Kurs R - Tutorial

Agnieszka Strzałka

2021-08-25

Contents

Chapter 1

Wstęp

Książka jest dodatkiem do kursu R i ma stanowić zbiór przykładów, które mogą zostać wykorzystane podczas pisania własnych skryptów R. Materiały będą na bieżąco aktualizowane tak aby wszystkie przedstawione przykłady były możliwe do odtworzenia.

Większość przedstawionych przykładów będzie się opierać na funkcjach zawartych w pakietach tidyverse, które tworzą spójną całość obejmującą wczytanie i analizę danych oraz ich wizualizację. Chociaż wiele z przedstawionych zadań może być wykonane korzystając z podstawowych funkcji R uważam, że nauczenie się od razu korzystania z pakietów tidyverse za bardziej korzystne gdyż zawarte w nich funkcje są często łatwiejsze i bardziej intuicyjne w stosowaniu, szczególnie gdy ktoś nie ma dużego doświadczenia w programowaniu ;)

```
##
## R is the lingua franca of statistical research. Work in all other languages
## should be discouraged.
##    -- Jan de Leeuw (as quoted by Matt Pocernich on R-help)
##       JSM 2003, San Francisco (August 2003)
```

Chapter 2

Skąd ściągnąć R

- Postawowe środowisko R najlepiej ściągnąć bezpośrednio ze strony R project
- Polecam również ściągnąć R Studio, które bardzo ułatwia pracę z R
- Podstawowe pakiety zostaną zainstalowane razem z R, kolejne można pobrać bezpośrednio przez R Studio (zakładka Packages), bardziej bioinformatyczne pakiety znajdują się na stronie bioconductor, wiele pakietów można też pobrac bezpośrednio ze strony Github korzystając z pakietu devtools.

```
##
## Friends don't let friends use Excel for statistics!
##    -- Jonathan D. Cryer (about problems with using Microsoft Excel for
##       statistics)
##       JSM 2001, Atlanta (August 2001)
```

2.1 Pomoc

- Pomoc do funkcji można otworzyć bezpośrednio w R wpisując: `?nazwa_funkcji` albo naciskając F1 podczas wpisywania nazwy funkcji, jednak opisy często są dość enigmatyczne i zakładają już pewne zrozumienie tematu
- Odpowiedzi na konkretne pytania najlepiej szukać w internecie wpisując po prostu: "How do sth R", najczęściej pojawiająca się strona to Stack Overflow
- Ciekawe pomysły i tutoriale pojawiają się też na stronie R-bloggers
- Interaktywny kurs R można też znaleźć na stronie datacamp, ale obecnie tylko pierwsze lekcje każdego kursu są darmowe
- Pomoc do ggplot2

- Polecam też książkę Przemysława Biecka "Przewodnik po pakiecie R", jedna z niewielu jakie są po polsku, pierwsze rozdziały można bezpłatnie pobrać ze strony autora
- Introduction to R for Biologists
- How to make any plot in ggplot2?
- Fundamentals of Data Visualization
- R Graphics Cookbook, 2nd edition
- No i oczywiście ja chętnie pomogę :)

```
##
## You need to get the hang of reading the online help. The information required
## is actually there in ?dotchart --- it's just tersely and obscurely expressed. A
## certain degree of optimism is required. You need to ***believe*** that the
## information is there; then ask yourself "What could they possibly mean by what
## they have written that would tell me what I need to know?".
##    -- Rolf Turner (on reading the help pages)
##       R-help (June 2013)
```

2.2 Packages

Żeby użyć funkcję z danego pakietu, który nie należy do podstawowych należy go najpierw zainstalować (Install w zakładce Packages), a potem załadować. (funkcje `library(nazwa)` albo `require(nazwa)`). Można też włączać pakiety w zakładce Packages.

Dobrze jest ładować tylko te pakiety, które są faktycznie potrzebne - nie przeciążamy pamięci i niektóre nazwy funkcji mogą się powtarzać w różnych pakietach.

Jeżeli ładowanie całego pakietu jest niepotrzebne albo prowadzi do konflktów można odwołać się do konkretnej funckji przy pomocy :: np. `readxl::read_excel()`.

