

SPRAWOZDANIE

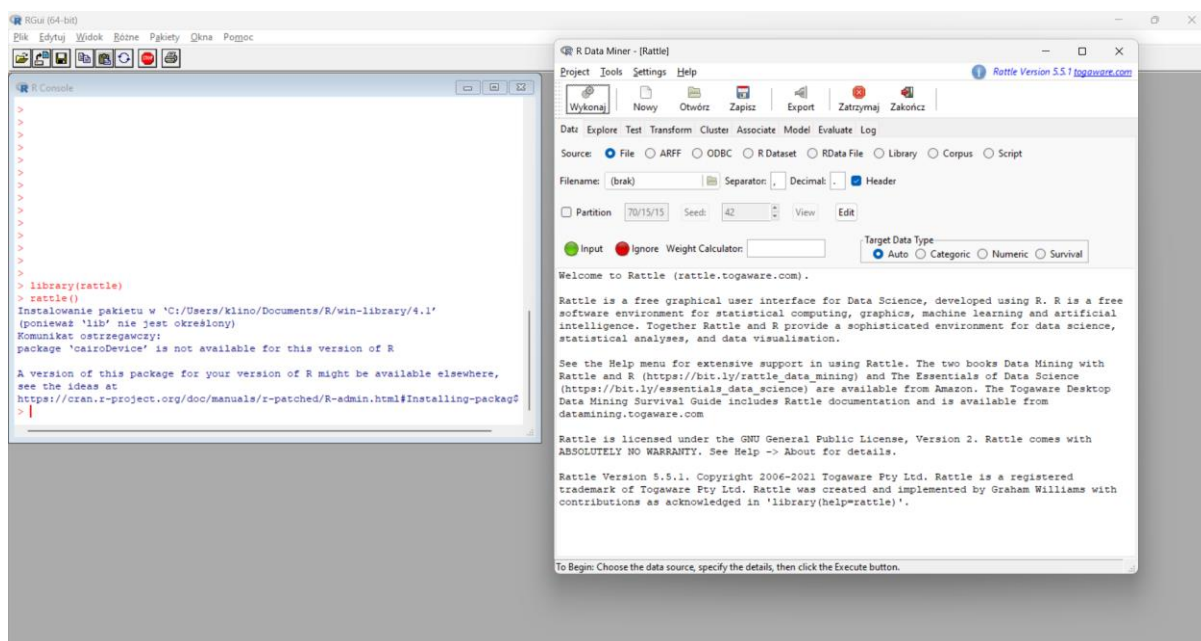
Zajęcia: Zbiory Big Data i Eksploracja Danych

Prowadząca: dr inż. Ruslana Ziubina

Laboratorium nr 1 Data rozpoczęcia: 13.10.2023 Temat: Eksploracja danych – podstawy języka R	Rafał Klinowski Informatyka II stopień, stacjonarne, Semestr 2, gr. a
---	--

Ćw. 1.

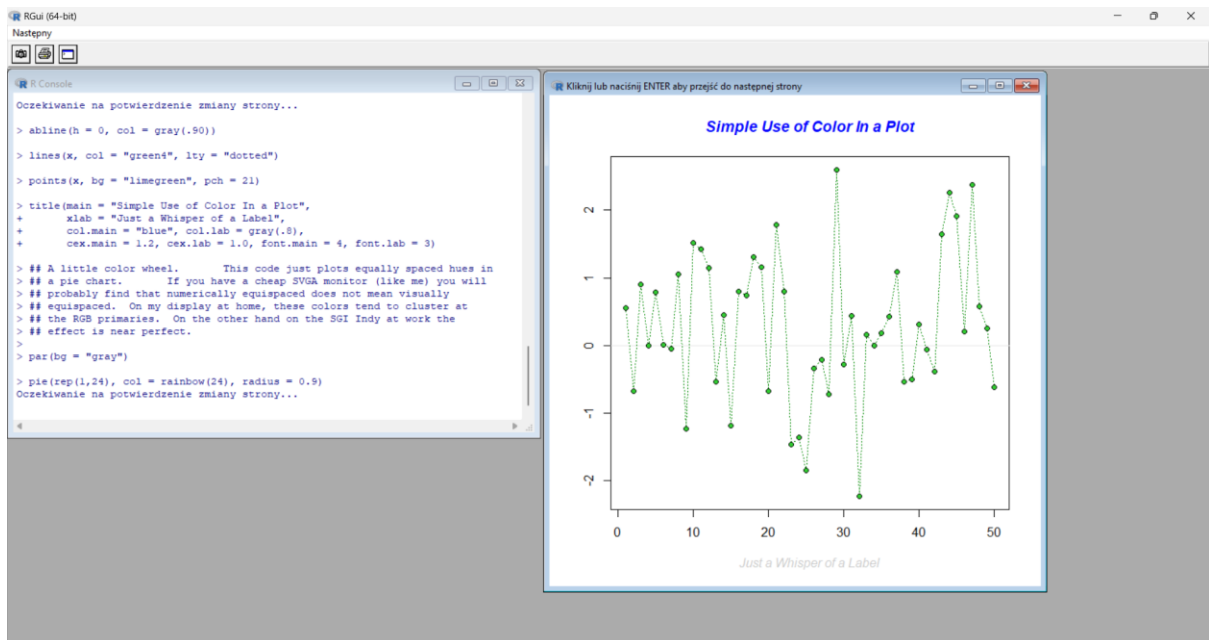
W celu zainstalowania Rattle konieczne było pobranie R w wersji 4.1.3 – w przypadku nowszych wersji występowały problemy z zależnością RGtk2. Po zainstalowaniu Rattle oraz RGtk2 uruchomiono środowisko.



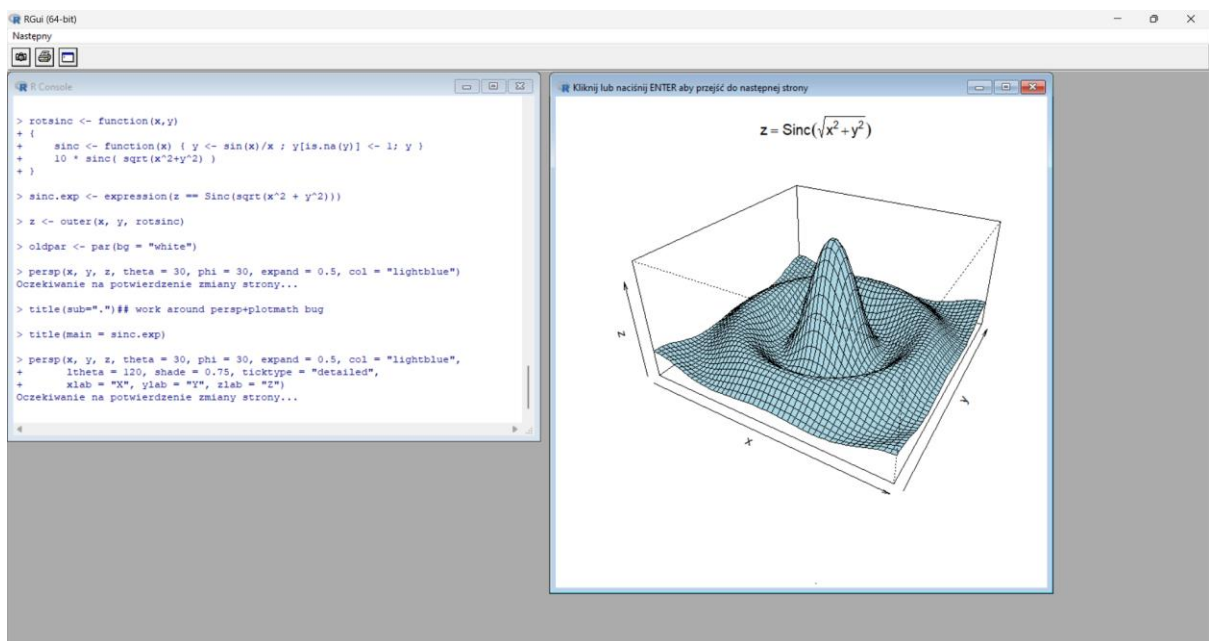
Rysunek 1. Uruchomione R i Rattle.

Ćw. 2.

Testowanie niektórych pakietów. W ramach tego ćwiczenia uruchomiono niektóre przykłady z pakietów „graphics” i „persp”.



