

# Podstawy programowania R

*Łukasz Wawrowski*



# Contents

|  |           |
|--|-----------|
| <b>Wprowadzenie</b>                                | <b>5</b>  |
| <b>1 Wprowadzenie do R</b>                         | <b>7</b>  |
| 1.1 R . . . . .                                    | 7         |
| 1.2 RStudio . . . . .                              | 7         |
| 1.3 Pakiety . . . . .                              | 10        |
| 1.4 Ważne informacje . . . . .                     | 10        |
| 1.5 R jako kalkulator . . . . .                    | 10        |
| <b>2 Struktury danych</b>                          | <b>13</b> |
| 2.1 Wektor . . . . .                               | 13        |
| 2.2 Macierz . . . . .                              | 21        |
| 2.3 Czynniki . . . . .                             | 25        |
| 2.4 Lista . . . . .                                | 26        |
| 2.5 Ramka danych . . . . .                         | 27        |
| <b>3 Przetwarzanie danych</b>                      | <b>31</b> |
| 3.1 Pakiet tidyverse . . . . .                     | 31        |
| 3.2 Import danych . . . . .                        | 31        |
| 3.3 Wybieranie kolumn . . . . .                    | 31        |
| 3.4 Filtrowanie . . . . .                          | 31        |
| 3.5 Dodawanie nowych zmiennych . . . . .           | 31        |
| 3.6 Grupowanie . . . . .                           | 31        |
| 3.7 Podsumowanie . . . . .                         | 31        |
| 3.8 Łączenie zbiorów . . . . .                     | 31        |
| 3.9 Wąska i szeroka reprezentacja danych . . . . . | 31        |
| <b>4 Programowanie w R</b>                         | <b>33</b> |
| 4.1 Funkcje . . . . .                              | 33        |
| 4.2 Pętle . . . . .                                | 33        |
| 4.3 Instrukcje warunkowe . . . . .                 | 33        |
| <b>5 Wizualizacja danych</b>                       | <b>35</b> |
| 5.1 Wbudowane funkcje . . . . .                    | 35        |
| 5.2 Pakiet ggplot2 . . . . .                       | 35        |



# Wprowadzenie

Literatura podstawowa:

- Przemysław Biecek - *Przewodnik po pakiecie R*
- Marek Gągolewski - *Programowanie w języku R. Analiza danych, obliczenia, symulacje.*
- Garret Golemund, Hadley Wickham - *R for Data Science*

Literatura dodatkowa:

- inne pozycje po polsku
- inne pozycje po angielsku



# Chapter 1

## Wprowadzenie do R

GNU R to interpretowany język programowania oraz środowisko do obliczeń statystycznych i wizualizacji wyników [Wikipedia 2017].

Robert A. Muenchen - The Popularity of Data Science Software

### 1.1 R

Bazowa wersja R jest do pobrania ze strony [r-project.org](http://r-project.org).

### 1.2 RStudio

RStudio to zintegrowane środowisko programistyczne (IDE) dla języka R dostępne za darmo na stronie RStudio.

Z R można także korzystać w Microsoft Visual Studio.

- **Katalog roboczy**

Ważnym pojęciem w R jest katalog roboczy (ang. working directory), który określa gdzie zostaną zapisane pliki, wykresy, zbiory, itp. jeśli nie podamy dokładnej ścieżki do pliku. Katalog roboczy przypisuje się z wykorzystaniem funkcji `setwd("ścieżka do katalogu")`, a jego wartość można sprawdzić funkcją `getwd()`. W RStudio przypisanie katalogu roboczego odbywa się w momencie utworzenia projektu.

- **Projekt**

Katalog na dysku, w którym znajdują się wszystkie pliki projektu wraz z plikiem o rozszerzeniu `.Rproj` skojarzonym z RStudio.

- **Korzystanie z pomocy**

Dostęp do pomocy odnośnie wybranej funkcji można uzyskać na dwa sposoby. Pierwszym z nich jest poprzedzenie nazwy funkcji w konsoli znakiem zapytania np. `?getwd` lub wywołanie funkcji `help` na nazwie funkcji `help("getwd")`. Drugim sposobem jest umieszczenie kursora w dowolnym miejscu nazwy funkcji i wciśnięcie klawisza F1.

Internet - przede wszystkim [stackoverflow](http://stackoverflow.com).

