R lengoaia: Lehenengo Urratsak

Usue Mori, Borja Calvo

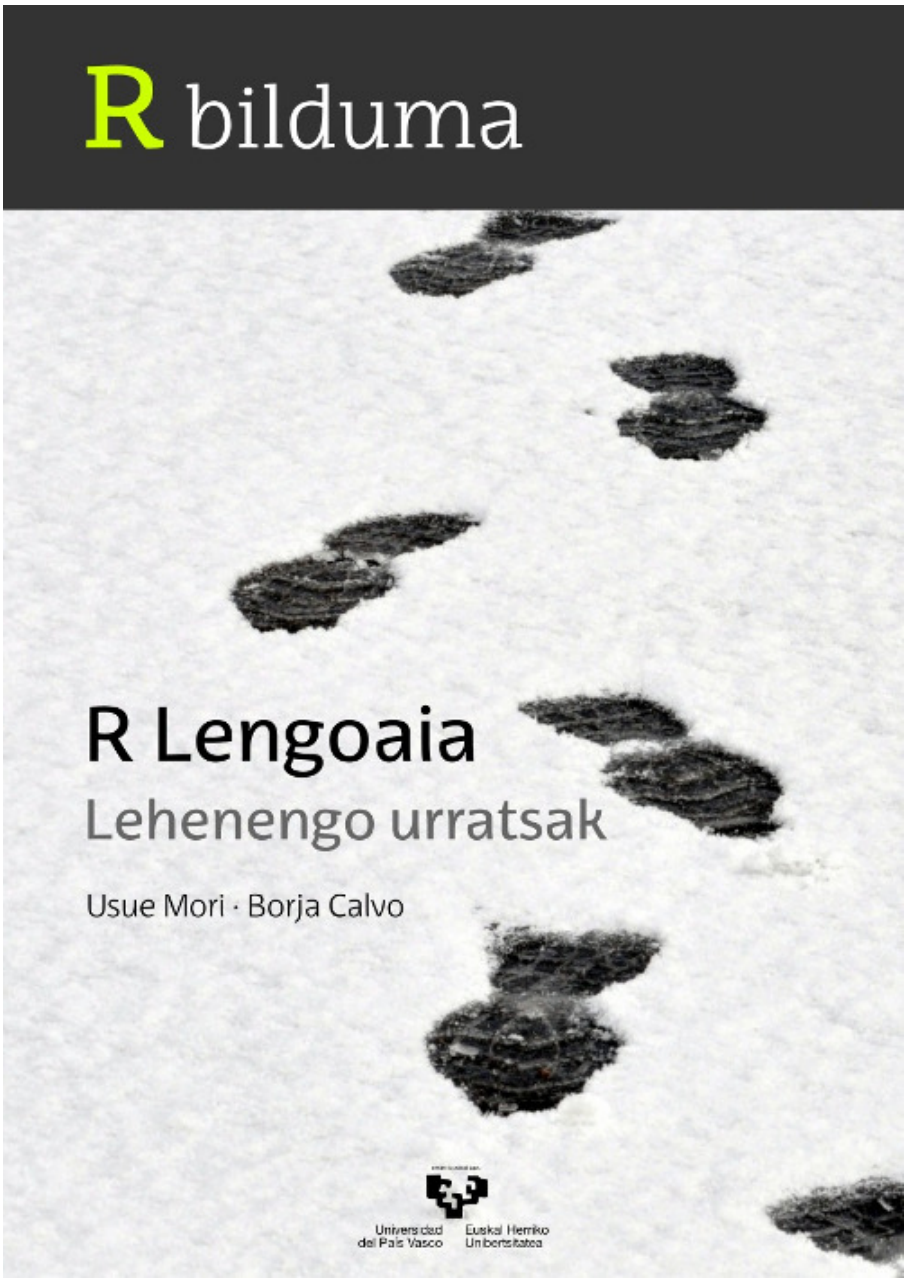
2018-03-12

Contents

Hitzaurrea

Liburuxka honetan R lenoaian lehenengo pausuak hartzeko behar duzun informazioa topatuko duzu. Agertzen diren kontzpetuak oso oinarrizkoak dira. Izan ere, liburuxka R lengaiari buruzko bilduma baten lehenengo alea da; hemen aipatzen diren gaietan sakontzeko beste hainbat liburuxka jasoko ditugu bilduma honetan.

R kodea · Datuak · Liburuaren PDFa


R bilduma
R Lengoaia
Lehenengo urratsak
Usue Mori · Borja Calvo
Universidad del País Vasco
Euskal Herriko Unibertsitatea

1 irudia.

1

Instalazioa eta lehen pausuak

R programazio lengoaia libre bat da, gehienbat estatistika eta datu analisira bideratuta dagoena. R programazio lengoaia interpretatua da, hau da, terminal batean sartzen diren aginduak interpretatu eta segidan exekutatu egiten dira, ondoren emaitzak berriro ere terminalean erakutsiz.

Hori dela eta, Rrekin diharduteko behar den gauza bakarra komando interpretea da, hots, R terminala. Edonola ere, gero ikusiko dugun moduan, lana errazteko, garapenerako ingurune integratuak (IDE-ak) ere erabil daitezke.

1.1 Rren instalazioa

Rren instalazioa gure sistema eragilearen araberakoa da; informazio guztia web ofizialean aurki daiteke.[1] Windows eta MacOS sistemen kasuan, instalazio fitxategiak eta jarraibideak ondorengo helbide hauetan aurki daitezke:

- **Windows**: http://cran.r-project.org/bin/windows/base/.
- **MacOS**: http://cran.r-project.org/bin/macosx/.

Linux sistematan, berriz, R instalatzeko biderik errazena distribuzioko pakete kudeatzailea erabiltzea da. Ubunturen kasuan, instalatu behar diren paketeak `r-base` eta `r-base-dev` dira. Edonola ere, Ubunturen repositorioetan dagoen bertsioa zaharkitua egon daiteke –Ubuntu bera azken bertsiokoa ez bada, batik bat–. Rren azken bertsioa instalatzeko, pakete kudeatzaileari Rko repositorioa gehitzea komeni da. Hau egiteko, ondoko komando hauek exekutatu behar dira[2]:

```
sudo add-apt-repository 'http://cran.r-project.org/bin/linux/ubuntu artful/'
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9
sudo apt-get update
sudo apt-get install r-base
sudo apt-get install r-base-dev
```

Jarraibide hauek Rren oinarrizko instalazioa burutzen dute eta beraz lengoaia hau erabiltzen hastea ahalbidetzen digute.

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

[2] Kontutan izan Rko repositorio hau Ubuntu 17.10 sistemarentzat dela baliagarria; Ubunturen beste bertsio bat izanez gero `artful`, sistemari dagokion izenarekin aldatu beharko duzu.

1.2 R konsola

Lehen esan bezala, R lengoaia interpretatua da, eta ondorioz, erabiltzen hasteko terminal bat besterik ez dugu behar. Noski, oinarrizko instalazioa egiten dugunean Rko terminala erabilgarri jartzen zaigu.

Rren terminalak itxura ezberdina hartzen du sistema eragile ezberdinetan. Mac eta Linux sistemetan terminala oinarrizkoa da, eta erabili ahal izateko ohiko terminalean `R` komandoa exekutatuko dugu.

Bertan, Rko edozein komando idatz dezakegu, emaitzak berriro ere terminalean lortuz. Ataza konplexuagoak burutzeko, ohikoena, *script*ak erabiltzea da. Rko *Script* bat, agindu zerrenda luze bat gordeko dituen fitxategi bat izango da (.R amaierakoa), bata bestearen atzetik exekutatuko direnak. *Script*ekin jarduteko testu lauak editatzeko programa bat eta Rren terminala besterik ez dira behar eta programatzaile askok bi elementu hauekin soilik egiten dute lan.

Windows sistemetan R terminalak itxura apur bat berezia dauka eta gainera, zenbait funtzionalitate gehigarri dauzka beste bi sistema eragileekin alderatuz. Adibidez, script editore bat eskeintzen du.

1.3 Garapenerako ingurune integratuak: RStudio

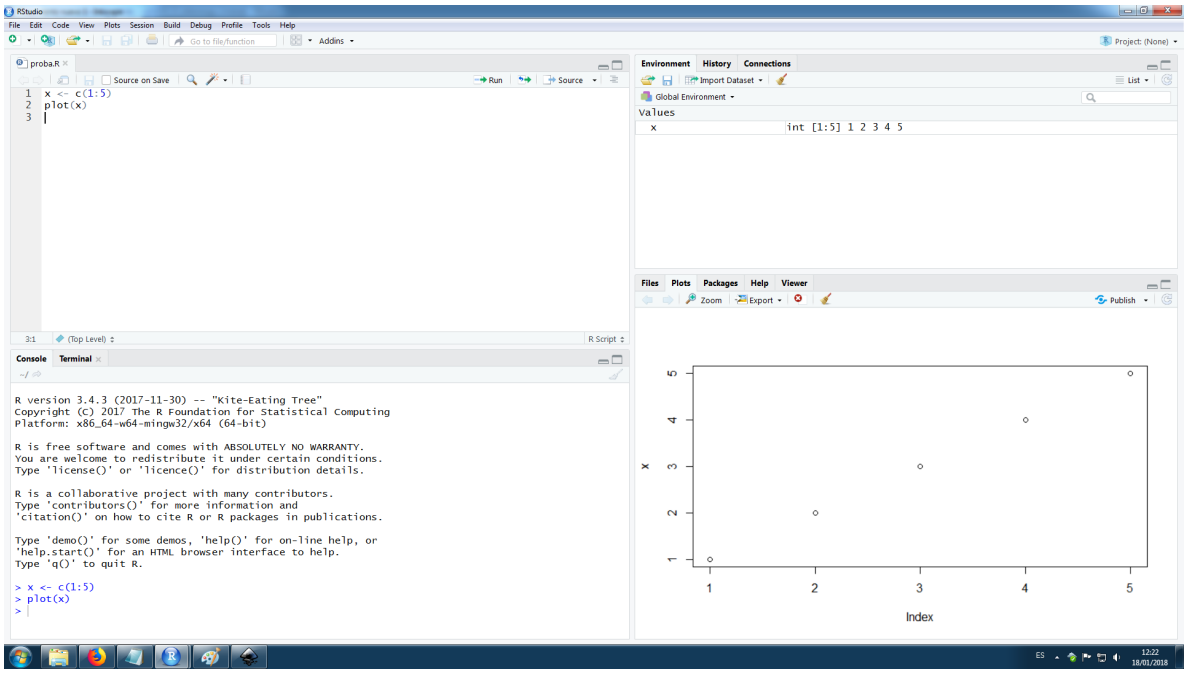
Nahiz eta terminala eta testu fitxategiak Rn lan egiteko nahikoak izan, badaude R lengoaian programatzeko garapen ingurune integratuak (IDE-ak). Garapen ingurune integratu bat software programa bat da programatzaileei haien software-a garatzerako garaian hainbat erraztasun/erosotasun eskaintzen dizkionak, hala nola, kode editore bat, debugger bat, kodearen auto-konpletatzen sistema bat, etab.

Gaur egun Rren IDE hedatuena RStudio da.[3]

RStudio-k bi IDE modalitate ditu: *desktop* eta *server*; lehena da, kasu gehienetan, instalatu beharko duguna. Honez gain, modalitate bakoitzerako, bi bertsio daude, bata dohakoa eta bestea profesionala. Guk dohakoa erabiliko dugu. Programa instalatzeko, beraz, RStudioren webunean *Download RStudio* botoian sakatu, *desktop* mota aukeratu eta ondoren *Open Source Edition* aukera. Bertan programaren hainbat bertsio izango ditugu eta gure sistemari dagokiona aukeratu beharko dugu.

RStudio zabaltzen dugunean hiru zatitan banaturiko lehio bat agertuko zaigu. Ezkerreko aldean R terminala izango dugu. Eskuinean, goiko partean, kargatutako R objektuak eta komandoen historiala izango ditugu. Beheko partean, berriz, aukera ezberdinak eskeintzen dituzten hainbat fitxa daude: fitxategiak, grafikoak, paketeak, laguntza eta bistaratzailea. Fitxategi bat zabaltzen dugunean ezkerreko zatia (hasieran terminala soilik zena) bitan banatuko da, script fitxategia –goian– eta terminala –behean– erakusteko.

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



1.1 irudia. RStudioren itxura

Aspektu honetan sakontzea ez denez kapitulu honen helburua, interesatuta egon ezkero, jarraian zerrendatzen diren webguneetan baliabide gehiago aurki ditzakezu:

- http://www.rstudio.com/products/rstudio/features/
- http://rmarkdown.rstudio.com/
- http://shiny.rstudio.com/

1.4 Lehenengo urratsak

Aldagaiek Rko objektuak gordeko dituzte. Beraz, aldagai bati balio zehatz bat esleitzeko `<-` sinboloa erabiliko dugu, edo baliokideki `=` sinboloa. Bata zein bestearen erabileraren inguruko irizpideak, estilo gidan aurki daitezke (A eranskinean). Adibidez, `x` aldagaiari, 2.45 balioa esleitzeko:

```
x <- 2.45
```

Hala, behar ezberdinak betetzeko, aldagai mota ezberdinak definitu daitezke Rn:

- **Numeric**: zenbaki errealak gordetzen dituzten aldagai motak dira.

```
x <- 3.2
```

- **Integer**: zenbaki osoak gordetzen dituzten aldagai motak dira.

```
x <- 1
```

- **Logic**: aldagai bitarrak gordetzen dituzten aldagai motak dira, `TRUE` edo `FALSE` balioak har ditzakete.

```
x <- TRUE
```

- **Character**: karaktereak eta karaktere kateak gordetzen dituzten aldagai motak dira.

```
x <- "kaixo"
```

- **Factor**: aldagai kualitatiboen balioak gordetzen dituzten aldagaiak dira, hau da, multzo finitu bateko balioak hartzen dituzten aldagaiek dira, adibidez 1, 2 edo 3. Ohartu, horrelako aldagai bat definitzea, apur bat konplexuagoa dela, izan ere, balioa esleitzeaz gain, aldagaiak har ditzakeen balio posibleen multzoa ere zehaztu behar ditugu era honetan:

```
x <- factor(1, levels=c(1,2,3))
```

Kasu honetan aldagaiak 1 balioa hartzen du eta balio posibleen multzoa {1,2,3} da. gainera, ikus daitekeen bezala, faktore motako objektu batek zein balio har ditzakeen adierazteko `levels` parametroa erabiltzen da.

Aldagai baten balioa ikusteko, bere izena idatzi eta enter sakatu behar dugu, besterik gabe. Gainera, ohartu, uneoro oso argi izan behar dugu zein motakoa den gure aldagaia, izan ere, honen arabera eragiketa batzuk ala besteak egin ahalko ditugu harekin.

Azkenik, ohartu R lengoaian, `NA` *balio galdu* a adierazteko erabiltzen dela; ez da nahastu behar `NaN` objektuarekin *ez-zenbaki* a (*not a number*) adierazten duena.

1.4.1 Eragiketa aritmetikoak

Aldagai numeriko edo osoen artean eragiketa aritmetikoak egin ditzakegu:

1.1 taula. : Eragiketa aritmetiko ohikoak

Sinboloa	Eragiketa
+	Batuketa
-	Kenketa
	Biderkaketa
/	Zatiketa
^	Berreketa
%%	Moduluak (x mod y)
%/%	Zatiketaren zati osoa

Adibidez,

```
x <- 2
y <- 3
x*y
```

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

1.4.2 Eragiketa logikoak

Eragiketa mota hauek beti TRUE (egia) ala FALSE (gezurra) itzuliko digute, eta horregatik deitzen dira eragiketa logikoak. Aldagaien gaineko baldintzak konprobatzeko balio dute. Hasteko, bi elementuren arteko konparaketak egin ditzakegu:

1.2 taula. : Eragiketa logiko ohikoak

Sinboloa	Eragiketa
< eta >	Handiago edo txikiago moduko konparaketak (soilik aldagai oso eta errealak)
<= eta >=	"txikiago edo berdin" eta "handiago edo berdin" motako konparaketak (soilik aldagai oso eta errealak)
==	Bi elementu berdinak diren konprobatzeko (edozein aldagai mota)
!=	Bi elementu ezberdinak diren konprobatzeko (edozein aldagai mota)

Adibidez,

```
x <- 5
y <- 7
x<y
```

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

Gainera, baldintza konplexuagoak eraikitzen lagunduko diguten beste operadore logiko batzuk ere badauzkagu:

1.3 taula. : Eragiketa logiko ohikoak

Sinboloa	Eragiketa
! *baldintza*	Baldintza baten kontrakoa konprobatzeko balio du.
baldintza1 | *baldintza2*	Baldintza bat edo bestea betetzen diren konprobatzen du.

Sinboloa	Eragiketa
baldintza1 & *baldintza2*	Bi baldintzak aldi berean betetzen direla konprobatzen du.
isTRUE(*baldintza*)	baldintza egia bada, TRUE itzuliko du eta bestela FALSE.