Każdy pakiet zawiera podstawowy opis funkcji, niektóre posiadają bardziej rozbudowane przykłady analiz jakie można przy ich pomocy wykonać - vignette. Jeżeli chcemy sprawdzić które pakiety zawierają winietki można to zrobić funkcją `browseVignettes`.

2.2.1 Lista pakietów, które pojawiają się w książce (uwaga może być niekompletna)

Przed rozpoczęciem należy mieć zainstalowane następujące pakiety: ggplot2, tidyr, knitr, dplyr, readr, readxl oraz najlepiej posiadać najnowszą wersję R (4.1.0 - "Camp Pontanezen") i R Studio (przynajmniej 1.4).

Pozostałe pakiety można pobrać tylko jeśli będą potrzebne.

Inne pakiety jakie pojawiają się to: w części dotyczącej wykresów: car, likert, gplots, VennDiagram, corrplot, hexbin, aplpack, GGally, ggthemes, ggthemr i w części dotyczącej statystyki i obróbki danych: car, MASS, nortest, modeest, moments, agricolae, drc, broom.

2.3 R Studio

Jest to obecnie najpopularniejszy dostępny edytor R, pozwalający na tworzenie projektów, pisanie i zarządzanie skryptami, pozwalający na łatwy dostęp do historii wpisywanych komend i tworzonych wykresów. Został zintegrowany z wieloma przydatnymi pakietami np. knitr, który pozwala tworzenie raportów w języku markdown, latex, prezentacji multimedialnych itp.

R Studio pozwala również na założenie projektu do konkretnego zadania, wszystkie pliki tworzone w trakcie pracy (wykresy, tabele, skrypty) zostaną umieszczone w folderze przypisanym do projektu. Jeżeli nasze dane wejściowe umieścimy w tym samym miejscu nie będzie konieczne podawanie całej ścieżki dostępu przy wczytywaniu pliku. Wykorzystanie projektów ułtwia uporządkowanie pracy. Każdy projekt można też poddać kontroli wersji przy pomocy Git i Github bezpośrednio z Rstudio. Więcej informacji na ten temat można znaleźć w Happy Git with R.

Cały Tutorial to zbiór skryptów napisanych w markdown, które zostały połączone w książkę dzięki pakietowi R bookdown. Wszystkie przykłady można skopiować albo przepisać do własnych skryptów R. Trzeba jedynie pamiętać o wcześniejszym załadowaniu odpowiednich pakietów i ewentualnie danych. Samodzielne wpisywanie nazw funkcji przyśpiesza proces zapamiętywania, a korzystając z autouzpełniania trzeba tak naprawdę pamiętać 2-3 pierwsze litery :)

Chapter 3

Dane

3.1 Wczytanie danych

Dane najłatwiej wczytać z plików .txt lub .csv. Można również z plików .xlsx lub .xls ale często trwa to dłużej i może wymagać zainstalowanych innych pakietów/programów.

W każdym przypadku niezbędne jest podanie ścieżki dostępu do pliku. Jeżeli znajduje się on folderze projektu to wystarczy jego nazwa. Dobrą praktyka jest przechowywanie danych w osobnym katalogu w obrębie projektu - wtedy ścieżka może wyglądać np. data/dane_1.txt. Jeżeli plik z danymi jest przechowywany gdzie indziej konieczne jest podanie pełnej ścieżki dostępu, ale można w tym wypadku korzystać z autouzupełniania klawiszem Tab.

3.1.1 Wczytanie danych z plików tekstowych

R Studio posiada funkcję Import Dataset (zakładka Environment), która pozwala na wczytanie danych wraz z wyborem podstawowych opcji jak separator, separator dziesiętny, obecność nagłówków. Opcja ta obejmuje podstawowe funkcje R oraz pakiety readr i readxl będące częścią tidyverse. Na początek jest łatwiej korzystać z opcji Load Dataset, ponieważ po jej użyciu zostaje również wygenerowany odpowiedni kod R, który można wkleić do własnych skryptów tak aby te same lub podobne dane wczytać już bez problemu następnym razem.