- **Komentarze**

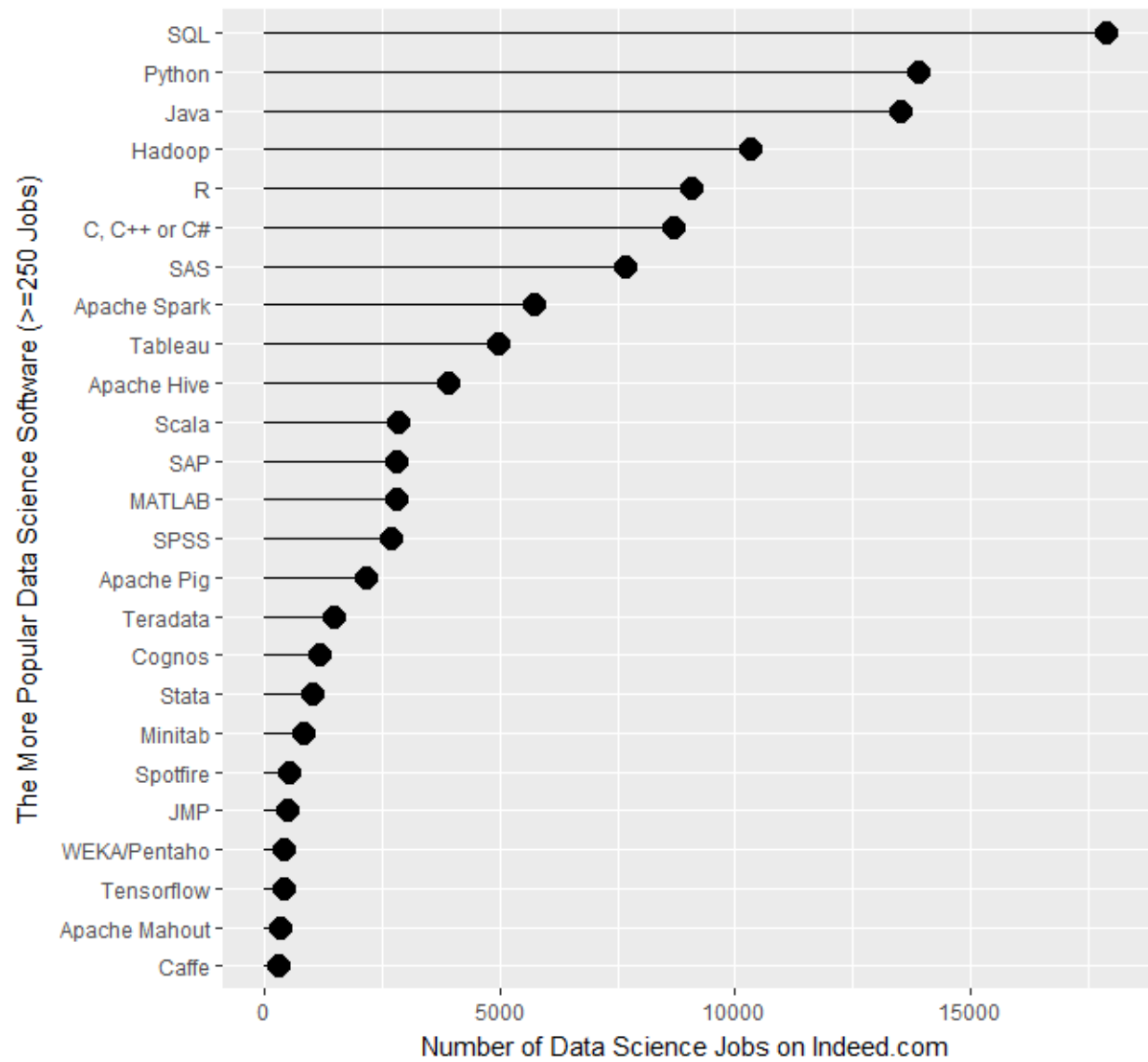


Figure 1.1:



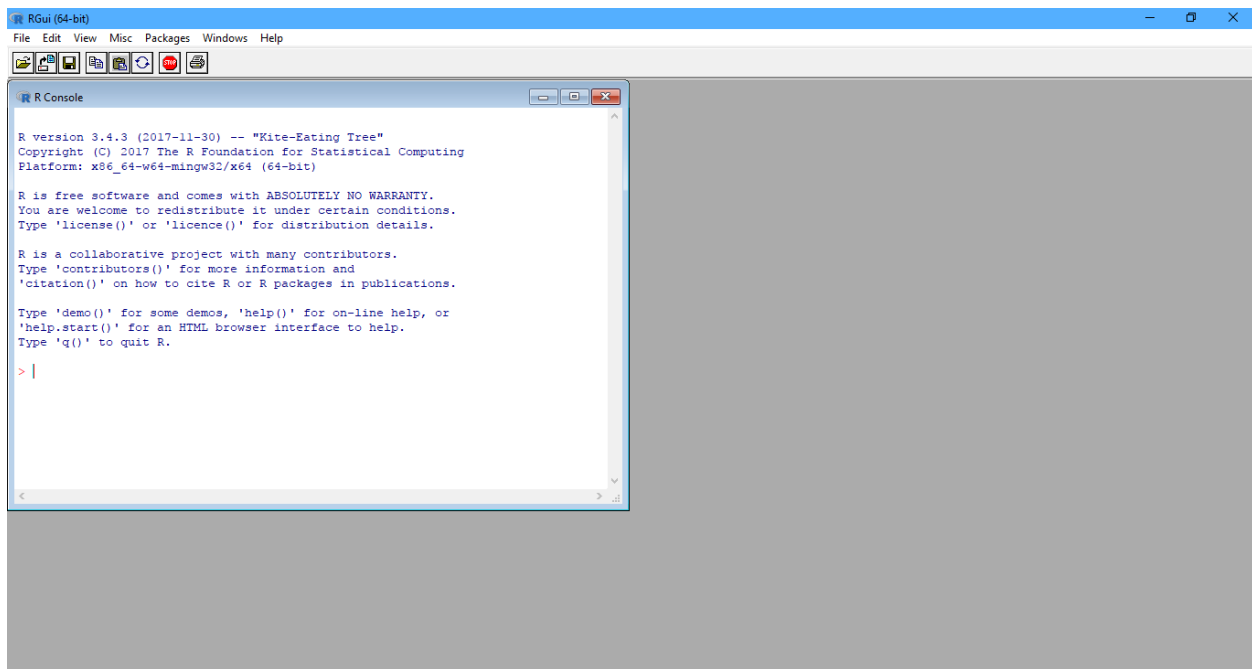


Figure 1.2:

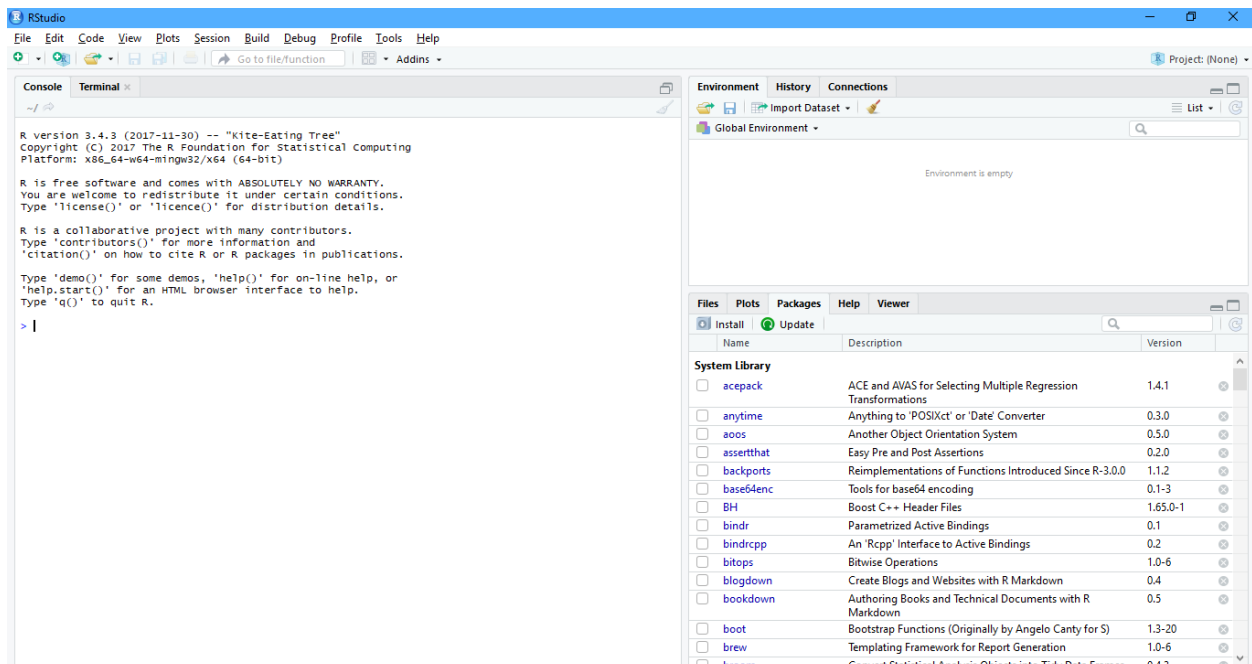


Figure 1.3:

Real programmers don't comment their code. If it was hard to write it should be hard to understand.

Dobrze napisany kod jest czytelny bez komentarzy. W R komentarze rozpoczynają się od symbolu `#`. Skrót klawiaturowy w RStudio to `CTRL + SHIFT + C` (do wstawiania i usuwania komentarzy).

- **Podpowiadanie składni**

RStudio ma zaimplementowaną funkcję podpowiadania składni. Listę możliwych funkcji i obiektów wywołuję się klawiszem `TAB` lub `CTRL + SPACJA` po wpisaniu co najmniej jednej litery. Kolejne naciśnięcie `TAB` lub `ENTER` powoduje uzupełnienie kodu o wybraną funkcję lub obiekt.

## 1.3 Pakiety

Podstawowe możliwości R są dosyć ograniczone. Rozszerzają je pakiety, których obecnie jest ponad 12 tysięcy. Można je przeglądać według kategorii w CRAN Task Views lub w wygodnej wyszukiwarce METACRAN i rdro.io.

## 1.4 Ważne informacje

R jest wrażliwy na wielkość liter.

Separatorem części dziesiętnej liczby jest kropka.

W codziennej pracy RStudio jest wygodniejsze, jednak długotrwałe obliczenia lepiej uruchamiać w trybie wsadowym w zwykłym R.

## 1.5 R jako kalkulator

Działania matematycznie w R:

| Operator                           | Operacja         |
|------------------------------------|------------------|
| <code>+</code>                     | dodawanie        |
| <code>-</code>                     | odejmowanie      |
| <code>*</code>                     | mnożenie         |
| <code>/</code>                     | dzielenie        |
| <code>^</code> lub <code>**</code> | potęgowanie      |
| <code>sqrt()</code>                | pierwiastkowanie |

W R istnieje także stała wbudowana `pi` przechowująca wartość liczby  $\pi$ .

Funkcja `factorial(x)` zwraca silnię z podanej wartości `x`, a `sign(x)` sprawdza znak wyrażenia i zwraca odpowiednio wartość -1 jeśli wyrażenie jest ujemne, 0 jeśli jest równe 0 i 1 dla wyrażeń dodatnich.

Funkcja `exp(x)` zwraca wartość wyrażenia  $e^x$ , natomiast funkcja `log(x)` zwraca logarytm z podanej liczby. Domyślnie jest to logarytm naturalny, ale można zmienić podstawę podając wartość argumentu `base`.

Funkcja `abs(x)` zwraca wartość absolutną wyrażenia.

### Ćwiczenie

Oblicz wartość wyrażenia:  $2 \cdot \sqrt{\pi} + \log_2 8$ .

Rozwiązanie:

```
2*sqrt(pi)+log(8,2)
```

```
## [1] 6.544908
```

### Zadania

Oblicz wartość wyrażen:

1.  $\frac{2^3 \cdot 6^2}{(\frac{1}{2})^2 \cdot (\frac{4}{5})^3}$
2.  $\sqrt[3]{\frac{6-3.5}{2^{11}}}$
3.  $\pi + \sqrt{e^4}$
4.  $5! - \log_{10} 100$
5.  $|1 - e|$



## Chapter 2

# Struktury danych

W R praktycznie wszystko jest obiektem. Może to być zbiór danych, ale także wykres czy mapa. Zasadnicza różnica to klasa tych obiektów i operacje jakie mogą być na nich wykonywane.

Funkcje w R wymagają jako argumentów określonych typów obiektów - stąd tak ważna jak znajomość istniejących struktur.