Adibidez,

```
x <- 2
y <- 6
```

```
!(x%%2==0) # x ez da bikoitia?
```

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

```
(x>3) | (y<7) # x 3 baino handiagoa EDO y 7 baino txikiagoa da?
```

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

```
isTRUE(x==2) # Egia da, x 2-ren berdina dela?
```

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

Eragiketa konplexuagoak egiteko, Rn funtzioak erabiltzen dira, baina hori aurrerago ikusiko dugu

1.5 Ariketak

Erantzun honako galderei, kasu bakoitzean soilik Rko kode lerro bakar bat erabiliz:

1. Ariketa Sortu aldagai berri bat, `x` izenekoa, 6 balioa hartzen duena eta beste bat, `y` izenekoa, 10 balioa hartzen duena.

2. Ariketa `x`aldagaiaren erroa 2.5 baino txikiagoa da?

3. Ariketa `x` eta `y` aldagaien arteko biderkadura bakoitia al da?

4. Ariketa Egia al da `x` aldagaia 3 baino txikiagoa edo berdina eta `y`aldagaia 5 baino handiagoa direla?

5. Ariketa Egia al da `y` aldagaia bikoitia edo 4ren multiploa dela?

6. Ariketa Kalkulatu `x` eta `y` aldagaien batuketa, 3rekin zatitzean lortzen dugun zati osoa.

2

Oinarrizko datu-egiturak

R lengoaian oinarrizkoak diren zenbait datu-egitura mota daude. Liburu honen helburua sarrera labur bat eskaintzea denez, bakarrik lau datu-egitura mota ohikoenak aztertuko ditugu, bektoreak, matrizeak, zerrendak eta `data-frame`ak.

2.1 Bektoreak

R lengoaiaren oinarrizko datu egitura bektorea da. Adibidez elementu bakarreko aldagai bat 1 tamainako bektore bezala ulertzen du Rk:

```
x <- 2.45
x
```

```
## [1] 2.45
```

Orainarte ulertzen ez genuen [1] sinboloak, horixe adierazten du hain zuzen, x bektorearen 1. elementua erakusten ari zatzaigula.

Elementu bat baino gehiagoko bektore bat sortzeko `c` funtzioa erabiliko dugu era honetan:

```
x <- c(2, 6, 1, 5)
x
```

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

```
y <- c(1, 4)
y
```

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

Bestalde, `c` funtzioa erabil dezakegu ere bi bektore bata bestearen atzean lotzeko:

```
z <- c(x, y)
z
```

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

Gainera, hurrenez hurreneko zenbakiez osatutako bektore bat sortzeko `:` eragilea erabili dezakegu, bai goruntz eta bai beherunzko bektoreak lortuz:

```
x <- c(2:7)
x
```

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

```
y <- c(7:2)
y
```

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

Ohartu, bektore bateko elementu guztiak mota berdinekoak izan behar dutela. Mota ezberdineko elementuak konbinatuz gero, Rk denak mota batekoak bihurtuko ditu, normalean guk nahi ez ditugun aldaketak eginez.

```
x <- c("kaixo", 3, 4, 5)
x
```

```
## [1] "kaixo" "3"     "4"     "5"
```

Aurreko adibidean, adibidez, zenbakiak ere karaktere bilakatu ditu.

Bestalde, bektore baten elementuak atzitzeko `[]` sintaxia erabiltzen da:

```
x <- c(1, 4, 7, 2, 5)
x[3]
```

```
## [1] 7
```

Adibidean ikus daitekeen bezala, edozein bektoreko lehenengo balioaren indizea 1 da. Gainera, Rk 0 luzerako bektoreak definitzeko aukera ere eskeintzen du:

```
bektore.hutsa <- vector()
bektore.hutsa
```

```
## logical(0)
```

Azkenik, bektoreen gaineko zenbait oinarrizko eragiketa egin ditzakegu Rn. Hasteko bi bektoreen arteko batuketa, kenketa eta eskalar batekin biderkaketa egin ditzakegu, hauek ohiko moduan egiten direlarik:

```
x <- c(1, 1, 1, 1, 1)
y <- c(1, 2, 3, 4, 5)
x+y
```

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

```
x*7
```

```
## [1] 7 7 7 7 7
```

Bektoreen arteko batuketa edo kenketa egitean, zer gertatzen da bektoreak tamaina ezberdinekoak direnean?. Pentsa genezake aginduak errore bat jaurtiko lukeela, baina hori ez da gertatzen dena, jarraian dagoen adibidean ikus daitekeen bezala.

```
x <- rep(5, 10)
y <- 1:5
x - y
```

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

```
z <- 1:3
x * z
```

```
## Warning in x * z: longer object length is not a multiple of shorter object
## length
```

```
##  [1]  5 10 15  5 10 15  5 10 15  5
```

Lehenengo adibidean `x` bektoreak 10 elementu ditu[1], baina `y` bektoreak bakarrik 5. Kenketa egiten dugunean lortzen dugun bektoreak 10 elementu ditu, `x` bektorearen lehenengo 5 elementuei `y` kenduta eta, jarraian, `x` bektorearen azken 5 elementuei `y` berriz ere kenduta. Emaitza hori R lengoaian bektoreak *birziklatu* egiten direlako lortzen da. Hau da, 10 tamainako bektore bat behar bada eta daukagun bektorearen tamaina soilik 5 bada, 10 tamainako bektore bat sortzen da 5 tamainakoa errepikatuz. Beste era batera esanda, lehenengo adibidean `x` bektoreari `1:5` kendu beharrean `c(1:5, 1:5)` kendu diogu.

Bigarren adibidean gauza bera gertatzen da, baina abisu bat jasotzen dugu, `z`rekin `x`ren tamainako bektorerik ezin delako sortu–izan ere, 10 ez da 3ren multiploa–. Birziklapena zenbait egoeratan oso praktikoa egingo zaigu –adibidez, `x` bektorearen elementu guztiei 1 gehitzeko `x + 1` besterik ez dugu egin behar–. Programazio estilo honek ordea, programatzailearen aldetik lengoaiaren ezagutza handia eskatzen du.

Bestalde, matematika arloan ondo definituta ez dauden beste zenbait eragiketa ere egin ditzakegu bektoreekin Rn. Adibidez, bi bektore biderkatu edo zatitu ditzakegu. Eragiketa hauek elementuz elementu aplikatuko dira:

```
x*y
```

```
##  [1]  5 10 15 20 25  5 10 15 20 25
```

```
x/y
```

```
##  [1] 5.000000 2.500000 1.666667 1.250000 1.000000 5.000000 2.500000
##  [8] 1.666667 1.250000 1.000000
```

Gainera, aurreko atalean ikusi ditugun eragiketa logikoak bektoreekin ere erabil ditzakegu, hauek bektoreko elementu bakoitzari banan banan aplikatzen zaizkiolarik:

```
x <- c(1, 3, 4, 5, 6, 7)
x > 5
```

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

Azkenik, Rko bektoreak algebrako multzoak bezala ulertzen baditugu, multzo teoriako operazioak ere burutu ditzakegu hauen gainean. Adibidez:

- **Bildura**: Bi bektoreetan dauden elementu guztiak hartuko ditu, errepikapenik gabe.
- **Ebakidura**: Bi bektoreetan aldi berean dauden elementuak hartuko ditu.
- **Barne:** Balio bat bektore baten barruan dagoen konprobatuko du, TRUE ala FALSE itzuliz.

Eragiketa hauen adibide batzuk honakoak dira:

[1] Ohartu, `rep` funtzioak bektore bat sortzen duela, elementu bat, 5 kasu honetan, errepikatuz hainbat aldiz, 10 kasu honetan.

```
x <- c(1, 2, 3, 6, 7, 8)
y <- c(1, 2, 5, 9, 10)
```

```
union(x, y) #Bildura
```

```
## [1]  1  2  3  6  7  8  5  9 10
```

```
intersect(x, y) #Ebakidura
```

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

```
6 %in% x #Barne
```

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

2.2 Matrizeak eta *array*ak

Matrizeak 2 dimentsiotako egiturak dira eta `matrix` funtzioaren bitartez sortu ditzakegu:

```
m <- matrix(1:12, nrow=3, byrow=FALSE)
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Ikusten dugunez, lehenengo parametro gisa, matrizean sartu nahi ditugun balioak jarriko ditugu bektore batean. Ondoren, errenkada kopurua zehaztuko dugu `nrow` parametroan (edo zutabe kopurua `ncol` parametroan). Azken parametroak matrizea zein ordenetan beteko den adierazten du, zutabeka (`byrow=FALSE`) edo errenkadaka (`byrow=TRUE`).

Gainera, errenkadei eta zutabeei izenak eman diezazkiekegu, `colnames` eta `rownames` funtzioak erabiliz:

```
colnames(m) <- c("W", "X", "Y", "Z")
rownames(m) <- c("A", "B", "C")
m
```

```
##   W X Y  Z
## A 1 4 7 10
## B 2 5 8 11
## C 3 6 9 12
```

Matrizeak 2 dimentsiotako egiturak dira, Rk ordea, hainbat dimentsiotako egiturak sortzeko aukera ematen digu. Horrelako objektuak sortzeko `array` funtzioa erabil daiteke, `dim` parametroan egituraren dimentsioak zehaztuz:

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

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

Ikusten dugunez, kasu honetan, `array`-ak 3 dimentsio dauzka. 1. dimentsioaren tamaina 1 da (matrizeen errenkada kopurua), bigarrenarena (matrizeen zutabeen kopurua) 2, eta hirugarrenarena 3 (matrize kopurua).

Matrizeen zein *array* -en elementuak atzitzeko kortxete sinpleak erabiltzen dira, dimentsio guztien indizeak –edo izenak– erabiliz. Hona hemen adibide batzuk:

```
m[1, 1]
```

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

```
m["A", "Y"]
```

```
## [1] 7
```

```
a[1, 1, 2]
```

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

Indizeak zenbaki bakar bat ez ezik, bektoreak ere izan daitezke:

```
m[c(1, 3), 2:3]
```

```
##   X Y
## A 4 7
## C 6 9
```

Gainera, elementuak atzitzeko beste bi sintaxi berezi ere existitzen dira. Indize jakin bat zehaztu ordez, dimentsio bat hutsik utziz gero, dimentsio horren elementu guztiak atzituko dira. Bestalde, aukeratutako indizeen aurretik `-` zeinua jarriz gero, indize horiei dagozkien elementuak ezik, beste guztiak atzituko dira:

```
m[,-1]
```

```
##   X Y  Z
```

```
## A 4 7 10
## B 5 8 11
## C 6 9 12
```

```
m[-(1:2) , ]
```

```
##  W  X  Y  Z
##  3  6  9 12
```

Gainera, Rn hainbat eragiketa matrizial ere egin ditzakegu:

- **A + B**: Matrizeen batuketa.
- **A - B**: Matrizeen kenketa.
- **A * B**: Elementuz elementu egindako matrize biderkaketa.
- **A / B**: Elementuz elementu egindako matrize zatiketa.
- **A %*% B**: Matrize biderkaketa algebraikoa.

Ohartu, lehen lau eragiketa hauetan matrizeen dimentsioak berdinak izatea komeni dela, nahi dugun emaitza lortu nahi badugu. Bestalde, azken eragiketan, matrizeen biderkaketa ohikoan, matrizeen dimentsioak (nxm) eta (mxp) modukoak izan behar dute, hau da, lehenengoaren zutabe kopurua, bigarrenaren errenkada kopuruaren berdina izan behar du.

Azkenik, bektoreei bezala, matrizeei ere operadore aritmetiko eta logikoak aplika dakizkiegu, operadoreak elementuz elementu aplikatzen direlarik:

```
m>4
```

```
##       W     X    Y    Z
## A FALSE FALSE TRUE TRUE
## B FALSE  TRUE TRUE TRUE
## C FALSE  TRUE TRUE TRUE
```

Ohartu, operadore konplexuak edo haien konbinazioak ere aplika ditzakegula:

```
((m>4) & (m%%2==0))
```

```
##       W     X     Y     Z
## A FALSE FALSE FALSE  TRUE
## B FALSE FALSE  TRUE FALSE
## C FALSE  TRUE FALSE  TRUE
```

Adibidez, kasu honetan, `m`-ko elementuak 4 baino handiagoak eta bikoitiak diren ala ez aztertzen dugu.

2.3 Zerrendak

Bektoreetan elementu guztiak mota berdinekoak izan behar dira. Alabaina, kasu askotan bektore heterogeneoak ere beharko ditugu; hauek `list` edo zerrenda motako objektuen bidez adierazten dira. Adibide gisa, zerrenda batean izena, e-posta eta telefono zenbakia gorde dezakegu:

```
contact <- list("Izena", "izena@server.com", 555652332)
contact
```

```
## [[1]]
## [1] "Izena"
##
## [[2]]
## [1] "izena@server.com"
##
## [[3]]
## [1] 555652332
```

Zerrenda bateko elementuak edozein objektu mota izan daitezke, baita bektoreak edo matrizeak ere:

```
zerrenda <- list(x, y, m)
zerrenda[[1]]
```

```
## [1] 1 2 3 6 7 8
```

```
zerrenda[[3]]
```

```
##   W X Y  Z
## A 1 4 7 10
## B 2 5 8 11
## C 3 6 9 12
```

Adibidean ikusten den moduan, zerrenda bateko elementuak atzitzeko kortxete bikoitza –`[[]]`– erabiltzen da[2].

Ondoren, zerrenda bateko objektu zehatz bat atzitu ondoren, bere gaineko atzipenak ere burutu ditzakegu. Adibidez honako aginduek, `zerrenda`ko 3. elementuko (1,2) posizioan dagoen balioa itzuliko digu, hau da, `m` matrizearen (1,2) posizioko balioa:

```
zerrenda[[3]][1,2]
```

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

2.4 data-frameak

Bektoreen gisa, matrizeak datu egitura oso erabilgarriak dira, baina eragozpen bat dute: bertan gordetako elementu guztiak mota berekoak izan behar dute. Hau da, matrize batean ezin dugu zutabe numeriko bat eta, aldi berean, *character* motako zutabe bat izan. Baina ohiko datu baseak mota ezberdinetako aldagaiak (zutabeak) izaten dituzte orohar.

[2]Kortxete bakarra erabiltzen badugu, hasiera batean, emaitza berdina dela pentsa genezake, baina ezberdintasu handi bat dago: zerrendan dagoen elementu soila lortu beharrean 1 tamainako zerrenda bat lortuko dugu, atzitu nahi dugun elementua gordeko duena.

Hau hala, R lengoaian, mota ezberdinetako zutabeak dituzten taulak *data.frame* objektu motaren bidez adierazi daitezke. Izatez, *data.frame*ak posizio bakoitzean tamaina berdineko bektore bat gordetzen duten zerrendak dira eta zerrenda bezala ere tratatu daitezke, baina haien erabilgarritasuna dela eta, izaera propioa eman zaie. Objektu mota hauek eraikitzeko *data.frame* funtzioa erabiltzen da:

```
df <- data.frame("String" =c("A", "B", "C"),
                 "Number" =10:12,
                 "Logical"=c(TRUE, TRUE, FALSE))
df
```

```
##   String Number Logical
## 1      A     10    TRUE
## 2      B     11    TRUE
## 3      C     12   FALSE
```

`data.frame` baten zutabeen edo aldagaien izenak ezagutzeko `names` atributua erabil dezakegu:

```
names(df)
```

```
## [1] "String"  "Number"  "Logical"
```

Gainera, *data frame* baten lehen lerroak edo azken lerroak ikusteko, `head` eta `tail` funtzioak erabil ditzakegu, ikusi nahi ditugun lerro kopurua zehaztuz:

```
head(df, n=2)
```

```
##   String Number Logical
## 1      A     10    TRUE
## 2      B     11    TRUE
```

```
tail(df, n=3)
```

```
##   String Number Logical
## 1      A     10    TRUE
## 2      B     11    TRUE
## 3      C     12   FALSE
```

Gainera, *data.frame*-etan atzipenak, matrizeetan edo zerrendetan bezala egin daitezke:

```
df[1, 2] # 1. errenkadako 2. elementua atzitzen du.
```

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

```
df[1, ]  # 1.errenkada atzitzen du
```

```
##   String Number Logical
## 1      A     10    TRUE
```

```
df[, 2] # 2. zutabea atzitzen du
```

```
## [1] 10 11 12
```

```
df[[2]] # 2. zutabea atzitzen du
```

```
## [1] 10 11 12
```

Baina, gainera, zutabeak izena badute, $ eragilea ere erabil dezakegu aldagai oso bat atzitzeko:

```
df$Logical
```

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

Era berean, aldagai berriak ere sor ditzakegu, $ eragilea eta eragiketa bektorialak erabiliz:

```
df$Number2 <- df$Number*2
df$Number3 <- df$Number > 11
df
```

```
##   String Number Logical Number2 Number3
## 1      A     10    TRUE      20   FALSE
## 2      B     11    TRUE      22   FALSE
## 3      C     12   FALSE      24    TRUE
```

Azkenik, ohartu, atzipen konplexuagoak egiteko, operadore logikoak ere erabil ditzakegula. Adibidez, `df` datu basetik soilik `Logical` aldagaia `TRUE` duten eta `Number` aldagaia gutxienez 11 duten errenkadak aukeratu nahi baditugu:

