1 Wprowadzenie do R

1.1 Skróty klawiszowe programu RStudio

- `CTRL+SHIFT+n` - tworzy nowy plik źródłowy
- `CTRL+ENTER` - przekazuje kod z edytora do konsoli R
- `CTRL+1` i `CTRL+2` - przenoszą karetkę między edytorem a konsolą
- `CTRL+F11` i `CTRL+F12` - przenoszą karetkę między otwartymi skryptami

1.2 System pomocy

```
?mean
help(mean)
```

1.3 Pakiety

- Pakiet to zestaw narzędzi, takich jak nowe funkcje wraz z dokumentacją oraz nowe zbiory danych, rozszerzających funkcjonalność programu R. Większość z nich znajduje się w repozytorium CRAN (*Comprehensive R Archive Network*).
- `install.packages(nazwa pakietu, dependencies = TRUE)` - instalacja pakietu
- `library(nazwa pakietu)` - ładowanie pakietu
- `detach(package:nazwa pakietu)` - usunięcie pakietu

```
install.packages("car")
library(car)
detach(package:car)
```

1.4 Wektory atomowe

1.4.1 Wektory wartości logicznych

- W R zdefiniowane są dwie stałe logiczne:
  - `TRUE` - prawda,
  - `FALSE` - fałsz.

```
FALSE
```

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

- Wektory można tworzyć przez złączanie. Wektor (ciąg) składający się z konkretnych wartości logicznych w określonej kolejności, można utworzyć za pomocą funkcji `c()` (od ang. **c**ombine - złącz).

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

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

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

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

- Długość wektora zwraca funkcja `length()` .

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

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

1.4.2 Wektory liczbowe

```
c(1, +2, -3, 2.3, -.4, 5.)
```

```
## [1]  1.0  2.0 -3.0  2.3 -0.4  5.0
```

- Do generowania ciągów arytmetycznych w R służą:
  - operator `:` (różnica równa się 1 lub -1),
  - funkcja `seq()` (od ang. **seq**uence, dowolne różnice).

```
c(-3:2, 4:0)
```

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

```
seq(1, 8, by = 2)
```

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

```
seq(1, 8, length.out = 6)
```

```
## [1] 1.0 2.4 3.8 5.2 6.6 8.0
```

1.4.3 Wektory napisów

- Ciągi dowolnych znaków drukowanych, zwane napisami, tworzymy wykorzystując apostrofy lub cudzysłów.

```
c("DSTA LIO", "informatyka", ",", "statystyka", "!")
```

```
## [1] "DSTA LIO"    "informatyka" ","           "statystyka"  "!"
```

```
length(c("DSTA LIO", "informatyka", ",", "statystyka", "!"))
```

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

1.4.4 Nazywanie obiektów

- W R obiekty nazywamy za pomocą jednego z następujących operatorów przypisania (ang. assignment operator):
  - `=`
  - `<-` (w RStudio skrót klawiszowy `ALT+-`)
  - `->`

```
x = 5
```

```
5 = x
```

```
## Error in 5 = x : invalid (do_set) left-hand side to assignment
```

```
y <- 6
```

```
6 -> y
```

```
x
```

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

```
y
```

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

- Wielu użytkowników programu R nie zaleca stosowania operatora `=` , ponieważ ma on również inne znaczenia, np. używa się go do ustalania wartości funkcji.
- Lepiej nie używać (poza ewentualnie komentarzami) polskich znaków diakrytycznych.

- Polecenie `ls()` podaje wszystkie aktualnie istniejące obiekty.
- Usunąć jakiś obiekt możemy za pomocą funkcji `rm()` .
- Wszystkie obiekty usuwamy poleceniem `rm(list = ls())` .

```
x <- 1:2
```

```
y <- list(1, 2)
```

```
ls()
```

```
## [1] "x" "y"
```

```
rm(x)
```

```
ls()
```

```
## [1] "y"
```

```
rm(list = ls())
ls()
```

```
## character(0)
```