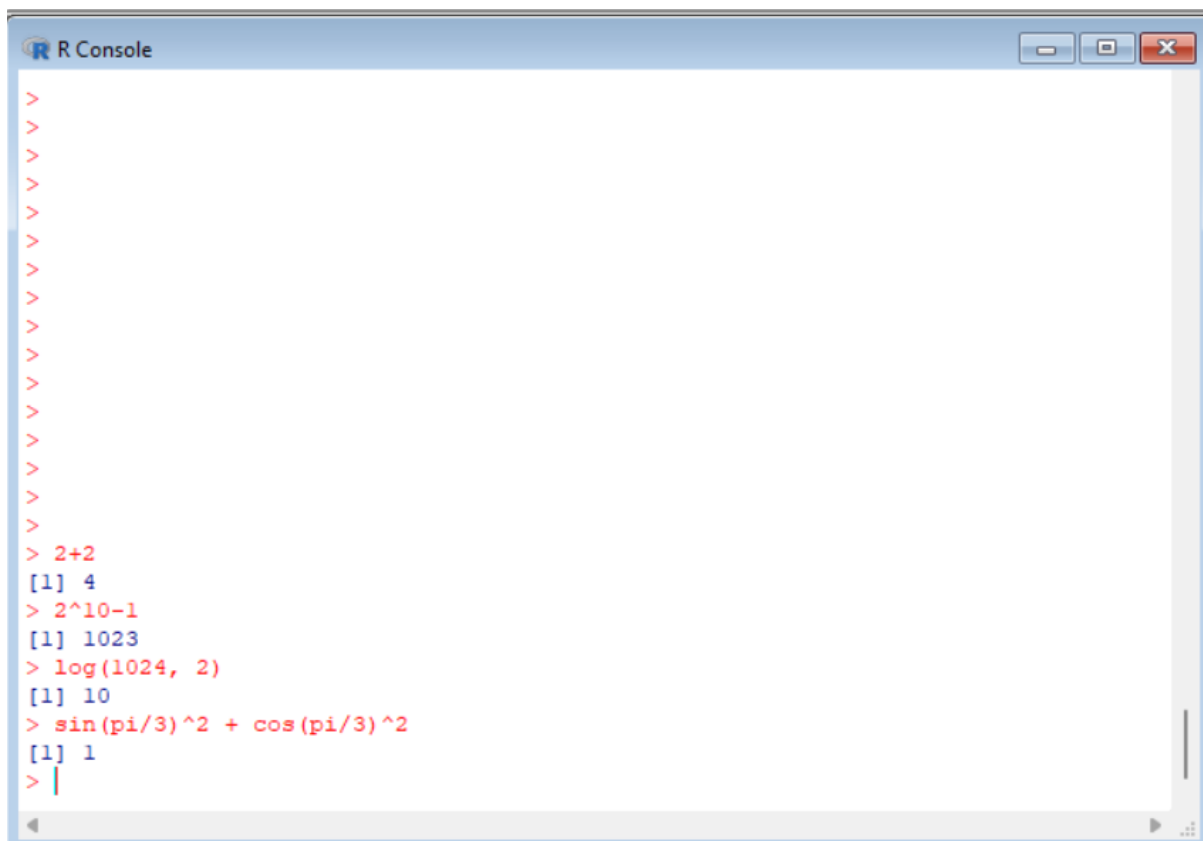
Rysunek 2. Wynik działania komendy `demo(graphics)`.



Rysunek 3. Wynik działania komendy `demo(persp)`.

Ćw. 3.

Używanie R jako kalkulatora do wykonywania pewnych operacji matematycznych.

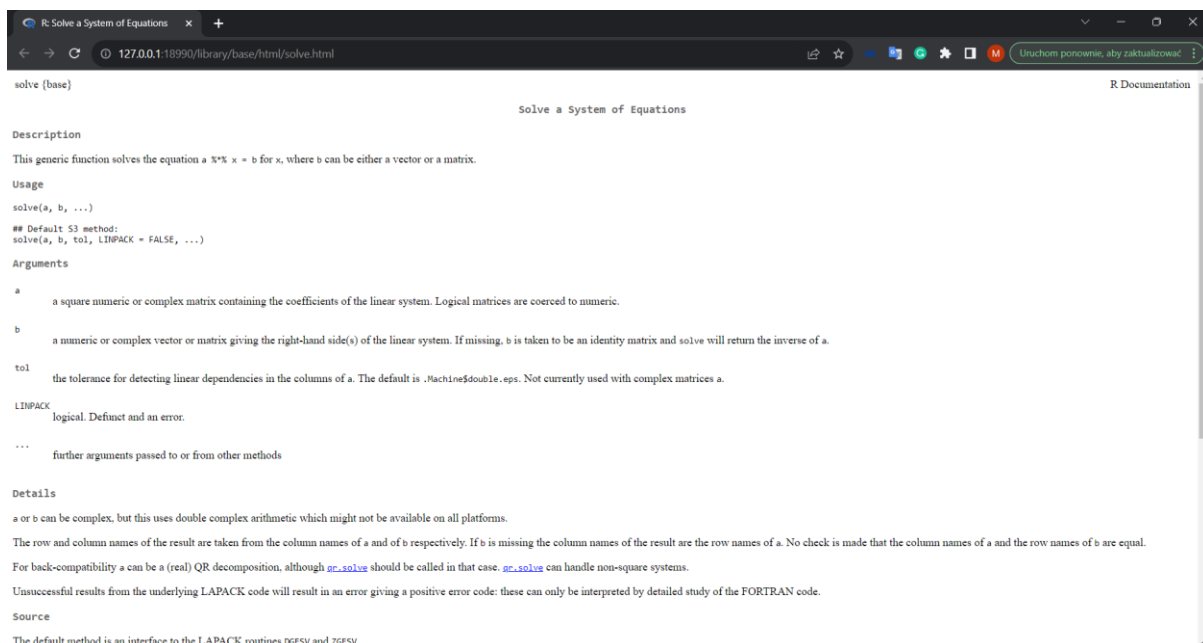


```
>
>
>
>
>
>
>
>
>
>
>
>
>
> 2+2
[1] 4
> 2^10-1
[1] 1023
> log(1024, 2)
[1] 10
> sin(pi/3)^2 + cos(pi/3)^2
[1] 1
> |
```

Rysunek 4. Pewne działania matematyczne w konsoli R.

Ćw. 4.

Uzyskiwanie pomocy dotyczącej pakietów R – w przeglądarce lub wewnątrz konsoli.



R: Solve a System of Equations

127.0.0.1:8990/library/base/html/solve.html

Unruhom ponownie, aby zaktualizować

solve (base)

Solve a System of Equations

R Documentation

Description

This generic function solves the equation $Ax = b$ for x , where b can be either a vector or a matrix.

Usage

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

Default S3 method:
solve(a, b, tol, LINPACK = FALSE, ...)

Arguments

a
a square numeric or complex matrix containing the coefficients of the linear system. Logical matrices are coerced to numeric.

b
a numeric or complex vector or matrix giving the right-hand side(s) of the linear system. If missing, b is taken to be an identity matrix and $solve$ will return the inverse of a .

tol
the tolerance for detecting linear dependencies in the columns of a . The default is `.Machine$double.eps`. Not currently used with complex matrices a .

LINPACK
logical. Default and an error:

...
further arguments passed to or from other methods

Details

a or b can be complex, but this uses double complex arithmetic which might not be available on all platforms.

The row and column names of the result are taken from the column names of a and of b respectively. If b is missing the column names of the result are the row names of a . No check is made that the column names of a and the row names of b are equal.

For back-compatibility a can be a (real) QR decomposition, although `qr.solve` should be called in that case. `qr.solve` can handle non-square systems.

Unsuccessful results from the underlying LAPACK code will result in an error giving a positive error code: these can only be interpreted by detailed study of the FORTRAN code.

Source

The default method is an interface to the LAPACK routines `DGESV` and `ZGESV`.

Rysunek 5. Interfejs pomocy w przeglądarce internetowej po wpisaniu komendy `help(solve)`.

```
R Console

solve> sh8 <- solve(h8)

solve> round(sh8 %*% h8, 3)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    0    0    0    0    0    0    0
[2,]    0    1    0    0    0    0    0    0
[3,]    0    0    1    0    0    0    0    0
[4,]    0    0    0    1    0    0    0    0
[5,]    0    0    0    0    1    0    0    0
[6,]    0    0    0    0    0    1    0    0
[7,]    0    0    0    0    0    0    1    0
[8,]    0    0    0    0    0    0    0    1

solve> A <- hilbert(4)

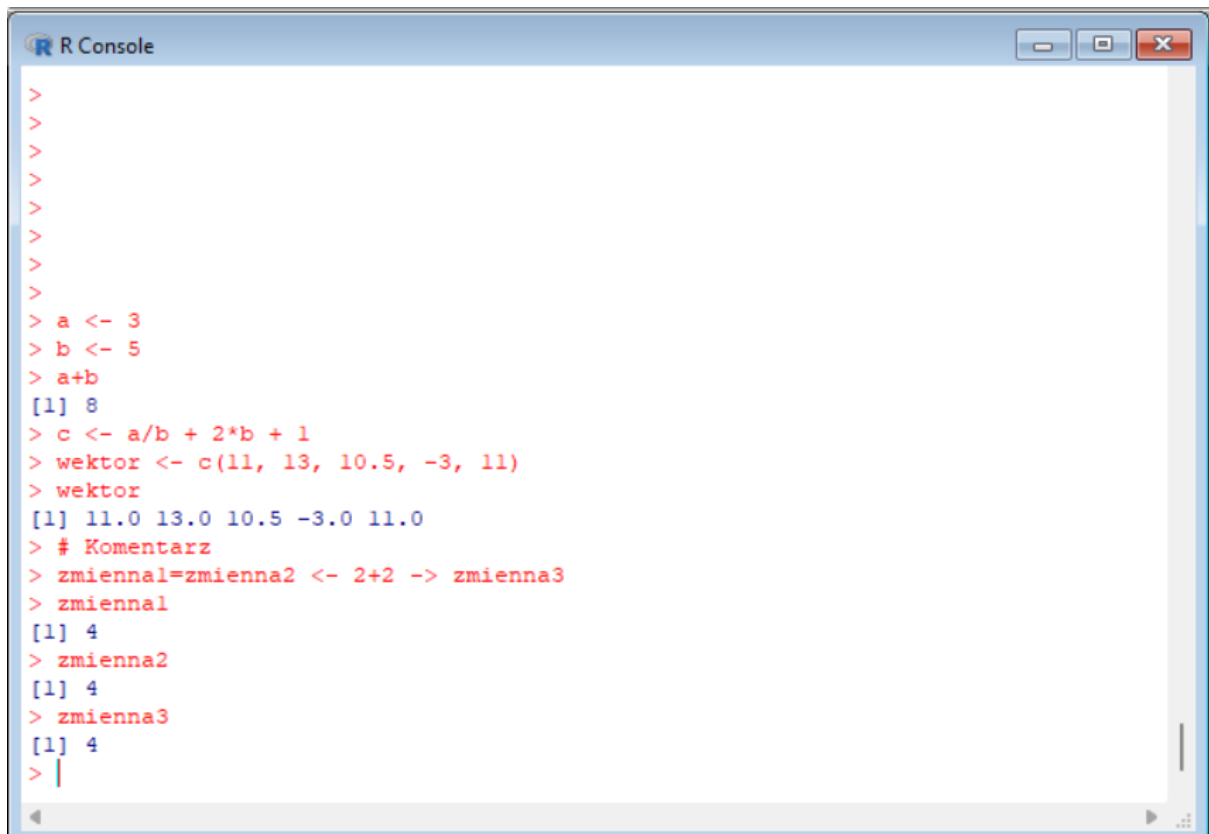
solve> A[] <- as.complex(A)

solve> ## might not be supported on all platforms
solve> try(solve(A))
      [,1]      [,2]      [,3]      [,4]
[1,] 16+0i -120+0i  240+0i -140+0i
[2,] -120+0i 1200+0i -2700+0i 1680+0i
[3,]  240+0i -2700+0i 6480+0i -4200+0i
[4,] -140+0i 1680+0i -4200+0i 2800+0i
> |
```

