Programowanie w R - podstawy

Magda G.

12 sierpnia 2018

## Poziom 0

### Podstawowe operacje arytmetyczne:

Z **R** można korzystać jak z kalkulatora. Wpisujemy bezpośrednio do konsoli:

# Dodawanie  
3 + 7

[1] 10

# Odejmowanie  
7 - 3

[1] 4

# Mnożenie  
3 \* 7

[1] 21

# Dzielenie  
7/3

[1] 2.333333

# Podnoszenie do kwadratu  
2^3

[1] 8

# Modulo - zwraca resztę z dzielenia  
8 %% 3

[1] 2

### Podstawowe funkcje matematyczne:

#### Logarytmy i potęgowanie

x=10  
log2(x) # logarithms o podstawie 2 z x

[1] 3.321928

log10(x) # logaritms o podstawie 10 z x

[1] 1

exp(x) # eksponenta z x

[1] 22026.47

#### Funkcje trygonometryczne

cos(x) # cosinus z x

[1] -0.8390715

sin(x) # sinus z x

[1] -0.5440211

tan(x) # tangens z x

[1] 0.6483608

acos(x) # arc-cosinus z x

[1] NaN

asin(x) # arc-sinus z x

[1] NaN

atan(x) #arc-tangens z x

[1] 1.471128

#### Inne funkcje matematyczne

abs(x) # wartość bezwzgledna z x

[1] 10

sqrt(x) # pierwiastek kwadratowy z x

[1] 3.162278

### Przypisywanie wartości do zmiennych

Zmienne służą do przechowywania wartości

Przykład: cena jabłek w złotówkach przypisana do zmiennej “cena\_jablek”

cena\_jablek <- 2 # Cena jabłek = 2 zł  
  
cena\_jablek = 2 # lub inny zapis

R jest wrażliwy na wielkość liter cena\_jablek, to nie to samo, co Cena\_jablek

cena\_jablek # wpisanie nazwy zmiennej wypisuje jej wartość

[1] 2

print(cena\_jablek) # działa tak samo

[1] 2

Na utworzonej zmiennej można wykonywać działania

5 \* cena\_jablek # pomnożyć cenę jabłek przez 5

[1] 10

cena\_jablek <- 5 # zmienić wartość zmiennej  
cena\_jablek

[1] 5

#### Tworzenie 2 zmiennych i wykonywanie obliczeń

# Wysokość prostokąta  
wys <- 10  
# Szerokość prostokąta  
szer <- 5  
# liczenie powierzchni prostokąta  
pow <- wys\*szer  
print(pow)

[1] 50

Funkcja **ls()** wyświetla listę zmiennych, które utworzyliśmy. W RStudio wszystkie zmienne wyświetlane sa w panelu *“Environment”*.

ls()

[1] "cena\_jablek" "pow" "szer" "wys" "x"

Kolekcja utworzonnych zmiennych nosi nazwę \*\*Workspace\*\*.   
Każda zmienna zajmuje pamięć komputera. Przy pracy z dużą ilością danych   
warto jest usuwać zbędne zmienne.

rm(wys, szer) # usuwanie zmiennych znajdujących się w nawiasie  
ls()

[1] "cena\_jablek" "pow" "x"

## Poziom 1

### Podstawowe typy danych

Dane dzielimy na: liczbowe, tekstowe i logiczne

# Dana liczbowa: Ile masz lat?  
wiek <- 28  
# Obiekt tekstowy: Jak masz na imię?  
imie <- "Bartek"  
# Obiekt logiczny: Jesteś naukowcem?  
# (yes/no) <=> (TRUE/FALSE)  
naukowiec <- TRUE

Wektor tekstowy tworzymy wykorzystując ("") lub (’’). Jeśli w tekście zmiennej znajduje się cudzysłów lub apostrof, należy zastsować znak ucieczki () lub w pzrypadku pojawienia się apostofu zdanie objąć cudzysłowem i odwrotnie

'My friend\'s name is "George"'

[1] "My friend's name is \"George\""

"My friend's name is \"George\""

[1] "My friend's name is \"George\""

Możemy sprawdzić typ wybranej zmiennej

class(wiek)

[1] "numeric"

class(imie)

[1] "character"

Stosując funkcje: **is.numeric()**, **is.character()**, **is.logical()** sprawdzamy czy zmienna ma konkretny typ:

is.numeric(wiek)

[1] TRUE

is.numeric(imie)

[1] FALSE

Możliwe jest zmienianie typu zmiennej

wiek

[1] 28

as.character(wiek) # zmienia daną liczbową na tekstową.

[1] "28"

as.numeric(wiek) # Odwrotna zmiana

[1] 28

### Wektory (Vector)

Wektor to kombinacja wielu zmiennych liczbowych, tekstowych lub logicznych. Tworzymy wektroy liczbowe, tekstowe i logiczne. Wektor może zawierać tylko jeden typ danych. Wektor tworzony jest przy użyciu funkcji **c()**

# Wiek przyjaciół w postaci wektora liczbowego  
wiek\_przyj <- c(27, 25, 29, 26)   
wiek\_przyj

[1] 27 25 29 26

# Imiona przyjaciół jako wektor tekstowy  
przyj <- c("Marta", "Ala", "Janek", "Piotr")  
przyj

[1] "Marta" "Ala" "Janek" "Piotr"

# Stan cywilny przyjaciół jako wektor logiczny  
# Czy jest żonaty/ zamężna? (yes/no <=> TRUE/FALSE)  
zajety <- c(TRUE, FALSE, TRUE, TRUE)  
zajety

[1] TRUE FALSE TRUE TRUE

Możlie jest nadawanie wartościom wektora etykiet, które mogą służyć do selekcjonowania odpowiednich danych. Służy do tego funkcja **names()**

# Wektor bez etykiet  
wiek\_przyj

[1] 27 25 29 26

# Przypisywanie etykiet  
names(wiek\_przyj) <- c("Marta", "Ala", "Janek", "Piotr")  
wiek\_przyj

Marta Ala Janek Piotr   
 27 25 29 26

# Tworzenie wektora z etykietami  
wiek\_przyj <- c(Marta = 27, Ala = 25, Janek = 29, Piotr = 26)  
wiek\_przyj

Marta Ala Janek Piotr   
 27 25 29 26

Każdy wektor ma określona długość. Do jej określania służy funkcja **length()**

# Liczba przyjaciół  
length(przyj)

[1] 4

#### Brakujące wartości

Brakujące wartości zapisywane są jako NA

# NA - nie ma informacji dotyczącej dzieci  
dzieci <- c(Marta = "yes", Ala = "yes", Janek = NA, Piotr = NA)  
dzieci

Marta Ala Janek Piotr   
"yes" "yes" NA NA

is.na(dzieci) # sprawdza czy wektor zawiera brakujące dane

Marta Ala Janek Piotr   
FALSE FALSE TRUE TRUE

Występuje też inny rodzaj brakującej wartości NaN, pojawiającej się   
w wyniku nieprawidłowej kalkulacji np. 0/0 = NaN. Funkcja is.na() obie te zmienne   
traktuje identycznie.

#### Wybieranie danych z wektorów

Selekcjonowanie danych

# Wybieramy przyjaciela nr 2  
przyj[2] # indeksowanie przy użyciu []

[1] "Ala"

# Wybieramy przyjaciela nr 2 i 4   
przyj[c(2, 4)]

[1] "Ala" "Piotr"

# Wybieramy przyjaciela od 1 do 3  
przyj[1:3] # R indeksuje od 1 a nie od 0

[1] "Marta" "Ala" "Janek"

# Możliwe jest stosowanie etykiet  
  
wiek\_przyj["Ala"]

Ala   
 25

# Pomijanie danych dzięki stosowaniu indeksowania ujemnego  
przyj[-2] # Wybieramy wszystkich poza 2

[1] "Marta" "Janek" "Piotr"

# Pominięcie przyjaciół 2 i 4  
przyj[-c(2, 4)]

[1] "Marta" "Janek"

# Pominięcie przyjaciół od 1 do 3  
przyj[-(1:3)]

[1] "Piotr"

# Selekcjonowanie z użyciem wektora logicznego  
przyj[zajety == TRUE]

[1] "Marta" "Janek" "Piotr"

# Selekcjonowanie starzszych niż 27 lat  
przyj[wiek\_przyj >= 27]

[1] "Marta" "Janek"

# Przyjaciele, którzy nie mają 27 lat  
przyj[wiek\_przyj != 27]

[1] "Ala" "Janek" "Piotr"

# Pozbywanie się danych brakujących  
dzieci

Marta Ala Janek Piotr   
"yes" "yes" NA NA

dzieci[!is.na(dzieci)] # Zachowuje dane różne od NA (!is.na())

Marta Ala   
"yes" "yes"

# Zastąpienie NA tekstem "no"  
dzieci[is.na(dzieci)] <- "no"  
dzieci

Marta Ala Janek Piotr   
"yes" "yes" "no" "no"

Operatory logiczne w R:  
  
\* "<"" : mniej niż  
\* ">" : więcej niż  
\* "<="" : mniej lub równe  
\* ">=" : więcej lub równe  
\* "==" : równe  
\* "!=" : nie równe

#### Obliczenia na wektorach

# podstawowe operacje arytmetyczne i funkcje mogą być zastosowane w dzialaniach na wektorach numerycznych.   
# Operacje wykonywane są na każdym elemencie wektora po kolei.  
  
# Pensja moich przyjaciół w złotych  
pensja <- c(2000, 1800, 2500, 3000)  
names(pensja) <- c("Marta", "Ala", "Janek", "Piotr") # Przypisanie etykiet  
pensja

Marta Ala Janek Piotr   
 2000 1800 2500 3000

# Przemnożenie przez 2  
pensja\*2

Marta Ala Janek Piotr   
 4000 3600 5000 6000

# Mnożenie przez różne współczynniki  
# Utworzenie wektora współczynników o tej samej długości, co wektor pensja  
wsp <- c(2, 1.5, 1, 3)  
pensja\*wsp # Pomnożenie wektorów

Marta Ala Janek Piotr   
 4000 2700 2500 9000

#### Inne użyteczne funkcje

x <- c(4, 16, 9, 11, 33, 7, 15, 80, 46)  
max(x) # Zwraca maksimum wektora x

[1] 80

min(x) # Zwraca minimum wektora x

[1] 4

range(x) # Zwraca zakres wektora x

[1] 4 80

length(x) # Zwraca liczbę elementów wektora x

[1] 9

sum(x) # Zwraca sumę elementów wektora x

[1] 221

prod(x) # Zwraca iloczyn elementów wektora x

[1] 80791603200

# Średnią x można policzyć na 2 sposoby  
sum(x)/length(x)

[1] 24.55556

mean(x)

[1] 24.55556

sd(x) # Odchylenie standardowe wektora x

[1] 24.81487

var(x) # Wariancja elementów wektora x

[1] 615.7778

sort(x) # Sortowanie wektora x rosnąco

[1] 4 7 9 11 15 16 33 46 80

sort(x, decreasing = TRUE) # Sortowanie wektora x malejąco

[1] 80 46 33 16 15 11 9 7 4

# Korzystajac z wcześniejszych danych możemy policzyć  
sum(pensja)

[1] 9300

mean(pensja)

[1] 2325

range(pensja)

[1] 1800 3000

### Macierze (Matrix)

Mcierze przypominają arkusze Excel’a i podobnie jak one składają się z wielu rzędów i kolumn. Składają się z wielu pojedynczych wektorów **jednego typu**. Wykorzystywane są do przechowywania danych tabelarycznych - zwykle pojedynczy rząd to osobnik/ obserwacja, a kolumna, to zmienna.

