

Наука о данных в R для программы
Цифровых гуманитарных
исследований

Г. А. Мороз, И. С. Поздняков

Оглавление

| | | |
|----------|--|------------|
| 1 | О курсе | 5 |
| 2 | Введение в R | 7 |
| 2.1 | Наука о данных | 7 |
| 2.2 | Установка R и RStudio | 8 |
| 2.3 | Полезные ссылки | 8 |
| 2.4 | Rstudio | 9 |
| 2.5 | Введение в R | 10 |
| 2.6 | Типы данных | 18 |
| 2.7 | Вектор | 21 |
| 2.8 | Матрицы (matrix) | 33 |
| 2.9 | Списки (list) | 35 |
| 2.10 | Data.frame | 37 |
| 2.11 | Начинаем работу с реальными данными | 40 |
| 2.12 | Препроцессинг данных в R | 43 |
| 3 | tidyverse: Загрузка и трансформация данных | 53 |
| 3.1 | Загрузка данных | 54 |
| 3.2 | tibble | 58 |
| 3.3 | dplyr | 60 |
| 3.4 | tidyr package | 75 |
| 4 | Визуализация данных | 79 |
| 4.1 | Зачем визуализировать данные? | 79 |
| 4.2 | Основы ggplot2 | 82 |
| 4.3 | Столбчатые диаграммы (barplots) | 93 |
| 4.4 | Факторы | 97 |
| 4.5 | Точки, джиттер (jitter), ящики с усами (boxplot), (violinplot) | 100 |
| 4.6 | Гистограммы | 102 |
| 4.7 | Функции плотности | 102 |
| 4.8 | Фасетизация | 102 |
| 4.9 | Визуализация комбинаций признаков | 102 |
| 5 | Условия и работа со списками | 107 |

| | | |
|-----------|---|------------|
| 6 | Представление данных: rmarkdown, github, shiny | 109 |
| 7 | Работа со строками | 111 |
| 8 | Работа с текстами: tidytext, udpipe | 113 |
| 9 | Сбор данных из интернета: rvest | 115 |
| 10 | Нестандартные данные: время, OCR, карты | 117 |
| 11 | Задания | 119 |
| 11.1 | Вектор | 119 |
| 11.2 | Вектор. Операции с векторами | 120 |
| 11.3 | Вектор. Индексирование | 121 |
| 11.4 | Списки | 121 |
| 11.5 | Матрицы | 123 |
| 11.6 | Датафрейм | 125 |
| 12 | Решения заданий | 127 |
| 12.1 | Вектор | 127 |
| 12.2 | Вектор. Операции с векторами | 129 |
| 12.3 | Вектор. Индексирование | 130 |
| 12.4 | Списки | 132 |
| 12.5 | Матрицы | 133 |
| 12.6 | Датафрейм | 136 |

Глава 1

О курсе

Материалы для курса Наука о данных для магистерской программы Цифровых гуманитарных исследований НИУ ВШЭ.

Глава 2

Введение в R

2.1 Наука о данных

Наука о данных — это новая область знаний, которая активно развивается в последнее время. Она находится на пересечении компьютерных наук, статистики и математики и трудно сказать, действительно ли это наука. При этом это движение развивается в самых разных научных направлениях, иногда даже оформляясь в отдельную отрасль:

- биоинформатика
- цифровые гуманитарные исследования
- датаждурналистика
- ...

Все больше книг “Data Scince for ...”:

- psychologists (Hansjörg, 2019)
- immunologists (Thomas and Pallett, 2019)
- buisness (Provost and Fawcett, 2013)
- public policy (Brooks and Cooper, 2013)
- fraud detection (Baesens et al., 2015)
- ...

Среди умений датасаентистов можно перечислить следующие:

- сбор и обработка данных
- трансформация данных
- визуализация данных
- моделирование данных
- представление полученных результатов

Все эти темы в той или иной мере будут представлены на нашем курсе.

2.2 Установка R и RStudio

В данной книге используется исключительно R (R Core Team, 2019), так что для занятий понадобятся:

- R
 - на Windows¹
 - на Mac²
 - на Linux³, также можно добавить зеркало и установить из командной строки:

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

- RStudio — IDE для R (можно скачать здесь⁴)
- и некоторые пакеты на R

Часто можно увидеть или услышать, что R — язык программирования для “статистической обработки данных”. Изначально это, конечно, было правдой, но уже давно R — это полноценный язык программирования, который при помощи своих пакетов позволяет решать огромный спектр задач. В данной книге используются следующая версия R:

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

```
## [1] "R version 3.6.1 (2019-07-05)"
```

Некоторые люди не любят устанавливать лишние программы себе на компьютер, несколько вариантов есть и для них:

- RStudio cloud⁵ — полная функциональность RStudio, пока бесплатная, но скоро это исправят;
- RStudio on rollApp⁶ — облачная среда, позволяющая разворачивать программы.

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 среда (IDE), которая позволяет в этом языке очень удобно работать.

2.3 Полезные ссылки

В интернете легко найти документацию и туториалы по самым разным вопросам в R, так что главный залог успеха — грамматно пользоваться поисковиком,

¹<https://cran.r-project.org/bin/windows/base/>

²<https://cran.r-project.org/bin/macosx/>

³<https://cran.rstudio.com/bin/linux/>

⁴<https://www.rstudio.com/products/rstudio/download/>

⁵<https://rstudio.cloud/>

⁶<https://www.rollapp.com/app/rstudio>

и лучше на английском языке.

- книга (Wickham and Grolemund, 2016)⁷ является достаточно сильной альтернативой всему курсу
- stackoverflow⁸ — сервис, где достаточно быстро отвечают на любые вопросы (не обязательно по R)
- RStudio community⁹ — быстро отвечают на вопросы, связанные с R
- русский stackoverflow¹⁰
- R-bloggers¹¹ — сайт, где собираются новинки, связанные с R
- чат¹², где можно спрашивать про R на русском (но почитайте правила чата¹³, перед тем как спрашивать)
- чат¹⁴ по визуализации данных, чат¹⁵ датажурналистов
- канал про визуализацию¹⁶, data-блог “Новой газеты”¹⁷, ...

2.4 Rstudio

Когда вы откроете RStudio первый раз, вы увидите три панели: консоль, окружение и историю, а также панель для всего остального. Если ткнуть в консоли на значок уменьшения, то можно открыть дополнительную панель, где можно писать скрипт.

⁷<https://r4ds.had.co.nz>

⁸<https://stackoverflow.com>

⁹<https://community.rstudio.com/>

¹⁰<https://ru.stackoverflow.com>

¹¹[https://www.r-bloggers.com/](https://www.r-bloggers.com)

¹²https://t.me/rlang_ru

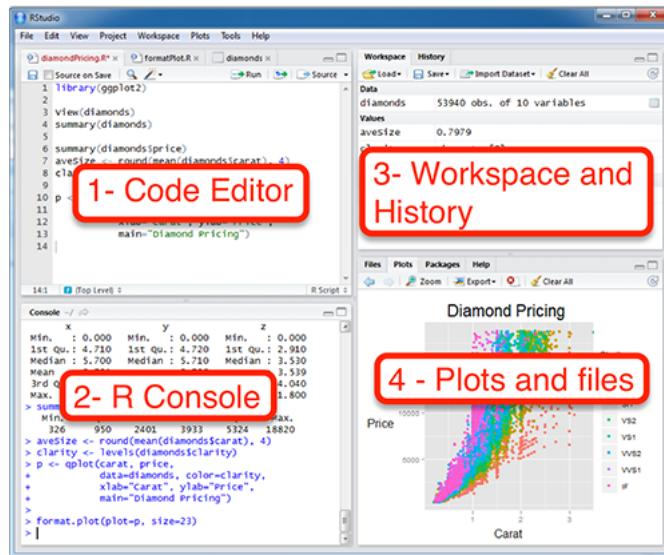
¹³<https://github.com/r-lang-group-ru/group-rules/blob/master/README.md>

¹⁴<https://t.me/joinchat/CxZg5goGc6rlWGjcv0YrpA>

¹⁵<https://t.me/ddjrus>

¹⁶<https://t.me/chartomojka>

¹⁷https://t.me/novaya_data



Существуют разные типы пользователей: одни любят работать в консоли (на картинке это 2 — R Console), другие предпочитают скрипты (1 — Code Editor). Консоль позволяет иметь интерактивный режим команда-ответ, а скрипт является по сути текстовым документом, фрагменты которого можно для отладки запускать в консоли.

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 последних 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.

2.5 Введение в R

2.5.1 R как калькулятор

Ой-ей, консоль, скрипт че-то все непонятно.

Давайте начнем с самого простого и попробуем использовать R как простой калькулятор. $+$, $-$, $*$, $/$, $^$ (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

```
## [1] 8
```

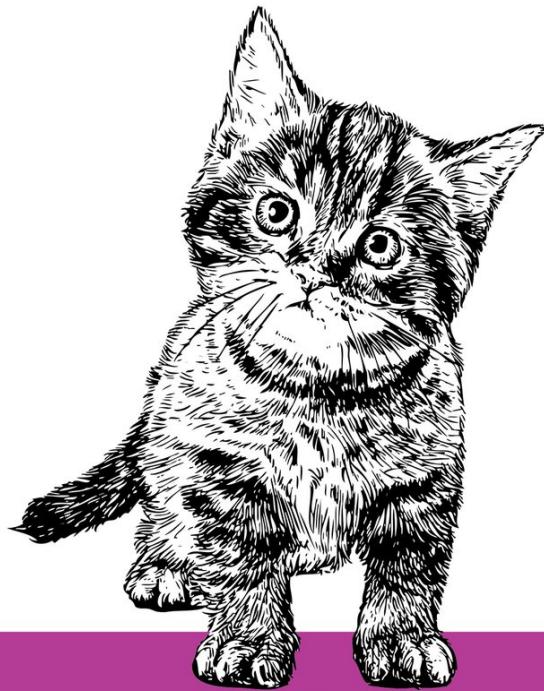
(2+2)*2

```
## [1] 8
```

Ничего сложного, верно? Вводим выражение и получаем результат. Порядок выполнения арифметических операций как в математике, так что не забывайте про скобочки.

Если Вы не уверены в том, какие операции имеют приоритет, то используйте скобочки, чтобы точно обозначить, в каком порядке нужно производить операции.

How to actually learn any new programming concept



Essential

Changing Stuff and
Seeing What Happens

O RLY?

@ThePracticalDev

2.5.2 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
```

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:

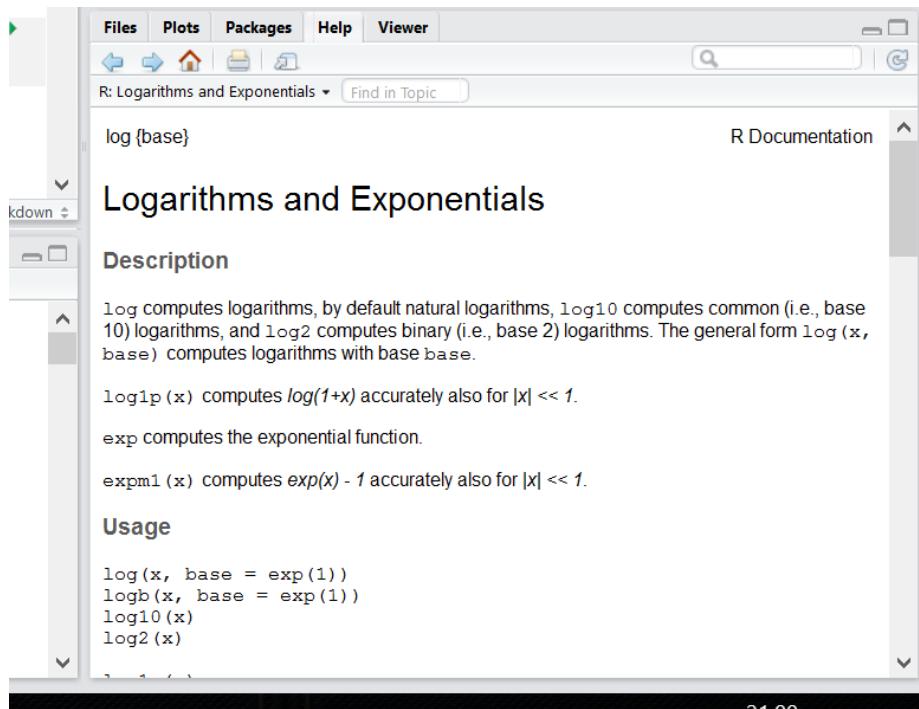
$2^3 = 8$

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:



Действительно, у этой функции есть еще аргумент `base` = . По дефолту он равен числу Эйлера ($2.7182818\dots$), т.е. функция считает натуральный логарифм. В большинстве функций R есть какой-то основной инпут — данные в том или ином формате, а есть и дополнительные параметры, которые можно прописывать вручную, если параметры по умолчанию нас не устраивают.

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

```
## [1] 3
```

...или просто (если Вы уверены в порядке аргументов):

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

```
## [1] 3
```

Более того, Вы можете использовать output одних функций как инпут 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 недописывать имена аргументов, 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 input для функций и многое-многое другое. В R очень крутые возможности работы с функциями. Поэтому подружитесь с функциями, они клевые.

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

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

```
## [1] 7
```

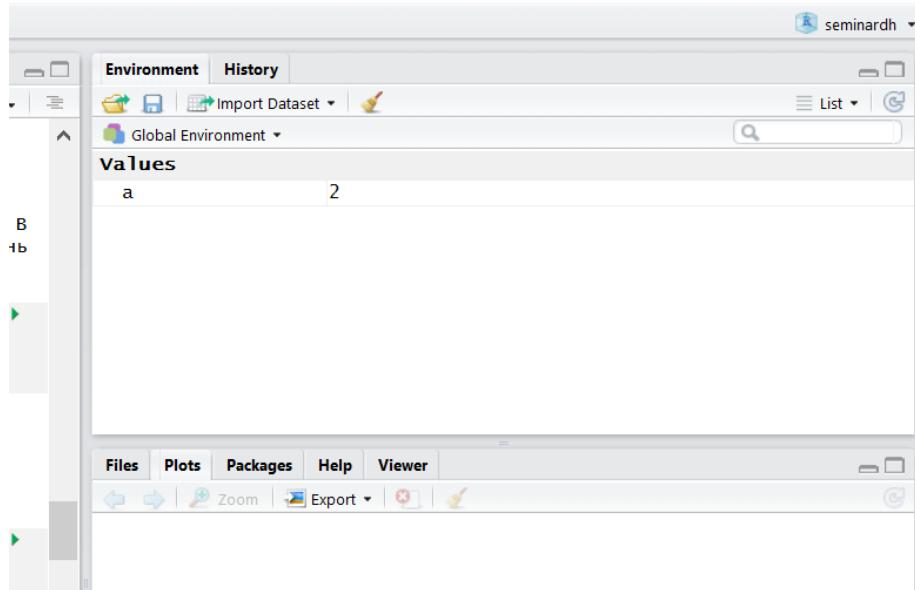
2.5.3 Переменные

Важная штука в программировании на практически любом языке — возможность сохранять значения в **переменных**. 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 - на Mac).

```
a <- 2
a
```

```
## [1] 2
```

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



Можно использовать переменные в функциях и просто вычислениях:

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

```
## [1] 8
```

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

```
## [1] 3
```

Вы можете сравнивать разные переменные:

```
a == b
```

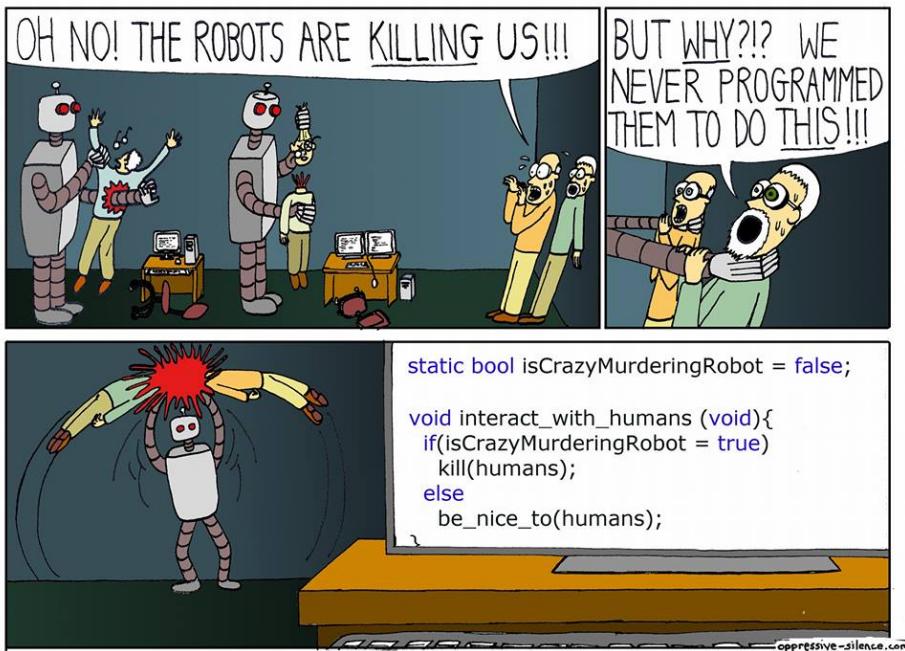
```
## [1] FALSE
```

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

```
a = b
a
```

```
## [1] 8
```

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 *неравенство*:

```

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
```

2.6 Типы данных

До этого момента мы работали только с числами (numeric):

```
class(a)
## [1] "numeric"
```

Вообще, в R много типов numeric: integer (целые), double (с десятичной дробью), complex (комплексные числа). Последние пишутся так: complexnumber <- 2+2i Однако в R с этим обычно можно вообще не заморачиваться, R сам будет конвертить между форматами при необходимости. Немного подробностей здесь:

Разница между numeric и integer¹⁸, Как работать с комплексными числами в R¹⁹

Теперь же нам нужно ознакомиться с двумя другими типами данных в R:

1. **character**: строки символов. Они должны выделяться кавычками. Можно использовать как ", так и ' (что удобно, когда строчка внутри уже содержит какие-то кавычки).

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

¹⁸<https://stackoverflow.com/questions/23660094/whats-the-difference-between-integer-class-and-numeric-cl>

¹⁹<http://www.r-tutor.com/r-introduction/basic-data-types/complex>

```
## [1] " "
class(s)
## [1] "character"
```

2. logical: просто TRUE или FALSE.

```
t1 <- TRUE
f1 <- FALSE

t1
```

```
## [1] TRUE

f1
## [1] FALSE
```

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

```
t2 <- T
f2 <- F
```

Это дурная практика, так как 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 вы можете догадаться, что результаты сравнения, например, числовых или строковых переменных вы можете сохранять в переменные тоже!

```
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, а это 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 не (!):

```
t1
```

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

```
!t1
```

```
## [1] FALSE
```

```
!!t1 # !
```

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

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

```
t1&t2
```

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

```
t1&f1
```

```
## [1] FALSE
```

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

```
t1 | f1
```

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

```
f1 | f2
```

```
## [1] FALSE
```

Если кому-то вдруг понадобиться другое ИЛИ — есть функция `xor()`, принимающий два аргумента.

Поздравляю, мы только что разобрались с самой занудной частью. Пора переходить к важному и интересному. ВЕКТОРАМ!

2.7 Вектор

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

P.S. 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, {2,1}. Вот последние два числа и будем считать вектором. Поэтому постарайтесь на время выбросить стрелочки из головы.

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 “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(" ", " ", " ")
```

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

Одна из самых мерзких и раздражающих причин ошибок в коде — это использование из кириллицы вместо с 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 шаг функция `seq()` посчитает сама:

```
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!

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

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

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

```
## [1] 1 2 2 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!"
```

2.7.1 Coercion

Что будет, если вы объедините два вектора с значениями разных типов? Ошибка? Мы уже обсуждали, что в *atomic* 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.

Например:

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

```
## [1] 0 2
```

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

```
2 + TRUE
```

```
## [1] 3
```

Это называется *coercion*. Более сложный пример:

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

```
## [1] "TRUE"   "3"      ""       "
```

У R есть иерархия коэрсинга: NULL < raw < logical < integer < double < complex < character < list < expression. Мы из этого списка еще многое не знаем, сейчас важно запомнить, что логические данные — TRUE и FALSE — превращаются в 0 и 1 соответственно, а 0 и 1 в строчки "0" и "1".

Если Вы боитесь полагаться на coercion, то можете воспользоваться функциями `as.`

```
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).

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

## Warning: NAs introduced by coercion

## [1] 1 2 NA
```

2.7.2 Операции с векторами

Все те арифметические операции, 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 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, Вы хотите извлечь корень из нескольких чисел, для этого не нужны никакие циклы (как это обычно делается в 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
## [8] 2.828427 3.000000 3.162278
```

2.7.3 Recycling

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 ресайклнга (*recycling* = правило переписывания). Это означает, что если короткий вектор кратен по длине длинному, то он будет повторяться короткий 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 ресайклнга: короткий вектор длиной 1 будет повторяться 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 ресайкинг некратных по длине векторов. См. здесь²⁰. А вот ресайкинг кратных по длине векторов — это очень удобная штука, которая используется очень часто.

2.7.4 Индексирование векторов

Итак, мы подошли к одному из самых сложных моментов. И одному из основных. От того, как хорошо вы научились 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 <- 1:10
n[1]
```

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

```
n[10]
```

```
## [1] 10
```

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 привыкли

²⁰<https://stackoverflow.com/questions/6555651/under-what-circumstances-does-r-recycle>

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] 1 2 20 4 5 6 7 8 9 10
```

Конечно, можно использовать целые векторы для индексирования:

```
n[4:7]

## [1] 4 5 6 7

n[10:1]

## [1] 10 9 8 7 6 5 4 20 2 1
```

Индексирование с минусом выдаст вам все значения вектора кроме выбранных (простите, пользователя Python, которые ожидают здесь отсчет с конца...):

```
n[-1]

## [1] 2 20 4 5 6 7 8 9 10

n[c(-4, -5)]

## [1] 1 2 20 6 7 8 9 10
```

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] 1 20 5 7 9
```

Ну а если они не равны, то тут будет снова работать правило ресайклинга!

