Adrian DUȘA
Bogdan OANCEA
Nicoleta CARAGEA
Ciprian ALEXANDRU
Nicolae Marius JULA
Ana Maria DOBRE

R cu aplicații în statistică

Referenți științifici:
Prof.univ.dr. Elena Druică
Universitatea din București
Prof.univ.dr. Tudorel Andrei
Academia de Studii Economice
Prof.univ.dr. Călin Vâlsan
Bishop's University, Canada
William's School of Business

Șos. Panduri nr. 90-92, 050663 București – ROMÂNIA
Tel./Fax: +40 214102384
E-mail: editura.unibuc@gmail.com
Internet: http://editura-unibuc.ro
Centru de vânzare:
Bd. Regina Elisabeta nr. 4-12,
030018 București – ROMÂNIA
Tel. +40 213053703

Tehnoredactare: ADRIAN DUȘA
Copertă: MARIUS JULA

Descrierea CIP a Bibliotecii Naționale a României
R cu aplicații în statistică / Adrian Dușa, Bogdan Oancea,
Nicoleta Caragea, … - București : Editura Universității din București,
2015
Conține bibliografie
Index
ISBN 978-606-16-0643-6
I. Dușa, Adrian
II. Oancea, Bogdan
III. Caragea, Nicoleta
004:311

Cuprins

Capitolul 1

Introducere

Nevoia de informații, din ce în ce mai diverse și mai complexe, dar și posibilitățile de calcul avansat cu ajutorul soft-urilor tot mai performante, au condus la crearea unui bazin imens de date care pot fi cu ușurință exploatate pe baza analizei statistice. Mediul R a devenit, în prezent, unul dintre cele mai utilizate instrumente de analiză statistică, fiind utilizat în mediile universitare și de cercetare academică, dar și în mediul de afaceri. Acest manual este destinat tuturor celor care doresc să învețe statistica, fiind un material introductiv de studiu, care prezintă un spectru larg de exemple, prezentări grafice și analiză a datelor, dezvoltate cu ajutorul R.

1.1 Ce este R?

Mediul R este un instrument de analiză date conceput de statisticieni, pentru statisticieni.
R este un sistem pentru analize statistice și reprezentare grafică creat de către Ross Ihaka și Robert Gentleman, profesori de statistică la Universitatea Auckland din Noua Zeelandă[1].
R este considerat un dialect al limbajului S creat de AT&T Bell Laboratories. S este disponibil sub forma software-ului S-PLUS, comercializat de compania Insightful. Există diferențe importante între cele două limbaje, R și S: acestea sunt documentate de către Ihaka & Gentleman (1996) sau se regăsesc în R-FAQ[2]. Astfel, numele limbajului R provine de la inițiala prenumelui creatorilor, dar este totodată și un omagiu adus limbajului S.

[1] Ihaka R. & Gentleman R. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5: 299–314.

[2] R-FAQ

1.2 Avantajele mediului R

În primul rând, R este open-source, fiind distribuit în mod gratuit sub licență *GNU - General Public Licence*[3]; dezvoltarea și distribuirea sunt în grija câtorva profesori și statisticieni, afiliați companiilor și universităților, cunoscuți sub denumirea generică de *R Development Core Team*.
Conform filosofiei *GNU*[4], software-ul open-source este caracterizat de libertatea acordată utilizatorilor săi de a-l utiliza, copia, distribui, studia, modifica și îmbunătăți. Mai exact, este vorba de patru forme de libertate acordate utilizatorilor:

- Libertatea de a utiliza programul, în orice scop (libertatea 0);
- Libertatea de a studia modul de funcționare a programului, și de a-l adapta nevoilor proprii (libertatea 1).
  Accesul la codul-sursă este o precondiție pentru aceasta;
- Libertatea de a redistribui copii, în scopul ajutorării aproapelui tău (libertatea 2);
- Libertatea de a îmbunătăți programul, și de a pune îmbunătățirile la dispoziția publicului, în folosul întregii societăți (libertatea 3). Accesul la codul-sursă este o precondiție pentru aceasta.

Faptul că este gratuit atrage automat avantajul competitiv în fața altor software-uri de analiză statistică, precum Stata, SAS și SPSS. Astfel, costurile alocate licenței de software dispar.
R este denumit de către Norman Nie, unul dintre fondatorii SPSS și CEO al Revolution Analytics, "cel mai puternic și flexibil limbaj de programare statistică din lume" (în engleză *"the most powerful and flexible statistical programming language in the world"*).[5] Dovadă a succesului pe care R îl are în știința datelor, s-au dezvoltat medii de integrare a acestuia în SAS și chiar SPSS. Este vorba despre modulul SAS/IML[6], care integrează limbajul R în SAS, și despre *translate2R*, un serviciu de translatare a codului SPSS direct în R dezvoltat de compania

[3] GNU
[4] GNU Philosophy
[5] Smith, D., 2010,"R is Hot", Revolution Analytics
[6] SAS/IML Module

eoda[7].
Totodată, R are susținerea comunității academice și științifice internaționale , dar și a multor companii internaționale. Din ce în ce mai multe companii utilizează R ca instrument de analiză date. Dintre acestea, menționăm: Google, Facebook, Mozilla, Twitter, The New York Times, The Economist, NewScientist, Lloyd's, Bing, Johnson&Johnson, Pfizer, Shell, Bank of America, Ford.[8]
R este susținut și de mediul academic. Marile universități din lume sprijină R, la fel cum sprijină și alte inițiative sau software-uri open-source, precum sistemul de operare Linux sau sistemul de preparare a documentelor LaTeX.
Alte puncte forte sunt flexibilitatea limbajului și reproductibilitatea analizelor. Utilizatorii pot întreprinde analize aplicate pe diferite seturi de date utilizând aceleași instrumente și același know-how.
Cu siguranță, cititorii vor descoperi singuri și alte avantaje.

1.3 Cum funcționează R

La prima vedere, mediul R poate părea prea complex pentru un non-specialist sau pentru un non-programator. În realitate, lucrurile se prezintă diferit. O caracteristică de seamă a mediului R este chiar flexibilitatea sa. În timp ce un software clasic afișează instantaneu rezultatele unei analize, R memorează aceste rezultate într-un obiect, astfel că o analiza poate fi efectuată fără afișarea vreunui rezultat. Utilizatorul poate fi surprins în această situație, însă o asemenea particularitate se poate dovedi foarte utilă. Se poate extrage doar partea din rezultat care prezintă interes. Spre exemplu, dacă utilizatorul execută o serie de 20 de analize de regresie și vrea să compare diferiți coeficienți de regresie, R poate afișa numai coeficienții estimați: astfel rezultatul poate avea o singură linie, în timp ce un software clasic poate deschide 20 de ferestre cu rezultate.

R este un limbaj obiectual (în engleză *object-oriented*). Toate comenzile din R sunt executate asupra obiectelor reținute în memoria activă a computerului; nu sunt utilizate fișiere temporare. Citirea și scrierea fișierelor sunt utilizate pentru input-ul și output-ul datelor și al rezultatelor. Utilizatorul execută analiza prin intermediul comenzilor. Rezultatele sunt afișate direct pe ecran, memorate într-un obiect, sau scrise pe disc (valabil de exemplu pentru grafice). Din moment ce

[7] translate2R - eoda
[8] Revolution Analytics, "Companies Using R"

rezultatele sunt obiecte, pot fi considerate date și analizate ca atare. Fișierele de date pot fi citite de pe calculatorul local sau de pe un server la distanță prin Internet.

De ce este R diferit de alte limbaje de programare?

În primul rând, R este un limbaj interpretat, nu unul compilat, ceea ce presupune că toate comenzile introduse prin tastatură sunt direct executate fără să fie necesară redactarea unui program complet, așa cum se procedează în majoritatea limbajelor de programare (C, Fortran etc.). În al doilea rând, sintaxa este foarte simplă și intuitivă. Spre exemplu, o regresie liniară poate fi efectuată cu ajutorul comenzii `lm(y ~ x)` care înseamnă "adecvarea modelului liniar având y ca variabilă de răspuns și x ca predictor". Pe parcursul cărții se vor analiza și alte exemple ce ilustrează flexibilitatea limbajului R.

Capitolul 2

Instalarea R

2.1 Mediul R

În această carte se studiază paşii de instalare a mediului R pentru sistemul de operare Windows. R este, însă, disponibil pentru orice sistem de operare, mai exact pentru o gamă largă de platforme UNIX, Windows și MacOS. R poate fi descărcat de pe site-ul CRAN (Comprehensive R Archive Network): `http://cran.r-project.org/`.

Pentru a instala R, se parcurg următoarele etape:

1. Se accesează website-ul R Project: `http://www.r-project.org`
2. Se selectează un *CRAN Mirror* din lista disponibilă, recomandabil a fi cel mai apropiat de locația utilizatorului (Ungaria sau Austria în cazul României);
3. Se descărcă versiunea *base* a mediului R, pentru sistemul de operare Windows, versiunea curentă fiind R 3.2.2;
4. Se execută fișierul R-3.2.2-win.exe ca la o instalare obișnuită de software.

La instalare, se va selecta automat versiunea de 32MB/64MB, în funcție de versiunea sistemului de operare.
Pentru a actualiza versiunea de R, se vor urma etapele descrise anterior. Instalarea mai multor versiuni de R pe același calculator nu reprezintă o problemă. Ele se vor instala în același folder, sub diferite subfoldere, și nu se vor afecta una pe cealaltă.

2.2 Prima sesiune de lucru în R

Pornind mediul R, se va deschide fereastra principală RGui cu sub-fereastra R Console, ce conține, printre altele, informații privind versiunea, copyright-ul, condițiile de redistribuire:

```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

O altă informație utilă este și modul în care se citează limbajul R în publicații, ce se poate obține astfel:

```
> citation()
```

Informațiile necesare pentru citarea pachetelor se vor obține, similar, prin comanda:

```
> citation("nume_pachet")
```

Se recomandă citarea limbajului și a pachetelor în orice publicație în care sunt menționate.
În Consola R, prompter-ul `>` așteaptă să se introducă instrucțiuni sau comenzi. După introducerea unei comenzi, executarea acesteia se va face în consolă cu tasta `Enter`. Spre exemplu, R poate fi utilizat ca și calculator:

```
> 2*5
[1] 10
> 1+sin(15)
[1] 1.650288
> 8 * 10 /2
[1] 40
```

Ordinea efectuării operațiilor respectă regulile matematice.
Dacă în alte programe de analiză statistică utilizatorii sunt obișnuiți să vizualizeze permanent datele de intrare și de ieșire, în R se lucrează cu acestea sub formă de obiecte. Astfel, datele de intrare și rezultatele analizelor pot fi memorate în obiecte, utilizând operatorul de atribuire (`<-`).
În realitate, se pot utiliza și alți operatori de atribuire. Aceștia pot reprezenta atribuirea valorii din dreapta către elementul din stânga, precum în exemplul de mai jos:

```
> variabila1 = 6
> variabila2 <- 8
```

De asemenea, atribuirea poate avea loc și în sens invers, când elementul din dreapta ia valoarea din stânga:

```
> 10 -> variabila3
```

Cu toate acestea, este recomandabil să se utilizeze operatorul `<-`, lăsând spațiu în jurul lui.

Pentru a afișa un obiect, se introduce numele lui, precum în exemplul de mai jos:

```
> x <- 2
> x
[1] 2
```

O alternativă pentru afișare este utilizarea funcției `print()`:

```
> print(x)
[1] 2
```

sau chiar utilizarea parantezelor, cu ajutorul cărora se realizează operațiunea dublă de atribuire și afișare pe ecran:

```
> (x <- 2)
[1] 2
```

În mod convențional, se va utiliza varianta cu paranteze pe tot parcursul acestei cărți.

Există câteva restricții atunci când se alege numele unui obiect, respectiv numele unei variabile:

- Numele obiectelor nu pot conține simboluri (`!`, `+`, `-`, \, #);
- Se recomandă utilizarea simbolurilor `.` sau _ în separarea a două cuvinte din numele unui obiect;
- Numele obiectelor pot conține numere, însă nu la începutul lor;
- R este un limbaj sensibil la majuscule (în engleză *case-sensitive*), astfel că `X` și `x` sunt două obiecte diferite.

Aceste recomandări sunt ilustrate în exemplele următoare. Mai jos sunt redate câteva reguli de bună practică:

```
> exemplu1 <- 1
> exemplul_ <- 10
> exemplu_1 <- 20
> o.variabila <- 30
```

În continuare, câteva exemple de utilizări eronate:

```
> 1exemplu <- 10
Error: unexpected input in "1ex"
> _exemplu <- 10
Error: unexpected input in "_"
> exemplu! <- 3
Error: unexpected '!' in "exemplu!"
# R este un limbaj sensibil la majuscule
> Varsta <- 10
> varsta
Error: object 'varsta' not found
```

Se observă că se va afișa eroare atunci când numele unui obiect începe cu simbolul _ sau cu un număr.
O altă particularitate a limbajului ilustrată în exemplul anterior este aceea că liniile care încep cu simbolul # sunt interpretate pe post de comentarii; ele nu sunt executate drept comenzi. Sunt utile pentru a formula explicații referitoare la analiza realizată direct în cadrul codului. R va interpreta drept comenzi orice este scris în dreapta semnului #.

Pentru a înțelege filosofia din spatele limbajului R în ceea ce privește stocarea obiectelor în memorie, se va urmări exemplul următor.
Se definește un obiect `x`, iar valoarea lui se atribuie ulterior obiectului `a`.

```
> (x <- 2)
[1] 2
> a <- x
```

Ulterior, valoarea lui `x` a fost modificată; cu toate acestea, valoarea lui `a` rămâne neschimbată.

```
> x
[1] 2
> a
[1] 2
> x <- 5
> a
[1] 2
```

În concluzie, atunci când unui obiect deja existent i se atribuie o valoare, conținutul acestuia va fi rescris cu noua valoare.
Dacă o comandă nu este completă la sfârșitul unui rând de la prompter-ul `>`, R va lansa un nou prompter, și anume simbolul `+` în linia sau liniile următoare. Acest simbol dispare atunci când comanda este completă:

```
> x <- 10
> y <- 15
> z <- x*
+ y
> z
[1] 150
```

Funcția `ls()` va lista toate obiectele din memorie:

```
> ls()
[1] "x" "y" "z"
```

O altă funcție utilă este `rm()`. Aceasta se utiliează pentru ştergerea unui obiect din memorie, ca de exemplu ștergerea lui `z`:

```
> rm(z)
> ls()
[1] "x" "y"
```

Un element de utilitate este ușurința cu care se poate naviga în istoricul comenzilor utilizate, prin folosirea tastelor cu săgeți ↑ și ↓. Astfel, la

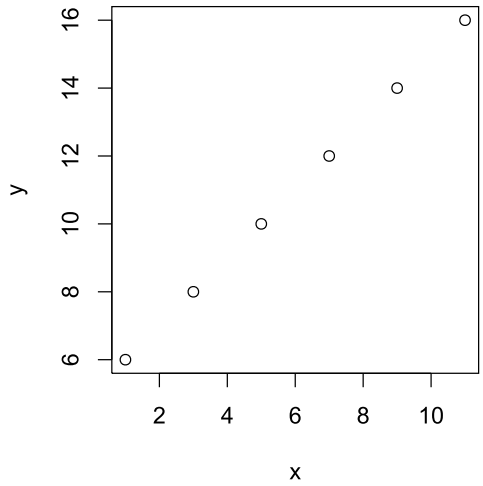
tastarea ↑, va apărea ultima comandă introdusă.

Pentru a încheia prima sesiune de lucru în R, se vor crea două obiecte și se vor reprezenta grafic.

```
> x <- c(1,3,5,7,9,11)
> y <- c(6,8,10,12,14,16)
> plot(x,y)
```

În urma acestor comenzi, va rezulta Figura 2.1.

Figura 2.1: Un prim grafic



O sesiune de R se va încheia prin comanda `q()` sau din meniul File -> Exit. La încheierea sesiunii, se observă că se poate salva atât script-ul (fișier cu extensia `.R`), cât și datele existente în spațiul de lucru (în engleză *workspace image*).

Capitolul 3

Pachete

R a devenit un standard pentru noua teorie statistică, în special în contextul importanței pe care o capătă reproductibilitatea rezultatelor unei cercetări sau analize (în engleză *Reproducible Research*). El este alcătuit dintr-o serie de instrumente de modelare, analiză și vizualizare a datelor. Acestea sunt organizate în biblioteci de funcții numite *pachete* (în engleză *packages*). Majoritatea acestor pachete se găsesc pe rețeaua de servere CRAN (The Comprehensive R Archive Network)[1]. Aceasta cuprinde servere ftp și web care stochează versiuni de cod si documentații R identice și actualizate. Utilizatorul va selecta *CRAN mirror* cea mai apropiată de localizarea sa geografică pentru a facilita conexiunea la server.
R este structurat în două părți conceptuale: pachete de bază și pachete contribuite (instalate de pe CRAN sau alte servere - GitHub, Bioconductor).

3.1 Tipuri de pachete

În R există două tipuri de pachete: pachetele de bază și pachetele contribuite. Pachetele de bază (în engleză *base packages*) sunt cele încărcate automat în memorie atunci când R este pornit. Acestea sunt următoarele: `base`, `compiler`, `datasets`, `graphics`, `grDevices`, `grid`, `methods`, `parallel`, `splines`, `stats`, `stats4`, `tcltk`.

Pachetele contribuite (în engleză *contributed packages*) pot fi utilizate numai după ce sunt încărcate în memorie. Câteva dintre aceste pachete sunt recomandate întrucât acoperă metodele statistice frecvent utilizate

[1] http://cran.r-project.org/

în analiza de date: `KernSmooth`, `MASS`, `Matrix`, `boot`, `class`, `cluster`, `codetools`, `foreign`, `lattice`, `mgcv`, `nlme`, `nnet`, `rpart`, `spatial`, `survival`.
Concret, instalarea unui pachet se realizează prin următoarea comandă:

```
> install.packages("spatial")
```

și încărcarea în memorie a unui pachet (de exemplu pachetul `spatial`)

Pentru a utiliza un anumit pachet, acesta trebuie încărcat în memorie, fie prin funcția `library()`, fie prin funcţia `require()`:

```
> library(spatial)
> require(spatial)
```

Funcția `library()` poate fi utilizată și pentru a vizualiza toate pachetele disponibile pentru a fi încărcate în memorie, iar funcția `search()` va lista toate pachetele încărcate deja în memorie:

```
> library()
> search()
```

În Consola R, instalarea pachetelor se poate face și din meniul *Packages*, cu opțiunea *Install Package(s)*. Opțiunea *Load Packages* va încărca pachete deja instalate. Opțiunea *Select Repositories* va selecta unul din serverele pe care se regăsesc pachete: CRAN, BioC, Omegahat sau R-Forge. Instalarea propriu-zisă a pachetelor se execută din opțiunea *Install.packages*, sau prin varianta de instalare din fișiere zip.
Pe lângă funcții, pachetele conțin și alte elemente și informații: versiunea, licența, autorul, o descriere, documentații și seturi de date. Pentru a accesa seturile de date dintr-un pachet (în exemplul de mai jos, pachetul `nlme`), se utilizează următoarea funcție:

```
> data(package = "nlme")
```

Accesarea documentației unui pachet se realizează prin funcția `help()`:

```
> help(nlme)
```

Pentru a elimina din memorie un pachet încărcat, se folosește funcția `detach()`:

```
> detach(package:nlme)
```

Se observă că numele pachetului se scrie în acest caz sub forma `package:nume_pachet`.
Actualizarea pachetelor se face prin comanda `update.packages()`, care va genera o listă de versiuni actuale disponibile ale pachetelor instalate şi va cere consimțământul utilizatorului prin `y/N/c` (Yes/No/Cancel):

```
> update.packages()
manipulate :
 Version 0.98.1103 installed in C:/R/R-3.2.0rc/library
 Version 1.0.1 available at http://cran.at.r-project.org
Update (y/N/c)?
```

Unele pachete sunt dependente de alte pachete. În acest caz, pachetele de care sunt dependente se vor instala fie automat, fie după acceptul dat de utilizator printr-o căsuță de dialog.
Pachetele sunt publicate și distribuite numai după un amplu proces de peer-review, realizat de către statisticieni și cercetători, membri ai *CRAN*. Sistemul de pachete, împreună cu infrastructura *CRAN*, pune la dispoziție un proces standardizat de documentare, validare, construire și distribuție a pachetelor către milioane de utilizatori din toată lumea. În prezent, pe CRAN se regăsesc 6635 pachete[2], creșterea acestora fiind exponențială de la an la an[3]. Multe organizații au nevoie de un server (în engleză *mirror*) al CRAN care să conțină numai o parte din pachete, relevante pentru activitatea lor. Venind în întâmpinarea acestei nevoi, a fost creat pachetul `miniCRAN`[4], care face posibil acest lucru prin determinarea rețelei de pachete dependente și instalarea lor.

3.2 CRAN Task Views

Un mod particular de a grupa pachetele după domeniile de analiză este reprezentat de așa numitele *CRAN Task Views*. Avantajul acestora este că, pentru un anumit domeniu (de exemplu, statistica oficială), se poate instala o suită de pachete, ce oferă funcționalități complete pentru analizele încorporate în domeniul respectiv. Câteva *CRAN Task Views* din cele 33 disponibile sunt:

- Econometrics - Computational Econometrics

[2]http://cran.r-project.org/web/packages/
[3]http://r4stats.com/2014/04/07/r-continues-its-rapid-growth/
[4]http://cran.r-project.org/web/packages/miniCRAN/index.html

- Graphics - Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
- OfficialStatistics - Official Statistics & Survey Methodology[5]
- SocialSciences - Statistics for the Social Sciences
- Time Series - Time Series Analysis

Pentru a instala aceste *Task Views*, trebuie instalat pachetul `ctv`:

```
> install.packages("ctv")
> library("ctv")
```

Ulterior, *CRAN Task Views* pot fi instalate, respectiv actualizate, prin funcțiile `install.views()` și `update.views()`. De exemplu:

```
> install.views("SocialSciences")
> update.views("SocialSciences")
```

[5]Este disponibilă și varianta în limba română: http://www.r-project.ro/ro/OfficialStatistics_RO.html

Capitolul 4

Interfețe grafice

4.1 Cele mai uzuale interfețe grafice

4.1.1 Consola R

Fereastra de Consolă R, prezentată și în capitolul anterior, este utilă atunci când utilizatorul are de executat câteva linii de cod. Pentru o analiză complexă, sau pentru a salva ce s-a lucrat, se va accesa meniul *File* și se va selecta *New Script*. Va apărea un editor R, unde se pot introduce coduri și care poate fi salvat cu extensia .R. Pentru a executa o linie de cod din editor, se va utiliza <Ctrl>-R. Se observă că linia de cod a fost executată în fereastra de consolă.

Consola R nu este totuși cea mai user-friendly interfață a mediului R. În secțiunea următoare se va prezenta o altă interfață grafică - *RStudio*.

4.1.2 RStudio

RStudio este un mediu integrat de dezvoltare (în engleză *integrated development environment*) pentru limbajul R. Există două categorii de versiuni:

- RStudio Desktop (cu varianta Open Source și varianta cu licență comercială): se utilizează pe calculatorul local;
- RStudio Server (cu variantele Open Source Edition și Professional Edition): permite utilizarea R într-o interfață în cadrul browser-ului..

RStudio[1] este disponibil pentru toate sistemele de operare: Windows, Mac OS X și Linux.

RStudio este considerat cel mai intuitiv și mai ușor de utilizat dintre toate interfețele grafice. Pune la dispoziția utilizatorului în aceeași fereastră atât consola, spațiul de lucru, scriptul cât și spațiul de vizualizare al pachetelor și al documentației. RStudio este structurat în patru zone de lucru, precum în Figura 4.1:

1. Editor de comenzi (*.R);
2. Consola;
3. Environment, History, Files;
4. Plots, Packages, Help, Viewer.

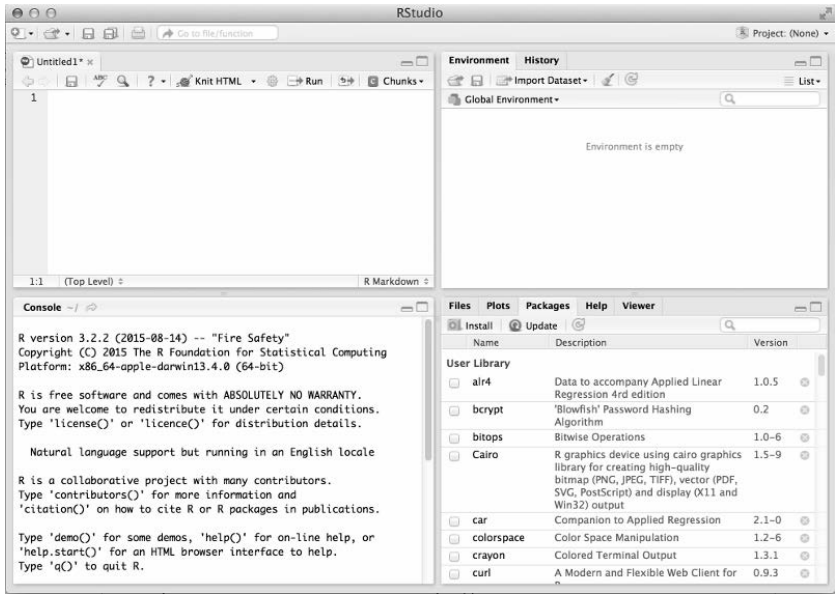


Figura 4.1: RStudio

Principalele caracteristici ale interfeței grafice RStudio sunt următoarele:

- Evidențierea sintaxei, completări ale codului și aliniere inteligentă;
- Executarea codului R direct din editor;
- Ușurința organizării folderelor și fișierelor de lucru prin proiecte;

[1] http://www.rstudio.com

- Căutare inteligentă în codul sursă și identificarea variabilelor și a funcțiilor;
- Identificarea definițiilor pentru funcții;
- Suport și documentație integrate;
- Debugging interactiv pentru identificarea și rezolvarea rapidă a erorilor;
- Instrumente avansate pentru dezvoltarea pachetelor.

O serie de comenzi și funcționalități ale mediului R vor fi executate cu ușurință în RStudio:

- Pachetele se instalează accesând butonul *Install* din tab-ul *Packages*.
- Pachetele se încarcă în memorie prin bifarea căsuței din fața denumirii pachetului, în tab-ul *Packages*.
- Documentația unui pachet se accesează prin click pe numele pachetului, în tab-ul *Packages*.
- Obiectele pot fi explorate cu ușurință în tab-ul *Environment*.
- Graficele și alte reprezentări grafice pot fi vizualizate și salvate, în tab-ul *Plots*.

O observație importantă este aceea că toate comenzile procesate din meniu sau din diferite butoane (cu mouse-ul), vor conduce automat la executarea funcțiilor specifice în Consolă.

Versiunea actuală de RStudio (v0.99) include un Data Viewer avansat, care permite sortare, filtrare, căutare în cadrul setului de date vizualizat. De asemenea, se pot aplica etichete variabilelor din seturile de date utilizate. RStudio nu oferă încă facilitatea salvării rezultatelor filtrărilor și sortărilor, însă se aşteaptă ca pe viitor sa fie dezvoltată şi această funcţionalitate.[2]

RStudio permite crearea, editarea și salvarea unei game largi de tipuri de fișiere: R Script, R Markdown, R Sweave, R HTML, R Presentation, R Documentation, Text, TEX, Markdown, XML, YAML, DCF, Shell, HTML, CSS, JavaScript, C/C++, Python și SQL.

[2]https://support.rstudio.com/hc/en-us/articles/205175388-Using-the-Data-Viewer

4.2 Alte interfețe grafice

4.2.1 RCommander

RCommander este o interfață grafică utilizată în special pentru analize statistice (Figura 4.2), având încorporate diferite funcționalități, precum: teste neparametrice, analize dimensionale, modelare prin regresii, distribuții continue, distribuții discrete, teste de ipoteze. De asemenea, oferă facilități la importul de date din diferite formate (text, Excel, Access, dBase, SPSS, SAS, Stata, Minitab) și la realizarea de grafice (histogramă, boxplot, nor de puncte, grafic de linii, plot condițional XY, strip chart, grafic de medii, diagramă bară, diagramă circulară, grafice diagnostic).

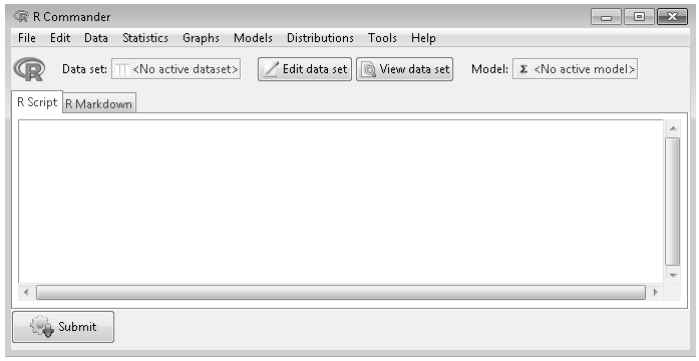


Figura 4.2: RCommander

Pachetul care permite rularea interfeței RCommander se numește `Rcmdr`. RCommander dispune și de o serie de plugin-uri, specializate pe diferite domenii de analiză.[3]

4.2.2 Rattle

Rattle (R Analytic Tool To Learn Easily)[4] este o interfață grafică utilizată pentru data mining, ce oferă facilități pentru teste statistice, transformări de seturi de date, clustere, modele decision tree, matrici eroare, analize de corelații, reprezentări grafice ale modelelor (Figura 4.3). Pentru a utiliza interfața grafică Rattle, se va instala pachetul `rattle`. După încărcarea în memorie, se va executa funcția `rattle()` :

[3] http://www.rcommander.com
[4] http://rattle.togaware.com/

```
> install.packages("rattle")
> library("rattle")
> rattle()
```

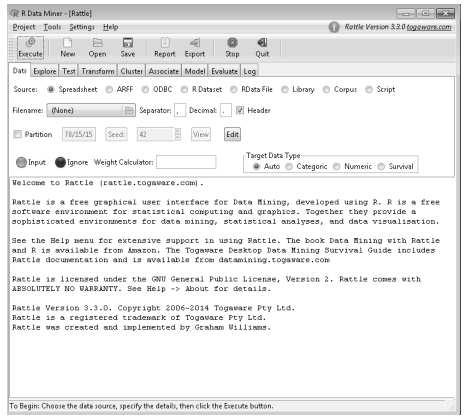


Figura 4.3: Rattle

Datele pot fi încărcate din fișiere de tip CSV, TXT, Excel, ARFF, ODBC, R Dataset, RData File, seturi de date din librăriile pachetelor, iar modelele construite și evaluate vor fi exportate ca PMML (predictive modelling markup language) sau ca scoruri.

4.2.3 Revolution R

Revolution R este o variantă comercială a mediului R, disponibilă gratuit pentru utilizarea în scopuri academice, utilă mai ales pentru manipularea big data. Revolution R, în varianta free, se prezintă precum în Figura 4.4.

Revolution R este dezvoltat și distribuit de compania Revolution Analytics[5], care promovează intens limbajul R în mediul business. Printre serviciile puse la dispoziție, această companie oferă și o certificare de utilizare a Revolution R - *AcademyR Certification* [6].

[5]http://revolutionanalytics.com/
[6]http://revolutionanalytics.com/academyr-certification

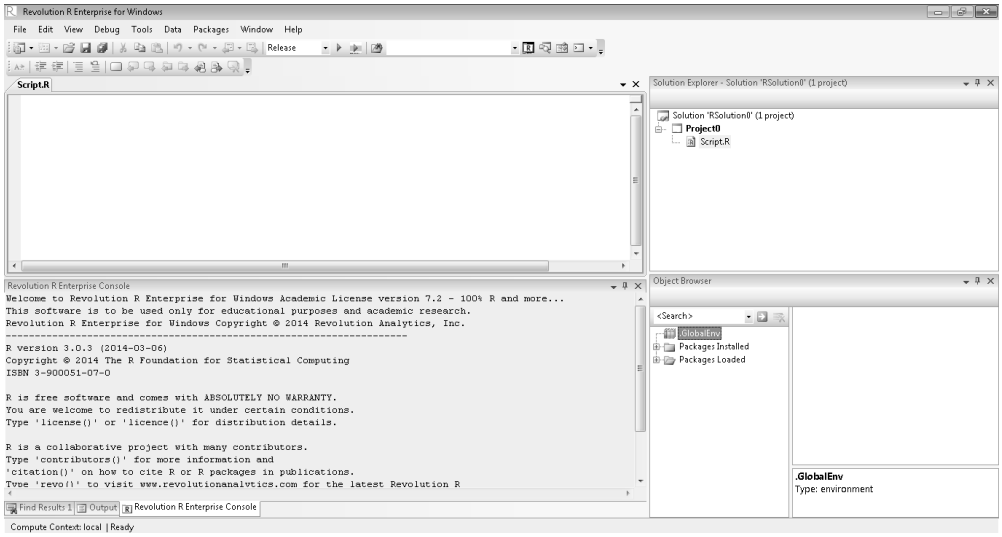


Figura 4.4: Revolution R

Capitolul 5

Resurse și comunitate online

5.1 Principalele resurse online

Utilizatorii mediului R interacționează inevitabil cu marea comunitate online. Entuziasmul utilizatorilor a dat naștere unei comunități dinamice, în continuă expansiune și activitate.
În cele ce urmează se vor prezenta principalele resurse online care ajută în utilizarea mediului R.

- Paginile de suport on-line pentru fiecare pachet (vignette)
  Acestea reprezintă o descriere a documentației care stă la baza fiecărui pachet de lucru, însoțită de exemple. Fiecare autor de pachet scrie realizează aceste pagini de suporr odată cu dezvoltarea pachetului respectiv.

- Documentația disponibilă pe website-ul CRAN: `http://cran.r-project.org`
  Aceste documentații sunt disponibile în două categorii: engleză și alte limbi.

- R-FAQ: `http://cran.r-project.org/doc/FAQ/R-FAQ.html`

- Motor de căutare specializat: `http://www.rseek.org`
  RSeek este un motor de căutare conceput pe platforma Google. În cadrul acestuia, căutarea se poate face pe mai multe secțiuni, ce includ: Funcții, Pachete, Cărți, Bloguri, Task Views etc.

- Website specializat cu știri și tutoriale (573 de bloggeri R): `http://www.r-bloggers.com`
  R Bloggers centralizează toate blogurile specializate pe mediul R.

- Grupuri Google Groups
- Grupuri LinkedIn
- Quick-R: `http://www.statmethods.net`
- R pentru utilizatori SPSS și SAS: `http://r4stats.com`
- Codul de bune practici elaborat de către Google: `http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`
- R Journal: `http://journal.r-project.org`
- MRAN (Managed R Archive Network): `http://mran.revolutionanalytics.com`
  MRAN este o arhivă de pachete și *Task Views*, realizată de Revolution Analytics, în care sunt redate graficele de dependențe dintre pachete.

5.2 Adresarea întrebărilor

Există o serie de website-uri și forumuri specializate, unde se pot adresa întrebări, dintre care:

- R-help mailing list: `http://www.r-project.org/mail.html`
- Stackoverflow: `http://stackoverflow.com/questions/tagged/r`
  Acest site este specializat pe probleme de programare, structură date, grafice.
- StatExchange: `http://stats.stackexchange.com`
  Secțiunea de analiză statistică a StatExchange este orientată mai degrabă pe probleme de natură statistică, decât pe programare.
- Talk Stats: `http://www.talkstats.com`
- CrossValidated: `http://stats.stackexchange.com`

O regulă nescrisă este aceea ca, înainte de a adresa o întrebare, să se consulte documentația disponibilă în Help sau să se utilizeze website-urile enumerate mai sus. Există șanse foarte mari ca întrebarea să fi fost deja adresată.
Atunci când se adresează întrebări comunității, este indicat să se țină cont de câteva reguli de bune practici. Astfel, în formularea întrebării trebuie incluse următoarele aspecte:

- Cum s-a ajuns la problemă?
- Care este output-ul așteptat?
- Ce nu este în regulă în output?/Care este output-ul obținut?
- Ce versiune de R/de pachet se utilizează?
- Ce sistem de operare se utilizează?
- Un exemplu minimal, care să conțină și datele necesare constatării problemei.

5.3 Alte resurse online

În această secțiune se vor prezenta alte website-uri care constituie resurse și exemplificări de aplicații ale mediului R:

- Gapminder: `http://www.gapminder.org`
  Gapminder este un proiect ce oferă instrumente de vizualizare a datelor panel, utilizând mediul R.
- Quandl: `https://www.quandl.com`
  Quandl este o bancă de date, ce cuprinde baze de date de la organizații naționale și internaționale, precum: Banca Mondială, Eurostat, etc. Accesarea unui set de date permite descărcarea lui în următoarele formate: R, Excel, CSV, JSON, XML, Python, Matlab. Pentru R există pachetul `Quandl`, ce are diferite opțiuni de descărcare și manipulare a datelor de pe acest website.
- Rapporter: `http://rapporter.net`
  Rapporter este un excelent instrument online, sub formă de platformă și cloud computing, de raportare și analiză date.

Capitolul 6

Tipuri de date și operații de bază

6.1 Numeric

Tipul de date numeric se mai întâlnește și sub denumirea de *double* și este de forma numerelor zecimale: *3.14, 5.24*. Acesta poate fi utilizat pentru a reprezenta variabile continue.

```
> x <- 3.14
> y <- 5.24
> z <- x + y
```

Funcția `is.numeric()` se utilizează pentru a verifica dacă variabila `z` este de tip numeric. Ca o alternativă, se poate folosi și funcția `typeof()`.

```
> is.numeric(z)
[1] TRUE
> typeof(z)
[1] "double"
```

Datele de tip *numeric* reprezintă aproximări ale numerelor reale. Din punct de vedere matematic, există o infinitate de astfel de numere, însă computerul poate reprezenta un număr finit. Se poate întâmpla ca numere precum 0.12 să nu fie reprezentate cu precizie, dar respectând aritmetica în virgulă mobilă[1]. O consecință este că trebuie tratată cu precauție compararea a două numere de tip *double*. Un exemplu în acest sens cu un rezultat surprinzător este oferit mai jos:

[1] http://web.cse.msu.edu/~cse320/Documents/FloatingPoint.pdf

```
> 0.2 + 0.2 + 0.2 == 0.6
[1] FALSE
```

Există două funcții în R pentru testarea egalității în acest caz: `identical()` și `all.equal()`:

```
> all.equal(0.6, 0.2 + 0.2 + 0.2)
[1] TRUE
> identical(0.6, 0.2 + 0.2 + 0.2)
[1] FALSE
```

Funcția `all.equal()` compară valorile în termeni de "egalitate apropiată" (în engleză *near equality*), în timp ce `identical()` le compară ținând cont de egalitatea exactă.

R reprezintă în mod corect valori numerice infinite, cum ar fi $\pm\infty$ cu `Inf` și `-Inf`, sau valori nenumerice cu `NaN` (*not a number*).

```
> (x <- 5/0)
[1] Inf
> class(x)
[1] "numeric"
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

6.2 Integer

Datele de tip întreg (în engleză *integer*) sunt numerele naturale, de forma: *3, 50, 210*. Ele pot fi utilizate pentru a reprezenta variabile cantitative, ca de exemplu numărul de copii dintr-o gospodărie. Funcția `class()` va returna tipul de date din obiect.

```
> nr_copii <- as.integer(5)
> is.integer(nr_copii)
[1] TRUE
> class(nr_copii)
[1] "integer"
```

În R, se pot realiza operații între date numerice și date integer.

```
> x <- as.integer(9)
> y <- 2.0
> z <- x/y
> z
[1] 4.5
```

Se observă că rezultatul este de tip numeric.

```
> class(z)
[1] "numeric"
```

6.3 Complex

Datele de tip *complex* sunt utilizate pentru a reprezenta numere complexe. Ele nu sunt foarte întâlnite în analizele statistice.
Funcțiile `as.complex()` și `complex()` creează astfel de date.

```
> test1 <- as.complex(-24+6i)
> sqrt(test1)
[1] 0.607714+4.936529i
> (test2 <- complex(3, real = 5, imaginary = 2))
[1] 5+2i 5+2i 5+2i
> typeof(test2)
[1] "complex"
```

Funcția `typeof()` va returna tipul obiectului..
O observație importantă este aceea că `-4` nu este perceput ca număr complex.

```
> sqrt(-4)
[1] NaN
Warning message:
In sqrt(-4) : NaNs produced
```

Pentru a fi de tip complex, trebuie scris sub forma `-4+0i`.

```
> sqrt(-4+0i)
[1] 0+2i
> sqrt(as.complex(-4))
[1] 0+2i
```

6.4 Logic

Un obiect de tip logic poate avea valorile `TRUE` sau `FALSE` și este utilizat pentru a indica dacă o condiție este adevarată sau falsă. Astfel de obiecte pot rezulta din expresii logice.

```
> x <- 2
> (y <- x > 4)
[1] FALSE
```

În exemplul anterior, obiectul `y` primește valoare logică pentru condiția `x > 4`.
Funcțiile de forma `is.tipdate()` (ca de exemplu, `is.numeric()`, `is.logical()` etc.) vor avea rezultat de tip logic:

```
> is.numeric(10.35)
[1] TRUE
> is.integer(10.35)
[1] FALSE
> is.complex(1+5i)
[1] TRUE
> is.logical(T)
[1] TRUE
```

Expresiile logice sunt realizate prin operatori logici.

torii logici de bază sunt: `&` (*și* logic), `|` (*sau* logic), și `!` (operatorul de negare):

```
> m <- TRUE
> n <- FALSE
> m & n          # m ŞI n
[1] FALSE
> m | n          # m SAU n
[1] TRUE
> !m             # negarea lui m
[1] FALSE
```

Aceștia sunt prezentați amănunțit în secțiunea 6.7.

Observație: Deși limbajul R acceptă scrierea `T` și `F` ca alternativă la `TRUE` și `FALSE`, se recomandă să se utilizeze cea de-a doua variantă. Observația este valabilă și pentru scrierea celor două valori în argumentele funcțiilor, ce se va studia în Capitolul 11.

6.5 Caracter

Datele de tip caracter sunt reprezentate de caractere, scrise între ghilimele duble (”). Spre exemplu, ”îmi”, ”place”, ”R” sunt date de tip caracter. Ele sunt definite astfel:

```
> (cuvinte <- c("imi", "place", "R"))
[1] "imi"        "place"      "R"
```

Un obiect particular cu date de tip caracter, predefinit în limbajul R, este `letters` sau `LETTERS`:

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

6.6 Conversia datelor

Diferențele dintre anumite tipuri de date sunt minore; prin urmare este posibilă conversia unui obiect dintr-un tip în altul prin schimbarea unor atribute ale sale. O asemenea conversie va fi efectuată cu o funcție de tipul `as.tipdate()`.

Rezultatul unei conversii depinde, evident, de atributele obiectului convertit. În general, conversia urmează reguli intuitive. Tabelul 6.1 oferă o privire de ansamblu asupra conversiei tipurilor de date.

Se observă că datele de tip logic, întreg și caracter pot fi convertite în numeric. Datele de tip întreg și anumite caractere (`"FALSE"`, `"TRUE"`, `"F"`, `"T"` pot fi convertite în tip logic. În fine, datele de tip întreg și logic pot fi convertite în caracter.

6.7 Operații de bază

În R, există trei tipuri principale de operatori, sintetizate în Tabelul 6.2. Pe lângă operatorii prezentați anterior, mai există și alții: `$`, `@`, `[`, `[[`, `:`, `?`, `<-`, `=`. O documentație completă a acestora se poate accesa cu `?Syntax`.

Tabelul 6.1: Conversia datelor

Conversie în	**Funcția**	**Reguli**		
numeric	`as.numeric`	F	→	0
		T	→	1
		`"1"`, `"2"`, …	→	1, 2, …
		`"A"`, …	→	`NA`
logic	`as.logical`	0	→	F
		alte numere	→	T
		`"FALSE"`, `"F"`	→	F
		`"TRUE"`, `"T"`	→	T
		alte caractere	→	`NA`
caracter	`as.character`	1, 2, …	→	`"1"`, `"2"`, …
		F	→	`"FALSE"`
		T	→	`"TRUE"`

Tabelul 6.2: Operatori aritmetici, de comparație, logici

Operatori					
Aritmetici		De comparație		Logici	
`+`	adunare	`<`	mai mic	`! x`	NU logic
`-`	scădere	`>`	mai mare	`x & y`	ȘI logic
`*`	înmulțire	`<=`	mai mic sau egal	`x && y`	id.
`/`	împărțire	`>=`	mai mare sau egal	`x | y`	SAU logic
`^` sau `**`	ridicare la putere	`==`	egal	`x || y`	id.
`%%`	modul	`!=`	diferit de	`xor(x, y)`	SAU exclusiv
`%/%`	împărțire cu rest				

Operatorii aritmetici și cei de comparație acționează asupra a două elemente, spre exemplu `x + y`, `a < b`. Operatorii aritmetici acționează nu doar asupra obiectelor de tip numeric sau complex, dar și asupra variabilelor logice; în cazul din urmă, valorile logice sunt transformate automat în numerice. Operatorii de comparație pot fi aplicați oricărui tip: ei returnează una sau câteva valori logice.

Operatorii logici sunt aplicați unuia sau mai multor obiecte de tip logic și returnează una sau mai multe valori logice. Operatorii "ȘI" și "SAU" există sub două forme: cel simplu actionează asupra fiecărui element din obiect și returnează tot atâtea valori logice câte comparații s-au făcut; cel dublu operează asupra primului element al obiectului.

Operatorii de comparație acționează asupra fiecărui element al ambelor obiecte de comparat (reciclând valorile celei mai scurte dacă este necesar) și astfel returnează un obiect de aceeași lungime.

```
> x <- 1:5
> y <- 1:5
> x == y
[1] TRUE TRUE TRUE TRUE TRUE
```

În continuare este oferit un alt exemplu ce ilustrează procesarea operatorilor de comparaţie în limbajul R. Se dorește afişarea poziţiilor dintr-un vector (a se vedea Secțiunea 9.1) care îndeplinesc anumite condiţii logice.

```
> x <- c(1:10)
> x[(x < 5) | (x > 8)]
[1]  1  2  3  4  9 10
```

Pentru a înţelege cum s-au procesat, de fapt, aceste comenzi, se va explica succesiunea de instrucţiuni pe care limbajul le execută. Într-o primă fază, se definit un obiect de tip vector, cu valori de la 1 la 10. Se doreşte afişarea poziţiilor din vector mai mici decât 5 sau mai mari decât 8.

```
> (x <- c(1:10))
 [1]  1  2  3  4  5  6  7  8  9 10
```

După afişarea verctorului, se verifică fiecare condiţie logică secundară în parte (`x < 5`, respectiv `x > 8`), iar apoi se verifică cea principală (`x < 5 | x > 8`):

```
> x < 5
 [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
> x > 8
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
> x < 5 | x > 8
 [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

În final, se vor căuta în vectorul `x` și se vor afișa poziţiile care îndeplinesc condiţia principală:

```
> x[c(TRUE,TRUE,TRUE,TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,TRUE)]
[1]  1  2  3  4  9 10
```

Capitolul 7

Citirea și scrierea datelor

Primul pas de urmat atunci când se începe lucrul în mediul R este citirea datelor cu care se lucrează sau importarea acestor date din diferite formate. Pentru studiul inițial al mediului de programare R, s-a considerat necesară prezentarea corespunzătoare următoarelor formate de fișiere: text, csv, SPSS, SAS și DBF.

7.1 Directorul de lucru

O etapă premergătoare lucrului în R este alegerea unui director de lucru, în care se vor stoca toate fișierele de lucru ale sesiunii respective. Calea directorului de lucru activ se obține prin funcția `getwd()`. Pentru modificarea acestuia se poate utiliza funcția `setwd()`.

Atunci când se definește calea directorului de lucru, trebuie ținut cont de faptul că R acceptă numai dublu backslash (`\\`) sau un slash simplu (`/`). De exemplu:

```
> setwd("C:/Documents/home")
```

sau:

```
> setwd("C:\\Documents\\home")
```

În RStudio, există în meniul *Session ->Set Working Directory* trei opțiuni de modificare a directorului de lucru:

1. To Source File Location – va rula automat comanda `setwd()` și va seta directorul de lucru în directorul de unde provin fișierele deja

deschise;
Dacă nu este niciun fișier deschis, această opțiune va fi inactivă.

2. To Files Pane Location – va seta directorul de lucru în secțiunea unde este instalat RStudio;
3. Choose Directory – se va deschide o fereastră cu meniu de alegere a directorului de lucru.

7.2 Citirea datelor

7.2.1 Citirea datelor din fișiere text și csv

Citirea datelor dintr-un fișier text (.txt) se poate face atât de pe calculatorul local, cât și de pe un URL. În primul caz, pentru citirea unui fișier text de pe hard-disk-ul local, se utilizează funcția `read.table()`:

```
> date_txt <- read.table("data.txt", header=TRUE)
```

`date_txt` reprezintă numele obiectului creat la importul datelor din fișierul `data.txt`. Argumentul `header=TRUE` indică faptul că primul rând din fișier conține numele variabilelor.

Importul datelor se poate face și în mod interactiv prin alegerea fișierului din My Computer, prin introducerea comenzii:

```
> mydata <- read.table(file.choose(), header=TRUE)
```

Importul datelor de pe un URL se realizează astfel:

```
date_url <- read.table("http://lib.stat.cmu.edu/jcgs/tu",
skip = 4, header = TRUE)
```

În acest exemplu, argumentul `skip=4` arată că primele 4 rânduri nu au fost citite.

Un alt caz particular este citirea datelor dintr-un fișier de tip csv, pentru care se utilizează funcția `read.csv()`. Această funcție este o variantă a funcției prezentate anterior (`read.table()`), având caracteristici comune. Se recomandă importul fișierelor de tip csv ca alternativă la al celor Excel. Deși R oferă și pachete pentru importul fișierelor Excel, funcționalitatea lor depinde de versiunea de Java instalată, iar din acest motiv instalarea lor poate fi uneori complicată.

7.2.2 Citirea datelor din fișiere SPSS

Citirea fișierelor SPSS se realizează cu pachetul `foreign`, utilizând funcția `read.spss()`:

```
> date_SPSS <- read.spss("data.sav", to.data.frame = TRUE)
```

În mod implicit, la import, nu se creează un dataframe; așadar, dacă utilizatorul are nevoie să lucreze cu obiect de tip dataframe se va seta parametrul `to.data.frame` cu valoarea `TRUE`. Această funcție citește fișierele create de comenzile `save` și `export` din SPSS.

O altă variantă pentru citirea fișierelor din format SPSS, care va genera automat un dataframe la import, este funcția `read_spss` din pachetul `haven`.

```
> date_SPSS <- read_spss("data.sav")
```

În fine, o altă alternativă este funcția `spss.get()` din pachetul `Hmisc`.

7.2.3 Citirea datelor din fișiere SAS

Importul datelor din fișiere SAS se face cu ajutorul aceluiași pachet menționat anterior, însă cu funcția specifică `read.xport()`. Această funcție importă fișiere de tip *XPORT SAS*.

```
> date_SAS <- read.xport("data.xpt")
```

Pentru fișiere de tip SAS7BDAT există un pachet specializat, numit `sas7bdat`:

```
> date_SAS <- read.sas7bdat("data.sas7bdat", debug = FALSE)
```

7.2.4 Citirea datelor din fișiere DBF

Datele din fișierele DBF se importă cu funcția `read.dbf()` din pachetul `foreign`.

