home\_work

DEAD|LINE

Zarva\_I

2024-09-17

##START

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

summary(cars)

## speed dist   
## Min. : 4.0 Min. : 2.00   
## 1st Qu.:12.0 1st Qu.: 26.00   
## Median :15.0 Median : 36.00   
## Mean :15.4 Mean : 42.98   
## 3rd Qu.:19.0 3rd Qu.: 56.00   
## Max. :25.0 Max. :120.00

## Including Plots

You can also embed plots, for example:



Note that the echo = FALSE parameter was added to the code chunk to prevent printing of the R code that generated the plot.

##Domashka

citation(“ggplot2”) .libPaths() library() source(“<https://bioconductor.org/biocLite.R>”)

###Bioconductor

if (!require("BiocManager", quietly = TRUE)) #установил Bioconductor  
 install.packages("BiocManager")  
  
BiocManager::install("BiocVersion")

## Bioconductor version 3.19 (BiocManager 1.30.23), R 4.4.1 (2024-06-14 ucrt)

## Warning: package(s) not installed when version(s) same as or greater than current; use  
## `force = TRUE` to re-install: 'BiocVersion'

## Installation paths not writeable, unable to update packages  
## path: C:/Program Files/R/R-4.4.1/library  
## packages:  
## boot, foreign, MASS, nlme, survival

## Old packages: 'BiocManager', 'cpp11', 'curl', 'data.table', 'digest', 'ps',  
## 'ragg', 'rmarkdown', 'tinytex', 'xfun'

###dplyr

#mutate() — изменяет переменные, добавляет новые;  
#select() — выбирает переменные;  
#filter() — фильтрует объекты по условиям;  
#summarise() — вычисляет сводные статистики;  
#arrange() — сортирует по переменным;  
#group\_by() — группирует по значениям переменных;  
#\*\_join() — группа глаголов для склеивания двух таблиц по ключу.  
  
#Пакет позволяет:  
#Поворачивать данные, то есть, преобразовывать их в длинный и широкий форматы: #pivot\_longer(), pivot\_wider();  
#Разворачивать данные из вложенных списков в простые таблицы: unnest\_longer(), unnest\_wider();  
#Наоборот, делать из таблиц вложенные переменные: nest(), unnest();  
#Разделять и объединять столбцы по разделителю строк: separate(), unite();  
#Заполнять отсутствующие значения определёнными значениями или удалять их: complete(), drop\_na(), fill(), replace\_na().

###readr, readxl, haven

#Три основных пакета, которые помогают читать данные.  
  
#readr предназначен для чтения самых распространённых форматов данных: \*.csv, \*.txt, \*.tsv;  
#redxl помогает читать файлы \*.xlsx или, иными словами, всё, что пересылается в формате Excel;  
#haven читает форматы \*.sas7bdat, \*.sap, \*.dta, \*.sav, \*.por, то есть данные из SAS.

###purrr

#Этот пакет может поначалу казаться сложным для понимания, однако на самом деле он просто расширяет функционал уже известного семейства функций \*apply().  
#Функция map() и её расширения позволяют итерироваться по элементам векторов или списков самыми разными способами

###tibble

#В базовом R уже есть data.frame, однако tibble позволяет создавать гораздо более приятные для работы таблицы данных.  
  
#В отличие от data.frame он не приводит строки к факторам автоматически. Это нужно сделать самостоятельно, однако таким образом мы всегда в точности знаем, что находится в каждой переменной датафрейма;  
#Имена переменных остаются такими же, какими были. Например, в переменной с названием "variable name" пробел не будет заменён на точку, вместо этого имя будет окружено апострофами (обычно находятся на букве "ё"), что сохраняет ожидания от названий;  
#tibble оценивает переменные лениво. По существу, это значит, что при создании мы можем объявлять одну переменную на основе другой (но эта другая должна быть указана первой).

###ggplot2 и ggpubr

#Для визуализации данных существует сразу два пакета: базовый ggplot, содержащий в себе почти всю необходимую для создания графики функциональность.  
#Дополнительные интересные особенности вроде автоматического расчёта и добавления на график p-value, реализованы в пакете ggpubr.