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

```
## [1] 1 20 5 7 9
```

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.

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

```
## a
## 1
```

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

Теперь посчитаем среднее вектора `n`:

```
mean(n)
```

```
## [1] 7.2
```

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  TRUE  TRUE  TRUE
```

А теперь используем его для индексирования вектора `n`:

```
n[larger]
```

```
## [1] 20 8 9 10
```

Можно все это сделать в одну строчку:

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

```
## [1] 20 8 9 10
```

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

2.7.5 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 этого в R есть NA. 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!

Иногда NA в данных очень бесит:

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

```
## [1] 1 2 20 4 NA 6 7 8 9 10
```

```
mean(n)
```

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

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.

Чтобы выбраться из этой непростой ситуации, используйте функцию `is.na()`:

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

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

Результат выполнения `is.na(n)` выдает FALSE в тех местах, где у нас числа и TRUE там, где у нас NA. Нам нужно сделать наоборот. Здесь нам понадобится оператор `!` (мы его уже встречали), который инвертирует логические значения:

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

```
## [1] 1 2 20 4 6 7 8 9 10
```

Ура, мы можем считать среднее!

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

```
## [1] 7.444444
```

Теперь Вы понимаете, зачем нужно отрицание (`!`)

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

```
?mean()
```

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

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

```
## [1] 7.444444
```

Eeeee!

NA может появляться в векторах других типов тоже. Кроме NA есть еще NaN — это разные вещи. NaN расшифровывается как Not a Number и получается в результате таких операций как 0/0.

2.7.6 В любой непонятной ситуации — ищите в поисковике

Если вдруг вы не знаете, что искать в хэлпе, или хэлпа попросту недостаточно, то... ищите в поисковике!



Нет ничего постыдного в том, чтобы искать в Интернете решения проблем. Это абсолютно нормально. Используйте силу интернета во благо и да помогут Вам Stackoverflow и бесчисленные R-туториалы!

Computer Programming To Be Officially Renamed “Googling Stack Overflow”
Source: <http://t.co/xu7acfXvFF> pic.twitter.com/iJ9k7aAVhd

— Stack Exchange July 20, 2015

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

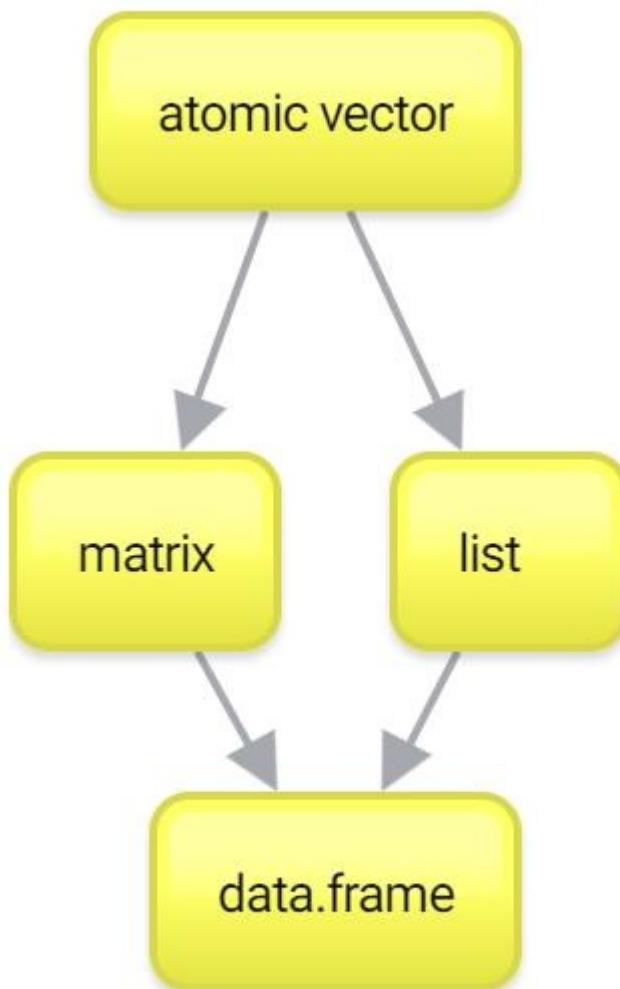
Кроме того, правильно загуглить проблему — не так уж и просто.

Does anyone ever get good at R or do they just get good at googling how to do things in R

— Lauren M. Seyler, Ph.D. https://twitter.com/mousquemere/status/1125522375141883907?ref_src=twsr%5Etfw May 6, 2019

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 — *data.frame*.



2.8 Матрицы (matrix)

Если вдруг Вас пугает это слово, то совершенно зря. Матрица — это всего лишь “двумерный” вектор: вектора, 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
```

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

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

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] 12

A[2:4, 1:3]
##      [,1] [,2] [,3]
## [1,]    2    7   12
## [2,]    3    8   13
## [3,]    4    9   14
```

Первый индекс — выбор строк, второй индекс — выбор колонок. 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[,1:3]
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
A[2:4,]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    7   12   17
## [2,]    3    8   13   18
## [3,]    4    9   14   19
```

```
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
```

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.

То, что матрица - это просто двумерный вектор, не является метафорой: в R матрица - это по сути своей вектор с дополнительными *атрибутами dim и dimnames*. Атрибуты — это неотъемлемые свойства объектов, 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 здесь²¹ или на стр. 99–101 книги “R in a Nutshell” (Adler, 2010).

²¹<https://perso.esiee.fr/~courivad/R/06-objects.html>

2.9 Списки (list)

Теперь представим себе вектор без ограничения на одинаковые данные внутри. И получим список!

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

## [[1]]
## [1] 42
##
## [[2]]
## [1] "      "
##
## [[3]]
## [1] TRUE
```

А это значит, что там могут содержаться самые разные данные, в том числе и другие списки и векторы!

```
lbig <- list(c("Wow", "this", "list", "is", "so", "big"), "16", 1)
lbig
```

```
## [[1]]
## [1] "Wow"  "this" "list" "is"    "so"    "big"
##
## [[2]]
## [1] "16"
##
## [[3]]
## [[3]][[1]]
## [1] 42
##
## [[3]][[2]]
## [1] "      "
##
## [[3]][[3]]
## [1] TRUE
```

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

```
str(lbig)
```

```
## List of 3
```

```
## $ : chr [1:6] "Wow" "this" "list" "is" ...
## $ : chr "16"
## $ :List of 3
## ..$ : num 42
## ..$ : chr "
## ..$ : logi TRUE
```

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

```
namedl <- list(age = 24, PhDstudent = T, language = "Russian")
namedl

## $age
## [1] 24
##
## $PhDstudent
## [1] FALSE
##
## $language
## [1] "Russian"
```

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

```
namedl$age
```

```
## [1] 24
```

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

```
namedl[1]
```

```
## $age
## [1] 24
```

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

```
class(namedl)
```

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

```
class(namedl[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:

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

```
## [1] 24
```

```
class(namedl[[1]])
```

```
## [1] "numeric"
```

Indexing lists in #rstats. Inspired by the Residence Inn pic.twitter.com/YQ6axb2w7t

— Hadley Wickham (@ href="https://twitter.com/hadleywickham/status/643381054758363136?ref_src=twsrc%5Etfw">September 14, 2015

Как и в случае с вектором, к элементу списка можно обращаться по имени.

```
namedl[['age']]
```

```
## [1] 24
```

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

Списки довольно часто используются в R, но реже, чем в Python. Со многими объектами в R, такими как результаты статистических тестов, объекты ggplot и т.д. 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 dataфрейм.

2.10 Data.frame

Итак, мы перешли к самому главному. Самому-самому. Датафреймы (**data.frames**). 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 dataфрейм.

```

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

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

str(df)

## 'data.frame': 5 obs. of 3 variables:
## $ name : Factor w/ 5 levels "Eugeny","Ivan",...: 2 1 3 4 5
## $ age  : num 26 34 23 27 26
## $ student: logi FALSE FALSE TRUE TRUE 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, список Вы представляли “горизонтально”. Если это так, то теперь “переверните” его у себя в голове. Так, чтоб названия векторов оказались сверху, а колонки стали столбцами. Поскольку длина всех этих векторов равна (обязательное условие!), то данные представляют собой табличку, похожую на матрицу. Но в отличие от матрицы, разные столбцы могут иметь разные типы данных: 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[2:3]
```

```
## [1] 34 23
```

Здесь мы сначала вытащили колонку age с помощью оператора \$. Результатом этой операции является числовой вектор, из которого 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    Ivan  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
```

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
```

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.

Датафреймы удобно просматривать в RStudio. Для этого нужно написать команду `View(df)` или же просто нажать на название 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... Но, конечно, интереснее все эти вещи делать руками, т.е. с помощью написания кода.

На этом пора заканчивать с введением и приступать к реальным данным.

2.11 Начинаем работу с реальными данными

Итак, пришло время перейти к реальным данным. Мы начнем с использования датасета (так мы будем называть любой набор данных) по Игре Престолов, а точнее, по книгам цикла “Песнь льда и пламени” Дж. Мартина. Да, будут спойлеры, но сериал уже давно закончился и сильно разошелся с книгами...

2.11.1 Рабочая папка и проекты

Для начала скачайте файл по ссылке²²

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:

```
Ошибка в file(file, "rt") :не могу открыть соединение Вдобавок: Предупреждение: В file(file, "rt") :не могу открыть файл 'character-deaths.csv': No such file or directory
```

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:

```
got <- read.csv("character-deaths.csv")
```

- 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:

²²<https://raw.githubusercontent.com/Pozdniakov/stats/master/data/character-deaths.csv>

```
got <- read.csv("character-deaths.csv")
```

Этот вариант использовать не рекомендуется. Как минимум, это сразу делает невозможным запустить скрипт на другом компьютере.

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

```
got <- read.csv("/Users/Username/Some_Folder/character-deaths.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:

```
got <- read.csv("https://raw.githubusercontent.com/Pozdniakov/stats/master/data/character-deaths.csv")
```

- 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 папочка, где у Вас лежат данные, хранятся скрипты, вспомогательные файлы и отчеты. Если нужно вернуться к другому проекту — просто открываете другой проект, с другими файлами и скриптами. 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.

2.11.2 Импорт данных

Как Вы уже поняли, импортирование данных — одна из самых муторных и неприятных вещей в R. Если у Вас получится с этим справиться, то все остальное — ерун-

да. 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 ее аналоге 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 отделет колонки друг от друга.

Для чтения данных из текстового файла есть довольно удобная функция `read.table()`. Почтайте хэлп по ней и ужаснитесь: столько разных параметров на входе! Но там же вы увидете функции `read.csv()`, `read.csv2()` и некоторые другие — по сути, это тот же `read.table()`, но с другими дефолтными параметрами, соответствующие формату файла, который мы загружаем. В данном случае используется формат `.csv`, что означает Comma Separated Values (Значения, Разделенные Запятыми). Это просто текстовый файл, в котором “закодирована” таблица: разные строчки разделяют разные строчки таблицы, а столбцы отделяются запятыми. 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 “другой” формат `.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”.

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

Итак, прочитаем наш файл. Для этого используем только параметр `file =`, который идет первым, и для параметра `stringsAsFactors =` поставим значение `FALSE`:

```
got <- read.csv("data/character-deaths.csv", stringsAsFactors = FALSE)
```

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. Но 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.

Можете проверить с помощью `View(got)`: все работает! Если же вылезает какая-то странная ерунда или же просто ошибка - попробуйте другие функции и покопаться с параметрами. Для этого читайте `Help`.

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.

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 экселецких форматов и стараться все сохранять в .csv.

Для работы с экселецкими файлами есть много пакетов: `readxl`, `xlsx`, `openxlsx`. Для чтения файлов SPSS, Stata, SAS есть пакет `foreign`. Что такое пакеты и как их устанавливать мы изучим позже.

2.12 Препроцессинг данных в R

Вчера мы узнали про основы языка R, про то, как работать с векторами, списками, матрицами и, наконец, датафреймами. Мы закончили день на загрузке данных, с чего мы и начнем сегодня:

```
got <- read.csv("data/character-deaths.csv", stringsAsFactors = F)
```

После загрузки данных стоит немного "осмотреть" получившийся датафрейм `got`.

2.12.1 Исследование данных

Ок, давайте немного поизучаем датасет. Обычно мы привыкли глазами пробегать по данным, листая строки и столбцы — и это вполне правильно и логично, от этого не нужно отучаться. Но мы можем дополнить наш базовый зрительно-поисковой инструментарий несколькими полезными командами.

Во-первых, вспомним другую полезную функцию `str()`:

```
str(got)
```

```
## 'data.frame': 917 obs. of 13 variables:
## $ Name : chr "Addam Marbrand" "Aegon Frey (Jinglebell)" "Aegon Targa...
## $ Allegiances : chr "Lannister" "None" "House Targaryen" "House Greyjoy" ...
## $ Death.Year : int NA 299 NA 300 NA NA 300 300 NA NA ...
## $ Book.of.Death : int NA 3 NA 5 NA NA 4 5 NA NA ...
## $ Death.Chapter : int NA 51 NA 20 NA NA 35 NA NA NA ...
## $ Book.Intro.Chapter: int 56 49 5 20 NA NA 21 59 11 0 ...
## $ Gender : int 1 1 1 1 1 1 0 1 1 ...
## $ Nobility : int 1 1 1 1 1 1 1 1 0 ...
## $ GoT : int 1 0 0 0 0 0 1 1 0 0 ...
## $ CoK : int 1 0 0 0 0 1 0 1 1 0 ...
## $ SoS : int 1 1 0 0 1 1 1 1 0 1 ...
## $ FfC : int 1 0 0 0 0 0 1 0 1 0 ...
## $ DWD : int 0 0 1 1 0 0 0 1 0 0 ...
```

Давайте разберемся с переменными в датафрейме:

Колонка `Name` — здесь все понятно. Важно, что эти имена записаны абсолютно по-разному: где-то с фамилией, где-то без, где-то в скобочках есть пояснения. Колонка `Allegiances` — к какому дому принадлежит персонаж. С этим сложно, иногда они меняют дома, здесь путаются сами семьи и персонажи, лояльные им. Особой разницы между `Stark` и `House Stark` нет. Следующие колонки `-Death Year`, `Book.of.Death`, `Death.Chapter`, `Book.Intro.Chapter` — означают номер главы, в которой персонаж впервые появляется, а так же номер книги, глава и год (от завоевания Вестероса Эйгоном Таргариеном), в которой персонаж умирает. `Gender` — 1 для мужчин, 0 для женщин. `Nobility` — дворянское происхождение персонажа. Последние 5 столбцов содержат информацию, появлялся ли персонаж в книге (всего книг пока что 5).

Другая полезная функция для больших таблиц — функция `head()`: она выведет первые несколько (по дефолту 6) строчек датафрейма.

```
head(got)
```

| | Name | Allegiances | Death.Year | Book.of.Death | | | | | |
|------|-------------------------|--------------------|------------|---------------|-----|-----|-----|-----|-----|
| ## 1 | Addam Marbrand | Lannister | NA | NA | | | | | |
| ## 2 | Aegon Frey (Jinglebell) | None | 299 | 3 | | | | | |
| ## 3 | Aegon Targaryen | House Targaryen | NA | NA | | | | | |
| ## 4 | Adrack Humble | House Greyjoy | 300 | 5 | | | | | |
| ## 5 | Aemon Costayne | Lannister | NA | NA | | | | | |
| ## 6 | Aemon Estermont | Baratheon | NA | NA | | | | | |
| ## 7 | Death.Chapter | Book.Intro.Chapter | Gender | Nobility | GoT | CoK | SoS | FfC | DWD |
| ## 1 | NA | 56 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

```
## 2      51      49      1      1      0      0      1      0      0
## 3     NA       5      1      1      0      0      0      0      1
## 4     20      20      1      1      0      0      0      0      1
## 5     NA       1      1      0      0      0      1      0      0
## 6     NA       1      1      0      1      1      0      0      0
```

Есть еще функция `tail()`. Догадайтесь сами, что она делает.

Для некоторых переменных полезно посмотреть таблицы частотности с помощью функции `table()`:

```
table(got$Allegiances)
```

```
##          Arryn Baratheon Greyjoy House Arryn
##             23      56      51           7
## House Baratheon House Greyjoy House Lannister House Martell
##             8       24      21           12
## House Stark House Targaryen House Tully House Tyrell
##            35      19       8           11
## Lannister      Martell Night's Watch None
##            81      25      116          253
## Stark         Targaryen      Tully Tyrell
##            73      17      22           15
## Wildling
##            40
```

Yay! Очень просто и удобно, не так ли? Функция `table()` может принимать сразу несколько столбцов. Это удобно для получения *таблицы сопряженности*:

```
table(got$Allegiances, got$Gender)
```

```
##          0   1
## Arryn    3  20
## Baratheon 6  50
## Greyjoy   4  47
## House Arryn 3   4
## House Baratheon 0   8
## House Greyjoy 1  23
## House Lannister 2  19
## House Martell 7   5
## House Stark 6  29
## House Targaryen 5  14
## House Tully 0   8
## House Tyrell 4   7
```

```
##   Lannister      12   69
##   Martell         7   18
##   Night's Watch    0 116
##   None            51 202
##   Stark           21   52
##   Targaryen        1   16
##   Tully            2   20
##   Tyrell           6    9
##   Wildling         16   24
```

2.12.2 Subsetting

Как мы обсуждали на прошлом занятии, мы можем сабсэттить (выделять часть датафрейма) датафрейм, обращаясь к нему и как к матрице: *датафрейм[вектор_с_номерами_строк, вектор_с_номерами_колонок]*

```
got[100:115, 1:2]
```

```
##                  Name Allegiances
## 100      Blue Bard House Tyrell
## 101     Bonifer Hasty Lannister
## 102          Borcas Night's Watch
## 103    Boremund Harlaw Greyjoy
## 104      Boros Blount Baratheon
## 105          Borroq Wildling
## 106      Bowen Marsh Night's Watch
## 107      Bran Stark House Stark
## 108    Brandon Norrey Stark
## 109          Brenett None
## 110  Brienne of Tarth Stark
## 111          Bronn Lannister
## 112      Brown Bernarr Night's Watch
## 113          Brusco None
## 114    Bryan Fossoway Baratheon
## 115      Bryce Caron Baratheon
```

и используя имена колонок:

```
got[508:515, "Name"]
```

```
## [1] "Mance Rayder"    "Mandon Moore"    "Maric Seaworth"  "Marei"
## [5] "Margaery Tyrell" "Marillion"       "Maris"           "Marissa Frey"
```

и даже используя вектора названий колонок!

```
got[508:515, c("Name", "Allegiances", "Gender")]
```

| | Name | Allegiances | Gender |
|--------|-----------------|-----------------|--------|
| ## 508 | Mance Rayder | Wildling | 1 |
| ## 509 | Mandon Moore | Baratheon | 1 |
| ## 510 | Maric Seaworth | House Baratheon | 1 |
| ## 511 | Marei | None | 0 |
| ## 512 | Margaery Tyrell | House Tyrell | 0 |
| ## 513 | Marillion | Arryn | 1 |
| ## 514 | Maris | Wildling | 0 |
| ## 515 | Marissa Frey | None | 0 |

Мы можем вытаскивать отдельные колонки как векторы:

```
houses <- got$Allegiances
unique(houses) # --- table()
```

| | | | |
|---------|-------------------|---------------|-------------------|
| ## [1] | "Lannister" | "None" | "House Targaryen" |
| ## [4] | "House Greyjoy" | "Baratheon" | "Night's Watch" |
| ## [7] | "Arryn" | "House Stark" | "House Tyrell" |
| ## [10] | "Tyrell" | "Stark" | "Greyjoy" |
| ## [13] | "House Lannister" | "Martell" | "House Martell" |
| ## [16] | "Wildling" | "Targaryen" | "House Arryn" |
| ## [19] | "House Tully" | "Tully" | "House Baratheon" |

Итак, давайте решим нашу первую задачу — вытащим в отдельный датасет всех представителей Ночного Дозора. Для этого нам нужно создать вектор логических значений — результат сравнений колонки Allegiances со значением "Night's Watch" и использовать его как вектор индексов для датафрейма.

```
vectornight <- got$Allegiances == "Night's Watch"
head(vectornight)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Теперь этот вектор с TRUE и FALSE нам надо использовать для индексирования строк. Но что со столбцами? Если мы хотим сохранить все столбцы, то после запятой внутри квадратных скобок нам не нужно ничего указывать:

```
nightswatch <- got[vectornight,]
head(nightswatch)
```

```
##                                     Name Allegiances Death.Year
## 7 Aemon Targaryen (son of Maekar I) Night's Watch      300
```

```

## 10          Aethan Night's Watch      NA
## 13          Alan of Rosby Night's Watch   300
## 16          Albett Night's Watch      NA
## 24          Alliser Thorne Night's Watch   NA
## 49          Arron Night's Watch      NA
##   Book.of.Death Death.Chapter Book.Intro.Chapter Gender Nobility GoT CoK
## 7            4           35          21     1      1  1  0
## 10           NA          NA          0     1      0  0  0
## 13           5           4           18    1      1  0  1
## 16           NA          NA          26    1      0  1  0
## 24           NA          NA          19    1      0  1  1
## 49           NA          NA          75    1      0  0  0
##   SoS FfC DwD
## 7     1   1   0
## 10    1   0   0
## 13    1   0   1
## 16    0   0   0
## 24    1   0   1
## 49    1   0   1

```

Вуаля! Все это можно сделать проще и в одну строку:

```
nightswatch <- got[got$Allegiances == "Night's Watch",]
```

И не забывайте про запятую!