Każdy obiekt w R możemy przypisać do tzw. obiektu nazwanego. Wówczas jest przechowywany w pamięci i można się do niego odwołać. Przypisanie odbywa się za pomocą operatora `<-`.

```
nazwa <- obiekt
obiekt -> nazwa
```

Można także przypisywać obiekty z wykorzystaniem znaku równości `=`, ale nie jest to zalecane ponieważ symbol ten jest używany w innych miejscach np. do deklarowania wartości argumentów w funkcji.

W R dostępna jest funkcja `assign`, która także umożliwia przypisanie nazwy do obiektu:

```
assign("nazwa", obiekt)
```

## 2.1 Wektor

Wektor jest najprostszym typem danych w R. Najczęściej korzysta się z trzech typów wektorów:

- logicznych
- liczbowych
- tekstowych

Wektor tworzy się z wykorzystaniem funkcji `c()`.

### 2.1.1 Wektor wartości logicznych

Przyjmuje wartości *prawda* lub *falsz*:

```
c(TRUE, FALSE, FALSE)
```

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

lub w skróconej wersji:

```
c(T, F, F)
```

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

Do sprawdzenia długości wektora używa się funkcji `length`:

```
length(c(T, F, F))
```

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

lub korzystając z obiektu nazwanego:

```
wart_log <- c(T,F,F)
length(wart_log)
```

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

Wektory można także utworzyć poprzez replikację określonej wartości lub wektora z wykorzystaniem funkcji `rep`. Funkcja ta przyjmuje co najmniej dwa argumenty: obowiązkowo `x` - wektor wejściowy oraz jeden z następujących: `times` - liczba powtórzeń elementów wektora `x`, `each` - liczba powtórzeń elementów wektora `x` (wyjaśnienie różnicy poniżej) lub `length.out` - oczekiwana długość wektora wynikowego.

Trzy równoważne zapisy:

```
rep(x = c(T,F), times = 3)
```

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

```
rep(c(T,F), times = 3)
```

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

```
rep(c(T,F), 3)
```

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

A tak to wygląda z argumentem `each`:

```
rep(c(T,F), each = 3)
```

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

Wykorzystanie argumentu `length.out`:

```
rep(c(T,F), length.out = 5)
```

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

## 2.1.2 Wektor wartości liczbowych

W wektorze możemy przechowywać także liczby:

```
c(1, 3, -5, 2.5, .6) # nie trzeba pisać zera przed ułamkiem
```

```
## [1] 1.0 3.0 -5.0 2.5 0.6
```

Połączenie dwóch wektorów to także wektor:

```
c(c(1,2,3), c(3.5,4,4.5))
```

```
## [1] 1.0 2.0 3.0 3.5 4.0 4.5
```

Pojedyncza liczba też jest jednoelementowym wektorem:

```
length(2)
```

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

Proste ciągi o różnicy równej 1 można generować wykorzystując dwukropek:

```
1:10
```

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

lub

```
c(-5:-1,1:5)
```

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

Do generowania ciągów liczbowych o różnych różnicach wykorzystuje się funkcję `seq`, która przyjmuje następujące argumenty. Wartość początkową `from`, wartość końcową `to` oraz jeden z następujących: `by` - krok lub `length.out` - oczekiwana długość wektora.

To samo co 1:10

```
seq(1, 10, 1)
```

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

Wartości niecałkowite:

```
seq(1, 2, 0.2)
```

```
## [1] 1.0 1.2 1.4 1.6 1.8 2.0
```

Wektor wartości malejących:

```
seq(10, 1, by=1) # błędny zapis
```

```
## Error in seq.default(10, 1, by = 1): wrong sign in 'by' argument
```

```
seq(10, 1, by=-1) # poprawny zapis
```

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

Tworzenie wektora w oparciu o argument `length.out` - funkcja sama dobiera krok:

```
seq(1, 7, length.out = 13)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
```

Do generowania liczb pseudolosowych służy funkcja `runif(n)`, która do poprawnego wywołania wymaga tylko jednego argumentu - długości wektora wynikowego. Domyślnie losowane są liczby z przedziału  $[0; 1]$  (tak jak w funkcji `los()` w Excelu), można to jednak zmienić podając odpowiednie wartości argumentów `min` i `max`.

```
runif(6)
```

```
## [1] 0.4613038 0.3760703 0.4049975 0.7187415 0.4607144 0.5527203
```

Obserwacje można także generować z innych rozkładów:

- `rnorm` - rozkład normalny,
- `rchisq` - rozkład  $\chi^2$ ,
- `rt` - rozkład t-studenta,
- itp.

Wykaz wszystkich dostępnych w R rozkładów uzyskamy wywołując polecenie `help("Distributions")`.

Za każdym uruchomieniem jednej z wymienionych wyżej funkcji losujących wartości z danego rozkładu otrzymamy inne wartości:

```
runif(5)
```

```
## [1] 0.6768611 0.3017154 0.1658514 0.5314126 0.9758312
```

```
runif(5)
```

```
## [1] 0.03936093 0.14406005 0.76353684 0.66516634 0.48137439
```

Powtarzalność wyników możemy uzyskać ustalając ziarno generatora:

```
set.seed(123)
```

```
runif(5)
```

```
## [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673
```

```
set.seed(123)
```

```
runif(5)
```

```
## [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673
```

### 2.1.3 Wektor wartości tekstowych

W wektorze może być przechowywany tekst - wówczas poszczególne elementy zapisujemy w cudzysłowie lub apostrofach:

```
c("ala", "ma", "kota")
```

```
## [1] "ala" "ma" "kota"
```

```
c('ala', 'ma', 'kota')
```

```
## [1] "ala" "ma" "kota"
```

W RStudio wygodniej używać cudzysłowu, ponieważ program automatycznie go zamyka.