Można również wczytać dane wpisując komendę (wygodne przy pisaniu skryptów) `read.table` albo `read.csv`. Argumenty: file określa nam plik, który wczytujemy, header - nagłówki, sep - separator np. "/t" to tabulator, dec - separator dziesiętny (domyślnie kropka), quote - obecność "". W przypadku nagłówków kolumn wpisanie `header=TRUE` spowoduje, że pierwszy

wiersz naszej tabeli zostanie potraktowany jako tytuły kolumn, `header=FALSE` oznacza domyślne nazwy kolumn - V1, V2, …

W pakiecie readr znajdują się analogiczne funckje: * read_delim - do plików tekstowych, funckja spróbuje zgadnąć jak rozdzielone są kolumny, można podać argumentem delim * read_csv i read_csv2 - do plików csv, odpowiednio do danych wykorzystujących kropki i przecinki jako separatory dziesiętne * read_tsv i read_table - do plików gdzie kolumny są rodzielone przy pomocy tab.

Wczytane dane będą widoczne w zakładce Environment. Kliknięcie na nazwę tabeli spowoduje otwarcie widoku danych podobnego do arkusza kalkulacyjnego. Można w nim dane sortować i filtrować, ale nie edytować.

Pierwsze albo ostanie wiersze można zobaczyć używając funkcji `head` albo `tail`. Strukturę danych pokaże funkcja `str`, a podstawowe informacje `summary`. Dobrze jest po wczytaniu danych sprawdzić czy wyglądają faktycznie tak jak miały ;)

```
dane1 <- read.table(file = "data/dane1.txt", header=TRUE, quote="\"")

head(dane1)
```

```
##     pomiar Szczep warunki
## 1 3.389738      A       1
## 2 5.992065      A       1
## 3 4.464553      A       1
## 4 4.546082      A       1
## 5 6.521751      A       1
## 6 5.713088      A       1
```

```
str(dane1)
```

```
## 'data.frame':    1800 obs. of  3 variables:
##  $ pomiar : num  3.39 5.99 4.46 4.55 6.52 ...
##  $ Szczep : chr  "A" "A" "A" "A" ...
##  $ warunki: int  1 1 1 1 1 1 1 1 1 ...
```

```
summary(dane1)
```

```
##      pomiar          Szczep             warunki
##  Min.   : 1.434   Length:1800        Min.   :1.0
##  1st Qu.: 4.450   Class :character   1st Qu.:1.0
##  Median : 5.478   Mode  :character   Median :1.5
##  Mean   : 5.846                      Mean   :1.5
##  3rd Qu.: 6.743                      3rd Qu.:2.0
##  Max.   :27.953                      Max.   :2.0
```

3.1.2 Wczytanie danych z plików excel

Do wczytania danych z plików xlsx można wykorzystać funkcję read_excel z pakietu readxl. Przy pracy z excelem należy jednak pamiętać żeby wczytywane dane nie zawierały formuł albo połączonych komórek oraz, że funkcja wczyta dane zawarte tylko w jednym arkuszu (sheet), ale można wybrać z której przy pomocy argumentu sheet.

```
library(readxl)

data_excel <- read_excel('data/test.xlsx')

data_excel
```

```
## # A tibble: 16 x 3
##        A     B     C
##    <dbl> <dbl> <dbl>
##  1     1     2     3
##  2     1     3     4
##  3     1     4     5
##  4     1     5     6
##  5     1     6     7
##  6     1     7     8
##  7     1     8     9
##  8     1     9    10
##  9     1    10    11
## 10     1    11    12
## 11     1    12    13
## 12     1    13    14
## 13     1    14    15
## 14     1    15    16
## 15     1    16    17
## 16     1    17    18
```

3.2 Zapisanie danych

Zapisać tabelę danych można korzystając z funkcji `write.table`, wystarczy określić, co chcemy zapisać i nazwę pliku wyjściowego. Możemy zapisywać pliki w formacie np. txt, csv. Jeżeli nie określimy ścieżki dostępu plik zostanie zapisany w folderze projektu.

