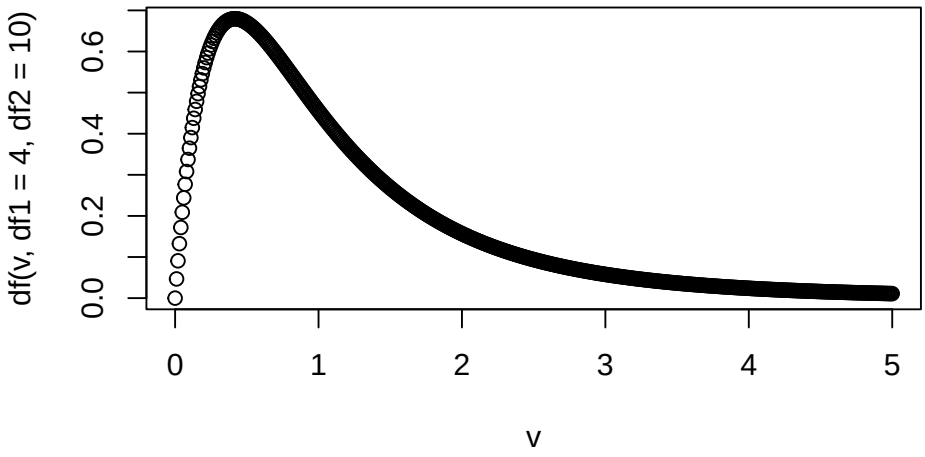
1. Для вдохновения посмотрите на вот эти графики.
2. Для реально плохих графиков вам придется покопаться с настройками темы. Посмотрите подсказку по темам `?theme`, попытайтесь что-то поменять в теме.
3. Экспериментируйте с разными геомами и необычными их применениями.
4. По изучайте дополнения к `gpplot2`.
5. Попробуйте подготовить интересные данные для этого графика.

26.26 Распределения

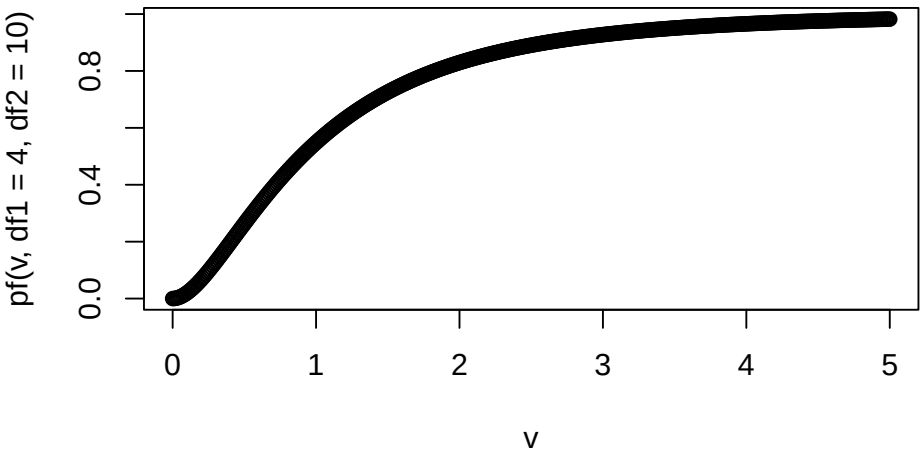
Выберите любое непрерывное распределение из представленных в базовом пакете `stats` или же в любом другом пакете. Найти все распределения пакета `stats` можно с помощью `?Distributions`. Подберите для него какие-нибудь параметры или используйте параметры по умолчанию.

> Я возьму F-распределение с параметрами `df1 = 4` и `df = 10`, но вы можете выбрать другое распределение.

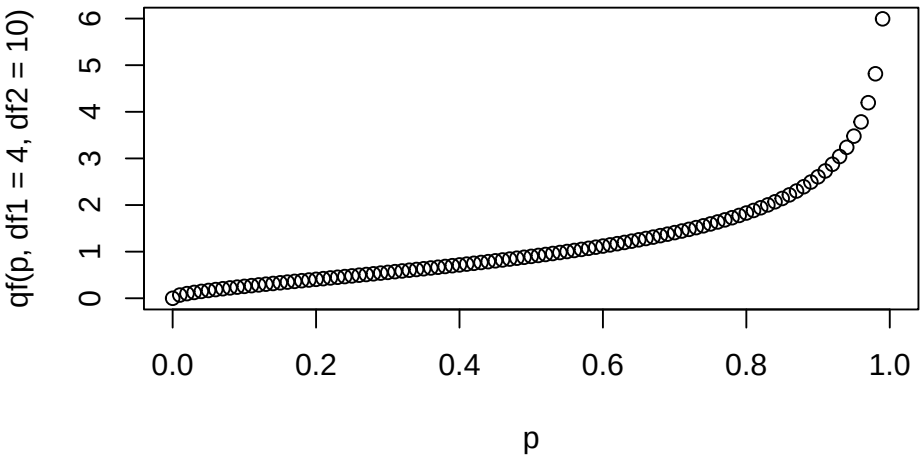
- Визуализируйте функцию плотности вероятности для выбранного распределения.



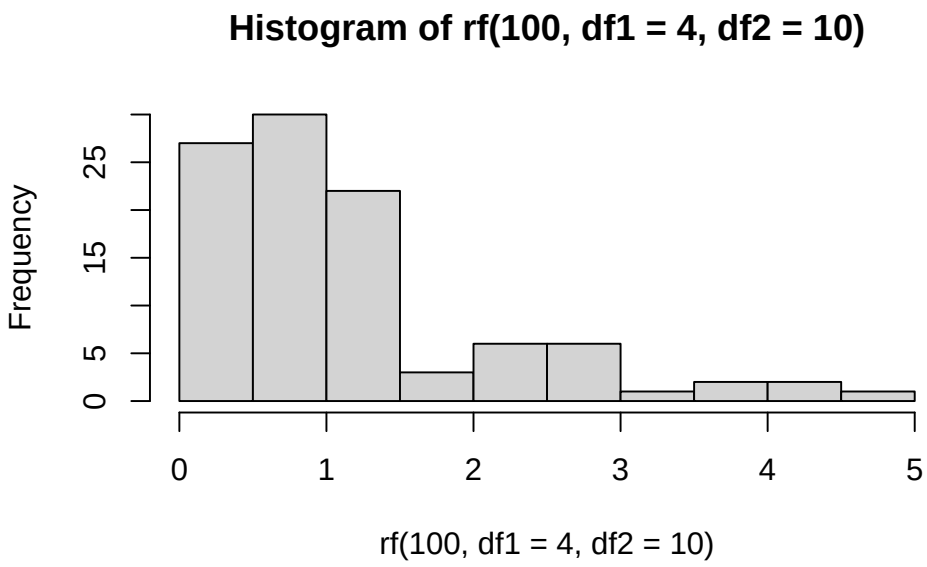
- Визуализируйте функцию накопленной плотности распределения для выбранной функции.



- Визуализируйте квантильную функцию для выбранного распределения.



- Сделайте выборку из 100 случайных значений из выбранного распределения и постройте гистограмму (функция `hist()`) для полученной выборки.



26.27 Одновыборочный t-test

- Представьте, что наши супергерои из набора данных `heroes` – это выборка из генеральной совокупности всех написанных и ненаписанных супергероев. Проведите одновыборочный t-тест для веса супергероев и числа 100 – предположительного среднего веса в генеральной совокупности всех супергероев. Проинтерпретируйте результат.

- Проведите одновыборочный t-тест для роста супергероев и числа 185 – предположительного среднего роста в генеральной совокупности всех супергероев. Проинтерпретируйте результат.

26.28 Двухвыборочный зависимый t-test

Для дальнейших заданий понадобится набор данных о результативности трех диет, который мы использовали во время занятия.

```
diet <- readr::read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/mas
```

- Посчитайте двухвыборочный зависимый t-тест для остальных диет: для диеты 2 и диеты 3. Проинтерпретируйте полученные результаты.

26.29 Двухвыборочный независимый t-test

- Сделайте независимый t-тест для сравнения массы испытуемых двух групп после диеты, сравнив вторую и третью группу. Проинтерпретируйте результаты.

- Сделайте независимый t-тест для сравнения массы испытуемых двух групп после диеты, сравнив первую и третью группу. Проинтерпретируйте результаты.

- Проведите независимый t-тест для сравнения массы супергероев с черными и белыми глазами.

```
heroes_white_black <- heroes %>%
  filter(`Eye color` %in% c("white", "black"))
```

26.30 Непараметрические аналоги t-теста

- Сравните вес первой и второй группы после диеты, используя тест Манна-Уитни. Сравните результаты теста Манна-Уитни с результатами t-теста? Проинтерпретируйте полученные результаты.

- Повторите задание для второй и третьей группы, а так же для первой и третьей группы.

- Сравните вес до и после для диеты 1, используя тест Уилкоксона. Сравните с результатами применения t-теста. Проинтерпретируйте полученные результаты.

- Сравните вес до и после для диеты 2 и диеты 3, используя тест Уилкоксона. Сравните с результатами применения t-теста. Проинтерпретируйте полученные результаты.

26.31 Критерий хи-квадрат Пирсона

- Связаны ли переменные Alignment и Gender? Используйте критерий хи-квадрат для проверки и проинтерпретируйте результаты

```
pub_good <- heroes %>%
  filter(Alignment %in% c("good", "bad")) %>%
  select(Alignment, Gender) %>%
  drop_na()
```

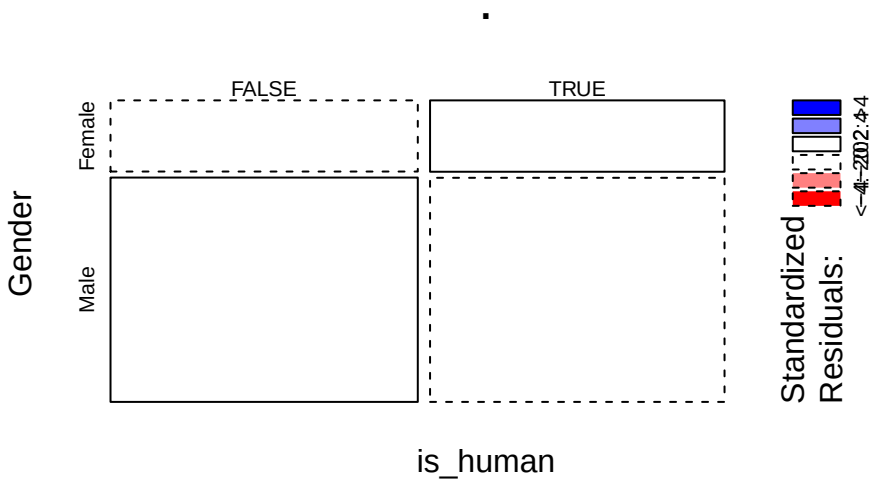
- Создайте в `heroes` новую колонку `is_human` логического типа, в которой будет `TRUE`, если супергерой принадлежит расе (`Race`) `"Human"`, и `FALSE` в случае если супергерой принадлежит другой расе.

- Посчитайте долю женщин для `"Human"` и всех остальных (`is_human` равен `TRUE` и `FALSE` соответственно). Перед этим удалите все строчки с `NA` в переменных `is_human` и `Gender`.

```
# A tibble: 2 x 2
  is_human `mean(Gender == "Female")`
  <lgl>                         <dbl>
1 FALSE                         0.241
2 TRUE                          0.242
```

- Сравните распределения частот для переменных `is_human` и `Gender` используя хи-квадрат Пирсона. Проинтерпретируйте результаты.

- Постройте мозаичный график для переменных `is_human` и `Gender`.



26.32 Исследование набора данных Backpack

Для следующих тем нам понадобится набор данных `Backpack` из пакета `Stat2Data`.

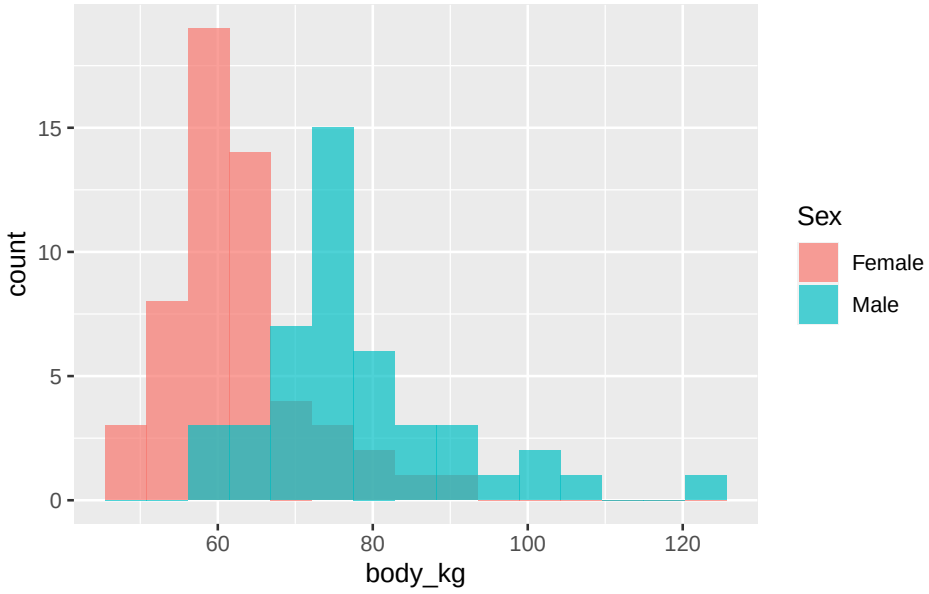
```
#install.packages("Stat2Data")
library(Stat2Data)
data(Backpack)
back <- Backpack %>%
  mutate(backpack_kg = 0.45359237 * BackpackWeight,
         body_kg = 0.45359237 * BodyWeight)
```

- Как различается вес рюкзака в зависимости от пола? Кто весит больше?

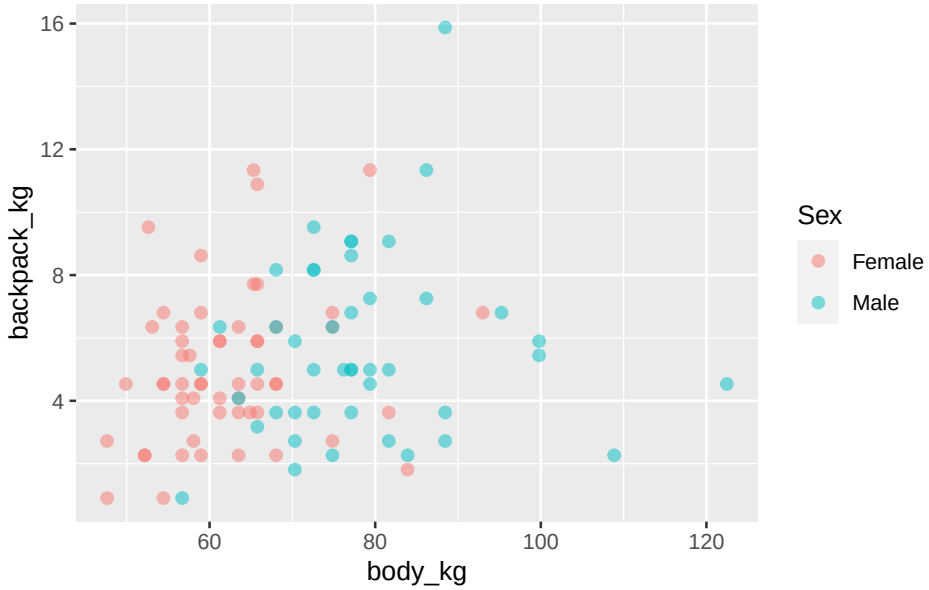
```
# A tibble: 2 x 2
  Sex    `mean(backpack_kg)`
  <fct>                <dbl>
1 Female                5.01
2 Male                  5.63
```

- Если допустить, что выборка репрезентативна, то можно ли сделать вывод о различии по среднему весу рюкзаков в генеральной совокупности?
- Повторите пункты 2 и 3 для веса самих студентов.

- Визуализируйте распределение этих двух переменных в зависимости от пола (используя `ggplot2`)



- Постройте диаграмму рассеяния с помощью `ggplot2`. Цветом закодируйте пол респондента.



26.33 Ковариация

- Посчитайте матрицу ковариаций для веса студентов и их рюкзаков в *фунтах.* Различаются ли результаты подсчета ковариации этих двух переменных от результатов подсчета ковариаций веса студентов и их рюкзаков в *килограммах*? Почему?

26.34 Коэффициент корреляции

- Посчитайте коэффициент корреляции Пирсона для веса студентов и их рюкзаков в *фунтах.* Различаются ли результаты подсчета коэффициента корреляции Пирсона (сам коэффициент, p-value) этих двух переменных от результатов подсчета корреляции Пирсона веса студентов и их рюкзаков в *килограммах*? Почему?
- Посчитайте коэффициент корреляции Пирсона для веса и роста супергероев из датасета `heroes`. Проинтерпретируйте результат.
- Теперь посчитайте коэффициент корреляции Спирмена и коэффициент корреляции Кэнделла для веса и роста супергероев из датасета `heroes`. Различаются ли результаты по сравнению с коэффициентом корреляции Пирсона? Почему?