Do tworzenia macierzy wykorzystuje sie funkcję cbind() lub rbind() w następujący sposób:

# Wektory liczbowe  
col1 <- c(5, 6, 7, 8, 9)  
col2 <- c(2, 4, 5, 9, 8)  
col3 <- c(7, 3, 4, 8, 7)  
  
# Połączenie wektorów kolumnami  
tab1 <- cbind(col1, col2, col3)  
tab1

col1 col2 col3  
[1,] 5 2 7  
[2,] 6 4 3  
[3,] 7 5 4  
[4,] 8 9 8  
[5,] 9 8 7

# Zmiana nazw rzędów  
rownames(tab1) <- c("row1", "row2", "row3", "row4", "row5")  
tab1

col1 col2 col3  
row1 5 2 7  
row2 6 4 3  
row3 7 5 4  
row4 8 9 8  
row5 9 8 7

cbind(): składa obiekty R kolumnami;  
rbind(): składa obiekty R rzędami;  
rownames(): wypisuje lub ustawia nazwy rzędów macierzy i jej podobnych;  
colnames(): wypisuje lub ustawia nazwy kolumn macierzy i jej podobnych

Do transponowania macierzy służy funkcja t()

t(tab1)

row1 row2 row3 row4 row5  
col1 5 6 7 8 9  
col2 2 4 5 9 8  
col3 7 3 4 8 7

Możliwe jest utworzenie macierzy z wykorzystaniem funkcji matrix()

# matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)  
 # data: opcjonalny wektor danych  
 # nrow, ncol: liczba rzędów i kolumn macierzy  
 # byrow: wartość logiczna. Jeśli FALSE (wartość wyjściowa) macierz wypełniana jest kolumnami,   
 # w przeciwnym przypadku - rzędami.  
 # dimnames: Lista 2 wektorów zawierająca, odpowiednio, nazwy rzędów i nazwy kolumn.  
  
mdat <- matrix(  
 data = c(1,2,3, 11,12,13),   
 nrow = 2, byrow = TRUE,  
 dimnames = list(c("row1", "row2"), c("C.1", "C.2", "C.3"))  
 )  
mdat

C.1 C.2 C.3  
row1 1 2 3  
row2 11 12 13

Wymiary macierzy możemy poznać dzięki następującym funkcjom:

ncol(tab1) # Liczba kolumn

[1] 3

nrow(tab1) # Liczba rzędów

[1] 5

dim(tab1) # Liczba kolumn i rzędów

[1] 5 3

Do wybierania elementów z macierzy (my\_data[row, col]) służą:

##### **1. dodatnie indeksy:**

# Wybierz rząd 2  
tab1[2, ]

col1 col2 col3   
 6 4 3

# Wybierz rzędy od 2 do 4  
tab1[2:4, ]

col1 col2 col3  
row2 6 4 3  
row3 7 5 4  
row4 8 9 8

# Wybierz kilka rzędów nie tworzących zakresu np.: 2 i 4  
tab1[c(2,4), ]

col1 col2 col3  
row2 6 4 3  
row4 8 9 8

# Wybierz kolomnę 3  
tab1[, 3]

row1 row2 row3 row4 row5   
 7 3 4 8 7

# Wybierz wartość znajdującą się w rzędzie 2 i kolummnie 3  
tab1[2, 3]

[1] 3

##### **2. nazwy kolumn/ rzędów lub jednocześnie - indeksy i nazwy**

# Wybierz kolumnę 2  
tab1[, "col2"]

row1 row2 row3 row4 row5   
 2 4 5 9 8

# Wybierz wartość z rzędu 3 i kolumny 2  
tab1[3, "col2"]

[1] 5

##### **3. indeksowanie wartościami ujemnymi - wykluczanie niektórych kolumn/ rzędów**

# Usuń kolumnę 1  
tab1[, -1]

col2 col3  
row1 2 7  
row2 4 3  
row3 5 4  
row4 9 8  
row5 8 7

##### **4. wektory logiczne**

col3 <- tab1[, "col3"]   
# Wybieranie kolumny (zmiennej), która będzie podstawą selekcji, tu: col3.   
# W col3 wybieramy elementy większe lub równe 4 i zatrzymujemy w macierzy   
# jedynie te wiersze, które spełniają ten warunek.  
# W kolumnie 3 przeprowadzone zostaje sprawdzenie warunku i R w pamięci   
# zapisuje wartości TRUE lub FALSE dla każdego elementu.   
# Zatrzymywane są wartości TRUE  
  
tab1[col3 >= 4, ]

col1 col2 col3  
row1 5 2 7  
row3 7 5 4  
row4 8 9 8  
row5 9 8 7

#### Obliczenia na macierzach

Mozliwe jest przeprowadzanie prostych kalkulacji na macierzach

tab1\*2 # Mnożenie elementów macierzy

col1 col2 col3  
row1 10 4 14  
row2 12 8 6  
row3 14 10 8  
row4 16 18 16  
row5 18 16 14

log2(tab1) # Oblicznaie logarytmu dla każdego elementu macierzy

col1 col2 col3  
row1 2.321928 1.000000 2.807355  
row2 2.584963 2.000000 1.584963  
row3 2.807355 2.321928 2.000000  
row4 3.000000 3.169925 3.000000  
row5 3.169925 3.000000 2.807355

rowSums(tab1) # Sumowanie wartości po rzędach

row1 row2 row3 row4 row5   
 14 13 16 25 24

colSums(tab1) # Sumowanie wartości po kolumnach

col1 col2 col3   
 35 28 29

rowMeans(tab1) # Średnia wartość w każdym rzędzie

row1 row2 row3 row4 row5   
4.666667 4.333333 5.333333 8.333333 8.000000

colMeans(tab1) # Średnia wartość w każdej kolumnie

col1 col2 col3   
 7.0 5.6 5.8

#### Kożystanie z funkcji apply()

Funkcji tej można użyć w celu wykonania obliczeń (na rzędach i kolumnach macierzy) z wykorzystaniem wbudowanych funkcji

###### **Uproszczona forma funkcji apply() jest następująca:**

apply(X, MARGIN, FUN),  
  
gdzie:  
X - macierz;  
MARGIN - możliwe wartości, to 1 dla rzędów lub 2 dla kolumn;  
FUN - wybrana funkcja do zastosowania na rzędach/ kolumnach.

# Policz średnią rzędów  
apply(tab1, 1, mean)

row1 row2 row3 row4 row5   
4.666667 4.333333 5.333333 8.333333 8.000000

# Oblicz/ wyszukaj medianę w rzędach  
apply(tab1, 1, median)

row1 row2 row3 row4 row5   
 5 4 5 8 8

# Policz średnie kolumn  
apply(tab1, 2, mean)

col1 col2 col3   
 7.0 5.6 5.8

### Czynniki (Factor)

Zmienne te reprezentują kategorie lub grupy danych. Do tworzenia czynników wykorzystuje się funkcję factor()

# Tworzenie  
grupa\_przyj <- factor(c(1, 2, 1, 2))  
grupa\_przyj

[1] 1 2 1 2  
Levels: 1 2

Zmienna *‘grupa\_przyj’* składa się z 2 kategorii: 1 i 2 - poziomu czynnika. Funkcja levels() wypisuje te poziomy.

# Jakie są poziomy zmiennej?  
levels(grupa\_przyj)

[1] "1" "2"

Możliwa jest zmiana nazw poziomów czynnikóW

# Zmiana nazw poziomów zmiennej  
levels(grupa\_przyj) <- c("przyjaciel", "kolega")  
grupa\_przyj

[1] przyjaciel kolega przyjaciel kolega   
Levels: przyjaciel kolega

Można wymusić kolejność wyświetlania poziomów czynników

# Zmiana porządku wyświetlania poziomów  
grupa\_przyj <- factor(grupa\_przyj,   
 levels = c("kolega", "przyjaciel"))  
grupa\_przyj

[1] przyjaciel kolega przyjaciel kolega   
Levels: kolega przyjaciel

summary(grupa\_przyj) # Wyświetla liczbę obserwacji należących do każdego poziomu

kolega przyjaciel   
 2 2

Do sprawdzenia czy dana zmienna jest czynnikiem służy funkcja logiczna is.factor() a funkcja as.factor() służy do zamiany innej zmiennej na czynnik

# Czy 'grupa\_przyj' to czynnik?  
is.factor(grupa\_przyj)

[1] TRUE

# Czy 'zajety' jest czynnikiem?  
is.factor(zajety)

[1] FALSE

# Zamienić 'zajety' na czynnik  
as.factor(zajety)

[1] TRUE FALSE TRUE TRUE   
Levels: FALSE TRUE

Do policzenia wartości z podziałem na grupy można wykorzystać czynniki. Funkcja tapply() posłuży do użycia funkcji (tu średniej) do obliczeń w grupach.

pensja

Marta Ala Janek Piotr   
 2000 1800 2500 3000

grupa\_przyj

[1] przyjaciel kolega przyjaciel kolega   
Levels: kolega przyjaciel

# Średnia pensja w grupach  
sred\_pens <- tapply(pensja, grupa\_przyj, mean)  
sred\_pens

kolega przyjaciel   
 2400 2250

# Wielkość każdej z grup  
tapply(pensja, grupa\_przyj, length)

kolega przyjaciel   
 2 2

# Tworzenie tabeli rozdzielczej/ krzyżowej  
table(grupa\_przyj)

grupa\_przyj  
 kolega przyjaciel   
 2 2

# Badanie zależności pomiędzy zmiennymi  
table(grupa\_przyj, zajety)

zajety  
grupa\_przyj FALSE TRUE  
 kolega 1 1  
 przyjaciel 0 2

### Ramki danych (Data Frame)

Ramka danych przypomina macierz, ale może zawierać zmienne różnego typu - kolumny z wartościami numerycznymi, logicznymi i tekstowymi. Rzędy sa pojedynczymi obserwacjami (np. osobnikami) a kolumny poszczególnymi zmiennymi. Ramki danych tworzy się przy użyciu funkcji data.frame()

