Введение в R

### Расчеты, операции с данными (арифметика)

1 + 1  
2 \* 3  
(4 + 5) \* 6  
2 ^ 3  
2 ^ 0.5  
sqrt(2)  
log10(100)  
log2(16)  
log(9, base = 3)  
abs(-4)

### Переменные

* Переменные: имена, присвоения, использование
* присвоение значений переменным

my\_variable = 1  
MyVariable = 1  
my.variable = 1

a = 1  
a=1  
a <- 1  
a = "kjfhs"  
a = 'kjfhs'

### Типы данных

Простые типы данных (числовые и строковые константы, векторы, факторы)

# number  
my\_number = 1  
  
# character  
my\_char = "bla\tbla\nbla"  
cat(my\_char)  
  
# boolean  
my\_bool1 = TRUE  
my\_bool2 = FALSE  
  
if(my\_bool1) {  
 print("TRUE")  
} else {  
 print("FALSE")  
}  
  
ifelse(my\_bool1, "TRUE", "FALSE")  
  
2<1  
ifelse(2<1, "TRUE", "FALSE")  
my\_bool3 = 2<1   
ifelse(my\_bool3, "TRUE", "FALSE")  
  
# vector набор данных одного типа (одни единицы измерения)  
my\_vector1 = 1:20  
my\_vector1.1 = rep(9, 30) # последовательность из 9 длиной 30  
length(my\_vector1.1)  
# подмножество из вектора  
my\_vector1[2:4]  
my\_vector2 = c(169, 180, 157, 165, 190, NA, 185, 164, 171) # NA = Not Available = пропущенные данные  
# выполняем операцию сравнения  
my\_vector2 > 170  
# выбираем значения, отвечающие критерию my\_vector2 > 40  
my\_vector2[my\_vector2 > 170]  
186 %in% my\_vector2  
190 %in% my\_vector2  
my\_vector2 == 190  
  
# factor vs vector  
# создаем вектор  
age = c("young", "middle-aged", "old", "Old", "young", "middle-aged", "middle-aged")  
age # просто перечень текстовых значений  
# преобразуем в фактор (текстовые значения в коды, текст отображается лишь для простоты восприятия)  
factor(age) # видим лишний уровень "Old", дублирующий "old"  
age = c("young", "middle-aged", "old", "old", "young", "middle-aged", "middle-aged")  
factor(age) # несортированный фактор  
factor(age, levels=c("young", "middle-aged", "old"), ordered = TRUE) # levels задает возрастающий порядок уровней

Сложные типы данных (матрицы, списки, таблицы данных)

# matrix - двухмерный набор значений одного типа (например, каких-то измерений)  
# создаем одномерный вектор  
rnorm(8\*12, mean=5000, sd=2000) # команда для создания 8\*12 случайных чисел с заданным средним и стандартным отклонением  
# помещаем его значения в матрицу  
my\_matrix1 = matrix(rnorm(8\*12, 5000, 2000), nrow=8, ncol=12) # пример матрицы 8х12 со случайными значениями  
my\_matrix1  
colnames(my\_matrix1) = as.character(1:12) # задаем название столбцов (становятся текстом!)  
library(tidyverse) # много всего полезного, в т.ч. работа со строками  
str\_split("ABCDEFGH", "", simplify = T) # генерим вектор из букв  
rownames(my\_matrix1) = str\_split("ABCDEFGH", "", simplify = T) # задаем название строк  
my\_matrix1 # выглядит знакомо ;)  
str(my\_matrix1)  
my\_matrix1[1,] # первая строка (номер строки)  
my\_matrix1[1,] = 1:12 # так можно модифицировать данные, замена должна соответствовать размеру ...  
my\_matrix1[2,] = NA # ... или можно заменять все позиции матрицы на одно значение  
my\_matrix1["A",] # первая строка (название строки)  
my\_matrix1[c(1,3),] # первая и третья строка (номера строк)  
my\_matrix1[c("A","C"),] # первая и третья строка (названия строк)  
my\_matrix1[, 1] # первая колонка (номер)  
my\_matrix1[, "1"] # первая колонка (название, поэтому в кавычках, это текст)  
my\_matrix1[3, 11]  
my\_matrix1[8, 12] = 100500  
my\_matrix1\*1000  
my\_matrix1  
ncol(my\_matrix1)  
nrow(my\_matrix1)  
t(my\_matrix1) # транспонировать матрицу, т.е. превратить колонки в строки  
  