Istnieje także stała zawierająca litery alfabetu:

```
letters
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

### 2.1.4 Przeciążanie wektora

Jeśli w wektorze pomieszczyliśmy kilka typów zmiennych to R przekształci poszczególne wartości, tak aby stracić jak najmniej informacji:

```
c(TRUE, 2, 5)
```

```
## [1] 1 2 5
```

```
c(3, "cztery", 5)
```

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

W pierwszym przypadku wartość TRUE została przekształcona na odpowiednik liczbowy - 1. Z kolei w drugim przykładzie podane liczby zostały przekonwertowane na tekst.



### 2.1.5 Operacje na wektorach

Na wektorach logicznych i liczbowych można wykonywać operacje arytmetyczne np. mnożenie:

```
1:10*2
```

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

Wektor liczbowy plus wektor liczbowy:

```
1:10 + c(1,2)
```

```
## [1] 2 4 4 6 6 8 8 10 10 12
```

Wektor liczbowy razy wektor liczbowy:

```
1:10 * c(1,2)
```

```
## [1] 1 4 3 8 5 12 7 16 9 20
```

Wektor liczbowy razy wektor logiczny:

```
1:10 * c(T, F)
```

```
## [1] 1 0 3 0 5 0 7 0 9 0
```

Długości obu wektorów muszą być odpowiednie:

```
1:10 * c(T,F,T)
```

```
## Warning in 1:10 * c(T, F, T): longer object length is not a multiple of
## shorter object length
```

```
## [1] 1 0 3 4 0 6 7 0 9 10
```

Dłuższy z wektorów musi być wielokrotnością krótszego.

Siłą rzeczy działania arytmetyczne na wektorach tekstowych nie są możliwe:

```
c("jeden", "dwa", "trzy", "cztery") * c(T,F)
```

```
## Error in c("jeden", "dwa", "trzy", "cztery") * c(T, F): non-numeric argument to binary operator
```

```
c("jeden", "dwa", "trzy", "cztery") + c(1,2)
```

```
## Error in c("jeden", "dwa", "trzy", "cztery") + c(1, 2): non-numeric argument to binary operator
```

### 2.1.6 Operacje agregujące

Na wektorach można także wykonywać operacje agregujące:

| Funkcja    | Działanie                        |
|------------|----------------------------------|
| mean()     | średnia elementów                |
| sum()      | suma elementów                   |
| prod()     | iloczyn elementów                |
| var()      | wariancja elementów              |
| sd()       | odchylenie standardowe elementów |
| median()   | mediana elementów                |
| quantile() | kwantyl danego rzędu             |
| min()      | minimum                          |
| max()      | maksimum                         |

Obliczenie skośności i kurtozy jest możliwe po zainstalowaniu pakietu `e1071`. Wówczas mamy dostęp do funkcji:

| Funkcja                 | Działanie          |
|-------------------------|--------------------|
| <code>skewness()</code> | skośność elementów |
| <code>kurtosis()</code> | kurtoza elementów  |

Suma wektora numerycznego:

```
sum(1:10)
```

```
## [1] 55
```

Suma i średnia wektora logicznego:

```
sum(c(T, F, F, T))
```

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

```
mean(c(T, F, F, T))
```

```
## [1] 0.5
```

Korzystanie z funkcji pochodzących z pakietów zewnętrznych wymaga wskazania skąd pochodzi dana funkcja. Można to zrobić na dwa sposoby: funkcją `library(pakiet)` - wówczas wszystkie funkcje z tego pakietu są wczytywane do pamięci i można do nich sięgać bezpośrednio lub wskazując przed nazwą funkcji z jakiego pakietu pochodzi.

Wczytanie pakietu:

```
library(e1071)
skewness(c(1,2,3,4,5,7,9,11,13))
```

```
## [1] 0.3451259
```

lub równoważnie:

```
e1071::skewness(c(1,2,3,4,5,7,9,11,13))
```

```
## [1] 0.3451259
```

Podsumowanie rozkładu wektora można także uzyskać z wykorzystaniem funkcji `summary(x)`:

```
summary(1:10)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00   3.25   5.50   5.50   7.75   10.00
```

Działa także na wektorach tekstowych:

```
summary(c("jeden", "dwa", "trzy", "cztery"))
```

```
##      Length      Class      Mode
##           4 character character
```

### 2.1.7 Sprawdzanie typu wektora

Do określenia typu wektora służy funkcja `typeof` lub `mode`.

```
typeof(wart_log)
```

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

Sprawdzenie czy obiekt jest danego typu odbywa się z wykorzystaniem dedykowanych funkcji z przyrostkiem `is`.

```
is.logical(wart_log)
```

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

```
is.character(wart_log)
```

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

### 2.1.8 Rzutowanie wektorów

Czasami jako np. argument funkcji będzie wymagany inny typ wektora aniżeli aktualnie posiadany w pamięci. Można wówczas spróbować przekształcić taki wektor z wykorzystaniem funkcji rozpoczynającej się od `as`:

```
typeof(wart_log)
```

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

```
as.numeric(wart_log)
```

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

```
typeof(as.numeric(wart_log))
```

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

### 2.1.9 Indeksowanie wektorów

Aby uzyskać dostęp do części wektora korzysta się z indeksatora w postaci nawiasów kwadratowych. Utworzymy nowy wektor zawierający liczby całkowite od 10 do 20:

```
wart_10_20 <- seq(10,20)
```

```
wart_10_20
```

```
## [1] 10 11 12 13 14 15 16 17 18 19 20
```

a następnie wybieramy trzecią obserwację:

```
wart_10_20[3]
```

```
## [1] 12
```

Możemy także odwołać się do większego zakresu:

```
wart_10_20[3:5]
```

```
## [1] 12 13 14
```

I wybranych elementów:

```
wart_10_20[c(1,3,5)]
```

```
## [1] 10 12 14
```

Wybór obserwacji większych od 15:

```
wart_10_20[wart_10_20>15]
```

```
## [1] 16 17 18 19 20
```

Z kolei następujący zapis zwróci nam wektor wartości logicznych:

```
wart_10_20 > 15
```

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

### 2.1.10 Wartości nieliczbowe

Brak danych w R jest przedstawiany jako wartość NA (ang. *not available*) i może powodować trudności z wywoływaniem niektórych funkcji:

```
v_na <- c(1,2,1,NA,1)
v_na
```

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

```
sum(v_na)
```

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

W związku z tym większość funkcji ma zaimplementowany dodatkowy argument służący do obsługi tego typu wartości, który najczęściej nie uwzględnia tych wartości w obliczeniach:

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

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

Oprócz braku danych podczas obliczeń możemy natrafić na wartości nieokreślone NaN (ang. *not a number*) oraz nieskończone Inf (ang. *infinity*).

```
0/0
```

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

```
1/0
```

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

```
sqrt(-10)
```

```
## Warning in sqrt(-10): NaNs produced
```

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

W R istnieje także wartość NULL, która jest podstawowym typem danych a nie wartością. NULL można traktować jako odpowiednik zbioru pustego. Jest stosowany np. w funkcjach, które niczego nie zwracają.

```
v_null <- c(1,2,1,NULL,1)
v_null
```

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

```
sum(v_null)
```

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

### 2.1.11 Zadania

1. Ile wynosi suma elementów większych od 10 dla następujących liczb: 12, 5, 20, 18, 8.5, 10, 4, 101, -2?
2. Utwórz następujący wektor: 2 0 0 4 0 0 6 0 0 8 0 0.
3. Dane są dwa wektory - a: 2, 3, 7, 8, 2, b: 9, 1, 2, 0, 2. Jakiego typu będzie wektor będący wynikiem działania  $a \leq b$ ?
4. Uzupełnij wektor `letters` o polskie litery diakrytyzowane. Jaka długość ma nowo utworzony wektor?
5. Wylosuj z rozkładu normalnego 1000 obserwacji z ziarnem równym 76. Ile wynosi kurtoza tych wartości?

## 2.2 Macierz

Macierze są wykorzystywane w R do przechowywania np. odległości pomiędzy punktami czy wskazywania sąsiedztwa obszarów geograficznych.

Do tworzenia macierzy służy funkcja `matrix`:

```
m <- matrix(1:6, nrow = 2, ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Z wykorzystaniem wybranych funkcji można sprawdzić wymiary macierzy, liczbę wierszy oraz kolumn:

```
dim(m)
```

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

```
ncol(m)
```

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

```
nrow(m)
```

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

Macierz może także zawierać tekst:

```
matrix(letters[1:9], nrow=3)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "d"  "g"
## [2,] "b"  "e"  "h"
## [3,] "c"  "f"  "i"
```

Domyślnie macierz układana jest kolumnami. Aby to zmienić należy dodać argument `byrow=TRUE`:

```
matrix(letters[1:9], nrow=3, byrow=TRUE)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "b"  "c"
## [2,] "d"  "e"  "f"
## [3,] "g"  "h"  "i"
```

Jeśli liczba elementów wejściowych jest mniejsza iloczyn podanej liczby kolumn i wierszy to w brakujące miejsce wstawiane są elementy z początku wektora wejściowego:

```
matrix(letters[1:7], nrow=3, byrow=TRUE)
```

```
## Warning in matrix(letters[1:7], nrow = 3, byrow = TRUE): data length [7] is
## not a sub-multiple or multiple of the number of rows [3]
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "b"  "c"
## [2,] "d"  "e"  "f"
## [3,] "g"  "a"  "b"
```

Z kolei macierz diagonalną posiadającą elementy niezerowe wyłącznie na przekątnej tworzy się z wykorzystaniem funkcji `diag`. Macierz jednostkowa o wymiarach  $4 \times 4$ :

```
diag(4)
```

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

Macierz diagonalna o wartościach 5 na przekątnej i wymiarach  $3 \times 3$

```
diag(5, nrow=3, ncol=3)
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    0
## [2,]    0    5    0
## [3,]    0    0    5
```

Funkcja `diag` umożliwia także ekstrakcję przekątnej z istniejącej już macierzy:

```
diag(matrix(letters[1:9], nrow=3))
```

```
## [1] "a" "e" "i"
```

### 2.2.1 Łączenie macierzy

Z wykorzystaniem funkcji `rbind` i `cbind` można odpowiednio łączyć obiekty wierszami (ang. *row bind*) lub kolumnami (ang. *col bind*):

```
rbind(m, c(99, 88, 77))
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## [3,]   99   88   77
```

```
cbind(m, matrix(101:104, nrow=2))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5  101  103
## [2,]    2    4    6  102  104
```

### 2.2.2 Indeksowanie macierzy

Dostęp do poszczególnych elementów macierzy odbywa się z wykorzystaniem nawiasów kwadratowych, ale można podać dwie wartości - obiekt[wiersz,kolumna]:

```
m[2,1] # drugi wiersz, pierwsza kolumna
```

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

```
m[2,] # tylko drugi wiersz
```

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

```
m[,1] # tylko pierwsza kolumna
```

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

```
m[,] # wszystkie obserwacje
```

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

```
## [1,]    1    3    5
```

```
## [2,]    2    4    6
```

```
m[] # wszystkie obserwacje
```

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

```
## [1,]    1    3    5
```

```
## [2,]    2    4    6
```

W ten sposób można dokonać modyfikacji konkretnych elementów macierzy:

```
m[2,1] <- 77
```

```
m
```

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

```
## [1,]    1    3    5
```

```
## [2,]   77    4    6
```