```
df[df$Logical==TRUE & df$Number>=11, ]
```

```
##   String Number Logical Number2 Number3
## 2      B     11    TRUE      22   FALSE
```

2.5 Ariketak

1. Ariketa Sortu honako bektoreak Rn:

- $x = (1, 3, 5, 7)$
- $y = (2, 4, 6, 8)$
- $z = (7, 5, 3, 1)$
- $t = (8, 6, 4, 2)$

2. Ariketa Egin x eta z bektoreen arteko kenketa (orden honetan). Bektore berriaren zein posizio dira -3 baino txikiagoak? -5 balioa lortutako kenduren artean agertzen al da?

3. Ariketa Sortu matrize bat, m izenekoa, x, y, z eta t bektoreak zutabetzat dituenak. Horretarako, berrerabili iada definituta dituzun aldagaiak. Ondoren, ordezkatu matrize honen 3. zutabea bere 2. eta 3. zutabeen baturagatik.

4. Ariketa Sortu zerrenda bat x, y, m eta zure izena gordetzen dituenak (orden horretan). Ondoren, atzitu zerrenda honetatik x bektorearen 2. elementua eta m matrizearen (2,3) posizioan dagoen elementua.

5. Ariketa `iris` datu basea Rn berez dagoen *data frame* bat da. Berau erabiliz egin honako ariketak:

- Zeintzuk dira datu base honen aldagaiak?
- Bistaratu datu basearen lehen 5 lerroak.
- Aukeratu, `Species` aldagaian `setosa` balioa duten indibiduoak.
- Aukeratu `Sepal.Length` 4.5 baino handiagoa eta `Petal.Width` 0.3 baino txikiagoa edo berdina duten indibiduoak.
- `Petal.Width` aldagaia cm-tan dago neurtuta. Sortu aldagai berri bat, `Petal.Width2` izenekoa aldagai bera m-tan ematen duena eta gehitu aldagai berri hau `iris` datu baseari.

3

Funtzioak

Rko objektu garrantzitsuenetako batzuk funtzioak dira. Funtzioak datuen gaineko operazio konplexuagoak egiteko erabiliko ditugu. Behin baino gehiagotan erabili beharko ditugun agindu zerrendak enkapsulatzeko balio dute eta behin definituta eta izen bat esleituta, nahi adina aldiz erabili ditzakegu.

3.1 Funtzioen erabilera

Hasteko, garrantzitsua da jakitea Rn iada funtzio asko inplementaturik daudela. Beraz, askotan, behar dugun funtzioa bilatu eta erabiltzea izango da gure lan bakarra.

Rko funtzio bat erabiltzeko, funtzioaren izena erabiliko dugu, eta parentesi artean, funtzio horri dagozkion parametro edo argumentuen balioak definitu beharko ditugu. Ohartu argumentu hauek Rko edozein motako objektuak izan daitezkeela. Adibidez, `abs` funtzioak, bektore numeriko edo matrize bat hartzen du argumentu bezala eta bertako elementuen balio absolutuak itzultzen ditu:

```
x <- c(-5:12)
x
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12
```

```
abs(x)
```

```
##  [1]  5  4  3  2  1  0  1  2  3  4  5  6  7  8  9 10 11 12
```

Funtzio batzuk, aurrekoak kasu, argumentu bakarra jasotzen dute eta aldiz, beste batzuetan argumentu bat baina gehiago definitu beharko ditugu. Adibidez, `seq` funtzioak sekuentzia erregularrak sortzeko balio du eta hiru argumentu hartzen ditu: sekuentziaren hasierako balio, sekuentziaren bukaerako balioa eta zenbakien arteko tartea:

```
seq(from=1, to=10, by=2)
```

```
## [1] 1 3 5 7 9
```

Erabili nahi ditugun funtzioak parametro asko dituztenean `do.call` funtzioa oso lagungarria izan daiteke. Parametroak zerrenda batean definituz gero, ondoren, `do.call` funtzioa honela erabil dezakegu era honetan:

```
args        <- list()
args$from      <- 1
args$to   <- 10
args$by <- 2
do.call(what=seq, args=args)
```

```
## [1] 1 3 5 7 9
```

Exekuzioa berdina da zuzenean edo `do.call` funtzioa erabiliz, baina hainbat kasutan azken hurbilketarekin kodearen antolaketa txukunagoa izango da. Gainera, parametroen zerrenda hau beste funtzio batekin ere berrerabili dezakegu.

Azkenik, ohartu, ez dela beharrezkoa argumentuen izena funtzioaren deian beti idaztea:

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

Argumentuen izenak idatzi nahi ez baditugu, kontuz ibili beharko gara argumentuen ordena zehazterakoan; erabiltzen ari garen funtzioak, bere definizioan, finkatuta daukan argumentuen ordenera egokitu beharko gara. Izan ere, ordena aldatu ezkero, ez dugu nahi dugun emaitza lortuko:

```
seq(2, 10, 1)
```

```
## [1]  2  3  4  5  6  7  8  9 10
```

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

Funtzio baten argumentuak zeintzuk diren eta zein ordenetan definituta dauden nola ikusi, hurrengo bi ataletan ikasiko dugu.

3.2 Paketeak

Rren ezaugarririk interesgarriena hedagarritasuna da; edonork gara ditzake funtzionalitate berriak dakartzaten paketeak. Beraz, Rko funtzio ezberdinak pakete desberdinetan daude eskuragai.

Rn paketeak instalatzeko bide ofiziala CRAN biltegiaren bitartez da –*Comprehensive R Archive Network*–. Bertan milaka pakete daude gordeta.[1]

Biltegi horretan dauden paketeak `install.packages` aginduarekin instala daitezke. Esate baterako, estatistiko batzuk kalkulatzeko beharko dugun **fBasics** paketea instalatzeko, ondokoa idatzi behar da R terminalean:

[1] Zerrenda osoa http://cran.r-project.org/web/packages/index.html helbidean dago.

```
install.packages("fBasics")
```

CRAN biltegiaz gain, badago oso hedatua dagoen beste bat: Bioconductor –http://bioconductor.org–. Biltegi horretan, bioinformatika arloan garatutako paketeak aurki daitezke gehien bat. Hala ere, pakete orokorrak ere aurki ditzakegu. Bioconductor biltegiaren kasuan paketeen instalazioa pixka bat ezberdina da. Adibidez **reshape2** paketea instalatzeko[2] adibidez, ondoko kodea exekutatu behar da R terminalean.

```
source("http://bioconductor.org/biocLite.R")
biocLite("reshape2")
```

Paketeen instalazioa behin bakarrik egin behar dugu ordenagailu bakoitzean. Hala ere, Rko saio berri bat hasten dugun bakoitzean behar ditugun paketeak kargatu beharko ditugu, hauek erabili ahal izateko. Horretarako `library` funtzioa erabiliko dugu:

```
library(fBasics)
```

3.3 R ren laguntza

Rren funtzio guztiak ezagutu eta haiek erabiltzen jakitea ezinezkoa da, izan ere milaka funtzio daude iada eskuragai eta paketeak denborarekin aldatzen dira, funtzio berriak sortuz, zaharrak ezabatuz, etab.

Beraz, Rko funtzio eta paketeei buruzko informazioa lortzeko Rren laguntza erabili beharko dugu. Horretarako, hainbat bide daude. Erabili nahi dugun funtzioaren izena zehazki ezagutzen badugu, honakoa idatziko dugu Rren terminalean:

```
help("funtzioaren izena")
```

Honek zuzenean aukeratutako funtzioaren laguntza orria zabalduko digu, RStudion bagaude eskuineko beheko leihoan eta bestela internet nabigatzailean. Honakoak ere efektu berdina izango du:

```
?funtzioaren izena
```

Defektuz, `help` funtzioak kargatuta dauden paketeetako funtzioen artean egingo du bilaketa. Kargatuta ez daukagun paketeren bateko funtzio baten laguntza behar badugu `try.all.packages = TRUE` argumentua gehitu beharko dugu `help` funtzioaren deian:

```
help("funtzioaren izena", try.all.packages = TRUE)
```

Funtzioaren izena ez badugu zehazki ezagutzen, `help.search()` edo `??` erabili ditzakegu, argumentu bezala, bilatu nahi dugunarekin erlazioa duen espresio bat jarriz:

```
??variance
```

Hau eginda, bilaketa bat egingo da eta erlazionatuta dauden funtzio, objektu eta pakete guztien

[2] Datu egiturak maneiatzeko eta eraldatzeko erabili daiteke pakete hau.

zerrenda itzuliko digu Rk. Ondoren, guk, behar duguna aukeratu beharko dugu hau guztiaren artean eta berriro ere laguntza orri batera iritsiko gara.

Rko laguntza orri guztiek egitura berdina dute, honako atal hauetaz osatuta:

- **Description**: Hautatutako funtzioak zer egiten duen azaltzen duen lerro pare bat.
- **Usage**: Funtzioari dei nola egin diezaiokegun adierazten da, kodea erabiliz.
- **Arguments**: Funtzioari zein sarrera argumentu eman behar dizkiogun adierazten da eta haien ordena, bakoitzean zein motako argumentua den azalduz eta defektuzko baliorik duen ala ez zehaztuz. Defektuzko baliorik ez duten argumentuei nahi eta nahiez eman beharko diegu balio bat.
- **Details**: Funtzioari buruzko azalpen luzeago bat ematen da, xehetasun garrantzitsu batzuk emanez.
- **Value**: Funtzioak irteeran zein balio itzultzen dituen adierazten digu, zer informazio gordetzen duten itzulitako objektuak eta zein motakoak diren adieraziz.
- **References**: Funtzioarekin edo bere inplementazioarekin erlazionatutako erreferentzia bibliografikoak aurkezten dira.
- **See Also**: Hautatutako funtzioarekin erlazionatutako Rko beste funtzio batzuen zerrenda bat.
- **Examples**: Funtzioaren erabileraren adibide batzuk eskeitzen dira.

Guretzako atal garrantzitsuenak **Description**, **Usage**, **Arguments** eta **Examples** izango dira.

Rren laguntza oso erabilgarria eta lagungarria da, beraz hau ondo erabiltzen jakitea ezinbestekoa da. Gainera, interneten ere informazio ugari dago Rko funtzio eta pakete ezberdinen inguruan. Gogoratu, bai laguntza eta bai interneteko bilaketak egitean ingelesa erabiliz emaitza askoz gehiago eta erabilgarriagoak lortuko ditugula.

3.4 Ohiko funtzioak

Honako taula honetan Rko ohiko funtzio batzuen laburpena egingo dugu.

3.1 taula. : Ohiko funtzio batzuk

Funtzioa	Funtzionalitatea
rm	Objektuak ezabatzeko
ls	Definituta dauden objektu guztien zerrenda itzultzen du
paste	Bi bektore kateatzen ditu eta karaktere bihurtzen ditu.
log, log10, log2	Oinarri ezberdinetako logaritmoak
exp	Funtzio esponentziala
sqrt	Erro karratua
cos, sin, tan, acos, asin, atan	Funtzio trigonometriko ohikoak

Funtzioa	Funtzionalitatea
seq	Sekuentzia erregularrak sortzeko
rep	Balio bat hainbat aldiz errepikatuz, bektore bat sortzeko
table	Maiztasun taula edo kontingentzia taulak sortzeko
length	Bektore edo zerrenda baten luzera ezagutzeko
dim	Objektuen dimentsioa ezagutzeko (matrizeak, *data frame*-ak eta *array*-ak)
is, mode	Objektu baten mota jakiteko
as.numeric, as.character, as.factor	Mota batetik bestera *casting*a egiteko.
na.omit	Bektore batean agertzen diren `NA` balioak ezabatzeko
unique	Bektore bateko errepikatzen ez diren elementuak lortzeko
append	Bektore bati balioak eransteko
rev	Bektore bati buelta emateko, elementuak atzekoz aurrera ordenatuz
sort	Bektore bateko elementuak ordenatzeko
order	Bektore bateko elementuen ordena itzultzen digu, txikienaren indizea lehenengo eta handienaren indizea azkena jarriz.
sign	Bektore bateko elementuen zeinuak itzultzen ditu
floor, ceiling, round, signif, trunc	Borobiltzeko funtzioak
max	Bektore batean balio maximoa
min	Bektore batean balio minimoa
sum	Bektore bateko elementuen batura egiteko.
cut	Aldagai jarraitu bat tarteetan biltzeko.
which	Bektore edo matrize batean aukeratutako baldintzak betetzen dituzten balioen indizeak itzultzen ditu
which.max, which.min	Bektore batean balio maximoaren edo minimoaren indizea itzultzen du.
cbind	*Data frame* (edo matrize) bati zutabeak gehitzeko
rbind	*Data frame* (edo matrize) bati errenkadak gehitzeko
t	Matrize baten transposatua.
diag	Matrize baten diagonala atzitzeko edo matrize diagonal bat sortzeko
subset	Bektore, matrize edo *data frame* batean, definitutako baldintzak betetzen dituzten indibiduoak aukeratzeko

Funtzioa	Funtzionalitatea
rowMeans	Matrize edo *data frame* baten errenkaden batezbestekoak kalkulatzeko.
rowSums	Matrize edo *data frame* baten errenkaden batukariak kalkulatzeko.
colMeans	Matrize edo *data frame* baten zutabeen batezbestekoak kalkulatzeko.
colSums	Matrize edo *data frame* baten zutabeen baturak kalkulatzeko.
summary	*Data frame* baten aldagai guztien edo aldagai konkretu baten estatistiko garrantzitsuenen laburpen bat
cumsum	Bektore batean batuketa metakorrak egiten ditu.
cumprod	Bektore batean biderkaketa metakorrak egiten ditu.

Funtzio guzti hauen erabilera hobeto ezagutu nahi badugu eta zenbait adibide ikusi nahi baditugu, R ren laguntzara jo dezakegu aurreko atalean azaldu bezala.

3.4.1 Estatistikoak kalkulatzeko funtzioak

Bereziki, goian aipatutako funtzioez gain, oinarrizko estatistikoak kalkulatzeko honako funtzio hauek erabili ditzakegu:

3.2 taula. : Ohiko estatistikoak kalkulatzeko funtzioak

Funtzioa	Funtzionalitatea
mean	Batezbestekoa
var	Kuasibariantza
sd	Kuasi desbiderapen estandarra
median	Mediana
quantile	Pertzentil, kuantil eta dezilak
range	Heina
skewness	Asimetria koefizientea (timeDate paketean)
kurtosis	Kurtosia (timeDate paketean)
ineq	Gini-ren indizea (ineq paketean)
Lc	Lorenz-en kurba irudikatzeko (ineq paketean)

Beste zenbait estatistiko, adibidez moda, bariantza, desbiderapen estandarra eta alborapena, hauei aurreko gaietan ikasi ditugun eragiketak eta transformazioak aplikatuta eta funtzio berriak sortuz lor ditzakegu, hurrengo atalean azalduko dugun moduan.

Adibidez, Rn aurki dezakegun `iris` datu baseko `Sepal.Width` aldagaiaren estatistiko batzuk kalkulatu ditzakegu:

```
library(timeDate)
library(ineq)
mean(iris$Sepal.Width)
```

```
## [1] 3.057333
```

```
quantile(iris$Sepal.Width)
```

```
##   0%  25%  50%  75% 100%
##  2.0  2.8  3.0  3.3  4.4
```

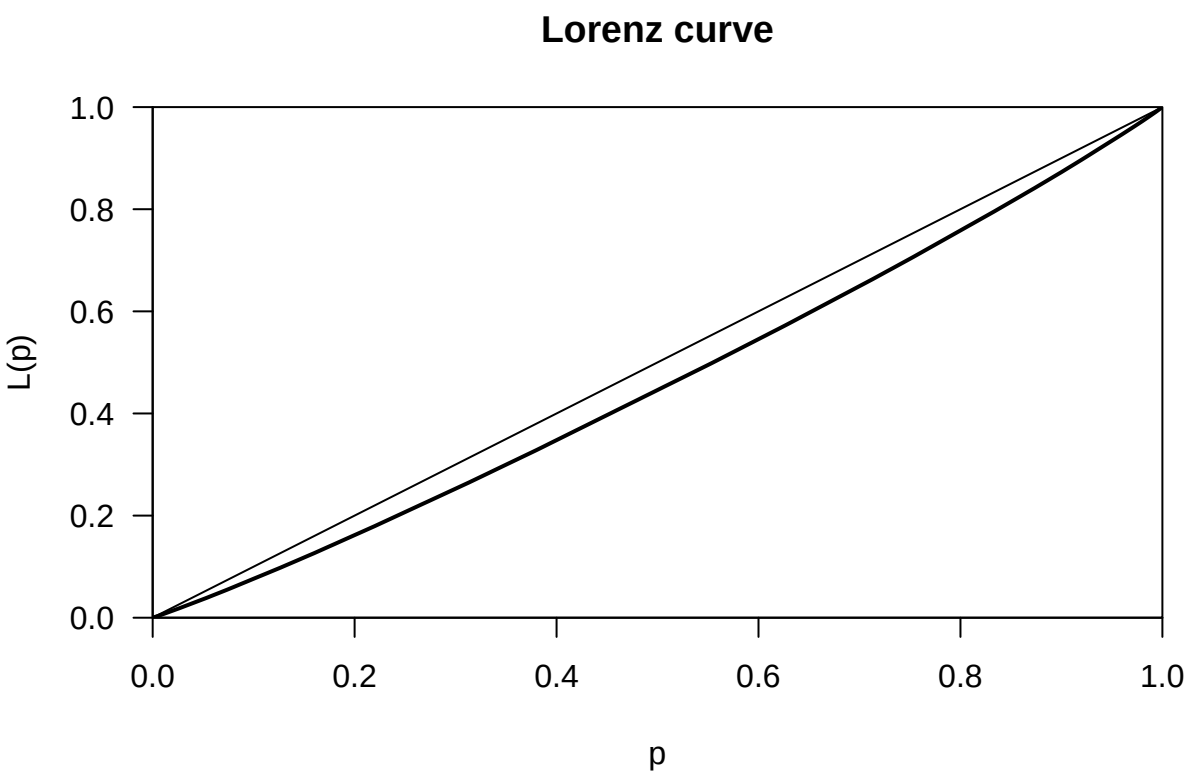
```
skewness(iris$Sepal.Width)
```

```
## [1] 0.3126147
## attr(,"method")
## [1] "moment"
```

```
ineq(iris$Sepal.Width)
```

```
## [1] 0.07915104
```

```
Lorenz <- Lc(iris$Sepal.Width, plot=TRUE)
```



Ohartu, aurreko kodea exekutatu ahal izateko `timeDate` eta `ineq` paketeak instalatuta izan behar direla.

Azkenik, aldagai jarraitu baten estatistikoak beste aldagai baten arabera kalkulatzeko `aggregate` funtzioa erabil dezakegu. Adibidez, honako kodeak `Sepal.Width` aldagaiaren asimetria koefizientea kalkulatzen du, `Species` aldagaiaren arabera.