###flextable

#Необходимо ещё и напечатать эти таблицы в виде, пригодном для восприятия коллегами без лишних проблем. Именно для этого нужен пакет flextable

###stringr

#Очень часто в данных есть строковые переменные, в которых записана некая текстовая информация, важная нам для исследования. Например, в одной ячейке записаны все возможные названия препарата, и стоит задача взять только тех участников исследования, у которых название препарата содержит подстроку "циклин".

###lubridate

#Помимо строковых данных часто встречаются и даты. С ними нельзя работать как с факторами, строками или, тем более, числами. Для этого существует специальная библиотека, которая может даже вычислить разницу между двумя датами с учётом високосных годов.

###DescTools, psych

#Существуют пакеты, в которых собраны статистические функции, позволяющие чуть ли не в одну строку подготовить базовый статистический отчёт.  
#В DescTools мы найдём полезные функции для статистических тестов и доверительных интервалов, а в psych функции для корреляционных матриц и расчёта сразу группы статистик.

###read.\*

#Выше мы видим пример файла csv с разделителем-запятой. Обсудим конкретно, какими функциями читается каждый формат:  
  
#read.csv() читает csv с запятой в качестве разделителя;  
#read.csv2() читает csv с точкой с запятой в качестве разделителя (кстати, этот же формат отлично читает Excel, сразу разбивая его на столбцы);  
#read.tsv() читает csv со знаком табуляции в качестве разделителя (часто этот формат сохраняют в файле с расширением .txt).  
#Однако, в каждой функции при этом можно указать параметры sep, quote, dec, которые, соответственно, устанавливают знаки: разделителя, кавычек, десятичного разделителя.

###write.\*

#write.csv(), write.csv2() печатают датафрейм в файлы csv с соответствующими разделителями, но их использование (особенно на Windows) может приводить к забавной вещи — поломке кодировки, когда символы в Excel не читаются человеком. Как решить эту проблему, мы узнаем в следующих шагах.

read\_tsv("data/raw/data\_tsv.tsv", skip = 0, n\_max = 10, col\_names = TRUE)

## Rows: 10 Columns: 13  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: "\t"  
## chr (3): Группа, Пол, Группа крови  
## dbl (10): Возраст, Рост, Базофилы\_E1, Эозинофилы\_E1, Гемоглобин\_E1, Эритроци...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

## # A tibble: 10 × 13  
## Группа Возраст Пол Рост `Группа крови` Базофилы\_E1 Эозинофилы\_E1  
## <chr> <dbl> <chr> <dbl> <chr> <dbl> <dbl>  
## 1 Группа 1 31 Женский 174 A (II) 0.422 0.646  
## 2 Группа 1 28 Женский 157 A (II) 0.327 4.97   
## 3 Группа 1 33 Женский 166 <NA> 0.799 3.39   
## 4 Группа 1 26 Женский 168 O (I) 0.0237 4.54   
## 5 Группа 1 33 Женский 170 A (II) 0.664 3.32   
## 6 Группа 1 28 Мужской 172 B (III) 0.481 2.79   
## 7 Группа 1 27 Мужской 157 A (II) 0.890 2.34   
## 8 Группа 1 31 Мужской 174 <NA> 0.858 3.98   
## 9 Группа 1 23 Женский 175 A (II) 0.383 3.39   
## 10 Группа 1 29 Женский 172 A (II) 0.281 7.94   
## # ℹ 6 more variables: Гемоглобин\_E1 <dbl>, Эритроциты\_E1 <dbl>,  
## # Базофилы\_E2 <dbl>, Эозинофилы\_E2 <dbl>, Гемоглобин\_E2 <dbl>,  
## # Эритроциты\_E2 <dbl>

###iris

library(datasets)  
data(iris)  
summary(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width   
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100   
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300   
## Median :5.800 Median :3.000 Median :4.350 Median :1.300   
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199   
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800   
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500   
## Species   
## setosa :50   
## versicolor:50   
## virginica :50   
##   
##   
##

#write\_csv(data, "data/raw/data\_csv.csv")  
  
#write\_excel\_csv(data, "data/raw/data\_csv.csv")  
  
#write\_csv2(data, "data/raw/data\_csv2.csv")  
  
#write\_excel\_csv2(data, "data/raw/data\_csv2.csv")  
  
#Ничего не работает...

