Știința Datelor Laborator 1

ȘD - Scop

Scopul laborataorelor de Știința Datelor este formarea următoarelor capacități:

- Matematice
- Programare
- Procesare a datelor
- De cercetare

ȘD - Scop Laboratorul 1

Scopul laboratarului 1 de Știința Datelor este:

- dezvoltarea capacității de programare în Limbajul R

Limbajul R

Istorie

R se trage din limbajul S, creat în 1976 la BELL, special pentru analiză statistică. Acesta a fost făcut în Fortran. Echivalentul lui din ziua de astăzi este Splus, care este proprietar. Dezvoltarea R a început în 1992 și s-a terminat o dată cu nașterea R pe 29 februarie 2000 . Avantajul lui este că e open-source și a transpus multe părți din Fortran în C. În ziua de astăzi este considerat cel mai specific limbaj pentru statistică.

Părți Importante

IDE-ul de dezvoltare R Studio oferă consolă, editor de fișiere, incorporează managementul de pachete, graficele si documentația. CRAN este arhiva de pachete pentru R. Rd este un mark-up asemănător latext pentru R, în care sunt făcute documentațiile funcțiilor. R conține foarte multe seturi de date incorporate precum celebrele iris și mtcars. Putem apela docuemntația pentru o funcție cu ?functie. Are un depanator incorporat foarte ușor de utilizat, asemănător celui din visual studio code.

Tipuri de bază - Clase atomice

Înainte să intrăm în detalii operatorul de atribuire în R este `<-`, `print` este o funcție de afișare a unui obiect, iar `class` este o funcție ce returnează tipul obiectului.

Caracter Primul tip de bază 'caracter', reprezentat unicode.

```
a <- 'a'
print(a)
```

```
## [1] "a"
```

```r
class(a)
```

```
## [1] "character"
```

Număr real Numere reale sunt reprezentate prin clasa numeric. Ele sunt stocate sub forma IEEE 754 double.

```r
a <- 1.3
print(a)
```

```
## [1] 1.3
```

```r
class(a)
```

```
## [1] "numeric"
```

Număr Întreg Numere intregi necesită litera L ca sufix în notare. Ele sunt memeorate pe 32 de biți.

```r
a <- 1L
print(a)
```

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

```r
class(a)
```

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

Număr complex Numerele complexe sunt reținute ca 2 valori IEEE 754 double.

```r
a <- 1+1i
print(a)
```

```
## [1] 1+1i
```

```r
class(a)
```

```
## [1] "complex"
```

Element logic Elemente logice au două valori: TRUE și FALSE.

```r
a <- TRUE
print(a)
```

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

```r
class(a)
```

```
## [1] "logical"
```

```r
summary(a)
```

```
##    Mode    TRUE
## logical       1
```

Clase de bază

Vector Prima clasă de bază este Vector. Aceasta poate stoca mai multe elemente de același tip sau clasă.

```r
a <- c(1,2,3)
print(a)
```

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

```r
class(a)
```

```
## [1] "numeric"
```

```r
b <- c('a','b','c')
print(b)
```

```
## [1] "a" "b" "c"
```

```r
class(b)
```

```
## [1] "character"
```

```r
summary(b)
```

```
##    Length     Class      Mode
##         3 character character
```

În caz de folosire a mai multor tipuri se va alege cel mai cuprinzător dintre ele. Prin cuprinzător se referă la tipul la care tuturor celălalte li se poate face conversia.

```r
a <- c(1,'a',3L,TRUE)
print(a)
```

```
## [1] "1"    "a"    "3"    "TRUE"
```

```r
class(a)
```

```
## [1] "character"
```

```r
a[4] == TRUE
```

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

```r
summary(a)
```

```
##    Length     Class      Mode
##         4 character character
```

Listă Dacă vrem să punem într-o grupare tipuri diferite, putem folosi clasa listă.

```r
a <- list(1,'a',3L,TRUE)
print(a)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] TRUE
```

```r
class(a)
```

```
## [1] "list"
```

```r
a[4] == TRUE
```