# tworzenie ramki danych  
Przyjaciele <- data.frame(  
 imie = przyj,  
 wiek = wiek\_przyj,  
 wzrost = c(180, 170, 185, 169),  
 zajety = zajety  
)  
  
Przyjaciele

imie wiek wzrost zajety  
Marta Marta 27 180 TRUE  
Ala Ala 25 170 FALSE  
Janek Janek 29 185 TRUE  
Piotr Piotr 26 169 TRUE

Do sprawdzenia, czy dany obiekt jest ramka danych służy funkcja is.data.frame()

is.data.frame(Przyjaciele) # ramka dancyh

[1] TRUE

is.data.frame(tab1) # macierz

[1] FALSE

# Zamiana macierzy w ramkę danych  
class(tab1)

[1] "matrix"

tab1\_2 <- as.data.frame(tab1)  
class(tab1\_2)

[1] "data.frame"

Ramkę danych można transpozować podobnie jak macierz. Jednakże rodzaje danych zostaja wtedy ujednolicone, często do typu tekstowego.

t(Przyjaciele)

Marta Ala Janek Piotr   
imie "Marta" "Ala" "Janek" "Piotr"  
wiek "27" "25" "29" "26"   
wzrost "180" "170" "185" "169"   
zajety " TRUE" "FALSE" " TRUE" " TRUE"

#### Wybieranie danych z ramki danych

Wyboru danych z kolumn lub rzeędów można dokonać używając i nazw i ich położenia w ramce danych (np. kolumna 1, kolumna 2 i td.)

##### **1. indeksowane po nazwie i położeniu**

# Wybór danych z kolumny 'imie'  
Przyjaciele$imie # Korzystamy ze znaku $

[1] Marta Ala Janek Piotr  
Levels: Ala Janek Marta Piotr

Przyjaciele[, 'imie'] # Wybieramy kolumnę o danej nazwie

[1] Marta Ala Janek Piotr  
Levels: Ala Janek Marta Piotr

# Wybieranie więcej niż 1 kolumny  
Przyjaciele[ , c(1, 3)] # Wybieramy 2 kolumny 1 i 3

imie wzrost  
Marta Marta 180  
Ala Ala 170  
Janek Janek 185  
Piotr Piotr 169

# Wybieramy wszystkie kolumny bez 1  
Przyjaciele[, -1] # Wartość ujemna wskazuje na to, że kolumna 1 ma zostać pominięta

wiek wzrost zajety  
Marta 27 180 TRUE  
Ala 25 170 FALSE  
Janek 29 185 TRUE  
Piotr 26 169 TRUE

##### **2. indeksowane na podstawie danych**

# Wybierz przyjaciół starszych lub 27-letnich  
  
Przyjaciele$wiek >= 27 # Znajduje rzędy spełniające ten warunek

[1] TRUE FALSE TRUE FALSE

# Wartości spełniające warunek oznaczane są TRUE  
  
# Zapis oznacza - wybrać wszystkie rzędy gdzie wiek >= 27 i wypisać wszystkie wartości kolumn w tym rzędzie  
Przyjaciele[Przyjaciele$wiek >= 27, ]

imie wiek wzrost zajety  
Marta Marta 27 180 TRUE  
Janek Janek 29 185 TRUE

# Ograniczenie liczby wyświetlanych kolumn  
# W wybranych rzędach wyświetli tylko 2 pierwsze kolumny; używamy położenia kolumny c(1, 2)  
Przyjaciele[Przyjaciele$wiek >= 27, c(1, 2)]

imie wiek  
Marta Marta 27  
Janek Janek 29

Przyjaciele[Przyjaciele$wiek >= 27, c("imie", "wiek")] # Uzyskujemy to samo, ale używając nazw kolumn

imie wiek  
Marta Marta 27  
Janek Janek 29

# Jeśli kryteria wyboru są długie i nie wygodne w używaniu, można zapisac je do zmiennych   
lat27 <- Przyjaciele$wiek >= 27  
lat27

[1] TRUE FALSE TRUE FALSE

cols <- c("imie", "wiek")  
cols

[1] "imie" "wiek"

Przyjaciele[lat27, cols] # zmienne umożliwiaja wybór odpowiednich danych

imie wiek  
Marta Marta 27  
Janek Janek 29

##### **3. Selekcja z zastosowaniem funkcji subset()**

# Wybieramy przyjaciół , gdzie wiek >= 27  
subset(Przyjaciele, wiek >= 27)

imie wiek wzrost zajety  
Marta Marta 27 180 TRUE  
Janek Janek 29 185 TRUE

Ramki danych można rozszerzać o kolejne dane

# Dodanie kolumny 'grupa' do 'Przyjaciele'  
Przyjaciele$grupa <- grupa\_przyj  
Przyjaciele

imie wiek wzrost zajety grupa  
Marta Marta 27 180 TRUE przyjaciel  
Ala Ala 25 170 FALSE kolega  
Janek Janek 29 185 TRUE przyjaciel  
Piotr Piotr 26 169 TRUE kolega

cbind(Przyjaciele, grupa = grupa\_przyj) # Przyłaczanie kolumny z użyciem cbind()

imie wiek wzrost zajety grupa grupa  
Marta Marta 27 180 TRUE przyjaciel przyjaciel  
Ala Ala 25 170 FALSE kolega kolega  
Janek Janek 29 185 TRUE przyjaciel przyjaciel  
Piotr Piotr 26 169 TRUE kolega kolega

# rbind() - przyłącza rzędy - obserwacje

W przypadku ramek danych z wartościami liczbowymi możliwe   
jest stosowanie funkcji rowSums(), olSums(), colMeans(), rowMeans()  
i apply() jak opisano dla macierzy

### Listy (List)

Lista, to zbiór obiektów, które mogą być wektorami, macierzami, ramkami danych i tp. Lista moze składać się ze wszystkich typów obiektów R.

#### Tworzenie listy

# Utwórz listę  
rodzina <- list(  
 matka = "Zuzanna",   
 ojciec = "Piotr",  
 siostry = c("Alicja", "Monika"),  
 wiek\_siostr = c(12, 22)  
 )  
  
rodzina

$matka  
[1] "Zuzanna"  
  
$ojciec  
[1] "Piotr"  
  
$siostry  
[1] "Alicja" "Monika"  
  
$wiek\_siostr  
[1] 12 22

# Nazwy elementów listy  
names(rodzina)

[1] "matka" "ojciec" "siostry" "wiek\_siostr"

# Liczba elementów listy   
length(rodzina)

[1] 4

Lista rodzina zawiera 4 składniki, do których można odnosić się niezależnie stosując **rodzina[[1]]**, **rodzina[[2]]** i td.

Do wybierania elementów z listy używa się ich nazw, bądź indeksów:  
"rodzina$matka" = "rodzina[[1]]"  
"rodzina$ojciec" = "rodzina[[2]]"

# Zastosowanie nazwy [1/2]  
rodzina$ojciec

[1] "Piotr"

# Zastosowanie nazwy [2/2]  
rodzina[["ojciec"]]

[1] "Piotr"

# zastosowanie indeksu  
rodzina[[1]]

[1] "Zuzanna"

rodzina[[3]]

[1] "Alicja" "Monika"

# Wybieranie elementu składowej listy  
# Wybierz ([1]) element z rodzina[[3]]  
rodzina[[3]][1]

[1] "Alicja"

#### Dodawanie kolejnych elementów do listy

# dodawanie elementów  
rodzina$dziadek <- "Jan"  
rodzina$babcia <- "Maria"  
rodzina

$matka  
[1] "Zuzanna"  
  
$ojciec  
[1] "Piotr"  
  
$siostry  
[1] "Alicja" "Monika"  
  
$wiek\_siostr  
[1] 12 22  
  
$dziadek  
[1] "Jan"  
  
$babcia  
[1] "Maria"

Możliwe jest łączenie list

list\_a <- tab1  
list\_b <- wiek\_przyj  
list\_c <- rodzina  
  
list\_abc <- c(list\_a, list\_b, list\_c) # Lista składająca się z połączonych elementów  
names(list\_abc)

[1] "" "" "" "" ""   
 [6] "" "" "" "" ""   
[11] "" "" "" "" ""   
[16] "Marta" "Ala" "Janek" "Piotr" "matka"   
[21] "ojciec" "siostry" "wiek\_siostr" "dziadek" "babcia"

length(list\_abc)

[1] 25

list\_abc[2]

[[1]]  
[1] 6

list\_abc[[2]]

[1] 6

list\_abc[[22]]

[1] "Alicja" "Monika"

list\_abc[[22]][2]

[1] "Monika"

## Poziom 2

### Importowanie danych

W RStudio zaimplementowano 'interface' graficzny ułatwiający importowanie danych

#### Importowanie danych z plików tekstowych

##### Podstawowe funkcje R do importowania plików

Funkcja read.table() jest podstawową funkcją do wczytywania danych tabelarycznych. Dane importowane sa jako ramka danych. W zależności od typu importowanych plików stsowane są wariacje funkcji read.table():

* read.csv() - dla plików ‘.csv’
* read.csv2() - wariant dla dla plików ‘.csv’ w krajach, gdzie wartości dziesiętne oddzielane są przecinkiem, a pola średnikiem (Polska)
* read.delim() - dla plików ‘.txt’ z wartościami dziesiętnymi oddzielanymi kropką
* read.delim2() - dla plików ‘.txt’ z wartościami dziesiętnymi oddzielanymi przecinkiem.

###### **Uproszczona formuła tych funkcji wygląda następująco:**

# wczytywanie danych tabelarycznych  
read.table(file, header = FALSE, sep = "", dec = ".")  
  
# wczytywanie (".csv")  
read.csv(file, header = TRUE, sep = ",", dec = ".", ...)  
  
# wariant dla (".csv") z wartościami dziesiętnymi oddzielanymi przecinkiem  
read.csv2(file, header = TRUE, sep = ";", dec = ",", ...)  
  