```
> date_DBF <- read.dbf("data.dbf", as.is = FALSE)
```

Această funcție convertește câmpurile de tip caracter în factor și importă fișierele ca dataframe-uri.

7.3 Scrierea datelor

7.3.1 Scrierea datelor în fișiere text

Un dataframe sau o matrice pot fi exportate în fișiere text cu ajutorul funcției `write.table()`:

```
> write.table(mydata, "output.txt", sep = "\t", row.names = FALSE)
```

Pentru salvarea unui obiect de orice tip, se poate utiliza comanda `save(x, y, z, file= "xyz.RData")`. Datele (care sunt numite în această fază *workspace* în jargonul limbajului R) pot fi încărcate mai târziu în memorie cu funcția `load("xyz.RData")`.

7.3.2 Scrierea datelor într-un fișier SPSS

Pentru a exporta date în fișiere SPSS, se utilizează funcția `write.foreign()` din pachetul `foreign`:

```
> write.foreign(date_SPSS, "output_SPSS.txt", "output_code.sps",
package = "SPSS")
```

Funcția va scrie un fișier text și un program SPSS care va citi acest fișier text.

7.3.3 Scrierea datelor într-un fișier SAS

Pachetul `foreign` este folosit și în cazul scrierii datelor din R în fișiere SAS. Astfel, se va utiliza următoarea comandă:

```
> write.foreign(date_SAS, "output_SAS.txt", "output_code.sas",
package = "SAS")
```

Funcția va scrie un fișier text și un program SAS care va citi acest fișier text.

7.3.4 Scrierea datelor într-un fișier DBF

Pentru exportul datelor (dataframe sau matrice) în format DBF, se utilizează funcția `write.dbf()` din pachetul `foreign`:

```
> write.dbf(date_DBF, file, factor2char = TRUE, max_nchar = 254)
```

În finalul acestui capitol, se consideră utilă o expunere a unui tablou de ansamblu asupra funcțiilor și pachetelor care pot fi utilizate pentru citirea și scrierea datelor.

Tabelul 7.1: Funcții pentru citirea și scrierea datelor

Pachet	Format	Citire	Scriere
`utils`	txt	`read.table()`	`write.table()`
	csv	`read.csv()`	`write.csv()`
`foreign`	SPSS	`read.spss()`	`write.foreign()`
	SAS	`read.xport()`	`write.foreign()`
	DBF	`read.dbf()`	`write.foreign()`
`haven`	SPSS	`read_sav()`	`write_sav()`
	SAS	`read_sas()`	-
`sas7bdat`	SAS	`read.sas7bdat()`	-

Capitolul 8

Grafice

Scopul acestui capitol este de a oferi câteva referințe pentru a contura o idee asupra particularităților mediului R în vizualizarea datelor.
Vizualizarea datelor (în engleză *data visualisation*) se referă la studierea reprezentării vizuale a datelor. R oferă o varietate remarcabilă de reprezentări grafice. Pentru un scurt demo, se pot utiliza comenzile `demo(graphics)` sau `demo(persp)`.

Pe lângă varietate, mediul R oferă multiple avantaje în ceea ce privește vizualizarea datelor:

- grafice integrate;
- flexibilitate și personalizare;
- reproductibilitate;
- calitate excelentă;
- posibilitate de salvare a graficelor în formate variate: PDF, PNG, JPG, TIFF, BMP, metafile, SVG, EPS etc.

În R, există două categorii principale de grafice, fiecare având la rândul ei două sub-categorii:

1. Grafice *low-level*
    - grafice *base*
    - grafice *grid*
2. Grafice *high-level*

- grafice *lattice*
- grafice *ggplot2*

Graficele *low-level* sunt cele în care fiecare detaliu (fiecare pixel, fiecare caracter) al graficului poate fi modificat sau customizat. Graficele *high-level* sunt grafice prefabricate, care au un anumit aspect predefinit. Graficele *grid* nu vor face obiectul studiului în această carte. Cele patru subcategorii (base, grid, lattice și ggplot) reprezintă medii grafice (în engleză *graphics environments*) ale mediului R. Înainte de a începe studierea acestora, se vor analiza câteva funcționalități specifice mediului R pentru reprezentările grafice.

8.1 Utilități pentru grafice: Parametri grafici

Realizarea graficelor poate fi îmbunătățită prin utilizarea parametrilor grafici. Aceștia pot fi utilizați fie ca opțiuni ale funcțiilor grafice (însă nu funcționează pentru toate), fie în cadrul funcției `par()`. Spre exemplu, comanda urmatoare:

```
> par(bg = "yellow")
```

va avea ca rezultat reprezentarea graficelor consecutive pe fundal galben.

Există zeci de parametri grafici, o parte dintre ei având funcționalități similare. Lista completă a acestor parametri poate fi vizualizată prin comanda `?par`. În continuare sunt prezentați cei mai uzuali parametri grafici:

- `adj`: controlează alinierea textului față de limita din stânga.
  Poate lua următoarele valori: `0` pentru aliniere stânga, `0.5` pentru centrare, `1` pentru aliniere dreapta, valorile > 1 deplasează textul înspre stânga, iar valorile negative înspre dreapta; pentru două valori date (de exemplu, `c(0, 0)`), a doua controlează alinierea verticală față de nivelul textului;
- `bg`: specifică culoarea fundalului (de exemplu, `bg="red"`, `bg="blue"`; lista culorilor disponibile este afișată cu `colors()`);
- `bty`: controlează tipul casetei din jurul graficului.
  Valorile permise sunt: `"o"`, `"l"`, `"7"`, `"c"`, `"u"` sau `"]"`; pentru `bty="n"` caseta nu este trasată;

- `cex`: o valoare ce controlează dimensiunea fontului și simbolurilor; parametrii următori au același control pentru axe (`cex.axis`), etichetele axelor (`cex.lab`), titlu (`cex.main`) și subtitlu (`cex.sub`);

- `col`: controlează culoarea simbolurilor; similar cu parametrul `cex`, și în acest caz există variantele: `col.axis`, `col.lab`, `col.main`, `col.sub`;

- `font`: o valoare de tip întreg care setează stilul textului (`1`: normal, `2`: italic, `3`: bold, `4`: bold italic); există și variantele: `font.axis`, `font.lab`, `font.main`, `font.sub`;

- `las`: o valoare de tip întreg care controlează orientarea etichetelor de pe axe (`0`: paralel cu axele, `1`: orizontal, `2`: perpendicular pe axe, `3`: vertical);

- `lty` controlează tipul liniilor, poate fi de tip întreg (`1`: neîntrerupt, `2`: striat (în engleză *dashed*), `3`: punctat (în engleză *dotted*), `4`: striat și punctat (în engleză *dotdash*), `5`: punctat lung (în engleză *longdash*), `6`: punctat dublu (în engleză *twodash*)), sau de tip șir de până la 8 caractere (între `"0"` și `"9"`) care specifică alternativ lungimea, în puncte sau pixeli, a elementelor reprezentate grafic și ale spațiilor goale, de exemplu `lty="44"` va avea același efect cu `lty=2`;

- `lwd`: o valoare numerică ce controlează grosimea liniilor;

- `mar`: un vector de 4 valori numerice care controlează coordonatele marginilor graficului, având forma `c(bottom, left, top, right)`; valorile implicite sunt `c(5, 4, 4, 2)`;

- `mfcol`: un vector de forma `c(nr,nc)` care împarte fereastra graficului într-o matrice cu `nr` linii și `nc` coloane, graficele fiind apoi desenate pe coloane;

- `mfrow`: identic cu `mfcol`, însă graficele sunt apoi desenate pe linii;

- `pch`: controlează tipul simbolului grafic, fie că este un întreg între 1 și 25, sau orice alt caracter fără `""`;

- `ps`: o valoare de tip întreg ce controlează dimensiunea textului și a simbolurilor;

- `pty`: un caracter ce specifică tipul zonei reprezentate grafic.
  Poate lua valorile: `"s"` pentru pătrat (în engleză *square*) sau `"m"` pentru maximal;

- `tck`: o valoare care specifică lungimea marcajelor de pe axe ca parte a celei mai mici lungimi sau înălțimi a graficului; pentru `tck=1` este desenată o grilă;
- `tcl`: identic cu `tck`, însă ca parte a înălțimii unei linii de text (în mod implicit `tcl=-0.5`);
- `xaxt`: pentru `xaxt="n"` axa x este setată însă nu desenată (parametru util împreună cu `axis(side=1, ...)`);
- `yaxt`: pentru `yaxt="n"` axa y este setată însă nu desenată (parametru util împreună cu `axis(side=2, ...)`).

8.2 Utilități pentru grafice: culori

În mediul R există câteva funcționalități specifice pentru culori. Mediul R lucrează cu palete de culori. Paleta implicită de culori se accesează cu următoarea comandă:

```
> palette()
[1] "black"   "red"     "green3"  "blue"
[5] "cyan"    "magenta" "yellow"  "gray"
```

Aceasta se poate modifica prin schimbarea parametrilor:

```
> palette(rainbow(5, start = 0.1, end = 0.5))
> palette()
[1] "#FF9900" "#EBFF00" "#70FF00" "#00FF0A" "#00FF85"
```

De asemenea, noua paletă creată poate fi setată ca implicită:

```
> palette("default")
```

Funcția `gray()` permite selectarea de nuanțe de gri, prin atribuirea de valori cuprinse între 0 și 1:

```
> gray(seq(0.1, 1, by = 0.3))
[1] "#1A1A1A" "#666666" "#B2B2B2" "#FFFFFF"
```

Funcția `colors()` listează toata gama de culori ce poate fi utilizată la realizarea graficelor. Pentru vizualizarea completă a gamei de culori

disponibile în mediul R, se recomandă consultarea unor surse suplimentare.[1]

`RColorBrewer` este un pachet specializat pe palete de culori, ce oferă câteva palete predefinite, care se pot previzualiza astfel:

```
> par(mar = c(0, 4, 0, 0))
> display.brewer.all()
```

Parametrul grafic `mar` cu valorile `0,4,0,0` va seta o margine în stânga graficului. Paletele de culori disponibile în acest pachet pot fi setate ca implicite, pentru realizarea graficelor din mediul grafic activ. Spre exemplu, se pot seta primele trei culori din paleta `Accent`, inclusă în pachetul `RColorBrewer`, astfel:

```
> palette_accent <- brewer.pal(3, "Accent")
> palette(palette_accent)
```

8.3 Grafice base

Pentru graficele *base* există câteva funcții specifice, ce produc diferite tipuri de reprezentări:

- `plot()` - diagrame xy;
- `barplot()` - diagrame bară;
- `boxplot()` - diagrame boxplot;
- `hist()` - histogramă;
- `pie()` - diagramă circulară;
- `dotchart()` - diagramă cu puncte;
- `image()`, `heatmap()`, `contour()`, `persp()` - grafice de tip imagine;
- `qqnorm()`, `qqline()`, `qqplot(0)` - grafice pentru comparația distribuțiilor;
- `pairs()`, `coplot(0)` - reprezentarea datelor multivariate.

[1] Earl F. Glynn, Chart of R Colors, http://research.stowers-institute.org/efg/R/Color/Chart/

În continuare se vor ilustra exemple ale celor mai importante și totodată uzuale funcții enumerate mai sus. Pentru a asigura reproductibilitatea graficelor, s-au utilizat obiecte create (matrici, dataframe-uri) sau seturi de date din pachete existente.

8.3.1 Diagrame xy

```
> set.seed(1234)
> y <- matrix(runif(30), ncol = 3, dimnames = list(letters[1:10],
 LETTERS[1:3]))
> plot(y[,1], y[,2])
```

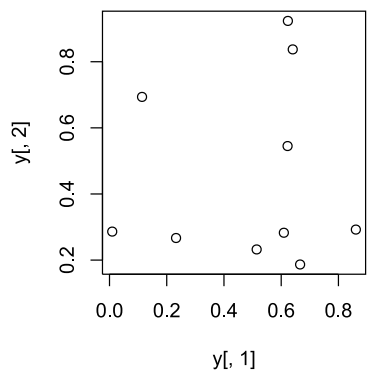
În exemplul anterior se generează o matrice, formată din 3 coloane și 30 de valori, în care s-au definit numele rândurilor și coloanelor cu primele 3 litere mici (rânduri) și primele 10 litere mari (coloane). Ulterior se reprezintă grafic prin funcția `plot()` primele două coloane ale matricei `y`, rezultând Figura 8.1a.

Există și posibilitatea reprezentării grafice a unei matrice *scatterplot*, precum în Figura 8.1b.

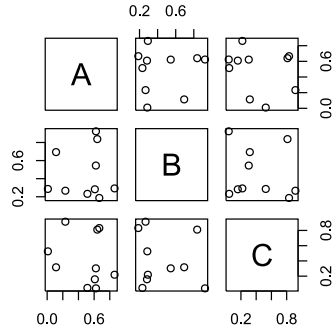
```
> pairs(y)
```

Figura 8.1

a. Diagramă xy



b. Matrice scatterplot



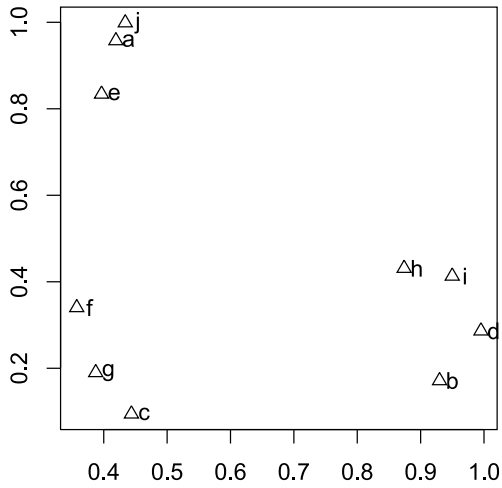
În matricea scatterplot, variabilele sunt scrise pe diagonală începând din stânga sus. Fiecare variabilă este reprezentată în funcţie de celelalte. De exemplu, pătratul al doilea din prima coloană este o diagramă xy individuală pentru variabilele `A` și `B`, cu `A` reprezentat pe axa Ox și `B` pe Oy. Aceeaşi diagramă xy este replicată în mijlocul primului rând.

În esenţă, diagramele din partea de sus a diagonalei sunt replicate, în oglindă, în partea de jos a diagonalei.

Se pot controla orice parametri grafici și se poate îmbunătăți graficul prin adăugarea oricăror informații adiționale, imaginația utilizatorului fiind singura limită. De exemplu, la graficul inițial se pot adăuga etichete valorilor reprezentate grafic cu ajutorul funcției `text()`, și se poate stabili simbolul punctelor cu argumentul `pch`:

```
> plot(y[,1], y[,2], pch = 2, xlab = "", ylab = "")
> text(0.02 + y[,1], y[,2], rownames(y))
```

Figura 8.2: Diagramă xy



În cadrul funcției `text()`, valoarea `+0.03` va deplasa etichetele valorilor în dreapta, pentru a nu se suprapune cu simbolurile grafice.

Alte argumente importante ale funcției `plot()` sunt:

- `mar` specifică dimensiunea marginilor din jurul suprafeței graficului: `c(bottom, left, top, right)`;
- `xlab`, `ylab` specifiă etichetele abscisei și ordonatei;
- `col` setează culoarea simbolurilor grafice;
- `pch` indică tipul simbolurilor grafice;

- `lwd` indică dimensiunea simbolurilor grafice;
- `cex` setează dimensiunea fonturilor;
- `legend` creează o legendă a graficului;
- `bg` setează backgroundul ariei de reprezentare grafică.

8.3.2 Diagrame bară

Se va considera matricea `y`, definită anterior. Se dorește reprezentarea grafică printr-o diagramă bară a primelor patru rânduri ale acesteia:

```
> barplot(y[1:4,], ylim = c(0, max(y[1:4,]) + 0.2),
beside  =TRUE,legend = letters[1:4])
```

În cadrul argumentului `ylim`, valoarea `+0.2` a însumat 0.2 la valoarea primelor patru rânduri din matrice, setând astfel limita axei *Oy*. Va rezulta diagrama cu bară verticală din Figura 8.3a.

Diagramele bară vor fi explicate mai detaliat în Capitolul 12.

8.3.3 Diagrame boxplot

Diagramele boxplot sunt utile pentru reprezentarea grafică a variabilelor numerice, ele indicând valorile extreme, minimul, maximul, quartilele și mediana.

În exemplul următor, s-a utilizat matricea `y`, pentru reprezentarea grafică sub forma unei diagrame boxplot:

```
> boxplot(y)
```

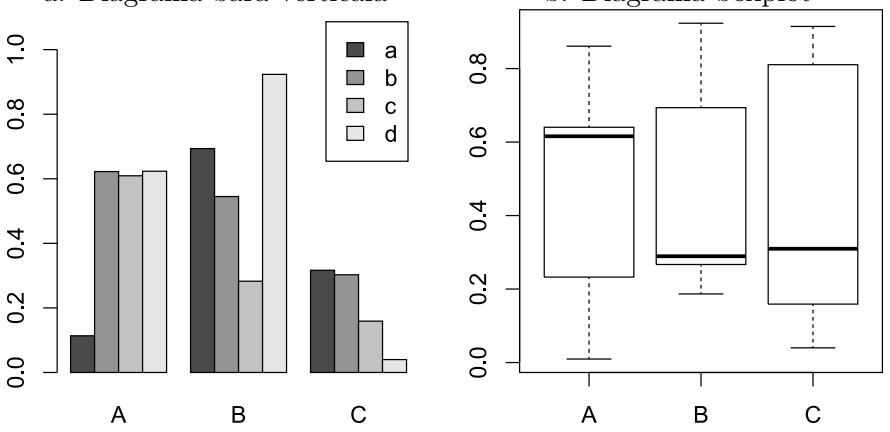
În Figura 8.3b, se observă că toate coloanele matricei au fost reprezentate pe același grafic. Mediana este indicată de linia din interiorul cadranelor, valoarea maximă este linia de sus și cea minimă linia inferioară din afara cadranului, iar cele două linii orizontale ce compun cadranul sunt quartila inferioară (25% din valori sunt mai mici decât această valoare), respectiv cea superioară (25% din valori sunt mai mari decât această valoare).

8.3.4 Histograme

Histogramele se utilizează pentru a reprezenta grafic frecvențele de apariție (caz în care argumentul `freq` ia valoarea `TRUE`) sau densitățile

Figura 8.3

a. Diagramă bară verticală b. Diagramă boxplot



de probabilitate (caz în care argumentul `freq` ia valoarea `FALSE`) pentru variabile cantitative. Intervalele de apariție pentru aceste frecvențe se setează cu argumentul `breaks`.
În continuare se presupune un vector de vârste pentru 30 de indivizi:

```
> varste <- c(25,21,35,43,51,67,50,45,34,37,28,15,16,20,54,61,
32,45,35,39,22,25,14,27,56,51,39,40,22,17)
```

Se va crea histograma acestor vârste:

```
hist(varste,freq=TRUE, breaks=6, main="")
```

Argumentul `main=""` a fost utilizat pentru ca graficul să nu aibă titlu (implicit, titlul ar fi fost "Histogram of y"). Din Figura 8.4a, se observă că sunt 5 persoane cu vârsta cuprinsă între 10 și 20 de ani, 7 persoane între 20 și 30 de ani, 8 între persoane 30 și 40 de ani ș.a.m.d.

8.3.5 Diagrame circulare

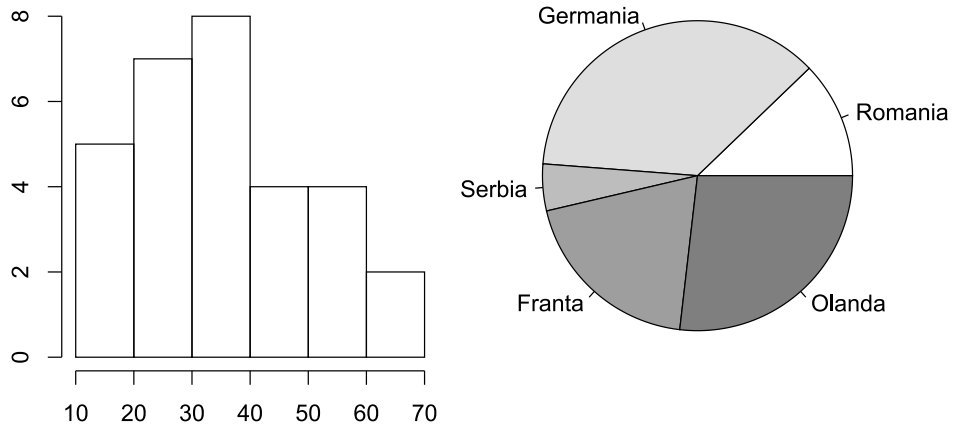
În exemplul următor se vor reprezenta sub formă de diagramă circulară anumite date (`valori`) pentru anumite state (`etichete`).

```
> valori <- c(10, 30, 4, 16, 22)
> etichete <- c("Romania", "Germania", "Serbia", "Franta",
"Olanda")
```

Funcția care va genera Figura 8.4b este următoarea:

Figura 8.4

a. Histogramă b. Diagramă circulară




```
> pie(valori, labels = etichete,
col = gray(seq(1.0,0.5, length = 5)))
```

Diagramele circulare sunt abordate amănunțit în Capitolul 12.

8.4 Grafice lattice

Lattice reprezintă un mediu grafic avansat al mediului R. Acesta este o implementare a sistemului Trellis din S-PLUS, care constă în grafice specializate pe reprezentarea unei variabile sau a relației între variabile, condiționată de una sau mai multe variabile (factori).[2] Deși sintaxa este similară cu cea din graficele *base*, *lattice* simplifică secvențele de instrucțiuni de reprezentare grafică de tip *high-level*.
Principala idee a graficelor *latice* este aceea a reprezentărilor condiționărilor multiple: un grafic bidimensional va fi împărțit în câteva grafice ținând cont de valorile unei a treia variabile.

Formula tipică a graficelor *lattice* este `graph_type(formula, data)`. `graph_type` reprezintă unul din tipurile de grafice disponibile (`barchart, bwplot, cloud, contourplot, densityplot, dotplot, histogram, levelplot, parallel, splom, stripplot, xyplot, wireframe`), iar formula specifică variabilele reprezentate și variabilele condiționale. Spre exemplu, `~x|A` se referă la reprezentarea grafică a variabilei numerice `x` pentru fiecare nivel al factorului `A`. Similar, `y~x |`

[2] http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/index.html

`A*B` va însemna reprezentarea relației între variabilele numerice `x` și `y` pentru fiecare combinație a nivelurilor factorilor A și B. `~x` va reprezenta grafic variabila `x`, de sine stătătoare.

Pentru a exemplifica acest tip de grafice, se va încărca în memorie pachetul `lattice`:

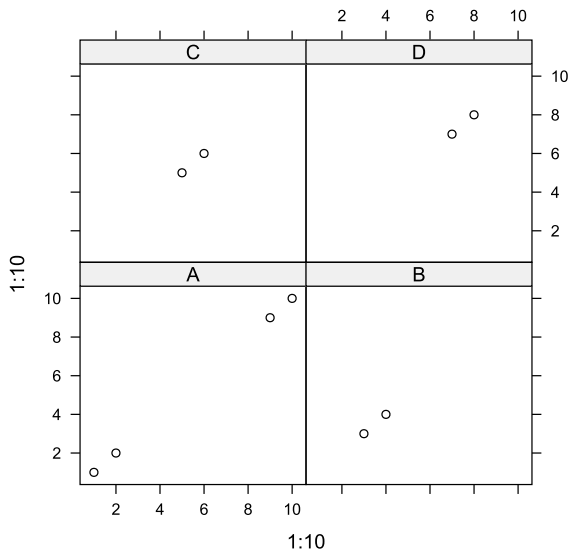
```
> library(lattice)
```

În continuare, se vor reprezenta grafic numerele de la 1 la 10, pentru fiecare nivel al primelor patru litere din alfabet.

```
> trellis.device(color = FALSE)
> xyplot(1:10~1:10|rep(LETTERS[1:4], each=2))
```

Prima linie de cod va dezactiva culorile din sistemul grafic, pentru ca graficul rezultat să conțină doar nuanțe alb-negru. Argumentul `each=2` arată că valorile se vor repeta de două ori pentru fiecare literă.

Figura 8.5: Diagramă xy - lattice



8.5 Grafice ggplot2

ggplot2 este un sistem avansat de grafice de tip *high-level*, ce are la bază teoria implementată de Leland Wilkinson în The Grammar of Graphics.[3]
Principalul avantaj al acestui sistem este simplificarea derulării etapelor pentru realizarea graficelor complexe.
În *ggplot2*, sintaxa este centrată în jurul funcției principale `ggplot()`, dar și a funcției `qplot()`.
Funcția `geom()` permite alegerea tipului de grafic: `geom_abline` (grafic cu abscisă și ordonată), `geom_area` (diagramă suprafață), `geom_bar` (diagramă bară verticală), `geom_boxplot` (diagramă boxplot) etc.[4] Pe lângă alegerea geometriei, sistemul *ggplot2* permite executarea unor transformări statistice asupra datelor, controlul asupra scalelor mapării, precum și setarea sistemelor de coordonate.

8.5.1 Grafice `qplot`

În această secțiune se va prezenta o parte din capabilitatea funcției `qplot()`, prin ilustrarea exemplelor de diagramă prin puncte și histogramă.

Funcția `qplot()` este similară funcției `plot()` studiată în Secțiunea 8.2. Aceasta poate avea ca argumente:

- x - coordonatele abscisei;
- y - coordonatele ordonatei;
- data - dataframe-ul din care se iau datele de intrare;
- xlim, ylim - limitele abscisei și ordonatei;
- alte argumente ce sunt valabile și pentru funcția `plot`.

Pentru a exemplifica aceste grafice, se vor crea două obiecte (eșantioane), `x` și `y`, fiecare de câte 20 de înregistrări:

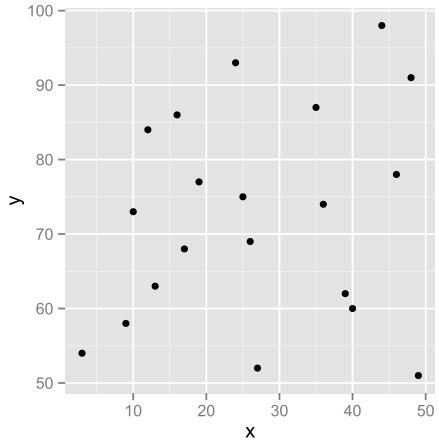
```
> x <- sample(1:50, 20)
> y <- sample(51:100, 20)
```

Se va realiza o diagramă cu puncte, prin atributul geometric de tip puncte, din care va rezulta Figura 8.6:

[3] Leland Wilkinson (1999) The Grammar of Graphics
[4] http://docs.ggplot2.org/current/

```
> qplot(x, y, geom = "point")
```

Figura 8.6: Diagramă cu puncte - qplot



În continuare, se va reprezenta un grafic cu linie de regresie. Pentru aceasta, se va utiliza setul de date `diamonds` din pachetul `ggplot2`. Acesta include informații din baza de date Diamond Search Engine[5], ce cuprinde date referitoare la carate și caracterirsticile acestora: tăieturi, culoare, claritate, adâncime, lățime. Din setul de date `diamonds` se va extrage un eșantion, format din primele 1000 de rânduri:

```
> diamonds_small <- diamonds[sample(nrow(diamonds), 1000), ]
```

Pentru a reprezenta grafic relația dintre preț și carate, dar și linia de regresie dintre acestea, se va executa următoarea comandă:

```
> qplot(carat, price, data = diamonds_small,
geom = c("point", "smooth"), method = lm)
```

Va rezulta Figura 8.7a.

În continuare, pentru realizarea unei histograme a variabilei `carat` se va executa comanda:

```
> qplot(carat, data = diamonds, geom = "histogram",
binwidth = 0.3)
```

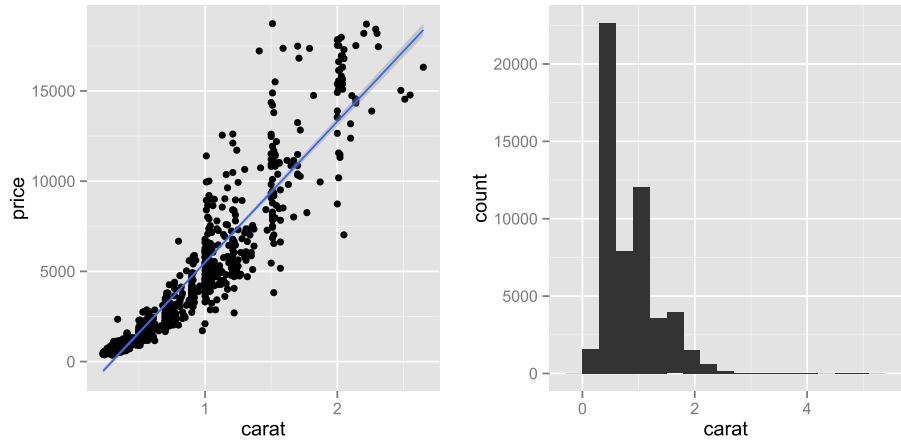
[5]Diamond Search Engine, http://www.diamondse.info/

Argumentul `binwidth = 0.3` specifică intervalele de apariție, acestea fiind vizibile și în Figura 8.7b.

Figura 8.7

a. Diagramă prin puncte cu linie de regresie - qplot

b. Histogramă - qplot



8.5.2 Grafice `ggplot`

Funcția acceptă două argumente: setul de date și mapările estetice oferite de funcția `aes()`, care permite setarea atributelor estetice ale variabilelor reprezentate.
Sintaxa generală a funcției `ggplot()` este:

```
> ggplot(data, aes(...)) + geom_*() +...+ stat_*() +...
```

`aes())`, `geom_*()` și `stat_*()` sunt argumente care setează atributele estetice, geometrice și statistice ale graficelor.

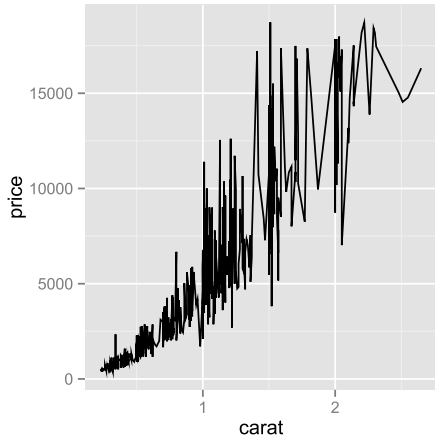
Temele graficelor pot fi accesate cu funcția `theme_get()` și pot fi schimbate cu `theme()`.

Graficele și caracteristicile lor pot fi stocate în obiecte. Se va lua drept exemplu reprezentarea grafică a două variabile din setul de date `diamonds_small`, creat în secțiunea 8.5.1.

```
> (p <- ggplot(diamonds_small,aes(carat, price)) + geom_line())
```

Funcția `summary()` oferă detalii ale caracteristicilor graficului `p`, reprodus în Figura 8.8.

Figura 8.8: Diagramă cu linii - ggplot




```
> summary(p)
data: carat, cut, color, clarity, depth, table, price, x, y, z [1000x10]
mapping:  x = carat, y = price
faceting: facet_null()
-----------------------------------
geom_line:
stat_identity:
position_identity: (width = NULL, height = NULL)
```

Salvarea graficului se poate face, de exemplu, într-un fișier *pdf*:

```
> ggsave(p, file = "myplot.pdf")
```

sau în format *JPG*:

```
> ggsave(p, file = "myplot.jpg")
```

O diagramă boxplot se poate realiza prin modificarea argumentului `geom_()`, astfel:

```
> (p <- ggplot(diamonds_small, aes(color, price/carat, fill=color)),
 geom_boxplot() + scale_fill_grey())
> p + guides(fill=FALSE)
```

În plus, argumentul `scale_fill_grey` a fost utilizat pentru ca graficul să fie colorat în nuanțe de gri, iar atributul `guides(fill=FALSE)` a eliminat legenda graficului, rezultând astfel Figura 8.9a.

Se observă că atributele statistice și geometrice se pot seta ulterior definirii graficului. În exemplul următor, în care se utilizează setul de date `diamonds`, se vor defini, rând pe rând, câteva atribute.

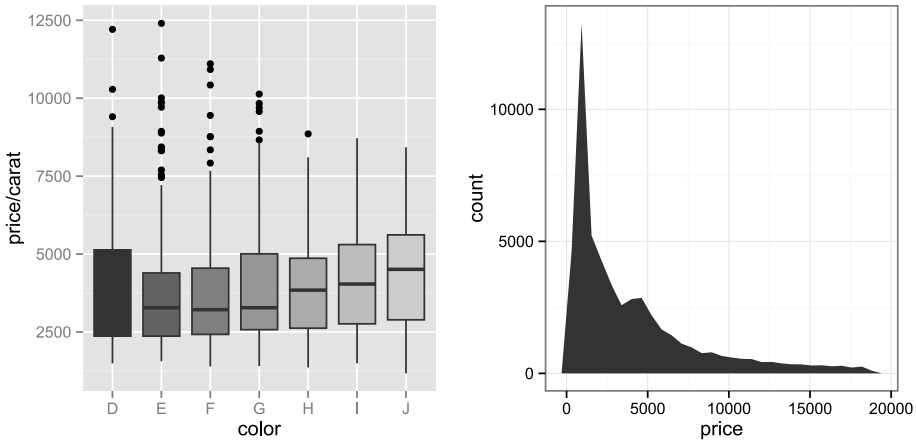
```
> (p <- ggplot(diamonds, aes(x = price)))
Error: No layers in plot
```

Această eroare apare deoarece nu a fost setat atributul geometric al graficului, atribut ce este obligatoriu. În continuare, se vor aplica atribute geometrice și statistice, urmând să rezulte Figura 8.9b, ce reprezintă o diagramă suprafață. Argumentul `theme_bw()` va seta tema alb-negru, eliminând tema gri (implicită sistemului ggplot).

```
> p + geom_histogram()
> p + geom_histogram(binwidth = 1000)
> p + stat_bin(geom = "area") + theme_bw()
```

Figura 8.9

a. Diagramă boxplot - ggplot b. Diagramă suprafață - ggplot



Capitolul 9

Structuri de date

9.1 Vectori

Vectorul reprezintă o colecție de valori care aparțin aceluiași tip de date. Dacă elementele unui vector sunt toate de tip numeric, vectorul este de tip numeric, iar dacă toate valorile sunt de tip caracter, va rezulta un vector de caractere.
Astfel, un vector poate fi utilizat pentru a reprezenta o singură variabilă dintr-un set de date. Pentru exemplificare se va crea un vector cu 5 elemente de tip numeric.

```
> vectornumeric <- c(2, 4, 3, 5, 8)
```

Se afișează tipul de date pe care il are vectorul:

```
> class(vectornumeric)
[1] "numeric"
```

Prin funcția `c(1:n)` mediul R crează vectorul de tip integer de n elemente.

```
> (vectorinteger <- c(1:5))
[1] 1 2 3 4 5
> class(vectorinteger)
[1] "integer"
```

Crearea unui vector de tip logic:

```
> x <- c(TRUE, FALSE)
> class(x)
[1] "logical"
```

Crearea unui vector de tip caracter:

```
> x <- c("a","b","c")
> class(x)
[1] "character"
```

Crearea unui vector de tip complex:

```
> x <- c(1+3i, 5+6i)
> class(x)
[1] "complex"
```

Prin funcția `vector()` se poate aloca spațiul necesar fără a atribui valori variabilei respective.

```
> (x <- vector("numeric", length=5))
[1] 0 0 0 0 0
```

Când se încearcă construirea unui vector cu elemente din clase diferite, acestea sunt transformate în elemente din aceeași clasă, iar vectorul este transformat în clasa cea mai cuprinzătoare. Astfel, din combinarea de valori logice sau numerice cu cele de tip caracter va rezulta un vector de tip caracter, iar alăturarea de valori logice cu cele de tip numeric va determina crearea unui vector numeric. Se poate crea un vector cu clase diferite, dar mediul R le va converti la o clasă unică. Crearea unui vector cu clase diferite care va fi transformat automat în caracter:

```
> (y <- c(1.5, "a"))
[1] "1.5" "a"
> class(y)
[1] "character"
```

Crearea unui vector care va fi transformat automat în numeric:

```
> (y <- c(TRUE, FALSE, 2))
[1] 1 2
> class(y)
[1] "numeric"
```

Se observă că valoare `TRUE` este transformată în numărul 1, iar valoarea `FALSE` este transformată în numărul 0.

Crearea unui vector cu elementele transformate în tip caracter:

```
> (y <- c("a", TRUE))
[1] "a"    "TRUE"
> class(y)
[1] "character"
```

9.2 Factori

O variabilă de tip factor este o colecție de valori care pot aparține unei mulțimi date de valori posibile. Un factor este similar cu un vector, cu excepția faptului că valorile dintr-un factor sunt limitate la o mulțime dată de valori posibile.

Un factor poate fi folosit pentru a reprezenta o variabilă categorială dintr-un set de date. Factorii sunt utilizați, de obicei, în modelarea statistică deoarece variabilele categoriale sunt tratate diferit de către modelele statistice în comparație cu variabilele continue. În cazul în care se cunoaște că variabila din seria de date este de tip categorial atunci este recomandat ca datele să se stocheze în variabile de tip factor pentru a fi tratate corect de către funcțiile de modelare.

Exemplul 1: pentru mulțimea numerelor {2,3,1,3,2,4,2,1,1,3,2,2,3,4} se pot identifica următoarele categorii: 1,2,3,4.

```
> vectornumere <- c(2,3,1,3,2,4,2,1,1,3,2,2,3,4)
> (fvectornumere <- as.factor(vectornumere))
 [1] 2 3 1 3 2 4 2 1 1 3 2 2 3 4
Levels: 1 2 3 4
```

Exemplul 2: sexul primilor zece respondenți la un sondaj este $\{F, M, M, F, M, F, F, M, F, M\}$, serie de date care conține două categorii: F-feminin, M-masculin.

```
> vectorsex <- c("F","M","M","F","M","F","F","M","F","M")
> class(vectorsex)
[1] "character"
> (fvectorsex <- as.factor(vectorsex))
 [1] F M M F M F F M F M
Levels: F M
```

Pentru a face conversia denumirilor implicite ale celor două categorii regăsite în exemplul precedent, respectiv F și M în Feminin și Masculin se utilizează o singură funcție numită `levels()`. Schimbarea denumirii categoriilor F și M, în Feminin, respectiv Masculin:

```
> levels(fvectorsex) <- c('Feminin','Masculin')
> fvectorsex
 [1] Feminin  Masculin Masculin Feminin  Masculin Feminin
 Feminin  Masculin Feminin  Masculin
Levels: Feminin Masculin
```

Variabilele de tip factor reprezintă o modalitate foarte eficientă de a stoca valori de tip caracter, deoarece fiecare valoare unică este stocată o singură dată, iar datele în sine sunt stocate ca un vector de numere întregi. Din acest motiv, funcția `read.table()` va converti automat variabilele de tip caracter în factori, cu excepția cazului în care este specificat altfel prin argumentul `as.is`. Se crează un vector numit `luni` cu o serie de valori reprezentând lunile de naştere a 17 angajaţi:

```
> luni <- c("Apr","Mar","Feb","Ian", "Noi","Oct","Dec","Apr",
            "Mai","Sep","Aug","Ian","Mai", "Iul","Iun","Dec",
             "Iun","Aug")
> fluni <- as.factor(luni)
```

Afişarea datelor într-un tabel de corelație între variabile, reprezentate prin denumirea categoriei, respectiv numerele de apariţie a categoriei:

```
> table(fluni)
fluni
Apr Aug Dec Feb Ian Iul Iun Mai Mar Noi Oct Sep
  2   2   2   1   2   1   2   2   1   1   1   1
```

Ordinea firească a lunilor nu este reflectată în rezultatul afișat de funcția `table()`, iar pentru ordonarea categoriilor se utilizează parametrul `levels` în cadrul funcției `factor()`, alături de parametrul `ordered=TRUE`.

```
> fluni <- factor(luni, levels = c("Ian","Feb", "Mar","Apr",
                                   "Mai", "Iun","Iul","Aug",
                                  "Sep","Oct", "Noi","Dec"),
                          ordered = TRUE)
```

Se verifică dacă primul element din vectorul `fluni` este mai mic decât al doilea element, inițial primul element fiind 'Apr', iar al doilea 'Mar':

```
> fluni[1] < fluni[2]
[1] FALSE
```

Afișarea datelor într-un tabel de frecvențe, reprezentate prin denumirea categoriei, respectiv frecvența de apariție a categoriei, dar de data aceasta ordonate după categoriile definite prin parametrul `levels`.

```
> table(fluni)
fluni
Ian Feb Mar Apr Mai Iun Iul Aug Sep Oct Noi Dec
  2   1   1   2   2   2   1   2   1   1   1   2
```

Analiza valorilor dintr-un vector de tip numeric se poate realiza cu funcția `cut()`, însoțită de argumentul `breaks` pentru a crea un factor cu intervale egale, considerate categorii, în care se vor regăsi valorile din vectorul analizat. Pentru exemplificare se poate crea un vector cu veniturile a 12 salariați, împărțite în quartile (în patru intervale egale ca număr de salariați):

```
> venituri <- c(1200, 1400, 1300, 1200, 1800, 1600, 2300,
                1400, 2500, 1500, 1800, 2300)

> fvenituri <- cut(venituri,
                   breaks = c(1200, 1375, 1550, 1925, 2500),
                   include.lowest = TRUE, dig.lab = 4)
> class(fvenituri)
[1] "factor"

> table(fvenituri)
fvenituri
[1200,1375] (1375,1550] (1550,1925] (1925,2500]
          3           3           3           3
```

În acest exemplu, valorile 1200 și 2500 reprezintă salariul minim și maxim, iar valorile 1375, 1550 și 1925 sunt cele trei quartile care împart veniturile în patru clase cu număr egal de salariați.

9.3 Matrici

O variabilă de tip matrice reprezintă colecție bidimensională de valori care au toate același tip. Valorile sunt aranjate în rânduri și coloane, iar în limbajul specific R, în observații și variabile. În fapt, matricea este un caz particular al unei variabile (în engleză *array*) n-dimensională cu doar două dimensiuni.

Se poate crea o matrice cu 2 rânduri și 5 coloane direct cu funcția `matrix()`.

```
> (m <- matrix(1:10, ncol = 5))
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Se observă că numerele de la 1 la 10 se completează în matrice pe coloane. De reținut că numele coloanelor este "`, j`", iar numele rândurilor este "`i,` ", spre exemplu prima coloană este "`, 1`", iar a doua linie este "`2`". Se afișează prima coloană:

```
> m[, 1]
[1] 1 2
```

Prima linie este:

```
> m[1, ]
[1] 1 3 5 7 9
```

Dacă se dorește afișarea valorii "8", care reprezintă elementul de la intersecția coloanei "4" cu linia "2", se utilizează comanda:

```
> m[2, 4]
[1] 8
```

Uneori există două serii de date care trebuie să formeze un tablou bidimensional, adică o matrice. Pentru exemplificare se consideră vectorul vârstelor angajaților: 25,43,27,36 și înălțimea acestora, exprimată în centrimetri: 175,180,168,183.

```
> varste <- c(25, 43, 27, 36)
> inaltime <- c(175, 180, 168, 183)
```

Putem concatena cele două serii pe coloane sau pe rânduri. Pe coloane vom utiliza funcția `cbind()`.

```
> (angajati <- cbind(varste, inaltime))
     varste inaltime
[1,]     25      175
[2,]     43      180
[3,]     27      168
[4,]     36      183
```

Pentru concatenarea pe rânduri se utilizează funcția `rbind()`.

```
> (angajati <- rbind(varste, inaltime))
          [,1] [,2] [,3] [,4]
varste      25   43   27   36
inaltime   175  180  168  183
```

Pentru crearea unei variabile cu trei dimensiuni se poate utiliza funcția `array()`, astfel:

```
> (v3dim <- array(1:12, dim = c(2, 3, 2)))
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Elementele din variabila tridimensională se pot afișa foarte ușor, acestea fiind parte dintr-un cub. Practic, se lucrează cu o colecție de două matrici cu dimensiunile de 2 rânduri pe 3 coloane.

```
> v3dim[, , 1]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Elementul care are valoarea 8 se găsește pe poziția: rândul 2, coloana 1, matricea 2, astfel:

```
> v3dim[2, 1, 2]
[1] 8
```

9.4 Liste

Lista este o colecție de elemente de tipuri diferite. Acestea se despart pe componente, iar în fiecare dintre acestea pot fi diferite tipuri de obiecte: vectori, matrici, dataframe-uri, liste.

```
> (lista <- list(48:52, "y", TRUE, 3+10i))
[[1]]
[1] 48 49 50 51 52

[[2]]
[1] "y"

[[3]]
[1] TRUE

[[4]]
[1] 3+10i

> class(lista)
[1] "list"
```

Numerele din parantezele pătrate duble reprezintă poziția componentelor din cadrul listei, iar numerele în paranteze simple reprezintă poziția elementelor din cadrul vectorului.

De exemplu, numărul 50 se găsește în prima componentă a listei, pe poziția a treia:

```
> lista[[1]][3]
[1] 50
```

Pentru a afișa componenta `i` dintr-o listă cu `n` componente se va proceda identic ca în cazul vectorilor. A doua componentă din listă este caracterul y:

```
> lista[[2]]
[1] "y"
```

Dar, elementul 2 este un vector de tip caracter care conține un singur element.

```
> class(lista[2])
[1] "list"
> class(lista[[2]])
[1] "character"
```

În exemplul precedent s-a observat o listă formată din patru vectori cu un singur element fiecare. Se construiește o listă cu vectori de dimensiuni

și tipuri diferite pentru a sesiza anumite diferențe în modul de acces la elementele componente.

```
> listacomplexa <- list(a = 15:30, b = "y",
                        c(TRUE, FALSE, TRUE, FALSE), 3+10i)
> class(listacomplexa)
[1] "list"
```

Lista `listacomplexa` conține patru vectori, astfel:
- primul cu denumirea `a` și valorile de la 15 la 30, inclusiv;
- al doilea cu numele `b`, care conține o singură valoare de tip caracter: `y`;
- al treilea este un vector de tip logic cu patru valori: `TRUE, FALSE, TRUE, FALSE`;
- ultimul conține un număr complex: `3+10i`.

```
> listacomplexa
$a
 [1] 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

$b
[1] "y"

[[3]]
[1]  TRUE FALSE  TRUE FALSE

[[4]]
[1] 3+10i
```

Se află numărul de componente din `listacomplexa`, prin comanda `length`:

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

Tipul datelor pentru prima compoentă din listă este:

```
> class(listacomplexa[[1]])
[1] "integer"
```

Dar același lucru se poate face și prin adresarea directă a numelui primului element, `a`, astfel:

```
> class(listacomplexa$a)
[1] "integer"
```

De asemenea se poate afla dimensiunea primei componente (un vector), prin două metode:

```
> length(listacomplexa[[1]])
[1] 16
> length(listacomplexa$a)
[1] 16
```

Tipul celorlalte elemente din listă se află în mod similar:

```
> class(listacomplexa[[2]])
[1] "character"
> class(listacomplexa[[3]])
[1] "logical"
> class(listacomplexa[[4]])
[1] "complex"
```

Pentru afișarea elementului `i` din primul vector se va utiliza comanda:

```
> listacomplexa[[1]][[3]]
[1] 17
```

Sau:

```
> listacomplexa$a[3]
[1] 17
```

O altă variantă, mai rar utilizată, este:

```
> listacomplexa[[c(1,3)]]
[1] 17
```

O adresare prin `listacomplexa[1,3]` nu va funcționa pentru că este specifică unei variabile de tip matrice și nu listă, precum cea din exemplul precedent. Utilizarea acestei comenzi va genera o eroare cu privire la numărul greșit de dimensiuni la care face referire:

```
> listacomplexa[1,3]
Error in listacomplexa[1, 3] : incorrect number of dimensions
```

Denumirea vectorilor/elementelor dintr-o listă pot fi afișate prin comanda `names()`:

```
> names(listacomplexa)
[1] "a" "b" ""  ""
```

În exemplul precedent doar primele două elemente au denumiri `a`, respectiv `b`.

9.5 Dataframe-uri

Un dataframe este o colecție de observații formată din unul sau mai mulți vectori și/sau factori de aceeași lungime, de oricare tip. Se vor lua ca exemplu cinci vectori care conțin informații privind caracteristicile unor autoturisme.

```
> marca <- c("Volskwagen","Renault","Fiat")
> motor <- c("Diesel","Benzina","Benzina")
> capacitate <- c(1600, 1900, 1200)
> putere <- c(75, 65, 55)
> transmisie <- c("Automata","Manuala","Manuala")
> (auto <- data.frame(marca, motor, capacitate, putere,
transmisie))
       marca   motor capacitate putere transmisie
1 Volskwagen  Diesel       1600     75   Automata
2    Renault Benzina       1900     65    Manuala
3       Fiat Benzina       1200     55    Manuala
```

După cum se observă, un dataframe este un fel de matrice în care fiecare coloană poate avea un tip diferit de date. În dataframe-ul `auto`, tipurile variabilelor sunt:

```
> unlist(lapply(auto, class))
     marca      motor capacitate     putere transmisie
  "factor"   "factor"  "numeric"  "numeric"   "factor"
```

Dataframe-urile sunt folosite mai ales pentru datele importate din diferite surse externe.

9.6 Operații aritmetice și logice cu vectori

Unul dintre cele mai mari avantaje ale utilizării mediului R în analiza statistică este faptul că a fost creat de statisticieni pentru statisticieni.

Demonstrația cea mai simplă se poate face prin ușurința cu care se realizează operațiile aritmetice și cele logice asupra vectorilor.

Adunarea unei valori la toate elementele unui vector se realizează prin simpla operație aritmetică, fără a utiliza structuri repetitive de parcurgere a elementelor vectorului. Un vector care să conțină numerele întregi de la 1 la 10 se crează astfel:

```
> (v <- 1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Se însumează valoarea "1" la toate elementele vectorului:

```
> (v <- v + 1)
 [1]  2  3  4  5  6  7  8  9 10 11
```

Adunarea a doi vectori este de asemenea foarte simplă, iar pentru aceasta se crează vectorii: `v1` și `v2`:

```
> v1 <- c(1, 2, 3, 4, 5)
> v2 <- c(10, 20, 30, 40, 50)
> v1 + v2
[1] 11 22 33 44 55
```

O altă operație poate fi înmulțirea valorilor celor doi vectori:

```
> v1 * v2
[1]  10  40  90 160 250
```

Dacă vectorii au lungimi diferite, operația aritmetică se va realiza repetând valorile vectorului mai scurt până la obținerea lungimii vectorului mai lung.

```
> v1 <- c(100, 200, 300, 400)
> v2 <- c(1, 2)
> v1 + v2
[1] 101 202 301 402
```

În cazul în care lungimea vectorului mai lung nu este perfect divizibilă cu lungimea vectorului mai scurt, comanda se va executa însă mediul R va afișa un mesaj de avertizare (în engleză *warning*):

```
> v1 <- c(100, 200, 300, 400)
> v2 <- c(1, 2, 3)
> v1 + v2
[1] 101 202 303 401
Warning message:
In v1 + v2 :
 longer object length is not a multiple of shorter object length
```

Subseturi

Selecția unor elemente dintr-un vector se realizează printr-un vector de tip logic transmis ca parametru vectorului în sine. De exemplu: se vor selecta din vectorul care conține numerele întregi de la 1 la 15 acele valori divizibile cu 3. Primul pas este crearea vectorului:

```
> (v <- 1:15)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

Pentru a afla numerele divizibile cu 3 se va utiliza operatorul `modulo` și anume %% care va returna restul împărțirii la 3.

```
> v %% 3
 [1] 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
```

Dacă se compară acest vector cu valoarea zero, se va crea un vector de tip logic cu valorile `FALSE` pentru valorile diferite de zero și `TRUE` pentru acele valori egale cu zero.

```
> v %% 3 == 0
 [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
 TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE
```

Vectorul de tip logic creat și se va transmite vectorului `v`, iar ca rezultat se obțin toate valorile divizibile cu 3.

```
> v[v %% 3 == 0]
[1]  3  6  9 12 15
```

9.7 Generarea datelor

Mediul R dispune de funcții foarte utile pentru generarea datelor atât de necesare la testări sau simulări, atunci când datele reale sau observate nu sunt disponibile. De fapt, când se discută de generarea datelor, în mod obișnuit, se face referire la generarea unor numere, aleator sau după o anumită regulă. Deoarece orice generare are în vedere un număr finit de valori, atunci se poate numi *secvență de numere*. În limba engleză este întâlnită denumirea *regular sequence*, prin care se înțelege că secvența de numere este generată după o anumită regulă.