26.35 ANOVA

Загрузите пингвиний датасет:

```
library(tidyverse)
penguins <- readr::read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/da
  mutate(id = row_number()) #      id
```

Давайте посчитаем арифметические средние по группам для различных признаков:

```
penguins %>%
  group_by(species) %>%
  summarise(across(ends_with("_mm") | ends_with("_g"),
```

```
                    mean,
                    na.rm = TRUE))
```

```
# A tibble: 3 x 5
  species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <chr>              <dbl>         <dbl>             <dbl>       <dbl>
1 Adelie              38.8          18.3              190.       3701.
2 Chinstrap           48.8          18.4              196.       3733.
3 Gentoo              47.5          15.0              217.       5076.
```

- Есть ли статистически значимые различия между видами пингвинов по массе тела (`body_mass_g`)? Если да, то между какими?
- Есть ли статистически значимые различия между видами пингвинов по длине ласт (`flipper_length_mm`)? Если да, то между какими?
- Есть ли взаимодействие между полом (`sex`) и видом (`species`) для длины ласт `flipper_length_mm`? Если да, то между какими группами?

Глава 27

Решения заданий

Задания, которые помечены звездочкой (*) можно пропускать: это задания повышенной сложности, в них требуется подумать над решением, а не просто применить выученные инструменты.

27.1 Начало работы в R

- Разделите 9801 на 9.

```
9801/9
```

```
[1] 1089
```

- Посчитайте логарифм от 2176782336 по основанию 6.

```
log(2176782336, 6)
```

```
[1] 12
```

- Теперь натуральный логарифм 10 и умножьте его на 5.

```
log(10)*5
```

```
[1] 11.51293
```

- С помощью функции `sin()` посчитайте $\sin(\pi)$, $\sin\left(\frac{\pi}{2}\right)$, $\sin\left(\frac{\pi}{6}\right)$.

  Значение π - зашитая в R константа (`pi`).

```
sin(pi)
```

```
[1] 1.224647e-16
```

```
sin(pi/2)
```

```
[1] 1
```

```
sin(pi/6)
```

```
[1] 0.5
```

27.2 Создание векторов

- Создайте вектор из значений 2, 30 и 4000.

```
c(2, 30, 4000)
```

```
[1]    2   30 4000
```

- Создайте вектор от 1 до 20.

```
1:20
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Создайте вектор от 20 до 1.

```
20:1
```

```
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

Функция `sum()` возвращает сумму элементов вектора на входе. Посчитайте сумму первых 100 натуральных чисел (т.е. всех целых чисел от 1 до 100).

```
sum(1:100)
```

```
[1] 5050
```

- Создайте вектор от 1 до 20 и снова до 1. Число 20 должно присутствовать только один раз!

```
c(1:20, 19:1)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18 17 16 15
[26] 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

- Создайте вектор значений 5, 4, 3, 2, 2, 3, 4, 5:

```
c(5:2, 2:5)
```

```
[1] 5 4 3 2 2 3 4 5
```

- Создайте вектор 2, 4, 6, … , 18, 20.

```
seq(2, 20, 2)
```

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

- Создайте вектор 0.1, 0.2, 0.3, …, 0.9, 1.

```
seq(0.1, 1, 0.1)
```

```
 [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- 2020 год — високосный. Следующий високосный год через 4 года — это будет 2024 год. Составьте календарь всех високосных годов XXI века, начиная с 2020 года.

  2100 год относится к XXI веку, а не к XXII.

```
seq(2020, 2100, 4)
```

```
 [1] 2020 2024 2028 2032 2036 2040 2044 2048 2052 2056 2060 2064 2068 2072 2076
[16] 2080 2084 2088 2092 2096 2100
```

- Создайте вектор, состоящий из 20 повторений “Хэй!”.

```
rep(" !", 20)
```

```
 [1] " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !"
[11] " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !"
```

- Как я и говорил, многие функции, работающие с одним значением на входе, так же прекрасно работают и с целыми векторами. Попробуйте посчитать квадратный корень чисел от 1 до 10 с помощью функции `sqrt()` и сохраните результат в векторе `roots`. Выведите содержание вектора `roots` в консоль.

```
roots <- sqrt(1:10)
roots
```

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278
```

- *Создайте вектор из одной единицы, двух двоек, трех троек, …. , девяти девяток.

```
rep(1:9, 1:9)
```

```
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9
[39] 9 9 9 9 9 9 9
```

27.3 Приведение типов

- Сделайте вектор `vec1`, в котором соедините `3`, а также значения `"  "` и `"    "`.

```
vec1 <- c(3, "  ", "    ")
vec1
```

```
[1] "3"      "  "    "    "
```

- Попробуйте вычесть `TRUE` из 10.

```
10 - TRUE
```

```
[1] 9
```

- Соедините значение `10` и `TRUE` в вектор `vec2`.

```
vec2 <- c(10, TRUE)
vec2
```

```
[1] 10  1
```

- Соедините вектор `vec2` и значение `"r"`:

```
c(vec2, "r")
```

```
[1] "10" "1"  "r"
```

- Соедините значения `10`, `TRUE`, `"r"` в вектор.

```
c(10, TRUE, "r")
```

```
[1] "10"   "TRUE" "r"
```

27.4 Векторизация

- Создайте вектор `p`, состоящий из значений 4, 5, 6, 7, и вектор `q`, состоящий из 0, 1, 2, 3.

```
p <- 4:7
p
```

```
[1] 4 5 6 7
```

```
q <- 0:3
q
```

```
[1] 0 1 2 3
```

- Посчитайте поэлементную сумму векторов `p` и `q`:

```
p + q
```

```
[1]  4  6  8 10
```

- Посчитайте поэлементную разницу `p` и `q`:

```
p - q
```

```
[1] 4 4 4 4
```

- Поделите каждый элемент вектора `p` на соответствующий ему элемент вектора `q`:

  О, да, Вам нужно делить на 0!

```
p / q
```

```
[1]      Inf 5.000000 3.000000 2.333333
```

- Возведите каждый элемент вектора `p` в степень соответствующего ему элемента вектора `q`:

```
p ^ q
```

```
[1]   1   5  36 343
```

- Умножьте каждое значение вектора `p` на 10.

```
p * 10
```

```
[1] 40 50 60 70
```

- Создайте вектор квадратов чисел от 1 до 10:

```
(1:10)^2
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

- Создайте вектор 0, 2, 0, 4, … , 18, 0, 20.

```
1:20 * 0:1
```

```
 [1]  0  2  0  4  0  6  0  8  0 10  0 12  0 14  0 16  0 18  0 20
```

- Создайте вектор 1, 0, 3, 0, 5, …, 17, 0, 19, 0.

```
1:20 * 1:0
```

```
 [1]  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19  0
```

- *Создайте вектор, в котором будут содержаться первые 20 степеней двойки.

```
2 ^ (1:20)
```

```
 [1]       2       4       8      16      32      64     128     256     512
[10]    1024    2048    4096    8192   16384   32768   65536  131072  262144
[19]  524288 1048576
```

- *Создайте вектор из чисел 1, 10, 100, 1000, 10000:

```
10 ^ (0:4)
```

```
[1]     1    10   100  1000 10000
```

- *Посчитать сумму последовательности $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \ldots + \frac{1}{50 \cdot 51}$.

```
sum(1 / (1:50 * 2:51))
```

```
[1] 0.9803922
```

- *Посчитать сумму последовательности $\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \frac{1}{2^{20}}$.

```
sum(1 / 2 ^ (0:20))
```

```
[1] 1.999999
```

- *Посчитать сумму последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$.

```
sum((3 * (1:10) - 2) / 3 ^ (0:9))
```

```
[1] 3.749174
```

- *Сколько чисел из последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$ больше чем 0.5?

```
sum((3 * (1:10) - 2) / 3 ^ (0:9) > 0.5)
```

```
[1] 3
```

27.5 Индексирование векторов

- Создайте вектор `troiki` со значениями 3, 6, 9, ..., 24, 27.

```
troiki <- seq(3, 27, 3)
troiki
```

```
[1]  3  6  9 12 15 18 21 24 27
```

- Извлеките 2, 5 и 7 значения вектора `troiki`.

```
troiki[c(2, 5, 7)]
```

```
[1]  6 15 21
```

- Извлеките *предпоследнее* значение вектора `troiki`.

```
troiki[length(troiki) - 1]
```

```
[1] 24
```

- Извлеките все значения вектора `troiki` *кроме* предпоследнего:

```
troiki[-(length(troiki) - 1)]
```

```
[1]  3  6  9 12 15 18 21 27
```

Создайте вектор `vec3`, скопировав следующий код:

```
vec3 <- c(3, 5, 2, 1, 8, 4, 9, 10, 3, 15, 1, 11)
```

- Найдите второй элемент вектора `vec3`.

```
vec3[2]
```

```
[1] 5
```

- Верните второй и пятый элемент вектора `vec3`.

```
vec3[c(2, 5)]
```

```
[1] 5 8
```

- Попробуйте извлечь сотое значение вектора `vec3`:

```
vec3[100]
```

```
[1] NA
```

- Верните все элементы вектора `vec3` *кроме* второго элемента.

```
vec3[-2]
```

```
 [1]  3  2  1  8  4  9 10  3 15  1 11
```

- Верните все элементы вектора `vec3` *кроме* второго и пятого элемента.

```
vec3[c(-2, -5)]
```

```
 [1]  3  2  1  4  9 10  3 15  1 11
```

- Найдите последний элемент вектора `vec3`.

```
vec3[length(vec3)]
```

```
[1] 11
```

- Верните все значения вектора `vec3` кроме первого и последнего.

```
vec3[c(-1, -length(vec3))]
```

```
 [1]  5  2  1  8  4  9 10  3 15  1
```

- Найдите все значения вектора `vec3`, которые больше 4.

```
vec3[vec3 > 4]
```

```
[1]  5  8  9 10 15 11
```

- Найдите все значения вектора `vec3`, которые больше 4, но меньше 10.

  Если хотите сделать это в одну строчку, то вам помогут логические операторы!

```
vec3[vec3 > 4 & vec3 < 10]
```

```
[1] 5 8 9
```

- Найдите все значения вектора `vec3`, которые меньше 4 или больше 10.

```
vec3[vec3 < 4 | vec3 > 10]
```

```
[1]  3  2  1  3 15  1 11
```

- Возведите в квадрат каждое значение вектора `vec3`.

```
vec3 ^ 2
```

```
 [1]   9  25   4   1  64  16  81 100   9 225   1 121
```

- *Возведите в квадрат каждое значение вектора на нечетной позиции и извлеките корень из каждого значения на четной позиции вектора `vec3`.

  Извлечение корня - это то же самое, что и возведение в степень 0.5.

```
vec3 ^ c(2, 0.5)
```

```
 [1]  9.000000  2.236068  4.000000  1.000000 64.000000  2.000000 81.000000
 [8]  3.162278  9.000000  3.872983  1.000000  3.316625
```

- Создайте вектор 2, 4, 6, … , 18, 20 как минимум 2 новыми способами.

  Знаю, это задание может показаться бессмысленным, но это очень базовая операция, с помощью которой можно, например, разделить данные на две части. Чем больше способов Вы знаете, тем лучше!

```
(1:20)[c(FALSE,TRUE)]
```

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

```
#(1:10)*2
```

27.6 Работа с пропущенными значениями

- Создайте вектор `vec4` со значениями 300, 15, 8, 2, 0, 1, 110, скопировав следующий код:

```
vec4 <- c(300, 15, 8, 20, 0, 1, 110)
vec4
```

```
[1] 300  15   8  20   0   1 110
```

- Замените все значения `vec4`, которые больше 20 на `NA`.

```
vec4[vec4 > 20] <- NA
```

- Проверьте полученный вектор `vec4`:

```
vec4
```

```
[1] NA 15  8 20  0  1 NA
```

- Посчитайте сумму `vec4` с помощью функции `sum()`. Ответ `NA` не считается!

```
sum(vec4, na.rm = TRUE)
```

```
[1] 44
```

27.7 Матрицы

- Создайте матрицу размером *4x4*, состоящую из единиц. Назовите ее `M1`.

```
M1 <- matrix(rep(1, 16), ncol = 4)
M1
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    1    1    1
[3,]    1    1    1    1
[4,]    1    1    1    1
```

- Поменяйте все некрайние значения матрицы `M1` (то есть значения на позициях [2,2], [2,3], [3,2] и [3,3]) на число 2.

```
M1[2:3, 2:3] <- 2
M1
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    2    2    1
[3,]    1    2    2    1
[4,]    1    1    1    1
```

- Выделите второй и третий столбик из матрицы `M1`.

```
M1[,2:3]
```

```
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    2    2
[4,]    1    1
```

- Сравните (==) вторую колонку и вторую строчку матрицы `M1`.

```
M1[,2] == M1[2,]
```

```
[1] TRUE TRUE TRUE TRUE
```

- Создайте матрицу `M2` из *пяти строк* и *шести столбцов*, в которой будут записаны значения *от 1 до 30* (сверху вниз, затем слева направо):

```
M2 <- matrix(1:30, ncol = 6)
M2
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    6   11   16   21   26
[2,]    2    7   12   17   22   27
[3,]    3    8   13   18   23   28
[4,]    4    9   14   19   24   29
[5,]    5   10   15   20   25   30
```

- Извлеките из матрицы `M2` все значения (в виде матрицы) *кроме значений из третьей строки и второго столбца.*

```
M2[-3, -2]
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   11   16   21   26
[2,]    2   12   17   22   27
[3,]    4   14   19   24   29
[4,]    5   15   20   25   30
```

- Умножьте каждое значение матрицы `M2` на 100.

```
M2 * 100
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  100  600 1100 1600 2100 2600
[2,]  200  700 1200 1700 2200 2700
[3,]  300  800 1300 1800 2300 2800
[4,]  400  900 1400 1900 2400 2900
[5,]  500 1000 1500 2000 2500 3000
```

- Возведите каждое значение матрицы `M2` *в квадрат.*

```
M2 ^ 2
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1   36  121  256  441  676
[2,]    4   49  144  289  484  729
[3,]    9   64  169  324  529  784
[4,]   16   81  196  361  576  841
[5,]   25  100  225  400  625  900
```

- Возведите каждое значение матрицы `M2` *в квадрат* и *вычтите* значения изначальной матрицы `M2`.

```
M2 ^ 2 - M2
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0   30  110  240  420  650
[2,]    2   42  132  272  462  702
[3,]    6   56  156  306  506  756
[4,]   12   72  182  342  552  812
[5,]   20   90  210  380  600  870
```

- *Создайте таблицу умножения (9x9) в виде матрицы. Сохраните ее в переменную `mult_tab`.

  Вам может понадобиться функция `rep()`.

```
mult_tab <- matrix(rep(1:9, rep(9,9))*(1:9), nrow = 9)
mult_tab
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    1    2    3    4    5    6    7    8    9
 [2,]    2    4    6    8   10   12   14   16   18
 [3,]    3    6    9   12   15   18   21   24   27
 [4,]    4    8   12   16   20   24   28   32   36
 [5,]    5   10   15   20   25   30   35   40   45
 [6,]    6   12   18   24   30   36   42   48   54
 [7,]    7   14   21   28   35   42   49   56   63
 [8,]    8   16   24   32   40   48   56   64   72
 [9,]    9   18   27   36   45   54   63   72   81
```

```
#
#outer(1:9, 1:9, "*")
#1:9 %o% 1:9
```

- *Из матрицы `mult_tab` выделите подматрицу, включающую в себя только строчки с 6 по 8 и столбцы с 3 по 7.

```
mult_tab[6:8, 3:7]
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   18   24   30   36   42
[2,]   21   28   35   42   49
[3,]   24   32   40   48   56
```

- *Создайте матрицу с логическими значениями, где `TRUE`, если в этом месте в таблице умножения (`mult_tab`) двузначное число и `FALSE`, если однозначное.

  Матрица - это почти вектор. К нему можно обращаться с единственным индексом.

```
mult_tab >= 10
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
 [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [2,] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 [3,] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [4,] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [5,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [6,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [7,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [8,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [9,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- *Создайте матрицу `mult_tab2`, в которой все значения `tab` меньше 10 заменены на 0.

```
mult_tab2 <- mult_tab
mult_tab2[mult_tab < 10] <- 0
mult_tab2
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    0    0    0    0    0    0    0    0    0
 [2,]    0    0    0    0   10   12   14   16   18
 [3,]    0    0    0   12   15   18   21   24   27
 [4,]    0    0   12   16   20   24   28   32   36
 [5,]    0   10   15   20   25   30   35   40   45
 [6,]    0   12   18   24   30   36   42   48   54
 [7,]    0   14   21   28   35   42   49   56   63
 [8,]    0   16   24   32   40   48   56   64   72
 [9,]    0   18   27   36   45   54   63   72   81
```

27.8 Списки

Дан список `list1`:

```
list1 = list(numbers = 1:5, letters = letters, logic = TRUE)
list1
```

```
$numbers
[1] 1 2 3 4 5

$letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

$logic
[1] TRUE
```

- Найдите первый элемент списка `list1`. Ответ должен быть списком длиной один.

```
list1[1]
```

```
$numbers
[1] 1 2 3 4 5
```

- Теперь найдите содержание первого элемента списка `list1` двумя разными способами. Ответ должен быть вектором.

