Наука о данных в R для программы Цифровых гуманитарных исследований

Г. А. Мороз, И. С. Поздняков

Оглавление

Глава 1

О курсе

Материалы для курса Наука о данных для магистерской программы Цифровых гуманитарные исследования 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 находиться 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
- вычислительная криминалистика
- цифровые гуманитарные исследования
- датажурналистика
- ...

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

- psychologists (Hansjörg, 2019)
- immunologists (Thomas and Pallett, 2019)
- business (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
- статистическое моделирование данных
- представление полученных результатов
- организация всей работы **воспроизводимым способом**

Большинство этих тем в той или иной мере будет представлено в 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 Windows[1]
  - на Mac[2]
  - на Linux[3], также можно добавить зеркало и установить из командной строки:

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

- RStudio — IDE для R (можно скачать здесь[4])
- 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 используется следующая версия R:

```
## [1] "R version 4.0.2 (2020-06-22)"
```

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

- RStudio cloud[5] — полная функциональность RStudio, пока бесплатная, но скоро это исправят;
- RStudio on rollApp[6] — 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.

[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

2.3 Полезные ссылки

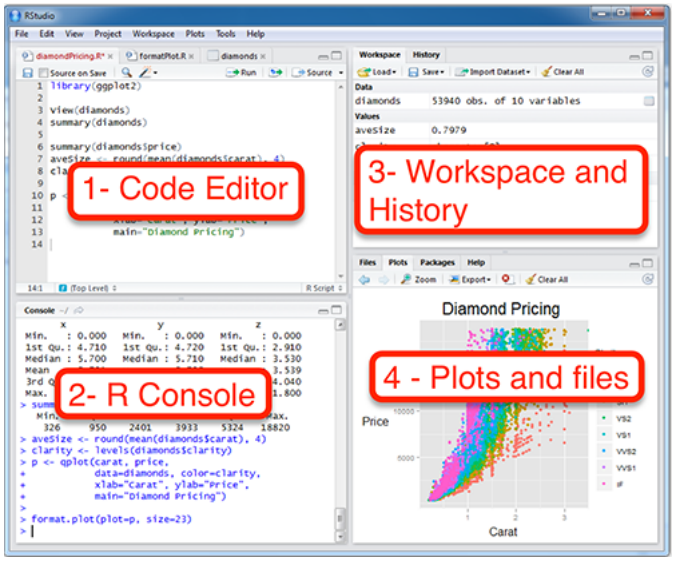
В интернете легко найти документацию и туториалы по самым разным вопросам в R, так что главный залог успеха — грамотно пользоваться поисковиком, и лучше на английском языке.

- книга (Wickham and Grolemund, 2016)[7] является достаточно сильной альтернативой всему курсу
- stackoverflow[8] — сервис, где достаточно быстро отвечают на любые вопросы (не обязательно по R)
- RStudio community[9] — быстро отвечают на вопросы, связанные с R
- русский stackoverflow[10]
- R-bloggers[11] — сайт, где собираются новинки, связанные с R
- чат[12], где можно спрашивать про R на русском (но почитайте правила чата[13], перед тем как спрашивать)
- чат[14] по визуализации данных, чат[15] датажурналистов
- канал про визуализацию[16], дата-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/
[12] https://t.me/rlang_ru
[13] https://github.com/r-lang-group-ru/group-rules/blob/master/README.md
[14] https://t.me/joinchat/CxZg5goGc6rlWGjcvOYrpA
[15] https://t.me/ddjrus
[16] https://t.me/chartomojka
[17] https://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 использовать 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. Подсказку по порядку выполнения операций в R можно получить с помощью следующей команды:

```
?Syntax
```

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

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:

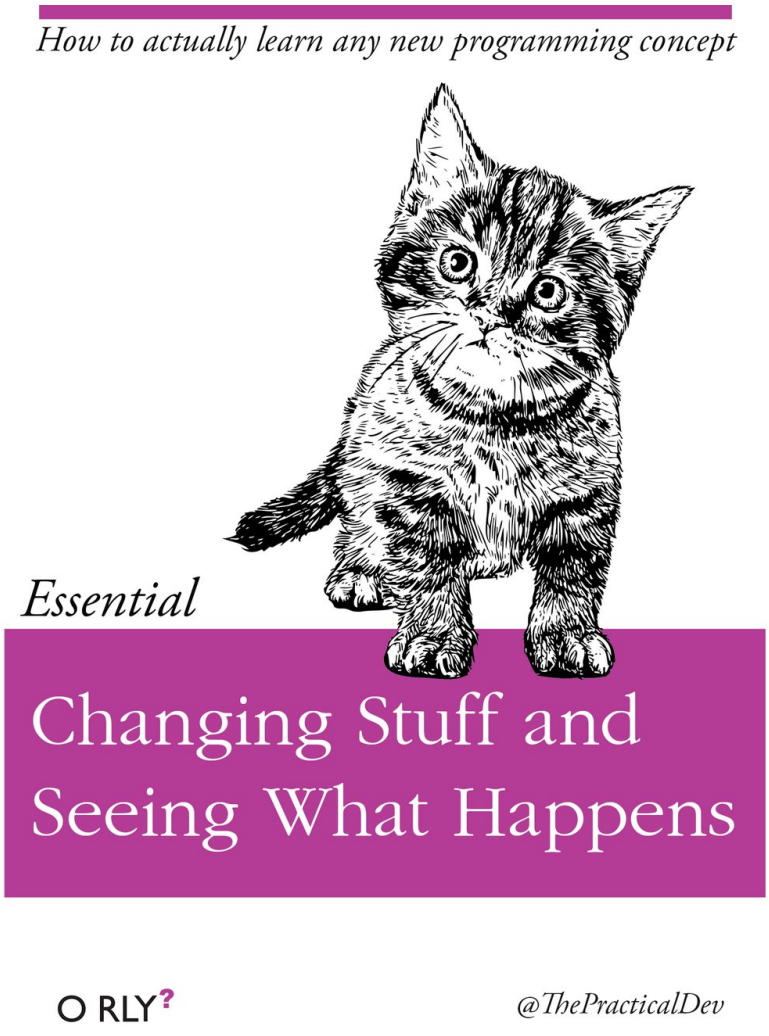

How to actually learn any new programming concept
Essential
Changing Stuff and Seeing What Happens
O RLY?
@ThePracticalDev

Рис. 2.1

```
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 выдают в ответ какие-то другие данные (или же функция записывает файл, рисует график и т.д.).

Данные на входе называются **аргументом** функции, а иногда — **параметром** функции. В обыденной речи часто говорят **инпут** (калька с английского *input*).

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:

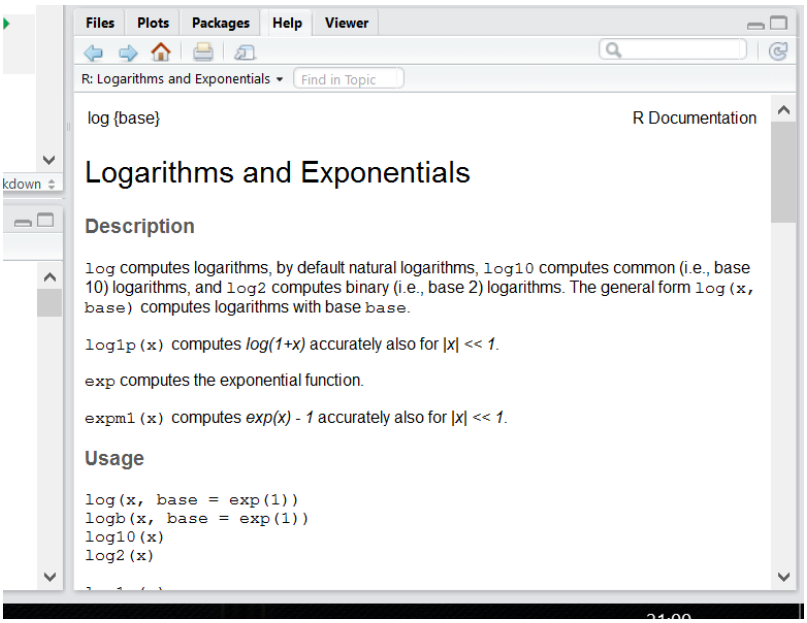


Рис. 2.2

Действительно, у этой функции есть еще аргумент *base =*. По умолчанию он равен числу Эйлера (2.7182818...), т.е. функция считает натуральный логарифм. В большинстве функций R есть какой-то основной инпут — данные в том или ином формате, а есть и дополнительные параметры, которые можно прописывать вручную, если параметры по умолчанию вас 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 результат выполнения одних функций в качестве аргумента для других:

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

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

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

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

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

А еще можно 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 инпут 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 Маке).

```
a <- 2
a
```

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

Справа от `<-` находится значение, которое вы хотите сохранить, или же какое-то выражение, результат которого вы хотите сохранить в эту переменную[18]:

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

```
a <- log(9, 3)
```

Слева от <- находится название будущей переменной. Название переменных может быть самым разным. Есть несколько ограничений для синтаксически валидных имен переменных: они должны включать в себя буквы, цифры, . или _, начинаться на букву (или точку, за которой не будет следовать цифра), не должны совпадать с коротким списком зарезервированных слов[19]. Короче говоря, название не должно включать в себя пробелы и большинство других знаков.

Нельзя: - `new variable` - `_new_variable` - `.1var` - `v-r`

Можно: - `new_variable` - `.new.variable` - `var_2`

Обязательно делайте названия переменных осмысленными! Старайтесь делать при этом их понятными и короткими, это сохранит вам очень много времени, когда вы (или кто-то еще) будете пытаться разобраться в написанном ранее коде. Если название все-таки получается длинным и состоящим из нескольких слов, то лучше всего использовать нижнее подчеркивание в качестве разделителя: `some_variable`[20].

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

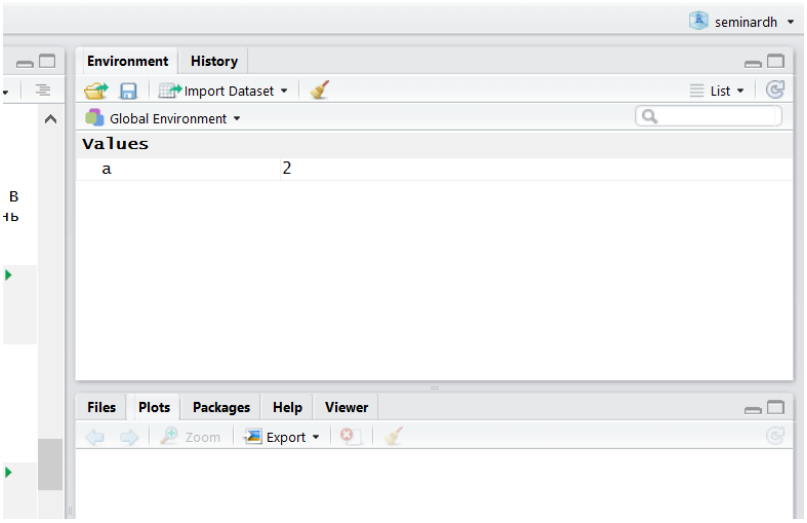


Рис. 2.3

Можно использовать переменные в функциях и просто вычислениях:

[19] `https://stat.ethz.ch/R-manual/R-devel/library/base/html/Reserved.html`

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

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

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

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

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

2.6 Логические операторы

Вы можете сравнивать разные переменные:

```
a == b
```

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

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

```
a = b #      ,          !
a
```

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

```
b
```

```
## [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 *не*равенство:

```
a <- 2
b <- 3

a == b
```

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

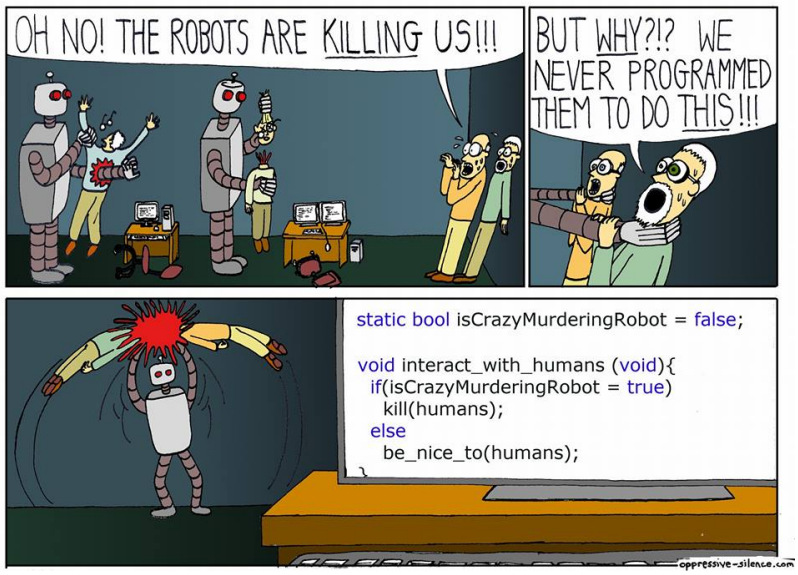


Рис. 2.4

```
a != b
```

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

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

Еще мы можем сравнивать на больше/меньше:

```
a > b
```

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

```
a < b
```

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

```
a >= b
```

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

```
a <= b
```

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

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

2.7 Типы данных

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

```
class(a)
```

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

На самом деле, в R три типа numeric: integer (целые), double (дробные), complex (комплексные числа)[21]. R сам будет конвертировать числа в нужный тип numeric при необходимости, поэтому этим можно не заморачиваться.

Если же все-таки нужно задать конкретный тип числа эксплицитно, то можно воспользоваться функциями `as.integer()`, `as.double()` и `as.complex()`. Кроме того, при создании числа можно поставить в конце L, чтобы обозначить число как integer:

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

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

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

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

Про double есть еще один маленький секрет. Дело в том, что дробные числа хранятся в R как числа с плавающей запятой двойной точности[22]. Дробные числа в компьютере могут быть записаны только с определенной степенью точности, поэтому иногда встречаются вот такие вот ситуации:

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

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

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

[22] `https://ru.wikipedia.org/wiki/_____`

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

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

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

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

1. **character**: строки символов. Они должны выделяться кавычками.

```
s <- '      !'
s
```

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

```
class(s)
```

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

Можно использовать как ", так и ' (что удобно, когда строчка внутри уже содержит какие-то кавычки).

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

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

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:        (do_set)
```

```
TRUE
```

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

```
T <- FALSE
T
```

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

Мы уже встречались с логическими значениями при сравнении двух числовых переменных. Теперь вы можете догадаться, что результаты сравнения, например, числовых или строковых переменных, можно тоже сохранять в переменные!

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

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

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

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

```
t1
```

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

```
!t1
```

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

```
!!t1 #          !
```

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

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

```
t1 & t2
```

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

```
t1 & f1
```

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

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

```
t1 | f1
```

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

```
f1 | f2
```

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

Если кому-то вдруг понадобится другое ИЛИ (строгое ЛИБО) — есть функция `xor()`, принимающая два аргумента.

Итак, мы только что разобрались с самой занудной (хотя и важной) частью - с основными типа данных в R и как с ними работать[23]. Пора переходить к чему-то более интересному и специфическому для R. Вперед к ВЕКТОРАМ!

2.8 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.

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

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

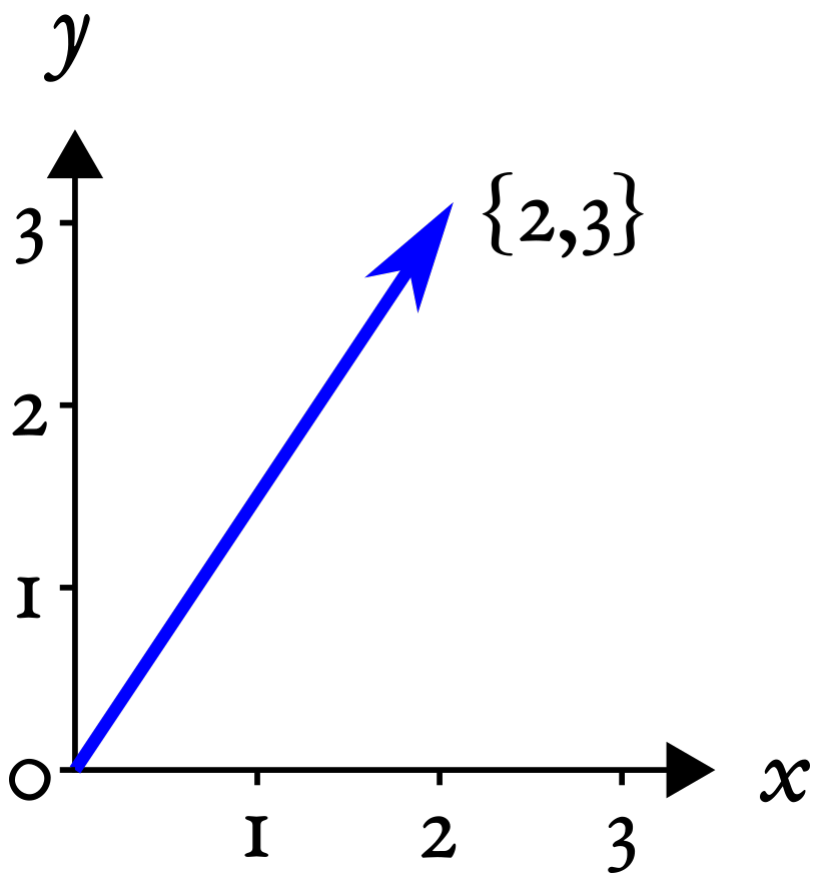


Рис. 2.5

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

На самом деле, мы уже работали с векторами в R, но, возможно, вы об этом даже не догадывались. Дело в том, что в R нет как таковых “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
```

Рис. 2.6

```
c(" ", " ", " ")
```

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

Одна из самых мерзких и раздражающих причин ошибок в коде — это использование из кириллицы вместо c из латиницы. Видите разницу? И я не вижу. А R видит. И об этом сообщает:

```
(3, 4, 5)
```

```
## Error in  (3, 4, 5):                " "
```

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!"
```

2.8.1 Приведение типов

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 пройзойдет 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 оба значения можно было объединить в вектор. То же самое произошло бы в случае операций с векторами:

```
2 + TRUE
```

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

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

Вот более сложный пример:

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

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

У R есть иерархия приведения типов:

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

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

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

```
as.numeric(c(T, F, F))
```

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

```
as.character(as.numeric(c(T, F, F)))
```

```
## [1] "0" "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:                        NA
```

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

> Один из распространенных примеров использования неявного приведения типов — использования функций sum() и mean() для подсчета в логическом векторе количества и доли TRUE соответсвенно. Мы будем много раз пользоваться этим приемом в дальнейшем!

2.8.2 Векторизация

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

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

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

```
n - m
```

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

```
n * m
```

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

```
n / m
```

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

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

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

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

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

```
n %*% m
```

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

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

В принципе, большинство функций в R, которые работают с отдельными значениями, так же хорошо работают и с целыми векторами. Скажем, 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 2.828427
##  [9] 3.000000 3.162278
```

Таких векторизованных функций в R очень много. Многие из них написаны на более низкоуровневых языках программирования (C, C++, FORTRAN), за счет че-

го использование таких функций приводит не только к более элегантному, лаконичному, но и к более быстрому коду.

> Векторизация в R — это очень важная фишка, которая отличает этот язык программирования от многих других. Если вы уже имеете опыт программирования на другом языке, то вам во многих задачах захочется использовать циклы типа `for` и `while` 4.2. Не спешите этого делать! В очень многих случаях циклы можно заменить векторизацией. Тем не менее, векторизация — это не единственный способ избавить от циклов типа `for` и `while` 5.5.1.

2.8.3 Ресайклинг

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

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

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

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

```
n * 2
```

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

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

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

```
## Warning in n + c(3, 4, 5):
##
```

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

Проблема в том, что эти предупреждения могут в неожиданный момент стать причиной ошибок. Поэтому не стоит полагаться на ресайклинг некратных по длине векторов. См. здесь[24]. А вот ресайклинг кратных по длине векторов — это очень удобная штука, которая используется очень часто.

2.8.4 Индексирование векторов

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

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

```
n <- 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 свыклись с этим), то в R Вам придется переучиться обратно. Здесь первый индекс — это 1, а последний равен длине вектора — ее можно узнать с помощью функции `length()`. С обоих сторон индексы берутся включительно.

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

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

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

[24] https://stackoverflow.com/questions/6555651/under-what-circumstances-does-r-recycle

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:

```
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
```

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

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

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

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

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

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

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

```
## [1]  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:

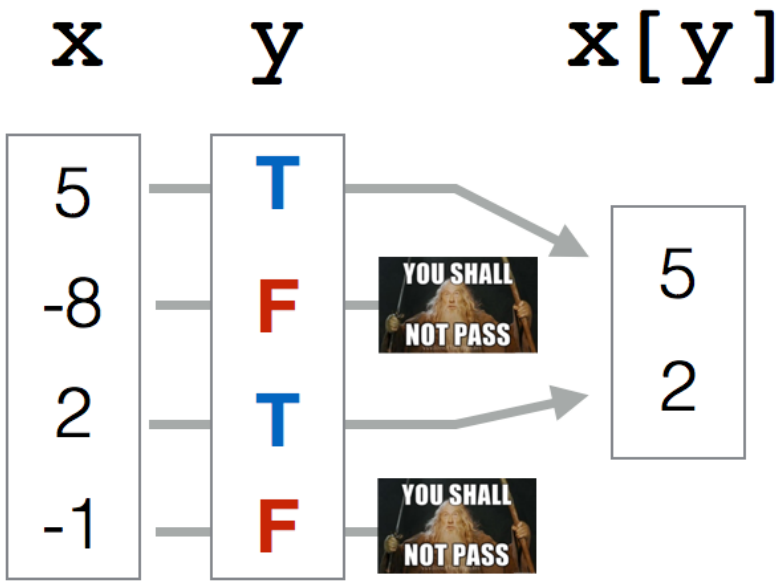


Рис. 2.7

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

```
## first
##     1
```

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

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

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

`letters` - это “зашитая” в R константа - вектор букв от a до z. Иногда это очень удобно! Кроме того, есть константа `LETTERS` - то же самое,

но заглавными буквами. А еще в R есть названия месяцев на английском и числовая константа `pi`.

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 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.

2.8.5 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 обозначения пропущенных значений в R есть специальное значение `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 FALSE
```

Результат выполнения `is.na(n)` выдает FALSE в тех местах, где у нас числа и TRUE там, где у нас NA. Чтобы вычленить из вектора n все значения кроме NA нам нужно, чтобы было наоборот: TRUE, если это не NA, FALSE, если это NA. Здесь нам понадобится логический оператор НЕ `!` (мы его уже встречали), 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 умолчанию `FALSE`. Вы знаете, что нужно делать!

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

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

> `NA` может появляться в векторах других типов тоже. На самом деле, `NA` - это специальное значение в логических векторах, тогда как в векторах других типов `NA` появляется как `NA_integer_`, `NA_real_`, `NA_complex_` или `NA_character_`, но R обычно сам все переводит в нужный формат и показывает как просто `NA`.
> Кроме `NA` есть еще `NaN` — это разные вещи. `NaN` расшифровывается как Not a Number и получается в результате таких операций как `0/0`.

2.8.6 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.

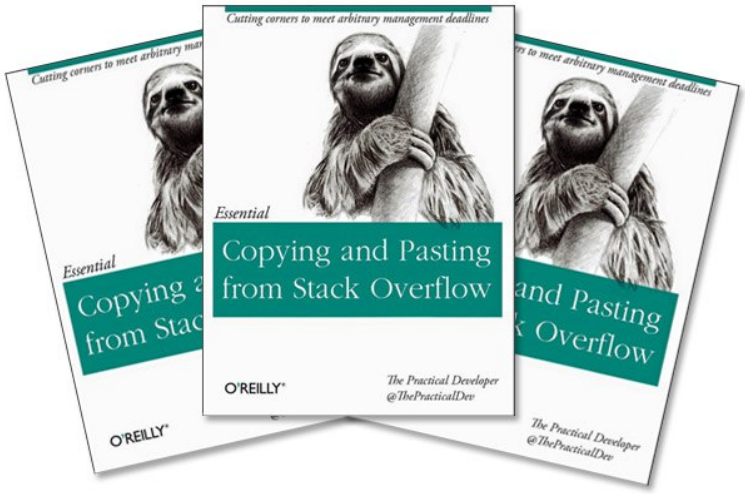


Рис. 2.8

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.?? href="https://twitter.com/mousquemere/status/1125522375141883907?ref_src=twsrc%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**.

2.9 Матрицы (matrix)

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

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

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

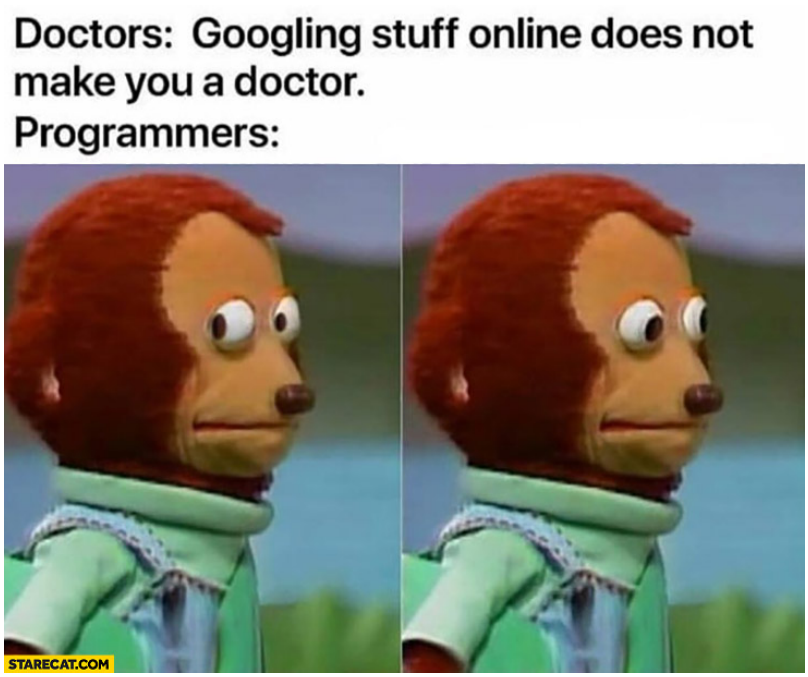


Рис. 2.9

```
## [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
```

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

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

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

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
```

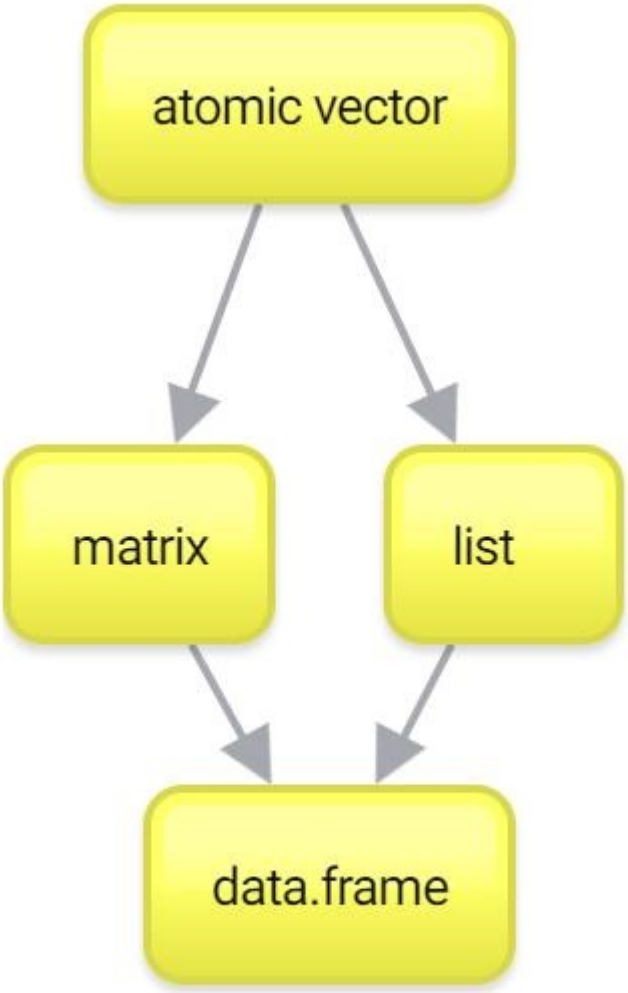


Рис. 2.10

```
## [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
```

Если мы выберем только одну колонку/строчку, то на выходе получим уже вектор, а не матрицу:

```
A[2,]
```

```
## [1]  2  7 12 17
```

Это называется “схлопыванием размерности”. Чтобы этого избежать, нужно поставить `drop = FALSE` после второй запятой внутри квадратных скобок.

```
A[2,, drop = FALSE]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    7   12   17
```

Для соединения двух или более матриц можно воспользоваться функциями `rbind()` и `cbind()` для соединения матриц по вертикали и по горизонтали соответственно.