###read\_excel

read\_excel("data/raw/data\_excel.xlsx", sheet = "data\_csv2")

## # A tibble: 100 × 13  
## Группа Возраст Пол Рост `Группа крови` Базофилы\_E1 Эозинофилы\_E1  
## <chr> <dbl> <chr> <dbl> <chr> <chr> <chr>   
## 1 Группа 1 31 Женский 174 A (II) 0,4222 0,6465   
## 2 Группа 1 28 Женский 157 A (II) 0,3270 4,9742   
## 3 Группа 1 33 Женский 166 NA 0,7994 3,3875   
## 4 Группа 1 26 Женский 168 O (I) 0,0237 4,5403   
## 5 Группа 1 33 Женский 170 A (II) 0,6636 3,3159   
## 6 Группа 1 28 Мужской 172 B (III) 0,4810 2,7863   
## 7 Группа 1 27 Мужской 157 A (II) 0,8899 2,3432   
## 8 Группа 1 31 Мужской 174 NA 0,8576 3,9788   
## 9 Группа 1 23 Женский 175 A (II) 0,3832 3,3896   
## 10 Группа 1 29 Женский 172 A (II) 0,2812 7,9352   
## # ℹ 90 more rows  
## # ℹ 6 more variables: Гемоглобин\_E1 <chr>, Эритроциты\_E1 <chr>,  
## # Базофилы\_E2 <chr>, Эозинофилы\_E2 <chr>, Гемоглобин\_E2 <chr>,  
## # Эритроциты\_E2 <chr>

###xlsx::write.xlsx()

#Существует несколько пакетов, позволяющих записывать данные в книги Excel: xlsx, openxlsx, writexl. Однако, здесь мы будем использовать openxlsx, поскольку он не требует установки Java и наиболее просто устанавливается почти на всех компьютерах.  
  
#write.xlsx(data, "data\_excel.xlsx", colNames = TRUE)  
  
#Вот и всё. У нас появляется файл Excel с записанными данными. Перейдём к бонусу для тех, у кого установлен Java.  
  
#openxlsx::write.xlsx(), openxlsx::write.xlsx2()  
  
#Прежде всего отметим, что дублирующая функция с цифрой 2 используется преимущественно для того, чтобы быстро записывать крайне большие датафреймы (более, чем 100 тысяч ячеек).  
  
#write.xlsx(data, "data\_excel.xlsx", sheetName = "data", col.names = TRUE, row.names = TRUE, append = FALSE)  
  
#Функция создаёт книгу Excel, а в ней лист с соответствующим именем. Аргументы col.names и row.names говорят, нужно ли записывать в файл имена столбцов и имена строк соответственно. Аргумент append нужен для того, чтобы...  
  
#write.xlsx(data, "data\_excel.xlsx", sheetName = "data\_2", col.names = TRUE, row.names = TRUE, append = TRUE)  
  
#...добавлять новые листы к уже существующей книге.   
  
#haven::read\_spss(), haven::read\_sas()  
  
#Очень редко, но приходится читать данные, которые выгружены напрямую из SPSS или SAS.  
  
#haven::write\_sav()   
  
#Точно так же можно записать данные, чтобы наш предполагаемый коллега мог загрузить их в SPSS.

###mean()

mean(c(20, 68, 45, 76, 41, 36, 13, 52, 77, 53, 70, 73))

## [1] 52

a1 <- c(1, -1, 5, -12, -12, 3, 8, -10, 0)  
a2 <- c(76, 65, 71, 16, 60, 29, 71, 46, 45, 41)  
a3 <- c(-2, 16, -3, 16, -9, 7, 31)  
a4 <- c(-19, -9, 19, 5, -14, 0, 34, -8, 34, 24, -11, 8, 33, 12, -6)  
a5 <- c(NA, NA, NA, NA, NA, NA, 3, NA, NA)  
a6 <- c(-13, 19, -24, NA, 30, 64, -53, NA, 50, 31, -58, -34, -3, -34, 77)  
  
mean(a1)

## [1] -2

mean(a2)

## [1] 52

mean(a3)

## [1] 8

mean(a4)

## [1] 6.8

mean(a5)