# pliki rozdzielane tabulatorami - rozdzielane "." oraz ","  
read.delim(file, header = TRUE, sep = "\t", dec = ".", ...)  
read.delim2(file, header = TRUE, sep = "\t", dec = ",", ...),  
  
gdzie:  
  
file: ścieżka do pliku  
sep: separator “\t” dla plików rozdzielanych tabulatorami  
header: wartość logiczna; TRUE - read.table() zakłada, że pierwszy rząd,   
to nagłówki kolumn. Jeśli tak nie jest, należy podać argument: header = FALSE.  
dec: znak używany, jako oddzielenie wartości dziesiętnych.  
  
np.:  
# Wczytywanie pliku .txt o nazwie "mtcars.txt"  
my\_data <- read.delim("mtcars.txt")  
  
# Wczytywanie pliku .csv o nazwie "mtcars.csv"  
my\_data <- read.csv("mtcars.csv")

Pliki, które znajdują się w bieżącym katalogu (aby sprawdzić w jakim katalogu pracujemy możemy wykorzystac fonkcje getwd()) nie wymagają podawania ścieżki dostępu, tylko ich nazwy. W innych przypadkach należy podać pełną ścieżkę dostępu. W celu ułatwienia wyboru pliku stworzono funkcję file.choose(), która umożliwia interaktywny wybór pliku i automatycznie uzupełnia ścieżkę dostępu

# Wczytywanie pliku .txt  
my\_data <- read.delim(file.choose())  
  
# Wczytywanie pliku .csv  
my\_data <- read.csv(file.choose())

Jeśli dane zawierają kolumny z tekstem, R może założyć, że są to czynniki, albo dane grupujące (np.: “good”, “good”, “bad”, “bad”, “bad”). Aby zapobiec przekształceniu tekstu w czynniki i zachowaniu typu danych tekstowych (*string*), należy użyć opcji stringsAsFactor = FALSE do funkcji read.delim(), read.csv() i read.table().

my\_data <- read.delim(file.choose(),   
 stringsAsFactor = FALSE)

Możliwe jest również podanie innego typu separatora np.: “|” jako opcji funkcji read.table()

my\_data <- read.table(file.choose(),   
 sep ="|", header = TRUE, dec =".")

##### Korzystanie z możliwości pakietu readr

Pakiet readr umożliwia szybkie i przyjazne użytkownikowi importowanie danych do R. W porównaniu do podstawowych funkcji R pakiet readr jest znacznie szybszy (ca. 10x), wyświetla pasek postępu i posiada pełną funkcjonalność funkcji natywnych R. Pakiet posiada funkcje dla: plików tekstowych, linii plików i całych plików. read\_delim() to podstawowa funkcja wczytywania plików z pakietu readr i w zależności od typu pliku imporotwanego istnieją warianty: read\_csv() - dla warości rozdzielanych przecinkami (‘,’); read\_csv2() - dla wartości rozdzielanych średnikami (‘;’); read\_tsv() - dla wartości rozdzielanych tabulatorami (‘’).

###### **Uproszczona formuła tych funkcji wygląda następująco:**

# wczytywanie danych tabelarycznych  
read\_delim(file, delim, col\_names = TRUE)  
  
# wczytywanie (".csv")  
read\_csv(file, col\_names = TRUE)  
  
# wariant dla (".csv") z wartościami dziesiętnymi oddzielanymi przecinkiem  
read\_csv2(file, col\_names = TRUE)  
  
# pliki rozdzielane tabulatorami  
read\_tsv(file, col\_names = TRUE)  
  
gdzie:  
  
file: ścieżka do pliku, link lub wektor z danymi. Pliki o rozszerzeniach   
.gz, .bz2, .xz, lub .zip są automatycznie rozpakowywane.   
Pliki rozpoczynające sie od “http://”, “https://”, “ftp://”, lub “ftps://” są automatycznie pobierane.   
delim: znak rozdzielający dane w pliku  
col\_names: TRUE lub FALSE lub wektor z wartościami będącymi nagłówkami kolumn.   
Jeśli TRUE - to pierwszy rząd uznawany jest jako nazwy kolumn.  
  
np.:  
# importowanie pliku tekstowego .txt o nazwie "mtcars.txt"  
my\_data <- read\_tsv("mtcars.txt")  
  
# importowanie pliku tekstowego .csv o nazwie "mtcars.csv"  
my\_data <- read\_csv("mtcars.csv")

Podobnie jak w przypadku natywnych funkcji R pliki znajdujące sie poza bieżącym katologiem muszą byc importowane poprzez podanie pełnej ścieżki dostepu i podobnie możliwe jest stosowanie funkcji file.choose()

# Wczytywanie pliku .txt  
my\_data <- read\_tsv(file.choose())  
  
# Wczytywanie pliku .csv  
my\_data <- read\_csv(file.choose())  
  
# wczytywanie pliku tekstowego z wyszczególnionym separatorem (tu: "|")  
my\_data <- read\_delim(file.choose(), sep = "|")

# install.packages("readr") # wykonać, jeśli nie zainstalowany   
library("readr")  
my\_data <- read\_csv("city\_commutes.csv")  
problems <- problems(my\_data)  
my\_data

# A tibble: 8 x 13  
 city smh\_commute smh\_density urban\_pop urban\_area urban\_density  
 <chr> <dbl> <int> <int> <dbl> <dbl>  
1 Los ~ 30.7 1042 12150996 NA NA   
2 Phoe~ 24.6 195 3629114 NA NA   
3 San ~ 33.6 503 NA NA NA   
4 Sydn~ 35 390 NA NA NA   
5 Mont~ 30 890 3519595 1545. 2278.  
6 Toro~ 34 1004 5132794 1751. 2931.  
7 Vanc~ 29.7 854 2264823 NA NA   
8 Madr~ 31 1251 624000 NA NA   
# ... with 7 more variables: metro\_pop <int>, metro\_area <dbl>,  
# metro\_density <dbl>, city\_pop <int>, city\_area <dbl>,  
# city\_density <dbl>, wp\_density <dbl>

problems

# tibble [0 x 4]  
# ... with 4 variables: row <int>, col <int>, expected <chr>, actual <chr>

Pakiet readr próbuje automatycznie wykryć rodzaj danych znajdujących się w każdej kolumnie. W sytuacji, w którrej rozpoznał dane błędnie, może pojawić sie wiele ostrzeżeń. Aby temu zapobiec lub naprawić można użyc dodakowago argumentu podczas importowania col\_type(), umożliwiającego podanie typu danych w kolumnach. Dostępne są następujące typy danych:

* col\_integer(): dane numeryczne (alias = “i”)
* col\_double(): dane liczbowe (alias = “d”).
* col\_logical(): wartości logiczne (alias = “l”)
* col\_character(): zachowuje tekst, nie zmienia na czynniki (alias = “c”).
* col\_factor(): czynniki lub zmienne grupujące (alias = “f”)
* col\_skip(): pomijanie kolumny (alias = “-” lub “\_“)
* col\_date() (alias = “D”), col\_datetime() (alias = “T”) i col\_time() (“t”) określa daty, daty i czas, oraz czas.

Przykładowo (kolumna x zawiera wartości numeryczne (i)   
a kolumna treatment = “character” (c):  
  
read\_csv("my\_file.csv", col\_types = cols(  
 x = "i", # wartości numeryczne  
 treatment = "c" # kolumna z tekstem  
))

##### Wczytywanie linii z pliku - funkcja read\_lines()

###### **Uproszczona formuła funkcji wygląda następująco:**

read\_lines(file, skip = 0, n\_max = -1L)  
  
  
 file: ścieżka do pliku  
 skip: liczba linii, które mają być pominięte zanim rozpocznie się wczytywanie  
 n\_max: liczba linii do wczytania. Jeśli n = -1, zostana wczytane wszystkie linie.

Funkcja read\_lines() zwraca wektor tekstowy, gdzie 1 elemnt, to 1 cały rząd

plik <- system.file("extdata/mtcars.csv", package = "readr") # plik demo  
dane <- read\_lines(plik) # wczytywanie danych z rzędów do kolejnych wektorów 1-elementowych  
head(dane)

[1] "\"mpg\",\"cyl\",\"disp\",\"hp\",\"drat\",\"wt\",\"qsec\",\"vs\",\"am\",\"gear\",\"carb\""  
[2] "21,6,160,110,3.9,2.62,16.46,0,1,4,4"   
[3] "21,6,160,110,3.9,2.875,17.02,0,1,4,4"   
[4] "22.8,4,108,93,3.85,2.32,18.61,1,1,4,1"   
[5] "21.4,6,258,110,3.08,3.215,19.44,1,0,3,1"   
[6] "18.7,8,360,175,3.15,3.44,17.02,0,0,3,2"

plik <- "city\_commutes.csv" # plik z bieżącego katalogu  
miasta <- read\_lines(plik, n\_max = 3) # wczytywanie 3 linii do 3 wektorów  
miasta

[1] "city,smh\_commute,smh\_density,urban\_pop,urban\_area,urban\_density,metro\_pop,metro\_area,metro\_density,city\_pop,city\_area,city\_density,wp\_density"  
[2] "Los Angeles,30.7,1042,12150996,,,13131431,12562,1045.329645,3976322,1302.15,3053.658949,3275.32"   
[3] "Phoenix,24.6,195,3629114,,,4737270,37725.1,125.5734246,1615017,1343.94,1201.703201,1204.61"

##### Wczytywanie całego pliku - funkcja read\_file

###### **Uproszczona formuła funkcji wygląda następująco:**

read\_file(file)

my\_file <- system.file("extdata/mtcars.csv", package = "readr") # demo  
read\_file(my\_file) # wczytanie całego pliku do 1 wektora

[1] "\"mpg\",\"cyl\",\"disp\",\"hp\",\"drat\",\"wt\",\"qsec\",\"vs\",\"am\",\"gear\",\"carb\"\n21,6,160,110,3.9,2.62,16.46,0,1,4,4\n21,6,160,110,3.9,2.875,17.02,0,1,4,4\n22.8,4,108,93,3.85,2.32,18.61,1,1,4,1\n21.4,6,258,110,3.08,3.215,19.44,1,0,3,1\n18.7,8,360,175,3.15,3.44,17.02,0,0,3,2\n18.1,6,225,105,2.76,3.46,20.22,1,0,3,1\n14.3,8,360,245,3.21,3.57,15.84,0,0,3,4\n24.4,4,146.7,62,3.69,3.19,20,1,0,4,2\n22.8,4,140.8,95,3.92,3.15,22.9,1,0,4,2\n19.2,6,167.6,123,3.92,3.44,18.3,1,0,4,4\n17.8,6,167.6,123,3.92,3.44,18.9,1,0,4,4\n16.4,8,275.8,180,3.07,4.07,17.4,0,0,3,3\n17.3,8,275.8,180,3.07,3.73,17.6,0,0,3,3\n15.2,8,275.8,180,3.07,3.78,18,0,0,3,3\n10.4,8,472,205,2.93,5.25,17.98,0,0,3,4\n10.4,8,460,215,3,5.424,17.82,0,0,3,4\n14.7,8,440,230,3.23,5.345,17.42,0,0,3,4\n32.4,4,78.7,66,4.08,2.2,19.47,1,1,4,1\n30.4,4,75.7,52,4.93,1.615,18.52,1,1,4,2\n33.9,4,71.1,65,4.22,1.835,19.9,1,1,4,1\n21.5,4,120.1,97,3.7,2.465,20.01,1,0,3,1\n15.5,8,318,150,2.76,3.52,16.87,0,0,3,2\n15.2,8,304,150,3.15,3.435,17.3,0,0,3,2\n13.3,8,350,245,3.73,3.84,15.41,0,0,3,4\n19.2,8,400,175,3.08,3.845,17.05,0,0,3,2\n27.3,4,79,66,4.08,1.935,18.9,1,1,4,1\n26,4,120.3,91,4.43,2.14,16.7,0,1,5,2\n30.4,4,95.1,113,3.77,1.513,16.9,1,1,5,2\n15.8,8,351,264,4.22,3.17,14.5,0,1,5,4\n19.7,6,145,175,3.62,2.77,15.5,0,1,5,6\n15,8,301,335,3.54,3.57,14.6,0,1,5,8\n21.4,4,121,109,4.11,2.78,18.6,1,1,4,2\n"