Można również dopisywać dane do istniejącego pliku ustawiając parametr `append = TRUE`. Domyślnie ten argument jest zawsze ustawiony jako `append =`

`FALSE` i jeżeli w nazwie pliku podamy istniejący plik to **zostanie on nadpisany bez ostrzeżenia**.

Można też dane zapisać w formacie rds. Można je potem otworzyć tylko w R, ale można w takim formacie zapisać każdy obiekt R, nie tylko tabele. Do zapisania danych służy funkcja saveRDS, a do wczytanie readRDS.

```
write.table(dane1, "data/dane4.txt")
```

3.3 Podstawowe typy danych w R

W R dane do zmiennej przypisujemy korzystając ze strzałki ;) `<-` albo `->`. Można też użyć `=`

Do najbardziej podstawowych typów danych należą: typ liczbowy, znakowy, logiczny i czynnikowy (o tym później). Typ danych można sprawdzić funkcją `class`

Najczęściej wykorzystywane rodzaje danych to wektor i ramka danych. Pozostałe to m.in. macierz i lista. W R można również pisać własne funkcje.

3.3.1 Wektor (vector)

Wektor to ciąg elementów tego samego typu np. liczbowy, znakowy. W R nawet pojedyncza cyfra jest wektorem. Wektor można stworzyć korzystając z funkcji:

- `c`. Sekwencję liczb można zapisać jako `1:10` - utworzy wektor liczb od 1 do 10.
- Przy bardziej skomplikowanych sekwencjach można użyć funkcji `seq`, ustawiamy start, koniec i co ile ma być kolejny element
- funkcję `rep`, podajemy jakie elementy mają zostać powtórzone i ile razy. `each` - powtarzanie każdego elementu, `times` - powtarzanie całego wektora.

```
a <- c(1,2,5,8)
a
```

```
## [1] 1 2 5 8
```

```
b <- c("A", "B", "V")
b
```

```
## [1] "A" "B" "V"
```

```
c <- 1:15
c
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
d <- seq(10, 100, by=10)
d
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

```
e <- rep(c(1,2,3), times=3)
e
```

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

```
e <- rep(c(1,2,3), each=3)
e
```

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

Do poszczególnych elementów wektora można się odwołać stosując nawiasy: `wektor[x]`, gdzie x to numer elementu. Można się odwołać do kilku elementów jednocześnie np. `wektor[1:3]`, `wektor[c(2,4)]`. Numeracja rozpoczyna się od 1. Elementy wektora mogą również mieć swoje nazwy i wtedy można je także wykorzystać do wyświetlania danych.

Ilość elementów wektora podaje funkcja `length`.

```
a <- c(1:5)
a[2]
```

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

```
b <- c( "jeden" = 1, "dwa" = 2, "trzy" = 3)
b[2]
```

```
## dwa
##   2
```

```
b["dwa"]
```

```
## dwa
##   2
```

```
length(a)
```

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

Na wektorach można wykonywać wszystkie operacje matematyczne. Przykładowo jeżeli dodamy do siebie dwa wektory to zawsze pierwszy element zostanie dodany do pierwszego elementu z drugiego wektora itd.

```
1:10 + 11:20
```

```
##  [1] 12 14 16 18 20 22 24 26 28 30
```

```
1:10 * 11:20
```

```
##  [1]  11  24  39  56  75  96 119 144 171 200
```

3.3.2 Ramka danych (data frame)

Ramka danych to po prostu kilka wektorów ułożonych w tabelę. Wektory mogą być różnego typu, powinny być tej samej długości. Jeżeli mają różną długość można uzupełnić brakujące elementy stosując NA - brak danych

Skoro każda kolumna ramki danych jest wektorem to można stosować na nich te same operacje matematyczne np. dodawać albo dzielić.

Ramkę danych tworzymy przy użyciu funkcji `data.frame`

Jeżeli chcemy coś zmienić w ramkę danych używamy `as.data.frame`

Podstawowe informacje o ramce danych

Funkcja	Opis
`nrow`	liczba wierszy
`ncol`	liczba kolumn
`colnames`	nazwy kolumn
`rownames`	nazwy wierszy
`dim`	wymiary