```
aggregate(iris$Sepal.Width, by=list(iris$Species), FUN=skewness)
```

```
##      Group.1           x
## 1     setosa  0.03872946
## 2 versicolor -0.34136443
## 3  virginica  0.34428489
```

3.5 Funtzio berriak sortzea

Rko funtzioak erabiltzeaz gain, behar denean, guk ere geure funtzio propioak definitu eta eraiki ditzakegu. Funtzioak sortzeko `function` funtzioa erabiltzen da era honetan:

```
funtzioaren.izena <- function(argumentuak){

  Aginduak

  return(objektua)
}
```

Adibidez, hurrengo adibidean cm-tan neurturako pertsonen altueren bektore bat metrotara pasatzen dugu:

```
conversion1 <- function(x){
  x/100
}

x <- c(165, 170, 155)
conversion1(x)
```

```
## [1] 1.65 1.70 1.55
```

Funtzioek, implementatzen duten azken aginduaren emaitza itzultzen dute. Edonola ere, nahi izanez gero, `return` agindua erabili daiteke, funtzioaren emaitza esplizituki adierazteko.

```
conversion1 <- function(x){
  return(x/100)
}
```

Ohartu, funtzioak sortzean bere argumentuak erazagutu edo definitu behar direla. Hauek izen bat izan behar dute eta balio lehenetsi (defektuzko balio) bat ere izan dezakete, nahi badugu. Adibidez, jarraian definituko dugun funtzioak cm-tan dagoen bektore bat metrotara pasako du, baina gainera, emaitzaren digitu dezimal konpurua kontrolatu ahalko dugu `digituak` izeneko argumentu baten bidez:

```
conversion2 <- function(x, digituak=1){
  m <- x/100
  round(m, digits=digituak)
}

x <- c(165, 170, 155)
conversion2(x)
```

```
## [1] 1.6 1.7 1.6
```

```
conversion2(x, digituak=2)
```

```
## [1] 1.65 1.70 1.55
```

Balio lehenetsirik ez duten parametroei, `x` kasu honetan, derrigorrez eman behar zaie balio bat funtzioa erabiltzean. Argumentu horiei balioa eman ezean, interpreteak errore bat jaurtizen du. Aldiz, balio lehenetsia daukaten parametroak ez ditugu zertan definitu, baliorik esleitu ezean Rk defektuzko balioa erabiliko baitu. Gainera, ohartu, funtzio baten barnean, beste funtzioei deitu diezaiekegula: kasu honetan `round` funtzioa erabili dugu zenbaki erreal bat borobiltzeko balio duena.

3.6 Ariketak

1. Ariketa Erabili laguntza `sample` funtzioa ezagutzeko. Zein parametro hartzen ditu sarreran? Hauetako baten batek ba al du defektuko baliorik? Zein balio itzultzen ditu? Zein motakoak dira objektu hauek? Sortu adibide bat, laguntza orrian agertzen denaren ezberdina, funtzio honen erabilera erakusteko.

2. Ariketa Jarraitu honako pausuak, bakoitzean soilik funtzio bat erabiliz:

- Definitu ezazu $x = (5, 4, 3, 2, 1)$ bektorea `seq` funtzioa erabiliz.
- Ordenatu x bektorea balio txikienetik handienera eta gorde `y` izeneko aldagaian.
- Gehitu `y` bektoreari 2 eta 4 balioak bukaeran.
- Aukeratu `y` bektorean errepikatzen ez diren elementuak eta gorde `z` bektorean.

3. Ariketa Kalkulatu honakoa $\sum_{i=10}^{100}(i^2 + 4i^5)$.

4. Ariketa Kalkulatu $x = (1, 4, 2, 4, 6, 1)$ bektorearen batezbestekoa 2 modu ezberdinetan:

- R n iada definituta dagoen funtzio bat erabiliz.
- Zuk definitutako funtzio bat erabiliz (OHARRA: Kontuan izan, bektoreak balio galduak baditu, hauek batezbestekoa kalkulatu aurretik kendu beharko ditugula.).

5. Ariketa Definitu funtzio bat, bektore baten desbiderapen estandarra (s_n) kalkulatuko duena eta erabili funtzio hau $x = (9, 8, 1, 3, 5, 1, 3, 5, 1, 4)$ bektorearen desbiderapen estandarra kalkulatzeko. (OHARRA: Kontuan izan, bektoreak balio galduak baditu, hauek batezbestekoa kalkulatu aurretik kendu beharko ditugula.)

6. Ariketa Egin aurreko ariketa berriro, horretarako Rn definitutako `sd` funtzioak itzulitako baliotik abiatuz. (OHARRA: Kontuan izan, bektoreak balio galduak baditu, hauek batezbestekoa kalkulatu aurretik kendu beharko ditugula.)

7. Ariketa Definitu datu bektore baten taula estatistikoa eraiki eta itzultzen duen funtzio bat, x_i, f_i, F_i, h_i eta H_i zutabeak dituena. Horretarako, goian azaldutako ohiko funtzio batzuetaz baliatu beharko zarete. Ondoren, erabili funtzio hori $x = (9, 8, 1, 3, 5, 1, 3, 5, 1, 4)$ laginaren taula lortzeko.

8. Ariketa Sortu bektore baten moda kalkulatuko duen funtzio bat eta, hau erabiliz, alborapena kalkulatuko duen beste funtzio bat. Kalkulatu funtzio hauek erabiliz $x = (9, 8, 1, 3, 5, 1, 3, 5, 1, 4)$ laginaren moda eta alborapena.

9. Ariketa `iris` datu basea Rn berez dagoen *data frame* bat da. Berau erabiliz egin honako ariketak:

- Aztertu zein motakoak diren aldagai guztiak `is` funtzioa erabiliz.
- Kalkulatu aldagai jarraitu guztien batezbestekoak Rko agindu bakar bat erabiliz.
- Kalkulatu *Petal.Width* izeneko aldagaiaren batezbesteko geometrikoa.
- Sortu aldagai berri bat *Petal.Width2* izenekoa, aldagai hau tarteetan banatzen duena eta kualitatibo bihurtzen duena (OHARRA: erabili `cut` funtzioa).

4

Datuak irakurri eta idatzi

Aurreko kapituluetan datu-egiturak nola sortu eta manipulatu ikusi dugu, baina askotan horrelako objektuetan jasotzen den informazioa nonbaitetik kargatu behar da. Are gehiago, kargatutako datuak landu ondoren, emaitzak gorde nahi izango dugu askotan. Kapitulu honetan datuak nola irakurtzen eta idazten diren ikusiko dugu. Gehien erabiltzen diren formatuak R lengoaiko berezko formatua eta csv-a dira.

4.1 Rko fitxategi formatua

R ingurunean sortu ditugun objektuak (bektoreak, matrizeak, etab.) zuzenean fitxategietan gorde daitezke `save` funtzioa erabiliz. Funtzioaren erabilera oso erraza da, lehenik objektu zerrenda bat pasatu behar diogu eta, gero, fitxategiaren izena.

```
matrize.bat <- matrix(1:100, ncol=10)
bektore.bat <- c("A","B","C")
save(matrize.bat, bektore.bat, file="fitxategia.RData")
```

Goiko kodea erabiliz sortutako bi objektuak, `matrize.bat` eta `bektore.bat` *fitxategia.RData* izeneko fitxategi batean gorde ditugu. Kontutan izan `file` argumentuari izen-bide osoa ez badiogu ematen, fitxategia R saioko lan-direktorioan gordeko dela.

Jakiteko zein den uneko lan-direktorioa `getwd` funtzioa erabil dezakegu. Direktorioa aldatzeko, berriz, `setwd` funtzioa erabil dezakegu edo, RStudio erabiliz gero, *Session* menuan *Set Working Directory* atalean aukera ezberdinak izango ditugu.

Goiko kodearekin sortzen dugun fitxategia formatu berezian dago, eta bakarrik Rn zabaldu daiteke. Horregatik, _*.RData_ hedapena erabiltzen da.

Gordetako informazioa R saioan berriro kargatzeko, `load` funtzioa erabil dezakegu.

```
rm("matrize.bat","bektore.bat") # Bi objektuak ezabatu
matrize.bat
```

```
## Error in eval(expr, envir, enclos): object 'matrize.bat' not found
```

```
load(file="fitxategia.RData") # Kargatu fitxategia
matrize.bat
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
```

```
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
```

Lehen esan bezala, kontuz fitxategiaren bide-izenarekin, fitxategia existitzen ez bada errore bat jasoko dugu eta.

```
load(file="asmatutako_izena.RData")
```

```
## Warning in readChar(con, 5L, useBytes = TRUE): cannot open compressed file
## 'asmatutako_izena.RData', probable reason 'No such file or directory'
```

```
## Error in readChar(con, 5L, useBytes = TRUE): cannot open the connection
```

4.2 Formatu estandarrak: CSV fitxategiak

R lengoaiaren fitxategi-formatua oso erabilgarria izan arren, datuak beste programekin trukatzeko ez du balio. Are gehiago, formatua bitarra denez, ezin dugu testu-editore arruntekin _*.RData_ formatuko fitxategietan gordetako edukia bistaratu. Hori dela eta, oso ohikoa da taulak gordetzeko formatu estandarrak erabiltzea.

Fitxategi-formatu hedatuena CSVa da (*comma separated values*, ingelesez). Fitxategi mota hauetan errenkadak taulen errenkadak dira eta zutabeak koma edo beste karaktere baten bidez banatzen dira. Hau da, honako testu hau testu-fitxategi batean (*adibidea.csv*) kopiatzen badugu …

```
1,2,3,4,5
5,4,3,2,1
1,1,1,1,1
2,2,2,2,2
```

… fitxategi horren edukia R sesioan karga dezakegu `read.csv` funtzioa erabiliz

```
taula <- read.csv(file="adibidea.csv")
taula
```

```
##   X1 X2 X3 X4 X5
## 1  5  4  3  2  1
## 2  1  1  1  1  1
## 3  2  2  2  2  2
```

Goiko adibidean, ohartu, fitxategian lau errenkada geneuzkala idatzita, baina kargatutako

taulan bakarrik hiru agertzen zaizkigu. Hori gertatzen da `read.csv` funtzioak lehenengo lerroa *goiburu*ak (zutabeen izenak, alegia) izatea espero duelako. Izan ere, `taula`objektuko zutabeen izenak (`X1`etik `X5`era) lehenengo errenkadako balioak dira, aurretik X gehituta, Rn zutabeen eta errenkedeen izenak ezin baitira zenbaki batekin hasi.

Arazoa konpontzeko biderik errazena (eta egokiena), fitxategian goiburua gehitzea da, honako hau idatziz, adibidez.

```
A,B,C,D,E
1,2,3,4,5
5,4,3,2,1
1,1,1,1,1
2,2,2,2,2
```

Beste konponbidea `read.csv` funtzioari goibururik ez dagoela esatea da, `header` argumentuari `FALSE` balioa esleituz.

```
taula <- read.csv(file="adibidea.csv", header=FALSE)
taula
```

```
##   V1 V2 V3 V4 V5
## 1  1  2  3  4  5
## 2  5  4  3  2  1
## 3  1  1  1  1  1
## 4  2  2  2  2  2
```

Ondoren, izenak jarri diezazkiokegu zutabeei honela:

```
colnames(taula) <- c("A", "B", "C", "D", "E")
taula
```

```
##   A B C D E
## 1 1 2 3 4 5
## 2 5 4 3 2 1
## 3 1 1 1 1 1
## 4 2 2 2 2 2
```

`read.csv` funtzioak beste argumentu erabilgarri gehiago ere baditu (denak ikusteko funtzioaren laguntza kontsultatu `?read.csv` exekutatuz):

- `sep` Argumentu honek zutabeak banatzeko erabili den karaketerea definitzeko balio du. Karaktere ohikoenak `,`, `;`, tartea eta tabuladorea dira. Azken hori adierazteko `\t` kodea erabil daiteke.
- `dec` Fitxategian agertzen diren balioak zenbakiak badira, Rk dezimalak `.` karakterearen bidez banatuta daudela suposatuko du. Beste karaktere bat erabili bada dezimalak adierazteko (`,`, adibidez), argumentu honen bidez definitu behar da interpretazioa ondo egiteko.
- `skip` Zenbait kasutan, fitxategiaren hasieran kargatu behar ez den informazioa egon daiteke (iruzkinak, esate baterako). Argumentu honekin hasierako zenbat lerro baztertu behar diren adieraz dezakegu.

Rko terminalean sortutako matrizeak edota `data.frame`ak CSV fitxategitan gorde daitezke, `write.csv` funtzioa erabiliz. Funtzionamendua `read.csv` funztioarenaren oso antzerakoa da,

argumentu gehienak berdinak baitira. Haintzat hartzeko gauza bakarra da `write.csv` funtzioak, defektuz, errenkaden izenak ere gordetzen dituela beti (edo, izenik ez badute, errenkadaren indizea). Informazio hori ez badugu nahi, `row.names` argumentuari `FALSE` balioa esleitu behar diogu.

```
write.csv(x=taula^2, file="adibidea_karratua.csv", row.names=FALSE)
taula.berbi <- read.csv(file="adibidea_karratua.csv")
taula.berbi
```

```
##    A  B C  D  E
## 1  1  4 9 16 25
## 2 25 16 9  4  1
## 3  1  1 1  1  1
## 4  4  4 4  4  4
```

Goiko kodean ikus daitekeen bezala, funtzioaren lehenengo atributua gorde nahi dugun taula da (gure kasuan,`taula`ren karratua).

4.3 Testu fitxategiak

Ohikoena ez izan arren, tarteka testu fitxategietatik irakurri edota testu fitxategietan idatzi beharko dugu. Testu fitxategietatik bai irakurtzeko eta bai bertan idazteko, hasteko, fitxategiarekin konexioa ezarri behar dugu, `file` funtzioa erabiliz. Funtzio horren argumentu nagusia `description` da, non fitxategiaren bide-izena idatzi beharko dugun.

```
fitxategia <- file(description="adibidea_karratua.csv")
```

Behin fitxategia irekita, edukia lerroz lerro irakur dezakegu `readLines` funtzioa erabiliz. Funtzio horrek karaktere-bektore bat itzuliko digu, non elementu bakoitza fitxategiaren lerro bat den.

```
lerroak <- readLines(fitxategia)
lerroak[2]
```

```
## [1] "1,4,9,16,25"
```

```
lerroak[3]
```

```
## [1] "25,16,9,4,1"
```

Oso garrantzitsua da fitxategia erabiltzen amaitu eta gero konexioa ixtea. Horretarako, `close` funtzioa erabil dezakegu.

```
close(fitxategia)
```

Fitxategia irakurri beharrean bertan idatzi behar badugu, prozedura antzerakoa erabiliko dugu (alegia, konexioa ezarri, idatzi eta gero konexioa itxi), baina oraingo honetan `writeLines` funtzioa erabiliko dugu nahi duguna idazteko.