# Если надо сложить одну переменную много разных типов данных - это список  
my\_list1 = list(a=c(1,2,4), b=c("one", "two", "three"), c=3, d=age) # сразу задаем значения  
my\_list1   
str(my\_list1)  
my\_list1$a # обращаемся к элементу  
my\_list1[["a"]] # обращаемся к элементу - второй способ  
b\_element = my\_list1[["a"]]  
b\_element  
b\_element[2]  
my\_list1[["a"]][2] # выбираем второй элемент вектора b из списка my\_list1   
my\_list1$e = "создаем новый элемент"  
my\_list1[["last\_one"]] = c("и", "последний", "элемент")  
str(my\_list1)  
  
# пример использования списка  
my\_list2 = list()  
my\_list2$description = "Плашка FIA"  
my\_list2$date = Sys.time() # эта команда выводит текущее время  
my\_list2$plate = my\_matrix1  
my\_list2$session = sessionInfo() # эта команда выводит данные о версии R и загруженных библиотеках (пригодится для воспроизводимости!)  
  
# а еще многие функции выводят списки  
t.test\_result = t.test(rnorm(8\*12, mean=5000, sd=2000), mu = 4500)  
str(t.test\_result)  
t.test\_result$p.value # из объекта типа лист можно доставать нужное значение  
t.test(rnorm(8\*12, mean=5000, sd=2000), mu = 4500)$p.value # или даже так  
  
# самый используемый тип данных - таблица данных (data.frame и ее расширения типа tibble, data.table)  
# такая таблица технически это список из разных типов колонок-векторов (численные, текстовые, факторные) одной длины  
my\_df1 = data.frame(a=c(1,2,4), b=c("a", "b", "d"))  
my\_df1  
str(my\_df1)  
# с таблицей данных можно обращаться как со списком, а можно как с матрицей  
# почти все данные, которые придется обрабатывать, будут в формате таблиц данных

Арифметические операции с векторами, матрицами, таблицами

my\_vector1 = 1:20  
my\_vector3 = 101:120  
my\_vector1 + 5  
my\_vector1 \* 5  
my\_vector1 + my\_vector2  
my\_vector4 = 1:21  
my\_vector1 + my\_vector3 # ошибка  
my\_vector1 ^ 2  
log10(my\_vector1)  
max(my\_vector1)  
min(my\_vector1)  
my\_vector2 = c(169, 180, 157, 165, 190, NA, 185, 164, 171)  
max(my\_vector2) # что-то пошло не так...  
na.omit(my\_vector2)  
max(na.omit(my\_vector2))

[Дополнительная информация](https://adv-r.hadley.nz/subsetting.html) про экстракцию подтаблиц и частей векторов, сортировку и т.п.

### Использование функций

# использование одной функции  
?rnorm  
rnorm(96) # аргументы по умолчанию: mean=0, sd=1  
rnorm(8\*12)  
rnorm(8\*12, mean=5000, sd=2000) # явно указанные аргументы  
rnorm(8\*12, sd=2000, mean=5000) # явно указанные аргументы в другом порядке - не важно!  
rnorm(8\*12, 5000, 2000) # аргументы указания названий в стандартном порядке (см. справку)  
rnorm(8\*12, s=2000, m=5000) # сокращенные названия аргументов - первые буквы должны однозначно указывать  
  
# некоторые функции принимают аргументы в виде формул вида value ~ grouping1 + grouping2 ...  
my\_data = data.frame(my\_values = c(1, 2, 3, 1, 4, 3),  
 my\_grouping = as.factor(c(1, 1, 1, 2, 2, 2)))  
library(car)  
leveneTest(my\_data$my\_values, my\_data$my\_grouping)  
with(my\_data, leveneTest(my\_values, my\_grouping)) # чтобы не писать my\_data$ перед каждой переменной  
leveneTest(my\_values ~ my\_grouping, data = my\_data) # пример формулы  
  