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

```r
summary(a)
```

```
##      Length Class  Mode
## [1,] 1      -none- numeric
## [2,] 1      -none- character
## [3,] 1      -none- numeric
## [4,] 1      -none- logical
```

Matrice Matricea, asemenea vectorilor, poate avea doar elemente de același tip.

```r
a <- matrix(c('a','b','c','d','e','f'),nrow=2,ncol=3)
print(a)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "c"  "e"
## [2,] "b"  "d"  "f"
```

```r
class(a)
```

```
## [1] "matrix" "array"
```

```r
summary(a)
```

```
##       V1                 V2                 V3
##  Length:2           Length:2           Length:2
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
```

Cadru de date Cadrul de date este o listă de vectori de lungimi egale.

```r
a <- data.frame(c('a','b','c','d','e','f'),1:6)
print(a)
```

```
##   c..a....b....c....d....e....f.. X1.6
## 1                               a    1
## 2                               b    2
## 3                               c    3
## 4                               d    4
## 5                               e    5
## 6                               f    6
```

```r
class(a)
```

```
## [1] "data.frame"
```

```r
summary(a)
```

```
##  c..a....b....c....d....e....f..      X1.6
##  Length:6                        Min.   :1.00
##  Class :character                1st Qu.:2.25
##  Mode  :character                Median :3.50
```

```
##                                 Mean   :3.50
##                                 3rd Qu.:4.75
##                                 Max.   :6.00
```

```
a <- data.frame(a=c('a','b','c','d','e','f'),b=1:6)
print(a)
```

```
##   a b
## 1 a 1
## 2 b 2
## 3 c 3
## 4 d 4
## 5 e 5
## 6 f 6
```

```
class(a)
```

```
## [1] "data.frame"
```

```
summary(a)
```

```
##       a                   b
##  Length:6           Min.   :1.00
##  Class :character   1st Qu.:2.25
##  Mode  :character   Median :3.50
##                     Mean   :3.50
##                     3rd Qu.:4.75
##                     Max.   :6.00
```

Factori O clasă specială este cea de factori. Acestia se folosesc atunci când avem un numar finit de valori text într-un vector. Pentru îmbunătățirea performanței, fiecare valoare textuala va primi o reprezentare numerică 1,2 . . . .

```
a <- c('a','b','c','d','e','f')
print(a)
```

```
## [1] "a" "b" "c" "d" "e" "f"
```

```
class(a)
```

```
## [1] "character"
```

```
summary(a)
```

```
##    Length     Class      Mode
##         6 character character
```

```
b <- factor(a)
print(b)
```

```
## [1] a b c d e f
## Levels: a b c d e f
```

```
class(b)
```

```
## [1] "factor"
```

```
summary(b)
```

```
## a b c d e f
```

```
## 1 1 1 1 1 1
```

Operații

Operațiile în interiorul R sunt asemănătoare celor din limbaje obișnuite, dar în același timp există și operații vectoriale, precum în MatLab / Octave.

```
a <- 1
b <- 2
a+b
```

Operații simple

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

```
a/b
```

```
## [1] 0.5
```

```
a%%b
```

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

```
a-b
```

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

```
a*b
```

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

```
a<b
```

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

```
a==b
```

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

```
a>b
```

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

```
1:6
```

Operații vectoriale

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

```
a <- c(1,2,3)
b <- c(4,5,6)
a+b
```

```
## [1] 5 7 9
```

```
a/b
```

```
## [1] 0.25 0.40 0.50
```

```
a%%b
```

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

```
a-b
```

```
## [1] -3 -3 -3
```

```
a*b
```

```
## [1]  4 10 18
```

```
a*3
```

```
## [1] 3 6 9
```

```
a<b
```