O funcție foarte întâlnită este `seq()` (în engleză *Sequence Generation*), care poate primi ca argumente:

- `from`: de la valoarea ...;

- `to`: până la valoarea ...;

- `by`: după regula de generare: ((to - from)/(length.out - 1));

- `length.out`: numărul de valori generate, iar implicit are valoarea `NULL`;

- `along.with`: de-a lungul, adică ia lungimea argumentului.

Exemplu, în care se prezintă trei metode:

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(length.out = 10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Dacă se încearcă generarea unei secvențe de nouă numere, dar toate cuprinse între 0 și 1 se utilizează următoarea formă a funcției:

```
> seq(0, 1, length.out = 9)
[1] 0.000 0.125 0.250 0.375 0.500 0.625 0.750 0.875 1.000
```

Afișarea numerelor pare din intervalul 0 - 10 se poate face prin schimbarea formulei (a regulii de generare), prin argumentul `by`, astfel:

```
> seq(0, 10, by = 2)
[1]  0  2  4  6  8 10
```

O altă funcție prin care se pot genera automat anumite date, dar după regula multiplicării, este: `rep()`, cu argumentele: `x`, `times` și `length.out`. Multiplicarea numărului 1 de 5 ori:

```
> rep(1, 5)
[1] 1 1 1 1 1
```

Multiplicarea secvenței de numere de la 1 la 3 de două ori:

```
> rep(1:3, 2)
[1] 1 2 3 1 2 3
```

Multiplicarea fiecărui număr din intervalul de la 1 la 3 de două ori:

```
> rep(1:3, each = 2)
[1] 1 1 2 2 3 3
```

Crearea unei serii de date prin generarea aleatoare de numere este importantă pentru funcțiile de analiză statistică. În această categorie, mediul R dispune de o multitudine de funcții, iar utilizarea uneia anume ține foarte mult de regula de generare dorită.

Funcția	**Distribuția**	**Funcția**	**Distribuția**
`rbeta()`	Beta	`rlogis()`	Logistic
`rbinom()`	Binomial	`rmultinom()`	Multinomial
`rcauchy()`	Cauchy	`rnbinom()`	Neg. Binomial
`rchisq()`	Chi-square	`rnorm()`	Normal
`rexp()`	Exponential	`rpois()`	Poisson
`rf()`	F	`rsignrank()`	Signed Rank
`rgamma()`	Gamma	`rt()`	Student's t
`rgeom()`	Geometric	`runif()`	Uniform
`rhyper()`	Hypergeometric	`rweibull()`	Weibull
`rlnorm()`	Log Normal	`rwilcox()`	Wilcoxon Rank Sum

Funcția `runif()` generează n numere aleatoare după o distribuție uniformă într-un interval dat. Argumentele funcției sunt:

- `n`: numărul de elemente;
- `min` și `max`: intervalul în care se vor genera numerele.

Generarea a 5 numere cu distribuție uniformă, în intervalul 1 și 10:

```
> runif(5, min = 1, max = 10)
[1] 6.949369 7.230502 2.810380 4.368844 7.493939
```

Instrucțiunile din exemplul precedent vor genera la fiecare execuție alte numere, dar dacă se dorește ca execuția instrucțiunilor să genereze mereu aceleași numere trebuie inițializat un parametru de sistem (`set.seed`) înainte de rularea oricărei funcții, astfel:

```
> set.seed(123)
> runif(5, min = 1, max = 10)
[1] 3.588198 8.094746 4.680792 8.947157 9.464206
```

Generarea unei serii de numere cu o distribuție normală se realizează cu funcția `rnorm()`, care poate primi argumentele:

- `n`: numărul de valori generate;
- `mean`: media;
- `sd`: abaterea standard (în engleză *standard deviation*).

Exemplu de generare a 5 numere cu distribuție normală, de medie 0 și abatere standard 1:

```
> set.seed(123)
> rnorm(5, mean = 0, sd = 1)
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

Alegerea aleatoare a unor numere dintr-o serie care există deja în memorie, generată sau încărcată dintr-o bază de date se realizează prin funcția `sample()`, cu următoarele argumente:

- `x`: vectorul de elemente din care se va realiza extragerea;
- `size`: numărul de elemente care se va extrage;
- `replace`: care poate lua valorile: `TRUE` sau `FALSE`, stabilind dacă numerele extrase vor fi introduse din nou în urnă, respectiv sunt eliminate din urnă.

În cazul în care funcția este utilizată fără argumente, se va realiza o extragere a tuturor elementelor din vectorul dat, în mod aleator, prin inițializarea unui vector și rearanjarea aleatoare a numerelor din vector, astfel:

```
> x <- 1:10
> sample(x)
 [1]  9  7  6 10  4  8  3  2  1  5
```

Se ia ca exemplu extragerea numerelor câștigătoare de la Loteria Română, la jocul 6 din 49. Mai întâi se generează numerele de la 1 la 49, apoi se extrag 6 numere din 49:

```
> set.seed(1234)
> sample(1:49, 6)
[1]  6 30 29 47 39 46
```

Ultimul exemplu este extragerea a unui număr de elemente dintr-un vector cu reintroducerea acestora în seria de date (așa numita eșantionare cu înlocuire), prin argumentul `replace`.

```
> set.seed(1234)
> sample(10, 5, replace = TRUE)
[1] 2 7 7 7 9
```

În exemplul precedent se observă că numărul 2 a fost extras de două ori, deoarece fiind introdus în urnă a avut din nou șansa de a fi extras.

Capitolul 10

Structuri de programare

Cunoașterea structurilor de programare, întâlnite și sub denumirea de bucle (în engleză *loops*), se va dovedi utilă când se dorește repetarea unei instrucțiuni sau a unui bloc întreg de instrucțiuni pentru toate elementele dintr-un vector, iar alteori doar pentru unele elemente din acel vector.

10.1 if - else

Instrucțiunea `if()` se utilizează după cum este cunoscută și din alte limbaje de programare, iar sintaxa cea mai simplă este `if (cond) expr` (dacă este îndeplinită condiția `cond` se execută instrucțiunea `expr`).

Evident, în funcție de cât de complexă este problema propusă spre rezolvare, sintaxa poate avea mai multe cazuri, toate desprinse din sintaxa completă, prin lipsa unor elemente:

```
if (conditie1) {
   # executa niște expresii
} else if (conditie2) {
   # executa alte expresii
} else {
   # executa alte expresii
}
```

O singură condiție și o singură instrucțiune:

```
if (conditie) expresie
```

Exemplu:

```
> vect <- 10:20
> if (vect[2] == 11) print ("elementul 2 are valoarea 11")
[1] "elementul 2 are valoarea 11"
```

O singură condiție și mai multe instrucțiuni:

```
if (conditie) {
   # executa niște expresii
}
```

Exemplu:

```
> vect <- 10:20
> if (vect[2] == 11) {
      vect[2] <- 33
      print ("elementul 2 are o altă valoare!")
  }
[1] "elementul 2 are are o altă valoare!"
```

O singură condiție cu două instrucțiuni, una pentru cazul îndeplinirii condiției, o alta pentru cazul contrar:

```
if (conditie) expresie else alta_expresie
```

Exemplu:

```
> vect <- 10:20
> if (vect[3] == 11) {
       print("elementul 3 are valoarea 11")
   } else {
       print ("elementul 3 nu are valoarea 11")
   }
[1] "elementul 3 nu are valoarea 11"
```

O singură condiție cu multiple instrucțiuni, un set pentru cazul îndeplinirii condiției, un alt set pentru cazul contrar:

```
if (conditie) {
   # executa niște expresii
} else {
   # executa alte expresii
}
```

Exemplu:

```
> vect <- 10:20
> if (vect[2] == 11) {
        print("elementul 2 are valoarea 11")
        vect[2] <- 33
   } else {
        print("elementul 2 nu are valoarea 11")
        vect[2] <- 11
   }
```

Mai există o versiune a structurii `if-else` numită vectorizată: `ifelse(condiție,expr1,expr2)`. Să luăm ca exemplu afișarea valorii `1` pentru toate valorile mai mici decât `15` și a valorii `0` pentru toate valorile mai mari sau egale cu `15` din vectorul `10:20`:

```
> (vect <- 10:20)
 [1] 10 11 12 13 14 15 16 17 18 19 20
> ifelse(datelemele < 15, 1, 0)
 [1] 1 1 1 1 1 0 0 0 0 0 0
```

10.2 for

Sintaxa instrucțiunii este asemănătoare cu cea cunoscută din alte limbaje, dar are totuși un specific dat de experiența celor care au creat limbajul în domeniul statisticii: `for(var in seq) expr`. Cu alte cuvinte, pentru toate valorile `var` din secvența de date `seq` se repetă instrucțiunile `expr`.

Repetă un set de instrucțiuni de un anumit număr de ori:

```
for (index in interval) {
# set de instructiuni
}
```

Variabila `index` primește pe rând valorile din interval și execută pentru fiecare valoare pe care o primește setul de instrucțiuni.

Afișarea unui salut de 5 ori:

```
> for (i in 1:2) print("Salut!")
[1] "Salut!"
[1] "Salut!"
```

Se observă că variabila `i` a luat valori între 1 şi 5, iar afișarea s-a făcut în mod corespunzător. Pentru execuția a două instrucțiuni pe fiecare buclă, acestea se vor cuprinde în acolade:

```
> for(i in 1:2){
    mesaj <- paste("Salut!", i)
    print(mesaj)
  }
[1] "Salut! 1"
[1] "Salut! 2"
```

În general, în lucrul cu vectori există mai multe posibilități de rezolvare pentru aceeași problemă. Spre exemplu, construirea unui vector cu patru elemente și afișarea acestora prin diverse metode:

```
> x <- c("a", "b")
```

Se afișează prin parcurgerea vectorului de la primul elementul patru, deoarece se cunoaște numărul de elemente:

```
> for(i in 1:2) print(x[i])
[1] "a"
[1] "b"
```

În cazul în care nu se cunoaște câte elemente conține vectorul, se poate afla cu instrucțiunea `length()`.

```
> for(i in seq(length(x))) print(x[i])
[1] "a"
[1] "b"
```

Parcurgerea unui vector de la primul la ultimul element se poate realiza și cu funcția `seq_along()`, care returnează un vector numeric care conține numărul fiecărui element de la `1` la `n`.

```
> for (i in seq_along(x)) print(x[i])
[1] "a"
[1] "b"
```

Un alt mod ar fi parcurgerea vectorului de caractere, de la prima la ultima literă, astfel:

```
> for (litera in x) print(litera)
[1] "a"
[1] "b"
```

Dacă este folosit împreună cu `if`, se poate sări peste un anumit număr de iterații:

```
> for (index in 1:15) {
   if (index > 12) print(index)
 }
[1] 13
[1] 14
[1] 15
```

10.3 while

Executarea unei instrucțiuni atâta timp cât este îndeplinită o condiție se poate face cu instrucțiunea `while()`, iar sintaxa este `while(conditie) expr`. Pentru execuția mai multor instrucțiuni, acestea se vor cuprinde în acolade.

```
> i <- 0
> while (i < 3) {
   i <- i + 1
   print(i)
 }
[1] 1
[1] 2
[1] 3
```

Dacă în setul de instrucțiuni, odată cu parcurgerea fiecărei iterații, valoarea condiției rămâne neschimbată poate rezulta o repetare infinită a setului de instrucțiuni, cunoscută sub denumirea de buclă infinită.

Exemplu de utilizare corectă:

```
> index <- 0
> while(index < 3){
   print(index)
   index <- index + 1
 }
[1] 0
[1] 1
[1] 2
```

Generarea unei bucle infinite, deoarece valoarea variabilei `index` nu se modifică prin parcurgerea setului de instrucțiuni:

```
> index <- 0
> while(index < 5){
   print(index)
 }
[1] 0
[1] 0
[1] 0
...
```

10.4 repeat

În cazul instrucțiunii `repeat` expresiile cuprinse în acolade vor fi executate până când este întâlnită instrucțiunea `break`. Sintaxa simplă este:

```
repeat {
# set de instructiuni
break
}
```

Aceasta înseamnă că trebuie să mai existe în cod o condiție care se va verifica și va conduce la executarea instrucțiunii de ieșire din buclă.

Exemplu:

```
> index <- 0
> repeat {
   index <- index + 1
   print(index)
   if(index >= 3) break()
}
[1] 1
[1] 2
[1] 3
```

Un exemplu de utilizare incorectă este omiterea instrucțiunii `break` sau neîndeplinirea condiției care duce la executarea acesteia.

```
> index <- 0
> repeat {
   print(index)
   if (index > 5) break
 }
```

Capitolul 11

Funcții

Funcțiile sunt obiecte de prim rang în R. Există funcții incluse în pachetele de bază ale R sau în cele instalate suplimentar și funcții create de utilizator. Se pune problema creării propriilor funcții atunci când pentru rezolvarea unei probleme este nevoie de scrierea mai multor instrucțiuni, iar acestea se repetă de mai multe ori, dar cu anumiți parametri care își pot schimba valorile. Parametri funcțiilor se mai numesc și argumente.

11.1 Declarare

O funcție se declară prin cuvântul cheie `function`.

Trebuie reținut că funcțiile pot fi acceptate de alte funcții ca argumente sau pot primi ca argumente alte funcții. Sintaxa este:

```
FunctieUtilizator <- function (<argumente>) {
   # set de instructiuni
   return(value)
}
```

Se va crea pentru exemplificare o funcție cu patru argumente fără a le declara tipul:

```
> FunctieUtilizator <- function(a, b, c, d) {
   print(paste("a =", a, "de tip:", class(a)))
   print(paste("b =", b, "de tip:", class(b)))
   print(paste("c =", c, "de tip:", class(c)))
```

```
   print(paste("d =", d, "de tip:", class(d)))
   if (a==1) "unu" else "diferit de unu"
 }
> FunctieUtilizator(1, "Nume", 3.14, TRUE)
[1] "a = 1 de tip: numeric"
[1] "b = Nume de tip: character"
[1] "c = 3.14 de tip: numeric"
[1] "d = TRUE de tip: logical"
[1] "unu"
> class(FunctieUtilizator)
[1] "function"
```

Se observă că au fost declarate patru argumente: a, b, c, d, iar în momentul apelării funcției, aceste argumente au primit tipul: `numeric`, `character`, `numeric`, respectiv `logical`. Ultima instrucțiune din cadrul funcției era verificarea valorii argumentului `a`, iar valoarea găsită (1) a determinat afișarea șirului de caractere ”unu”. Ultima instrucțiune executată a fost `class()`, care a afișat tipul obiectului `FunctieUtilizator`, și anume ”`function`”.

Se crează o altă funcție pentru însumarea a două numere, iar argumentele a și b vor fi inițializate cu valoarea `0`, adică li se va atribui valoarea implicită `0`:

```
> FunctieSuma <- function(a = 0, b = 0) {
   print(paste("a =", a, "de tip:", class(a)))
   print(paste("b =", b, "de tip:", class(b)))
   return(a + b)
 }
> FunctieSuma(1, 2)
[1] "a = 1 de tip: numeric"
[1] "b = 2 de tip: numeric"
[1] 3
```

Prin apelarea funcției `FunctieSuma()` cu argumentele 1 și 2 rezultatul va fi, desigur, 3, iar tipul variabilelor va fi afișat, ambele fiind de tip `numeric`.
Dacă se apelează funcția `FunctieSuma()` cu argumente non-numerice, mediul R va genera o eroare:

```
> FunctieSuma("Nume", "Prenume")
[1] "a = Nume de tip: character"
[1] "b = Prenume de tip: character"
Error in a + b : non-numeric argument to binary operator
```

Se observă că deși s-au inițializat argumentele cu valoarea 0, în momentul apelării funcției cu șiruri de caractere, ”Nume” și ”Prenume”, mediul R a schimbat automat tipul argumentelor, din `numeric` în `character`, dar la instrucțiunea de însumare a generat o eroare prin care arată că argumentele ne-numerice au fost transmise unui operator aritmetic. În acel moment execuția funcției a fost oprită. Astfel de erori trebuie să fie preîntâmpinate prin verificări prealabile, necesare înainte de operațiile avute în vedere, în funcție de specificul acestora.

11.2 Valori returnate

În secțiunea precedentă s-a utilizat deja `return` pentru a întoarce sistemului rezultatul însumării celor două argumente.
Se modifică ultima funcție creată pentru a realiza doar calculul, fără a mai afișa tipul argumentelor primite:

```
> FunctieSuma <- function(a = 0, b = 0) {
   return(a + b)
 }
> FunctieSuma(1, 2)
[1] 3
```

Rezultatul însumării numerelor 1 cu 2 a fost 3, iar după această operație simplă se crează o altă variabilă în care se înmulțește cu 2 rezultatul returnat de funcție și se obține valoarea 6.

```
> (variabila <- FunctieSuma(1, 2) * 2)
[1] 6
```

Mai mult decât atât, se poate apela funcția prin ea însăși, astfel:

```
> FunctieSuma(FunctieSuma(1, 2), 3)
[1] 6
```

Mediul R va executa mai întâi însumarea numerelor 1 cu 2, iar rezultatul va fi transmis din nou funcției create, dar de data aceasta ca argument, la care va adăuga argumentul 3. Rezultatul final returnat de funcție a fost 6.

11.3 Argumente locale și globale

Argumentele unei funcții pot fi aflate cu instrucțiunea `args()`.

```
> FunctieSuma <- function(a = 0, b = 0) {
   return(a + b)
 }
> args(FunctieSuma)
function (a = 0, b = 0)
NULL
```

Adesea, numele argumentelor sunt utile, mai ales în funcțiile cu mai multe argumente, iar valorile implicite sunt uneori necesare pentru ca funcția să ruleze fără erori. Inițializarea argumentelor este recomandată pentru cele care sunt utilizate obligatoriu în interiorul funcției.

Argumentul `...` se utilizează atunci când numărul argumentelor nu este cunoscut. Spre exemplu se poate realiza o funcție de însumare a mai multor numere.

```
> FunctieSuma <- function(...) {
   return(sum(...))
 }
> FunctieSuma(1,2,3)
[1] 6
```

Capitolul 12

Statistică descriptivă în R

12.1 Introducere

Statistica este o știință care implică diverse tipuri de date, care sunt utilizate pentru a le clasifica și organiza, iar apoi pentru a le interpreta și analiza.

Indiferent de domeniul la care este aplicată, statistica (cel puțin la nivel descriptiv) poate fi utilizată în mod similar. Fie că este vorba de obiecte fizice, sau entități economice, ori unități sociale, acestea pot fi împărțite pe categorii, măsurate în mod numeric iar descrierea acestora se face la fel.

Există așadar niște entități de bază ale statisticii, utilizate în cadrul operațiunii descriptive a variabilelor. Acestea sunt nivelurile de măsurare:

- nominal
- ordinal
- interval
- raport

O variabilă este orice caracteristică ce variază de la o unitate observată la alta (tip de activitate economică pentru unități economice, prețul pentru bunuri și servicii, înălțimea pentru oameni, etc.) Unele dintre acestea pot fi măsurate în mod numeric, altele definesc niște categorii pe care oamenii le interpretează calitativ.

În funcție de modul cum este măsurată o anumită caracteristică, se pot utiliza anumite operațiuni descriptive sau altele. Primele două niveluri

de măsurare definesc așa numitele variabile categoriale (sau calitative), iar ultimele două pe cele numerice (sau cantitative, ori metrice).

Variabilele nominale sunt cele care împart unitățile de studiu în categorii pentru care ordinea nu este importantă. De exemplu împărțirea localităților pe urban și rural, într-o astfel de colecție nu contează care dintre categorii este prima sau a doua. Este doar o enumerare a acestora, iar ele pot fi prezentate în orice ordine.

Ca și la variabilele nominale, cele ordinale împart unitățile observate în categorii, însă (așa cum se spune numele), ordinea acestora este foarte importantă. De exemplu, dacă unitățile economice sunt ordonate după criteriul mărimii, în categoriile "foarte mari", "mari", "mici" și "foarte mici", se poate vedea foarte clar că aceste categorii sunt ordonate într-o anumită direcție (în acest caz în ordine descendentă a mărimii).

În chestionarele cantitative, categoriile pot fi reprezentate de niște numere: 1. Urban și 2. Rural, iar în calculator se introduc numerele 1 și 2 în locul categoriilor. Aceste numere sunt doar niște simple înlocuitoare pentru categoriile respective, însă nu au semnificație numerică. În matematică, cifra 2 este mai mare decât cifra 1 însă în acest exemplu cifra 2 (mediu rural) nu este matematic mai mare decât cifra 1 (mediu urban). Cifrele sunt utilizate doar pentru a face referire la categoriile respective, nu se poate spune că ruralul este dublul urbanului.

Variabilele numerice, pe de altă parte, sunt cele pentru care o anumită caracteristică poate fi măsurată printr-un număr pe baza căruia se pot efectua operațiuni matematice. Vârsta unei persoane, sau greutatea acesteia sunt caracteristici cantitative, care pot fi măsurate foarte precis cu ajutorul unor numere. Acest lucru este valabil pentru ambele niveluri de măsurare, interval și raport, singura diferență între acestea fiind faptul că nivelul de măsurare raport are o caracteristică suplimentară față de nivelul interval, anume faptul că are zero absolut.

O caracteristică numerică poate sau nu să aibă zero absolut (valoarea la care se constată absența completă a caracteristicii studiate). De exemplu, coeficientul de inteligență a unei persoane este o variabilă care nu are zero absolut, întrucât testul pe baza căruia a fost alocat un coeficient sau altul este relativ (am putea spune chiar subiectiv). O valoare egală cu zero la un test de inteligență nu înseamnă absența completă a inteligenței, așadar valoarea zero pentru această variabilă nu este absolută.

Pentru anumite caracteristici fizice sau geografice (cum sunt înălțimea, greutatea sau distanța), valoarea zero este absolută. Pentru greutate, zero înseamnă absența completă a greutății, iar pentru distanță valoare zero înseamnă absența completă a distanței dintre două puncte (este punctul

de pornire, valoarea de la care începe distanța).

Se va vedea că toate aceste distincții sunt importante pentru a putea distinge între diferitele operațiuni descriptive care pot fi aplicate unor variabile. Este foarte important, întrucât anumite operațiuni nu pot fi aplicate decât pentru anumite variabile, de exemplu nu se poate calcula o medie aritmetică între categoriile Urban și Rural.

În fine, în cadrul variabilelor numerice mai trebuie făcută o ultimă distincție între variabilele discrete și variabilele continue. Cele discrete au un număr finit și numărabil de valori posibile, iar cele continue au un număr infinit de valori. Numărul de copii reprezintă un exemplu de variabilă discretă, în timp ce greutatea are un număr infinit de valori. Fiecare persoană are o greutate unică la un infinit de zecimale posibile, probabilitatea ca două persoane să aibă exact aceeași greutate fiind practic zero.

În R, există anumite obiecte care se potrivesc cu cele patru niveluri de măsurare. Corespondentul variabilelor categoriale poartă denumirea de "factor" în limbajul R, iar pentru variabilele numerice există obiecte numerice.

Variabilele nominale în R sunt denumite simplu "factor", iar cele ordinale poartă denumirea de "ordered factor". Pentru ambele tipuri, categoriile poartă denumirea de "level" (care în limba română poate fi tradus ca "nivel" și uneori se mai utilizează ca atare în locul categoriilor).

```
> factor(c("B", "A", "C"))
[1] B A C
Levels: A B C
```

Așa cum a fost arătat la secțiunea 9.2, dacă ordinea categoriilor (nivelurilor) nu este specificată în vreun fel anume atunci R le ordonează, în mod implicit, în ordine alfabetică. Dacă se dorește o ordine anume, atunci se poate utiliza argumentul `levels` din funcția `factor()`.

```
> factor(c("B", "A", "C"), levels = c("C", "A", "B"))
[1] B A C
Levels: C A B
```

Foarte mulți utilizatori confundă ordonarea nivelurilor cu variabilele ordinale, însă aici nu este vorba decât de o simplă aranjare într-o altă ordine decât cea alfabetică. Este ca și cum s-ar rearanja categoriile "galben", "verde" și "albastru" într-o altă ordine decât cea alfabetică

"albastru", "galben" și "verde". Această rearanjare a categoriilor nu înseamnă că o variabilă este măsurată la nivel ordinal, ci tot la nivel nominal (unde ordinea aranjării acestora nu contează deloc).

Pentru variabilele ordinale există un argument suplimentar numit `ordered`:

```
> factor(c("B", "A", "C"), levels = c("C", "A", "B"),
ordered = TRUE)
[1] B A C
Levels: C < A < B
```

Spre deosebire de exemplele anterioare, se poate observa că există o anumită ordine a categoriilor: prima fiind `C`, apoi `A`, apoi `B`, între ele fiind semnul `<`.

Un exemplu concret de variabilă ordinală îl reprezintă cele cu scală de răspuns de tip Likert:

```
> set.seed(12345)
> (var1 <- sample(1:4, 20, replace = TRUE))
 [1] 3 4 4 4 2 1 2 3 3 4 1 1 3 1 2 2 2 2 1 4

> (var1 <- factor(var1, labels=c("Foarte puțin", "Puțin",
"Mult", "Foarte mult"), ordered=TRUE))
 [1] Mult         Foarte mult  Foarte mult  Foarte mult
 [5] Puțin        Foarte puțin Puțin        Mult
 [9] Mult         Foarte mult  Foarte puțin Foarte puțin
[13] Mult         Foarte puțin Puțin        Puțin
[17] Puțin        Puțin        Foarte puțin Foarte mult
Levels: Foarte puțin < Puțin < Mult < Foarte mult
```

Prima comandă din acest exemplu crează obiectul `var1`, care conține 20 de numere cu valori între 1 și 4 (de remarcat includerea întregii comenzi între două paranteze, cu rolul de a efectua comanda și de a afișa rezultatul acesteia pe ecran, în același timp). În prealabil, a fost utilizată funcția `set.seed()` pentru ca numerele generate aleator să fie aceleași la o utilizare ulterioară a comenzii `sample()`.

Valorile generate nu au semnificație matematică, doar înlocuiesc categoriile "Foarte puțin", "Puțin", "Mult" și "Foarte mult", care au în mod evident o ordonare precisă de la extrema foarte puțin la extrema foarte mult. A doua comandă transformă obiectul `var1` într-o variabilă ordinală, prin apelarea funcției `factor()` și a argumentelor acesteia

`ordered` și `labels`, dintre care ultimul nu face altceva decât să acorde etichete valorilor de la 1 la 4.

În acest exemplu nu a mai fost nevoie de argumentul `levels`, întrucât acestea au fost preluate în mod automat de către R în ordine crescătoare a numerelor (la fel cum în cazul literelor se aranjau automat în ordine alfabetică).

Pentru variabilele numerice, în R nu se face distincție între nivelurile interval și raport, acest lucru fiind interpretabil numai pentru utilizatori. Calculatorul nu cunoaște a priori dacă o valoare zero reprezintă un zero absolut sau unul relativ, doar oamenii pot să facă această distincție.

În continuarea acestui capitol, vor fi prezentate metode grafice și metode numerice de sumarizare a datelor, utilizate pentru fiecare nivel de măsurare în parte. Există metode grafice atât pentru variabile categoriale cât și pentru variabile numerice, iar metodele de sumarizare numerică vor fi aplicate numai variabilelor numerice, cu excepția unor cazuri foarte izolate în care vor putea fi aplicate și unor tipuri speciale de variabile ordinale.

12.2 Descrierea variabilelor categoriale

Pentru aceste tipuri de variabile, sunt posibile doar formele descriptive care nu fac apel la operațiuni matematice. Tabelele de frecvență reprezintă o bună metodă de sumarizare, iar în mod grafic pot fi efectuate o serie de grafice, în marea lor majoritate de tip bară (bar chart) sau circulare (pie chart).

12.2.1 Tabele de frecvență

Tabelele de frecvență sunt printre cele mai utilzate forme de sumarizare a datelor categoriale. Ele prezintă, într-o ordine oarecare, categoriile variabilei respective, precum și frecvența de apariție a categoriei respective (altfel spus, câte unități de observație există în eșantion, care fac parte dintr-o categorie sau alta).

Se poate utiliza exemplul anterior cu obiectul `var1` care conține răspunsuri pe o scală de tip Likert:

```
> table(var1)
var1
Foarte puțin        Puțin         Mult  Foarte mult
           5            6            4            5
```

Multe programe de analiză statistică (cum este spre exemplu SPSS) afișează în fereastra de rezultate niște tabele de frecvență cu un design mult mai sugestiv, iar de obicei categoriile sunt prezentate pe linii iar frecvențele pe coloane. Aceast aparent minus al lui R este de fapt un mare plus, întrucât pentru toate celelalte pachete statistice informațiile prezentate în fereastra de rezultate sunt pur descriptive, sunt afișate și nu mai pot fi utilizate în continuare.

Pe de altă parte, R este un limbaj superior întrucât orice obiect (inclusiv cele de tip rezultat) pot fi stocate și reutilizate ulterior. Următorul cod este sugestiv:

```
> tab.fr <- table(var1)
> tab.fr[4]
Foarte mult
          5
```

Prima linie de cod efectuează tabelul de frecvențe pe variabila `var1` și o stochează în obiectul `tab.fr`, iar a doua linie de cod arată că un obiect de tip tabel de frecvențe în R se poate comporta întocmai ca un vector. A fost interogată a patra categorie a tabelului, iar pe ecran a fost afișată împreună cu frecvența de apariție a acesteia.

Așadar, R este mult mai flexibil în ceea ce privește utilizarea și reutilizarea obiectelor, indiferent că vorbim de date de intrare sau de rezultate ale unor acțiuni asupra acestora. În plus, obiectele rezultat din R pot fi ușor transformate pentru a fi trimise către o multitudine de alte medii de utilizare. De exemplu, există pachete în R care transformă astfel de tabele de frecvență în format HTML, altele care transformă în format LaTeX pentru publicații profesionale, altele care preiau aceste tabele și construiesc o imagine (de tip .jpg, .png etc.) cu un design particular, în principiu orice este posibil, singura limită fiind imaginația utilizatorului.

Un exemplu de formatare profesională este prezentat în tabelul 12.1, în care sunt prezente:

- valorile de la 1 la 4 generate anterior;
- etichetele acestora (adică cele patru categorii de răspuns);
- frecvențele absolute (F.A.), numărul de observații din fiecare categorie;
- frecvențele relative (F.R.), obținute prin împărțirea frecvențelor absolute la totalul de 20, suma acestora fiind 1.

Tabelul 12.1: Tabel de frecvență pentru variabila `var1`

		F.A.	F.R.
1.	Foarte puțin	5	0.25
2.	Puțin	6	0.30
3.	Mult	4	0.20
4.	Foarte mult	5	0.25
	Total	20	1.00

Acesta este un tip de tabel potrivit pentru tipărirea în publicații profesionale, obținut pe baza obiectului rezultat din R. Sunt mai multe pachete în R care pot genera astfel de tabele profesionale, unul dintre acestea fiind `Sweave`.

În exemplul de mai sus, distribuția de frecvențe relative este simplu de obținut prin împărțirea la volumul eșantionului, obținut fie cu:

```
> length(var1)
[1] 20
```

sau cu:

```
> sum(tab.fr)
[1] 20
```

La rândul lor, frecvențele relative pot fi obținute fie cu:

```
> tab.fr/sum(tab.fr)
var1
Foarte puțin        Puțin         Mult  Foarte mult
        0.25         0.30         0.20         0.25
```

sau cu:

```
> prop.table(tab.fr)
var1
Foarte puțin        Puțin         Mult  Foarte mult
        0.25         0.30         0.20         0.25
```

12.2.2 Reprezentări grafice

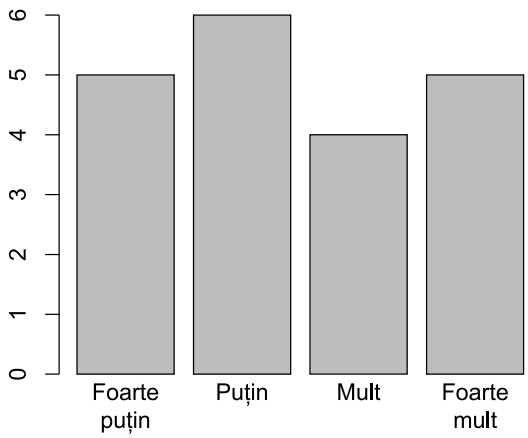
O altă modalitate des utilizată pentru descrierea variabilelor calitative sunt reprezentările grafice. Mediul R are unul dintre cele mai puternice motoare grafice existente în prezent, dezvoltate în laboratoarele Bell.

În cazul variabilor categoriale, cele mai utilizate reprezentări grafice sunt diagramele bară (în engleză *bar chart*) și diagramele circulare (în engleză *pie chart*).

Diagramele bară sunt formate dintr-o serie de bare orizontale sau verticale a căror lungime (respectiv înălțime) sunt egale cu frecvențele absolute ale categoriilor reprezentate.

```
> barplot(tab.fr, names.arg = c("Foarte\nputin", "Puțin\n",
"Mult\n", "Foarte\nmult"))
```

Figura 12.1: Diagramă bară verticală



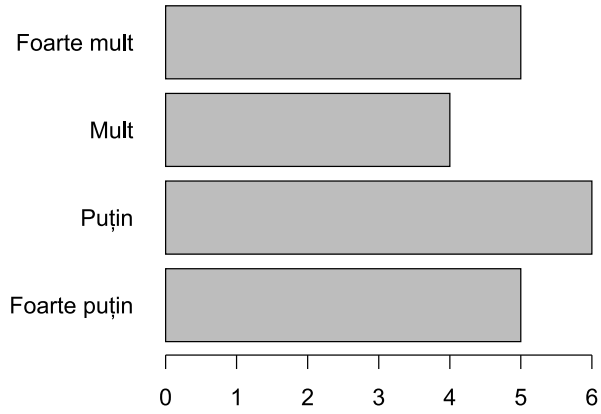
În figura 12.1, cele patru bare verticale corespund frecvențelor absolute din tabelul 12.1, respectiv 5 pentru categoria "Foarte puțin", 6 pentru categoria "Puțin", 4 pentru categoria "Mult" și 5 pentru categoria "Foarte mult".

Argumentul "`names.arg`" are și un efect cosmetic, prin introducerea caracterului "`\n`" pentru a afișa etichetele pe două rânduri, un truc foarte util atunci când etichetele sunt prea lungi pentru lățimea barelor.

O versiune mai bună a diagramei bară pe verticală este cea cu barele dispuse pe orizontală, așa cum este prezentat în figura 12.2, obținută cu comenzile:

```
> par(mar = c(2,6,1,1))
> barplot(tab.fr, horiz = TRUE, las = 1)
```

Figura 12.2: Diagramă bară orizontală



Spre deosebire de comanda pentru diagrama verticală, au fost utilizate două noi argumente: `horiz` controlează orientarea pe orizontală a barelor, în timp ce parametrul `las` controlează orientarea etichetelor (în acest caz, valoarea 1 însemnând tot pe orizontală). De această dată nu a mai fost necesară modificarea etichetelor întrucât pe orizontală există suficient spațiu pentru scrierea acestora.

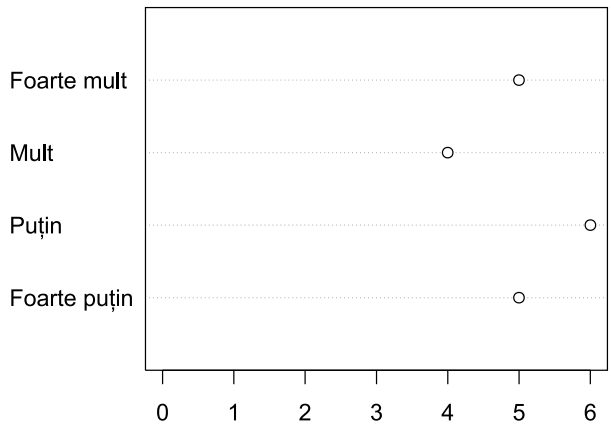
Comanda `par()` controlează o întreagă serie de parametri grafici, printre care argumentul `mar` pentru dimensiunea marginilor de jos, stânga, sus și dreapta, valorile fiind reprezentate în număr de linii.

După cum se poate vedea, există o multitudine de parametri și argumente disponibile pentru a controla diverse aspecte vizuale ale unei diagrame. Spre exemplu, la comanda `barplot()` mai este un argument numit `col` care controlează culorile barelor (poate fi utilizată o singură culoare pentru toate barele sau un vector de culori de lungime egală cu numărul de bare), sau un argument numit `legend.text` care produce o legendă atunci când este cazul etc.

Există un principiu de bază în construirea diagramelor, care prevede obținerea unui echilibru între cerneala consumată și informația transmisă. Situația ideală este atunci când se transmite maxim de informație cu minim de cerneală consumată. Din acest motiv, un alt tip de diagramă care transmite aceeași informație dar cu mai puțină cerneală consumată este afișat în figura 12.3:

```
> par(mar = c(2,6,1,1))
> dotchart(tab.fr, xlim = c(0, 6))
```

Figura 12.3: Diagramă cu puncte



Diagramele bară sunt printre cele mai utilizate, atât în publicațiile profesionale cât și în cadrul documentelor uzuale. La fel de utilizată este și diagrama circulară (în engleză *pie chart*), care prezintă nu atât frecvențele absolute ale categoriilor din tabel cât mai degrabă frecvențele relative ale acestora (desigur, așa cum s-a arătat anterior, frecvențele relative se obțin foarte ușor prin împărțirea la total).

Următoarele comenzi produc diagrama circulară din figura 12.4, în care s-a utilizat pe post de culori, în argumentul `col`, o serie de patru nuanțe de gri obținute cu comanda `gray()`:

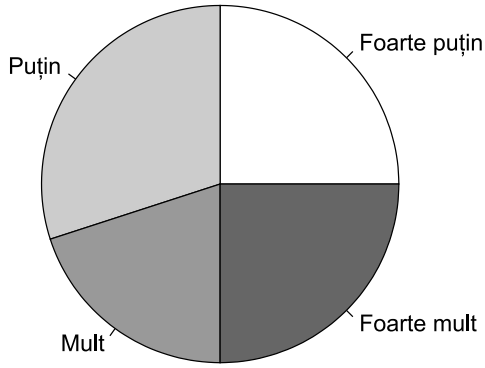
```
> par(mar = c(0, 0, 0, 0))
> pie(tab.fr, col = gray(seq(1.0, 0.4, length = 4)))
```

La fel ca în cazul tuturor celorlalte diagrame, se pot controla orice fel de parametri grafici și îmbunătăți diagrama prin adăugarea oricăror informații adiționale. De exemplu, ar fi interesant de vizualizat și frecvențele relative pentru fiecare felie în parte, întrucât în varianta ei predefinită diagrama circulară nu afișează decât etichetele categoriilor.

Codul necesar obținerii diagramei din figura 12.5 ajunge să fie ușor mai complex, însă permite flexibilitate maximă în adăugarea sau scoaterea fiecărui detaliu.

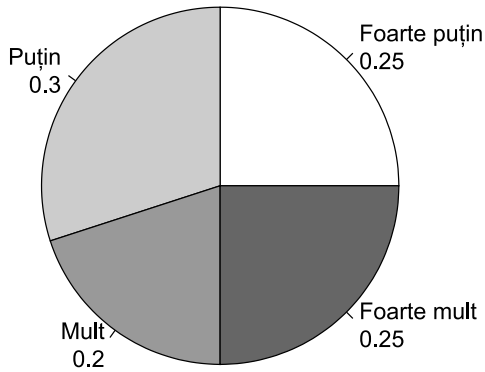
```
> par(mar = c(0, 0, 0, 0))
```

Figura 12.4: Diagramă circulară




```
> pie(tab.fr, col = gray(seq(1.0, 0.4, length = 4)),
      labels = paste(names(tab.fr),
                     prop.table(tab.fr),
                     sep = "\n"))
```

Figura 12.5: Diagramă circulară îmbunătățită



În acest exemplu s-a făcut apel la argumentul `labels` al comenzii `pie()`, unde s-a lipit cu comanda `paste()` toate frecvențele relative de fiecare etichetă în parte. Etichetele se regăsesc chiar în obiectul `tab.fr` și pot fi extrase cu comanda `names()`. Cele două seturi de valori (etichetele și frecvențele relative) au fost separate cu același caracter "`\n`", pentru așezarea frecvențelor relative pe un alt rând (altfel ar fi fost așezate în dreapta etichetelor).

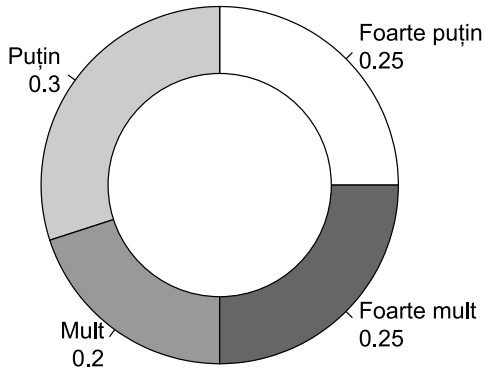
Spre deosebire de diagramele bară sau cea cu puncte, diagrama circulară

este însă evitată în publicațiile profesionale, în ciuda utilizării ei foarte largi în documentele uzuale. Toată lumea o folosește, însă puțini știu că de fapt diagramele circulare sunt extrem de neclare atunci când proporțiile categoriilor sunt aproximativ egale.

Există studii de percepție vizuală care arată foarte clar o ușurință în interpretarea diferențelor pe o diagramă bară și o dificultate perceptuală atunci când se afișează aceleași date pe o diagramă circulară. Acest lucru se întâmplă deoarece ochiul este nevoit să evalueze nu doar diferențele dintre categorii, dar și diferențele dintre ariile reprezentate de feliile diagramei, un lucru dificil mai ales dacă există multe categorii cu multe culori (culorile închise par mai mici, culorile deschise par mai largi).

O diagramă considerată mai puțin "nocivă" este cea inelară (în engleză *donut chart*), în care ochiul nu mai este forțat să perceapă arii largi ci mai degrabă un fel de bară îndoită, ca în figura 12.6.

Figura 12.6: Diagramă inelară



Există mai multe metode de a obține o astfel de diagramă, cu funcții dedicate în pachete speciale, însă efectul final este cel care contează și probabil cea mai ușoară metodă este de a utiliza același cod utilizat pentru figura 12.5 și se adăugă pur și simplu un cerc deasupra, umplut cu culoarea albă. Un astfel de cerc poate fi obținut foarte ușor adăugând următorul cod:

```
> par(new = TRUE)
> symbols(0, 0, 0.5, bg = "white", inches = FALSE)
```

Argumentul `new=TRUE` din comanda `par()` spune că un nou grafic va fi adăugat peste cel deja existent (altfel noul grafic îl va înlocui pe cel

dinainte), iar comanda `symbols()` este cea care adaugă un cerc în centrul diagramei (la coordonatele 0 pe orizontală și 0 pe verticală), care să acopere jumătate din totalul ariei (valoarea de 0.5 pe a treia poziție).

Ambele tipuri de diagrame circulare (inclusiv cea inelară) sunt la fel de dificile perceptual, în plus utilizează și mai multă cerneală decât o diagramă bară. Se transmite aceeași informație, însă utilizând mult mai multe elemente (barele sunt îndoite, culorile segmentelor sunt diferite etc.)

Dintre toate formele de reprezentare grafică a variabilelor categoriale, sunt de preferat diagramele bară orizontale, sau chiar diagrama cu puncte, pentru cei care vor să realizeze prezentări profesionale.

12.3 Descrierea variabilelor numerice

În cazul variabilelor numerice, există mai multe posibilități descriptive, însă printre acestea nu se mai numără tabelele de frecvențe și diagramele specifice variabilelor categoriale. Motivul este unul foarte bun, întrucât un tabel de frecvențe arată toate valorile posibile ale unei variabile, împreună cu frecvența de apariție a fiecărei valori.

Variabilele numerice pot avea extrem de multe valori (cele continue pot avea chiar un infinit de valori posibile) astfel încât a construi un tabel de frecvențe cu atât de multe linii este un nonsens. De aceea, tabelele de frecvențe sunt (în cele mai multe cazuri) specifice doar variabilelor categoriale. Există doar două situații în care pot fi construite tabele de frecvență și pe baza variabilelor numerice.

Prima situație este cea a variabilelor numerice discrete cu foarte puține valori (să spunem cu cel mult 10 sau 15 valori posibile), de exemplu numărul de copii dintr-o familie. În cvasi-majoritatea cazurilor, numărul de copii se situează undeva între 0 și 4, din ce în ce mai rar la un număr mai mare și extrem de rar peste 10 sau mai mult. Dacă se fixează o limită maximă la 15 copii, putem fi asigurați că fiecare familie se va regăsi într-unul dintre aceste numere, iar 15 linii sunt suficient de puține pentru a putea construi un tabel de frecvență al numărului de copii (chiar dacă această variabilă este numerică).

A doua situație este cea în care variabila metrică este grupată pe intervale (sau pe clase) de valori. Vârsta respondentului este un astfel de exemplu, care poate fi recodificată pe intervale (mai multe sau mai puține, mai înguste sau mai late, în funcție de întrebările de cercetare specifice).

Într-o situație ipotetică de cercetare, ar putea fi interesant de a vedea

diferențele dintre cele trei mari categorii de vârste: tineri, adulți și vâstnici. Pentru fiecare dintre cele trei categorii se poate specifica o limită minimă și una maximă: tineri între 16 și 40 de ani, adulți între 41 și 65 de ani și vârstnici între 66 și 90 ani și peste. Fiecare individ se regăsește într-unul dintre aceste intervale, așadar fiecare individ aparține uneia dintre aceste categorii determinate de intervalele de vârstă. Prin recodificare se obțin doar trei categorii, așadar se poate construi un tabel de frecvențe, pe modelul tabelului 12.2.

În felul acesta, o variabilă numerică poate fi transformată (prin recodificarea valorilor) într-o variabilă categorială care poate fi utilizată mai departe pentru a fi descrisă cu ajutorul metodelor arătate la secțiunea precedentă. Codul de mai jos arată generarea unei variabile cu vârste a 150 de respondenți între 16 și 90 ani, plus recodificarea acesteia în variabila `varec` pe cele trei intervale.

```
> set.seed(12345)
> varsta <- sample(16:90, 150, replace = TRUE)
> varec <- rep(2, length(varsta)) # initializare
> varec[varsta < 41] <- 1
> varec[varsta > 65] <- 3
```

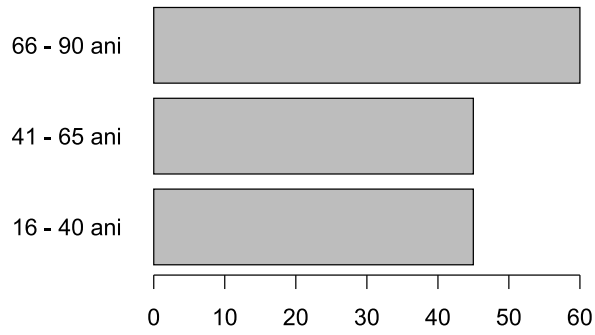
La fel ca în cazul variabilelor categoriale, pe baza acestui tabel de frecvențe se poate genera și o reprezentare grafică, așa cum este ilustrată în figura 12.7.

În general, fiecare dintre secțiunile următoare sunt caracterizate atât de măsuri numerice cât și de reprezentări grafice, așa încât nu mai este necesară o secțiune separată pentru reprezentări, acestea fiind prezentate printre măsurile numerice asociate în cadrul fiecărei secțiuni în parte.

Tabelul 12.2: Tabel de frecvență pentru variabila `varec`

		F.A.
1.	16 - 40 ani	45
2.	41 - 65 ani	45
3.	66 - 90 ani	60
	Total	150

Figura 12.7: Diagramă bară pentru variabila `varec`



12.3.1 Măsuri ale tendinței centrale

În afară de cazurile rare în care o variabilă numerică poate fi descrisă cu ajutorul tabelelor de frecvență, probabil cele mai comune măsuri descriptive care se pot realiza pe aceste tipuri de variabile sunt cele care se referă la centrul distribuției.

Ideea conceptului de centralitate se referă la necesitatea statistică de a descrie locul unde se află cele mai multe elemente din variabila respectivă. În orice populație există elemente tipice și elemente atipice, ultimele fiind considerate raritați. Foarte puțini oameni sunt extrem de bogați (elemente atipice), în timp ce marea masă a populației au venituri relativ asemănătoare (elemente tipice). Așadar, rolul unei măsuri descriptive pentru tendința centrală este să arate, printr-un singur număr, care este valoarea tipică pentru marea majoritate a elementelor (adică locul în care sunt adunate cele mai multe dintre aceste elemente).

Cea mai simplă modalitate de a determina o valoare tipică este de a urmări care dintre valorile variabilei apare cel mai des (pe ideea că dacă o anumită valoare este specifică multor elemente atunci ea reprezintă marea majoritate a elementelor), ceea ce în statistică se numește "mod".

Modul este așadar una dintre măsurile tendinței centrale, determinată de valoarea variabilei cu frecvența cea mai mare de apariție. În R, a determina modul este o simplă chestiune de a construi un tabel de frecvențe cu toate valorile existente (nu toate valorile posibile, doar cele existente) apoi selectată acea valoare asociată cu maximul frecvenței.