```
fitxategia <- file(description="testu_fitxategia.txt")
writeLines("Nire testu fitxategia", con=fitxategia)
writeLines("----------------------", con=fitxategia)
writeLines("\tLerro bat", con=fitxategia)
writeLines("\tBeste lerro bat", con=fitxategia)
close(fitxategia)
```

Goiko adibideak *testu_fitxategia.txt* izeneko fitxategi bat sortzen du. Bertan, lau lerro idatzi ditugu, bakoitza `writeLines` funtzioari behin dei eginez. Adibidean oso ohikoa den karaktere berezi bat agertzen da, `\t`, tabuladorea sartzeko erabiltzen dena.

4.3.1 Terminalaren irteera

Kasu berezi moduan, R terminalean agertzen dena ere fitxategi batean idatzi/kopiatu daiteke `sink` funtzioa erabiliz. Fitxategi hauek normalean _*.Rout_ hedapena izaten dute. `sink` funtzioa exekutatzen dugunean, hortik aurrera terminalean agertzen diren irteera guztiak aukeratutako fitxategi horretan idatziko dira.

```
sink(file="Rko_irteera.Rout")
4+4
```

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

```
c("Testu", "bat")
```

```
## [1] "Testu" "bat"
```

```
sink()

fitxategia <- file("Rko_irteera.Rout")
readLines(con=fitxategia)
```

```
## character(0)
```

```
close(fitxategia)
```

Goiko adibidean `sink` funtzioa bi bider erabili dugu. Lehenengoan irteera(k) *Rko_irteera.Rout* fitxategian idazteko da. Bigarrena, argumenturik ez duena, fitxategiarekin konexioa itxi eta berriro irteerak terminalean idazteko da. Irteera bakarrik fitxategian agertzea nahi ez badugu (hau da, terminalean ere agertzea nahi badugu), `split` argumentuari `TRUE` balioa eman behar diogu.

```
sink(file="Rko_irteera.Rout", split=TRUE)
4+4
```

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

```
c("Testu", "bat")
```

```
## [1] "Testu" "bat"
```

```
sink()

fitxategia <- file("Rko_irteera.Rout")
readLines(con=fitxategia)
```

```
## character(0)
```

```
close(fitxategia)
```

4.4 Hedapenak

Aurreko ataletan CSV eta testuko fitxategiak nola idatzi/irakurri ikusi dugu. Atal honetan bi hedapen ikusiko ditugu. Alde batetik, fitxategi konprimatuak nola erabili aztertuko dugu eta, bestetik, sarean dauden fitxategiak nola atzitu erakutsiko dugu.

4.4.1 Fitxategi konprimatuak

Testu fitxategiak (CSVak barne) ez dira oso bide eraginkorra informazioa gordetzeko. Hori dela eta, zenbait egoeratan informazioa konprimatzeko premia izango dugu. R lengoaian informazioa fitxategi konprimatutan gordetzea oso erraza da, `gzfile` funtzioa erabiliz. Adibidez, aurreko ataletan erabili dugun matrizea fitxategi konprimatu batean gordetzeko hau egin dezakegu.

```
fitxategia <- gzfile(description="adibidea_bikoitza.csv.gz")
write.csv(x=taula*2, file=fitxategia, row.names=FALSE)
```

Irakurtzea are errazagoa da, fitxategia konprimatuta egon arren era berdinean irakurtzen baita.

```
taula.bikoitza <- read.csv(file="adibidea_bikoitza.csv.gz")
taula.bikoitza
```

```
##    A B C D  E
## 1  2 4 6 8 10
## 2 10 8 6 4  2
## 3  2 2 2 2  2
## 4  4 4 4 4  4
```

4.4.2 Sareko fitxategiak

Kapitulu honetan zehar `file` funtzioa erabili dugu diskoan dauden fitxategiekin konexioak ezartzeko. Fitxategiak zerbitzari batean egonez gero, `url` funtzioa erabil dezakegu konexioa ezartzeko. Hortik aurrera, era berdinean irakur dezakegu fitxategia.

```
f <- url(description="http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat")
edukia <- read.csv(file=f, header=FALSE, sep=" ")
head(edukia)
```

```
##   V1  V2   V3    V4 V5
## 1  1 307  930 36.58  0
## 2  2 307  940 36.73  0
## 3  3 307  950 36.93  0
## 4  4 307 1000 37.15  0
## 5  5 307 1010 37.23  0
## 6  6 307 1020 37.24  0
```

4.5 Ariketak

1. Ariketa Gorde .RData fitxategi batean Rko saioan dituzun objetu guztiak. Gogoratu `ls` funtzioak saioko objektuen zerrenda itzultzen duela.

2. Ariketa Erabili testu editore bat (edo kalkulu-orri bat) goiburua duen CSV fitxategi bat sortzeko. Bertan hiru zutabe sortu, `Izena`, `NAN` eta `Nota` izenekoak. Sartu asmatutako datuak (nota zenbaki bat izan behar da), gorde fitxategia eta kargatu R terminalean. Ondo kargatu dira datuak? Kargatutako datuetan `Nota` zutabea `numeric` da? Ez bada, nola konponduko zenuke?

3. Ariketa Aurreko ariketan sortutako datuak erabiliz, idatzi, testu fitxategi batean, "`Izena`, zure nota `Nota` da." esaldia ikasle bakoitzarentzat, errenkada bakoitzeko lerro bat idatziz.

4. ariketa Sartu Eustateko webgunean eta menuan aukeratu Gaiak. Bertan aukeratu Biztanleria->Biztanleriaren Mugimenduak->Jaiotzak eta aukeratu "Estatistika taulak". Aukeren artean topatu "100 neska-izen ugarienen zerrenda" eta datuak csv formatuan deskargatu. Kargatu datuak eta fitxategi batean idatzi, komaz banatuta, gehien errepikatzen diren 10 izenak (lerro bat urte bakoitzeko).

5. Ariketa Sartu Eustateko webgunean eta menuan aukeratu Datu bankua. Bertan aukeratu Gizartea->Hezkuntza->Eskolatze-maila eta bertan lehenengo sarrera aukeratu. Egin eremu aukeraketa bat, esportatu datuak CSV fitxategi batera eta R terminalean kargatu datuak.

5

Kontrol-egiturak eta begiztak

Beste lengoaia guztietan bezala, R n kontrol-egitura eta begizta tipikoak erabil ditzakegu. Jarraian, egitura horien sintaxia eta zenbait adibide jasotzen dira.

5.1 `if` agindua

Exekuzio baldintzatua ahalbidetzen duen egiturarik sinpleenak `if` eta `else if` aginduak dira. Funtzio hauek parametro bakarra dute –balio logiko bat itzultzen duen espresio bat, hain zuzen–. Hona hemen agindu hauen sintaxia.

```
k <- 5
if (k < 2){
  message("k balioa 2 baino txikiagoa da")
}else if (k < 3){
  message("k balioa 2 eta 3 balioen artean dago")
}else{
  message("k balioa 3 edo handiagoa da")
}
```

```
## k balioa 3 edo handiagoa da
```

5.2 `switch` agindua

`if` agindua baldintza baten arabera bi kode blokeen artean aukeratzeko erabiltzen da; bi aukera baino gehiago egonez gero, `switch` agindua erabil dezakegu. Funtzio horren lehenengo parametroa zenbaki edo `string` bat itzultzen duen espresio bat da. Ondoren, definitutako espresioak itzuli dezakeen balio posible bakoitzari dagozkion kode blokeak datoz. Hona hemen sintaxiaren adibide pare bat.

```
a <- 1
b <- 2
switch(a+b,
       "A"={
         message("Option A selected")
```

```
       },
       "B"={
         message("Option B selected")
       },
       "C"={
         message("Option C selected")
       })
```

```
## Option C selected
```

```
opt <- "B"
switch(opt,
       "A"={
         message("Option A selected")
       },
       "B"={
         message("Option B selected")
       },
       "C"={
         message("Option C selected")
       })
```

```
## Option B selected
```

Lehenengo adibidean espresioak zenbaki bat itzultzen du; kode blokeetatik zenbaki horri dagokiona exekutatzen da –emaitza i bada, i. blokea, alegia–. Bigarren adibidean, berriz, espresioaren emaitza *string* bat da eta beraz izen bereko blokea exekutatuko da. Azken aukera hau erabiltzeko, noski, kode blokeek izena izan behar dute. Berez, zilegi da izenik ez duen bloke bat uztea. Bloke honek kode lehenetsia definituko du: espresioak itzultzen duen *string*a ez badago kode blokeen izenen artean, izenik gabeko kode blokea exekutatuko da.

5.3 `for` agindua

Iterazio kopuru ezaguna dituzten begiztak definitzeko `for` agindua erabili ohi da. Honetarako, `for` funtzioaren barruan, `aldagaia in zerrenda` motako espresio bat sortu beharko dugu; iterazio bakoitzean aldagaiak zerrendako balio bat hartuko du. Hona hemen adibide sinple pare bat.

```
lst <- c("A", "B", "C", "D", "E")
for (l in lst)
  message("Uneko elementua:", l, "\n")
```

```
## Uneko elementua:A
```

```
## Uneko elementua:B
```

```
## Uneko elementua:C
## Uneko elementua:D
## Uneko elementua:E
```

```
s <- 0
for (i in 1:5) {
  s <- s + i
}
s
```

```
## [1] 15
```

5.4 while agindua

Kode bloke jakin bat baldintza bat bete arte behin eta berriro exekutatu behar denean `while` funtzioa erabiliko dugu. Funtzio horrek `logical` balio bat itzultzen duen espresio bat du parametro bakartzat. Hona hemen sintaxiaren adibide bat.

```
i <- 1
while(i < 5) {
  message("i aldagaia txikiegia da (", i, ") ...\n")
  i <- i + 1
}
```

```
## i aldagaia txikiegia da (1) ...
## i aldagaia txikiegia da (2) ...
## i aldagaia txikiegia da (3) ...
## i aldagaia txikiegia da (4) ...
```

5.5 Bestelako aginduak

Ikusi ditugun aginduez gain, badaude kodearen exekuzioa kontrolatzeko erabil daitezkeen beste hiru funtzio, `repeat`, `break` eta `next`. Informazio gehiago lortzeko, `?Control` exekuta dezakezu.

5.6 Begiztak eta paralelizazioa

R lengoaian dauden funtzio gehienak bektorialak dira eta, beraz, lengoaia eragiketa bektorialak egiteko dago optimizaturik. Hori agerian gelditzen da jarraian dagoen adibidean.

```
n <- 1000000
x <- 1:n
y <- n:1

system.time({
  x + y
})
```

```
##    user  system elapsed
##   0.005   0.004   0.009
```

```
system.time({
  for (i in 1:n){
    x[i] + y[i]
  }
})
```

```
##    user  system elapsed
##   0.973   0.000   0.972
```

Adibide honetan bi bektore sortu ditugu, 1etik `n`rako balio osoak hartzen ditu batek eta `n`tik 1erakoak besteak. Rn bi bektore hauen osagaiak elementuz elementu batzeko `+` eragilea erabil dezakegu zuzenean. Beraz, esan dezakegu, Rn batuketa eragilea bektoriala dela. Are gehiago, eragiketa hau, bektoriala izateaz gain paralelizaturik dago, hau da posizio ezberdinetako batuketak aldi berean egiten dira. Beste hainbat legoaiatan bi bektore elementuz elementu batzeko begizta bat erabili beharko genuke, bigarren kode zatian egiten den bezala. Alabaina, Rn hau ez da estrategiarik egokiena, `+` eragilearen paralelizazioa apurtzen baitugu. Hau honela, adibideko bi kodeek berdina egiten dute baina lehenengoa askoz ere azkarragoa da.

Goiko adibide sinpletik ondorio garrantzitsu bat atera behar dugu: Oro har, Rn programatzerako garaian begiztak saihesten saiatu behar gara. Are gehiago, Rn funtzio asko daude definituta eta optimizatuta eta, beraz, ezer inplementatu aurretik, garatu nahi dugun kodea aurrera eramateko behar ditugun funtzioak iada ez direla existitzen egiaztatzea da egokiena.

R ren oinarrizko funtzio gehienak bektorialki funtzionatzeko daude optimizaturik. Zenbait kasutan ordea, guk sortutako funtzioren bat bektorialki aplikatzeko beharra izango dugu; hau burutzeko `apply` motako funtzioak erabil ditzakegu. Existitzen diren guztietatik, ondorengo hiruak azalduko ditugu zehaztasun gehiagorekin: `sapply`, `lapply` eta `apply`.

5.6.1 `sapply` funtzioa

Zerrenda edo bektore baten elementu guztiei funtzio bat aplikatzeko `sapply` funtzioa erabiltzen da; Emaitza, bektore bat edo matrize bat izango da, aukeratutako funtzioak itzultzen duen emaitza motaren arabera. Adibide moduan, `character` bektore batek posizio bakoitzean daukan *string*aren karaktere kopurua kalkulatuko dugu `nchar` funtzioa era bektorialean aplikatuz:

```
chr.vector <- c("Apply", "a", "Function", "over", "a",
                "List", "or", "Vector")
sapply(X=chr.vector, FUN=nchar)
```

```
##    Apply        a Function     over        a     List       or   Vector
##        5        1        8        4        1        4        2        6
```

5.6.2 `lapply` funtzioa

`lapply` funtzioak `sapply`ek egiten duen gauza bera egiten du, baina emaitza zerrenda batean gordetzen da matrize batean beharrean. Erabilera berdina da, bakarrik emaitza aldatzen da.

```
lapply(X=chr.vector, FUN = nchar)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 8
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 1
##
## [[6]]
## [1] 4
##
## [[7]]
## [1] 2
##
## [[8]]
## [1] 6
```

5.6.3 `apply` funtzioa

Ikusi ditugun funtzioek bektoreekin eta zerrendekin dihardute, dimentsio bakarreko egiturekin, alegia. Bi dimentsioko egitura bat badugu –matrize edo `data.frame` bat–, `apply` funtzioa erabil dezakegu funtzioak bektorialki aplikatzeko. Funtzioaren erabilera antzerakoa da, baina oraingo honetan bi dimentsio ditugunez, beste parametro bat behar dugu, funtzioa ea errenkadaka, zutabeka edo elementuka aplikatu behar den adierazteko. Parametro hau `MARGIN` da, eta 1 balioa esleitzen badiogu, funtzioa errenkadaka aplikatuko da. Argumentuari 2 balioa esleitzen badiogu, berriz, funtzioa zutabeka aplikatuko da. Azkenik, funtzioa elementuz elementu aplikatu nahi badugu, argumentuari `c(1,2)` bektorea esleitu beharko diogu.

