install.packages("UsingR") # Инсталиране на пакети

# Преди да инсталираме пакети, желателно е да проверим дали пакетът вече не е инсталиран.

# Това става с реда

installed.packages()

# Резултатът е матрица, която съдържа името на пакета, директорията на пакета, версията

# и т.н. Ако само се нуждаем от имената на инсталираните пакети, то най-добре е да изпозлваме

# командата по-долу

row.names(installed.packages())

# Връща ни вектор с имената. Векторът може да се присвои на променлива

library(UsingR) # Зареждане на пакета

# - Как можем да потърсим помощ в R?

# - С помощта на ?името\_на\_функцията и help("името\_на\_функцията"). Ако не се сещаме

# името на функцията, която ни трябва, то можем да напишем част от това, която ни трябва

?mean

help("sd")

??linear # Надяваме се да намерим някаква функция за линейна регресия. Намерихме я - stats::lm

# Начините за присвояване на обект към променлива са: "=", "<-" или "->". Последняит е

# напълно излишен но го има

x = 5

y <- 2\*x + 6

"Austria" -> z

x; y; z

# Основни структури от данни

# 1. Вектор/масив

# С комaндата "c(...)" се създава масив от елементи. В един векторът може да се

# съдържат елементи от различни тип - числа, стрингове и дори листа/обекти.

# Ако имаме наличие на лист или обект, то векторът се превръща в лист.

v1 <- c() # Създаване на правен вектор

v1 <- c(1, 4, 6., 4, 7, 12., -17) # Създаване на вектор, който съдържа числа

v1

# За да добавим нови елементи към вектора, най-лесно е да използваме отново командата "c(...)"

v2 <- c("a", "b", "c")

v2

v12 <- c(v1, v2) # Новият вектор съдържа елементите на двата вектора v1 и v2

v12

# Какъв е типа на елементите в новия вектор? Това се проверява с функцията "str(...)"

str(v1) # Числов вектор

str(v2) # Стрингов вектор

str(v12) # Автоматично е cast-нат е до стрингов вектор

# Добре, а какво би станало, ако използваме дати?

v3 <- as.Date(c("2015-01-01", "2016-07-08"))

# С функцията as.Date конвертираме стрингов вектор (или числов) във вектор с дати

?as.Date

v13 <- c(v1 ,v3)

v13

str(v13) # Числов вектор

# Забележете, че датите са integer числа, които броят дните от 1970-01-01 до посочената дата

# - А можем ли да кажем на R, че искаме непременно векторът да бъде в числов вид

# - Да и не само в числов. Можем да му кажем, че искаме да го cast-нем в стрингов или

# във вектор, съдържащ дати. Това, разбира се, може да доводе до загуба на информация

as.numeric(v12)

# Стойността "NA" в последните три елемента от вектора, показва, че имаме липса на

# информация. Това се дължи на факта, че се опитахме да представим стрингов вектор в

# числова форма.

as.character(v1) # Тук нямаме проблем, защото всеки елемент може да се представи в стрингова форма.

# Да пробваме да преобразуваме числов вектор във вектор с дати

as.Date(v1)

# Дава грешка. Защо?

# Защото не сме посочили начална дата, от която да тръгне броенето на дните. Началната

# дата се задава с параметъра "origin"

as.Date(v1, origin = "2011-01-01")

v1

# length(...) - взима дължината на вектор

length(v1)

# Взимане на елемент от вектор

v1[1] # Индексацията започва от 1

v1[c(1, length(v1))] # Нов вектор, който съдържа първия и последния елемент от вектора

v1[c(3, 3, 3, 3)] # Вектор, който има дължина 4 и стойностите му са третия елемент от v1

1:8 # Създаваме редица, която съдържа елементите от 1 до 8

8:1 # Създаваме редица, която съдържа елементите от 8 до 1

# Горните два реда са еквивалентни съответно на долните два

seq(from = 1, to = 8, by = 1)

seq(from = 8, to = 1, by = -1)

# Искаме да вектор, който да съдържа 3, 6, 1, 2, 3, 4 елементи от вектора v1

v1[c(3, 6, 1:4)]

# В R можем да създадем подвектор на v1, като му кажем кои стойности НЕ ИСКАМЕ да присъстват

v1[-c(3, 6)]

v1[-(3:6)]

# Можем да добавяме/умножаваме с число даден вектор. Също така можем да събираме и

# умножаваме два вектора.

# За да не ни се налага да се чудим какви числа да измисляме всеки път, то най-добре е

# всичко до оставим в ръцете на "съдбата". Тоест да генерираме нашите числа на случаен принцип.

# Създаваме 3 вектора с генерирани псевдослучайни величини. За да получаваме винаги една и

# съща редица от числа, то трябва винаги да стартираме от една и съща начална позиция. За тази

# цел използваме командата set.seed(...).

# Искаме числата, които се генерират в трите вектора, да се падат с равни вероятности. Тоест

# P(1) = P(2) = ... = P(n-1) = P(n), където P(x) е вероятнсотта да се падне числото x.

# В упражненията ще учите подробно различните вероятностни разпределения, но за момента е

# достатъчно да знаем, че този вид разпределение се нарича "равномерно".

# На долния ред е показан пример на равномерно разпределение

hist(trunc(runif(10^3, 1, 5.9999)), col = "red", main = "Histogram",

xlab = "Pseudo random numbers")

#

set.seed(1806)

v4 <- trunc(runif(n = 20, min = 1, max = 40.99999))

set.seed(2713)

v5 <- round(runif(n = length(v4), min = 1, max = 40.99999))

set.seed(189)

v6 <- round(runif(n = length(v4) - 7, min = 1, max = 40.99999))

