

Zaawansowane typy danych

Statystyka I z R

Bartosz Maćkiewicz

Zaawansowane typy danych

W tej części kursu będziemy omawiać bardziej zaawansowane typy danych, z którymi możemy pracować w R.

Dygresja: `dolna_granica:górna_granica`

Na początek nauczymy się tworzenia wektorów liczb całkowitych (`integer`) za pomocą konstrukcji `dolna_granica:górna_granica`.

Jest to bardzo często wykorzystywany sposób, który pojawiać będzie się przez cały kurs.

```
dwadziescia_liczb <- 1:20 # Tworzymy wektor typu integer z liczbami od 1 do  
↪ 20  
dwadziescia_liczb # sprawdzamy czy rzeczywiście się utworzył
```

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

```
class(dwadziescia_liczb) # sprawdzamy jego typ
```

```
[1] "integer"
```

Jeżeli jako pierwszą podamy większą liczbę, to R utworzy ciąg malejący.

```
5:1
```

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

Jeżeli interesują nas bardziej wyrafinowane sekwencje liczb, to powinniśmy zainteresować się funkcją `seq`. Poniżej tworzymy sekwencję zaczynającą się od 5, w której każda kolejna liczba jest większa o 3. Nie przekraczamy podanej granicy 55, więc ostatnią liczbą będzie 53. Więcej o tej funkcji można znaleźć w dokumentacji (`?seq`)

```
seq(from = 5, to = 55, by = 3)
```

```
[1]  5  8 11 14 17 20 23 26 29 32 35 38 41 44 47 50 53
```

Macierze

Utwórzmy też macierz o wymiarach 4x5. Kolejne liczby z wektora `dwadziescia_liczb` wypełniają po kolei kolumny naszej macierzy. Zrobimy to za pomocą funkcji `matrix`.

```
matrix(dane, wiersze, kolumny)
```

- `dane` - wartości, którymi chcemy wypełnić macierz
- `wiersze` - liczba wierszy
- `kolumny` - liczba kolumn

Warto w tym miejscu zwrócić uwagę na dwa fakty.

1. R domyślnie wypełnia macierze kolumnami
2. Jeżeli jakaś funkcja przyjmuje jako argumenty wymiary jakiejś dwuwymiarowej struktury danych, to najpierw podaje się liczbę wierszy a dopiero potem liczbę kolumn. Oto mnemotechniczna rymowanka, która pozwala zapamiętać to zachowanie: *wiersze pierwsze*.

```
macierz <- matrix(dwadziescia_liczb, 4,5)
macierz
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]     1     5     9    13    17
[2,]     2     6    10    14    18
[3,]     3     7    11    15    19
[4,]     4     8    12    16    20
```

Za pomocą argumentu `byrow` możemy jednak wypełniać macierz wierszami.

```
macierz_wiersze <- matrix(dwadziescia_liczb, 4,5, byrow = TRUE)
macierz_wiersze
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	2	3	4	5
[2,]	6	7	8	9	10
[3,]	11	12	13	14	15
[4,]	16	17	18	19	20

Do macierzy możemy dodawać nowe wiersze i kolumny za pomocą funkcji `cbind` i `rbind`.

```
print(cbind(macierz_wiersze, c(1000,2000,3000,400))) # analogicznie rbind
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	1	2	3	4	5	1000
[2,]	6	7	8	9	10	2000
[3,]	11	12	13	14	15	3000
[4,]	16	17	18	19	20	400

```
print(rbind(macierz_wiersze, macierz_wiersze)) # możemy nawet "złączyć" dwie
↪ macierze w ten sam sposób
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	2	3	4	5
[2,]	6	7	8	9	10
[3,]	11	12	13	14	15
[4,]	16	17	18	19	20
[5,]	1	2	3	4	5
[6,]	6	7	8	9	10
[7,]	11	12	13	14	15
[8,]	16	17	18	19	20

Na macierzach podobnie jak na wektorach możemy wykonywać operacje element-po-elemente, na przykład mnożenie.