```
list1[[1]]
```

```
[1] 1 2 3 4 5
```

```
list1$numbers
```

```
[1] 1 2 3 4 5
```

- Теперь возьмите первый элемент содержания первого элемента списка `list1`. Ответ должен быть вектором.

```
list1[[1]][1]
```

```
[1] 1
```

- Создайте список `list2`, содержащий в себе два списка `list1`. Один из них будет иметь имя `pupa`, а другой — `lupa`.

```
list2 = list(pupa = list1, lupa = list1)
list2
```

```
$pupa
$pupa$numbers
[1] 1 2 3 4 5

$pupa$letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

$pupa$logic
[1] TRUE


$lupa
$lupa$numbers
[1] 1 2 3 4 5

$lupa$letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

$lupa$logic
[1] TRUE
```

- *Извлеките первый элемент списка `list2`, из него — второй подэлемент, а из него — третье значение.

```
list2[[1]][[2]][3]
```

```
[1] "c"
```

27.9 Датафрейм

- Запустите команду `data(mtcars)` чтобы загрузить встроенный датафрейм с информацией про автомобили. Каждая строчка датафрейма - модель автомобиля, каждая колонка - отдельная характеристика. Подробнее см. `?mtcars`.

```
data(mtcars)
mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Изучите структуру датафрейма `mtcars` с помощью функции `str()`.

```
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- Найдите значение третьей строчки четвертого столбца датафрейма `mtcars`.

```
mtcars[3, 4]
```

```
[1] 93
```

- Извлеките первые шесть строчек и первые шесть столбцов датафрейма `mtcars`.

```
mtcars[1:6, 1:6]
```

```
                   mpg cyl disp  hp drat    wt
Mazda RX4         21.0   6  160 110 3.90 2.620
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875
Datsun 710        22.8   4  108  93 3.85 2.320
Hornet 4 Drive    21.4   6  258 110 3.08 3.215
Hornet Sportabout 18.7   8  360 175 3.15 3.440
Valiant           18.1   6  225 105 2.76 3.460
```

- Извлеките колонку `wt` датафрейма `mtcars` - массу автомобиля в тысячах фунтов.

```
mtcars$wt
```

```
 [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070
[13] 3.730 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840
[25] 3.845 1.935 2.140 1.513 3.170 2.770 3.570 2.780
```

- Извлеките колонки из `mtcars` в следующем порядке: `hp`, `mpg`, `cyl`.

```
mtcars[, c("hp", "mpg", "cyl")]
```

```
                   hp  mpg cyl
Mazda RX4         110 21.0   6
Mazda RX4 Wag     110 21.0   6
Datsun 710         93 22.8   4
Hornet 4 Drive    110 21.4   6
Hornet Sportabout 175 18.7   8
Valiant           105 18.1   6
Duster 360        245 14.3   8
Merc 240D          62 24.4   4
Merc 230           95 22.8   4
Merc 280          123 19.2   6
Merc 280C         123 17.8   6
Merc 450SE        180 16.4   8
Merc 450SL        180 17.3   8
Merc 450SLC       180 15.2   8
Cadillac Fleetwood  205 10.4   8
Lincoln Continental 215 10.4   8
Chrysler Imperial   230 14.7   8
Fiat 128           66 32.4   4
Honda Civic        52 30.4   4
Toyota Corolla     65 33.9   4
Toyota Corona      97 21.5   4
Dodge Challenger  150 15.5   8
AMC Javelin       150 15.2   8
Camaro Z28        245 13.3   8
Pontiac Firebird  175 19.2   8
Fiat X1-9          66 27.3   4
Porsche 914-2      91 26.0   4
Lotus Europa      113 30.4   4
Ford Pantera L    264 15.8   8
Ferrari Dino      175 19.7   6
Maserati Bora     335 15.0   8
Volvo 142E        109 21.4   4
```

- Посчитайте *количество* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
sum(mtcars$cyl == 4)
```

```
[1] 11
```

- Посчитайте *долю* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
mean(mtcars$cyl == 4)
```

```
[1] 0.34375
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) в датафрейме `mtcars`.

```
mtcars[mtcars$hp >= 100, ]
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (hp) и 4 цилиндрами (cyl) в датафрейме mtcars.

```
mtcars[mtcars$hp >= 100 & mtcars$cyl == 4, ]
```

```
              mpg cyl  disp  hp drat    wt qsec vs am gear carb
Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Volvo 142E   21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

- Посчитайте максимальную массу (wt) автомобиля в выборке, воспользовавшись функцией max():

```
max(mtcars$wt)
```

```
[1] 5.424
```

- Посчитайте минимальную массу (wt) автомобиля в выборке, воспользовавшись функцией min():

```
min(mtcars$wt)
```

```
[1] 1.513
```

- Найдите строчку датафрейма mtcars с самым легким автомобилем.

```
mtcars[mtcars$wt == min(mtcars$wt), ]
```

```
              mpg cyl disp  hp drat    wt qsec vs am gear carb
Lotus Europa 30.4   4 95.1 113 3.77 1.513 16.9  1  1    5    2
```

- Извлеките строчки датафрейма mtcars с автомобилями, масса которых ниже средней массы.

```
mtcars[mtcars$wt < mean(mtcars$wt), ]
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Масса автомобиля указана в тысячах фунтов. Создайте колонку `wt_kg` с массой автомобиля в килограммах. Результат округлите до целых значений с помощью функции `round()`.

  1 фунт = 0.45359237 кг.

```
mtcars$wt_kg <- round(mtcars$wt * 1000 * 0.45359237)
mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb wt_kg
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4  1188
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4  1304
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1  1052
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1  1458
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2  1560
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1  1569
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4  1619
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2  1447
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2  1429
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4  1560
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4  1560
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3  1846
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3  1692
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3  1715
Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98  0  0    3    4  2381
Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82  0  0    3    4  2460
```

```
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4  2424
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1   998
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2   733
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1   832
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1  1118
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2  1597
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2  1558
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4  1742
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2  1744
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1   878
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2   971
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2   686
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4  1438
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6  1256
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8  1619
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2  1261
```

27.10 Условные конструкции

- Создайте вектор `vec5`, скопировав следующий код:

```
vec5 <- c(5, 20, 30, 0, 2, 9)
```

- Создайте новый строковый вектор, где на месте чисел больше 10 в `vec5` будет стоять “большое число”, а на месте остальных чисел – “маленькое число”.

```
ifelse(vec5 > 10, "большое число", "маленькое число")
```

```
[1] "маленькое число" "большое число"   "большое число"   "маленькое число"
[5] "маленькое число" "маленькое число"
```

- Загрузите файл heroes_information.csv в переменную `heroes`.

```
heroes <- read.csv("data/heroes_information.csv",
                   stringsAsFactors = FALSE,
                   na.strings = c("-", "-99"))
```

- Создайте новою колонку `hair` в `heroes`, в которой будет значение `"Bold"` для тех супергероев, у которых в колонке `Hair.color` стоит `"No Hair"`, и значение `"Hairy"` во всех остальных случаях.

```
heroes$hair <- ifelse(heroes$Hair.color == "No Hair", "Bold", "Hairy")
head(heroes)
```

```
  X          name Gender Eye.color              Race Hair.color Height
1 0        A-Bomb   Male    yellow             Human    No Hair    203
2 1    Abe Sapien   Male      blue     Icthyo Sapien    No Hair    191
3 2      Abin Sur   Male      blue           Ungaran    No Hair    185
4 3   Abomination   Male     green Human / Radiation    No Hair    203
5 4       Abraxas   Male      blue     Cosmic Entity      Black     NA
6 5 Absorbing Man   Male      blue             Human    No Hair    193
         Publisher Skin.color Alignment Weight  hair
1     Marvel Comics       <NA>      good    441  Bold
2 Dark Horse Comics       blue      good     65  Bold
3         DC Comics        red      good     90  Bold
4     Marvel Comics       <NA>       bad    441  Bold
5     Marvel Comics       <NA>       bad     NA Hairy
6     Marvel Comics       <NA>       bad    122  Bold
```

- Создайте новою колонку `tall` в `heroes`, в которой будет значение `"tall"` для тех супергероев, у которых в колонке `Height` стоит число больше 190, значение `"short"` для тех супергероев, у которых в колонке `Height` стоит число меньше 170, и значение `"middle"` во всех остальных случаях.

```
# heroes$tall <- dplyr::case_when(
#   heroes$Height > 190 ~ "tall",
#   heroes$Height < 170 ~ "short",
#   TRUE ~ "middle"
# )
heroes$tall <- ifelse(heroes$Height > 190,
                      "tall",
                      ifelse(heroes$Height < 170,
                             "short",
                             "middle"))
```

27.11 Создание функций

- Создайте функцию `plus_one()`, которая принимает число и возвращает это же число + 1.

```
plus_one <- function(x) x + 1
```

- Проверьте функцию `plus_one()` на числе 41.

```
plus_one(41)
```

```
[1] 42
```

- Создайте функцию `circle_area()`, которая вычисляет площадь круга по радиусу согласно формуле πr^2.

```
circle_area <- function(r) pi * r ^ 2
```

- Посчитайте площадь круга с радиусом 5.

```
circle_area(5)
```

```
[1] 78.53982
```

- Создайте функцию `cels2fahr()`, которая будет превращать градусы по Цельсию в градусы по Фаренгейту.

```
cels2fahr <- function(x) x * 9 / 5 + 32
```

- Проверьте на значениях -100, -40 и 0, что функция `cels2fahr()` работает корректно.

```
cels2fahr(c(-100, -40, 0))
```

```
[1] -148  -40   32
```

- Напишите функцию `highlight()`, которая принимает на входе строковый вектор, а возвращает тот же вектор, но дополненный значением `"***"` в начале и конце вектора. Лучше всего это рассмотреть на примере:

```
highlight <- function(x) c("***", x, "***")
```

```
highlight(c(" ", "   !"))
```

```
[1] "***"     " "       "   !" "***"
```

- Теперь сделайте функцию `highlight` более гибкой. Добавьте в нее параметр `wrapper =`, который по умолчанию равен `"***"`. Значение параметра `wrapper =` и будет вставлено в начало и конец вектора.

```
highlight <- function(x, wrapper = "***") c(wrapper, x, wrapper)
```

- Проверьте написанную функцию на векторе `c(" ", "   !")`.

```
highlight(c(" ", "   !"))
```

```
[1] "***"     " "       "   !" "***"
```

```
highlight(c(" ", "   !"), wrapper = "__")
```

```
[1] "__"      " "       "   !" "__"
```

- Создайте функцию `na_n()`, которая будет возвращать количество `NA` в векторе.

```
na_n <- function(x) sum(is.na(x))
```

- Проверьте функцию `na_n()` на векторе:

```
na_n(c(NA, 3:5, NA, 2, NA))
```

```
[1] 3
```

- Напишите функцию `factors()`, которая будет возвращать все делители числа в виде числового вектора.

  Здесь может понадобиться оператор для получения остатка от деления: `%%`.

```
factors <- function(x) (1:x)[x %% (1:x) == 0]
```

- Проверьте функцию `factors()` на простых и сложных числах:

```
factors(3)
```

```
[1] 1 3
```

```
factors(161)
```

```
[1]   1   7  23 161
```

```
factors(1984)
```

```
 [1]    1    2    4    8   16   31   32   62   64  124  248  496  992 1984
```

- *Напишите функцию `is_prime()`, которая проверяет, является ли число простым.

  Здесь может пригодиться функция `any()` - она возвращает `TRUE`, если в векторе есть хотя бы один `TRUE`.

```
is_prime <- function(x) !any(x%%(2:(x-1)) == 0)
#is_prime <- function(x) length(factors(x)) == 2 #                    factors()
```

- Проверьте какие года были для нас простыми, а какие нет:

```
is_prime(2017)
```

```
[1] TRUE
```

```
is_prime(2019)
```

```
[1] FALSE
```

```
2019/3 #2019        3
```

```
[1] 673
```

```
is_prime(2020)
```

```
[1] FALSE
```

```
is_prime(2021)
```

```
[1] FALSE
```

- *Создайте функцию `monotonic()`, которая возвращает `TRUE`, если значения в векторе не убывают (то есть каждое следующее - больше или равно предыдущему) или не возврастают.

  Полезная функция для этого — `diff()` — возвращает разницу соседних значений.

```
monotonic <- function(x) all(diff(x)>=0) | all(diff(x)<=0)
```

```
monotonic(1:7)
```

```
[1] TRUE
```

```
monotonic(c(1:5,5:1))
```

```
[1] FALSE
```

```
monotonic(6:-1)
```

```
[1] TRUE
```

```
monotonic(c(1:5, rep(5, 10), 5:10))
```

```
[1] TRUE
```

Бинарные операторы типа + или %in% тоже представляют собой функции. Более того, мы можем создавать свои бинарные операторы! В этом нет особой сложности — нужно все так же создавать функцию (для двух переменных), главное окружать их % и название обрамлять обратными штрихами `. Например, можно сделать свой бинарный оператор %notin%, который будет выдавать TRUE, если значения слева *нет* в векторе справа:

```
`%notin%` <- function(x, y) ! (x %in% y)
1:10 %notin% c(1, 4, 5)
```

```
 [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- *Создайте бинарный оператор %without%, который будет возвращать все значения вектора слева без значений вектора справа.

```
`%without%` <- function(x, y) x[!x %in% y]
```

```
c(" ", " ", " ", "   ", " ", "   ") %without% c(" ", " ")
```

```
[1] " "     "   " " " "     "   "
```

- *Создайте бинарный оператор %between%, который будет возвращать TRUE, если значение в векторе слева накходится в *диапазоне* значений вектора справа:

```
`%between%` <- function(x, y) x >= min(y) & x <= max(y)
```

```
1:10 %between% c(1, 4, 5)
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

27.12 Проверка на адекватность

- Создайте функцию `trim()`, которая будет возвращать вектор без первого и последнего значения (вне зависимости от типа данных).

```
trim <- function(x) x[c(-1, -length(x))]
```

- Проверьте, что функция `trim()` работает корректно:

```
trim(1:7)
```

```
[1] 2 3 4 5 6
```

```
trim(letters)
```

```
 [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[20] "u" "v" "w" "x" "y"
```

- Теперь добавьте в функцию `trim()` параметр `n =` со значением по умолчанию 1. Этот параметр будет обозначать сколько значений нужно отрезать слева и справа от вектора.

```
trim <- function(x, n = 1) x[c(-1:-n, (-length(x)+n-1):-length(x))]
```

- Проверьте полученную функцию:

```
trim(letters)
```

```
 [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[20] "u" "v" "w" "x" "y"
```

```
trim(letters, n = 2)
```

```
 [1] "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u"
[20] "v" "w" "x"
```

- Сделайте так, чтобы функция `trim()` работала корректно с `n = 0`, т.е. функция возвращала бы исходный вектор без изменений.

```
trim <- function(x, n = 1) {
  if (n == 0) return(x)
  x[c(-1:-n, (-length(x)+n-1):-length(x))]
}
```

```
trim(letters, n = 0)
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

- *Теперь добавьте проверку на адекватность входных данных: функция `trim()` должна выдавать ошибку, если `n` = меньше нуля или если `n` = слишком большой и отрезает все значения вектора:

```
trim <- function(x, n = 1) {
  if (n < 0) stop("n                !")
  l <- length(x)
  if (n > ceiling(l/2) - 1) stop("n          !")
  if (n == 0) return(x)
  x[c(-1:-n, (-l+n-1):-l)]
}
```

- *Проверьте полученную функцию `trim()`:

```
trim(1:6, 3)
```

```
Error in trim(1:6, 3): n          !
```

```
trim(1:6, -1)
```

```
Error in trim(1:6, -1): n                !
```

27.13 Семейство функций apply()

- Создайте матрицу `M2`:

```
M2 <- matrix(c(20:11, 11:20), nrow = 5)
M2
```

```
     [,1] [,2] [,3] [,4]
[1,]   20   15   11   16
[2,]   19   14   12   17
[3,]   18   13   13   18
[4,]   17   12   14   19
[5,]   16   11   15   20
```

- Посчитайте максимальное значение матрицы `M2` по каждой строчке.

```
apply(M2, 1, max)
```

```
[1] 20 19 18 19 20
```

- Посчитайте максимальное значение матрицы `M2` по каждому столбцу.

```
apply(M2, 2, max)
```

```
[1] 20 15 15 20
```

- Посчитайте среднее значение матрицы `M2` по каждой строке.

```
apply(M2, 1, mean)
```

```
[1] 15.5 15.5 15.5 15.5 15.5
```

- Посчитайте среднее значение матрицы `M2` по каждому столбцу.

```
apply(M2, 2, mean)
```

```
[1] 18 13 13 18
```

- Создайте список `list3`:

```
list3 <- list(
  a = 1:5,
  b = 0:20,
  c = 4:24,
  d = 6:3,
  e = 6:25
  )
```

- Найдите максимальное значение каждого вектора списка `list3`.

```
sapply(list3, max)
```

```
a  b  c  d  e
5 20 24  6 25
```

- Посчитайте сумму каждого вектора списка `list3`.

```
sapply(list3, sum)
```

```
 a   b   c   d   e
15 210 294  18 310
```

- Посчитайте длину каждого вектора списка `list3`.

```
sapply(list3, length)
```

```
a  b  c  d  e
5 21 21  4 20
```

- Напишите функцию `max_item()`, которая будет принимать на входе список, а возвращать - (первый) самый длинный его элемент.

