Wprowadzenie do R

Alicja Szabelska[1] Joanna Zyprych-Walczak[1]

[1]Katedra Metod Matematycznych i Statystycznych
Uniwersytet Przyrodniczy w Poznaniu

PAZUR - warsztaty, październik 2014

Rozkład jazdy..

1. Wprowadzenie do R i RStudio
2. Obiekty w R
   - Podstawowe operacje na obiektach
   - I jeszcze trochę o obiektach...
3. Wczytywanie danych
4. Pierwsze kroki do ...
   - funkcja 'if', 'ifelse'
   - funkcja 'for'
   - funkcja 'while'
   - Rodzina funcji apply
5. Jak to zobaczyć?
   - Funkcje plot, lines, curve
   - Dodatkowe elementy związane z tworzeniem wykresów
   - Histogramy
   - Boxplot
   - Wykresy słupkowe

Przejdźmy do..

Parę słów o R

- Inspiracja - jezyk programowania S
- Roberta Gentlemana i Rossa Ihake (Wydział Statystyki Uniwersytetu w Auckland)
- Projekt GNU - powszechna dostępność dla wszystkich użytkowników
- Dziedziny wykorzystania: liniowe i nieliniowe programowanie, statystyczne testowanie, analiza czasowa, analiza dyskryminacyjna, metody numeryczne, algebra, analiza eksperymentu...
- 8 podstawowych pakietów zawartych podczas instalacji, 5900 pakietów dodatkowych
- Zalety: Powszechność, darmowość i ogromna funkcjonalność

Parę słów o RStudio

- Darmowość i ogólnodostępność na wszystkich systamach
- Łatwe zarzadzanie wieloma plikami za pomocą projektów
- Szybkie instalowanie pakietów
- Łatwe edytowanie składni (podświetlanie tekstu, uzupełnianie istniejących nazw funkcji, zmiennych i słów kluczowych)
- Wsparcie dla pakietów: `knitr`, `Sweave`, `shiny`
- Podgląd danych oraz wykresów w osobnych oknach

Instalacja

- Instalacja R - http://www.r-project.org
- Instalacja RStudio - http://rstudio.org/download/desktop

Pakiety i dane w nich zawarte

- Pakiety R są zbiorem funkcji i danych.
- Pełna lista pakietów: http://cran.r-project.org/web/packages
- Instalacja pakietów - *install.packages()*, ładowanie pakietów do pamięci - *library()*
- Wywoływanie danych zawartych w pakiecie: *data('nazwa danych')*

Pierwsze uruchomienie

Szukanie pomocy

?sum, help(sum)
apropos(sum)
example(matrix)
??sum, help.search('sum')

Przydatne skróty klawiszowe w RStudio

uzupełnianie nazw funkcji i obiektów -'tab'
wyświetlanie kodu funkcji - klawisz F2
wyświetlanie pomocy na temat funkcji - klawisz F1
zamknięcie programu - ctrl+q
zamknięcie skryptu - ctrl+w

Pierwsze uruchomienie - ćwiczenia

- Zainstaluj pakiet ‘aplpack’, dowiedz się czegoś o funkcji ‘faces’ i narysuj bazując na tych informacjach zestaw twarzy z tematyką świąt Bożego Narodzenia.
- Znajdź, jakie funkcje istnieją w R, które powiązane są ze słowem kluczowym ‘ test’
- Zainstaluj pakiet ’effects’. Sprawdź czego dotyczą dane TitanicSurvival.

Podstawy składni R

- Znak przypisania: <- lub =
- Nazwy obiektów mogą zawierać: duże i małe litery (UWAGA: wielkość znaków jest rozróżnialna),
  liczby (UWAGA: nazwy nie mogą się zaczynac od liczby),
  znaki interpunkcyjne . oraz _ (UWAGA: nazwy nie mogą się zaczynac od _)
- Korzystając z funkcji jej argumenty są umieszczane w okrągłych nawiasach () i rozdzielane są za pomocą przecinka

Przejdźmy do..

Obiekty

W R dostępne są następujące typy obiektów:

- wektor (funkcja *c()*)
- macierz (funkcja *matrix()*)
- ramka danych (funkcja *data.frame()*)
- lista (funkcja *list()*)

Obiekty mogą być następujących klas:

- znakowa (funkcja *character()*)
- logiczna (funkcja *logical()*)
- numeryczna (funkcja *numeric()*)
- czynnikowa (funkcja *factor()*)
- pusta (funkcja *NULL*)

Podstawowe operacje arytmetyczne

Funkcja w R	Opis	Przykład użycia
$+, -$	dodawanie, odejmowanie	$1 + 2$,$3 - 2$
$*, /$	mnożenie, dzielenie	$5 * 4$,$8/4$
sqrt(4),^	pierwiastkowanie, potęgowanie	sqrt(4), 3^ 5
%%	reszta z dzielenia	10 %% 3
log, log2, exp	logarytm, eksponenta	log(2),log2(5), exp(1)
round	zaokrąglenie liczby	round(3.8), round(pi)
abs	wartość bezwzględna	abs(-34)

Operatory logiczne

Operator	Opis	Przykład użycia
TRUE(T), FALSE(F)	prawda, fałsz	T, F
!	zaprzeczenie	!T, !FALSE
==	równość	5==3
!=	nierówność	"Kasia"!="Krysia"
&	oraz	TRUE & FALSE
|	lub	F|T
%in%	czy znajduje się w wektorze	c(3,5) %in% 1:4
>, <,	większe, mniejsze	4<7
>=, <=	większe lub równe, mniejsze lub równe	3>=2
&&	sprawdza wszystkie elementy	1:7>=4 && 1:7<6
||	sprawdza wszystkie elementy	1:7>=4 || 1:7<6
all	sprawdza czy wszystkie elementy spełniają warunek	all(1:7>=4)
any	sprawdza czy którykolwiek z elementów spełnia warunek	any(1:7>=4)
is.<obiekt/typ>	sprawdza czy obiekt jest danej klasy	is.vector(1:4)
which	zwraca indeksy obiektu spełniające warunek	which(1:10>4)

Wektory

Przykłady wektorów:

```
A <- 1:4 # wektor numeryczny zawierający wartości 1, 2, 3, 4
B <- c(1, 2, 5.3, 6, -2, 4) # wektor numeryczny
C <- c("jeden", "dwa", "trzy") # wektor znakowy
D <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE) # wektor logiczny
```

Łączenie dwóch wektorów:

```
x <- c(2,3,5,2,7,1)
y <- c(10,15,12)
z <- c(x, y)
z
```

```
## [1]  2  3  5  2  7  1 10 15 12
```

Wektory - c.d.

Alternatywne metody tworzenia wektorów:

```
rep(c(1,2), times = 3) # Jeśli chcemy powielić jakiś wektor parę razy
```

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

```
rep(c(1,2), each = 3)  # Jeśli chcemy powielić elementy wektora parę razy
```

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

```
seq(from = 1 , to= 10, by =2 ) # Tworzy sekwencję liczb (od 1 do 10 co 2)
```

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

Odwoływanie się do elementów wektora

```
x[1:3]# Odwoładnie się do elementów wektora x stojących na pozycjach 1:3

## [1] 2 3 5

x[c(2,4)] # Wyciąganie podzbioru 2 i 4 z wektora x

## [1] 3 2

x[-c(2,3)] # Podzbiór wektora x bez elementu 2 i 3

## [1] 2 2 7 1

a<-x>5 # Generuje wektor logiczny o wartościach (T or F)
b<-x[x>6] # Podzbiór wektora x - wyciągane są tylko te wartości wektora x
# które są większe od 6 (możliwe relacje dla wyłączania podzbiorów <,
# <=, >, >=, ==, i !=).
```

Wektory - użyteczne funkcje

- length() - zwraca liczbę elementów wektora
- sum() - wyznacza sumę wszystkich elementów wektora
- prod()- wyznacza iloczyn wszystkich elementów wektora
- cumsum(), cumprod() - wyznacza sumę i iloczyn skumulowany
- sort() - sortuje wektor
- order() - podaje indeksy posortowanych elementów
- diff() - wyznacza różnice pomiędzy elementami oddalonymi od siebie o wartość parametru lag
- print() - wyświetla obiekty
- cat() - wyświetla obiekty łącząc je w jeden napis
- paste() - tworzy łańcuch tekstowy z podanych argumentów

Przykłady użycia funkcji:

```
s <- c(1,1,3,4,7,11)
length(s)

## [1] 6

sum(s) # 1+1+3+4+7+11

## [1] 27

prod(s) # 1*1*3*4*7*11

## [1] 924

cumsum(s)

## [1]  1  2  5  9 16 27

diff(s) # 1-1, 3-1, 4-3, 7-4, 11-7

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

Przykłady użycia funkcji:

```
zm <- 'zmienna'
paste(zm, 1:6, sep='_')

## [1] "zmienna_1" "zmienna_2" "zmienna_3" "zmienna_4"
## [5] "zmienna_5" "zmienna_6"

paste(zm,1:3)

## [1] "zmienna 1" "zmienna 2" "zmienna 3"

iter<-1:6
cat("iteration = ", iter <- iter + 1, "\n")

## iteration =  2 3 4 5 6 7

x<-'I love R'
print(x)

## [1] "I love R"
```

Wektory - ćwiczenia

- Wprowadź dowolne wektory x,y,z. Wykonaj następujące operacje na danych x,y,z:
  - y-z, x+y, x/2, ln(x) – cos y
- Stwórz dane, które będą zawierały 8 jedynek i zapisz je pod zmienną cc, a następnie utwórz dane zawierające 199 zer i zapisz pod zmienną d.
- Oblicz: $10^2 + 11^2 + ... + 20^2$,
  $\sqrt{(log(1))} + \sqrt{(log(10))} + ... + \sqrt{(log(100))}$
- Użyj funkcji rep żeby utworzyć następujące dane
  - 1 1 1 1 1 1 1 1
  - 1 2 1 2 1 2 1 2 1 2 1 2
  - 0 0 0 0 0 0 0 0 1 1 1 1
  - 4 3 3 2 2 2 1 1 1 1
  - 23 23 23 32 42 42
  - „A" „B" „A" „B" „A" „B"

Macierze

Przykłady macierzy:

```
macierz1 <- matrix(1:6,3,2)
macierz1

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

macierz2 <- matrix(1:6,3,2,byrow=T)
macierz2

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

Macierze

Przykłady macierzy:

```
macierz3 <- matrix(c(1, 2, 5.3, 6, -2, 4), 2, 3)
macierz3

##      [,1] [,2] [,3]
## [1,]    1  5.3   -2
## [2,]    2  6.0    4

macierz4<-matrix(c(1, 2, 5.3, 6, -2, 4), 3, 2)
macierz4

##      [,1] [,2]
## [1,]  1.0    6
## [2,]  2.0   -2
## [3,]  5.3    4
```

Macierze

Łączenie dwóch macierzy:

```
cbind(1:4,5:8)
```

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

```
rbind(1:4,5:8)
```

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

Odwoływanie się do elementów macierzy

```
a<-matrix(1:9,3,3)
a[,1] # pierwsza kolumna

## [1] 1 2 3

a[1,] # pierwszy wiersz

## [1] 1 4 7

a[1,1] # pierwszy element (i=1,j=1)

## [1] 1

diag(a) ## digonalna

## [1] 1 5 9
```

Macierze - użyteczne funkcje

- dim() - zwraca wymiar macierzy
- %*% - wyznacza iloczyn macierzy
- t()- wyznacza transpozycję macierzy
- det() - wyznacza determinantę macierzy
- solve() - wyznacza macierz odwrotną
- ncol(), nrow() - zwraca liczbę kolumn, wierszy macierzy
- colnames(), rownames() - zwraca nazwy kolumn, wierszy
- colSums(), rowSums() - zwraca sumę kolumn, wierszy

Przykłady użycia funkcji:

```
C<-matrix(1:4,2,2)
dim(C)

## [1] 2 2

colnames(C)<-c('C1','C2')
rownames(C)<-c('R1','R2')
colSums(C)

## C1 C2
##  3  7

rowSums(C)

## R1 R2
##  4  6
```

Macierze - ćwiczenia

1. Zadeklaruj poniższe macierze:

$$A = \begin{pmatrix} 1 & 2 & -1 & 0 \\ 3 & -2 & 4 & 5 \\ 2 & 6 & 5 & -3 \\ 0 & 1 & 5 & -4 \end{pmatrix}$$

$$B = \begin{pmatrix} 3 & 6 \\ 4 & 0 \\ 2 & -1 \\ 1 & 1 \end{pmatrix}$$

Oblicz wyznacznik macierzy **A**, znajdź iloczyn **AB**, znajdź macierz transponowaną $\mathbf{B}^T$.