1.5 Operatory arytmetyczne

- Do działania na wektorach liczbowych (czasem również zespolonych) można używać następujących binarnych operatorów arytmetycznych:
  - `+` (dodawanie),
  - `-` (odejmowanie),
  - `*` (mnożenie),
  - `/` (dzielenie rzeczywiste),
  - `^` (potęgowanie),
  - `%%` (reszta z dzielenie (modulo)),
  - `%/%` (dzielenie całkowite (bez reszty)).
- Operatory arytmetyczne są zwektoryzowane (ang. vectorized), tzn. dla wektorów $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ i $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ o tej samej długości n w wyniku działania $\mathbf{x} \diamondsuit \mathbf{y}$ uzyskujemy wektor

$$\mathbf{w} = (x_1 \diamondsuit y_1, x_2 \diamondsuit y_2, \ldots, x_n \diamondsuit y_n).$$

  Czyli operacje tego typu wykonywane są element po elemencie (ang. elementwise). Unikamy w tej sposób ,,jawnej" pętli (pętla jest ,,ukryta" w kodzie operatora), co może pozwolić na przyśpieszenie obliczeń.

```
7 %% 3
```

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

```
1:3 + c(3, 4, 5)
```

```
## [1] 4 6 8
```

- W przypadku, gdy wektory będące argumentami operatorów binarnych są różnej długości, stosowana jest tak zwana reguła zawijania (ang. recycling rule). Powiela ona niejako krótszy wektor tak, aby uzgodnić jego długość dłuższym wektorem. Niech $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ i $\mathbf{y} = (y_1, y_2, \ldots, y_m)$, gdzie bez straty ogólności $m \geqslant n$. Wtedy wynikiem działania jest m-elementowy wektor postaci (dla odpowiedniego l)

$$\mathbf{x} \diamondsuit \mathbf{y} = (x_1 \diamondsuit y_1, \ldots, x_n \diamondsuit y_n, x_1 \diamondsuit y_{n+1}, x_2 \diamondsuit y_{n+2}, \ldots, x_l \diamondsuit y_m).$$

```
x <- c(1, 3, 5, 8, 1, 3, 0, 6)
x * c(1, 3)
```

```
## [1]  1  9  5 24  1  9  0 18
```

```
x <- c(1, 3, 5, 8, 1, 3, 0)
x * c(1, 3)
```

```
## Warning in x * c(1, 3): długość dłuszego obiektu nie jest wielokrotnością
## długości krótszego obiektu
```

```
## [1]  1  9  5 24  1  9  0
```

1.6 Operatory logiczne i relacyjne

- Rozważamy następujące operatory i funkcje logiczne:
  - `!x` (negacja),
  - `x | y` (alternatywa),
  - `x & y` (koniunkcja).
- Do porównywania wektorów służą następujące operatory relacyjne:
  - `x < y` (czy mniejsze?),
  - `x > y` (czy większe?),
  - `x <= y` (czy nie większy?),

    - `x >= y` (czy nie mniejszy?),
    - `x == y` (czy równy?),
    - `x != y` (czy nierówny?).
- Można je stosować na wektorach dowolnych typów. Jednak wynikiem ich działania jest zawsze wektor logiczny.

```
(1:7) == (7:1)
```

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

```
c(TRUE, FALSE) < 1
```

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

1.7 Indeksowanie wektorów

- Wartości elementów każdego wektora leżą na ściśle określonych pozycjach oznaczonych kolejnymi liczbami naturalnymi (`1:length(x)`).
- Do elementów wektora odwołujemy się poprzez nawiasy kwadratowe `[]` .

```
x <- 1:5
x[2]
```

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

```
x[2:4]
```

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

```
x[-2]
```

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

```
x[-(2:4)]
```

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

```
# x[c(1, -2)]
## Error in x[c(1, -2)] : only 0's may be mixed with negative subscripts
x[c(TRUE, TRUE, FALSE, TRUE, FALSE)]
```

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

```
x[x < 4]
```

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

- Nawiasów kwadratowych możemy również użyć do zmiany elementów danego wektora.

```
x[2] <- 6
x
```

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

```
x[c(2, 4)] <- c(4, 2)
x
```

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

```r
x[c(2, 4)] <- 6
x
```

```
## [1] 1 6 3 6 5
```