```
macierz * macierz_wiersze # mnożenie R-owe, "element-po-elemente"
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	10	27	52	85
[2,]	12	42	80	126	180
[3,]	33	84	143	210	285
[4,]	64	136	216	304	400

Możemy również wykonywać operacje algebraiczne na macierzach.

```
a <- matrix(c(2,4,6,8,10,12,14,18), 2,4)
a
```

```
      [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    4    8   12   18
```

```
b <- matrix(c(9,8,7,6,5,4,3,2), 4,2)
b
```

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

```
a %*% b # iloczyn macierzy
```

```
      [,1] [,2]
[1,]   220   92
[2,]   292  124
```

Inne przydatne podstawowe operacje:

- t - transpozycja

```
# inny sposób tworzenia macierzy za pomocą funkcji cbind (*column bind*)
macierz_2 <- cbind(30:40, 25:35, 130:140)
t(macierz_2) # Transpozycja
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
[1,]   30   31   32   33   34   35   36   37   38   39   40
[2,]   25   26   27   28   29   30   31   32   33   34   35
[3,]  130  131  132  133  134  135  136  137  138  139  140
```

- rowSums, colSums, rowMeans, colMeans - sumy/średnie kolumnami/wierszami

```
rowMeans(macierz_2) # analogiczne colMeans
```

```
[1] 61.66667 62.66667 63.66667 64.66667 65.66667 66.66667 67.66667 68.66667
[9] 69.66667 70.66667 71.66667
```

```
colSums(macierz_2) # analogicznie rowSums
```

```
[1] 385 330 1485
```

O wymiarach macierzy możemy się dowiedzieć za pomocą funkcji `dim`, `ncol`, `nrow`

```
dim(macierz_2) # wymiary
```

```
[1] 11 3
```

```
ncol(macierz_2) # liczba kolumn
```

```
[1] 3
```

```
nrow(macierz_2) # liczba wierszy
```

```
[1] 11
```

Dla naszych celów ta wiedza powinna wystarczyć, statystycy z socjologii prosili jednak o podstawy algebry w R. Polecam więc zajrzenie w stosownym czasie do np. [Algebra w R](#) (polecam również samą stronę!)

Listy

Listy w przeciwieństwie do wektorów przechowują sekwencje wartości różnych typów. Możemy w niej przechowywać wektory różnych typów o różnej długości a nawet inne listy. W tym sensie są podobne do list z niektórych innych języków programowania. Listy tworzymy za pomocą funkcji `list`.

```
lista <- list(c(1,2,3,4,5), # pierwszy element: wektor typu `numeric`
             c('Adam', 'Marysia', 'Zosia', 'Tosia'), # drugi element: wektor
             ↪ typu `character`
             TRUE, # trzeci element: wektor typu `logical`
             TRUE, # czwarty element: wektor typu `logical`
             list( # piąty element: lista
                 c(-1,-2,-3) # pierwszy element: wektor typu `numeric`
               )
             )
lista
```

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

```
[[2]]  
[1] "Adam"      "Marysia" "Zosia"    "Tosia"
```

```
[[3]]  
[1] TRUE
```

```
[[4]]  
[1] TRUE
```

```
[[5]]  
[[5]][[1]]  
[1] -1 -2 -3
```

Elementom listy możemy nadawać nazwy. Dlatego w niektórych sytuacjach możemy traktować je jak słowniki z innych języków programowania, przechowujące pary klucz-wartość.

Nazwane listy również tworzymy przy pomocy funkcji `list`.

```
lista_nazwana <- list(liczby_od_jeden_do_piec = c(1,2,3,4,5),  
                     imiona = c('Adam', 'Marysia', 'Zosia', 'Tosia'),  
                     czy_r_jest_fajny = TRUE,  
                     czy_zajecia_sa_fajne = TRUE,  
                     lista_zagniezdzona = list(c(-1,-2,-3)))  
lista_nazwana
```

```
$liczby_od_jeden_do_piec  
[1] 1 2 3 4 5
```

```
$imiona  
[1] "Adam"      "Marysia" "Zosia"    "Tosia"
```

```
$czy_r_jest_fajny  
[1] TRUE
```

```
$czy_zajecia_sa_fajne  
[1] TRUE
```

```
$lista_zagniezdzona  
$lista_zagniezdzona[[1]]  
[1] -1 -2 -3
```

Do list wrócimy jeszcze kiedy będziemy mówić o indeksowaniu.