### 2.2.3 Operacje na macierzach

Na macierzach można wywołać szereg operacji:

| Operator/funkcja         | Działanie                          |
|--------------------------|------------------------------------|
| <code>a %*% b</code>     | mnożenie macierzy a i b            |
| <code>t(a)</code>        | transpozycja macierzy a            |
| <code>det(a)</code>      | wyznacznik macierzy a              |
| <code>solve(a)</code>    | macierz odwrotna z a               |
| <code>solve(a, b)</code> | rozwiązanie układu $a \cdot x = b$ |

Rozważmy dwie macierze:

```
a <- matrix(c(2, 3, 4, 2, 1, 2, 1, 3, 2), nrow = 3)
```

```
b <- matrix(6:1, ncol=2)
```

```
a;b
```

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

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

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

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

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

```
## [1,]    6    3
```

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

Aby przeprowadzić mnożenie macierzy **a** i **b**, liczba kolumn macierzy **a** musi być równa liczbie wierszy w macierzy **b**. Z kolei rozmiar macierzy wyjściowej to liczba wierszy macierzy **a** i liczba kolumn macierzy **b**.

```
a %*% b
```

```
##      [,1] [,2]
## [1,]   26   11
## [2,]   35   14
## [3,]   42   18
```

Transpozycja macierzy **b**:

```
t(b)
```

```
##      [,1] [,2] [,3]
## [1,]    6    5    4
## [2,]    3    2    1
```

Wyznacznik macierzy **a**:

```
det(a)
```

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

Macierz odwrotna do macierzy **a**:

```
solve(a)
```

```
##      [,1] [,2] [,3]
## [1,] -0.6666667 -0.3333333 0.8333333
## [2,] 1.0000000 0.0000000 -0.5000000
## [3,] 0.3333333 0.6666667 -0.6666667
```

Wyznaczenie macierzy **x** w równaniu  $\mathbf{a} \cdot \mathbf{x} = \mathbf{b}$ :

```
solve(a,b)
```

```
##      [,1] [,2]
## [1,] -2.333333 -1.833333
## [2,] 4.000000 2.500000
## [3,] 2.666667 1.666667
```

```
a %*% solve(a,b)
```

```
##      [,1] [,2]
## [1,]    6    3
## [2,]    5    2
## [3,]    4    1
```

```
b
```

```
##      [,1] [,2]
## [1,]    6    3
## [2,]    5    2
## [3,]    4    1
```

## 2.2.4 Zadanie

Estymator parametrów beta w metodzie najmniejszych kwadratów jest dany wzorem:



$$b = (X'X)^{-1}X'y$$

Zmienna  $x_1$  przyjmuje wartości 2,4,1,6,9,3,2,9,10,7, zmienna  $x_2$  1.5,0.2,0.1,2,3.1,1.2,0.4,2.9,2.5,1.9, a zmienna  $x_0$  to wektor jedynek. Te trzy zmienne tworzą macierz  $X$ . Z kolei wartości zmiennej  $y$  są następujące 12,15,10,19,26,13,13,21,29,18. Wyznacz wartość  $b$ .

## 2.3 Czynniki

Czynnik (ang. *factor*) służy do przechowywania danych jakościowych o małej liczbie kategorii, mierzonych na skali nominalnej i porządkowej.

Rozważmy informacje o wykształceniu:

```
wyk <- rep(c("podstawowe", "średnie", "wyższe"), c(5,3,2))
wyk
```

```
## [1] "podstawowe" "podstawowe" "podstawowe" "podstawowe" "podstawowe"
## [6] "średnie"      "średnie"      "średnie"      "wyższe"      "wyższe"
```

i dokonajmy transformacji na czynnik:

```
wyk_f <- factor(wyk)
wyk_f
```

```
## [1] podstawowe podstawowe podstawowe podstawowe podstawowe średnie
## [7] średnie      średnie      wyższe      wyższe
## Levels: podstawowe średnie wyższe
```

Funkcja `summary()` wywołana na czynniku zwraca wynik innego typu aniżeli na wektorze tekstowym:

```
summary(wyk)
```

```
##      Length      Class      Mode
##          10 character character
```

```
summary(wyk_f)
```

```
## podstawowe      średnie      wyższe
##           5           3           2
```

Jeśli chcemy zaakcentować fakt, że zmienne są mierzone na skali porządkowej dodajemy argument `ordered=TRUE`:

```
wyk_of <- factor(wyk, ordered = TRUE)
wyk_of
```

```
## [1] podstawowe podstawowe podstawowe podstawowe podstawowe średnie
## [7] średnie      średnie      wyższe      wyższe
## Levels: podstawowe < średnie < wyższe
```

W łatwy sposób możemy edytować etykiety:

```
levels(wyk_of) <- c("pod.", "śr.", "wyż.")
wyk_of
```

```
## [1] pod. pod. pod. pod. pod. śr.  śr.  śr.  wyż. wyż.
## Levels: pod. < śr. < wyż.
```

Czynniki mają szczególne znaczenie w przypadku tworzenia wykresów, gdy chcemy określić porządek wyświetlania.

## 2.4 Lista

Listy to ciągi złożone z elementów o dowolnych typach. Mogą przydać się w szczególności przy budowaniu funkcji, które zwracają tylko jedną wartość. Wówczas dane różnego typu mogą być zawarte w takiej liście.