  Для этого вам может понадобиться функция `which.max()`, которая возвращает индекс максимального значения (первого, если их несколько).

```
max_item <- function (x) x[[which.max(sapply(x, length))]]
```

- Проверьте функцию `max_item()` на списке `list3`.

```
max_item(list3)
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Теперь мы сделаем сложный список `list4`:

```
list4 <- list(1:3, 3:40, list3)
```

- Посчитайте длину каждого вектора в списке, в т.ч. для списка внутри. Результат должен быть списком с такой же структорой, как и изначальный список `list4`.

  Для этого может понадобиться функция `rapply()`: **recursive lapply**

```
rapply(list4, length, how = "list")
```

```
[[1]]
[1] 3

[[2]]
[1] 38

[[3]]
[[3]]$a
[1] 5

[[3]]$b
[1] 21

[[3]]$c
[1] 21

[[3]]$d
[1] 4

[[3]]$e
[1] 20
```

- *Загрузите набор данных `heroes` и посчитайте, сколько `NA` в каждом из столбцов.

  Для этого удобно использовать ранее написанную функцию `na_n()`.

```
sapply(heroes, na_n)
```

```
        X       name     Gender  Eye.color       Race Hair.color     Height
        0          0         29        172        304        172        217
Publisher Skin.color  Alignment     Weight       hair       tall
        0        662          7        239        172        217
```

- *Используя ранее написанную функцию `is_prime()`, напишите функцию `prime_numbers()`, которая будет возвращать все простые числа до выбранного числа.

```
is_prime <- function(x) !any(x %% (2:(x - 1)) == 0)
prime_numbers <- function(x) (2:x)[sapply(2:x, is_prime)]
```

```
prime_numbers(200)
```

```
 [1]   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71
[20]  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167
[39] 173 179 181 191 193 197 199
```

27.14 magrittr::%>%

```
library(tidyverse)
```

- Перепишите следующие выражения, используя %>%:

```
1:10 %>%
  sum() %>%
  sqrt()
```

```
[1] 7.416198
```

```
  -5:5 %>%
    min() %>%
    abs()
```

```
[1] 5
```

```
  2 %>% c("      ", ., "   ", sqrt(.))
```

```
[1] "      "       "2"               "   "           "1.4142135623731"
```

```
  10:39 %>%
    matrix(nrow = 5) %>%
    apply(1, mean)
```

```
[1] 22.5 23.5 24.5 25.5 26.5
```

27.15 Выбор столбцов: `dplyr::select()`

Для выполнения следующих заданий нам понадобятся датасеты `heroes` и `powers`, которые можно загрузить, используя следующие команды:

```
  library(tidyverse)
  heroes <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
                     na = c("-", "-99"))
  powers <- read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/d
```

- Выберете первые 4 столбца в `powers`.

```
  powers %>%
    select(1:4)
```

```
# A tibble: 667 x 4
   hero_names    Agility `Accelerated Healing` `Lantern Power Ring`
   <chr>         <lgl>   <lgl>                 <lgl>
 1 3-D Man       TRUE    FALSE                 FALSE
 2 A-Bomb        FALSE   TRUE                  FALSE
```

```
 3 Abe Sapien    TRUE    TRUE                  FALSE
 4 Abin Sur      FALSE   FALSE                 TRUE
 5 Abomination   FALSE   TRUE                  FALSE
 6 Abraxas       FALSE   FALSE                 FALSE
 7 Absorbing Man FALSE   FALSE                 FALSE
 8 Adam Monroe   FALSE   TRUE                  FALSE
 9 Adam Strange  FALSE   FALSE                 FALSE
10 Agent Bob     FALSE   FALSE                 FALSE
# i 657 more rows
```

- Выберите все столбцы от `Reflexes` до `Empathy` в тиббле `powers`:

```
powers %>%
  select(Reflexes:Empathy)
```

```
# A tibble: 667 x 7
   Reflexes Invulnerability `Energy Constructs` `Force Fields` `Self-Sustenance`
   <lgl>    <lgl>           <lgl>               <lgl>          <lgl>
 1 FALSE    FALSE           FALSE               FALSE          FALSE
 2 FALSE    FALSE           FALSE               FALSE          TRUE
 3 TRUE     FALSE           FALSE               FALSE          FALSE
 4 FALSE    FALSE           FALSE               FALSE          FALSE
 5 FALSE    TRUE            FALSE               FALSE          FALSE
 6 FALSE    TRUE            FALSE               FALSE          FALSE
 7 FALSE    TRUE            FALSE               FALSE          FALSE
 8 FALSE    FALSE           FALSE               FALSE          FALSE
 9 FALSE    FALSE           FALSE               FALSE          FALSE
10 FALSE    FALSE           FALSE               FALSE          FALSE
# i 657 more rows
# i 2 more variables: `Anti-Gravity` <lgl>, Empathy <lgl>
```

- Выберите все столбцы тиббла `powers` кроме первого (`hero_names`):

```
powers %>%
select(!hero_names)
```

```
# A tibble: 667 x 167
   Agility `Accelerated Healing` `Lantern Power Ring` `Dimensional Awareness`
   <lgl>   <lgl>                 <lgl>                <lgl>
 1 TRUE    FALSE                 FALSE                FALSE
 2 FALSE   TRUE                  FALSE                FALSE
```

```
 3 TRUE    TRUE                 FALSE                FALSE
 4 FALSE   FALSE                TRUE                 FALSE
 5 FALSE   TRUE                 FALSE                FALSE
 6 FALSE   FALSE                FALSE                TRUE
 7 FALSE   FALSE                FALSE                FALSE
 8 FALSE   TRUE                 FALSE                FALSE
 9 FALSE   FALSE                FALSE                FALSE
10 FALSE   FALSE                FALSE                FALSE
# i 657 more rows
# i 163 more variables: `Cold Resistance` <lgl>, Durability <lgl>,
#   Stealth <lgl>, `Energy Absorption` <lgl>, Flight <lgl>,
#   `Danger Sense` <lgl>, `Underwater breathing` <lgl>, Marksmanship <lgl>,
#   `Weapons Master` <lgl>, `Power Augmentation` <lgl>,
#   `Animal Attributes` <lgl>, Longevity <lgl>, Intelligence <lgl>,
#   `Super Strength` <lgl>, Cryokinesis <lgl>, Telepathy <lgl>, ...
```

27.16 Выбор строк: `dplyr::slice()` и `dplyr::filter()`

- Выберите только те строчки, в которых содержится информация о супергероях тяжелее 500 кг.

```
heroes %>%
  filter(Weight > 500)
```

```
# A tibble: 6 x 11
   ...1 name       Gender `Eye color` Race         `Hair color` Height Publisher
  <dbl> <chr>      <chr>  <chr>       <chr>        <chr>         <dbl> <chr>
1   203 Darkseid   Male   red         New God      No Hair       267   DC Comics
2   283 Giganta    Female green       <NA>         Red            62.5 DC Comics
3   331 Hulk       Male   green       Human / Rad~ Green         244   Marvel C~
4   373 Juggernaut Male   blue        Human        Red           287   Marvel C~
5   549 Red Hulk   Male   yellow      Human / Rad~ Black         213   Marvel C~
6   575 Sasquatch  Male   red         <NA>         Orange        305   Marvel C~
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о *женщинах*-супергероях тяжелее 500 кг.

```
heroes %>%
  filter(Weight > 500 & Gender == "Female")
```

```
# A tibble: 1 x 11
   ...1 name    Gender `Eye color` Race  `Hair color` Height Publisher
  <dbl> <chr>   <chr>  <chr>       <chr> <chr>         <dbl> <chr>
1   283 Giganta Female green       <NA>  Red            62.5 DC Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о супергероях человеческой расы (`"Human"`) женского пола. Из этих супергероев возьмите первые 5.

```
heroes %>%
  filter(Race == "Human" & Gender == "Female") %>%
  slice(1:5)
```

```
# A tibble: 5 x 11
   ...1 name         Gender `Eye color` Race  `Hair color` Height Publisher
  <dbl> <chr>        <chr>  <chr>       <chr> <chr>         <dbl> <chr>
1    38 Arachne      Female blue        Human Blond           175 Marvel Comics
2    63 Batgirl      Female green       Human Red             170 DC Comics
3    65 Batgirl IV   Female green       Human Black           165 DC Comics
4    72 Batwoman V   Female green       Human Red             178 DC Comics
5    96 Black Canary Female blue        Human Blond           165 DC Comics
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

27.17 Сортировка строк: `dplyr::arrange()`

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по возрастанию* `Height`.

```
heroes %>%
  select(name, Gender, Height) %>%
  arrange(Height)
```

```
# A tibble: 734 x 3
   name            Gender Height
   <chr>           <chr>   <dbl>
 1 Utgard-Loki     Male     15.2
 2 Bloodwraith     Male     30.5
 3 King Kong       Male     30.5
 4 Anti-Monitor    Male     61
```

```
 5 Giganta         Female   62.5
 6 Krypto          Male     64
 7 Yoda            Male     66
 8 Jack-Jack       Male     71
 9 Howard the Duck Male     79
10 Godzilla        <NA>    108
# i 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по убыванию* `Height`.

```
heroes %>%
  select(name, Gender, Height) %>%
  arrange(desc(Height))
```

```
# A tibble: 734 x 3
   name          Gender Height
   <chr>         <chr>   <dbl>
 1 Fin Fang Foom Male      975
 2 Galactus      Male      876
 3 Groot         Male      701
 4 MODOK         Male      366
 5 Wolfsbane     Female    366
 6 Onslaught     Male      305
 7 Sasquatch     Male      305
 8 Ymir          Male      305.
 9 Rey           Female    297
10 Juggernaut    Male      287
# i 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки сначала по `Gender`, затем *по убыванию* `Height`.

```
heroes %>%
  select(name, Gender, Height) %>%
  arrange(Gender, desc(Height))
```

```
# A tibble: 734 x 3
   name      Gender Height
   <chr>     <chr>   <dbl>
 1 Wolfsbane Female    366
```

```
 2 Rey       Female    297
 3 Bloodaxe  Female    218
 4 Thundra   Female    218
 5 Hela      Female    213
 6 Frenzy    Female    211
 7 She-Hulk  Female    201
 8 Ardina    Female    193
 9 Starfire  Female    193
10 Valkyrie  Female    191
# i 724 more rows
```

27.18 Уникальные значения: `dplyr::distinct()`

- Извлеките уникальные значения столбца `Eye color` из тиббла `heroes`.

```
heroes %>%
  distinct(`Eye color`)
```

```
# A tibble: 23 x 1
   `Eye color`
   <chr>
 1 yellow
 2 blue
 3 green
 4 brown
 5 <NA>
 6 red
 7 violet
 8 white
 9 purple
10 black
# i 13 more rows
```

- Извлеките уникальные значения столбца `Hair color` из тиббла `heroes`.

```
heroes %>%
  distinct(`Hair color`)
```

```
# A tibble: 30 x 1
   `Hair color`
   <chr>
 1 No Hair
 2 Black
 3 Blond
 4 Brown
 5 <NA>
 6 White
 7 Purple
 8 Orange
 9 Pink
10 Red
# i 20 more rows
```

27.19 Создание колонок: `dplyr::mutate()` и `dplyr::transmute()`

- Создайте колонку `height_m` с ростом супергероев в метрах, затем выберите только колонки `name` и `height_m`.

```
heroes %>%
  mutate(height_m = Height/100) %>%
  select(name, height_m)
```

```
# A tibble: 734 x 2
   name          height_m
   <chr>            <dbl>
 1 A-Bomb            2.03
 2 Abe Sapien        1.91
 3 Abin Sur          1.85
 4 Abomination       2.03
 5 Abraxas          NA
 6 Absorbing Man     1.93
 7 Adam Monroe      NA
 8 Adam Strange      1.85
 9 Agent 13          1.73
10 Agent Bob         1.78
# i 724 more rows
```

- Создайте новою колонку hair в heroes, в которой будет значение “Bold” для тех супергероев, у которых в колонке

Hair.color стоит “No Hair”, и значение “Hairy” во всех остальных случаях. Затем выберите только колонки `name`, `Hair color`, `hair`.

```
heroes %>%
  mutate(hair = ifelse(`Hair color` == "No Hair", "Bold", "Hairy")) %>%
  select(name, `Hair color`, hair)
```

```
# A tibble: 734 x 3
   name          `Hair color` hair
   <chr>         <chr>        <chr>
 1 A-Bomb        No Hair      Bold
 2 Abe Sapien    No Hair      Bold
 3 Abin Sur      No Hair      Bold
 4 Abomination   No Hair      Bold
 5 Abraxas       Black        Hairy
 6 Absorbing Man No Hair      Bold
 7 Adam Monroe   Blond        Hairy
 8 Adam Strange  Blond        Hairy
 9 Agent 13      Blond        Hairy
10 Agent Bob     Brown        Hairy
# i 724 more rows
```

27.20 Агрегация: `dplyr::group_by() %>% summarise()`

- Посчитайте количество супергероев по расам и отсортируйте по убыванию. Извлеките первые 5 строк.

```
heroes %>%
  count(Race, sort = TRUE) %>%
  slice(1:5)
```

```
# A tibble: 5 x 2
  Race              n
  <chr>         <int>
1 <NA>            304
2 Human           208
3 Mutant           63
4 God / Eternal    14
5 Cyborg           11
```

- Посчитайте средний рост по полу.

```
heroes %>%
  group_by(Gender) %>%
  summarise(height_mean = mean(Height, na.rm = TRUE))
```

```
# A tibble: 3 x 2
  Gender height_mean
  <chr>        <dbl>
1 Female        175.
2 Male          192.
3 <NA>          177.
```

27.21 Соединение датафреймов: *_join

Создайте тиббл `web_creators`, в котором будут супергерои, которые могут плести паутину, т.е. у них стоит `TRUE` в колонке `Web Creation` в тиббле `powers`.

```
powers_web <- powers %>%
  select(hero_names, `Web Creation`)
web_creators <- left_join(heroes, powers_web, by = c("name" = "hero_names")) %>%
  filter(`Web Creation`)
web_creators
```

```
# A tibble: 16 x 12
    ...1 name             Gender `Eye color` Race  `Hair color` Height Publisher
   <dbl> <chr>            <chr>  <chr>       <chr> <chr>         <dbl> <chr>
 1    33 Anti-Venom       Male   blue        Symb~ Blond           229 Marvel C~
 2    38 Arachne          Female blue        Human Blond           175 Marvel C~
 3   161 Carnage          Male   green       Symb~ Red             185 Marvel C~
 4   335 Hybrid           Male   brown       Symb~ Black           175 Marvel C~
 5   479 Mysterio         Male   brown       Human No Hair         180 Marvel C~
 6   580 Scarlet Spider ~ Male   brown       Clone Brown           193 Marvel C~
 7   597 Silk             Female brown       Human Black            NA Marvel C~
 8   620 Spider-Girl      Female blue        Human Brown           170 Marvel C~
 9   621 Spider-Gwen      Female blue        Human Blond           165 Marvel C~
10   622 Spider-Man       Male   hazel       Human Brown           178 Marvel C~
11   623 Spider-Man       <NA>   red         Human Brown           178 Marvel C~
12   624 Spider-Man       Male   brown       Human Black           157 Marvel C~
13   673 Toxin            Male   blue        Symb~ Brown           188 Marvel C~
14   674 Toxin            Male   black       Symb~ Blond           191 Marvel C~
15   689 Venom            Male   blue        Symb~ Strawberry ~    191 Marvel C~
```

```
16   692 Venompool        Male   <NA>        Symb~ <NA>            226 Marvel C~
# i 4 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>,
#   `Web Creation` <lgl>
```

- Найдите всех супергероев, которые присутствуют в `heroes`, но отсутствуют в `powers`. Ответом должен быть строковый вектор с именами супергероев.

```
anti_join(heroes, powers, by = c("name" = "hero_names")) %>%
  pull(name)
```

```
 [1] "Agent 13"          "Alfred Pennyworth" "Arsenal"
 [4] "Batgirl III"       "Batgirl V"         "Beetle"
 [7] "Black Goliath"     "Black Widow II"    "Blaquesmith"
[10] "Bolt"              "Boomer"            "Box"
[13] "Box III"           "Captain Mar-vell"  "Cat II"
[16] "Cecilia Reyes"     "Clea"              "Clock King"
[19] "Colin Wagner"      "Colossal Boy"      "Corsair"
[22] "Cypher"            "Danny Cooper"      "Darkside"
[25] "ERG-1"             "Fixer"             "Franklin Storm"
[28] "Giant-Man"         "Giant-Man II"      "Goliath"
[31] "Goliath"           "Goliath"           "Guardian"
[34] "Hawkwoman"         "Hawkwoman II"      "Hawkwoman III"
[37] "Howard the Duck"   "Jack Bauer"        "Jesse Quick"
[40] "Jessica Sanders"   "Jigsaw"            "Jyn Erso"
[43] "Kid Flash II"      "Kingpin"           "Meteorite"
[46] "Mister Zsasz"      "Mogo"              "Moloch"
[49] "Morph"             "Nite Owl II"       "Omega Red"
[52] "Paul Blart"        "Penance"           "Penance I"
[55] "Plastic Lad"       "Power Man"         "Renata Soliz"
[58] "Ronin"             "Shrinking Violet"  "Snake-Eyes"
[61] "Spider-Carnage"    "Spider-Woman II"   "Stacy X"
[64] "Thunderbird II"    "Two-Face"          "Vagabond"
[67] "Vision II"         "Vulcan"            "Warbird"
[70] "White Queen"       "Wiz Kid"           "Wondra"
[73] "Wyatt Wingfoot"    "Yellow Claw"
```

- Найдите всех супергероев, которые присутствуют в `powers`, но отсутствуют в `heroes`. Ответом должен быть строковый вектор с именами супергероев.

```
  anti_join(powers, heroes, by = c("hero_names" = "name")) %>%
    pull(hero_names)
```

```
 [1] "3-D Man"           "Bananaman"         "Bizarro-Girl"
 [4] "Black Vulcan"      "Blue Streak"       "Bradley"
 [7] "Clayface"          "Concrete"          "Dementor"
[10] "Doctor Poison"     "Fire"              "Hellgramite"
[13] "Lara Croft"        "Little Epic"       "Lord Voldemort"
[16] "Orion"             "Peek-a-Boo"        "Queen Hippolyta"
[19] "Reactron"          "SHDB"              "Stretch Armstrong"
[22] "TEST"              "Tommy Clarke"      "Tyrant"
```

27.22 Tidy data

- Для начала создайте тиббл `heroes_weight`, скопировав код:

```
  heroes_weight <- heroes %>%
    filter(Publisher %in% c("DC Comics", "Marvel Comics")) %>%
    group_by(Gender, Publisher) %>%
    summarise(weight_mean = mean(Weight, na.rm = TRUE)) %>%
    drop_na()
  heroes_weight
```

```
# A tibble: 4 x 3
# Groups:   Gender [2]
  Gender Publisher     weight_mean
  <chr>  <chr>               <dbl>
1 Female DC Comics            76.8
2 Female Marvel Comics        80.1
3 Male   DC Comics           113.
4 Male   Marvel Comics       134.
```

  Функция `drop_na()` позволяет выбросить все строчки, в которых встречается `NA`.