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

```
cbind(A, A)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    6   11   16    1    6   11   16
## [2,]    2    7   12   17    2    7   12   17
## [3,]    3    8   13   18    3    8   13   18
## [4,]    4    9   14   19    4    9   14   19
## [5,]    5   10   15   20    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 здесь[25] или на стр. 99–101 книги "R in a Nutshell" (Adler, 2010).

2.10 Списки (list)

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

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

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

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

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

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

[25] https://perso.esiee.fr/~courivad/R/06-objects.html

```
##
## [[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(complex_list)
```

```
## 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:

```
named_list <- list(age = 24, phd_student = T, language = "Russian")
named_list
```

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

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

```
named_list$age
```

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

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

```
named_list[1]
```

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

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

```
class(named_list)
```

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

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

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

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

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

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

```
class(named_list[[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.

```
named_list[['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. Особенно это характерно для данных, выкачанных из веб-страниц (HTML страницы, XML данные) или полученных с помощью API различных веб-сайтов (например, в формате JSON). Но обычно после этого стоит как можно скорее превратить список в датафрейм.

2.11 Датафрейм

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 датафрейм.

```
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   : chr  "Ivan" "Eugeny" "Lena" "Misha" ...
##  $ 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 имет 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 это нужно написать команду `View(df)` или же просто нажать на названии нужной переменной из списка вверху справа (там где Environment). Тогда увидите табличку, очень похожую на Excel и тому подобные программы для работы с таблицами. Там же есть и всякие возможности для фильтрации, сортировки и поиска... 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.

Глава 3

Импорт данных

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

3.1 Рабочая папка и проекты RStudio

Для начала скачайте файл по ссылке[2]

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

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

```
## Warning in file(file, "rt"):              'heroes_information.csv': No
## such file or directory
```

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

[2] `https://raw.githubusercontent.com/agricolamz/2020-2021-ds4dh/master/data/heroes_information.csv`

```
## Error in file(file, "rt"):
```

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

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

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

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

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

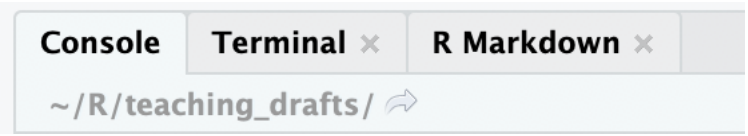


Рис. 3.1

- Гора идет к Магомету: изменение рабочей папки.

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

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

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

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

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

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

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

- Магомет использует кнопочный интерфейс: Import Dataset.

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

- Гора находит Магомета в интернете.

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

```
heroes <- read.csv("https://raw.githubusercontent.com/agricolamz/2020-2021-ds4dh/master/data/hero
```

- 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.

> При закрытии проекта все переменные по умолчанию тоже будут сохраняться, а при открытии — восстанавливаться. Это очень удобно, хотя некоторые рекомендуют от этого отказаться[3]. Это можно сделать во вкладке `Tool - Global Options...`

3.1.1 Табличные данные: текстовые и бинарные данные

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

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

Текстовый файл означает, что такой файл можно открыть в программе “Блокнот” или аналоге (например, TextEdit на macOS) и увидеть напечатанный текст: скрипт, роман или упорядоченный набор цифр и букв. Нас сейчас интересует

[3] `https://r4ds.had.co.nz/workflow-projects.html`

именно последний случай. Таблица может быть представлена как текст: отдельные строчки в файле будут разделять разные строчки таблицы, а какой-нибудь знак-разделитель отделять колонки друг от друга.

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

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

Как говорилось ранее, в качестве разделителя ячеек по горизонтали — то есть разделителя между столбцами — используется запятая. С этим связана одна проблема: в некоторых странах (в т.ч. и России) принято использовать запятую для разделения дробной части числа, а не точку, как это делается в большинстве стран мира. Поэтому есть альтернативный вариант формата .csv, где значения разделены точкой с запятой (`;`), а дробные значения - запятой (`,`). В этом и различие функций `read.csv()` и `read.csv2()` — первая функция предназначена для "международного" формата, вторая - для (условно) "Российского". Оба варианта формата имеют расширение .csv, поэтому заранее понять какой именно будет вариант довольно сложно, приходится либо пробовать оба, либо заранее открывать файл в текстовом редакторе.

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

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

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

FALSE:

```
heroes <- read.csv("data/heroes_information.csv", stringsAsFactors = FALSE)
```

> Параметр `stringsAsFactors =` задает то, как будут прочитаны строковые значения - как уже знакомые нам строки или как факторы. По сути, факторы - это примерно то же самое, что и character, но закодированные числами. Когда-то это было придумано для экономии используемых времени и памяти, сейчас же обычно становится просто лишней морокой. 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. Если у вас версия R выше 4.0, то `stringsAsFactors =` будет `FALSE` по умолчанию.

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

3.2 Проверка импортированных данных

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

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

```
str(heroes)
```

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

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

На что нужно обращать внимание?

1. Прочитаны ли пропущенные значения как `NA`. По умолчанию пропущенные значения обозначаются пропущенной строчкой или "NA", но встречаются самые разнообразные варианты. Возможные варианты кодирования пропущенных значений можно задать в параметре `na.strings =` функции `read.table()` и ее вариантов. В нашем датасете как раз такая ситуация, где нужно самостоятельно задавать, какие значения будут прочитаны как `NA`. Попытайтесь самостоятельно догадаться, как именно.

2. Прочитаны ли те столбики, которые должны быть числовыми, как `int` или `num`. Если в колонке содержатся числа, а написано `chr` (= `"character"`) или `Factor` (в случае если `stringsAsFactors = TRUE`), то, скорее всего, одна из строчек содержит в себе нечисловые знаки, которые не были прочитаны как `NA`.

3. Странные названия колонок. Это может случиться по самым разным причинам, но в таких случаях стоит открывать файл в другой программе и смотреть первые строчки. Например, может оказаться, что первые несколько строчек — пустые или что первая строчка не содержит название столбцов (тогда для параметра `header =` нужно поставить `FALSE`)

4. Вместо строковых данных у вас кракозябры. Это означает проблемы с кодировкой. В первую очередь попробуйте выставить значение `"UTF-8"` для параметра `encoding =` в функции для чтения файла:

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

В случае если это не помогает, попробуйте разобрать[4], что это за кодировка.

5. Все прочиталось как одна колонка. В этом случае, скорее всего, неправильно подобран разделить колонок — параметр `sep =`. Откройте файл в текстовом редакторе, чтобы понять какой нужно использовать.

6. В отдельных строчках все прочиталось как одна колонка, а в остальных нормально. Скорее всего, в файле есть значения типа `\` или `"`, которые в функциях `read.csv()`, `read.delim()`, `read.csv2()`, `read.delim2()` читаются как символы для закавычивания значений. Это может понадобиться, если у вас в таблице есть строковые значения со знаками `,` или `;`, которые могут восприниматься как разделитель столбцов.

[4] `https://www.artlebedev.ru/decoder/`

7. Появились какие-то новые числовые колонки. Возможно неправильно поставлен разделитель дробной части. Обычно это либо . (`read.table()`, `read.csv()`, `read.delim()`), либо , (`read.csv2()`, `read.delim2()`).

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

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

3.3 Экспорт данных

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

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

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

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

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

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

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

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

3.4 Импорт таблиц в бинарном формате: таблицы Excel, SPSS

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

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

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

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

3.5 Быстрый импорт данных

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

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

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

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
##
## -- Column specification --------------------------------
## cols(
##   X1 = col_double(),
##   name = col_character(),
##   Gender = col_character(),
```

```
##   `Eye color` = col_character(),
##   Race = col_character(),
##   `Hair color` = col_character(),
##   Height = col_double(),
##   Publisher = col_character(),
##   `Skin color` = col_character(),
##   Alignment = col_character(),
##   Weight = col_double()
## )
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

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

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

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

```
## Rows: 734
## Columns: 11
## Delimiter: ","
## chr [8]: name, Gender, Eye color, Race, Hair color, Publisher, Skin color, Alignment
## dbl [3]: ...1, Height, Weight
##
## Use `spec()` to retrieve the guessed column specification
## Pass a specification to the `col_types` argument to quiet this message
```

```
## # A tibble: 734 x 11
##      ...1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
```

```
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black           -99 Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        -     Blond           -99 NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        -     Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

- Пакет `data.table` - это не просто пакет, а целый фреймворк для работы с R, основной конкурент tidyverse. Одна из основных фишек `data.table` - быстрота работы. Это касается не только процессинга данных, но и их загрузки и записи. Поэтому некоторые используют функции `data.table` для чтения и записи данных в отдельности от всего остального пакета - они даже и называются соответствующе: `fread()` и `fwrite()`, где **f** означает **f**ast[5].

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

```
##        V1            name Gender Eye color              Race       Hair color
##    1:   0          A-Bomb   Male    yellow             Human          No Hair
##    2:   1      Abe Sapien   Male      blue     Icthyo Sapien          No Hair
##    3:   2        Abin Sur   Male      blue           Ungaran          No Hair
##    4:   3     Abomination   Male     green Human / Radiation          No Hair
##    5:   4         Abraxas   Male      blue     Cosmic Entity            Black
##   ---
## 730: 729 Yellowjacket II Female      blue             Human Strawberry Blond
## 731: 730            Ymir   Male     white       Frost Giant          No Hair
## 732: 731            Yoda   Male     brown    Yoda's species            White
## 733: 732         Zatanna Female      blue             Human            Black
## 734: 733            Zoom   Male       red                 -            Brown
##       Height         Publisher Skin color Alignment Weight
##    1:  203.0     Marvel Comics          -      good    441
##    2:  191.0 Dark Horse Comics       blue      good     65
##    3:  185.0         DC Comics        red      good     90
##    4:  203.0     Marvel Comics          -       bad    441
##    5:  -99.0     Marvel Comics          -       bad    -99
##   ---
## 730:  165.0     Marvel Comics          -      good     52
## 731:  304.8     Marvel Comics      white      good    -99
## 732:   66.0      George Lucas      green      good     17
```

[5] А еще friendly: `fread()` обычно самостоятельно хорошо угадывает формат таблицы на входе. `vroom` тоже так умеет.

```
## 733:  170.0         DC Comics           -      good     57
## 734:  185.0         DC Comics           -       bad     81
```

Чем же пользоваться среди всего этого многообразия? Бенчмарки[6][7] показывают, что быстрее всех `vroom` и `data.table`. Если же у вас нет задачи ускорить работу кода на несколько миллисекунд или прочитать датасет на много миллионов строк, то стандартного `read.csv()` (если вы работаете в базовом R) и `readr::read_csv()` (если вы работаете в tidyverse) должно быть достаточно.

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

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

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

[6] `https://www.danielecook.com/speeding-up-reading-and-writing-in-r/`

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

Глава 4

Условные конструкции и циклы

4.1 Выражения `if`, `else`, `else if`

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

```
if (    )
```

Вот так это будет работать на практике:

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

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

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

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

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

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

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

```
if (    )       else
```

Работает это так:

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

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

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

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

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

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

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

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

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

```
} else {
  "   "
}
```

```
## Warning in if (number > 0) {:          > 1,
##

## Warning in if (number < 0) {:          > 1,
##

## [1] "           "
```

R выдает сообщение, что используется только первое значение логического вектора внутри условия. Остальные просто игнорируются. Как же посчитать для всего вектора сразу?

4.2 Циклы `for`

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

```
for(       in           )
```

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

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

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

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

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

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

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

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

Если же нет подходящей векторизованной функции, то можно воспользоваться семейством функций `apply()` (см. @ref(apply_f)).

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

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

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

```
  }
}
number_descriptions
```

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

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

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

Альтернатива сочетанию условных конструкций и циклов `for` является использование встроенной функции `ifelse()`. Функция `ifelse()` принимает три аргумента - 1) условие (т.е. просто логический вектор, состоящий из `TRUE` и `FALSE`), 2) что выдавать в случае `TRUE`, 3) что выдавать в случае `FALSE`. На выходе получается вектор такой же длины, как и изначальный логический вектор (условие).

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

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

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

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

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

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

Достаточно симпатичное решение предлогает пакет `dplyr` (основа tidyverse) — функция `case_when()`, которая работает с использованием формулы:

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

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

Глава 5

Функциональное программирование в R

5.1 Создание функций

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

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

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

```
## [1] 9
```

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

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

```
## [1] 9
```

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

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

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

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

```
## [1] 9
```

Вообще, фигурные скобки используются для того, чтобы выполнить серию выражений, но вернуть только результат выполнения последнего выражения. Это можно использовать, чтобы не создавать лишних временных переменных в глобальном окружении.

Мы можем оставить в функции параметры по умолчанию.

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

```
## [1] 9
```

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

```
## [1] 27
```

В R работают **ленивые вычисления (lazy evaluations)**. Это означает, что параметры функций будут только когда они понадобятся, а не заранее. Например, эта функция не будет выдавать ошибку, если мы не зададим параметр `we_will_not_use_this_parameter =`, потому что он нигде не используется в расчетах.

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

```
## [1] 9
```

5.2 Проверка на адекватность

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

Хорошо написанные функции не только выдают правильный результат на все возможные адекватные данные на входе, но и не дают получить правдоподобные результаты при неадекватных входных данных. Как вы уже знаете, если на входе у вас имеются пропущенные значения, то многие функции будут в ответ тоже выдавать пропущенные значения. И это вполне осознанное решение, которое позволяет избегать ситуаций вроде той, когда около одной пятой научных статей по генетике содержало ошибки в приложенных данных[1] и замечать пропущенные значения на ранней стадии. Кроме того, можно проводить проверки на адекватность входящих данных (sanity check).

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

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

Проверим, что функция работает верно:

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

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

[1] https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-1044-7

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

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

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

```
## [1] 0.002796802
```

В некоторых случаях ответ будет совершенно точно некорректным, хотя функция все посчитает и выдаст ответ, как будто так и надо. Например, если какой-то из аргументов функции `imt()` будет меньше или равен 0. В этом случае нужно прописать проверку на это условие, и если это действительно так, то выдать пользователю ошибку.

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

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

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

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

Ну а в дальнейшем я советую ознакомиться со средствами отладки кода в R[2] для того, чтобы научиться справляться с ошибками в своем коде на более продвинутом уровне.

[2] https://adv-r.hadley.nz/debugging.html

5.3 Когда и зачем создавать функции?

Когда стоит создавать функции? Существует “правило трех”[3] — если у вас есть три куска очень похожего кода, то самое время превратить код в функцию. Это очень условное правило, но, действительно, стоит избегать копипастинга в коде. В этом случае очень легко ошибиться, а сам код становится нечитаемым.

Есть и другой подход к созданию функций: их стоит создавать не столько для того, чтобы использовать тот же код снова, сколько для абстрагирования от того, что происходит в отдельных строчках кода. Если несколько строчек кода были написаны для того, чтобы решить одну задачу, которой можно дать понятное название (например, подсчет какой-то особенной метрики, для которой нет готовой функции в R), то этот код стоит обернуть в функцию. Если функция работает корректно, то теперь не нужно думать над тем, что происходит внутри нее. Вы ее можете мысленно представить как операцию, которая имеет определенный вход и выход — как и встроенные функции в R.

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

5.4 Функции как объекты первого порядка

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

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

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

```
## [[1]]
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x7fd99b92ac58>
## <environment: namespace:base>
##
## [[2]]
## function (..., na.rm = FALSE)  .Primitive("min")
##
## [[3]]
```

[3] https://en.wikipedia.org/wiki/Rule_of_three_(computer_programming)

```
## .Primitive("{")
```

Мы можем создать список из функций! Зачем — это другой вопрос, но ведь можем же!

Еще можно создавать функции внутри функций [4], использовать функции в качестве аргументов функций, сохранять функции как переменные. Пожалуй, самое важное из этого всего - это то, что функция может быть аргументом в функции. Не просто название функции как строковая переменная, не результат выполнения функции, а именно сама функция. Это лежит в основе использования семейства функций `apply()` @ref(apply_f) и многих фишек tidyverse.

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

5.5 Семейство функций apply()

5.5.1 Применение `apply()` для матриц

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

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

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

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

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

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

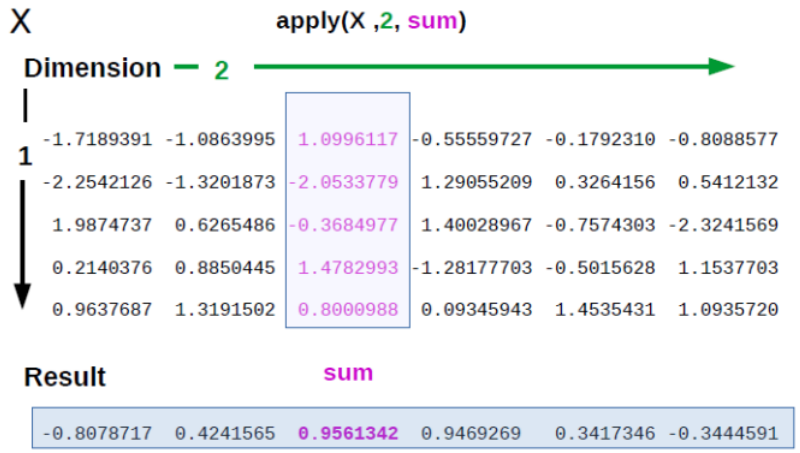


Рис. 5.1: apply

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

Давайте разберем на примере:

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

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

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

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

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

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
```

```
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

> Конкретно для подсчета сумм и средних по столбцам и строкам в R есть функции `colSums()`, `rowSums()`, `colMeans()` и `rowMeans()`, которые можно использовать как альтернативы `apply()` в данном случае.

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

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

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

```
apply(A, 1, weighted.mean, w = c(0.2, 0.4, 0.3, 0.1))
```

```
## [1] 4.9 5.9 6.9
```

5.5.2 Анонимные функции

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

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

> Питонистам знакомо понятие **лямбда-функций**. Да, это то же самое.

Например, мы можем посчитать отклонения от среднего без называния этой функции:

```
apply(A, 1, function(x) x - mean(x)) #
```

```
##      [,1] [,2] [,3]
## [1,] -4.5 -4.5 -4.5
## [2,] -1.5 -1.5 -1.5
## [3,]  1.5  1.5  1.5
## [4,]  4.5  4.5  4.5
```

```
apply(A, 2, function(x) x - mean(x)) #
```

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

```
## [2,]    0    0    0    0
## [3,]    1    1    1    1
```

```
apply(A, c(1,2), function(x) x - mean(x)) #                , . .
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    0
## [2,]    0    0    0    0
## [3,]    0    0    0    0
```

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

```
apply(A, 1, function(whatevername) whatevername - mean(whatevername))
```

```
##      [,1] [,2] [,3]
## [1,] -4.5 -4.5 -4.5
## [2,] -1.5 -1.5 -1.5
## [3,]  1.5  1.5  1.5
## [4,]  4.5  4.5  4.5
```

5.5.3 Другие функции семейства `apply()`

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

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

```
## $some
## [1] 10
##
## $list
## [1] 26
```

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

```
## some list
##   10   26
```

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

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

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

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

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

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

> Еще одна альтернатива - это векторизация невекторизованной функции с помощью `Vectorize()`. Эта функция просто оборачивает функцию одним из вариантов `apply()`.

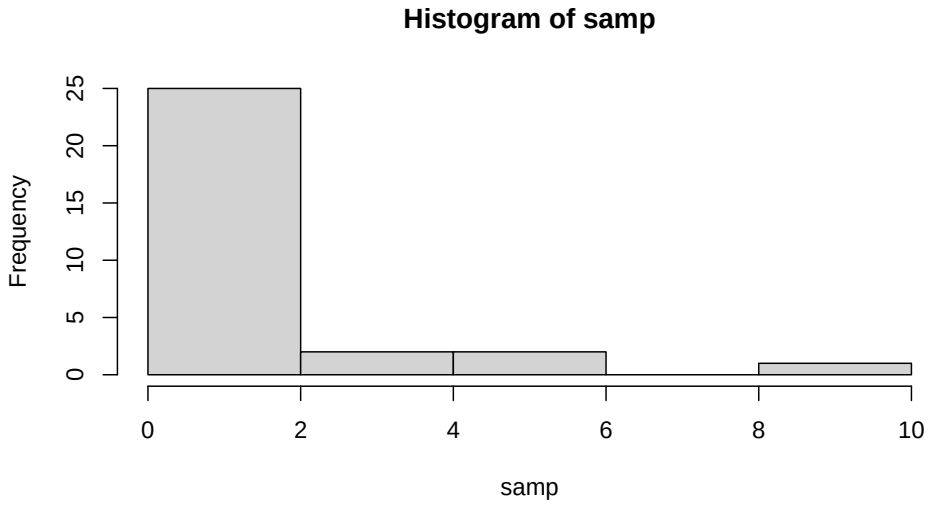
Можно применять функции `lapply()` и `sapply()` на датафреймах. Поскольку фактически датафрейм - это список из векторов одинаковой длины (см. 2.11), то итерироваться эти функции будут по колонкам:

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

```
##           X        name      Gender   Eye.color        Race  Hair.color
##   "integer" "character" "character" "character" "character" "character"
##      Height   Publisher  Skin.color   Alignment      Weight
##   "numeric" "character" "character" "character"   "integer"
```

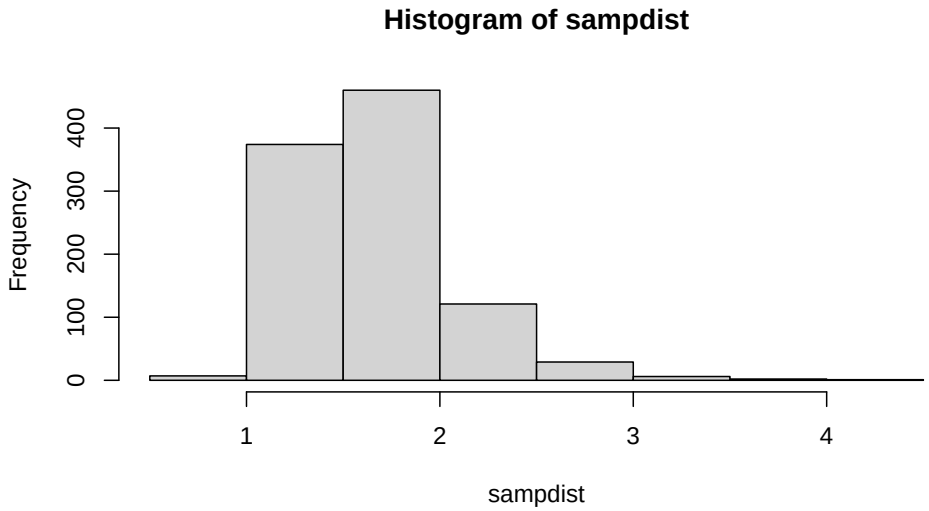
Еще одна функция из семейства `apply()` - функция `replicate()` - самый простой способ повторить одну и ту же операцию много раз. Обычно эта функция используется при симуляции данных и моделировании. Например, давайте сделаем выборку из логнормального распределения:

```
samp <- rlnorm(30)
hist(samp)
```



А теперь давайте сделаем 1000 таких выборок и из каждой возьмем среднее:

```
sampdist <- replicate(1000, mean(rlnorm(30)))
hist(sampdist)
```



> Про функции для генерации случайных чисел и про визуализацию мы поговорим в следующие дни.

Если хотите познакомиться с семейством `apply()` чуточку ближе, то рекомендую вот этот туториал[5].

В заключение стоит сказать, что семейство функций `apply()` — это очень силь-

[5] https://www.datacamp.com/community/tutorials/r-tutorial-apply-family

ное колдунство, но в tidyverse оно практически полностью перекрывается функциями из пакета `purrr`. Впрочем, если вы поняли логику `apply()`, то при желании вы легко сможете переключиться на альтернативы из пакета `purrr`.

Глава 6

Введение в tidyverse

6.1 Вселенная tidyverse

tidyverse - это не один, а целое множество пакетов. Есть ключевые пакеты (ядро тайдиверса), а есть побочные - в основном для работы со специфическими видами данных.

tidyverse[1] — это набор пакетов:

- *ggplot2*, для визуализации
- *tibble*, для работы с тибблами, продвинутый вариант датафрейма
- *tidyr*, для формата tidy data
- *readr*, для чтения файлов в R
- *purrr*, для функционального программирования (замена семейства функций *apply())
- *dplyr*, для преобразованиия данных
- *stringr*, для работы со строковыми переменными
- *forcats*, для работы с переменными-факторами

Полезно также знать о следующих пакетах, не включенных в ядро, но также считающихся частью тайдиверса:

- *vroom*, для быстрой загрузки табоичных данных
- *readxl*, для чтения .xls и .xlsx
- *jsonlite*, для работы с JSON
- *xml*, для работы с XML
- *DBI*, для работы с базами данных
- *rvest*, для веб-скреппинга
- *lubridate*, для работы с временем
- *tidytext*, для работы с текстами и корпусами

[1] `https://www.tidyverse.org`

- *glue*, для продвинутого объединения строк
- *magrtittr*, с несколькими вариантами pipe оператора
- *tidymodels*, для моделирования и машинного обучения[2]
- *dtplyr*, для ускорения `dplyr` за счет перевод синтаксиса на `data.table`

И это еще не все пакеты tidyverse! Есть еще много других небольших пакетов, которые тоже считаются частью tidyverse. Кроме официальных пакетов tidyverse есть множество пакетов, которые пытаются соответствовать принципам tidyverse и дополняют его.

Все пакеты tidyverse объединены tidy философией и взаимосовместимым синтаксисом. Это означает, что, во многих случаях даже не нужно думать о том, из какого именно пакета тайдиверса пришла функция. Можно просто установить и загрузить пакет `tidyverse`.

```
install.packages("tidyverse")
```

Пакет `tidyverse` — это такой пакет с пакетами[3].