```
m <- matrix(1:200, ncol=20)
apply(m, MARGIN=1, FUN=median)
```

```
##  [1]  96  97  98  99 100 101 102 103 104 105
```

```
apply(m, MARGIN=2, FUN=median)
```

```
##  [1]   5.5  15.5  25.5  35.5  45.5  55.5  65.5  75.5  85.5  95.5 105.5
## [12] 115.5 125.5 135.5 145.5 155.5 165.5 175.5 185.5 195.5
```

Goiko adibidean dauden bi deiek matrize baten mediana konputatzen dute, lehenengo kasuan errenkadaka eta bigarren kasuan zutabeka.

5.6.4 Noiz erabili `for` eta noiz ez

Oro har, begiztek kodearen eraginkortasunean eragin handia dute, baina eragin hori iterazio kopuruaren araberakoa da; begiztak iterazio gutxi baditu, eragina txikia izango da eta, hortaz, `for` egiturak erabil daitezke[1]. Izan ere, kodearen ulergarritasuna dela eta, kasu horietan begiztak erabiltzea komenigarria da, `apply` motako funtzioak baino errazagoak baitira interpretatzeko.

Tamainaz gain, badago beste aspektu bat `apply` motako funtzioen eraginkortasuna baldintzatzen duena: sekuentzialtasuna. Kasu batzuetan, kodearen natura sekuentziala izango da, iterazio bakoitzean aurrekoan lortutako emaitza erabili behar dugulako, adibidez. Kasu horietan `apply` funtzioak lortzen duen paralelizazioa ez da erabilgarria, hortaz, ez dugu ezer irabazten. Hortaz, horrelako kasuetan `for` egiturak erabiltzea egokia da.

5.7 Ariketak

1. Ariketa Idatzi `for` begizta bat 1tik 100rako zenbaki bikoiti guztiak idazten dituena. Egin ariketa hau bera begiztarik erabili gabe.

2. Ariketa Idatzi funtzio bat, `maximo` izenekoa, begizta bat erabiliz bektore bateko balio maximoa bilatu eta itzultzen duena. Begiztarik erabili gabe, nola egingo zenuke hau?

[1] Are gehiago, kasu batzuetan begiztak eraginkorragoak izan daitezke.

3. Ariketa Definitu honako matrizea Rn:

$$A_{m,n} = \begin{pmatrix} 1 & 3 & 1 & 4 \\ 2 & 1 & 7 & 6 \\ 5 & 3 & 1 & 1 \\ 3 & 3 & 7 & 2 \end{pmatrix} \tag{5.1}$$

Sortu matrize honen zutabe guztien batezbestekoak gordetzen dituen bektore bat `for` begizta bat erabiliz. Egin ariketa berbera `apply` erabiliz. Errepikatu ariketa Rko oinarrizko funtzio bat erabiliz.

4. Ariketa Egin funtzio bat zenbaki bat lehena den konprobatzen duena: lehena bada "x zenbakia lehena da" esaldia pantallaratuko du terminalean eta bestela, "x zenbakia ez da lehena", kasu bakoitzean, `x` dagokion zenbakiarekin ordezkatuz.

5. ariketa Definitu funtzio bat, argumentu bezala zenbaki natural bat `n` jasoko duena, eta Fibonacci-ren segidako lehenengo `n` zenbakiak gordetzen dituen bektore bat itzultzen duena. Erabili `while` agindua. (OHARRA: Fibonnaciren segida ezagutzen ez baduzue, begiratu bere definizioa Wikipedian).

6. ariketa Definitu zerrenda bat, posizio bakoitzean honako bektore bat duena:

- $x = (1, 2, 3)$\$
- $y = (7, 5, 1, 4)$\$
- $z = (1, 2, 1, 2, 1)$\$
- $t = (a, b, c, d, e)$\$

Erabili `for` begizta bat zerrenda horretako elementu bakoitzaren luzera lortzeko. Egin ariketa bera, `lapply` komandoa erabiliz.

6

Grafikoak Rn

R lengoaiak datuak bistaratzeko aukera asko ematen ditu. Kapitulu honetan oinarrizko instalazioak dakartzan aukera ohikoenak aztertuko ditugu. Ondoren, sistema aurreratu batzuei sarrera labur bat emango diegu. Amaitzeko, grafikoak fitxategietan nola gorde daitezkeen azalduko dugu.

6.1 Oinarrizko grafikoak

Rko oinarrizko instalazioak hainbat funtzio ditu grafiko mota ezberdinak egiteko. Horrez gain, grafikoen itxura kontrolatzeko eta grafiko konposaketak sortzeko funtzioak ere eskuragarri ditugu pakete estrarik instalatu gabe. Atal honetan Rrekin datozen funtzio nagusienak aztertuko ditugu.

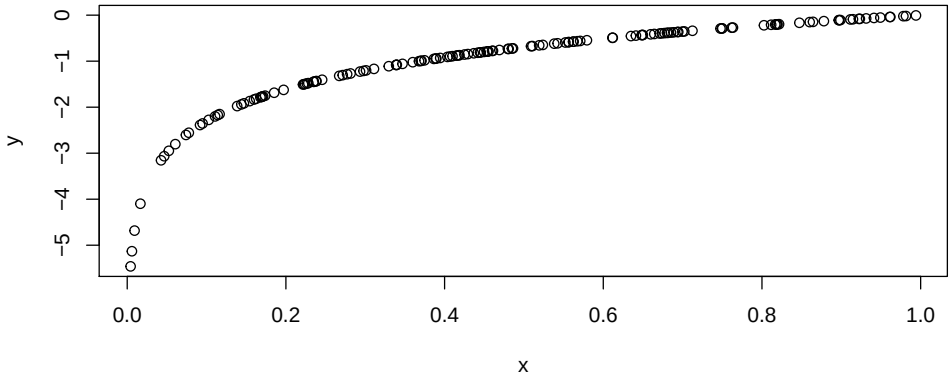
6.1.1 `plot`, `lines` eta `points` funtzioak

R lengoaian grafikoak egiteko funtziorik sinpleena `plot` funtzioa da. Funtzio hau objektu mota ezberdinentzako inplementatuta dagoenez, funtzio berdinak emaitza ezberdinak eman ditzake. Kasurik sinpleenean dispertsio-grafikoak egiteko erabil dezakegu, bi bektore, `x` eta `y`, emanik:

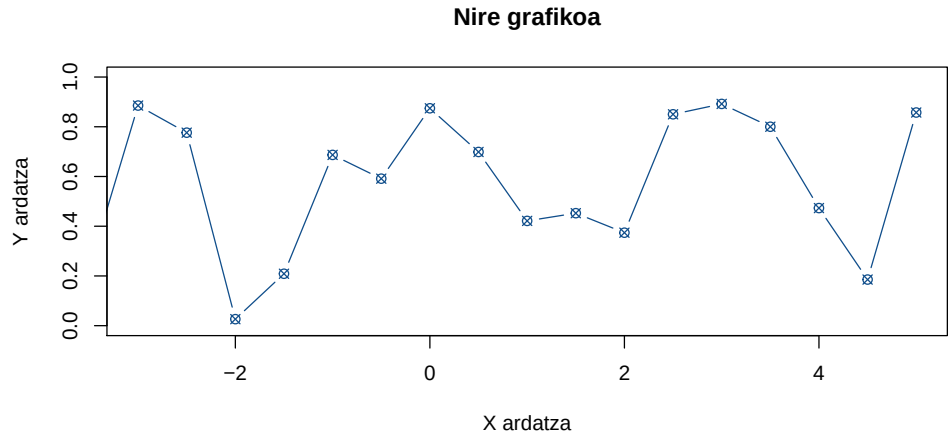
```
x <- runif(150) # Ausazko 150 zenbaki sortzen ditu [0,1] tartean.
y <- log(x)
plot(x=x, y=y)
```

Emaitza 6.1 irudian dago. Ikus daitekeen bezala,`plot` funtzioak bi bektoreak hartzen ditu eta, elementu bakoitzeko, puntu bat gehitzen dio grafikoari, X koordenatua `x` bektoretik hartuz eta Y koordenatua ybektoretik. Funtzioak beste hainbat parametro ditu. Geroago (6.1.3 atalean: Parametro grafikoak) xehetasun gehiagokin ikusiko ditugu aukera horiek. Bitartean, hona gehien erabiltzen diren argumentuak:

- `xlim` eta `ylim` - Biak bi tamainako bektoreak dira eta X edota Y ardatzaren mugak adierazteko erabil daitezke (bektorearen lehenengo elementua balio minimoa eta bigarren elementua balio maximoa)
- `main` eta `sub` - Grafikoari titulua eta azpititulua gehitzeko erabil daitezke. Bien kasuan karaktere-bektore bat pasatu behar diegu.
- `xlab` eta `ylab` - Grafikoaren ardatzen izenak aldatzeko erabiltzen dira bi argumentu hauek.



6.1 irudia. Dispertsio-grafiko sinple bat



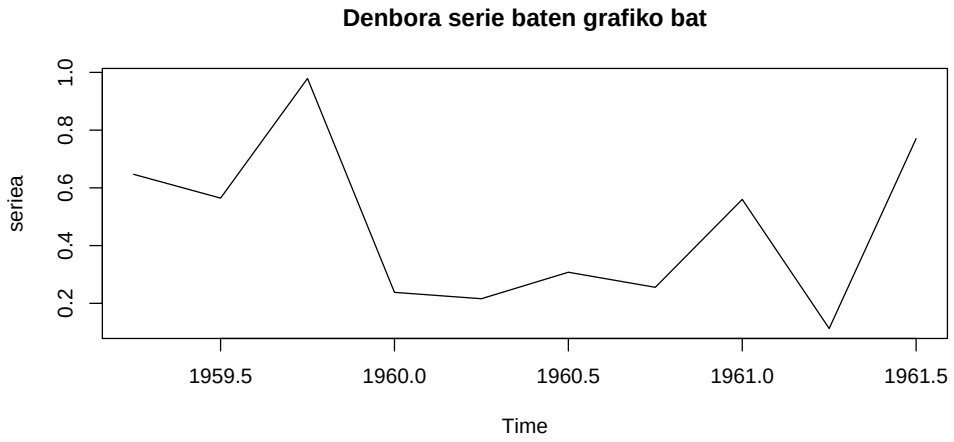
6.2 irudia. 'plot' funtzioaren erabileraren beste adibide bat

- `col` - Puntuen kolorea aldatzeko erabil daiteke. Argumentu honek zenbakizko balioak edo karaktere-bektoreak jaso ditzake[1].
- `pch` - Argumentu honek puntuak adierazteko erabiltzen den karakterea adierazten du. Zenbaki oso bat (1etik 25era) edo karaktere bat izan daiteke.
- `type` - Puntuak bistaratu beharrean marra bat marraztu nahi badugu, argumentu honi `'l'` balioa eman behar diogu, edo `'b'` puntuak eta marra marraztu nahi badugu. Balio horiez gain beste hainbat aukera ditugu (ikusi funtzioaren laguntza).

Ikus dezagun adibide bat argumentu guzti hauekin (emaitza 6.2 irudian).

```
x <- seq(from=-5, to=5, by=0.5)
y <- runif(length(x)) # x-en tamainako bektore bat, [0,1] tarteko ausazko zenbakiekin.
plot(x=x, y=y, xlim=c(-3, 5), ylim=c(0, 1),
     type='b', main="Nire grafikoa",
     xlab="X ardatza", ylab="Y ardatza",
     col="dodgerblue4", pch=13)
```

[1] Karaktere-bekotoreek kolorearen izena adierazten dute. Gauzak horrela, `col='red'` edo `col='green'` idatzi dezakegu. Aukera guztiak dokumentu honetan topa ditzakezu.



6.3 irudia. Denbora serie baten grafikoa

Lehen esan bezala, `plot` funtzioa generikoa da. Horrek esan nahi du hainbat objektu motarentzat inplementatuta dagoela; Lortutako emaitza objektu motaren araberakoa izango da. Adibide moduan, denbora serie bat sortuko dugu `ts` funtzioa erabiliz eta, ondoren, denbora serie hori marraztuko dugu (emaitza 6.3 irudian).

```
seriea <- ts(runif(10), frequency = 4, start = c(1959, 2))
plot(seriea, main="Denbora serie baten grafiko bat")
```

Adibidean ikus daitekeen bezala, zenbait parametroen balioa ezarrita dago, marraztu nahi dugun objektua `ts` klasekoa delako. Adibidez, automatikoki, lerro diagrama bat burutu da.

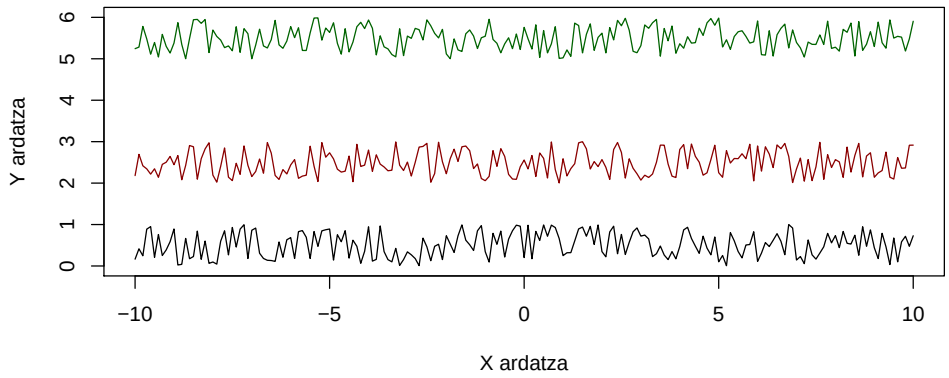
Batzuetan, sortutako grafikoei beste marra edo puntu multzo batzuk gehitu beharko diegu. Horretarako bi funtzio ditugu, `points` eta `lines`. Funtzio horien funtzionamendua `plot` funtzioaren berdina da, baina emaitza grafiko berri batean sartu beharrean uneko grafikoan gehitzen da.

```
x <- seq(from=-10, to=10, by=0.1)
y.1 <- runif(n=length(x), min=0, max=1) # 0 eta 1 arteko 201 ausazko zenbaki
y.2 <- runif(n=length(x), min=2, max=3) # 2 eta 3 arteko 201 ausazko zenbaki
y.3 <- runif(n=length(x), min=5, max=6) # 5 eta 6 arteko 201 ausazko zenbaki

plot(x=x, y=y.1, type='l',
     xlab="X ardatza", ylab="Y ardatza", ylim=c(0,6))

lines(x=x, y=y.2, col="darkred")
lines(x=x, y=y.3, col="darkgreen")
```

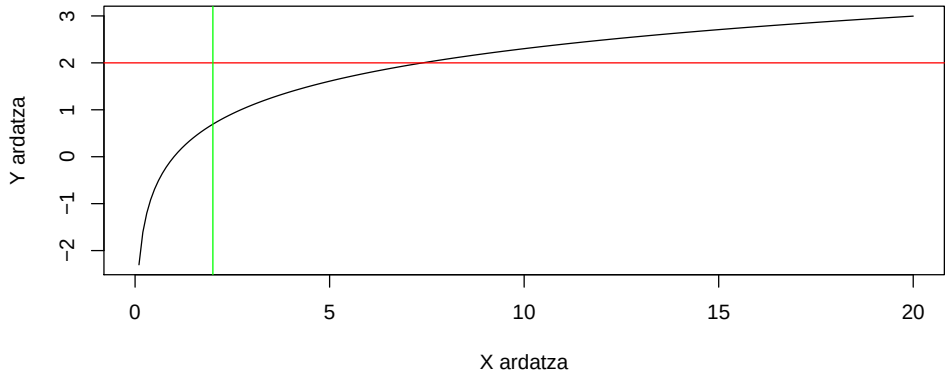
Goiko kodearekin sortutako grafikoa 6.4 irudian ikus dezakezu.

Gainera, grafiko bati lerro horizontal edo bertikal bat gehitu nahi badiogu, `abline` funtzioa erabili dezakegu:

```
x <- seq(from=0, to=20, by=0.1)
y <- log(x)
```



6.4 irudia. Grafiko batean marra bat baino gehiago izan ditzakegu



6.5 irudia. Grafiko batean marra bat baino gehiago izan ditzakegu

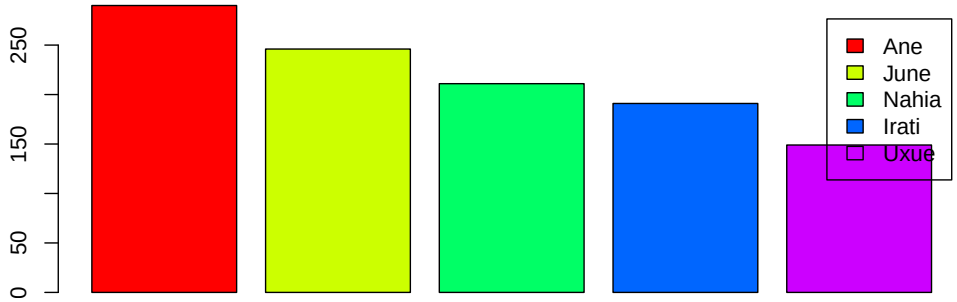
```
plot(x=x, y=y, type='l',
     xlab="X ardatza", ylab="Y ardatza")

abline(h=2, col="red") # Lerro horizontala
abline(v=2, col="green") # Lerro bertikala
```

6.1.2 Beste ohiko grafiko mota batzuk

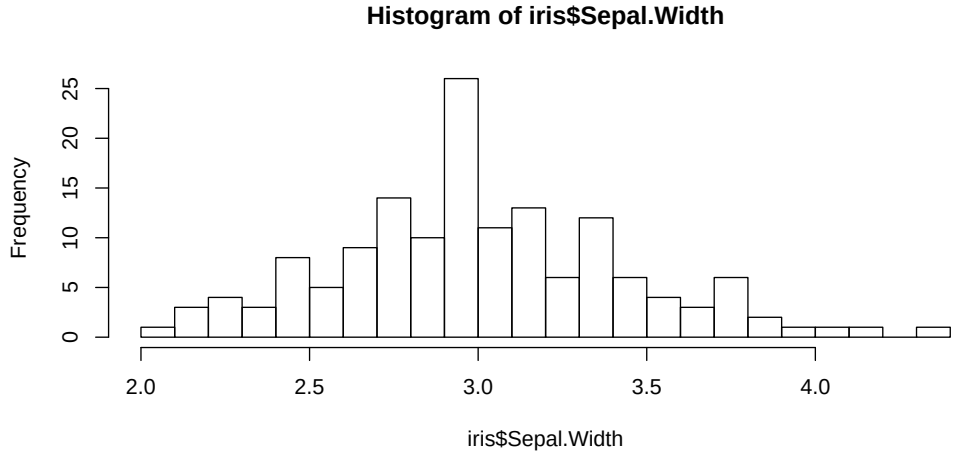
Dispertsio- eta marra-grafikoaz gain, badaude grafiko mota batzuk oso erabiliak direnak. Atal honetan batzuk aipatuko ditugu, barra-diagramak, histogramak eta kutxa-diagramak.

Aldagai kualitatibo edo diskretu bat irudikatzeko barra-diagramak `barplot` funtzioa erabiliz sor ditzakegu. Ikus dezagun adibide bat (emaitza 6.6).