```
> tab.fr <- table(varsta)
> max(tab.fr)
```

```
[1] 5
> names(tab.fr)[tab.fr == max(tab.fr)]
[1] "54" "62" "70" "76"
```

Maximul de apariție a unei vârste este de cinci ori, iar patru dintre vârstele existente în variabila `varsta` prezintă această frecvență: 54 ani, 62 ani, 70 ani și 76 ani, toate acestea apar cu aceeași frecvență maximă de 5 ori.

Aceasta este ceea ce se numește o variabilă multi-modală, cu mai multe moduri. Există cazuri în care o variabilă poate avea un singur mod (variabilă uni-modală), sau cu două moduri (variabilă bi-modală) sau, cum este cazul aici, cu mai multe moduri. Din acest motiv, modul este o doar primă încercare de a determina centrul distribuției, însă nu este foarte eficient dacă mai multe asemenea valori există în setul de date.

O altă măsură a tendinței centrale, de altfel cea mai utilizată dintre toate, este chiar media variabilei, care se calculează după formula clasică a sumei valorilor supra numărul de valori:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{12.1}$$

În R, calcularea mediei se face trivial cu:

```
> mean(varsta)
[1] 55.45333
```

Media este un bun estimator al tendinței centrale, câtă vreme toate valorile sunt mai mult sau mai puțin asemănătoare, pe o distribuție normală (cele mai multe valori în mijlocul distribuției, din ce în ce mai puține la stânga și la dreapta, la o distanță egală față de mijloc).

Există situații în care unele valori sunt extrem de diferite de celelalte, acestea fiind denumite "valori extreme" (în engleză *outliers*). Spre exemplu, veniturile a 10 persoane pot fi:

```
> (venit <- c(1750, 800, 1200, 1350, 1450, 2000, 1400, 1050,
1300, 1500))
 [1] 1750  800 1200 1350 1450 2000 1400 1050 1300 1500
> mean(venit)
[1] 1380
```

Media veniturilor celor 10 persoane este de 1380 lei, iar structura acestora este relativ asemănătoare, se încadrează între o limită minimă de 800 lei

și una maximă de 2000 lei, foarte des întâlnite la marea majoritate a oamenilor din România. Dificultățile de interpretare apar atunci când între valorile observate apar și unele extreme (ori foarte mici, ori foarte mari în comparație cu celelalte). Să spunem că la seria de date existentă se adaugă încă o persoană cu un venit foarte mare:

```
> (venit <- c(venit, 10000))
 [1] 1750  800 1200 1350 1450 2000 1400 1050 1300 1500 10000
> mean(venit)
[1] 2163.636
```

La cele 10 venituri observate anterior a fost adăugat încă unul, cu o valoare de 10000 lei lunar, iar media tuturor celor 11 venituri s-a modificat la valoarea de 2163.64 lei (cu rotunjire la două zecimale). Această valoare a venitului mediu, însă, nu mai poate fi considerată reprezentativă pentru primele zece valori, deoarece toate se încadrează până la un maxim de 2000 lei, iar media este mai mare chiar decât această valoare maximă. De cealaltă parte, această valoare a venitului mediu nu reprezintă nici ultimul venit cu valoarea de 10000 lei, așadar atunci când există valori extreme în date, media nu mai poate fi considerată o măsură corectă a tendinței centrale.

Se poate observa cu certitudine că media este influențată foarte mult de valorile extreme, acestea acționând ca niște poli de atracție: media se deplasează în direcția în care se află valoarea extremă, de cele mai multe ori în afara zonei unde se află marea majoritate a elementelor.

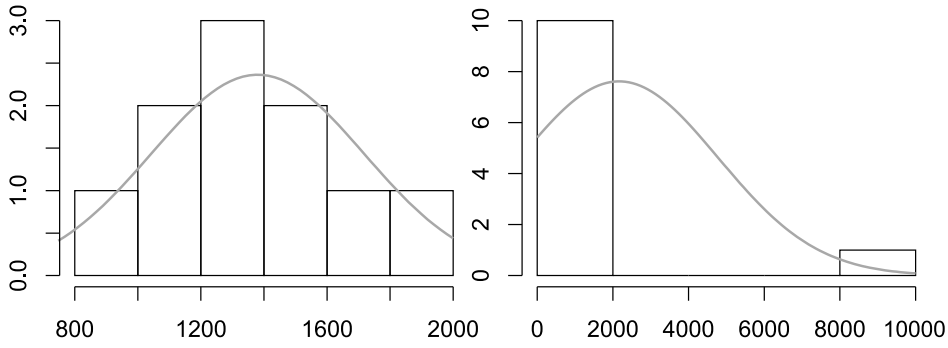
Toate aceste particularități au o strânsă legătură cu forma distribuției pe care o prezintă variabila de interes: dacă distribuția este una normală, media stă în centrul distribuției (deci reprezintă marea majoritate a valorilor observate), iar dacă distribuția valorilor nu este normală (cu alungire fie spre stânga fie spre dreapta), atunci există valori extreme între valorile observate. Forma distribuției este așadar o primă formă de identificare a existenței valorilor extreme.

Figura 12.8 reprezintă un exemplu tipic de distorsionare a formei distribuției atunci când sunt prezente valori extreme. În partea stângă este o distribuție normală, simetrică a variabilei înainte de adăugarea valorii de 10000 lei, iar în partea dreaptă este o distribuție alungită la dreapta, după introducerea acelei valori.

Forma alungită a distribuției este un prim indiciu de existență a valorilor extreme în setul de date. La secțiunea 12.3.3 vor fi prezentate și alte modalități de identificare a acestor valori, însă pentru moment inspecția vizuală este importantă. Se poate observa că marea majoritate a valorilor

Figura 12.8: Distribuția variabilei `venit`

a. distribuție normală b. distribuție alungită



sunt grupate în intervalul 0 - 2000 lei, iar media egală cu 2163.64 nu reprezintă această majoritate.

Din acest motiv, este introdusă o a treia măsură a tendinței centrale numită "mediană". Mediana este valoarea din mijlocul unei distribuții de valori ordonate crescător. În cazul variabilei `venit`, înainte de introducerea valorii extreme, ordonarea arată cam așa:

```
> sort(venit[-11])
 [1]  800 1050 1200 1300 1350 1400 1450 1500 1750 2000
> median(venit[-11])
[1] 1375
```

Codul de mai sus afișează valorile ordonate crescător, mai puțin ultima (a 11a) introdusă anterior. Sunt 10 valori, iar mediana se află la mijlocul distanței dintre a cincea și a șasea valoare, adică între 1350 și 1400. Următoarea comandă afișează valoarea medianei, egală cu 1375 lei. Pentru verificarea vizuală a procesului este imperativă ordonarea valorilor, însă comanda `median()` face acest lucru în mod predefinit.

Următoarele linii de cod afișează toate valorile, ordonate de la cea mai mică (800) până la cea mai extremă valoare (10000). În total, acum sunt 11 valori iar mediana este cea din mijloc, adică a șasea:

```
> sort(venit)
 [1]  800 1050 1200 1300 1350 1400 1450 1500 1750 2000 10000
> median(venit)
[1] 1400
```

La modul general, pentru o variabilă cu n valori, mediana se regăsește la poziția numărul $\frac{n+1}{2}$. Dacă n este un număr impar, atunci mediana este chiar valoarea din mijlocul seriei de date, iar dacă n este un număr impar atunci mediana se calculează la mijlocul distanței dintre valorile din centrul seriei de date.

Utilizarea medianei are multiple avantaje asupra utilizării mediei, în special atunci când există valori extreme în seria de date. Pentru cele 10 valori inițiale, mediana era egală cu 1375 lei, iar după introducerea valorii extreme mediana a devenit egală cu 1400 lei. Ambele valori ale medianei se află între minima de 800 și maxima de 2000 lei care definesc marea majoritate a elementelor, deci mediana reprezintă o foarte bună estimare a centrului distribuției.

Mai mult, dacă în loc de zece mii lei ar fi fost introdusă valoarea de un milion de lei, media s-ar fi deplasat și mai mult în partea dreaptă (92163.64 lei), însă mediana ar fi rămas tot la valoarea de 1400 lei. Indiferent de magnitudinea valorilor extreme, mediana rămâne tot în centrul distribuției, așadar este imună la acțiunea acestor valori.

În cazul unei distribuții nomale, simetrice, toate cele trei măsuri (media, mediana și modul) sunt egale și se află în exact același loc pe centrul distribuției, iar dacă distribuția este alungită media nu mai poate fi o bună estimare a tendinței centrale.

Înainte de introducerea valorii extreme media era egală cu 1380 lei, iar mediana cu 1375 lei. După introducerea valorii extreme, media a devenit egală cu 2163.64 lei însă mediana a rezistat în centru, la valoarea de 1400 lei.

Ambele valori ale medianei (1375, respectiv 1400 lei) se află în aceeași zonă cu valoarea mediei pentru o distribuție normală (1380 lei) așadar se demonstrează faptul că mediana este cea mai bună măsură pentru determinarea tendinței centrale, indiferent dacă seria de date are sau nu valori extreme (și poate mai important, indiferent de magnitudinea acestor valori extreme).

12.3.2 Măsuri ale variației

Măsurile tendinței centrale ajută la descrierea primară a unei populații de studiu, însă nu oferă o imagine completă a acesteia. Sunt multe situații în care mai multe serii de date au aceleași valori ale mediei și medianei, însă acestea poti fi extrem de diferite.

Un exemplu ar putea fi constituit din cele trei serii de date din tabelul de mai jos:

S1:	1	2	3	4	5	6	7	8
S2:	2	2	4	4	5	5	7	7
S3:	4	4	4	4	5	5	5	5

Atât media, cât și mediana pentru toate cele trei serii de date au exact aceeași valoare, anume 4.5 (doar valorile modale diferă, însă toate seriile au mod multiplu). Judecând doar din perspectiva tendinței centrale, fără inspecția tuturor seriilor de date s-ar putea constata că sunt asemănătoare.

```
> s1 <- 1:8
> s2 <- rep(c(2,4,5,7), each = 2)
> s3 <- rep(4:5, each = 4)
> c(mean(s1), median(s1))
[1] 4.5 4.5
> c(mean(s2), median(s2))
[1] 4.5 4.5
> c(mean(s3), median(s3))
[1] 4.5 4.5
```

În mod evident însă, ele diferă foarte mult sub aspectul varietății datelor: prima serie este extrem de variată cu opt numere diferite între ele, a doua serie este mai puțin variată cu doar patru numere unice iar a treia este cea mai puțin variată cu doar două numere unice. Descrierea acestor serii de date trebuie completată cu alte măsuri care să completeze imaginea formată în cadrul tendinței centrale.

Aceste măsuri adiționale se numesc măsuri ale variației și oferă o altă descriere asupra datelor, din perspectiva diferenței (sau invers, asemănării) dintre elemente. În limbaj statistic, diferențele sunt acumulate în conceptul de "eterogenitate" iar asemănările în conceptul opus de "omogenitate".

O variabilă este denumită eterogenă dacă elementele care o alcătuiesc sunt foarte diferite între ele și este denumită omogenă dacă elementele care o alcătuiesc sunt asemănătoare între ele. Sunt concepte opuse, întrucât o variabilă extrem de eterogenă este în același timp o variabilă foarte puțin omogenă și invers.

În cazul celor trei serii de date, este evident faptul că prima serie este cea mai eterogenă dintre toate iar cea de a treia este cea mai omogenă. Această diferențiere însă poate fi realizată și cu ajutorul unor măsuri numerice care să ateste cât de eterogenă și cât de omogenă este o variabilă.

Este nevoie așadar de o măsură care, dacă are o valoare mare să se poată spune că variază foarte mult (este eterogenă), iar dacă are o valoare mică atunci să se poată spune că o variabilă variază foarte puțin (este omogenă).

Prima dintre aceste măsuri ale variației se numește amplitudine (în engleză *range*), calculată ca diferența dintre valoarea maximă și cea minimă dintr-o serie de date, pe principiul că dacă o variabilă este omogenă atunci va avea o distanță mică între minim și maxim, iar dacă variază mult va avea o plajă largă de valori (așadar implicit o distanță largă între minim și maxim).

```
> range(s1)
[1] 1 8
> range(s2)
[1] 2 7
> range(s3)
[1] 4 5
```

Judecând prin intermediul acestei măsuri, prima serie de date are o amplitudine egală cu $8-1=7$, a doua o amplitudine egală cu $7-2=5$ iar a treia o amplitudine egală cu $5-4=1$. Aceasta este o primă modalitate de inspecție a variației unor date, însă (similar cu modul în măsurile tendinței centrale) are o aplicabilitate foarte redusă întrucât este puternic influențată de existența valorilor extreme. Spre exemplu, dacă ultimul număr din a treia serie de date ar fi fost egal cu 100, am fi dedus că amplitudinea ar fi egală cu 96 cu concluzia unei variații foarte mari, când în fapt a treia serie de date este cea mai omogenă dintre toate cele trei serii.

O altă posibilitate de a măsura cantitatea de variație dintr-o variabilă este calcularea distanței dintre fiecare observație până la un punct de referință, acela fiind media variabilei. Ideea este relativ ușor de înțeles: dacă aceste distanțe sunt mari, atunci datele variază mult, iar dacă distanțele sunt mici atunci datele variază puțin.

Această distanță între fiecare observație și punctul de referință (media) se face cu formula $x_i - \overline{x}$, unde $i = 1, 2, \ldots, n$.

```
> s1 - mean(s1)
[1] -3.5 -2.5 -1.5 -0.5  0.5  1.5  2.5  3.5
```

În seria de date `s1` fiecare valoare are o distanță față de medie, așa încât toate aceste distanțe trebuie utilizate pentru a obține o singură măsură

numerică. O opțiune ar fi calcularea mediei tuturor distanțelor, pe aceeași idee: dacă media distanțelor este un număr mare, valorile sunt foate variate, iar dacă această medie a distanțelor este un număr mic, valorile sunt omogene.

Dificultatea cu acest raționament nu constă în complexitatea de calcul ci în însăși natura de calcul a oricărei medii: este suma valorilor împărțită la numărul de valori. Dat fiind însă că unele distanțe sunt negative (valorile sunt mai mici decât media variabilei) iar altele sunt pozitive (valorile sunt mai mari decât media variabilei), ele se anulează reciproc iar suma acestor distanțe este egală cu zero, implicit media distanțelor este egală cu sero.

Pentru a anula efectul diferențelor de semn, o metodă foarte utilizată în matematică este ridicarea la pătrat, iar formula completă a mediei distanțelor este:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1} \tag{12.2}$$

Formula 12.2 arată o sumă de distanțe ridicate la pătrat, împărțite la numărul valorilor (n) minus 1. Întrucât distanțele față de medie sunt ridicate la pătrat, formula se notează cu s^2 și se numește varianță (în engleză *variance*), care este cea de a doua măsură a variației.

```
> var(s1)
[1] 6
> var(s2)
[1] 3.714286
> var(s3)
[1] 0.2857143
```

După cum se poate observa, varianța respectă principiul enunțat mai devreme (dacă este un număr mare, variabila este eterogenă, iar dacă este un număr mic, variabila este omogenă), însă unitatea de măsură a acestei valori este ridicată la pătrat. Pentru a o aduce înapoi la unitatea de măsură originală a variabilei, se definește cea de a treia măsură a variației. Efectul ridicării la pătrat se anulează prin extragerea radicalului, așa încât se introduce abaterea standard s (în engleză *standard deviation*) care se calculează ca radical din varianță:

$$s = \sqrt{s^2} \tag{12.3}$$

Abaterea standard respectă și ea principiul menționat, cu avantajul adițional că se exprimă în aceeași unitate de măsură ca a variabilei pe baza căreia a fost calculată.

```
> sd(s1)
[1] 2.44949
> sd(s2)
[1] 1.927248
> sd(s3)
[1] 0.5345225
```

Cea mai mare abatere standard este cea pentru seria de date $s1$ (care are variația cea mai mare a datelor) însă valoarea de 2.45 este mare doar în comparație cu media de 4.5. Dacă media variabilei ar fi egală cu 100, o abatere standard egală cu 2.45 ar indica în fapt o variabilă foarte omogenă.

În concluzie, valoarea abaterii standard se judecă în comparație cu valoarea mediei variabilei: dacă reprezintă o proporție mică din medie, variația este mică iar dacă reprezintă o proporție mare din medie, variația este mare.

12.3.3 Măsuri ale poziționării

Cea de a treia serie de măsuri utile pentru descrierea variabilelor se referă la poziționarea elementelor unele față de celelalte. Uneori este util să se identifice în ce zonă se poziționează anumite valori: în prima parte a datelor, sau în a doua? Între cele mai mici, sau între cele mai mari?

Cu specific la cercetarea socială, în ce categorie face parte o persoană, sub aspectul venitului de exemplu? Între cei mai săraci, sau între cei mai bogați, sau undeva pe la mijloc? Astfel de întrebări au sens, de aceea se procedează (ca și în cazul medianei la descrierea tendinței centrale) la ordonarea tuturor valorilor în sens crescător, după care se stabilește o împărțire a acestora în zone distincte cu ajutorul unor jaloane care se numesc generic "quantile".

Spre exemplu, mediana este o astfel de quantilă întrucât împarte observațiile (valorile variabilei) în două jumătăți egale ca număr de valori. Dacă se dorește împărțirea în trei părți egale, jaloanele utilizate se numesc "terțile". Dacă se dorește împărțirea în patru părți egale jaloanele se numesc "quartile", pentru 10 părți egale se numesc "decile", pentru 100 părți egale se numesc "percentile" etc. Toate aceste jaloane (terțile, quartile, decile etc.) fac parte dintr-o familie generică numită "quantile".

În exemplul anterior cu ordinea veniturilor, o aplicație foarte utilă este împărțirea acestora pe decile: se adună veniturile celor mai săraci 10% dintre locuitori, se adună și veniturile celor mai bogați 10% dintre locuitori, apoi se face raportul acestor două venituri cumulate: dacă acest raport este unul extrem de mare (veniturile bogaților de foarte multe ori mai mare decât veniturile săracilor) atunci societatea poate fi descrisă ca inegalitară, iar dacă acest raport este doar rezonabil mare (veniturile bogaților mai mari, însă nu enorme față de ale săracilor) atunci societatea poate fi descrisă ca egalitară.

La împărțirea observațiilor pe quartile, sunt necesare trei jaloane: una pentru împărțirea în două jumătăți, apoi pentru fiecare jumatate câte încă un jalon pentru împărțirea fiecăreia în alte două jumătăți egale. Aceste jaloane sunt denumite Q_1, Q_2 și Q_3 (unde Q_2 este chiar mediana variabilei). Împreună cu cea mai mică și cea mai mare valoare din variabilă (minima și maxima) formează cele cinci valori care, alături de medie, alcătuiesc cea mai cunoscută și mai utilizată formă de sumarizare numerică a datelor.

```
> summary(venit)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    800    1250    1400    2164    1625   10000

> summary(venit[-11])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    800    1225    1375    1380    1488    2000
```

În exemplul venitului, valoarea minimă este 800, iar în absența valorii de 10000, cea maximă este 2000. Q_1 este egală cu1200, Q_2 (mediana) este egală cu 1375 iar Q_3 este egală cu 1500. Cele trei quartile sunt extrem de informative și au multiple utilizări.

O primă utilizare a quartilelor este de a identifica tendința centrală și pentru a determina dacă distribuția datelor este una normală sau alungită. La secțiunea 12.3.1 se arată că în cazul unei distribuții normale media este în același punct cu mediana, iar dacă media este foarte departe de mediană atunci distribuția prezintă o alungire.

În cazul variabilei `venit`, în absența valorii de 10000 diferența dintre medie și mediană este una foarte mică (1380 față de 1375) așa încât se poate concluziona că distribuția valorilor este foarte apropiată de normalitate (cu o doar foarte, foarte mică deplasare spre dreapta). În situația când valoarea de 10000 este introdusă în setul de valori, diferența dintre medie și mediană devine foarte mare (2164 față de 1400) ceea ce sugerează o puternică deplasare spre dreapta a distribuției și

existența unor valori extreme. În general, atunci când se identifică o distribuție deplasată de la normalitate, este un prim indiciu pentru existența unor posibile valori extreme în cadrul variabilei.

Dacă prima utilizare a quartilelor se referea la măsurile tendinței centrale, cea de a doua utilizare a acestora se referă la măsurile variației. Ca și în cazul mediei (care este sensibilă la acțiunea valorilor extreme), s-a arătat și în cazul amplitudinii că este foarte sensibilă la aceleași valori extreme. Când valoarea maximă este extrem de mare (sau invers, valoarea minimă extrem de mică) față de restul datelor, amplitudinea nu mai poate fi considerată o bună măsură pentru descrierea variației datelor. Cu ajutorul quartilelor însă poate fi construită o imagine mai bună indiferent de existența valorilor extreme.

Întrucât valorile extreme apar la capetele din stânga sau dreapta ale valorilor ordonate crescător, atunci când se ignoră primul sfert și ultimul sfert dintre valori putem fi siguri că valorile rămase în mijloc nu mai conțin niciun fel de valori extreme. Așadar, între Q_1 și Q_3 se află jumătate dintre valorile variabilei iar distanța dintre acestea se numește abatere interquartilă și oferă o imagine similară cu amplitudinea: $AIQ = Q_3 - Q_1$.

Dacă abaterea interquartilă (AIQ) este un număr mic, datele variază puțin (sunt omogene), iar dacă este un număr mare, datele variază mult (sunt eterogene). Numărul respectiv este mare sau mic, în comparație cu mediana variabilei (la fel cum abaterea standard este mare sau mică în comparație cu media).

O a treia utilizare posibilă a quartilelor este aceea de identificare a valorilor extreme. Forma alungită a distribuției valorilor este doar un indiciu al existenței acestora, însă cu ajutorul quartilelor pot fi identificate cu certitudine care dintre valori sunt extreme.

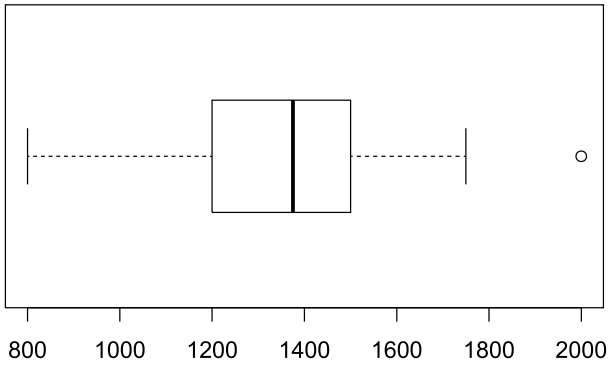
În acest scop, este utilizată o altă diagramă numită *boxplot*, aceasta mai fiind cunoscută (în limba engleză) și sub numele de diagramă *box and whiskers*.

```
> boxplot(venit[-11], horizontal = TRUE)
```

În figura 12.9 este prezentată o astfel de diagramă, în care poziția cutiei pe axa orizontală este determinată de quartilele Q_1 și Q_3. În interiorul acestei cutii este o bară verticală, a cărei poziție este determinată de cea de a doua quartilă Q_2.

Valorile extreme sunt cele din afara liniilor punctate (o singură astfel de valoare se află în partea dreaptă). Oricine ar fi intuit că o valoare egală cu

Figura 12.9: Diagramă boxplot



10000 este una extremă, însă foarte puțin evident a fost faptul că valoarea de 2000 este una extremă, în comparație cu restul datelor.

O valoare extremă este o valoare vare se află în afara unor praguri, definite de 1.5 abateri interquartile în stânga quartilei Q_1 și tot de 1.5 abateri interquartile în dreapta quartilei Q_3. În cazul variabilei `venit`, cea de a treia quartilă are o valoare de 1488 iar abaterea interquartilă are o valoare de $1488 - 1225 = 263$, ceea ce înseamnă că pragul din dreapta este de $Q_3 + 1.5 \cdot AIQ = 1488 + 1.5 \cdot 263 = 1882.5$ iar orice valoare peste acest prag se definește ca fiind extremă. Cum valoarea de 2000 este mai mare decât acest prag, este o valoare extremă.

Desigur, valoarea de 10000 este cu siguranță extremă, cu diferența între valori extreme moderate (cum este 2000) și valori extreme puternice (cum este 10000).

Capitolul 13

Analiza econometrică în R. Modelul linear de regresie

În accepțiunea generală, econometria reprezintă aplicarea statisticii matematice în analiza datelor economice, în scopul testării teoriilor și/sau a realizării de prognoze.[1]

Pornind de la ideea lui Frish, 1933, pp. 1-2, conform căreia: "Experiența a arătat că fiecare dintre aceste trei puncte de vedere, adică statistica, teoria economică și matematica sunt condiții necesare, dar nu reprezintă în sine o condiție suficientă pentru o înțelegere reală a relațiilor cantitative din viața economică modernă. Doar unificarea tuturor celor trei dă putere de analiză. Și econometria constituie tocmai această unificare".

13.1 Regresia unifactorială

13.1.1 Ecuația de regresie

De exemplu, din teoria economică se cunoaște faptul că veniturile (X) și consumul familiilor (Y) sunt două variabile care nu evoluează independent una de alta, mai exact, consumul depinde în mare măsură de nivelul veniturilor. Se acceptă, pentru început, ipoteza că la nivelul întregii populații consumul depinde linear de venit. Se poate scrie consumul ca funcție de venit astfel:

$$Y = f(X, e) \tag{13.1}$$

unde `e` simbolizează ceilalți factori care contribuie la formarea

[1] Pentru detalierea analizei econometrice, a se vedea Jula (Jula)

comportamentului de consum. Fie `M(Y/X)` valoarea anticipată a consumului atunci când venitul atinge valoarea X:

$$M(Y/X) = a_0 + a_1 X \tag{13.2}$$

Daca se interpretează variabila `M(Y/X)` ca fiind valoarea medie a consumului calculată pentru toate familiile care înregistrează un anumit nivel X al venitului, atunci:

- a_1 - înclinația marginală spre consum a familiilor din populația analizată.
- a_0 - nivelul consumului atunci când venitul familiei este zero.

Dar consumul efectiv înregistrat (Y) nu este întotdeauna egal cu `M(Y/X)` – valoarea anticipată a consumului pentru un venit dat X. Consumul efectiv al familiilor ar trebui să fie scris:

$$Y = M(Y/X) + e \tag{13.3}$$

ecuația de regresie a lui Y în funcție de X, sau

$$Y = a_0 + a_1 X + e \tag{13.4}$$

Modelarea consumului (Y) în funcție de venit (X) înseamnă stabilirea unor valori $\widehat{a_0}$ și $\widehat{a_1}$ pentru parametrii a_0 și a_1 astfel încât înlocuind aceste valori într-o ecuație de forma $\widehat{Y} = \widehat{a_0} + \widehat{a_1} X$, cu X_t cunoscut, să se obțină valori $\widehat{Y_t}$ cât mai apropiate de cele înregistrate în eșantionul selectat (Y_t). Se definește diferența dintre consumul estimat $\widehat{Y_t}$ și cel înregistrat efectiv Y_t, numită *variabila reziduală*:

$$u_t = Y_t - \widehat{Y_t}, t = 1, 2, ..., n \tag{13.5}$$

13.1.2 Metoda celor mai mici pătrate

Criteriul aplicat în cazul metodei celor mai mici pătrate este următorul: dreapta care asigură cea mai bună ajustare a punctelor empirice (dreapta de regresie) este aceea pentru care se minimizează suma pătratelor abaterilor dintre punctele de pe grafic și punctele care au aceiași abscisă pe dreapta de regresie, abaterile fiind măsurate vertical.
Analitic, se notează $F(\widehat{a_0}, \widehat{a_1})$ suma pătratelor abaterilor `u` dintre valorile

înregistrate ale variabilei Y și valorile calculate $\widehat{Y}$ (adică $F(\widehat{a_0}, \widehat{a_1})$ măsoară suma pătratelor valorilor variabilei reziduale):

$$F(\widehat{a_0}, \widehat{a_1})) = \sum_{t=1}^{n} u_t^2 = \sum_{t=1}^{n} (Y_t - \widehat{Y_t})^2 = \sum_{t=1}^{n} (Y_t - \widehat{a_0} - \widehat{a_1} Y_t)^2 \qquad (13.6)$$

Metoda celor mai mici pătrate constă în determinarea, pentru un set dat de observații, a acelor valori $\widehat{a_0}$ și $\widehat{a_1}$ care minimizează funcția $F(\widehat{a_0}, \widehat{a_1})$.

Se obține așa-numitul sistem de ecuații normale:

$$\begin{cases} \sum_{t=1}^{n} Y_t = n\widehat{a_0} + \widehat{a_1} \sum_{t=1}^{n} X_t \\ \sum_{t=1}^{n} X_t Y_t = \widehat{a_0} \sum_{t=1}^{n} X_t + \widehat{a_1} \sum_{t=1}^{n} X_t^2 \end{cases} \qquad (13.7)$$

Exemplu
Dintr-o cercetare statistică se presupune că au fost identificate 5 seturi a câte 25 de valori (de exemplu, venituri, investiții, cheltuieli cu cercetarea etc.). Fie *X*1, *X*2, *X*3, *X*4 și *Y* seriile respective. Trebuie să se indentifice dacă acești factori influențează veniturile, în ce măsură și ce grad încredere se poate avea în informațiile obținute.
Prima etapă pentru modelarea econometrică în R presupune încărcarea datelor:

```
> datele_mele <- data.frame(
X1 = c(3, 2, 0.8, 2.5, 2, 1.4, 2.5, 2.5, 3, 1.4, 1, 1.2, 1.6, 1.8,
       1, 2.8, 3.5, 2.6, 2.4, 3.4, 1.6, 1.9, 3.5, 1.6, 3),
X2 = c(1.3, 2.8, 1.5, 0.2, 1.8, 4, 1.8, 2, 0.5, 2.8, 3.2, 2.5, 1.3,
       2.2, 3.5, 1.1, 0, 0.2, 2, 1.2, 3, 3, 0.6, 3.2, 0.3),
X3 = c(2.1, 2.4, 1.1, 1.4, 1.9, 2.7, 2.2, 2.3, 1.7, 2.1, 2.1, 1.8,
     1.5, 2, 2.3, 2, 1.7, 1.4, 2.2, 2.3, 2.3, 2.5, 2.1, 2.4, 1.6),
X4 = c(2.4, 1.6, 0.6, 2, 1.6, 1.1, 2, 2, 2.4, 1.1, 0.8, 1, 1.3, 1.4,
       0.8, 2.2, 2.8, 2.1, 1.9, 2.7, 1.3, 1.5, 2.8, 1.3, 2.4),
Y  = c(2, 0.5, 1.5, 3, 1, 0, 2.1, 1.8, 3, 0.7, 0.5, 1, 1.4, 1.2,
       0.8, 2.3, 3.5, 3.8, 1.8, 2.6, 0.8, 1.2, 4.2, 0.8, 2.5))
```

Pentru a vizualiza structura creată, mai exact primele 6 înregistrări, se folosește comanda:

```
> head(datele_mele)
   X1  X2  X3  X4   Y
1 3.0 1.3 2.1 2.4 2.0
```

```
2 2.0 2.8 2.4 1.6 0.5
3 0.8 1.5 1.1 0.6 1.5
4 2.5 0.2 1.4 2.0 3.0
5 2.0 1.8 1.9 1.6 1.0
6 1.4 4.0 2.7 1.1 0.0
```

Pentru modelul unifactorial, trebuie să se identifice dacă una dintre variabile, de exemplu $X1$, influențează starea și dinamica variabilei Y. Modelul linear unifactorial este

$$Y_t = a_0 + a_1 X_{1t} + e_t \tag{13.8}$$

În R, rezolvarea modelului prin aplicarea metodei celor mai mici pătrate presupune următoarele comenzi:

```
> regresie <- lm(Y ~ X1, data = datele_mele)
```

Rezultatele estimării modelului se pot afișa cu ajutorul funcției `summary()`:

```
> summary(regresie)

Call:
lm(formula = Y ~ X1, data = datele_mele)

Residuals:
    Min      1Q  Median      3Q     Max
-1.0801 -0.3303 -0.1797  0.2954  1.5452

Coefficients:
            Estimate Std. Error t value   Pr(>|t|)
(Intercept)  -0.6690     0.3747  -1.785   0.0874 .
X1            1.1245     0.1626   6.914   4.76e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6518 on 23 degrees of freedom
Multiple R-squared:  0.6752,Adjusted R-squared:  0.661
F-statistic: 47.81 on 1 and 23 DF,  p-value: 4.761e-07
```

Rezultatele obținute sugerează următoarea relație lineară dintre variabila $X1$ și Y:

$$\widehat{Y}_t = -0.669 + 1.1245 * X_{1t} \tag{13.9}$$

Coeficientul de determinare are valoarea $R^2 = 0.6752$, iar coeficienții modelului sunt semnificativ diferiți de zero, la un prag de 10% constanta din model (concret, $100 - 8.74\%$) și la un prag mai mare de 99.99% coeficientul variabilei $X1$.

Pentru a accesa și alte informații despre rezultatele regresiei, se pot apela o serie de comenzi:

```
> output <- summary(regresie)
> SSR <- deviance(regresie)
> LL <- logLik(regresie)
> Grade_de_libertate <- regresie$df
> Yest <- regresie$fitted.values
> Coef <- regresie$coefficients
> Resid <- regresie$residuals
> s <- output$sigma
> RSquared <- output$r.squared
> CovMatrix <- s^2 * output$cov
```

unde

- SSR - suma pătratelor reziduurilor
- LL - logaritm din funcția de verosimilitate
- Yest - vectorul valorilor estimate
- Resid - vectorul reziduurilor
- s - abaterea standard estimată a erorilor
- CovMatrix - matricea varianță-covarianță a coeficienților (apelabilă și utilizând `vcov()`)

13.2 Regresia multifactorială

13.2.1 Regresia bifactorială

Regresia bifactorială (sau bivariată) presupune analiza evoluției unei varibile Y în raport cu doi factori de influență. Această legătură se presupune că este sugerată de o anumită teorie economică (de exemplu, în marketing se sugerează o legătură între nivelul vânzărilor și investiția în reclamă, respectiv preț).

Un exemplu de model bifactorial este:

$$Y_t = a_0 + a_1 X_{1t} + a_2 X_{2t} + e_t \tag{13.10}$$

Valoare estimată $\widehat{Y}$ va fi:

$$\widehat{Y_t} = \widehat{a_0} + \widehat{a_1} X_{1t} + \widehat{a_2} X_{2t} \tag{13.11}$$

Dacă se notează cu u_t diferența dintre valoarea înregistrată și valoarea estimată a variabilei Y, atunci:

$$Y_t = \widehat{Y} + u_t \tag{13.12}$$

La fel ca în modelul unifactorial, se definește funcția de pierdere

$$F(\widehat{a_0}, \widehat{a_1}, \widehat{a_2}) = u'u = \sum_{t=1}^{n} u_t^2 = \sum_{t=1}^{n} (Y_t - \widehat{a_0} - \widehat{a_1} X_{1t} - \widehat{a_2} X_{2t})^2 \tag{13.13}$$

care se minimizează în raport cu estimatorii $\widehat{a_0}, \widehat{a_1}, \widehat{a_2}$.

Exemplu

Modelul bi-factorial presupune ca 2 factori explicativi, X_1 și X_2, să influențeze o variabilă explicată, Y.
În R se folosesc următoarele comenzi:

```
> regresie2 <- lm(Y ~ X1 + X2, data = datele_mele)
> summary(regresie2)
```

Programul returnează următoarele rezultate:

```
Call:
lm(formula = Y ~ X1 + X2, data = datele_mele)

Residuals:
     Min       1Q   Median       3Q      Max
-0.71506 -0.17149  0.01414  0.12197  1.05774

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.9836     0.5379   3.688  0.00129 **
X1            0.4405     0.1634   2.696  0.01318 *
X2           -0.6387     0.1150  -5.553  1.4e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4301 on 22 degrees of freedom
Multiple R-squared:  0.8647,Adjusted R-squared:  0.8524
F-statistic: 70.32 on 2 and 22 DF,  p-value: 2.774e-10
```

Conform rezultatelor, relația dintre variabilele $X1$, $X2$ și Y are forma:

$$\widehat{Y_t} = 1.9836 + 0.4405 * X_{1t} - 0.6387 * X_{2t} \tag{13.14}$$

și este semnificativă din perspectiva testelor uzuale (R^2, testele de semnifacție ale estimatorilor, testul F).

13.2.2 Regresia multivariată

Regresia multifactorială (sau multivariată) presupune existența unor k variabile explicative. Fie X_{it} valoarea variabilei explicative i, în înregistrarea t, unde i = 1, 2,... k iar t = 1, 2,... n. Dacă se acceptă ipoteza că relația dintre Y_t și variabilele explicative este lineară, atunci se poate scrie:

$$\begin{cases} Y_1 = a_0 + a_1 X_{11} + a_2 X_{21} + \cdots + a_k X_{k1} + e_1 \\ Y_2 = a_0 + a_1 X_{12} + a_2 X_{22} + \cdots + a_k X_{k2} + e_2 \\ Y_3 = a_0 + a_1 X_{13} + a_2 X_{23} + \cdots + a_k X_{k3} + e_3 \\ \vdots \\ Y_t = a_0 + a_1 X_{1t} + a_2 X_{2t} + \cdots + a_k X_{kt} + e_t \\ \vdots \\ Y_n = a_0 + a_1 X_{1n} + a_2 X_{2n} + \cdots + a_k X_{kn} + e_n \end{cases} \tag{13.15}$$

Cu notațiile:

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ Y_n \end{pmatrix} \tag{13.16}$$

$$X = \begin{pmatrix} 1 & X_{11} & X_{21} \cdots & X_{k1} \\ 1 & X_{12} & X_{22} \cdots & X_{k2} \\ 1 & X_{13} & X_{23} \cdots & X_{k3} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_{1n} & X_{2n} \cdots & X_{kn} \end{pmatrix} \tag{13.17}$$

$$A = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} \tag{13.18}$$

$$e = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{pmatrix} \tag{13.19}$$

sistemul poate fi scris matriceal astfel:

$$Y = XA + e \tag{13.20}$$

Rezolvarea sistemul va avea ca rezultat obținerea următoarei forme pentru vectorul estimatorilor:

$$\widehat{A} = (X'X)^{-1}X'Y \tag{13.21}$$

Exemplu
Fie următoarele modele lineare:
- modelul `regresie3`:

$$Y_t = a_0 + a_1X_{1t} + a_2X_{2t} + a_3X_{3t} + e_t \tag{13.22}$$

- modelul `regresie4`

$$Y_t = a_0 + a_1X_{1t} + a_2X_{2t} + a_3X_{3t} + a_4X_{4t} + e_t \tag{13.23}$$

În detaliu este afișat modelul *regresie*4.

```
> regresie3 <- lm(Y ~ X1 + X2 + X3, data = datele_mele)
> regresie4 <- lm( ~ ., data = datele_mele)
> summary(regresie4)

Call:
lm(formula = Y ~ ., data = datele_mele)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73831 -0.19110 -0.00296  0.15243  0.86870

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.1128     0.5355   3.945   0.0008 ***
X1           -3.8260     3.7352  -1.024   0.3179
X2           -2.5528     1.1984  -2.130   0.0458 *
X3            3.7555     2.3345   1.609   0.1234
X4            2.9481     4.3294   0.681   0.5037
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4215 on 20 degrees of freedom
Multiple R-squared:  0.8819,Adjusted R-squared:  0.8583
F-statistic: 37.34 on 4 and 20 DF,  p-value: 5.184e-09
```

Se observă că pentru *regresie*4 s-a folosit forma `Y ~ .`, adică influența tuturor variabilelor explicative din setul de date disponibil:

$$\widehat{Y_t} = 2.1128 - 3.8260 * X_{1t} - 2.5528 * X_{2t} + 3.7555 * X_{3t} + 2.9481 * X_{4t} \tag{13.24}$$

13.3 Testarea semnificației estimatorilor

Estimatorii din modelul linear, calculați prin metoda celor mai mici pătrate, sunt variabile aleatoare repartizate normal și au media egală cu valoarea parametrului pe care îl estimează (sunt nedeplasați). Fiind variabile aleatoare, estimatorii pot avea – evident, cu probabilități diferite – orice valoare situată într-un anumit interval. Dacă acest interval conține și valoarea zero, atunci, cu o probabilitate care poate fi calculată, estimatorii pot lua valoarea zero. De exemplu, în modelul unifactorial, intensitatea legăturii dintre X și Y este dată de valoarea parametrului a_1, estimat prin $\widehat{a_1}$. Dacă estimatorul $\widehat{a_1}$ poate fi zero cu o

probabilitate $P(\widehat{a_1} = 0)$ suficient de mare, înseamnă că există riscul ca între variabilele X și Y să nu fie nici o legătură, sau cel puțin se poate afirma că eșantionul selectat nu oferă argumente statistice suficient de puternice care să justifice ipoteza unei legături între X și Y de tipul celei admise prin modelul de regresie lineară prezentat. Pentru anumite tipuri de probleme este important și testul privind semnificația estimatorului $\widehat{a_0}$, deoarece admiterea ipotezei $\widehat{a_0} = 0$ ar putea avea implicații teoretice deosebite (Jula, Jula).

Procedura uzuală aplicată pentru testarea semnificației parametrilor din modelul unifactorial de regresie lineară urmărește testarea ipotezei nule H_0: parametrii nu diferă semnificativ de zero, contra ipotezei alternative H_1: parametrii din ecuația de regresie sunt, în valoare absolută, strict pozitivi. Atunci, sub ipoteza H_0, statistica

$$t_{\widehat{a_i}} = \frac{\widehat{a_i}}{s_{\widehat{a_i}}} \tag{13.25}$$

urmează o distribuție t-Student cu `n - k - 1` grade de libertate. Se respinge ipoteza de nul H_0: $\widehat{a_i} = 0$ (X_i nu influențează Y) dacă valoarea absolută a acestui test este mai mare decât o valoare critică obținută din tabelele distribuției `t (Student)`.

Exemplu

```
> summary(regresie4)

Call:
lm(formula = Y ~ ., data = datele_mele)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73831 -0.19110 -0.00296  0.15243  0.86870

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.1128     0.5355   3.945   0.0008 ***
X1           -3.8260     3.7352  -1.024   0.3179
X2           -2.5528     1.1984  -2.130   0.0458 *
X3            3.7555     2.3345   1.609   0.1234
X4            2.9481     4.3294   0.681   0.5037
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.4215 on 20 degrees of freedom
Multiple R-squared:  0.8819,Adjusted R-squared:  0.8583
F-statistic: 37.34 on 4 and 20 DF,  p-value: 5.184e-09
```

Semnificația estimatorilor se deduce din ultima coloană:

- estimatorul $\widehat{a_0}$ este semnificativ (peste 99%)
- estimatorul $\widehat{a_1}$ NU este semnificativ (probabilitatea de a greși respingând ipoteza de nul este 31.79%, mai mare decât pragul standard de 5%)
- estimatorul $\widehat{a_2}$ este semnificativ (peste 95%)
- estimatorul $\widehat{a_3}$ NU este semnificativ
- estimatorul $\widehat{a_4}$ NU este semnificativ

Codurile de semnificație `Signif.codes` ajută la identificare rapidă a esitmatorilor semnificativi.
Conform rezultatelor obținute, modelul ar trebui respecificat (de exemplu, prin eliminarea unei variabile explicative).

Pentru exemplificarea celorlate teste, se continuă cu acest model, cu mențiunea că în practică analiza ar trebui oprită aici și modelul respecificat.

13.4 Coeficientul de determinare și criterii de specificare

13.4.1 Coeficientul de determinare

Prin metoda celor mai mici pătrate se determină acea ecuație de regresie pentru care suma pătratelor abaterilor dintre datele înregistrate și cele calculate este cea mai mică posibilă, pentru clasa de modele respective.
Problema care se ridică, în continuare, este aceea a măsurii în care variația exogenei poate explica evoluția variabilei endogene.

Se definește coeficientul de determinare R^2 ca fiind partea din variația lui Y care poate fi atribuită variației lui X, pentru eșantionul analizat:

$$R^2 = 1 - \frac{\sum_{t=1}^{n} u_t^2}{\sum_{t=1}^{n} (Y_t - \bar{Y})^2} \tag{13.26}$$

Coeficientul de determinare este o mărime pozitivă și subunitară. Cu cât R^2 este mai aproape de unu, cu atât modelul se apropie mai mult de procesul economic modelat. Interpretarea obișnuită a coeficientului de determinare este următoarea: din variația totală a lui Y, $(R^2) * 100\%$ ar putea fi atribuită variației lui X. În afirmația precedentă înlocuirea formulării *ar putea fi atribuită* cu formularea *este cauzată de* poate duce la concluzii eronate, în special atunci când analiza econometrică se referă la serii de timp.

O altă măsură a acurateței ajustării este **coeficientului de determinare corectat**, calculat astfel:

$$\overline{R^2} = 1 - \frac{n-1}{n-k-1}(1-R^2) \tag{13.27}$$

unde `n` este dimensiunea eșantionului și `k` numărul de variabile explicative.

Exemplu

```
> #coeficientul de determinare
> (summary(regresie)$r.squared)
[1] 0.6751684
> #coeficientul de determinare ajustat
> (summary(regresie)$adj.r.squared)
[1] 0.6610453
> (summary(regresie2)$adj.r.squared)
[1] 0.852435
> (summary(regresie3)$adj.r.squared)
[1] 0.8618953
> (summary(regresie4)$adj.r.squared)
[1] 0.8582758
```

Conform rezultatelor, modelul se îmbunătățește prin includerea variabilei $X2$, respectiv $X3$, calitatea acestuia scăzând în momentul includerii variabilei $X4$.

13.4.2 Specificarea modelului multifactorial

Dacă prin includerea unei (unor) variabile suplimentare suma pătratelor reziduurilor scade mai repede decât numărul gradelor de libertate, din punct de vedere econometric se justifică reținerea în model a variabilei (variabilelor) respective.

De asemenea, se poate demonstra următoarea proprietate: dacă valoarea absolută a testului t pentru un parametru din ecuația de regresie lineară multiplă este mai mică decât 1, atunci, eliminând din model variabila explicativă asociată, valoarea coeficientului de determinare corectat va crește; dacă se elimină o variabilă pentru care *t statistic* este mai mare decât 1, valoarea coeficientului de determinare corectat se va reduce.
S-au definit mai multe așa numite criterii informaționale:

- Criteriul informațional Akaike (AIC)
- Criteriul informațional Schwartz (BIC)
- Criteriul informațional Hannan-Quinn (HQ)

O condiție pentru includerea unei noi variabile explicative este ca prin această re-specificare a modelului să se obțină o valoare mai mică pentru criteriile menționate.

Exemplu
Deoarece dimensiunea eșantionului este relativ mică, se recomandă utilizarea testului BIC:

```
> BIC(regresie, regresie2, regresie3, regresie4)
          df      BIC
regresie   3 57.11696
regresie2  4 38.43460
regresie3  5 38.83405
regresie4  6 41.47996
```

Testul indică modelul 2 (Y în funcție de variabilele X_1 și X_2) ca fiind cel mai bun.

13.5 Testarea ipotezelor referitoare la erorile din model

13.5.1 Normalitatea distribuției erorilor

Normalitatea distribuției erorilor reprezintă o condiție esențială pentru evaluarea calității estimatorilor calculați prin metoda celor mai mici pătrate, în cazul modelelor lineare de regresie. Aceasta deoarece majoritatea rezultatelor referitoare la regresia lineară au fost dezvoltate pornind de la ipoteza normalității distribuției erorilor.

Consecințele nerespectării ipotezei de normalitate a erorilor sunt:

1. Estimatorii parametrilor din model sunt nedeplasați și consistenți.
2. Estimatorii parametrilor din model nu sunt eficienți.
3. Estimatorii parametrilor din model nu au proprietatea de maximă verosimilitate.
4. Testul *t statistic* (Student) aplicat pentru analiza semnificației estimatorilor nu este valid

Unul dintre cele mai cunoscute teste privind normalitatea erorilor este testul Jarque-Bera (Jarque şi Bera, 1980a,b), test care folosește momentele necentrate ale reziduurilor pentru a estima conformitatea distribuția erorilor. Momentul centrat de ordin r al erorilor este:

$$\mu_r = \frac{\sum_{t=1}^{n} u_t^r}{n} \tag{13.28}$$

Testul este formulat astfel:

$$JB = n\left[\frac{1}{6} * \frac{\mu_3^2}{\mu_2^3} + \frac{1}{24} * \left(\frac{\mu_4}{\mu_2^2} - 3\right)^2\right] + n\left(\frac{3}{2} * \frac{\mu_1^2}{\mu_2} - \frac{\mu_3 * \mu_1}{\mu_2^2}\right) \tag{13.29}$$

ipoteza de nul a testului Jarque-Bera este H_0: *erorile sunt normal distribuite* și este evaluată contra ipotezei alternative H_1: *erorile urmează o altă distribuție din familia distribuțiilor de tip Pearson.*

Jarque & Bera au demonstrat că, sub ipoteza de nul, statistica JB urmează asimptotic o distribuție χ^2 cu 2 grade de libertate. ipoteza de nul (normalitatea distribuției erorilor) este respinsă dacă valoarea testului JB este mai mare decât valoarea teoretică din distribuția $\chi_2^2(\alpha)$, unde α este pragul de semnificație (probabilitatea erorii de tipul I.)

Atenuarea consecințelor non-normalității distribuției erorilor

Se demonstrează că testele bazate pe momentele centrate ale reziduurilor sunt foarte sensibile la prezență punctelor atipice (în engleză *outliers*)[2]. Dacă non-normalitatea este datorată existenței punctelor de acest tip, atunci problema se rezolvă prin explicarea prezenței punctelor respective. Atunci când ipoteza de normalitate a distribuției erorilor este invalidată,

[2]Jula D., 2003, Introducere în econometrie, Editura Professional Consulting, București, pag. 211-213

de obicei se aplică anumite tehnici de transformare a seriilor de date, de obicei prin logaritmare.

Exemplu
Testul Jarque-Bera nu se află în pachetul standard. Testul implementat in R folosește momentele cetrate ale erorilor și are valori identice cu cel prezentat anterior pentru modele care au termen liber. Se încărca pachetul `tseries`:

```
> library(tseries)
```

Aplicarea testului pentru modelul unifactorial duce la următoarele rezultate:

```
> jarque.bera.test(summary(regresie)$residuals)

Jarque Bera Test

data:  summary(regresie)$residuals
X-squared = 2.2203, df = 2, p-value = 0.3295
```

Interpretarea este: dacă se respinge ipoteza de nul, conform căreia erorile sunt normal distribuite, probabilitatea de a greși este de aproximativ 33%. În consecință, ipoteza de nul nu este respinsă.
Obs. Testul are semnificație asimptotică, astfel încât în cazul modelului analizat, dimensiunea relativ mică a eșantionului poate duce la interpretări eronate ale testului.

Alte teste de normalitate a distribuției erorilor în R se găsesc în pachetul de funcții `nortest`:

- `ad.test` - testul Anderson-Darling
- `cvm.test` - testul Cramer-von Mises
- `lillie.test` - testul Lilliefors (Kolmogorov-Smirnov)
- `pearson.test` - testul χ^2 - Pearson
- `sf.test` - testul Shapiro-Francia

Exemplu
Se încarcă pachetul `nortest`. Dacă nu este descărcat, se va folosi comanda `> install.packages("nortest")` Testul Anderson-Darling:

```
> ad.test(summary(regresie)$residuals)

Anderson-Darling normality test

data:  summary(regresie)$residuals
A = 0.5968, p-value = 0.1085
```

Pentru a aplica testul Cramer-von Mises, se folosește:

```
> cvm.test(summary(regresie)$residuals)

Cramer-von Mises normality test

data:  summary(regresie)$residuals
W = 0.1067, p-value = 0.08509
```

Pentru a aplica testul Lilliefors (Kolmogorov-Smirnov):

```
> lillie.test(summary(regresie)$residuals)

Lilliefors (Kolmogorov-Smirnov) normality test

data:  summary(regresie)$residuals
D = 0.1561, p-value = 0.1202
```

Pentru a aplica testul χ^2 - Pearson:

```
> pearson.test(summary(regresie)$residuals)

Pearson chi-square normality test

data:  summary(regresie)$residuals
P = 11.16, p-value = 0.0483
```

Pentru a aplica testul Shapiro-Francia:

```
> sf.test(summary(regresie)$residuals)

Shapiro-Francia normality test

data:  summary(regresie)$residuals
W = 0.9425, p-value = 0.1497
```

Toate testele indică normalitatea distribuției erorilor la pragul standard de semnificație.

13.5.2 Autocorelarea erorilor

În prezența autocorelării erorilor este afectată calitatea estimatorilor calculați prin metoda celor mai mici pătrate pentru parametrii modelului de regresie. Autocorelarea erorilor presupune existența unei covariante nenule între erorile din ecuația de regresie.
În modelele economice de regresie lineară se întâlnesc des situații în care erorile sunt autocorelate. În special, autocorelarea erorilor apare în modelele construite pentru seriile de timp. Principalele cauze care determină fenomenul respectiv sunt:

1. omiterea din model a unor variabile explicative cu influență semnificativă asupra variabilei endogene;
2. ignorarea prezenței unor relații nelineare între variabile și
3. imposibilitatea evitării unor erori de măsurare.

Consecințe ale autocorelării erorilor
În prezența autocorelării erorilor, estimatorii calculați prin metoda celor mai mici pătrate pentru parametrii modelului de regresie lineară rămân nedeplasați și consistenți, deoarece în demonstrarea proprietăților respective nu s a folosit ipoteza de lipsă a autocorelării erorilor (ca, de altfel, nici ipoteza de lipsă a heteroscedasticității). În schimb, estimatorii nu sunt eficienți, în sensul că există estimatori ai parametrilor modelului care au o dispersie mai mică decât dispersia estimatorilor calculați prin metoda celor mai mici pătrate.

Mai mult, se demonstrează că dacă erorile sunt autocorelate, atunci dispersia erorilor și dispersiile estimatorilor sunt subestimate. În aceste condiții, valorile testelor de semnificație t Student sunt supradimensionate, iar parametrii pot apărea ca semnificativi chiar dacă în realitate acest lucru este fals.
Dacă fenomenul de autocorelare a erorilor din modelul de regresie lineară este ignorat, iar pentru estimarea parametrilor se folosește metoda celor mai mici pătrate, atunci, sintetic, consecințele sunt următoarele:

- Estimatorii parametrilor din model sunt nedeplasați și consistenți.

- Estimatorii parametrilor din model nu sunt eficienți și nu au proprietatea de maximă verosimilitate.
- Estimatorii calculați pentru dispersia și covariația parametrilor sunt deplasați, nu sunt consistenți și nu sunt eficienți.
- Testul *t statistic (Student)* aplicat pentru analiza semnificației estimatorilor nu este valid.
- Valorile *t Student* calculate pentru estimarea semnificației parametrilor sunt supradimensionate, ceea ce sugerează o semnificație a parametrilor mai mare decât este în realitate.
- Abaterea standard a erorilor este subdimensionată față de valoarea reală și, în consecință, coeficientul de determinare R^2 este supradimensionat, ceea ce indică o ajustare mai bună decât este în realitate.

Testul Durbin – Watson (Durbin şi Watson, 1950, 1951) este cea mai cunoscută procedură utilizată pentru identificarea autocorelării de ordinul întâi a erorilor din modelele de regresie lineară. Statistica Durbin – Watson se calculează astfel:

$$dw = \frac{\sum_{t=2}^{n}(u_t - u_{t-1})^2}{\sum_{t=1}^{n} u_t^2} \tag{13.30}$$

Din tabelele testului bilateral Durbin – Watson se selectează (pornind de la nivelul de semnificație ales – de obicei, 0.05 sau 0.01), valorile critice dL și dU, pentru k – numărul de variabile explicative din model și n – dimensiunea eșantionului.

- Se acceptă ipoteza H_0 – lipsa autocorelării de ordinul I, dacă $dU \leq dw \leq 4 - dU$
- Se respinge H_0 dacă $dw \leq dL$ sau $dw \leq 4dU$.
- Dacă $dL \leq dw \leq dU$ sau $4 - dU \leq dw \leq 4 - dL$, testul este neconcludent.

Exemplu
Testul Durbin-Watson se găsește în pachetul `lmtest` și se folosește apelând funcția `dwtest()`:

```
> library(lmtest)
```

```
> dwtest(regresie)

Durbin-Watson test

data:  regresie
DW = 1.9033, p-value = 0.3929
alternative hypothesis: true autocorrelation is greater than 0
```

Potrivit testului DW, se respinge ipoteza de autocorelare a erorilor (de ordinul I). Dacă s-ar accepta o asemnea ipoteză, probabilitatea de eroare (eroarea de ordinul I) ar fi de 39.3%, mai mare decât pragul standard de 5%.
Obs. Dimensiunea relativ mică a eșantionului poate duce la interpretări eronate ale testelor.

13.5.3 Multicolinearitatea

Una dintre ipotezele de fundamentare a modelului de regresie lineară multifactorială afirmă faptul că nu există nici o relație lineară între două sau mai multe variabile explicative (absența colinearității). În practica modelării însă, o asemenea ipoteză este extrem de greu de îndeplinit, deoarece între variabilele economice există multiple legături de intercondiționare. O renunțare la ipoteza de independență a variabilelor explicative din modelele de regresie creează probleme în ceea ce privește estimarea parametrilor și calitatea estimatorilor.

Consecințe ale multicolinearității

- Dacă două sau mai multe variabile explicative din modelul de regresie multiplă sunt perfect corelate, estimatorii parametrilor nu pot fi calculați prin metoda celor mai mici pătrate.
- Dacă anumite variabile explicative sunt relativ puternic corelate, estimatorii obținuți prin metoda celor mai mici pătrate sunt lineari, normal distribuiți, nedeplasați, consistenți și de maximă
- Efectul multicolinearității se manifestă în creșterea abaterii standard a estimatorilor calculați pentru parametrii modelului, ceea ce reduce valoarea testului t statistic (Student). Aceasta face estimatorii mai puțin semnificativi (posibil chiar nesemnificativi). Totuși, testul t rămâne valid.

- Se reduce precizia estimatorilor calculați pentru parametrii modelului, în sensul că abaterea standard mare duce la creșterea intervalului de încredere în care sunt garantați parametrii.
- Deoarece covarianța între variabilele explicative corelate relativ puternic poate fi mare (în valoare absolută), interpretarea parametrilor individuali este dificilă.

Există anumite elemente ce pot indica prezența multicolinearității:

- Coeficienții de corelație lineară, calculați pentru perechile de variabile explicative din model, sunt mari în valoare absolută (sunt, în modul, apropiați de $\pm$ 1).
- Determinantul matricei $(X'X)$ are valori în apropierea lui zero. Coeficientul de determinare R^2 este mare, iar valorile testelor t (Student), calculate pentru parametrii modelului sunt mici.
- Estimatorii parametrilor sunt sensibili la specificarea modelului.
- Multicolinearitatea este identificată prin aplicarea unor proceduri formale

Testul Belsley, Kuh & Welsch (BKW)

Testul BKW pornește de la analiza valorilor proprii ale matricei de covarianță a estimatorilor și se bazează pe observația că dacă există un număr mare de valori proprii în apropierea lui zero, atunci există riscul unor dependențe lineare între variabile.[3]
Pentru aplicarea testului se calculează așa-numita CN – condiție de număr (în engleză *condition number*), ca raport între cea mai mare (λ_{max}) și cea mai mică (λ_{min}) valoare proprie a matricei de covarianță a estimatorilor.
Totodată, se calculează `CI` – condiția de indice (în engleză *condition index*), după relația:

$$CI_i = \sqrt{\frac{\lambda_{max}}{\lambda_i}} \tag{13.31}$$

pentru fiecare valoare proprie a matricei respective (λ_i). O valoare mare a coeficienților CI indică prezența multicolinearității. Practic, se consideră că o valoare CI $<$ 10, nu ridică probleme, 10 $<$ CI $<$ 30,

[3] Jula D., Jula N.-M., 2015, *Econometrie*, Editura Mustang, București

înseamnă o colinearitate moderată, iar o valoare CI > 30 este asociată cu o colinearitate severă.
De asemenea, se calculează descompunerea dispersiei estimatorilor, în funcție de valorile proprii λ_i. Dacă proporția din dispersia totală a unui estimator, asociată celui mai mare CI este superioară unui anumit prag (de exemplu, 50%), atunci variabila respectivă este corelată puternic cu celelalte variabile explicative din model.

Atunci când este determinată o situație de multicolinearitate, există metode de atenuare:

- Eliminarea unor variabile explicative
- Realizarea unor observații suplimentare asupra variabilelor din model (se mărește volumul eșantionului)
- Prelucrarea primară a datelor (calculul ritmurilor de modificare, a sporurilor, indicilor, logaritmarea valorilor observate etc.)
- Regresia ridge

Exemplu
Testul BKW se găsește în pachetul `perturb`:

```
> library(perturb)
> #testez multicolinearitatea
> colldiag(regresie4, add.intercept = FALSE, scale = FALSE)
Condition
Index Variance Decomposition Proportions
           X1    X2    X3    X4
1    1.000 0.000 0.000 0.000 0.000
2    2.714 0.000 0.002 0.000 0.000
3  123.667 0.016 0.930 0.930 0.034
4  264.606 0.984 0.068 0.070 0.965
```

Valorile din coloana `Condition Index` peste 30 indică o colinearitate severă.
Există două valori ale CI superioare pragului de 30, ceea ce indică existența unei colinearități severe între variabilele explicative.

Obs. Dimensiunea relativ mică a eșantionului poate duce la interpretări eronate ale testelor.

Testul VIF Pentru a identifica ce variabilă trebuie eliminată, se calculează indicatorul VIF (în engleză *Variance Inflation Factors*), utilizând pachetul `car`:

```
> library(car)
> vif(regresie4)
       X1        X2        X3        X4
1261.4577  261.9420  113.8169 1083.2679
```

Conform rezultatelor, se elimină variabila $X1$ (cel mai mare coeficient) și se reface modelul de regresie fără această variabilă:

```
> regresie5 <- lm(Y ~ X2 + X3 + X4, data = datele_mele)
> summary(regresie5)

Call:
lm(formula = Y ~ X2 + X3 + X4, data = datele_mele)

Residuals:
     Min       1Q   Median       3Q      Max
-0.70422 -0.20367  0.03907  0.17408  0.92982

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.0815     0.5353   3.888 0.000848 ***
X2           -2.0849     1.1092  -1.880 0.074096 .
X3            2.8307     2.1554   1.313 0.203238
X4           -1.2518     1.3917  -0.899 0.378614
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.422 on 21 degrees of freedom
Multiple R-squared:  0.8757,Adjusted R-squared:  0.8579
F-statistic: 49.32 on 3 and 21 DF,  p-value: 1.106e-09
```

Se verifică din nou valoarea CI:

```
> colldiag(regresie5, add.intercept = FALSE, scale = FALSE)
Condition
Index Variance Decomposition Proportions
```

```
           X2    X3    X4
1    1.000 0.000 0.000 0.000
2    2.831 0.002 0.000 0.002
3  106.627 0.998 1.000 0.998
> vif(regresie5)
       X2        X3        X4
223.88494  96.79392 111.67316
```

Se elimină și variabila $X2$, se obține:

```
> regresie6 <- lm(Y ~ X3 + X4, data = datele_mele)
> summary(regresie6)

Call:
lm(formula = Y ~ X3 + X4, data = datele_mele)

Residuals:
     Min       1Q   Median       3Q      Max
-0.71706 -0.12487 -0.03149  0.14217  1.09997

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.8325     0.5477   3.346  0.00293 **
X3           -1.1996     0.2321  -5.169 3.50e-05 ***
X4            1.3524     0.1395   9.694 2.12e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4456 on 22 degrees of freedom
Multiple R-squared:  0.8548,Adjusted R-squared:  0.8416
F-statistic: 64.75 on 2 and 22 DF,  p-value: 6.054e-10
```

Se verifică valoarea CI:

```
> colldiag(regresie6)
Condition
Index Variance Decomposition Proportions
          intercept X3    X4
1   1.000 0.003     0.004 0.014
2   5.539 0.022     0.100 0.840
3  13.530 0.975     0.896 0.146
```

În cazul de față, modelul are toți estimatorii semnificativi și nu prezintă multicolinearitate. Coeficientul de determinare este 0.8548. De asemenea, erorile sunt distribuite normal:

```
> jarque.bera.test(summary(regresie6)$residuals)

Jarque Bera Test

data:  summary(regresie6)$residuals
X-squared = 1.8367, df = 2, p-value = 0.3992
```

și nu prezintă autocorelare de ordin 1:

```
> dwtest(regresie6)

Durbin-Watson test

data:  regresie6
DW = 1.6106, p-value = 0.1423
alternative hypothesis: true autocorrelation is greater than 0
```

Capitolul 14

Regresia logistică

14.1 Precizări preliminare

În capitolele anterioare, în care a fost prezentată regresia liniară, obiectivul principal era acela de a modela variabila dependentă (y), considerată variabilă continuă, prin estimarea parametrilor ecuației de regresie pe baza metodei celor mai mici pătrate. Astfel de metode sunt denumite metode statistice parametrice sau cantitative.

Problema esențială care va fi pusă în discuție în acest capitol este de a evidenția legătura între fenomenele socio-economice prin intermediul instrumentarului statistic, pe baza caracteristicilor calitative. Astfel de metode impun un mod de abordare diferit față de metodele parametrice, făcând parte din grupa modelelor de regresie liniară generalizată (*generalized linear model - GLM*). Modele liniare generalizate au fost formulate de către John Nelder și Robert Wedderburn ca o modalitate de unificare a diverselor modele statistice, inclusiv cele de regresie logistică și regresie Poisson.

Regresia logistică este un model probabilistic de analiză statistică a legăturii dintre două sau mai multe fenomene, pe baza anumitor caracteristici, rezultatul fiind o *variabilă categorială*. În cazul regresiei logistice se pune problema predicției probabilității ca variabila rezultat să înregistreze una dintre categoriile de răspuns posibile, estimarea parametrilor ecuației de regresie respectând criteriul verosimilității maxime[1].

Există situații în care variabila dependentă poate înregistra două sau

[1]Regresia liniară modelează/estimează valoarea aşteptată a unei cantități necunoscute ca o combinaţie liniară a unui set de valori observate (predictori). Acest model de regresie se potriveşte atunci când variabila dependentă este distribuită normal.

mai multe categorii de răspuns; dacă există două categorii, variabila este de tip binar sau dihotomic (de exemplu, variabila sex poate înregistra două valori: masculin și feminin) – în acest caz se aplică *regresia logistică binomială*; dacă există mai multe categorii de răspuns ale variabilei rezultat, se aplică *regresia logistică multinomială* (de exemplu, variabila nivel de educație poate înregistra mai multe categorii: scăzut, mediu sau înalt). Mai mult, dacă aceste categorii sunt ordonate, adică variabila dependentă este ordinală, se aplică *regresia logistică ordinală*.

Modelele de regresie pentru care variabila dependentă este calitativă pot fi de tip probit sau logit, fiind diferite în ceea ce privește specificarea distribuției erorilor. Dacă distribuția cumulată a erorilor este o funcție logistică rezultă un model de tip *logit*. Dacă distribuția cumulată a erorilor urmează o distribuție normală, rezultă un model de tip *probit*.

Regresia logistică poate avea una sau mai multe variabile independente (x_i), denumite și predictori, sau variabile explicative; în modelele de regresie logistică, variabilele explicative pot fi continue și/sau categoriale.

14.2 Regresia logistică simplă

Variabilele binare reprezintă cea mai comună formă de variabile categoriale, metodele de regresie prezentate în secțiunile următoare având o importanță fundamentală în modelarea fenomenelor din diverse domenii din sfera economico-socială, pe baza analizei legăturilor dintre variabile.

Pentru început, se va considera un model de regresie logistică simplă, în care variabila dependentă (y) este dihotomică (binară), având două categorii (0 și 1) și în care există o singură variabilă independentă (x), considerată cantitativă.

Pentru a facilita înțelegerea regresiei logistice, se va considera un experiment în care răspunsul este măsurat ca *succes* sau *eșec*, categorii de răspuns ce pot fi codificate cu 1, respectiv cu 0. Astfel de variabile dependente pot fi întâlnite în viața reală, de regulă, atunci când se dorește predicția apartenenței unei persoane la două clase (categorii) diferite, de exemplu:

- utilizează/nu utilizează internetul;
- face parte/nu face parte din categoria de interes;
- va cumpăra/nu va cumpăra un anumit produs;

- face/nu face accident în primul an după ce a obținut permisul de conducere;
- va vota/nu va vota;
- are/nu are cont pe Facebook etc.

Se va imagina un model prin care se estimează probabilitatea ca o anumită persoană să decidă contractarea unui credit bancar în funcție de venitul lunar disponibil. Un model adecvat ar putea prezice că o schimbare cu 10 lei a venitului lunar ar putea avea ca rezultat: să fie de două ori mai mult sau mai puțin probabil ca persoana să decidă contractarea creditului. Dar ce înseamnă "de două ori mai mult" în termeni de probabilitate? Nu poate însemna literalmente a dubla valoarea probabilității (de exemplu, 50% să devină 100%). Se poate spune, însă, că șansele de a decide contractarea creditului bancar se pot dubla. Astfel de situații, care permit variabilei rezultat să aibă o distribuție arbitrară (alta decât distribuția normală), pot fi modelate prin metoda regresiei logistice.

Așadar, regresia logistică estimează probabilitatea ca un caz să fie inclus într-una sau alta din categoriile definite de variabila dependentă (succes sau eșec) în funcție de variabila independentă (x). Cu alte cuvinte, pe baza regresiei logistice se va putea estima dacă evenimentul: *va avea loc* sau *nu va avea loc*. Dacă evenimentul va avea loc, atunci apariția acestuia are loc cu probabilitatea p.

14.2.1 Reprezentarea grafică a legăturii între două variabile

Se consideră un set de date care conține valorile a două variabile:

- y reprezintă apartenența/non-apartenența unui individ la categoria *beneficiari de prestații sociale*;
- x este variabilă numerică, reprezentând venitul mediu lunar individual, exprimat în unități monetare.

Cunoscând datele referitoare la venitul mediul lunar obținut de $n = 35$ persoane și cunoscând şi dacă aceste persoane sunt beneficiare de sprijin financiar din partea statului (sprijin care se acordă familiilor cu venituri reduse), se poate construi un model cu care se poate răspunde la întrebarea: *care este probabilitatea ca persoana să beneficieze de sprijin financiar din partea statului* $(y = 1)$ *în condițiile unui venit mediu lunar dat* (x)?

```
> date <- data.frame(obs = 1:35, y = c(1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1), x=c(502, 1550, 980, 1901, 766, 803, 2121, 2594, 204, 2801,
882, 1533, 355, 795, 533, 1935, 217, 150, 1482, 102, 573, 312,
1198, 2011, 1549, 755, 139, 589, 480, 117, 208, 993, 190, 121, 418))
```

Unde:

- `obs` = observația
- `y` beneficiar/non-beneficiar
- `x` = venitul mediu lunar

Evenimentele aleatoare determinate de variabila x (venitul mediu lunar al beneficiarilor de prestații sociale) conținute în clasa de evenimente $y = 1$ (categoria beneficiarilor de prestații sociale) se obțin cu funcția `subset()`:

```
> date2 <- subset(date, y == 1)
```

O posibilitate de reprezentare a legăturii dintre două variabile este aceea de a face un grafic de tip *scatterplot*[2], adică norul de puncte care se stabilesc la intersecția valorilor celor două variabile, pentru fiecare observație. În R, pentru a reprezenta grafic legătura între o variabilă dependentă binară și o variabilă independentă cantitativă, se utilizează funcția `plot()`.

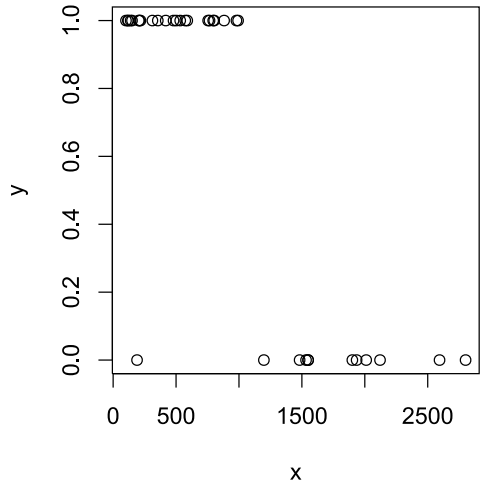
```
> date <- read.table(file.choose(), head = TRUE)
> head(date)
  y    x
1 1  502
2 0 1550
3 1  980
4 0 1901
5 1  766
6 1  803

> attach (date)
> plot(x,y)
```

În figura următoare se observă că variabila dependentă y tinde să ia valoarea 0 la valori mai mici ale lui x și valoarea 1 pe măsură ce valorile lui x cresc.

[2]Graficul care prezintă legătura dintre două variabile poartă numele de corelogramă

Figura 14.1: Reprezentarea grafică a legăturii între o variabilă dependentă y binară și o variabilă independentă x cantitativă



Punctele de pe grafic nu pot oferi informații despre forma dependenței dintre variabile, ca în cazul regresiei liniare. Motivul este acela că nu există o variație a lui y (există doar două valori posibile, respectiv 0 și 1).

14.2.2 Șansele de succes. Ecuația de regresie logistică simplă

Modelul de regresie logistică este legat direct de noțiunea de șanse de succes (*odds*, în limba engleză), notat Ω, reprezentând raportul dintre probabilitatea de succes și probabilitatea de eșec a evenimentului:

$$\Omega = \frac{p}{1-p} \tag{14.1}$$

unde:
p este probabilitatea ca y să ia valoarea 1, adică fenomenul să înregistreze un succes, probabilitate condiționată de valorile variabilei independente x;
$(1-p)$ este probabilitatea ca y să ia valoarea 0, adică evenimentul să înregistreze un eșec, de asemenea probabilitatea de insucces fiind condiționată de influența lui x.

De exemplu:

- când $p = 0,6$ rezultă $1 - p = 0,4$ şi $\Omega = \frac{0,6}{0,4} = 1,5$; înseamnă că

probabilitatea de succes a evenimentului este de 1,5 mai mare decât probabilitatea de eșec;

- când $p = 0,99$ rezultă $1 - p = 0,01$ şi $\Omega = \frac{0,99}{0,01} = 99$; înseamnă că probabilitatea de succes este de 99 mai mare decât probabilitatea de eșec.

Pot exista trei situații:

- când $\Omega > 1$, evenimentul este mai probabil să înregistreze un succes (adică este mai probabil să aibă loc);
- când $\Omega < 1$, evenimentul este mai probabil să nu aibă loc;
- când $\Omega = 1$, există șanse egale ca evenimentul să se întâmple sau nu.

În timp ce probabilitatea ia valori cuprinse între 0 și 1, Ω poate lua valori pozitive în intervalul $(0, +)$. Prin logaritmarea șanselor de succes se recurge la așa-zisa transformare logistică, denumită și *logit*, ce poate lua valori cuprinse în intervalul $(-, +)$. Transformarea logit este necesară pentru a proiecta probabilitatea p din intervalul (0,1) în intervalul $(-, +)$, fapt necesar în procesul de estimare a parametrilor ecuației de regresie. Așadar, când p tinde spre valoarea 1, șansele de succes tind spre $+$ și funcția *logit* tinde tot spre $+$. Când p tinde spre 0, șansele de succes tind spre 0, iar funcția logit tinde spre $-$. Când probabilitățile de succes și de eșec sunt egale, $\Omega = 1$, iar *logit* ia valoarea 0.

Modelul de regresie logistică simplă este dat de formula:

$$ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x \tag{14.2}$$

β_0 și β_1 sunt parametrii ecuației de regresie, denumiți și coeficienți logit; Modelul de regresie logistică are formă liniară dacă se aplică funcția *logit* probabilității p:

$$logit(p) = \beta_0 + \beta_1 x \tag{14.3}$$

Modelul se mai poate scrie[3]:

$$\frac{p}{1-p} = e^{\beta_0 + \beta_1 x} \tag{14.4}$$

[3]Pe baza unei proprietăți matematice a funcției exponențiale $e^{ln(A)} = A$

Adică,

$$\Omega = e^{\beta_0+\beta_1 x} \tag{14.5}$$

unde:

$$p = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \tag{14.6}$$

În exemplul dat, se poate constata că, pentru o observație din setul de 35 de perechi de valori (x, y), dacă $p > 0,5$, atunci este mai probabil ca observația să aparțină grupului caracterizat de $y = 1$, adică persoana să fie beneficiar de prestații sociale. Această condiție este echivalentă cu $\Omega > 1$, adică $logit > 0$.

Expresia grafică a regresiei logistice are o formă specială, denumită *sigmoid (S-shaped)*, care decurge din natura logaritmică a relației dintre valorile variabilei independente și cele ale variabilei rezultat.

Formula care estimează probabilitatea de producere cu succes a evenimentului în funcție de valorile variabilei independente implică faptul că $p(x)$ crește sau descrește în formă de S.

Sigmoidul ia aspecte particulare în funcție de natura relației dintre variabila independentă și variabila rezultat. Pentru datele considerate în exemplul anterior, în care variabila dependentă este binară (beneficiar/non-beneficiar de prestații sociale), iar variabila explicativă este venitul mediu lunar, forma sigmoidului este ca o reprezentare în oglindă a literei S. Această reprezentare (Figura 14.2) se datorează faptului că legătura dintre cele două variabile este inversă.

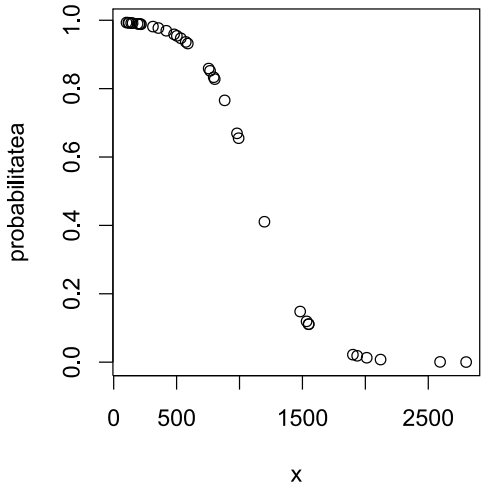
```
> probabilitatea <- exp(5.495958 - 0.004889 * x) /
(1 + exp(5.495958 - 0.004889 * x))
> plot(x, probabilitatea)
```

14.2.3 Estimarea parametrilor de regresie

În cazul regresiei logistice, estimarea parametrilor ecuației de regresie se bazează pe maximizarea probabilității de producere cu succes a evenimentului, nu pe minimizarea sumei pătratelor, ca în cazul regresiei liniare.

Pentru a explica metoda de calcul a coeficienților de regresie, se va introduce un concept nou: verosimilitatea (*likelihood*). Se consideră că

Figura 14.2: Reprezentarea grafică a curbei de regresie logistică în cazul legăturii între variabila binară beneficiar/non-beneficiar și venit



un model este - mai mult sau mai puțin – verosimil atunci când, folosind variabilele independente din model, se pot estima corect valorile variabilei dependente y. Conceptul de verosimilitate poate fi utilizat în calcularea coeficienților de regresie pe baza unui algoritm iterativ, denumit metoda verosimilității maxime (*Maximum-Likelihood Estimation - MLE*). Metoda a fost introdusă de R. A. Fischer în 1912 și se bazează pe transformarea variabilei dependente într-o variabilă de tip *logit* (logaritmul natural al șansei ca evenimentul să se producă sau nu).

Problema estimării coeficienților logit este de a determina, pe baza setului de valori observate, acele valori ale parametrilor β_0 şi β_1 astfel încât modelul matematic $logit(p) = \beta_0 + \beta_1 x$ să poată descrie cât mai credibil/corect fenomenul observat, în sensul că este capabil să descrie comportamentul fenomenului și în alte puncte care nu fac parte din mulțimea inițială de observații. Altfel spus, problema constă în găsirea acelor valori pentru parametrii modelului care vor asigura verosimilitate maximă. Aceste valori se vor nota cu $\hat{\beta}_0$, $\hat{\beta}_1$ şi constituie așa numitele estimații ale parametrilor modelului în sensul verosimilității maxime.

În exemplul în care se cunosc datele referitoare la venitul mediul lunar obținut de $n = 35$ persoane, modelul de regresie logistică estimează probabilitatea ca persoana să beneficieze de sprijin financiar din partea statului ($y = 1$) în condițiile unui venit mediu lunar dat (x). Determinarea coeficienților logit se face pe baza unui algoritm care realizează testarea aleatoare a verosimilității modelului de regresie

logistică pentru mai multe perechi de valori ale parametrilor (β_0 şi β_1). Modelul cel mai potrivit va fi acela pentru care un set de valori ale parametrilor (β_0 şi β_1) asigură verosimilitate maximă.

Algoritmul se bazează pe un set de iterații prin care se testează verosimilitatea modelului, comparând două câte două, perechi de valori ale coeficienților (β_0, β_1). Pentru prima iterație se generează aleatoriu o pereche de valori ale parametrilor (β_0, β_1). Se determină apoi direcția și mărimea cu care parametrii se modifică, cu scopul de a mări verosimilitatea logaritmică (*log likelihood - LL*); procesul se repetă de mai multe ori, până ce LL nu se mai modifică semnificativ, în sensul că se localizează punctul de maxim al funcției logaritmice de verosimilitate în planul parametrilor ($\hat{\beta}_0$, $\hat{\beta}_1$). Astfel, s-au găsit (s-au estimat) coeficienții de regresie pentru care modelul are verosimilitate maximă, pentru datele pe baza cărora s-au calculat.

Metoda verosimilității maxime s-a dezvoltat în ultimii ani, în special datorită creșterii capacității de calcul prin utilizarea computerelor performante și a unor software-uri de analiză statistică.

În R, funcția de bază utilizată pentru modelarea fenomenelor care au ca răspuns variabile categoriale este funcția `glm()` (*generalized linear models*), care are următoarea structură:

```
> glm(formula, family, data, weights, subset, ...)
```

Unde argumentele funcției sunt următoarele:

- formula – reprezintă ecuația modelului de regresie;
- family – este forma distribuției de probabilitate; se poate alege una din cele șase opțini: gaussian, binomial, poisson, Gamma, inverse gaussian, quasi. În cazul regresiei logistice, distribuția de probabilitate este binomială;
- data – reprezintă setul de date în care se găsesc valorile variabilei dependente și ale celei explicative.

Modelele de regresie liniară generalizate sunt modele de regresie liniară extinse care permit modelarea variabilei dependente categoriale pe baza unei funcții de legătură (în engleză *link function*); variabila de răspuns este o variabilă discretă, valorile sale urmând una dintre distribuțiile de probabilitate (de exemplu: normală, binomială, Poisson). În general, dacă se cunoaște că variabila dependentă este dihotomică (family=binomial), funcția `glm()` va întoarce în R rezultatul regresiei logistice.

Utilizând datele din exemplul dat, se aplică funcția `glm()` care creează un model de regresie liniară generalizat (GLM) pentru o distribuție binomială.

```
> mylogit <- glm(y ~ x, data = date_logit, family = "binomial")
> summary(mylogit)
```

Funcția `glm()` returnează următorul rezultat:

```
Call:
> glm(formula = y ~ x, family = "binomial", data = date_logit)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.0257  -0.1424   0.1487   0.3452   0.9198

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  5.495958   1.782551   3.083  0.00205 **
x           -0.004889   0.001684  -2.903  0.00370 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 45.004  on 34  degrees of freedom
Residual deviance: 15.782  on 33  degrees of freedom
AIC: 19.782

Number of Fisher Scoring iterations: 6
```

Conform rezultatelor obținute, modelul de regresie logistică poate fi scris:

$$ln\left(\frac{p}{1-p}\right) = 5.495958 - 0.004889 \times Venit \tag{14.7}$$

Semnificația parametrilor

Ca şi în cazul regresiei liniare, atenția se îndreaptă, în special asupra coeficientului de regresie β_1 - care exprimă modificarea cantității *logit* atunci când x crește cu o unitate, în cazul în care x este variabilă continuă,

sau în cazul binar, când x este 1. Uneori, este mai ușor de interpretat e^{β_1} - care semnifică efectul generat de coeficientul de regresie β_1 asupra șanselor de succes. Pentru a vedea ce se întâmplă cu raportul de șanse dacă x crește cu o unitate (x devine $x+1$), se porneşte de la ecuația de regresie logistică simplă:

$$\Omega_{(x)} = e^{\beta_0+\beta_1 x} = e^{\beta_0} \times e^{\beta_1 x} \tag{14.8}$$

Iar când x devine $x+1$, raportul de șanse devine:

$$\Omega_{(x)} = e^{\beta_0+\beta_1(x+1)} = e^{\beta_0} \times e^{\beta_1 x} \times e^{\beta_1} \tag{14.9}$$

Cele două ecuații ale raportului de șanse diferă prin înmulțirea cu e^{β_1}. Așadar, în cazul regresiei logistice, variabilele independente au efecte multiplicative asupra șanselor de succes. Pentru a vedea care sunt șansele de succes pentru grupul persoanelor care au caracteristica $x = x+1$ faţă de grupul persoanelor caracterizate de x, se poate calcula raportul șanselor între cele două grupuri (în engleză *odds ratio*):

$$OR = \frac{\Omega_{(x+1)}}{\Omega_{(x)}} = \frac{e^{\beta_0+\beta_1(x+1)}}{e^{\beta_0+\beta_1 x}} = \frac{e^{\beta_0} \times e^{\beta_1 x} \times e^{\beta_1}}{e^{\beta_0} \times e^{\beta_1 x}} = e^{\beta_1} \tag{14.10}$$

Aşadar, e^{β_1} reprezintă raportul de șanse care arată ce se întâmplă atunci când x se modifică cu o unitate. Există trei situații posibile:

- $e^{\beta_1} > 1$, situație în care creșterea cu o unitate a lui x conduce la sporirea șanselor de succes (de exemplu, dacă $e^{\beta_1} = 1,5$ înseamnă că șansele de succes cresc, în medie, de 1,5 ori, sau cu 50%, atunci când x creşte cu o unitate);
- $e^{\beta_1} < 1$, situație în care creșterea cu o unitate a lui x conduce la reducerea șanselor de succes (de exemplu, dacă $e^{\beta_1} = 0,5$ înseamnă că șansele de succes scad, în medie, la jumătate, sau cu 50%, atunci când x crește cu o unitate);
- $e^{\beta_1} = 1$, situație în care orice valoarea ar lua x, aceasta nu produce nici un efect asupra șanselor de succes ($e^{\beta_1} = 1$, dacă $\beta_1 = 0$).

Cazul 1. Variabila independentă x este o variabilă continuă

Se consideră exemplul în care variabila dependentă (y) reprezintă apartenența/non-apartenența unui individ la categoria beneficiari de prestații sociale, iar x este variabilă numerică, reprezentând venitul

mediu lunar individual, exprimat în unități monetare. Șansele de succes sunt determinate de raportul între probabilitatea de a fi beneficiar și probabilitatea de a fi non-beneficiar.
În acest caz, modelul cel mai potrivit, care asigură verosimilitatea maximă, este dat de ecuația:

$$p(x) = \frac{e^{5.495958-0.004889\times x}}{1+e^{5.495958-0.004889\times x}} \tag{14.11}$$

Termenul liber al ecuației de regresie (în engleză *intercept*) este $\beta_0 = 5.495958$; parametrul $\beta_1 = 0.004889$ exprimă scăderea logaritmului natural din raportul de șanse cu 0.004889, atunci când venitul mediul lunar individual crește, în medie, cu o unitate. $e^{\beta_1} = 0.995123 < 1$ reprezintă raportul de șanse, semnificând faptul că, atunci când venitul crește cu o unitate, șansele ca persoana să beneficieze de prestații sociale scad, în medie, la 99,5% (scad cu 0,5%).

Cazul 2. Variabila independentă x este o variabilă binară

Pentru această situație, se consideră un set de observații privind decizia a $n = 30$ persoane de a cumpăra un bun de folosință îndelungată în gospodărie (0 – nu cumpără, 1 - cumpără), în funcție de statutul ocupațional pe piața forței de muncă (0 - neangajat, 1 - angajat).
Atunci când x este o variabilă dihotomică și ia valoarea 0 (neangajat) sau 1 (angajat), observațiile se împart în următoarele categorii/clase:

- categoria persoanelor angajate pe piața forței de muncă pentru care probabilitatea de a decide asupra cumpărării bunului este $\frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$, când $x = 1$ (angajat) și $y = 1$ (decide să cumpere);
- categoria persoanelor neangajate pe piața forței de muncă pentru care probabilitatea de a decide asupra cumpărării bunului este $\frac{e^{\beta_0}}{1+e^{\beta_0}}$, când $x = 0$ (neangajat) și $y = 1$ (decide să cumpere);
- categoria persoanelor angajate pentru care probabilitatea de a decide să nu cumpere bunul este $\frac{1}{1+e^{\beta_0+\beta_1 x}}$, când $x = 1$ (angajat) și $y = 0$ (decide să nu cumpere);
- categoria persoanelor neangajate pentru care probabilitatea de a decide asupra cumpărării bunului este $\frac{1}{1+e^{\beta_0}}$, când $x = 0$ (neangajat) şi $y = 0$ (decide să nu cumpere).

Se aplică funcția `glm()` setului de date:

```
> date <- data.frame(obs = 1:35,
y = c(1,0,1,0,1,1,0,0,1,0,1,0,1,1,1,0,1,1,0,1,1,1,0,0,0,1,1,1,
    1,1,1,1,0,1,1),
x=c(1,1,1,1,1,1,0,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,0,0,0,1,1,1,
    1,1,1,1,0,1,1))
```

Unde:

- `obs` = observația
- `y` = cumpără/nu cumpără
- `x` = statutul ocupațional

Funcția `glm()` returnează următorul rezultat:

```
Call:
glm(formula = y ~ x, family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.07821  -0.00008   0.49518   0.49518   0.49518

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -19.57    3584.67  -0.005    0.996
x              21.60    3584.67   0.006    0.995

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 45.004  on 34  degrees of freedom
Residual deviance: 18.597  on 33  degrees of freedom
AIC: 22.597

Number of Fisher Scoring iterations: 18
```

Conform rezultatelor obținute, modelul de regresie logistică poate fi scris:

$$ln\left(\frac{p}{1-p}\right) = -19.57 + 21.60 \times statut\_ocupational \qquad (14.12)$$

În acest caz, parametrul β_1 exprimă creșterea logaritmului natural din raportul de șanse cu 21,60, atunci când x ia valoarea 1 (persoana este

angajată pe piața forței de muncă).
$e^{\beta_1} = e^{21.6} = 2.4 \times 10^9 > 1$ reprezintă raportul de șanse, semnificând faptul că, atunci când x ia valoarea 1, șansele de succes cresc de 2.4×10^9 ori. Altfel spus, şansele ca o persoană să cumpere bunul cresc foarte mult dacă aceasta este angajată pe piaţa forţei de muncă.

Rolul de predicție al modelului de regresie

Rolul de predicție al modelului rezultă din posibilitatea de estimare a valorilor pe care le poate lua probabilitatea ca evenimentul să aibă loc $(y = 1)$ sub influența valorii pe care o are predictorul (x).
De exemplu, pentru primul caz, în care variabila independentă este continuă, probabilitatea ca o persoană să beneficieze de prestaţii sociale, având un venit mediu lunar de 100 lei este de:

$$p(x) = \frac{e^{5.495958 - 0.004889 \times 100}}{1 + e^{5.495958 - 0.004889 \times 100}} = 0.993354 \qquad (14.13)$$

Altfel spus, există 99,33% șanse ca o persoană cu un venit de 100 lei să beneficieze de ajutor financiar din partea statului.

În R, utilizând datele din exemplul de mai sus, se estimează probabilitatea ca un individ să beneficieze de prestații sociale $(y = 1)$ dacă are un venit de $x = 100$ lei.
Se defineşte un nou data frame, denumit `newdata`.

```
> newdata <- data.frame (x = 100)
```

Pentru a estima probabilitatea ca un individ să beneficieze de prestații sociale, în cazul în care venitul său este de 100 lei, se aplică funcția `predict()` modelului de regresie definit `mylogit`.

```
> prob_estim <-  predict(mylogit, newdata, type = "response")
```

Funcţia `predict()` returnează următorul rezultat:

```
0.993354
```

Pentru a estima probabilitatea ca un individ să beneficieze de prestații sociale, în cazul în care venitul său este de 900 lei, se aplică funcția `predict()` modelului de regresie definit `mylogit`.

```
> newdata <- data.frame (x = 900)
> prob_estim <-  predict(mylogit, newdata, type = "response")
```

Funcţia `predict()` returnează următorul rezultat:

```
0.7495187
```

Așadar, pe baza datelor din setul considerat, pentru un individ cu un venit de 900 lei, probabilitatea estimată ca acesta să fie beneficiar de prestații sociale este 74,95%. Analog, dacă se doreşte estimarea probabilităţii ca o persoană să beneficieze de prestații sociale în funcție de venitul mediu lunar de care dispune, se va observa că probabilitatea scade foarte mult cu creșterea venitului (peste un anumit prag de venit, care poate fi considerat venitul mediu pe economie). De exemplu, la un venit mediu lunar de 2.000 lei, probabilitatea estimată ca persoana să beneficieze de prestații sociale este de 1,36%.

Pentru cel de al doilea exemplu, în care variabila independentă este binară, probabilitatea ca persoana să decidă să cumpere un bun de folosință îndelungată în gospodărie, fiind angajată pe piața forței de muncă, este de:

$$p(x) = \frac{e^{-19,57+21,60}}{1+e^{-19,57+21,60}} \tag{14.14}$$

În R, utilizând datele privind decizia de cumpărare, se estimează probabilitatea ca persoana să cumpere bunul ($y = 1$) dacă este angajat $x = 1$:

```
> newdata <- data.frame(x = 1)
```

Se aplică funcția `predict()` modelului de regresie definit `mylogit`:

```
> prob_estimata <- predict(mylogit, newdata, type = "response")
```

Funcţia `predict()` returnează următorul rezultat:

```
0.8846154
```

Pe baza celor 30 de observații, se estimează o probabilitate de 88,46% ca o persoană care este angajată pe piața forței de muncă să cumpere bunul de folosință îndelungată în gospodărie.

Interpretarea rezultatelor regresiei logistice trebuie făcută cu reținere – modelul de predicție este limitat la variabilele independente utilizate în model, dar pot exista alți predictori (x) care au fost voluntar sau nu, ignorați în model. În exemplul prezentat, probabilitatea de a beneficia

de prestații sociale poate fi influențată și de alte variabile predictive, cum ar fi numărul de membri ai gospodăriei din care individul face parte, nivelul de educație, statutul ocupațional etc. În cazul regresiei logistice simple, astfel de predictori au fost ignorați intenționat, pentru a sublinia existența unei singure variabile independente. Există, însă, și situații în care anumite variabile independente nu sunt introduse în modelul de regresie din motive legate, în principal, de indisponibilitatea datelor sau de faptul că acestea nu se pot cuantifica/măsura.

14.2.4 Testarea modelului de regresie logistică

După estimarea coeficienților de regresie este important să se răspundă la câteva întrebări referitoare la alegerea modelului și la calitatea estimărilor:
- există o influență semnificativă a variabilei independente x asupra variabilei dependente y?
- ce procent din variația variabilei dependente poate fi explicat de modelul de regresie considerat?
- cât de potrivit este modelul de regresie considerat pentru a explica variația variabilei dependente?

Evaluarea modelului de regresie conține două etape:

- în prima etapă se determină dacă există sau nu variabile independente care nu au o influență semnificativă asupra dependentei, prin testarea semnificației coeficienților de regresie asociați acestora;
- în cea de a doua etapă se evaluează – prin intermediul unor măsuri stabilite convențional – dacă modelul este adecvat (în engleză *goodness-of-fit*); această etapă presupune și evaluarea capacității de predicție a modelului.

O măsură des utilizată pentru testarea semnificaţiei coeficienţilor de regresie, precum şi pentru testarea adecvării modelului de regresie logistică, este *-2LL (-2 Log Likelihood)*. Se ştie că verosimilitatea (*likelihood*) este probabilitatea ca valorile variabilei dependente y să poată fi estimate (corect/credibil) pe baza valorilor observate ale predictorilor x.

De ce *-2LL*? Fiind o probabilitate, verosimilitatea ia valori în intervalul $[0,1]$, variațiile acesteia fiind foarte mici. Din acest motiv, verosimilitatea se logaritmează, iar valoarea rezultată variază într-un

interval mult mai larg $(-\infty, 0]$. Această valoare poartă denumirea de *verosimilitate logaritmică* (*LL - Log Likelihood*). Mai mult, pentru ca verosimilitatea să poată fi mai ușor interpretată, s-a stabilit prin convenție ca LL să se înmulțească cu -2, cu scopul de a fi o valoare mai mare și pozitivă.

Așadar, în testarea modelelor de regresie logistică se utilizează valoarea *-2LL* cu o largă aplicabilitate, fiind similară cu suma pătratelor erorilor din regresia liniară. *-2LL* se compară pentru mai multe variante de modele de regresie. Se spunem că modelul este mai potrivit, cu cât valoarea *-2LL* este mai mică (condiția de verosimilitate maximă implică o condiție de minim pentru valoarea *-2LL*).

Modelele de regresie pot fi construite în mai multe variante, prin introducerea în model, rând pe rând a variabilelor explicative. Problema de interes este de a testa dacă introducerea unei noi variabile independente în model mărește sau nu verosimilitatea modelului.

În cazul regresiei logistice simple, se compară două modele: un model restrâns care nu conține variabila independentă (conține numai termenul liber β_0, *intercept only* – denumirea modelului în limba engleză) şi un model extins care conține variabila independentă (în limba engleză *intercept and covariates*). Diferența între valorile *-2LL* corespunzătoare celor două modele arată dacă verosimilitatea modelului, în ansamblu, crește sau scade la introducerea variabilei independente în model.

Testarea se face cu ajutorul testului χ^2 (în limba engleză *Chi-Square*), deoarece *diferența între valorile* $-2LL$ [4], calculată pentru oricare două câte două modele posibile, este o variabilă aleatoare care urmează o distribuție χ^2. Testul χ^2 poartă și denumirea de testul Wald sau testul raportului de verosimilități (*Likelihood Ratio Test*):

$$\chi^2 = -2LL_0 - (-2LL_M) = -2ln\left(\frac{LL_0}{LL_M}\right) \tag{14.15}$$

unde:
$-2LL_0$ reprezintă verosimilitatea logaritmică a modelului care nu conține variabila independentă; aceasta este denumită și verosimilitate inițială, iar modelul de regresie este considerat un model restrâns, utilizat ca bază de comparație pentru alte modele posibile care conțin una sau mai multe variabile independente;
$-2LL_M$ reprezintă verosimilitatea logaritmică a modelului care conține variabila independentă, (modelul extins).

[4] diferența între valorile $-2LL$ poate fi privită ca o abatere care, ca și în cazul sumei pătratelor erorilor din regresia liniară, trebuie minimizată

Un model adecvat (în engleză *fitted model*) este acela care îndeplinește criteriul verosimilității maxime, adică χ^2 tinde la zero. Altfel spus, diferența dintre cele două verosimilități este minimă. Un model perfect este acela in care modelele au aceeași verosimilitate $LL_0 = LL_M$.

Este important de precizat că cel mai bun model de regresie nu este acela care explică totul, ci acela care are cea mai mare putere de explicație cu cel mai mic număr de variabile independente. Ideal ar fi să se găsească un număr mic de variabile independente care să fie puternic corelate cu variabila dependentă, iar explicația variației să fie făcută doar pe baza acestora, chiar dacă mai rămâne o cantitate (preferabil mică) de eroare.

În cazul regresiei liniare, un model adecvat este acela în care suma erorilor cumulate este cât mai aproape de suma erorilor explicate de regresie, astfel încât suma valorilor reziduale[5] este cât mai mică ($SST = SSR + SSE$).

Verosimilitatea logaritmică $-2LL_0$ poate fi privită similar cu variația totală din testul ANOVA, în cazul regresiei liniare (SST[6] – *Sum of Squares Total*), iar $-2LL_M$ este similar cu variația explicată de modelul regresie (SSR[7] – *Sum of Squares Regression*).

Exemplu: În continuare, se evaluează modelul de regresie logistică pentru exemplul în care variabila dependentă (`y`) reprezintă apartenența/non-apartenența unui individ la categoria beneficiari de prestații sociale, iar `x` este variabilă numerică, reprezentând venitul mediu lunar individual, exprimat în unități monetare:

$$ln\left(\frac{p}{1-p}\right) = 5.495958 - 0.004889 \times venitul\_mediu\_lunar \qquad (14.16)$$

Ecuaţia de regresie logistică estimează, pe baza unui număr de observaţii (n=35), probabilitatea ca individul să fie beneficiar de prestaţii sociale (p) pentru orice valoare pe care o poate lua variabila independentă (venit mediul lunar).

[5]suma pătratelor valorilor reziduale se calculează după formula: $SEE = \sum_i (y_i - \hat{y}_i)^2$, unde y_i sunt valorile observate ale variabilei dependente, iar $\hat{y}_i$ sunt valorile estimate pe baza regresiei ale variabilei dependente

[6]suma pătratelor erorilor cumulate se calculează după formula: $SST = \sum_i (y_i - \overline{y})^2$, unde y_i sunt valorile observate ale variabilei dependente, iar $\overline{y}$ este media acesteia

[7]suma pătratelor erorilor explicate de regresie se calculează după formula: $SSR = \sum_i (\hat{y}_i - \overline{y})^2$, unde $\hat{y}_i$ sunt valorile estimate pe baza regresiei ale variabilei dependente, iar $\overline{y}$ este media acesteia

Pentru prima etapă de evaluare a modelului de regresie se verifică dacă variabila independentă (venit mediu lunar) are sau nu o influenţă semnificativă asupra variabilei dependente (beneficiar/non-beneficiar). Pentru aceasta, se verifică dacă coeficientul de regresie estimat ($\beta_1 = -0.004889$) este semnificativ statistic.

Aplicând în R funcția `glm()` setului de date considerat, aceasta returnează următorul rezultat:

```
Call:
> glm(formula = y ~ x, family = "binomial", data = date_logit)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.0257  -0.1424   0.1487   0.3452   0.9198

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  5.495958   1.782551   3.083  0.00205 **
x           -0.004889   0.001684  -2.903  0.00370 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 45.004  on 34  degrees of freedom
Residual deviance: 15.782  on 33  degrees of freedom
AIC: 19.782

Number of Fisher Scoring iterations: 6
```

Conform rezultatului obținut, coeficientul este semnificativ statistic (semnificativ diferit de zero, conform ipotezei de nul), cu un nivel de semnificație de 1% (‘**’ 0.01). Altfel spus, valoarea coeficientului de regresie este estimată cu o probabilitate asociată de $0.00370 < 0.01$ (încredere de 99%). În această situație, se poate afirma că venitul mediu lunar influențează probabilitatea ca individul să beneficieze de prestații sociale, astfel încât prin creșterea cu o unitate a venitului său, șansele lui să fie beneficiar scad, în medie, la 99.5% ($e^{\beta_1} = 0.995123$).

În general, nivelul maxim de semnificație acceptat este 5% (‘*’ 0.05). Semnificația coeficienților de regresie o putem judeca după numărul de steluțe, a căror interpretare o găsim în codurile de semnificație generate în tabloul de rezultate returnate de funcția `glm()`.

Calculul limitelor intervalelor de încredere cu care au fost estimați parametrii ecuației de regresie se realizează cu funcția `confint()`:

```
> confint(mylogit, level = 0.95)
                    2.5 %      97.5 %
(Intercept)  2.802837013 10.20701174
x           -0.009456652 -0.00238618
```

Așadar, intervalul de încredere pentru β_1, pentru un nivel de semnificație de 5% este $[-0.009456652, -0.0238618]$.

Pentru cea de a doua etapă de evaluare, care constă în testarea modelului de regresie logistică - în ansamblu, procedură care arată dacă modelul este adecvat - se utilizează testul χ^2 care este inclus în R prin funcția `anova()`.

```
> anova(mylogit, test = "Chisq")
Analysis of Deviance Table

Model: binomial, link: logit

Response: y

Terms added sequentially (first to last)


     Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                    34     45.004
x     1   29.222        33     15.782 6.453e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Funcția compară, pe baza testului raportului de verosimilități, cele două modele de regresie logistică (modelul restrâns și modelul extins):

```
glm(formula = y ~ 1, family = "binomial")
glm(formula = y ~ x, family = "binomial")
```

În condițiile în care probabilitatea este mai mică decât cea pentru care modelul extins se acceptă cu încredere de 95% ($Pr(> Chi) = 6.453e - 08 < 0.05$), se poate afirma că testul χ^2 confirmă faptul că venitul mediu lunar influențează semnificativ șansele ca o

persoană să fie sau nu beneficiar de prestații sociale. Valoarea χ^2, calculată ca diferență între valorile $-2LL$ corespunzătoare celor două modele, este returnată în tabloul de rezultat al funcției `anova()` sub denumirea *deviance* și este egală cu 29.22[8].

Numărul de grade de libertate (df) – reprezintă o măsură a numărului de unități de informație independente pe baza căruia este estimat un parametru. Pentru determinarea valorii χ^2 pentru un model de regresie cu k variabile independente (x_k), numărul gradelor de libertate ($df = n - k - 1$) este egal cu numărul observațiilor (n) minus numărul de parametri adiționali[9] ($k + 1$). În cazul unei ecuații de regresie logistică simplă, χ^2 este calculat cu $n - 2$ grade de libertate. Pentru exemplul dat, χ^2 este determinat cu 33 grade de libertate.

Alte măsuri pentru evaluarea modelului de regresie logistică

În regresia logistică nu există un indicator absolut similar raportului de determinare R^2 din regresia liniară. S-au dezvoltat însă indicatori similari. Astfel, există R^2 al lui *Cox & Snell*, o măsură care mai este denumită pseudo-R^2 definit prin relația:

$$R^2 = 1 - \left[\frac{-2LL_0}{-2LL_M}\right]^{2/n} \tag{14.17}$$

Unde LL_0 este logaritm din maximul funcției de verosimilitate pentru modelul restrâns (care conține doar termenul liber), iar LL_M este logaritm din maximul funcției de verosimilitate pentru modelul cu variabile independente incluse, iar n reprezintă numărul de observații.

Valoarea R^2 al lui *Cox & Snell* nu ajunge, de obicei, la 1 (în regresia liniară R^2 variază între 0 și 1), astfel încât este dificil de interpretat. Din acest motiv acest R^2 a fost introdusă de *Nagelkerke* o modificare prin care R^2 atinge valoarea 1. R^2 al lui *Nagelkerke* este cel mai folosit dintre toate măsurile de testare a adecvării modelului de regresie logistică, având

[8]O valoare calculată χ^2 mai mare decât valoarea tabelară a distribuției χ^2, pentru un anumit număr de grade de libertate și un anumit nivel de încredere, se traduce prin afirmația că modelul de regresie logistică extins (modelul care conține variabila independentă) este mai adecvat decât modelul restrâns (modelul care conține numai termenul liber). Valorile distribuției se găsesc în tabele precalculate, pentru diferite grade de libertate și diferite niveluri de încredere.