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

```
a==b
```

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

```
a>b
```

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

```
a <- matrix(1:6,nrow=2,ncol=3)
b <- matrix(11:6,nrow=2,ncol=3)
a+b
```

```
##      [,1] [,2] [,3]
## [1,]   12   12   12
## [2,]   12   12   12
```

```
a/b
```

```
##            [,1]      [,2]      [,3]
## [1,] 0.09090909 0.3333333 0.7142857
## [2,] 0.20000000 0.5000000 1.0000000
```

```
a%%b
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    0
```

```
a-b
```

```
##      [,1] [,2] [,3]
## [1,]  -10   -6   -2
## [2,]   -8   -4    0
```

```
a*b
```

```
##      [,1] [,2] [,3]
## [1,]   11   27   35
## [2,]   20   32   36
```

```
a*4
```

```
##      [,1] [,2] [,3]
## [1,]    4   12   20
## [2,]    8   16   24
```

```
a<b
```

```
##      [,1] [,2]  [,3]
## [1,] TRUE TRUE  TRUE
## [2,] TRUE TRUE FALSE
```

```
a==b
```

```
##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,] FALSE FALSE  TRUE
```

```
a>b
```

```
##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE
```

```
a <- matrix(c(1:4),nrow=2,ncol=2)
b <- matrix(c(7:10),nrow=2,ncol=2)
a%*%b
```

```
##      [,1] [,2]
## [1,]   31   39
## [2,]   46   58
```

Selectie

Un lucru important în știința datelor este selectarea unor fragmente din date. Pentru aceasta, exista diferiti operatori.

```
?summary
```

Operatorul [] Operatorul [] returnează un subset conform condițiilor din interior, având aceeași clasă ca obiectul inițial.

```
a <- list('a',1,'c',4,'e',6)
print(a[1])
```

```
## [[1]]
## [1] "a"
```

```
class(a[1])
```

```
## [1] "list"
```

```
summary(a[1])
```

```
##      Length Class  Mode
## [1,] 1      -none- character
```

Se pot folosi indecși negativi care reprezintă indecsii elementelor care nu vor fi luate în considerare

```
a <- list('a',1,'c',4,'e',6)
print(a[-1])
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "c"
```

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

Operatorul [[]] și $ Operatorii [[]] și $ sunt asemănători cu operatorul [], dar in acest caz valoarea de return va avea clasa elementului.

```
a <- list('a',1,'c',4,'e',6)
print(a[[1]])
```

```
## [1] "a"
```

```
class(a[[1]])
```

```
## [1] "character"
```

```
summary(a[[1]])
```

```
##    Length     Class      Mode
##         1 character character
```

Subnetare Logică În mod implicit condiția din interiorul operatorilor se referă la indecși selectați, dar în același timp poate fi un vector de elemente logice

```
a <- 1:6
print(a[a<3])
```

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

```
a <- 6:1
print(a)
```

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

```
print(sort(a))
```

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

```
print(order(a))
```

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

```
print(a[order(a)])
```

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

NA/NAN/Inf În R există anumite valori speciale pe care nu dorim să le considerăm în operațiile noastre asupra datelor.

```
a <- list('a',1,3,NA,NaN,Inf)
b <- c(5,1,3,NA,NaN,Inf)
d <- c('a',1,3,NA,NaN,Inf)
print(a)
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] NA
##
## [[5]]
## [1] NaN
##
## [[6]]
## [1] Inf
```

```
0/0
```

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

```
Inf-Inf
```

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

```
NA + 1
```

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

```
NA * 0
```

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

```
NaN * 0
```

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

```
Inf * 0
```

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

```
NA==NaN
```

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

```
is.na(a)
```

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

```
is.nan(b)
```

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

```
is.infinite(b)
```

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

```
is.nan(d)
```

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

```r
is.infinite(d)
```

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

```r
is.nan(NaN)
```

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

```r
is.infinite(Inf)
```

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

Secvențe de Control

Ca în majoritatea limbajelor R deține secvențe de control care nu sunt indicate spre utilizare doar ca ultim resort când nu putem folosi operații vectoriale sau funcții predefinite. #### IF if (condiție1) {} else if (condiție2) {} else {}

```r
a <- 2
if (a<1) {print(a*2)} else if (a>3) {print(a*3)} else {print(a/2)}
```

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

FOR for(x in X) {} – poate fi utilizat next

```r
a <- 1:6
for(elem in a) {
  if(elem<3) next
  print(elem)
}
```

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

```r
for(elem in a) {
  print(elem)
}
```

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

```r
for(elem in a) {
  if(elem==3) break
  print(elem)
}
```