1.8 Wybrane funkcje wbudowane

- Program R zawiera wiele zwektoryzowanych funkcji matematycznych dla wektorów liczbowych lub czasem zespolonych. W wyniku ich działania uzyskujemy wektor tej samej długości co wektor wejściowy, którego wartości są wyznaczane przez przekształcenie każdego elementu daną funkcją.
- `abs()` - wartość bezwzględna, `sign()` - znak liczby, `floor()` - funkcja „podłoga", `ceiling()` - funkcja „sufit", `trunc()` - obcięcie części ułamkowej liczby ($\lfloor x \rfloor$ dla $x \geqslant 0$, $\lceil x \rceil$ dla $x < 0$), `round(x, digits = 0)` - zaokrąglenie `x` do `digits` miejsc po kropce dziesiętnej, `sqrt()` - pierwiastek kwadratowy, `exp()` - funkcja wykładnicza, `log(x, base = exp(1))` - logarytm o podstawie `base`, `gamma()` - funkcja gamma, `beta(a, y)` - funkcja beta, `choose(n, k)` - współczynnik dwumianowy, `sin()`, `cos()`, `tan()` - funkcje trygonometryczne, `asin()`, `acos()`, `atan()` - funkcje cyklometryczne, `Conj(z)` - liczba sprzężona do `z`, `Re(z)` - część rzeczywista `z`, `Im(z)` - część urojona `z`, `Mod(z)` - moduł `z`, `Arg(z)` - argument `z`.

```r
abs(-2:2)
```

```
## [1] 2 1 0 1 2
```

```r
ceiling(c(-1.99, -0.1, 0.1, 1.99))
```

```
## [1] -1  0  1  2
```

1.9 Zadania

Zadanie 1. Otwórz program RStudio. Następnie utwórz nowy skrypt i zapisz go jako, na przykład, `wprowadzenie_do_R_zadania.R` . W tym skrypcie możesz napisać rozwiązania następujących zadań.

Zadanie 2. Użyj funkcji `rep()` , aby utworzyć wektor logiczny, zaczynając od trzech wartości prawda, następnie czterech wartości fałsz, po których następują dwie wartości prawda i wreszcie pięć wartości fałsz. Przypisz ten wektor logiczny do zmiennej `x` . Na koniec przekonwertuj ten wektor na wektor numeryczny. Jak zmieniły się wartości prawda i fałsz?

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

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

Zadanie 3. Palindromem nazywamy wektor, którego elementy czytane od końca tworzą ten sam wektor co elementy czytane od początku. Utwórz taki wektor 100 liczb przy czym pierwsze 20 liczb to kolejne liczby naturalne, następnie występuje 10 zer, następnie 20 kolejnych liczb parzystych, a pozostałe elementy określone są przez palindromiczność (warunek symetrii).

```
##   [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  0  0  0
##  [24]  0  0  0  0  0  0  0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32
##  [47] 34 36 38 40 40 38 36 34 32 30 28 26 24 22 20 18 16 14 12 10  8  6  4
##  [70]  2  0  0  0  0  0  0  0  0  0  0 20 19 18 17 16 15 14 13 12 11 10  9
##  [93]  8  7  6  5  4  3  2  1
```

Zadanie 4. Z wektora `letters` wybierz litery na pozycjach 5, 10, 15, 20, 25.

```
## [1] "e" "j" "o" "t" "y"
```

Zadanie 5. Utwórz wektor liczb naturalnych od 1 do 1000, a następnie zamień liczby parzyste na ich odwrotności.

```
## [1] 1 0.5 3 0.25 5 0.1666667 ...
```

Zadanie 6. Uporządkuj elementy wektora $(6, 3, 4, 5, 2, 3)$ od największego do najmniejszego wykorzystując funkcję `order()` .

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

Zadanie 7. Wyznacz znaki elementów wektora $(-1{,}876; -1{,}123; -0{,}123; 0; 0{,}123; 1{,}123; 1{,}876)$. Następnie zaokrąglij elementy tego wektora do dwóch miejsc po przecinku. Na koniec wyznacz część całkowitą każdego elementu nowego wektora.