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

Подключение пакета `tidyverse` автоматически приводит к подключению ядра tidyverse, остальные же пакеты нужно подключать дополнительно при необходимости.

6.2 Загрузка данных с помощью `readr`

Стандартной функцией для чтения `.csv` файлов в R является функция `read.csv()`, но мы будем использовать функцию `read_csv()` из пакета `readr`. Синтаксис функции `read_csv()` очень похож на `read.csv()`: первым аргументом является путь к файлу (в том числе можно использовать URL), некоторые остальные параметры тоже совпадают.

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

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
##
## -- Column specification --------------------------------
## cols(
```

[2] Как и пакет `tidyverse`, `tidymodels` — это пакет с несколькими пакетами.

[3] `https://cs11.pikabu.ru/post_img/big/2019/03/12/11/1552415351186680692.jpg`

```
##   X1 = col_double(),
##   name = col_character(),
##   Gender = col_character(),
##   `Eye color` = col_character(),
##   Race = col_character(),
##   `Hair color` = col_character(),
##   Height = col_double(),
##   Publisher = col_character(),
##   `Skin color` = col_character(),
##   Alignment = col_character(),
##   Weight = col_double()
## )
```

Подробнее про импорт данных, в том числе в tidyverse, смотри в @ref(real_data).

6.3 tibble

Когда мы загрузили данные с помощью `read_csv()`, то мы получили `tibble`, а не `data.frame`:

```
class(heroes)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Тиббл (`tibble`) - это такой "усовершенствованный" `data.frame`. Почти[4] все, что работает с `data.frame`, работает и с тибблами. Однако у тибблов есть свои дополнительные фишки. Самая очевидная из них - более аккуратный вывод в консоль:

```
heroes
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
```

[4] https://www.jumpingrivers.com/blog/the-trouble-with-tibbles/

```
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Выводятся только первые 10 строк, если какие-то колонки не влезают на экран, то они просто перечислены внизу. Ну а тип данных написан прямо под названием колонки.

Функции различных пакетов tidyverse сами конвертируют в тиббл при необходимости. Если же нужно это сделать самостоятельно, то можно это сделать так:

```
heroes_df <- as.data.frame(heroes) #
class(heroes_df)
```

```
## [1] "data.frame"
```

```
as_tibble(heroes_df) #
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

> В дальнейшем мы будем работать только с tidyverse, а это значит, что только с тибблами, а не обычными датафреймами. Тем не менее, тибблы и датафреймы будут в дальнейшем использоваться как синонимы.

Можно создавать тибблы вручную с помощью функции `tibble()`, которая работает аналогично функции `data.frame()`:

```
tibble(
  a = 1:3,
  b = letters[1:3]
)
```

```
## # A tibble: 3 x 2
##       a b
##   <int> <chr>
## 1     1 a
## 2     2 b
## 3     3 c
```

6.4 magrittr::%>%

Оператор %>% называется “пайпом” (pipe), т.е. “трубой”. Он означает, что следующая функция (справа от пайпа) принимает на вход в качестве первого аргумента результат выполнения предыдущей функции (той, что слева). Фактически, это примерно то же самое, что и вставлять результат выполнения функции в качестве первого аргумента в другую функцию. Просто выглядит это красивее и читабельнее. Как будто данные пропускаются через трубы функций или конвеерную ленту на заводе, если хотите. А то, что первый параметр функции - это почти всегда данные, работает нам здесь на руку. Этот оператор взят из пакета `magrittr`[5]. Возможно, даже если вы не захотите пользоваться tidyverse, использование пайпов Вам понравится.

Важно понимать, что пайп не дает какой-то дополнительной функциональности или дополнительной скорости работы[6]. Он создан исключительно для читабельности и комфорта.

С помощью пайпов вот эту команду...

```
sum(sqrt(abs(sin(1:22))))
```

```
## [1] 16.72656
```

...можно переписать вот так:

```
1:22 %>%
  sin() %>%
  abs() %>%
  sqrt() %>%
  sum()
```

```
## [1] 16.72656
```

[5]Если быть точным, то оператор %>% был импортирован во все основные пакеты tidyverse, а сам пакет `magrittr` не входит в базовый набор tidyverse. Тем не менее, в самом `magrittr` есть еще несколько интересных операторов.

[6]Даже наоборот, использование пайпов незначительно снижает скорость выполнения команды.

В очень редких случаях результат выполнения функции нужно вставить не на первую позицию (или же мы хотим использовать его несколько раз). В этих случаях можно использовать `.`, чтобы обозначить, куда мы хотим вставить результат выполнения выражения слева от %>%.

```
"       !" %>%
  c("--", ., "--")
```

```
## [1] "--"           "       !" "--"
```

6.5 Главные пакеты tidyverse: `dplyr` и `tidyr`

`dplyr`[7] — это самая основа всего `tidyverse`. Этот пакет предоставляет основные функции для манипуляции с тибблами. Пакет `dplyr` является наследником и более усовершенствованной версией `plyr`, так что если увидите использование пакета `plyr`, то, скорее всего, скрипт был написан очень давно.

Пакет `tidyr` дополняет `dplyr`, предоставляя полезные функции для тайдификации тибблов. Тайдификация ("аккуратизация") данных означает приведение табличных данных к такому формату, в котором:

- Каждая переменная имеет собственный столбец
- Каждый наблюдение имеет собственную строку
- Каждое значение имеет свою собственную ячейку

Впрочем, многие функции `dplyr` часто используются при тайдификации, так же как и многие функции `tidyr` имеет применение вне тайдификации. В общем, функционал этих двух пакетов несколько смешался, поэтому мы будем рассматривать их вместе. А чтобы представлять, какая функция относится к какому пакету (хотя запоминать это необязательно), я буду использовать запись с двумя двоеточиями `::`, которая обычно используется для использования функции без подгрузки всего пакета, при первом упоминании функции.

Пакет `tidyr` — это более усовершенствованная версия пакета `reshape2`, который в свою очередь является усовершенствованной версией `reshape`. По аналогии с `plyr`, если вы видите использование этих пакетов, то это указывает на то, что перед вами морально устаревший код.

Код с использованием `dplyr` и `tidyr`сильно непохож на то, что мы видели раньше. Большинство функций `dplyr` и `tidyr` работают с целым тибблом сразу, принимая его в качестве первого аргумента и возвращая измененный тиббл. Это позволяет превратить весь код в последовательный набор применяемых функций, соединенный пайпами. На практике это выглядит очень элегантно, и вы в этом скоро убедитесь.

[7] Есть споры о том, как это правильно читать[8]. Используемые варианты: *диплаер*, *диплюр*, *диплир*.

6.6 Работа с колонками тиббла

6.6.1 Выбор колонок: `dplyr::select()`

Функция `dplyr::select()` позволяет выбирать колонки по номеру или имени (кавычки не нужны).

```
heroes %>%
  select(1,5)
```

```
## # A tibble: 734 x 2
##       X1 Race
##    <dbl> <chr>
##  1     0 Human
##  2     1 Icthyo Sapien
##  3     2 Ungaran
##  4     3 Human / Radiation
##  5     4 Cosmic Entity
##  6     5 Human
##  7     6 <NA>
##  8     7 Human
##  9     8 <NA>
## 10     9 Human
## # ... with 724 more rows
```

```
heroes %>%
  select(name, Race, Publisher, `Hair color`)
```

```
## # A tibble: 734 x 4
##    name          Race              Publisher         `Hair color`
##    <chr>         <chr>             <chr>             <chr>
##  1 A-Bomb        Human             Marvel Comics     No Hair
##  2 Abe Sapien    Icthyo Sapien     Dark Horse Comics No Hair
##  3 Abin Sur      Ungaran           DC Comics         No Hair
##  4 Abomination   Human / Radiation Marvel Comics     No Hair
##  5 Abraxas       Cosmic Entity     Marvel Comics     Black
##  6 Absorbing Man Human             Marvel Comics     No Hair
##  7 Adam Monroe   <NA>              NBC - Heroes      Blond
##  8 Adam Strange  Human             DC Comics         Blond
##  9 Agent 13      <NA>              Marvel Comics     Blond
## 10 Agent Bob     Human             Marvel Comics     Brown
## # ... with 724 more rows
```

Обратите внимание, если в названии колонки присутствует пробел или, например, колонка начинается с цифры или точки и цифры, то это синтаксически

невалидное имя (2.5.3). Это не значит, что такие названия колонок недопустимы. Но такие названия колонок нужно обособлять ‘ грависом (правый штрих, на клавиатуре находится там же где и буква ё и ~).

Еще обратите внимание на то, что функции tidyverse не изменяют сами изначальные тибблы/датафреймы. Это означает, что если вы хотите полученный результат сохранить, то нужно добавить присвоение:

```
heroes_some_cols <- heroes %>%
  select(name, Race, Publisher, `Hair color`)
heroes_some_cols
```

```
## # A tibble: 734 x 4
##    name          Race              Publisher         `Hair color`
##    <chr>         <chr>             <chr>             <chr>
##  1 A-Bomb        Human             Marvel Comics     No Hair
##  2 Abe Sapien    Icthyo Sapien     Dark Horse Comics No Hair
##  3 Abin Sur      Ungaran           DC Comics         No Hair
##  4 Abomination   Human / Radiation Marvel Comics     No Hair
##  5 Abraxas       Cosmic Entity     Marvel Comics     Black
##  6 Absorbing Man Human             Marvel Comics     No Hair
##  7 Adam Monroe   <NA>              NBC - Heroes      Blond
##  8 Adam Strange  Human             DC Comics         Blond
##  9 Agent 13      <NA>              Marvel Comics     Blond
## 10 Agent Bob     Human             Marvel Comics     Brown
## # ... with 724 more rows
```

6.7 Мини-язык tidyselect для выбора колонок

Для выбора столбцов (не только в `select()`, но и для других функций tidyverse) используется специальный мини-язык tidyselect из одноименного пакета[9]. tidyselect дает очень широкие возможности для выбора колонок.

Можно использовать оператор `:` для выбора нескольких соседних колонок (по аналогии с созданием числового вектора с шагом 1).

```
heroes %>%
  select(name:Publisher)
```

```
## # A tibble: 734 x 7
##    name       Gender `Eye color` Race          `Hair color` Height Publisher
##    <chr>      <chr>  <chr>       <chr>         <chr>         <dbl> <chr>
```

[9]Как и в случае с `magrittr`, пакет `tidyselect` не содержатся в базовом tidyverse, но функции импортируются основными пакетами tidyverse.

```
##  1 A-Bomb     Male   yellow      Human         No Hair         203 Marvel Comics
##  2 Abe Sapien Male   blue        Icthyo Sapien No Hair         191 Dark Horse C~
##  3 Abin Sur   Male   blue        Ungaran       No Hair         185 DC Comics
##  4 Abominati~ Male   green       Human / Radi~ No Hair         203 Marvel Comics
##  5 Abraxas    Male   blue        Cosmic Entity Black            NA Marvel Comics
##  6 Absorbing~ Male   blue        Human         No Hair         193 Marvel Comics
##  7 Adam Monr~ Male   blue        <NA>          Blond            NA NBC - Heroes
##  8 Adam Stra~ Male   blue        Human         Blond           185 DC Comics
##  9 Agent 13   Female blue        <NA>          Blond           173 Marvel Comics
## 10 Agent Bob  Male   brown       Human         Brown           178 Marvel Comics
## # ... with 724 more rows
```

```
heroes %>%
  select(name:`Eye color`, Publisher:Weight)
```

```
## # A tibble: 734 x 7
##    name         Gender `Eye color` Publisher       `Skin color` Alignment Weight
##    <chr>        <chr>  <chr>       <chr>           <chr>        <chr>      <dbl>
##  1 A-Bomb       Male   yellow      Marvel Comics   <NA>         good         441
##  2 Abe Sapien   Male   blue        Dark Horse Com~ blue         good          65
##  3 Abin Sur     Male   blue        DC Comics       red          good          90
##  4 Abomination  Male   green       Marvel Comics   <NA>         bad          441
##  5 Abraxas      Male   blue        Marvel Comics   <NA>         bad           NA
##  6 Absorbing M~ Male   blue        Marvel Comics   <NA>         bad          122
##  7 Adam Monroe  Male   blue        NBC - Heroes    <NA>         good          NA
##  8 Adam Strange Male   blue        DC Comics       <NA>         good          88
##  9 Agent 13     Female blue        Marvel Comics   <NA>         good          61
## 10 Agent Bob    Male   brown       Marvel Comics   <NA>         good          81
## # ... with 724 more rows
```

Используя ! можно вырезать ненужные колонки.

```
heroes %>%
  select(!X1)
```

```
## # A tibble: 734 x 10
##    name  Gender `Eye color` Race  `Hair color` Height Publisher `Skin color`
##    <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>     <chr>
##  1 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~ <NA>
##  2 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~ blue
##  3 Abin~ Male   blue        Unga~ No Hair         185 DC Comics red
##  4 Abom~ Male   green       Huma~ No Hair         203 Marvel C~ <NA>
##  5 Abra~ Male   blue        Cosm~ Black            NA Marvel C~ <NA>
##  6 Abso~ Male   blue        Human No Hair         193 Marvel C~ <NA>
##  7 Adam~ Male   blue        <NA>  Blond            NA NBC - He~ <NA>
```

```
##  8 Adam~ Male   blue        Human Blond           185 DC Comics <NA>
##  9 Agen~ Female blue        <NA>  Blond           173 Marvel C~ <NA>
## 10 Agen~ Male   brown       Human Brown           178 Marvel C~ <NA>
## # ... with 724 more rows, and 2 more variables: Alignment <chr>, Weight <dbl>
```

```
heroes %>%
  select(!(Gender:Height))
```

```
## # A tibble: 734 x 6
##       X1 name          Publisher         `Skin color` Alignment Weight
##    <dbl> <chr>         <chr>             <chr>        <chr>      <dbl>
##  1     0 A-Bomb        Marvel Comics     <NA>         good         441
##  2     1 Abe Sapien    Dark Horse Comics blue         good          65
##  3     2 Abin Sur      DC Comics         red          good          90
##  4     3 Abomination   Marvel Comics     <NA>         bad          441
##  5     4 Abraxas       Marvel Comics     <NA>         bad           NA
##  6     5 Absorbing Man Marvel Comics     <NA>         bad          122
##  7     6 Adam Monroe   NBC - Heroes      <NA>         good          NA
##  8     7 Adam Strange  DC Comics         <NA>         good          88
##  9     8 Agent 13      Marvel Comics     <NA>         good          61
## 10     9 Agent Bob     Marvel Comics     <NA>         good          81
## # ... with 724 more rows
```

Другие известные нам логические операторы (`&` и `|`) тоже работают в tidyselect.

В дополнение к логическим операторам и `:`, в tidyselect есть набор вспомогательных функций, работающих исключительно в контексте выбора колонок с помощью tidyselect.

Вспомогательная функция `last_col()` позволит обратиться к последней колонке тиббла:

```
heroes %>%
  select(name:last_col())
```

```
## # A tibble: 734 x 10
##    name  Gender `Eye color` Race  `Hair color` Height Publisher `Skin color`
##    <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>     <chr>
##  1 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~ <NA>
##  2 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~ blue
##  3 Abin~ Male   blue        Unga~ No Hair         185 DC Comics red
##  4 Abom~ Male   green       Huma~ No Hair         203 Marvel C~ <NA>
##  5 Abra~ Male   blue        Cosm~ Black            NA Marvel C~ <NA>
##  6 Abso~ Male   blue        Human No Hair         193 Marvel C~ <NA>
##  7 Adam~ Male   blue        <NA>  Blond            NA NBC - He~ <NA>
##  8 Adam~ Male   blue        Human Blond           185 DC Comics <NA>
```

```
##  9 Agen~ Female blue        <NA>  Blond           173 Marvel C~ <NA>
## 10 Agen~ Male   brown       Human Brown           178 Marvel C~ <NA>
## # ... with 724 more rows, and 2 more variables: Alignment <chr>, Weight <dbl>
```

А функция `everything()` позволяет выбрать все колонки.

```
heroes %>%
  select(everything())
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

При этом `everything()` не будет дублировать выбранные колонки, поэтому можно использовать `everything()` для перестановки колонок в тиббле:

```
heroes %>%
  select(name, Publisher, everything())
```

```
## # A tibble: 734 x 11
##    name  Publisher    X1 Gender `Eye color` Race  `Hair color` Height
##    <chr> <chr>     <dbl> <chr>  <chr>       <chr> <chr>         <dbl>
##  1 A-Bo~ Marvel C~     0 Male   yellow      Human No Hair         203
##  2 Abe ~ Dark Hor~     1 Male   blue        Icth~ No Hair         191
##  3 Abin~ DC Comics     2 Male   blue        Unga~ No Hair         185
##  4 Abom~ Marvel C~     3 Male   green       Huma~ No Hair         203
##  5 Abra~ Marvel C~     4 Male   blue        Cosm~ Black            NA
##  6 Abso~ Marvel C~     5 Male   blue        Human No Hair         193
##  7 Adam~ NBC - He~     6 Male   blue        <NA>  Blond            NA
##  8 Adam~ DC Comics     7 Male   blue        Human Blond           185
##  9 Agen~ Marvel C~     8 Female blue        <NA>  Blond           173
## 10 Agen~ Marvel C~     9 Male   brown       Human Brown           178
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
```

```
## #   Alignment <chr>, Weight <dbl>
```

Впрочем, для перестановки колонок удобнее использовать специальную функцию `relocate()` (@ref()) Можно даже выбирать колонки по паттернам в названиях. Например, с помощью `ends_with()` можно выбрать все колонки, заканчивающиеся одинаковым суффиксом:

```
heroes %>%
  select(ends_with("color"))
```

```
## # A tibble: 734 x 3
##    `Eye color` `Hair color` `Skin color`
##    <chr>       <chr>        <chr>
##  1 yellow      No Hair      <NA>
##  2 blue        No Hair      blue
##  3 blue        No Hair      red
##  4 green       No Hair      <NA>
##  5 blue        Black        <NA>
##  6 blue        No Hair      <NA>
##  7 blue        Blond        <NA>
##  8 blue        Blond        <NA>
##  9 blue        Blond        <NA>
## 10 brown       Brown        <NA>
## # ... with 724 more rows
```

Аналогично, с помощью функции `starts_with()` можно найти колонки с одинаковым префиксом, с помощью `contains()` — все колонки с выбранным паттерном в любой части названия колонки[10].

```
heroes %>%
  select(starts_with("Eye") & ends_with("color"))
```

```
## # A tibble: 734 x 1
##    `Eye color`
##    <chr>
##  1 yellow
##  2 blue
##  3 blue
##  4 green
##  5 blue
##  6 blue
##  7 blue
##  8 blue
```

[10] Выбранный паттерн будет найден посимвольно, если же вы хотите искать по регулярным выражениям, то вместо `contains()` нужно использовать `matches()`.

```
##  9 blue
## 10 brown
## # ... with 724 more rows
```

```
heroes %>%
  select(contains("eight"))
```

```
## # A tibble: 734 x 2
##    Height Weight
##     <dbl>  <dbl>
##  1    203    441
##  2    191     65
##  3    185     90
##  4    203    441
##  5     NA     NA
##  6    193    122
##  7     NA     NA
##  8    185     88
##  9    173     61
## 10    178     81
## # ... with 724 more rows
```

Ну и наконец, можно выбирать по содержимому колонок с помощью `where()`. Это напоминает применение `sapply()`(@ref(apply_other})) на датафрейме для индексирования колонок: в качестве аргумента для `where` принимается функция, которая применяется для каждой из колонок, после чего выбираются только те колонки, для которых было получено `TRUE`.

```
heroes %>%
  select(where(is.numeric))
```

```
## # A tibble: 734 x 3
##       X1 Height Weight
##    <dbl>  <dbl>  <dbl>
##  1     0    203    441
##  2     1    191     65
##  3     2    185     90
##  4     3    203    441
##  5     4     NA     NA
##  6     5    193    122
##  7     6     NA     NA
##  8     7    185     88
##  9     8    173     61
## 10     9    178     81
## # ... with 724 more rows
```

Функция `where()` дает невиданную мощь. Например, можно выбрать все колонки без `NA`:

```
heroes %>%
  select(where(function(x) !any(is.na(x))))
```

```
## # A tibble: 734 x 3
##       X1 name          Publisher
##    <dbl> <chr>         <chr>
##  1     0 A-Bomb        Marvel Comics
##  2     1 Abe Sapien    Dark Horse Comics
##  3     2 Abin Sur      DC Comics
##  4     3 Abomination   Marvel Comics
##  5     4 Abraxas       Marvel Comics
##  6     5 Absorbing Man Marvel Comics
##  7     6 Adam Monroe   NBC - Heroes
##  8     7 Adam Strange  DC Comics
##  9     8 Agent 13      Marvel Comics
## 10     9 Agent Bob     Marvel Comics
## # ... with 724 more rows
```

6.7.1 Переименование колонок: `dplyr::rename()`

Внутри `select()` можно не только выбирать колонки, но и переименовывать их:

```
heroes %>%
  select(id = X1)
```

```
## # A tibble: 734 x 1
##       id
##    <dbl>
##  1     0
##  2     1
##  3     2
##  4     3
##  5     4
##  6     5
##  7     6
##  8     7
##  9     8
## 10     9
## # ... with 724 more rows
```

Однако удобнее для этого использовать специальную функцию `dplyr::rename()`.

Синтаксис у нее такой же, как и у `select()`, но `rename()` не выбрасывает колонки, которые не были упомянуты.

```
heroes %>%
  rename(id = X1)
```

```
## # A tibble: 734 x 11
##       id name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Для массового переименования колонок можно использовать функцию `rename_with()`. Эта функция так же использует tidyselect синтаксис для выбора колонок (по умолчанию выбираются все колонки) и применяет функцию в качестве аргумента, которая изменяет

```
heroes %>%
  rename_with(make.names)
```

```
## # A tibble: 734 x 11
##       X1 name  Gender Eye.color Race  Hair.color Height Publisher Skin.color
##    <dbl> <chr> <chr>  <chr>     <chr> <chr>       <dbl> <chr>     <chr>
##  1     0 A-Bo~ Male   yellow    Human No Hair       203 Marvel C~ <NA>
##  2     1 Abe ~ Male   blue      Icth~ No Hair       191 Dark Hor~ blue
##  3     2 Abin~ Male   blue      Unga~ No Hair       185 DC Comics red
##  4     3 Abom~ Male   green     Huma~ No Hair       203 Marvel C~ <NA>
##  5     4 Abra~ Male   blue      Cosm~ Black          NA Marvel C~ <NA>
##  6     5 Abso~ Male   blue      Human No Hair       193 Marvel C~ <NA>
##  7     6 Adam~ Male   blue      <NA>  Blond          NA NBC - He~ <NA>
##  8     7 Adam~ Male   blue      Human Blond         185 DC Comics <NA>
##  9     8 Agen~ Female blue      <NA>  Blond         173 Marvel C~ <NA>
## 10     9 Agen~ Male   brown     Human Brown         178 Marvel C~ <NA>
## # ... with 724 more rows, and 2 more variables: Alignment <chr>, Weight <dbl>
```

6.7.2 Перестановка колонок: `dplyr::relocate()`

Для изменения порядка колонок можно использовать функцию `relocate()`. Она тоже работает похожим образом на `select()` и `rename()`[11]. Как и `rename()`, функция `relocate()` не выкидывает неиспользованные колонки:

```
heroes %>%
  relocate(Publisher)
```

```
## # A tibble: 734 x 11
##    Publisher    X1 name  Gender `Eye color` Race  `Hair color` Height
##    <chr>     <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl>
##  1 Marvel C~     0 A-Bo~ Male   yellow      Human No Hair         203
##  2 Dark Hor~     1 Abe ~ Male   blue        Icth~ No Hair         191
##  3 DC Comics     2 Abin~ Male   blue        Unga~ No Hair         185
##  4 Marvel C~     3 Abom~ Male   green       Huma~ No Hair         203
##  5 Marvel C~     4 Abra~ Male   blue        Cosm~ Black            NA
##  6 Marvel C~     5 Abso~ Male   blue        Human No Hair         193
##  7 NBC - He~     6 Adam~ Male   blue        <NA>  Blond            NA
##  8 DC Comics     7 Adam~ Male   blue        Human Blond           185
##  9 Marvel C~     8 Agen~ Female blue        <NA>  Blond           173
## 10 Marvel C~     9 Agen~ Male   brown       Human Brown           178
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

При этом `relocate()` имеет дополнительные параметры `.after =` и `.before =`, которые позволяют выбирать, куда поместить выбранные колонки.

```
heroes %>%
  relocate(Publisher, .after = name)
```

```
## # A tibble: 734 x 11
##       X1 name  Publisher Gender `Eye color` Race  `Hair color` Height
##    <dbl> <chr> <chr>     <chr>  <chr>       <chr> <chr>         <dbl>
##  1     0 A-Bo~ Marvel C~ Male   yellow      Human No Hair         203
##  2     1 Abe ~ Dark Hor~ Male   blue        Icth~ No Hair         191
##  3     2 Abin~ DC Comics Male   blue        Unga~ No Hair         185
##  4     3 Abom~ Marvel C~ Male   green       Huma~ No Hair         203
##  5     4 Abra~ Marvel C~ Male   blue        Cosm~ Black            NA
##  6     5 Abso~ Marvel C~ Male   blue        Human No Hair         193
##  7     6 Adam~ NBC - He~ Male   blue        <NA>  Blond            NA
##  8     7 Adam~ DC Comics Male   blue        Human Blond           185
##  9     8 Agen~ Marvel C~ Female blue        <NA>  Blond           173
## 10     9 Agen~ Marvel C~ Male   brown       Human Brown           178
```

[11] `relocate()` не позволяет переименовывать колонки в отличие от `select()` и `rename()`

```
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

`relocate()` очень хорошо работает в сочетании с выбором колонок с помощью tidyselect. Например, можно передвинуть в одно место все колонки с одним типом данных:

```
heroes %>%
  relocate(Publisher, where(is.numeric), .after = name)
```

```
## # A tibble: 734 x 11
##    name  Publisher    X1 Height Weight Gender `Eye color` Race  `Hair color`
##    <chr> <chr>     <dbl>  <dbl>  <dbl> <chr>  <chr>       <chr> <chr>
##  1 A-Bo~ Marvel C~     0    203    441 Male   yellow      Human No Hair
##  2 Abe ~ Dark Hor~     1    191     65 Male   blue        Icth~ No Hair
##  3 Abin~ DC Comics     2    185     90 Male   blue        Unga~ No Hair
##  4 Abom~ Marvel C~     3    203    441 Male   green       Huma~ No Hair
##  5 Abra~ Marvel C~     4     NA     NA Male   blue        Cosm~ Black
##  6 Abso~ Marvel C~     5    193    122 Male   blue        Human No Hair
##  7 Adam~ NBC - He~     6     NA     NA Male   blue        <NA>  Blond
##  8 Adam~ DC Comics     7    185     88 Male   blue        Human Blond
##  9 Agen~ Marvel C~     8    173     61 Female blue        <NA>  Blond
## 10 Agen~ Marvel C~     9    178     81 Male   brown       Human Brown
## # ... with 724 more rows, and 2 more variables: `Skin color` <chr>,
## #   Alignment <chr>
```

Последняя важная функция для выбора колонок — `pull()`. Эта функция делает то же самое, что и индексирование с помощью $, т.е. вытаскивает из тиббла вектор с выбранным названием. Это лучше вписывается в логику tidyverse, поскольку позволяет извлечь колонку из тиббла с использованием пайпа:

```
heroes %>%
  select(Height) %>%
  pull() %>%
  head()
```

```
## [1] 203 191 185 203  NA 193
```

```
heroes %>%
  pull(Height) %>%
  head()
```

```
## [1] 203 191 185 203  NA 193
```

У функции `pull()` есть аргумент `name =`, который позволяет создать проименованный вектор:

```
heroes %>%
  pull(Height, name) %>%
  head()
```

```
##        A-Bomb    Abe Sapien      Abin Sur   Abomination       Abraxas
##           203           191           185           203            NA
## Absorbing Man
##           193
```

В отличие от базового R, tidyverse нигде не сокращает имплицитно результат вычислений до вектора, поэтому функция `pull()` - это основной способ извлечения колонки из тиббла как вектора.