```
altuera <- c(290, 246, 211, 191, 149)
etiketa <- c("Ane", "June", "Nahia", "Irati", "Uxue")
kolorea <- rainbow(5)
barplot(height=altuera, legend.text=etiketa, col=kolorea)
```



6.6 irudia. Barra-diagrama baten adibidea



6.7 irudia. Histograma baten adibidea

Barra-diagramak egiteko barra bakoitzaren altuera, hau da, modalitate bakoitzaren maiztasuna (absolutua edo erlatiboa) behar dugu. Horrez gain, barra bakoitzari etiketa bat eta kolore bat eslei diezaiokegu[2]. Barra-diagrama konplexuagoak egin daitezke, altuera-bektore bat pasatu beharrean matrize bat pasatuz. Kontsulta ezazu funtzioaren laguntza informazio gehiago izateko.

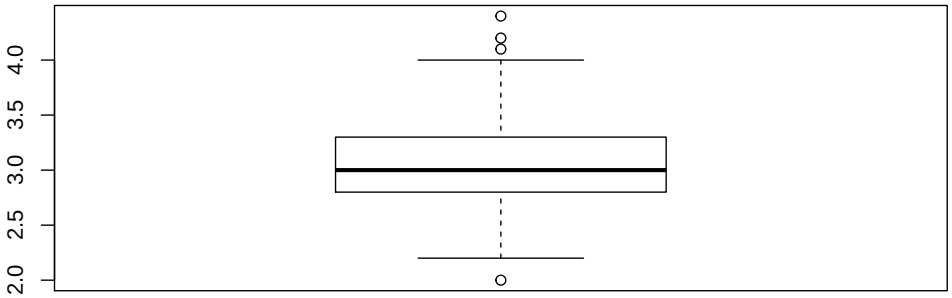
Aldagai jarraitu (edo modalitate askoko aldagai diskretu) bat badugu, histograma baten bidez irudikatu dezakegu. R lengoaian histogramak `hist` funtzioa erabiliz era sinplean sor daitezke. Adibidez Rn topa daitekeen `iris` datu baseko `Sepal.Width` aldagaia irudikatu dezakegu:

```
hist(iris$Sepal.Width, breaks=25)
```

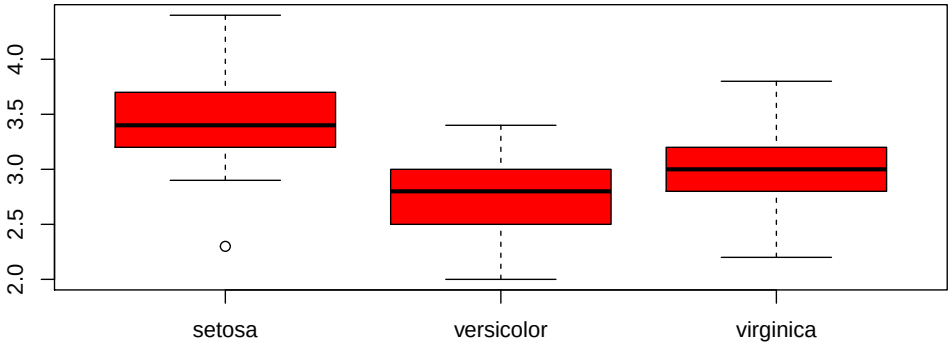
Emaitza @reffig(fig:hist) irudian ikus dezakezu. Beste funtzioak bezala, `hist` funtzioak hainbat parametro ditu emaitzaren itxura aldatzeko. Parametro horien artean interesgarriena `breaks` da, zeinek X ardatza zein tartetan banatuko den adierazten duen.

Atal honekin amaitzeko, kutxa-diagramak nola egiten diren ikusiko dugu. Kasu honetan, `boxplot` funtzioari bektore bakar bat pasatzen badiogu kutxa bakarra sortuko du.

[2]Kasu honetan koloreak eskuz definitu beharrean **paleta** bat sortu dugu, `rainbow` funtzioa erabiliz. Laburki, funtzio honi zenbat kolore behar ditugun pasatzen diogu eta funtzioak kolore bektore bat itzultzen digu. Paletak sortzeko beste hainbat funtzio daude: `hsv`, `hcl`, `gray`, `heat.colors` eta `topo.colors`



6.8 irudia. Kutxa-diagrama adibide bat



6.9 irudia. Kutxa-diagrama adibide bat

```
boxplot(iris$Sepal.Width)
```

Askotan kutxa ezberdinak irudikatu nahi ditugu aldi berean, adibidez, bi aldagai (bat jarraitua eta bat kualitatiboa) aztertu nahi baditugu aldi berean. Adibidez, irudika dezagun `iris` datu baseko `Sepal.Width` aldagai jarraitua, `Species` aldagaiaren arabera (ikusi emaitza 6.9):

```
boxplot(iris$Sepal.Width~iris$Species, col="red")
```

Ohartu, `boxplot` funtzioari ere parametro gehiago gehitu diezazkiogula, kolorea adibidez.

6.1.3 Parametro grafikoak

Aurreko atalean zenbait parametro ikusi ditugu. Berez, parametro asko daude grafikoen itxura kontrolatzeko (letra tamainak, lerroen lodierak, etab.). Parametro-zerrenda osoa ikustea ezinezkoa da (`?par` exekutatuz lor dezakezu), beraz atal honetan bereziki interesgarriak diren parametro batzuk aztertuko ditugu:

- `bty` - Parametro honekin grafikoen *markoa* kontrola dezakegu. Gehien bat marko hori kentzeko erabiltzen da, `bty='n'` aukera gehituz.
- `cex` - Letra eta karaktereen tamaina aldatzeko parametroa. Zenbakizko balioak hartzen ditu.
- `lwd` - Marren lodiera aldatzeko parametroa.
- `lty` - Marra mota aldatzeko parametroa. Zenbakizko balioak hartzen ditu.

- `xlog` eta `ylog` - Balio logikoak. `TRUE` bada, X edota Y ardatza(k) eskala logaritmikoan marraztuko dira.

6.1.4 `layout` funtzioa

Batzuetan irudi berdinean grafiko bat baino gehiago sartu nahi dugu. R lengoaian `layout` funtzioa erabil dezakegu leihoa (edo fitxategia) zatitan banatzeko, gero zati bakoitzean grafiko bat marrazteko. Zati horiek matrize baten bidez definitzen dira, non matrizeak leiho nagusia adierazten duen eta matrizeko balioak (zenbaki kontsekutiboak) leihoan sortuko ditugun zatiak identifikatzen dituzten, bakoitzaren ordena zehaztuz.

Ikus dezagun adibide bat, kontzeptua argitzeko. Lehenik, matrize bat sortuko dugu hiru grafiko marrazteko eskema deskribatzeko. Eskema horretan leiho nagusia horizontalki bi zatitan banatuko dugu, eta beheko zatian grafiko bakarra sartuko dugu. Goiko zatitan, berriz, bi plot sartuko ditugu, biak tamaina berdinekoak. Eskema hori adierazteko ondoko matrizea behar dugu:

```
eskema <- matrix(c(1,2,3,3), byrow=TRUE, ncol=2)
eskema
```

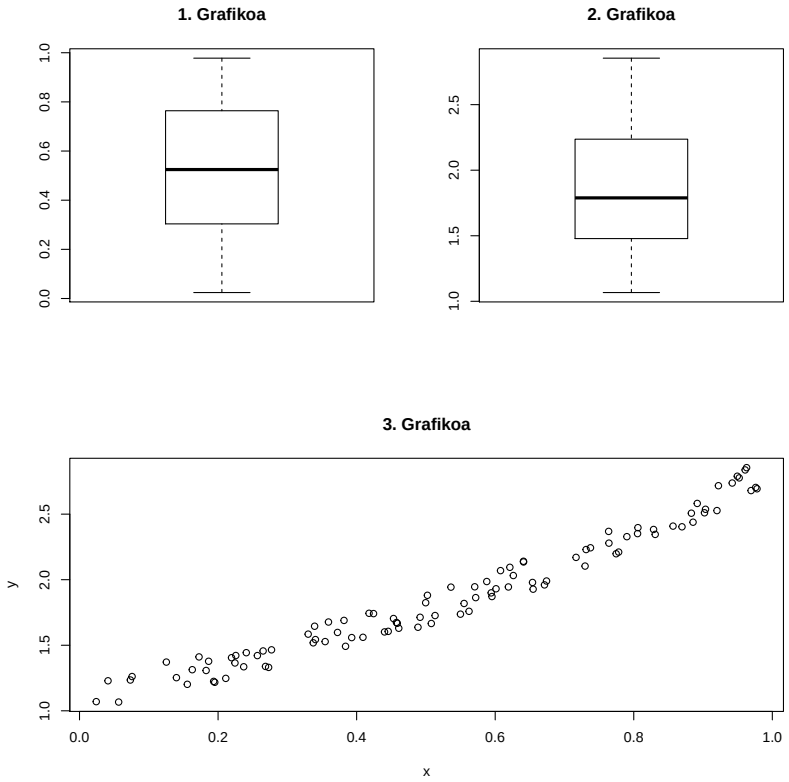
```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    3
```

Leihoa eskema horren arabera zatitzeko `layout` funtzioa erabili behar dugu. Ondoren, hiru plot sartuko ditugu. Kontutan izan lehenengo plota `1` balioa erabiliz definitu dugun zatian marraztuko dela, bigarrena `2` balioa erabiliz definitutakoan eta azkena `3` balioaren bidez adirazitako zatian.

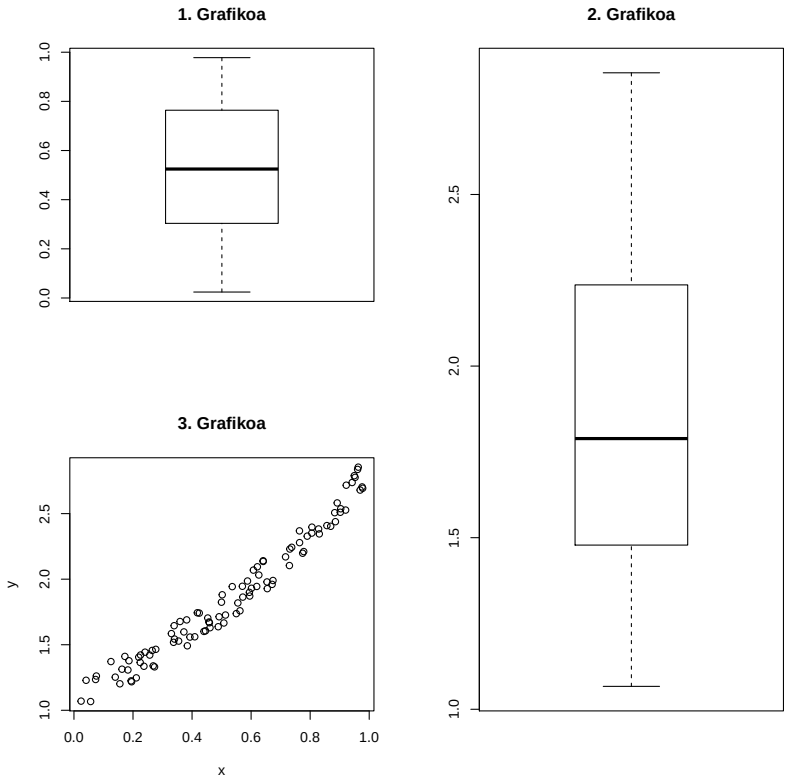
```
layout(mat=eskema)

x <- runif(100)
y <- exp(x) + runif(100, 0, 0.25)

boxplot(x, main="1. Grafikoa")

boxplot(y, main="2. Grafikoa")

plot(x, y, main="3. Grafikoa")
```

Goiko kodearekin sortutako grafikoak 6.10 irudian ikus dezakezu. Era berean, beste konfigurazio posible bat 6.11 irudian ikus dezakegu eta kode honen bitartez lortzen da:

```
eskema <- matrix(c(1,2,3,2), byrow=TRUE, ncol=2)
layout(mat=eskema)
boxplot(x, main="1. Grafikoa")

boxplot(y, main="2. Grafikoa")
```



6.10 irudia. Grafiko bat baino gehiago batera marraz daitezke



6.11 irudia. Grafiko bat baino gehiago batera marraz daitezke

```
plot(x, y, main="3. Grafikoa")
```

6.2 Grafikoak fitxategietan gorde

Orain arte grafikoak sortu ditugu non sortzen diren kontutan izan gabe. R lengoaian **graphical device** kontzeptua dugu, alegia, grafikoak sortzeko euskarriak. Bi euskarri mota ditugu, sistemaren leihoak (orain arte erabili ditugunak) eta fitxategiak.

Hainbat fitxategi mota erabil ditzakegu euskarri moduan, hala nola, PDFak eta png-ak. Fitxategi mota bakoitza sortzeko funtzio bat dugu (`pdf`, `png`, `jpg`, etab.). Oro har, funtzio guztietan hiru parametro nagusi ditugu, `file`, fitxategiaren bide-izena adierazteko eta `width` eta `height`, grafikoaren tamaina adierazteko. Tamainaren aldetik, kontutan izan behar da irudi bektorialeko fitxategietan (PDF, adibidez) tamainak **hazbetetan** adierazten direla defektuz, baina bit-mapako fitxategietan tamainak pixeletan jarri behar dira. Puntu hori oso garrantzitsua da, ezer gehiago aldatzen ez badugu letraren tamainan eragina izango baitu (geroz eta fitxategi txikiagoak, orduan eta letra tamaina handiagoa).

Grafiko bat fitxategi batean sortzeko hiru pausu hartu behar ditugu. Lehenengoa, fitxategia

sortu, bigarrena, grafikoa egin, hirugarrena, fitxategia itxi. Hona hemen adibide bat, PDF fitxategi batekin.

```
pdf(file="grafiko_adibidea.pdf", width=8, heigh=6)

barplot(height=altuera, legend.text=etiketa, col=kolorea)

dev.off()
```

```
## pdf
##   2
```

Azken agindua, `dev.off()`, uneko euskarria ixteko erabiltzen da. Adibidean uneko euskarri hori PDF bat denez, agindu horrekin fitxategia ixten dugu (uneko euskarria leiho bat izanez gero, leiho hori itxiko genuke). Kontutan izan R interpreteak, beste zer edo zer esan ezean, uneko euskarri erabiltzen duela, beraz, grafiko berri bat egin nahi dugunean, aurretik `dev.off` funtzioa erabili behar da irekita egon daitezkeen euskarri guztiak ixteko.

Grafikoak zuzenean fitxategietan sortu nahi ez badugu, leihoan dagoen grafikoa kopiatu dezakegu `dev.copy2pdf` funtzioa erabiliz edo, RStudio erabiltzen badugu, interfazeak ematen duen aukera erabiliz.

6.3 Beste sistema batzuk

R lengoaiaren oinarrizko instalazioak grafikoak egiteko aukera asko ematen ditu, baina badaude pakete asko eskuragarri grafiko sofistikatuagoak egiteko. Atal honetan, horietako batzuk aztertuko ditugu laburki.

6.3.1 ggplot2 paketea

Gaur egun Rn grafikoak egiteko pakete hedatuena *ggplot2* da. Pakete honek grafikoak egiteko filosofia berezi bat inplementatzen du. Alde batetik, guk datu batzuk izango digutu (pentsatu matrize bat dugula, non errenkada bakoitza item bat den eta zutabe bakoitzak item hori deskribatzen duen aldagai bat den). Bestetik, datu horiek bistaratu nahi ditugu eta, horretarako, aldagaiak (zutabeak, alegia) eta grafikoaren elementu estetikoak (kokapena, forma, kolorea, etab.) **mapeatu** behar ditugu. Dispertsio-grafiko batean, adibidez, normalean bi aldagai izaten ditugu, eta aldagai horiek grafikoaren bi ezaugarri estetikokin mapeatzen ditugu, bat kokapen horizontalarekin (X ardatza) eta bestea kokapen bertikalarekin (Y ardatza). Horrelako grafiko batean informazio gehiago sartzerik badaukagu, beste elementu estetiko erabiliz, hala nola, puntuen tamaina eta kolorea.

ggplot2 paketea erabiltzeko datuak formatu konkretu batean egon behar dira: `data.frame` bat non errenkada bakoitza puntu bat den eta zutabe bakoitza aldagai bat. Datuak formatu horretan ez badaude, grafikoa egin aurretik prestatu behar ditugu. Jarraian dagoen adibidean

Eustatetik deskargatutako errolda datuak erabiliko ditugu. Zehazki, lurralde bakoitzeko, urtez urte zenbat emakume eta gizon jaio diren izango dugu.

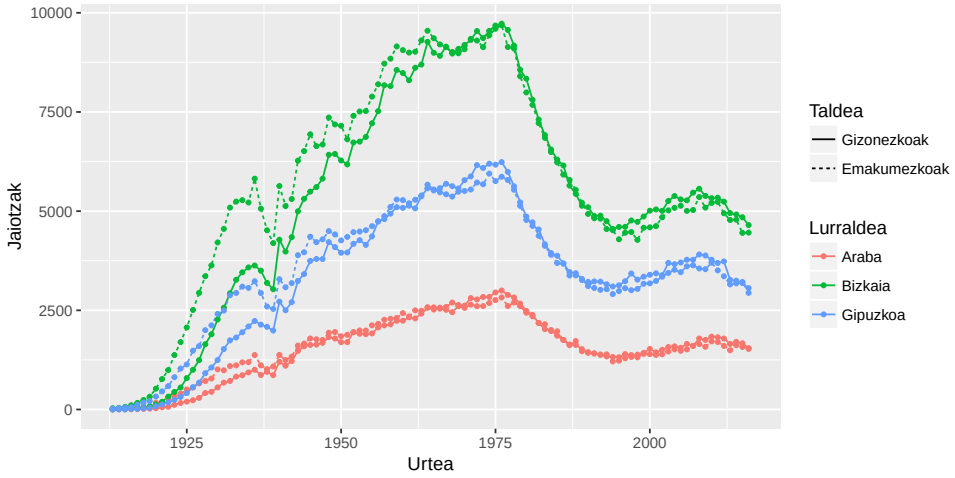
```
datu.gordinak <- read.csv("biztanleria.csv", header=FALSE,
                          stringsAsFactors=FALSE)
head(datu.gordinak)
```