#### Importowanie danych z Excel (xls|xlsx) do R

Odpowiednie przygotowanie pliku Excel

1. Nazwy rzędów i kolumn
2. Pierwszy rząd ma zawierać nazwy kolumn. **Kolumny zwykle reprezentuja zmienne**
3. W pierwszej kolumnie należy umieścic nazwy rzędów. **Zwykle rzędy reprezentują obserwacje**
4. Każdy rząd powinien być **unikalny** - należy usunąć lub zastąpić zduplikowane nazwy
5. Nazwy powinny być zgodne z konwencją wykorzstywaną przez R
6. Unikać **spacji** w nazwach: dlugi\_skok, dlugi.skok - dobre nazwy; dlugi skok - zła nazwa kolumny/ rzędu
7. Unikać nazw z **symbolami specjalnymi**: ?, $, \*, +, #, (, ), -, /, }, {, |, >, < i tp. Jedym wyjątkiem jest podkreślenie
8. Nie używać cyfr na początku nazwy: sport\_100m i x100m - dobre nazwy; 100m - zła nazwa kolumny/ rzędu iv) R jest **wrażliwy na wielkość czcionki**: Nazwa, NAZWA, nazwa - to różne nazwy
9. Usuń puste rzędy ze swoich danych
10. Usuń wszystkie komentarze z arkuszy
11. Zastąp brakujące dane wartościa **NA**
12. kolumny z datami powinny mieć format **DD/MM/RRRR**

Uporządkowany plik najlepiej zapsać jako .txt (plik tekstowy) lub .csv   
(plik z warościami rozdzielanymi przecinkami), co ułatwia importowanie danych do R.   
Nie jest to konieczne - można zachować format Excel.

##### 1. “Przeklejanie danych”

###### W wybranym pliki Excel wybrać i przekopiować zakres danych (ctrl + c)

dane <- read.table(file = "clipboard", sep = "\t", header=TRUE)

##### 2. Importowanie danych przy użyciu pakietu readxl

To podejście wymaga zainstalowania i załadowanie pakietu readxl

install.packages("readxl")  
library("readxl")`  
   
dane <- read\_excel("my\_file.xls")  
 lub  
dane <- read\_excel("my\_file.xlsx")

###### Jeśli brakujące dane oznaczone są jakimś innym znakiem niż pusta komórka, należy go wyszczególnić

dane <- read\_excel("data.xlsx", na = "---")

###### Kod zakłada, że plik znajduje się w katalogu, w którym pracujemy. Aby sprawdzć ścieżkę do obecnego katalogu używa się getwd(). Funkcja file.choose() pozwala na interaktywne wybieranie pliku.

dane <- read\_excel(file.choose())  
   
podobnie w przypadku innych rodzajów plików:  
pliki .txt # dane <- read.delim(file.choose())  
pliki .csv # dane <- read.csv(file.choose())

###### Konkretny arkusz Excel wskazuje się z zastosowaniem opcji sheet = wykorzystując jego nazwę lub numer

dane <- read\_excel("data.xlsx", sheet = "data") # nazwa arkusza  
dane <- read\_excel("data.xlsx", sheet = 2) # nr arkusza

#### Importowanie danych z internetu

Funkcje read.delim(), read.csv() i read.table() można wykorzystać do importowania danych z sieci np.:

# web <- read.delim("http://www.sthda.com/upload/boxplot\_format.txt")  
# head(web)

Podobną funkcjonalność mają funkcje read\_delim(), read\_csv() i read\_tsv() z pakietu readr

# install.packages("readr") # wykonać, jeśli nie zainstalowany  
library("readr")  
# my\_data <- read\_tsv("http://www.sthda.com/upload/boxplot\_format.txt")  
# head(my\_data)

### Eksportowanie danych - podstawowe funkcje R

##### **Formuła funkcji write.table**

write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", ...)  
  
write.csv(...) # zapisywanie w formacie rozdzielanym przecinkami  
write.csv2(...) # zapisywanie w formacie rozdzielanym średnikami  
  
Argumenty:  
x - obiekt do zapisania; najlepiej macierz lub ramka danych. Każdy inny rodzaj danych zamieniany będzie   
na ramke danych  
file - ścieżka dostępu i nazwa pliku, do którego dane będą zapisywane; "" oznacza wypisanie   
w konsoli/ terminalu   
append - wartość logiczna; jeślieśli TRUE - wartości będą dopisywane do pliku, jeśli FALSE - plik   
zostanie nadpisany.  
quote - wartość logiczna; jeśli TRUE, każdy z elementów otaczany będzie cudzysłowami, wartości liczbowe   
zamieniane są tym sposobem w indeksy, jeśli FALSE - wartości nie są wstawiane w cudzysłów  
sep - znak rozdziału pól; " " - spacja, "\t" - tabulator; "," - przecinek  
... - inne opcje zapisu

### Eksportowanie danych z wykorzystaniem pakietu readr

##### **Formuła funkcji pakietu readr**

Zapisuje dane x, obiekty R, do pliku o określonej   
nazwie i ścieżce dostępu:  
  
# plik wyjściowy rozdzielany przecinkami  
write\_csv(x, path, na = "NA", append = FALSE,  
col\_names = !append)  
  
# plik wyjściowy z wybranym separatorem  
write\_delim(x, path, delim = " ", na = "NA",  
append = FALSE, col\_names = !append)  
  
# plik CSV dla Excel'a  
write\_excel\_csv(x, path, na = "NA", append =  
FALSE, col\_names = !append)  
  
# zapisywanie całego obiektu do 1 elementu tekstowego  
write\_file(x, path, append = FALSE)  
  
# zapisywanie wektora do pliku jako 1 elementu  
write\_lines(x,path, na = "NA", append = FALSE)  
  
# zapis z kompresowaniem pliku  
write\_rds(x, path, compress = c("none", "gz",  
"bz2", "xz"), ...)  
  
# plik wyjściowy rozdzielany tabulatorami  
write\_tsv(x, path, na = "NA", append = FALSE,  
col\_names = !append)  
  
Argumenty:  
x - ramka danych zapisywana na dysk  
path - ścieżka dostępu dla tworzonego pliku  
delim - znak używany do oddzielania wartości - pojedynczy znak; wartość   
domyślna to (" ") spacja   
na - wartość wstawiana w przypadku brakujących danych; domyślnie NA  
append - wartość logiczna; jeślieśli TRUE - wartości będą dopisywane do pliku,   
jeśli FALSE - plik zostanie nadpisany.  
col\_names - określa, czy uwzględnić nagłówki przy zapisie

### Tibbles alternatywa dla ramek danych

W porównaniu do tradycyjnej funkcji data.frame() (tworzącej ramiki danych) nowa funkcja data\_frame() (tworząca TIBBLES) :

* nie zamienia łańcuchów na czynniki
* nie zmienia nazw zmiennych
* nie tworzy nazw rzędów

# install.packages("tibble") # jeśli wymagana jest instalacja 'od'hash'ować  
library("tibble")  
  
# Tworzenie ramki danych poleceniem data.frame()  
przyjaciele <- data.frame(  
 imie = c("Marta", "Ala", "Janek", "Piotr"),  
 wiek\_przyj = c(27, 25, 29, 26),  
 wzrost = c(180, 170, 185, 169),  
 zajety = c(TRUE, FALSE, TRUE, TRUE)  
)  
  
przyjaciele

imie wiek\_przyj wzrost zajety  
1 Marta 27 180 TRUE  
2 Ala 25 170 FALSE  
3 Janek 29 185 TRUE  
4 Piotr 26 169 TRUE

przyjaciele2 <- data\_frame(  
 imie = c("Marta", "Ala", "Janek", "Piotr"),  
 wiek\_przyj = c(27, 25, 29, 26),  
 wzrost = c(180, 170, 185, 169),  
 zajety = c(TRUE, FALSE, TRUE, TRUE)  
)  
  
przyjaciele2

# A tibble: 4 x 4  
 imie wiek\_przyj wzrost zajety  
 <chr> <dbl> <dbl> <lgl>   
1 Marta 27 180 TRUE   
2 Ala 25 170 FALSE   
3 Janek 29 185 TRUE   
4 Piotr 26 169 TRUE

#### Konwertowanie danych do ‘tibbles’

Jeśli do importowania danych użyto funkcji pakietu 'readr', to nie ma konieczności przekształcania danych,   
bo importowane są jako tbl\_df (tibble data frame).

Do konwertowania danych zaimportowanych/ utworzonych jako ramki danych, listy, macierze wykorzystuje sie funkcję as\_data\_frame z pakietu tibble

library("tibble")  
  
data("iris") # korzystamy z wbudowanych danych 'iris'  
class(iris) # klasa danych

[1] "data.frame"

head(iris, 6)

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

# Convert iris data to a tibble  
irysy <- as\_data\_frame(iris) # konwertowanie do tibbles  
class(irysy) # klasa danych

[1] "tbl\_df" "tbl" "data.frame"

irysy

# A tibble: 150 x 5  
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
 <dbl> <dbl> <dbl> <dbl> <fct>   
 1 5.1 3.5 1.4 0.2 setosa   
 2 4.9 3 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 3.6 1.4 0.2 setosa   
 6 5.4 3.9 1.7 0.4 setosa   
 7 4.6 3.4 1.4 0.3 setosa   
 8 5 3.4 1.5 0.2 setosa   
 9 4.4 2.9 1.4 0.2 setosa   
10 4.9 3.1 1.5 0.1 setosa   
# ... with 140 more rows

Możliwe jest konwertowanie odwrotne - ‘tibbles’ do ramek danych przy użyciu funkcji as.data.frame(my\_data)

##### **Zalety ‘tibbles’ w porównaniu do ramek danych:**

* ‘Tibbles’ mają przyjazny sposób wyświetlania - pokazują tylko 10 pierwszych rzędów i wszystkie kolumny, które mieszczą sie na ekranie - jest to szczególne przydatne, gdy pracuje się z dużymi zestawami danych.
* Każda kolumna ma podany w nagłówku rodzaj danych, które zawiera: <dbl> dla danych liczbowych (double); <fct> - dla czynników (factor); <chr> - dla tekstowych (character) i <lgl> dla logicznych (logical).
* Możliwa jest zmiana standardowego sposobu wyświetlania z zastosowaniem opcji: options(tibble.print\_max = 20, tibble.print\_min = 6) # zmaian wyświetlania maksymalnej i minimalnej ilości wierszy; options(tibble.print\_max = Inf) # wyświetlanie wszystkich rzędów ; options(tibble.width = Inf) # wyświetlanie wszystkich kolumn.
* Wyselekcjonowane dane zawsze będą zapisywane jako ‘tibble’ - nie trzeba stosować opcji drop = FALSE, co było konieczne w przypadku tradycyjnych ramek danych.