Rysunek 6. Przykłady użycia biblioteki po wpisaniu komendy `example(solve)`.

Ćw. 5.

Definiowanie i wykorzystywanie zmiennych oraz komentarze.



```
>
>
>
>
>
>
>
> a <- 3
> b <- 5
> a+b
[1] 8
> c <- a/b + 2*b + 1
> wektor <- c(11, 13, 10.5, -3, 11)
> wektor
[1] 11.0 13.0 10.5 -3.0 11.0
> # Komentarz
> zmienna1=zmienna2 <- 2+2 -> zmienna3
> zmienna1
[1] 4
> zmienna2
[1] 4
> zmienna3
[1] 4
> |
```

Rysunek 7. Wynik w konsoli po wykonaniu ciągu instrukcji z instrukcji laboratoryjnej.

Ćw. 6.

Obiekty w przestrzeni roboczej R oraz atrybuty.

```

R Console
>
> library(rattle)
> objects()
[1] "a"      "A"      "b"      "c"      "h8"     "hilbert"
[7] "oldpar" "opar"   "rotsinc" "sh8"    "sinc.exp" "wektor"
[13] "x"      "y"      "z"      "zmienna1" "zmienna2" "zmienna3"
> attributes(weather)
$class
[1] "tbl_df"      "tbl"        "data.frame"

$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 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
[163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
[181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
[199] 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
[217] 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
[235] 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
[253] 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
[271] 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
[289] 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
[307] 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
[325] 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
[343] 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
[361] 361 362 363 364 365 366

$names
 [1] "Date"      "Location"   "MinTemp"    "MaxTemp"
 [5] "Rainfall"  "Evaporation" "Sunshine"    "WindGustDir"
 [9] "WindGustSpeed" "WindDir9am" "WindDir3pm"  "WindSpeed9am"
[13] "WindSpeed3pm" "Humidity9am" "Humidity3pm" "Pressure9am"
[17] "Pressure3pm"  "Cloud9am"    "Cloud3pm"    "Temp9am"
[21] "Temp3pm"     "RainToday"   "RISK_MM"     "RainTomorrow"

```

Rysunek 8. Podgląd obiektów w przestrzeni R oraz atrybutów.

```

> x <- 1:6
> x
[1] 1 2 3 4 5 6
> rm(x)
> x
BŁĄD: nie znaleziono obiektu 'x'

```

Rysunek 9. Sposób usunięcia obiektu z pamięci.

Ćw. 7.

Usuwanie obiektów z pamięci R.

```

> objects()
[1] "a"      "A"      "b"      "c"      "h8"      "hilbert"
[7] "oldpar" "opar"   "rotsinc" "sh8"    "sinc.exp" "wektor"
[13] "y"      "z"      "zmienna1" "zmienna2" "zmienna3"
> rm(wektor, a, b)
> wektor
BŁĄD: nie znaleziono obiektu 'wektor'

```

Rysunek 10. Wyświetlenie oraz usunięcie niektórych obiektów z pamięci. Usunięte obiekty nie mogą być już wykorzystywane (np. wyświetlone).

Ćw. 8.

Operator przypisania, typy zmiennych.

```

<
> my_apples <- 5
> my_apples
[1] 5
> my_oranges <- 19
> my_fruit <- my_apples + my_oranges
> my_fruit
[1] 24
>
>
> my_numeric = 42
> my_numeric
[1] 42
> my_numeric <- "forty-two"
> my_numeric
[1] "forty-two"
> my_logical <- FALSE
> my_logical
[1] FALSE
> |

```

Rysunek 11. Wykonanie pewnych prostych operacji na zmiennych, oraz zapisywanie wartości różnego typu do zmiennych.

Ćw. 9.

Wykorzystanie funkcji print do wypisywania informacji na ekran.

```

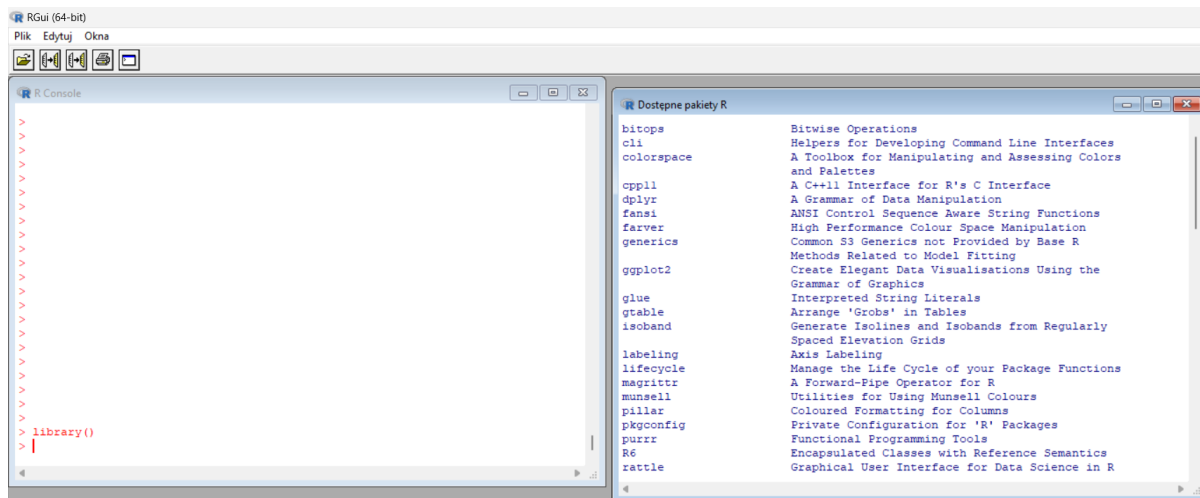
> wynik <- sin(pi/6)
> print(wynik)
[1] 0.5
>
> print(sin(60*pi/180), digits=11)
[1] 0.86602540378
> print("Napis w cudzysłowie", quote=TRUE)
[1] "Napis w cudzysłowie"
> print("Napis bez cudzysłowów", quote=FALSE)
[1] Napis bez cudzysłowów
> print(c(1,3,NA,5,3,NA,NA,14), na.print="Brak wartości")
[1] 1 3 Brak wartości 5 3
[6] Brak wartości Brak wartości 14

```

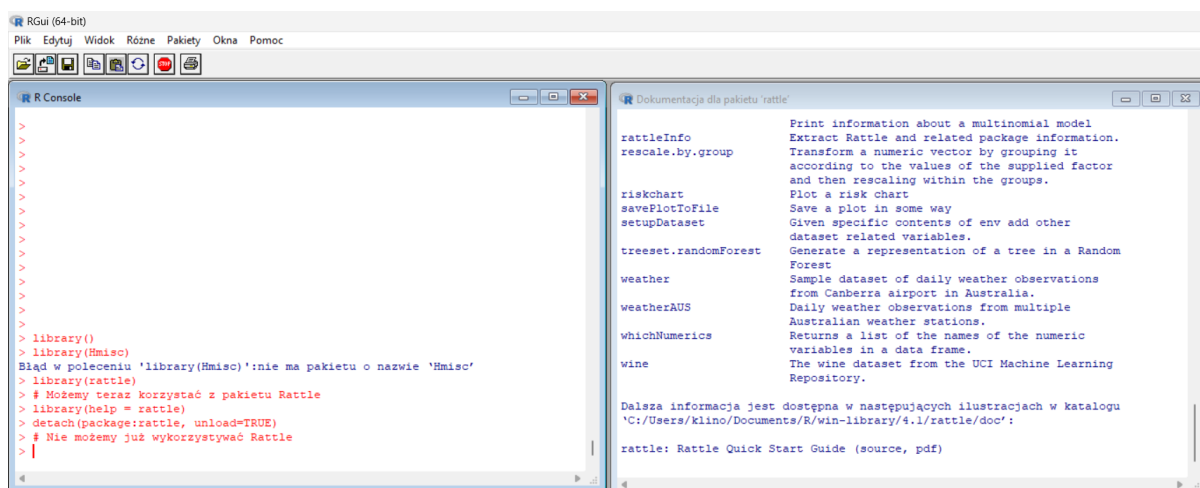
Rysunek 12. Wyniki wykorzystania funkcji print() w kilku różnych przypadkach i z dodatkowymi argumentami.