6.8 Работа со строками тиббла

6.8.1 Выбор строк по номеру: `dplyr::slice()`

Начнем с выбора строк. Функция `dplyr::slice()` выбирает строчки по их числовому индексу.

```
heroes %>%
  slice(1:3)
```

```
## # A tibble: 3 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
## 2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
## 3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

6.8.2 Выбор строк по условию: `dplyr::filter()`

Функция `dplyr::filter()` делает то же самое, что и `slice()`, но уже по условию. Причем для условий нужно использовать не векторы из тиббла, а название колонок (без кавычек) как будто бы они были переменными в окружении.

```
heroes %>%
  filter(Publisher == "DC Comics")
```

```
## # A tibble: 215 x 11
```

```
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  2     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  3    13 Alan~ Male   blue        <NA>  Blond           180 DC Comics
##  4    16 Alfr~ Male   blue        Human Black           178 DC Comics
##  5    19 Amazo Male   red         Andr~ <NA>            257 DC Comics
##  6    27 Anim~ Male   blue        Human Blond           183 DC Comics
##  7    31 Anti~ Male   yellow      God ~ No Hair          61 DC Comics
##  8    35 Aqua~ Male   blue        <NA>  Blond            NA DC Comics
##  9    36 Aqua~ Male   blue        Atla~ Black           178 DC Comics
## 10    37 Aqua~ Male   blue        Atla~ Blond           185 DC Comics
## # ... with 205 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

6.8.3 Семейство функций `slice()`

У функции `slice()` есть множество родственников, которые объединяют функционал обычного `slice()` и `filter()`. Например, с помощью функций `dplyr::slice_max()` и `dplyr::slice_min()` можно выбрать заданное количество строк, содержащих наибольшие или наименьшие значения по колонке соответственно:

```
heroes %>%
  slice_max(Weight, n = 3)
```

```
## # A tibble: 3 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   575 Sasq~ Male   red         <NA>  Orange          305 Marvel C~
## 2   373 Jugg~ Male   blue        Human Red             287 Marvel C~
## 3   203 Dark~ Male   red         New ~ No Hair         267 DC Comics
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

```
heroes %>%
  slice_min(Weight, n = 3)
```

```
## # A tibble: 3 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   346 Iron~ Male   blue        <NA>  No Hair          NA Marvel C~
## 2   302 Groot Male   yellow      Flor~ <NA>            701 Marvel C~
## 3   350 Jack~ Male   blue        Human Brown            71 Dark Hor~
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Функция `slice_sample()` позволяет выбирать заданное количество случайных строчек:

```
heroes %>%
  slice_sample(n = 3)
```

```
## # A tibble: 3 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   213 Dead~ Male   brown       Human Brown           185 DC Comics
## 2    38 Arac~ Female blue        Human Blond           175 Marvel C~
## 3   578 Scar~ Male   blue        Human Brown           183 DC Comics
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Или же долю строчек:

```
heroes %>%
  slice_sample(prop = .01)
```

```
## # A tibble: 7 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   272 Gala~ Male   black       Cosm~ Black           876 Marvel C~
## 2    54 Atom~ Male   brown       <NA>  Black            NA DC Comics
## 3   543 Raph~ Male   <NA>        Muta~ No Hair          NA IDW Publ~
## 4   649 Sylar Male   <NA>        <NA>  <NA>             NA NBC - He~
## 5   190 Crys~ Female green       Inhu~ Red             168 Marvel C~
## 6   269 Fran~ <NA>   blue        <NA>  Grey            188 Marvel C~
## 7   168 Cham~ Male   brown       Muta~ Brown           175 Marvel C~
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

Если поставить значение параметра `prop` = равным 1, то таким образом можно перемешать порядок строчек в тиббле:

```
heroes %>%
  slice_sample(prop = 1)
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1   304 Guy ~ Male   blue        Huma~ Red             188 DC Comics
##  2   528 Prof~ Male   blue        Muta~ No Hair         183 Marvel C~
##  3   634 Star~ Female green       Tama~ Auburn          193 DC Comics
##  4   627 Spid~ Female brown       <NA>  Brown           173 Marvel C~
##  5   183 Colo~ Male   <NA>        <NA>  <NA>             NA DC Comics
```

```
##  6   245 Etri~ Male   red         Demon No Hair         193 DC Comics
##  7   409 Livi~ <NA>   yellow      <NA>  <NA>            198 Marvel C~
##  8   462 Mock~ Female blue        Human Blond           175 Marvel C~
##  9   293 Gori~ Male   yellow      Gori~ Black           198 DC Comics
## 10   152 Capt~ Male   blue        <NA>  Brown           188 Team Epi~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

6.8.4 Удаление строчек с NA: `tidyr::drop_na()`

Если нужно выбрать только строчки без пропущенных значений, то можно воспользоваться удобной функцией `tidyr::drop_na()`.

```
heroes %>%
  drop_na()
```

```
## # A tibble: 50 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  2     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  3    34 Apoc~ Male   red         Muta~ Black           213 Marvel C~
##  4    39 Arch~ Male   blue        Muta~ Blond           183 Marvel C~
##  5    41 Ardi~ Female white       Alien Orange          193 Marvel C~
##  6    56 Azaz~ Male   yellow      Neya~ Black           183 Marvel C~
##  7    74 Beast Male   blue        Muta~ Blue            180 Marvel C~
##  8    75 Beas~ Male   green       Human Green           173 DC Comics
##  9    92 Biza~ Male   black       Biza~ Black           191 DC Comics
## 10   108 Blac~ Male   red         Demon White           191 Marvel C~
## # ... with 40 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Можно выбрать колонки, наличие `NA` в которых будет приводить к удалению соответствующих строчек (не затрагивая другие строчки, в которых есть `NA` в остальных столбцах).

```
heroes %>%
  drop_na(Weight)
```

```
## # A tibble: 495 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
```

```
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  6     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  7     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
##  8     9 Agen~ Male   brown       Human Brown           178 Marvel C~
##  9    10 Agen~ Male   <NA>        <NA>  <NA>            191 Marvel C~
## 10    11 Air-~ Male   blue        <NA>  White           188 Marvel C~
## # ... with 485 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Для выбора колонок в `drop_na()` используется tidyselect, с которым мы недавно познакомились (6.7).

6.8.5 Сортировка строк: `dplyr::arrange()`

Функция `dplyr::arrange()` сортирует строчки от меньшего к большему (или по алфавиту - для текстовых значений) по выбранной колонке.

```
heroes %>%
  arrange(Weight)
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1   346 Iron~ Male   blue        <NA>  No Hair          NA Marvel C~
##  2   302 Groot Male   yellow      Flor~ <NA>            701 Marvel C~
##  3   350 Jack~ Male   blue        Human Brown            71 Dark Hor~
##  4   272 Gala~ Male   black       Cosm~ Black           876 Marvel C~
##  5   731 Yoda  Male   brown       Yoda~ White            66 George L~
##  6   255 Fin ~ Male   red         Kaka~ No Hair         975 Marvel C~
##  7   330 Howa~ Male   brown       <NA>  Yellow           79 Marvel C~
##  8   396 Kryp~ Male   blue        Kryp~ White            64 DC Comics
##  9   568 Rock~ Male   brown       Anim~ Brown           122 Marvel C~
## 10   208 Dash  Male   blue        Human Blond           122 Dark Hor~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Чтобы отсортировать в обратном порядке, воспользуйтесь функцией `desc()`.

```
heroes %>%
  arrange(desc(Weight))
```

```
## # A tibble: 734 x 11
```

```
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1   575 Sasq~ Male   red         <NA>  Orange        305   Marvel C~
##  2   373 Jugg~ Male   blue        Human Red           287   Marvel C~
##  3   203 Dark~ Male   red         New ~ No Hair       267   DC Comics
##  4   283 Giga~ Female green       <NA>  Red            62.5 DC Comics
##  5   331 Hulk  Male   green       Huma~ Green         244   Marvel C~
##  6   549 Red ~ Male   yellow      Huma~ Black         213   Marvel C~
##  7   119 Bloo~ Female blue        Human Brown         218   Marvel C~
##  8   718 Wolf~ Female green       <NA>  Auburn        366   Marvel C~
##  9   657 Than~ Male   red         Eter~ No Hair       201   Marvel C~
## 10     0 A-Bo~ Male   yellow      Human No Hair       203   Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Можно сортировать по нескольким колонкам сразу. В таких случаях удобно в качестве первой переменной выбирать переменную, обозначающую принадлежность к группе, а в качестве второй — континуальную числовую переменную:

```
heroes %>%
  arrange(Gender, desc(Weight))
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1   283 Giga~ Female green       <NA>  Red            62.5 DC Comics
##  2   119 Bloo~ Female blue        Human Brown         218   Marvel C~
##  3   718 Wolf~ Female green       <NA>  Auburn        366   Marvel C~
##  4   591 She-~ Female green       Human Green         201   Marvel C~
##  5   320 Hela  Female green       Asga~ Black         213   Marvel C~
##  6   686 Valk~ Female blue        <NA>  Blond         191   Marvel C~
##  7   596 Sif   Female blue        Asga~ Black         188   Marvel C~
##  8   271 Frig~ Female blue        <NA>  White         180   Marvel C~
##  9   667 Thun~ Female green       <NA>  Red           218   Marvel C~
## 10   592 She-~ Female blue        Huma~ No Hair       183   Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

6.9 Создание колонок: `dplyr::mutate()` и `dplyr::transmute()`

Функция `dplyr::mutate()` позволяет создавать новые колонки в тиббле.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  select(name, imt) %>%
  arrange(desc(imt))
```

```
## # A tibble: 734 x 2
##    name          imt
##    <chr>       <dbl>
##  1 Utgard-Loki 2510.
##  2 Giganta     1613.
##  3 Red Hulk     139.
##  4 Darkseid     115.
##  5 Machine Man  114.
##  6 Thanos       110.
##  7 Destroyer    108.
##  8 A-Bomb       107.
##  9 Abomination  107.
## 10 Hulk         106.
## # ... with 724 more rows
```

`dplyr::transmute()` - это аналог `mutate()`, который не только создает новые колонки, но и сразу же выкидывает все старые:

```
heroes %>%
  transmute(imt = Weight/(Height/100)^2)
```

```
## # A tibble: 734 x 1
##      imt
##    <dbl>
##  1 107.
##  2  17.8
##  3  26.3
##  4 107.
##  5  NA
##  6  32.8
##  7  NA
##  8  25.7
##  9  20.4
## 10  25.6
## # ... with 724 more rows
```

Внутри `mutate()` и `transmute()` мы можем использовать либо векторизованные операции (длина новой колонки должна равняться длине датафрейма), либо операции, которые возвращают одно значение. В последнем случае значение будет одинаковым на всю колонку, т.е. будет работать правило ресайклинга

(2.8.3):

```
heroes %>%
  transmute(name, weight_mean = mean(Weight, na.rm = TRUE))
```

```
## # A tibble: 734 x 2
##    name          weight_mean
##    <chr>               <dbl>
##  1 A-Bomb               112.
##  2 Abe Sapien           112.
##  3 Abin Sur             112.
##  4 Abomination          112.
##  5 Abraxas              112.
##  6 Absorbing Man        112.
##  7 Adam Monroe          112.
##  8 Adam Strange         112.
##  9 Agent 13             112.
## 10 Agent Bob            112.
## # ... with 724 more rows
```

Однако в функциях `mutate()` и `transmute()` правило ресайклинга не будет работать в остальных случаях: если полученный вектор будет не равен 1 или длине датафрейма, то мы получим ошибку.

```
heroes %>%
  mutate(one_and_two = 1:2)
```

```
## Error: Problem with `mutate()` input `one_and_two`.
## x Input `one_and_two` can't be recycled to size 734.
## i Input `one_and_two` is `1:2`.
## i Input `one_and_two` must be size 734 or 1, not 2.
```

Это не баг, а фича: авторы пакета `dplyr` считают, что ресайклинг кратных друг другу векторов — это слишком удобное место для выстрелов себе в ногу. Поэтому в таких случаях разработчики `dplyr` рекомендуют использовать функцию `rep()`, знакомую нам уже очень давно (??atomic)).

```
heroes %>%
  mutate(one_and_two = rep(1:2, length.out = nrow(.)))
```

```
## # A tibble: 734 x 12
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
```

```
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 4 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>, one_and_two <int>
```

6.10 Агрегация данных в тиббле

Аггрегация по группам - это очень часто возникающая задача, например, это может использоваться для усреднения данных по испытуемым или условиям. Сделать аггрегацию в датафрейме удобной Хэдли Уикхэм пытался еще в предшественнике dplyr, пакете plyr. dplyr позволяет делать аггрегацию очень симпатичным и понятным способом. Аггрегация в dplyr состоит из двух этапов: группировки (group_by()) и подытоживания (summarise()). Начнем с последнего.

Функция dplyr::summarise()[12] позволяет аггрегировать данные в тиббле. Работает она очень похоже на mutate(), но если внутри mutate() используются векторизованные функции, возвращающие вектор такой же длины, что и колонки, использовавшиеся для расчетов, то в summarise() используются функции, которые возвращают вектор длиной 1. Например, min(), mean(), max() и т.д. Можно создавать несколько колонок через запятую (это работает и для mutate()).

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  summarise(min(imt, na.rm = TRUE),
            max(imt, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   `min(imt, na.rm = TRUE)` `max(imt, na.rm = TRUE)`
##                      <dbl>                    <dbl>
## 1                   0.0814                    2510.
```

В dplyr есть дополнительные суммирующие функции для более удобного индексирования в стиле tidyverse. Например, функции dplyr::nth(), dplyr::first() и dplyr::last(), которые позволяют вытаскивать значения из вектора по индексу (что-то вроде slice(), но для векторов)

[12] У функции dplyr::summarise() есть синоним dplyr::summarize(), которая делает абсолбтно то же самое. Просто потому что в американском английском и британском английском это слово пишется по-разному.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  arrange(imt) %>%
  summarise(first = first(imt),
            tenth = nth(imt, 10),
            last = last(imt))
```

```
## # A tibble: 1 x 3
##    first tenth  last
##    <dbl> <dbl> <dbl>
## 1 0.0814  16.7    NA
```

В отличие от `mutate()`, функции внутри `summarise()` вполне позволяют функциям внутри возвращать вектор из нескольких значений, создавая тиббл такой же длины, как и получившийся вектор.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  arrange(imt) %>%
  summarise(imt_range = range(imt, na.rm = TRUE)) #     range()                    :
```

```
## # A tibble: 2 x 1
##   imt_range
##       <dbl>
## 1    0.0814
## 2 2510.
```

`group_by()` - это функция для группировки данных в тиббле по дискретной переменной для дальнейшей аггрегации с помощью `summarise()`. После применения `group_by()` тиббл будет выглядеть так же, но у него появятся атрибут `groups`[13]:

```
heroes %>%
  group_by()
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
```

[13]Снять группировку можно с помощью функции `ungroup()`.

```
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Если после этого применить на тиббле функцию `summarise()`, то мы получим не тиббл длиной один, а тиббл со значением для каждой из групп.

```
heroes %>%
  mutate(imt = Weight/(Height/100)^2) %>%
  group_by(Gender) %>%
  summarise(min(imt, na.rm = TRUE),
            max(imt, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 3
##   Gender `min(imt, na.rm = TRUE)` `max(imt, na.rm = TRUE)`
##   <chr>                     <dbl>                     <dbl>
## 1 Female                  15.5                      1613.
## 2 Male                     0.0814                   2510.
## 3 <NA>                    16.3                       114.
```

6.11 Подсчет строк: `dplyr::n()`, `dplyr::count()`

Для подсчет количества значений можно воспользоваться функцией `n()`.

```
heroes %>%
  group_by(Gender) %>%
  summarise(n = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   Gender     n
##   <chr>  <int>
## 1 Female   200
## 2 Male     505
## 3 <NA>      29
```

Функция `n()` вместе с `group_by()` внутри `filter()` позволяет удобным образом “отрезать” от тиббла редкие группы...

```
heroes %>%
  group_by(Race) %>%
  filter(n() > 10) %>%
  select(name, Race)
```

```
## # A tibble: 611 x 2
## # Groups:   Race [6]
##    name          Race
##    <chr>         <chr>
##  1 A-Bomb        Human
##  2 Abomination   Human / Radiation
##  3 Absorbing Man Human
##  4 Adam Monroe   <NA>
##  5 Adam Strange  Human
##  6 Agent 13      <NA>
##  7 Agent Bob     Human
##  8 Agent Zero    <NA>
##  9 Air-Walker    <NA>
## 10 Ajax          Cyborg
## # ... with 601 more rows
```

или же наоборот, выделить только маленькие группы:

```
heroes %>%
  group_by(Race) %>%
  filter(n() == 1) %>%
  select(name, Race)
```

```
## # A tibble: 34 x 2
## # Groups:   Race [34]
##    name          Race
##    <chr>         <chr>
##  1 Abe Sapien    Icthyo Sapien
##  2 Abin Sur      Ungaran
##  3 Alien         Xenomorph XX121
##  4 Azazel        Neyaphem
##  5 Bizarro       Bizarro
##  6 Boba Fett     Human / Clone
##  7 Darth Maul    Dathomirian Zabrak
##  8 Fin Fang Foom Kakarantharaian
##  9 Gamora        Zen-Whoberian
## 10 Gladiator     Strontian
## # ... with 24 more rows
```

Таблицу частот можно создать без `group_by()` и `summarise(n = n())`. Функция

`count()` заменяет эту конструкцию:

```
heroes %>%
  count(Gender)
```

```
## # A tibble: 3 x 2
##   Gender     n
##   <chr>  <int>
## 1 Female   200
## 2 Male     505
## 3 <NA>      29
```

Эту таблицу частот удобно сразу проранжировать, указав в параметре `sort =` значение `TRUE`.

```
heroes %>%
  count(Gender, sort = TRUE)
```

```
## # A tibble: 3 x 2
##   Gender     n
##   <chr>  <int>
## 1 Male     505
## 2 Female   200
## 3 <NA>      29
```

> Функция `count()`, несмотря на свою простоту, является одной из наиболее используемых в tidyverse.

6.12 Уникальные значения: `dplyr::distinct()`

`dplyr::distinct()` - это более быстрый аналог `unique()`, позволяет извлекать уникальные значения для одной или нескольких колонок.

```
heroes %>%
  distinct(Gender)
```

```
## # A tibble: 3 x 1
##   Gender
##   <chr>
## 1 Male
## 2 Female
## 3 <NA>
```

```
heroes %>%
  distinct(Gender, Race)
```

```
## # A tibble: 81 x 2
##    Gender Race
##    <chr>  <chr>
##  1 Male   Human
##  2 Male   Icthyo Sapien
##  3 Male   Ungaran
##  4 Male   Human / Radiation
##  5 Male   Cosmic Entity
##  6 Male   <NA>
##  7 Female <NA>
##  8 Male   Cyborg
##  9 Male   Xenomorph XX121
## 10 Male   Android
## # ... with 71 more rows
```

Иногда нужно аггрегировать данные, но при этом сохранить исходную структуру тиббла. Например, нужно посчитать размер групп или посчитать средние значения по группе для последующего сравнения с индивидуальными значениями. В tidyverse это можно сделать с помощью сочетания `group_by()` и `mutate()` (вместо `summarise()`):

```
heroes %>%
  group_by(Race) %>%
  mutate(Gender_n = n()) %>%
  select(Race, name, Gender, Gender_n)
```

```
## # A tibble: 734 x 4
## # Groups:   Race [62]
##    Race              name          Gender Gender_n
##    <chr>             <chr>         <chr>     <int>
##  1 Human             A-Bomb        Male        208
##  2 Icthyo Sapien     Abe Sapien    Male          1
##  3 Ungaran           Abin Sur      Male          1
##  4 Human / Radiation Abomination   Male         11
##  5 Cosmic Entity     Abraxas       Male          4
##  6 Human             Absorbing Man Male        208
##  7 <NA>              Adam Monroe   Male        304
##  8 Human             Adam Strange  Male        208
##  9 <NA>              Agent 13      Female      304
## 10 Human             Agent Bob     Male        208
## # ... with 724 more rows
```

Результаты аггрегации были записаны в отдельную колонку, при этом значения этой колонки внутри одной группы повторяются.

6.13 Трансформация нескольких колонок: `dplyr::across()`

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

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
##
## -- Column specification --------------------------------
## cols(
##   X1 = col_double(),
##   name = col_character(),
##   Gender = col_character(),
##   `Eye color` = col_character(),
##   Race = col_character(),
##   `Hair color` = col_character(),
##   Height = col_double(),
##   Publisher = col_character(),
##   `Skin color` = col_character(),
##   Alignment = col_character(),
##   Weight = col_double()
## )
```

```
heroes
```

```
## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <chr>,
## #   Alignment <chr>, Weight <dbl>
```

Допустим, вы хотите посчитать среднюю массу и рост, группируя по полу супергероев. Можно посчитать это внутри одного `summarise()`, используя запятую:

```
heroes %>%
  group_by(Gender) %>%
  summarise(height = mean(Height),
            weight = mean(Weight))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 3
##   Gender height weight
##   <chr>   <dbl>  <dbl>
## 1 Female     NA     NA
## 2 Male       NA     NA
## 3 <NA>       NA     NA
```

Если таких колонок будет много, то это уже станет сильно неудобным, нам придется много копировать код, а это чревато ошибками и очень скучно.

Поэтому в `dplyr` есть функция для операций над несколькими колонками сразу: `dplyr::across()`[14]. Эта функция работает похожим образом на функции семейства `apply()` и использует tidyselect для выбора колонок.

Таким образом, конструкции с функцией `across()` можно разбить на три части:

1. Выбор колонок с помощью tidyselect. Здесь работают все те приемы, которые мы изучили при выборе колонок (6.7).
2. Собственно применение функции `across()`. Первый аргумент `.col` — колонки, выбранные на первом этапе с помощью tidyselect, по умолчанию это `everything()`, т.е. все колонки. Второй аргумент `.fns` — это функция или целый список из функций, которые будут применены к выбранным колонкам. Если функции требуют дополнительных аргументов, то они могут быть перечислены внутри `across()`.
3. Использование `summarise()` или другой функции `dplyr`. В этом случае в качестве аргумента для функции используется результат работы функции `across()`.

Вот такой вот бутерброд выходит. Давайте посмотрим, как это работает на практике и посчитаем среднее значение по колонкам `Height` и `Weight`.

[14] Функция `across()` появилась в пакете `dplyr` относительно недавно, до этого для работы с множественными колонками в tidyverse использовались многочисленные функции `*_at()`, `*_if()`, `*_all()`, например, `summarise_at()`, `summarise_if()`, `summarize_all()`. Эти функции до сих пор присутствуют в `dplyr`, но считаются устаревшими. Другая альтернатива - использование пакета `purrr` (**??**) или семейства функций `apply()`(@ref(apply_f)).

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height, Weight), mean))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 3
##   Gender Height Weight
##   <chr>   <dbl>  <dbl>
## 1 Female     NA     NA
## 2 Male       NA     NA
## 3 <NA>       NA     NA
```

Здесь мы столкнулись с уже известной нам проблемой: функция `mean()` при столкновении хотя бы с одним `NA` будет возвращать `NA`, если мы не изменим параметр `na.rm =`. Как и в случае с функциями семейства `apply()` (@ref(apply_f)), дополнительные параметры для функции можно перечислить через запятую после самой функции:

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height, Weight), mean, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 3
##   Gender Height Weight
##   <chr>   <dbl>  <dbl>
## 1 Female   175.   78.8
## 2 Male     192.  126.
## 3 <NA>     177.  129.
```

До этого мы просто использовали выбор колонок по их названию. Но именно внутри `across()` использование tidyselect раскрывается как удивительно элегантный и мощный инструмент. Например, можно посчитать среднее для всех numeric колонок:

```
heroes %>%
  drop_na(Height, Weight) %>%
  group_by(Gender) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 4
##   Gender    X1 Height Weight
```

```
##   <chr>  <dbl>  <dbl>  <dbl>
## 1 Female  394.   174.   78.3
## 2 Male    369.   193.  126.
## 3 <NA>    375.   182   129.
```

Или длину строк для строковых колонок. Для этого нам понадобится вспомнить, как создавать анонимные функции (@ref(anon_f)).