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

```
## [1] -1.88 -1.12 -0.12  0.00  0.12  1.12  1.88
```

```
## [1] -2 -2 -1  0  0  1  1
```

Zadanie 8. Wyznacz pierwiastek kwadratowy z każdej liczby naturalnej od 1 do 100 milionów. Najpierw wykonaj to polecenie korzystając z odpowiedniej funkcji wbudowanej w R, a następnie wykorzystując potęgowanie. Który sposób działa szybciej? **Wskazówka:** Do badania długości czasu działania programu można wykorzystać funkcję `Sys.time()`.

```
## Time difference of 1.485525 secs
## Time difference of 9.706759 secs
## [1] 1 1.414214 1.732051 2 2.236068 2.44949 ...
```

Zadanie 9. W pakiecie `schoolmath` znajduje się zbiór danych `primlist`, który zawiera liczby pierwsze pomiędzy 1 a 9999999.

- Znajdź największą liczbę pierwszą mniejszą od 1000.
- Ile jest liczb pierwszych większych od 100 a mniejszych od 500?

```
## [1] 997
```

```
## [1] 73
```

Zadanie 10. Wyznacz wszystkie kombinacje wartości wektorów (a, b) i $(1, 2, 3)$ za pomocą funkcji `rep()` i `paste()`.

```
## [1] "a1" "a2" "a3" "b1" "b2" "b3"
```

Zadanie 11. Utwórz wektor 30 napisów następującej postaci: `liczba.litera` , gdzie liczba to kolejne liczby naturalne od 1 do 30 a litera to trzy wielkie litery `X` , `Y` , `Z` występujące cyklicznie.

```
##  [1] "1.X"  "2.Y"  "3.Z"  "4.X"  "5.Y"  "6.Z"  "7.X"  "8.Y"  "9.Z"  "10.X"
## [11] "11.Y" "12.Z" "13.X" "14.Y" "15.Z" "16.X" "17.Y" "18.Z" "19.X" "20.Y"
## [21] "21.Z" "22.X" "23.Y" "24.Z" "25.X" "26.Y" "27.Z" "28.X" "29.Y" "30.Z"
```

Zadanie 12. W pewnych sytuacjach przydatna może się okazać tzw. kategoryzacja zmiennych, czyli inny podział na kategorie niżby wynikał z danych. Wygeneruj 100 obserwacji, które są odpowiedziami na pytania ankiety, każda odpowiedź może przyjąć jedną z wartości: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’. Dokonaj kategoryzacji w taki sposób, aby kategoria 1 obejmowała odpowiedzi ‘a’ i ‘b’, 2 odpowiedzi ‘c’ i ‘d’ oraz 3 odpowiedź ‘e’. **Wskazówka:** Wykorzystaj funkcję `sample()` oraz funkcję `recode()` z pakietu `car` .

```
## Loading required package: carData
```

```
##   [1] "d" "b" "e" "d" "a" "e" "d" "b" "b" "d" "d" "d" "e" "d" "c" "d" "e"
##  [18] "b" "e" "b" "c" "d" "d" "c" "a" "c" "d" "b" "c" "b" "e" "a" "c" "a"
##  [35] "e" "a" "a" "b" "a" "c" "b" "c" "a" "c" "a" "b" "e" "a" "c" "c" "b"
##  [52] "e" "b" "d" "d" "a" "e" "c" "e" "c" "d" "d" "a" "d" "d" "c" "a" "d"
##  [69] "a" "b" "e" "e" "e" "a" "b" "b" "b" "e" "c" "d" "d" "c" "b" "d" "e"
##  [86] "b" "a" "c" "c" "a" "e" "a" "e" "a" "b" "c" "c" "e" "d" "c"
```

```
##   [1] 2 1 3 2 1 3 2 1 1 2 2 2 3 2 2 2 3 1 3 1 2 2 2 2 1 2 2 1 2 1 3 1 2 1 3
##  [36] 1 1 1 1 2 1 2 1 2 1 1 3 1 2 2 1 3 1 2 2 1 3 2 3 2 2 2 1 2 2 2 1 2 1 1
##  [71] 3 3 3 1 1 1 1 3 2 2 2 2 1 2 3 1 1 2 2 1 3 1 3 1 1 2 2 3 2 2
```