Теперь попробуем вытащить одновременно всех Одичалых (*Wildling*) и всех представителей Ночного Дозора. Это можно сделать, используя оператор | (ИЛИ) при выборе колонок:

```
nightwatch_wildling <- got[got$Allegiances == "Night's Watch" | got$Allegiances == "Wi
```

```

##                               Name Allegiances Death.Year
## 7  Aemon Targaryen (son of Maekar I) Night's Watch   300
## 10                     Aethan Night's Watch      NA
## 13                     Alan of Rosby Night's Watch   300
## 16                     Albett Night's Watch      NA
## 24                     Alliser Thorne Night's Watch   NA
## 49                     Arron Night's Watch      NA
##   Book.of.Death Death.Chapter Book.Intro.Chapter Gender Nobility GoT CoK
## 7            4           35          21     1      1  1  0
## 10           NA          NA          0     1      0  0  0
## 13           5           4           18    1      1  0  1
## 16           NA          NA          26    1      0  1  0

```

```

## 24          NA          NA      19      1      0      1      1
## 49          NA          NA      75      1      0      0      0
##   SoS FfC DwD
## 7   1   1   0
## 10  1   0   0
## 13  1   0   1
## 16  0   0   0
## 24  1   0   1
## 49  1   0   1

```

Кажется очевидным следующий вариант: `got[got$Allegiances == c("Night's Watch", "Wildling"),]`. Однако это выдаст не совсем то, что нужно, хотя результат может показаться верным на первый взгляд. Попробуйте самостоятельно ответить на вопрос, что происходит в данном случае и чем результат отличается от предполагаемого.

Подсказка: вспомните правило recycling.

Для таких случаев есть удобный оператор `%in%`, который позволяет сравнить каждое значение вектора с целым набором значений. Если значение вектора хотя бы один раз встречается в векторе справа от `%in%`, то результат — TRUE:

```
1:6 %in% c(1,4,5)
```

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

```
nightwatch_wildling <- got[got$Allegiances %in% c("Night's Watch", "Wildling"),]
head(nightwatch_wildling)
```

```

##           Name Allegiances Death.Year
## 7 Aemon Targaryen (son of Maekar I) Night's Watch      300
## 10                      Aethan Night's Watch      NA
## 13                      Alan of Rosby Night's Watch      300
## 16                      Albett Night's Watch      NA
## 24                      Alliser Thorne Night's Watch      NA
## 49                      Arron Night's Watch      NA
##   Book.of.Death Death.Chapter Book.Intro.Chapter Gender Nobility GoT CoK
## 7          4          35          21      1      1      1      0
## 10         NA         NA          0      1      0      0      0
## 13         5          4          18      1      1      0      1
## 16         NA         NA          26      1      0      1      0
## 24         NA         NA          19      1      0      1      1
## 49         NA         NA          75      1      0      0      0
##   SoS FfC DwD
## 7   1   1   0
## 10  1   0   0

```

```
## 13   1   0   1
## 16   0   0   0
## 24   1   0   1
## 49   1   0   1
```

2.12.3 Создание новых колонок

Давайте создадим новую колонку, которая будет означать, жив ли еще персонаж (по книгам). Заметьте, что в этом датасете, хоть он и посвящен смертям персонажей, нет нужной колонки. Мы можем попытаться “вытащить” эту информацию. В колонках `Death.Year`, `Death.Chapter` и `Book.of.Death` стоит `NA` у многих персонажей. Например, у `Arya Stark`, которая и по книгам, и по сериалу живее всех живых и мертвых:

```
got[got$name == "Arya Stark",]

##           Name Allegiances Death.Year Book.of.Death Death.Chapter
## 56 Arya Stark      Stark        NA          NA          NA
##             Book.Intro.Chapter Gender Nobility GoT CoK SoS FfC DwD
## 56                      2     0       1   1   1   1   1   1
```

Следовательно, если в `Book.of.Death` стоит `NA`, мы можем предположить, что Джордж Мартин еще не занес своей карающей руки над этим героем.

Мы можем создать новую колонку `IsAlive`:

```
got$IsAlive <- is.na(got$Book.of.Death)
```

2.12.4 data.table vs. tidyverse

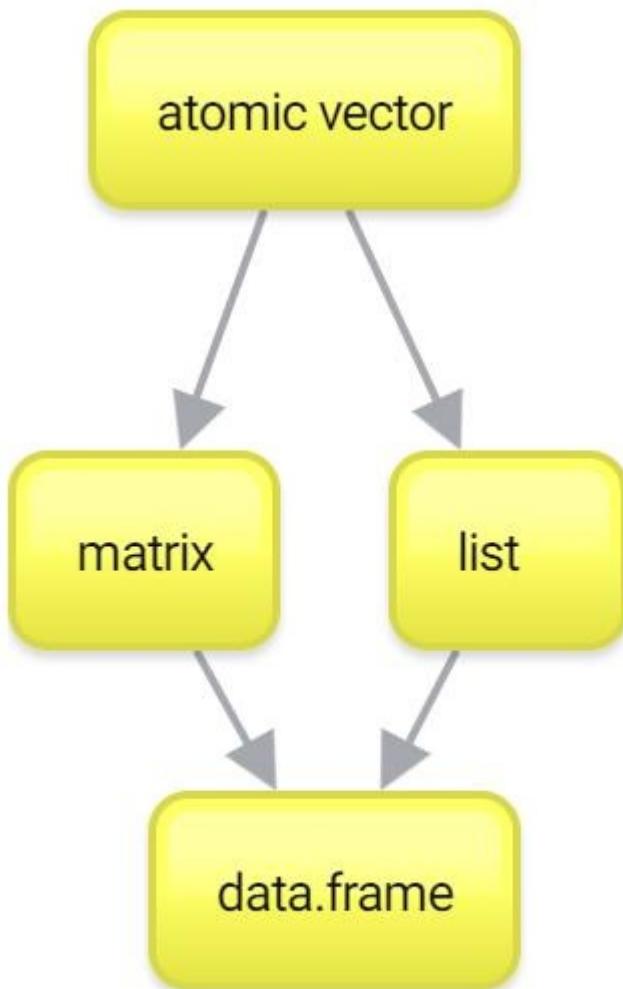
В принципе, с помощью базового R можно сделать все, что угодно. Однако базовые инструменты R — не всегда самые удобные. Идея сделать работу с датафреймами в R еще быстрее и удобнее сподвигла разработчиков на создание новых инструментов — `data.table` и `tidyverse` (`dplyr`). Это два конкурирующих подхода, которые сильно перерабатывают язык, хотя это по-прежнему все тот же R — поэтому их еще называют “диалектами” R.

Оба подхода обладают своими преимуществами и недостатками, но на сегодняшний день `tidyverse` считается более популярным. Основное преимущество этого подхода — в относительной легкости освоения. Обычно код, написанный в `tidyverse` можно примерно понять, даже не владея им.

Преимущество `data.table` — в суровом лаконичном синтаксисе и наиболее эффективных алгоритмах. Последние обеспечивают очень серьезный прирост в скорости в работе с данными. Чтение файлов и манипуляция данными может

быть на порядки быстрее, поэтому если Ваш датасет с трудом пролезает в оперативную память компьютера, а исполнение скрипта занимает длительное время - стоит задуматься о переходе на `data.table`.

Что из этого учить — решать Вам, но знать оба совсем не обязательно: они решают те же самые задачи, просто совсем разными способами. За `data.table` — скорость, за `tidyverse` — понятность синтаксиса. Очень советую почитать обсуждение на эту тему здесь²³.



²³<https://stackoverflow.com/questions/21435339/data-table-vs-dplyr-can-one-do-something-well-the-other-cant-or-doesnt>

Глава 3

tidyverse: Загрузка и трансформация данных

*tidyverse*¹ — это набор пакетов:

- *ggplot2*, для визуализации
- *tibble*, для работы с тиблами, современный вариант датафрейма
- *tidyverse*, для формата tidy data
- *readr*, для чтения файлов в R
- *purrr*, для функционального программирования
- *dplyr*, для преобразования данных
- *stringr*, для работы со строковыми переменными
- *forcats*, для работы с переменными-факторами

Полезно также знать о следующих:

- *readxl*, для чтения .xls и .xlsx
- *jsonlite*, для работы с JSON
- *rvest*, для веб-скраппинга
- *lubridate*, для работы с временем
- *tidytext*, для работы с текстами и корпусами
- *broom*, для перевода в tidy формат статистические модели

```
library("tidyverse")
```

¹<https://www.tidyverse.org>

3.1 Загрузка данных

3.1.1 Рабочая директория

Все в R происходит где-то. Нужно загружать файлы с данными, нужно их куда-то сохранять. Желательно иметь для каждого проекта некоторую отдельную папку на компьютере, куда складывать все, относящееся к этому проекту. Две команды позволяют определить текущую рабочую директорию (`getwd()`) и (`setwd(.../path/to/your/directory)`).

3.1.2 Форматы данных: .csv

Существует много форматов данных, которые придумали люди. Большинство из них можно загрузить в R. Так как центральный объект в R – таблица $n \times k$, то и работать мы большую часть времени будем с таблицами. Наиболее распространенные способы хранить данные сейчас это `.csv` (разберем в данном разделе) и `.json` (разберем в разделе 5).

`.csv` (comma separated values) – является обычным текстовым файлом, в котором перечислены значения с некоторым фиксированным разделителем: запятой, табуляцией, точка с запятой, пробел и др. Такие файлы обычно открываются LibreOffice, а в Microsoft Excel нужны некоторые трюки².

3.1.3 Загрузка данных: `readr`, `readxl`

Стандартной функцией для чтения `.csv` файлов в R является функция `read.csv()`, но мы будем использовать функцию `read_csv()` из пакета `readr`.

```
read_csv("...")
```

Вместо многоточия может стоять:

- название файла (если он, есть в текущей рабочей директории)

```
read_csv("my_file.csv")
```

- относительный путь к файлу (если он, верен для текущей рабочей директории)

```
read_csv("data/my_file.csv")
```

- полный путь к файлу (если он, верен для текущей рабочей директории)

²<https://superuser.com/questions/291445/how-do-you-change-default-delimiter-in-the-text-import-in-excel>

```
read_csv("/home/user_name/work/data/my_file.csv")
```

- интернет ссылка (тогда, компьютер должен быть подключен к интернету)

```
read_csv("https://my_host/my_file.csv")
```

Для чтения других форматов .csv файлов используются другие функции:

- `read_tsv()` – для файлов с табуляцией в качестве разделителя
- `read_csv2()` – для файлов с точкой с запятой в качестве разделителя
- `read_delim(file = "...", delim = "...")` – для файлов с любым разделителем, задаваемым аргументом `delim`

Стандартной практикой является создавать первой строкой .csv файлов названия столбцов, поэтому по умолчанию функции `read_...` () будут создавать таблицу, считая первую строку названием столбцов. Чтобы изменить это поведение следует использовать аргумент `col_names = FALSE`.

Другая проблема при чтении файлов – кодировка и локаль. На разных компьютерах разные локали и дефолтные кодировки, так что имеет смысл знать про аргумент `locale(encoding = "UTF-8")`.

Попробуйте корректно считать в R файл по этой ссылке³.

```
## # A tibble: 3 x 3
##   cyrillic ipa_symbols greek
##   <chr>     <chr>       <chr>
## 1
## 2
## 3
```

Благодаря `readxl` пакету Также данные можно скачать напрямую из файлов .xls (функция `read_xls`) и .xlsx (функция `read_xlsx`), однако эти функции не умеют читать из интернета.

```
library("readxl")
xlsx_example <- read_xlsx("...")
```

Существует еще один экстравагантный способ хранить данные: это формат файлов R .RData. Создадим `data.frame`:

³https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/data/scary_letters.csv

```
my_df <- data.frame(letters = c("a", "b"),
                     numbers = 1:2)
my_df

##   letters numbers
## 1         a        1
## 2         b        2
```

Теперь можно сохранить файл...

```
save(my_df, file = "data/my_df.RData")
```

удалить переменную...

```
rm(my_df)
my_df
```

Error in eval(expr, envir, enclos): object 'my_df' not found
и загрузить все снова:

```
load("data/my_df.RData")
my_df
```

```
##   letters numbers
## 1         a        1
## 2         b        2
```

3.1.3.1 Misspelling dataset

Этот датасет я переработал из данных, собранных для статьи The Gyllenhaal Experiment⁴, написанной Расселом Гольденбергом и Мэттом Дэниэлсом для издания *pudding*⁵. Они анализировали ошибки в правописании при поиске имен и фамилий звезд.

```
misspellings <- read_csv("https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/misspellings.csv")
```

```
## Parsed with column specification:
## cols(
##   correct = col_character(),
##   spelling = col_character(),
```

⁴<https://pudding.cool/2019/02/gyllenhaal/>

⁵<https://pudding.cool>

```

##   count = col_double()
## )

misspellings

## # A tibble: 15,477 x 3
##   correct  spelling   count
##   <chr>    <chr>     <dbl>
## 1 deschanel deschanel 18338
## 2 deschanel dechanel  1550
## 3 deschanel deschannel 934
## 4 deschanel deschenel  404
## 5 deschanel deshanel  364
## 6 deschanel dechannel 359
## 7 deschanel deschanelle 316
## 8 deschanel dechanelle 192
## 9 deschanel deschanell 174
## 10 deschanel deschenal 165
## # ... with 15,467 more rows

```

В датасете следующие переменные:

- correct – корректное написание фамилии
- spelling – написание, которое сделали пользователи
- count – количество случаев такого написания

3.1.3.2 diamonds

diamonds

```

## # A tibble: 53,940 x 10
##   carat cut      color clarity depth table price     x     y     z
##   <dbl> <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23 Ideal     E     SI2     61.5    55   326  3.95  3.98  2.43
## 2 0.21 Premium   E     SI1     59.8    61   326  3.89  3.84  2.31
## 3 0.23 Good      E     VS1     56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium  I     VS2     62.4    58   334  4.2   4.23  2.63
## 5 0.31 Good      J     SI2     63.3    58   335  4.34  4.35  2.75
## 6 0.24 Very Good J     VVS2    62.8    57   336  3.94  3.96  2.48
## 7 0.24 Very Good I     VVS1    62.3    57   336  3.95  3.98  2.47
## 8 0.26 Very Good H     SI1     61.9    55   337  4.07  4.11  2.53
## 9 0.22 Fair       E     VS2     65.1    61   337  3.87  3.78  2.49
## 10 0.23 Very Good H    VS1     59.4    61   338   4     4.05  2.39
## # ... with 53,930 more rows

```

```
?diamonds
```

3.2 tibble

Пакет `tibble` – является альтернативой штатного датафрейма в R. Существует встроенная переменная `month.name`:

```
month.name
```

```
## [1] "January"    "February"   "March"      "April"       "May"
## [6] "June"        "July"        "August"     "September"  "October"
## [11] "November"   "December"
```

Можно создать датафрейм таким образом:

```
data.frame(id = 1:12,
           months = month.name,
           n_letters = nchar(months))
```

```
## Error in nchar(months): cannot coerce type 'closure' to vector of type 'character'
```

Однако переменная `months` не создана пользователем, так что данный код выдаст ошибку. Корректный способ сделать это базовыми средствами:

```
data.frame(id = 1:12,
           months = month.name,
           n_letters = nchar(month.name))
```

| | <code>id</code> | <code>months</code> | <code>n_letters</code> |
|-------|-----------------|---------------------|------------------------|
| ## 1 | 1 | January | 7 |
| ## 2 | 2 | February | 8 |
| ## 3 | 3 | March | 5 |
| ## 4 | 4 | April | 5 |
| ## 5 | 5 | May | 3 |
| ## 6 | 6 | June | 4 |
| ## 7 | 7 | July | 4 |
| ## 8 | 8 | August | 6 |
| ## 9 | 9 | September | 9 |
| ## 10 | 10 | October | 7 |
| ## 11 | 11 | November | 8 |
| ## 12 | 12 | December | 8 |

Одно из отличий `tibble` от базового датафрейма – возможность использовать создаваемые “по ходу пьесы” переменные”

```
tibble(id = 1:12,
       months = month.name,
       n_letters = nchar(months))

## # A tibble: 12 x 3
##       id months   n_letters
##   <int> <chr>     <int>
## 1     1 January      7
## 2     2 February     8
## 3     3 March        5
## 4     4 April        5
## 5     5 May          3
## 6     6 June         4
## 7     7 July         4
## 8     8 August       6
## 9     9 September    9
## 10    10 October     7
## 11    11 November    8
## 12    12 December    8
```

Если в окружении пользователя уже есть переменная с датафреймом, его легко можно переделать в `tibble` при помощи функции `as_tibble()`:

```
df <- data.frame(id = 1:12,
                  months = month.name)

df

##   id   months
## 1 1   January
## 2 2  February
## 3 3    March
## 4 4   April
## 5 5     May
## 6 6    June
## 7 7    July
## 8 8   August
## 9 9 September
## 10 10 October
## 11 11 November
## 12 12 December

as_tibble(df)
```

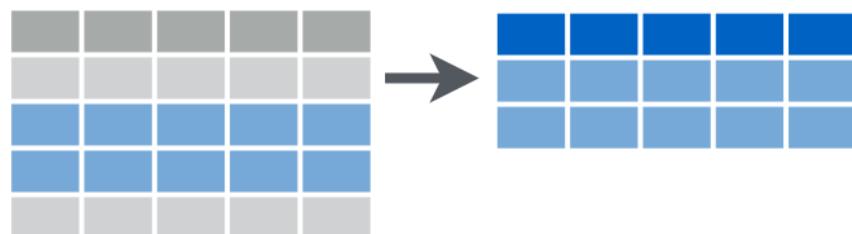
```
## # A tibble: 12 x 2
##       id months
##   <int> <fct>
## 1     1 January
## 2     2 February
## 3     3 March
## 4     4 April
## 5     5 May
## 6     6 June
## 7     7 July
## 8     8 August
## 9     9 September
## 10    10 October
## 11    11 November
## 12    12 December
```

Функционально `tibble` от `data.frame` ничем не отличается, однако существует ряд несущественных отличий. Кроме того стоит помнить, что многие функции из `tidyverse` возвращают именно `tibble`, а не `data.frame`.

3.3 dplyr

В сжатом виде содержание этого раздела хранится вот здесь⁶.

3.3.1 dplyr::filter()



Эта функция фильтрует строчки по условиям, основанным на столцах.

Сколько неправильных произношений, которые написали меньше 10 юзеров?

```
misspellings %>%
  filter(count < 10)
```

⁶<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

```
## # A tibble: 14,279 x 3
##   correct spelling   count
##   <chr>    <chr>     <dbl>
## 1 deschanel deshanael     9
## 2 deschanel daychanel     9
## 3 deschanel deschaneles    9
## 4 deschanel dashenel     9
## 5 deschanel deschenael    9
## 6 deschanel deechanel     9
## 7 deschanel deichanel     9
## 8 deschanel dechantel    9
## 9 deschanel deychanel     9
## 10 deschanel daschenell   9
## # ... with 14,269 more rows
```

%>% — конвеер (pipe) отправляет результат работы одной функции в другую.

```
sort(sqrt(abs(sin(1:22))), decreasing = TRUE)

## [1] 0.9999951 0.9952926 0.9946649 0.9805088 0.9792468 0.9554817 0.9535709
## [8] 0.9173173 0.9146888 0.8699440 0.8665952 0.8105471 0.8064043 0.7375779
## [15] 0.7325114 0.6482029 0.6419646 0.5365662 0.5285977 0.3871398 0.3756594
## [22] 0.0940814

1:22 %>%
  sin() %>%
  abs() %>%
  sqrt() %>%
  sort(., decreasing = TRUE) #           ?

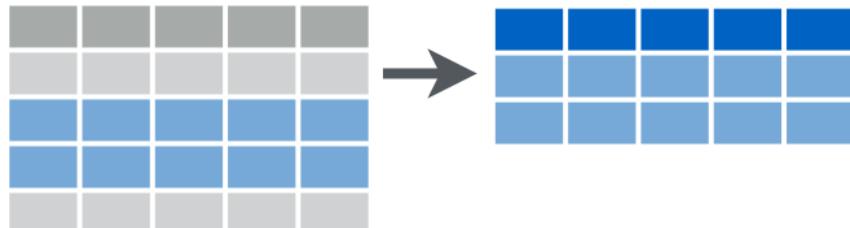
## [1] 0.9999951 0.9952926 0.9946649 0.9805088 0.9792468 0.9554817 0.9535709
## [8] 0.9173173 0.9146888 0.8699440 0.8665952 0.8105471 0.8064043 0.7375779
## [15] 0.7325114 0.6482029 0.6419646 0.5365662 0.5285977 0.3871398 0.3756594
## [22] 0.0940814
```

Конвееры в *tidyverse* пришли из пакета *magrittr*. Иногда они работают не корректно с функциями не из *tidyverse*.