[9]Parametrii ecuației de regresie și termenul liber.

următoarea formulă de calcul:

$$R^2 = \frac{1 - \left[\frac{-2LL_0}{-2LL_M}\right]^{2/n}}{1 - (2LL_0)^{2/n}} \tag{14.18}$$

Alte criterii de specificare pentru evaluarea acurateții ajustării modelului de regresie sunt AIC (*Akaike's Information Criterion*) și BIC (*Bayesian Information Criterion*), definiți prin următoarele formule de calcul:

$$AIC = -2LL_k + 2k \tag{14.19}$$

$$BIC = -2LL_k + 2 \times log(n) \tag{14.20}$$

unde k este numărul de parametri estimați, iar `n` reprezintă numărul de observații. BIC mai este cunoscut și drept criteriul Schwartz (care l-a argumentat). Ambele criterii compară buna potrivire a modelului cu un alt model de referință, astfel încât să se obțină un model cu un număr minim de parametri. Vor fi preferate modelele pentru care criteriile (AIC sau BIC) au valori mai mici. Altfel spus, cea mai mică valoare a criteriului, obținută la fiecare model analizat, indică modelul care se potrivește cel mai bine la datele analizate (în engleză *the best fit model*). Ambii indicatori necesită condiția ca erorile (valorile reziduale) să fie normal distribuite.

14.3 Regresia logistică multiplă

Regresia logistică multiplă modelează relația dintre o mulțime de variabile independente x_i (categoriale, continue) și o variabilă dependentă dihotomică (nominală, binară) y. Modelul de regresie logistică multiplă este dat de formula:

$$ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k \tag{14.21}$$

sau

$$logit(p) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k \tag{14.22}$$

Ca și în cazul modelului de regresie logistică simplă, modelul de regresie multiplă se mai poate scrie:

$$\frac{p}{1-p} = e^{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k} \tag{14.23}$$

Sau, exprimat prin șansele de succes:

$$\Omega = \frac{p}{1-p} \tag{14.24}$$

Modelul poate fi scris sub forma:

$$\Omega = e^{\beta_0 + \sum_k \beta_k x_k} \tag{14.25}$$

Unde p este probabilitatea ca y să ia valoarea 1, adică fenomenul să înregistreze un succes, iar $1-p$ este probabilitatea ca y să ia valoarea 0, adică să se înregistreze un eșec.
β_0,β_1, ..., β_k sunt parametrii ecuației de regresie; k – numărul de observații.

Transformarea valorii logit în probabilități.

$$p = \frac{e^{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k}}{1 + e^{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k}} \tag{14.26}$$

Raportul de șanse, care compară șansele a două grupuri de populație caracterizate de valori diferite înregistrate de variabila independentă (x_j), în condițiile în care toate celelalte variabile independente rămân neschimbate ($x_i = \text{const.}, i \neq j$), este exprimat prin relaţia:

$$OR = \frac{\Omega_{(x_j+1)}}{\Omega_{(x_j)}} = \frac{e^{\beta_0 + \beta_j (x_j+1)}}{e^{\beta_0 + \beta_j x_j}} = \frac{e^{\beta_0} \times e^{\beta_j x_j} \times e^{\beta_j}}{e^{\beta_0} \times e^{\beta_j x_j}} = e^{\beta_j} \tag{14.27}$$

Așadar, e^{β_j} reprezintă raportul de șanse care arată ce se întâmplă atunci când x_j se modifică cu o unitate, iar celelalte variabile independente neavând nicio influență asupra modificării variabilei rezultative. Ca și în cazul regresiei logistice simple, există trei situații posibile:

- $e^{\beta_j} > 1$, situație în care creșterea cu o unitate a lui x_j conduce la sporirea șanselor de succes ale grupului de populație caracterizat de valoarea $x_j + 1$ (de exemplu, dacă $e^{\beta_j} = 1,5$ spunem că șansele de succes cresc, în medie, de 1,5 ori, sau cu 50%, atunci când x_j crește cu o unitate);
- $e^{\beta_j} < 1$, situație în care creșterea cu o unitate a lui x_j conduce la reducerea șanselor de succes ale grupului de populație caracterizat de valoarea $x_j + 1$ (de exemplu, dacă $e^{\beta_j} = 0,5$ spunem că șansele de succes scad, în medie, la jumătate, sau cu 50%, atunci când x_j crește cu o unitate);

- $e^{\beta_j} = 1$, situație în care orice valoare ar lua x_j, aceasta nu produce nici un efect asupra șanselor de succes ($e^{\beta_j} = 1$, dacă $\beta_j = 0$), adică grupurile de populație caracterizate de aceleași valori ale variabilei independente au șanse de succes egale.

Influența variabilelor explicative se analizează independent una de cealaltă, în sensul că variația uneia dintre acestea conduce la variația variabilei dependente, în condițiile în care toate celelalte variabile independente rămân constante.
Exemplu:
Se consideră exemplul în care o firmă își propune selectarea unor persoane care să lucreze într-un anumit departament. Se cunosc date privind vechimea în muncă (x_1) și nivelul de educație (x_2), pentru 100 de persoane din firmă, unele dintre acestea lucrând deja în departamentul pentru care se fac noi angajări. Se dorește ca, în funcție de datele disponibile, să fie selectate persoane cu aceleași caracteristici/profil ca cele care lucrează deja, după "modelul" care este considerat "de succes". Variabila rezultat (y) este binară (1 - persoana lucrează în departament, 0 - nu lucrează în departament). Vechimea în muncă (x_1) se poate situa în două intervale de valori ($x_1 = 1$, dacă persoanele au o vechime în muncă de 3-5 ani și $x_1 = 2$, dacă persoanele au o vechime în muncă de 6-10 ani). Nivelul de educație (x_2) poate avea, de asemenea, două valori ($x_2 = 1$, dacă persoanele au nivel mediu de educație şi $x_2 = 2$, dacă au absolvit învățământ superior).
Modelul de regresie logistică conține două variabile factoriale este dat de formula:

$$ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{14.28}$$

Unde p – este probabilitatea ca o persoană cu o anumită vechime în muncă și cu un anumit nivel de educație să lucreze în departament (probabilitatea de succes). Care sunt șansele de succes ale unei persoane, pentru ca aceasta să fie angajată în departament?

Se aplică funcția `glm()` care returnează următoarele rezultate:

```
Call:
glm(formula = y ~ x1 + x2, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8416  -0.1278   0.1961   0.1961   3.1022
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -6.202      2.251  -2.755  0.00587 **
x1            -2.449      1.123  -2.181  0.02916 *
x2             6.297      1.320   4.770 1.84e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 134.602  on 99  degrees of freedom
Residual deviance:  37.511  on 97  degrees of freedom
AIC: 43.511

Number of Fisher Scoring iterations: 7
```

Așadar, parametrii estimați pe baza datelor observate sunt: $\beta_0 = -6.202$, $\beta_1 = -2.449$, $\beta_2 = 6.297$.

- Pentru variabila vechime în muncă, $e^{\beta_1} = e^{-2.449} = 0.08636 < 1$; suntem în situația în care creșterea cu o unitate a lui x_1 (de la $x_1 = 1$ - grupa de vechime 3-5 ani, la $x_1 = 2$ - grupa de vechime 6-10 ani), în condițiile în care persoanele au același nivel de educație, conduce la reducerea șanselor de succes a persoanelor pentru a fi angajate în departament (spunem că șansele de succes scad, în medie, la 8.63%, atunci când persoana are vechime în muncă mai mare de 5 ani); această afirmație este susținută de faptul că parametrul estimat β_1 este semnificativ statistic, în sensul că nivelul de încredere ($p = 0.02916$) se află sub pragul de 0.05.
- Pentru variabila nivel de educație, $e^{\beta_2} = e^{6.297} = 542.9406 > 1$; suntem în situația în care creșterea cu o unitate a lui x_2 (de la $x_2 = 1$ - nivel mediu de educație, la $x_2 = 2$ - nivel superior de educație, în condițiile în care persoanele au aceeași vechime în muncă, conduce la creșterea șanselor de angajare (spunem că șansele de succes cresc, în medie, de 543 ori, atunci când persoana are studii superioare); parametrul estimat β_2 este semnificativ statistic.

Dacă toate cele 100 persoane ar avea același nivel de educație și aceeași vechime în muncă ($x_1 = \text{const.}$) şi ($x_2 = \text{const.}$), acestea ar avea șanse egale de a fi selectate pentru a fi angajate. Aceasta ar fi situația în care orice valoare ar avea variabilele independente, aceasta nu produce nici un efect asupra șanselor de succes ($e^{\beta_j} = 1$, dacă $\beta_j = 0$).

14.4 Regresia logistică multinomială

Modelul de regresie logistică multinomială este o generalizare a regresiei logistice binomiale; în acest caz, variabila dependentă categorială are mai mult de două categorii de răspuns. Modelul este cunoscut în econometrie și ca regresia logistică politomică (*polytomous logistic regression*), model de alegere discretă (*discrete choice model*) sau regresia logistică cu răspuns multiplu (*logit model for multinomial responses*).

Se notează cu Y variabila dependentă cu J categorii de răspuns.

$$Y_i = \begin{cases} categoria - 1 \\ categoria - 2 \\ categoria - 3 \\ \quad ... \\ categoria - J \end{cases} \tag{14.29}$$

Pentru observația i, categoriile de răspuns se înregistrează cu probabilitățile:

$$p_i = \begin{cases} p_{i1} \\ p_{i2} \\ p_{i3} \\ ... \\ p_{ij} \\ ... \\ p_{iJ} \end{cases} \tag{14.30}$$

$j = 1, 2, ..., J$
Dacă $J > 2$, vor fi $J - 1$ ecuații de regresie logistică, categoria J fiind considerată *categorie de referință*:

$$ln\left(\frac{p_{i1}}{1 - p_{iJ}}\right) = \beta_{10} + \beta_{11} \times x_{i1} + \beta_{12} \times x_{i2} + ... + \beta_{1k} \times x_{ik} = \beta_1' \times x_i \tag{14.31}$$

$$ln\left(\frac{p_{i2}}{1 - p_{iJ}}\right) = \beta_{20} + \beta_{21} \times x_{i1} + \beta_{22} \times x_{i2} + ... + \beta_{2k} \times x_{ik} = \beta_2' \times x_i \tag{14.32}$$

.....

$$ln\left(\frac{p_{ij}}{1-p_{iJ}}\right) = \beta_{j0} + \beta_{j1} \times x_{i1} + \beta_{j2} \times x_{i2} + ... + \beta_{jk} \times x_{ik} = \beta'_j \times x_i \tag{14.33}$$

.....

$$ln\left(\frac{p_{iJ}}{1-p_{iJ}}\right) = \beta_{J0} + \beta_{J1} \times x_{i1} + \beta_{J2} \times x_{i2} + ... + \beta_{Jk} \times x_{ik} = \beta'_J \times x_i \tag{14.34}$$

Fiecare dintre aceste ecuații sunt caracterizate de propriile valori ale parametrilor de regresie β_j. În general, putem considera o singură ecuație de regresie, corespunzătoare observației i și categoriei de răspuns j:

$$ln\left(\frac{p_{ij}}{1-p_{iJ}}\right) = \beta_{j0} + \beta_{j1} \times x_{i1} + \beta_{j2} \times x_{i2} + ... + \beta_{jk} \times x_{ik} = \beta'_j \times x_i \tag{14.35}$$

Modelul poate fi scris sub forma șansei de succes:

$$\Omega = \frac{p_{ij}}{1-p_{ij}} = e^{\beta_{j0}+\beta_{j1}\times x_{i1}+\beta_{j2}\times x_{i2}+...+\beta_{jk}\times x_{ik}} \tag{14.36}$$

Pentru a scrie transformările valorilor logit în probabilități, se consideră următoarele cazuri:

Cazul 1. $J = 2 (j = 1, 2)$ Regresia logistică binomială

$$ln\left(\frac{p_{i1}}{1-p_{i1}}\right) = \beta_{10} + \beta_{11} \times x_{i1} + \beta_{12} \times x_{i2} + ... + \beta_{1k} \times x_{ik} = \beta'_1 \times x_i \tag{14.37}$$

Probabilitatea ca observația i să înregistreze prima din cele două categorii de răspuns, evenimentul fiind considerat un succes, este:

$$p_{i1} = \frac{e^{\beta_{10}+\beta_{11}\times x_{i1}+\beta_{12}\times x_{i2}+...+\beta_{1k}\times x_{ik}}}{1+e^{\beta_{10}+\beta_{11}\times x_{i1}+\beta_{12}\times x_{i2}+...+\beta_{1k}\times x_{ik}}} = \frac{e^{\beta'_1\times x_i}}{1+e^{\beta'_1\times x_i}} \tag{14.38}$$

Probabilitatea ca observația i să înregistreze cea de a doua categorie de răspuns, evenimentul fiind considerat un eșec, este:

$$p_{i2} = 1 - p_{i1} = 1 - \frac{e^{\beta_{10}+\beta_{11}\times x_{i1}+\beta_{12}\times x_{i2}+...+\beta_{1k}\times x_{ik}}}{1+e^{\beta_{10}+\beta_{11}\times x_{i1}+\beta_{12}\times x_{i2}+...+\beta_{1k}\times x_{ik}}} = \frac{1}{1+e^{\beta_1'\times x_i}} \tag{14.39}$$

Șansele de succes se reduc la cazul regresiei logistice binomiale multiple:

$$\Omega = \frac{p_{i1}}{1-p_{i1}} = e^{\beta_{10}+\beta_{11}\times x_{i1}+\beta_{12}\times x_{i2}+...+\beta_{1k}\times x_{ik}} = e^{\beta_1'\times x_i} \tag{14.40}$$

Dacă se consideră cazul regresiei logistice binomiale simple, caracterizată de o singură variabilă independentă, șansele de succes devin:

$$\Omega = \frac{p}{1-p} = e^{\beta_0+\beta_1\times x_i} \tag{14.41}$$

Cazul 2. $J > 2$
Regresia logistică multinomială

Generalizarea probabilității de succes, corespunzătoare observației i și categoriei de răspuns $j < J$:

$$p_{ij} = \frac{e^{\beta_J'\times x_i}}{1+\sum_{j=1}^{J-1} e^{\beta_j'\times x_i}} \tag{14.42}$$

când $j < J$.

Probabilitatea de eșec, corespunzătoare observației i și categoriei de răspuns $j = J$:

$$p_{iJ} = \frac{1}{1+\sum_{j=1}^{J-1} e^{\beta_j'\times x_i}} \tag{14.43}$$

când $j = J$.

Așadar, se compară șansele categoriei j să înregistreze un succes, față de categoria J, pentru un set de valori ale variabilelor independente $(x_{i1},...,x_{ik})$. Pentru a interpreta rezultatele modelului de regresie logistică multinomială, se determină raportul de șanse. Acesta compară șansele a două grupuri de populație caracterizate de valori diferite

înregistrate de o variabilă independentă (x_{ik} și $x_{ik}+1$), în condițiile în care toate celelalte variabile independente sunt identice:

$$OR = \frac{\Omega_{(x_{ik}+1)}}{\Omega_{(x_{ik})}} = \frac{e^{\beta_{j0}+\beta_{j1}\times x_{i1}+\beta_{j2}\times x_{i2}+...+\beta_{jk}\times(x_{ik}+1)}}{e^{\beta_{j0}+\beta_{j1}\times x_{i1}+\beta_{j2}\times x_{i2}+...+\beta_{jk}\times x_{ik}}} = e^{\beta_{jk}} \qquad (14.44)$$

β_{jk} - reprezintă logaritmul natural din raportul de șanse pentru categoria de răspuns j versus J, în condițiile unei creșteri cu o unitate a variabilei independente x_{ik}. Categoriile de răspuns pot fi *nominale* (de exemplu: statutul ocupațional – angajat, șomer, pensionar) sau *ordinale* (de exemplu: nivelul de educație - scăzut, mediu, superior). Pentru a facilita înțelegerea regresiei logistice multinomiale, se va considera un experiment în care răspunsul este măsurat prin mai multe categorii de răspuns.

Exemplu: Statutul ocupațional al unei persoane poate fi influențat de vârstă, sex, naționalitate, nivelul educațional.

Se importă setul de date care cuprinde 32.937 de persoane, pentru care se cunosc următoarele date:

- grupa de vârstă
- sexul
- naționalitatea
- nivelul de educație

```
> Date_multinom <- read.table(file.choose(), header =  TRUE)
> head(Date_multinom)

     STOCUP grv SEX   NAT   NIVE
1 pensionar >64   1 Roman  mediu
2 pensionar >64   2 Roman  mediu
3 pensionar >64   1 Roman  mediu
4 pensionar >64   1 Roman  mediu
5 pensionar >64   2 Roman scazut
6 pensionar >64   1 Roman  mediu

> tail(Date_multinom)

         STOCUP   grv SEX   NAT   NIVE
32932 pensionar 45-54   1 Roman scazut
32933 pensionar   >64   1 Roman scazut
```

```
32934 pensionar   >64   1 Roman scazut
32935 pensionar   >64   2 Roman scazut
32936      Elev   <25   1 Roman scazut
32937      Elev   <25   1 Roman scazut
```

Descrierea variabilelor:

STOCUP - variabila dependentă (y_k) - statutul ocupațional, care are următoarele categorii de răspuns: salariat[10], casnică, pensionar, elev, student.

```
> levels(Date_multinom$STOCUP)
[1] "aSalariat" "Casnica"   "Elev"  "pensionar" "Student"
```

grv – variabilă factorială (x_{1k}) – grupa de vârstă, cu următoarele categorii:
$<$ 25 ani
25-34 ani
35-44 ani
45-54 ani
55-64 ani
$>$ 64 ani

```
> levels(Date_multinom$grv)
[1] "<25"   ">64"   "25-34" "35-44" "45-54" "55-64"
```

SEX – variabilă factorială (x_{2k}) – sexul: 1- masculin, 2- feminin.

```
> levels(as.factor(Date_multinom$SEX))
[1] "1" "2"
```

NAT – variabilă factorială (x_{3k}) – naționalitatea: român, maghiar, rom, german.

```
> levels(Date_multinom$NAT)
[1] "German"  "Maghiar" "Roman" "Rom"
```

NIVE – variabilă factorială (x_{4k}) – nivelul de educaţie: scăzut (primar şi gimnazial), mediu (liceal şi postliceal), superior (universitar şi postuniversitar).

[10]În dataframe s-a introdus denumirea categoriei salariat ca fiind aSalariat, deoarece funcția `multinom()` care returnează coeficienții de regresie ordonează alfabetic categoriile variabilelor, categoria de referință fiind considerată prima categorie astfel ordonată. Intenția este de a calcula probabilitățile de a avea un alt statut ocupațional decât cel de salariat, funcție de profilul dat de variabilele independente.

```
> levels(Date_multinom$NIVE)
[1] "mediu"    "scazut"   "superior"
```

Calculăm coeficienții de regresie pe baza funcției `multinom()`, furnizată în pachetul de date `nnet`[11].

```
> Regresia_multinom <- multinom( STOCUP ~  NAT + NIVE,
data = Date_multinom)

initial  value 53010.056522
iter  10 value 20902.541315
iter  20 value 20127.512540
iter  30 value 19878.836261
iter  40 value 19872.668464
iter  50 value 19872.549771
iter  50 value 19872.549770
final  value 19872.549770
```

Prima rulare a modelului include valoarea finală, rezultată în urma unui număr de iterații, a $-LL$ (19872.549770). Se cunoaște că, în testarea modelelor de regresie logistică se utilizează valoarea $-2LL$ cu o largă aplicabilitate. $-2LL$ se compară pentru mai multe variante de modele de regresie. Se spune că modelul este mai potrivit, cu cât valoarea $-2LL$ este mai mică (condiția de verosimilitate maximă implică o condiție de minim pentru valoarea $-2LL$).

Funcția `summary()` returnează în R următorul rezultat:

```
> summary(Regresia_multinom)

Call:
multinom(formula = STOCUP ~ NAT + NIVE, data = Date_multinom)

Coefficients:
          (Intercept) NATMaghiar  NATRoman   NATRom
Casnica     -8.749036   8.860484  8.453346  10.373517
Elev         6.108923  -7.949381 -7.748737  -7.785677
pensionar   11.837338  -7.912068 -8.019252  -9.568915
Student      9.930317  -9.327672 -9.554188 -19.197345
           NIVEscazut NIVEsuperior
Casnica     0.6634735   -2.9176685
```

[11]Creat pentru rezolvarea modelelor de regresie multinomială

```
Elev         5.7622362  -17.9037431
pensionar    1.1815414    0.1005348
Student    -13.1616631   -1.3704095

Std. Errors:
           (Intercept) NATMaghiar  NATRoman     NATRom
Casnica      0.1783185  0.2556346 0.1578634 3.694178e-01
Elev         0.3603144  0.3283281 0.2875170 4.637509e-01
pensionar    0.2601339  0.2984994 0.2560432 4.472484e-01
Student      0.5723380  0.6385659 0.5758090 2.079786e-05
             NIVEscazut NIVEsuperior
Casnica    1.932547e-01 1.029483e+00
Elev       2.725434e-01 6.198365e-09
pensionar  1.374942e-01 2.312425e-01
Student    1.043614e-05 4.130807e-01

Residual Deviance: 39745.1
AIC: 39793.1
```

La primul grup de coeficienți se observă următoarele:
- se compară fiecare categorie a variabilei rezultat cu categoria salariat (de referință): casnică vs. salariat, elev vs. salariat, pensionar vs. salariat, student vs. salariat;
Ecuația modelului, pentru j=casnică, poate fi scrisă:

$$ln\left(\frac{p_{casnica}}{1-p_{salariat}}\right) = \beta_{casnica0} + \beta_{casnicaNATMaghiar} \times x_{NATMaghiar} + \\ + \beta_{casnicaNATRoman} \times x_{NATRoman} + \beta_{casnicaNATRom} \times x_{NATRom} + \\ + \beta_{casnicaNIVEscazut} \times x_{NIVEscazut} + \beta_{casnicaNIVEsuperior} \times x_{NIVEsuperior}$$

Creșterea cu o unitate a variabilei NIVE (trecerea persoanei de la nivelul mediu de educație la nivelul superior) este asociată cu scăderea cu 2.9176685 a logaritmului din șansele de succes de a fi casnică vs. salariat ($\beta_{casnicaNIVEsuperior}$).

Analog, se scriu și ecuațiile pentru j=elev, pensionar, student.

$$ln\left(\frac{p_{elev}}{1-p_{salariat}}\right) = \beta_{elev0} + \beta_{elevNATMaghiar} \times x_{NATMaghiar} + \\ + \beta_{elevNATRoman} \times x_{NATRoman} + \beta_{elevNATRom} \times x_{NATTRom} + \\ + \beta_{elevNIVEscazut} \times x_{NIVEscazut} + \beta_{elevNIVEsuperior} \times x_{NIVEsuperior}$$

$$ln\left(\frac{p_{pensionar}}{1-p_{salariat}}\right) = \beta_{pensionar0} + \beta_{pensionarNATMaghiar} \times x_{NATMaghiar} + \\ + \beta_{pensionarNATRoman} \times x_{NATRoman} + \beta_{pensionarNATRom} \times x_{NATRom} + \\ + \beta_{pensionarNIVEscazut} \times x_{NIVEscazut} + \beta_{pensionarNIVEsuperior} \times x_{NIVEsuperior}$$

$$ln\left(\frac{p_{student}}{1-p_{salariat}}\right) = \beta_{student0} + \beta_{studentNATMaghiar} \times x_{NATMaghiar} + \\ + \beta_{studentNATRoman} \times x_{NATRoman} + \beta_{studentNATRom} \times x_{NATRom} + \\ + \beta_{studentNIVEscazut} \times x_{NIVEscazut} + \beta_{studentNIVEsuperior} \times x_{NIVEsuperior}$$

Creșterea cu o unitate a variabilei `NIVE` (trecerea persoanei de la nivelul mediu de educație la nivelul superior) este asociată cu scăderea cu 1.3704095 a logaritmului din șansele de succes de a fi student vs. salariat ($\beta_{studentNIVEsuperior}$). Altfel spus, raportul de șanse student vs. salariat fiind $e^{\beta_{studentNIVEsuperior}} = 0.2540029$. Pentru a înțelege mai bine modelul, se estimează probabilitățile asociate fiecărei categorii de răspuns, pentru variabilele independente, utilizând funcția `fitted()`.

```
> pp <- fitted(Regresia_multinom)
> head(pp)
```

```
   aSalariat     Casnica        Elev pensionar      Student
1 0.02044502 0.015211468 0.003966664 0.9305959 2.978093e-02
2 0.02044502 0.015211468 0.003966664 0.9305959 2.978093e-02
3 0.02044502 0.015211468 0.003966664 0.9305959 2.978093e-02
4 0.02044502 0.015211468 0.003966664 0.9305959 2.978093e-02
5 0.00470564 0.006797438 0.290378206 0.6981187 1.318050e-08
6 0.02044502 0.015211468 0.003966664 0.9305959 2.978093e-02
```

Pentru a putea estima modificările variabilei dependente, asociate influenței uneia din cele două variabile, se va crea un set de date în care una dintre variabile va fi menținută la un nivel constant (se consideră că toate persoanele din eșantion au același nivel de educație[12]). Astfel, se consideră că o persoană poate avea un anumit statut ocupațional sub influența unui singur factor: naționalitatea.

[12]Nivelul de educație mediu – pentru a fi calculat, variabila categorială `NIVE` trebuie transformată în variabilă numerică

```
> class(Date_multinom$NIVE) <- "numeric"
> dataNAT <- data.frame(NAT = c("Roman", "Maghiar", "Rom",
"German"), NIVE = mean(Date_multinom$NIVE))
> head(dataNAT)
     NAT     NIVE
1   Roman 1.896864
2 Maghiar 1.896864
3   Rom 1.896864
4  German 1.896864
```

Se estimează probabilitățile asociate fiecărei categorii ale statutului ocupațional (salariat, casnică, elev, pensionar, student), cu ajutorul funcției `predict()`.

```
> Regresia_multinom <- multinom( STOCUP ~  NAT + NIVE,
data = Date_multinom)
> predict(Regresia_multinom, newdata = dataNAT, "probs")

  aSalariat      Casnica      Elev pensionar      Student
1 7.022915e-03 7.327580e-03 0.2332427 0.7514264 9.804386e-04
2 6.386280e-03 1.042621e-02 0.1880070 0.7940681 1.112418e-03
3 1.174666e-02 1.100228e-01 0.5532080 0.3250226 1.307448e-14
4 2.695821e-07 1.919673e-11 0.2018602 0.7937035 4.436075e-03
```

O altă cale de a înțelege mai bine rezultatele modelului este de a vedea care sunt probabilitățile estimate asociate fiecărei categorii ale variabilei NIVE din cadrul fiecărui nivel de educație NAT. Se construiește un nou dataframe, apoi se atașează probabilitățile estimate pentru fiecare categorie a celor două variabile independente, pentru toate observațiile din setul de date:

```
> dNIVE <- data.frame(NAT = rep(c("Roman", "Maghiar", "Rom",
"German"), each = 3), NIVE = rep(c(1:3),4))
> pp.NIVE <- cbind(dNIVE, predict(Regresia_multinom,
newdata = dNIVE, type = "probs", se = TRUE))

> head(pp.NIVE)
      NAT NIVE   aSalariat     Casnica       Elev
1   Roman    1 0.015760418 0.015732183 0.12143089
2   Roman    2 0.006341103 0.006649950 0.24912891
3   Roman    3 0.002162092 0.002382100 0.43314300
4 Maghiar    1 0.013888385 0.021692427 0.09485246
```

```
5 Maghiar    2 0.005790334 0.009501505 0.20165006
6 Maghiar    3 0.002072372 0.003572637 0.36801066
  pensionar      Student
1 0.8191016 2.797492e-02
2 0.7372192 6.608110e-04
3 0.5622996 1.322811e-05
4 0.8388079 3.075884e-02
5 0.7823052 7.528929e-04
6 0.6263285 1.582008e-05
```

Se calculează probabilităţile ca persoana să aibă un anumit statut ocupaţional, pentru fiecare nivel de educaţie (NIVE) şi fiecare categorie de naţionalitate (NAT).

```
> by(pp.NIVE[, 3:5], pp.NIVE$NAT, colMeans)

pp.NIVE$NAT: German
   aSalariat      Casnica         Elev
2.937637e-07 2.049519e-11 2.329729e-01
------------------------------------------------------------
pp.NIVE$NAT: Maghiar
  aSalariat     Casnica        Elev
0.007250364 0.011588856 0.221504392
------------------------------------------------------------
pp.NIVE$NAT: Roman
  aSalariat     Casnica        Elev
0.008087871 0.008254745 0.267900933
------------------------------------------------------------
pp.NIVE$NAT: Rom
 aSalariat    Casnica       Elev
0.01409629 0.12874010 0.56057516
```

Capitolul 15

Baze de date

Încă de la început se face precizarea că mediul R este expert în analiza de date, iar datele se pot regăsi oriunde, de la calculatorul utilizatorului, pe un server din rețeaua locală sau oriunde în lume, într-o rețea de calculatoare, la care se poate avea acces prin Internet.

Mediul R nu este un sistem de gestiune a bazelor de date (SGBD), deci nu se ocupă de stocarea datelor pe diverse suporturi, ci doar de citirea în memoria calculatorului și analiza acestora. În cazul unor baze de date de dimensiuni foarte mari, de ordinul gigaocteților, vor fi utilizate tehnologii speciale pentru manipularea datelor, acestea neputând fi încărcate în memoria locală.

Există mai multe modalități de citire a datelor, iar cele mai utilizate sunt:

- Exportul datelor din sistemul de gestiune al bazelor de date într-un fișier de tip CSV sau TXT și citirea datelor din acest fișier într-o variabilă din R;
- Importul direct din sistemul de gestiune al bazelor de date într-o variabilă de tip dataframe.

15.1 Terminologie specifică R

În limbajul specific domeniului statisticii există denumirile de seturi de date, observații și variabile. Un tabel cu terminologia adoptată pe scară largă de către programatorii și analiștii care utilizează R poate crea o imagine mai corectă a termenilor specifici.

R	SQL/RDBMS/Excel
data frame	table
observație (în engleză *observation*)	rând (în engleză *row*)
variabilă (în engleză *variable*)	coloană (în engleză *column*)

15.2 Interfețe pentru accesarea bazelor de date

Accesul direct la datele dintr-un SGBD se realizează printr-o interfață (în engleză *DBI interface*) între *R* și SGBD. Cele mai cunoscute pachete pentru citirea datelor din diversele SGBD-uri sunt:

- `RJDBC` acces la bazele de date prin interfața JDBC;
- `RMySQL` interfața pentru MySQL;
- `RODBC` acces la bazele de date de tip ODBC;
- `ROracle` driver pentru baza de date Oracle;
- `RpgSQL` interfața pentru baza de date PostgreSQL;
- `RSQLite` interfața pentru SQLite.

Funcțiile de citire din diverse tipuri de baze de date și în același timp foarte ușor de utilizat sunt incluse în pachetul `foreign`:

Funcția	Format
`read.dbf()`	DBF
`read.spss()`	SPSS
`read.dta()`	Stata
`read.ssd()`	SAS
`read.octave()`	Octave
`read.mtp()`	Minitab
`read.systat()`	Systat

O parte din aceste funcții au fost studiate în Capitolul 7.

15.2.1 JDBC

Pachetul `RJDBC` utilizează sistemul JDBC pentru a permite conectarea la o bază de date. Astfel, orice bază de date care recunoaște driver-ul JDBC

poate fi accesată prin funcțiile disponibile în acest pachet. Un exemplu este dat în rândurile care urmează, dar trebuie adaptat pe baza de date specifică serverului utilizat.

```
> library(RJDBC)
> drv <- JDBC("com.mysql.jdbc.Driver",
"/path/mysql-connector-java-5.1.34.jar", identifier.quote="`")
> con1 <- dbConnect(drv, "jdbc:mysql://localhost/test",
"user", "pwd")
> dbWriteTable(con1, "TEST_TABLE", test_table)
> dbGetQuery(con1, "select count(*) from TEST_TABLE")
> myTable <- dbReadTable(con1, "TEST_TABLE")
> dbDisconnect(con1)
```

15.2.2 MySQL

Specific mediului R este ca pentru o anumită problematică să existe mai multe soluții. Astfel, accesarea unei baze de date de tip MySQL se poate realiza printr-o conexiune de tip ODBC sau în mod direct, prin accesarea serverului respectiv. Pentru exemplele de mai jos a fost utilizat sistemul de operare Windows 7 (64-Bit), pe care a fost instalat serverul MySQL (a se vedea https://dev.mysql.com/downloads/mysql/).

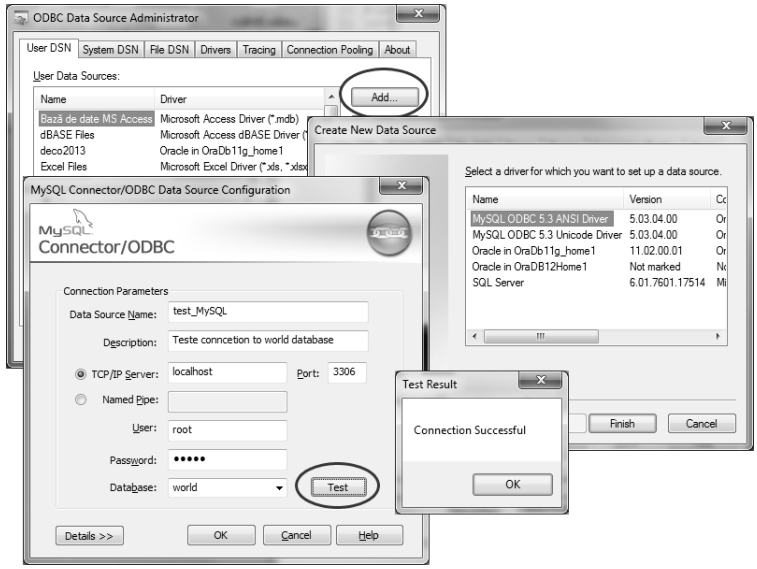
În paragrafele următoare sunt prezentate pe scurt ambele metode de accesare iar prima este cu driverul de tip ODBC, care poate fi descărcat și instalat de pe link-ul *Windows (x86, 64-bit), MSI Installer*, de pe pagina: https://dev.mysql.com/downloads/connector/odbc/. După instalare se crează conexiunea din Control Panel > Administrative Tools > Data Sources, astfel:

Așa cum se observă din imaginea de mai sus s-a creat conexiunea `test_MySQL`, pentru a o accesa din mediul R.

```
> library(RODBC)
> mysql <- odbcConnect("test_MySQL")
> (tables <- sqlTables(mysql))
  TABLE_CAT TABLE_SCHEM      TABLE_NAME TABLE_TYPE REMARKS
1     world                        city      TABLE
2     world                     country      TABLE
3     world             countrylanguage      TABLE
```

Se observă că tabelele disponibile sunt: `city`, `country` și `countrylanguage`, iar pentru exemplificare se afișează primele înregistrări din prima tabelă.

Figura 15.1: Conexiune ODBC-MySQL




```
> country <- sqlQuery(mysql, "SELECT * FROM country")
> head(country, 3)
  Code        Name     Continent                    Region
1  ABW       Aruba North America                 Caribbean
2  AFG Afghanistan          Asia Southern and Central Asia
3  AGO      Angola        Africa            Central Africa
  SurfaceArea IndepYear Population LifeExpectancy  GNP
1         193        NA     103000           78.4  828
2      652090      1919   22720000           45.9 5976
3     1246700      1975   12878000           38.3 6648
  GNPOld             LocalName Capital Code2
1    793                 Aruba     129    AW
2     NA Afganistan/Afqanestan       1    AF
3   7984                Angola      56    AO
                              GovernmentForm
1 Nonmetropolitan Territory of The Netherlands
2                             Islamic Emirate
3                                     Republic
             HeadOfState
1                Beatrix
2           Mohammad Omar
3 JosÃ© Eduardo dos Santos
```

Pentru a accesa cu succes datele, trebuie avute în vedere tabelele

disponibile pe sistemul de lucru.

```
> install.packages("RMySQL")
> library(RMySQL)
> mydb <- dbConnect(MySQL(), user = 'root', password = 'admin',
dbname = 'city', host = 'localhost')
> (tables <- dbListTables(mydb))
> (myfields <- dbListFields(mydb, 'city'))
> mycity <- dbSendQuery(mydb, 'select * from city')
> head(mycity)
```

Adăugarea unei observații în baza de date din serverul MySQL, prin crearea unui dataframe temporar:

```
> mydata <- data.frame(Nume = c("Ionescu", "Popescu"),
Prenume = c("Mihai", "Maria"), Venit = c("2500", "2800"))
> dbSendQuery(mydb, 'drop venituri if exists venituri')
> dbWriteTable(mydb, name='venituri', value=mydata)
```

15.2.3 ODBC

Prin interfața ODBC se pot accesa diverse tipuri de fișiere, pe principalele sisteme de operare, Linux/Unix, Windows, OS X. Pachetul `RODBC` pune la dispoziție o serie de funcții prin care mediul R poate interacționa cu sistemul client-server de pe calculator. O posibilitate este utilizarea pachetului `RODBC`.

```
> install.packages("RODBC")
> library(RODBC)
```

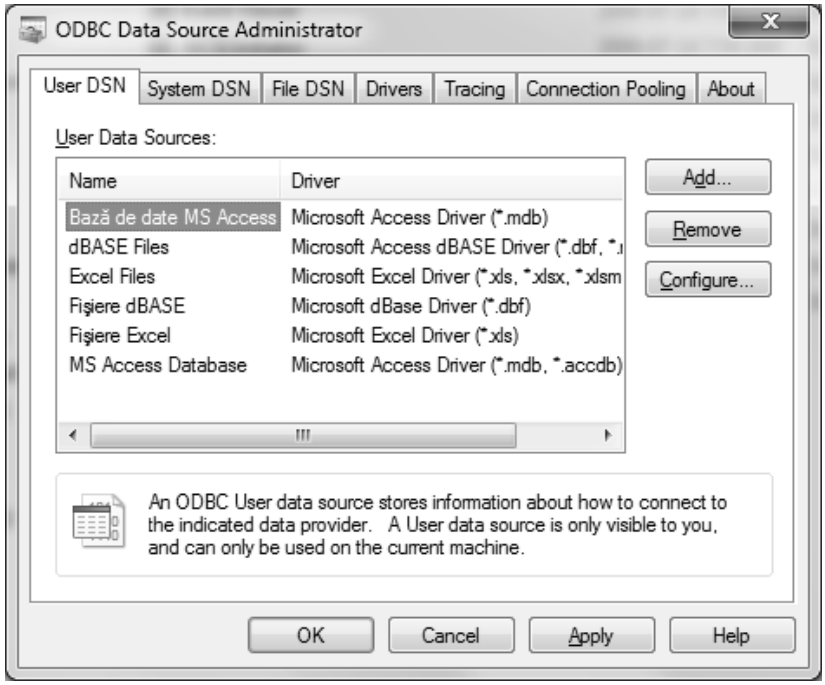
Prin sistemul ODBC se pot realiza legături la baze de date de tip Microsoft Access (*.mdb), dBase Files (*.dbf) și Microsoft Excel (*.xls).

În exemplele de mai jos sunt descrise câteva funcții de bază pentru manipularea datelor:

- `odbcConnect()` - deschide o legătură/conexiune către o bază de date ODBC, prin argumentele: `dsn` (*data source name*), `uid` (*username ID*), `pwd` (*parola de autentificare*);

```
> dsn.name <- "dsn_name"
> user.name <- "guest"
```

Figura 15.2: Sistemul de administrare ODBC




```
> pwd0 <- "1234"
> con1 <- odbcConnect(dsn=dsn.name, uid=user.name, pwd = pwd0)
```

- `odbcClose()` - închide o legătură/conexiune deschisă în prealabil, iar ca atribut primește numele conexiunii;

```
> odbcClose(con1)
```

- `odbcCloseAll()` - închide toate conexiunile existente;

```
> odbcCloseAll()
```

- `sqlTables()` - afișează denumirea tuturor tabelelor existente în baza de date ODBC;

Pentru Microsoft Access se poate specifica tipul tabelelor dacă se vrea ca cele de tip `system` să nu fie cuprinse în lista returnată.

```
> tables.list <- sqlTables(con1, tableType = "TABLE")
```

- `sqlColumns()` - afișează denumirea coloanelor din tabela specificată, din baza de date ODBC, deschisă în prealabil;

```
> table.name <- "Incomes"
> col.list <- sqlColumns(con1, table.name)
```

- `sqlFetch()` - citește total sau parțial o tabelă din baza de date ODBC;

Returnarea întregului conținut al tabelei:

```
> alltable <- sqlFetch(con1, table.name)
```

Returnarea primelor 20 de observații/rânduri:

```
> seltable <- sqlFetch(channel, "USArrests", max = 20)
```

- `sqlQuery()` - transmite o interogare de tip SQL către o bază de date ODBC și întoarce rezultatul interogării;

Selecția persoanelor cu veniturile mai mari de 800 lei, ordonate după nume:

```
> selectie <- sqlQuery(con1, "select Name, Income from Incomes
where Income > 800 order by Name"))
```

15.2.4 Oracle

Conectarea la o bază de date de tip Oracle se face prin pachetul `ROracle`, iar pe calculator trebuie să existe Oracle Instant Client sau Oracle Database Client. Codul prin care se face conexiunea este descris mai jos:

```
> library(ROracle)
> drv <- dbDriver("Oracle")
> con1 <- dbConnect(drv, "user", "pwd")
```

Crearea unei tabele noi:

```
> dbWriteTable(con1, "TEST_TABLE", test_table)
```

Transmiterea unei interogări pentru tabela creată:

```
> dbGetQuery(con1, "select count(*) from TEST_TABLE")
> myTable <- dbReadTable(con1, "TEST_TABLE")
> dbDisconnect(con1)
```

15.3 Funcții de manipulare a datelor

Citirea datelor dintr-o bază de date sau doar dintr-un fișier reprezintă doar primul pas în pregătirea datelor pentru analiză. Un altă etapă foarte importantă este prelucrarea datelor în vederea realizării analizelor. Practic, în această fază, se realizează explorarea, selecția și corectarea variabilelor. În măsura în care este nevoie se vor crea variabile noi care vor avea ca bază de plecare una sau mai multe variabile existente în baza de date.

15.3.1 Selecții

De câte ori se lucrează cu baze de date se ridică problema dacă se utilizează toate observațiile/variabilele sau doar o parte dintre ele, iar uneori sunt necesare toate variabilele sau doar un set mai restrâns. Toate aceste aspecte devin și mai importante în cazul bazelor de date de dimensiuni foarte mari, de ordinul zecilor de gigaocteți.

Rânduri/observații

Primul aspect prezentat este selectarea observațiilor/rândurilor care îndeplinesc o anumită condiție, iar ca exemplu se încarcă setul de date `airquality` din pachetul `datasets`.

```
> data("airquality", package = "datasets")
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

O primă verificare a datelor se referă la câte observații și câte variabile sunt în dataframe.

```
> dim(airquality)
[1] 153   6
```

Deci, în exemplul prezentat sunt 153 de observații și 6 variabile. Numărul de nivele pe care le are variabila `Month` se poate afla astfel:

```
> levels(as.factor(airquality$Month))
[1] "5" "6" "7" "8" "9"
```

Instrucțiunile de mai sus sunt specifice oricărei explorări preliminare a setului de date. Se vor selecta doar observațiile care conțin în variabila `Month` valoarea 6, prin instrucțiunea `subset(dataset_name, nume_variabila == 'valoare')`.

```
> selectie <- subset(airquality, Month == 6)
> dim(selectie)
[1] 30  6
> head(selectie, 3)
   Ozone Solar.R Wind Temp Month Day
32    NA     286  8.6   78     6   1
33    NA     287  9.7   74     6   2
34    NA     242 16.1   67     6   3
```

Coloane/Variabile

Selectarea doar a variabilelor/coloanelor de interes se poate realiza tot prin instrucțiunea `subset()`, dar cu argumentul `select`, astfel:

```
> selectie <- subset(airquality, select = c(Wind, Temp))
> dim(selectie)
[1] 153   2
> head(selectie, 3)
  Wind Temp
1  7.4   67
2  8.0   72
3 12.6   74
```

În cazul se dorește selecția mai multor coloane consecutive se poate utiliza operatorul ":". Selecția variabilelor `Wind`, `Temp`, `Month` și `Day`, adică a variabilelor 3, 4, 5 și 6.

```
> selectie1 <- subset(airquality, select = c(3:6))
```

Același rezultat este obținut și prin utilizarea numelor variabilelor:

```
> selectie2 <- subset(airquality, select = c(Wind:Day))
```

Se poate verfica dacă cele două selecții sunt identice:

```
> all.equal(selectie1, selectie2)
[1] TRUE
```

Totodată, se pot selecta variabilele de interes (`Wind`, `Temp` și `Month`) și observațiile care îndeplinesc o anumită condiție (valorile `5` și `6` din variabila `Month`):

```
> selectie <- subset(airquality, Month == 5 | Month == 6,
select = c(Wind:Month))
> dim(selectie)
[1] 61  3
> head(selectie, 3)
  Wind Temp Month
1  7.4   67     5
2  8.0   72     5
3 12.6   74     5
```

Pentru selecții se mai pot utiliza și caracteristicile dataframe-ului.

- Selectarea tuturor observațiilor/rândurilor care au o anumită valoare într-o variabilă:
  `NumeDataFrame[NumeDataFrame$NumeVariabila == Valoare, ]`
- Selectarea anumitor variabile/coloane:
  `NumeDataFrame[, NumeVariabila]`

Se exemplifică ambele cazuri. Primul, selecția observațiilor/ înregistrărilor/ rândurilor care au valoarea `6` în variabila `Month`:

```
> selectie <- airquality[airquality$Month == 6, ]
> dim(selectie)
[1] 30  6
> head(selectie, 3)
   Ozone Solar.R Wind Temp Month Day
32    NA     286  8.6   78     6   1
33    NA     287  9.7   74     6   2
34    NA     242 16.1   67     6   3
```

Al doilea caz, afișarea primelor 3 observații selectate, doar coloanele `Wind` și `Temp`.

```
> selectie <- airquality[, c("Wind", "Temp")]
> dim(selectie)
[1] 153   2
> head(selectie, 3)
  Wind Temp
1  7.4   67
2  8.0   72
3 12.6   74
```

Există un dezavantaj, deoarece aici nu mai poate fi utilizat operatorul ':', instrucțiunea `selectie <- airquality[, c(Wind:Temp)]` va genera o eroare în acest caz:

```
> selectie <- airquality[, c(Wind:Temp)]
Error in `[.data.frame`(airquality, , c(Wind:Temp)) :
  object 'Wind' not found
```

15.3.2 Sortări

Sortarea sau ordonarea elementelor unui vector sau a unui dataframe se poate face crescător sau descrescător prin utilizarea funcțiilor `sort()` sau `order()`. Totuși, trebuie acordată atenție la diferențele dintre cele două funcții. Funcția `sort()` returnează un vector cu elementele ordonate corespunzător opțiunilor date prin argumente, crescător, respectiv descrescător. În schimb, funcția `order()` returnează un vector de aceeași lungime cu cel transmis funcției ca argument, dar având ca valori pozițiile pe care ar trebui să le ocupe elementele din vectorul sursă, în ordine crescătoare sau descrescătoare.

```
> data("airquality", package="datasets")
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> head(airquality$Temp)
[1] 67 72 74 62 56 66
```

Se sortează primele 6 elemente și se afișează rezultatul:

```
> sort(head(airquality$Temp))
[1] 56 62 66 67 72 74
```

Se ordonează primele 6 elemente, dar se observă o diferență:

```
> order(head(airquality$Temp))
[1] 5 4 6 1 2 3
```

Funcția `order()` a returnat pozițiile pe care le au cele 6 elemente, dar ordonate crescător, adică pe prima poziție se va plasa elementul de pe poziția originală `5` (`56`), pe a doua poziție se va situa elementul de poziția `4` din vectorul dat (`62`), pe poziția a treia va fi elementul `6` din vectorul dat, și tot așa până la a șasea poziție care va fi ocupată de elementul `3` din vector (`74`). Cu alte cuvinte, prin funcția `order()` se poate ordona dataframe-ul după orice variabilă, astfel:

```
> airquality <- airquality[order(airquality$Temp), ]
> head(airquality)
   Ozone Solar.R Wind Temp Month Day
5     NA      NA 14.3   56     5   5
18     6      78 18.4   57     5  18
25    NA      66 16.6   57     5  25
27    NA      NA  8.0   57     5  27
15    18      65 13.2   58     5  15
26    NA     266 14.9   58     5  26
```

Dacă se dorește o sortare a elementelor în ordine descrescătoare, se utilizează argumentul `decreasing = TRUE`.

```
> airquality <- airquality[order(airquality$Temp,
                                 decreasing = TRUE), ]
> head(airquality)
    Ozone Solar.R Wind Temp Month Day
120    76     203  9.7   97     8  28
122    84     237  6.3   96     8  30
123    85     188  6.3   94     8  31
121   118     225  2.3   94     8  29
126    73     183  2.8   93     9   3
127    91     189  4.6   93     9   4
```

În cazul în care variabila după care se ordonează observațiile are valori NA, adică sunt necompletate, aceste observații pot fi puse la sfârșitul dataframe-ului, prin argumentul `na.last = TRUE`, ca în exemplul de mai jos:

```
> airquality <- airquality[order(airquality$Ozone,
                                 na.last = TRUE), ]
> head(airquality)
    Ozone Solar.R Wind Temp Month Day
21      1       8  9.7   59     5  21
23      4      25  9.7   61     5  23
18      6      78 18.4   57     5  18
147     7      49 10.3   69     9  24
11      7      NA  6.9   74     5  11
76      7      48 14.3   80     7  15
```

Iar dacă trebuie ca aceste observații să fie omise se atribuie argumentului `na.last` valoarea `NA`. Atenție, deoarece această comandă va face să se piardă observații din dataframe.

```
> dim(airquality)
[1] 153   6
> airquality <- airquality[order(airquality$Ozone,
                                 na.last = NA), ]
> head(airquality)
    Ozone Solar.R Wind Temp Month Day
21      1       8  9.7   59     5  21
23      4      25  9.7   61     5  23
18      6      78 18.4   57     5  18
76      7      48 14.3   80     7  15
11      7      NA  6.9   74     5  11
147     7      49 10.3   69     9  24
> dim(airquality)
[1] 116   6
```

Se observă că în dataframe au rămas doar 116 observații față de 153 câte au fost încărcate inițial, dar uneori este util să se renunțe la aceste observații care pot conduce la rezultate greșite din cauza elementelor necompletate (a valorilor `NA` din variabile).

15.3.3 Concatenări

Îmbinarea sau concatenarea a două seturi de date se poate realiza doar dacă există o variabilă în comun, numită de obicei cheie, deoarece valorile trebuie să îndeplinească anumite caracteristici pentru a realiza o concatenare corectă. Funcția `merge()` este utilizată pentru a realiza concatenarea a două seturi de date. Se crează un dataframe cu numele angajaților:

```
> (df1 <- data.frame(ID = 1:3, Nume = c("Ion", "Petre",
"Vasile")))
  ID   Nume
1  1    Ion
2  2  Petre
3  3 Vasile
```

Se crează un alt dataframe cu salariile, dar cu același ID, astfel încât un anumit salariu să corespundă unui anumit salariat, identificat prin `ID`:

```
> (df2 <- data.frame(ID = 1:3, Salariu = c(1400, 1800, 1500)))
  ID Salariu
1  1    1400
2  2    1800
3  3    1500
```

Se concatenează cele două seturi de date după variabila `ID`:

```
> (df3 <- merge(df1, df2, by = "ID"))
  ID   Nume Salariu
1  1    Ion    1400
2  2  Petre    1800
3  3 Vasile    1500
```

Exemplul de mai sus a avut ca rezultat concatenarea celor două seturi de date și realizarea unui dataframe cu același număr de observații, dar cu toate coloanele reunite, altfel spus o concatenare unu-la-unu. Dacă cele două seturi de date au numere diferite de observații, trebuie atenție la rezultat, pentru că se vor multiplica valorile dintr-un set de date pentru a se completa celălalt set de date, evident, având criteriu principal variabila cheie.