Macierze - ćwiczenia

2. Dana jest macierz $\mathbf{A}$:

$$A = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix}$$

Obliczyć $\mathbf{A}^2 - 6\mathbf{A} + 4\mathbf{I}$, gdzie $\mathbf{I}$ jest macierzą jednostkową postaci:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

3. Mając:

$$A = \begin{pmatrix} 1 & -1 & 5 \\ 2 & 1 & -4 \end{pmatrix},$$

$$B = \begin{pmatrix} -2 & -1 & 2 \\ 3 & 1 & 4 \end{pmatrix},$$

$$C = \begin{pmatrix} 0 & -3 & 9 \\ 3 & 18 & -6 \end{pmatrix}$$

Oblicz:

* $\mathbf{A} + \mathbf{B}$, * $\mathbf{A} - \mathbf{B}$, * $2\mathbf{A} + 3\mathbf{B}$, * $\mathbf{A}^T - 4\mathbf{B}^T$,

Macierze - ćwiczenia

4. Zadeklaruj macierz A:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Następnie, korzystając z funkcji R:
* Wyznacz liczbę wierszy i kolumn macierzy **A**.
* Wyznacz sumę wszystkich elementów macierzy **A**.
* Wyznacz sumy wszytkich elementów w poszczególnych kolumnach macierzy **A**.
* Wyznacz sumy wszytkich elementów w poszczególnych wierszach macierzy **A**.
* Oblicz $A_{11} + A_{32}$.
* Wyświetl zawartość drugiej kolumny.
* Wyświetl zawartość pierwszego wiersza.

Przykłady ramek danych

```
data.frame(imie=c('Asia','Kasia','Basia'), wiek=c(24,30,15),
           oczy=factor(c('piwne','szare','piwne')))

##    imie wiek  oczy
## 1  Asia   24 piwne
## 2 Kasia   30 szare
## 3 Basia   15 piwne

data(iris) # wczytanie danych dostępnych w R
head(iris) # wyświetlanie kilku pierwszych wartości

##   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
```

Odwoływanie się do elementów ramek danych

```
iris[1:10,1] # pierwsze dziesięć elementów z pierwszej kolumny

##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9

iris[1:10,'Sepal.Length'] # lub równoznacznie

##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9

iris$Sepal.Length[1:10] # lub równoznacznie

##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9

iris$'Sepal.Length'[1:10] # lub...

##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

Ramki danych - użyteczne funkcje

- head() - wyświetla pierwsze rekordy zbioru
- attach() - pozwala odnosić się do nazw zmiennych znajdujących się bezpośrednio w danych
- detach() - likwiduje możliwość bezpośredniego odnoszenia się do zmiennych
- attributes(), str() - pozwala wyświetlić informacje o obiekcie
- complete.cases() - sprawdza ile wierszy w zbiorze zawiera wszystkie informacje
- duplicated() - sprawdza ile wierszy jest powtórzonych
- summary() - wyznacza podstawowe statystyki dla obiektu
- table() - wyznacza tabelę z liczbą wsytąpień danego czynnika
- subset() - określa podzbiór danego zbioru, spełniający określone warunki

Przykłady użycia funkcji:

```
data(trees)
head(trees)

##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7

Girth

## Error:  nie znaleziono obiektu 'Girth'

attach(trees)
Girth

##  [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3
## [12] 11.4 11.4 11.7 12.0 12.9 12.9 13.3 13.7 13.8 14.0 14.2
## [23] 14.5 16.0 16.3 17.3 17.5 17.9 18.0 18.0 20.6
```

Przykłady użycia funkcji - c.d.

```
attributes(trees)

## $names
## [1] "Girth"  "Height" "Volume"
##
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
## [19] 19 20 21 22 23 24 25 26 27 28 29 30 31
##
## $class
## [1] "data.frame"

str(trees)

## 'data.frame': 31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

Przykłady użycia funkcji - c.d.

```
complete.cases(trees)
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [12] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [23] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
summary(trees)
```

```
##      Girth           Height       Volume
##  Min.   : 8.3   Min.   :63   Min.   :10.2
##  1st Qu.:11.1   1st Qu.:72   1st Qu.:19.4
##  Median :12.9   Median :76   Median :24.2
##  Mean   :13.2   Mean   :76   Mean   :30.2
##  3rd Qu.:15.2   3rd Qu.:80   3rd Qu.:37.3
##  Max.   :20.6   Max.   :87   Max.   :77.0
```

Przykłady użycia funkcji - c.d.

```
ankieta<-data.frame(odpowiedzi=c('T','N','T','T','N','X','N','X','T'),
                    wiek=c(16,23,22,65,45,32,24,12,56))
table(ankieta$odpowiedzi)

##
## N T X
## 3 4 2

subset(ankieta,wiek>20)

##   odpowiedzi wiek
## 2          N   23
## 3          T   22
## 4          T   65
## 5          N   45
## 6          X   32
## 7          N   24
## 9          T   56
```

Ramki danych - ćwiczenia

Załaduj dane TitanicSurvival z pakietu effects. Spróbuj odpowiedzieć na następujące pytania:

1. Ile jest osób w tym zestawie danych?
2. Ile jest w nich kobiet, a ile mężczyzn?
3. Ile kobiet przeżyło? Ilu mężczyzn przeżyło?
4. Ile kobiet z pierwszej klasy miało powyżej 30 lat?
5. Ilu dzieci poniżej 5 lat przeżyło?
6. Jaka jest proporcja (w %) pomiędzy mężczyznami, a kobietami w 3 klasie?
7. Jaka jest proporcja pomiędzy przeżytymi kobietami, a mężczyznami?

Przykłady list

```
pierwsza.lista<-list(imie=c('Asia','Kasia','Basia'), sredni.wiek=23,
                     macierz=matrix(1:4,2,2))
pierwsza.lista

## $imie
## [1] "Asia"  "Kasia" "Basia"
##
## $sredni.wiek
## [1] 23
##
## $macierz
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

str(pierwsza.lista)

## List of 3
##  $ imie       : chr [1:3] "Asia" "Kasia" "Basia"
##  $ sredni.wiek: num 23
##  $ macierz    : int [1:2, 1:2] 1 2 3 4
```

Odwoływanie się do elementów list

```
pierwsza.lista$imie # odniesienie do elementu 'imię'

## [1] "Asia"  "Kasia" "Basia"

pierwsza.lista$imie[1] # odniesienie do pierwszego elementu tej zmiennej

## [1] "Asia"

pierwsza.lista[[1]][1] # to samo, lecz z wykorzystaniem numeru kolejnego

## [1] "Asia"

pierwsza.lista$macierz[1,] # pierwszy wiersz elementu macierz

## [1] 1 3
```

Operacje na plikach i katalogach

```
sessionInfo()

## R version 3.1.1 (2014-07-10)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
##
## locale:
## [1] pl_PL.UTF-8/pl_PL.UTF-8/pl_PL.UTF-8/C/pl_PL.UTF-8/pl_PL.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## other attached packages:
## [1] knitr_1.6
##
## loaded via a namespace (and not attached):
## [1] evaluate_0.5.5 formatR_1.0    highr_0.3
## [4] stringr_0.6.2  tools_3.1.1
```

Operacje na plikach i katalogach

```
# zwraca ścieżkę ustawioną jako domyślną (warto z tego korzystać ponieważ
# nie trzeba wtedy wpisywać za każdym razem ścieżki. Natomiast jeżeli
# działamy w RStudio i tworzymy projekty domyślna ścieżka jest tożsama
# z tą opisaną w pakiecie)
getwd()
# jeżeli chcemy zmienić ścieżkę dostępu używamy funkcji setwd()
setwd('/Users/')
# funkcja list files zwraca nam informację (a dokładnie wektor) o tym
# jakie pliki znajdują się w domyślnym folderze
list.files()
```

Operacje na plikach i katalogach

```
# usuwamy pliki z dysku za pomocą file.remove()
file.remove()
# zapisujemy obiekty znajdujace się w przestrzeni roboczej do pliku.
# UWAGA! Pliki zapisane w systemie 32 bitowym nie otworza się w
# systemie 64 bitowym!
save.image()
# funkcją save() możemy zapisać wybrany obiekt lub obiekty (a nie całą
# przestrzeń) do pliku z rozszerzeniem .RData
save()
# ładujemy pliki do przestrzeni roboczej za pomocą load()
load()
# kończymy działanie R (z poziomu konsoli) za pomocą q()
```

Przejdźmy do..

Wczytywanie danych

Odczytywanie danych z plików

- txt - funkcja *read.table()*, *read.delim()*
- csv - funkcja *read.csv()*,*read.csv2()*
- xlsx, xls - funcja *xlsx()*, *xls()* (pakiet XLConnect)

Wczytywanie danych z plików tekstowych

```
A <- read.table(file = 'dane.txt', # podajemy plik wraz ze ścieżką
                header=T, # określamy czy w pliku znajduje sie nagłówek
                sep=';') # określamy co jest separatorem

## trochę bardziej rozbudowany sposób wczytania plików
A <- read.table(file = 'dane.txt', header = TRUE, sep = ';',
                dec = ',',  # określa symbol miejsca dziesiętnego
                na.strings = 'NA', # określa oznaczanie braków w pliku
                colClasses = c('numeric','numeric','character'),
                # przy dużych plikach przyspiesza wczytywanie danych
                stringsAsFactors = FALSE) # domyślnie kolumny
                # zapisywane są jako factor
```

Wczytywanie danych z Excela

```
### wczytanie plików Excela
library(XLConnect)
readWorksheetFromFile('plik.xlsx',
                      sheet='Arkusz', ### nazwa arkusza,
# można również podać wiele arkuszy lub kolejne liczby
                      header = T, ### czy w pierwszym
# wierszu znajdują się nazwy kolumn
                      startRow = 1,
                      startCol = 2)

### istnieje również drugi sposób wczytania plików
skoroszyt<-loadWorkbook('plik.xlsx') ### wczytujemy
# skoroszyt do obiektu
getSheets(skoroszyt) ### pobieramy nazwy arkuszy
dane<-readWorksheet(object = skoroszyt,  ### wczytujemy
# wybrany przez nas arkusz (lub arkusze)
                    sheet = 'Arkusz')
```

Wczytywanie danych - ćwiczenia

Załaduj dane studenci.xlsx i odpowiedz na następujące pytania:

- Ile jest kobiet, a ilu mężczyzn studiuje Leśnictwo?
- Jaka jest średnia stypendium naukowego dla kobiet, a jaka dla mężczyzn?
- Ile kobiet studiuje rolnictwo?
- Ile studentów leśnictwa nie ma stypendium naukowego?

Wczytywanie danych - ćwiczenia

Załaduj dane.txt (separatorem jest tutaj tabulator) i odpowiedz na następujące pytania:

- Ile rodzin mieszka w mieście, a ile na wsi?
- Ile duzych rodzin mieszka w mieście, a ile dużych mieszka na wsi?
- Ile rodzin jedzie na wakacje?
- Ile rodzin ze wsi, a ile z miasta jedzie na wakacje?
- Jaki jest maksymalny dochód rodzin dużych (malych) mieszkających w miescie, a jaki na wsi?

Przejdźmy do..

1. Wprowadzenie do R i RStudio
2. Obiekty w R
3. Wczytywanie danych
4. Pierwsze kroki do ...
   - funkcja 'if', 'ifelse'
   - funkcja 'for'
   - funkcja 'while'
   - Rodzina funcji apply
5. Jak to zobaczyć?

Instrukcje warunkowe - if...else...

W R można korzystać z funkcji warunkowych typu if...else... Dzięki tej funkcji można uzależnić wykonywanie fragmentu kodu od spełnienia pewnego warunku logicznego. Składnia tej funkcji ma następującą postać:

```
if (warunek) {
    polecenia
}
```

lub

```
if (warunek) {
    polecenia
} else if (warunek) {
    polecenia
} else {
    polecenia
}
```

Przykład użycia funkcji if...else

```
liczba <- sample(1:100,1)
liczba

## [1] 88

if (liczba %% 3 == 0){
  cat("ta liczba jest wielokrotnością 3\n")
} else {
  cat("ta liczba nie jest wielokrotnością 3\n")
}

## ta liczba nie jest wielokrotnością 3
```

Instrukcje warunkowe - funkcja *ifelse()*

Gdy chcemy wynonać ciąg poleceń warunkowych, możemy wykorzystać funkcję *ifelse()*. Ma ona następującą składnię:

```
ifelse(warunek, polecenie_1, polecenie_2)
# tutaj warunek może być jedną wartością logiczną lub wektorem logicznym
```

Przykład użycia tej funkcji:

```
ifelse(2:5 == 1, "hmmm... coś tu nie pasuje", "R potrafi liczyć")

## [1] "R potrafi liczyć" "R potrafi liczyć" "R potrafi liczyć"
## [4] "R potrafi liczyć"
```

Pętla for

Tak jak w każdym języku programowania w R również można tworzyć pętle. Schemat jej użycia wygląda następująco:

```
for (iterator in zbiór){
  polecenia
}
# iterator jest zmienną, która w każdym kroku pętli przyjmuje kolejne
# wartości ze obiektu zbiór
```

Pętla for ma z góry określoną liczbę iteracji.