```
x <- data.frame(a=1:5, b=rep("A",5), c=c(T,T,T,F,NA))
x
```

```
##   a b     c
## 1 1 A  TRUE
## 2 2 A  TRUE
## 3 3 A  TRUE
## 4 4 A FALSE
## 5 5 A    NA
```

```
colnames(x)
```

```
## [1] "a" "b" "c"
```

Do elementów ramki danych również można się odwołać przy pomocy nawiasów: `ramka[x,y]`, gdzie x to numer wiersza, a y numer kolumny. Jeżeli podamy jedynie numer kolumny otrzymamy wszystkie jej elementy.

Innym sposobem jest wykorzystanie nazw kolumn i $ np `ramka$kolumna_1`. W ten sposób można łatwo dodać nowe kolumny do ramki danych.

Podobnie możemy usuwać kolumny i wiersze z ramki np. `ramka<-ramka[,-1]` usunie pierwszą kolumnę.

```
ramka <- data.frame(kol1=c(1:10), kol2=c(11:20))

ramka
```

```
##    kol1 kol2
## 1     1   11
## 2     2   12
## 3     3   13
## 4     4   14
## 5     5   15
## 6     6   16
## 7     7   17
## 8     8   18
## 9     9   19
## 10   10   20
```

```
ramka$kol_3 <- ramka$kol1 + ramka$kol2

ramka
```

```
##    kol1 kol2 kol_3
## 1     1   11    12
## 2     2   12    14
## 3     3   13    16
## 4     4   14    18
## 5     5   15    20
## 6     6   16    22
## 7     7   17    24
## 8     8   18    26
## 9     9   19    28
## 10   10   20    30
```

Funkcje zawarte w pakietach tidyverse (w tym read_delim i inne) często zamiast zwykłej ramki danych używają ulepszonej struktury zwanej tibble. W codziennym użytkowaniu nie ma między nimi wielu różnic. Tibble są bezpieczniejsze od zwykłych data.frame, ponieważ nigdy nie zmieniają typów danych w kolumnach. Może się jednak czasem zdarzyć że funkcje starszych pakietów nie będą akceptować tibble zamiast data.frame (można łatwo zmienić przy pomocy `as.data.frame`.

3.3.3 Matryca (matrix)

Matryca jest trochę podobna do ramki danych, ale może zawierać tylko jeden typ danych np. liczbowe. Może mieć więcej wymiarów niż dwa. Tworzymy funkcją `matrix`, zmiana istniejących danych na matrycę - `as.matrix`.

Indeksowanie z matrycach działa tak samo jak w ramkach danych - `[wiersz, kolumna]`.

```
matrix(0, 4, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
```

```
matrix(1:15, 3, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

```
(x <- matrix(1:9, 3,3))
```

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

```
x[1,3]
```

```
## [1] 7
```

3.3.4 Lista (list)

Lista to taki rozbudowany wektor ;) i najbardziej elastyczny typ danych w R. Każdy element listy może być innego rodzaju, mieć inną długość, można stworzyć listę, której elementami będzie np. ramka danych, wektor, wykres i nawet inna lista.

Listę tworzymy funkcją `list`, podobnie jak w wektorze każdy element może mieć swoją nazwę. Wiele funkcji jako swój wynik zwraca listę, więc często przydaje się wiedza jak dostać się do poszczególnych elementów.

Funkcją `str` możemy wyświetlić podsumowanie wszystkich elementów listy.

Do poszczególnych części można dostać się przez nazwy albo indeksowanie [] albo [[]]. Przy zastosowaniu nawiasów [] uzyskany element nadal będzie częścią listy.

```
lista <- list( ramka = data.frame(a=rnorm(10), b="test"),
               wektor = seq(1,16,by=3),
               tekst = "To jest lista" )

lista
```

```
## $ramka
##             a    b
## 1  -2.0531244 test
## 2   0.7015323 test
## 3   0.3268519 test
## 4   1.0487510 test
## 5   0.3900792 test
## 6   0.7802174 test
## 7  -1.1907613 test
## 8  -0.3541666 test
```

```
## 9   0.1383231 test
## 10  0.6498310 test
##
## $wektor
## [1]  1  4  7 10 13 16
##
## $tekst
## [1] "To jest lista"
```

```
str(lista)
```

```
## List of 3
##  $ ramka :'data.frame':  10 obs. of  2 variables:
##   ..$ a: num [1:10] -2.053 0.702 0.327 1.049 0.39 ...
##   ..$ b: chr [1:10] "test" "test" "test" "test" ...
##  $ wektor: num [1:6] 1 4 7 10 13 16
##  $ tekst : chr "To jest lista"
```

```
lista$tekst
```

```
## [1] "To jest lista"
```

```
lista[3]
```

```
## $tekst
## [1] "To jest lista"
```

```
lista[[3]]
```

```
## [1] "To jest lista"
```

```
# pierwszy element wektora, będącego drugim elementem listy
lista[[2]][1]
```