```
> (df1 <- data.frame(ID = c(1:3,2), Nume = c("Ion", "Petre",
```

```
"Vasile", "Maria")))
  ID   Nume
1  1    Ion
2  2  Petre
3  3 Vasile
4  2 Maria
> (df2 <- data.frame(ID = c(1:3,3), Salariu = c(1400, 1800,
1500, 2000)))
  ID Salariu
1  1    1400
2  2    1800
3  3    1500
4  3    2000
> (df3 <- merge(df1, df2, by = "ID"))
  ID   Nume Salariu
1  1    Ion    1400
2  2  Petre    1800
3  2  Maria    1800
4  3 Vasile    1500
5  3 Vasile    2000
```

Se observă că s-a realizat concatenarea, dar rezultatul este mai mult ca sigur greșit din punctul de vedere al datelor, deoarece nici sursele pe baza cărora s-a efectuat funcția `merge` nu au fost bune (atât în `df1`, cât și în `df2` există mai multe ID-uri identice).

Căutarea și eliminarea observațiilor duplicat

Există două funcții care pot fi utilizate pentru căutarea și/sau eliminarea observațiilor duplicat, `duplicated()` și `unique()`. Funcția `duplicated()` identifică toate elementele duplicat dintr-un vector și returnează un vector de tip logic, de lungimea celui verificat, cu valoarea `TRUE` pentru toate elementele care sunt duplicat și `FALSE` în caz contrar.

Se crează un vector de 100 de valori luate aleator, în care se pot regăsi duplicate, prin menționarea argumentului `replace = TRUE`:

```
> set.seed(50)
> x1 <- sample(1:100, replace = TRUE)
```

În exemplul de mai jos se separă în doi vectori distincți valorile duplicat de celelalte:

```
> duplicate0 <- x1[duplicated(x1)]
```

```
> length(duplicate0)
[1] 39
> faraduplicate0 <- x1[!duplicated(x1)]
> length(faraduplicate0)
[1] 61
```

Spre deosebire de funcția `duplicated()`, funcția `unique()` întoarce un vector/dataframe care conține doar valorile unice, eliminându-le pe cele duplicat:

```
> unice0 <- unique(x1)
> length(unice0)
[1] 61
```

Se poate verifica că rezultatele, obținute prin cele două metode, pentru vectorii care conțin doar valorile unice, sunt identice:

```
> all.equal(faraduplicate, unice0)
[1] TRUE
```

15.3.4 Tratarea factorilor

Stabilirea nivelelor

Nivelele sunt specifice variabilelor de tip factor, iar pentru exemplificare se va utiliza tot setul de date cu care s-a lucrat.

```
> data("airquality", package = "datasets")
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

Se poate reconsidera variabila `Month` ca fiind de tip factor, astfel:

```
> class(airquality$Month)
[1] "integer"
> airquality$Month <- as.factor(airquality$Month)
```

```
> levels(airquality$Month)
[1] "5" "6" "7" "8" "9"
```

Se observă că există cinci nivele care reprezintă lunile Mai până la Septembrie.

Redenumirea nivelelor

Schimbarea denumirii nivelelor pentru variabila de tip factor `Month` se realizează foarte simplu, prin atribuirea unui vector de caractere cu noile denumiri, în cazul prezentat, schimbarea din cifre în denumirea lunilor respective:

```
> levels(airquality$Month) <- c("Mai", "Iun", "Iul", "Aug", "Sep")
```

Schimbarea denumirilor se va reflecta în tot setul de date:

```
> levels(airquality$Month)
[1] "Mai"    "Iun"    "Iul"    "Aug"    "Sep"
> head(airquality, 3)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67   Mai   1
2    36     118  8.0   72   Mai   2
3    12     149 12.6   74   Mai   3
> tail(airquality)
    Ozone Solar.R Wind Temp Month Day
151    14     191 14.3   75   Sep  28
152    18     131  8.0   76   Sep  29
153    20     223 11.5   68   Sep  30
```

Prin transformarea tipului variabilei în factor se pot obține informații specifice variabilelor categoriale, astfel:

```
> table(airquality$Month)
       Mai       Iun       Iul       Aug       Sep
        31        30        31        31        30
```

Ceea ce înseamnă că în dateframe sunt 31 de observații care au valoarea 5(Mai) în variabila `Month`, 30 de observații pentru Iunie, etc., valori care coincid chiar cu zilele calendaristice din lunile respective.

Schimbarea ordinii nivelelor unui factor

Ordinea nivelelor unei variabile de tip factor este foarte importantă în utilizarea variabilei ca argument al funcțiilor, spre exemplu `multinom()`.

Motivul este unul simplu și anume, se stabilește nivelul de referință dintr-o analiză specifică.

În exemplul următor, prin funcția `relevel`, se stabilește ca valoare de referință luna Iunie.

```
> airquality$Month <- relevel(airquality$Month, ref = "Iun")
```

Se observă că după execuția acestei funcții primul nivel afișat este `Iun`:

```
> levels(airquality$Month)
[1] "Iun"    "Mai"    "Iul"    "Aug"    "Sep"
```

Totuși, în baza de date, variabilă rămâne cu aceleași valori, dar va fi tratată complet diferit de funcțiile de analiză.

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67   Mai   1
2    36     118  8.0   72   Mai   2
3    12     149 12.6   74   Mai   3
4    18     313 11.5   62   Mai   4
5    NA      NA 14.3   56   Mai   5
6    28      NA 14.9   66   Mai   6
```

15.4 Prelucrarea tabelelor

Adăugarea unei variabile

O variabilă poate fi adăugată prin mai multe metode, iar cea mai simplă este atribuirea directă a unei valori, calculată sau constantă, noii variabile. Pentru exemplificare se va utiliza setul de date `airquality`, în care se va adăuga o variabilă `TempC` prin care să transformă gradele Farenheit în grade Celsius. Pentru aceasta se va utiliza formula `C = (F - 32) / 1.8`, cu rotunjirea rezultatului la 2 zecimale.

```
> data("airquality", package="datasets")
> head(airquality, 3)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

```
> class(airquality$Temp)
[1] "integer"
> airquality$TempC <- round((airquality$Temp - 32) / 1.8, 2)
> head(airquality, 3)
  Ozone Solar.R Wind Temp Month Day TempC
1    41     190  7.4   67     5   1 19.44
2    36     118  8.0   72     5   2 22.22
3    12     149 12.6   74     5   3 23.33
```

O altă metodă este utilizarea funcției `transform()`.

```
> airquality <- transform(airquality,
TempC = round((Temp - 32) / 1.8, 2))
> head(airquality, 3)
  Ozone Solar.R Wind Temp Month Day TempC
1    41     190  7.4   67     5   1 19.44
2    36     118  8.0   72     5   2 22.22
3    12     149 12.6   74     5   3 23.33
```

Ștergerea unei variabile

Îndepărtarea unei variabile dintr-un set de date se face simplu, prin atribuirea valorii `NULL` acelei variabile. Din exemplul de mai sus și se dă un exemplu, prin ștergerea variabilei nou introduse.

```
> airquality$TempC <- NULL
> head(airquality, 3)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

Redenumirea variabilelor

Există mai multe metode pentru redenumirea variabilelor unui set de date. În primul rând, pentru schimbarea tuturor denumirilor se poate folosi funcția `names()`.

```
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> names(airquality) <- c("Ozon", "Solar", "Vant", "Temperatura",
"Luna", "Ziua")
> names(airquality)
[1] "Ozon" "Solar" "Vant" "Temperatura" "Luna" "Ziua"
```

Totuși, rareori se schimbă toate denumirile și e bine să se cunoască câteva metode pentru a schimba doar una sau doar câteva denumiri de variabile. Deoarece funcția `names()` întoarce un vector cu denumirile, tot cu ajutorul ei se poate executa și atribuirea unei valori doar elementului x din respectivul vector. Spre exemplu, se schimbă denumirea variabilei `Vant`, care este a treia în vectorul denumirilor:

```
> names(airquality)[3] <- "Wind"
```

Se verifică prin afişarea datelor:

```
> head(airquality, 3)
  Ozon Solar Wind Temperatura Luna Ziua
1   41   190  7.4          67    5    1
2   36   118  8.0          72    5    2
3   12   149 12.6          74    5    3
```

O altă metodă este prin funcția `colnames()`.

```
> colnames(airquality)[3] <- "Vant"
```

Vizualizarea denumirilor confirmă schimbarea realizată.

```
> colnames(airquality)
[1] "Ozon" "Solar" "Vant" "Temperatura" "Luna" "Ziua"
> names(airquality)
[1] "Ozon" "Solar" "Vant" "Temperatura" "Luna" "Ziua"
```

Depinde cât de multe informaţii sunt cunoscute despre setul de date, iar dacă nu se cunoaște a câta variabilă este cea a cărei denumire trebuie schimbată atunci se foloseşte o altă metodă. Mai întâi se încarcă datele:

```
> install.packages("data.table")
> library(data.table)
```

Se schimbă denumirea variabilei `Temperatura` cu `Temp`.

```
> setnames(airquality, "Temperatura", "Temp")
> names(airquality)
[1] "Ozon"  "Solar" "Vant"  "Temp"  "Luna"  "Ziua"
```

O altă funcție interesantă este `rename()`, din pachetul `plyr`.

```
> install.packages("plyr")
> library(plyr)
> airquality <- rename(airquality, c('Luna' = 'Month', 'Ziua' = 'Day'))
> names(airquality)
[1] "Ozon"  "Solar" "Vant"  "Temp"  "Month" "Day"
```

Rearanjarea variabilelor

Variabilele dintr-un dataframe pot fi rearanjate după numărul variabilei sau după denumirea acesteia.

După numărul avut de fiecare variabilă:

```
> data("airquality", package = "datasets")
> names(airquality)
[1] "Ozone" "Solar.R" "Wind" "Temp" "Month" "Day"
> airquality <- airquality[c(2, 3, 4, 6, 1, 5)]
> names(airquality)
[1] "Solar.R" "Wind" "Temp" "Day" "Ozone" "Month"
```

După denumirea variabilelor:

```
> data("airquality", package = "datasets")
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> airquality <- airquality[c("Solar.R", "Wind", "Temp", "Day",
"Month", "Ozone")]
> names(airquality)
[1] "Solar.R" "Wind" "Temp" "Day" "Month" "Ozone"
```

Capitolul 16

Big Data și R

16.1 Ce reprezintă Big Data

Dacă în urmă cu 20-30 de ani volumele de date de ordinul zecilor sau sutelor de MB păreau foarte mari, astăzi situația s-a schimbat radical. Unul dintre motivele care au contribuit la această schimbare este apariția și dezvoltarea rețelei Internet și a sistemului World Wide Web. Orice activitate este practic înregistrată într-o bază de date: fie că executăm o căutare utilizând un motor de căutare pe Internet, fie că vizităm site-ul unui magazin electronic sau consultăm presa online, toate aceste activități se înregistrează undeva. Companiile de telefonie mobilă, marile lanțuri de supermarketuri înregistrează date despre clienții lor, despre vânzări. Volumele de date strânse astfel sunt de ordinul exabytes (2^60 bytes adică aproximativ 10^18 bytes). Procesarea acestor informații aduce avantaje majore companiilor care dețin aceste date. Să luăm un simplu exemplu. Dacă la începutul anilor '90 comercianții strângeau date zilnice despre vânzări și se mulțumeau dacă reușeau să obțină informații despre vânzările pe categorii de produse, astăzi procesarea informațiilor obținute prin scanarea codurilor de bare conduce la cunoașterea comportamentului consumatorilor la nivel individual (în cazul comerțului electronic) ceea ce constituie un mare avantaj. Se pot cunoaște informații detaliate despre produsele vândute, orele la care au fost vândute, se pot determina stocurile produse existente într-un magazin pe locații (rafturi).

În domeniul fizicii, experimentele efectuate cu ajutorul *Large Hadron Collider* pot produce în jur de 500 Exabytes pe zi ceea ce înseamnă aproape 150 milioane Petabytes anual (Brumfield, 2011). O asemenea cantitate de informație este greu de stocat și prelucrat chiar și cu tehnica de azi. Deocamdată doar un procent de 0.001% din toate datele

produse sunt stocate și prelucrate.

Alte domenii alte științei cunosc de asemenea o explozie a cantității de informații care se produc: Centrul NASA pentru Simulări în Domeniul Climei lucrează cu volume de date de ordinul zecilor de Petabytes (Webster, 2012), în astronomie, telescoapele moderne strâng informație de ordinul Petabytes zilnic.

Rețelele de socializare și site-urile de comerț electronic colectează cantități uriașe de date. Spre exemplu Facebook are un depozit de date de peste 300 PB (Vagata şi Wilfong, 2014) cu o rată de creștere de circa 300TB zilnic. Se estimează că pe rețeaua Facebook sunt înregistrați peste 500 de milioane de utilizatori și se stochează în jur de 50 de miliarde de poze încărcate de aceștia. eBay.com deține baze de date de ordinul a 90 PB (Tay, 2013) referitoare la tranzacțiile efectuate de clienții săi în timp ce Amazon.com stochează milioane de tranzacții pe zi.

Acest volum uriaș de date care se produc și se stochează zilnic a dat naștere la conceptul de "big data". Prima referire la acest nou concept se regăsește într-un raport al companiei de consultanță META Group (devenit între timp Gartner) (Laney, 2001) din anul 2001 unde se afirma că asistăm la o creștere a producției de date la nivel mondial care se caracterizează prin trei dimensiuni (cei 3 V):

- *volumul* datelor care este într-o continuă creștere. Se estimează că rata de creștere a volumului datelor este exponențială;
- *viteza* cu care se produc datele care este de asemenea în creștere. Companiile trec de la aplicații de tip batch la aplicații în timp real producând date cu o viteză foarte mare;
- *varietatea* datelor care se traduce prin creșterea tipurilor de date produse și stocate de companii precum și prin diversificarea surselor de date. Marea majoritate a datelor care se produc azi sunt date nestructurate.

Pe lângă aceste trei caracteristici de bază au mai fost identificate si alte caracteristici ale "big data":

- *variabilitatea* datelor se referă la inconsistențe care pot apare în date la diferite momente de timp ceea ce cauzează dificultăți în prelucrarea lor;
- *veridicitatea* datelor care reprezintă o problemă foarte importantă pentru cel care folosește datele;

- *complexitatea* datelor care se manifestă atunci când datele provin din surse diferite și trebuie conectate și corelate pentru a putea extrage informațiile necesare.

În 2012 aceeași companie Gartner dă o nouă definiție conceptului de *big data*, care reprezintă active informaționale de volum mare, care se produc cu viteză mare și/sau de mare varietate, care necesită noi forme de procesare pentru a facilita luarea de decizii mai bune, descoperirea de noi cunoștințe și optimizarea proceselor" (Mark şi Laney, 2012). O altă definiție dată conceptului de "big data" afirmă că acesta înseamnă "datele păstrate și prelucrate în cantități imense, datorită unor medii de stocare mai ieftine, unor metode de procesare mai rapide și unor algoritmi mai performanți" (Mayer-Schönberger şi Cukier, 2013).

Noțiunea de "big data" este foarte dinamică: ceea ce se consideră acum ca încadrându-se în acest concept poate deveni ceva uzual peste câțiva ani. Se poate verifica dacă într-adevăr așa stau lucrurile prin analizarea capacității de a schimba/transfera informație la nivel mondial prin intermediul rețelelor de comunicații: dacă în 1986 volumul de date schimbat prin intermediul acestor rețele a fost de aproximativ 280 Petabytes, în 2000 acesta a crescut la 2.2 Exabytes iar în 2014 se estimează că se va ajunge la 670 Exabytes (Economist, 2010).

Pentru a putea profita de această cantitate uriașă de date care se produc azi organizațiile trebuie să dispună de tehnologii speciale. Instrumentele software clasice nu pot prelucra eficient volume așa de mari de date. Au apărut astfel noi instrumente specializate pentru stocarea și procesarea volumelor mari de date: Hadoop, Apache Spark, Pentaho Business Analytics, baze de date de tip NoSQL etc.

Din acest punct de vedere se va investiga modul în care R poate fi folosit la procesarea și analiza volumelor mari de date. Se vor prezenta o serie de pachete R specializate în prelucrarea datelor care nu pot fi tratate în mod obișnuit datorită volumului: pachetul `bigmemory` și pachete înrudite `biganalytics`, `bigtabultate`, `biglm`, `bigalgebra` precum și pachetul `ff` . În partea a doua a acestui capitol se va prezenta sistemul software *Hadoop* care s-a impus ca o tehnologie de bază în stocarea și procesarea volumelor mari de date și modul cum acesta poate fi interfațat cu R pentru a permite analiza datelor. În acest sens vor fi prezentate trei modalități prin care date stocate cu ajutorul *Hadoop* pot fi accesate pentru prelucrare în R: *Streaming* , *Rhipe* și *RHadoop*.

16.2 Pachete R pentru tratarea volumelor mari de date

Sistemul R prezintă o serie de limitări în ceea ce privește procesarea masivelor de date de dimenisuni foarte mari. Cea mai importantă limitare provine din modul de lucru cu datele: R încarcă în memoria RAM toate datele pe care le procesează. Dimensiunea limitată a memoriei RAM impune și o limită volumului de date ce pot fi procesate în R. Lucrul cu masive mari de date poate conduce foarte rapid la epuizarea spațiului disponibil de memorie. Chiar dacă teoretic un sistem de operare pe 32 de biți poate accesa 4 GB de memorie RAM, practic sistemul R va arunca o excepție atunci când se depășește limita de 2 GB. Pe sistemele de 64 de biți actuale limita de memorie este mult mai mare, de 8 TB. Cu toate acestea, utilizatorul nu va putea lucra cu matrici sau dataframe de dimenisuni foarte mari pentru că în R nu există deocamdată un tip de întreg pe 64 de biți. Pentru a indexa elementele unei matrici se folosesc indici reprezentați ca întregi pe 32 de biți ceea ce conduce la o limitare a dimensiunii matricei. Cu toate că sistemele de operare actuale folosesc mecanismul memoriei virtuale și teoretic pot aloca obiecte mai mari decât dimensiunea memoriei RAM, nu este recomandat să alocăm obiecte de dimensiuni foarte mari deoarece sistemul de operare va folosi partiția de *swap* foarte frecvent ceea ce conduce de regulă la apariția fenomenului de *trashing*.

Un exemplu foarte simplu care arată limitele R în ceea ce privește alocarea obiectelor de dimensiuni mari este următorul:

```
> x <- rep(0, 2^30)
Error: cannot allocate vector of size 8.0 Gb
```

Exemplul precedent a fost rulat pe R versiunea 2.15.2 sub sistemul de operare Linux Fedora Core 17 pe 64 de biți, pe un calculator cu 4 GB de memorie RAM și cu o partiție de *swap* de 6 GB. Același exemplu puțin modificat, dar rulat pe o versiune de R pe 32 de biți sub sistemul de operare Windows 7 conduce la o limitare și mai mare legată dimensiunea obiectelor care pot fi create:

```
> x <- rep(0, 2^28)
Error: cannot allocate vector of size 2.0 Gb
```

O soluție posibilă la aceste probleme ar fi utilizarea unei memorii RAM mai mari (dar aceasta este destul de scumpă) și folosirea R pe 64 de

biți pe calculatoare cu sisteme de operare tot pe 64 de biți. S-a arătat totuși că a avea la dispoziție mai multă memorie RAM nu rezolvă în întregime problema obiectelor foarte mari întrucât în R utilizatorul nu are la dispoziție un tip de date întreg pe 64 de biți cu care să poata indexa elementele acestor obiecte.

Din fericire R pune la dispoziția utilizatorilor o serie de pachete cu care se pot procesa volumele foarte mari de date. Dintre acestea vor fi prezentate două pachete care sunt folosite cu succes în prezent:

- `bigmemory`;
- `ff`.

O altă limitare a sistemului R constă în faptul că utilizează doar un singur nucleu (core) al procesorului cu toate că astăzi toate procesoarele uzuale sunt multi-nucleu (multicore). Atunci când trebuie procesate volume foarte mari de date este util ca operațiile de prelucrare a datelor să se execute în paralel pentru a scurta timpul de execuție. și în acest caz R dispune de pachete specializate care permit efectuarea unor operații de prelucrare în paralel, utilizând astfel în mod eficient toate nucleele procesorului. Dintre acestea va fi prezentat doar pachetul *`MapReduce`* care implementează un model de calcul distribuit cu același nume.

16.2.1 Familia de pachete "bigmemory"

O primă soluție de prelucrare a masivelor de date de dimenisuni foarte mari în R constă în utilizarea pachetului `bigmemory` (Kane et al., 2013) care permite declararea unor obiecte de tipul *big.matrix* și procesarea lor cu ajutorul rutinelor puse la dispoziție de pachete înrudite: `biganalytics`, `bigtabulate`, `bigalgebra`. Deoarece pachetul `bigmemory` este scris in limbajul C++, tipul de dată al elementelor matricilor va fi dictat de tipurile de date disponibile în C++: `int`, `double`, `short`, `char`. Pachetul *bigmemory* permite utilizarea memoriei partajate pentru matrici cu implementarea unor mecanisme de excluziune mutuală transparente utilizatorului. De asemenea, `bigmemory` permite crearea unor obiecte care depășesc dimensiunea memoriei RAM, folosind fișierele ca suport de memorie pentru aceste obiecte ceea ce face posibil ca datele dintr-un obiect ce are în spatele său un fișier să poată fi partajate nu numai de către procesele de pe acel calculator dar și în interiorul unui cluster de calculatoare.

Se vor prezenta pe scurt principalele funcții incluse în pachetul `bigmemory`. Crearea unei matrici se realizează astfel :

```
> install.packages("bigmemory")
> library("bigmemory")
> (A <- big.matrix(10, 10, type = "integer", init = 0))
An object of class "big.matrix"
Slot "address":
<pointer: 0xa5f5b0>
```

A fost instalat pachetul `bigmemory` (deoarece acesta nu este instalat implicit) apoi s-a creat o matrice `A` cu 10 de linii și 10 de coloane, cu elemente de tip întreg inițializate cu zero. Tastarea numelui matricei în consola R nu conduce la afișarea valorilor elementelor matricei ci arată faptul că `A` este de un pointer către o structură de date.

Se vor inițializa elementele de pe coloana 1 a matricei cu valoarea 10 și se vor afișa primele 6 linii ale matricei:

```
> for (k in 1:10) {
     A[k,1] = 10
 }
> head(A)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   10    0    0    0    0    0    0    0    0     0
[2,]   10    0    0    0    0    0    0    0    0     0
[3,]   10    0    0    0    0    0    0    0    0     0
[4,]   10    0    0    0    0    0    0    0    0     0
[5,]   10    0    0    0    0    0    0    0    0     0
[6,]   10    0    0    0    0    0    0    0    0     0
```

Din acest exemplu simplu se observă ușor că elementele unei matrice de tipul *big.matrix* se accesează într-un mod asemănător cu elementele unei matrice obișnuite. Conținutul unei matrice de tipul *big.matrix* poate fi salvat pe disc sub forma unui fișier text ca în exemplul următor:

```
write.big.matrix(A, "A.csv", sep = ',')
```

Matricea `A` este salvată în fișierul *A.csv*, parametrul `sep = ','` indicând faptul că elementele matricei vor fi separate prin caracterul `','`.

O matrice de tipul *big.matrix* poate fi creată și inițializată pe baza datelor dintr-un fișier de tip *csv*. Se va prezenta în continuare un

exemplu în care se va crea o matrice plecând de la fișierul de date *NYSE-2000-2001.tsv.gz*[1]. După dezarhivare, se obține fișierul *NYSE-2000-2001.tsv* care este un fișier de tip *tsv tab separated values*. Fișierul conține informații despre tranzacțiile la bursa din New York în anii 2000 și 2001. Pe prima coloană avem simbolul bursei (NYSE) apoi un simbol al companiilor care au tranzacționat acțiuni (stock_symbol), data tranzacției (date), prețul de deschidere al acțiunilor (stock_price_open), valoarea maximă pe care o atinge o acțiune în cursul unei zile (stock_price_high), valoarea minimă a acțiunii (stock_price_low), valoarea acțiunii la închiderea zilei (stock_price_close), valoarea volumului tranzacțiilor (stock_volume) și valoarea ajustată a prețului de închidere al acțiunii (stock_price_adj_close). Conținutul fișierului poate fi citit în mod normal și afișat ca în exemplul următor:

```
> se_data <- read.table(file = "NYSE-2000-2001.tsv",
header = TRUE, sep = "\t")
> head(se_data)
```

În acest exemplu a fost creat un obiect de tipul *data.frame* denumit *se_data*. În continuare se va citi conținutul acestui fișier și se va construi o matrice de tipul *big.matrix* cu datele din acesta. Matricea construită va introduce două elemente noi. Va fi o matrice de tipul *big.matrix* dar care va avea ca suport extern de memorare a datelor un fișier binar. Acest lucru este recomdandat pentru matrici foarte mari care depășesc dimensiunea memoriei RAM. Descrierea matricei va fi de asemenea salvată într-un fișier pe disc.

```
> x <- read.big.matrix("NYSE-2000-2001.tsv", sep = "\t",
type = "double", header = TRUE, backingfile = "stocks.bin",
descriptorfile = "stocks.desc")
> head(x)
exchange stock_symbol date stock_price_open stock_price_high
[1,]       NA          NA 2001             12.55            12.80
[2,]       NA          NA 2001             12.50            12.55
[3,]       NA          NA 2001             12.59            12.59
[4,]       NA          NA 2001             12.45            12.60
[5,]       NA          NA 2001             12.61            12.61
[6,]       NA          NA 2001             12.40            12.78
```

[1] Acesta poate fi descărcat de la următoarea adresă: https://s3.amazonaws.com/hw-sandbox/tutorial1/NYSE-2000-2001.tsv.gz

```
stock_price_low stock_price_close stock_volume
[1,]           12.42              12.80        11300
[2,]           12.42              12.55         4800
[3,]           12.50              12.57         5400
[4,]           12.45              12.55         5400
[5,]           12.61              12.61         1400
[6,]           12.40              12.60        18200
 stock_price_adj_close
[1,]           6.91
[2,]           6.78
[3,]           6.79
[4,]           6.78
[5,]           6.76
[6,]           6.75
```

Cu ajutorul codului prezentat aici s-a citit conținutul fișierului NYSE-2000-2001.tsv și a fost construită o matrice `x` cu elemente de tip *double*. Parametrul `header = TRUE` indică faptul că primul rând conține denumirile coloanelor din fișier și nu date efective. Parametrul `backingfile = "stocks.bin"` indică numele fișierului binar unde vor fi copiate datele matricei. Parametrul `descriptorfile = "stocks.desc"` specifică numele fișierului unde va fi salvată descrierea structurii matricei nou create.

Din afișarea primelor 6 rânduri ale conținutului matricei se constată că acele coloane din fișier care conțin valori care nu sunt de tipul declarat al elementelor matricei (*double*) au fost completate cu valoarea specială *NA* iar valoarea de pe cea de-a treia coloană a fost trunchiată. Dacă totuși se doreste a avea valori valide și pe primele trei coloane (exchange, stock_symbol respectiv date) se va proceda în felul următor: se va citi conținutul fișierului și se va crea un obiect de tip *data.frame* (ca în primul exemplu) care va fi transformat apoi într-un obiect de tip *big.matrix*. La transformare, valorile de tip șir de caractere vor fi transformate în valori de tip *factor* apoi vor fi convertite în valori numerice. În final noua matrice va fi salvată sub forma unui fișier *.csv*

```
> se_data <- read.table(file = "NYSE-2000-2001.tsv",
header = TRUE, sep = "\t")
> x1 <- as.big.matrix(se_data, type = "double")
> x1[1:3, 1:5]
exchange stock_symbol date stock_price_open stock_price_high
1        1           107  519              12.55            12.80
2        1           107  518              12.50            12.55
```

```
3        1          107  517            12.59        12.59

> write.big.matrix(x1, "stocks.csv", sep = ",", col.names = TRUE)
```

Se observă cum valorile de pe primele trei coloane pe care erau de tip șir de caracter respectiv dată calendaristică au fost transformate în valori numerice.

Se propune în continuare rezolvarea următorului exercițiu: se va pleca de la fișiserul "stocks.csv" pe care îl vom citi și se va construi un obiect `x2` de tipul *big.matrix* , apoi se va adăuga o nouă coloană matricei `x2` și se vor stoca aici valoarea medie a fiecarei tranzacții, calculată ca medie aritmetică între valoarea minimă și cea maximă a acțiunii tranzacționate. Se va calcula apoi valoarea medie a tranzancțiilor pentru fiecare companie în parte, pe toată perioada de înregistrare a datelor.

```
> library(bigmemory)
> x2 <- read.big.matrix("stocks.csv", type = "double",
header = TRUE,  backingfile = "stocks1.bin",
descriptorfile = "stocks1.desc", extraCols = "avg")
> media <- function(x) {
return ((x[,'stock_price_low'] + x[,'stock_price_high'])/2)
}
> head(x2[,'avg'])
[1] NA NA NA NA NA NA
> x2[,"avg"] = media(x2)
> head(x2[,"avg"])
[1] 12.610 12.485 12.545 12.525 12.610 12.590
> install.packages("bigtabulate")
> library(bigtabulate)
> media2 <- function (x) {
return (mean(x[,c("stock_price_high","stock_price_low")]))
 }
> indices <- bigsplit(x2, "stock_symbol")
> s <- sapply(indices, function(i) media2(x2[i,]))
> s
>  1        2        3        4        5        6
> 42.02971  6.11286 15.51390 43.54175 22.69849 47.06930
```

Fișierul *stocks.csv* este citit cu ajutorul funcției `read.big.matrix()` care crează o matrice `x2` cu elemente de tip `double`. Totodată se crează și un fișier denumit *stocks1.bin* care va memora pe suport extern (hard-disc) datele matricei, având în acest fel posibilitatea de a lucra cu

matrici mai mari decât dimensiunea memoriei RAM. Descrierea obiectului `x2` va fi salvată în fișierul *stocks1.desc*. Această descriere este utilă atunci când dorim să reutilizăm matricea `x2`, operația de încărcare în memorie fiind mult mai rapidă citind direct descrierea matricei din *stocks1.desc* si datele din fișierul *stocks1.bin*. Ultimul parametru `extraCols = "avg"` va avea ca efect adăugarea unei noi coloane matricei `x2`, în plus față de cele citite din fișier. Această coloană va avea numele *"avg"* și va fi folosită pentru a calcula valoarea medie a unei acțiuni. Se declară apoi o funcție `media()` care calculează valoarea medie a unei acțiuni ca medie aritmetică între valoarea maximă și valoarea minimă din cursul unei zile de tranzacționare. Apelarea pentru prima dată a funcției `head(x2[,'avg'])` arată că deocamdată pe coloana *"avg"* nu există nicio valoare. După instrucțiunea `x2[,'avg'] = media(x2)` se constată că pe fiecare rând al matricei a fost calculată valoarea medie.

Se instalează apoi pachetul `bigtabulate` și se declară o altă funcție `media2()` care va fi apelată pentru a calcula valoarea medie a acțiunilor pentru fiecare companie în parte. Funcția `bigsplit(x2, 'stock_symbol')` partiționează rândurile matricei în funcție de valoarea coloanei *'stock_symbol'* (simbolul companiei) creând câte un grup de indici de rânduri pentru fiecare valoare distinctă găsită pe coloana *'stock_symbol'*. În final, cu ajutorul `sapply` se aplică funcția `media2()` pe fiecare grup de indici în parte și se va calcula valoarea medie pentru fiecare grup de rânduri.

În continuare se exemplifică câteva funcții uzuale care pot fi aplicate unei matrici de tipul *big.matrix*.

```
> dimnames(x2)
[[1]]
NULL

[[2]]
 [1] "exchange"          "stock_symbol"      "date"
 [4] "stock_price_open"  "stock_price_high"  "stock_price_low"
 [7] "stock_price_close" "stock_volume"
 [9] "stock_price_adj_close"
[10] "avg"

> dim(x2)
[1] 812989     10

> typeof(x2)
```

```
[1] "double"

> rownames(x2)
NULL

> colnames(x2)
 [1] "exchange"          "stock_symbol"      "date"
 [4] "stock_price_open"  "stock_price_high"
 [6] "stock_price_low"
 [7] "stock_price_close" "stock_volume"
 [9] "stock_price_adj_close"
[10] "avg"

> nrow(x2)
[1] 812989

> ncol(x2)
[1] 10

> summary(x2)
    Length      Class       Mode
   8129890 big.matrix         S4
```

În practică apare deseori situația în care este nevoie să se efectueze o copie a unei matrici. Se va exemplifica acestă operație pornind de la matricea `x2` și se vor copia coloanele "stock_symbol", "stock_price_open", "stock_price_high", "stock_price_low" și "stock_price_close" într-o nouă matrice. Se vor selecta pentru copiere primele 1000 de rânduri din matricea `x2` și se vor copia într-o nouă matrice `x3`, descrierea obiectului `x3` fiind salvată în fișierul *x3.desc* iar datele în fișierul binar *x3.bin*.

```
> x3 <- deepcopy(x2, cols = c("stock_symbol", "stock_price_open",
"stock_price_high", "stock_price_low", "stock_price_close"),
rows = 1:1000, backingfile = "x3.bin",
descriptorfile = "x3.desc")

> options(bigmemory.allow.dimnames=TRUE)
> colnames(x3) <- c("stock_symbol", "stock_price_open",
"stock_price_high", "stock_price_low", "stock_price_close")
> nrow(x3)
[1] 1000
```

Pentru copierea unei (sub)matrici este necesară funcția `deepcopy()` întrucât în mod normal R permite copierea unui obiect iar în acest caz s-ar fi copiat obiectul `x2` care este de fapt un pointer la o structură de date care memorează matricea propriu-zisă și nu s-ar fi copiat datele care compun matricea. Parametrul `cols` permite specificarea unui subset de coloane care vor fi copiate, parametrul `rows` permite specificarea unui subset de rânduri care se vor copia iar parametrii `backingfile` și `descriptorfile` specifică fișierul binar de date respectiv descriptorul matricei. Se observă ca în urma execuției funcției `deepcopy()` au fost copiate doar datele matricei, nu și denumirile coloanelor. Denumirile coloanelor au fost restabilite ulterior cu ajutorul funcției `colnames(x3)`.

Filtrarea datelor dintr-o matrice de tip *big.matrix* se realizează foarte eficient cu ajutorul funcției `mwhich()` care evită crearea unor masive de date temporare ca în cazul funcției `which()`. Obiectele de tipul *big.matrix* pot avea dimensiuni foarte mari iar crearea unor masive temporare necesare operațiilor de filtrare poate conduce la epuizarea rapidă a memoriei RAM. Sintaxa funcției `mwhich()` este prezentată în continuare.

```
mwhich(x, cols, vals, comps, op = 'AND')
```

Semnificația argumentelor este următoarea:

- `x` matricea asupra căreia se aplică operația de filtrare;
- `cols` un vector care conține indici sau nume de coloane;
- `vals` o listă de vectori de lungime 1 sau 2, fiecare element din listă corespunzând unui element din `cols`; elementele vectorii de lungime 1 vor fi folosite pentru a testa egalități în timp ce elementele din vectorii de lungime 2 vor fi folosite pentru testarea inegalităților;
- `comps` o listă de operatori de comparare, câte unul pentru fiecare element din `cols`:
  - 'eq' operatorul egalitate;
  - 'neq' operatorul inegalitate;
  - 'le' operatorul $\leq$;
  - 'lt' operatorul $<$;
  - 'ge' operatorul $\geq$;
  - 'gt' operatorul $>$;

- op poate fi AND sau OR, fiind folosit pentru combinarea rezultatelor operațiilor de comparare.

Funcția mwhich() întoarce ca rezultat un vector cu indicii rândurilor care satisfac criteriile de filtrare. În continuare sunt prezentate câteva exemple de utilizare a acestei funcții.

```
> index1 <- mwhich(x3, c(3, 3, 4), list(10, 12, 12),
list("gt", "lt", "lt"), "AND")
```

Din matricea x3 se selectează rândurile pentru care valorile de pe coloana 3 sunt mai mari decât 10 și mai mici decât 12 iar valorile de pe coloana 4 sunt mai mici decât 12.

```
> index2 = mwhich(x3, 3:4, list(12, 12), list("eq", "eq"), "AND")
> index2
[1] 166
> head(x3[index2,])
     stock_symbol  stock_price_open  stock_price_high
              107                12                12

     stock_price_low stock_price_close
     12                12
```

Din matricea x3 sunt selectate rândurile pentru care au valoarea 12 pe coloana 3 și tot valoarea 12 pe coloana 4.

```
> index3 <- mwhich(x3, 4, 15, "gt")
> head(x3[index3, ])
       stock_symbol   stock_price_open  stock_price_low
[1,]   53             18.0              18.65
[2,]   53             18.4              18.66
[3,]   53             17.9              18.49
[4,]   53             17.4              18.60
[5,]   53             17.5              17.86
[6,]   53             17.6              18.00
  stock_price_high stock_price_close
[1,] 18.00          18.65
[2,] 17.90          17.90
[3,] 17.89          18.49
[4,] 17.40          17.90
```

```
[5,] 17.50          17.50
[6,] 17.60          17.70
```

Din matricea `x3` sunt selectate rândurile pentru care valorile de pe coloana 4 sunt mai mari decât 15.

Prelucrarea datelor memorate în obiecte de tipul *big.matrix* se poate realiza cu funcții incluse în pachete specializate. Se va exemplifica estimarea unui model de regresie liniar pentru cazul în care datele sunt memorate în obiecte *big.matrix*. Estimarea modelelor de regresie liniare în cazul în care datele au dimensiuni rezonabile se realizeaza cu ajutorul funcției `lm()`. În cazul în care se lucrează cu date de tipul *big.matrix* care pot avea dimensiuni foarte mari utilizarea funcției `lm()` nu mai este posibila. Pentru această situație se va folosi o funcție echivalentă denumită `biglm.big.matrix()` care se găsește în pachetul `biganalytics`. În exemplul prezentat mai jos am utilizat setul de date *trees* și se estimează modelul de regresie următor:

$$Volume_i = a \cdot Girth_i + b \cdot Height_i + \epsilon_i$$

```
> install.packages("biganalytics")
> library(biganalytics)
> data(trees)
> head(trees)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
> mat <- as.big.matrix(trees)
> lin_Model <- biglm.big.matrix(Volume ~ Girth + Height, data = mat)
> summary(lin_Model)
Large data regression model: biglm(formula = formula,
data = data, ...)
Sample size =  31
                Coef     (95%      CI)     SE      p
(Intercept) -57.9877 -75.2641 -40.7112 8.6382 0.0000
Girth         4.7082   4.1796   5.2367 0.2643 0.0000
Height        0.3393   0.0789   0.5996 0.1302 0.0091
```

Matricile *big.matrix* pot fi partajate între mai multe sesiuni R și pot fi

prelucrate în paralel. În exemplul următor se vor starta două sesiuni R și se va lucra cu aceeași matrice care va fi partajată între cele două procese. În prima sesiune R se rulează următoarea secvență de cod:

```
> library(bigmemory)
> library(biganalytics)
> x <- read.big.matrix("stocks.csv", sep = ",", shared = TRUE,
header = TRUE)
> descriptor <- describe(x)
> dput(descriptor, "desc_partajare.txt")
> media <- colmean(x, 4)
> media
stock_price_open
        27.39365
```

Au fost citite datele din fișierul *stocks.csv* creat anterior și s-a construit matricea `x`. În variabila `descriptor` s-a obținut descriptorul matricei `x` care a fost salvat în fișierul *desc_partajare.txt* iar apoi s-a aplicat funcția `colmean()` pentru calculul mediei valorilor de pe coloana 4. Matricea `x` a fost creată cu parametrul `shared = TRUE` ceea ce permite ca datele matricei să poată fi partajate între mai multe procese.

Se va starta o a doua sesiune R și se va executa următoarea secvență de cod:

```
> library(bigmemory)
> library(biganalytics)
> d <- dget("desc_partajare.txt")
> y <- attach.big.matrix(d)
> media2 <- colmean(y, 4)
> media2
stock_price_open
        27.39365
```

În variabila `d` se citește descrierea matricei salvată în fișierul *desc_partajare.txt* de către prima instanță R apoi cu ajutorul funcției `attach.big.matrix()` se crează o nouă matrice `y` bazată pe informațiile salvate în descriptorul `d`, cu aceleași date ca și matricea `x` din prima instanță R. Acest lucru este posibil deoarece datele matricei `x` sunt memorate într-o zonă de memorie partajată care poate fi accesată de mai multe procese. În final, se aplică funcția `colmean()` pentru calculul mediilor valorilor de pe coloana 4 și se observă că se obține același rezultat ca și în prima instanță, ceea ce confirmă faptul că cele două instanțe R folosesc aceleași date partajate.

Folosirea matricilor de tipul *big.matrix* ca argumente ale funcțiilor trebuie facută cu precauție deoarece o astfel de variabilă este de fapt un pointer la o structură de date. Dacă o funcție modifică valorile elementelor matricei, aceste modificări vor avea efect și la ieșirea din funcția respectivă ceea ce este echivalent cu faptul că funcția utilizează apelul prin referință în cazul acestui tip de date.

16.2.2 Pachetul "ff"

Pachetul `ff` (D. Adler, 2007) a fost proiectat pentru a facilita prelucrarea volumelor mari de date, astfel încât limitările sistemului R amintite în secțiunea 16.2 să poată fi depășite. Memorarea datelor se realizează în fișiere binare ("ff" - "flat files") într-un format nativ, pe hard disc, permițând astfel lucrul cu vectori și masive de date care nu pot fi încărcate în memoria RAM. Vectorii și masivele de date sunt citite de pe disc iar în memorie sunt încărcate doar porțiuni care sunt necesare prelucrărilor. Obiectele create cu ajutorul funcțiilor din pachetul `ff` sunt văzute de utilizatori ca vectori sau masive de date R obișnuite care pot fi accesate în mod uzual cu ajutorul notației `[ ]` cu toate că datele nu sunt încărcate în întregime în memoria RAM. Pachetul `ff` furnizează utilizatorilor tipurile de date *ff, ffdf* care sunt analogul vectorilor, matricilor multidimensionale și tipului *data.frame*, având posibilitatea de a importa sau exporta datele din sau în fișiere externe. Analog pachetului `bigmemory` fișierele *ff* pot fi utilizate partajat de mai multe sesiuni R. Va fi prezentată utilizarea pachetului `ff` prin câteva exemple simple.

```
> install.packages("ff")
> library("ff")
> v <- ff(vmode = "double", length = 100)
> v
ff (open) double length=100 (100)
  [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]        [93]  [94]
    0     0     0     0     0     0     0     0     :     0     0
 [95]  [96]  [97]  [98]  [99] [100]
   0     0  0     0     0     0
> v[1:100] <- 5
> v
ff (open) double length=100 (100)
  [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]        [93]  [94]
    5     5     5     5     5     5     5     5     :     5     5
 [95]  [96]  [97]  [98]  [99] [100]
```

```
  5     5     5     5     5     5
> length(v)
[1] 100
```

În acest exemplu a fost creat un vector `v` de tipul *ff* cu 100 de elemente de tipul *double* apoi s-au inițializat toate elementele vectorului cu valoarea 5. Se observă că elementele vectorului se accesează la fel ca și în cazul unui vector obișnuit.

```
> mat <- ff(vmode = "double", dim = c(3,10))
> mat
ff (open) double length=30 (30) dim=c(3,10) dimorder=c(1,2)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    0    0    0    0    0    0    0    0     0
[2,]    0    0    0    0    0    0    0    0    0     0
[3,]    0    0    0    0    0    0    0    0    0     0

> mat[,3] <- 6
> mat
ff (open) double length=30 (30) dim=c(3,10) dimorder=c(1,2)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    0    6    0    0    0    0    0    0     0
[2,]    0    0    6    0    0    0    0    0    0     0
[3,]    0    0    6    0    0    0    0    0    0     0


> mat2 <- ff(filename="mat2.bin", vmode="double", dim=c(3,10))
```

Obiectul `mat` este o matrice cu 3 linii și 10 coloane cu elemente de tip *double* iar coloana 3 a fost inițializată cu valoarea 6 și apoi a fost afișat conținutul matricei. Obiectul `mat2` este tot o matrice cu 3 linii și 10 coloane dar datele sunt memorate în fișierul *mat2.bin* specificat cu ajutorul parametrului `filename`. În acest fel se poate lucra cu masive de date care depășesc capacitatea memoriei RAM.

Următorul exemplu crează un obiect de tipul *ffdf* care este echivalentul unui obiect de tip *data.frame*. Se va folosi setul de date `rock` inclus în R.

```
> data(rock)
> a <- ff(rock$area)
> p <- ff(rock$peri)
> s <- ff(rock$shape)
```

```
> pr <- ff(rock$perm)
> ffrocks <- ffdf(area = a, peri = p, shape = s, perm = pr)
```

code ffrocks este un masiv de date de tipul *ffdf* inițializat cu valorile obținute din setul de date `rock` încărcat în memoria RAM. Dacă dorim să construim un obiect *fffd* cu date dintr-un fișier de pe disc se folosește funcția `read.csv.ffdf()` ca în exemplul următor în care se citesc date din fișierul *NYSE-2000-2001.tsv* utilizat și în secțiunea precedentă.

```
> s <- read.csv.ffdf(file = "NYSE-2000-2001.tsv", header = TRUE,
sep = "\t")
```

Argumentul `file` specifică numele fișierului de unde vor fi citite datele, `header = TRUE` indică faptul că primul rând conține denumirile coloanelor iar `sep = ""` indică faptul că valorile din fișier sunt separate cu caracterul "tab". Operația inversă, cea de salvare a unui masiv *ffdf* într-un fișier se realizează cu ajutorul funcției `write.csv.ffdf()` ca în exemplul următor în care datele din `ffrocks` sunt salvate într-un fișier de tip *.csv* denumit *rocks.csv*.

```
> write.csv.ffdf(ffrocks, file = "rocks.csv")
```

Datele din masivele de date `ff` sau `ffdf` pot fi prelucrate cu ajutorul funcțiilor din pachetul `biglm`. Pentru exemplificare se va estima următorul model de regresie liniară plecând de la datele încarcate în masivul `ffrocks`:

$$Permi = a \cdot Area_i + b \cdot Peri_i + c \cot Shape_i + \epsilon_i$$

```
> install.packages("biglm")
> library(biglm)
> lm <- biglm(perm ~ area + peri + shape, data = ffrocks)
> summary(lm)
Large data regression model: biglm(perm ~ area + peri +
shape, data = ffrocks)
Sample size =  48
                Coef      (95%       CI)       SE      p
(Intercept) 485.6180  168.8015  802.4345 158.4083 0.0022
area          0.0913    0.0413    0.1413   0.0250 0.0003
peri         -0.3440   -0.4463   -0.2418   0.0511 0.0000
shape       899.0693 -114.8327 1912.9712 506.9510 0.0761
```

Utilizarea funcției `biglm()` aici are numai scop demostrativ întrucât `ffrocks` are doar 48 de linii. În cazuri practice în care volumul datelor este foarte mare se vor aplica tehnici speciale. În exemplul următor se va genera o matrice *ff* cu 100000000 linii și trei coloane denumite "Y", "X1" și respectiv "X2". Se va inițializa apoi matricea cu numere aleatoare și în final se va estima modelul de regresie următor:

$$Y_i = a \cdot X1_i + b \cdot X2_i + \epsilon_i, \qquad i = 1...100000000 \tag{16.1}$$

```
> nr <- 100000000
> nc <- 3
> block = 100000
> a <- ff(vmode="double", dim=c(nr,nc),
dimnames = list(NULL, c("Y", "X1", "X2")))
> ffrowapply(
 {
 n <- i2 - i1 + 1
 epsilon <- rnorm(n)
 a[i1:i2,] <- epsilon + matrix(rnorm(n*nc), n, nc)
 },
 X = a, BATCHSIZE = block)

> model <- Y ~ X1 + X2
> first <- TRUE

> ffrowapply({
   if (first){
     first <- FALSE
     m <- biglm(model, as.data.frame(a[i1:i2,,drop = FALSE]))
   } else {
     m <- update(m, as.data.frame(a[i1:i2,,drop = FALSE]))
   }
 },
 X = a, BATCHSIZE = block)

> summary(m)
Large data regression model: biglm(model,
as.data.frame(a[i1:i2, ,
drop = FALSE]))
Sample size =  100000000
              Coef    (95%    CI)    SE      p
(Intercept) 0.0002 -0.0001 0.0004 1e-04 0.1248
X1          0.3334  0.3332 0.3336 1e-04 0.0000
```

```
X2          0.3333  0.3331 0.3335 1e-04 0.0000
```

Variabila `nr` conține numărul de rânduri iar variabila `nc` numărul de coloane ale matricei `a` de tipul *ff* care va fi declarată în continuare. Variabila `block` care are valoarea 100000 este folosită pentru a declara dimensiunea (exprimata în număr de rânduri) a unui bloc din matricea `a` care va fi prelucrat la un moment dat. Divizarea matricei și prelucrarea ei pe blocuri de dimensiuni mai reduse permite prelucrarea unor matrici foarte mari și evitarea epuizării memoriei RAM. În continuare se construiește matricea `a` cu elemente de tip *double* cu `nr` linii și `nc` coloane. În plus, coloanele matricei vor avea și denumiri: prima coloană se va numi "Y", următoarele "X1" respectiv "X2". Matricea `a` va fi inițializat prin aplicarea funcției `ffrowapply()`. Această funcție permite prelucrarea matricei pe blocuri pentru a evita epuizarea memoriei RAM. Primul argument al funcției `ffrowapply()` este o expresie (în cazul nostru cea inclusă între { }) care se va aplica pe blocuri formate din rândurile cu indicii `i1:i2`. Argumentul *X* primește ca valoare obiectul *ff* care va fi prelucrat, în cazul nostru matricea `a`. Expresia care se execută repetat pentru fiecare bloc al matricei are ca scop generarea unor valori folosind distribuția normală de medie 0 și abatere standard 1 (`rnorm`). În continuare se declară o variabilă booleană `first` inițializată cu valoarea TRUE și variabila `model` care conține modelul liniar ce urmează a fi estimat. Pentru estimarea modelului se va utiliza tot funcția `ffrowapply()` care estimează modelul pentru primul bloc de date cu ajutorul funcției `biglm()` apoi estimările sunt actualizate prin aplicarea succesivă a funcției `update()` pentru următoarele blocuri de date.

Utilizarea pachetului `ff` sau `bigmemory` atunci când trebuie prelucrate volume de date foarte mari rămâne la alegerea utilizatorului, ambele pachete prezentând performanțe similare. Folosind funcția `system.time()` care indică timpul necesar execuției unei secvențe de cod se vor testa performanțele celor două pachete pentru câteva operații: calculul valorii maxime și a valorii medii de pe o coloană a unei matrici. Din rezultatul afișat în secvența de cod următoare se poate constata că timpul de execuție este similar în cazul utilizării celor două pachete.