Przykłady użycia pętli for

```
for (i in 1:5) cat(paste0("w tym kroku wartość zmiennej i to ", i, "\n"))

## w tym kroku wartość zmiennej i to 1
## w tym kroku wartość zmiennej i to 2
## w tym kroku wartość zmiennej i to 3
## w tym kroku wartość zmiennej i to 4
## w tym kroku wartość zmiennej i to 5

# jako zbiór względem którego pętla jest wykonywana może być również
# użyty wektor tekstowy:

imiona <- c("Kasia", "Asia", "Basia")
for (i in imiona) cat(paste0("w tym kroku zmienna i to ", i, "\n"))

## w tym kroku zmienna i to Kasia
## w tym kroku zmienna i to Asia
## w tym kroku zmienna i to Basia
```

Przykłady użycia pętli for - cd

W niektórych przypadkach użycie pętli może wiązać się z dużą ilością obliczeń. O ile to możliwe warto wykorzystywać obliczenia macierzowe lub inne dostępne funcje, które działają szybciej.

```
Tabliczka <- matrix(NULL, 10, 10)
for (i in 1:10) {
  for (j in 1:10) {
    Tabliczka[i,j] <- i*j
  }
}

# to samo można zrobić za pomocą mnożenia macierzy:

x <- 1:10
Tabliczka1 <- x %*% t(x)

# lub korzystając z funkcji outer():

Tabliczka2 <- outer(x, x, '*')
```

Pętla `while`

W przeciwieństwie do pętli `for` pętla `while` nie ma z góry określonej liczby powtórzeń. Jest ona wykonywana tak długo, jak spełniony jesy postawiony warunek. Schemat jej wywołania jest następujący:

```
while (warunek) {
  polecenia
}
```

Istotne jest przy tworzeniu tej pętli jest zadbanie o to, aby w którymś momencie warunek nie został spełniony i pętla mogła zostać zakończona.

Przykłady użycia pętli while

```
i <- 0
while (i < 5) {
  i <- i +1
  print(i)
}

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

set.seed(100)
while ((x <- sample(100,1)) %% 5 != 0)
  cat(x, ' ')

## 31  26  56  6  47  49  82  38
```

Rodzina funkcji apply

Do wykonywania obliczeń na różnego typu obiektach można zamiast pętli wykorzystywać również funkcje z rodziny apply. Są one często szybszą alternatywą do używania pętli i łatwym sposobem na wyznaczenie różnych charakterystyk obiektów. Funcje z tej rodziny wykonują operacje określone przez argument FUN. Główną różnicą między tymi funkcjami jest typ obiektów, na których operacje są wykonywane. Do rodziny apply należą m.in. funkcje:

- apply() - wykonywana na kolumnach lub wierszach macierzy
- lapply() - wykonywana na elementach listy
- sapply() - wykonywana na elementach wektora (może zastępować pętlę for)
- tapply()/by() - wykonywana na podzbiorach wektora (oraz macierzy lub listy - funkcja by)

Przykłady użycia funkcji apply

```
library("effects")
dane <- TitanicSurvival[,c(1:2,4)]
apply(dane,2,table)

## $survived
##
##  no yes
## 809 500
##
## $sex
##
## female   male
##    466    843
##
## $passengerClass
##
## 1st 2nd 3rd
## 323 277 709
```

Przykłady użycia funkcji by

```
by(TitanicSurvival$age, TitanicSurvival$sex, mean, na.rm = T)

## TitanicSurvival$sex: female
## [1] 28.69
## ------------------------------------------------
## TitanicSurvival$sex: male
## [1] 30.59
```

Przykłady użycia funkcji sapply

```
czy.mlodszy.niz.10 <- function(x) (x < 10)
wyn <- sapply(TitanicSurvival$age, FUN = czy.mlodszy.niz.10)
head(wyn)

## [1] FALSE  TRUE  TRUE FALSE FALSE FALSE

summary(wyn)

##    Mode   FALSE    TRUE    NA's
## logical     964      82     263
```

Tworzenie własnych funkcji

Oprócz isteniących już funkcji w środowisku R można również tworzyć własne funkcje dostosowane do naszych potrzeb. Schemat tworzenia funkcji jest następujący

```
nazwa_funkcji <- function(lista_argumentow){
  polecenia
}
```

Warto przyjąć zasadę przy tworzeniu funkcji, że im jest ona krótsza, tym lepiej - łatwiej ją modyfikować i znajdować ewentualne błedy.
Jeśli w kodzie wykorzystujemy jakieś stałe warto pomyśleć, czy nie dodać tej stałej jako argumentu funkcji, który może przyjmować różne wartości.

Przykłady własnych funkcji

```
moja.silnia <- function(n)
{
  x <- 1
  for (i in 1:n)
  {
    x=x*i
  }
  print(x)
}

moja.silnia(4)

## [1] 24
```

Przykłady własnych funkcji

```
ktory.dzisiaj <- function()
{
  cat("R mowi, ze dzis jest ")
  cat(format(Sys.time(), "%A %d %B"))
}

ktory.dzisiaj()

## R mowi, ze dzis jest środa 15 października
```

Pierwsze kroki programowania - ćwiczenia

- Stwórz wektor x, korzystając z funkcji sample(1:35). Następnie stwórz wektor y, który dla każdego elementu wektora x zwracałby wartość funkcji

$$f(x) = \begin{cases} 0 & \text{if } x \geq 6 \\ x - 4 & \text{if } 6 < x \geq 10 \\ 36 - x & \text{if } 10 < x \geq 35 \end{cases}$$

- Stwórz funkcję, która dla wektorów x i y o tej samej długości, wyznacza następujące macierze:
  - macierz A, której elementy to $a_{ij} = x_i * y_j$.
  - macierz B, której elementy to $b_{ij} = x_i / y_j$.
  - wektor d, której elementy to $d_i = x_i * y_i$.
  - main - nazwa wykresu
- Stwórz funkcję, która jako argument przyjmuje wektor liczb, a jako wynik zwraca 3 najmniejsze i 3 największe liczby. Jeśli podany wektor jest krótszy niż 3 liczby, to wyświetl napis źa krótki argument".
- Stwórz funkcję zależną od parametru n, która zwraca n liczb naturalnych, które nie zawierają liczb 2 i 3 jako dzielników.

Przejdźmy do..

1. Wprowadzenie do R i RStudio
2. Obiekty w R
3. Wczytywanie danych
4. Pierwsze kroki do ...
5. Jak to zobaczyć?
   - Funkcje plot, lines, curve
   - Dodatkowe elementy związane z tworzeniem wykresów
   - Histogramy
   - Boxplot
   - Wykresy słupkowe

Funkcja *plot()* i *lines()*

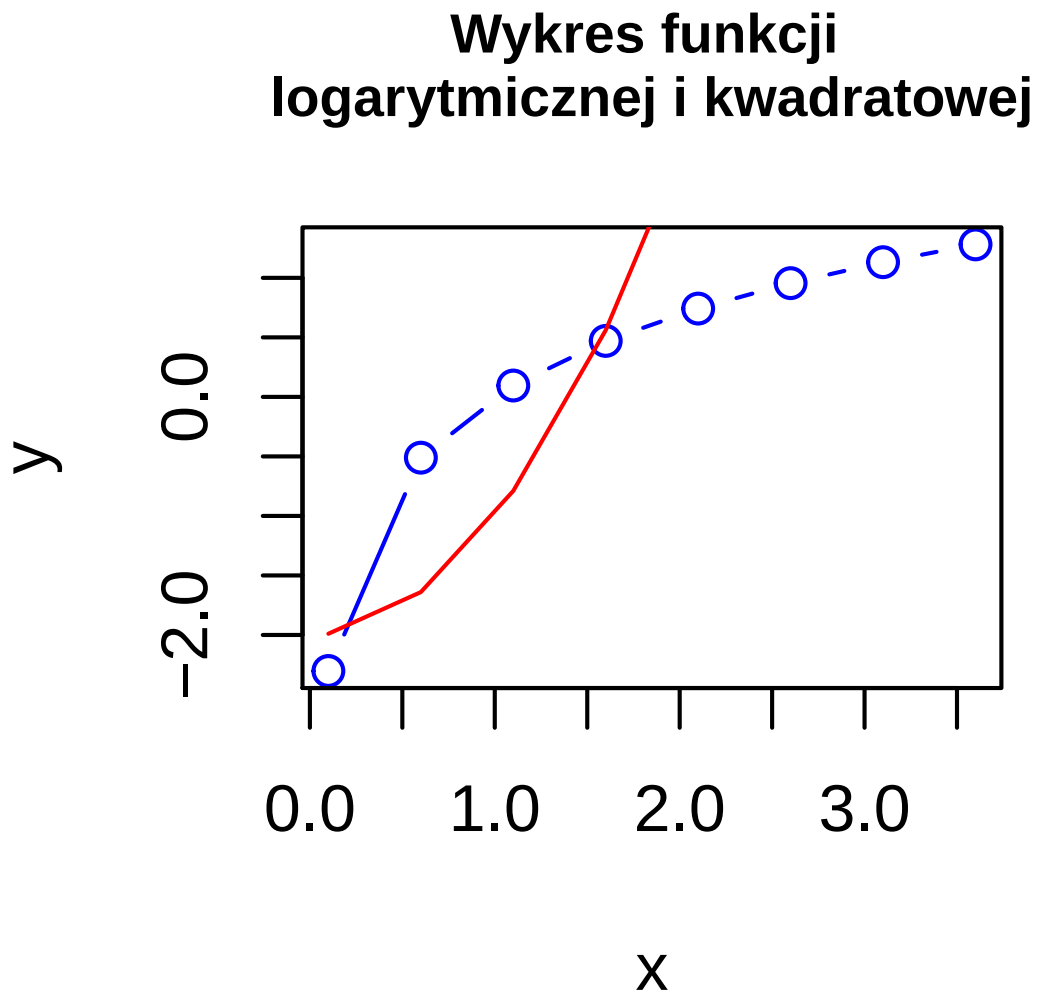
```
plot(x, y, type = "b", col = "black", main = NULL, ...)
lines(x, y = NULL, type = "b", ...)
```

Przydatne parametry funkcji *plot()* i *lines()*

- type - rodzaj linii
- col - kolor linii
- main - nazwa wykresu
- xlab/ylab - etykieta osi x/y
- cex - wielkość czcionki
- lwd - szerokość linii

```
x <- seq(.1,4,by=.5)
plot(x, log(x), type = "b", main = "Wykres funkcji logarytmicznej
     i kwadratowej", col = "blue")
lines(x, x^2-2, type = "l", col = "red")
```

Funkcja *plot()* i *lines()*



Funkcja *curve()*

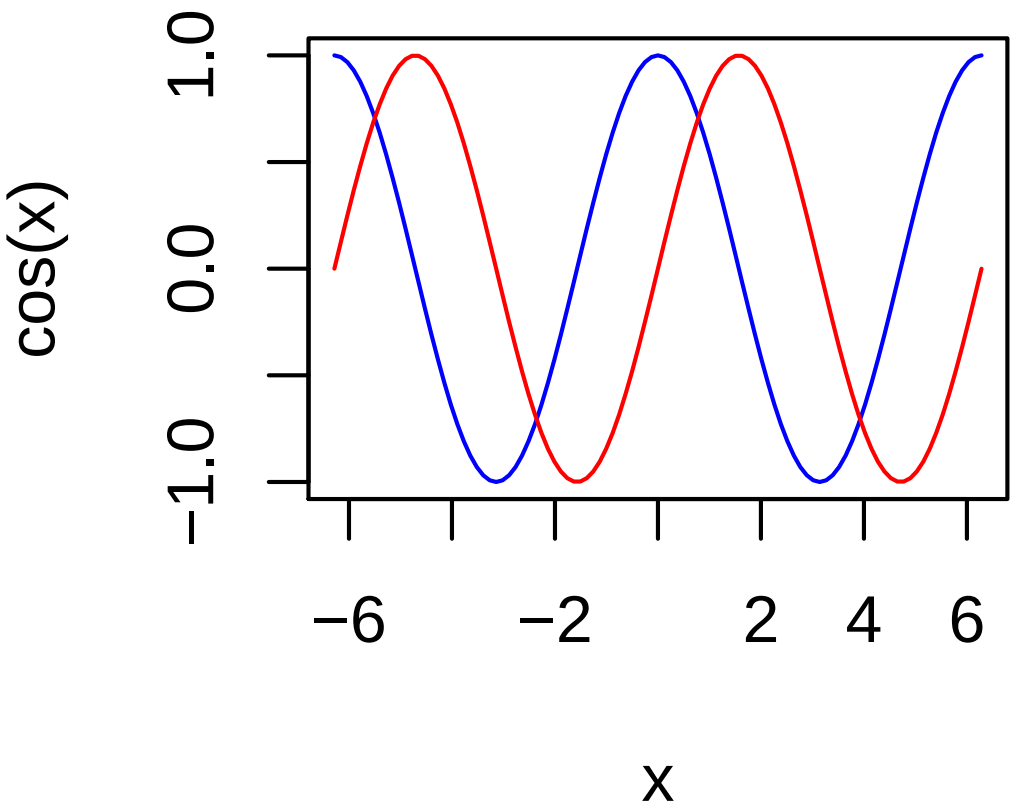
```
curve(expr, from = NULL, to = NULL, n = 101, ...)
```

Przydatne parametry funkcji *curve()*

- expr - nazwa funkcji lub wyrażenie matematyczne
- from/to - zakres przedziału, w którym funkcja jest rysowana
- n - ilość argumentów, dla których wartość funkcji jest liczona
- add - jeśli TRUE dodaje wykres do już istniejącego

```
curve(cos, from = -2*pi, to = 2*pi, type = "l", col = "blue")
curve(sin, from = -2*pi, to = 2*pi, type = "l", col = "red", add = T)
```