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

WHILE while(condiție1) {}

```
a <- 2
b <- 4
while(a<b) {
  a <- a+2
  b <- b+1
}
print(a)
```

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

REPEAT repeat {} – utilizare break

```
a <- 2
b <- 4
repeat {
  a <- a+2
  b <- b+1
  if(a>=b) {
    break
  }
}
print(a)
```

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

Funcții

Un mod de a modulariza codulul este prin intermediul funcțiilor.

```
my_sum <- function(a,b) {
  a <- a+1
  b <- b/2
  a+b
}
my_sum(4,6)
```

Definitii ale funcțiilor

```
## [1] 8
```

```
my_iter <- function(a,b) {
  a <- a*2
  a+3
}
my_iter(4)
```

```
## [1] 11
```

```
my_iter(4,5)
```

```
## [1] 11
```

Valori implicite.

```
my_sum_2 <- function(a,b=2) {
  a <- a+1
  b <- b/2
  a+b
}
my_sum_2(4)
```

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

```
my_iter_2 <- function(a=3,b) {
  a <- a*2
  a+3
}
my_iter_2(4)
```

```
## [1] 11
```

```
my_iter_2(4,5)
```

```
## [1] 11
```

```
my_iter_2(b=4)
```

```
## [1] 9
```

```
formals(my_sum_2)
```

```
## $a
##
##
## $b
## [1] 2
```

Argumentul ... .

```
my_sum_3 <- function(a,...) {
  a <- a+1
  b <- list(...)
  print(b)
  a + b[[1]]
}
my_sum_3(4,5,6,a=7,8)
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 6
##
## [[4]]
## [1] 8
```

```
## [1] 12
```

```
my_sum_4 <- function(..., a) {
  a <- a+1
```

```
  b <- list(...)
  a + b[[1]]
}
my_sum_4(4,5,6,a=7,8)
```

```
## [1] 12
```

După parametrul ..., este recomandat să avem valori implicite.

```
my_sum_4 <- function(..., a=5) {
  a <- a+1
  b <- list(...)
  a + b[[1]]
}
my_sum_4(4,5,6,a=7,8)
```

```
## [1] 12
```

```
my_sum_4(4,5,6,7,8)
```

```
## [1] 10
```

```
a <- 1:10
max(a)
```

Funcții predefinite

```
## [1] 10
```

```
sum(a)
```

```
## [1] 55
```

```
min(a)
```

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

```
median(a)
```

```
## [1] 5.5
```

Funcții iterative lapply(listă, funcție) – aplică funcția asupra fiecărui element al listei (returnează listă mereu)

```
iter <- function(a) {a+1}
a <- list(1,2,3,4,5,6)
print(a)
```

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

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

```
lapply(a, iter)
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 4
##
## [[4]]
## [1] 5
##
## [[5]]
## [1] 6
##
## [[6]]
## [1] 7
```

```
a <- list(1:6)
print(a)
```

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

```
lapply(a, iter)
```

```
## [[1]]
## [1] 2 3 4 5 6 7
```

sapply(listă,funcție) – analog lapply, dar poate returna vector, matrice sau listă

```
a <- list(1,2,3,4,5,6)
print(a)
```

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

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

```
sapply(a, iter)
```

```
## [1] 2 3 4 5 6 7
```

```
a <- list(1:6)
print(a)
```

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

```
sapply(a, iter)
```

```
##      [,1]
## [1,]    2
## [2,]    3
## [3,]    4
## [4,]    5
## [5,]    6
## [6,]    7
```

apply(vector/matrice, rânduri/coloana, funcție) – c(1,3) păstrează dimensiunea 1 și 3 a unei matrice cel puțin tridimensională

```
a <- matrix(c(1:6),nrow=2,ncol=3)
print(a)
```