## [1] NA

###median()

b1 <- c(-92, -50, 54, 55, 84, 52, -55, -23, 36, -11, 22, 11, -7)  
b2 <- c(1, 9, NA, 88, 2, NA, 42, NA, 4, 68, NA)  
b3 <- c(-15, 71, 77, 36, 66, -21, -48, -8)  
b4 <- c(19, 89, 78, 38, 8, 17, 25, 60, 8, 43, 29, 6, 62, 41, 69, 97, 61, 83, 25, 24)  
b5 <- c(-91, -33, 13, 34, 34, 75, -80, -35, -90, -72, 70, 67, -100, -94, -18)  
  
median(b1)

## [1] 11

median(b2, na.rm = TRUE)

## [1] 9

median(b3)

## [1] 14

median(b4)

## [1] 39.5

median(b5)

## [1] -33

###min(), max()

c1 <- c(68.92, 44.15, 34.2, 34.12, 37.7, 73.95, 36.9, 59.26, 31.06, 55.79, 73.92, 68.04, 53.73, 90.7, 39.66)  
c2 <- c(90.48, 31.16, 44.4, 21.94, 84.37, 53.15, 81.15, 47.86, 63.23, 46.75, 102.73)  
c3 <- c(48.11, 45.3, 58.42, 51.64, 62.07, 57.26, 49.69, 93.29, 81.18, 44.78, 55.1, 76.74, 58.08)  
c4 <- c(17.24, 35.77, 57.57, 30.15, 43.27, 77.56, 72.19, 40.45, 46.2, 39.92)  
c5 <- c(60.22, 31.91, 72.71, 52.49, 46.21, 60.39, 60.09)  
  
min(c1, na.rm = FALSE)

## [1] 31.06

max(c1, na.rm = FALSE)

## [1] 90.7

min(c2, na.rm = FALSE)

## [1] 21.94

max(c2, na.rm = FALSE)

## [1] 102.73

min(c3, na.rm = FALSE)

## [1] 44.78

max(c3, na.rm = FALSE)

## [1] 93.29

min(c4, na.rm = FALSE)

## [1] 17.24

max(c4, na.rm = FALSE)

## [1] 77.56

min(c5, na.rm = FALSE)

## [1] 31.91

max(c5, na.rm = FALSE)

## [1] 72.71

###quantile()

d1 <- c(80.94, 44.46, 46.33, 65.1, 66.42, 104.43, 53.15, 48.41, 12.88, 51.1, 43.03, 40.3, 33.71, 55.1, 22.17)  
d2 <- c(26.17, 97.73, 24.81, 53.62, 87.72, 45.19, 45.7, 69.63, 36.76, 7.17)  
d3 <- c(63.92, 35.85, 26.9, 48.92, 43.1, 66.94, 47.06, 56.54, 29.1, 58.88)  
d4 <- c(32.05, 93.85, 85.52, 56.69, 23.69, 11.29, 51.44, 63.09, 65.65, 35.73, 60.15, 30.93, -4.2)  
  
quantile(d1, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE, type = 7)

## 0% 25% 50% 75% 100%   
## 12.880 41.665 48.410 60.100 104.430

quantile(d2, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE, type = 7)

## 0% 25% 50% 75% 100%   
## 7.1700 28.8175 45.4450 65.6275 97.7300

quantile(d3, probs = seq(0, 1, 0.025), na.rm = FALSE, names = TRUE, type = 7)

## 0% 2.5% 5% 7.5% 10% 12.5% 15% 17.5%   
## 26.90000 27.39500 27.89000 28.38500 28.88000 29.94375 31.46250 32.98125   
## 20% 22.5% 25% 27.5% 30% 32.5% 35% 37.5%   
## 34.50000 36.03125 37.66250 39.29375 40.92500 42.55625 43.69400 44.58500   
## 40% 42.5% 45% 47.5% 50% 52.5% 55% 57.5%   
## 45.47600 46.36700 47.15300 47.57150 47.99000 48.40850 48.82700 50.25350   
## 60% 62.5% 65% 67.5% 70% 72.5% 75% 77.5%   
## 51.96800 53.68250 55.39700 56.71550 57.24200 57.76850 58.29500 58.82150   
## 80% 82.5% 85% 87.5% 90% 92.5% 95% 97.5%   
## 59.88800 61.02200 62.15600 63.29000 64.22200 64.90150 65.58100 66.26050   
## 100%   
## 66.94000

quantile(d4, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE, type = 7)

