Statystyczna Analiza Danych – laboratorium

Typy i struktury danych w R

Dorota Celińska-Kopczyńska

Uniwersytet Warszawski

Typy i struktury danych

- W R dostępne są różnorodne struktury do przechowywania danych
- Różnią się typem danych, które mogą przechowywać, sposobem tworzenia, złożonością struktury i notacją wykorzystywaną, np. do dostępu do poszczególnych elementów

Numeric

- Przechowuje wartości **dziesiętne**
- Domyślnie używany do reprezentacji liczb

```
x = 10.5          # przypisz wartosc dziesietna
x                 # wypisz wartosc x
class(x)          # wypisz nazwe klasy, do ktorej nalezy x
```

Numeric a Integer, inne klasy do reprezentacji liczb

- Domyślnie, typ numeric przechowuje też *wartości, które nazwalibyśmy* integer (typ całkowitoliczbowy)
- To samo dzieje się w przypadku typu double (typ zmiennoprzecinkowy podwójnej precyzji)
- W większości zastosowań nie ma to znaczenia, ale warto wiedzieć, że integer, double i numeric nie są tą samą klasą w R

```
k = 1              # k to liczba calkowita, ale R domyslnie przechowa ja jako numeric!
class(k)           # wypisz nazwe klasy, do ktorej nalezy k  -- zwraca "numeric"
is.integer(k)      # sprawdz, czy k to integer
k <- as.integer(k) # zmien klase, do ktorej nalezy k na integer
class(k)           # teraz to integer!
```

Character

- Reprezentuje **napisy**
- Można skonwertować obiekty innych klas na napisy, wykorzystując funkcję `as.character()`

```
name = "Dorota"
name
x = as.character(3.14)
x               # wypisz napis x
class(x)        # wypisz nazwe klasy, do ktorej nalezy x
```

Logical

- Typ logiczny, wartość służąca do porównań TRUE (prawda) albo FALSE (fałsz)
- Standardowymi logicznymi operatorami są "&" (i), "|" (lub) oraz "!" (negacja)

```
x = 1; y = 2   # przykladowe wartosci
z = x > y      # zapisze pod z wynik zapytania czy x jest wieksze niz y
z              # wypisz wartosc z
class(z)       # wypisz klase, do ktorej nalezy z
               # dzialania na typach logicznych
u = TRUE; v = FALSE
u & v          # u I v
u | v          # u LUB v
!u             # NIE u
```

Uwaga: R zwraca uwagę na wielkość liter! Przykładowo, **c()** a **C()** albo `TRUE` a ‘‘`True`’’ lub `"TRUE"`!

Factor

- Jako typ factor przechowywane są zmienne, które przyjmują ograniczoną liczbę wartości: zmienne kategoryczne
- Dane przechowywane są jako wektor z wartościami integer z odpowiadającym im zbiorem wartości typu character
- Wartości character używane są do wyświetlania wartości factor. Domyślnie uporządkowane alfabetycznie
- Do typu factor można przekonwertować zarówno napisy jak i liczby

```
data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
fdata = factor(data)
fdata
rdata = factor(data,labels=c("I","II","III"))
rdata
```

Vector

- Jednowymiarowe tablice
- **Vector** zawiera dane **jednego** typu, np. numeric, character, logical. Nie można mieszać typów w tym samym wektorze!
- Domyślnie tworzony funkcją `c()`
- Skalar to jednoelementowy wektor. Wykorzystywane do przechowywania stałych.

```
a <- c(1,2,3,4,5,-10,12) # numeric vector
b <- c("R", "is", "not", "very", "difficult") # character vector
c <- c(TRUE, TRUE, TRUE, FALSE, FALSE, TRUE) # logical vector
```

Inne sposoby tworzenia wektorów

- `seq` – tworzy **sekwencję** liczb
- `vector` – tworzy vector danego typu, wykorzystując dane tego typu
- `double(), integer(), character(), logical()` – tworzy vector danej długości i dla typu danych

```
integers <- vector("integer", 10) # inicjalizuje (pusty) vector zawierajacy integer
character(10)  # inicjalizuje (pusty) vector zawierajacy character
logical(10) # inicjalizuje (pusty) vector zawierajacy wartosci logical -- 0 to FALSE
seq(10) # tworzy sekwencje liczb
seq(10,20,2) # tworzy sekwencje 10,12,14,16,18,20
c(10:12) # tworzy sekwencje 10,11,12
```

Przydatne funkcje

Akcja	Funkcja	Przykład
Odwołanie do elementu	nazwa[indeks]	`wektor[2]`
Odwołanie do wielu elementów	nazwa[c()]	`wektor[c(1,2,3,7:10)]`
Warunkowe odwołanie do elementów	nazwa[warunek]	`wektor[wektor < 3]`
Długość wektora	`length()`	`length(wektor)`
Suma elementów	`sum()`	`sum(wektor)`
Najwyższa wartość	`max()`	`max(wektor)`