- Превратите тиббл `heroes_weight` в широкий тиббл:

```
  heroes_weight %>%
    pivot_wider(names_from = "Publisher", values_from = "weight_mean")
```

```
# A tibble: 2 x 3
# Groups:   Gender [2]
  Gender `DC Comics` `Marvel Comics`
  <chr>        <dbl>           <dbl>
1 Female        76.8            80.1
2 Male         113.            134.
```

- Затем превратите его обратно в длинный тиббл:

```
heroes_weight %>%
  pivot_wider(names_from = "Publisher", values_from = "weight_mean") %>%
  pivot_longer(cols = !Gender,
               names_to = "Publisher",
               values_to = "weight_mean")
```

```
# A tibble: 4 x 3
# Groups:   Gender [2]
  Gender Publisher     weight_mean
  <chr>  <chr>               <dbl>
1 Female DC Comics            76.8
2 Female Marvel Comics        80.1
3 Male   DC Comics           113.
4 Male   Marvel Comics       134.
```

- Сделайте `powers` длинным тиббломом с тремя колонками: `hero_names`, `power` (названгие суперсилы) и `has` (наличие суперсилы у данного супергероя).

```
powers %>%
  pivot_longer(cols = !hero_names,
               names_to = "power",
               values_to = "has")
```

```
# A tibble: 111,389 x 3
   hero_names power                has
   <chr>      <chr>                <lgl>
 1 3-D Man    Agility              TRUE
 2 3-D Man    Accelerated Healing  FALSE
 3 3-D Man    Lantern Power Ring   FALSE
 4 3-D Man    Dimensional Awareness FALSE
 5 3-D Man    Cold Resistance      FALSE
 6 3-D Man    Durability           FALSE
```

```
 7 3-D Man    Stealth              FALSE
 8 3-D Man    Energy Absorption    FALSE
 9 3-D Man    Flight               FALSE
10 3-D Man    Danger Sense         FALSE
# i 111,379 more rows
```

- Сделайте тиббл `powers` обратно широким, но с новой структурой: каждая строчка означает суперсилу, а каждая колонка - супергероя (за исключением первой колонки - названия суперсилы).

```
powers %>%
  pivot_longer(cols = !hero_names,
               names_to = "power",
               values_to = "has") %>%
  pivot_wider(names_from = hero_names,
              values_from = has)
```

```
# A tibble: 167 x 668
   power          `3-D Man` `A-Bomb` `Abe Sapien` `Abin Sur` Abomination Abraxas
   <chr>          <lgl>     <lgl>    <lgl>        <lgl>      <lgl>       <lgl>
 1 Agility        TRUE      FALSE    TRUE         FALSE      FALSE       FALSE
 2 Accelerated H~ FALSE     TRUE     TRUE         FALSE      TRUE        FALSE
 3 Lantern Power~ FALSE     FALSE    FALSE        TRUE       FALSE       FALSE
 4 Dimensional A~ FALSE     FALSE    FALSE        FALSE      FALSE       TRUE
 5 Cold Resistan~ FALSE     FALSE    TRUE         FALSE      FALSE       FALSE
 6 Durability     FALSE     TRUE     TRUE         FALSE      FALSE       FALSE
 7 Stealth        FALSE     FALSE    FALSE        FALSE      FALSE       FALSE
 8 Energy Absorp~ FALSE     FALSE    FALSE        FALSE      FALSE       FALSE
 9 Flight         FALSE     FALSE    FALSE        FALSE      FALSE       TRUE
10 Danger Sense   FALSE     FALSE    FALSE        FALSE      FALSE       FALSE
# i 157 more rows
# i 661 more variables: `Absorbing Man` <lgl>, `Adam Monroe` <lgl>,
#   `Adam Strange` <lgl>, `Agent Bob` <lgl>, `Agent Zero` <lgl>,
#   `Air-Walker` <lgl>, Ajax <lgl>, `Alan Scott` <lgl>, `Alex Mercer` <lgl>,
#   `Alex Woolsly` <lgl>, Alien <lgl>, `Allan Quatermain` <lgl>, Amazo <lgl>,
#   Ammo <lgl>, `Ando Masahashi` <lgl>, Angel <lgl>, `Angel Dust` <lgl>,
#   `Angel Salvadore` <lgl>, Angela <lgl>, `Animal Man` <lgl>, ...
```

27.23 Операции с несколькими колонками: across()

- Посчитайте количество NA в каждой колонке, группируя по полу (Gender).

```
na_n <- function(x) sum(is.na(x))
heroes %>%
  group_by(Gender) %>%
  summarise(across(everything(), na_n))
```

```
# A tibble: 3 x 11
  Gender  ...1  name `Eye color`  Race `Hair color` Height Publisher
  <chr>  <int> <int>       <int> <int>        <int>  <int>     <int>
1 Female     0     0          41    98           38     56         0
2 Male       0     0         121   184          123    147         0
3 <NA>       0     0          10    22           11     14         0
# i 3 more variables: `Skin color` <int>, Alignment <int>, Weight <int>
```

- Посчитайте количество NA в каждой колонке, которая заканчивается на "color", группируя по полу (Gender).

```
na_n <- function(x) sum(is.na(x))
heroes %>%
  group_by(Gender) %>%
  summarise(across(ends_with("color"), na_n))
```

```
# A tibble: 3 x 4
  Gender `Eye color` `Hair color` `Skin color`
  <chr>        <int>        <int>        <int>
1 Female          41           38          186
2 Male           121          123          449
3 <NA>            10           11           27
```

- Найдите (первую) самую длинную строчку для каждой колонки с character типом данных, группируя по полу (Gender).

  Для расчета количества значений в строке есть функция nchar(), для расчета индекса (первого) максимального значения есть функция which.max().

```
longest_char <- function(x) x[which.max(nchar(x))]
heroes %>%
  group_by(Gender) %>%
  summarise(across(where(is.character), longest_char))
```

```
# A tibble: 3 x 8
  Gender name    `Eye color` Race  `Hair color` Publisher `Skin color` Alignment
  <chr>  <chr>   <chr>       <chr> <chr>        <chr>     <chr>        <chr>
1 Female Negaso~ yellow (wi~ Huma~ Strawberry ~ Dark Hor~ orange       neutral
2 Male   Drax t~ yellow / r~ Dath~ Strawberry ~ Dark Hor~ orange / wh~ neutral
3 <NA>   Captai~ yellow (wi~ God ~ Orange / Wh~ Marvel C~ grey         neutral
```

- Создайте из тиббла `heroes` новый тиббл, в котором числовые значения `Height` и `Weight` заменены на следующие строковые значения: если у супергероя рост или вес выше среднего по колонке, то " ", если его/ее рост или вес ниже или равен среднему, то " ".

```
higher_than_average <- function(x) ifelse(x > mean(x, na.rm = TRUE),
                                          "  ",
                                          "  ")
heroes %>%
  mutate(across(c(Height, Weight),
                higher_than_average))
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>        <chr>  <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         ~ Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         ~ Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         ~ DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         ~ Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black        <NA>   Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         ~ Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond        <NA>   NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           ~ DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           ~ Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           ~ Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <chr>
```

- Создайте из тиббла `heroes` новый тиббл, в котором числовые значения `Height` и `Weight` заменены на следующие строковые

значения: если у супергероя *внутри соответствующей группы по полу* рост или вес выше среднего по колонке, то "X", если его/ее рост или вес ниже или равен среднему *внутри соответствующей группы по полу*, то "X", где *X* — соответствующий пол (Gender).

```
heroes %>%
  group_by(Gender) %>%
  mutate(across(c(Height, Weight),
                higher_than_average)) %>%
  ungroup() %>%
  mutate(across(c(Height, Weight),
                ~paste(., " ", Gender)))
```

```
# A tibble: 734 x 11
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>        <chr>  <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         ~ Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         ~ Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         ~ DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         ~ Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black        NA  ~ Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         ~ Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond        NA  ~ NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           ~ DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           ~ Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           ~ Marvel C~
# i 724 more rows
# i 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <chr>
```

27.24 Описательная статистика

Для выполнения задания создайте вектор height из колонки Height датасета heroes, удалив в нем NA.

```
height <- heroes %>%
  drop_na(Height) %>%
  pull(Height)
```

- Посчитайте среднее в векторе height.

```
mean(height)
```

```
[1] 186.7263
```

- Посчитайте усеченное среднее в векторе `height` с усечением 5% значений с обоих сторон.

```
mean(height, trim = 0.05)
```

```
[1] 182.5846
```

- Посчитайте медиану в векторе `height`.

```
median(height)
```

```
[1] 183
```

- Посчитайте стандартное отклонение в векторе `height`.

```
sd(height)
```

```
[1] 59.25189
```

- Посчитайте межквартильный размах в векторе `height`.

```
IQR(height)
```

```
[1] 18
```

- Посчитайте ассиметрию в векторе `height`.

```
psych::skew(height)
```

```
[1] 8.843432
```

Посчитайте эксцесс в векторе `height`.

```
psych::kurtosi(height)
```

```
[1] 105.0297
```

Примените функции для получения множественных статистик на векторе `height`.

```
summary(height)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   15.2   173.0   183.0   186.7   191.0   975.0
```

```
psych::describe(height)
```

```
   vars   n   mean    sd median trimmed   mad  min max range skew kurtosis   se
X1    1 517 186.73 59.25    183  182.02 11.86 15.2 975 959.8 8.84   105.03 2.61
```

```
skimr::skim(height)
```

Таблица 27.1: Data summary

Name	height
Number of rows	517
Number of columns	1
Column type frequency:	
numeric	1
Group variables	None

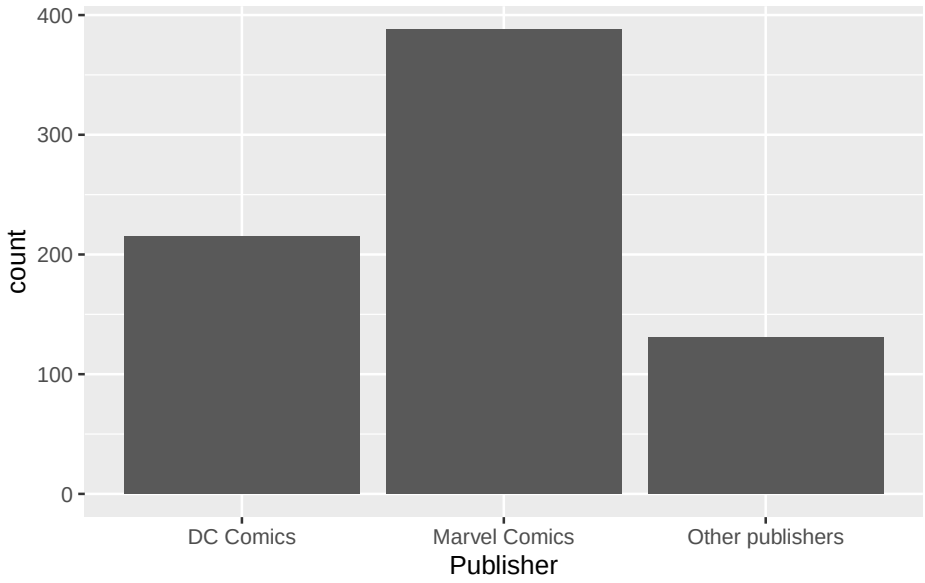
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
data	0	1	186.73	59.25	15.2	173	183	191	975	▇▁▁▁▁

27.25 Построение графиков в ggplot2

- Нарисуйте столбиковую диаграмму (`geom_bar()`), которая будет отражать количество супергероев издателей `"Marvel Comics"`, `"DC Comics"` и всех остальных (отдельным столбиком) из датасета `heroes`.

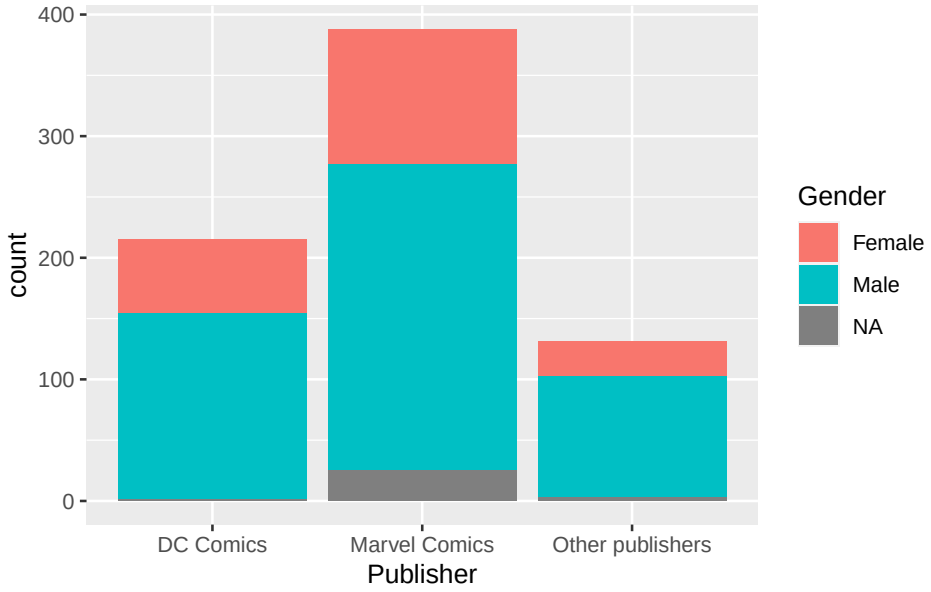
```
heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  #mutate(Publisher = fct_lump(Publisher, 2)) %>% #                    ,       forc
  ggplot(aes(x = Publisher)) +
  geom_bar()
```



- Добавьте к этой диаграме заливку цветом (`fill =`) в зависимости от распределения `Gender` внутри каждой группы.

