Duomenų analizės įvadas

2-1 Dalis

Justas Mundeikis

2019 m. vasario 26 d.

1. Dalies turinys

Šioje dalyje

2-1 R programavimo dalyje susipažinsime su

- if-else
- for, while, repeat loop
- R funkcijomis
- Datos formatais

Valdymo struktūros (control structures) leidžia valdyti programų veikimą, priklausomai nuo tam tikrų aplinkybių:

- if, else: testuoja tam tikrą aplinkybę
- for: vykdo programą tam tikrą iteracijų skaičių
- while: vykdo programą kol egzistuoja tam tikros aplinkybės
- repeat: vykdo nesibaigiančią iteraciją
- break: nutraukia iteracijos procesą
- next: peršoka 1 iteraciją
- return: nutraukia funkciją

if

```
# 1
if(<condition>) {
        ## do something
}

#2
if(<condition>) {
        ## do something
} else{
        ## do something
}

#3
if(<condition>) {
        ## do something
} else if(<condition>) {
        ## do something
} else{
        ## do something
}
```

if

```
x <- 5

if(x>3){
        y <- 5
} else{
        y <- 0
}

# tapatu

y <- if (x>3) {
        5
} else{
        0
}
```

for

- for loop dažniausiai naudojami iteruoti tam tikriems veiksmams, žinant, kiek kartų iteracija turi trukti.
- galima naudoti "i" arba bet kokią kitą raidę / stringą

```
for (i in 1:5){
        print(i)
}

for (values in 1:5){
        print(values)
}

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

for

- visi šie for loops veikia vienodai

```
x <- c("a", "b", "c", "d", "e", "f")

for (i in 1:6) {
        print(x[i])
}

for (i in seq_along(x)){
        print(x[i])
}

for (raide in x) {
        print(raide)
}

for(i in 1:6) print(x[i])
```

for

- nested for loop
- retai naudojama, sunkiai suprantama, geriau nekišt nagų

```
for (i in seq_len(nrow(x))) {
        for (j in seq_len(ncol(x))) {
                print(x[i,j])
        }

}

seq_len()
```

while

- while testuoja aplinkybes, jeigu ok, atlieka veiksmą, pabaigus vėl testuoja ir t.t.
- while loop gali testis neribotą skaičių iteracijų, tad atsargiai
- galima sutikti, kai bandoma iteruoti optimizavimo uždavinius, kur while (<condition>) yra siekiama vertė

```
count <-0

while(count<5000){
        print(count)
        count <-count+1
}
```

while

- pvz., čia visai neaišku, kada baigsis iteracija

```
z<- 5

while(z>=3 && z<=10){
        print(z)
        #rbinom(n, size, prob)
        coin <- rbinom(1,1,0.5)
        if(coin==1){
                z<- z+1
        }else{
                z<- z-1
        }
}
```

repeat, break, next

- repeat inicializuoja begalinės trukmės loop
- vienintelis būdas sustabdyti, su break
- pavojinga funkcija!

```
x0 <- 1
tol <- 1e-8

repeat{
        x1 <- computeEstimate() #pvz kokia nors optimizavimo ↩
            funkcija
        if(abs(x1-x0)<tol){
                break
        }else{
                x0 <-x1
        }
}
```

repeat, break, next

- next komanda peršoka prie sekančios iteracijos

```
for (i in 1:100) {
        if(i<=30){
                ## skips the first 30
                next
        }
        print(i)

}
```

R Funkcijos

- Užrašome savo pirmą funkciją
- Ką ji daro?

```
add2 <- function(x,y){
        x+y
}

add2(3,5)
```

R Funkcijos

- 2 funkcija
- Ką ji daro?

```
above10 <- function(x){
        use <- x>10
        x[use]
}

c <- seq(1:20)

above10(c)
```

R Funkcijos

- Patobuliname antrą funkciją: 2 argumentai ir antras standartizuotas argumentas

```
above <- function(x,y){
        use <- x>y
        x[use]
}
above(c,6)

above <- function(x,y=10){
        use <- x>y
        x[use]
}

above(c)
above(c,2)
```

R Funkcijos

- Funkcija, kuri apskaičiuoja stulpelių vidurkius:

```
column_mean <- function(y){
        nc <- ncol(y)
        means <- numeric(nc)
        for (i in 1:nc){
                means[i] <-mean(y[,i])
        }
        means
}

library(datasets)
column_mean(airquality)
```

R Funkcijos

- Kai kurių stulepelių nepavyko apskaičiuoti, nes kai kuriuos reikšmės NA
- na.rm=TRUE funkcijoje mean(), leidžia apskaičiuoti įverčius pašalinant NA

```
column_mean <- function(y){
        nc <- ncol(y)
        means <- numeric(nc)
        for (i in 1:nc){
                means[i] <-mean(y[,i], na.rm = TRUE )
        }
        means
}
```

R Funkcijos

- Funkcijos savaime yra R objektai, kuriuos galima perduoti kaip argumentus kitoms funkcijoms
- Funkcijos gali būti *nested* viena į kitą
- Funkcijos rezultatas - paskutinė R išraiška funkcijos viduje
- *Formal arguments* - predefinuoti argumentai, tai palengvina funkcijų naudojimą (pvz., read.csv: sep=",")