3.3.2 dplyr::slice()

Эта функция фильтрует строчки по индексу.



```
misspellings %>%
  slice(3:7)
```

```
## # A tibble: 5 x 3
##   correct    spelling   count
##   <chr>      <chr>     <dbl>
## 1 deschanel deschannel  934
## 2 deschanel deschenel   404
## 3 deschanel deshanel    364
## 4 deschanel dechannel   359
## 5 deschanel deschanelle 316
```

3.3.3 dplyr::select()

Эта функция позволяет выбрать столбцы.



```
diamonds %>%
  select(8:10)
```

```
## # A tibble: 53,940 x 3
```

```

##      x     y     z
##  <dbl> <dbl> <dbl>
## 1  3.95  3.98  2.43
## 2  3.89  3.84  2.31
## 3  4.05  4.07  2.31
## 4  4.2   4.23  2.63
## 5  4.34  4.35  2.75
## 6  3.94  3.96  2.48
## 7  3.95  3.98  2.47
## 8  4.07  4.11  2.53
## 9  3.87  3.78  2.49
## 10 4     4.05  2.39
## # ... with 53,930 more rows

diamonds %>%
  select(color:price)

## # A tibble: 53,940 x 5
##   color clarity depth table price
##   <ord> <ord>   <dbl> <dbl> <int>
## 1 E     SI2     61.5   55    326
## 2 E     SI1     59.8   61    326
## 3 E     VS1     56.9   65    327
## 4 I     VS2     62.4   58    334
## 5 J     SI2     63.3   58    335
## 6 J     VVS2    62.8   57    336
## 7 I     VVS1    62.3   57    336
## 8 H     SI1     61.9   55    337
## 9 E     VS2     65.1   61    337
## 10 H    VS1     59.4   61    338
## # ... with 53,930 more rows

diamonds %>%
  select(-carat)

## # A tibble: 53,940 x 9
##   cut      color clarity depth table price     x     y     z
##   <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 Ideal    E     SI2     61.5   55    326  3.95  3.98  2.43
## 2 Premium  E     SI1     59.8   61    326  3.89  3.84  2.31
## 3 Good    E     VS1     56.9   65    327  4.05  4.07  2.31
## 4 Premium I     VS2     62.4   58    334  4.2   4.23  2.63
## 5 Good    J     SI2     63.3   58    335  4.34  4.35  2.75
## 6 Very Good J    VVS2    62.8   57    336  3.94  3.96  2.48
## 7 Very Good I    VVS1    62.3   57    336  3.95  3.98  2.47

```

```
## 8 Very Good H     SI1      61.9    55    337  4.07  4.11  2.53
## 9 Fair          E     VS2      65.1    61    337  3.87  3.78  2.49
## 10 Very Good H   VS1      59.4    61    338   4     4.05  2.39
## # ... with 53,930 more rows
```

```
diamonds %>%
  select(-c(carat, cut, x, y, z))
```

```
## # A tibble: 53,940 x 5
##       color clarity depth table price
##       <ord>   <ord>   <dbl> <dbl> <int>
## 1 E        SI2      61.5    55    326
## 2 E        SI1      59.8    61    326
## 3 E        VS1      56.9    65    327
## 4 I        VS2      62.4    58    334
## 5 J        SI2      63.3    58    335
## 6 J        VVS2     62.8    57    336
## 7 I        VVS1     62.3    57    336
## 8 H        SI1      61.9    55    337
## 9 E        VS2      65.1    61    337
## 10 H       VS1      59.4    61    338
## # ... with 53,930 more rows
```

```
diamonds %>%
  select(cut, depth, price)
```

```
## # A tibble: 53,940 x 3
##       cut      depth price
##       <ord>    <dbl> <int>
## 1 Ideal      61.5   326
## 2 Premium    59.8   326
## 3 Good       56.9   327
## 4 Premium    62.4   334
## 5 Good       63.3   335
## 6 Very Good  62.8   336
## 7 Very Good  62.3   336
## 8 Very Good  61.9   337
## 9 Fair        65.1   337
## 10 Very Good 59.4   338
## # ... with 53,930 more rows
```

3.3.4 dplyr::arrange()

Эта функция сортирует (строки по алфавиту, а числа по порядку).

```

misspellings %>%
  arrange(count)

## # A tibble: 15,477 x 3
##   correct spelling count
##   <chr>     <chr>    <dbl>
## 1 deschanel deschil     1
## 2 deschanel deshauneil   1
## 3 deschanel deschmuel    1
## 4 deschanel deshannle    1
## 5 deschanel deslanges    1
## 6 deschanel deshoenel    1
## 7 deschanel dechadel    1
## 8 deschanel dooschaney   1
## 9 deschanel dishana     1
## 10 deschanel deshaneil   1
## # ... with 15,467 more rows

diamonds %>%
  arrange(desc(carat), price)

## # A tibble: 53,940 x 10
##   carat cut      color clarity depth table price     x     y     z
##   <dbl> <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 5.01 Fair      J      I1     65.5    59 18018 10.7  10.5  6.98
## 2 4.5  Fair     J      I1     65.8    58 18531 10.2  10.2  6.72
## 3 4.13 Fair     H      I1     64.8    61 17329 10     9.85  6.43
## 4 4.01 Premium   I      I1     61      61 15223 10.1  10.1  6.17
## 5 4.01 Premium   J      I1     62.5    62 15223 10.0  9.94  6.24
## 6 4  Very Good  I      I1     63.3    58 15984 10.0  9.94  6.31
## 7 3.67 Premium   I      I1     62.4    56 16193 9.86  9.81  6.13
## 8 3.65 Fair     H      I1     67.1    53 11668 9.53  9.48  6.38
## 9 3.51 Premium   J      VS2    62.5    59 18701 9.66  9.63  6.03
## 10 3.5 Ideal     H     I1     62.8    57 12587 9.65  9.59  6.03
## # ... with 53,930 more rows

### dplyr::distinct() Эта функция возвращает уникальные значения в столбце или комбинации столбцов.

misspellings %>%
  distinct(correct)

## # A tibble: 15 x 1
##   correct
##   <chr>
## 1 deschanel

```

```

##      <chr>
## 1 deschanel
## 2 mclachlan
## 3 galifianakis
## 4 labeouf
## 5 macaulay
## 6 mcconaughey
## 7 minaj
## 8 morissette
## 9 poehler
## 10 shyamalan
## 11 kaepernick
## 12 mcgwire
## 13 palahniuk
## 14 picabo
## 15 johansson

misspellings %>%
  distinct(spelling)

## # A tibble: 15,462 x 1
##       spelling
##      <chr>
## 1 deschanel
## 2 dechanel
## 3 deschannel
## 4 deschenel
## 5 deshanel
## 6 dechannel
## 7 deschanelle
## 8 dechanelle
## 9 deschanell
## 10 deschenal
## # ... with 15,452 more rows

diamonds %>%
  distinct(color, cut)

## # A tibble: 35 x 2
##       color     cut
##      <ord> <ord>
## 1 E      Ideal
## 2 E      Premium
## 3 E      Good
## 4 I      Premium

```

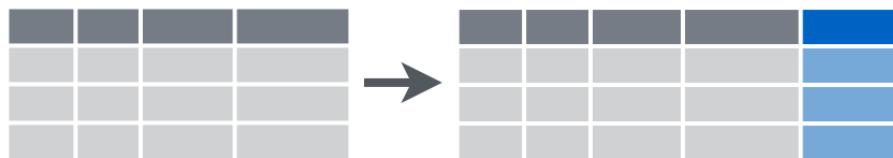
```
## 5 J Good
## 6 J Very Good
## 7 I Very Good
## 8 H Very Good
## 9 E Fair
## 10 J Ideal
## # ... with 25 more rows
```



Во встроенным в tidyverse датасете starwars отфильтруйте существ выше 180 (height) и весом меньше 80 (mass) и выведите уникальные значения мест, откуда они происходят (homeworld).

3.3.5 dplyr::mutate()

Эта функция позволяет создать новые переменные.



```
misspellings %>%
  mutate(misspelling_length = nchar(spelling),
        id = 1:n())
```

```
## # A tibble: 15,477 x 5
##   correct spelling   count misspelling_length     id
##   <chr>    <chr>   <dbl>              <int>      <int>
## 1 deschanel deschanel  18338                 9       1
## 2 deschanel dechannel  1550                  8       2
## 3 deschanel deschannel  934                  10      3
## 4 deschanel deschenel   404                  9       4
## 5 deschanel deshanel   364                  8       5
## 6 deschanel dechannel   359                  9       6
## 7 deschanel deschanelle  316                 11      7
## 8 deschanel dechanelle   192                 10      8
## 9 deschanel deschanell   174                 10      9
## 10 deschanel deschenal   165                 9      10
## # ... with 15,467 more rows
```



Создайте переменную с индексом Кетле⁷: $\frac{mass}{height^2}$ для всех существ датасета `starwars`. Сколько героев страдают ожирением (т. е. имеют индекс массы тела больше 30)? (Не забудьте перевести рост из сантиметров в метры).

3.3.6 `dplyr::group_by(...)` %>% `summarise(...)`

Эта функция позволяет сгруппировать переменные по какому-то из столбцов и получить какой-нибудь вывод из описательной статистики (максимум, минимум, последний, первый, среднее, медиану и т. п.).



```
misspellings %>%
  summarise(min(count), mean(count))
```

```
## # A tibble: 1 x 2
##   `min(count)` `mean(count)`
##       <dbl>        <dbl>
## 1           1        21.8
```

```
misspellings %>%
  group_by(correct) %>%
  summarise(mean(count))
```

```
## # A tibble: 15 x 2
##   correct      `mean(count)`
##   <chr>          <dbl>
## 1 deschanel     25.9
## 2 galifianakis  8.64
## 3 johansson    74.8
## 4 kaepernick    29.1
## 5 labeouf       61.2
## 6 macaulay      17.6
## 7 mcconaughey   7.74
## 8 mcgwire       55.3
## 9 mclachlan     14.8
```

```

## 10 minaj          140.
## 11 morissette    55.2
## 12 palahniuk     10.2
## 13 picabo         23.2
## 14 poehler        65.3
## 15 shyamalan      16.9

misspellings %>%
  group_by(correct) %>%
  summarise(my_mean = mean(count))

## # A tibble: 15 x 2
##   correct      my_mean
##   <chr>        <dbl>
## 1 deschanel     25.9
## 2 galifianakis  8.64
## 3 johansson     74.8
## 4 kaepernick    29.1
## 5 labeouf       61.2
## 6 macaulay      17.6
## 7 mcconaughey   7.74
## 8 mcgwire       55.3
## 9 mclachlan     14.8
## 10 minaj        140.
## 11 morissette   55.2
## 12 palahniuk    10.2
## 13 picabo        23.2
## 14 poehler       65.3
## 15 shyamalan     16.9

```

Если нужно посчитать количество вхождений, то можно использовать функцию n() в summarise() или же функцию count():

```

misspellings %>%
  group_by(correct) %>%
  summarise(n = n())

## # A tibble: 15 x 2
##   correct      n
##   <chr>     <int>
## 1 deschanel   1015
## 2 galifianakis 2633
## 3 johansson   392
## 4 kaepernick   779
## 5 labeouf      449

```

```
## 6 macaulay      1458
## 7 mcconaughey   2897
## 8 mcgwire       262
## 9 mclachlan     1054
## 10 minaj        200
## 11 morissette   478
## 12 palahniuk    1541
## 13 picabo       460
## 14 poehler      386
## 15 shyamalan    1473
```

```
misspellings %>%
  count(correct)
```

```
## # A tibble: 15 x 2
##   correct      n
##   <chr>     <int>
## 1 deschanel    1015
## 2 galifianakis 2633
## 3 johansson    392
## 4 kaepernick    779
## 5 labeouf      449
## 6 macaulay     1458
## 7 mcconaughey   2897
## 8 mcgwire       262
## 9 mclachlan     1054
## 10 minaj        200
## 11 morissette   478
## 12 palahniuk    1541
## 13 picabo       460
## 14 poehler      386
## 15 shyamalan    1473
```



А что будет, если в датасете misspellings создать переменную n и оставить отсортировать по переменным correct и n?

Можно даже отсортировать результат:

```
misspellings %>%
  count(correct, sort = TRUE)
```

```
## # A tibble: 15 x 2
##   correct      n
##   <chr>     <int>
```

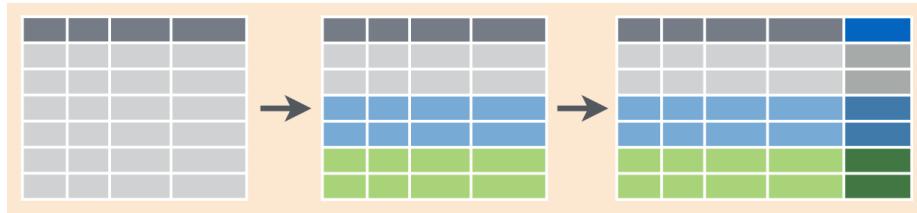
```
## 1 mcconaughey    2897
## 2 galifianakis  2633
## 3 palahniuk     1541
## 4 shyamalan     1473
## 5 macaulay       1458
## 6 mclachlan      1054
## 7 deschanel       1015
## 8 kaepernick     779
## 9 morissette     478
## 10 picabo        460
## 11 labeouf        449
## 12 johansson     392
## 13 poehler        386
## 14 mcgwire        262
## 15 minaj          200
```

Если вы хотите создать не какое-то саммари, а целый дополнительный столбец с этим саммари вместо функции `summarise()` нужно использовать функцию `mutate()`:

```
misspellings %>%
  group_by(correct) %>%
  mutate(my_mean = mean(count))
```

```
## # A tibble: 15,477 x 4
## # Groups:   correct [15]
##   correct spelling   count my_mean
##   <chr>    <chr>     <dbl>    <dbl>
## 1 deschanel deschanel  18338    25.9
## 2 deschanel dechanel   1550     25.9
## 3 deschanel deschannel  934     25.9
## 4 deschanel deschenel   404     25.9
## 5 deschanel deshanel    364     25.9
## 6 deschanel dechannel   359     25.9
## 7 deschanel deschanelle  316     25.9
## 8 deschanel dechanelle  192     25.9
## 9 deschanel deschanell   174     25.9
## 10 deschanel deschenal   165     25.9
## # ... with 15,467 more rows
```

Схематически это выглядит так:



В датасете starwars запишите в отдельную переменную среднее значение роста (height) по каждой расе (species).

3.3.7 bind_...

3.3.8 dplyr::..._join()

Эти функции позволяют соединять датафреймы.

```
languages <- data_frame(
  languages = c("Selkup", "French", "Chukchi", "Polish"),
  countries = c("Russia", "France", "Russia", "Poland"),
  iso = c("sel", "fra", "ckt", "pol")
)
languages

## # A tibble: 4 x 3
##   languages countries iso
##   <chr>     <chr>    <chr>
## 1 Selkup     Russia    sel
## 2 French     France    fra
## 3 Chukchi    Russia    ckt
## 4 Polish     Poland    pol

country_population <- data_frame(
  countries = c("Russia", "Poland", "Finland"),
  population_mln = c(143, 38, 5))
country_population

## # A tibble: 3 x 2
##   countries population_mln
##   <chr>           <dbl>
## 1 Russia            143
## 2 Poland             38
## 3 Finland            5
```

```
inner_join(languages, country_population)
```

```
## Joining, by = "countries"  
## # A tibble: 3 x 4  
##   languages countries iso   population_mln  
##   <chr>     <chr>    <chr>        <dbl>  
## 1 Selkup     Russia    sel         143  
## 2 Chukchi    Russia    ckt         143  
## 3 Polish     Poland    pol          38
```

```
left_join(languages, country_population)
```

```
## Joining, by = "countries"  
## # A tibble: 4 x 4  
##   languages countries iso   population_mln  
##   <chr>     <chr>    <chr>        <dbl>  
## 1 Selkup     Russia    sel         143  
## 2 French     France    fra         NA  
## 3 Chukchi    Russia    ckt         143  
## 4 Polish     Poland    pol          38
```

```
right_join(languages, country_population)
```

```
## Joining, by = "countries"  
## # A tibble: 4 x 4  
##   languages countries iso   population_mln  
##   <chr>     <chr>    <chr>        <dbl>  
## 1 Selkup     Russia    sel         143  
## 2 Chukchi    Russia    ckt         143  
## 3 Polish     Poland    pol          38  
## 4 <NA>       Finland   <NA>          5
```

```
anti_join(languages, country_population)
```

```
## Joining, by = "countries"  
## # A tibble: 1 x 3  
##   languages countries iso  
##   <chr>     <chr>    <chr>  
## 1 French     France    fra
```

```
anti_join(country_population, languages)
```

```
## Joining, by = "countries"
```

```
## # A tibble: 1 x 2
##   countries population_mln
##   <chr>          <dbl>
## 1 Finland           5
```

```
full_join(country_population, languages)
```

```
## Joining, by = "countries"
```

```
## # A tibble: 5 x 4
##   countries population_mln languages iso
##   <chr>          <dbl> <chr>    <chr>
## 1 Russia           143  Selkup    sel
## 2 Russia           143  Chukchi   ckt
## 3 Poland            38  Polish    pol
## 4 Finland            5  <NA>     <NA>
## 5 France            NA French   fra
```

Mutating Joins

| x1 | x2 | x3 |
|----|----|----|
| A | 1 | T |
| B | 2 | F |
| C | 3 | NA |

dplyr::left_join(a, b, by = "x1")
Join matching rows from b to a.

| x1 | x3 | x2 |
|----|----|----|
| A | T | 1 |
| B | F | 2 |
| D | T | NA |

dplyr::right_join(a, b, by = "x1")
Join matching rows from a to b.

| x1 | x2 | x3 |
|----|----|----|
| A | 1 | T |
| B | 2 | F |

dplyr::inner_join(a, b, by = "x1")
Join data. Retain only rows in both sets.

| x1 | x2 | x3 |
|----|----|----|
| A | 1 | T |
| B | 2 | F |
| C | 3 | NA |
| D | NA | T |

dplyr::full_join(a, b, by = "x1")
Join data. Retain all values, all rows.

3.4 tidyverse package

Давайте посмотрим на датасет с количеством носителей разных языков в Индии согласно переписи 2001 года (данные из Википедии):

```
langs_in_india_short <- read_csv("https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/datasets/langs_in_india_short.csv")
```

```
## Parsed with column specification:
## cols(
##   language = col_character(),
##   n_L1_sp = col_double(),
##   n_L2_sp = col_double(),
##   n_L3_sp = col_double(),
##   n_all_sp = col_double()
## )
```

- Short format

```
langs_in_india_short
```

```
## # A tibble: 12 x 5
##   language     n_L1_sp    n_L2_sp    n_L3_sp    n_all_sp
##   <chr>        <dbl>      <dbl>      <dbl>      <dbl>
## 1 Hindi        422048642  98207180  31160696  551416518
## 2 English       226449     86125221  38993066  125344736
## 3 Bengali       83369769    6637222  1108088   91115079
## 4 Telugu        74002856    9723626  1266019   84992501
## 5 Marathi       71936894    9546414  2701498   84184806
## 6 Tamil          60793814    4992253  956335    66742402
## 7 Urdu           51536111    6535489  1007912   59079512
## 8 Kannada        37924011   11455287  1396428   50775726
## 9 Gujarati       46091617    3476355  703989    50271961
## 10 Odia          33017446    3272151  319525    36609122
## 11 Malayalam     33066392    499188   195885    33761465
## 12 Sanskrit       14135      1234931  3742223   4991289
```

- Long format

```
## # A tibble: 48 x 3
##   language type     n_speakers
##   <chr>    <chr>      <dbl>
## 1 Hindi    n_L1_sp    422048642
## 2 Hindi    n_L2_sp    98207180
## 3 Hindi    n_L3_sp    31160696
## 4 Hindi    n_all_sp   551416518
## 5 English   n_L1_sp    226449
## 6 English   n_L2_sp    86125221
## 7 English   n_L3_sp    38993066
## 8 English   n_all_sp   125344736
## 9 Bengali   n_L1_sp    83369769
## 10 Bengali  n_L2_sp    6637222
## # ... with 38 more rows
```

- Short format → Long format: `tidy::pivot_longer()`

```
langs_in_india_short %>%
  pivot_longer(names_to = "type", values_to = "n_speakers", n_L1_sp:n_all_sp) ->
  langs_in_india_long
```

```
langs_in_india_long
```

```
## # A tibble: 48 x 3
##   language type     n_speakers
```

```

##   <chr>   <chr>      <dbl>
## 1 Hindi    n_L1_sp    422048642
## 2 Hindi    n_L2_sp    98207180
## 3 Hindi    n_L3_sp    31160696
## 4 Hindi    n_all_sp   551416518
## 5 English  n_L1_sp    226449
## 6 English  n_L2_sp    86125221
## 7 English  n_L3_sp    38993066
## 8 English  n_all_sp   125344736
## 9 Bengali  n_L1_sp    83369769
## 10 Bengali n_L2_sp    6637222
## # ... with 38 more rows
  • Long format → Short format: tidyverse::pivot_wider()

```

```

langs_in_india_long %>%
  pivot_wider(names_from = "type", values_from = "n_speakers")->
  langs_in_india_short
langs_in_india_short

```

```

## # A tibble: 12 x 5
##   language   n_L1_sp   n_L2_sp   n_L3_sp   n_all_sp
##   <chr>       <dbl>     <dbl>     <dbl>     <dbl>
## 1 Hindi        422048642  98207180  31160696  551416518
## 2 English      226449     86125221  38993066  125344736
## 3 Bengali      83369769   6637222   1108088   91115079
## 4 Telugu       74002856   9723626   1266019   84992501
## 5 Marathi      71936894   9546414   2701498   84184806
## 6 Tamil         60793814   4992253   956335    66742402
## 7 Urdu          51536111   6535489   1007912   59079512
## 8 Kannada      37924011   11455287  1396428   50775726
## 9 Gujarati     46091617   3476355   703989   50271961
## 10 Odia         33017446   3272151   319525   36609122
## 11 Malayalam   33066392   499188    195885   33761465
## 12 Sanskrit     14135     1234931   3742223  4991289

```


Глава 4

Визуализация данных

```
library("tidyverse")
```

4.1 Зачем визуализировать данные?