## 0% 25% 50% 75% 100%   
## -4.20 30.93 51.44 63.09 93.85

###var(), sd()

e1 <- c(47.44, 62.44, 20.44, 72.75, 77.86, 13.74, 28.2, 50.47, 59.19, 69.04)  
e2 <- c(49.31, 44.47, 14.04, 44.43, 49.18, 40.73, 44.65, 41.91, 80.38, 80.09)  
e3 <- c(57.96, 20.81, 8.92, 14.03, 61.02, 25.69, 21.22, 49.56, 25.64, 28.31)  
e4 <- c(76.22, 65, 19.69, 29.84, 37.18, 70.93, 64.78, 61.66, 49.03, 51.56)  
e5 <- c(92.11, 56, 47.89, 62.96, 47.41, 37.05, 73.96, 53, 52.37, 85.23)  
  
var(e1, na.rm = TRUE)

## [1] 507.3136

sd(e1, na.rm = TRUE)

## [1] 22.52362

var(e2, na.rm = TRUE)

## [1] 372.5051

sd(e2, na.rm = TRUE)

## [1] 19.30039

var(e3, na.rm = TRUE)

## [1] 334.519

sd(e3, na.rm = TRUE)

## [1] 18.28986

var(e4, na.rm = TRUE)

## [1] 347.8641

sd(e4, na.rm = TRUE)

## [1] 18.65111

var(e5, na.rm = TRUE)

## [1] 313.3289

sd(e5, na.rm = TRUE)

## [1] 17.7011

###IQR()

f1 <- c(80.94, 44.46, 46.33, 65.1, 66.42, 104.43, 53.15, 48.41, 12.88, 51.1, 43.03, 40.3, 33.71, 55.1, 22.17)  
f2 <- c(26.17, 97.73, 24.81, 53.62, 87.72, 45.19, 45.7, 69.63, 36.76, 7.17)  
f3 <- c(63.92, 35.85, 26.9, 48.92, 43.1, 66.94, 47.06, 56.54, 29.1, 58.88)  
f4 <- c(32.05, 93.85, 85.52, 56.69, 23.69, 11.29, 51.44, 63.09, 65.65, 35.73, 60.15, 30.93, -4.2)  
  
IQR(f1, na.rm = FALSE, type = 7)

## [1] 18.435

IQR(f2, na.rm = FALSE, type = 7)

## [1] 36.81

IQR(f3, na.rm = FALSE, type = 7)

## [1] 20.6325

IQR(f4, na.rm = FALSE, type = 7)

## [1] 32.16

###length()

#Приведём лайфхак, косвенно связанный с количеством значений вектора.   
  
#Зачастую в работе нам нужно включить в таблицы количество значений без пропущенных значений и количество именно пропущенных значений:  
  
#sum(!is.na(vec)): количество значений без учёта пропущенных;  
#sum(is.na(vec)): количество пропущенных значений.

###Стандартная ошибка среднего sd(x)/sqrt(length(x))

d1 <- c(47.44, 62.44, 20.44, 72.75, 77.86, 13.74, 28.2, 50.47, 59.19, 69.04)  
d2 <- c(49.31, 44.47, 14.04, 44.43, 49.18, 40.73, 44.65, 41.91, 80.38, 80.09)  
d3 <- c(57.96, 20.81, 8.92, 14.03, 61.02, 25.69, 21.22, 49.56, 25.64, 28.31)  
d4 <- c(76.22, 65, 19.69, 29.84, 37.18, 70.93, 64.78, 61.66, 49.03, 51.56)  
d5 <- c(92.11, 56, 47.89, 62.96, 47.41, 37.05, 73.96, 53, 52.37, 85.23)  
  
sd(d1)/sqrt(length(d1))

## [1] 7.122595

sd(d2)/sqrt(length(d2))

## [1] 6.10332

sd(d3)/sqrt(length(d3))

## [1] 5.783761

sd(d4)/sqrt(length(d4))

## [1] 5.898

sd(d5)/sqrt(length(d5))

## [1] 5.597579

###summary()

###psych::describe()

data <- read\_rds("data/raw/numeric\_data.rds")  
describe(data, na.rm = TRUE, skew = FALSE, ranges = TRUE)