### Podstawowe wykresy

#### **Wykres punktowy**

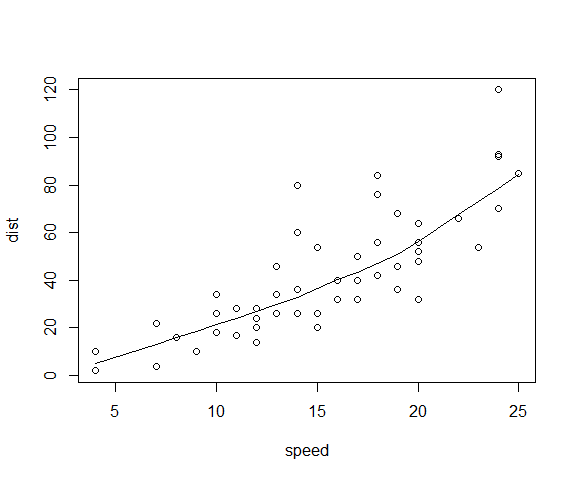
Stosowany głównie do przedstawiania zależności między zmienną x i y. Może być zastosowany do 1 zmiennej, w takim przypadku na osi x pojawiają się warości porządkowe.

Formuła

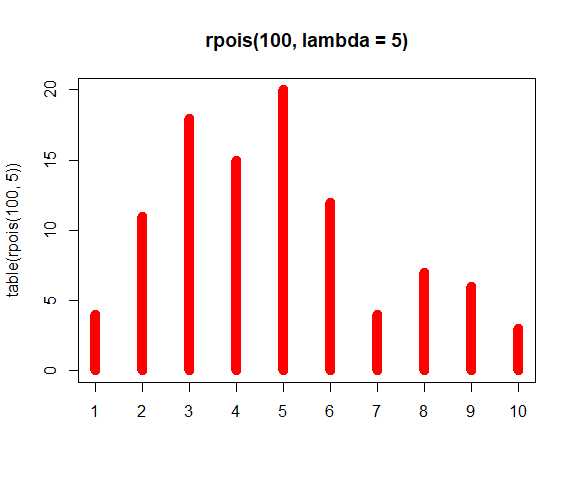
plot(x, y, ...)  
  
Argumenty  
  
x - koordynaty punktów na wykresie (może być wygenerowany automatycznie, jeżeli używamy danych z wektora)  
y - koordynaty wykresu; nie są podawane, gdy x jest wektorem  
... - dodatkowe informacje związane z wykresem

Przykłady

require(stats) # dla lowess, rpois, rnorm  
plot(cars)  
lines(lowess(cars))



plot(table(rpois(100, 5)), type = "h", col = "red", lwd = 10,  
 main = "rpois(100, lambda = 5)") # wykres dla danych dyskretnych wygenerowanych losowo, rozkład Poisson'a



#### **Wykres słupkowy**

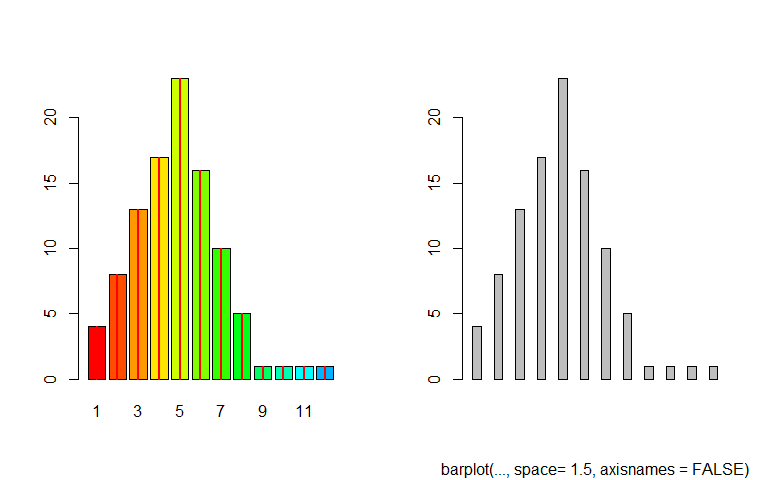
Służy głównie do przedstawiania danych kategorycznych

Formuła

barplot(height, ...)  
  
Argumenty  
  
height - wektor lub macierz opisująca słupki (wysokość)  
width - opcjonalnie podawany wektor określający szerokość słupków  
... - inne opcje wykresu

Przykłady

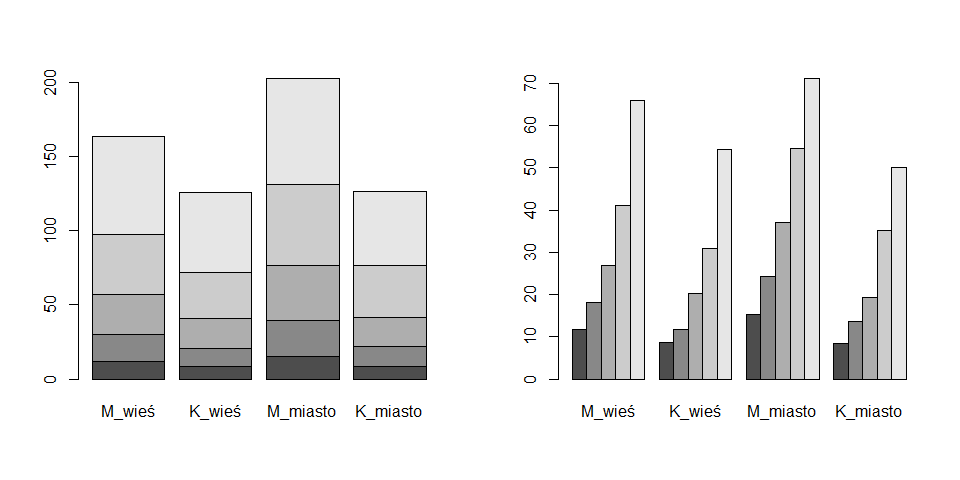
par(mfrow = c(1,2))  
require(grDevices) # kolory wykresu  
tN <- table(Ni <- stats::rpois(100, lambda = 5))  
r <- barplot(tN, col = rainbow(20))  
lines(r, tN, type = "h", col = "red", lwd = 2) # type = "h" to wykres słupkowy  
  
barplot(tN, space = 1.5, axisnames = FALSE,  
 sub = "barplot(..., space= 1.5, axisnames = FALSE)")



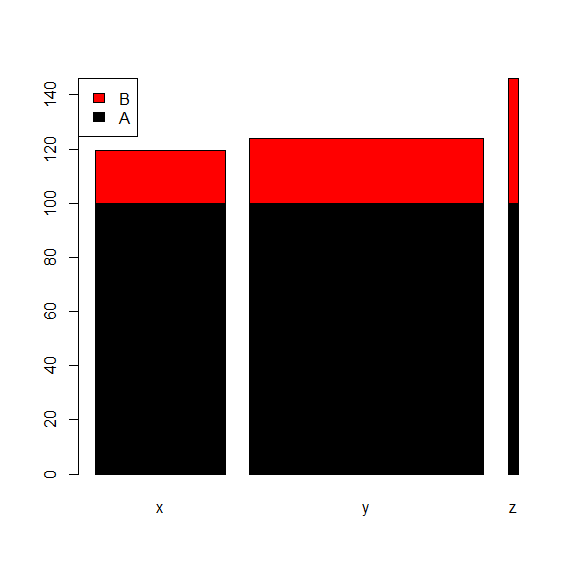
VADeaths # Liczba zgonóW w satnie Virginia w różnych kategoriach wiekowych

Rural Male Rural Female Urban Male Urban Female  
50-54 11.7 8.7 15.4 8.4  
55-59 18.1 11.7 24.3 13.6  
60-64 26.9 20.3 37.0 19.3  
65-69 41.0 30.9 54.6 35.1  
70-74 66.0 54.3 71.1 50.0

par(mfrow = c(1,2))  
barplot(VADeaths, names.arg = c("M\_wieś", "K\_wieś", "M\_miasto", "K\_miasto"))  
barplot(VADeaths, beside = TRUE, names.arg = c("M\_wieś", "K\_wieś", "M\_miasto", "K\_miasto"))



# legenda  
barplot(height = cbind(x = c(465, 91) / 465 \* 100,  
 y = c(840, 200) / 840 \* 100,  
 z = c(37, 17) / 37 \* 100),  
 beside = FALSE,  
 width = c(465, 840, 37),  
 col = c(1, 2),  
 legend.text = c("A", "B"),  
 args.legend = list(x = "topleft"))



#### **Wykres pudełkowy**

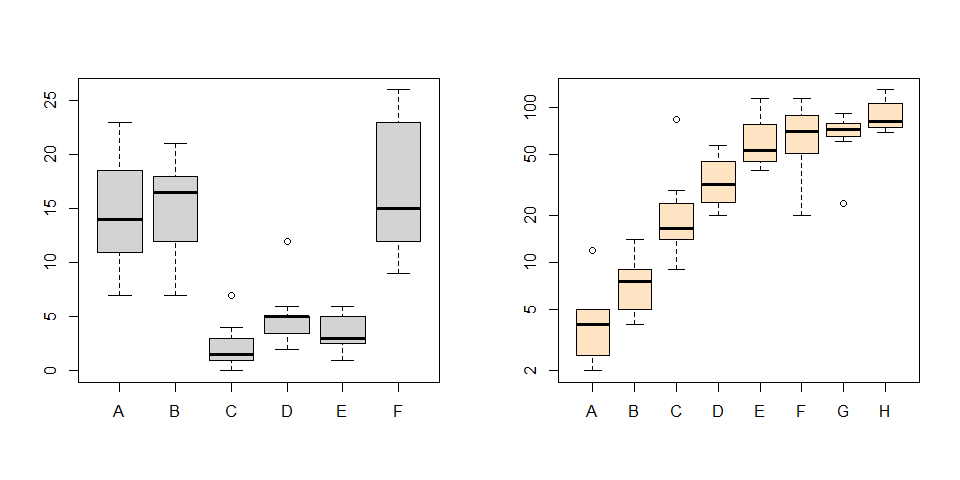
Służy do wizualizacji rozrzutu danych oraz jego porównywania między badanymi grupami