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(where(is.character),
                   function(x) mean(nchar(x), na.rm = TRUE)))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 8
##   Gender  name `Eye color`  Race `Hair color` Publisher `Skin color` Alignment
##   <chr>  <dbl>       <dbl> <dbl>        <dbl>     <dbl>        <dbl>     <dbl>
## 1 Female  9.04        4.68  6.42         5.05      11.5         4.57      3.88
## 2 Male    9.05        4.53  6.75         5.48      11.4         5.02      3.78
## 3 <NA>    9.48        5.16 10.1          6.44      11.9         4         3.96
```

Или же даже посчитать и то, и другое внутри одного `summarise()`!

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE),
            across(where(is.character),
                   function(x) mean(nchar(x), na.rm = TRUE)))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 11
##   Gender    X1 Height Weight  name `Eye color`  Race `Hair color` Publisher
##   <chr>  <dbl>  <dbl>  <dbl> <dbl>       <dbl> <dbl>        <dbl>     <dbl>
## 1 Female  395.   175.   78.8  9.04        4.68  6.42         5.05      11.5
## 2 Male    357.   192.  126.   9.05        4.53  6.75         5.48      11.4
## 3 <NA>    329    177.  129.   9.48        5.16 10.1          6.44      11.9
## # ... with 2 more variables: `Skin color` <dbl>, Alignment <dbl>
```

Внутри одного `across()` можно применить не одну функцию к каждой из выбранных колонок, а сразу несколько функций для каждой из колонок. Для этого нам нужно использовать список функций (желательно - проименованный).

```
heroes %>%
  group_by(Gender) %>%
```

```
  summarise(across(c(Height, Weight),
                   list(min = min,
                        mean = mean,
                        max = max),
                   na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 7
##   Gender Height_min Height_mean Height_max Weight_min Weight_mean Weight_max
##   <chr>       <dbl>       <dbl>      <dbl>      <dbl>       <dbl>      <dbl>
## 1 Female       62.5        175.        366         41        78.8        630
## 2 Male         15.2        192.        975          2       126.         900
## 3 <NA>        108          177.        198         39       129.         383
```

Вот нам и понадобился список функций (@ref(functions_objects))!

```
heroes %>%
  group_by(Gender) %>%
  summarise(across(c(Height, Weight),
                   list(min = function(x) min(x, na.rm = TRUE),
                        mean = function(x) mean(x, na.rm = TRUE),
                        max = function(x) max(x, na.rm = TRUE),
                        na_n = function(x, ...) sum(is.na(x)))
                   )
            )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 9
##   Gender Height_min Height_mean Height_max Height_na_n Weight_min Weight_mean
##   <chr>       <dbl>       <dbl>      <dbl>       <int>      <dbl>       <dbl>
## 1 Female       62.5        175.        366          56         41        78.8
## 2 Male         15.2        192.        975         147          2       126.
## 3 <NA>        108          177.        198          14         39       129.
## # ... with 2 more variables: Weight_max <dbl>, Weight_na_n <int>
```

Хотя основное применение функции `across()` — это массовое подытоживание с помощью `summarise()`, `across()` можно использовать и с другими функциями `dplyr`. Например, можно делать массовые операции с колонками с помощью `mutate()`:

```
heroes %>%
  mutate(across(is.character, as.factor))
```

```
## Warning: Problem with `mutate()` input `..1`.
## i Predicate functions must be wrapped in `where()`.
##
##   # Bad
##   data %>% select(is.character)
##
##   # Good
##   data %>% select(where(is.character))
##
## i Please update your code.
## This message is displayed once per session.
## i Input `..1` is `across(is.character, as.factor)`.

## Warning: Predicate functions must be wrapped in `where()`.
##
##   # Bad
##   data %>% select(is.character)
##
##   # Good
##   data %>% select(where(is.character))
##
## i Please update your code.
## This message is displayed once per session.

## # A tibble: 734 x 11
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <fct> <fct>  <fct>       <fct> <fct>         <dbl> <fct>
##  1     0 A-Bo~ Male   yellow      Human No Hair         203 Marvel C~
##  2     1 Abe ~ Male   blue        Icth~ No Hair         191 Dark Hor~
##  3     2 Abin~ Male   blue        Unga~ No Hair         185 DC Comics
##  4     3 Abom~ Male   green       Huma~ No Hair         203 Marvel C~
##  5     4 Abra~ Male   blue        Cosm~ Black            NA Marvel C~
##  6     5 Abso~ Male   blue        Human No Hair         193 Marvel C~
##  7     6 Adam~ Male   blue        <NA>  Blond            NA NBC - He~
##  8     7 Adam~ Male   blue        Human Blond           185 DC Comics
##  9     8 Agen~ Female blue        <NA>  Blond           173 Marvel C~
## 10     9 Agen~ Male   brown       Human Brown           178 Marvel C~
## # ... with 724 more rows, and 3 more variables: `Skin color` <fct>,
## #   Alignment <fct>, Weight <dbl>
```

Менее очевидный способ применения `across()` - использование `across()` внутри `count()` вместе с функцией `n_distinct()`, которая считает количество уникальных значений в векторе. Это позволяет посмотреть таблицу частот для группирующих переменных:

```
heroes %>%
  count(across(where(function(x) n_distinct(x) <= 6)))
```

```
## # A tibble: 11 x 3
##    Gender Alignment     n
##    <chr>  <chr>     <int>
##  1 Female bad          35
##  2 Female good        161
##  3 Female neutral       4
##  4 Male   bad         165
##  5 Male   good        316
##  6 Male   neutral      18
##  7 Male   <NA>          6
##  8 <NA>   bad           7
##  9 <NA>   good         19
## 10 <NA>   neutral       2
## 11 <NA>   <NA>          1
```

6.14 Соединение датафреймов: bind_rows(), bind_cols()

Для начала создадим следующие тибблы и сохраним их как `dc`, `marvel` и `other_publishers`:

```
dc <- heroes %>%
  filter(Publisher == "DC Comics") %>%
  group_by(Gender) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
dc
```

```
## # A tibble: 3 x 2
##   Gender weight_mean
##   <chr>        <dbl>
## 1 Female        76.8
## 2 Male         113.
## 3 <NA>         NaN
```

```
marvel <- heroes %>%
  filter(Publisher == "Marvel Comics") %>%
```

```
  group_by(Gender) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
marvel
```

```
## # A tibble: 3 x 2
##   Gender weight_mean
##   <chr>        <dbl>
## 1 Female        80.1
## 2 Male         134.
## 3 <NA>         129.
```

```
other_publishers <- heroes %>%
  filter(!(Publisher %in% c("DC Comics","Marvel Comics"))) %>%
  group_by(Gender) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
other_publishers
```

```
## # A tibble: 3 x 2
##   Gender weight_mean
##   <chr>        <dbl>
## 1 Female        70.8
## 2 Male         111.
## 3 <NA>         NaN
```

Несколько тибблов можно объединить вертикально с помощью функции `bind_rows()`. Для корректного объединения тибблы должны иметь одинаковые названия колонок.

```
bind_rows(dc, marvel)
```

```
## # A tibble: 6 x 2
##   Gender weight_mean
##   <chr>        <dbl>
## 1 Female        76.8
## 2 Male         113.
## 3 <NA>         NaN
## 4 Female        80.1
```

```
## 5 Male         134.
## 6 <NA>         129.
```

Чтобы соединить тибблы горизонтально, воспользуйтесь функцией `bind_cols()`.

```
bind_cols(dc, marvel)
```

```
## New names:
## * Gender -> Gender...1
## * weight_mean -> weight_mean...2
## * Gender -> Gender...3
## * weight_mean -> weight_mean...4
```

```
## # A tibble: 3 x 4
##   Gender...1 weight_mean...2 Gender...3 weight_mean...4
##   <chr>                <dbl> <chr>                <dbl>
## 1 Female                76.8 Female                80.1
## 2 Male                 113.  Male                 134.
## 3 <NA>                 NaN   <NA>                 129.
```

Функции `bind_rows()` и `bind_cols()` могут работать не только с двумя, но сразу с несколькими датафреймами.

```
bind_rows(dc, marvel, other_publishers)
```

```
## # A tibble: 9 x 2
##   Gender weight_mean
##   <chr>        <dbl>
## 1 Female        76.8
## 2 Male         113.
## 3 <NA>         NaN
## 4 Female        80.1
## 5 Male         134.
## 6 <NA>         129.
## 7 Female        70.8
## 8 Male         111.
## 9 <NA>         NaN
```

На входе в функции `bind_rows()` и `bind_cold()` можно подавать как сами датафреймы или тибблы через запятую, так и список из датафреймов/тибблов.

```
heroes_list_of_df <- list(DC = dc,
                          Marvel = marvel,
                          Other = other_publishers)
bind_rows(heroes_list_of_df)
```

```
## # A tibble: 9 x 2
##   Gender weight_mean
##   <chr>        <dbl>
## 1 Female        76.8
## 2 Male         113.
## 3 <NA>         NaN
## 4 Female        80.1
## 5 Male         134.
## 6 <NA>         129.
## 7 Female        70.8
## 8 Male         111.
## 9 <NA>         NaN
```

Чтобы не потерять, из какого датафрейма какие данные, можно указать любое строковое значение (название будущей колонки) для необязательного аргумента `.id =`.

```
bind_rows(heroes_list_of_df, .id = "Publisher")
```

```
## # A tibble: 9 x 3
##   Publisher Gender weight_mean
##   <chr>     <chr>        <dbl>
## 1 DC        Female        76.8
## 2 DC        Male         113.
## 3 DC        <NA>         NaN
## 4 Marvel    Female        80.1
## 5 Marvel    Male         134.
## 6 Marvel    <NA>         129.
## 7 Other     Female        70.8
## 8 Other     Male         111.
## 9 Other     <NA>         NaN
```

`bind_rows()` обычно используется, когда ваши данные находятся в разных файлах с одинаковой структурой. Тогда вы можете прочитать все таблицы в папке, сохранить их в качестве списка из датафреймов и объединить в один датафрейм с помощью `bind_rows()`.

6.15 Соединение датафреймов: *_join

В реальности иногда возникает ситуация, когда нужно соединить две таблички, у которых есть общий столбец (или несколько столбцов), но все остальные столбцы различаются. Эти две таблички нужно объединить (*join*). Эта задача обычно возникает не очень часто, обычно это происходит один-два раза в одном проекте, когда нужно дополнить имеющиеся данные дополнительной информацией

извне или объединить два набора данных, обрабатывавшихся в разных программах. Всякий раз, когда такая задача возникает, это доставляет много боли. `dplyr` предлагает интуитивно понятный инструмент для объединения тибблов - семейство функций `*_join()`.

Возьмем для примера два тиббла `band_members` и `band_instruments`, встроенных в `dplyr` специально для демонстрации работы функций `*_join()`.

```
band_members
```

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

- `left_join()`:

```
band_members %>%
  left_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

```
band_members %>%
  left_join(band_instruments, by = "name")
```

```
## # A tibble: 3 x 3
##   name  band    plays
```

```
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

```
band_members %>%
  left_join(band_instruments2, by = c("name" = "artist"))
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

- right_join():

```
band_members %>%
  right_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
## 3 Keith <NA>    guitar
```

- full_join():

```
band_members %>%
  full_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

- inner_join():

```
band_members %>%
  inner_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

- semi_join():

```
band_members %>%
  semi_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 2 x 2
##   name  band
##   <chr> <chr>
## 1 John  Beatles
## 2 Paul  Beatles
```

- anti_join():

```
band_members %>%
  anti_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 1 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
```

6.16 Tidy data: `tidyr::pivot_longer()`, `tidyr::pivot_wider()`

Принцип tidy data предполагает, что каждая строчка содержит в себе одно измерение, а каждая колонка - одну характеристику. Тем не менее, это не говорит однозначно о том, как именно хранить повторные измерения. Их можно хранить как одну колонку для каждого измерения (широкий формат) и как две колонки: одна колонка - для идентификатора измерения, другая колонка - для записи самого измерения.

Это лучше понять на примере. Например, вес до и после прохождения курса. Как это лучше записать - как два числовых столбца (один испытуемый - одна строка) или же создать отдельную "группирующую" колонку, в которой будет написано время измерения, а в другой - измеренные значения (одно измерение - одна строка)?

- **Широкий формат:**

Студент	До курса по R	После курса по R
Маша	70	63
Рома	80	74
Антонина	86	71

- **Длинный"формат:**

Студент	Время измерения	Масса (кг)
Маша	До курса по R	70
Рома	До курса по R	80
Антонина	До курса по R	86
Маша	После курса по R	63
Рома	После курса по R	74
Антонина	После курса по R	71

На самом деле, оба варианта приемлимы, оба варианта возможны в реальных данных, а разные функции и статистические пакеты могут требовать от вас как длинный, так и широкий форматы.

Таким образом, нам нужно научиться переводить из широкого формата в длинный и наоборот.

- `tidyr::pivot_longer()`: из *широкого* в *длинный* формат
- `tidyr::pivot_wider()`: из *длинного* в *широкий* формат

```
new_diet <- tibble(
  student = c("   ", "   ", "     "),
  before_r_course = c(70, 80, 86),
  after_r_course = c(63, 74, 71)
)
new_diet
```

```
## # A tibble: 3 x 3
##   student  before_r_course after_r_course
##   <chr>              <dbl>          <dbl>
```

```
## 1                    70             63
## 2                    80             74
## 3                   86             71
```

Тиббл `new_diet` - это пример широкого формата данных.

Превратим тиббл `new_diet` длинный:

```
new_diet %>%
  pivot_longer(cols = before_r_course:after_r_course,
               names_to = "measurement_time",
               values_to = "weight_kg")
```

```
## # A tibble: 6 x 3
##   student  measurement_time weight_kg
##   <chr>    <chr>                <dbl>
## 1        before_r_course         70
## 2        after_r_course          63
## 3        before_r_course         80
## 4        after_r_course          74
## 5      before_r_course         86
## 6      after_r_course          71
```

А теперь обратно в короткий:

```
new_diet %>%
  pivot_longer(cols = before_r_course:after_r_course,
               names_to = "measurement_time",
               values_to = "weight_kg") %>%
  pivot_wider(names_from = "measurement_time",
              values_from = "weight_kg")
```

```
## # A tibble: 3 x 3
##   student  before_r_course after_r_course
##   <chr>              <dbl>          <dbl>
## 1                    70             63
## 2                    80             74
## 3                   86             71
```

Глава 7

Задания

7.1 Начало работы в R

- Разделите 9801 на 9.

```
## [1] 1089
```

- Посчитайте логарифм от 8912162342 по основанию 6.

```
## [1] 12
```

- Теперь натуральный логарифм 10 и умножьте его на 5.

```
## [1] 11.51293
```

- С помощью функции `sin()` посчитайте $\sin(\pi), \sin\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{6}\right)$.

  Значение π - зашитая в R константа (`pi`).

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

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

```
## [1] 0.5
```

7.2 Создание векторов

- Создайте вектор из значений 2, 30 и 4000.

```
## [1]    2   30 4000
```

- Создайте вектор от 1 до 20.

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

- Создайте вектор от 20 до 1.

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

Функция `sum()` возвращает сумму элементов вектора на входе. Посчитайте сумму первых 100 натуральных чисел (т.е. всех целых чисел от 1 до 100).

```
## [1] 5050
```

- Создайте вектор от 1 до 20 и снова до 1. Число 20 должно присутствовать только один раз!

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18 17 16 15
## [26] 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

- Создайте вектор значений 5, 4, 3, 2, 2, 3, 4, 5:

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

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

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

- Создайте вектор 0.1, 0.2, 0.3, ..., 0.9, 1.

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- 2020 год — високосный. Следующий високосный год через 4 года — это будет 2024 год. Составьте календарь всех високосных годов XXI века, начиная с 2020 года.

  2100 год относится к XXI веку, а не к XXII.

```
##  [1] 2020 2024 2028 2032 2036 2040 2044 2048 2052 2056 2060 2064 2068 2072 2076
## [16] 2080 2084 2088 2092 2096 2100
```

- Создайте вектор, состоящий из 20 повторений "Хэй!".

```
##  [1] "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !"
## [11] "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !" "  !"
```

- Как я и говорил, многие функции, работающие с одним значением на входе, так же прекрасно работают и с целыми векторами. Попробуйте посчитать квадратный корень чисел от 1 до 10 с помощью функции `sqrt()` и сохраните результат в векторе `roots`.

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

- Давайте убедимся, что это действительно квадратные корни. Для этого возведите все значения вектора `roots` в квадрат!

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

- Если все верно, то того же самого можно добиться поэлементным умножением вектора `roots` на себя.

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

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

```
##  [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9
## [39] 9 9 9 9 9 9 9
```

7.3 Приведение типов

- Сделайте вектор `vec1`, в котором соедините `3`, а также значения `"  "` и `"    "`.

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

- Попробуйте вычесть `TRUE` из 10.

```
## [1] 9
```

- Соедините значение `10` и `TRUE` в вектор `vec2`.

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

- Соедините вектор `vec2` и значение `"r"`:

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

- Соедините значения `10`, `TRUE`, `"r"` в вектор.

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

7.4 Векторизация

- Создайте вектор `p`, состоящий из значений 4, 5, 6, 7, и вектор `q`, состоящий из 0, 1, 2, 3.

```
## [1] 4 5 6 7
```

```
## [1] 0 1 2 3
```

- Посчитайте поэлементную сумму векторов `p` и `q`:

```
## [1]  4  6  8 10
```

- Посчитайте поэлементную разницу `p` и `q`:

```
## [1] 4 4 4 4
```

- Поделите каждый элемент вектора `p` на соответствующий ему элемент вектора `q`:

  О, да, Вам нужно делить на 0!

```
## [1]       Inf 5.000000 3.000000 2.333333
```

- Возведите каждый элемент вектора p в степень соответствующего ему элемента вектора q:

```
## [1]   1   5  36 343
```

- Умножьте каждое значение вектора p на 10.

```
## [1] 40 50 60 70
```

- Создайте вектор квадратов чисел от 1 до 10:

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

- Создайте вектор 0, 2, 0, 4, ... , 18, 0, 20.

```
##  [1]  0  2  0  4  0  6  0  8  0 10  0 12  0 14  0 16  0 18  0 20
```

- Создайте вектор 1, 0, 3, 0, 5, ..., 17, 0, 19, 0.

```
##  [1]  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19  0
```

- *Создайте вектор, в котором будут содержаться первые 20 степеней двойки.

```
##  [1]       2       4       8      16      32      64     128     256     512
## [10]    1024    2048    4096    8192   16384   32768   65536  131072  262144
## [19]  524288 1048576
```

- *Создайте вектор из чисел 1, 10, 100, 1000, 10000:

```
## [1]     1    10   100  1000 10000
```

- *Посчитать сумму последовательности $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \ldots + \frac{1}{50 \cdot 51}$.

```
## [1] 0.9803922
```

- *Посчитать сумму последовательности $\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \frac{1}{2^{20}}$.

```
## [1] 1.999999
```

- *Посчитать сумму последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$.

```
## [1] 3.749174
```

- *Сколько чисел из последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$ больше чем 0.5?

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

7.5 Индексирование векторов

- Создайте вектор `troiki` со значениями 3, 6, 9, ..., 24, 27.

```
## [1]  3  6  9 12 15 18 21 24 27
```

- Извлеките 2, 5 и 7 значения вектора `troiki`.

```
## [1]  6 15 21
```

- Извлеките предпоследнее значение вектора `troiki`.

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

- Извлеките все значения вектора `troiki` *кроме* предпоследнего:

```
## [1]  3  6  9 12 15 18 21 27
```

Создайте вектор `vec3`:

```
vec3 <- c(3, 5, 2, 1, 8, 4, 9, 10, 3, 15, 1, 11)
```

- Найдите второй элемент вектора `vec3`.

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

- Верните второй и пятый элемент вектора `vec3`.

```
## [1] 5 8
```

- Попробуйте извлечь сотое значение вектора `vec3`:

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

- Верните все элементы вектора `vec3` *кроме* второго элемента.

```
##  [1]  3  2  1  8  4  9 10  3 15  1 11
```

- Верните все элементы вектора `vec3` *кроме* второго и пятого элемента.

```
##  [1]  3  2  1  4  9 10  3 15  1 11
```

- Найдите последний элемент вектора `vec3`.

```
## [1] 11
```

- Верните все значения вектора `vec3` кроме первого и последнего.

```
##  [1]  5  2  1  8  4  9 10  3 15  1
```

- Найдите все значения вектора `vec3`, которые больше 4.

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

- Найдите все значения вектора `vec3`, которые больше 4, но меньше 10.

  Если хотите сделать это в одну строчку, то вам помогут логические операторы!

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

- Найдите все значения вектора `vec3`, которые меньше 4 или больше 10.

```
## [1]  3  2  1  3 15  1 11
```

- Возведите в квадрат каждое значение вектора `vec3`.

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

- *Возведите в квадрат каждое значение вектора на нечетной позиции и извлеките корень из каждого значения на четной позиции вектора `vec3`.

  Извлечение корня - это то же самое, что и возведение в степень 0.5.

```
##  [1]  9.000000  2.236068  4.000000  1.000000 64.000000  2.000000 81.000000
##  [8]  3.162278  9.000000  3.872983  1.000000  3.316625
```

- Создайте вектор 2, 4, 6, ... , 18, 20 как минимум 2 новыми способами.

  Знаю, это задание может показаться бессмысленным, но это очень базовая операция, с помощью которой можно, например, разделить данные на две части. Чем больше способов Вы знаете, тем лучше!

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

7.6 Работа с пропущенными значениями

- Создайте вектор `vec4` со значениями 300, 15, 8, 2, 0, 1, 110:

```
vec4 <- c(300, 15, 8, 20, 0, 1, 110)
vec4
```

```
## [1] 300  15   8  20   0   1 110
```

- Замените все значения `vec4`, которые больше 20 на `NA`.
- Проверьте полученный вектор `vec4`:

```
## [1] NA 15  8 20  0  1 NA
```

- Посчитайте сумму `vec4` с помощью функции `sum()`. Ответ `NA` не считается!

```
## [1] 44
```

7.7 Матрицы

- Создайте матрицу 4x4, состоящую из единиц. Назовите ее `M1`.

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

```
## [4,]    1    1    1    1
```

- Поменяйте все некрайние значения матрицы `M1` (то есть значения на позициях [2,2], [2,3], [3,2] и [3,3]) на число 2.

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

- Выделите второй и третий столбик из матрицы `M1`.

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

- Сравните (==) вторую колонку и вторую строчку матрицы `M1`.

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

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

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    1    2    3    4    5    6    7    8    9
##  [2,]    2    4    6    8   10   12   14   16   18
##  [3,]    3    6    9   12   15   18   21   24   27
##  [4,]    4    8   12   16   20   24   28   32   36
##  [5,]    5   10   15   20   25   30   35   40   45
##  [6,]    6   12   18   24   30   36   42   48   54
##  [7,]    7   14   21   28   35   42   49   56   63
##  [8,]    8   16   24   32   40   48   56   64   72
##  [9,]    9   18   27   36   45   54   63   72   81
```

- *Из матрицы `mult_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 (`mult_tab`) двузначное число и `FALSE`, если однозначное.

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

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
##  [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [2,] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [3,] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [4,] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [5,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [6,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [7,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [8,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [9,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- *Создайте матрицу `mult_tab2`, в которой все значения `tab` меньше 10 заменены на 0.

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    0    0    0    0    0    0    0    0    0
##  [2,]    0    0    0    0   10   12   14   16   18
##  [3,]    0    0    0   12   15   18   21   24   27
##  [4,]    0    0   12   16   20   24   28   32   36
##  [5,]    0   10   15   20   25   30   35   40   45
##  [6,]    0   12   18   24   30   36   42   48   54
##  [7,]    0   14   21   28   35   42   49   56   63
##  [8,]    0   16   24   32   40   48   56   64   72
##  [9,]    0   18   27   36   45   54   63   72   81
```

7.8 Списки

Дан список `list1`:

```
list1 = list(numbers = 1:5, letters = letters, logic = TRUE)
list1
```

```
## $numbers
## [1] 1 2 3 4 5
##
## $letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $logic
## [1] TRUE
```

- Найдите первый элемент списка `list1`. Ответ должен быть списком длиной один.

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

- Теперь найдите содержание первого элемента списка `list1` двумя разными способами. Ответ должен быть вектором.

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

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

- Теперь возьмите первый элемент содержания первого элемента списка `list1`. Ответ должен быть вектором.

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

- Создайте список `list2`, содержащий в себе два списка `list1`. Один из них будет иметь имя `pupa`, а другой — `lupa`.

```
## $pupa
## $pupa$numbers
## [1] 1 2 3 4 5
##
## $pupa$letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $pupa$logic
## [1] TRUE
##
##
## $lupa
## $lupa$numbers
## [1] 1 2 3 4 5
##
## $lupa$letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $lupa$logic
## [1] TRUE
```

- *Извлеките первый элемент списка `list2`, из него — второй полэлемент, а из него — третье значение.

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

7.9 Датафрейм

- Запустите команду `data(mtcars)` чтобы загрузить встроенный датафрейм с информацией про автомобили. Каждая строчка датафрейма - модель автомобиля, каждая колонка - отдельная характеристика. Подробнее см. `?mtcars`.

```
data(mtcars)
mtcars
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9          27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Изучите структуру датафрейма `mtcars` с помощью функции `str()`.

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- Найдите значение третьей строчки четвертого столбца датафрейма `mtcars`.

```
## [1] 93
```

- Извлеките первые шесть строчек и первые шесть столбцов датафрейма `mtcars`.

```
##                   mpg cyl disp  hp drat    wt
## Mazda RX4         21.0   6  160 110 3.90 2.620
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875
## Datsun 710        22.8   4  108  93 3.85 2.320
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215
## Hornet Sportabout 18.7   8  360 175 3.15 3.440
## Valiant           18.1   6  225 105 2.76 3.460
```

- Извлеките колонку `wt` датафрейма `mtcars` - массу автомобиля в тысячах фунтов.

```
##  [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070
## [13] 3.730 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840
## [25] 3.845 1.935 2.140 1.513 3.170 2.770 3.570 2.780
```

- Извлеките колонки из `mtcars` в следующем порядке: `hp`, `mpg`, `cyl`.

```
##                     hp  mpg cyl
## Mazda RX4          110 21.0   6
## Mazda RX4 Wag      110 21.0   6
## Datsun 710          93 22.8   4
## Hornet 4 Drive     110 21.4   6
## Hornet Sportabout  175 18.7   8
## Valiant            105 18.1   6
## Duster 360         245 14.3   8
## Merc 240D           62 24.4   4
## Merc 230            95 22.8   4
## Merc 280           123 19.2   6
## Merc 280C          123 17.8   6
```

```
## Merc 450SE          180 16.4   8
## Merc 450SL          180 17.3   8
## Merc 450SLC         180 15.2   8
## Cadillac Fleetwood  205 10.4   8
## Lincoln Continental 215 10.4   8
## Chrysler Imperial   230 14.7   8
## Fiat 128             66 32.4   4
## Honda Civic          52 30.4   4
## Toyota Corolla       65 33.9   4
## Toyota Corona        97 21.5   4
## Dodge Challenger    150 15.5   8
## AMC Javelin         150 15.2   8
## Camaro Z28          245 13.3   8
## Pontiac Firebird    175 19.2   8
## Fiat X1-9            66 27.3   4
## Porsche 914-2        91 26.0   4
## Lotus Europa        113 30.4   4
## Ford Pantera L      264 15.8   8
## Ferrari Dino        175 19.7   6
## Maserati Bora       335 15.0   8
## Volvo 142E          109 21.4   4
```

- Посчитайте *количество* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
## [1] 11
```

- Посчитайте *долю* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
## [1] 0.34375
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) в датафрейме `mtcars`.

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
```

```
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) и 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
##               mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Volvo 142E   21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

- Посчитайте максимальную массу (`wt`) автомобиля в выборке, воспользовавшись функцией `max()`:

```
## [1] 5.424
```

- Посчитайте максимальную массу (`wt`) автомобиля в выборке, воспользовавшись функцией `min()`:

```
## [1] 1.513
```

- Найдите строчку датафрейма `mtcars` с самым легким автомобилем.

```
##               mpg cyl disp  hp drat    wt qsec vs am gear carb
## Lotus Europa 30.4   4 95.1 113 3.77 1.513 16.9  1  1    5    2
```

- Извлеките строчки датафрейма `mtcars` с автомобилями, масса которых ниже средней массы.

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Масса автомобиля указана в тысячах фунтов. Создайте колонку `wt_kg` с массой автомобиля в килограммах. Результат округлите до целых значений с помощью функции `round()`.

  1 фунт = 0.45359237 кг.

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb wt_kg
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4  1188
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4  1304
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1  1052
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1  1458
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2  1560
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1  1569
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4  1619
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2  1447
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2  1429
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4  1560
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4  1560
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3  1846
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3  1692
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3  1715
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4  2381
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4  2460
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4  2424
## Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1   998
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2   733
## Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1   832
## Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1  1118
## Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2  1597
## AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2  1558
## Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4  1742
## Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2  1744
## Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1   878
## Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2   971
## Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2   686
## Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4  1438
## Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6  1256
## Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8  1619
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2  1261
```

7.10 Условные конструкции

- Создайте вектор `vec5`:

```
vec5 <- c(5, 20, 30, 0, 2, 9)
```

- Создайте новый строковый вектор, где на месте чисел больше 10 в vec5 будет стоять “большое число”, а на месте остальных чисел — “маленькое число”.

```
## [1] "         " "         "    "         "    "         "
## [5] "         " "          "
```

- Загрузите файл heroes_information.csv в переменную heroes.