Ćw. 10.

Pakiety w R i ich załadowanie.



Rysunek 13. Wynik działania polecenia `library()`.



Rysunek 14. Ładowni pakietu, uzyskanie pomocy związanej z konkretnym pakietem i usuwanie pakietu z pamięci.

Ćw. 11.

Liczby, NaN, nieskończoności w wyniku niektórych operacji.

```
> a <- log(-3)
Komunikat ostrzegawczy:
W poleceniu 'log(-3)': wyprodukowano wartości NaN
> a
[1] NaN
> 1/0
[1] Inf
> -1/0
[1] -Inf
```

Rysunek 15. Uzyskane wartości NaN oraz +/- nieskończoność przy podaniu nieprawidłowych argumentów do operacji matematycznych.

Ćw. 12.

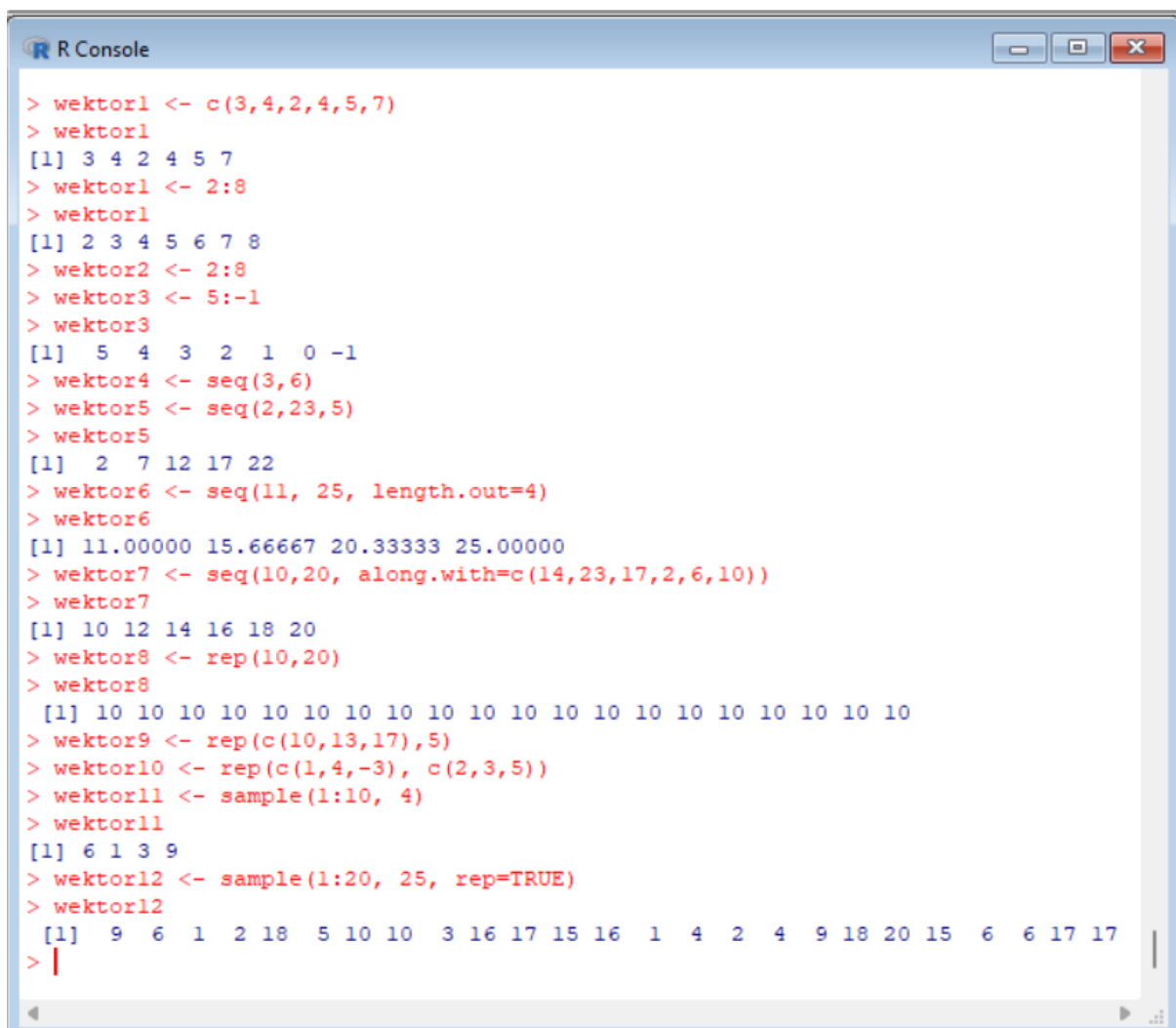
łańcuchy znaków.

```
> a <- "Dziś jest piątek"
> a
[1] "Dziś jest piątek"
> cat(a)
Dziś jest piątek>
> paste("Witaj ", " świecie")
[1] "Witaj  świecie"
```

Rysunek 16. Wypisanie łańcucha znaków z formatowaniem i bez formatowania, skorzystanie z funkcji paste() łączącej łańcuchy.

Ćw. 13.

Wektory w R.



```
R Console
> wektor1 <- c(3,4,2,4,5,7)
> wektor1
[1] 3 4 2 4 5 7
> wektor1 <- 2:8
> wektor1
[1] 2 3 4 5 6 7 8
> wektor2 <- 2:8
> wektor3 <- 5:-1
> wektor3
[1] 5 4 3 2 1 0 -1
> wektor4 <- seq(3,6)
> wektor5 <- seq(2,23,5)
> wektor5
[1] 2 7 12 17 22
> wektor6 <- seq(11, 25, length.out=4)
> wektor6
[1] 11.00000 15.66667 20.33333 25.00000
> wektor7 <- seq(10,20, along.with=c(14,23,17,2,6,10))
> wektor7
[1] 10 12 14 16 18 20
> wektor8 <- rep(10,20)
> wektor8
[1] 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
> wektor9 <- rep(c(10,13,17),5)
> wektor10 <- rep(c(1,4,-3), c(2,3,5))
> wektor11 <- sample(1:10, 4)
> wektor11
[1] 6 1 3 9
> wektor12 <- sample(1:20, 25, rep=TRUE)
> wektor12
[1] 9 6 1 2 18 5 10 10 3 16 17 15 16 1 4 2 4 9 18 20 15 6 6 17 17
> |
```

Rysunek 17. Sposoby tworzenia wektorów w R – skorzystanie z funkcji: c, seq, rep, sample – wraz z przykładowymi wartościami.

- c() – pozwala tworzyć wektor z podanych wartości, w tym innych wektorów

- seq() – pozwala tworzyć wektor z liczb w danym przedziale od-do, opcjonalnie z danym krokiem
- rep() – pozwala tworzyć wektor z danej ilości powtórzeń danego elementu
- sample() – pozwala tworzyć wektor liczb losowych z danego przedziału, z powtórzeniami lub bez

```
> length(wektor11)
[1] 4
```

Rysunek 18. Uzyskanie długości wektora.

```
> numeric_vector <- c(1,10,49)
> character_vector <- c("a","b","c")
> boolean_vector <- c(TRUE,FALSE,TRUE,FALSE)
> mixed_vector <- c(1, "X")
> mixed_vector
[1] "1" "X"
```

Rysunek 19. Utworzenie wektorów liczb, znaków oraz wartości logicznych. Typów nie można mieszać – w przypadku wektora mixed_vector, liczba została przekonwertowana na tekst.

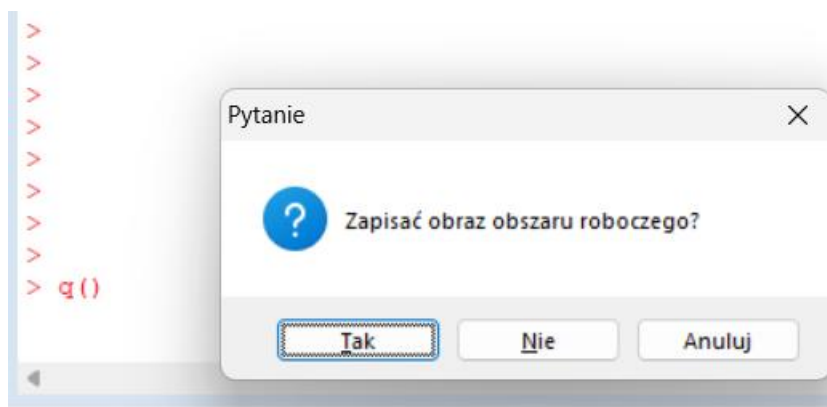
=

```
> wektor <- c(2:8)
> wektor^2
[1] 4 9 16 25 36 49 64
> 1 / wektor
[1] 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
> wektor - 2
[1] 0 1 2 3 4 5 6
> wektor
[1] 2 3 4 5 6 7 8
> c(wektor, 0, 3:5, wektor)
[1] 2 3 4 5 6 7 8 0 3 4 5 2 3 4 5 6 7 8
> wektor[1]
[1] 2
> wektor [-1]
[1] 3 4 5 6 7 8
> wektor[wektor>0]
[1] 2 3 4 5 6 7 8
```

Rysunek 20. Operacje na wektorach. Wypisywanie danych elementów wektora.