Formuła

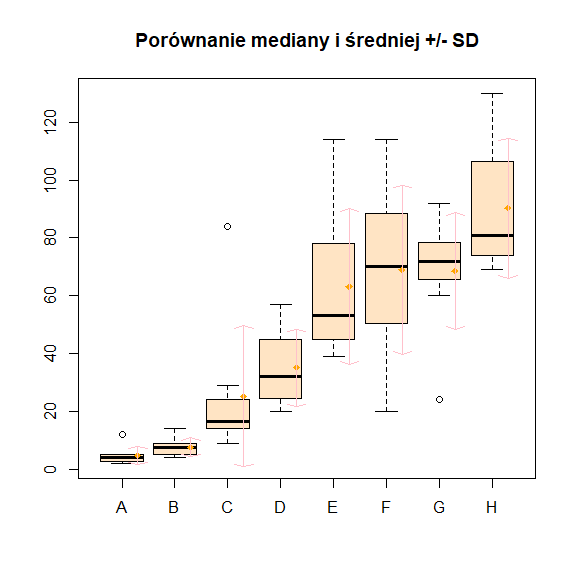
boxplot(x, ...) # dla danych  
boxplot(formula, data, ...) # dla formuły  
   
Argumenty:  
formula - formuła typu y ~ grp, gdzie y jest wektorem numerycznym powstałym przez grupowanie danych względem czynnika grupującego 'grp' (zwykle czynnika)  
data - ramka danych lub lista, na której została zastosowana formuła   
x - wektor liczbowy lub lista składająca się z wektorów, an podstawie, których ma być utworzony wykres

Przykłady

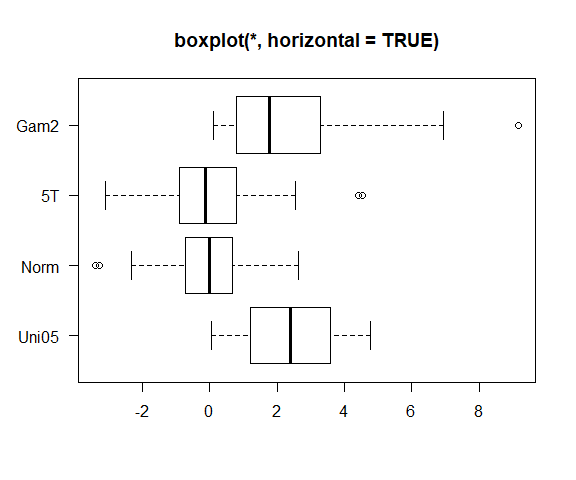
par(mfrow = c(1,2))  
boxplot(count ~ spray, data = InsectSprays, col = "lightgray") # wykorzystanie formuły  
boxplot(decrease ~ treatment, data = OrchardSprays,  
 log = "y", col = "bisque")



rb <- boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisque")  
title("Porównanie mediany i średniej +/- SD")  
mn.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, mean)  
sd.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, sd)  
xi <- 0.3 + seq(rb$n)  
points(xi, mn.t, col = "orange", pch = 18)  
arrows(xi, mn.t - sd.t, xi, mn.t + sd.t,  
 code = 3, col = "pink", angle = 75, length = .1)



mat <- cbind(Uni05 = (1:100)/21, Norm = rnorm(100),  
 `5T` = rt(100, df = 5), Gam2 = rgamma(100, shape = 2))  
df.mat <- as.data.frame(mat)  
par(las = 1) # horyzontalne ustawienie etykiet  
boxplot(df.mat, main = "boxplot(\*, horizontal = TRUE)", horizontal = TRUE)



#### **1-wymiarowy wykres punktowy**

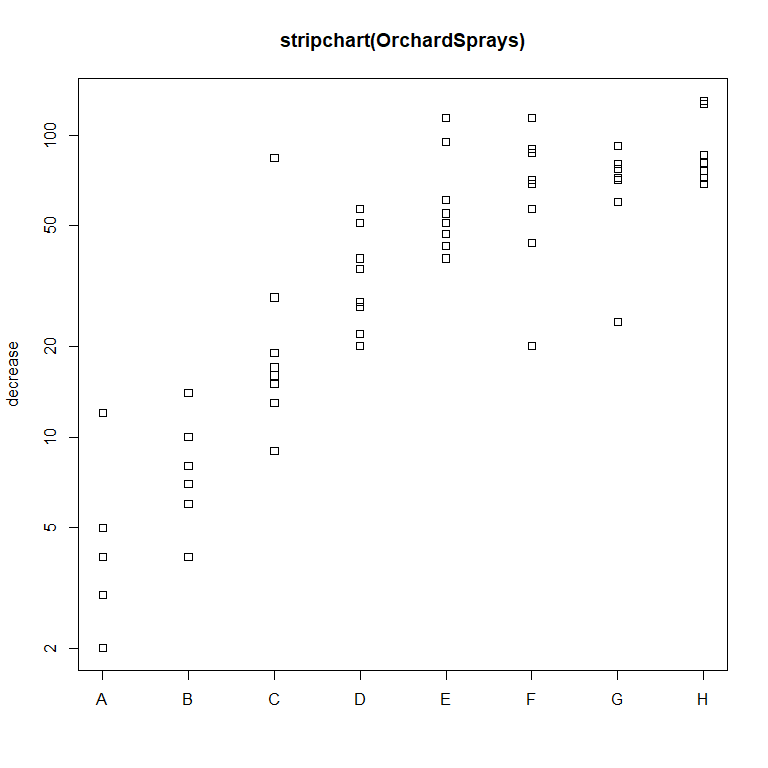
Przedstawia rozkład danych w postaci punktów i jest dobrą alternatywą dla wykresów pudełkowych, gdy dane są małoliczne

Formuła

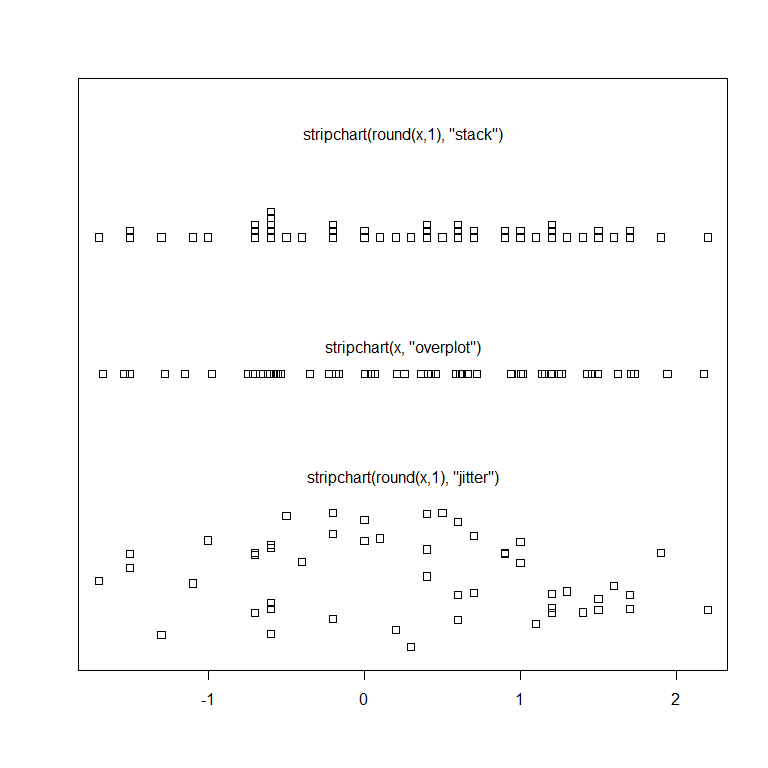
stripchart(x, data, ...)  
   
Argumenty:  
x - dane liczbowe w postaci wektora lub listy wektorów liczbowych (każdy odpowiadający komponencie wykresu). Jeśłi używamy formuły y ~ g do grupowania danych  
dane dzielone sa na poziomy odpowiadające poziomom 'g'  
data - ramka danych lub lista, z której pobierane będa dane  
... - inne parametry

Przykłady

stripchart(decrease ~ treatment,  
 main = "stripchart(OrchardSprays)",  
 vertical = TRUE, log = "y", data = OrchardSprays)



x <- stats::rnorm(50) # dane losowe o rozkładzie normalnym  
xr <- round(x, 1)  
stripchart(x) ; m <- mean(par("usr")[1:2])  
text(m, 1.04, "stripchart(x, \"overplot\")")  
stripchart(xr, method = "stack", add = TRUE, at = 1.2)  
text(m, 1.35, "stripchart(round(x,1), \"stack\")")  
stripchart(xr, method = "jitter", add = TRUE, at = 0.7)  
text(m, 0.85, "stripchart(round(x,1), \"jitter\")")



#### **Histogram**

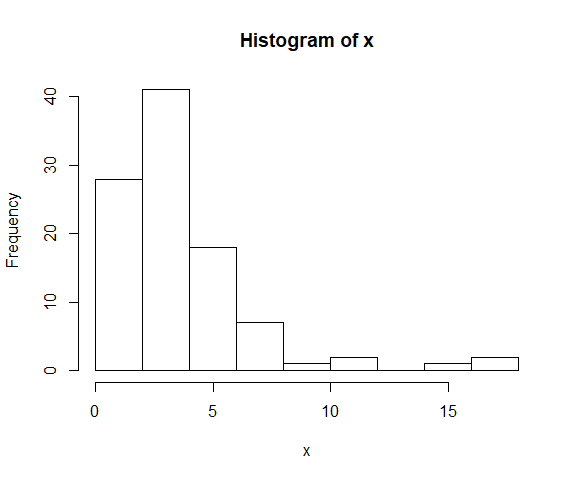
Przedstawia liczebność danych podzielonych na zakresy

Formuła

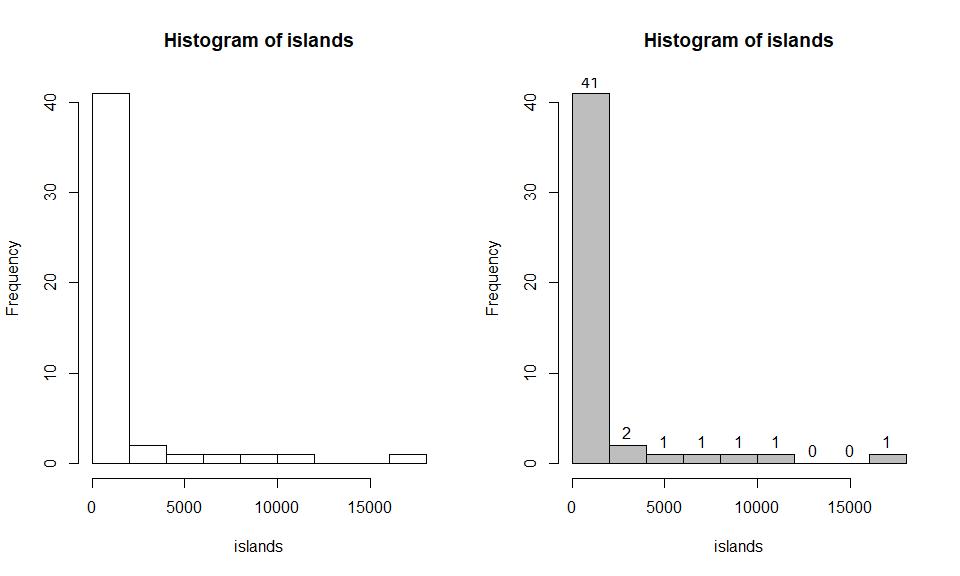
hist(x, breaks, ...)  
   