## vars n mean sd median min max range se  
## Возраст 1 100 30.25 3.98 30.50 21.00 42.00 21.00 0.40  
## Рост 2 100 167.70 5.77 168.00 155.00 181.00 26.00 0.58  
## Базофилы\_E1 3 100 0.65 0.38 0.65 -0.22 1.72 1.94 0.04  
## Эозинофилы\_E1 4 100 3.71 2.15 3.73 -1.23 8.43 9.66 0.21  
## Гемоглобин\_E1 5 100 11.86 1.78 11.71 5.35 16.23 10.88 0.18  
## Эритроциты\_E1 6 100 4.10 0.67 4.08 2.82 5.73 2.91 0.07  
## Базофилы\_E2 7 100 1.06 0.38 1.06 0.19 2.12 1.94 0.04  
## Эозинофилы\_E2 8 100 4.72 2.15 4.74 -0.21 9.45 9.66 0.21  
## Гемоглобин\_E2 9 100 12.58 1.78 12.43 6.07 16.95 10.88 0.18  
## Эритроциты\_E2 10 100 6.42 0.67 6.40 5.14 8.04 2.91 0.07

###table(), prop.table()

data1 <- read\_rds("data/raw/factor\_data.rds")  
table(data1$Группа, data1$`Группа крови`)

##   
## O (I) A (II) B (III) AB (IV) <NA>  
## \xc3\xf0\xf3\xef\xef\xe0 1 10 16 11 4 9  
## \xc3\xf0\xf3\xef\xef\xe0 2 15 18 5 4 8

###tibble()

#Особенности tibble:  
  
#tibble не изменяют тип ввода. Если вы вводили переменные типа character, то такими они и будут. Если numeric, то будут numeric. В том числе это позволяет даже вводить в качестве значений ячеек списки!  
#tibble не меняет имена переменных: если вы ввели имя с пробелом, то он не будет заполнен точкой или иным знаком, однако, нужно оборачивать имена в апострофы (чаще всего находится на клавише буквы "ё" в верхнем левом углу клавиатуры). Это следует делать всякий раз, когда имя переменной отличается от простой строки на латинице без иных знаков. Например: columnname не требует апострофов, а `column name` уже требует;  
#Оценивает аргументы лениво и последовательно, что мы разберём в следующем шаге;  
#Не использует имена строк;  
#При выводе таблицы данных в печать автоматически будут показаны только первые 10 строк и все столбцы, которые поместятся на экран. Благодаря этому даже очень большой датафрейм не заставит наш компьютер зависнуть.  
#Заметка: чтобы превратить data.frame в tibble, достаточно просто применить функцию as\_tibble()

tibble(var\_first = 1:10, var\_second = ifelse(var\_first < 5, var\_first + 100, var\_first)) tibble(var = 1:10, var = var - 10000000) tibble(var 1 = 1:10, var 2 = var 1 \* 100) tibble(var\_first = 1:10, var\_first = ifelse(var\_first < 5, var\_first + 100, var\_first)) tibble(var 2 = 10:1, var 3 = var 1 - 10) tibble(var\_1 = c(1:10) - 100, var\_2 = 1:100)

###View() ###tibble::add\_column()

#data %>% add\_column(column\_name = 1:10, .before = NULL, .after = NULL)  
  
#data: просто имя датафрейма, к которому мы хотим добавить столбец;  
#column\_name: это имя нового столбца. Оно может быть любым, не только таким, как в примере;  
#.before: номер уже существующего столбца, перед которым нужно поставить новый;  
#.after: то же, но уже после которого нужно поставить новый. Хитрый приём: если нужно поставить переменную в конец датафрейма, то в значение можно поставить Inf.

###tibble::add\_row()

#data %>% add\_row(var\_1 = 1, var\_2 = "value", .before = NULL, .after = NULL)  
  
#data: просто имя датафрейма, к которому мы хотим добавить столбец;  
#var\_1, var\_2: это имя нового столбца. Оно может быть любым, не только таким, как в примере;  
#.before: номер уже существующей строки, перед которым нужно поставить новый;  
#.after: то же, но уже после которого нужно поставить новый. Хитрый приём: если нужно поставить переменную в низ датафрейма, то в значение можно поставить Inf

###dplyr::row\_number()

###dplyr::bind\_cols()

#Мы можем сделать из двух и более таблиц одну, склеив их столбцы.  
  
#data\_1 %>% bind\_cols(data\_2) %>% bind\_cols(data\_3)