4.1.1 КвартетAnscombe

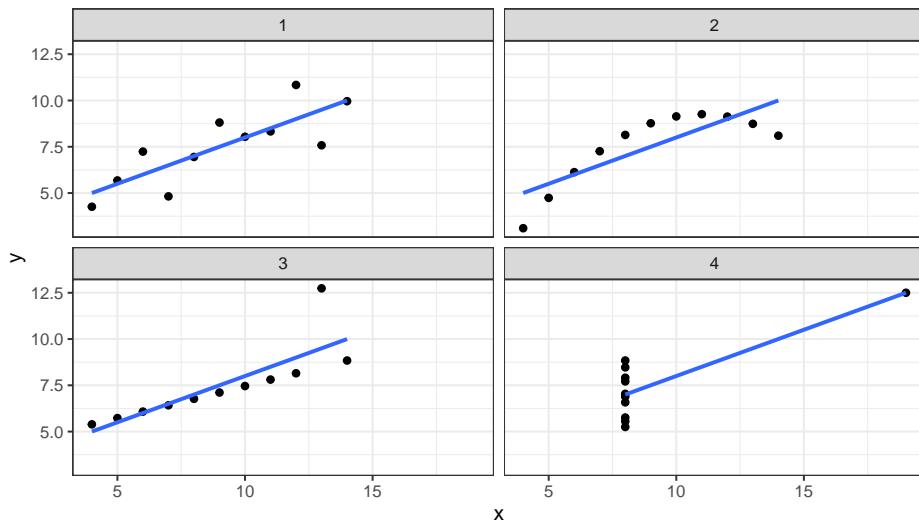
В работе Anscombe, F. J. (1973). "Graphs in Statistical Analysis" представлен следующий датасет:

```
quartet <- read_csv("https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/data/anscombe.csv")
quartet

## # A tibble: 44 x 4
##       id dataset     x     y
##   <dbl>   <dbl> <dbl> <dbl>
## 1     1      1    10  8.04
## 2     1      1    10  9.14
## 3     1      1    10  7.46
## 4     1      1     8  6.58
## 5     2      2     1  6.95
## 6     2      2     2  8.14
## 7     2      2     3  6.77
## 8     2      2     4  5.76
## 9     3      3     1  7.58
## 10    3      3     2  8.74
## # ... with 34 more rows
```

```
quartet %>%
  group_by(dataset) %>%
  summarise(mean_X = mean(x),
            mean_Y = mean(y),
            sd_X = sd(x),
            sd_Y = sd(y),
            cor = cor(x, y),
            n_obs = n()) %>%
  select(-dataset) %>%
  round(2)
```

```
## # A tibble: 4 x 6
##   mean_X mean_Y  sd_X  sd_Y    cor n_obs
##     <dbl>   <dbl> <dbl> <dbl>  <dbl>   <dbl>
## 1      9     7.5  3.32  2.03  0.82     11
## 2      9     7.5  3.32  2.03  0.82     11
## 3      9     7.5  3.32  2.03  0.82     11
## 4      9     7.5  3.32  2.03  0.82     11
```



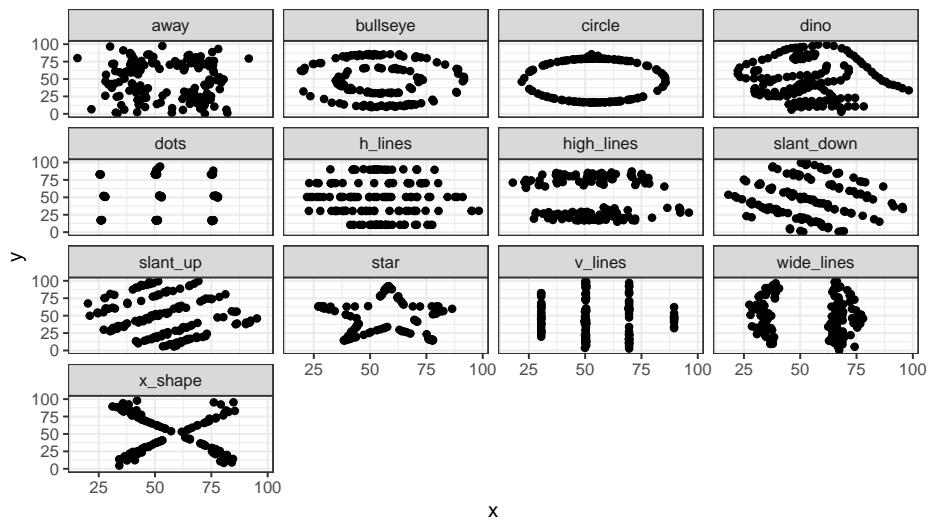
4.1.2 Датазаурус

В работе Matejka and Fitzmaurice (2017) “Same Stats, Different Graphs”¹ были представлены следующие датасеты:

¹<https://www.autodeskresearch.com/sites/default/files/SameStats-DifferentGraphs.pdf>

```
datasaurus <- read_csv("https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/data/datasaurus.csv")
datasaurus
```

```
## # A tibble: 1,846 x 3
##   dataset      x      y
##   <chr>    <dbl> <dbl>
## 1 dino     55.4  97.2
## 2 dino     51.5  96.0
## 3 dino     46.2  94.5
## 4 dino     42.8  91.4
## 5 dino     40.8  88.3
## 6 dino     38.7  84.9
## 7 dino     35.6  79.9
## 8 dino     33.1  77.6
## 9 dino     29.0  74.5
## 10 dino    26.2  71.4
## # ... with 1,836 more rows
```



```
datasaurus %>%
  group_by(dataset) %>%
  summarise(mean_X = mean(x),
            mean_Y = mean(y),
            sd_X = sd(x),
            sd_Y = sd(y),
            cor = cor(x, y),
            n_obs = n()) %>%
  select(-dataset) %>%
```

```
round(1)
```

```
## # A tibble: 13 x 6
##   mean_X mean_Y sd_X  sd_Y   cor n_obs
##   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 54.3   47.8  16.8  26.9 -0.1   142
## 2 54.3   47.8  16.8  26.9 -0.1   142
## 3 54.3   47.8  16.8  26.9 -0.1   142
## 4 54.3   47.8  16.8  26.9 -0.1   142
## 5 54.3   47.8  16.8  26.9 -0.1   142
## 6 54.3   47.8  16.8  26.9 -0.1   142
## 7 54.3   47.8  16.8  26.9 -0.1   142
## 8 54.3   47.8  16.8  26.9 -0.1   142
## 9 54.3   47.8  16.8  26.9 -0.1   142
## 10 54.3   47.8  16.8  26.9 -0.1   142
## 11 54.3   47.8  16.8  26.9 -0.1   142
## 12 54.3   47.8  16.8  26.9 -0.1   142
## 13 54.3   47.8  16.8  26.9 -0.1   142
```

4.2 Основы ggplot2

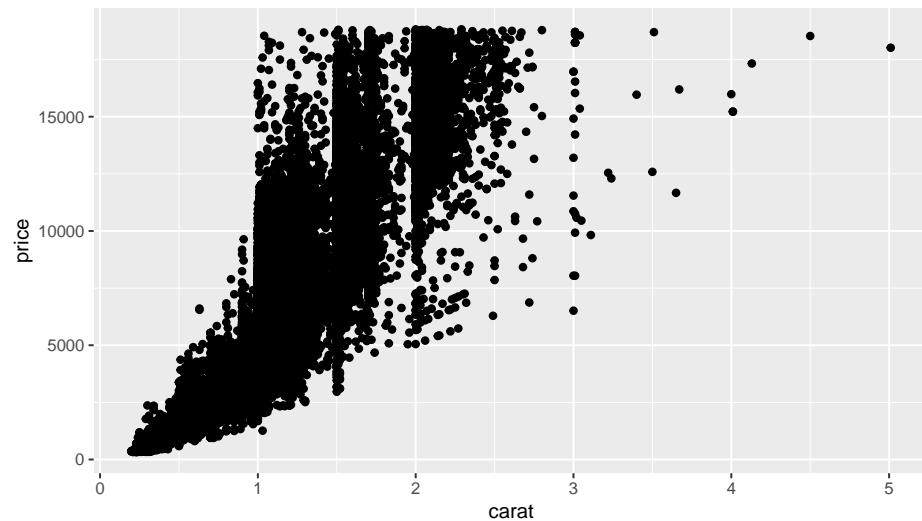
Пакет `ggplot2` – современный стандарт для создания графиков в R. Для этого пакета пишут массу расширений².

4.2.1 Диаграмма рассеяния (Scaterplot)

- `ggplot2`

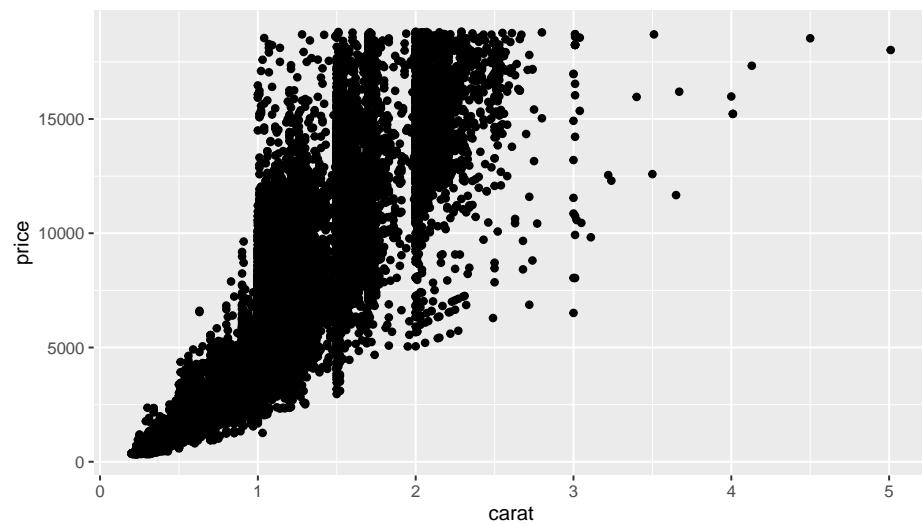
```
ggplot(data = diamonds, aes(carat, price)) +
  geom_point()
```

²<http://www.ggplot2-exts.org/gallery/>



```
• dplyr, ggplot2
```

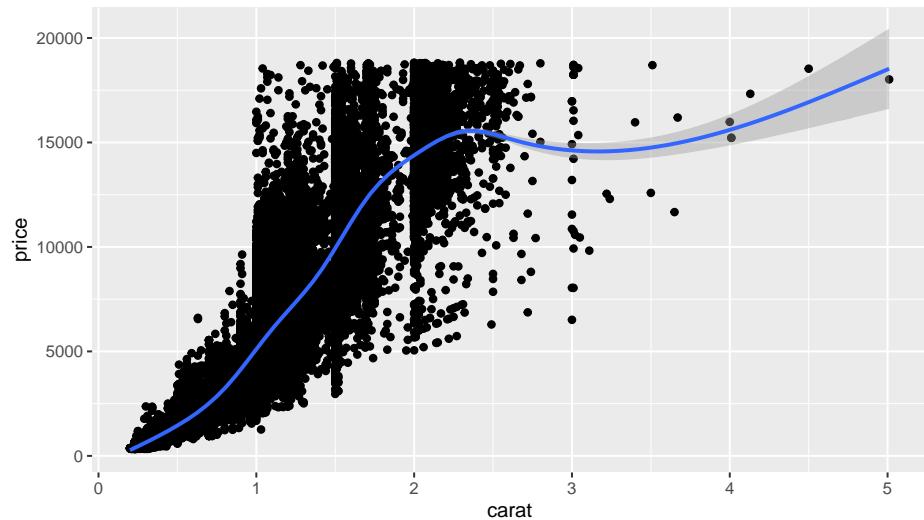
```
diamonds %>%
  ggplot(aes(carat, price))+
  geom_point()
```



4.2.2 Слои

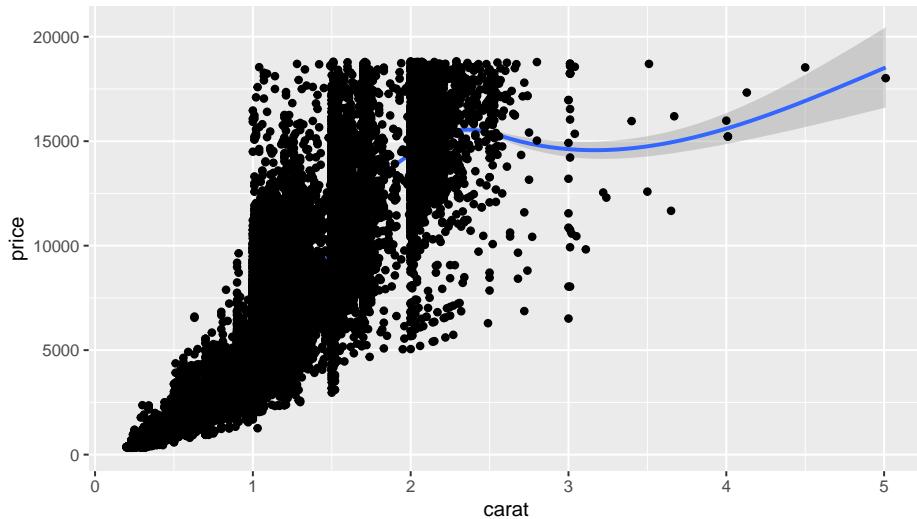
```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



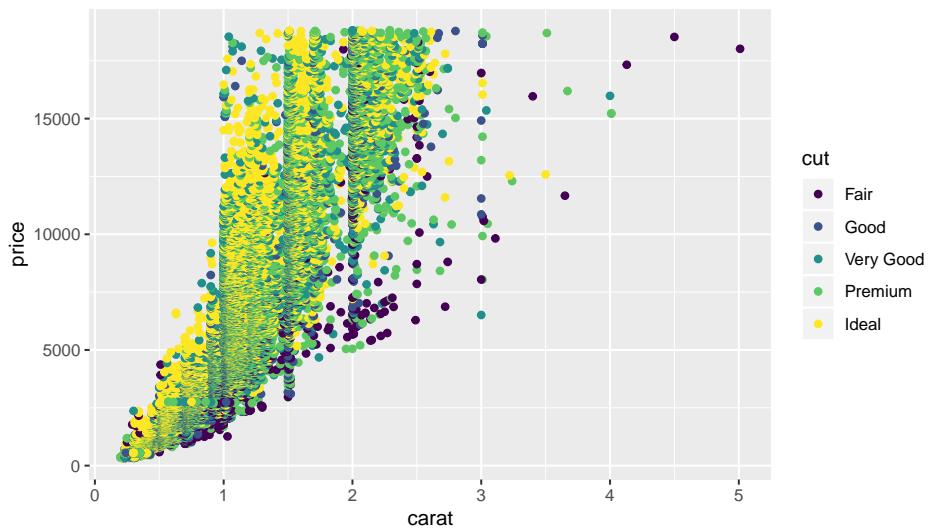
```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_smooth() +
  geom_point()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

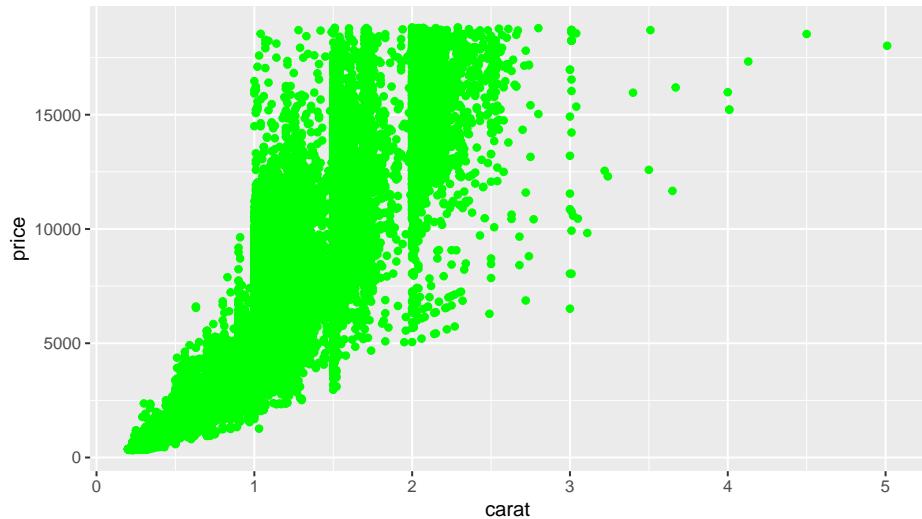


4.2.3 aes()

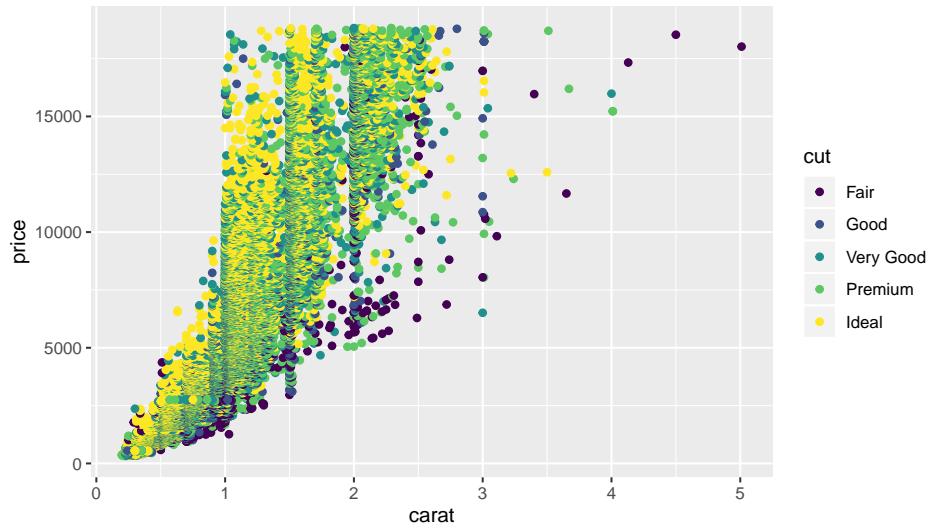
```
diamonds %>%
  ggplot(aes(carat, price, color = cut)) +
  geom_point()
```



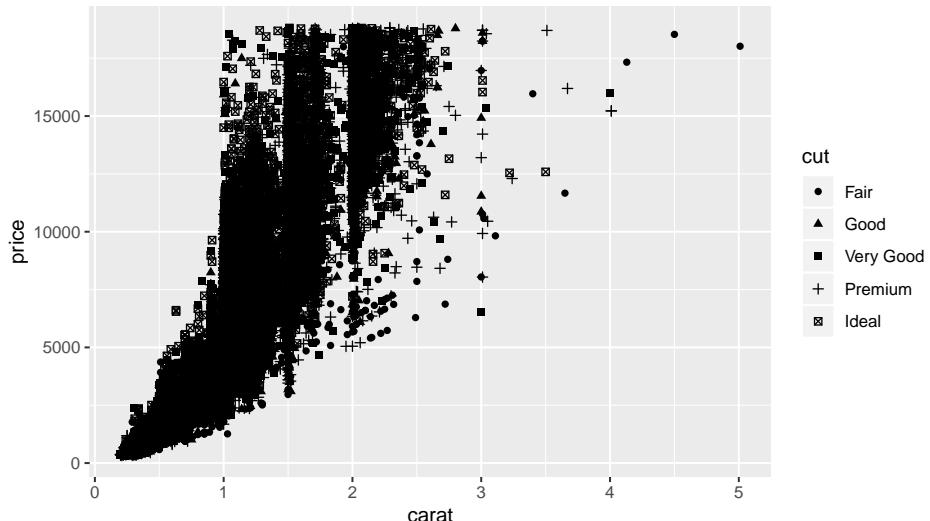
```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_point(color = "green")
```



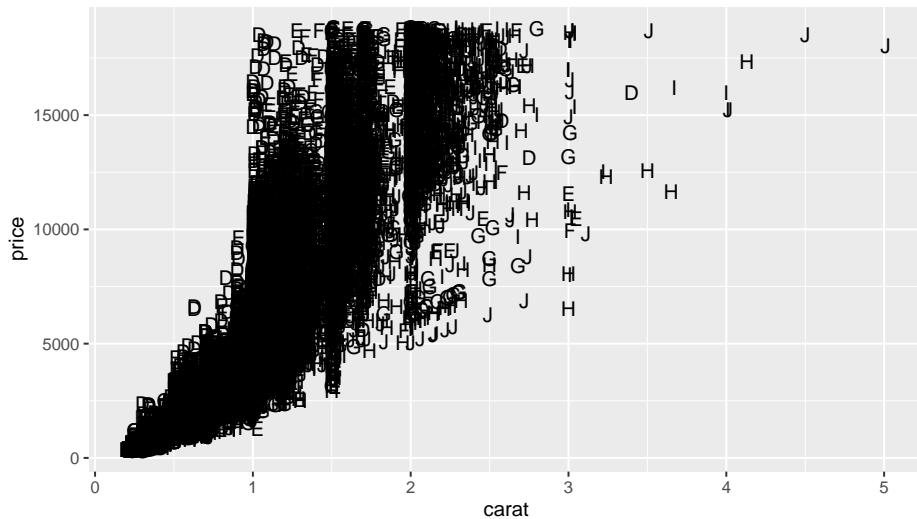
```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_point(aes(color = cut))
```