Argumenty:  
x - wektor, na podstawie, którego budowany jest histogram   
breaks - dane dotyczące podziału danych na zakresy (bins)  
freq - jeśli TRUE, dane przedstawione są jako częstości, jeśli FALSE - wykreślana jest gęstość funkcji (wartość wszystkich elementów wykresu sumowana jest do 1)

Przykłady

require(stats)  
set.seed(14)   
x <- rchisq(100, df = 4) # losowe generowanie danych  
hist(x)



par(mfrow = c(1, 2))  
hist(islands)  
utils::str(hist(islands, col = "gray", labels = TRUE))



List of 6  
 $ breaks : num [1:10] 0 2000 4000 6000 8000 10000 12000 14000 16000 18000  
 $ counts : int [1:9] 41 2 1 1 1 1 0 0 1  
 $ density : num [1:9] 4.27e-04 2.08e-05 1.04e-05 1.04e-05 1.04e-05 ...  
 $ mids : num [1:9] 1000 3000 5000 7000 9000 11000 13000 15000 17000  
 $ xname : chr "islands"  
 $ equidist: logi TRUE  
 - attr(\*, "class")= chr "histogram"

#### **Wykres mozaikowy**

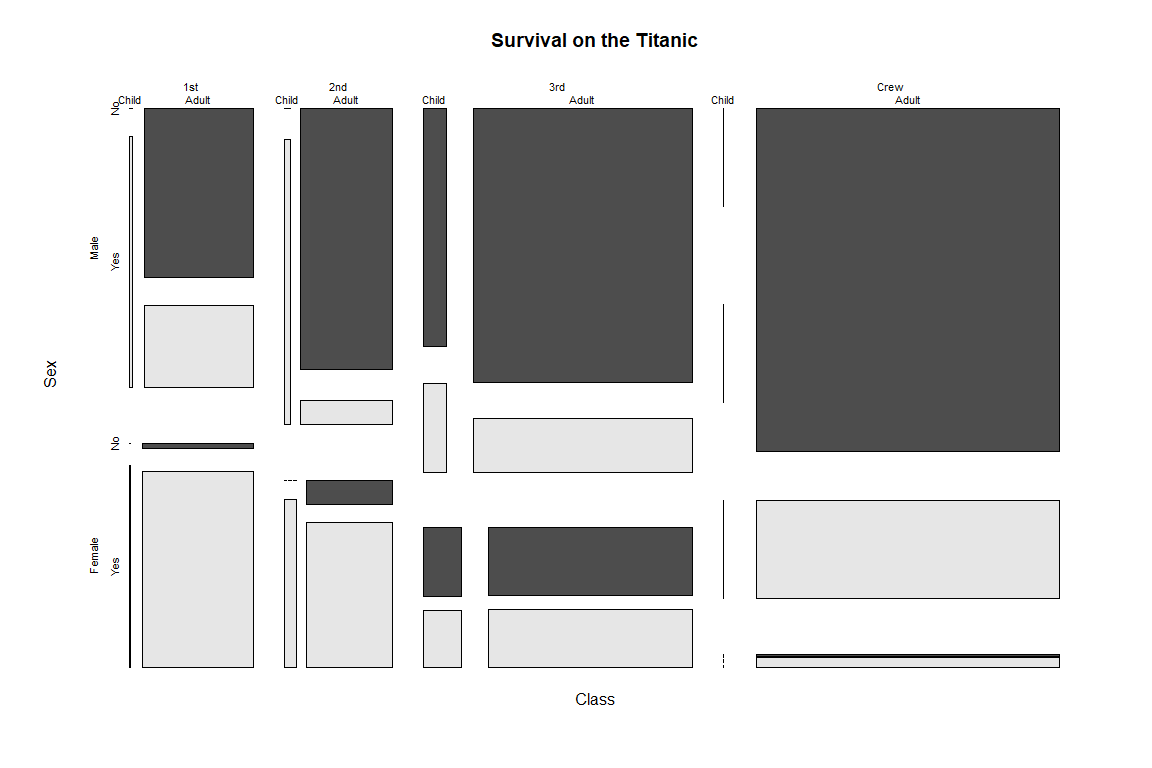
Przedsatwia dane zebrane w tabeli krzyżowej

Formuła

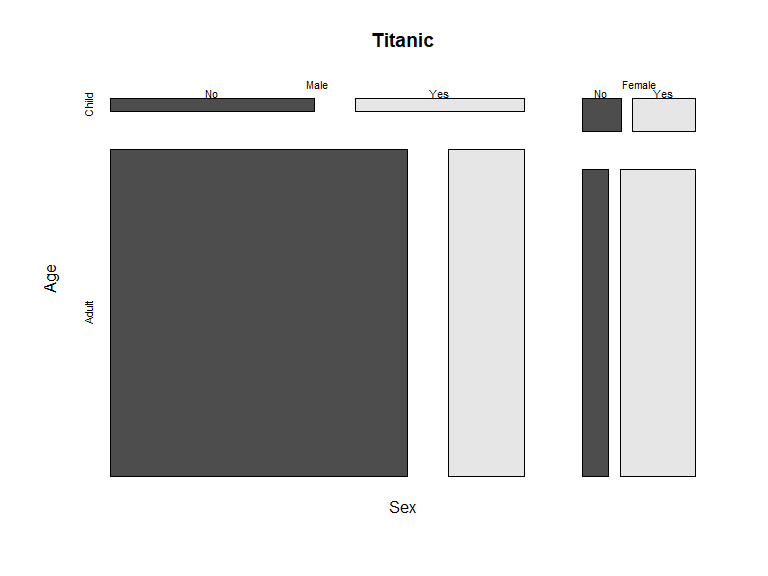
mosaicplot(x, ...) # tabela danych  
mosaicplot(formula, data, ...) # formula zależności danych  
   
Argumenty:  
x - tabela krzyżowa w formie macierzy  
formula - formuła grupująca dane, które chcemy przedstawić na wykresie

Przykłady

require(stats)  
mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)



require(stats)  
mosaicplot(~ Sex + Age + Survived, data = Titanic, color = TRUE) # formuła dla danych stabelaryzowanych



### Podstawy programowania

**Pętla FOR**

for (ZMIENNA in SEKWENCJA){ WYKONAJ }

Przykład:

for (i in 1:4){  
j <- i + 10  
print(j)  
}

[1] 11  
[1] 12  
[1] 13  
[1] 14

**Pętla WHILE**

while (WARUNEK){ WYKONAJ }

Przykład:

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

[1] 0  
[1] 1  
[1] 2  
[1] 3  
[1] 4

i

[1] 5

**Instrukcja warunkowa IF … ELSE …**

if (WARUNEK){ WYKONAJ 1 } else { WYKONAJ 2 }

Przykład:

a <- c(2,4,7,1,1,3,5)  
  
if (i > 3){  
print("Yes")  
} else {  
print("No")  
}

[1] "Yes"

for (i in a){  
if (i > 3){  
print("Yes")  
} else {  
print("No")  
}}

[1] "No"  
[1] "Yes"  
[1] "Yes"  
[1] "No"  
[1] "No"  
[1] "No"  
[1] "Yes"

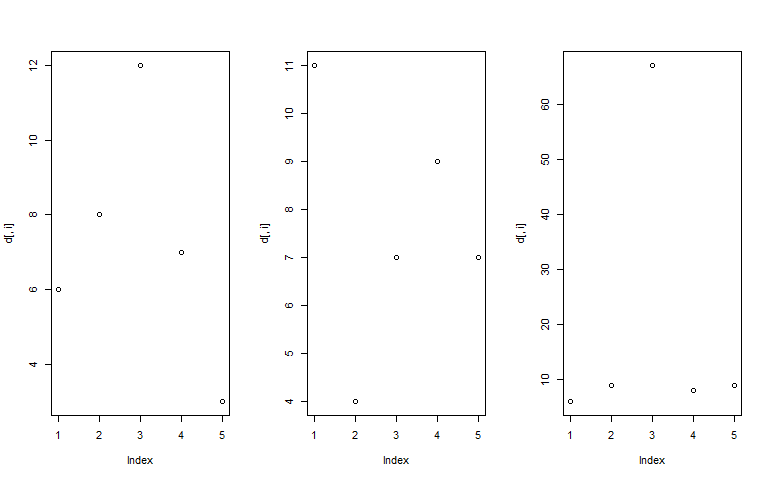
liczba <- 1233  
if (liczba %% 2 == 0) {  
 cat("liczba parzysta")  
} else {  
 cat("liczba nieparzysta")  
}

liczba nieparzysta

# Zagnieżdżenie funkcji  
  
a <- c(6,8,12,7,3)  
b <- c(11,4,7,9,7)  
c <- c(6,9,67,8,9)  
d <- data.frame(a,b,c)  
  
for (i in 1:length(d)){  
print(mean(d[,i]))  
i <- i + 1}

[1] 7.2  
[1] 7.6  
[1] 19.8

par(mfrow = c(1,3))  
for (i in 1:length(d)){  
plot(d[,i])  
i <- i + 1}



Do policznia średniej, czy też zastosowania innej funkcji na kolumnach lub wierszach ramki danych można wykorzystać wcześniej opisana funkcję apply

apply(d,2,mean)

a b c   
 7.2 7.6 19.8

**Funkcje**

nazwa\_funkcji <- function(ZMIENNA){ WYKONAJ return(NOWA\_ZMIENNA) }

Przykład:

square <- function(x){  
squared <- x\*x  
return(squared)  
}  
square(5)

[1] 25

square(c(2,3,8))

[1] 4 9 64