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

- Создайте новою колонку hair в heroes, в которой будет значение "Bold" для тех супергероев, у которых в колонке Hair.color стоит "No Hair", и значение "Hairy" во всех остальных случаях.

```
##   X           name Gender Eye.color               Race Hair.color Height
## 1 0         A-Bomb   Male    yellow              Human    No Hair    203
## 2 1     Abe Sapien   Male      blue      Icthyo Sapien    No Hair    191
## 3 2       Abin Sur   Male      blue            Ungaran    No Hair    185
## 4 3    Abomination   Male     green Human / Radiation    No Hair    203
## 5 4        Abraxas   Male      blue      Cosmic Entity      Black     NA
## 6 5 Absorbing Man   Male      blue              Human    No Hair    193
##            Publisher Skin.color Alignment Weight  hair
## 1      Marvel Comics       <NA>      good    441  Bold
## 2 Dark Horse Comics       blue      good     65  Bold
## 3          DC Comics        red      good     90  Bold
## 4      Marvel Comics       <NA>       bad    441  Bold
## 5      Marvel Comics       <NA>       bad     NA Hairy
## 6      Marvel Comics       <NA>       bad    122  Bold
```

- Создайте новою колонку tall в heroes, в которой будет значение "tall" для тех супергероев, у которых в колонке Height стоит число больше 190, значение "short" для тех супергероев, у которых в колонке Height стоит число меньше 170, и значение "middle" во всех остальных случаях.

7.11 Создание функций

- Создайте функцию plus_one(), которая принимает число и возвращает это же число + 1.
- Проверьте функцию plus_one() на числе 41.

```
plus_one(41)
```

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

- Создайте функцию `circle_area`, которая вычисляет площадь круга по радиусу согласно формуле πr^2.
- Посчитайте площадь круга с радиусом 5.

```
## [1] 78.53982
```

- Создайте функцию `cels2fahr()`, которая будет превращать градусы по Цельсию в градусы по Фаренгейту.
- Проверьте на значениях -100, -40 и 0, что функция `cels2fahr()` работает корректно.

```
cels2fahr(c(-100, -40, 0))
```

```
## [1] -148  -40   32
```

- Напишите функцию `highlight()`, которая принимает на входе строковый вектор, а возвращает тот же вектор, но дополненный значением `"***"` в начале и конце вектора. Лучше всего это рассмотреть на примере:

```
highlight(c(" ", "   !"))
```

```
## [1] "***"     " "       "   !" "***"
```

- Теперь сделайте функцию `highlight` более гибкой. Добавьте в нее параметр `wrapper =`, который по умолчанию равен `"***"`. Значение параметра `wrapper =` и будет вставлено в начало и конец вектора.
- Проверьте написанную функцию на векторе `c("  "   !")`.

```
highlight(c(" ", "   !"))
```

```
## [1] "***"     " "       "   !" "***"
```

```
highlight(c(" ", "   !"), wrapper = "__")
```

```
## [1] "__"      " "       "   !" "__"
```

- Создайте функцию `trim()`, которая будет возвращать вектор без первого и последнего значения (вне зависимости от типа данных).
- Проверьте, что функция `trim()` работает корректно:

```
trim(1:7)
```

```
## [1] 2 3 4 5 6
```

```
trim(letters)
```

```
##  [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
## [20] "u" "v" "w" "x" "y"
```

- Теперь добавьте в функцию `trim()` параметр `n =` со значением по умолчанию 1. Этот параметр будет обозначать сколько значений нужно отрезать слева и справа от вектора.
- Проверьте полученную функцию:

```
trim(letters)
```

```
##  [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
## [20] "u" "v" "w" "x" "y"
```

```
trim(letters, n = 2)
```

```
##  [1] "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u"
## [20] "v" "w" "x"
```

- Сделайте так, чтобы функция `trim()` работала корректно с `n = 0`, т.е. функция возвращала бы исходный вектор без изменений.

```
trim(letters, n = 0)
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

- *Теперь добавьте проверку на адекватность входных данных: функция `trim()` должна выдавать ошибку, если `n =` меньше нуля или если `n =` слишком большой и отрезает все значения вектора:
- *Проверьте полученную функцию `trim()`:

```
trim(1:6, 3)
```

```
## Error in trim(1:6, 3): n          !
```

```
trim(1:6, -1)
```

```
## Error in trim(1:6, -1): n                  !
```

- Создайте функцию `na_n()`, которая будет возвращать количество `NA` в векторе.
- Проверьте функцию `na_n()` на векторе:

```
na_n(c(NA, 3:5, NA, 2, NA))
```

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

- Напишите функцию `factors()`, которая будет возвращать все делители числа в виде числового вектора.

  Здесь может понадобиться оператор для получения остатка от деления: `%%`.
- Проверьте функцию `factors()` на простых и сложных числах:

```
factors(3)
```

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

```
factors(161)
```

```
## [1]   1   7  23 161
```

```
factors(1984)
```

```
##  [1]    1    2    4    8   16   31   32   62   64  124  248  496  992 1984
```

- *Напишите функцию `is_prime()`, которая проверяет, является ли число простым.

  Здесь может пригодиться функция `any()` - она возвращает `TRUE`, если в векторе есть хотя бы один `TRUE`.
- Проверьте какие года были для нас простыми, а какие нет:

```
is_prime(2017)
```

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

```
is_prime(2019)
```

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

```
2019/3 #2019        3
```

```
## [1] 673
```

```
is_prime(2020)
```

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

```
is_prime(2021)
```

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

- *Создайте функцию `monotonic()`, которая возвращает `TRUE`, если значения в векторе не убывают (то есть каждое следующее - больше или равно предыдущему) или не возврастают.

  Полезная функция для этого — `diff()` — возвращает разницу соседних значений.

```
monotonic(1:7)
```

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

```
monotonic(c(1:5,5:1))
```

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

```
monotonic(6:-1)
```

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

```
monotonic(c(1:5, rep(5, 10), 5:10))
```

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

Бинарные операторы типа + или `%in%` тоже представляют собой функции. Более того, мы можем создавать свои бинарные операторы! В этом нет особой сложности — нужно все так же создавать функцию (для двух переменных), главное

окружать их % и название обрамлять обратными штрихами ‘. Например, можно сделать свой бинарный оператор %notin%, который будет выдавать TRUE, если значения слева *нет* в векторе справа:

```
`%notin%` <- function(x, y) ! (x %in% y)
1:10 %notin% c(1, 4, 5)
```

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

- *Создайте бинарный оператор %without%, который будет возвращать все значения вектора слева без значений вектора справа.

```
c(" ", " ", " ", "   ", " ", "   ") %without% c(" ", " ")
```

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

- *Создайте бинарный оператор %between%, который будет возвращать TRUE, если значение в векторе слева накходится в *диапазоне* значений вектора справа:

```
1:10 %between% c(1, 4, 5)
```

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

7.12 Семейство функций apply()

- Создайте матрицу M2:

```
M2 <- matrix(c(20:11, 11:20), nrow = 5)
M2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   20   15   11   16
## [2,]   19   14   12   17
## [3,]   18   13   13   18
## [4,]   17   12   14   19
## [5,]   16   11   15   20
```

- Посчитайте максимальное значение матрицы M2 по каждой строчке.

```
## [1] 20 19 18 19 20
```

- Посчитайте максимальное значение матрицы M2 по каждому столбцу.

```
## [1] 20 15 15 20
```

- Посчитайте среднее значение матрицы `M2` по каждой строке.

```
## [1] 15.5 15.5 15.5 15.5 15.5
```

- Посчитайте среднее значение матрицы `M2` по каждому столбцу.

```
## [1] 18 13 13 18
```

- Создайте список `list3`:

```
list3 <- list(
  a = 1:5,
  b = 0:20,
  c = 4:24,
  d = 6:3,
  e = 6:25
  )
```

- Найдите максимальное значение каждого вектора списка `list3`.

```
##  a  b  c  d  e
##  5 20 24  6 25
```

- Посчитайте сумму каждого вектора списка `list3`.

```
##   a   b   c   d   e
##  15 210 294  18 310
```

- Посчитайте длину каждого вектора списка `list3`.

```
##  a  b  c  d  e
##  5 21 21  4 20
```

- Напишите функцию `max_item()`, которая будет принимать на входе список, а возвращать - (первый) самый длинный его элемент.

  Для этого вам может понадобиться функция `which.max()`, которая возвращает индекс максимального значения (первого, если их несколько).

- Проверьте функцию `max_item()` на списке `list3`.

```
max_item(list3)
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Теперь мы сделаем сложный список `list4`:

```
list4 <- list(1:3, 3:40, list3)
```

- Посчитайте длину каждого вектора в списке, в т.ч. для списка внутри. Результат должен быть списком с такой же структорой, как и изначальный список `list4`.

  Для этого может понадобиться функция `rapply()`: **recursive lapply**

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 38
##
## [[3]]
## [[3]]$a
## [1] 5
##
## [[3]]$b
## [1] 21
##
## [[3]]$c
## [1] 21
##
## [[3]]$d
## [1] 4
##
## [[3]]$e
## [1] 20
```

- *Загрузите набор данных `heroes` и посчитайте, сколько `NA` в каждом из столбцов.

  Для этого удобно использовать ранее написанную функцию `na_n()`.

```
##          X       name     Gender  Eye.color       Race Hair.color     Height
##          0          0         29        172        304        172        217
##  Publisher Skin.color  Alignment     Weight       hair       tall
##          0        662          7        239        172        217
```

- *Используя ранее написанную функцию `is_prime()`, напишите функцию `prime_numbers()`, которая будет возвращать все простые числа до выбранного числа.

```
prime_numbers(200)
```

```
##  [1]   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71
## [20]  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167
## [39] 173 179 181 191 193 197 199
```

7.13 magrittr::%>%

- Перепишите следующие выражения, используя %>%:

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

```
## [1] 7.416198
## [1] 7.416198
```

```
abs(min(-5:5))
```

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

```
c("     ", 2, "   ", sqrt(2))
```

```
## [1] "     "       "2"               "   "           "1.4142135623731"
## [1] "     "       "2"               "   "           "1.4142135623731"
```

7.14 Выбор строк: `dplyr::slice()` и `dplyr::filter()`

- Выберите только те строчки, в которых содержится информация о супергероях тяжелее 500 кг.

```
## # A tibble: 6 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   203 Dark~ Male   red         New ~ No Hair       267   DC Comics
## 2   283 Giga~ Female green       <NA>  Red            62.5 DC Comics
## 3   331 Hulk  Male   green       Huma~ Green         244   Marvel C~
## 4   373 Jugg~ Male   blue        Human Red           287   Marvel C~
## 5   549 Red ~ Male   yellow      Huma~ Black         213   Marvel C~
## 6   575 Sasq~ Male   red         <NA>  Orange        305   Marvel C~
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о *женщинах*-супергероях тяжелее 500 кг.

```
## # A tibble: 1 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   283 Giga~ Female green       <NA>  Red            62.5 DC Comics
```

```
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о супергероях человеческой расы (`"Human"`) женского пола. Из этих супергероев возьмите первые 5.

```
## # A tibble: 5 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1    38 Arac~ Female blue        Human Blond           175 Marvel C~
## 2    63 Batg~ Female green       Human Red             170 DC Comics
## 3    65 Batg~ Female green       Human Black           165 DC Comics
## 4    72 Batw~ Female green       Human Red             178 DC Comics
## 5    96 Blac~ Female blue        Human Blond           165 DC Comics
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

7.15 Выбор столбцов: `dplyr::select()`

- Выберете первые 4 столбца в `powers`.

```
## # A tibble: 667 x 4
##    hero_names    Agility `Accelerated Healing` `Lantern Power Ring`
##    <chr>         <lgl>   <lgl>                 <lgl>
##  1 3-D Man       TRUE    FALSE                 FALSE
##  2 A-Bomb        FALSE   TRUE                  FALSE
##  3 Abe Sapien    TRUE    TRUE                  FALSE
##  4 Abin Sur      FALSE   FALSE                 TRUE
##  5 Abomination   FALSE   TRUE                  FALSE
##  6 Abraxas       FALSE   FALSE                 FALSE
##  7 Absorbing Man FALSE   FALSE                 FALSE
##  8 Adam Monroe   FALSE   TRUE                  FALSE
##  9 Adam Strange  FALSE   FALSE                 FALSE
## 10 Agent Bob     FALSE   FALSE                 FALSE
## # ... with 657 more rows
```

- Выберите все столбцы от `Reflexes` до `Empathy` в тиббле `powers`:

```
## # A tibble: 667 x 7
##    Reflexes Invulnerability `Energy Constru~ `Force Fields` `Self-Sustenanc~
##    <lgl>    <lgl>           <lgl>            <lgl>          <lgl>
##  1 FALSE    FALSE           FALSE            FALSE          FALSE
##  2 FALSE    FALSE           FALSE            FALSE          TRUE
##  3 TRUE     FALSE           FALSE            FALSE          FALSE
##  4 FALSE    FALSE           FALSE            FALSE          FALSE
##  5 FALSE    TRUE            FALSE            FALSE          FALSE
##  6 FALSE    TRUE            FALSE            FALSE          FALSE
```

```
##  7 FALSE    TRUE           FALSE          FALSE          FALSE
##  8 FALSE    FALSE          FALSE          FALSE          FALSE
##  9 FALSE    FALSE          FALSE          FALSE          FALSE
## 10 FALSE    FALSE          FALSE          FALSE          FALSE
## # ... with 657 more rows, and 2 more variables: `Anti-Gravity` <lgl>,
## #   Empathy <lgl>
```

- Выберите все столбцы тиббла powers кроме первого (hero_names):

```
## # A tibble: 667 x 167
##    Agility `Accelerated He~ `Lantern Power ~ `Dimensional Aw~ `Cold Resistanc~
##    <lgl>   <lgl>            <lgl>            <lgl>            <lgl>
##  1 TRUE    FALSE            FALSE            FALSE            FALSE
##  2 FALSE   TRUE             FALSE            FALSE            FALSE
##  3 TRUE    TRUE             FALSE            FALSE            TRUE
##  4 FALSE   FALSE            TRUE             FALSE            FALSE
##  5 FALSE   TRUE             FALSE            FALSE            FALSE
##  6 FALSE   FALSE            FALSE            TRUE             FALSE
##  7 FALSE   FALSE            FALSE            FALSE            TRUE
##  8 FALSE   TRUE             FALSE            FALSE            FALSE
##  9 FALSE   FALSE            FALSE            FALSE            FALSE
## 10 FALSE   FALSE            FALSE            FALSE            FALSE
## # ... with 657 more rows, and 162 more variables: Durability <lgl>,
## #   Stealth <lgl>, `Energy Absorption` <lgl>, Flight <lgl>, `Danger
## #   Sense` <lgl>, `Underwater breathing` <lgl>, Marksmanship <lgl>, `Weapons
## #   Master` <lgl>, `Power Augmentation` <lgl>, `Animal Attributes` <lgl>,
## #   Longevity <lgl>, Intelligence <lgl>, `Super Strength` <lgl>,
## #   Cryokinesis <lgl>, Telepathy <lgl>, `Energy Armor` <lgl>, `Energy
## #   Blasts` <lgl>, Duplication <lgl>, `Size Changing` <lgl>, `Density
## #   Control` <lgl>, Stamina <lgl>, `Astral Travel` <lgl>, `Audio
## #   Control` <lgl>, Dexterity <lgl>, Omnitrix <lgl>, `Super Speed` <lgl>,
## #   Possession <lgl>, `Animal Oriented Powers` <lgl>, `Weapon-based
## #   Powers` <lgl>, Electrokinesis <lgl>, `Darkforce Manipulation` <lgl>, `Death
## #   Touch` <lgl>, Teleportation <lgl>, `Enhanced Senses` <lgl>,
## #   Telekinesis <lgl>, `Energy Beams` <lgl>, Magic <lgl>, Hyperkinesis <lgl>,
## #   Jump <lgl>, Clairvoyance <lgl>, `Dimensional Travel` <lgl>, `Power
## #   Sense` <lgl>, Shapeshifting <lgl>, `Peak Human Condition` <lgl>,
## #   Immortality <lgl>, Camouflage <lgl>, `Element Control` <lgl>,
## #   Phasing <lgl>, `Astral Projection` <lgl>, `Electrical Transport` <lgl>,
## #   `Fire Control` <lgl>, Projection <lgl>, Summoning <lgl>, `Enhanced
## #   Memory` <lgl>, Reflexes <lgl>, Invulnerability <lgl>, `Energy
## #   Constructs` <lgl>, `Force Fields` <lgl>, `Self-Sustenance` <lgl>,
## #   `Anti-Gravity` <lgl>, Empathy <lgl>, `Power Nullifier` <lgl>, `Radiation
## #   Control` <lgl>, `Psionic Powers` <lgl>, Elasticity <lgl>, `Substance
## #   Secretion` <lgl>, `Elemental Transmogrification` <lgl>,
## #   `Technopath/Cyberpath` <lgl>, `Photographic Reflexes` <lgl>, `Seismic
## #   Power` <lgl>, Animation <lgl>, Precognition <lgl>, `Mind Control` <lgl>,
```

```
## #   `Fire Resistance` <lgl>, `Power Absorption` <lgl>, `Enhanced
## #   Hearing` <lgl>, `Nova Force` <lgl>, Insanity <lgl>, Hypnokinesis <lgl>,
## #   `Animal Control` <lgl>, `Natural Armor` <lgl>, Intangibility <lgl>,
## #   `Enhanced Sight` <lgl>, `Molecular Manipulation` <lgl>, `Heat
## #   Generation` <lgl>, Adaptation <lgl>, Gliding <lgl>, `Power Suit` <lgl>,
## #   `Mind Blast` <lgl>, `Probability Manipulation` <lgl>, `Gravity
## #   Control` <lgl>, Regeneration <lgl>, `Light Control` <lgl>,
## #   Echolocation <lgl>, Levitation <lgl>, `Toxin and Disease Control` <lgl>,
## #   Banish <lgl>, `Energy Manipulation` <lgl>, `Heat Resistance` <lgl>,
## #   `Natural Weapons` <lgl>, ...
```

7.16 Сортировка строк: `dplyr::arrange()`

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по возрастанию* `Height`.

```
## # A tibble: 734 x 3
##    name            Gender Height
##    <chr>           <chr>   <dbl>
##  1 Utgard-Loki     Male     15.2
##  2 Bloodwraith     Male     30.5
##  3 King Kong       Male     30.5
##  4 Anti-Monitor    Male     61
##  5 Giganta         Female   62.5
##  6 Krypto          Male     64
##  7 Yoda            Male     66
##  8 Jack-Jack       Male     71
##  9 Howard the Duck Male     79
## 10 Godzilla        <NA>    108
## # ... with 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по убыванию* `Height`.

```
## # A tibble: 734 x 3
##    name          Gender Height
##    <chr>         <chr>   <dbl>
##  1 Fin Fang Foom Male     975
##  2 Galactus      Male     876
##  3 Groot         Male     701
##  4 MODOK         Male     366
##  5 Wolfsbane     Female   366
##  6 Onslaught     Male     305
##  7 Sasquatch     Male     305
##  8 Ymir          Male     305.
```

```
##  9 Rey           Female   297
## 10 Juggernaut    Male     287
## # ... with 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки сначала по `Gender`, затем *по убыванию* `Height`.

```
## # A tibble: 734 x 3
##    name      Gender Height
##    <chr>     <chr>   <dbl>
##  1 Wolfsbane Female    366
##  2 Rey       Female    297
##  3 Bloodaxe  Female    218
##  4 Thundra   Female    218
##  5 Hela      Female    213
##  6 Frenzy    Female    211
##  7 She-Hulk  Female    201
##  8 Ardina    Female    193
##  9 Starfire  Female    193
## 10 Valkyrie  Female    191
## # ... with 724 more rows
```

7.17 Уникальные значения: `dplyr::distinct()`

- Извлеките уникальные значения столбца `Eye color` из тиббла `heroes`.

```
## # A tibble: 23 x 1
##    `Eye color`
##    <chr>
##  1 yellow
##  2 blue
##  3 green
##  4 brown
##  5 <NA>
##  6 red
##  7 violet
##  8 white
##  9 purple
## 10 black
## # ... with 13 more rows
```

- Извлеките уникальные значения столбца `Hair color` из тиббла `heroes`.

```
## # A tibble: 30 x 1
##    `Hair color`
##    <chr>
```

```
##  1 No Hair
##  2 Black
##  3 Blond
##  4 Brown
##  5 <NA>
##  6 White
##  7 Purple
##  8 Orange
##  9 Pink
## 10 Red
## # ... with 20 more rows
```

7.18 Создание колонок: `dplyr::mutate()` и `dplyr::transmute()`

- Создайте колонку `height_m` с ростом супергероев в метрах, затем выберите только колонки `name` и `height_m`.

```
## # A tibble: 734 x 2
##    name          height_m
##    <chr>            <dbl>
##  1 A-Bomb            2.03
##  2 Abe Sapien        1.91
##  3 Abin Sur          1.85
##  4 Abomination       2.03
##  5 Abraxas          NA
##  6 Absorbing Man     1.93
##  7 Adam Monroe      NA
##  8 Adam Strange      1.85
##  9 Agent 13          1.73
## 10 Agent Bob         1.78
## # ... with 724 more rows
```

- Создайте новою колонку `hair` в heroes, в которой будет значение `"Bold"` для тех супергероев, у которых в колонке `Hair.color` стоит `"No Hair"`, и значение `"Hairy"` во всех остальных случаях. Затем выберите только колонки `name`, `Hair color`, `hair`.

```
## # A tibble: 734 x 3
##    name          `Hair color` hair
##    <chr>         <chr>        <chr>
##  1 A-Bomb        No Hair      Bold
##  2 Abe Sapien    No Hair      Bold
##  3 Abin Sur      No Hair      Bold
##  4 Abomination   No Hair      Bold
##  5 Abraxas       Black        Hairy
```

```
##  6 Absorbing Man No Hair      Bold
##  7 Adam Monroe   Blond        Hairy
##  8 Adam Strange  Blond        Hairy
##  9 Agent 13      Blond        Hairy
## 10 Agent Bob     Brown        Hairy
## # ... with 724 more rows
```

7.19 Агрегация: dplyr::group_by() %>% summarise()

- Посчитайте количество супергероев по расам и отсортируйте по убыванию. Извлеките первые 5 строк.

```
## # A tibble: 5 x 2
##   Race              n
##   <chr>         <int>
## 1 <NA>            304
## 2 Human           208
## 3 Mutant           63
## 4 God / Eternal    14
## 5 Cyborg           11
```

- Посчитайте средний пост по полу.

```
## # A tibble: 3 x 2
##   Gender height_mean
##   <chr>        <dbl>
## 1 Female        175.
## 2 Male          192.
## 3 <NA>          177.
```

7.20 Операции с несколькими колонками: across()

- Посчитайте количество NA в каждой колонке, группируя по полу (Gender).

```
## # A tibble: 3 x 11
##   Gender    X1  name `Eye color`  Race `Hair color` Height Publisher
##   <chr>  <int> <int>       <int> <int>        <int>  <int>     <int>
## 1 Female     0     0          41    98           38     56         0
## 2 Male       0     0         121   184          123    147         0
## 3 <NA>       0     0          10    22           11     14         0
## # ... with 3 more variables: `Skin color` <int>, Alignment <int>, Weight <int>
```

- Посчитайте количество NA в каждой колонке, которая заканчивается на "color", группируя по полу (Gender).

```
## # A tibble: 3 x 4
##   Gender `Eye color` `Hair color` `Skin color`
##   <chr>        <int>        <int>        <int>
## 1 Female          41           38          186
## 2 Male           121          123          449
## 3 <NA>            10           11           27
```

- Создайте из тиббла `heroes` новый тиббл с колонками `name`, `Height` и `Weight`, где для каждого героя содержится значение " ", если его рост или вес выше среднего по колонке и " ", если ниже или равен среднему.

```
## # A tibble: 734 x 3
##    name          Height        Weight
##    <chr>         <chr>         <chr>
##  1 A-Bomb
##  2 Abe Sapien
##  3 Abin Sur
##  4 Abomination
##  5 Abraxas       <NA>          <NA>
##  6 Absorbing Man
##  7 Adam Monroe   <NA>          <NA>
##  8 Adam Strange
##  9 Agent 13
## 10 Agent Bob
## # ... with 724 more rows
```

- Создайте из тиббла `heroes` новый тиббл с колонками `Gender`, `name`, `Height` и `Weight`, где для каждого героя содержится значение " ", если его рост или вес выше среднего по колонке и " ", если ниже или равен среднему *внутри соответствующей группы по полу*.

```
## # A tibble: 734 x 4
## # Groups:   Gender [3]
##    Gender name          Height        Weight
##    <chr>  <chr>         <chr>         <chr>
##  1 Male   A-Bomb
##  2 Male   Abe Sapien
##  3 Male   Abin Sur
##  4 Male   Abomination
##  5 Male   Abraxas       <NA>          <NA>
##  6 Male   Absorbing Man
##  7 Male   Adam Monroe   <NA>          <NA>
##  8 Male   Adam Strange
##  9 Female Agent 13
## 10 Male   Agent Bob
## # ... with 724 more rows
```

7.21 Соединение датафреймов: *_join {#task_join}

Создайте тиббл `web_creators`, в котором будут супергерои, которые могут плести паутину, т.е. у них стоит TRUE в колонке `Web Creation` в тиббле `powers`.

```
## # A tibble: 16 x 12
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1    33 Anti~ Male   blue        Symb~ Blond           229 Marvel C~
##  2    38 Arac~ Female blue        Human Blond           175 Marvel C~
##  3   161 Carn~ Male   green       Symb~ Red             185 Marvel C~
##  4   335 Hybr~ Male   brown       Symb~ Black           175 Marvel C~
##  5   479 Myst~ Male   brown       Human No Hair         180 Marvel C~
##  6   580 Scar~ Male   brown       Clone Brown           193 Marvel C~
##  7   597 Silk  Female brown       Human Black            NA Marvel C~
##  8   620 Spid~ Female blue        Human Brown           170 Marvel C~
##  9   621 Spid~ Female blue        Human Blond           165 Marvel C~
## 10   622 Spid~ Male   hazel       Human Brown           178 Marvel C~
## 11   623 Spid~ <NA>   red         Human Brown           178 Marvel C~
## 12   624 Spid~ Male   brown       Human Black           157 Marvel C~
## 13   673 Toxin Male   blue        Symb~ Brown           188 Marvel C~
## 14   674 Toxin Male   black       Symb~ Blond           191 Marvel C~
## 15   689 Venom Male   blue        Symb~ Strawberry ~    191 Marvel C~
## 16   692 Veno~ Male   <NA>        Symb~ <NA>            226 Marvel C~
## # ... with 4 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>,
## #   `Web Creation` <lgl>
```

7.22 Tidy data

- Для начала создайте тиббл `heroes_weight`, скопировав код:

```
heroes_weight <- heroes %>%
  filter(Publisher %in% c("DC Comics", "Marvel Comics")) %>%
  group_by(Gender, Publisher) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE)) %>%
  drop_na()
heroes_weight
```

```
## # A tibble: 4 x 3
## # Groups:   Gender [2]
##   Gender Publisher     weight_mean
##   <chr>  <chr>               <dbl>
## 1 Female DC Comics            76.8
## 2 Female Marvel Comics        80.1
```

```
## 3 Male   DC Comics           113.
## 4 Male   Marvel Comics       134.
```

Функция `drop_na()` позволяет выбросить все строчки, в которых встречается `NA`.