# использование нескольких функций: среднее полученного распределения  
my\_distr1 = rnorm(8\*12, sd=2000, mean=5000) # вывод функции можно сохранять в переменную  
mean(my\_distr1)  
  
# можно вкладывать функции друг в друга, направляя вывод одной функции на вход (аргумент) другой  
mean(rnorm(8\*12, sd=2000, mean=5000))  
  
# можно создавать pipelines (об этом далее будет более подробно)  
8\*12 %>% # это значение будет первым аргументом следующей (rnorm)  
 rnorm(sd=2000, mean=5000) %>% # вывод этой функции будет первым аргументом следующей (mean)  
 mean() %>% # это значение будет первым аргументом следующей (round)  
 round(0)

### Собственная функция

# пишем функцию для коэффициента вариации  
CV = function(x) sd(x)/mean(x)\*100 # возвращает результат преобразования  
my\_vector6 = rnorm(30, 50, 5)  
CV(my\_vector6)

# пишем функцию для вычисления дисперсии, которая нам выдаст несколько значений  
Var = function(x) {  
 result = list()  
 result$data = x  
 result$sum\_sq = sum((x-mean(x))^2)  
 result$n = length(x)  
 result$value = result$sum\_sq/(result$n - 1)  
 return(result) # задаем объект, который будет выводить функция (здесь это список)  
}

# Работа с переменными, вычисления

* математические операторы и функции (+, -, ^, \*, log, abs, mean)
* переменные и как их называть

my\_variable = 1  
MyVariable = 1  
my.variable = 1

* присвоение значений переменным

a = 1  
a=1  
a <- 1  
a = "kjfhs"  
a = 'kjfhs'

# Ввод данные

* Ввод в текстовом виде

weight\_of\_my\_students = c(65, 54, NA, 67, 45, 78) # NA - Not Available

* Импорт из XLSX (пакет readxl::read\_())
* Импорт из TXT, CSV, DAT (read.table(), read.csv()…)
* Импорт из других форматов (HTML, XML, кастомные форматы выдачи приборов и т.п.) - можно найти пакеты/советы на форумах/написать самому

# обратимся к функции read\_excel из пакета readxl  
my\_xlsx\_data = readxl::read\_excel("my\_data.xlsx") # см. полезные доп. аргументы!  
# пакет tidyverse нам нужен постоянно, поэтому загрузим его сразу  
library(tidyverse) # очень полезный пакет, который загружает много других полезных пакетов  
my\_csv\_data = read\_csv("my\_data.xlsx")  
# но можно и напрямую указывать из какого пакета функция, не загружая его  
my\_csv\_data = readr::read\_csv("my\_data.xlsx")  
# чтобы найти в каком пакете функция можно нагуглить, либо выполнить комманду  
getAnywhere("read\_csv")

* Вставка скопированных данных из Excel

my\_table = read.table(text="сюда вставлять таблицу") # далее пример  
my\_table = read.table(text="v1 v2  
1 2  
11 22  
111 222  
1.3 7  
4 NA", header=TRUE, dec=".") # header=TRUE если есть названия переменных, dec="." устанавливает точку десятичным разделителем  
# анализируем структуру открытой таблицы  
str(my\_table)

# Типы данных

# number  
my\_number = 1  
  
# character  
my\_char = "bla\tbla\nbla"  
cat(my\_char)  
  
# boolean  
my\_bool1 = TRUE  
my\_bool2 = FALSE  
  
# vector  
my\_vector1 = 1:20  
# подмножество из вектора  
my\_vector1[2:4]  
my\_vector2 = c(1, 2, 30, 50, 100, 150)  
# выполняем операцию сравнения  
my\_vector2 > 100  
# выбираем значения, отвечающие критерию my\_vector2 > 40  
my\_vector2[my\_vector2 > 40]  
  