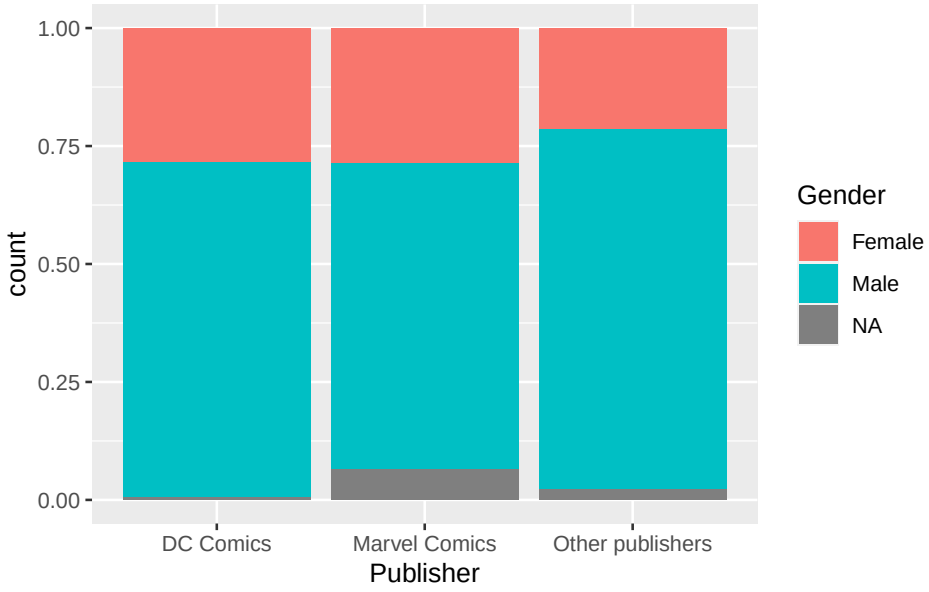
```
heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  #mutate(Publisher = fct_lump(Publisher, 2)) %>% #                    ,       forc
```

```
  ggplot(aes(x = Publisher, fill = Gender)) +
  geom_bar()
```



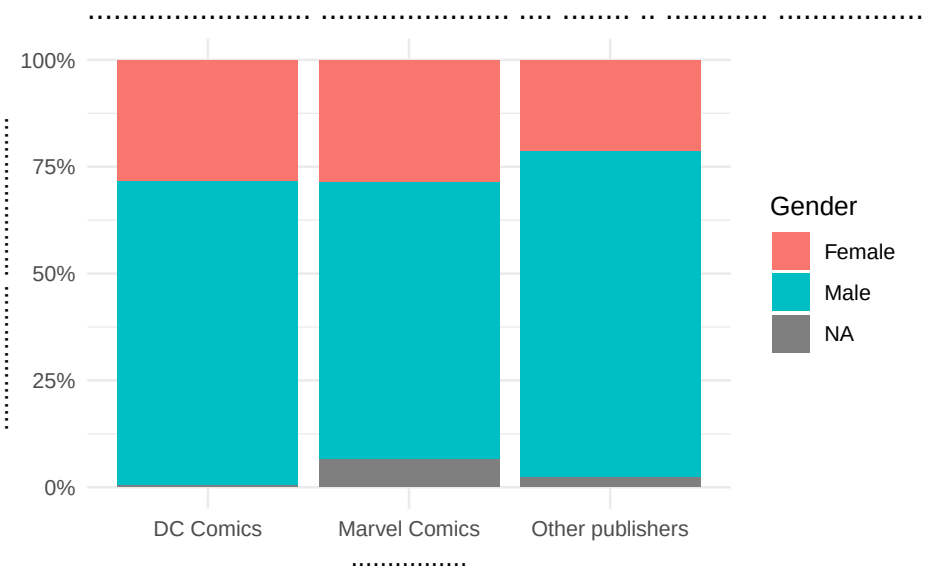
- Сделайте так, чтобы каждый столбик был максимальной высоты (`position = "fill"`).

```
heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  #mutate(Publisher = fct_lump(Publisher, 2)) %>% #                    ,       forcats
  ggplot(aes(x = Publisher, fill = Gender)) +
  geom_bar(position = "fill")
```



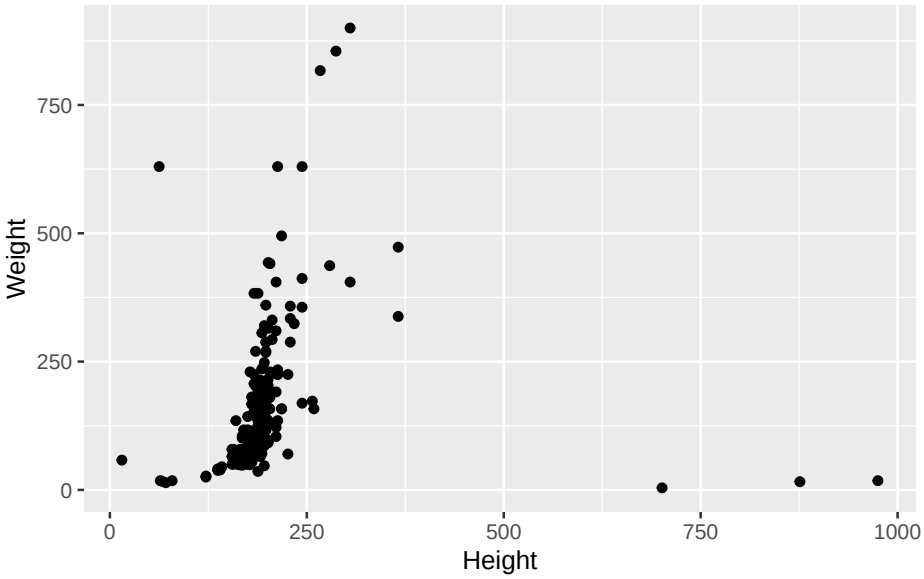
- Финализируйте график, задав ему описания осей (например, функция `labs()`), использовав процентную шкалу (`scale_y_continuous(labels = scales::percent)`) и задав тему `theme_minimal()`.

```
heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  #mutate(Publisher = fct_lump(Publisher, 2)) %>% #                    ,       for
  ggplot(aes(x = Publisher, fill = Gender)) +
  geom_bar(position = "fill") +
  labs(title = "                                    ",
       x = "     ",
       y = "             ")+
  scale_y_continuous(labels = scales::percent) +
  theme_minimal()
```



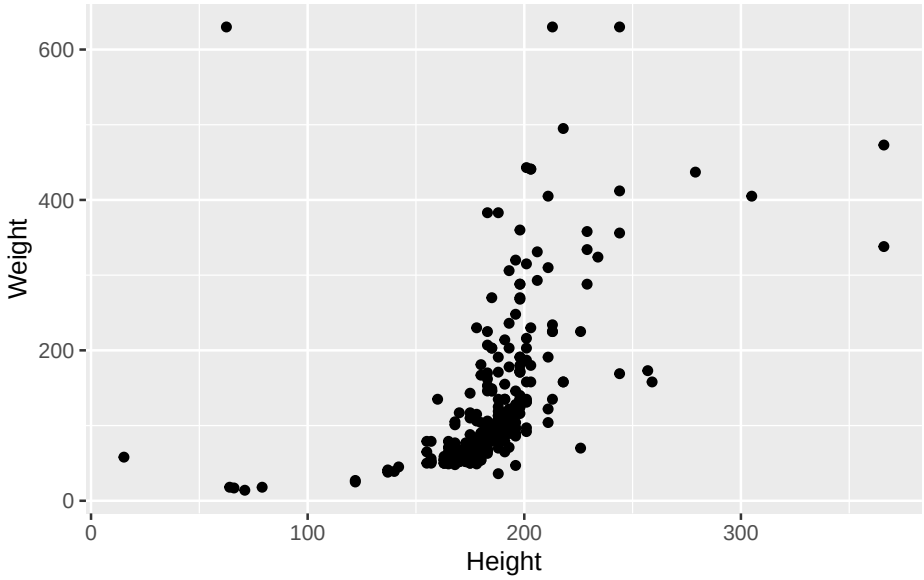
Создайте диаграмму рассеяния для датасета `heroes`, для которой координаты по оси *x* будут взяты из колонки `Height`, а координаты по оси *y* — из колонки `Weight`.

```
heroes %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point()
```



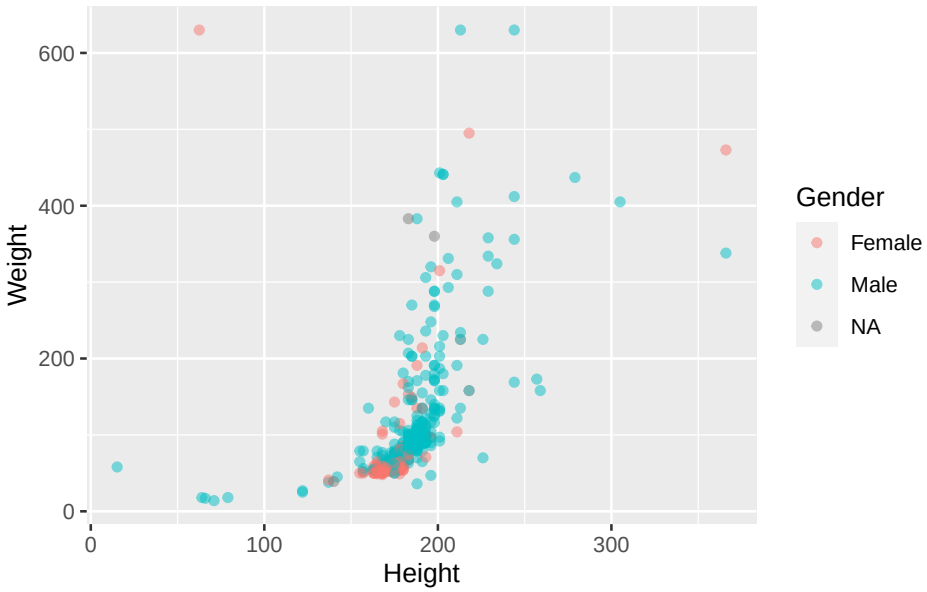
- Удалите с графика все экстремальные значения, для которых `Weight` больше или равен 700 или `Height` больше или равен 400. (*Подсказка:* это можно делать как средствами `ggplot2`, так и функцией `filter()` из `dplyr`).

```
heroes %>%
  filter(Weight < 700 & Height < 400) %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point()
```



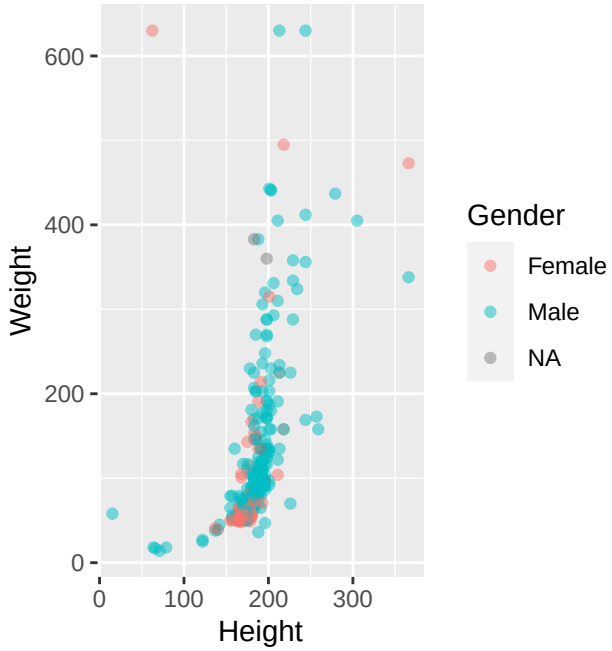
- Раскрасьте точки в зависимости от `Gender`, сделайте их полупрозрачными (параметр `alpha` =).

```
heroes %>%
  filter(Weight < 700 & Height < 400) %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point(aes(colour = Gender), alpha = 0.5)
```



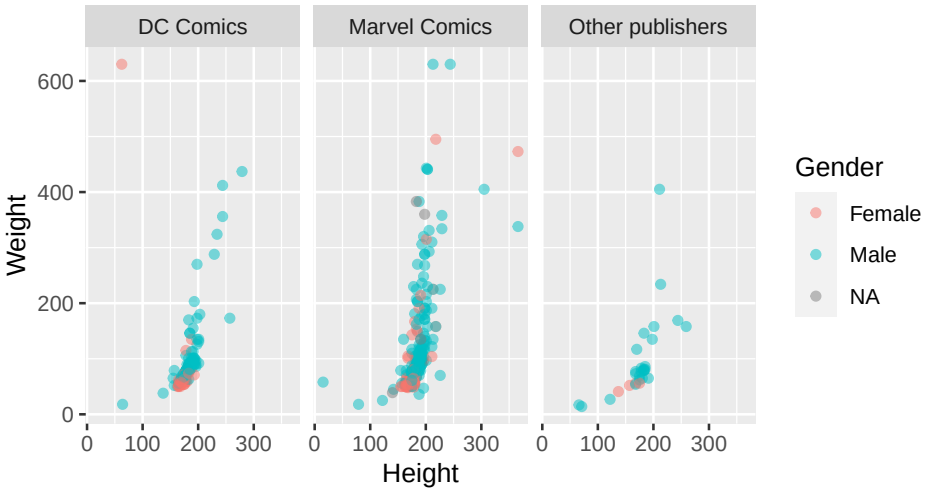
- Сделайте так, чтобы координатная плоскость имела соотношение 1:1 шкал по оси *x* и *y*. Этого можно добиться с помощью функции `coord_fixed()`.

```
heroes %>%
  filter(Weight < 700 & Height < 400) %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point(aes(colour = Gender), alpha = 0.5) +
  coord_fixed()
```



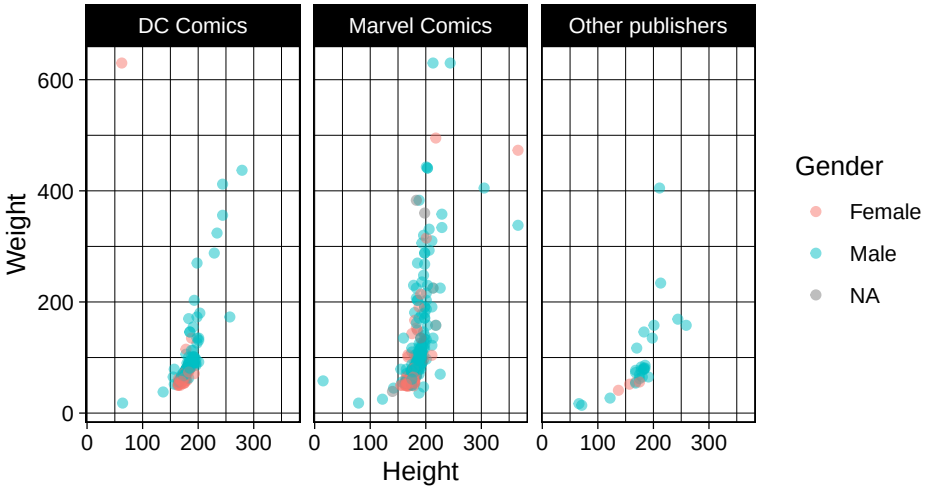
Разделите график (`facet_wrap()`) на три: для `"DC Comics","Marvel Comics"` и всех остальных.

```
heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  filter(Weight < 700 & Height < 400) %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point(aes(colour = Gender), alpha = 0.5) +
  coord_fixed() +
  facet_wrap(~Publisher)
```



- Используйте для графика тему `theme_linedraw()`.

```
heroes %>%
  mutate(Publisher = ifelse(Publisher %in% c("Marvel Comics", "DC Comics"),
                            Publisher,
                            "Other publishers")) %>%
  filter(Weight < 700 & Height < 400) %>%
  ggplot(aes(x = Height, y = Weight)) +
  geom_point(aes(colour = Gender), alpha = 0.5) +
  coord_fixed() +
  facet_wrap(~Publisher)+
  theme_linedraw()
```



- – Постройте новый график (или возьмите старый) по датасетам `heroes` и/или `powers` и сделайте его некрасивым! Чем *хуже* у вас получится график, тем лучше. Желательно, чтобы этот график был по-прежнему графиком, а не произведением абстрактного искусства. Разница очень тонкая, но она есть.

Вот несколько подсказок для этого задания:

1. Для вдохновения посмотрите на вот эти графики.
2. Для реально плохих графиков вам придется покопаться с настройками темы. Посмотрите подсказку по темам `?theme`, попытайтесь что-то поменять в теме.
3. Экспериментируйте с разными геомами и необычными их применениями.
4. По изучайте дополнения к `gpplot2`.
5. Попробуйте подготовить интересные данные для этого графика.

, !

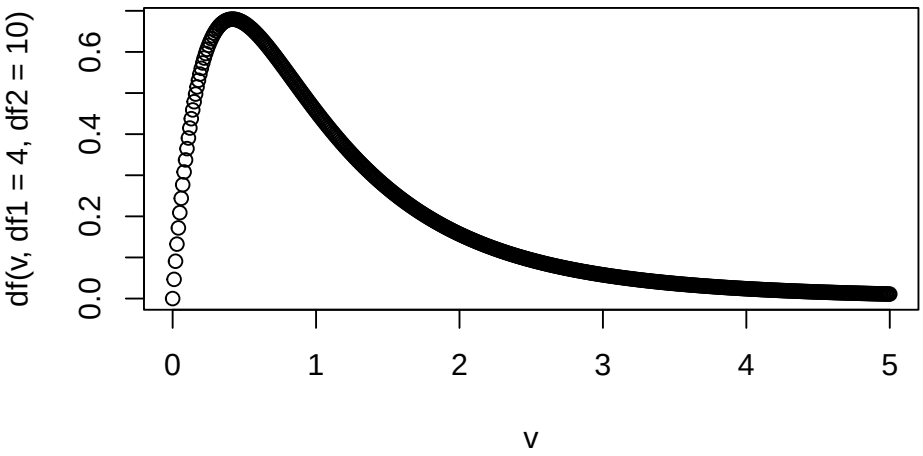
27.26 Распределения

Выберите любое непрерывное распределение из представленных в базовом пакете `stats` или же в любом другом пакете. Найти все распределения пакета `stats` можно с помощью `?Distributions`. Подберите для него какие-нибудь параметры или используйте параметры по умолчанию.

> Я возьму F-распределение с параметрами `df1 = 4` и `df = 10`, но вы можете выбрать другое распределение.