- Превратите тиббл `heroes_weight` в широкий тиббл:

```
## # A tibble: 2 x 3
## # Groups:   Gender [2]
##   Gender `DC Comics` `Marvel Comics`
##   <chr>        <dbl>           <dbl>
## 1 Female        76.8            80.1
## 2 Male         113.            134.
```

- Затем превратите его обратно в длинный тиббл:

```
## # A tibble: 4 x 3
## # Groups:   Gender [2]
##   Gender Publisher     weight_mean
##   <chr>  <chr>               <dbl>
## 1 Female DC Comics            76.8
## 2 Female Marvel Comics        80.1
## 3 Male   DC Comics           113.
## 4 Male   Marvel Comics       134.
```

Глава 8

Решения заданий

8.1 Начало работы в R

- Разделите 9801 на 9.

```
9801/9
```

```
## [1] 1089
```

- Посчитайте логарифм от 8912162342 по основанию 6.

```
log(2176782336, 6)
```

```
## [1] 12
```

- Теперь натуральный логарифм 10 и умножьте его на 5.

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

```
## [1] 11.51293
```

- С помощью функции `sin()` посчитайте $\sin(\pi), \sin\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{6}\right)$.

  Значение π - зашитая в R константа (`pi`).

```
sin(pi)
```

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

```
sin(pi/2)
```

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

```
sin(pi/6)
```

```
## [1] 0.5
```

8.2 Создание векторов

- Создайте вектор из значений 2, 30 и 4000.

```
c(2, 30, 4000)
```

```
## [1]    2   30 4000
```

- Создайте вектор от 1 до 20.

```
1:20
```

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

- Создайте вектор от 20 до 1.

```
20:1
```

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

Функция `sum()` возвращает сумму элементов вектора на входе. Посчитайте сумму первых 100 натуральных чисел (т.е. всех целых чисел от 1 до 100).

```
sum(1:100)
```

```
## [1] 5050
```

- Создайте вектор от 1 до 20 и снова до 1. Число 20 должно присутствовать только один раз!

```
c(1:20, 19:1)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18 17 16 15
## [26] 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

- Создайте вектор значений 5, 4, 3, 2, 2, 3, 4, 5:

```
c(5:2, 2:5)
```

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

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

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

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

- Создайте вектор 0.1, 0.2, 0.3, ..., 0.9, 1.

```
seq(0, 1, 0.1)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- 2020 год — високосный. Следующий високосный год через 4 года — это будет 2024 год. Составьте календарь всех високосных годов XXI века, начиная с 2020 года.

  2100 год относится к XXI веку, а не к XXII.

```
seq(2020, 2100, 4)
```

```
##  [1] 2020 2024 2028 2032 2036 2040 2044 2048 2052 2056 2060 2064 2068 2072 2076
## [16] 2080 2084 2088 2092 2096 2100
```

- Создайте вектор, состоящий из 20 повторений “Хэй!”.

```
rep(" !", 20)
```

```
##  [1] " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !"
## [11] " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !" " !"
```

- Как я и говорил, многие функции, работающие с одним значением на входе, так же прекрасно работают и с целыми векторами. Попробуйте посчитать квадратный корень чисел от 1 до 10 с помощью функции `sqrt()` и сохраните результат в векторе `roots`.

```
roots <- sqrt(1:10)
roots
```

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

- Давайте убедимся, что это действительно квадратные корни. Для этого возведите все значения вектора `roots` в квадрат!

```
roots ^ 2
```

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

- Если все верно, то того же самого можно добиться поэлементным умножением вектора `roots` на себя.

```
roots * roots
```

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

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

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

```
##  [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9
## [39] 9 9 9 9 9 9 9
```

8.3 Приведение типов

- Сделайте вектор `vec1`, в котором соедините `3`, а также значения `"  "` и `"    "`.

```
vec1 <- c(3, "  ", "    ")
vec1
```

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

- Попробуйте вычесть `TRUE` из `10`.

```
10 - TRUE
```

```
## [1] 9
```

- Соедините значение `10` и `TRUE` в вектор `vec2`.

```
vec2 <- c(10, TRUE)
vec2
```

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

- Соедините вектор vec2 и значение "r":

```
c(vec2, "r")
```

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

- Соедините значения 10, TRUE, "r" в вектор.

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

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

8.4 Векторизация

- Создайте вектор p, состоящий из значений 4, 5, 6, 7, и вектор q, состоящий из 0, 1, 2, 3.

```
p <- 4:7
p
```

```
## [1] 4 5 6 7
```

```
q <- 0:3
q
```

```
## [1] 0 1 2 3
```

- Посчитайте поэлементную сумму векторов p и q:

```
p + q
```

```
## [1]  4  6  8 10
```

- Посчитайте поэлементную разницу p и q:

```
p - q
```

```
## [1] 4 4 4 4
```

- Поделите каждый элемент вектора p на соответствующий ему элемент вектора q:

  О, да, Вам нужно делить на 0!

```
p / q
```

```
## [1]      Inf 5.000000 3.000000 2.333333
```

- Возведите каждый элемент вектора p в степень соответствующего ему элемента вектора q:

```
p ^ q
```

```
## [1]   1   5  36 343
```

- Умножьте каждое значение вектора p на 10.

```
p * 10
```

```
## [1] 40 50 60 70
```

- Создайте вектор квадратов чисел от 1 до 10:

```
(1:10)^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

- Создайте вектор 0, 2, 0, 4, ... , 18, 0, 20.

```
1:20 * 0:1
```

```
##  [1]  0  2  0  4  0  6  0  8  0 10  0 12  0 14  0 16  0 18  0 20
```

- Создайте вектор 1, 0, 3, 0, 5, ..., 17, 0, 19, 0.

```
1:20 * 1:0
```

```
##  [1]  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19  0
```

- *Создайте вектор, в котором будут содержаться первые 20 степеней двойки.

```
2 ^ (1:20)
```

```
##  [1]       2       4       8      16      32      64     128     256     512
## [10]    1024    2048    4096    8192   16384   32768   65536  131072  262144
## [19]  524288 1048576
```

- *Создайте вектор из чисел 1, 10, 100, 1000, 10000:

```
10 ^ (0:4)
```

```
## [1]     1    10   100  1000 10000
```

- *Посчитать сумму последовательности $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \ldots + \frac{1}{50 \cdot 51}$.

```
sum(1 / (1:50 * 2:51))
```

```
## [1] 0.9803922
```

- *Посчитать сумму последовательности $\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \frac{1}{2^{20}}$.

```
sum(1 / 2 ^ (0:20))
```

```
## [1] 1.999999
```

- *Посчитать сумму последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$.

```
sum((3 * (1:10) - 2) / 3 ^ (0:9))
```

```
## [1] 3.749174
```

- *Сколько чисел из последовательности $1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \ldots + \frac{28}{19683}$ больше чем 0.5?

```
sum((3 * (1:10) - 2) / 3 ^ (0:9) > 0.5)
```

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

8.5 Индексирование векторов

- Создайте вектор `troiki` со значениями 3, 6, 9, ..., 24, 27.

```
troiki <- seq(3, 27, 3)
troiki
```

```
## [1]  3  6  9 12 15 18 21 24 27
```

- Извлеките 2, 5 и 7 значения вектора `troiki`.

```
troiki[c(2, 5, 7)]
```

```
## [1]  6 15 21
```

- Извлеките предпоследнее значение вектора `troiki`.

```
troiki[length(troiki) - 1]
```

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

- Извлеките все значения вектора `troiki` *кроме* предпоследнего:

```
troiki[-(length(troiki) - 1)]
```

```
## [1]  3  6  9 12 15 18 21 27
```

Создайте вектор `vec3`:

```
vec3 <- c(3, 5, 2, 1, 8, 4, 9, 10, 3, 15, 1, 11)
```

- Найдите второй элемент вектора `vec3`.

```
vec3[2]
```

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

- Верните второй и пятый элемент вектора `vec3`.

```
vec3[c(2, 5)]
```

```
## [1] 5 8
```

- Попробуйте извлечь сотое значение вектора `vec3`:

```
vec3[100]
```

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

- Верните все элементы вектора `vec3` *кроме* второго элемента.

```
vec3[-2]
```

```
##  [1]  3  2  1  8  4  9 10  3 15  1 11
```

- Верните все элементы вектора `vec3` *кроме* второго и пятого элемента.

```
vec3[c(-2, -5)]
```

```
##  [1]  3  2  1  4  9 10  3 15  1 11
```

- Найдите последний элемент вектора `vec3`.

```
vec3[length(vec3)]
```

```
## [1] 11
```

- Верните все значения вектора `vec3` кроме первого и последнего.

```
vec3[c(-1, -length(vec3))]
```

```
##  [1]  5  2  1  8  4  9 10  3 15  1
```

- Найдите все значения вектора `vec3`, которые больше 4.

```
vec3[vec3 > 4]
```

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

- Найдите все значения вектора `vec3`, которые больше 4, но меньше 10.

  Если хотите сделать это в одну строчку, то вам помогут логические операторы!

```
vec3[vec3 > 4 & vec3 < 10]
```

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

- Найдите все значения вектора `vec3`, которые меньше 4 или больше 10.

```
vec3[vec3 < 4 | vec3 > 10]
```

```
## [1]  3  2  1  3 15  1 11
```

- Возведите в квадрат каждое значение вектора `vec3`.

```
vec3 ^ 2
```

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

- *Возведите в квадрат каждое значение вектора на нечетной позиции и извлеките корень из каждого значения на четной позиции вектора vec3.

  Извлечение корня - это то же самое, что и возведение в степень 0.5.

```
vec3 ^ c(2, 0.5)
```

```
##  [1]  9.000000  2.236068  4.000000  1.000000 64.000000  2.000000 81.000000
##  [8]  3.162278  9.000000  3.872983  1.000000  3.316625
```

- Создайте вектор 2, 4, 6, ... , 18, 20 как минимум 2 новыми способами.

  Знаю, это задание может показаться бессмысленным, но это очень базовая операция, с помощью которой можно, например, разделить данные на две части. Чем больше способов Вы знаете, тем лучше!

```
(1:20)[c(FALSE,TRUE)]
```

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

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

8.6 Работа с пропущенными значениями

- Создайте вектор vec4 со значениями 300, 15, 8, 2, 0, 1, 110:

```
vec4 <- c(300, 15, 8, 20, 0, 1, 110)
vec4
```

```
## [1] 300  15   8  20   0   1 110
```

- Замените все значения vec4, которые больше 20 на NA.

```
vec4[vec4 > 20] <- NA
```

- Проверьте полученный вектор vec4:

```
vec4
```

```
## [1] NA 15  8 20  0  1 NA
```

- Посчитайте сумму `vec4` с помощью функции `sum()`. Ответ `NA` не считается!

```
sum(vec4, na.rm = TRUE)
```

```
## [1] 44
```

8.7 Матрицы

- Создайте матрицу 4x4, состоящую из единиц. Назовите ее `M1`.

```
M1 <- matrix(rep(1, 16), ncol = 4)
M1
```

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

- Поменяйте все некрайние значения матрицы `M1` (то есть значения на позициях [2,2], [2,3], [3,2] и [3,3]) на число 2.

```
M1[2:3, 2:3] <- 2
M1
```

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

- Выделите второй и третий столбик из матрицы `M1`.

```
M1[,2:3]
```

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

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

- Сравните (==) вторую колонку и вторую строчку матрицы `M1`.

```
M1[,2] == M1[2,]
```

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

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

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

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    1    2    3    4    5    6    7    8    9
##  [2,]    2    4    6    8   10   12   14   16   18
##  [3,]    3    6    9   12   15   18   21   24   27
##  [4,]    4    8   12   16   20   24   28   32   36
##  [5,]    5   10   15   20   25   30   35   40   45
##  [6,]    6   12   18   24   30   36   42   48   54
##  [7,]    7   14   21   28   35   42   49   56   63
##  [8,]    8   16   24   32   40   48   56   64   72
##  [9,]    9   18   27   36   45   54   63   72   81
```

```
#
#outer(1:9, 1:9, "*")
#1:9 %o% 1:9
```

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

```
mult_tab[6:8, 3:7]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   18   24   30   36   42
## [2,]   21   28   35   42   49
## [3,]   24   32   40   48   56
```

- *Создайте матрицу с логическими значениями, где `TRUE`, если в этом месте в таблице умножения (`mult_tab`) двузначное число и `FALSE`, если однозначное.

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

```
mult_tab >= 10
```

```
##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
##  [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [2,] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [3,] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [4,] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [5,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [6,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [7,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [8,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [9,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- *Создайте матрицу `mult_tab2`, в которой все значения `tab` меньше 10 заменены на 0.

```
mult_tab2 <- mult_tab
mult_tab2[mult_tab < 10] <- 0
mult_tab2
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    0    0    0    0    0    0    0    0    0
##  [2,]    0    0    0    0   10   12   14   16   18
##  [3,]    0    0    0   12   15   18   21   24   27
##  [4,]    0    0   12   16   20   24   28   32   36
##  [5,]    0   10   15   20   25   30   35   40   45
##  [6,]    0   12   18   24   30   36   42   48   54
##  [7,]    0   14   21   28   35   42   49   56   63
##  [8,]    0   16   24   32   40   48   56   64   72
##  [9,]    0   18   27   36   45   54   63   72   81
```

8.8 Списки

Дан список `list1`:

```
list1 = list(numbers = 1:5, letters = letters, logic = TRUE)
list1
```

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

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $logic
## [1] TRUE
```

- Найдите первый элемент списка `list1`. Ответ должен быть списком длиной один.

```
list1[1]
```

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

- Теперь найдите содержание первого элемента списка `list1` двумя разными способами. Ответ должен быть вектором.

```
list1[[1]]
```

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

```
list1$numbers
```

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

- Теперь возьмите первый элемент содержания первого элемента списка `list1`. Ответ должен быть вектором.

```
list1[[1]][1]
```

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

- Создайте список `list2`, содержащий в себе два списка `list1`. Один из них будет иметь имя `pupa`, а другой — `lupa`.

```
list2 = list(pupa = list1, lupa = list1)
list2
```

```
## $pupa
## $pupa$numbers
## [1] 1 2 3 4 5
##
## $pupa$letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
##
## $pupa$logic
## [1] TRUE
##
##
## $lupa
## $lupa$numbers
## [1] 1 2 3 4 5
##
## $lupa$letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $lupa$logic
## [1] TRUE
```

- *Извлеките первый элемент списка `list2`, из него — второй полэлемент, а из него — третье значение.

```
list2[[1]][[2]][3]
```

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

8.9 Датафрейм

- Запустите команду `data(mtcars)` чтобы загрузить встроенный датафрейм с информацией про автомобили. Каждая строчка датафрейма - модель автомобиля, каждая колонка - отдельная характеристика. Подробнее см. `?mtcars`.

```
data(mtcars)
mtcars
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

```
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Изучите структуру датафрейма `mtcars` с помощью функции `str()`.

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- Найдите значение третьей строчки четвертого столбца датафрейма `mtcars`.

```
mtcars[3, 4]
```

```
## [1] 93
```

- Извлеките первые шесть строчек и первые шесть столбцов датафрейма `mtcars`.

```
mtcars[1:6, 1:6]
```

```
##                   mpg cyl disp  hp drat    wt
## Mazda RX4         21.0   6  160 110 3.90 2.620
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875
## Datsun 710        22.8   4  108  93 3.85 2.320
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215
## Hornet Sportabout 18.7   8  360 175 3.15 3.440
## Valiant           18.1   6  225 105 2.76 3.460
```

- Извлеките колонку `wt` датафрейма `mtcars` - массу автомобиля в тысячах фунтов.

```
mtcars$wt
```

```
##  [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070
## [13] 3.730 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840
## [25] 3.845 1.935 2.140 1.513 3.170 2.770 3.570 2.780
```

- Извлеките колонки из `mtcars` в следующем порядке: `hp`, `mpg`, `cyl`.

```
mtcars[, c("hp", "mpg", "cyl")]
```

```
##                     hp  mpg cyl
## Mazda RX4          110 21.0   6
## Mazda RX4 Wag      110 21.0   6
## Datsun 710          93 22.8   4
## Hornet 4 Drive     110 21.4   6
## Hornet Sportabout  175 18.7   8
## Valiant            105 18.1   6
## Duster 360         245 14.3   8
## Merc 240D           62 24.4   4
## Merc 230            95 22.8   4
## Merc 280           123 19.2   6
## Merc 280C          123 17.8   6
## Merc 450SE         180 16.4   8
## Merc 450SL         180 17.3   8
## Merc 450SLC        180 15.2   8
## Cadillac Fleetwood 205 10.4   8
## Lincoln Continental 215 10.4   8
## Chrysler Imperial  230 14.7   8
## Fiat 128            66 32.4   4
```

```
## Honda Civic          52 30.4   4
## Toyota Corolla       65 33.9   4
## Toyota Corona        97 21.5   4
## Dodge Challenger    150 15.5   8
## AMC Javelin         150 15.2   8
## Camaro Z28          245 13.3   8
## Pontiac Firebird    175 19.2   8
## Fiat X1-9            66 27.3   4
## Porsche 914-2        91 26.0   4
## Lotus Europa        113 30.4   4
## Ford Pantera L      264 15.8   8
## Ferrari Dino        175 19.7   6
## Maserati Bora       335 15.0   8
## Volvo 142E          109 21.4   4
```

- Посчитайте *количество* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
sum(mtcars$cyl == 4)
```

```
## [1] 11
```

- Посчитайте *долю* автомобилей с 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
mean(mtcars$cyl == 4)
```

```
## [1] 0.34375
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) в датафрейме `mtcars`.

```
mtcars[mtcars$hp >= 100, ]
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
```

```
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Найдите все автомобили мощностью не менее 100 лошадиных сил (`hp`) и 4 цилиндрами (`cyl`) в датафрейме `mtcars`.

```
mtcars[mtcars$hp >= 100 & mtcars$cyl == 4, ]
```

```
##               mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Volvo 142E   21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

- Посчитайте максимальную массу (`wt`) автомобиля в выборке, воспользовавшись функцией `max()`:

```
max(mtcars$wt)
```

```
## [1] 5.424
```

- Посчитайте максимальную массу (`wt`) автомобиля в выборке, воспользовавшись функцией `min()`:

```
min(mtcars$wt)
```

```
## [1] 1.513
```

- Найдите строчку датафрейма `mtcars` с самым легким автомобилем.

```
mtcars[mtcars$wt == min(mtcars$wt), ]
```

```
##               mpg cyl disp  hp drat    wt qsec vs am gear carb
## Lotus Europa 30.4   4 95.1 113 3.77 1.513 16.9  1  1    5    2
```

- Извлеките строчки датафрейма `mtcars` с автомобилями, масса которых ниже средней массы.

```
mtcars[mtcars$wt < mean(mtcars$wt), ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Масса автомобиля указана в тысячах фунтов. Создайте колонку `wt_kg` с массой автомобиля в килограммах. Результат округлите до целых значений с помощью функции `round()`.

  1 фунт = 0.45359237 кг.

```
mtcars$wt_kg <- round(mtcars$wt * 1000 * 0.45359237)
mtcars
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb wt_kg
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4  1188
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4  1304
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1  1052
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1  1458
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2  1560
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1  1569
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4  1619
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2  1447
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2  1429
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4  1560
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4  1560
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3  1846
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3  1692
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3  1715
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4  2381
```

```
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4  2460
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4  2424
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1   998
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2   733
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1   832
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1  1118
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2  1597
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2  1558
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4  1742
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2  1744
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1   878
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2   971
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2   686
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4  1438
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6  1256
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8  1619
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2  1261
```

8.10 Условные конструкции

- Создайте вектор `vec5`:

```
vec5 <- c(5, 20, 30, 0, 2, 9)
```

- Создайте новый строковый вектор, где на месте чисел больше 10 в `vec5` будет стоять “большое число”, а на месте остальных чисел — “маленькое число”.

```
ifelse(vec5 > 10, "       ", "        ")
```

```
## [1] "        " "       "   "       "   "        "
## [5] "        " "         "
```

- Загрузите файл heroes_information.csv в переменную `heroes`.

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

- Создайте новою колонку `hair` в `heroes`, в которой будет значение `"Bold"` для тех супергероев, у которых в колонке `Hair.color` стоит `"No Hair"`, и значение `"Hairy"` во всех остальных случаях.

```
heroes$hair <- ifelse(heroes$Hair.color == "No Hair", "Bold", "Hairy")
head(heroes)
```

```
##   X          name Gender Eye.color              Race Hair.color Height
## 1 0        A-Bomb   Male    yellow             Human    No Hair    203
## 2 1    Abe Sapien   Male      blue     Icthyo Sapien    No Hair    191
## 3 2      Abin Sur   Male      blue           Ungaran    No Hair    185
## 4 3   Abomination   Male     green Human / Radiation    No Hair    203
## 5 4       Abraxas   Male      blue     Cosmic Entity      Black     NA
## 6 5 Absorbing Man   Male      blue             Human    No Hair    193
##           Publisher Skin.color Alignment Weight  hair
## 1     Marvel Comics       <NA>      good    441  Bold
## 2 Dark Horse Comics       blue      good     65  Bold
## 3         DC Comics        red      good     90  Bold
## 4     Marvel Comics       <NA>       bad    441  Bold
## 5     Marvel Comics       <NA>       bad     NA Hairy
## 6     Marvel Comics       <NA>       bad    122  Bold
```

- Создайте новою колонку `tall` в `heroes`, в которой будет значение `"tall"` для тех супергероев, у которых в колонке `Height` стоит число больше 190, значение `"short"` для тех супергероев, у которых в колонке `Height` стоит число меньше 170, и значение `"middle"` во всех остальных случаях.

```
# heroes$tall <- dplyr::case_when(
#   heroes$Height > 190 ~ "tall",
#   heroes$Height < 170 ~ "short",
#   TRUE ~ "middle"
# )
heroes$tall <- ifelse(heroes$Height > 190,
                      "tall",
                      ifelse(heroes$Height < 170,
                             "short",
                             "middle"))
```

8.11 Создание функций

- Создайте функцию `plus_one()`, которая принимает число и возвращает это же число + 1.

```
plus_one <- function(x) x + 1
```

- Проверьте функцию `plus_one()` на числе 41.

```
plus_one(41)
```

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

- Создайте функцию `circle_area`, которая вычисляет площадь круга по радиусу согласно формуле πr^2.

```
circle_area <- function(r) pi * r ^ 2
```

- Посчитайте площадь круга с радиусом 5.

```
circle_area(5)
```

```
## [1] 78.53982
```

- Создайте функцию `cels2fahr()`, которая будет превращать градусы по Цельсию в градусы по Фаренгейту.

```
cels2fahr <- function(x) x * 9 / 5 + 32
```

- Проверьте на значениях -100, -40 и 0, что функция `cels2fahr()` работает корректно.

```
cels2fahr(c(-100, -40, 0))
```

```
## [1] -148  -40   32
```

- Напишите функцию `highlight()`, которая принимает на входе строковый вектор, а возвращает тот же вектор, но дополненный значением `"***"` в начале и конце вектора. Лучше всего это рассмотреть на примере:

```
highlight <- function(x) c("***", x, "***")
```

```
highlight(c(" ", "   !"))
```

```
## [1] "***"     " "       "   !" "***"
```

- Теперь сделайте функцию `highlight` более гибкой. Добавьте в нее параметр `wrapper =`, который по умолчанию равен `"***"`. Значение параметра `wrapper =` и будет вставлено в начало и конец вектора.

```
highlight <- function(x, wrapper = "***") c(wrapper, x, wrapper)
```

- Проверьте написанную функцию на векторе `c("  "    !")`.

```
highlight(c(" ", "    !"))
```

```
## [1] "***"     " "       "    !" "***"
```

```
highlight(c(" ", "    !"), wrapper = "__")
```

```
## [1] "__"      " "       "    !" "__"
```

- Создайте функцию `trim()`, которая будет возвращать вектор без первого и последнего значения (вне зависимости от типа данных).

```
trim <- function(x) x[c(-1, -length(x))]
```

- Проверьте, что функция `trim()` работает корректно:

```
trim(1:7)
```

```
## [1] 2 3 4 5 6
```

```
trim(letters)
```

```
##  [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
## [20] "u" "v" "w" "x" "y"
```

- Теперь добавьте в функцию `trim()` параметр `n =` со значением по умолчанию 1. Этот параметр будет обозначать сколько значений нужно отрезать слева и справа от вектора.

```
trim <- function(x, n = 1) x[c(-1:-n, (-length(x)+n-1):-length(x))]
```

- Проверьте полученную функцию:

```
trim(letters)
```

```
##  [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
## [20] "u" "v" "w" "x" "y"
```

```
trim(letters, n = 2)
```

```
##  [1] "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u"
## [20] "v" "w" "x"
```

- Сделайте так, чтобы функция `trim()` работала корректно с `n = 0`, т.е. функция возвращала бы исходный вектор без изменений.

```
trim <- function(x, n = 1) {
  if (n == 0) return(x)
  x[c(-1:-n, (-length(x)+n-1):-length(x))]
}
```

```
trim(letters, n = 0)
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

- *Теперь добавьте проверку на адекватность входных данных: функция `trim()` должна выдавать ошибку, если `n =` меньше нуля или если `n =` слишком большой и отрезает все значения вектора:

```
trim <- function(x, n = 1) {
  if (n < 0) stop("n                !")
  l <- length(x)
  if (n > ceiling(l/2) - 1) stop("n          !")
  if (n == 0) return(x)
  x[c(-1:-n, (-l+n-1):-l)]
}
```

- *Проверьте полученную функцию `trim()`:

```
trim(1:6, 3)
```

```
## Error in trim(1:6, 3): n          !
```

```
trim(1:6, -1)
```

```
## Error in trim(1:6, -1): n                !
```

- Создайте функцию `na_n()`, которая будет возвращать количество `NA` в векторе.

```
na_n <- function(x) sum(is.na(x))
```

- Проверьте функцию `na_n()` на векторе:

```
na_n(c(NA, 3:5, NA, 2, NA))
```

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

- Напишите функцию `factors()`, которая будет возвращать все делители числа в виде числового вектора.

  Здесь может понадобиться оператор для получения остатка от деления: `%%`.

```
factors <- function(x) (1:x)[x %% (1:x) == 0]
```

- Проверьте функцию `factors()` на простых и сложных числах:

```
factors(3)
```

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

```
factors(161)
```

```
## [1]   1   7  23 161
```

```
factors(1984)
```

```
##  [1]    1    2    4    8   16   31   32   62   64  124  248  496  992 1984
```

- *Напишите функцию `is_prime()`, которая проверяет, является ли число простым.

  Здесь может пригодиться функция `any()` - она возвращает `TRUE`, если в векторе есть хотя бы один `TRUE`.