```
f <- function (<arguments>){
        ##  funkcijos veikla
}
```

R Funkcijos

-

```
?sd
sd(x, na.rm = FALSE)

data <- rnorm(100)

sd(data)
sd(x=data)
sd(x=data, na.rm = TRUE)
sd(na.rm = TRUE, x=data)
sd(na.rm = TRUE, data)
```

R Funkcijos

- Pozicinis vs leksinis, dalinis funkcijų argumentų *matching*
  - Leksikinis pilnas matching
  - Leksikinis dalinis, bet unikalus matching
  - Pozicinis matching
- Patarimas bent jau pradžioje išrašykite argumentus su jų apibrėžimu, padeda greičiau išmokti ir padarysite mažiau klaidų

```
args(lm)
function (formula, data, subset, weights, na.action, method = "qr↩
    ", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok↩
     = TRUE, contrasts = NULL, offset, ...)

mydata <- data.frame(x=rnorm(1000), y=rnorm(1000))

lm(data=mydata, y~x, model = FALSE, 1:100)
lm(y~x, data=mydata, 1:100, model = FALSE)
lm(y~x, dat=mydata, 1:100, mod = FALSE)
lm (formula= "y~x", data=mydata, subset=1:100,model = FALSE)
plot(lm(formula= "y~x", data=mydata, subset=1:100,model = FALSE))
```

R Funkcijos - Lazy evaluation

- *Lazy evaluation* reiškia, jog argumentai funkcijoje panaudojami tada, kai ir jeigu, jų reikia

```
f<-function(a,b){
          a^2
}
f(2) #positional matching a=2
[1] 4

f <- function(a,b){
        print(a)
        print(b)
}

f(10)

[1] 10
Error in print(b) : argument "b" is missing, with no default
```

R Funkcijos - ...

- ... indikuoja argmentus, kurie perduodami kitai funkcijai
- *Generic functions* naudoja ... *methods* (šiuo metu nesvarbu...)

```
myplot <- function(x,y,type="l",...){
        plot(x,y,type=type,...)
}

plot(mydata$x, mydata$y)
myplot(mydata$x, mydata$y)
```

R Funkcijos - ...

- ... indikuoja argmentus, kurie perduodami kitai funkcijai
- *Generic functions* naudoja ... *methods* (šiuo metu nesvarbu...)

```
myplot <- function(x,y,type="l",...){
        plot(x,y,type=type,...)
}

plot(mydata$x, mydata$y)
myplot(mydata$x, mydata$y)
```

R Funkcijos - ...

- ... arba kai funkcija negali žinoti, kokie argumentai bus pateikti, arba kiek jų
- Tačiau po jų, būtina teisingai išrašyti argumentus

```
args(paste)
function (..., sep = " ", collapse = NULL)

args(cat)
function (..., file = "", sep = " ", fill = FALSE, labels = NULL,
          append = FALSE)
paste("a", "b", "c", sep = ",")
[1] "a,b,c"

paste("a", "b", "c", se = ",")
[1] "a b c ,"
```

Data ir laikas

- Datos turi Date klasę
- Laikas gali būti POSIXct arba POSIXlt klasės
- Data išsaugoma kaip dienų skirtumas lyginant su 1970-01-01
- Laikas išsaugomas kaip sekundžių skirtumas lyginant su 1970-01-01

Data ir laikas

- Character string su data galima paversti į datos klasės objektą

```
x <- as.Date("2019-03-27")

x
[1] "2019-03-27"

class(x)
[1] "Date"

unclass(x)
[1] 17982
```

Data ir laikas

- Laikas gali b8ti POSIXct arba POSIXlt klasės
- POSIXct išsaugo laiką kaip skaičių
- POSIXlt išsaugo laiką kaip *list* su daug papildomos informacijos
- Naudingos funkcijos
  - weekdays
  - months
  - quarters

```
x <- Sys.time()
x
class(x)
p <- as.POSIXlt(x)
p
unclass(p)
names(unclass(p))
p$sec

unclass(x)
```

Data ir laikas

- Komanda strptime padeda iš *charackter string* nuskaityti datą
- Praktikoje patartina geriau naudotis paketai tokiais kaip *lubridate*, *zoo*

```
datestring <- c("2019 January 21, 21:15","2019 February 14, ↩
    14:14")
x <- strptime(datestring, format="%Y %B %d, %H:%M")
x
class(x)

#but not with lithuanian names
datestring <- c("2019 Sausis 21, 21:15","2019 Vasaris 14, ↩
    14:14")
x <- strptime(datestring, format="%Y %B %d, %H:%M")
x
```

Data ir laikas

- Su datomis galima pasižaisti:

```
x <- as.Date("2019-03-27")
class(x)

y <- strptime("2019 January 21, 21:15", format="%Y %B %d, %H:%M")

x-y

as.POSIXct(x)-y
as.POSIXlt(x)-y
```

Data ir laikas

- Su datomis galima pasižaisti:

```
x <- as.Date("2016-02-28"); y <- as.Date("2016-03-01")
y-x
x <- as.Date("2016-02-28"); y <- as.Date("2016-02-29")
y-x

lt <- as.POSIXct("2019-02-27 08:00:00", tz = "EET")
us <- as.POSIXct("2019-02-27 08:00:00", tz = "EST")
us-lt
```

Tvarkingas kodavimas

- Visada rašykite kodą su (plain text) editoriumi
- Naudokite *identing* (8 spaces) (CRTL+I @R)
- Max eilučių ilgis: 80 ženklų
- Apribokite funkcijas: 1 funkcija - 1 operacija