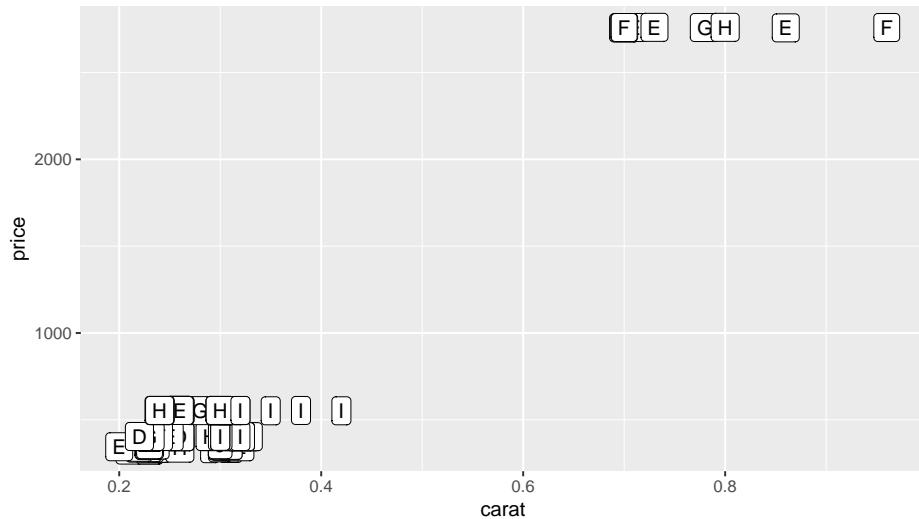
```
diamonds %>%
  ggplot(aes(carat, price, shape = cut)) +
  geom_point()
```



```
diamonds %>%
  ggplot(aes(carat, price, label = color)) +
  geom_text()
```

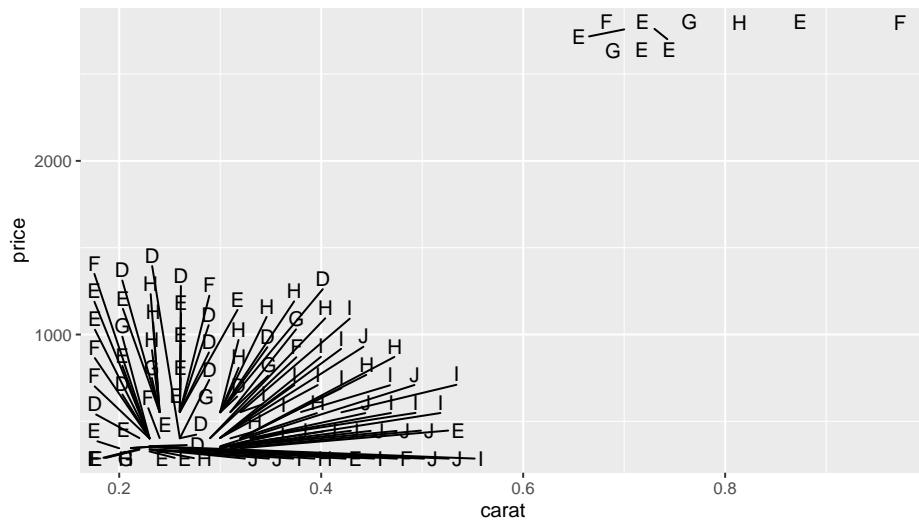


```
diamonds %>%
  slice(1:100) %>%
  ggplot(aes(carat, price, label = color)) +
  geom_label()
```



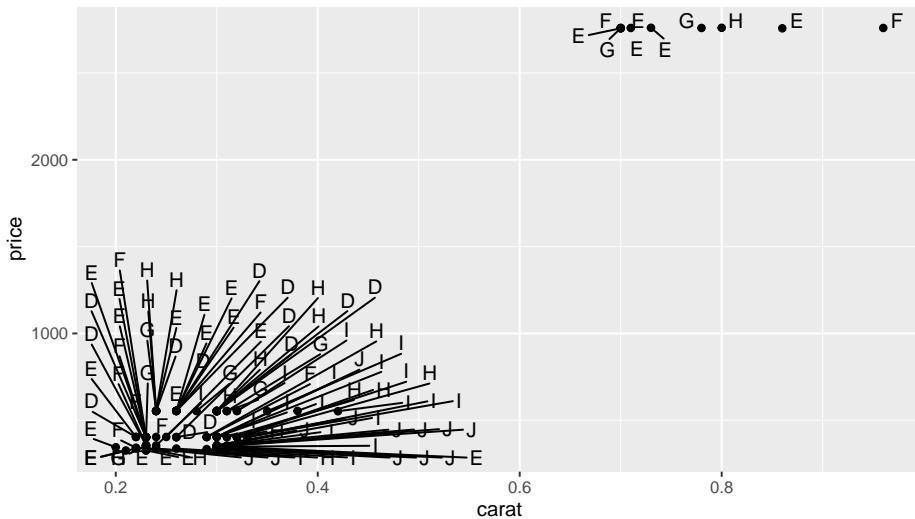
Иногда аннотации налекают друг на друга:

```
library(ggrepel)
diamonds %>%
  slice(1:100) %>%
  ggplot(aes(carat, price, label = color)) +
  geom_text_repel()
```

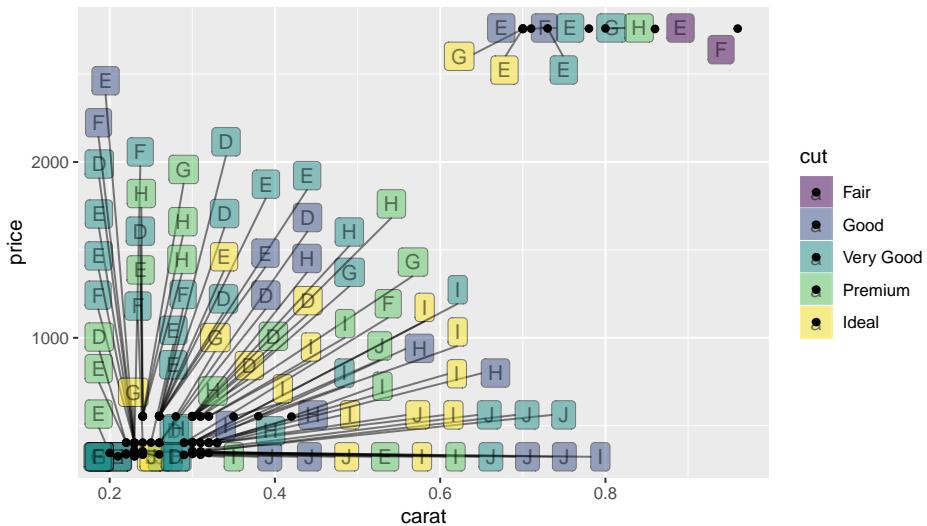


```
diamonds %>%
  slice(1:100) %>%
```

```
ggplot(aes(carat, price, label = color))+  
  geom_text_repel() +  
  geom_point()
```

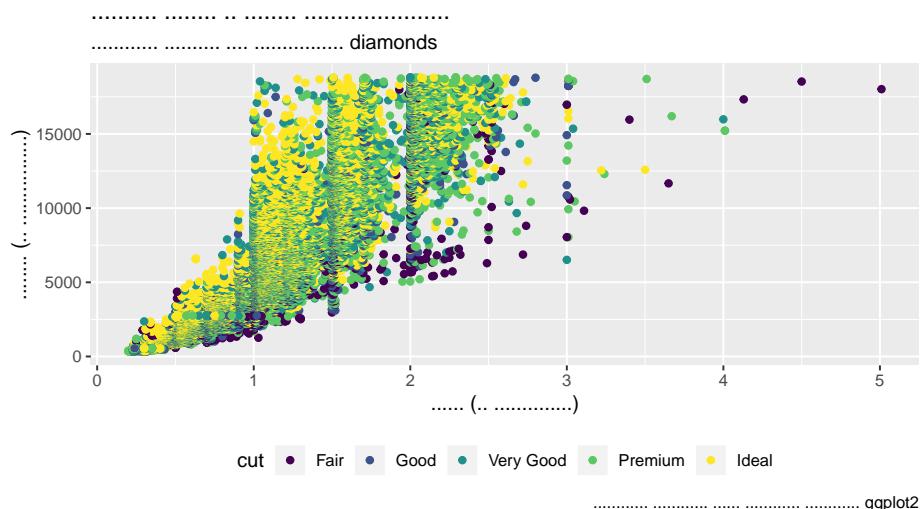


```
diamonds %>%  
  slice(1:100) %>%  
  ggplot(aes(carat, price, label = color, fill = cut)) + # fill  
  geom_label_repel(alpha = 0.5) + # alpha  
  geom_point()
```

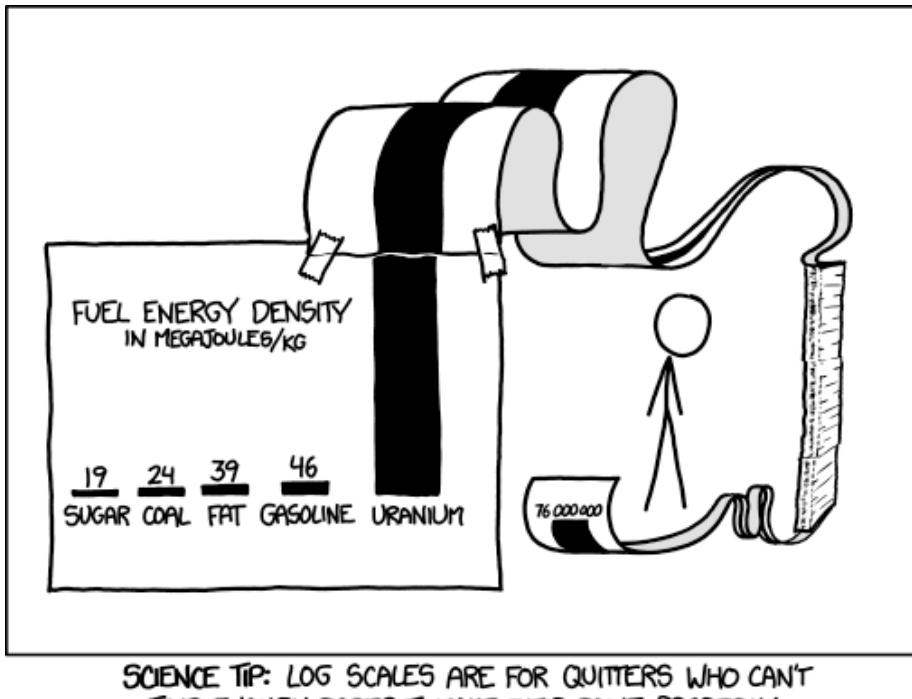


4.2.4 Оформление

```
diamonds %>%
  ggplot(aes(carat, price, color = cut)) +
  geom_point() +
  labs(x = " ( )",
       y = " ( )",
       title = "",
       subtitle = " diamonds",
       caption = " ggplot2") +
  theme(legend.position = "bottom") # theme()
```



4.2.5 Логарифмические шкалы

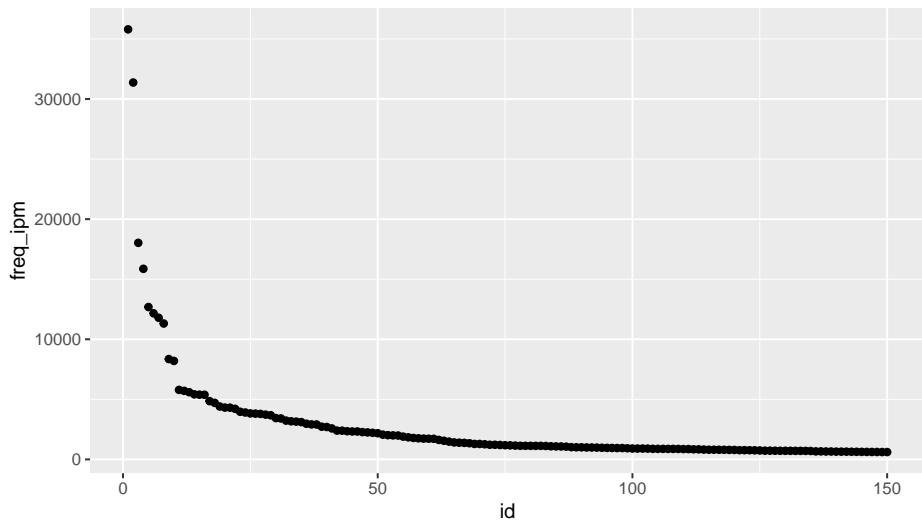


Рассмотрим словарь [Ляшевской, Шарова 2011]

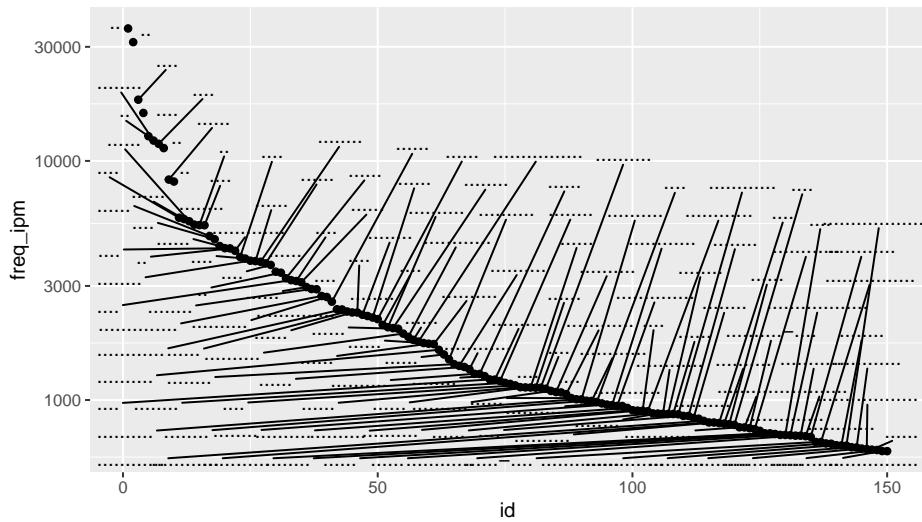
```
freqdict <- read_tsv("https://github.com/agricolamz/DS_for_DH/raw/master/data/freq_dict_2011.csv")

## Parsed with column specification:
## cols(
##   lemma = col_character(),
##   pos = col_character(),
##   freq_ipm = col_double()
## )

freqdict %>%
  arrange(desc(freq_ipm)) %>%
  mutate(id = 1:n()) %>%
  slice(1:150) %>%
  ggplot(aes(id, freq_ipm))+
  geom_point()
```



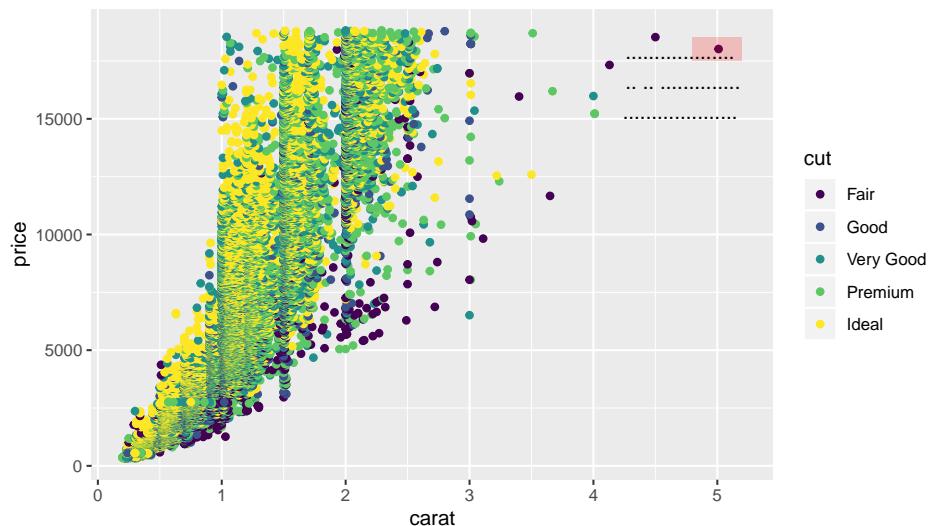
```
freqdict %>%
  arrange(desc(freq_ipm)) %>%
  mutate(id = 1:n()) %>%
  slice(1:150) %>%
  ggplot(aes(id, freq_ipm, label = lemma)) +
  geom_point() +
  geom_text_repel() +
  scale_y_log10()
```



4.2.6 `annotate()`

Функция `annotate` добавляет геом к графику.

```
diamonds %>%
  ggplot(aes(carat, price, color = cut)) +
  geom_point() +
  annotate(geom = "rect", xmin = 4.8, xmax = 5.2,
          ymin = 17500, ymax = 18500, fill = "red", alpha = 0.2) +
  annotate(geom = "text", x = 4.7, y = 16600,
          label = "... \n ...")
```



4.3 Столбчатые диаграммы (barplots)

Одна и та же информация может быть представлена в агрегированном и не агрегированном варианте:

```
misspelling <- read_csv("https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/data/misspelling.csv")
```

```
## Parsed with column specification:
## cols(
##   correct = col_character(),
##   spelling = col_character(),
##   count = col_double()
## )
```

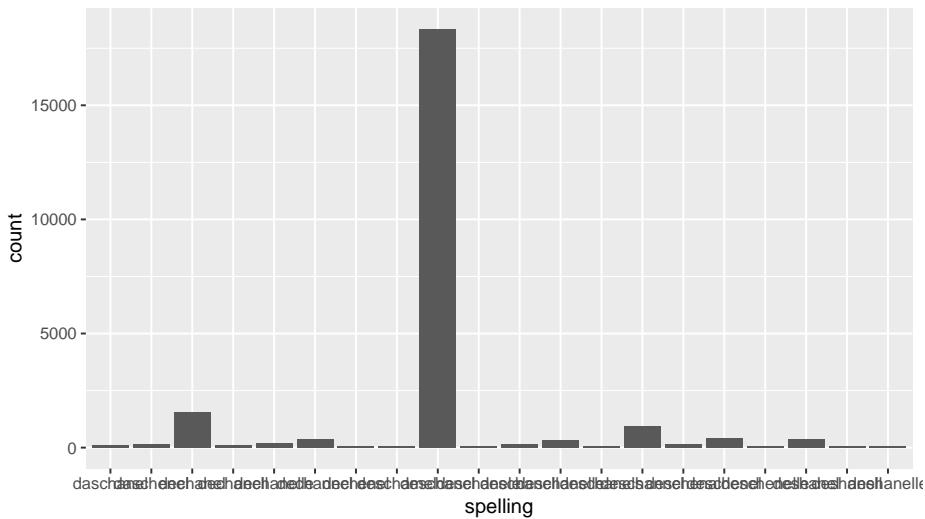
```
misspelling
```

```
## # A tibble: 15,477 x 3
##   correct    spelling   count
##   <chr>      <chr>     <dbl>
## 1 deschanel deschanel  18338
## 2 deschanel dechanel   1550
## 3 deschanel deschannel  934
## 4 deschanel deschenel   404
## 5 deschanel deshanel    364
## 6 deschanel dechannel   359
## 7 deschanel deschanelle  316
## 8 deschanel dechanelle  192
## 9 deschanel deschanell   174
## 10 deschanel deschenal   165
## # ... with 15,467 more rows
```

- переменные `spelling` **аггрегированы**: для каждого значения представлено значение в столбце `count`, которое обозначает количество каждого из написаний
- переменные `correct` **неаггрегированы**: в этом столбце она повторяется, для того, чтобы сделать вывод, нужно отдельно посчитать количество вариантов

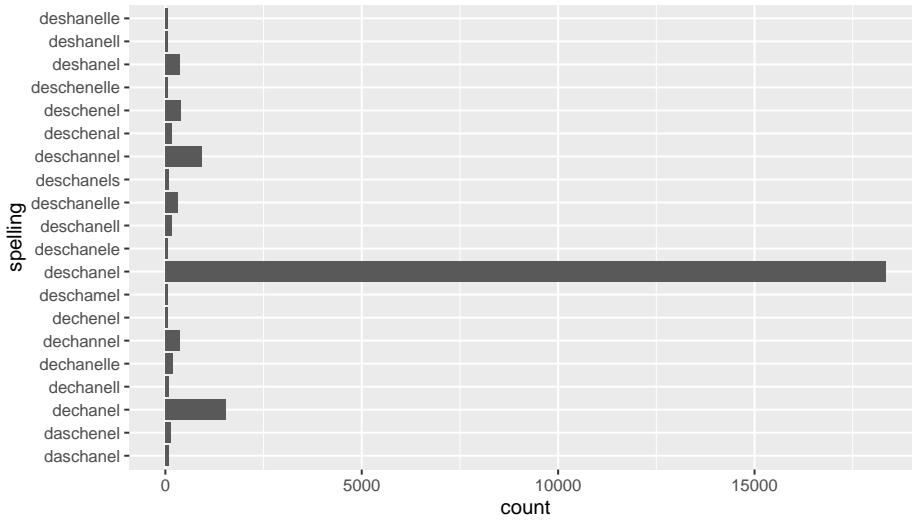
Для агрегированных данных используется `geom_col()`

```
misspelling %>%
  slice(1:20) %>%
  ggplot(aes(spelling, count)) +
  geom_col()
```

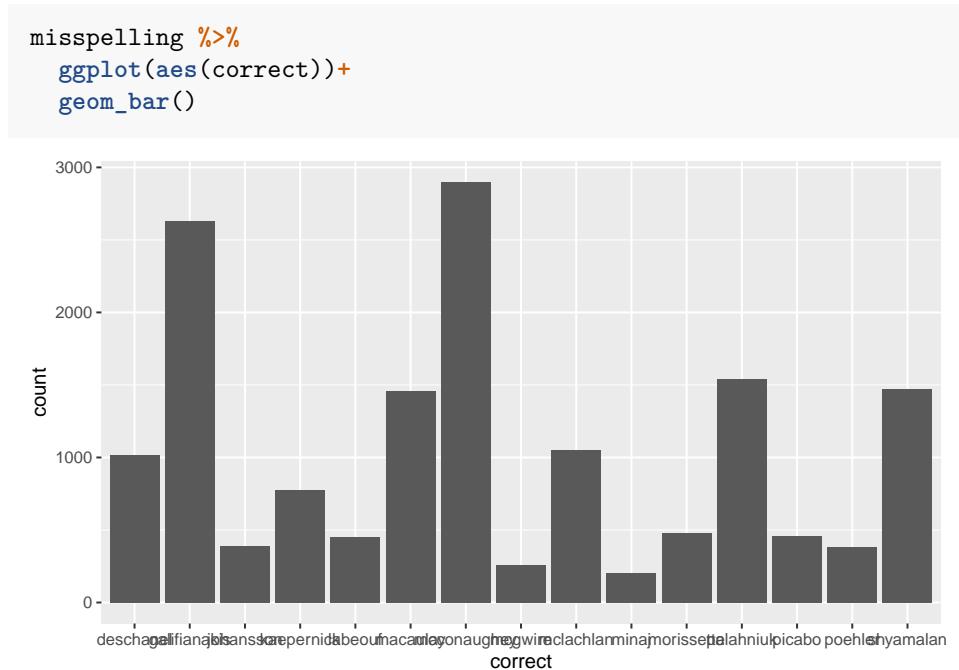


Перевернем оси:

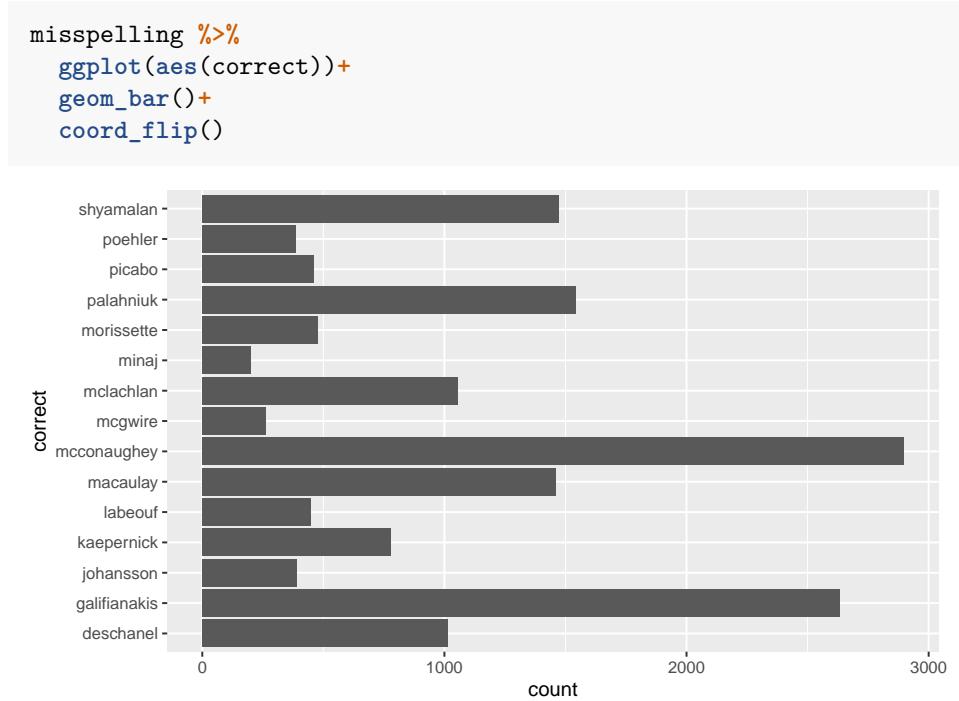
```
misspelling %>%
  slice(1:20) %>%
  ggplot(aes(spelling, count)) +
  geom_col() +
  coord_flip()
```



Для агрегированных данных используется `geom_bar()`



Перевернем оси:



4.4 Факторы

Как можно заметить по предыдущему разделу, переменные на графике упорядочены по алфавиту. Чтобы это исправить нужно обсудить факторы:

```
my_factor <- factor(misspelling$correct)
head(my_factor)

## [1] deschanel deschanel deschanel deschanel deschanel
## 15 Levels: deschanel galifianakis johansson kaepernick ... shyamalan

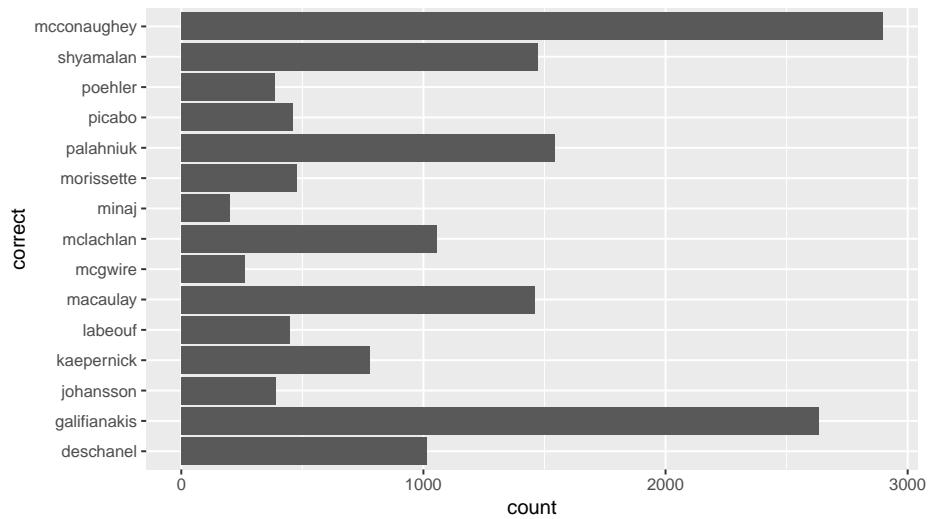
levels(my_factor)

##  [1] "deschanel"      "galifianakis"   "johansson"     "kaepernick"
##  [5] "labeouf"        "macaulay"       "mcconaughey"   "mcgwire"
##  [9] "mclachlan"      "minaj"         "morissette"    "palahniuk"
## [13] "picabo"         "poehler"       "shyamalan"

levels(my_factor) <- rev(levels(my_factor))
head(my_factor)

## [1] shyamalan shyamalan shyamalan shyamalan shyamalan
## 15 Levels: shyamalan poehler picabo palahniuk morissette ... deschanel

misspelling %>%
  mutate(correct = factor(correct, levels = c("deschanel",
                                              "galifianakis",
                                              "johansson",
                                              "kaepernick",
                                              "labeouf",
                                              "macaulay",
                                              "mcgwire",
                                              "mclachlan",
                                              "minaj",
                                              "morissette",
                                              "palahniuk",
                                              "picabo",
                                              "poehler",
                                              "shyamalan",
                                              "mcconaughey")))) %>%
  ggplot(aes(correct))+
  geom_bar()+
  coord_flip()
```



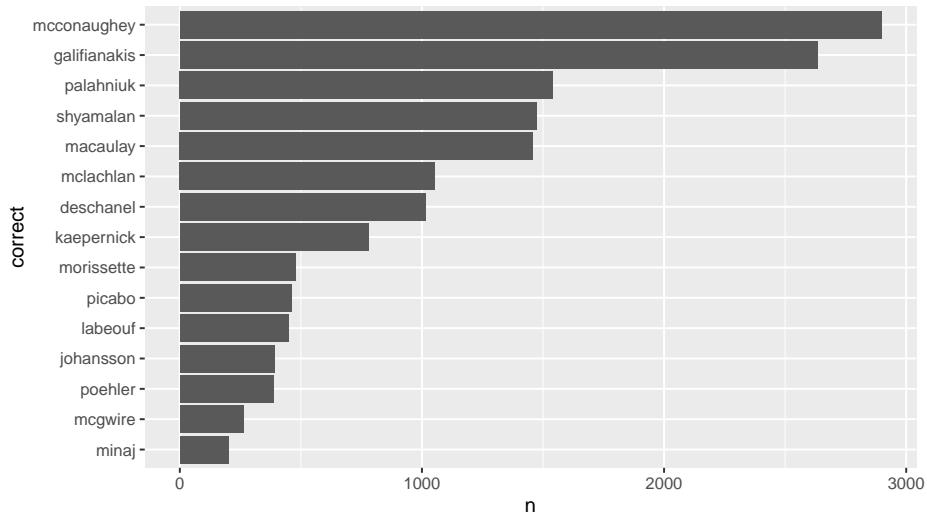
Иногда полезной бывает функция `reorder()`:

```
misspelling %>%
  count(correct)
```

```
## # A tibble: 15 x 2
##   correct      n
##   <chr>     <int>
## 1 deschanel    1015
## 2 galifianakis  2633
## 3 johansson     392
## 4 kaepernick     779
## 5 labeouf       449
## 6 macaulay      1458
## 7 mcconaughey    2897
## 8 mcgwire        262
## 9 mclachlan     1054
## 10 minaj         200
## 11 morissette    478
## 12 palahniuk    1541
## 13 picabo        460
## 14 poehler       386
## 15 shyamalan     1473
```

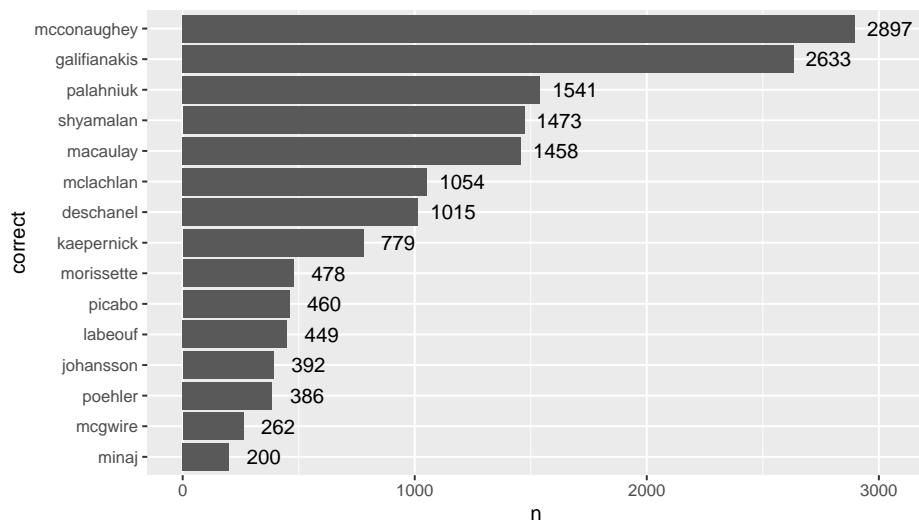
```
misspelling %>%
  count(correct) %>%
  mutate(correct = reorder(correct, n)) %>%
  ggplot(aes(correct, n)) +
```

```
geom_col()+
coord_flip()
```



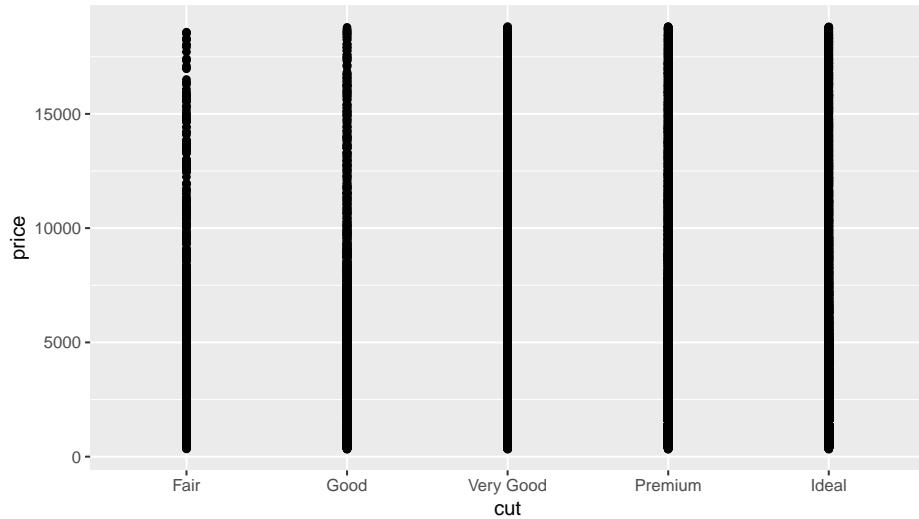
Можно совмещать разные geom_...:

```
misspelling %>%
count(correct) %>%
mutate(correct = reorder(correct, n)) %>%
ggplot(aes(correct, n, label = n))+
geom_col()+
geom_text(nudge_y = 150)+
coord_flip()
```



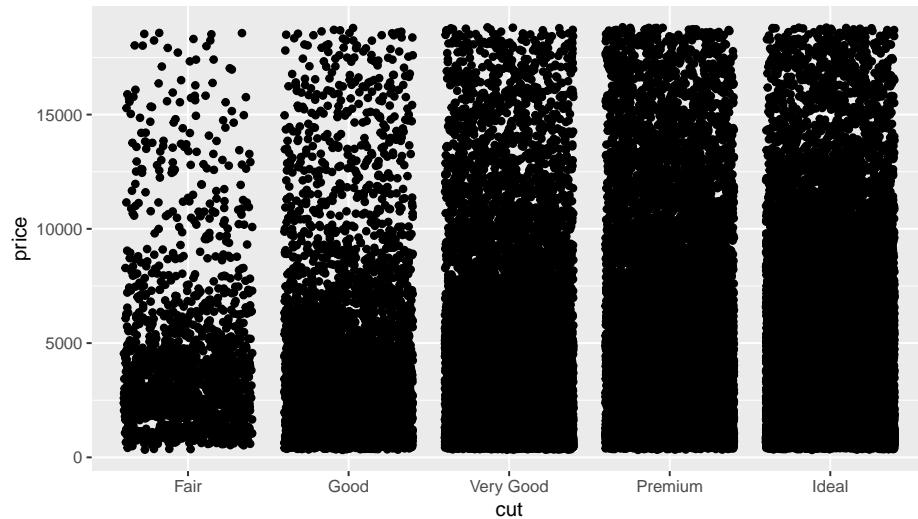
4.5 Точки, джиттер (jitter), ящики с усами (boxplot), (violinplot)

```
diamonds %>%
  ggplot(aes(cut, price)) +
  geom_point()
```

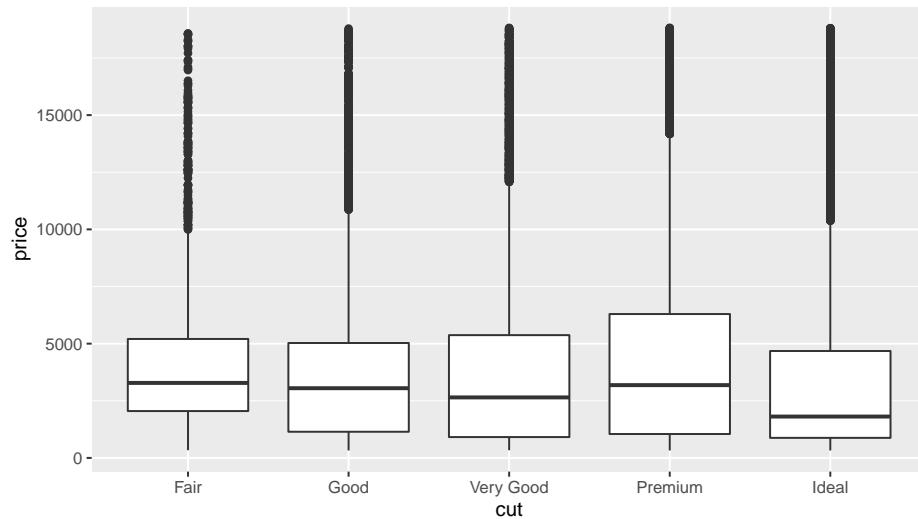


4.5. ТОЧКИ, ДЖИТТЕР (JITTER), ЯЩИКИ С УСАМИ (BOXPLOT), (VIOLINPLOT) 101

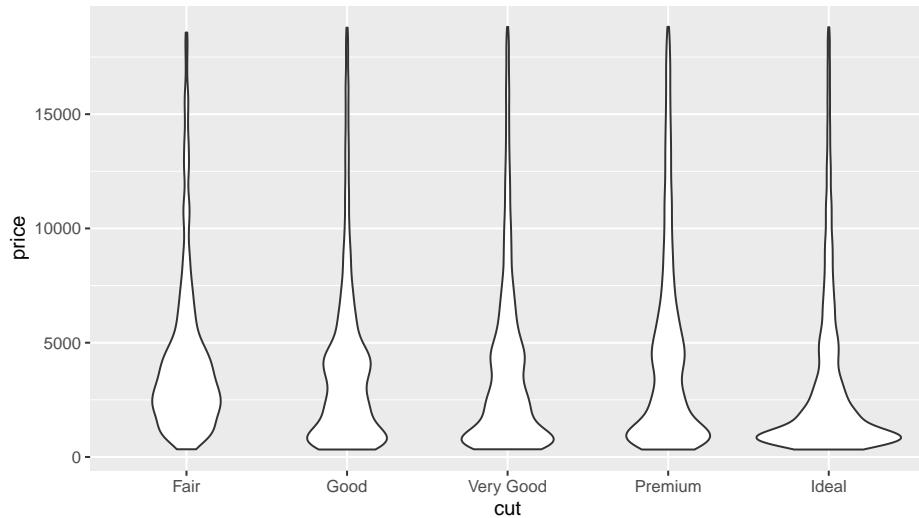
```
diamonds %>%
  ggplot(aes(cut, price)) +
  geom_jitter()
```



```
diamonds %>%
  ggplot(aes(cut, price)) +
  geom_boxplot()
```



```
diamonds %>%
  ggplot(aes(cut, price)) +
  geom_violin()
```



4.6 Гистограммы

4.7 Функции плотности

4.8 Фасетизация

4.9 Визуализация комбинаций признаков

4.9.1 Потоковая Диаграмма (Sankey diagram)

Один из способов визуализации отношений между признаками называется потоковая диаграмма³.

```
library("ggforce")
zhadina <- read_csv("https://raw.githubusercontent.com/agricolamz/DS_for_DH/master/data.csv")
```

```
## Parsed with column specification:
## cols(
##   word_1 = col_character(),
##   word_2 = col_character(),
```

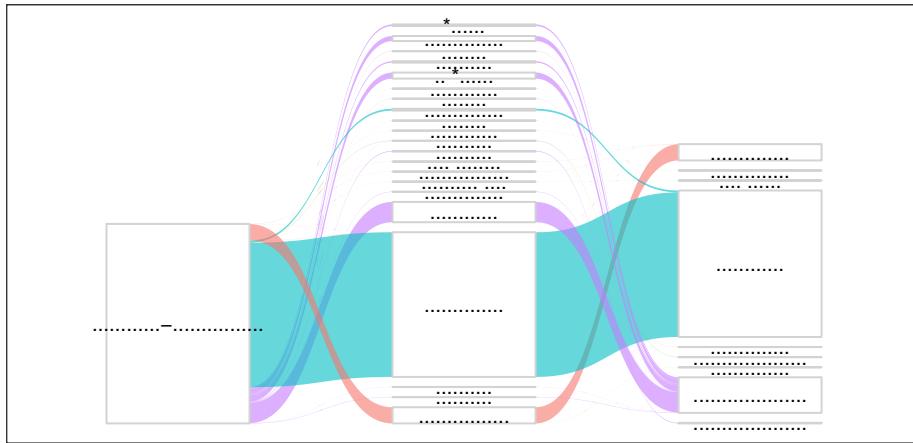
³https://en.wikipedia.org/wiki/Sankey_diagram

```

##   word_3 = col_character(),
##   type = col_character(),
##   n = col_double()
## )

zhadina %>%
  gather_set_data(1:3) %>%
  ggplot(aes(x, id = id, split = y, value = n)) +
  geom_parallel_sets(aes(fill = type), alpha = 0.6, axis.width = 0.5) +
  geom_parallel_sets_axes(axis.width = 0.5, color = "lightgrey", fill = "white") +
  geom_parallel_sets_labels(angle = 0) +
  theme_no_axes() +
  theme(legend.position = "bottom")

```



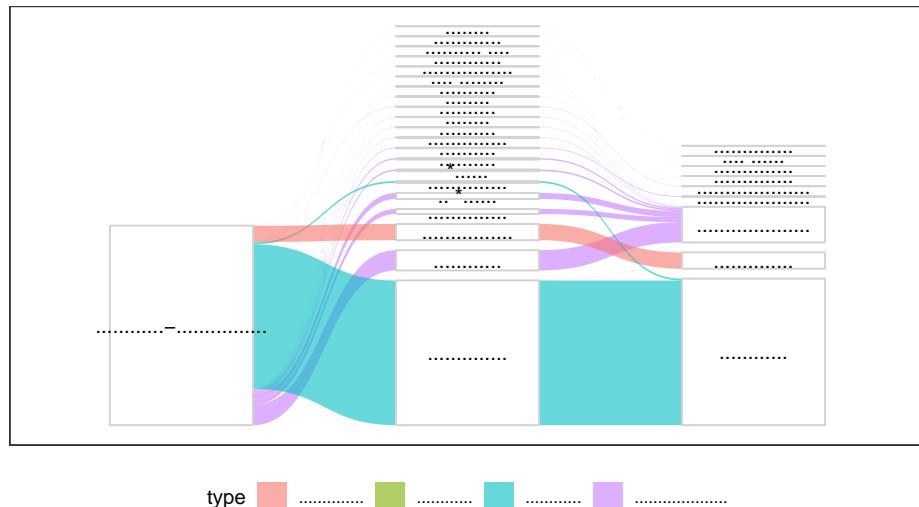
type

А как поменять порядок? Снова факторы.

```

zhadina %>%
  gather_set_data(1:3) %>%
  mutate(y = reorder(y, n)) %>%
  ggplot(aes(x, id = id, split = y, value = n)) +
  geom_parallel_sets(aes(fill = type), alpha = 0.6, axis.width = 0.5) +
  geom_parallel_sets_axes(axis.width = 0.5, color = "lightgrey", fill = "white") +
  geom_parallel_sets_labels(angle = 0) +
  theme_no_axes() +
  theme(legend.position = "bottom")

```



Можно донастроить, задав собственный порядок в аргументе `levels` функции `factor()`.