```
is_prime <- function(x) !any(x%%(2:(x-1)) == 0)
#is_prime <- function(x) length(factors(x)) == 2 #                    factors()
```

- Проверьте какие года были для нас простыми, а какие нет:

```
is_prime(2017)
```

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

```
is_prime(2019)
```

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

```
2019/3 #2019        3
```

```
## [1] 673
```

```
is_prime(2020)
```

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

```
is_prime(2021)
```

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

- *Создайте функцию `monotonic()`, которая возвращает `TRUE`, если значения в векторе не убывают (то есть каждое следующее - больше или равно предыдущему) или не возвращают.

  Полезная функция для этого — `diff()` — возвращает разницу соседних значений.

```
monotonic <- function(x) all(diff(x)>=0) | all(diff(x)<=0)
```

```
monotonic(1:7)
```

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

```
monotonic(c(1:5,5:1))
```

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

```
monotonic(6:-1)
```

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

```
monotonic(c(1:5, rep(5, 10), 5:10))
```

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

Бинарные операторы типа + или %in% тоже представляют собой функции. Более того, мы можем создавать свои бинарные операторы! В этом нет особой сложности — нужно все так же создавать функцию (для двух переменных), главное окружать их % и название обрамлять обратными штрихами ‘. Например, можно сделать свой бинарный оператор %notin%, который будет выдавать TRUE, если значения слева *нет* в векторе справа:

```
`%notin%` <- function(x, y) ! (x %in% y)
1:10 %notin% c(1, 4, 5)
```

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

- *Создайте бинарный оператор %without%, который будет возвращать все значения вектора слева без значений вектора справа.

```
`%without%` <- function(x, y) x[!x %in% y]
```

```
c(" ", " ", " ", "   ", " ", "   ") %without% c(" ", " ")
```

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

- *Создайте бинарный оператор %between%, который будет возвращать TRUE, если значение в векторе слева накходится в *диапазоне* значений вектора справа:

```
`%between%` <- function(x, y) x >= min(y) & x <= max(y)
```

```
1:10 %between% c(1, 4, 5)
```

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

8.12 Семейство функций apply()

- Создайте матрицу M2:

```
M2 <- matrix(c(20:11, 11:20), nrow = 5)
M2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   20   15   11   16
## [2,]   19   14   12   17
## [3,]   18   13   13   18
## [4,]   17   12   14   19
## [5,]   16   11   15   20
```

- Посчитайте максимальное значение матрицы M2 по каждой строчке.

```
apply(M2, 1, max)
```

```
## [1] 20 19 18 19 20
```

- Посчитайте максимальное значение матрицы M2 по каждому столбцу.

```
apply(M2, 2, max)
```

```
## [1] 20 15 15 20
```

- Посчитайте среднее значение матрицы M2 по каждой строке.

```
apply(M2, 1, mean)
```

```
## [1] 15.5 15.5 15.5 15.5 15.5
```

- Посчитайте среднее значение матрицы M2 по каждому столбцу.

```
apply(M2, 2, mean)
```

```
## [1] 18 13 13 18
```

- Создайте список list3:

```
list3 <- list(
  a = 1:5,
  b = 0:20,
  c = 4:24,
  d = 6:3,
  e = 6:25
  )
```

- Найдите максимальное значение каждого вектора списка list3.

```
sapply(list3, max)
```

```
##  a  b  c  d  e
##  5 20 24  6 25
```

- Посчитайте сумму каждого вектора списка `list3`.

```
sapply(list3, sum)
```

```
##   a   b   c   d   e
##  15 210 294  18 310
```

- Посчитайте длину каждого вектора списка `list3`.

```
sapply(list3, length)
```

```
##  a  b  c  d  e
##  5 21 21  4 20
```

- Напишите функцию `max_item()`, которая будет принимать на входе список, а возвращать - (первый) самый длинный его элемент.

  Для этого вам может понадобиться функция `which.max()`, которая возвращает индекс максимального значения (первого, если их несколько).

```
max_item <- function (x) x[[which.max(sapply(x, length))]]
```

- Проверьте функцию `max_item()` на списке `list3`.

```
max_item(list3)
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Теперь мы сделаем сложный список `list4`:

```
list4 <- list(1:3, 3:40, list3)
```

- Посчитайте длину каждого вектора в списке, в т.ч. для списка внутри. Результат должен быть списком с такой же структорой, как и изначальный список `list4`.

  Для этого может понадобиться функция `rapply()`: **recursive lapply**

```
rapply(list4, length, how = "list")
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 38
##
## [[3]]
## [[3]]$a
## [1] 5
##
## [[3]]$b
## [1] 21
##
## [[3]]$c
## [1] 21
##
## [[3]]$d
## [1] 4
##
## [[3]]$e
## [1] 20
```

- *Загрузите набор данных `heroes` и посчитайте, сколько `NA` в каждом из столбцов.

  Для этого удобно использовать ранее написанную функцию `na_n()`.

```
sapply(heroes, na_n)
```

```
##          X       name     Gender  Eye.color       Race Hair.color     Height
##          0          0         29        172        304        172        217
##  Publisher Skin.color  Alignment     Weight       hair       tall
##          0        662          7        239        172        217
```

- *Используя ранее написанную функцию `is_prime()`, напишите функцию `prime_numbers()`, которая будет возвращать все простые числа до выбранного числа.

```
is_prime <- function(x) !any(x %% (2:(x - 1)) == 0)
prime_numbers <- function(x) (2:x)[sapply(2:x, is_prime)]
```

```
prime_numbers(200)
```

```
##  [1]   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71
## [20]  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167
```

```
## [39] 173 179 181 191 193 197 199
```

```
library(tidyverse)
heroes <- read_csv("data/heroes_information.csv",
                   na = c("-", "-99"))
powers <- read_csv("data/super_hero_powers.csv")
```

8.13 magrittr::%>%

- Перепишите следующие выражения, используя %>%:

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

```
## [1] 7.416198
```

```
1:10 %>%
  sum() %>%
  sqrt()
```

```
## [1] 7.416198
```

```
abs(min(-5:5))
```

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

```
-5:5 %>%
  min() %>%
  abs()
```

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

```
c("     ", 2, "   ", sqrt(2))
```

```
## [1] "     "       "2"               "   "           "1.4142135623731"
```

```
2 %>% c("     ", ., "   ", sqrt(.))
```

```
## [1] "     "       "2"               "   "           "1.4142135623731"
```

8.14 Выбор строк: dplyr::slice() и dplyr::filter()

- Выберите только те строчки, в которых содержится информация о супергероях тяжелее 500 кг.

```
heroes %>%
  filter(Weight > 500)
```

```
## # A tibble: 6 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   203 Dark~ Male   red         New ~ No Hair       267   DC Comics
## 2   283 Giga~ Female green       <NA>  Red            62.5 DC Comics
## 3   331 Hulk  Male   green       Huma~ Green         244   Marvel C~
## 4   373 Jugg~ Male   blue        Human Red           287   Marvel C~
## 5   549 Red ~ Male   yellow      Huma~ Black         213   Marvel C~
## 6   575 Sasq~ Male   red         <NA>  Orange        305   Marvel C~
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о *женщинах*-супергероях тяжелее 500 кг.

```
heroes %>%
  filter(Weight > 500 & Gender == "Female")
```

```
## # A tibble: 1 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1   283 Giga~ Female green       <NA>  Red            62.5 DC Comics
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

- Выберите только те строчки, в которых содержится информация о супергероях человеческой расы (`"Human"`) женского пола. Из этих супергероев возьмите первые 5.

```
heroes %>%
  filter(Race == "Human" & Gender == "Female") %>%
  slice(1:5)
```

```
## # A tibble: 5 x 11
##      X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##   <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
## 1    38 Arac~ Female blue        Human Blond           175 Marvel C~
## 2    63 Batg~ Female green       Human Red             170 DC Comics
```

```
## 3    65 Batg~ Female green       Human Black           165 DC Comics
## 4    72 Batw~ Female green       Human Red             178 DC Comics
## 5    96 Blac~ Female blue        Human Blond           165 DC Comics
## # ... with 3 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>
```

8.15 Выбор столбцов: `dplyr::select()`

- Выберете первые 4 столбца в powers.

```
powers %>%
  select(1:4)
```

```
## # A tibble: 667 x 4
##    hero_names    Agility `Accelerated Healing` `Lantern Power Ring`
##    <chr>         <lgl>   <lgl>                 <lgl>
##  1 3-D Man       TRUE    FALSE                 FALSE
##  2 A-Bomb        FALSE   TRUE                  FALSE
##  3 Abe Sapien    TRUE    TRUE                  FALSE
##  4 Abin Sur      FALSE   FALSE                 TRUE
##  5 Abomination   FALSE   TRUE                  FALSE
##  6 Abraxas       FALSE   FALSE                 FALSE
##  7 Absorbing Man FALSE   FALSE                 FALSE
##  8 Adam Monroe   FALSE   TRUE                  FALSE
##  9 Adam Strange  FALSE   FALSE                 FALSE
## 10 Agent Bob     FALSE   FALSE                 FALSE
## # ... with 657 more rows
```

- Выберите все столбцы от Reflexes до Empathy в тиббле powers:

```
powers %>%
  select(Reflexes:Empathy)
```

```
## # A tibble: 667 x 7
##    Reflexes Invulnerability `Energy Constru~ `Force Fields` `Self-Sustenanc~
##    <lgl>    <lgl>           <lgl>            <lgl>          <lgl>
##  1 FALSE    FALSE           FALSE            FALSE          FALSE
##  2 FALSE    FALSE           FALSE            FALSE          TRUE
##  3 TRUE     FALSE           FALSE            FALSE          FALSE
##  4 FALSE    FALSE           FALSE            FALSE          FALSE
##  5 FALSE    TRUE            FALSE            FALSE          FALSE
##  6 FALSE    TRUE            FALSE            FALSE          FALSE
##  7 FALSE    TRUE            FALSE            FALSE          FALSE
##  8 FALSE    FALSE           FALSE            FALSE          FALSE
```

```
##  9 FALSE    FALSE           FALSE           FALSE          FALSE
## 10 FALSE    FALSE           FALSE           FALSE          FALSE
## # ... with 657 more rows, and 2 more variables: `Anti-Gravity` <lgl>,
## #   Empathy <lgl>
```

- Выберите все столбцы тиббла powers кроме первого (hero_names):

```
powers %>%
select(!hero_names)
```

```
## # A tibble: 667 x 167
##    Agility `Accelerated He~ `Lantern Power ~ `Dimensional Aw~ `Cold Resistanc~
##    <lgl>   <lgl>            <lgl>            <lgl>            <lgl>
##  1 TRUE    FALSE            FALSE            FALSE            FALSE
##  2 FALSE   TRUE             FALSE            FALSE            FALSE
##  3 TRUE    TRUE             FALSE            FALSE            TRUE
##  4 FALSE   FALSE            TRUE             FALSE            FALSE
##  5 FALSE   TRUE             FALSE            FALSE            FALSE
##  6 FALSE   FALSE            FALSE            TRUE             FALSE
##  7 FALSE   FALSE            FALSE            FALSE            TRUE
##  8 FALSE   TRUE             FALSE            FALSE            FALSE
##  9 FALSE   FALSE            FALSE            FALSE            FALSE
## 10 FALSE   FALSE            FALSE            FALSE            FALSE
## # ... with 657 more rows, and 162 more variables: Durability <lgl>,
## #   Stealth <lgl>, `Energy Absorption` <lgl>, Flight <lgl>, `Danger
## #   Sense` <lgl>, `Underwater breathing` <lgl>, Marksmanship <lgl>, `Weapons
## #   Master` <lgl>, `Power Augmentation` <lgl>, `Animal Attributes` <lgl>,
## #   Longevity <lgl>, Intelligence <lgl>, `Super Strength` <lgl>,
## #   Cryokinesis <lgl>, Telepathy <lgl>, `Energy Armor` <lgl>, `Energy
## #   Blasts` <lgl>, Duplication <lgl>, `Size Changing` <lgl>, `Density
## #   Control` <lgl>, Stamina <lgl>, `Astral Travel` <lgl>, `Audio
## #   Control` <lgl>, Dexterity <lgl>, Omnitrix <lgl>, `Super Speed` <lgl>,
## #   Possession <lgl>, `Animal Oriented Powers` <lgl>, `Weapon-based
## #   Powers` <lgl>, Electrokinesis <lgl>, `Darkforce Manipulation` <lgl>, `Death
## #   Touch` <lgl>, Teleportation <lgl>, `Enhanced Senses` <lgl>,
## #   Telekinesis <lgl>, `Energy Beams` <lgl>, Magic <lgl>, Hyperkinesis <lgl>,
## #   Jump <lgl>, Clairvoyance <lgl>, `Dimensional Travel` <lgl>, `Power
## #   Sense` <lgl>, Shapeshifting <lgl>, `Peak Human Condition` <lgl>,
## #   Immortality <lgl>, Camouflage <lgl>, `Element Control` <lgl>,
## #   Phasing <lgl>, `Astral Projection` <lgl>, `Electrical Transport` <lgl>,
## #   `Fire Control` <lgl>, Projection <lgl>, Summoning <lgl>, `Enhanced
## #   Memory` <lgl>, Reflexes <lgl>, Invulnerability <lgl>, `Energy
## #   Constructs` <lgl>, `Force Fields` <lgl>, `Self-Sustenance` <lgl>,
## #   `Anti-Gravity` <lgl>, Empathy <lgl>, `Power Nullifier` <lgl>, `Radiation
## #   Control` <lgl>, `Psionic Powers` <lgl>, Elasticity <lgl>, `Substance
## #   Secretion` <lgl>, `Elemental Transmogrification` <lgl>,
```

```
## #   `Technopath/Cyberpath` <lgl>, `Photographic Reflexes` <lgl>, `Seismic
## #   Power` <lgl>, Animation <lgl>, Precognition <lgl>, `Mind Control` <lgl>,
## #   `Fire Resistance` <lgl>, `Power Absorption` <lgl>, `Enhanced
## #   Hearing` <lgl>, `Nova Force` <lgl>, Insanity <lgl>, Hypnokinesis <lgl>,
## #   `Animal Control` <lgl>, `Natural Armor` <lgl>, Intangibility <lgl>,
## #   `Enhanced Sight` <lgl>, `Molecular Manipulation` <lgl>, `Heat
## #   Generation` <lgl>, Adaptation <lgl>, Gliding <lgl>, `Power Suit` <lgl>,
## #   `Mind Blast` <lgl>, `Probability Manipulation` <lgl>, `Gravity
## #   Control` <lgl>, Regeneration <lgl>, `Light Control` <lgl>,
## #   Echolocation <lgl>, Levitation <lgl>, `Toxin and Disease Control` <lgl>,
## #   Banish <lgl>, `Energy Manipulation` <lgl>, `Heat Resistance` <lgl>,
## #   `Natural Weapons` <lgl>, ...
```

8.16 Сортировка строк: `dplyr::arrange()`

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по возрастанию* `Height`.

```
heroes %>%
  select(name, Gender, Height) %>%
  arrange(Height)
```

```
## # A tibble: 734 x 3
##    name            Gender Height
##    <chr>           <chr>   <dbl>
##  1 Utgard-Loki     Male     15.2
##  2 Bloodwraith     Male     30.5
##  3 King Kong       Male     30.5
##  4 Anti-Monitor    Male     61
##  5 Giganta         Female   62.5
##  6 Krypto          Male     64
##  7 Yoda            Male     66
##  8 Jack-Jack       Male     71
##  9 Howard the Duck Male     79
## 10 Godzilla        <NA>    108
## # ... with 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки *по убыванию* `Height`.

```
heroes %>%
  select(name, Gender, Height) %>%
  arrange(desc(Height))
```

```
## # A tibble: 734 x 3
##    name          Gender Height
##    <chr>         <chr>   <dbl>
##  1 Fin Fang Foom Male      975
##  2 Galactus      Male      876
##  3 Groot         Male      701
##  4 MODOK         Male      366
##  5 Wolfsbane     Female    366
##  6 Onslaught     Male      305
##  7 Sasquatch     Male      305
##  8 Ymir          Male      305.
##  9 Rey           Female    297
## 10 Juggernaut    Male      287
## # ... with 724 more rows
```

- Выберите из тиббла `heroes` колонки `name`, `Gender`, `Height` и отсортируйте строчки сначала по `Gender`, затем *по убыванию* `Height`.

```
heroes %>%
  select(name, Gender, Height) %>%
  arrange(Gender, desc(Height))
```

```
## # A tibble: 734 x 3
##    name      Gender Height
##    <chr>     <chr>   <dbl>
##  1 Wolfsbane Female    366
##  2 Rey       Female    297
##  3 Bloodaxe  Female    218
##  4 Thundra   Female    218
##  5 Hela      Female    213
##  6 Frenzy    Female    211
##  7 She-Hulk  Female    201
##  8 Ardina    Female    193
##  9 Starfire  Female    193
## 10 Valkyrie  Female    191
## # ... with 724 more rows
```

8.17 Уникальные значения: `dplyr::distinct()`

- Извлеките уникальные значения столбца `Eye color` из тиббла `heroes`.

```
heroes %>%
  distinct(`Eye color`)
```

```
## # A tibble: 23 x 1
##    `Eye color`
##    <chr>
##  1 yellow
##  2 blue
##  3 green
##  4 brown
##  5 <NA>
##  6 red
##  7 violet
##  8 white
##  9 purple
## 10 black
## # ... with 13 more rows
```

- Извлеките уникальные значения столбца `Hair color` из тиббла `heroes`.

```
heroes %>%
  distinct(`Hair color`)
```

```
## # A tibble: 30 x 1
##    `Hair color`
##    <chr>
##  1 No Hair
##  2 Black
##  3 Blond
##  4 Brown
##  5 <NA>
##  6 White
##  7 Purple
##  8 Orange
##  9 Pink
## 10 Red
## # ... with 20 more rows
```

8.18 Создание колонок: `dplyr::mutate()` и `dplyr::transmute()`

- Создайте колонку `height_m` с ростом супергероев в метрах, затем выберите только колонки `name` и `height_m`.

```
heroes %>%
  mutate(height_m = Height/100) %>%
  select(name, height_m)
```

```
## # A tibble: 734 x 2
##    name          height_m
##    <chr>            <dbl>
##  1 A-Bomb            2.03
##  2 Abe Sapien        1.91
##  3 Abin Sur          1.85
##  4 Abomination       2.03
##  5 Abraxas          NA
##  6 Absorbing Man     1.93
##  7 Adam Monroe      NA
##  8 Adam Strange      1.85
##  9 Agent 13          1.73
## 10 Agent Bob         1.78
## # ... with 724 more rows
```

- Создайте новою колонку hair в heroes, в которой будет значение “Bold” для тех супергероев, у которых в колонке Hair.color стоит “No Hair”, и значение “Hairy” во всех остальных случаях. Затем выберите только колонки `name`, `Hair color`, `hair`.

```
heroes %>%
  mutate(hair = ifelse(`Hair color` == "No Hair", "Bold", "Hairy")) %>%
  select(name, `Hair color`, hair)
```

```
## # A tibble: 734 x 3
##    name          `Hair color` hair
##    <chr>         <chr>        <chr>
##  1 A-Bomb        No Hair      Bold
##  2 Abe Sapien    No Hair      Bold
##  3 Abin Sur      No Hair      Bold
##  4 Abomination   No Hair      Bold
##  5 Abraxas       Black        Hairy
##  6 Absorbing Man No Hair      Bold
##  7 Adam Monroe   Blond        Hairy
##  8 Adam Strange  Blond        Hairy
##  9 Agent 13      Blond        Hairy
## 10 Agent Bob     Brown        Hairy
## # ... with 724 more rows
```

8.19 Агрегация: `dplyr::group_by() %>% summarise()`

- Посчитайте количество супергероев по расам и отсортируйте по убыванию. Извлеките первые 5 строк.

```
heroes %>%
  count(Race, sort = TRUE) %>%
  slice(1:5)
```

```
## # A tibble: 5 x 2
##   Race              n
##   <chr>         <int>
## 1 <NA>            304
## 2 Human           208
## 3 Mutant           63
## 4 God / Eternal    14
## 5 Cyborg           11
```

- Посчитайте средний пост по полу.

```
heroes %>%
  group_by(Gender) %>%
  summarise(height_mean = mean(Height, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   Gender height_mean
##   <chr>        <dbl>
## 1 Female        175.
## 2 Male          192.
## 3 <NA>          177.
```

8.20 Операции с несколькими колонками: `across()`

- Посчитайте количество `NA` в каждой колонке, группируя по полу (`Gender`).

```
na_n <- function(x) sum(is.na(x))
heroes %>%
  group_by(Gender) %>%
  summarise(across(everything(), na_n))
```

```
## # A tibble: 3 x 11
##   Gender    X1  name `Eye color`  Race `Hair color` Height Publisher
##   <chr>  <int> <int>       <int> <int>        <int>  <int>     <int>
## 1 Female     0     0          41    98           38     56         0
## 2 Male       0     0         121   184          123    147         0
## 3 <NA>       0     0          10    22           11     14         0
## # ... with 3 more variables: `Skin color` <int>, Alignment <int>, Weight <int>
```

- Посчитайте количество NA в каждой колонке, которая заканчивается на "color", группируя по полу (Gender).

```
na_n <- function(x) sum(is.na(x))
heroes %>%
  group_by(Gender) %>%
  summarise(across(ends_with("color"), na_n))
```

```
## # A tibble: 3 x 4
##   Gender `Eye color` `Hair color` `Skin color`
##   <chr>        <int>        <int>        <int>
## 1 Female          41           38          186
## 2 Male           121          123          449
## 3 <NA>            10           11           27
```

- Создайте из тиббла heroes новый тиббл с колонками name, Height и Weight, где для каждого героя содержится значение " ", если его рост или вес выше среднего по колонке и " ", если ниже или равен среднему.

```
higher_than_average <- function(x) ifelse(x > mean(x, na.rm = TRUE),
                                          "  ",
                                          "  ")
heroes %>%
  transmute(name,
            across(c(Height, Weight),
                   higher_than_average))
```

```
## # A tibble: 734 x 3
##    name          Height        Weight
##    <chr>         <chr>         <chr>
##  1 A-Bomb
##  2 Abe Sapien
##  3 Abin Sur
##  4 Abomination
##  5 Abraxas       <NA>          <NA>
##  6 Absorbing Man
##  7 Adam Monroe   <NA>          <NA>
##  8 Adam Strange
##  9 Agent 13
## 10 Agent Bob
## # ... with 724 more rows
```

- Создайте из тиббла heroes новый тиббл с колонками Gender, name, Height и Weight, где для каждого героя содержится значение " ", если его

рост или вес выше среднего по колонке и " ", если ниже или равен среднему *внутри соответствующей группы по полу*.

```
heroes %>%
  group_by(Gender) %>%
  transmute(name,
            across(c(Height, Weight),
                   higher_than_average))
```

```
## # A tibble: 734 x 4
## # Groups:   Gender [3]
##    Gender name          Height        Weight
##    <chr>  <chr>         <chr>         <chr>
##  1 Male   A-Bomb
##  2 Male   Abe Sapien
##  3 Male   Abin Sur
##  4 Male   Abomination
##  5 Male   Abraxas       <NA>          <NA>
##  6 Male   Absorbing Man
##  7 Male   Adam Monroe   <NA>          <NA>
##  8 Male   Adam Strange
##  9 Female Agent 13
## 10 Male   Agent Bob
## # ... with 724 more rows
```

8.21 Соединение датафреймов: *_join {#solution_join}

Создайте тиббл `web_creators`, в котором будут супергерои, которые могут плести паутину, т.е. у них стоит `TRUE` в колонке `Web Creation` в тиббле `powers`.

```
powers_web <- powers %>%
  select(hero_names, `Web Creation`)
web_creators <- left_join(heroes, powers_web, by = c("name" = "hero_names")) %>%
  filter(`Web Creation`)
web_creators
```

```
## # A tibble: 16 x 12
##       X1 name  Gender `Eye color` Race  `Hair color` Height Publisher
##    <dbl> <chr> <chr>  <chr>       <chr> <chr>         <dbl> <chr>
##  1    33 Anti~ Male   blue        Symb~ Blond           229 Marvel C~
##  2    38 Arac~ Female blue        Human Blond           175 Marvel C~
##  3   161 Carn~ Male   green       Symb~ Red             185 Marvel C~
##  4   335 Hybr~ Male   brown       Symb~ Black           175 Marvel C~
```

```
##  5   479 Myst~ Male   brown       Human No Hair         180 Marvel C~
##  6   580 Scar~ Male   brown       Clone Brown           193 Marvel C~
##  7   597 Silk  Female brown       Human Black            NA Marvel C~
##  8   620 Spid~ Female blue        Human Brown           170 Marvel C~
##  9   621 Spid~ Female blue        Human Blond           165 Marvel C~
## 10   622 Spid~ Male   hazel       Human Brown           178 Marvel C~
## 11   623 Spid~ <NA>   red         Human Brown           178 Marvel C~
## 12   624 Spid~ Male   brown       Human Black           157 Marvel C~
## 13   673 Toxin Male   blue        Symb~ Brown           188 Marvel C~
## 14   674 Toxin Male   black       Symb~ Blond           191 Marvel C~
## 15   689 Venom Male   blue        Symb~ Strawberry ~    191 Marvel C~
## 16   692 Veno~ Male   <NA>        Symb~ <NA>            226 Marvel C~
## # ... with 4 more variables: `Skin color` <chr>, Alignment <chr>, Weight <dbl>,
## #   `Web Creation` <lgl>
```

8.22 Tidy data

- Для начала создайте тиббл `heroes_weight`, скопировав код:

```
heroes_weight <- heroes %>%
  filter(Publisher %in% c("DC Comics", "Marvel Comics")) %>%
  group_by(Gender, Publisher) %>%
  summarise(weight_mean = mean(Weight, na.rm = TRUE)) %>%
  drop_na()
heroes_weight
```

```
## # A tibble: 4 x 3
## # Groups:   Gender [2]
##   Gender Publisher     weight_mean
##   <chr>  <chr>               <dbl>
## 1 Female DC Comics            76.8
## 2 Female Marvel Comics        80.1
## 3 Male   DC Comics           113.
## 4 Male   Marvel Comics       134.
```

  Функция `drop_na()` позволяет выбросить все строчки, в которых встречается `NA`.

- Превратите тиббл `heroes_weight` в широкий тиббл:

```
heroes_weight %>%
  pivot_wider(names_from = "Publisher", values_from = "weight_mean")
```

```
## # A tibble: 2 x 3
```

```
## # Groups:   Gender [2]
##   Gender `DC Comics` `Marvel Comics`
##   <chr>        <dbl>           <dbl>
## 1 Female        76.8            80.1
## 2 Male         113.            134.
```

- Затем превратите его обратно в длинный тиббл:

```
heroes_weight %>%
  pivot_wider(names_from = "Publisher", values_from = "weight_mean") %>%
  pivot_longer(cols = !Gender,
               names_to = "Publisher",
               values_to = "weight_mean")
```

```
## # A tibble: 4 x 3
## # Groups:   Gender [2]
##   Gender Publisher     weight_mean
##   <chr>  <chr>               <dbl>
## 1 Female DC Comics            76.8
## 2 Female Marvel Comics        80.1
## 3 Male   DC Comics           113.
## 4 Male   Marvel Comics       134.
```

Литература

Adler, J. (2010). *R in a nutshell: A desktop quick reference*. "O'Reilly Media, Inc.".

Baesens, B., Van Vlasselaer, 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.