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

```
apply(a,1, iter)
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    4    5
## [3,]    6    7
```

```
apply(a,2, iter)
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    3    5    7
```

mapply(funcție, vector, vector) – aplică o funcție având ca argumente un element din x și un element din y

```
a <- 1:6
b <- 3:8
mapply(sum,a, b)
```

```
## [1]  4  6  8 10 12 14
```

tapply(vector, factori, funcție) – aplică funcția grupând elementele vectorului în funcție de factori

```
a <- 1:6
tapply(a,a<3, iter)
```

```
## $`FALSE`
## [1] 4 5 6 7
##
## $`TRUE`
## [1] 2 3
```

split(vector/data.frame/list,matrice, factori) – imparte vectorul in elemente de listă în funcție de factori (list(factori,factori))

```
a <- 1:6
split(a,a<3)
```

```
## $`FALSE`
## [1] 3 4 5 6
##
## $`TRUE`
## [1] 1 2
```

```
split(a,list(a<3,a%%2==0))
```

```
## $FALSE.FALSE
## [1] 3 5
##
## $TRUE.FALSE
## [1] 1
##
## $FALSE.TRUE
## [1] 4 6
##
## $TRUE.TRUE
## [1] 2
```

Funcții anonime Pentru funcțiile predefinite, putem folosi într-un anumit moment funcții anonime.

```
a <- list(1,2,3,4,5,6)
print(a)
```

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

```
lapply(a, function(x){x*2})
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 6
##
## [[4]]
## [1] 8
##
## [[5]]
## [1] 10
##
## [[6]]
## [1] 12
```

Operatori Putem defini noi operatori binari utilizând formatul %nume_operator% si definind funcția pe care o execută operatorul asupra "%p%". Atenție! La definire trebuie să folosim ghilimele, iar la utilizare putem folosi formatul inițial.

```
a <- 1
b <- 2
"%p%" <- function(x,y) {x*2+y}
a %p% b
```

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

Simulare/Experiment

Foarte important în procesul științific este reproducerea experimentelor și simulăriilor. Adeseori pentru a nu luat în cosniderare toate cazuriilor se alege un caz aleator sau se generează numere aleatorii când nu avem date de intrare precise. Pentru a avea un experiment ce poate fi reprodus se folosește funcția de setare a seed-ului:

```
set.seed(10)
```

Dacă dorim să generăm 10 numere aleatorii cu media 0 și deviația standard 1 folosim funcție:

```
rnorm(10, mean=0, sd=1)
```

```
##  [1]  0.01874617 -0.18425254 -1.37133055 -0.59916772  0.29454513  0.38979430
##  [7] -1.20807618 -0.36367602 -1.62667268 -0.25647839
```

Dacă dorim să alege un număr de 3 valorii aleatoriu dintr-un vector fără înlocuire:

```
sample(1:10, 3, replace=FALSE)
```

```
## [1] 8 7 6
```

În caz de vă loviți de erori puteți folosi funcțiile traceback() și debug(funcție).

Exercitii

1

Creati un vector x cu 20 de elemente, folosind functia rnorm. Hint: Puteti folosi mean=0, sd=0.5.

2

Construiti un vector cu elementele din x de la pozițiile 3, 5, 7, 11, 13, 19.

3

Construiti un vector cu elementele din x, mai puțin cele de la pozițiile 1, 5, 7, 16, 18. Hint: Folositi indecsi negativi

4

Simulati 4 aruncari cu zarul, folosind functia 'sample'. Rezultatele vor fi afisate intr-un vector, ca rezultat al functiei. Hint: Parametrul replace, setat pe True, permite repetarea unei valori random.

5

Sa presupunem ca vrem sa simulam aruncarea unei monede inechitabile, cu 2 fete. Aceasta moneda are o probabilitate de 0.3 pentru 'ban' si o probabilitate de 0.7 pentru 'stema'. Reprezentam 'banul' cu 0 si 'stema' cu 1. Construiti, folosind functia 'sample', un vector care simuleaza cele 100 de aruncari. Hint: In functia 'sample' se poate adauga un parametru extra, numit 'prob'.

6

Se da urmatoarea lista:

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
```

Folosind lapply, calculati media pentru a, beta si logic.

7

Se da codul de mai jos:

```
A<-c(1:9)
B<-c(1:12)
C<-c(1:15)
my.lst<-list(A,B,C)
```

Folosind sapply, creati o noua lista de tipul 2 * x, unde x reprezinta elementele din my.lst.