# runif - функция за генериране на псевдо случайни равномерно разпределени числа.

# Първият параметър е за броя на случайните числа. Вторият и третият показват обхвата на

# възможните числа.

# Препоръчвам да я разгледате, защото се използва при Монте Карло методите за оптимизации

v4 + 3

2\*v4

v4/7 + 11

v4\*v5 # Скаларно произведение

v4\*v6

# Дава предупреждение, защото дължината на втория вектор не е кратна на първия.

# Ето защо започва умножението отначало.

# - А друго какво мога да правя? Мога ли да сменям числа във вектора

# - Може, разбира се.

v4.prime <- v4

v4[c(1, 2, 3)] <- c(3, 2, 1)

v4 <- v4.prime

v4[c(1, 2, 3, 4)] <- c(100, -100)

# -------

# Функции

# име\_на\_функцията <- function(параметри) {

#

# }

func1 <- function(a, b, c) {

return(a + b + c)

}

func1(1, 2, 3)

# Функциите в R могат да имат стойности по подразбиране

func2 <- function(a, b = 0, c = 0) {

return(a + b + c)

}

func2(1)

func2(1, 2)

func2(1, c = 3)

# Функциите в R могат да приемат параметрите си в различен ред

func2(a = 1, c = 3, b = 2)

func2(b = 2, c = 3)

# Връща грешка, защото не сме задали стойност по подразбиране на параметъра "a" във функцията

# Това лесно може да се избегне с едно условие във функцията

func3 <- function(a, b = 0, c = 0) {

if(missing(a)) {

print("You don't enter value for \"a\". The function generate normal distributed value")

a <- rnorm(n = 1)

}

return(a + b + c)

}

# Командата "missing" проверява дали имаме стойност за параметъра "a"

# С "print()" извеждаме съобщение

# С "rnorm(n = 1)" генерираме една (n = 1) нормално разпределена случайна величина

# с очакване 0 и стандартно отклонение 1

func3(a = 1, b = 2, c = 3)

func3(b = 2, c = 3)

func4 <- function(a, b = 2, c = 3) {

if(missing(a)) {a <- NA}

Obj <- NULL

Obj$number1 <- a

Obj$number2 <- b

Obj$number3 <- c

Obj # return(Obj)

}

func4(1, 2, 3)

# if-else условия

# if() {

#

# } else if() {

#

# } else {

#

# }

# Цикли

# for(...) {}

# Цикълът for представлява foreach итерация. Конструкцията е проста -

# променлива %in% вектор

v7 <- c("a", "b", "c", "d", "e")

for(i in v7) {print(i)}

for(i in 1:length(v7)) {print(v7[i])}

for(i in length(7):1) {print(v7[i])}

for(i in 1:length(v7)) {

if(i == 4) {

print("----MISS")

next

}

print(v7[i])

}

# С командата "next" пропускаме итерация

counter <- 0

while(counter < 5) {

counter <- counter + 1

print(counter)

}

# do - while

counter <- 0

repeat {

counter <- counter + 1

print(counter)

if(counter > 5) {

break

}

}

# С командата "break" излизаме от най-близкия цикъл

# Циклите в R са бавни. Ето защо е подходящо в някои случаи да използвате методите

# apply, lapply, sapply, vapply и tapply. Тези методи ще ги представим след малко.

# -------

# 2. Матрица

# С функцията "matrix" в R създаваме матрица от предварително зададено множество от

# стойности - вектор или лист със стойности. Освен множеството с елементи, трябва да

# посочим и формата на матрицата - брой редове и колони. Стойностите на матрицата се

# пълнят по колони. За да напълним матрицата със стойности по редове, трябва да използваме

# параметъра byrow = TRUE

l1 <- lapply(1:12, function(x) {x}) # лист

M1 <- matrix(data = l1, nrow = 4, ncol = 3)

M1 <- matrix(data = 1:12, nrow = 4, ncol = 3)

# Горните два реда са еквивалентни

M2 <- matrix(data = 1:12, nrow = 4, ncol = 3, byrow = T)

M1

M2

# Взимане на елемент, ред, колона и подматрица от матрица

M3 <- matrix(data = c(1:28), nrow = 7, ncol = 4, byrow = TRUE)

M3[2, ] # Взимане на ред

M3[, 3] # Взимане на цолона

M3[1, 3] # Взимане на елемент

M3[c(1, 2), 3] # Взимаме 1 и 2 елемент от 3 колона

M3[c(1, 2), c(3, 4)] # Взимаме подматрица

# Операции с матрици

M3 + 4 # Добавяме число към матрицата

M3 + 5\*c(1:7) # Добавяме по вектор към всеки от четирите колони

M3 + 5\*c(1:4) # Добавяме ред вектор всеки от седемте реда на матрицата

M3 + 5\*c(1:2) # R автоматично удвоява вектора до дължина 4 и добавяме вектора към всеки ред

# Аналогично е и при умножение на матрица с вектор

M4 <- matrix(1:8, nrow = 4, ncol = 2)

M34 <- M3 %\*% M4 # Стандартно умножение на две матрици

M34

# M(7x4) \* M(4x2) = M(7x2)

dim(M34) # Връща размера на матрицата

nrow(M34) # Връща броя на редовете

ncol(M34) # Връща броя на колоните

# Функцията apply се прилага върху матрици, data.frame и други производни структури.

# Първият параметър е множеството от данни, вторият показва по редове (1) или колони (2) искаме

# да направим трансформациите, а третият е самата функция

apply(M3, 2, function(x, a) {sum(x)}, a)

apply(M3, 1, function(x) {sum(x)})