Funkcja *curve()*



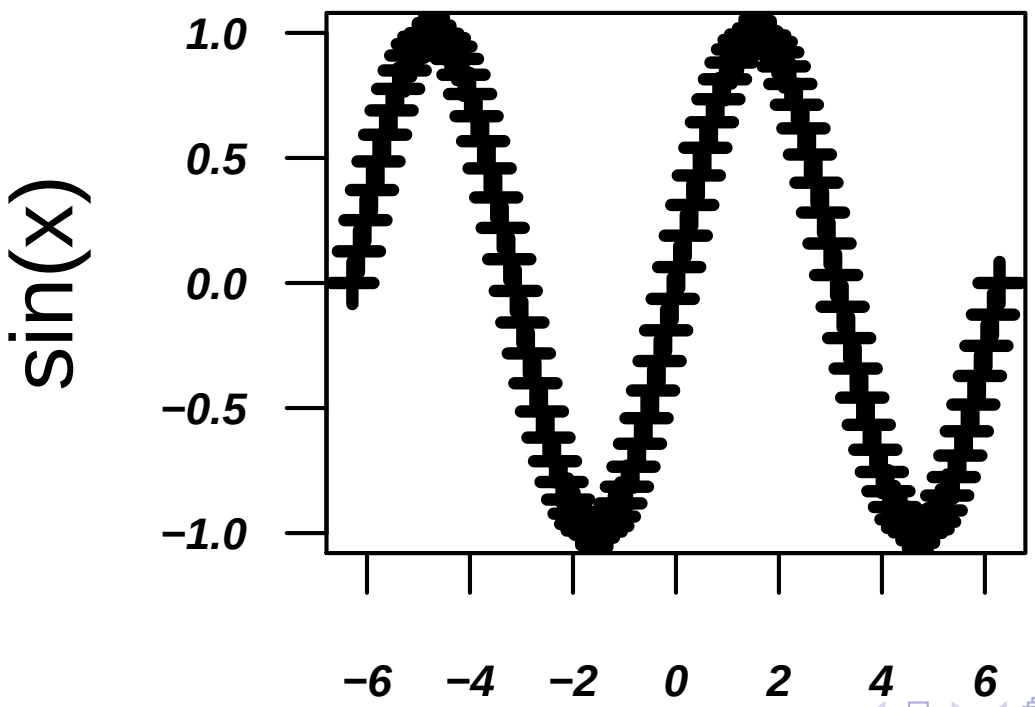
Wykres na indywidualne potrzeby

Dodatkowe argumenty funkcji graficznych

- adj - wyrównanie tekstu np. etykiet osi
- ann - określenie, czy etykiety osi mają być wpisywane
- bg/fg - kolor tła/"pierwszego planu"
- bty - kontrolowanie rysowania ramki wokół wykresu
- cex/cex.axis/cex.lab/cex.main - skala rozmiaru tekstu (domyślnie ma wartość 1, dla innych wartości jest przeskalowywany) na całym rysunku/ osiach/ etykietach/ tytule
- family/font - typ czcionki/styl czcionki
- las - kierunek tekstu na osiach
- lty/lwd - wzór linii/grubość linii
- pch - wybór symbolu, który ma być użyty do zaznaczania punktów na wykresie
- xaxt/yaxt - określa, czy oś X/Y ma być rysowana
- xlim/ylim - zakres danych umieszczony na osi X/Y

Wykres na indywidualne potrzeby

```
x<-seq(-2*pi, 2*pi, length.out = 100)
plot(x, sin(x), family = "sans", cex.lab = 1.3, cex.axis = 0.7,
     font.axis = 4, las = 1, lwd = 3, pch = 3)
```



Tytuł, opisy osi wykresu

```
title(main = NULL, sub = NULL, xlab = NULL, ylab = NULL, line = NA, ...)
```

Przydatne parametry funkcji *title()*

- main - tytuł wykresu
- sub - podtytuł wykresu
- xlab/ylab - etykiety osi x/y
- line - określa odległość od brzegu okna graficznego; liczony jest w liniach tekstu

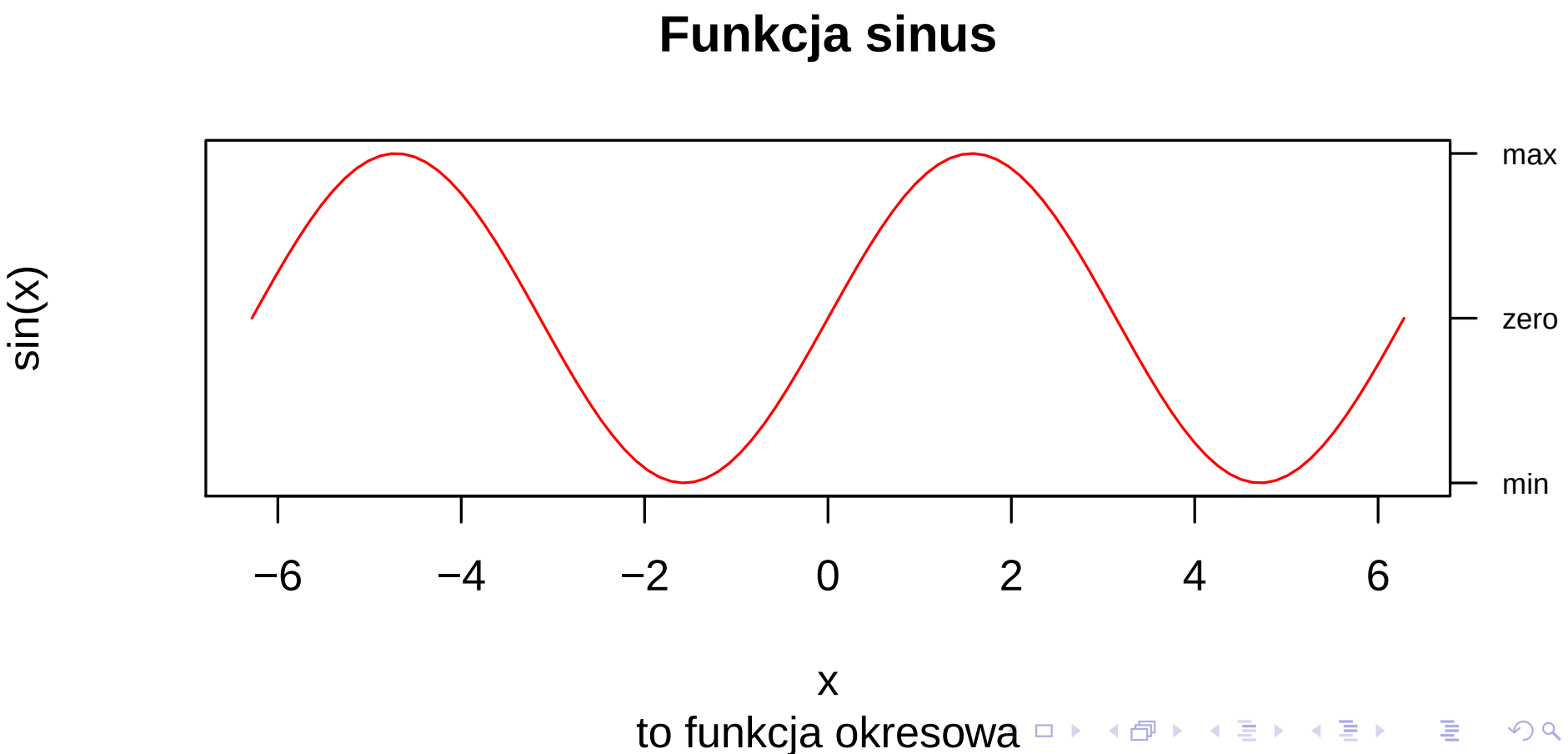
```
axis(side, at = NULL, labels = TRUE, tick = TRUE, pos = NA, ...)
```

Przydatne parametry funkcji *axis()*

- side - określa po której stronie wykresu będą rysowane osie
- at - wektor punktów, w których rysowane są znaczniki
- labels - określa czy i jakie oznaczenia mają być umieszczone przy znacznikach
- tick - określa, czy znaczniki mają czy umieszczane
- pos - określa pozycję umieszczenia osi; liczony jest we współrzędnych rysunku

Funkcja *title()* i *axis()*

```
plot(x, sin(x), type = "l", col = "red", yaxt = "n")
title(main = "Funkcja sinus", sub = "to funkcja okresowa")
axis(side = 4, at = c(-1, 0, 1), labels = c("min", "zero", "max"),
     cex.axis = 0.7, las=1)
```



Legenda wykresu

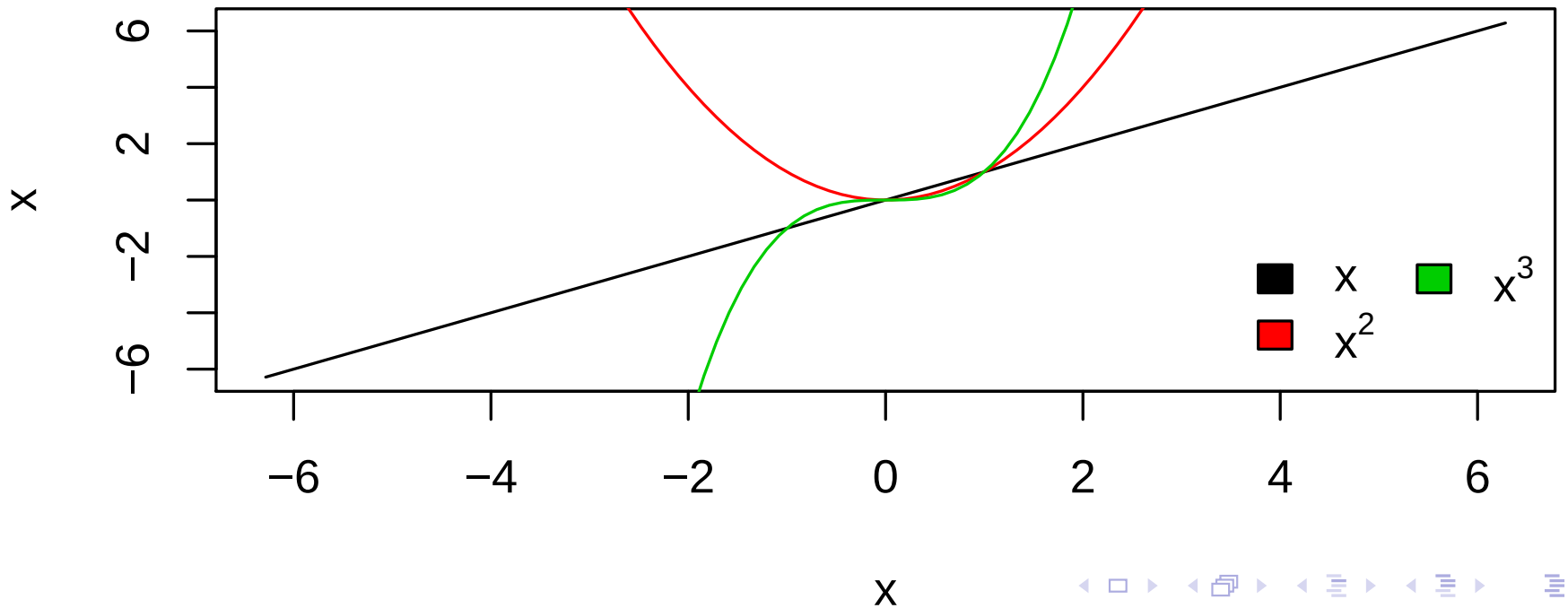
```
legend(x, y = NULL, legend, fill = NULL, border = "black", col, lty,
lwd, pch, bty = ’ o", x.intersp = 1, text.width = NULL, ncol = 1, horiz
= FALSE, title = NULL, ...)
```

Parametry funkcji *text()*

- x/y - współrzędne, gdzie legenda ma być umieszczona (może być również umieszczony jako napis np. "top")
- legend - wektor opisów, które mają być umieszczone w legendzie
- fill - określa kolor, jakim mają być rysowane kwadraty przy opisach legendy
- lwd/lty - parametry linii, ktore mają być rysowane przy kolejnych wykresach (występują też odpowiedniki tych argumentów dotuczące ogramowania całej legendy np. box.lty itp)
- x.intersp/yintersp - kontrola odległości pomiędzy pionowymi/poziomymi elementami legendy
- ncol - liczba kolumn, w których mają być opisy
- horiz - określa, czy kolejne opisy mają być umieszczanie pionowo, czy poziomo

Funkcja *legend()*

```
plot(x, x, type = "l", col = 1)
lines(x, x^2, type = "l", col = 2)
lines(x, x^3, type = "l", col = 3)
legend("bottomright", c(expression(x), expression(x^2), expression(x^3)),
       fill = 1:3, ncol = 2, bty = "n")
```



Napisy na wykresie

```
text(x, y = NULL, labels = seq_along(x), srt, ...)
```

Parametry funkcji *text()*

- x/y - współrzędne, gdzie tekst ma być umieszczony
- labels - tekst, który ma być umieszczony
- srt - kąt nachylenia tekstu do osi X