Ćw. 14.

Wyjście ze środowiska R – możliwe zapisanie danych z pamięci.



Rysunek 21. Próba wyjścia z R – pytanie o zapisanie obszaru roboczego (obiektów w pamięci).

Ćw. 15.

Tworzenie małych zbiorów danych. Zbiory zostały utworzone z wykorzystaniem funkcji `c()`.

```

> S.win <- sum(Wingcrd)
> S.win
[1] 440.5
> sum(Head)
BŁĄD: nie znaleziono obiektu 'Head'
> Z <- cbind(Wingcrd, Tarsus, Wt)
> Z
      Wingcrd Tarsus   Wt
[1,]    59.0   22.3  9.5
[2,]    55.0   19.7 13.8
[3,]    53.5   20.8 14.8
[4,]    55.0   20.3 15.2
[5,]    52.5   20.8 15.5
[6,]    57.5   21.5 15.6
[7,]    53.0   20.6 15.6
[8,]    55.0   21.5 15.7
> Z[,1]
[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0
> Z[1:8, 1]
[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0
> Z[2, ]
      Wingcrd   Tarsus      Wt
      55.0      19.7      13.8
> Z[2, 1:4]
Błąd w poleceniu 'Z[2, 1:4]':indeks jest poza granicami
> Z[2, 1:3]
      Wingcrd   Tarsus      Wt
      55.0      19.7      13.8
> dim(Z)
[1] 8 3
> Z2 <- rbind(Wingcrd, Tarsus, Wt)
> Z2
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
Wingcrd 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0
Tarsus  22.3 19.7 20.8 20.3 20.8 21.5 20.6 21.5
Wt       9.5 13.8 14.8 15.2 15.5 15.6 15.6 15.7

```

Rysunek 22. Operacje na małym zbiorze danych – utworzenie macierzy, uzyskiwanie niektórych kolumn lub wierszy, sumowanie elementów, itp.

Ćw. 16.

Typ danych „factor” - czynniki.

```

> wzrost <- rep(c("niski", "średni", "wysoki"), c(4,3,5))
> wzrost
[1] "niski" "niski" "niski" "niski" "średni" "średni" "średni" "wysoki"
[9] "wysoki" "wysoki" "wysoki" "wysoki"
> factor_wzrost <- factor(wzrost
+ )
> factor_wzrost
[1] niski niski niski niski średni średni średni wysoki wysoki wysoki
[11] wysoki wysoki
Levels: niski średni wysoki

```

Rysunek 23. Konwersja wektora na factor.

```

> levels(factor_wzrost)
[1] "niski" "średni" "wysoki"
> table(factor_wzrost)
factor_wzrost
niski średni wysoki
      4      3      5
> summary(factor_wzrost)
niski średni wysoki
      4      3      5

```

Rysunek 24. Operacje na typie factor.

```

> as.vector(factor_wzrost)
[1] "niski" "niski" "niski" "niski" "średni" "średni" "średni" "wysoki"
[9] "wysoki" "wysoki" "wysoki" "wysoki"
> as.integer(factor_wzrost)
[1] 1 1 1 1 2 2 2 3 3 3 3 3

```

Rysunek 25. Konwersja odwrotna – z typu factor na wektor lub liczby.

```

> factor2 <- factor(c(2,3,2,2,3,4), levels=1:5)
> factor2
[1] 2 3 2 2 3 4
Levels: 1 2 3 4 5
> x <- factor2 + 2
Komunikat ostrzegawczy:
W poleceniu 'Ops.factor(factor2, 2)': '+' nie ma sensu dla czynników
> factor2[3] <- "nijaki"
Komunikat ostrzegawczy:
W poleceniu '['[<-factor`(`*tmp*`, 3, value = "nijaki")':
niepoprawny poziom czynnika, wygenerowano wartość NA

```

Rysunek 26. Operacje arytmetyczne czy próba podstawienia pod zmienną kończą się błędem.

```

> typ_v <- c("wolno", "szybko", "średnio", "pyr pyr")
> predkosc <- rep(typ_v, c(12,33,2,100))
> v <- sample(predkosc, 50, replace=TRUE)
> v
[1] "pyr pyr" "pyr pyr" "szybko" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr"
[8] "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr"
[15] "pyr pyr" "pyr pyr" "wolno" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr"
[22] "pyr pyr" "szybko" "pyr pyr" "szybko" "szybko" "pyr pyr" "średnio"
[29] "pyr pyr" "pyr pyr" "wolno" "pyr pyr" "szybko" "pyr pyr" "pyr pyr"
[36] "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr"
[43] "pyr pyr" "pyr pyr" "pyr pyr" "pyr pyr" "szybko" "szybko" "pyr pyr"
[50] "szybko"
> fv <- factor(v, ordered=TRUE, levels=c("wolno", "szybko", "średnio", "pyr pyr"))
> levels(fv)
[1] "wolno" "szybko" "średnio" "pyr pyr"
> |

```

Rysunek 27. Ustalenie porządku podczas tworzenia factor.

```

> da2 <- fv[2]
> da15 <- fv[15]
> da2>da15
[1] FALSE

```

Rysunek 28. Porównywanie elementów factor zgodnie z kolejnością.

```

> fv <- factor(v, ordered=TRUE, levels=c("pyr pyr", "wolno", "średnio", "szybko")
> da2 <- fv[2]
> da15 <- fv[15]
> da2>da15
[1] FALSE
> da2
[1] pyr pyr
Levels: pyr pyr < wolno < średnio < szybko
> da15
[1] pyr pyr
Levels: pyr pyr < wolno < średnio < szybko

```

Rysunek 29. Porównanie po zmianie kolejności.

```

> wiek <- sample(1:100, 16, replace=TRUE)
> wiek
[1] 90 7 66 44 77 43 7 20 17 61 35 51 16 18 21 26
> cut(wiek, c(0,18,26,100))
[1] (26,100] (0,18] (26,100] (26,100] (26,100] (26,100] (0,18] (18,26]
[9] (0,18] (26,100] (26,100] (26,100] (0,18] (0,18] (18,26] (18,26]
Levels: (0,18] (18,26] (26,100]
> kat_wiekowe <- cut(wiek, c(0,18,26,100))
> table(kat_wiekowe)
kat_wiekowe
(0,18] (18,26] (26,100]
      5      3      8

```

Rysunek 30. Podział wektora za pomocą funkcji cut().

```

> kat <- factor(rep(c("klI", "klII", "klIII", "klIV"), 4))
> str(split(wiek, kat))
List of 4
 $ klI : int [1:4] 90 77 17 16
 $ klII: int [1:4] 7 43 61 18
 $ klIII: int [1:4] 66 7 35 21
 $ klIV : int [1:4] 44 20 51 26

```

Rysunek 31. Podział na grupy z wykorzystaniem wektora i faktora.

Ćw. 17.

Listy w R.

```

> x <- c(2,3,2,5,4,3,6,7,8)
> lista_x <- list(srednia=mean(x), minimum=min(x), maksimum=max(x))
> str(lista_x)
List of 3
 $ srednia : num 4.44
 $ minimum : num 2
 $ maksimum: num 8

```

Rysunek 32. Utworzenie prostej listy kilku wartości na podstawie wektora.

```

> print(lista_x$maksimum)
[1] 8
> print(lista_x[[3]])
[1] 8

```

Rysunek 33. Sposoby odwołania się do pojedynczych elementów listy.

```

> a<-list(1:5, LETTERS[1:15], list("a","ó","ę"))
> a[[2]]
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
> a[2]
[[1]]
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"

> a[[2]][10]
[1] "J"
> a[2][10]
[[1]]
NULL

```

Rysunek 34. Utworzenie i odwoływanie się do bardziej skomplikowanej listy.

Ćw. 18.

Ramka danych w R.

```

> tabela.danych1<-data.frame(LETTERS[1:10],1:10,rep(c(F,T),5))
> names(tabela.danych1) <- c("Inicjał","Kolejność","Czy Parzysty")
> tabela.danych1
  Inicjał Kolejność Czy Parzysty
1      A          1      FALSE
2      B          2       TRUE
3      C          3      FALSE
4      D          4       TRUE
5      E          5      FALSE
6      F          6       TRUE
7      G          7      FALSE
8      H          8       TRUE
9      I          9      FALSE
10     J         10       TRUE

```

Rysunek 35. Utworzenie ramki danych i ustawienie nazw kolumn.

```

> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 $
> table(iris$Sepal.Length, factor(iris$Species))

      setosa versicolor virginica
4.3         1           0         0
4.4         3           0         0
4.5         1           0         0
4.6         4           0         0
4.7         2           0         0
4.8         5           0         0
4.9         4           1         1
5          8           2         0

```

Rysunek 36. Wykorzystanie wbudowanej ramki danych zawierającej zbiór danych „Iris”.

Ćw. 19.

Instrukcje w R.

```
> ifelse(1:7 < 4, "mniej", "więcej")
[1] "mniej" "mniej" "mniej" "więcej" "więcej" "więcej" "więcej"
> ifelse(sin(1:5)>0, (1:5)^2, (1:5)^3)
[1] 1 4 9 64 125
>
> liczba <- 1313
> switch(class(liczba), logical = , numeric = cat("typ liczbowy lub logiczny"), $
typ liczbowy lub logiczny>
> for (i in 1:5) { cat(paste("aktualna wartość zmiennej i to ", i, "\n")) }
aktualna wartość zmiennej i to 1
aktualna wartość zmiennej i to 2
aktualna wartość zmiennej i to 3
aktualna wartość zmiennej i to 4
aktualna wartość zmiennej i to 5
```

Rysunek 37. Wykonywanie podstawowych instrukcji w R – instrukcje warunkowe, pętle.