Macierze

- **Matrix** to **dwuwymiarowa** tablica przechowująca dane **jednego** typu
- Każda kolumna musi być tej samej długości
- Tworzona z wykorzystaniem funkcji `matrix()`

```
# ogolna formula:
# nazwa-matrix <- matrix(vector, nrow=liczba_wierszy, ncol=liczba_kolumn, byrow=logical)
# vector zawiera elementy macierzy
# byrow wskazuje, czy macierz ma byc wypelniana wierszami (byrow = TRUE)
# czy kolumnami (byrow = FALSE); domyslnie jest kolumnami
# logical wskazuje, ze oczekujemy tu zmiennej tego typu
# jesli wektor ma mniej niz nrowncol elementow, nastapi jego "recykling"
x <- matrix(1:20, nrow=5, ncol=4)
```

Indeksowanie

```
# x to nazwa macierzy
# struktura ma dwa wymiary, wiec DWIE wartosci sa nam potrzebne
# pierwszy indeks wskazuje wiersz
# drugi indeks wskazuje kolumne

x[,4] # czwarta kolumna macierzy (wszystkie wiersze)
x[3,] # trzeci wiersz macierzy (wszystkie kolumny)
x[2:4,1:3] # wiersze 2,3,4 z kolumn 1,2,3
x[c(4,7), 1] # wiersze 4 i 7 kolumny 1
```

Przydatne funkcje

Akcja	Funkcja	Przykład
wartości własne	`eigen()`	`eigen(macierz)`
wykładnik	`det()`	`det(macierz)`
transpozycja	`t()`	`t(macierz)`
macierz odwrotna	`solve()`	`solve(macierz)`
suma elementów kolumn	`colSums()`	`colSums(macierz)`
suma elementów wierszy	`rowSums()`	`rowSums(macierz)`
średnia z elementów kolumn	`colMeans()`	`colMeans(macierz)`
średnia z elementów wierszy	`rowMeans()`	`rowMeans(macierz)`

Array

- Podobne do macierzy, ale mogą zawierać więcej niż dwa wymiary danych jednego typu.
- Tworzone z wykorzystaniem funkcji `array()`
- `help(array)` zawiera szczegóły

List

- Uporządkowana, jednowymiarowa kolekcja obiektów
- Listy mogą zawierać obiekty różnych typów
- Tworzone z wykorzystaniem funkcji `list()`

```
# przyklad listy zawierajacej 4 skladniki -
# napis, wektor liczb, macierz i skalar
# skladniki potrzebuja nazw, zeby mozna bylo sie latwo do nich odwolac
w <- list(name="Teresa", mynumbers=a, mymatrix=y, age=5.3)
# przyklad listy zawierajacej dwie listy
v <- c(list1,list2)
```

Odwołanie się do elementu

```
# x to nazwa listy
# Mozna uzyskac elementy listy, wykorzystujac operator  [[]]
# albo nazwy elementow
x[[2]] # drugi komponent listy
x[["name"]] # komponent o nazwie "name" w liscie
x$name     # odwolanie za pomoca nazwy elementu
```

Data Frame

- **Data Frame** (ramka danych) jest generalizacją macierzy
- Kolumny mogą zawierać dane różnych typów (np., numeric, character, factor)
- Popularna struktura wśród analityków danych

```
studentID <- c(1:4)
studentName <- c("Adam", "Beata", "Cecylia", "Dawid")
studentMajor <- c("Inf", "Inf", "Mat", "Mat")
studentMajor <- factor(studentMajor)
students <- data.frame(studentID, studentName, studentMajor)
students
str(students)
class(students)
```

Odwołanie do elementów

```
# x to nazwa data frame
# można odwołać się po indeksach, podobnie jak do elementow macierzy
# albo wykorzystujac nazwy kolumn
x[,2] # druga kolumna
x[3,] # trzeci wiersz
x[,"age"] # wszystkie elementy kolumny o nazwie "age"
x$name    # odwolanie z wykorzystaniem nazwy kolumny
```

Alternatywy dla Data Frame

- **tibble** (tidyverse) *modern version of data frame*
- Tibble nie zmienia typu danych (np. napis na factor), nazw zmiennych, nie dodaje row.names().
- **data.table** (data.table) *enhanced version of data frame*
- Nie zmienia napisu na factor, optymalizuje niektóre operacje, umożliwia stosowanie kodu podobnego do SQL/Staty

```
library(tidyverse)
as_tible(iris)
tibble(x = 1:5, y = 1, z = x ^ 2 + y)

library(data.table)
setDT(iris)
as.data.table(iris)
DT = data.table(
  ID = c("b","b","b","a","a","c"),
  a = 1:6,
  b = 7:12,
  c = 13:18
)
```

Konwersje

- Typy danych i struktur mogą być zmieniane
- Najczęściej możemy je zmienić, stosując funkcję `as.[nazwa]()`, np. `as.character()`, `as.data.frame()`