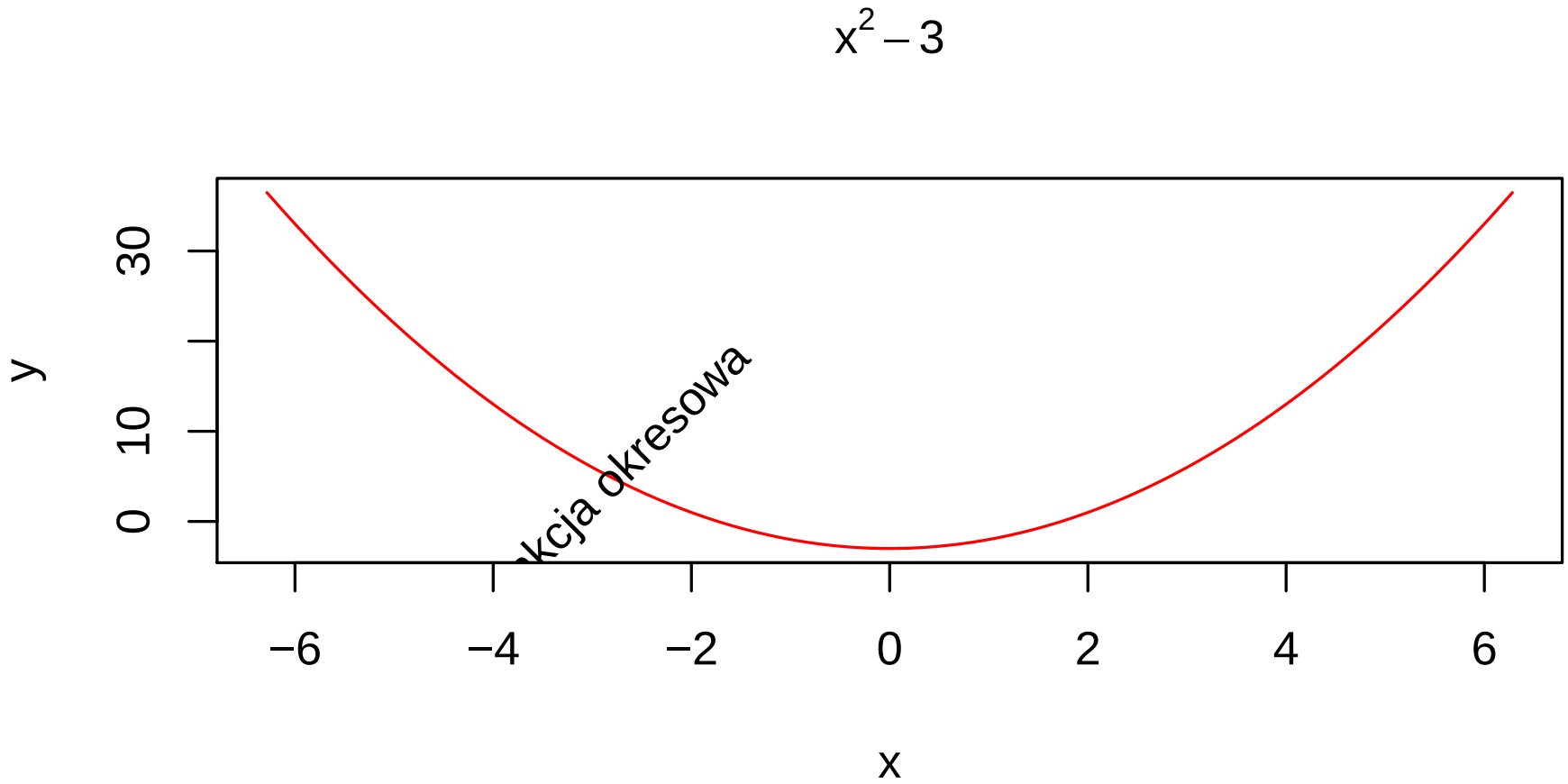
```
mtext(text, side = 3, line = 0, outer = FALSE, at = NA, ...)
```

Parametry funkcji *mtext()*

- text - tekst, który ma być umieszczony
- side - strona wykresu, po której ma być wykres umieszczony
- line - odległość (mierzona w liniach tekstu) od krawędzi wykresu, gdzie wykres ma być umieszczony
- outer - użycie zewnętrzych mrginesów

Funkcja *text()* i *mtext()*

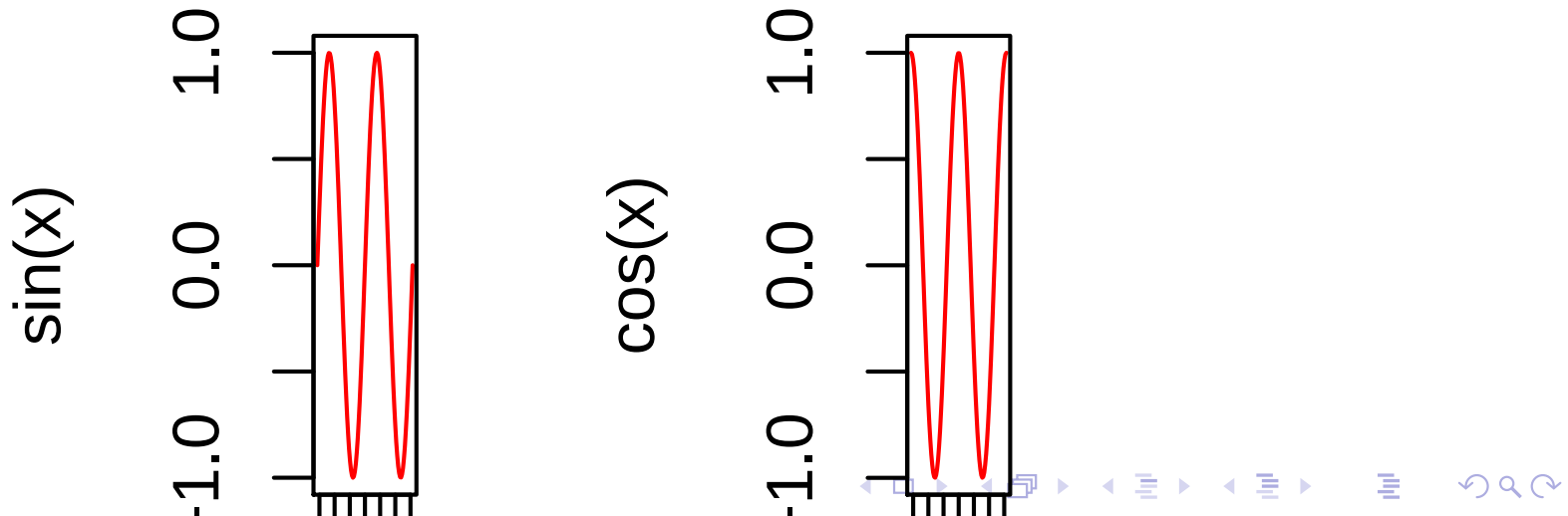
```
plot(x, x^2-3, type = "l", col = "red", ylab = "y")
mtext(expression(x^2-3), 3, line= 2)
text(-pi, 0.5, "... to funkcja okresowa", srt = 45)
```



Wiele wykresów na rysunku

Do wywołania kilku wykresów na jednym rysunku może posłużyć funkcja `par(mfrow=c(k,l))` lub `par(mfcol=c(k,l))`. Różnią się one kierunkiem umieszczania kolejnych wykresów (od lewej do prawej lub od góry do dołu).

```
par(mfrow = c(1, 2))
plot(x, sin(x), type = "l", col = "red")
plot(x, cos(x), type = "l", col = "red")
```



Funkcja *hist()*

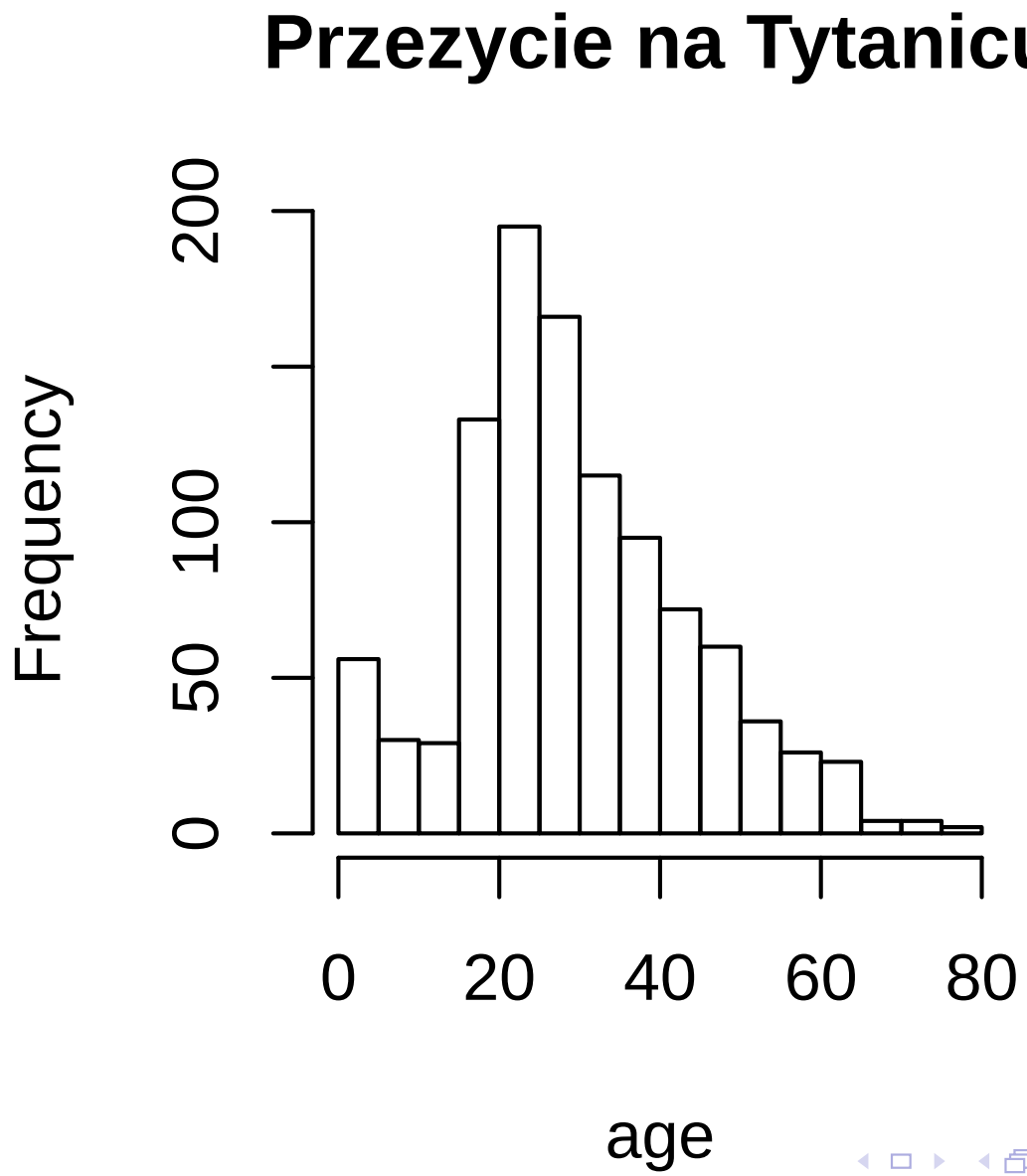
```
hist(x, breaks = 'Sturges', freq = NULL, probability =
!freq, plot = TRUE, labels = FALSE, ...)
```

Przydatne parametry funkcji *hist()*

- breaks - podział zakresu zmienności
- freq - wybór, czy wykres ma wykorzystywać frakcje, czy liczebności
- plot - czy wykres ma być rysowany, czy tylko dane do wykresu mają być wyznaczone
- labels - wektor nazw, które mają odpowiadać kolejnym podziałom