Ramki danych (*data frame*)

Ramki danych (*data frame*) są typem danych, z którym w praktyce będziemy pracować najczęściej. O ramkach danych można myśleć jako np. o tabelach Excela, w której każdy wiersz reprezentuje jedną obserwację.

Tworzenie ramek danych jest dość podobne do tworzenia list. To, co w listach było nazwami elementów, w przypadku ramek danych będzie nazwami kolumn. Poniżej znajduje się kilka przydatnych sposobów tworzenia ramek danych. Na razie pomijamy najczęściej wykorzystywany w praktyce sposób. Z reguły bowiem ramki danych będziemy tworzyć wczytując dane z pliku. Tym jednak zajmiemy się później.

Podstawowy sposób stworzenia ramki danych wygląda tak:

```
x <- c('Kasia', 'Zosia', 'Marysia', 'Tosia', 'Zbyszek')
y <- factor(c('F', 'F', 'F', 'F', 'M'), levels = c('F', 'M'))
z <- c(2, 4, 5, 8, 2)

# ramki danych tworzy się w zasadzie tak samo jak listy
dane <- data.frame(imie = x,
                   plec = y,
                   liczba_jabluszek = z
                   )

dane
```

	imie	plec	liczba_jabluszek
1	Kasia	F	2
2	Zosia	F	4
3	Marysia	F	5
4	Tosia	F	8
5	Zbyszek	M	2

Jeśli mamy listę, której elementy są wektorami równej długości, to możemy przekonwertować ją na ramkę danych.

```
lista_nie_ramka <- list(imie = x,
                        plec = y,
                        liczba_jabluszek = z)
data.frame(lista_nie_ramka)
```

	imie	plec	liczba_jabluszek
1	Kasia	F	2
2	Zosia	F	4
3	Marysia	F	5
4	Tosia	F	8
5	Zbyszek	M	2

W ramce danych nie tylko kolumny mogą mieć swoją nazwę. Możemy ustalić, że również każdy wiersz będzie miał nazwę. Uczynimy to przekazując funkcji `data.frame` argument `row.names` w postaci wektora z nazwami.

```
data.frame(plec = y,
           liczba_jabluszek = z,
           row.names = x)
```

	plec	liczba_jabluszek
Kasia	F	2
Zosia	F	4
Marysia	F	5
Tosia	F	8
Zbyszek	M	2

Do postaci ramki danych możemy przekonwertować również macierz. Czasami jest to przydatne. Służy do tego funkcja `as.data.frame`.

```
ramka_z_macierzy <- as.data.frame(macierz) # możemy utworzyć z macierzy
ramka_z_macierzy
```

	V1	V2	V3	V4	V5
1	1	5	9	13	17
2	2	6	10	14	18
3	3	7	11	15	19
4	4	8	12	16	20

Powiedzieliśmy sobie, że w ramkach danych kolumny oraz wiersze mogą mieć nazwy. Jeśli chodzi o kolumny, to funkcji `colnames` zwraca wektor z nazwami kolumn ramki danych.

```
dane
```

	imie	plec	liczba_jabluszek
1	Kasia	F	2
2	Zosia	F	4
3	Marysia	F	5

4	Tosia	F	8
5	Zbyszek	M	2

```
colnames(dane)
```

```
[1] "imie"          "plec"          "liczba_jabluszek"
```

Za pomocą tej samej funkcji możemy zmienić też nazwy. Wystarczy, że nadpiszemy zwrócony przez funkcję `colnames` wektor swoim własnym wektorem, w którym zawrzemy nowe nazwy kolumn. To samo możemy zrobić również z nazwami wierszy, korzystając w funkcji `rownames`.

```
ramka_z_macierzy
```

	V1	V2	V3	V4	V5
1	1	5	9	13	17
2	2	6	10	14	18
3	3	7	11	15	19
4	4	8	12	16	20

```
# Zastępujemy domyślnie nazwy kolumn (V1, V2, V3...) swoimi własnymi.
colnames(ramka_z_macierzy) <- c('Jeden', 'Dwa', 'Trzy', 'Cztery', 'Pięć')
ramka_z_macierzy
```

	Jeden	Dwa	Trzy	Cztery	Pięć
1	1	5	9	13	17
2	2	6	10	14	18
3	3	7	11	15	19
4	4	8	12	16	20