# factor vs vector  
# создаем вектор  
age = c("young", "middle-aged", "old", "Old", "young", "middle-aged", "middle-aged")  
age # просто перечень текстовых значений  
# преобразуем в фактор (текстовые значения в коды, текст отображается лишь для простоты восприятия)  
factor(age) # видим лишний уровень "Old", дублирующий "old"  
age = c("young", "middle-aged", "old", "old", "young", "middle-aged", "middle-aged")  
factor(age) # несортированный фактор  
factor(age, levels=c("young", "middle-aged", "old"), ordered = TRUE) # levels задает возрастающий порядок уровней  
  
# data.frame  
head(iris) # iris - встроенный пример таблицы данных

## Таблица данных - учимся создавать подтаблицы

# первая строка  
iris[1,]  
# первый столбец  
iris[,1]  
  
# пробуем доставать разные строки, столбцы и значения на основе разных критериев  
iris[1:15,]  
iris[,1:3]  
iris[, c(1, 3, 5)]  
iris[, c("Petal.Width", "Species")]  
iris$Sepal.Length  
iris$Sepal.Length[1:10]  
iris$Species == "versicolor"  
iris$Petal.Length[iris$Species == "versicolor"]  
iris[iris$Species == "versicolor",]  
iris$Sepal.Length > 7  
iris[iris$Sepal.Length > 7,]

## Операции с векторами и таблицами

# векторы (vectors)  
my\_vector2 = 1:10  
my\_vector3 = (1:10)\*10  
1:10\*10  
(1:10)\*10  
my\_vector2 + my\_vector3  
my\_vector2 - 1  
(my\_vector2)^2  
sum(my\_vector2)  
my\_vector2 > 3  
  
# таблицы ()  
iris\_subset = iris[1:10, 1:4]  
iris\_subset > 3.2

# Проверка нормальности распределения

Будем проверять распределение эмпирических данных (vers) и сгенерированных, соответствующих нормальному (nd) и ненормальному (nnd) распределению.

vers = iris[iris$Species == "versicolor","Sepal.Length"]  
  
# гистограмма  
hist(vers)  
?hist # справка  
hist(vers, breaks = 20)  
  
# кривая плотности вероятности распределения  
density(vers)  
plot(density(vers))  
rug(vers)  
stem(vers) # графический аналог описан тут http://www.dataanalytics.org.uk/Data%20Analysis/R%20Monographs/Dot%20Histogram.R  
  
# пример нормального распределения  
# для воспроизводимости генератора случайных чисел задаем "стартовое число"  
set.seed(123)  
nd = rnorm(100, mean=50, sd=5) # задаем желаемые параметры  
hist(nd,breaks=20)  
plot(density(nd))  
  
# пример НЕнормального распределения  
# для воспроизводимости генератора случайных чисел задаем "стартовое число"  
set.seed(123)  
nnd = rchisq(100, 20, 5) # задаем желаемые параметры  
hist(nnd,breaks=20)  
plot(density(nnd))  
  
# критерий нормальности - shapiro.test  
shapiro.test(vers)  
st = shapiro.test(vers)  
str(st)  
st$p.value  
st$p.value > 0.05 # TRUE => нормальное распределение  
shapiro.test(nnd)$p.value # < 0.05 => ненормальное распределение  
  
# визуальная оценка нормальности  
qqnorm(vers) # нормально  
qqline(vers)  
qqnorm(nnd) # ненормально  
qqline(nnd)

# Работаем с нормально распределенными данными

# параметры нормального распределения  
mean(vers)  
sd(vers)  
  
# z-трансформация  
scale(vers)  
  
# сами пишем функцию для проведения z-трансформации  
# Версия 1 - длинная и поэтапная  
z\_transformation = function(x){  
 step1 = mean(x)  
 step2 = x - step1  
 step3 = sd(x)  
 step4 = step2/step3  
 return(step4) # maybe just step4  
}  
  
# Версия 2 - лаконичная  
z\_transformation = function(x) (x-mean(x))/sd(x)

Кстати, нашел отличную [статью](https://www.mathsisfun.com/data/standard-deviation.html) про то, почему стандантартное отклонение не есть сумма отклонений и не сумма модулей, а именно корень из суммы квадратов.