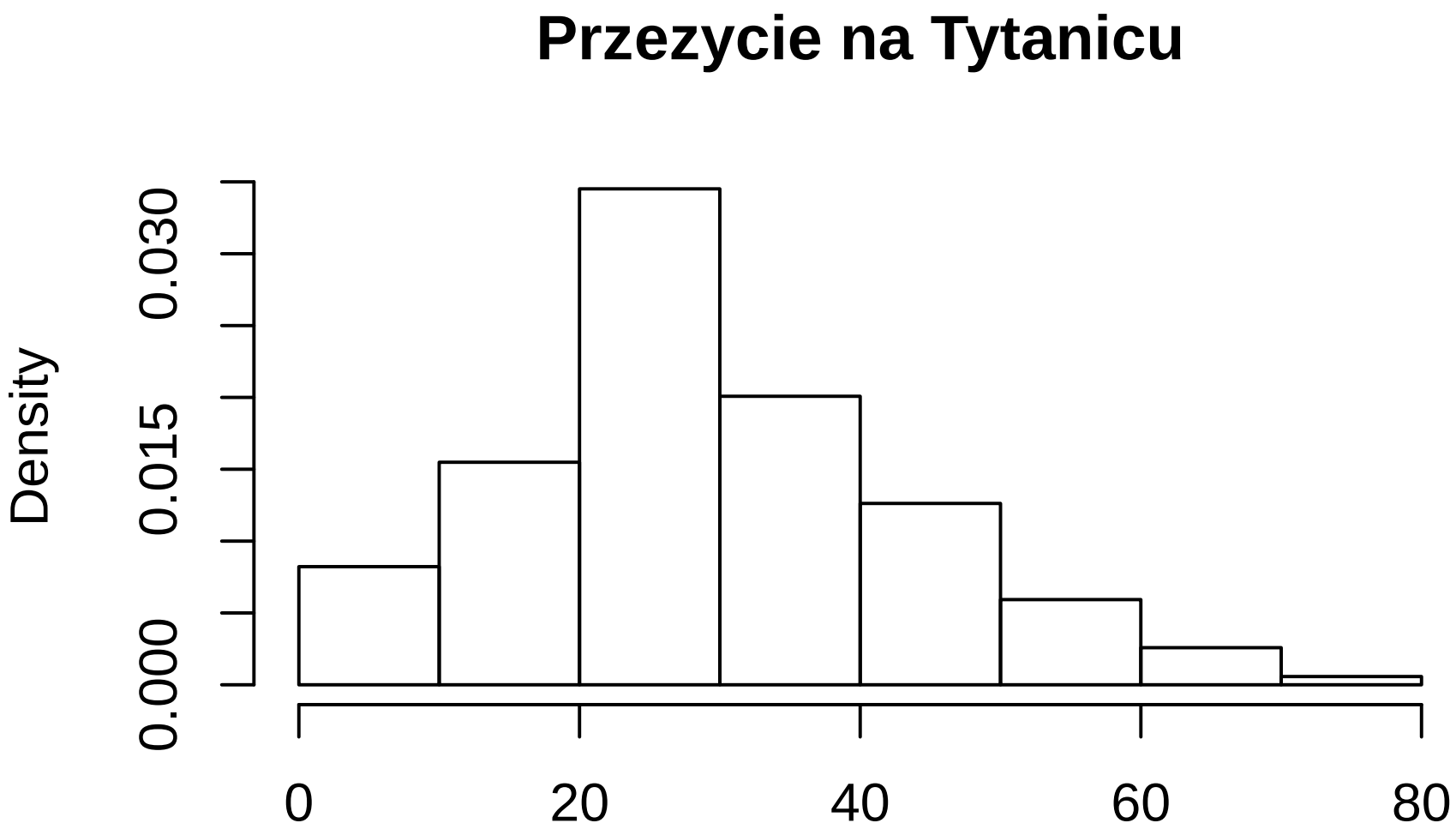
```
attach(TitanicSurvival)
hist(age, main = "Przezycie na Tytanicu")
```

Funkcja *hist()*



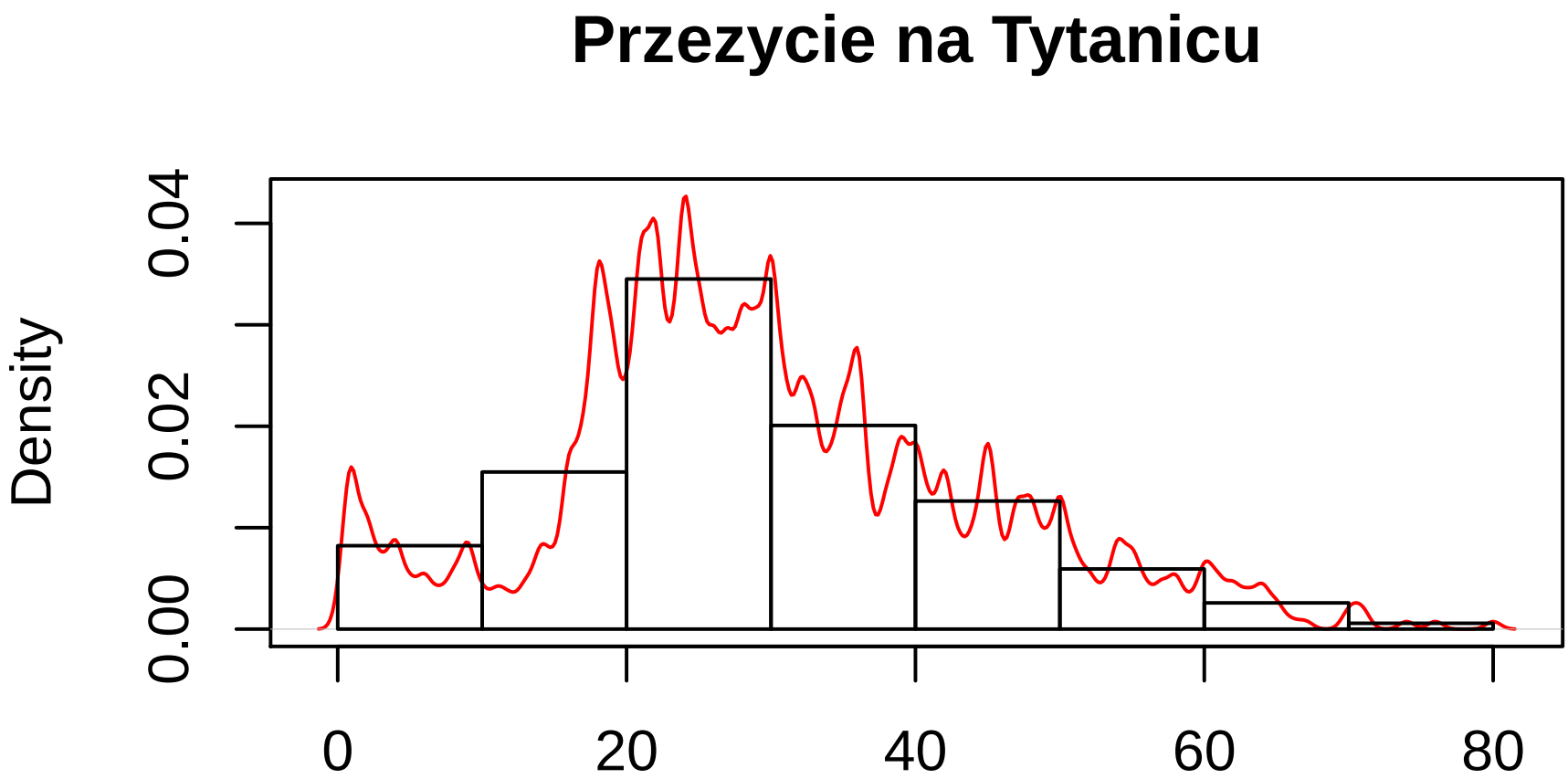
Funkcja *hist()*

```
hist(age, breaks=10, freq=F, main = "Przezycie na Tytanicu")
```



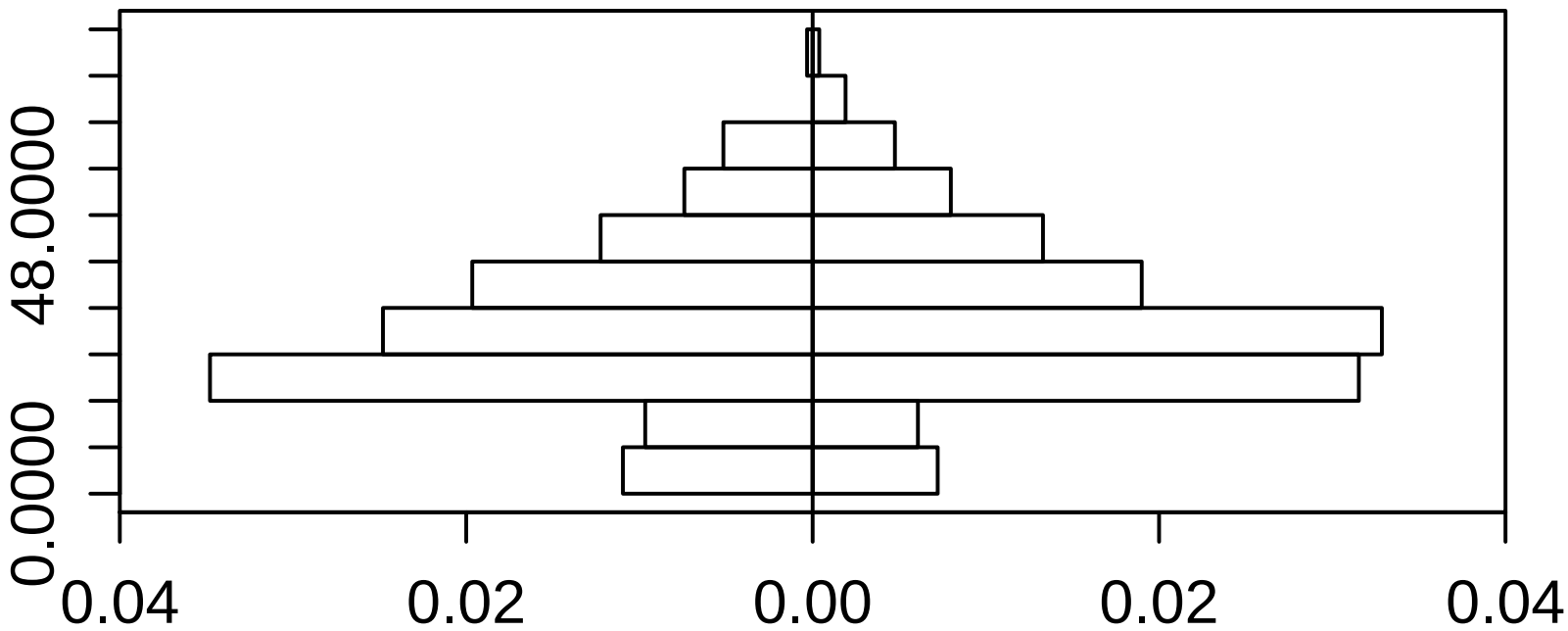
Funkcja *hist()*

```
plot(density(age, bw=0.5, na.rm = T),
     main = "Przezycie na Tytanicu", col = "red")
hist(age, breaks=10, freq=F, add=T)
```



Funkcja *histbackback()*{Hmisc}

```
library("Hmisc")
grupy <- split(age, sex)
histbackback(grupy, brks = (0:10)*8, probability = T)
```



Funkcja *boxplot()*

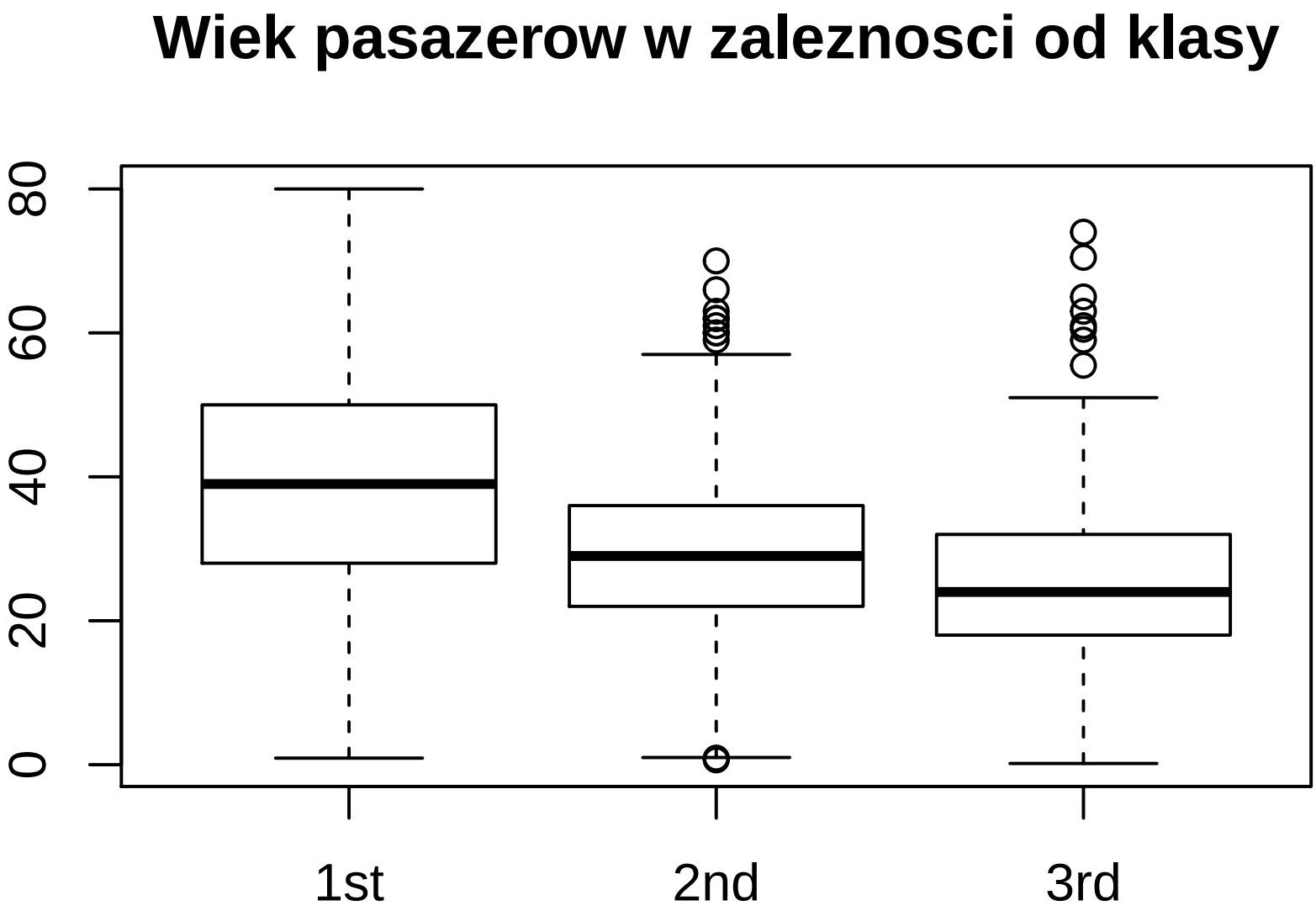
```
boxplot(formula, data = NULL, ..., subset, na.action = NULL)
```