Ćw. 20.

Utwórz na trzy różne sposoby wektor składający się z elementów 10, 9, 8, 7, 6.

```
> wek1 <- c(10,9,8,7,6)
> wek2 <- c(10:6)
> wek3 <- seq(10,6)
> wek1
[1] 10 9 8 7 6
> wek2
[1] 10 9 8 7 6
> wek3
[1] 10 9 8 7 6
```

Rysunek 38. Utworzone trzy identyczne wektory na trzy różne sposoby.

Ćw. 21.

Oblicz wyznacznik macierzy:

$$\begin{vmatrix} -1 & 4 \\ 2 & 1 \end{vmatrix}$$

```
> macierz <- matrix(c(-1,2,4,1), nrow=2, ncol=2)
> macierz
      [,1] [,2]
[1,]   -1    4
[2,]    2    1
> wyznacznik <- det(macierz)
> wyznacznik
[1] -9
```

Rysunek 39. Utworzenie macierzy i obliczenie jej wyznacznika.

Ćw. 22.

Dokonaj mnożenia macierzy:

$$\begin{bmatrix} 1 & 2 \\ -2 & 4 \\ 1 & 7 \end{bmatrix} \quad \begin{vmatrix} 1 & 3 & 5 \\ -4 & -1 & 2 \end{vmatrix}$$

```
> macierz1 <- matrix(c(1,-2,1,2,4,7), nrow=3, ncol=2)
> macierz2 <- matrix(c(1,-4,3,-1,5,2), nrow=2, ncol=3)
> macierz1
      [,1] [,2]
[1,]    1    2
[2,]   -2    4
[3,]    1    7
> macierz2
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]   -4   -1    2
>
> wynik <- macierz1 %*% macierz2
> wynik
      [,1] [,2] [,3]
[1,]   -7    1    9
[2,]  -18  -10   -2
[3,]  -27   -4   19
```

Rysunek 40. Utworzenie dwóch macierzy i ich mnożenie.

Ćw. 23.

Wyświetl za pomocą jednej instrukcji sześciany liczb od 1 do 20.

```
> print(c(1:20)^3)
[1]    1    8   27   64  125  216  343  512  729 1000 1331 1728 2197 2744 3375
[16] 4096 4913 5832 6859 8000
```

Rysunek 41. Jedna instrukcja użyta do wyświetlenia sześciąt liczb od 1 do 20.

Ćw. 24.

Wyświetl za pomocą jednej instrukcji kwadraty liczb parzystych od 16 do 40.

```
> print(seq(16,40,2)^2)
[1]  256  324  400  484  576  676  784  900 1024 1156 1296 1444 1600
```

Rysunek 42. Jedna instrukcja użyta do wyświetlenia sześciąt liczb parzystych od 16 do 40.

Ćw. 25.

Napisać funkcję liczącą wartość ciągu Fibonacciego dla podanego n.

```

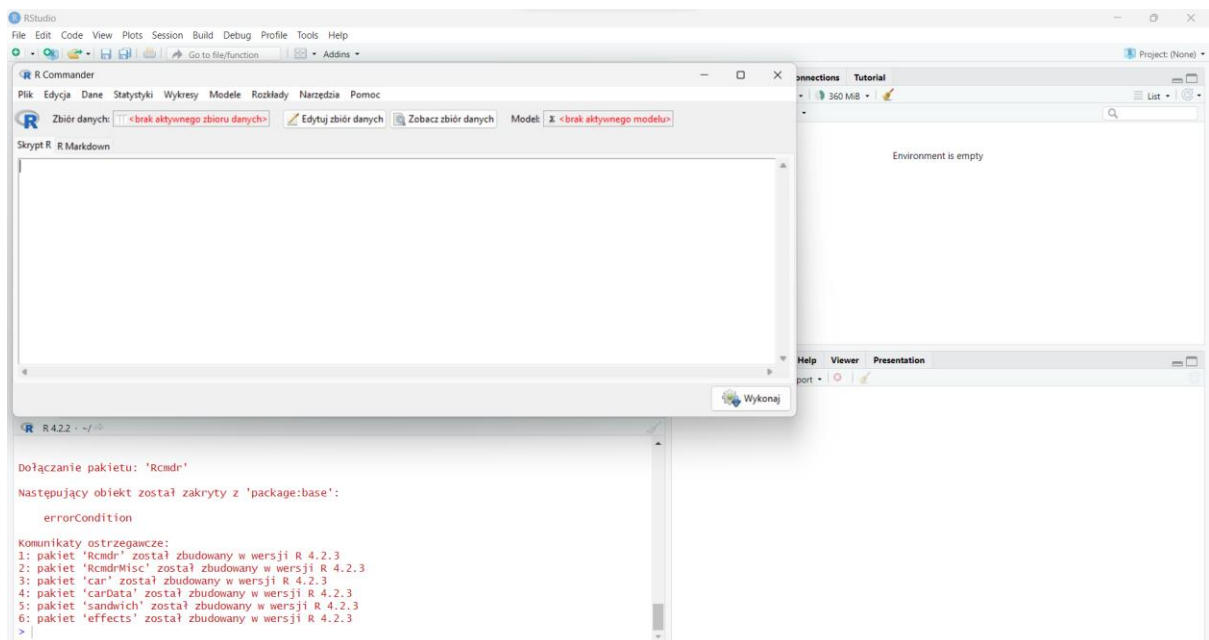
> fib <- function(n) {
+   if (n<=0) {
+     return(-1)
+   }
+   else if (n==1) {
+     return(0)
+   }
+   else if (n==2) {
+     return(1)
+   }
+   else {
+     a <- 0
+     b <- 1
+     for (i in 3:n) {
+       c <- a + b
+       a <- b
+       b <- c
+     }
+     return(b)
+   }
+ }
> fib(10)
[1] 34

```

Rysunek 43. Stworzona funkcja i przykład jej użycia dla $n=10$. Funkcja została zapisana w osobnym pliku źródłowym w celu prostszego użycia.

Ćw. 26.

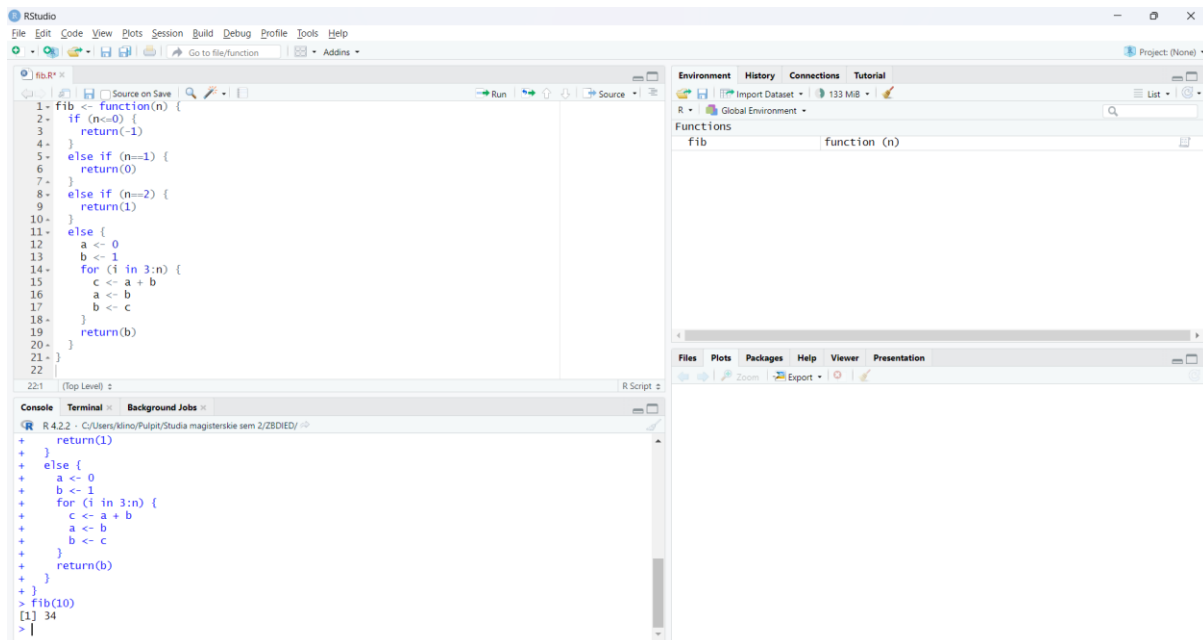
Pakiet R Commander – środowisko graficzne z poziomu R.



Rysunek 44. Ekran po instalacji i uruchomieniu środowiska graficznego R Commander.

Ćw. 27.