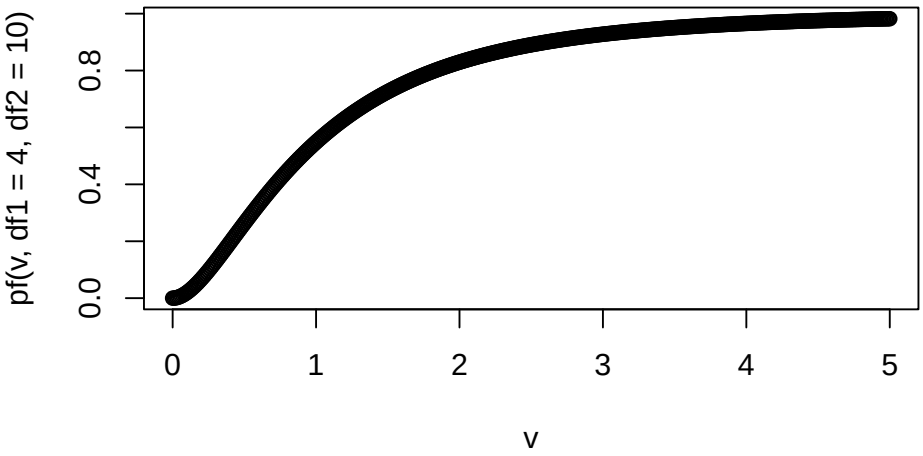
- Визуализируйте функцию плотности вероятности для выбранного распределения.

```
v <- seq(0, 5, 0.01)
plot(v, df(v, df1 = 4, df2 = 10))
```



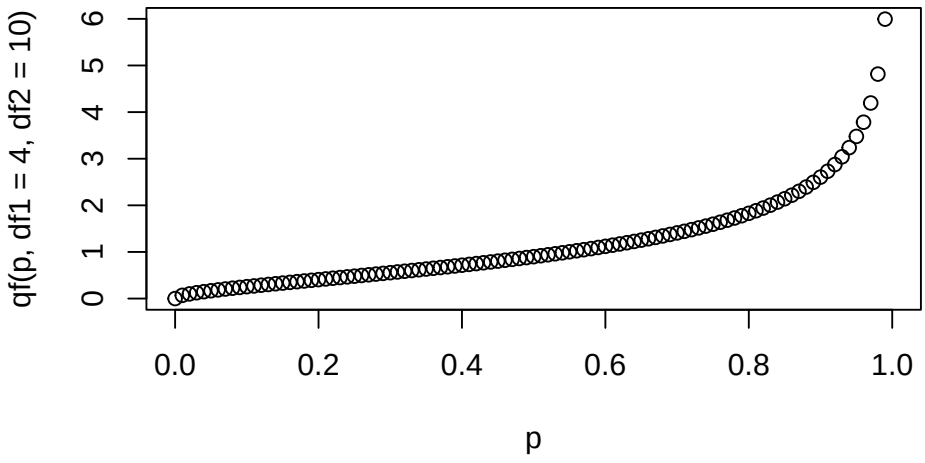
- Визуализируйте функцию накопленной плотности распределения для выбранной функции.

```
plot(v, pf(v, df1 = 4, df2 = 10))
```



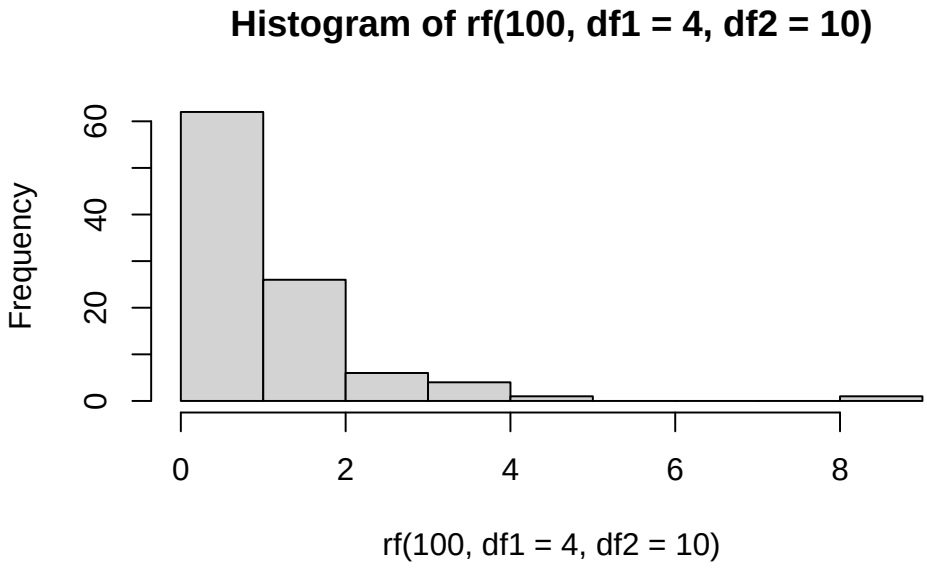
- Визуализируйте квантильную функцию для выбранного распределения.

```
p <- seq(0, 1, .01)
plot(p, qf(p, df1 = 4, df2 = 10))
```



- Сделайте выборку из 100 случайных значений из выбранного распределения и постройте гистограмму (функция `hist()`) для полученной выборки.

```
hist(rf(100, df1 = 4, df2 = 10))
```



27.27 Одновыборочный t-test

- Представьте, что наши супергерои из набора данных `heroes` — это выборка из генеральной совокупности всех написанных и ненаписанных супергероев. Проведите одновыборочный t-тест для веса супергероев и числа 100 — предположительного среднего веса в генеральной совокупности всех супергероев. Проинтерпретируйте результат.

```
t.test(heroes$Weight, mu = 100)
```

```
    One Sample t-test

data:  heroes$Weight
t = 2.6174, df = 494, p-value = 0.009133
alternative hypothesis: true mean is not equal to 100
95 percent confidence interval:
 103.0549 121.4501
sample estimates:
mean of x
 112.2525
```

p-value меньше .05, поэтому мы можем отклонить нулевую гипотезу о том, что среднее для веса в генеральной совокупности, из которой вы взяли выборку супергероев, равно 100. Мы принимаем ненулевую гипотезу о том, что в генеральной совокупности средний вес *не равен* 100.

- Проведите одновыборочный t-тест для роста супергероев и числа 185 — предположительного среднего роста в генеральной совокупности всех супергероев. Проинтерпретируйте результат.

```
t.test(heroes$Height, mu = 185)
```

```
    One Sample t-test

data:  heroes$Height
t = 0.66246, df = 516, p-value = 0.508
alternative hypothesis: true mean is not equal to 185
```

```
95 percent confidence interval:
 181.6068 191.8458
sample estimates:
mean of x
 186.7263
```

p-value больше .05, поэтому мы *не* можем отклонить нулевую гипотезу о том, что среднее для роста в генеральной совкупности, из которой вы взяли выборку супергероев, равно 185.

27.28 Двухвыборочный зависимый t-test

```
diet <- readr::read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/mas
```

- Посчитайте двухвыборочный зависимый t-тест для остальных диет: для диеты 2 и диеты 3. Проинтерпретируйте полученные результаты.

```
diet2 <- diet %>%
  filter(Diet == 2)
t.test(diet2$pre.weight, diet2$weight6weeks, paired = TRUE)
```

```
    Paired t-test

data:  diet2$pre.weight and diet2$weight6weeks
t = 6.231, df = 26, p-value = 1.36e-06
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 2.027715 4.024137
sample estimates:
mean difference
       3.025926
```

p-value меньше .05, поэтому мы можем отклонить нулевую гипотезу о том, что масса до и после диеты одинаковая. Мы принимаем альтернативную гипотезу о различии средних в генеральной совокупности.

- Сделайте независимый t-тест для сравнения массы испытуемых двух групп после диеты, сравнив первую и третью группу. Проинтерпретируйте результаты.

```
diet3 <- diet %>%
  filter(Diet == 3)
t.test(diet3$pre.weight, diet3$weight6weeks, paired = TRUE)
```

```
    Paired t-test

data:  diet3$pre.weight and diet3$weight6weeks
t = 11.167, df = 26, p-value = 2.03e-11
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 4.200493 6.095803
sample estimates:
mean difference
       5.148148
```

p-value меньше .05, поэтому мы можем отклонить нулевую гипотезу о том, что масса до и после диеты одинаковая. Мы принимаем альтернативную гипотезу о различии средних в генеральной совокупности.

- Проведите независимый t-тест для сравнения массы супергероев с черными и белыми глазами.

```
heroes_white_black <- heroes %>%
  filter(`Eye color` %in% c("white", "black"))
```

```
t.test(heroes_white_black$Weight ~ heroes_white_black$`Eye color`)
```

```
    Welch Two Sample t-test

data:  heroes_white_black$Weight by heroes_white_black$`Eye color`
t = -0.16381, df = 19.842, p-value = 0.8715
alternative hypothesis: true difference in means between group black and group white is not equal
95 percent confidence interval:
 -88.69024  75.78115
sample estimates:
mean in group black mean in group white
           108.0000            114.4545
```

p-value больше .05, поэтому мы не можем отклонить нулевую гипотезу о том, что массы супергероев с черными и белыми глазами одинаковые.

27.29 Двухвыборочный независимый t-test

- Сделайте независимый t-тест для сравнения веса испытуемых двух групп после диеты, сравнив вторую и третью группу. Проинтерпретируйте результаты.

```
diet23 <- diet %>%
  filter(Diet %in% 2:3)
t.test(weight6weeks ~ Diet, data = diet23, paired = FALSE)
```

```
    Welch Two Sample t-test

data:  weight6weeks by Diet
t = -0.15686, df = 49.774, p-value = 0.876
alternative hypothesis: true difference in means between group 2 and group 3 is not equ
95 percent confidence interval:
 -5.471327  4.678734
sample estimates:
mean in group 2 mean in group 3
       68.08519        68.48148
```

p-value больше .05, поэтому мы не можем отклонить нулевую гипотезу о том, что масса до и после диеты одинаковая.

- Сделайте независимый t-тест для сравнения веса испытуемых двух групп после диеты, сравнив первую и третью группу. Проинтерпретируйте результаты.

```
diet13 <- diet %>%
  filter(Diet %in% c(1,3))
t.test(weight6weeks ~ Diet, data = diet13, paired = FALSE)
```

```
    Welch Two Sample t-test

data:  weight6weeks by Diet
t = 0.46818, df = 48.072, p-value = 0.6418
alternative hypothesis: true difference in means between group 1 and group 3 is not equ
95 percent confidence interval:
 -3.602450  5.789487
sample estimates:
mean in group 1 mean in group 3
       69.57500        68.48148
```

p-value больше .05, поэтому мы не можем отклонить нулевую гипотезу о том, что масса до и после диеты одинаковая.

27.30 Непараметрические аналоги t-теста

- Сравните вес первой и второй группы после диеты, используя тест Манна-Уитни. Сравните результаты теста Манна-Уитни с результатами t-теста? Проинтерпретируйте полученные результаты.

```
diet12 <- diet %>%
  filter(Diet %in% c(1,2))
wilcox.test(weight6weeks ~ Diet, data = diet12, paired = FALSE)
```

```
    Wilcoxon rank sum test with continuity correction

data:  weight6weeks by Diet
W = 368, p-value = 0.4117
alternative hypothesis: true location shift is not equal to 0
```

В обоих случаях *p-value* больше 0.05, мы не можем отклонить нулевую гипотезу об отсутствии различий.

- Повторите задание для второй и третьей группы, а так же для первой и третьей группы.

```
wilcox.test(weight6weeks ~ Diet, data = diet23, paired = FALSE)
```

```
    Wilcoxon rank sum test with continuity correction

data:  weight6weeks by Diet
W = 340.5, p-value = 0.6843
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(weight6weeks ~ Diet, data = diet13, paired = FALSE)
```

```
    Wilcoxon rank sum test with continuity correction

data:  weight6weeks by Diet
W = 346, p-value = 0.6849
alternative hypothesis: true location shift is not equal to 0
```

В обоих случаях *p-value* больше 0.05, как и для соответствующих t-тестов. Мы не можем отклонить нулевую гипотезу об отстутствии различий между второй и третьей диетой, между первой и третьей диетой.

- Сравните вес до и после для диеты 1, используя тест Уилкоксона. Сравните с результатами применения t-теста. Проинтерпретируйте полученные результаты.

```
diet1 <- diet %>%
  filter(Diet == 1)
wilcox.test(diet1$pre.weight, diet1$weight6weeks, paired = TRUE)
```

```
    Wilcoxon signed rank test with continuity correction

data:  diet1$pre.weight and diet1$weight6weeks
V = 299, p-value = 2.203e-05
alternative hypothesis: true location shift is not equal to 0
```

И t-тест, и тест Уилкоксона дают *p-value* ниже 0.05. Мы можем отклонить нулевую гипотезу об отсутствии различий.

- Сравните вес до и после для диеты 2 и диеты 3, используя тест Уилкоксона. Сравните с результатами применения t-теста. Проинтерпретируйте полученные результаты.

```
wilcox.test(diet2$pre.weight, diet2$weight6weeks, paired = TRUE)
```

```
    Wilcoxon signed rank test with continuity correction

data:  diet2$pre.weight and diet2$weight6weeks
V = 313, p-value = 5.419e-05
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(diet3$pre.weight, diet3$weight6weeks, paired = TRUE)
```

```
    Wilcoxon signed rank test with continuity correction

data:  diet3$pre.weight and diet3$weight6weeks
V = 378, p-value = 5.912e-06
alternative hypothesis: true location shift is not equal to 0
```

В обоих случаях и t-тест, и тест Уилкоксона дают *p-value* ниже 0.05. Мы можем отклонить нулевую гипотезу об отсутствии различий.

27.31 Критерий хи-квадрат Пирсона

- Связаны ли переменные Alignment и Gender? Используйте критерий хи-квадрат для проверки и проинтерпретируйте результаты

```
pub_good <- heroes %>%
  filter(Alignment %in% c("good", "bad")) %>%
  select(Alignment, Gender) %>%
  drop_na()
```

```
table(pub_good) %>%
  summary()
```

```
Number of cases in table: 677
Number of factors: 2
Test for independence of all factors:
    Chisq = 18.096, df = 1, p-value = 2.1e-05
```

- Создайте в `heroes` новую колонку `is_human` логического типа, в которой будет `TRUE`, если супергерой принадлежит расе (`Race`) `"Human"`, и `FALSE` в случае если супергерой принадлежит другой расе.

```
heroes %>%
  mutate(is_human = Race == "Human")
```

```
# A tibble: 734 x 12
    ...1 name          Gender `Eye color` Race     `Hair color` Height Publisher
   <dbl> <chr>         <chr>  <chr>       <chr>    <chr>         <dbl> <chr>
 1     0 A-Bomb        Male   yellow      Human    No Hair         203 Marvel C~
 2     1 Abe Sapien    Male   blue        Icthyo ~ No Hair         191 Dark Hor~
 3     2 Abin Sur      Male   blue        Ungaran  No Hair         185 DC Comics
 4     3 Abomination   Male   green       Human /~ No Hair         203 Marvel C~
 5     4 Abraxas       Male   blue        Cosmic ~ Black            NA Marvel C~
 6     5 Absorbing Man Male   blue        Human    No Hair         193 Marvel C~
 7     6 Adam Monroe   Male   blue        <NA>     Blond            NA NBC - He~
 8     7 Adam Strange  Male   blue        Human    Blond           185 DC Comics
 9     8 Agent 13      Female blue        <NA>     Blond           173 Marvel C~
10     9 Agent Bob     Male   brown       Human    Brown           178 Marvel C~
# i 724 more rows
```

```
# i 4 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>,
#   is_human <lgl>
```

- Посчитайте долю женщин для "Human" и всех остальных (is_human равен TRUE и FALSE соответственно). Перед этим удалите все строчки с NA в переменных is_human и Gender.

```
heroes %>%
  mutate(is_human = Race == "Human") %>%
  drop_na(is_human, Gender) %>%
  group_by(is_human) %>%
  summarise(mean(Gender == "Female"))
```

```
# A tibble: 2 x 2
  is_human `mean(Gender == "Female")`
  <lgl>                           <dbl>
1 FALSE                           0.241
2 TRUE                            0.242
```

- Сравните распределения частот для переменных is_human и Gender используя хи-квадрат Пирсона. Проинтерпретируйте результаты.

```
heroes %>%
  mutate(is_human = Race == "Human") %>%
  drop_na(is_human, Gender) %>%
  select(is_human, Gender) %>%
  table() %>%
  chisq.test(correct = FALSE)
```

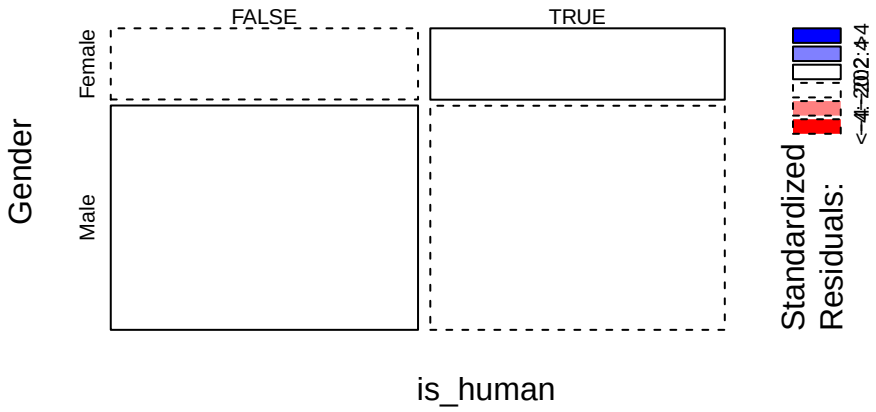
```
    Pearson's Chi-squared test

data:  .
X-squared = 0.00037447, df = 1, p-value = 0.9846
```

Мы не можем отвергнуть нулевую гипотезу о независимости расы (Человек/не-человек) и пола.