Tworzenie prostej listy:

```
l <- list(TRUE, c(1,2,3,4), "element tekstowy")
l
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] 1 2 3 4
##
## [[3]]
## [1] "element tekstowy"
```

Już na pierwszy rzut oka widać bardziej złożoną strukturę listy. W związku z tym odwoływanie do poszczególnych elementów będzie trochę się różnić od wektorów czy macierzy.

```
l[2] # druga lista
```

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

```
l[[2]] # zawartość listy
```

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

```
l[[2]][3] # trzeci element wektora drugiej listy
```

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

Listę można także rozwinąć do wektora z wykorzystaniem funkcji `unlist`:

```
unlist(l)
```

```
## [1] "TRUE"          "1"              "2"
## [4] "3"             "4"              "element tekstowy"
```

Poszczególne elementy listy można nazwać:

```
ln <- list(log=TRUE, num=c(1,2,3,4), tekst="element tekstowy")
ln
```

```
## $log
## [1] TRUE
##
## $num
## [1] 1 2 3 4
##
## $tekst
## [1] "element tekstowy"
```

Wówczas można uzyskać do nich dostęp poprzez symbol `$` i podaną nazwę:

```
ln$num

## [1] 1 2 3 4
ln[[2]] # normalne indeksowanie nadal działa

## [1] 1 2 3 4
ln$num[2]

## [1] 2
```

## 2.5 Ramka danych

Ramka danych to tabela, która przypomina tą z Excela zawierająca dane o różnych typach. Tworzona za pomocą funkcji `data.frame`:

```
df <- data.frame(plec=c("m", "k", "k", "m", "k", "m", "m", "m"),
                  wzrost=c(173, 170, 163, 178, 169, 180, 175, NA),
                  pali=c(T, F, F, F, T, F, NA, T))
```

W RStudio po wybraniu tego obiektu w zakładce **Environment** pojawia się przyjazne okno do przeglądania oraz poglądowego filtrowania i sortowania danych ze zbioru.

Możemy zobaczyć podsumowanie całego zbioru wywołując na nim funkcję `summary()`:

```
summary(df)

##   plec      wzrost      pali
## k:3   Min.   :163.0   Mode :logical
## m:5   1st Qu.:169.5   FALSE:4
##       Median :173.0   TRUE :3
##       Mean   :172.6   NA's :1
##       3rd Qu.:176.5
##       Max.   :180.0
##       NA's   :1
```

Ramki danych można indeksować w taki sam sposób jak macierze lub z wykorzystaniem operatora `$`:

```
df[,2] # druga kolumna

## [1] 173 170 163 178 169 180 175 NA

df$wzrost # kolumna wzrost

## [1] 173 170 163 178 169 180 175 NA

df[,c("plec", "pali")]

##   plec  pali
## 1    m  TRUE
## 2    k FALSE
## 3    k FALSE
## 4    m FALSE
## 5    k  TRUE
## 6    m FALSE
## 7    m   NA
## 8    m  TRUE
```

Z kolei do wyboru obserwacji można wykorzystać warunek:

```
df[df$plec=="m",]
```

```
##   plec wzrost  pali
## 1    m    173  TRUE
## 4    m    178 FALSE
## 6    m    180 FALSE
## 7    m    175   NA
## 8    m     NA  TRUE
```

Wyodrębnienie informacji o wzroście tylko dla kobiet i wyznaczenie średniej:

```
wzrost_k <- df$wzrost[df$plec == "k"]
wzrost_k
```

```
## [1] 170 163 169
```

```
mean(wzrost_k)
```

```
## [1] 167.3333
```

Widzimy, że dla mężczyzn nie udało się ustalić wszystkich informacji i jeden z nich nie ma podanego wzrostu, a dla drugiego brakuje informacji o paleniu papierosów. Możemy usunąć braki danych w kolumnach korzystając z funkcji `complete.cases()`:

```
df[complete.cases(df$wzrost),] # tylko zmienna wzrost
```

```
##   plec wzrost  pali
## 1    m    173  TRUE
## 2    k    170 FALSE
## 3    k    163 FALSE
## 4    m    178 FALSE
## 5    k    169  TRUE
## 6    m    180 FALSE
## 7    m    175   NA
```

```
df[complete.cases(df),] # wszystkie zmienne
```

```
##   plec wzrost  pali
## 1    m    173  TRUE
## 2    k    170 FALSE
## 3    k    163 FALSE
## 4    m    178 FALSE
## 5    k    169  TRUE
## 6    m    180 FALSE
```

Zbiory danych przechowywane są także w R i pochodzą z różnych pakietów. Wywołując funkcję `data("zbior")` ładujemy dany zbiór do pamięci. Do szybkiego podglądu zebranych danych służy funkcja `head()`, która domyślnie wyświetla 6 pierwszych obserwacji ze zbioru:

```
data("iris")
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1           3.5           1.4           0.2  setosa
## 2           4.9           3.0           1.4           0.2  setosa
## 3           4.7           3.2           1.3           0.2  setosa
## 4           4.6           3.1           1.5           0.2  setosa
## 5           5.0           3.6           1.4           0.2  setosa
```

```
## 6          5.4          3.9          1.7          0.4 setosa
```

### 2.5.1 Zadania

Załaduj do pamięci zbiór o nazwie ChickWeight.

1. Ile razy jedzenie otrzymał kurczak o numerze 15?
2. Ile wynosi mediana wagi kurczaka o numerze 35?
3. Ile średnio ważyły kurczaki na diecie nr 1, a ile na diecie nr 2?



## Chapter 3

# Przetwarzanie danych

3.1 Pakiet tidyverse

3.2 Import danych

3.3 Wybieranie kolumn

3.4 Filtrowanie

3.5 Dodawanie nowych zmiennych

3.6 Grupowanie

3.7 Podsumowanie

3.8 Łączenie zbiorów

3.9 Wąska i szeroka reprezentacja danych





## Chapter 4

# Programowanie w R

### 4.1 Funkcje

### 4.2 Pętle

### 4.3 Instrukcje warunkowe



## Chapter 5

# Wizualizacja danych

### 5.1 Wbudowane funkcje

### 5.2 Pakiet ggplot2