```
> library(bigmemory)
> library(ff)

> # cu pachetul ff
> nr <- 10000000
> nc <- 3
```

```
> block <- 10000
> x1 <- ff(vmode = "double", dim = c(nr,nc))
> ffrowapply(
{
n <- i2 - i1 + 1
epsilon <- rnorm(n)
x1[i1:i2,] <- epsilon + matrix(rnorm(n*nc), n, nc)
},
X = x1, BATCHSIZE = block)

> system.time(max(x1[,3]))
   user  system elapsed
   0.20    0.05    0.25
> system.time(mean(x1[,2]))
   user  system elapsed
   0.19    0.04    0.24

> # cu pachetul bigmemory
> x2 <- big.matrix(nr, nc, type = "double", backingfile = "x2.bin",
descriptorfile = "x2.desc")
> for(i in 1:3) {
x2[,i] <- x1[,i]
 }
> system.time(max(x2[,3]))
   user  system elapsed
   0.23    0.05    0.29
> system.time(mean(x2[,2]))
   user  system elapsed
   0.16    0.06    0.22
```

16.2.3 Alte pachete R pentru calculul paralel

În cadrul acestei secțiuni va fi prezentat pachetul `mapReduce` care implementează în R un model de programare distribuită denumit *Map-Reduce*. Acest model prezentat pentru prima data în (Dean şi Ghemawat, 2004) are la bază divizarea unui job de dimensiuni mari în mai multe joburi de dimensiuni mai mici care produc date de ieșire, apoi datele de ieșire produse de fiecare job sunt combinate într-un singur rezultat. Această abordare corespunde modelului clasic de programare *divide and conquer*. Joburile rezultate în urma fazei de divizare pot fi distribuite pentru execuție mai multor calculatoare, mai multor procesoare de pe același calculator sau mai multor nuclee (core) ale

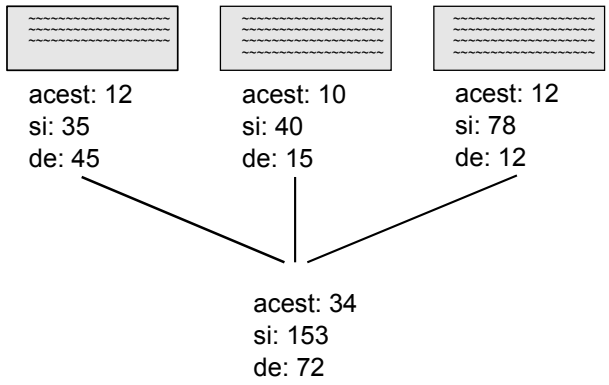


Figura 16.1: Modelul Map Reduce

aceluiași procesor. Modelul *Map-Reduce* presupune rezolvarea unei probleme în doi pași:

- primul pas denumit *map*: se execută aceeași operație în paralel pe setul de date inițial iar ca rezultat se va genera câte o pereche de tipul cheie-valoare pentru fiecare înregistrare (rând) din datele de intrare;
- al doilea pas este denumit *reduce* și constă în gruparea elementelor cu aceeași cheie și calcularea unei valori pentru fiecare grup.

Un exemplu clasic de aplicare a acestei paradigme de programare constă în numărarea aparițiilor fiecărui cuvânt într-un text. Presupunând că există mai multe cărți (texte) și se dorește o listă cu toate cuvintele din aceste cărți și numărul de apariții al fiecărui cuvânt, procesul de calcul se poate desfășura ca în figura 16.1. În prima etapă datele sunt împărțite în mai multe seturi și pentru fiecare set (text) se generează o serie de perechi (cuvânt, nr_apariții). În a doua etapă, aceste perechi sunt grupate după cheie (cuvânt) iar pentru toate perechile cu aceeași cheie numărul de apariții este însumat.

Întrucât pachetul `mapReduce` nu este disponibil în arhiva standard CRAN, acest pachet va fi instalat plecând de la codul sursă [2]. Presupunând că fișierul cu codul sursă al pachetului, mapReduce_1.2.6.tar.gz, a fost salvat în directorul curent de lucru, acesta se poate instala astfel:

```
> install.packages("mapReduce_1.2.6.tar.gz", repos = NULL,
```

[2]codul sursă se poate descărca de la următoarea adresa: http://cran.r-project.org/src/contrib/Archive/mapReduce/mapReduce_1.2.6.tar.gz

```
type = "source")
* installing *source* package 'mapReduce' ...
** package 'mapReduce' successfully unpacked and MD5 sums
** checked
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded

* DONE (mapReduce)
```

Acest pachet implementează practic o singură funcție și anume:

```
mapReduce( map, ..., data, apply = sapply)
```

unde argumentul `ap` este o expresie care în urma evaluării are ca rezultat un vector ce va fi folosit apoi la divizarea datelor de intrare în părți care vor fi prelucrate în paralel. Următorul argument reprezentat aici cu `...` reprezintă una sau mai multe expresii care vor fi evaluate în etapa *reduce* pe fiecare partiție a datelor separat. `data` reprezintă datele de intrare care pot fi un vector, o matrice sau un obiect de tipul `data.frame`. Ultimul argument `apply` reprezintă o funcție care va fi utilizată pentru paralelizarea joburilor *reduce*. Implicit acest argument are valoarea `sapply`, dar se poate folosi orice altă funcție care poate executa joburile în paralel pe mai multe nuclee ale aceluiași procesor sau pe procesoare diferite conectate între ele care sunt disponibile în pachete specializate precum *multicore* sau *snow*. Se va exemplifica utilizarea modelului de programare *Map-Reduce* pe setul de date `iris` care face parte din distribuția R.

```
> library(mapReduce)
> data(iris)
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
> mapReduce(map = Species, min_petal_length = min(Petal.Length),
average_petal_length = mean(Petal.Length),
max_petal_length = max(Petal.Length), data = iris)
      min_petal_length average_petal_length max_petal_length
setosa             1.0              1.462              1.9
versicolor         3.0              4.260              5.1
virginica          4.5              5.552              6.9
```

`iris` este un obiect de tipul *dataframe* care conține informații despre mai multe specii de irișI al căror nume este memorat pe coloana "Species". În exemplul prezentat aici, setul de date este divizat în mai multe partiții după numele speciei de iris în prima fază *map*. În continuare au fost specificate trei funcții care se vor aplica în faza *reduce* și care calculează valoarea minimă, maximă și medie a petalelor pentru fiecare specie de iris.

16.3 Introducere în Hadoop

Hadoop este un sistem software de tip open source care are ca principal scop prelucrarea distribuită a seturilor mari de date utilizând clustere de calculatoare (White, 2012). *Hadoop* este dezvoltat în limbajul de programare Java și este o platformă de tip *middleware* care se rulează pe un cluster de stații de lucru. Aplicațiile care folosesc platforma *Hadoop* pot fi dezvoltate în limbajul Java dar și în alte limbaje precum R, Ruby sau Phyton. *Hadoop* poate fi descărcat gratuit de la adresa http://hadoop.apache.org. Printre utilizatorii platformei *Hadoop* se numără companii precum Yahoo![3] sau Facebook (Vagata şi Wilfong, 2014). Sistemul *Hadoop* este alcătuit în principal din:

- Hadoop Distributed File System (HDFS) - un sistem de fișiere distribuit, de mare performanță;
- Hadoop YARN - un subsistem care are ca rol planificarea joburilor și managementul resurselor clusterului de calculatoare;
- Hadoop Map-Reduce - un sistem de procesare paralelă a seturilor foarte mari de date care implementează modelul de programare distribuită *MapReduce*.

Descris pe scurt, *Hadoop* este un sistem software care furnizează utilizatorilor săi un sistem de fișiere distribuit foarte fiabil și un sistem

[3]Hadoop at Yahoo!, Yahoo! Developer Network, 2014, http://developer.yahoo.com/hadoop/

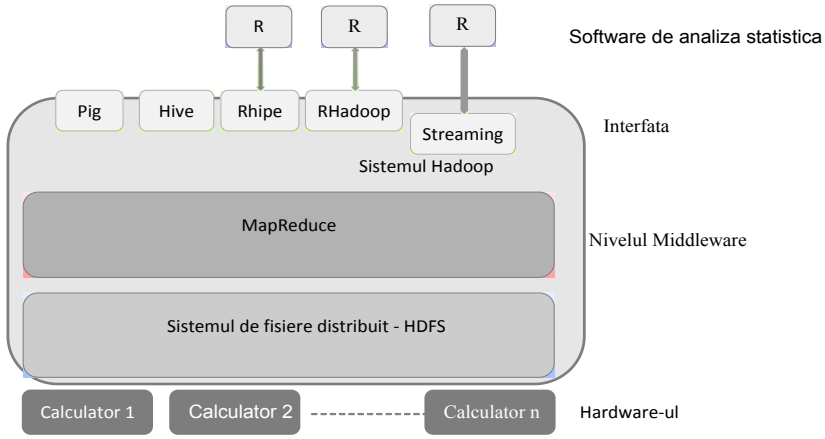


Figura 16.2: Sistemul software Hadoop

de analiză și prelucrare a datelor. *Hadoop* poate fi instalat și rulat atât pe clustere cu câteva calculatoare cât și pe clustere cu mii de calculatoare, cu un grad de toleranță la defecte foarte ridicat. În momentul actual, *Hadoop* este un standard *de-facto* în procesarea și stocarea volumelor foarte mari de date și este folosit de toți marii actori din industria software. Structura sistemului *Hadoop* poate fi urmărită în figura 16.2.

16.3.1 Sistemul de fișiere distribuit HDFS

Sistemul de fișiere HDFS este bazat pe o arhitectură de tip client-server. Este un sistem de fișiere cu toleranță ridicată la erori și este proiectat să poată fi rulat pe calculatoare cu resurse limitate. HDFS furnizează acces la date cu viteză ridicată, fiind ideal pentru aplicații care lucrează cu volume mari de date, de ordinul sutelor de GB sau TB. Sistemul de fișiere HDFS este un sistem de tip ”append only”, adică un fișier care a fost creat, populat cu date și apoi închis nu mai poate fi modificat ulterior. Această caracteristică simplifică modul de asigurare a coerenței datelor din fișiere. HDFS furnizează aplicațiilor facilități de aducere a prelucrărilor acolo unde sunt stocate datele deoarece este mult mai eficient să fie migrate instrucțiunile de prelucrare a datelor decât datele. Acest lucru reduce din traficul de date prin rețeaua de interconectare a calculatoarelor crescând astfel eficiența aplicațiilor. HDFS constă într-un nod (server) denumit `NameNode` care este rulat pe un server master și unul sau mai multe noduri (clienți) de tip `DataNode` care gestionează unitățile de stocare ale datelor atașate respectivelor noduri (calculatoare) din rețea. `NameNode` are scopul

de a gestiona spațiul de nume al sistemului de fișiere HDFS și a realiza operații de deschidere, închidere sau redenumire a fișierelor. Fișierele care vor fi memorate de HDFS sunt divizate în mai multe blocuri de date care sunt stocate de mai multe `DataNode`, responsabile cu realizarea operațiilor efective de scriere/citire a datelor. Maparea blocurilor de date pe noduri se realizează de către `NameNode`. Atât aplicația `NameNode` cât și `DataNode` sunt scrise în Java și pot fi rulate practic pe orice calculator care suportă Java.

Toleranța la erori hardware este obținută prin replicarea blocurilor de date pe mai multe calculatoare. Toate fișierele sunt divizate în blocuri de date de dimensiune egală care sunt apoi distribuite pe `DataNode`. Un bloc de date este copiat pe mai multe noduri, astfel încât dacă un nod nu mai poate funcționa din cauza unei erori hardware, copii ale datelor sunt disponibile de alte noduri din rețea.

16.3.2 Subsistemul Map-Reduce

Peste sistemul de fișiere distribuit HDFS rulează nucleul care implementează modelul de programare `MapReduce`. Acesta constă într-un proces denumit JobTracker care primește de la clienți joburi de tip `MapReduce` în vederea planificării acestora pentru execuție. Procesul JobTracker trimite prelucrările (job-uri) către procesele de tip TaskTracker care rulează pe nodurile din clusterul de calculatoare, încercând să mențină prelucrările cât mai aproape de datele care trebuiesc prelucrate. Dacă un proces TaskTracker nu răspunde într-un anumit interval de timp prestabilit sau se termină cu eroare, procesul JobTracker va replanifica respectivele prelucrări. Fiecare proces de tip TaskTracker pornește câte o mașină virtuală Java pentru fiecare job în parte pentru a evita ca TaskTracker-ul însuși să își încheie execuția dacă job-ul care trebuie executat va conduce la terminarea execuției mașinii virtuale Java în caz de eroare. TaskTracker-ul și JobTracker-ul comunică periodic pentru actualizarea stării sistemului. Structura sistemului Hadoop poate fi urmarită în figura 16.3.

16.4 Integrarea între R și Hadoop pentru procesarea volumelor mari de date

În prezent există un număr foarte mare de pachete R sau script-uri de prelucrare și analiză a datelor. Utilizarea acestora împreună cu Hadoop ar presupune în mod normal rescrierea lor în Java, limbajul natural pentru Hadoop, însă activitatea de rescriere poate conduce la multe

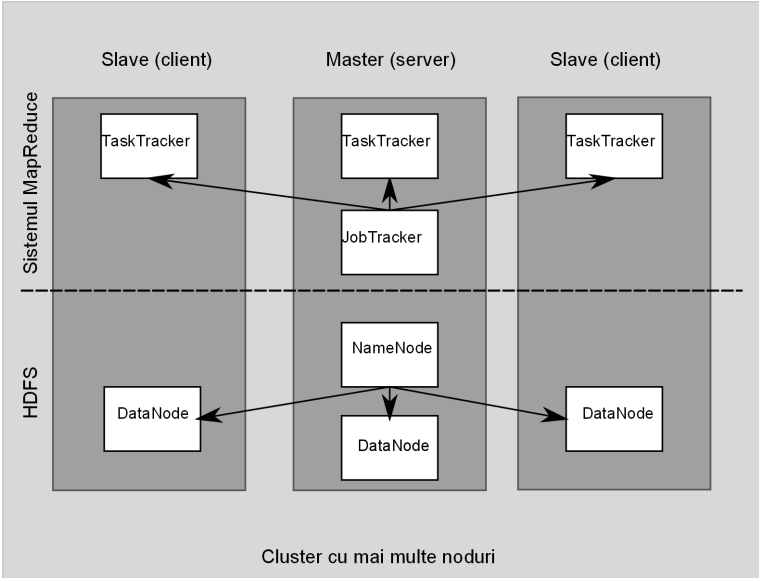


Figura 16.3: Structura sistemului software Hadoop

erori. De aceea este mult mai eficient să se interfațeze sistemul Hadoop cu R astfel încât să se poată prelucra cu script-uri R date memorate cu Hadoop (Holmes, 2012). Un alt motiv pentru a construi o interfață între R și Hadoop constă în faptul că R încarcă datele în memorie în vederea prelucrărilor ceea ce poate fi o limitare serioasă în ceea ce privește volumul datelor prelucrate. Pentru exemplele din secțiunile următoare s-a folosit o mașină virtuală Oracle VM VirtualBox în care s-a importat o imagine (sandbox) ce conține preinstalată o distribuție Hadoop. Mașina virtuală VirtualBox poate fi descărcată gratuit[4] iar procesul de instalare este foarte simplu. Distribuția de Hadoop utilizată în exemplele din acest capitol este cea furnizată de Hortonworks. De la adresa hortonworks.com/hdp/downloads se descarcă fișierul imagine pentru VirtualBox: *Hortonworks+Sandbox+2.0+VirtualBox.ova*. În continuare se pornește mașina virtuală Oracle VM VirtualBox și se importă fișierul imagine astfel: *File - Import Appliance* apoi se selectează fișierul *Hortonworks+Sandbox+2.0+VirtualBox.ova*. Acest fișier conține o imagine a sistemului de operare CentOS (care este o versiune de Linux) cu o distribuție de Hadoop preinstalată. Un tutorial simplu referitor la această distribuție se poate găsi la adresa: http://hortonworks.com/hdp/docs. Mașina virtuală se pornește

[4]Oracle VirtualBox se poate descărca de la adresa https://www.virtualbox.org/wiki/Downloads

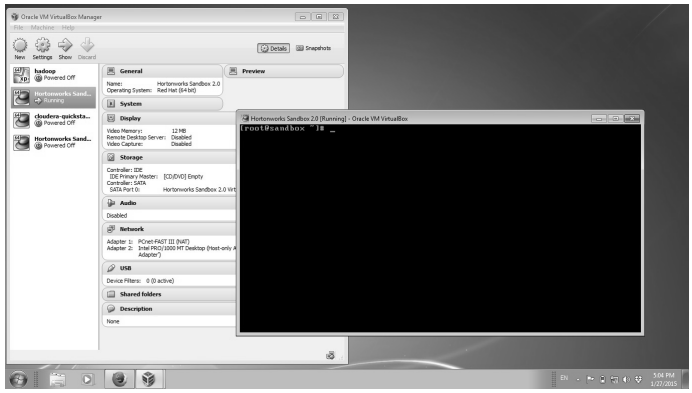


Figura 16.4: Mașina virtuală care rulează Hadoop

selectând *Machine* - *Start* din meniul principal al Oracle VM VirtualBox.

16.4.1 R și Streaming

Streaming reprezintă o tehnologie integrată în Hadoop care permite rularea job-urilor `MapReduce` cu ajutorul oricărui script sau program executabil care este capabil să citească și să scrie date de la consola standard (*stdin* respectiv *stdout* folosind denumirile consacrate deja in mediul Unix/Linux). Aceasta înseamnă că se poate folosi tehnologia Streaming împreună cu script-uri R atât în faza Map cât și în faza Reduce întrucât script-urile R pot citi/scrie date de la stdin/stdout. Se vor folosi comenzile Hadoop pentru a lansa în execuție job-uri care folosesc Streaming, argumentele plasate în linia de comandă indicând script-urile care rulează funcția Map respectiv funcția Reduce. Pentru exemplificare se va porni mașina virtuală Oracle Virtual Box și se va realiza o conexiune la consola Linux (CentOS) folosind drept username *root* iar ca parolă *hadoop*.

O linie de comandă care rulează un job `MapReduce` cu cele două funcții implementate în script-uri R arată astfel:

```
$ ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/contrib
/streaming/*.jar \
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input input_data.txt \
-output output \
-mapper /home/tst/src/map.R \
```

```
-reducer /home/tst/src/reduce.R \
-file /home/tst/src/map.R \
-file /home/tst/src/reduce.R
```

Variabila HADOOP_HOME trebuie să conțină calea către directorul unde este instalat Hadoop. În acest exemplu se presupune că fișierul cu datele de intrare denumit *input_data.txt* a fost deja copiat din sistemul de fișiere local în sistemul HDFS. Semnificația parametrilor din linia de comandă este următoarea:

- `inputformat org.apache.hadoop.mapred.TextInputFormat` specifică formatul datelor de intrare (în cazul de față un fisier text);
- `input input_data.txt` specifică fișierul care conține datele de intrare;
- `output` specifică directorul de ieșire; în acest director vor fi scrise rezultatele job-ului;
- `mapper /home/tst/src/map.R` specifică script-ul care implementează funcția Map; în exemplul prezentat acest script este denumit map.R și este localizat în directorul /home/tst/src/;
- `reducer /home/tst/src/reduce.R` specifică script-ul care implementează funcția Reduce; în exemplul prezentat acest script este denumit reduce.R și este localizat în directorul /home/tst/src/;
- `file /home/tst/src/map.R` indică faptul că script-ul Map.R trebuie copiat în sistemul de fișiere distribuit pentru a fi disponibil mașinilor/nodurilor care vor rula job-ul map-reduce;
- `file /home/tst/src/reduce.R` indică faptul că script-ul Map.R trebuie copiat în sistemul de fișiere distribuit pentru a fi disponibil mașinilor/nodurilor care vor rula job-ul map-reduce.

Se va exemplifica utilizarea tehnologiei HadoopStreaming împreună cu R pe două exemple concrete. Primul exemplu este foarte simplu și constă în calcularea Se vor scrie două scripturi R, unul care implementează funcția `map()`, celălalt funcția `reduce()`. Primul script denumit sugestiv *map.R* este redat în continuare:

```
#! /usr/bin/Rscript
```

```
library(HadoopStreaming)
args <- c()
arg <- hsCmdLineArgs(args, openConnections = TRUE)
mapper <- function(x) {
     # 1. se despart liniile de text în cuvinte care sunt
     # memorate într-un vector
    words <- unlist(strsplit(x, " "))
     # 2. dacă există cuvinte care nu conțin nimic ele
     # sunt eliminate
    words <- words[!(words == '')]
     # 3. se crează un obiect de tip dataframe cu o singură
     # coloană 'Word'
    df <- data.frame('Word' == words)
     # 4. se adaugă o coloană denumită 'Count' inițializată
     # cu valoarea 1
    df[,'Count'] = 1
     # se scrie obiectul dataframe la consolă pentru a fi
     # citit de către reducer
    hsWriteTable(df[,c('Word','Count')], file = arg$outcon,
    sep=',')
}

# se citește o linie de text de la consolă
hsLineReader(arg$incon, chunkSize = arg$chunksize, FUN = mapper)

if (!is.na(arg$infile)) {
  close(arg$incon)
}

if (!is.na(arg$outfile)) {
  close(arg$outcon)
}
```

Pentru a putea executa acest script trebuie în prealabil instalat pachetul HadoopStreaming. Scriptul începe cu încărcarea pachetului HadoopStreaming după care se creează un vector `args` care este transmis ca argument funcției `hsCmdLineArgs`. Această funcție preia argumentele din linia de comandă cu care este rulat scriptul map.R permițând specificarea fișierelor de intrare, ieșire, separatorul între valori și opțional poate deschide conexiunile de intrare/ieșire. În continuare se declară o funcție denumită `mapper()` care preia o linie de text din fișierul de intrare, separă cuvintele, creează un obiect de tip dataframe cu două coloane, pe prima memorând cuvintele textului iar

pe a doua valoarea 1. Această funcție creează deci perechi (cheie, valoare) unde cheia este reprezentată de fiecare cuvânt iar drept valoare avem deocamdată 1. Aceste perechi sunt apoi scrise în fișierul de ieșire, adică la consolă cu ajutorul funcției `hsWriteTable()`. Scriptul map.R conține apoi un apel al funcției `hsLineReader()` care citește o linie din fișierul de intrare (specificat de argumentul `arg$incon`) și aplică funcția `mapper()` pe această linie (`FUN = mapper`).

Al doilea script denumit reduce.R definește funcția `reducer()` care însumează numărul aparițiilor pentru fiecare cuvânt și afișează pe ecran cuvântul urmat de numărul de apariții. În cadrul acestui script se apelează funcția `hsTableReader()` care citește din fișierul de intrare (specificat prin argumentul arg$incon) pachete de date cu care crează obiecte de tip `data.frame` asigurându-se că toate rândurile care conțin aceeași valoare a cheii sunt împachetate în același obiect `data.frame` (prin argumentul `singleKey = TRUE`) și trimite acest obiect unei funcții specificate prin argumentul `FUN`.

```
#!/usr/bin/Rscript
library(HadoopStreaming)
args <- c()
arg <- hsCmdLineArgs(args, openConnections = TRUE)

reducer <- function(d) {
    cat(d[1,'Word'], sum(d$Count), '\n', sep = ',')
}
# definim numele coloanelor și tipurile lor
cols = list(Word = '', Count = 0)
hsTableReader(arg$incon, cols, chunkSize = arg$chunkSize,
    skip = 0,
    sep = ',', keyCol = 'Word', singleKey = T, ignoreKey = F,
    FUN = reducer)

# inchid fisiere deschise (stdin, stdout)
if (!is.na(arg$infile)) {
  close(arg$incon)
}

if (!is.na(arg$outfile)) {
  close(arg$outcon)
}
```

Pentru a rula cele două scripturi ca un job `MapReduce` se va porni mașina virtuală care conține imaginea Hadoop, și se va face conectarea la consolă

folosind drept username *root* și parolă *hadoop*. Pentru scopul de a testa scripturile, acestea se pot rula fără a implica Hadoop tastând următoarea comandă în consola Linux:

```
[root@sandbox ~]#cat cuvinte.txt | ./map.R -m | sort |
./reduce.R -r
```

Argumentul `-m` transmis script-ului *map.R* indică faptul că acesta rulează funcția `map()` în timp ce argumentul `-r` transmis script-ului *reduce.R* indică faptul că acesta rulează funcția `reduce()`. Acest mod de execuție este util în special pentru depanarea programelor. Se observă faptul că între funcția `map()` și `reduce()` a fost introdusă comanda *sort* deoarece funcția `reduce()` așteaptă ca perechile de intrare să fie sortate.

Pentru a rula acest job în mediul Hadoop se vor introduce în linia de comandă următoarele:

```
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop fs -put cuvinte.txt
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop fs -rmr output
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop jar
/usr/lib/hadoop-mapreduce/hadoop-streaming-2.2.0.2.0.6.0-76.jar \
-input /cuvinte.txt \
-output output \
-mapper "map.R -m" a \
-reducer "reduce.R -r" \
-file ./map.R -file ./reduce.R
[root@sandbox ~]${HADOOP_HOME}/bin/hadoop fs -cat output/part*
```

Prima comandă realizează copierea fișierului *cuvinte.txt* din sistemul de fișiere local în HDFS. A doua comandă șterge directorul *output* din HDFS care este directorul standard în care sunt scrise rezultatele unui job `MapReduce` pentru a ne asigura că acesta nu există deja de la un job precedent. Urmează apoi linia care lansează efectiv jobul `MapReduce`, semnficația parametrilor fiind prezentată anterior. Ultima comandă afișează rezultatul jobului. În mod implicit, rezultatul unui job `MapReduce` este stocat sub forma unor fișiere cu numele *part000, part001*, etc.

Al doilea exemplu utilizează fișierul *NYSE-2000-2001.tsv* care a fost deja descărcat și cu care s-a lucrat anterior. Pentru început se va calcula valoarea medie a fiecărei acțiuni ca medie aritmetică între valoarea minimă și cea maximă din cursul unei zile. Pentru aceasta avem nevoie doar de funcția Map care este implementată în script-ul următor salvat într-un fișier cu numele *media_zilnica.R*:

```
#! /usr/bin/Rscript
# se dezactivează mesajele de avertizare
options(warn = -1)
# se redirecționeaza orice output către dispozitivul null
sink("/dev/null")

#se deschide fișierul de intrare - consola
input <- file("stdin", "r")

#se citesc datele de intrare, linie cu linie
while (length(currentLine <- readLines(input,n = 1,warn = FALSE)) > 0)
{
# valorile de pe o linie citită sunt memorate într-un vector
fields <- unlist(strsplit(currentLine, ","))
# se face media aritmetică a valorilor de pe pozițiile
# 5 și 6 din vector, acolo unde sunt memorate
# valorile minime respectiv maxime ale unei acțiuni
lowHigh <- c(as.double(fields[5]), as.double(fields[6]))
mean <- mean(lowHigh)
# se anulează redirecționarea mesajelor către dispozitivul null
sink()
# se afișeaza simbolul bursei, al acțiunii și valoarea medie
cat(fields[1], fields[2], mean, "\n", sep = ",")
sink("/dev/null")
}

#se închide fișierul de intrare
close(input)
```

Script-ul conține comentarii sugestive. Execuția acestui script fără a face uz de Hadoop se poate realiza astfel:

```
[root@sandbox ~]#cat NYSE-2000-2001.tsv | media_zilnica.R
```

Dacă se dorește să se execute scriptul folosind mediul Hadoop atunci se procedează astfel:

```
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop fs -put
NYSE-2000-2001.tsv /
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop fs -rmr output
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop jar
/usr/lib/hadoop-mapreduce/hadoop-streaming-2.2.0.2.0.6.0-76.jar \
```

```
-D mapreduce.job.reduces=0 \
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input /NYSE-2000-2001.tsv  \
-output output \
-mapper ./media_zilnica.R \
-file ./media_zilnica.R \
hadoop fs -cat output/part*
```

Se constată că s-a folosit un argument prin care se specifică faptul că există decât partea `Map: -D mapreduce.job.reduces=0`. În continuare se dorește calcularea mediei pe fiecare tip de acțiune în parte. În acest caz funcția Map este cea implementată anterior și salvată în fișierul *media_zilnica.R*. Funcția Reduce va fi implementată cu ajutorul script-ului *media.R* prezentat în continuare. Script-ul începe prin dezactivarea afișării mesajelor de eroare apoi se redirecționează orice output către dispozitivul null. În continuare se declară o funcție `outputAVG` care primește ca argumente denumirea unei acțiuni și valorile medii zilnice, calculează media acestor valori și afișează apoi la consolă denumirea acțiunii și valoarea ei medie. Se declară apoi fișierul de intrare care va fi consola (standard input) și se citește linie cu linie output-ul produs de funcția `Map` pentru aceeași acțiune (aceeași valoare a cheii), se crează un vector denumit `means` care va conține valorile medii emise de funcția `Map`. În final se închide fișierul de intrare.

```
options(warn = -1)
sink("/dev/null")

outputAVG <- function(stock, means) {
stock_mean <- mean(means)
sink()
cat(stock, stock_mean, "\n", sep = ",")
sink("/dev/null")
}

input <- file("stdin", "r")
prevKey <- ''
means <- numeric(0)

while(length(currentLine <- readLines(input,
        n = 1,warn = FALSE)) > 0) {
fields <- unlist(strsplit(currentLine, ",")
key <- fields[2]
mean <- as.double(fields[3])
```

```
if(identical(prevKey, '') || identical(prevKey, key)) {
prevKey <- key
means <- c(means, mean)
} else {
outputAVG(prevKey, means)
prevKey <- key
means <- numeric(0)
}
}
if(!identical(prevKey, '')) {
outputAVG(prevKey, means)
}
close(input)
```

Execuția acestui job fără a apela la Hadoop se poate realiza astfel:

```
cat NYSE-2000-2001.tsv | ./media_zilnica.R | sort --key=1,2 |
./media.R
```

Comanda `cat` afișează conținutul fișierului de date la consolă de unde este citit de către script-ul *media_zilnica.R*. Output-ul este apoi sortat după coloanele 1 și 2 (comanda sort –key=1,2) întrucât funcția `Reduce` așteaptă ca perechile (cheie, valoare) să fie sortate după chei. În final datele sunt prelucrate de funcția `Reduce` implementată cu ajutorul scriptului *media.R*.

Execuția acestui job cu ajutorul Hadoop se realizează astfel:

```
[root@sandbox ~]# ${HADOOP_HOME}/bin/hadoop fs -put \
NYSE-2000-2001.tsv
[root@sandbox ~]# ${HADOOP_HOME}/bin/hadoop fs -rmr output
[root@sandbox ~]# ${HADOOP_HOME}/bin/hadoop jar
/usr/lib/hadoop-mapreduce/hadoop-streaming2.2.0.2.0.6.0-76.jar \
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input /NYSE-2000-2001.tsv -output output \
-mapper ./media_zilnica.R  -reducer ./media.R \
-file ./media_zilnica.R -file ./media.R
[root@sandbox ~]# hadoop fs -cat output/part*
```

16.4.2 RHadoop

RHadoop este un proiect de tip open source dezvoltat de Revolution Analytics[5] care furnizează o integrare pe partea client între R și Hadoop. Acesta permite rularea job-urilor de tip `MapReduce` din R și constă într-o colecție de mai multe pachete:

- `plyrmr` - furnizează funcții de procesare a datelor structurate de tip *plyr*, dispunând de operații de manipulare a seturilor mari de date memorate de Hadoop;
- `rmr` - conține o colecție de funcții care furnizează implementarea modelului `MapReduce` în R;
- `rdfs` - reprezintă o interfață între R și HDFS, furnizând operații de management al fișierelor stocate în HDFS din R;
- `rhbase` - este o interfață între R și HBase și furnizează funcții de management al bazelor de date HBase în R.

Instalarea RHadoop este foarte simplă, cu toate că RHadoop depinde de alte pachete R. Pentru a lucra cu RHadoop se va instala R și toate pachetele ce aparțin RHadoop pe fiecare `DataNode` din clusterul Hadoop:

```
> install.packages("RJSONIO")
> install.packages("itertools")
> install.packages("digest")
> install.packages("rJava")
> install.packages("Rcpp")
> install.packages("functional")
> install.packages("reshape2")
> install.packages("plyr")
> install.packages("caTools")
```

Pachetul `rmr` se va instala din arhiva ce conține codul sursă:

```
> install.packages("rmr2_3.1.1.tar.gz", repo = NULL,
type = "source")
```

Asemănător se instalează și celelalte pachete care compun RHadoop: `rdfs, plyrmr, rhbase`. Structura unui program R care utilizează RHadoop pentru rularea unui job `MapReduce` este următoarea:

[5] http://www.revolutionanalytics.com/

```
> library(rmr2)
> map <- function(k,v) { …}
> reduce <- function(k,vv) { …}
> mapreduce(
input = "data.txt",
output = "output",
textinputformat = rawtextinputformat,
map = map,
reduce = reduce
)
```

În primul rând se încarcă în memorie pachetul `rmr`, apoi urmează definirea funcției `map()` care primește ca argumente o pereche (cheie, valoare). Se definește apoi funcția `reduce()` care primește ca argumente o cheie și o listă de valori ce corespund acelei chei. În continuare este definit și lansat în execuție jobul `MapReduce` prin apelul funcției `mapreduce()`. Aceasta are mai multe argumente care definesc fișierul cu datele de intrare, fișierul unde vor fi scrise rezultatele, formatul datelor de intrare, numele funcției `map()` și respectiv al funcției `reduce()`.

Se arată modul de utilizare a RHadoop prin două exemple concrete (Prajapati, 2013). În primul exemplu se va folosi ca fișier de date fișierul *NYSE-2000-2001.tsv* utilizat și în exemplele anterioare și se dorește calcularea valorii medii a fiecărei acțiuni pe toată perioada de tranzacționare (medie artimetică între valoarea maximă și cea minimă din cursul unei zile). În continuare este redat script-ul R (salvat cu numele media-rhadoop.R) care realizează acest lucru.

```
#!/usr/bin/Rscript
library(rmr2)
rmr.options(backend = "local")

map <- function(k,v) {
keyval(k, v)
}

reduce <- function(k,vv) {
keyval(k, mean(as.numeric(unlist(vv))))
}

csv.reader =
  function(con, nrecs){
    lines = readLines(con, 1)
```

```
    if(length(lines) == 0)
      NULL
    else {
      delim = strsplit(lines, split = ",")
      keyval(
        sapply(delim, function(x) x[2]),
        list(
          sapply(delim, function(x) x[5]),
          sapply(delim, function(x) x[6])))
    }
}

mapreduce(
input = "/NYSE-2000-2001.tsv",
output = "output",
input.format = make.input.format(mode = "text", format =
    csv.reader),
output.format = "text",
map = map,
reduce = reduce
)
```

Script-ul începe prin încărcarea pachetului `rmr2` apoi urmează instrucțiunea `rmr.options(backend="local")` care indică faptul că fișierul de date se află pe sistemul de fișiere local și nu trebuie implicat sistemul Hadoop în accesarea lui. Acest mod de execuție este util pentru a testa și depana script-ul. În continuare este declarată funcția `map()` care nu face decât să emită perechi de tipul (cheie, valoare). Funcția `reduce()` primește ca argumente o cheie și o listă de valori corespunzătoare cheii, calculând media acestor valori. Pentru citirea datelor din fișierul de intrare declarăm funcția `csv.reader()` care citește linie cu linie fișierul de intrare, fiecare linie este apoi separată în cuvinte (prin apelul funcției `strsplit()`) si se formează perechi (cheie, valoare) unde *cheia* reprezintă denumirea unei acțiuni (se găsește pe poziția a doua a fiecărei linii a fișierului de intrare) iar *valoare* este o listă cu valorile de pe pozițiile 5 și 6 din linia de intrare (aici se găsesc valorile minime și maxime ale fiecărei acțiuni într-o zi bursieră). În finalul script-ului se apelează funcția `mapreduce()` care primește ca argumente denumirea fișierului cu datele de intrare, denumirea fișierului unde vor fi scrise rezultatele (output), funcția care citește datele de intrare (argumentul input.format), formatul datelor de ieșire (text), denumirile funcțiilor `map()` și `reduce()`.

După ce se șterge linia `rmr.options(backend="local")`, execuția acestui job cu ajutorul Hadoop se realizează din consola mașinii virtuale astfel:

```
[root@sandbox ~]#export HADOOP_CMD=/usr/bin/hadoop
[root@sandbox ~]#export HADOOP_STREAMING=/usr/lib/hadoop
-mapreduce/hadoop-streaming-2.2.0.2.0.6.0-76.jar
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop fs -put
NYSE-2000-2001.tsv
[root@sandbox ~]#${HADOOP_HOME}/bin/hadoop fs -rmr output
[root@sandbox ~]#./media-rhadoop.R
[root@sandbox ~]#hadoop fs -cat output/part*
```

Al doilea exemplu de utilizare a RHadoop se referă la estimarea unui model de regresie liniară folosind metoda celor mai mici pătrate. Există și alte modalități de a estima modele de regresie liniare cu R si Hadoop în afară de cel prezentat aici, totul depinde de problema care trebuie rezolvată și imaginația analistului în a transpune problema în termeni `MapReduce`. Un model de regresie liniar are forma următoare:

$$y_i = \beta_1 \times x_{i1} + \cdots + \beta_p \times x_{ip} + \epsilon_i = \boldsymbol{x}_i^T \times \boldsymbol{\beta} + \epsilon_i, \quad i = 1, \ldots, n \quad (16.2)$$

unde y_i este variabila dependentă iar vectorul x_i (de dimensiune p) reprezintă regresorii luați în calcul (variabilele explicative, independente), i ia valori de la 1 la n.

Cele n ecuații pot fi puse sub o formă matricială:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} x_{1,1} & \cdots & x_{1,p} \\ x_{2,1} & \cdots & x_{2,p} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,p} \end{pmatrix}, \beta = \begin{pmatrix} \beta_1^T \\ \beta_2^T \\ \vdots \\ \beta_p^T \end{pmatrix}, \epsilon = \begin{pmatrix} \epsilon_1^T \\ \epsilon_2^T \\ \vdots \\ \epsilon_n^T \end{pmatrix} \quad (16.3)$$

sau:

$$\boldsymbol{y} = X \times \beta + \epsilon \quad (16.4)$$

Vectorul β este vectorul parametrilor care trebuie estimați. Metoda celor mai mici pătrate care va fi folosită minimizează suma pătratelor reziduurilor. Formula de calcul pentru estimarea vectorului β este (Gujarati, 1995) :

$$\hat{\beta} = (X^T X)^{-1} X^T y \quad (16.5)$$

Se calculează un produs de matrici, X^TX, apoi rezultatul se inversează, $(X^TX)^{-1}$. Se calculează apoi produsul matrice-vector X^Ty și se înmulteste cu rezultatul intermediar $(X^TX)^{-1}$. Aceste calcule sunt echivalente cu rezolvarea unui sistem liniar:

$$X^TX\beta = X^Ty \tag{16.6}$$

unde X^TX este matricea sistemului liniar, X^Ty este termenul liber iar β este necunoscuta. Rezolvarea acestui sistem este echivalentă cu următoarea operație matricială:

$$\beta = (X^TX)^{-1}X^Ty \tag{16.7}$$

În R există o funcție predefinită pentru astfel de probleme (rezolvarea sistemelor liniare) : `solve(a, b, ...)`. Această funcție are doi parametri: matricea sistemului și termenul liber. În cazul de față această funcție va fi apelată astfel:

$$solve(X^TX, X^Ty) \tag{16.8}$$

Tot ce trebuie facut este calculul transpusei matricei X și apoi înmulțirea cu X, apoi cu y. Să spunem că rezolvăm o problemă în care avem 20000 de observații ($n = 20000$) și 15 variabile independente (x_i). În această situație matricea X va avea dimensiunile $(20000, 15)$ iar vectorul y va fi de dimensiune 20000. Se va pleca de la ipoteza că o matrice $A(20000, 15)$ nu poate fi stocată în memoria unui singur calculator iar calculul transpusei și înmulțirea de asemenea nu poate fi efectuată de un singur calculator. În schimb, apelul:

$$solve(X^TX, X^Ty) \tag{16.9}$$

poate fi executat foarte ușor de un singur calculator. De ce? Dacă încercăm să calculăm dimensinunea matricei X^TX constatăm că avem de efectuat un produs între două matrici de următoarele dimensiuni : $(15, 20000) \times (20000, 15) = (15, 15)$. Adică rezultatul este o matrice de dimensiune $(15, 15)$ și este perfect fezabil ca acest rezultat să fie stocat și prelucrat pe un singur calculator. De asemenea, dimensiunea produsului X^Ty este $(15, 20000) \times (20000, 1) = (15, 1)$. Deci și acesta poate fi stocat si prelucrat usor pe un singur calculator.

Obiectivul este utilizarea Hadoop pentru memorarea datelor inițiale și pentru efectuarea celor două produse. Apelul final $solve(X^TX, X^Ty)$ va fi efectuat în mod clasic, pe un singur calculator. Se va defini matricea

X cu valori aleatoare (urmând repartiția normală). Numărul de elemente ale matricei este $20000 \times 15 = 300000$:

```
> X <- matrix(rnorm(300000), ncol = 15)
```

Se va adăuga apoi o coloană nouă la matricea X care va conține valorile $1, 2, 3 \ldots 20000$ (numărul de rânduri ale matricei X). Vom vedea în continuare de ce este nevoie de această coloană nou introdusă.

```
> X1 <- cbind(1:nrow(X), X)
```

Funcția `cbind()` efectuează o concatenare pe coloane a celor două argumente. Rezultatul va fi o matrice de dimensiune `(20000, 16)` în care prima coloană conține valorile `1,2,3,...,nrow(X)`. Rezultatul va fi scris în sistemul de fișiere distribuit HDFS:

```
> X1 <- to.dfs(X1)
```

Se definește vectorul y (ale cărui elemente sunt tot numere aleatoare distribuite normal):

```
> y <- as.matrix(rnorm(20000))
```

Până acum au fost definite datele de intrare. Într-o aplicație reală acestea vor proveni din fișiere deja stocate în HDFS. Primul job map-reduce va fi definit cu scopul de a calcula produsul $X^T X$. Se va scrie mai întâi funcția `map()`:

```
> mapper = function (., Xr) {
    Xr = Xr[,-1]
    #print(dim(Xr))
    keyval(1, list(t(Xr) %*% Xr))
}
```

Aici `Xr[,-1]` înseamnă toate rândurile și coloanele matricei Xr mai puțin coloana 1. După cum a fost arătat anterior, coloana 1 conține valorile `1,2,3,...,nrow(X)`. Funcția `map()` va primi la intrare blocuri de date formate din rânduri întregi ale matricei X. S-a notat un astfel de bloc cu Xr. Dacă în timpul execuției se dorește vizualizarea dimensiunilor matricei Xr se decomentează linia `#print(dim(Xr))`. Pe calculatorul pe care s-a testat scriptul, `mapper()` este apelat de 3 ori cu

primele 7810 rânduri ale matricei X, apoi cu următoarele 7811 rânduri iar în final cu ultimele 4379 rânduri. În acest caz nu este de interes cheia pe care o primește mapper-ul. Operatorul %*% realizează înmulțirea a două matrici. Această funcție va realiza produse de submatrici de dimensiuni mai mici decât dimensiunea matricei problemei inițiale și va trimite rezultatele către funcția `reduce()` care le va însuma. Funcția `t(X)` calculează transpusa matricei X. Dacă se analizează algoritmul clasic de înmulțire a două matrici pentru calculul X^TX se va constata că se pot forma matrici de dimensiuni mai mici (se reține doar m rânduri din matricea inițială, cu $m < n$, n fiind numărul total de rânduri) care se înmulțesc și apoi se adună rezultatele parțiale. De aceea funcția `reducer()` va avea sarcina să adune rezultatele parțiale emise de `mapper()`:

```
> reducer = function(., Y) {
    keyval(1, list(Reduce('+', Y))
}
```

Pentru detalii privind funcția `Reduce()` se utilizează `help(Reduce)` într-o consolă R. Această funcție este similară funcției similare din limbajul de programare funcțională LISP. Pentru calculul produsului X^TX se va utiliza:

```
> XtX <- values(
        from.dfs(
            mapreduce(
                input = X1,
                map = mapper,
                reduce = reducer,
                combine = T
            )
        )
)[[1]]
```

Funcția `mapreduce()` va scrie rezultatul sub forma unor perechi (cheie, valoare) în sistemul de fișiere HDFS de unde este accesat cu ajutorul funcției `from.dfs(...)`. Funcția `values()` extrage doar valorile din perechile (cheie, valoare). Asemănător se va calcula X^Ty:

```
> mapper2 = function (., Xr) {
    yr = y[Xr[,1],]
    Xr = Xr[,-1]
```

```
    keyval(1, list(t(Xr) %*% yr))
}
```

`yr = y[Xr[,1],]` reține din vectorul y doar elementele corespunzătoare liniilor matricei Xr. Hadoop va apela mapper-ul cu blocuri de rânduri ale matricei originale `X1` și este nevoie să fie selectate aceleași rânduri și din y. Pentru acest lucru s-a introdus coloana nouă în `X1` (a se vedea instrucțiunea `X1 <- cbind(1:nrow(X), X)`).

```
Xty <- values(
    from.dfs(
        mapreduce(
            input = X1,
            map = mapper2,
            reduce = reducer,
            combine = T
        )
    )
)[[1]]
```

A fost folosit parametrul `combine = T` pentru a combina toate perechile (cheie, valoare) întrucât mapper-ul emite o singură cheie (1). În final se apelează:

```
solve(XtX, Xty)
```

Toate aceste instrucțiuni vor fi salvate într-un script denumit *linear-regression-rhadoop.R* care este listat mai jos:

```
#!/usr/bin/Rscript

library(rmr2)
X <- matrix(rnorm(300000), ncol = 15)
X1 <- cbind(1:nrow(X), X)
X1 <- to.dfs(X1)
y <- as.matrix(rnorm(20000))

mapper = function (., Xr) {
    Xr <- Xr[,-1]
    #print(dim(Xr))
    keyval(1, list(t(Xr) %*% Xr))
}
```

```
reducer = function(., A) {
    keyval(1, list(Reduce('+', A)))
}

mapper2 = function (., Xr) {
    yr <- y[Xr[,1],]
    Xr <- Xr[,-1]
    keyval(1, list(t(Xr) %*% yr))
}

XtX <- values(
    from.dfs(
        mapreduce(
            input = X1,
            map = mapper,
            reduce = reducer,
            combine = T
        )
    )
)[[1]]

Xty <- values(
    from.dfs(
      mapreduce(
        input = X1,
        map = mapper2,
        reduce = reducer,
        combine = T
      )
    )
)[[1]]

beta <- solve(XtX, Xty)
beta
```

Execuția acestui script în mediul RHadoop se realizează astfel:

```
[root@sandbox ~]# export HADOOP_CMD=/usr/bin/hadoop
[root@sandbox ~]# export HADOOP_STREAMING=/usr/lib/hadoop-
mapreduce/hadoop-streaming-2.2.0.2.0.6.0-76.jar
[root@sandbox ~]# ${HADOOP_HOME}/bin/hadoop fs -rmr output
[root@sandbox ~]# ./linear-regression-rhadoop.R
```

16.4.3 Rhipe

Denumirea Rhipe provine de la "R and Hadoop Integrated Programming Environment" și este un priect open source care furnizează o integrare strânsă între R și Hadoop. Analiza și prelucarea datelor se face direct în R, Rhipe furnizând utilizatorilor R aceleași facilități ale Hadoop care se găsesc și în limbajul Java. Pachetul poate fi descărcat de la adresa `http://www.datadr.org`. Instalarea Rhipe este un proces un pic mai dificil: pe fiecare `DataNode` trebuie instalat R, Protocol Buffers (o bibliotecă de funcții destinate serializării obiectelor) și Rhipe. Aceasta presupune ca R să fie compilat ca o bibliotecă partajată pe fiecare nod, Google Protocol Buffers să fie de asemenea compilat și instalat pe fiecare nod și în final să fie instalat RHipe.

Rhipe este un pachet R care permite rularea job-urilor `MapReduce` din R. Utilizatorul R va scrie funcțiile `map()` și `reduce()` ca funcții obișnuite R și Rhipe va avea apoi grijă de rularea acestora în mediul Hadoop. Intrările funcțiilor `map()` și `reduce()` sunt transferate folosind o schemă de codificare implementată de Protocol Buffers către o bibliotecă C (Rhipe). Structura generală a unui script R care utilizează Rhipe este prezentată în continuare:

```
1 library(Rhipe)
2 rhinit(TRUE, TRUE);
3 map <- expression ( {lapply (map.values, function(mapper)…)})
4 reduce <- expression(
5     pre = {…},
6     reduce = {…},
7     post = {…},
8 )
9 x <- rhmr(
10    map = map, reduce = reduce,
11          ifolder = inputPath,
12          ofolder = outputPath,
13          inout = c('text', 'text'),
14          jobname = 'my job name'))
15 rhex(x)
```

Script-ul începe cu încărcarea în memorie a pachetului Rhipe (linia 1) și inițializarea sa (linia 2). Linia 3 definește expresia *map* care va fi executată de task-ul *Map*. Liniile 4-8 conțin definiția expresiei *reduce* care constă în 3 apeluri. Blocul definit prin `pre=...` (linia 5) este apelat pentru fiecare cheie unică emisă de funcția `map()` înainte ca aceste valori

să fie trimise către blocul `reduce`. Blocul `reduce` (linia 6) este apelat cu un vector de valori drept argument iar în final blocul `post=..` (linia 7) este apelat pentru a emite perechile (cheie, valoare) de ieșire. Pe linia 9 este apelată funcția `rhmr()` care construiește practic job-ul `MapReduce` iar apelul `rhex(x)` de pe linia 15 lansează job-ul `MapReduce` în mediul Hadoop. Rhipe furnizează de asemenea funcții de comunicare cu Hadoop în timpul procesului `MapReduce` precum `rhcollect()` care permite scrierea datelor sau `rhstatus()` care returnează starea unui job.

Index

Bibliografie

Brumfield, G. (2011). High-energy physics: Down the petabyte highway. *Nature 469 pp. 282-283*.

D. Adler, O. Nenadic, W. Z. C. G. (2007). The *ff* package: Handling large data sets in R with memory mapped pages of binary flat files. *use!R 2007 Conference, Iowa State University*. `http://wsopuppenkiste.wiso.uni-goettingen.de/ff/ff_1.0/inst/doc/ff.pdf`.

Dean, J. şi S. Ghemawat (2004). Mapreduce: Simplified data processing on large clusters. În *OSDI'04, 6th Symposium on Operating Systems Design and Implementation*, pp. 137–150. USENIX, in cooperation with ACM SIGOPS.

Durbin, J. şi G. Watson (1950). Testing for serial correlation in least squares regression, i. *Biometrika 37*(3-4), 409–428.

Durbin, J. şi G. Watson (1951). Testing for serial correlation in least squares regression, ii. *Biometrika 38*(1-2), 159–179.

Economist, T. (2010, June). Data, data everywhere. Technical report. Available as `http://www.economist.com/node/15557443#sthash.jgntF194.dpbs`.

Frish, R. (1933). Editor's note. *Econometrica 1*(1), 2.

Gujarati, D. N. (1995). *Basic Econometrics* (Third ed.). McGraw Hill.

Holmes, A. (2012). *Hadoop in practice*. Manning Publications, New Jersey.

Jarque, C. M. şi A. Bera (1980a). Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics Letters 6*(3), 255–259.

Jarque, C. M. şi A. Bera (1980b). Efficient tests for normality, homoscedasticity and serial independence of regression residuals: Monte carlo evidence. *Economics Letters 7*(4), 313–318.

Jula, Dorin; Jula, N.-M. *Econometrie*. Mustang, București.

Kane, M. J.; J. Emerson şi S. Weston (2013). Scalable strategies for computing with massive data. *Journal of Statistical Software 55*(14), 1–19. `http://www.jstatsoft.org/v55/i14/`.

Laney, D. (2001, February). 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group. Available as `http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf`.

Mark, B. şi D. Laney (2012, June). The importance of big data: A definition. Technical report, Gartner. Available as `https://www.gartner.com/doc/2057415/importance-big-data-definition`.

Mayer-Schönberger, V. şi K. Cukier (2013). *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt.

Prajapati, V. (2013). *Big Data Analytics with R and Hadoop*. Packt Publishing.

Tay, L. (2013). Inside eBay's 90 pb data warehouse. Available as `http://www.itnews.com.au/News/342615,inside-ebay8217s-90pb-data-warehouse.aspx`.

Vagata, P. şi K. Wilfong (2014). Scaling the facebook data warehouse to 300 PB. Available as `https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb`.

Webster, P. (2012). Climate change simulation: Nasa's weather supercomputer. *CSC World*.

White, T. (2012). *Hadoop: The Definitive Guide, 3rd Edition*. O'Reilly Media.