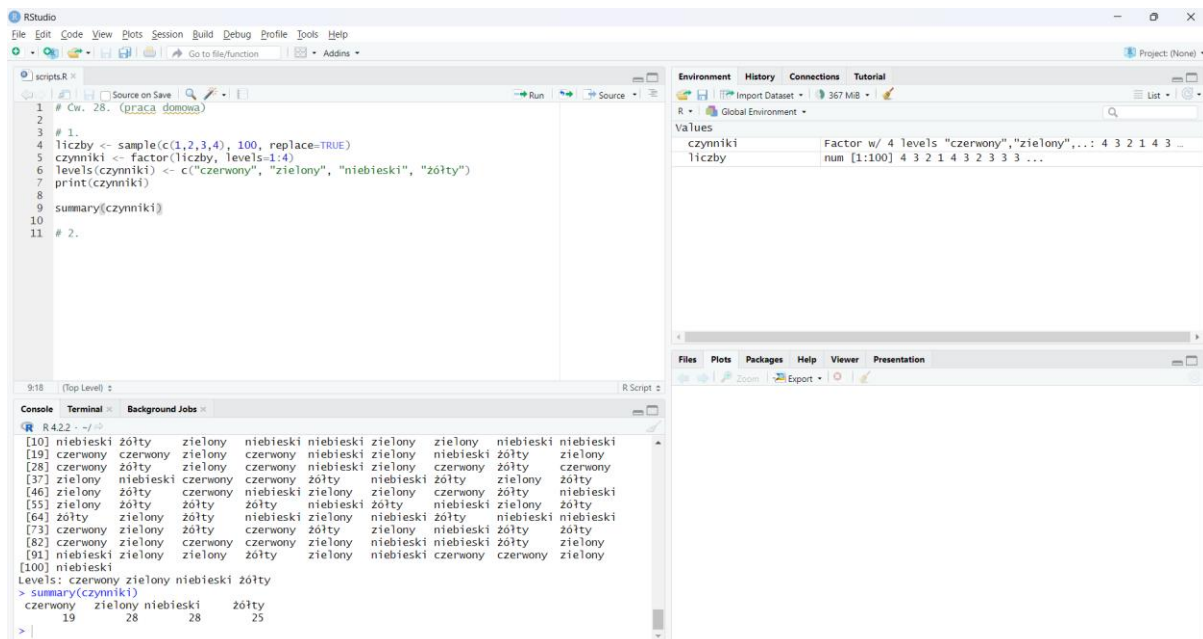
Środowisko RStudio.



Rysunek 45. Uruchomione środowisko RStudio z utworzonym wcześniej plikiem zawierającym funkcję z ciągiem Fibonacciego. Środowisko posiada podstawowe kolorowanie składni, konsolę oraz podgląd zmiennych w pamięci (w tym tabeli). Wyświetlają się w nim również utworzone wykresy.

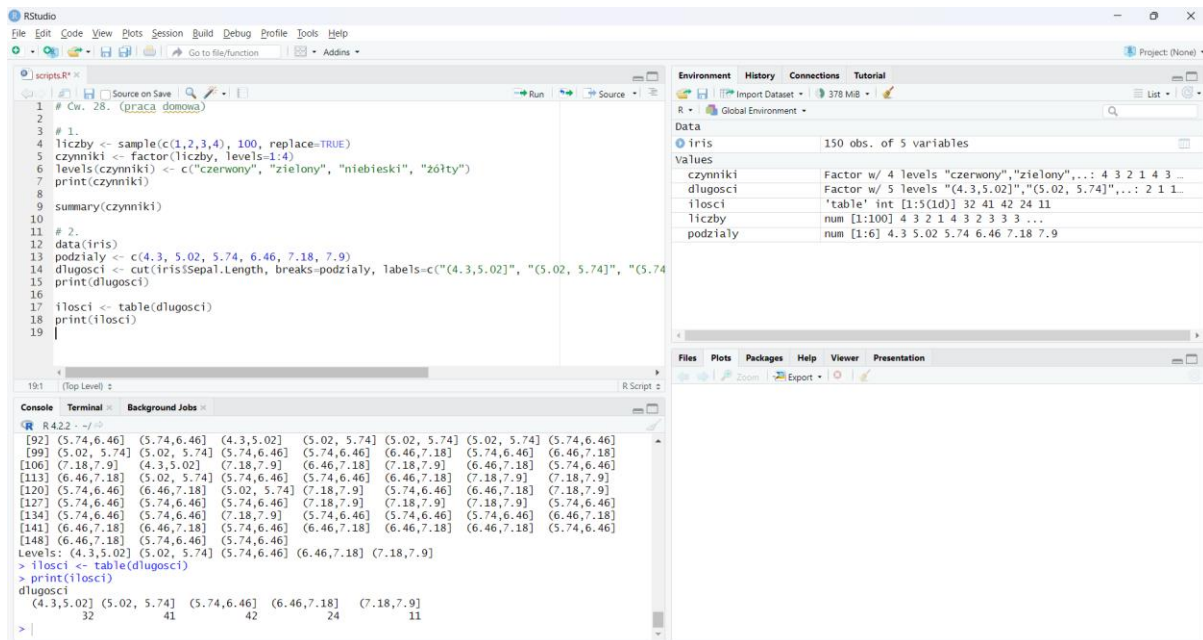
Ćw. 28.

1. Utwórz wektor 100 liczb naturalnych...



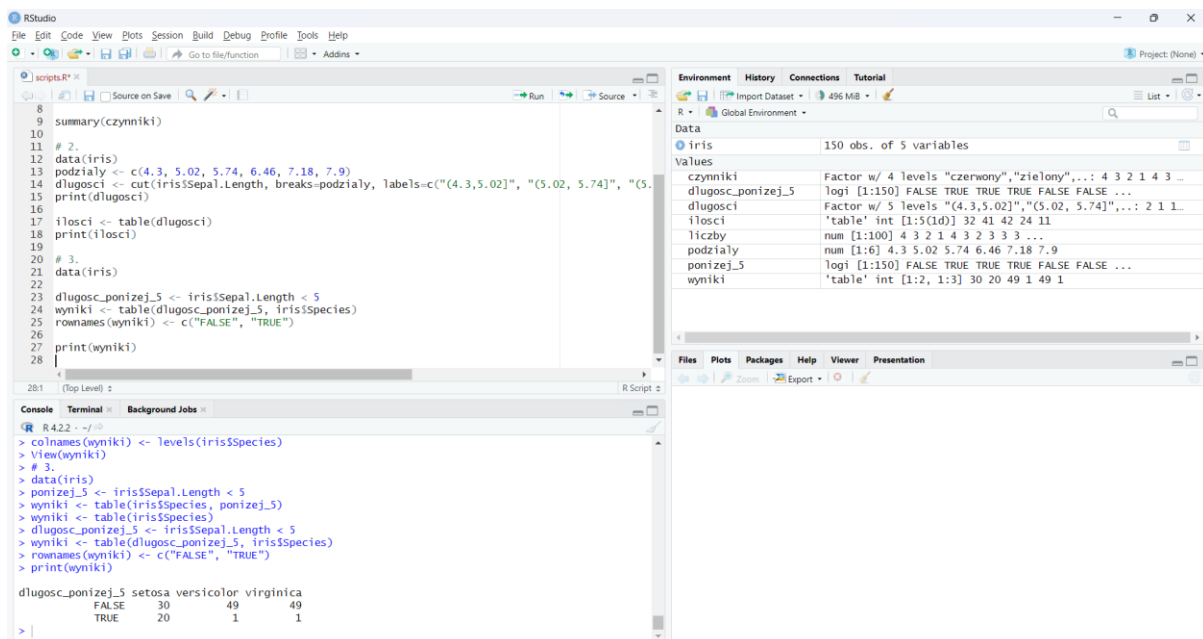
Rysunek 46. Ekran RStudio z kodem rozwiązującym ten podpunkt oraz wynikami działania.

2. Podziel Sepal.Length na 5 grup...



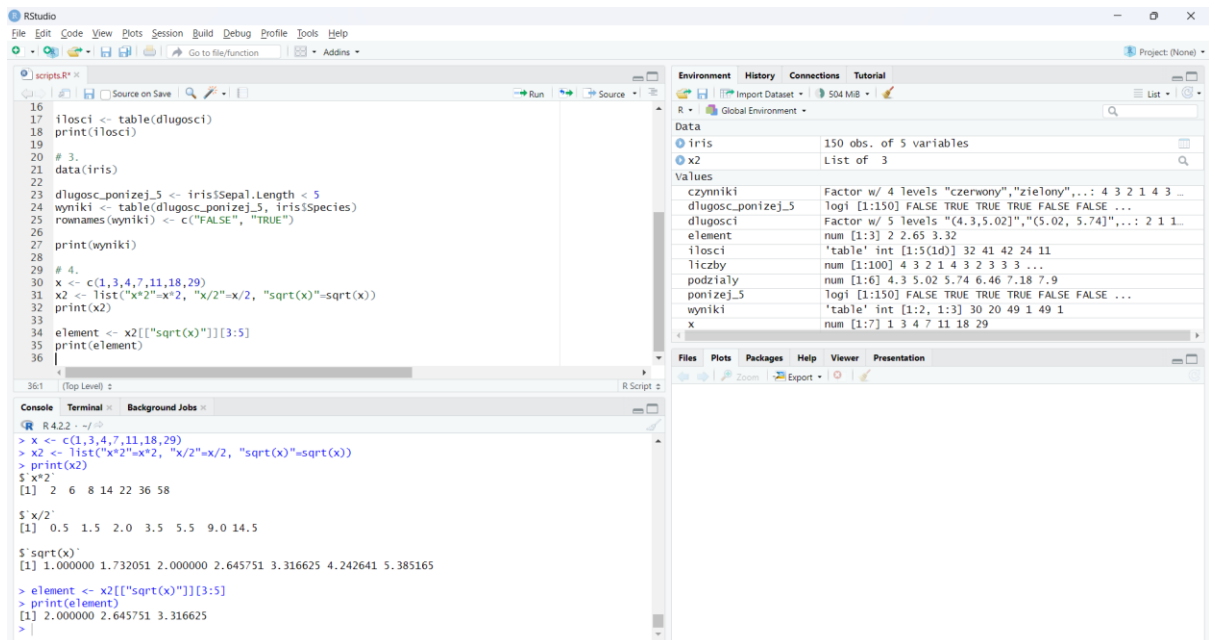
Rysunek 47. Kod źródłowy i wyniki działania kodu wykonującego to polecenie.