Przydatne parametry funkcji *boxplot()*

- formula - zapisujemy jako y zmienna grupująca, pozwala na stworzenie kilku boxplotów, każdy na podstawie danych z jednej grupy
- subset - wektor określający, podzbiór użytych danych
- outline - jeśli ma wartość TRUE wartości odstające mają być rysowane
- notch - jeśli ma wartość TRUE na wykresie zaznaczany jest przedział ufności dla mediany
- horizontal - określenie orientacji rysowanych wykresów

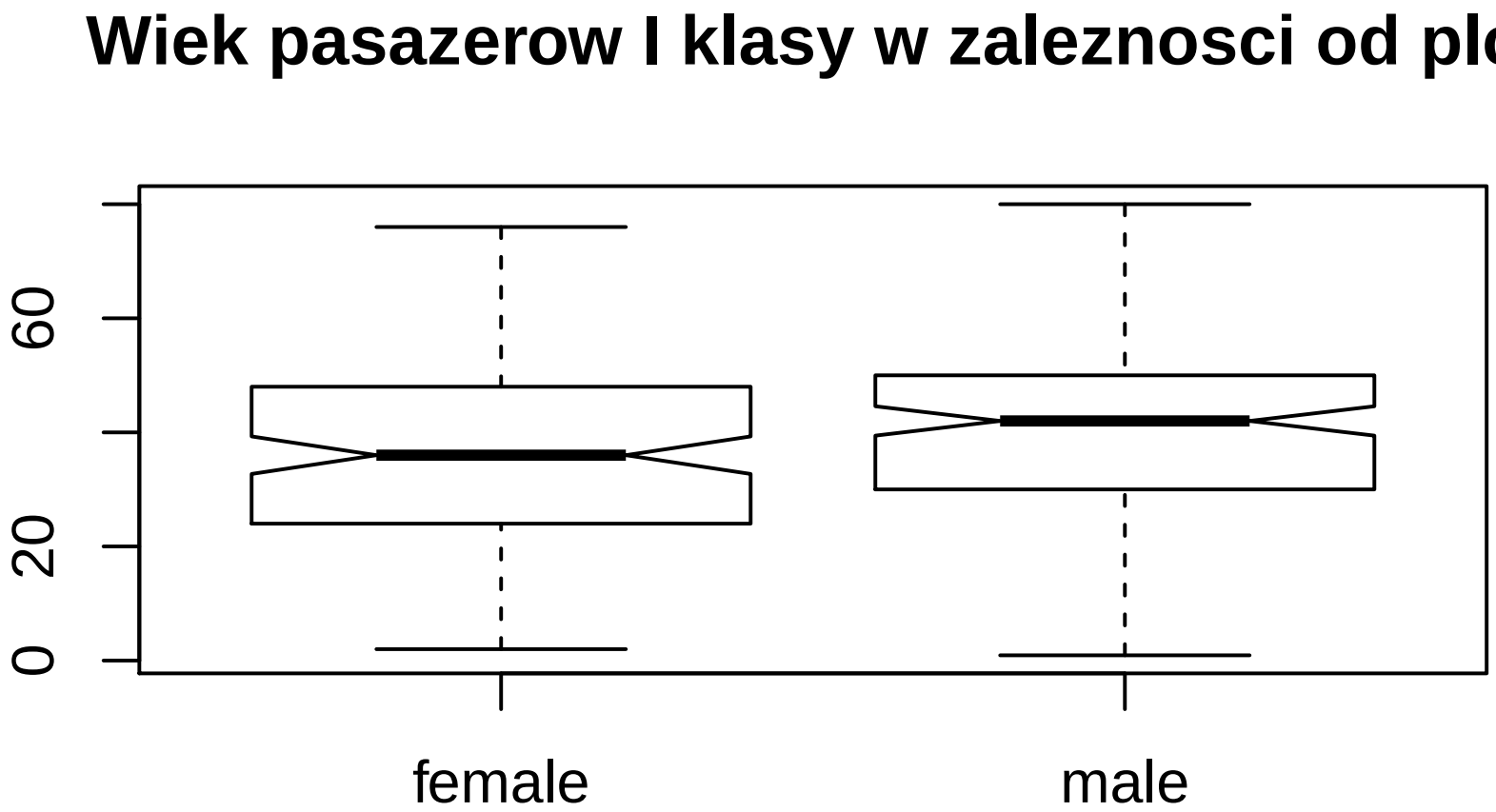
```
boxplot(age ~ passengerClass)
title("Wiek pasazerow w zaleznosci od klasy")
```

Funkcja *boxplot()*



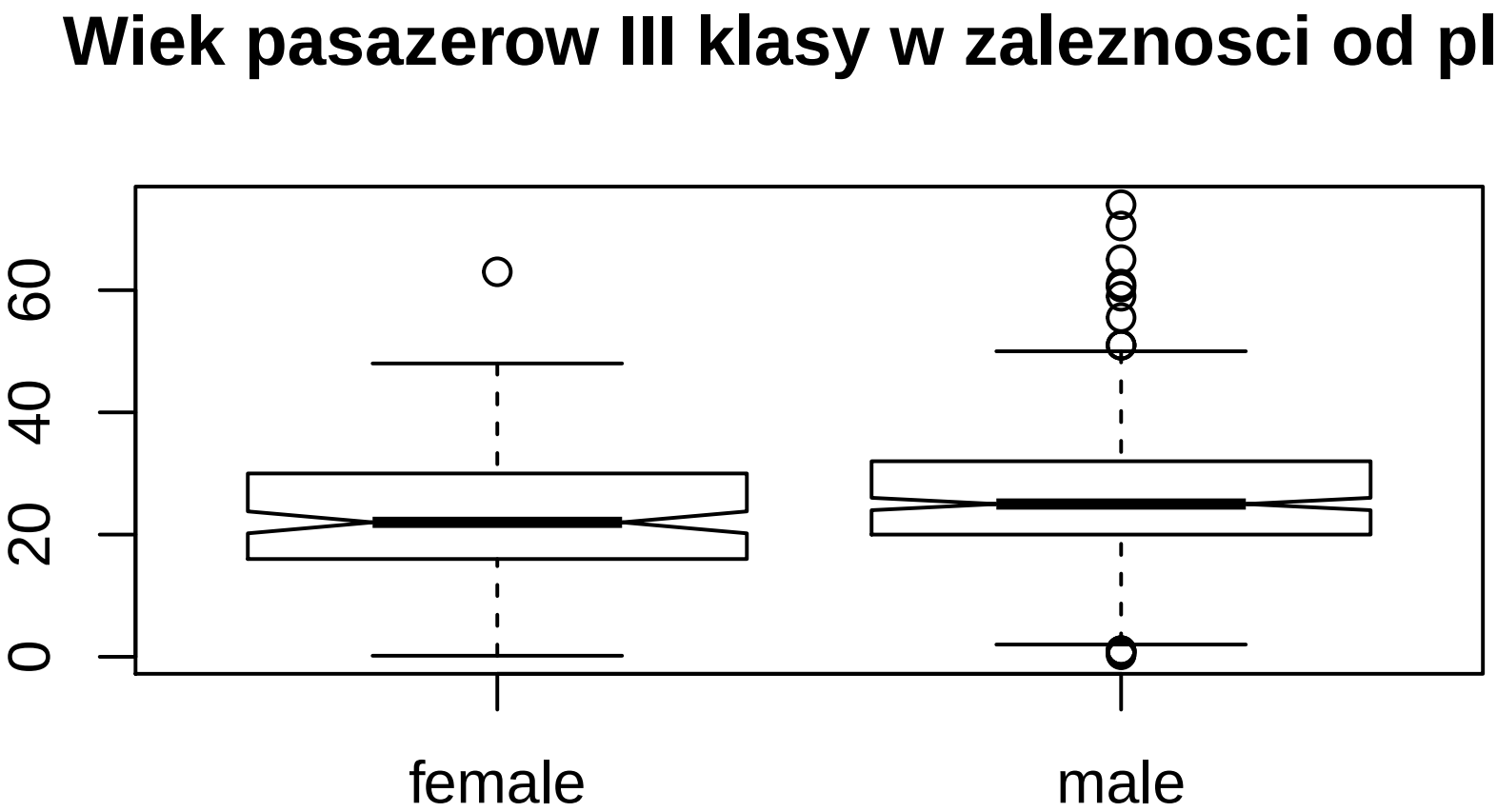
Funkcja *boxplot()*

```
boxplot(age ~ sex, subset = passengerClass == "1st", notch = T)
title("Wiek pasazerow I klasy w zaleznosci od plci")
```



Funkcja *boxplot()*

```
boxplot(age ~ sex, subset =
        passengerClass == "3rd", notch = T)
title("Wiek pasazerow III klasy w zaleznosci od plci")
```



Funkcja *barplot()*

```
barplot(height, width = 1, space = NULL, names.arg = NULL,
beside = FALSE, horiz = FALSE, density = NULL, angle = 45,
...)
```

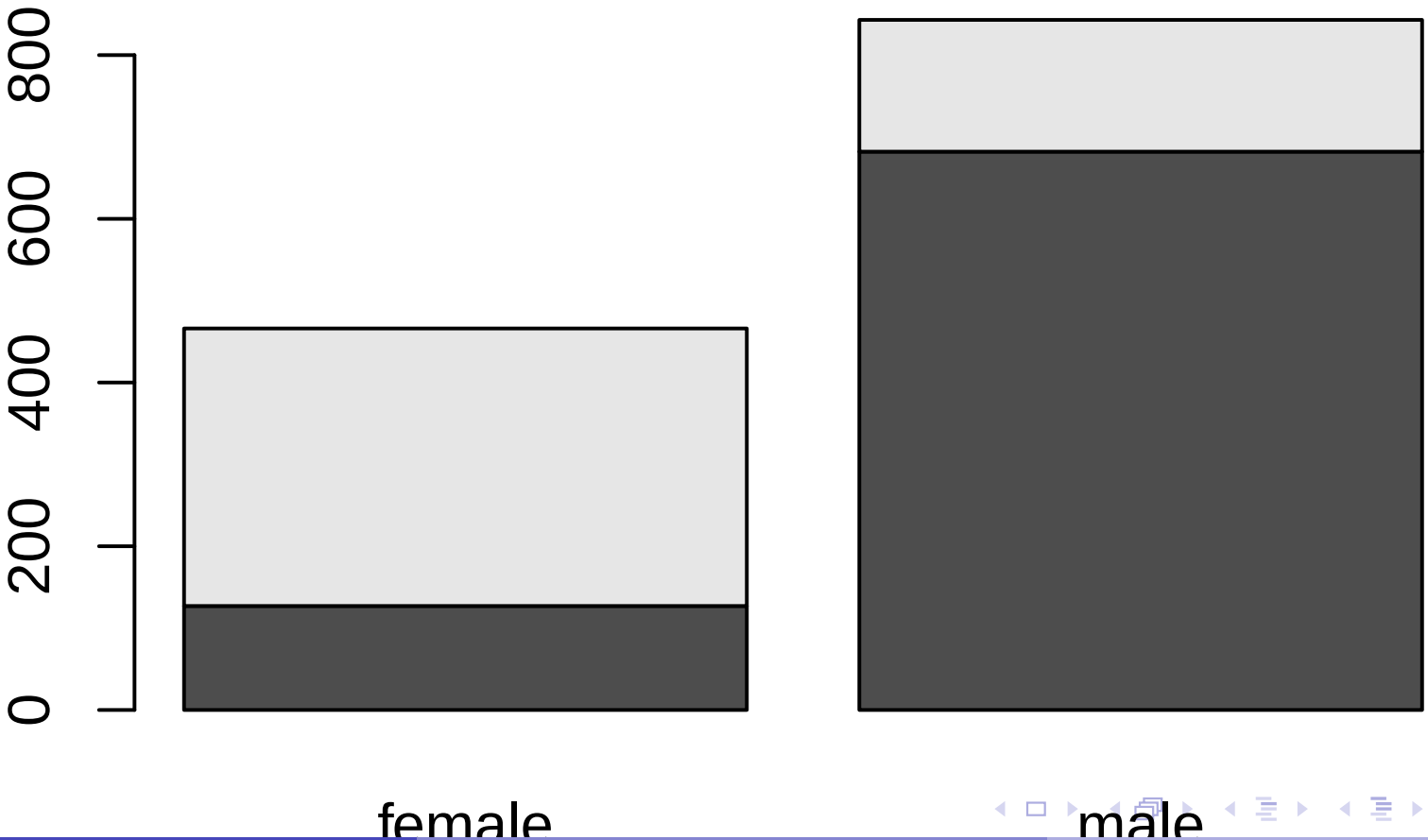
Przydatne parametry funkcji *boxplot()*

- height - dane, które mają być umieszczone na wykresie
- width - szerokość kolumn
- space - odległość pomiędzy kolumnami (mierzona jako procent szerokości)
- names.arg - nazwy elementów umieszczone pod kolumnami
- beside - jeśli TRUE kolumny z jednej grupy są rysowane jedna nad drugą, w przeciwnym razie są rysowane obok siebie

```
barplot(table(survived,sex), width = 1)
```