4.9.2 UpSet Plot

Если диаграмма Sankey визуализирует попарные отношения между переменными, то график UpSet потенциально может визуализировать все возможные комбинации и является хорошей альтернативой диаграмме Вена, с большим количеством переменных (см. эту статью Лауры Эллис⁴).

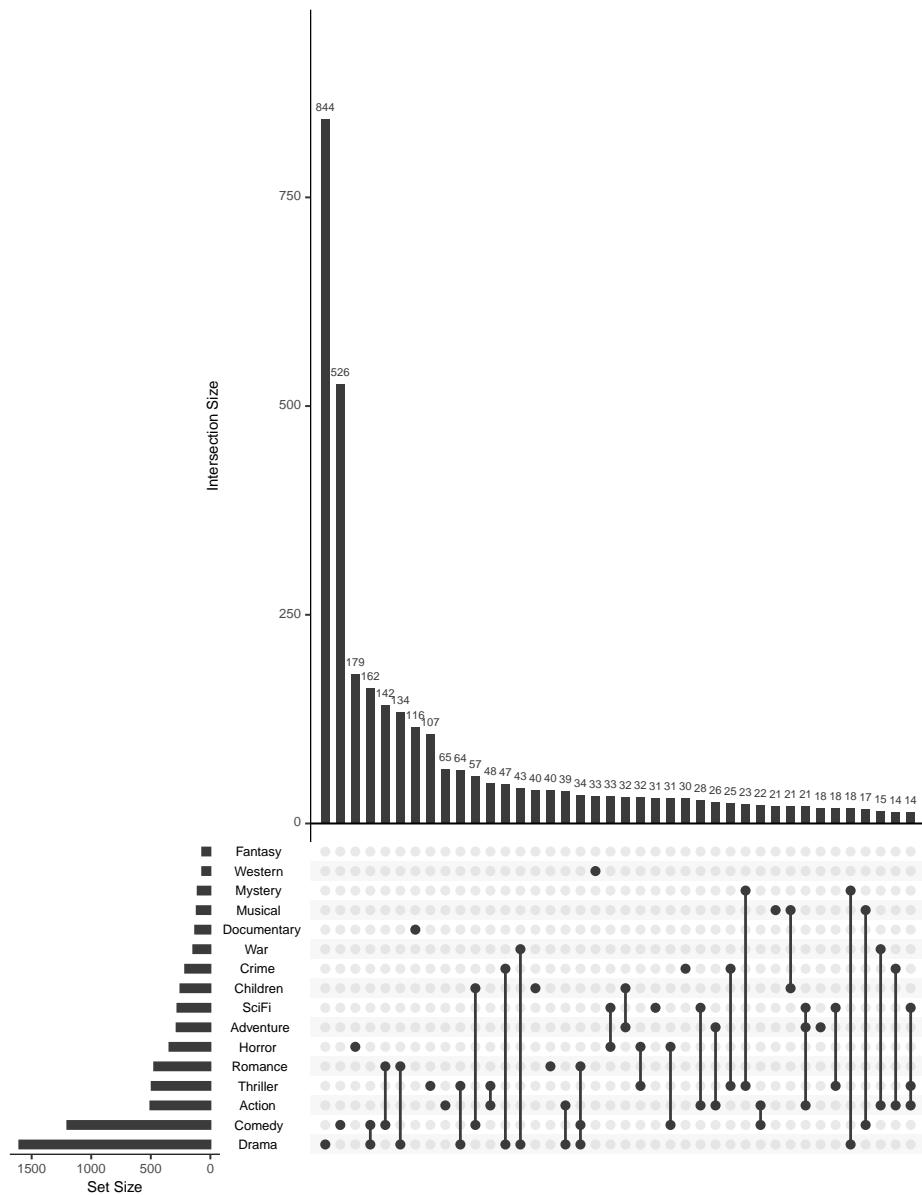
```
library(UpSetR)
movies <- read.csv( system.file("extdata", "movies.csv", package = "UpSetR"), header=TRUE)
str(movies)

## 'data.frame': 3883 obs. of 21 variables:
## $ Name      : Factor w/ 3883 levels "...And Justice for All (1979)",...
## $ ReleaseDate: int  1995 1995 1995 1995 1995 1995 1995 1995 1995 ...
## $ Action     : int  0 0 0 0 0 1 0 0 1 1 ...
## $ Adventure  : int  0 1 0 0 0 0 0 1 0 1 ...
## $ Children   : int  1 1 0 0 0 0 0 1 0 0 ...
## $ Comedy     : int  1 0 1 1 1 0 1 0 0 0 ...
## $ Crime      : int  0 0 0 0 0 1 0 0 0 0 ...
## $ Documentary: int  0 0 0 0 0 0 0 0 0 0 ...
## $ Drama      : int  0 0 0 1 0 0 0 0 0 0 ...
## $ Fantasy    : int  0 1 0 0 0 0 0 0 0 0 ...
## $ Noir       : int  0 0 0 0 0 0 0 0 0 0 ...
## $ Horror     : int  0 0 0 0 0 0 0 0 0 0 ...
```

⁴<https://www.littlemissdata.com/blog/set-analysis>

```
## $ Musical    : int  0 0 0 0 0 0 0 0 0 0 0 0 ...
## $ Mystery    : int  0 0 0 0 0 0 0 0 0 0 0 0 ...
## $ Romance    : int  0 0 1 0 0 0 1 0 0 0 0 0 ...
## $ SciFi      : int  0 0 0 0 0 0 0 0 0 0 0 0 ...
## $ Thriller   : int  0 0 0 0 0 1 0 0 0 1 0 0 ...
## $ War         : int  0 0 0 0 0 0 0 0 0 0 0 0 ...
## $ Western    : int  0 0 0 0 0 0 0 0 0 0 0 0 ...
## $ AvgRating  : num  4.15 3.2 3.02 2.73 3.01 3.88 3.41 3.01 2.66 3.54 ...
## $ Watches    : int  2077 701 478 170 296 940 458 68 102 888 ...
```

```
upset(movies[,3:19], nsets = 16, order.by = "freq")
```



Глава 5

Условия и работа со списками

Глава 6

Представление данных: rmarkdown, github, shiny

Глава 7

Работа со строками

Глава 8

Работа с текстами: tidytext, udpipe

Глава 9

Сбор данных из интернета: rvest

Глава 10

Нестандартные данные: время, OCR, карты

Глава 11

Задания

11.1 Вектор

- Посчитайте логарифм от 8912162342 по основанию 6
- ```
[1] 12.7867
```
- Теперь натуральный логарифм 10 и умножьте его на 5
- ```
## [1] 11.51293
```
- 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
```
- 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
[24] 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```
- Создайте вектор 2, 4, 6, ..., 18, 20
- ```
## [1] 2 4 6 8 10 12 14 16 18 20
```
- 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 6 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8
```
- ```
## [36] 8 9 9 9 9 9 9 9 9
```
- Сделайте вектор vec, в котором соедините 3, а также значения " " и " ".

```
## [1] "3"      " "      " "      "
  • Вычесть TRUE из 10

## [1] 9
  • Соедините значение 10 и TRUE в вектор vec

## [1] 10  1
  • Соедините вектор vec и значение "r":

## [1] "10" "1"  "r"
  • Соедините значения 10, TRUE, "r" в вектор.

## [1] "10"  "TRUE" "r"
```

11.2 Вектор. Операции с векторами

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
```

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
```

11.3 Вектор. Индексирование

Создайте вектор `vec1`:

```
vec1 <- c(3, 5, 2, 1, 8, 4, 9, 10, 3, 15, 1, 11)
```

- Найдите второй элемент вектора `vec1`:
- ```
[1] 5
```
- Найдите последний элемент вектора `vec1`
- ```
## [1] 11
```
- Найдите все значения вектора `vec1`, которые больше 4
- ```
[1] 5 8 9 10 15 11
```
- Найдите все значения вектора `vec1`, которые больше 4, но меньше 10
- ```
## [1] 5 8 9
```
- Возведите в квадрат каждое значение вектора `vec1`
- ```
[1] 9 25 4 1 64 16 81 100 9 225 1 121
```
- Возведите в квадрат каждое нечетное значение вектора и извлеките корень каждого четного значения `vec1`
- ```
## [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 `vec2`, в котором будут значения все значения `vec1`, которые меньше 10 будут заменены на `NA`.
- ```
[1] NA NA NA NA NA NA 10 NA 15 NA 11
```
- Посчитайте сумму `vec2` с помощью функции `sum()`. Ответ `NA` не считается!
- ```
## [1] 36
```
- 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!
- ```
integer(0)
```

### 11.4 Списки

Дан список `list_1`:

```
list_1 = list(numbers = 1:5, letters = letters, logic = TRUE)
list_1
```

```
$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"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
##
$logic
[1] TRUE
```

- Найдите первый элемент списка. Ответ должен быть списком.

```
$numbers
[1] 1 2 3 4 5
```

- Теперь найдите содержание первого элемента списка двумя разными способами. Ответ должен быть вектором.

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

Теперь возьмите первый элемент содержания первого элемента списка. Ответ должен быть вектором.

```
[1] 1
```

Создайте список `list_2`, содержащий в себе два списка `list_1` с именами `pupa` и `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"
[18] "r" "s" "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"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
##
$lupa$logic
[1] TRUE
```

Извлеките первый элемент списка, из него - второй полэлемент, а из него - третье значение

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

## 11.5 Матрицы

- Создайте матрицу 4x4, состоящую из единиц. Назовите ее M

```
[,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 M (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 M

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

- Сравните (==) вторую колонку и вторую строчку матрицы M

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

- Создайте таблицу умножения (9x9) в виде матрицы. Сохраните ее в переменную 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
```

- Из матрицы `tab` выделите подматрицу, включающую в себя только строчки 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 (`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 `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
```

## 11.6 Датафрейм

- Кто является 274ым персонажем в got датафрейме? Из какого он дома?

```
Name Allegiances
274 Gendry None
```

- Найдите имена всех персонажей из дома (Allegiances) "Tyrell" и "House Tyrell".

```
[1] "Alerie Hightower" "Alla Tyrell" "Alyn Ambrose"
[4] "Arryk (Guard)" "Arwyn Oakheart" "Bayard Norcross"
[7] "Blue Bard" "Butterbumps" "Elinor Tyrell"
[10] "Erryk (Guard)" "Garlan Tyrell" "Hobber Redwyne"
[13] "Horas Redwyne" "Janna Tyrell" "Kerwin"
[16] "Leo Tyrell" "Leonette Fossoway" "Loras Tyrell"
[19] "Mace Tyrell" "Margaery Tyrell" "Megga Tyrell"
[22] "Meredyth Crane" "Olenna Redwyne" "Paxter Redwyne"
[25] "Randyll Tarly" "Talbert Serry"
```

- Создайте новый датафрейм greyjoy\_women, который будет включать в себя только женщин Грейджоев ("Greyjoy", "House Greyjoy")

```
Name Allegiances Death.Year Book.of.Death
58 Asha Greyjoy House Greyjoy NA NA
248 Falia Flowers Greyjoy NA NA
313 Gwin Goodbrother Greyjoy NA NA
319 Gysella Goodbrother Greyjoy NA NA
806 Three-Tooth Greyjoy NA NA
Death.Chapter Book.Intro.Chapter Gender Nobility GoT CoK SoS FfC DwD
58 NA 11 0 1 0 1 0 1 1
248 NA 29 0 0 0 0 0 1 0
313 NA 1 0 1 0 0 0 1 0
319 NA 1 0 1 0 0 0 1 0
806 NA 11 0 0 0 0 0 1 0
Is.Alive
58 TRUE
248 TRUE
313 TRUE
319 TRUE
806 TRUE
```

- Сколько всего женских персонажей в книгах “Песни льда и пламени”?

```
[1] 157
```

- Сколько всего женских персонажей дворянского происхождения в книгах “Песни льда и пламени”?

```
[1] 84
```

- Посчитайте процентную (!) долю знати от общего числа персонажей (*Nobility*) в *Night's Watch*.

```
[1] 9.482759
```

- Посчитайте процентную (!) долю знати от общего числа персонажей (*Nobility*) у *Lannister*.

```
[1] 71.60494
```

- Какая из книг цикла самая кровавая? Для ответа на этот вопрос подсчитайте таблицу частот для колонки *got\$Book.of.Death*:

Это можно сделать с помощью функции *table()*, но в дальнейшем Вы узнаете и другие способы - подобная задача возникает достаточно часто.

```

1 2 3 4 5
49 73 97 27 61
```

## Глава 12

# Решения\_заданий

### 12.1 Вектор

- Посчитайте логарифм от 8912162342 по основанию 6

```
log(8912162342, 6)
```

```
[1] 12.7867
```

- Теперь натуральный логарифм 10 и умножьте его на 5

```
log(10)*5
```

```
[1] 11.51293
```

- 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
```

- 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
[24] 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

- Создайте вектор 2, 4, 6, ..., 18, 20

```
seq(2,20, 2)
```

```
[1] 2 4 6 8 10 12 14 16 18 20
```

- Создайте вектор из одной единицы, двух двоек, трех троек, ..., девяти десяток

```
rep(1:9, 1:9)
```

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

```
[36] 8 9
```

- Сделайте вектор vec, в котором соедините 3, а также значения " " и " ".

```
vec <- c(3, " ", " ")
```

```
vec
```

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

- Вычесть TRUE из 10

```
10 - TRUE
```

```
[1] 9
```

- Соедините значение 10 и TRUE в вектор vec

```
vec <- c(10, TRUE)
```

```
vec
```

```
[1] 10 1
```

- Соедините вектор vec и значение "r":

```
c(vec, "r")
```

```
[1] "10" "1" "r"
· Соедините значения 10, TRUE, "r" в вектор.
```

```
c(10, TRUE, "r")
```

```
[1] "10" "TRUE" "r"
```

## 12.2 Вектор. Операции с векторами

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
```

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
```

### 12.3 Вектор. Индексирование

Создайте вектор vec1:

```
vec1 <- c(3, 5, 2, 1, 8, 4, 9, 10, 3, 15, 1, 11)
```

- Найдите второй элемент вектора vec1:

```
vec1[2]
```

```
[1] 5
```

- Найдите последний элемент вектора vec1

```
vec1[length(vec1)]
```

```
[1] 11
```

- Найдите все значения вектора vec1, которые больше 4

```
vec1[vec1>4]
```

```
[1] 5 8 9 10 15 11
```

- Найдите все значения вектора vec1, которые больше 4, но меньше 10

```
vec1[vec1>4 & vec1<10]
```

```
[1] 5 8 9
```

- Возведите в квадрат каждое значение вектора vec1

```
vec1^2
```

```
[1] 9 25 4 1 64 16 81 100 9 225 1 121
```

- Возведите в квадрат каждое нечетное значение вектора и извлеките корень каждого четного значения vec1

```
vec1 ^ 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 vec2, в котором будут значения все значения vec1, которые меньше 10 будут заменены на NA.

```
vec2 <- vec1
vec2[vec2<10] <- NA
vec2
```

```
[1] NA NA NA NA NA NA 10 NA 15 NA 11
```

- Посчитайте сумму vec2 с помощью функции sum(). Ответ NA не считается!

```
sum(vec2, na.rm = TRUE)
```

```
[1] 36
```

- 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(F,T)]
```

```
integer(0)
```

```
#(1:10)*2
```

## 12.4 Списки

Дан список `list_1`:

```
list_1 = list(numbers = 1:5, letters = letters, logic = TRUE)
list_1

$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"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
##
$logic
[1] TRUE
```

- Найдите первый элемент списка. Ответ должен быть списком.

```
list_1[1]
```

```
$numbers
[1] 1 2 3 4 5
```

- Теперь найдите содержание первого элемента списка двумя разными способами. Ответ должен быть вектором.

```
list_1[[1]]
```

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

```
list_1$numbers
```

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

Теперь возьмите первый элемент содержания первого элемента списка. Ответ должен быть вектором.

```
list_1[[1]][1]
```

```
[1] 1
```

Создайте список `list_2`, содержащий в себе два списка `list_1` с именами `pira` и `lupa`.

```
list_2 = list(pupa = list_1, lupa = list_1)
list_2

$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"
[18] "r" "s" "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"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
##
$lupa$logic
[1] TRUE
```

Извлеките первый элемент списка, из него - второй полэлемент, а из него - третье значение

```
list_2[[1]][[2]][3]
```

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

## 12.5 Матрицы

- Создайте матрицу 4x4, состоящую из единиц. Назовите ее M

```
M <- matrix(rep(1, 16), ncol = 4)
M

[,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 M (то есть значения на позициях [2,2], [2,3], [3,2] и [3,3]) на число 2.

```
M[2:3, 2:3] <- 2
M
```

```
[,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 M

```
M[,2:3]
```

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

- Сравните (==) вторую колонку и вторую строчку матрицы M

```
M[,2] == M[2,]
```

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

- Создайте таблицу умножения (9x9) в виде матрицы. Сохраните ее в переменную tab:

```
tab <- matrix(rep(1:9, rep(9,9))*rep(1:9), nrow = 9)
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 %% 1:9
```

- Из матрицы `tab` выделите подматрицу, включающую в себя только строчки с 6 по 8 и столбцы с 3 по 7.

```
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 (`tab`) двузначное число и FALSE, если однозначное.

Матрица - это почти вектор. К нему можно обращаться с единственным индексом.

```
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 `tab2`, в которой все значения `tab` меньше 10 заменены на 0.

```
tab2 <- tab
tab2[tab<10] <- 0
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
```

## 12.6 Датафрейм

- Кто является 274ым персонажем в `got` датафрейме? Из какого он дома?

```
got[274, 1:2]
```

```
Name Allegiances
274 Gendry None
```

- Найдите имена всех персонажей из дома (`Allegiances`) "Tyrell" и "House Tyrell".

```
got[got$Allegiances %in% c("Tyrell", "House Tyrell"), "Name"]
```

```
[1] "Alerie Hightower" "Alla Tyrell" "Alyn Ambrose"
[4] "Arryk (Guard)" "Arwyn Oakheart" "Bayard Norcross"
[7] "Blue Bard" "Butterbumps" "Elinor Tyrell"
[10] "Erryk (Guard)" "Garlan Tyrell" "Hobber Redwyne"
[13] "Horas Redwyne" "Janna Tyrell" "Kerwin"
[16] "Leo Tyrell" "Leonette Fossoway" "Loras Tyrell"
[19] "Mace Tyrell" "Margaery Tyrell" "Megga Tyrell"
[22] "Meredyth Crane" "Olenna Redwyne" "Paxter Redwyne"
[25] "Randyll Tarly" "Talbert Serry"
```

- Создайте новый датафрейм `greyjoy_women`, который будет включать в себя только женщин Грейджоев ("Greyjoy", "House Greyjoy")

```
greyjoy_women <- got[got$Allegiances %in% c("Greyjoy", "House Greyjoy") & got$Gender ==
```

|        | Name                | Allegiances   | Death.Year | Book.of.Death |
|--------|---------------------|---------------|------------|---------------|
| ## 58  | Asha Greyjoy        | House Greyjoy | NA         | NA            |
| ## 248 | Falia Flowers       | Greyjoy       | NA         | NA            |
| ## 313 | Gwin Goodbrother    | Greyjoy       | NA         | NA            |
| ## 319 | Gysella Goodbrother | Greyjoy       | NA         | NA            |
| ## 806 | Three-Tooth         | Greyjoy       | NA         | NA            |

```
Death.Chapter Book.Intro.Chapter Gender Nobility GoT CoK SoS FfC DwD
58 NA 11 0 1 0 1 0 1 1
248 NA 29 0 0 0 0 0 1 0
313 NA 1 0 1 0 0 0 1 0
319 NA 1 0 1 0 0 0 1 0
806 NA 11 0 0 0 0 0 1 0
Is.Alive
58 TRUE
248 TRUE
313 TRUE
319 TRUE
806 TRUE
```

- Сколько всего женских персонажей в книгах “Песни льда и пламени”?

```
sum(got$Gender == 0)
```

```
[1] 157
```

- Сколько всего женских персонажей дворянского происхождения в книгах “Песни льда и пламени”?

```
sum((got$Gender == 0) & (got$Nobility == 1))
```

```
[1] 84
```

- Посчитайте процентную (!) долю знать от общего числа персонажей (Nobility) в Night's Watch.

```
mean(got[got$Allegiances == "Night's Watch", "Nobility"])*100
```

```
[1] 9.482759
```

- Посчитайте процентную (!) долю знать от общего числа персонажей (Nobility) у Lannister.

```
mean(got[got$Allegiances == "Lannister", "Nobility"])*100
```

```
[1] 71.60494
```

- Какая из книг цикла самая кровавая? Для ответа на этот вопрос подсчитайте таблицу частот для колонки got\$Book.of.Death:

Это можно сделать с помощью функции `table()`, но в дальнейшем Вы узнаете и другие способы - подобная задача возникает достаточно часто.

```
table(got$Book.of.Death)
```

```

1 2 3 4 5
49 73 97 27 61
```

# Литература

- Adler, J. (2010). *R in a nutshell: A desktop quick reference.* "O'Reilly Media, Inc."
- Baesens, B., Van Lasselaer, V., and Verbeke, W. (2015). *Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection.* John Wiley & Sons.
- Brooks, H. and Cooper, C. L. (2013). *Science for public policy.* Elsevier.
- Hansjörg, N. (2019). *Data Science for Psychologists.* self published.
- Provost, F. and Fawcett, T. (2013). *Data Science for Business: What you need to know about data mining and data-analytic thinking.* O'Reilly Media, Inc.
- R Core Team (2019). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.
- Thomas, N. and Pallett, L. (2019). *Data Science for Immunologists.* CreateSpace Independent Publishing Platform.
- Wickham, H. and Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data.* O'Reilly Media, Inc.