- Постройте мозаичный график для переменных is_human и Gender.

```
heroes %>%
  mutate(is_human = Race == "Human") %>%
  drop_na(is_human, Gender) %>%
  select(is_human, Gender) %>%
  table() %>%
  mosaicplot(shade = TRUE)
```



27.32 Исследование набора данных Backpack

Для следующих тем нам понадобится набор данных `Backpack` из пакета `Stat2Data`.

```
#install.packages("Stat2Data")
library(Stat2Data)
data(Backpack)
back <- Backpack %>%
  mutate(backpack_kg = 0.45359237 * BackpackWeight,
         body_kg = 0.45359237 * BodyWeight)
```

- Как различается вес рюкзака в зависимости от пола? Кто весит больше?

```
back %>%
  group_by(Sex) %>%
  summarise(mean(backpack_kg))
```

```
# A tibble: 2 x 2
  Sex    `mean(backpack_kg)`
  <fct>                <dbl>
1 Female                5.01
2 Male                  5.63
```

- Если допустить, что выборка репрезентативна, то можно ли сделать вывод о различии по среднему весу рюкзаков в генеральной совокупности?

```
t.test(backpack_kg ~ Sex, data = back)
```

```

    Welch Two Sample t-test

data:  backpack_kg by Sex
t = -1.1782, df = 86.25, p-value = 0.242
alternative hypothesis: true difference in means between group Female and group Male is
95 percent confidence interval:
 -1.6892365  0.4320067
sample estimates:
mean in group Female   mean in group Male
            5.006010             5.634625
```

p-value больше .05, поэтому мы не можем отклонить нулевую гипотезу о том, что масса до и после диеты одинаковая.

- Повторите пункты 2 и 3 для веса самих студентов.

```
back %>%
  group_by(Sex) %>%
  summarise(mean(body_kg))
```

```
# A tibble: 2 x 2
  Sex    `mean(body_kg)`
  <fct>            <dbl>
1 Female            62.3
2 Male              78.1
```

```
t.test(body_kg ~ Sex, data = back)
```

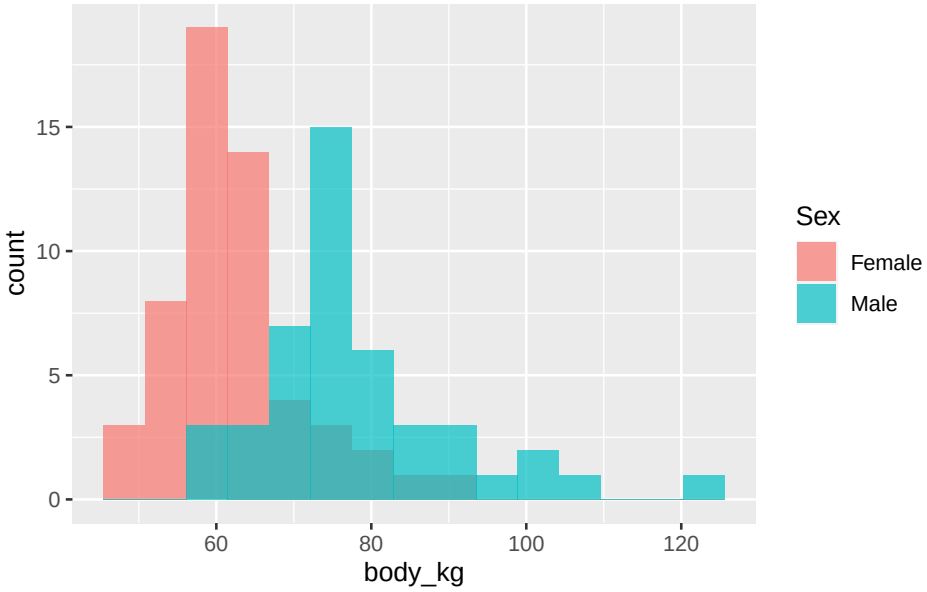
```
    Welch Two Sample t-test

data:  body_kg by Sex
t = -7.0863, df = 77.002, p-value = 5.704e-10
alternative hypothesis: true difference in means between group Female and group Male is not equal
95 percent confidence interval:
 -20.32511 -11.40803
sample estimates:
mean in group Female   mean in group Male
            62.28236             78.14893
```

p-value меньше .05, поэтому мы можем отклонить нулевую гипотезу о том, что масса студентов-мужчин и студентов-женщин одинаковая. Мы принимаем альтернативную гипотезу о различии средних в генеральной совокупности.

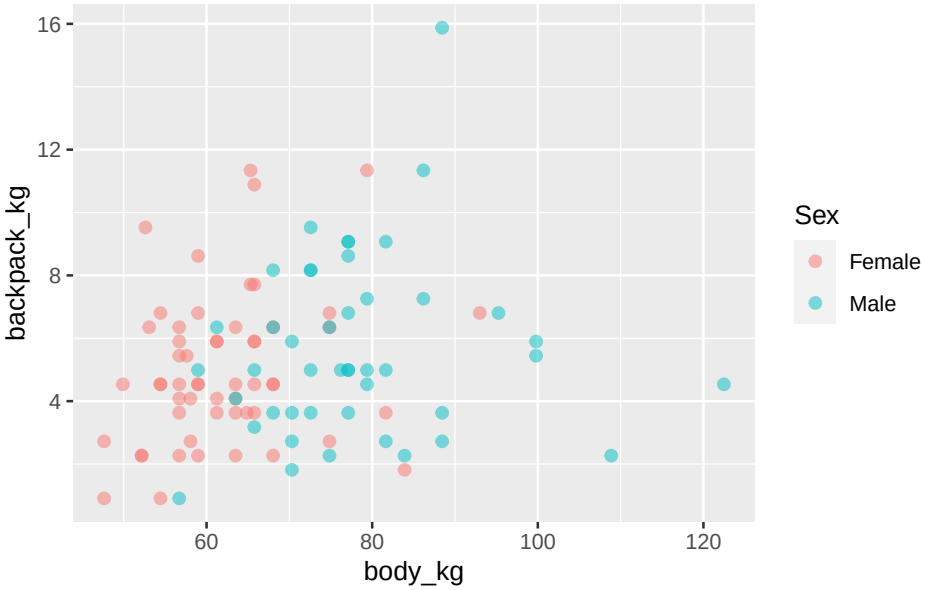
- Визуализируйте распределение этих двух переменных в зависимости от пола (используя `ggplot2`)

```
library(ggplot2)
ggplot(back)+
  geom_histogram(aes(x = body_kg, fill = Sex), bins = 15, position = "identity", alpha = 0.7)
```



- Постройте диаграмму рассеяния с помощью `ggplot2`. Цветом закодируйте пол респондента.

```
ggplot(back, aes(x = body_kg, y = backpack_kg))+
  geom_point(aes(colour = Sex), alpha = 0.5, size = 2)
```



27.33 Ковариация

- Посчитайте матрицу ковариаций для веса студентов и их рюкзаков в *фунтах*. Различаются ли результаты подсчета ковариации этих двух переменных от результатов подсчета ковариаций веса студентов и их рюкзаков в *килограммах*? Почему?

```
back %>%
  select(BodyWeight, BackpackWeight) %>%
  cov()
```

```
                BodyWeight BackpackWeight
BodyWeight       864.20960       32.08788
BackpackWeight    32.08788       33.23677
```

Результаты различаются, потому что значение ковариации зависит от размерности исходных шкал.

27.34 Коэффициент корреляции

- Посчитайте коэффициент корреляции Пирсона для веса студентов и их рюкзаков в *фунтах*. Различаются ли результаты подсчета коэффициента корреляции Пирсона (сам коэффициент, p-value) этих двух переменных от результатов подсчета корреляции Пирсона веса студентов и их рюкзаков в *килограммах*? Почему?

```
cor.test(back$BackpackWeight, back$BodyWeight)
```

```
    Pearson's product-moment correlation

data:  back$BackpackWeight and back$BodyWeight
t = 1.9088, df = 98, p-value = 0.05921
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.007360697  0.371918344
sample estimates:
      cor
0.1893312
```

Результаты не различаются, потому что значение ковариации не зависит от размерности исходных шкал.

- Посчитайте коэффициент корреляции Пирсона для веса и роста супергероев из датасета `heroes`. Проинтерпретируйте результат.

```
cor.test(heroes$Weight, heroes$Height)
```

```
    Pearson's product-moment correlation

data:  heroes$Weight and heroes$Height
t = 4.3555, df = 488, p-value = 1.619e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.1066877 0.2772717
sample estimates:
      cor
0.1934412
```

p-value меньше 0.05, поэтому мы можем отклонить нулевую гипотезу об отсутствии линейной связи между ростом и весом и принять альтернативную гипотезу о том, что такая связь есть. Судя по знаку и размеру корреляции, чем выше рост, тем выше вес супергероя, но эта связь достаточно слабая.

- Теперь посчитайте коэффициент корреляции Спирмена и коэффициент корреляции Кэнделла для веса и роста супергероев из датасета `heroes`. Различаются ли результаты по сравнению с коэффициентом корреляции Пирсона? Почему?

```
cor.test(heroes$Weight, heroes$Height, method = "spearman")
```

```
    Spearman's rank correlation rho

data:  heroes$Weight and heroes$Height
S = 3915061, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.8003344
```

```
cor.test(heroes$Weight, heroes$Height, method = "kendall")
```

```
    Kendall's rank correlation tau

data:  heroes$Weight and heroes$Height
z = 21.548, p-value < 2.2e-16
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.6751664
```

В обоих случаях p-value меньше 0.05, поэтому мы можем отклонить нулевую гипотезу об отсутствии связи между ростом и весом и принять альтернативную гипотезу о том, что такая связь есть. Сильное различие между коэффициентами корреляции указывает на нелинейность этой связи, либо же на наличие значительных выбросов в данных.

27.35 ANOVA

Загрузите пингвиний датасет:

```
library(tidyverse)
penguins <- readr::read_csv("https://raw.githubusercontent.com/Pozdniakov/tidy_stats/master/da
  mutate(id = row_number()) #      id
```

Давайте посчитаем арифметические средние по группам для различных признаков:

```
penguins %>%
  group_by(species) %>%
  summarise(across(ends_with("_mm") | ends_with("_g"),
                   mean,
                   na.rm = TRUE))
```

```
# A tibble: 3 x 5
  species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <chr>              <dbl>         <dbl>             <dbl>       <dbl>
1 Adelie              38.8          18.3              190.       3701.
2 Chinstrap           48.8          18.4              196.       3733.
3 Gentoo              47.5          15.0              217.       5076.
```

- Есть ли статистически значимые различия между видами пингвинов по массе тела (body_mass_g)? Если да, то между какими?

```
aov(body_mass_g ~ species, data = penguins) %>%
  summary()
```

```
             Df    Sum Sq  Mean Sq F value Pr(>F)
species       2 146864214 73432107   343.6 <2e-16 ***
Residuals   339  72443483   213698
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
2 observations deleted due to missingness
```

```
aov(body_mass_g ~ species, data = penguins) %>%
  TukeyHSD()
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = body_mass_g ~ species, data = penguins)

$species
                      diff       lwr       upr     p adj
Chinstrap-Adelie   32.42598 -126.5002  191.3522 0.8806666
Gentoo-Adelie    1375.35401 1243.1786 1507.5294 0.0000000
Gentoo-Chinstrap 1342.92802 1178.4810 1507.3750 0.0000000
```

- Есть ли статистически значимые различия между видами пингвинов по длине ласт (`flipper_length_mm`)? Если да, то между какими?

```
aov(flipper_length_mm ~ species, data = penguins) %>%
  summary()
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
species       2  52473   26237   594.8 <2e-16 ***
Residuals   339  14953      44
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
2 observations deleted due to missingness
```

```
aov(flipper_length_mm ~ species, data = penguins) %>%
  TukeyHSD()
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = flipper_length_mm ~ species, data = penguins)

$species
                     diff       lwr       upr p adj
Chinstrap-Adelie  5.869887  3.586583  8.153191     0
Gentoo-Adelie    27.233349 25.334376 29.132323     0
Gentoo-Chinstrap 21.363462 19.000841 23.726084     0
```

- Есть ли взаимодействие между полом (`sex`) и видом (`species`) для длины ласт `flipper_length_mm`? Если да, то между какими группами?

Решение с помощью `aov()`:

```
aov(flipper_length_mm ~ species * sex, data = penguins) %>%
  TukeyHSD()
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = flipper_length_mm ~ species * sex, data = penguins)

$species
                     diff      lwr       upr p adj
Chinstrap-Adelie  5.72079  3.76593  7.675649     0
Gentoo-Adelie    27.13255 25.48814 28.776974     0
Gentoo-Chinstrap 21.41176 19.38766 23.435867     0

$sex
                diff      lwr      upr p adj
male-female 6.849169 5.629796 8.068542     0

$`species:sex`
                                       diff        lwr        upr     p adj
Chinstrap:female-Adelie:female    3.9407736   0.574682   7.306865 0.0113141
Gentoo:female-Adelie:female      24.9123760  22.060733  27.764019 0.0000000
Adelie:male-Adelie:female         4.6164384   1.933019   7.299857 0.0000191
Chinstrap:male-Adelie:female     12.1172442   8.751153  15.483336 0.0000000
Gentoo:male-Adelie:female        33.7464631  30.934167  36.558759 0.0000000
Gentoo:female-Chinstrap:female   20.9716024  17.469931  24.473274 0.0000000
Adelie:male-Chinstrap:female      0.6756648  -2.690427   4.041756 0.9925701
Chinstrap:male-Chinstrap:female   8.1764706   4.244498  12.108443 0.0000001
Gentoo:male-Chinstrap:female     29.8056895  26.335986  33.275393 0.0000000
Adelie:male-Gentoo:female       -20.2959376 -23.147581 -17.444295 0.0000000
Chinstrap:male-Gentoo:female    -12.7951318 -16.296803  -9.293460 0.0000000
Gentoo:male-Gentoo:female         8.8340871   5.860850  11.807324 0.0000000
Chinstrap:male-Adelie:male        7.5008058   4.134714  10.866897 0.0000000
Gentoo:male-Adelie:male          29.1300247  26.317729  31.942320 0.0000000
Gentoo:male-Chinstrap:male       21.6292189  18.159515  25.098922 0.0000000
```

Решение с помощью пакета {ez}

```
library(ez)
ez_species_sex <- ezANOVA(data = penguins %>% drop_na(flipper_length_mm, species, sex),
        dv = .(flipper_length_mm),
        wid = .(id),
        between = .(species, sex),
        return_aov = TRUE)
```

```
TukeyHSD(ez_species_sex$aov)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = formula(aov_formula), data = data)

$species
                     diff      lwr       upr p adj
Chinstrap-Adelie  5.72079  3.76593  7.675649     0
Gentoo-Adelie    27.13255 25.48814 28.776974     0
Gentoo-Chinstrap 21.41176 19.38766 23.435867     0

$sex
                diff      lwr      upr p adj
male-female 6.849169 5.629796 8.068542     0

$`species:sex`
                                        diff        lwr        upr     p adj
Chinstrap:female-Adelie:female     3.9407736   0.574682   7.306865 0.0113141
Gentoo:female-Adelie:female       24.9123760  22.060733  27.764019 0.0000000
Adelie:male-Adelie:female          4.6164384   1.933019   7.299857 0.0000191
Chinstrap:male-Adelie:female      12.1172442   8.751153  15.483336 0.0000000
Gentoo:male-Adelie:female         33.7464631  30.934167  36.558759 0.0000000
Gentoo:female-Chinstrap:female    20.9716024  17.469931  24.473274 0.0000000
Adelie:male-Chinstrap:female       0.6756648  -2.690427   4.041756 0.9925701
Chinstrap:male-Chinstrap:female    8.1764706   4.244498  12.108443 0.0000001
Gentoo:male-Chinstrap:female      29.8056895  26.335986  33.275393 0.0000000
Adelie:male-Gentoo:female        -20.2959376 -23.147581 -17.444295 0.0000000
Chinstrap:male-Gentoo:female     -12.7951318 -16.296803  -9.293460 0.0000000
Gentoo:male-Gentoo:female          8.8340871   5.860850  11.807324 0.0000000
Chinstrap:male-Adelie:male         7.5008058   4.134714  10.866897 0.0000000
Gentoo:male-Adelie:male           29.1300247  26.317729  31.942320 0.0000000
Gentoo:male-Chinstrap:male        21.6292189  18.159515  25.098922 0.0000000
```

Глава 28

Библиография

Adler, Joseph. 2010. *R in a nutshell: A desktop quick reference*. " O'Reilly Media, Inc.".

Ritchie, Stuart, и Elliot Tucker-Drob. 2018. «How Much Does Education Improve Intelligence? A Meta-Analysis». *Psychological Science* 29 (июнь): 095679761877425. https://doi.org/10.1177/0956797618774253.

Wickham, Hadley. 2010. «A layered grammar of graphics». *Journal of Computational and Graphical Statistics* 19 (1): 3–28. https://doi.org/10.1198/jcgs.2009.07098.

Wilkinson, Leland. 2005. *The Grammar of Graphics (Statistics and Computing)*. Berlin, Heidelberg: Springer-Verlag.