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

3.3.5 Typ liczbowy (numeric), znakowy (character), logiczny (logical)

- Typ liczbowy przechowuje liczby całkowite i rzeczywiste. Kropką dziesiętną jest kropka :). Na liczbach można łatwo prowadzić podstawowe działania matematyczne - dodawanie, odejmowanie, mnożenie itp. Takie same operacje możemy prowadzić na wektorach.

```
2*4
```

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

```
A <- c(2,4,6)
B <- c(1,2,3)

A*B
```

```
## [1]  2  8 18
```

- Typ znakowy zawiera napisy umieszczone pomiędzy "" albo ''. Napisy można wyświetleć np. funkcją `cat`, możemy wymusić pisanie kolejnego elementu w nowej linii poprzez `"\n"`. Sklejanie np. tekstu i liczb można wykonać funkcją `paste`.

Do pracy z typem znakowym można wykorzystać pakiet stringr.

```
" To jest napis "
```

```
## [1] " To jest napis "
```

```
cat("coś", "tam")
```

```
## coś tam
```

```
cat(" coś","\n","tam")
```

```
##  coś
##  tam
```

```
a <- 1

cat("Wynik to", a)
```

```
## Wynik to 1
```

```
paste("Wynik równa się", a)
```

```
## [1] "Wynik równa się 1"
```

- Typ logiczny przechowuje tylko wartości: TRUE (T) i FALSE (F) oraz NA (brak danych). Nazwy TRUE i FALSE są w R zastrzeżone, ale T i F już nie. Dlatego lepiej używać pełnych nazw, bo może się zdarzyć, że do T albo F zostanie przypisana jakaś zmienna.

```
wektor <- c(TRUE, FALSE, TRUE)
wektor
```

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

```
summary(wektor)
```

```
##    Mode   FALSE    TRUE
## logical       1       2
```

3.3.6 Typ czynnikowy (factor)

Służy do przechowywania wartości występujących w kilku kategoriach np. płeć, wykształcenie itp. Podczas wczytywania danych R z wykorzystaniem podstawowych funckji, ale nie z pakietu readr, automatycznie zamieni kolumny zawierające tekst na typ czynnikowy, chyba że zaznaczymy opcję `stringAsFactors=FALSE`.

Typ czynnikowy jest przydatny podczas robienia wykresów, gdy chcemy pogrupować dane pod względem jakiejś kategorii. Również legendy (kolejność elementów) są tworzone w oparciu o poziomy czynnika. Może się zdarzyć, że kolejność wybrana przez R (często alfabetyczna) nie będzie odpowiednia. Można łatwo przestawić poziomy korzystając z funkcji `factor`. W tej samej funkcji argument labels pozwala na zmianę nazw kolejnych poziomów.

Jeżeli chcemy zmienić kolejność elementów na wykresie np. boxplotów, słupków najlepiej zrobić to zmieniając poziomy faktora w danych.

Poziomy factor można wyświetlić przy pomocy funkcji `levels`.