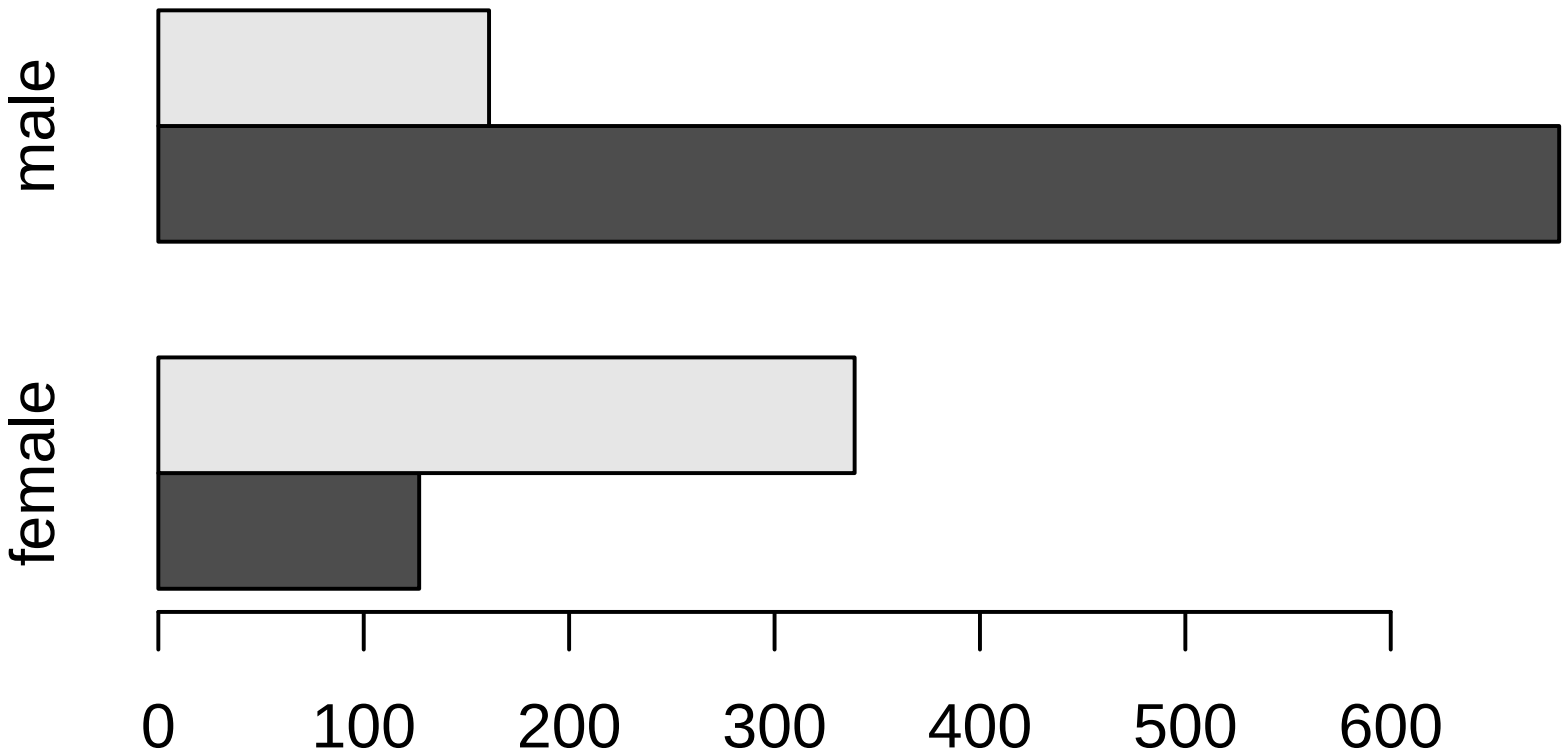
Funkcja *boxplot()*

```
barplot(table(survived, sex), width = 1)
```



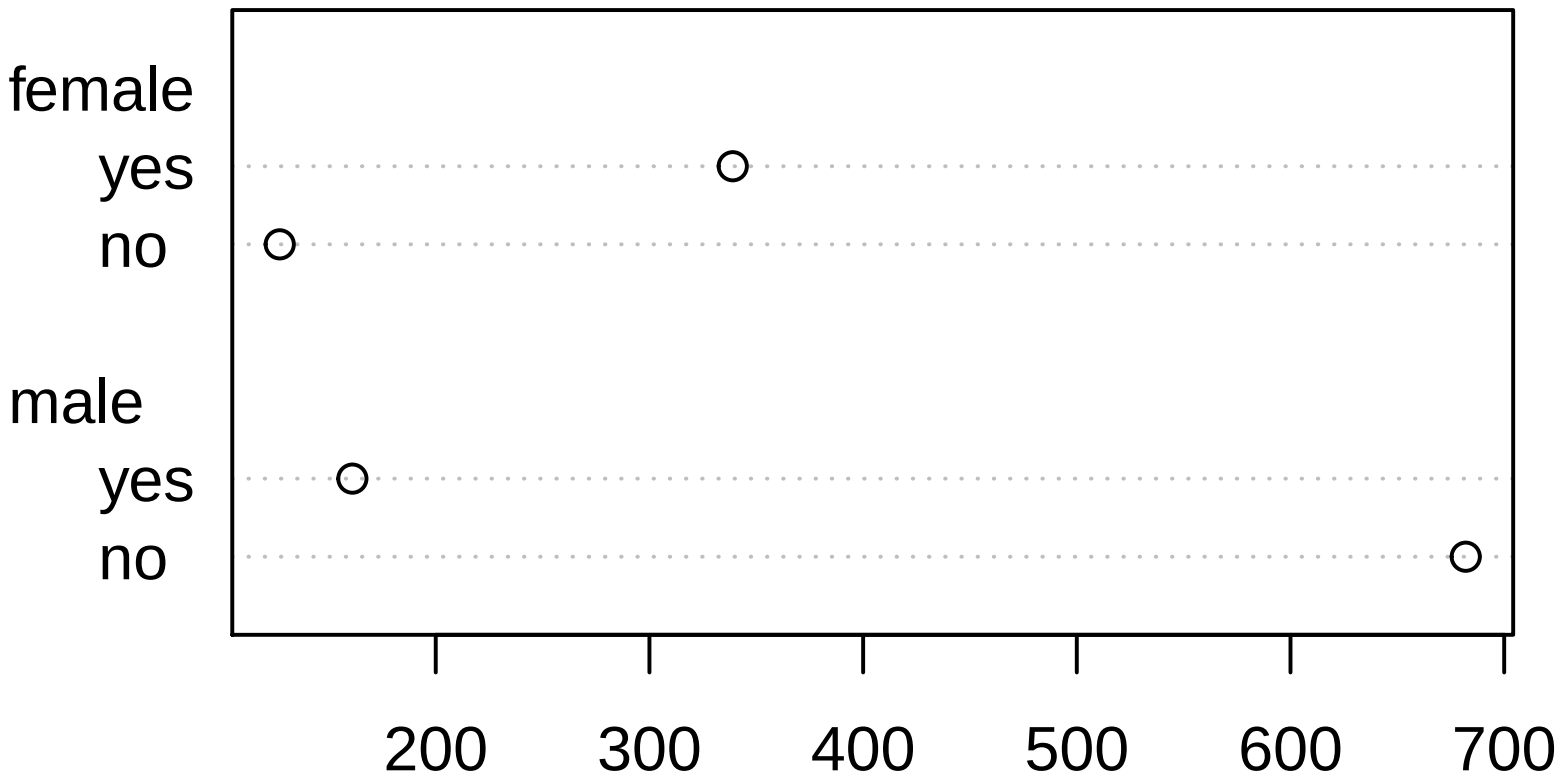
Funkcja *boxplot()*

```
barplot(table(survived, sex), horiz = T, beside = T, width = 1)
```



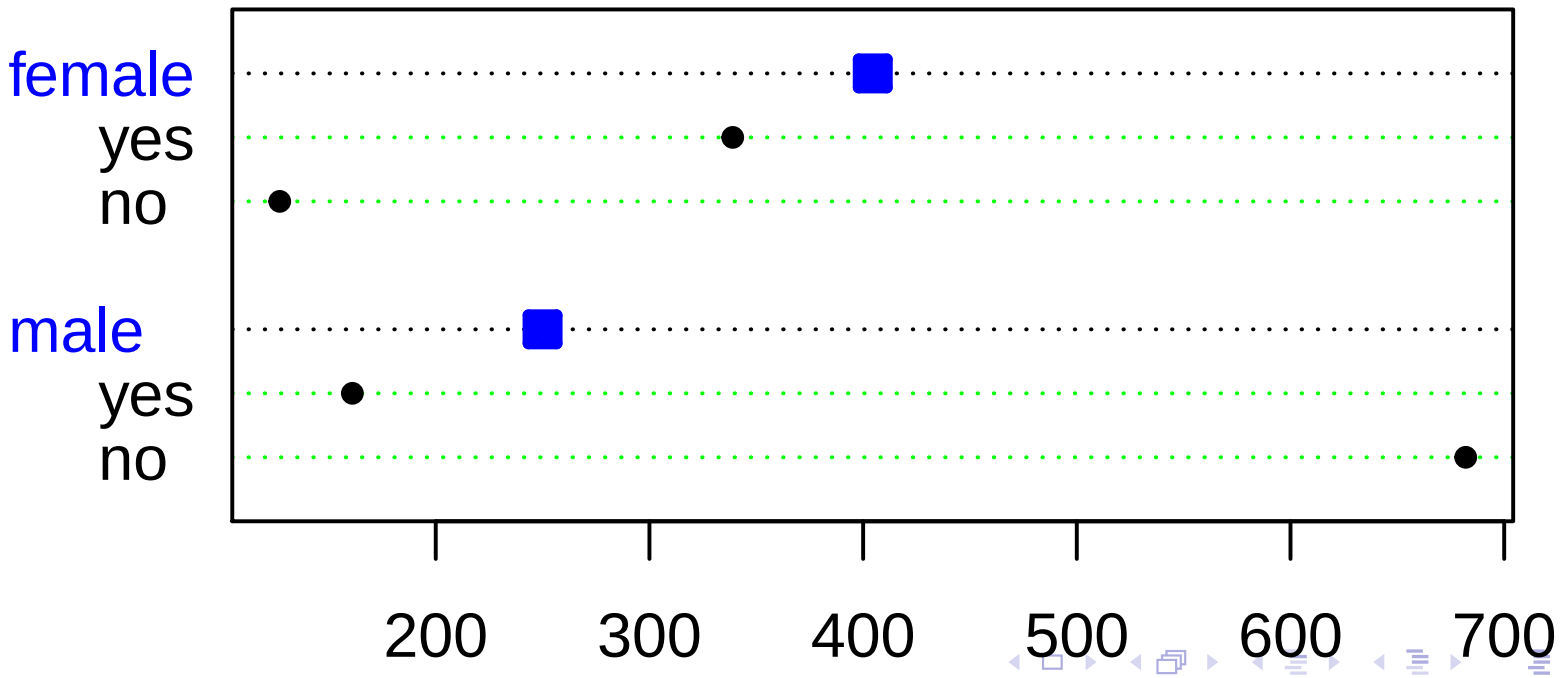
Funkcja *dotchart()*

```
dotchart(table(survived,sex))
```



Funkcja *dotchart()*

```
dane<-table(survived,sex)
srednie<-rowMeans(dane)
dotchart(dane, gdata = srednie, pch = 20, gpch = 7, lwd = 3,
         gcolor = "blue", lcolor = "green")
```



Inne przydatne funkcje graficzne

- *abline()* - rysowanie prostych
- *arrows()* - rysowanie strzałek
- *polygon()* - rysowanie krzywych zamkniętych
- *rect()* - rysowanie prostokątów
- *symbols()* - rysowanie symboli

Wykresy - ćwiczenia

- Wczytaj dane Studenci.xlsx. Na jego podstawie proszę stworzyć wykres rozrzutu (funkcja *plot()* typu punktowego), gdzie na osi X znajdować się będzie wiek studentów, a na osi Y wysokość stypendium. Proszę zaznaczyć kolorem czerwonym kobiety, a niebieskim mężczyzn. Znaczniki danych niech bedą zamalowanymi kropkami. Nazwij osie wykresu.
- Utwórz wykres dwuczęściowy z wykorzystaniem funkcji *par()* i parametru `mfrow`. Na górnym wykresie narysuj w przedziale $[-5, 5]$ następujące funkcje: $y = x^2$; $y = (x-2)^2$; $y = (x-2)^2 + 3$; $y = x^2 + 3$; $y = (x+1)^2 - 2$. Dodaj linie $y = 0$ w kolorze czarnym. Każda funkcja niech będzie narysowana innym kolorem. Zaznacz punkty odpowiadające argumentom -0.2 i 0.8. Na wykresie umieść legendę z opisem wzorów. Nadaj tytuł: "Wykresy funkcji przesunietych"i dodaj w dowolnym miejscu tekst: "Wykonał(a)...". Na dolnym wykresie narysuj na podstawie pliku Studenci.xlsx histogram dla wysokości stypendium z podziałem na płeć.
- Na podstawie danych TitanicSurvival wykonaj opisujący przeżycie pasażerów różnych klas. Dodaj etykiety klas. Dodaj tekst określający dokładne wartości. Zaznacz średnie wartości na wykresie.