3. Podaj częstość występowania Sepal.Length...



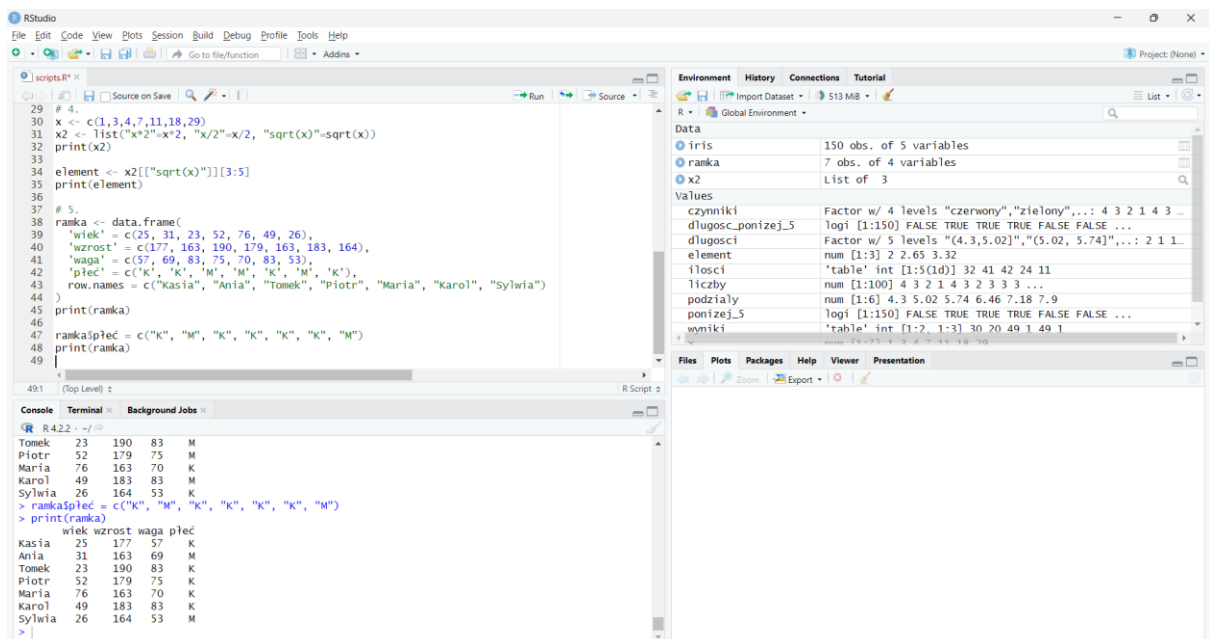
Rysunek 48. Kod tworzący tabelę o dwóch wierszach – TRUE i FALSE – oraz kolumnach odpowiadających gatunkom kwiatów.

4. Utwórz listę dla wektora...



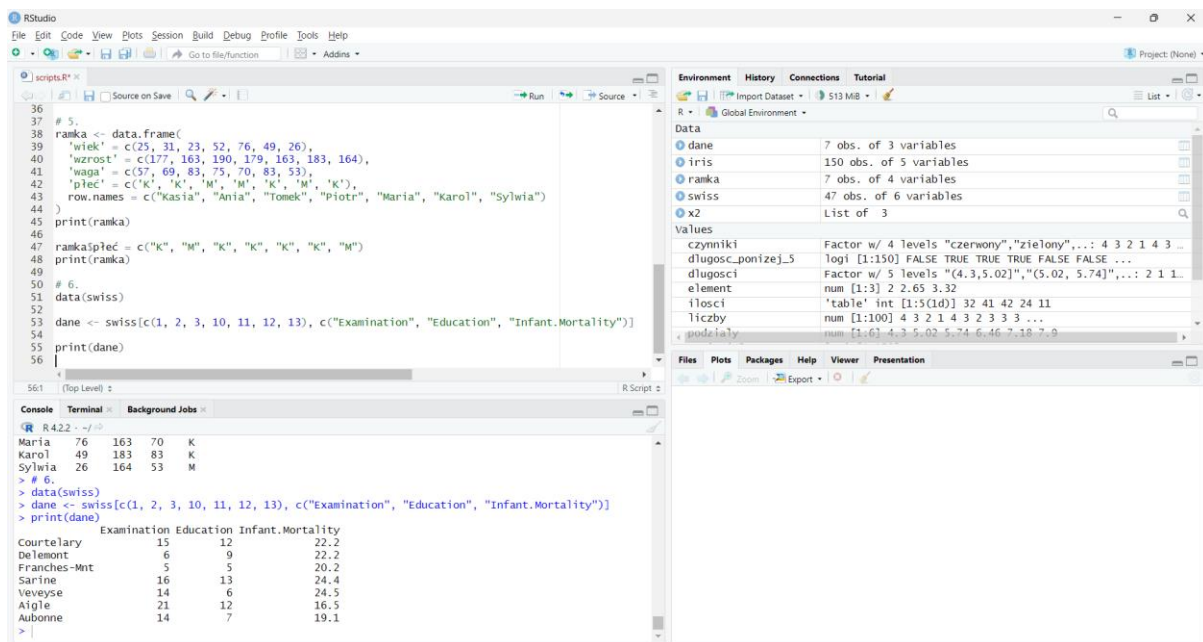
Rysunek 49. Utworzenie listy z nazwanymi elementami oraz wyświetlenie trzech kolejnych wartości pierwiastka z x zgodnie z instrukcją.

5. Utwórz data.frame...



Rysunek 50. Utworzenie data.frame zgodnie z instrukcją i zmiana kolumny z płcią.

6. Zbiór danych swiss...



Rysunek 51. Kod źródłowy rozwiązujący polecenie 6.

Poniżej można znaleźć cały kod źródłowy napisany do wykonania Ćwiczenia 28.

```

# Ćw. 28. (praca domowa)

# 1.
liczby <- sample(c(1,2,3,4), 100, replace=TRUE)
czynniki <- factor(liczby, levels=1:4)
levels(czynniki) <- c("czerwony", "zielony", "niebieski", "żółty")
print(czynniki)

summary(czynniki)

# 2.
data(iris)
podzialy <- c(4.3, 5.02, 5.74, 6.46, 7.18, 7.9)
dlugosci <- cut(iris$Sepal.Length, breaks=podzialy, labels=c("(4.3,5.02]",
"(5.02, 5.74]", "(5.74,6.46]", "(6.46,7.18]", "(7.18,7.9]"),
include.lowest=TRUE)
print(dlugosci)

ilosci <- table(dlugosci)
print(ilosci)

# 3.
data(iris)

dlugosc_ponizej_5 <- iris$Sepal.Length < 5
wyniki <- table(dlugosc_ponizej_5, iris$Species)
rownames(wyniki) <- c("FALSE", "TRUE")

print(wyniki)

# 4.
x <- c(1,3,4,7,11,18,29)
x2 <- list("x*2"=x*2, "x/2"=x/2, "sqrt(x)"=sqrt(x))
print(x2)

element <- x2[["sqrt(x)"]][3:5]
print(element)

# 5.
ramka <- data.frame(
  'wiek' = c(25, 31, 23, 52, 76, 49, 26),
  'wzrost' = c(177, 163, 190, 179, 163, 183, 164),
  'waga' = c(57, 69, 83, 75, 70, 83, 53),
  'płeć' = c('K', 'K', 'M', 'M', 'K', 'M', 'K'),
  row.names = c("Kasia", "Ania", "Tomek", "Piotr", "Maria", "Karol",
"Sylwia")
)
print(ramka)

ramka$płeć = c("K", "M", "K", "K", "K", "K", "M")
print(ramka)

# 6.
data(swiss)

dane <- swiss[c(1, 2, 3, 10, 11, 12, 13), c("Examination", "Education",
"Infant.Mortality")]

print(dane)

```

Wnioski.

Instalacja pakietu Rattle była wyzwaniem – w przypadku użycia nowszej wersji R występowały problemy z zależnością Rattle „RGtk2”. Dopiero po zainstalowaniu wersji R 4.1.3 i zainstalowaniu zgodnie z instrukcją na stronie pakietu, było możliwe poprawne uruchomienie Rattle.

Podczas laboratorium zapoznałem się z podstawami języka R, który jest w dużym stopniu zorientowany wokół danych – wektorów, list, ramek danych, itp. Standard języka zawiera sporo funkcjonalności do manipulacji danymi, i to jest największym plusem tego środowiska. Dodawanie nowych elementów do wektora, kolumn czy wierszy do ramki danych i tym podobne operacje są niezwykle proste i intuicyjne, a wiele wbudowanych operatorów działa nie tylko dla pojedynczych wartości, ale całych kolekcji danych. W kontekście w jakim używamy R, to takie operacje są bardziej istotne i częściej wykorzystywane niż standardowe instrukcje programistyczne.

W niektórych częściach ćwiczenia laboratoryjnego wykorzystano RStudio, będące najpopularniejszym środowiskiem programistycznym dla języka R. Zawiera ono podstawowe funkcje takie jak: podpowiadanie nazw, kolorowanie składni czy łatwy dostęp do środowiska roboczego, posiada wbudowany podgląd zmiennych w pamięci, wyświetlanie utworzonych wykresów i, poza obsługą plików źródłowych, konsolę R.