Do pracy z typem czynnikowym można wykorzystać pakiet forcats będący częścią tidyverse.

```
faktor <- factor(c("A", "B", "C", "A", "A","C"))

faktor
```

```
## [1] A B C A A C
## Levels: A B C
```

```
str(faktor)
```

```
##  Factor w/ 3 levels "A","B","C": 1 2 3 1 1 3
```

```
levels(faktor)
```

```
## [1] "A" "B" "C"
```

```
summary(faktor)
```

```
## A B C
## 3 1 2
```

```
# Zmiana poziomów

faktor2 <- factor(faktor, levels=c("B", "C", "A"), labels = c("BB", "CC", "AA"))
faktor2
```

```
## [1] AA BB CC AA AA CC
## Levels: BB CC AA
```

3.3.7 Funkcje

Większość operacji na danych w R wykonujemy za pomocą funkcji. Pakiety są to właściwie zbiory funkcji, często dotyczących jednego konkretnego zagadnienia.

Każda funkcja zawiera przynajmniej jeden argument, który jest niezbędny do jej działania. Nazwy wszystkich argumentóœ możemy sprawdzić korzystając z pomocy. Często nie jest konieczne podanie wartości wszystkich argumentów, gdyż mogą mieć ustawione wartości domyślne.

Jeżeli podajemy wartości argumentóœ możemy je wpisywać do funkcji kolejno (tak jak są wypisane w pomocy) albo korzystając z nazw. Jeżeli nie pamiętamy wszystkich nazw można sobie pomóc przy pomocy klawisza Tab :) Kolejne podawane argumenty należy rozdzielać przecinkami np. `funkcja(a=1, b=2, c="model")`. Do argumentu możemy też podać wektor wartości korzystając z `c` np. `funkcja(a=c(1,2,5))`

Np. funkcja licząca średnią to `mean`. Zawiera trzy argumenty:

- x - oznacza obiekt, z którego ma być policzona średnia
- trim (domyślnie 0) oznacza część obserwacji, która ma zostać przycięta z każdej strony x

- na.rm (domyślnie FALSE) - czy mają być brane pod uwagę wartości NA.

```
?mean
```

```
mean {base}  R Documentation
```

```
Arithmetic Mean
```

```
Description
```

```
Generic function for the (trimmed) arithmetic mean.
```

```
Usage
```

```
mean(x, ...)
```

```
## Default S3 method: mean(x, trim = 0, na.rm = FALSE, ...)
```

```
wek <- c(1,5,9,2,7,3,5,9,2,4)

mean(wek)
```

```
## [1] 4.7
```

```
mean(x=wek)
```

```
## [1] 4.7
```

```
mean(x=wek, trim=0.2, na.rm=TRUE)
```

```
## [1] 4.333333
```

W R możliwe jest pisanie własnych funkcji. Nie jest to trudne, a pozwala na uniknięcie wielokrotnego powtarzanie tego samego kodu. Dużo łatwiej jesy zmienić/poprawić jedną funkcję niż dwadzieścia razy wklejone te same 10 linijek kodu :)

Poniżej krótki przykład tworzenia funkcji. Więcej informacji można znaleźć np. w książce R for Data Science.

Załóżmy że chemy napisać funckję która wczyta dane z pliku txt i doda do tego pliki nową kolumnę będącą sumą dwóch pierwszych.

Kod potrzebny żeby to wykonać:

```
data <- read.table('data/test_funkcja.txt')

data$nowa_kolumna <- data$x + data$y
```

A teraz funckja która wykonuje te same czynności. Nazwa funkcji jest dowolna, chociaż lepiej jest nie nadpisywać już istniejących funkcji, a jej nazwa powinna opisywać to co funkcja będzie robić. W nawiasach () należy podać nazwy argumentów z ewentualnymi wartościami domyślnymi, a w nawiadach {} kod który ma zostać wykonany. Na końcu funkcji warto użyć return() - determinuje jaki wynik zwróci funkcja.

```
dodaj_kolumne <- function(file){

  data <- read.table(file)

  data$nowa_kolumna <- data$x + data$y

  return(data)
}
```

Każdą funkcję po napisaniu należy sprawdzić

```
dodaj_kolumne('data/test_funkcja.txt')
```

```
##     x  y nowa_kolumna
## 1   1 11           12
## 2   2 12           14
## 3   3 13           16
## 4   4 14           18
## 5   5 15           20
## 6   6 16           22
## 7   7 17           24
## 8   8 18           26
## 9   9 19           28
## 10 10 20           30
```