```
##     V1      V2          V3           V4      V5          V6           V7
## 1   NA     EAE         EAE          EAE   Araba       Araba        Araba
## 2   NA Guztira Gizonezkoak Emakumezkoak Guztira Gizonezkoak Emakumezkoak
## 3 2016   18182        9255         8927    3074        1546         1528
## 4 2015   18946        9703         9243    3245        1669         1576
## 5 2014   19456        9870         9586    3327        1706         1621
## 6 2013   19293        9871         9422    3146        1653         1493
##        V8          V9          V10      V11         V12          V13
## 1 Bizkaia     Bizkaia      Bizkaia Gipuzkoa    Gipuzkoa     Gipuzkoa
## 2 Guztira Gizonezkoak Emakumezkoak  Guztira Gizonezkoak Emakumezkoak
## 3    9108        4649         4459     6000        3060         2940
## 4    9298        4846         4452     6403        3188         3215
## 5    9702        4917         4785     6427        3247         3180
## 6    9726        4950         4776     6421        3268         3153
```

Ikus daitekeenez, datuak ez daude behar dugun formatuan eta, hortaz, taula eraldatu behar dugu. Horretarako `lapply` funtzioa erabiliko dugu zerrenda batean lurralde eta talde bakoitzeko datuak ateratzeko eta, ondoren, `do.call` funtzioa erabiliko dugu datu guztiak bata bestearen atzetik jartzeko.

```
a <- lapply(2:ncol(datu.gordinak),
            FUN=function(i){
              return(data.frame(Urtea=datu.gordinak[-(1:2), 1],
                                Lurraldea=datu.gordinak[1, i],
                                Taldea=datu.gordinak[2, i],
                                Jaiotzak=as.numeric(datu.gordinak[-(1:2), i])))
            })
datu.prozesatuak <- do.call(rbind, a)
datu.prozesatuak[sample(1:nrow(datu.prozesatuak), size=10), ]
```

```
##     Urtea Lurraldea       Taldea Jaiotzak
## 916  1933   Bizkaia Emakumezkoak     5241
## 903  1946   Bizkaia Emakumezkoak     6640
## 947  2006  Gipuzkoa      Guztira     7385
## 838  2011   Bizkaia Emakumezkoak     5233
## 184  1937       EAE  Gizonezkoak     6506
## 699  1942   Bizkaia      Guztira     9652
## 196  1925       EAE  Gizonezkoak     1402
## 34   1983       EAE      Guztira    26230
## 315  2014     Araba      Guztira     3327
## 528  2009     Araba Emakumezkoak     1581
```



6.12 irudia. Populazioaren eboluzioa, **ggplot2** paketearekin sortutako grafiko baten bidez bistaratuta

Behin datuak prest, hainbat motako grafikoak egin ditzakegu `ggplot` erabiliz. Lehenengoa marra-grafiko bat izango da, mapeo hauek eginez:

- X ardatza - Urtea
- Y ardatza - Jaiotzak
- Kolorea - Lurraldea (EAE kenduta)
- Lerro-mota - Taldea ('Guztira' kenduta)

Datu guztiak erabiliko ez ditugunez, lehenik aukeraketa egin behar dugu.

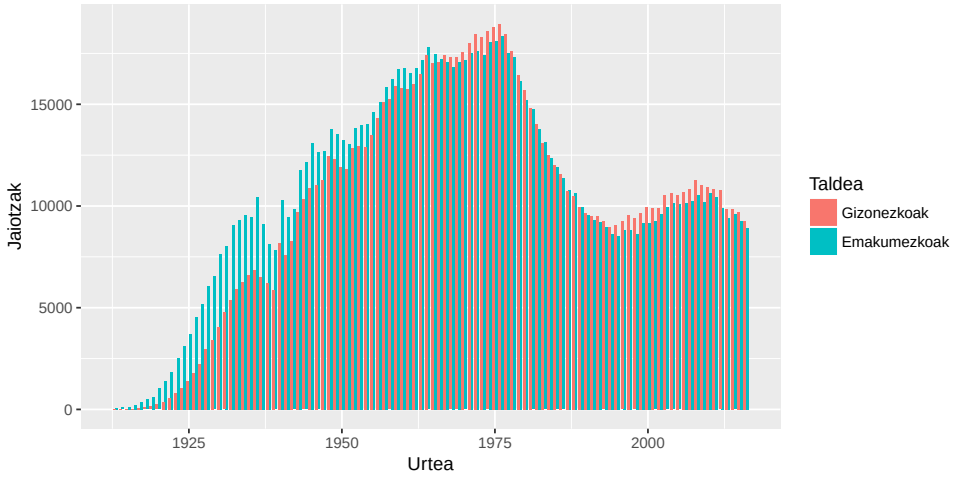
```
df <- subset(datu.prozesatuak,
             subset=datu.prozesatuak$Lurraldea!='EAE' &
               datu.prozesatuak$Taldea!='Guztira')
```

Grafikoa egiteko lehenik `ggplot` funtzioa erabili behar dugu, datuak pasatzeko eta egin nahi dugun mapeoa ezartzeko. Ondoren, nahi ditugun **geruza** guztiak **gehi** ditzakegu `geom_` hasten diren funtzioekin. Gure kasuan marrak egingo ditugu (`geom_line`) eta, gainetik, puntuak jarriko ditugu (`geom_point`).

```
library(ggplot2)
ggplot(data=df, mapping=aes(x=Urtea, y=Jaiotzak,
                            color=Lurraldea, linetype=Taldea)) +
  geom_line() + geom_point(size=1)
```

Sortutako grafikoa 6.12 irudian dago. Ikus daitekeen bezala, mapeoa egiteko `aes` funtzioa erabili behar dugu eta, zuzenean, elementu estetikoei `data.frame` egituran ditugun aldagaiak esleitzen diegu. **ggplot2** paketak duen abantaila da era sinplean oso plot mota ezberdinak egin ditzakegula. Ikusi, adibidez, jarraian dagoen adibidea (datu berdinak erabiliz, baina kasu honetan kopuru totalekin) (emaitza 6.13 irudian):

```
df <- subset(datu.prozesatuak,
             subset=datu.prozesatuak$Lurraldea=='EAE' &
```



6.13 irudia. **ggplot2** paketeko beste adibide bat

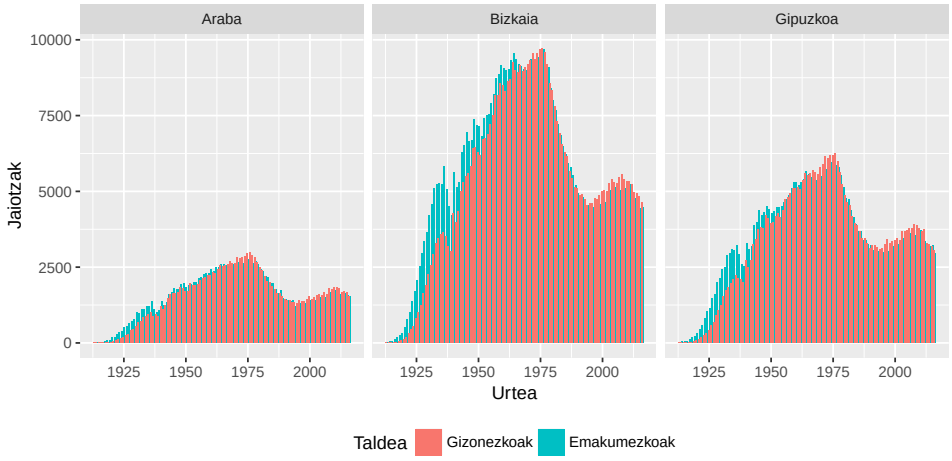
```
                datu.prozesatuak$Taldea!='Guztira')
ggplot(df, mapping=aes(x=Urtea, y=Jaiotzak, fill=Taldea)) +
  geom_bar(stat="identity", position="dodge")
```

Kasu honetan barra-grafikoa egin dugu, `geom_bar` funtzioa erabiliz. Funtzioan bi argumentu ditugu, lehenengoa zuzenean pasatzen dizkiogun Y balioak hartzeko esateko eta bestea kolore ezberdineko barrak elkarrondoan jartzeko esateko. Hare gehiago, era sinple batean grafiko ezberdinak sartu daitezke (emaitza 6.14 irudian):

```
df <- subset(datu.prozesatuak,
             subset=datu.prozesatuak$Lurraldea!='EAE' &
                    datu.prozesatuak$Taldea!='Guztira')
ggplot(df, mapping=aes(x=Urtea, y=Jaiotzak, fill=Taldea)) +
  geom_bar(stat="identity", position="dodge") +
  facet_grid(.~Lurraldea) + theme(legend.position="bottom")
```

6.3.2 Grafiko interaktiboak

ggplot2 paketea grafikoak egiteko gehien erabiltzen dena da, baina ez bakarra. Badaude hainbat pakete oso interesgarriak direnak eta, horien artean, batzuk grafiko interaktiboak sortzeko erabil daitezke. Horien artean **googleVis** paketea dugu, zeinek Google Chart Tools erabiltzen du grafikoak egiteko. Interaktiboak direnez, ez dira zuzenean leiho batean bistaratzen, paketeak web orriak sortzen ditu grafikoak bistaratzeko. Informazio gehiago paketearen dokumentazioan topa dezakezue.



6.14 irudia. Grafiko bat baino gehiago irudi batean, **ggplot2** paketea erabiliz

6.4 Ariketak

1. ariketa Irudikatu, Rko oinarrizko funtzioak erabiliz, ondo funtzio hauek:

- $y = e^x$
- $y = sin(x)$
- $x^2 + y^2 = 100$

2. ariketa Irudikatu goiko funtzioak **ggplot2** paketea erabiliz.

3. ariketa Bilatu nola egin zurtoin-eta-hosto diagramak R lengoaian eta sortu grafiko bat nahi duzun datuekin (hartu adibidez, klaseko apunteetako adibidea).

4. ariketa Sektore-diagramak egiteko `pie` funtzioa erabil dezakezue. Funtzioaren laguntza kontsultatuz, egin horrelako grafiko bat 6.1.2 ataleko nesken izeneko datuak bistaratzeko

5. ariketa Kargatu datuen karpetan dagoen *Europako_populazioa.csv* fitxategia. Bertan Europako lurraldeen populazioaren eboluzioa duzue, 2008tik 2017ra. Egin marra-grafiko bat Suediako populazioa urtez urte erakusteko. Grafikoan bi marra egon behar dira, bat genero bakoitzeko. Egin plot berdina Danimarkarako.

A

R Programazio Estilo-Gida

Liburuan zehar hainbat adibide daude metaheuristikoen erabilera eta inplementazioa erakusteko; adibide hauek R lengoaian daude inplementatuta. Lengoaia interpretatu bat aukeratzearen arrazoia ondorengoa da: Liburuko kodea oinarritzat hartuta, probak egitea erraza izan dadin. Alabaina, liburuko kodea ulergaitza balitz, nekez lortuko genuke gure helburua.

Kodearen ulergarritasuna hobetzearren programazioan estilo-gidak erabili ohi dira. Askotan, horrelako gidak lengoaia garatzen duen taldeak berak idazten ditu, baina R ren kasuan ez da horrela; Hare gehiago, ez dago ezta ere *de facto*koa den estilorik. Dena dela, horrek ez du esan nahi R programazio estilo-gidarik ez dagoenik. Liburu honetan jarraituko ditugun irizpideak (gehienak, behintzat) hainbat gidetatik ateratakoak dira. Bereziki, azpian aipatzen ditugun gidetan oinarritu gara hemen proposatutako estiloa sortzeko:

- Google's R style guide.[1] Googlek eragin handiko hainbat estilo-gida sortu ditu eta, arlo honetan, erreferente nagusienetariko bat da.
- Hadley Wickham's guide.[2] H. Wickhamek **ggplot2** paketea garatu zuen eta gaur egun RStudio konpainian dihardu. Haren gida *Advanced R* liburuan aurki daiteke.
- Bioconductor's coding style.[3] Aurreko eranskinean ikusi dugun bezala, Bioconductor R pakete-biltegi nagusienetariko bat da.

Goian apiatutako gidak ez ezik, beste hainbat dokumentu ere erabili ditugu gure estilo-gida definitzeko, hala nola, C. Guillespies's guide[4], Bernd Bischi' guide[5], G. William's style[6], Genolini's introduction to `S4` (**?**), P.E. Johnson's analysis (**?**) and R. Bååth's paper on naming conventions (**?**).

A.1 Izenak

Estilo-giden artean, aldagaien eta funtzioen izenak sortzeko arauak dira, ziurrenik, heterogenidade handien dutenak. Gai honi buruz oso artikulu interesgarri bat aurki daiteke *The R Journal* aldizkarian (**?**). Artikulu honek gehien erabiltzen diren estrategiak laburbiltzen ditu

[1] http://google-styleguide.googlecode.com/svn/trunk/Rguide.xml
[2] http://adv-r.had.co.nz/Style.htm
[3] http://bioconductor.org/developers/how-to/coding-style/
[4] https://csgillespie.wordpress.com/2010/11/23/r-style-guide/
[5] https://github.com/tudo-r/PackagesInfo/wiki/R-Style-Guide
[6] http://rattle.togaware.com/rattle-rstyle.html

eta, ondoren, CRAN biltegian dauden paketeak aztertzen ditu, estrategia bakoitzaren erabilera erakusteko.

Googleko gidak dio, aldagaien izenei dagokienez, letra xeheak erabili behar direla eta hitzak puntu ikurra erabiliz banandu behar direla –adibidez, `aldagaiaren.izena`–. Gidak formatu hura lehenetsi arren, hitzak letra larriz banatzea ere onartzen dela dio –alegia, `aldagaiarenIzena`–.

Funtzioen kasuan, berriz, puntuak ez erabiltzeko dio gidak; Hitzak letra larriak erabiliz banatu behar dira, lehenengo letra, kasu honetan, larria izanik –esate baterako, `NireFuntzioa`–.

Wickham-ek aldagaien zein funtzioen kasuan hitzak azpimarra erabiliz banandu behar direla dio eta Bioconductor gidak, berriz, letra larriz banatzea gomendatzen du.

Izenen formatua ez da gustuaren kontu bat soilik, erabilera ere –hots, tradizioa– aintzat hartzekoa da. **?** artikuluan egiten den azterketak tradizio hori zein den islatzen du. Bertan, ikus daiteke aldagaien izenei dagokienez, Googleko gidaren gomendioak direla erabilienak. Funtzioen izenen kasuan aldiz, CRAN biltegitik ateratako datuen arabera, Bioconductor gidan proposatutako formatua da nagusi.

Gida gutxik jasotzen dute klaseei izenak emateko arauak. Horietako bat Bioconductor-enarena da, zeinek izenak letra larriz hastea eta hitzak letra larriak erabiliz banatzea proposatzen duen.

Hau dena kontutan hartuz, hone hemen arau batzuk izenak sortzeko.

R1: Aldagaien izenak. Letra xehez idatzi behar dira eta hitzak puntu ikurraren bitartez banandu behar dira.

`nire.aldagaia`	✓
`nire_aldagaia`	×
`Nire.Aldagaia`	×
`nireAldagaia`	×
`NireAldagaia`	×
`nirealdagaia`	×

**R2: Funtzioen izenak.* Lehenengo letra xehea izan behar da eta hitzak letra larrien bitartez banandu behar dira.

`nireFuntzioa`	✓
`nire_funtzioa`	×
`nire.funtzioa`	×
`Nire.funtzioa`	×
`NireFuntzioa`	×
`nirefuntzioa`	×

R3: Klaseen izenak. Lehenengo letra larria izan behar da eta hitzak letra larrien bitartez banandu behar dira.

`NireKlasea`	✓

`nire_klasea`	×
`nire.klasea`	×
`nireKlasea`	×
`Nire.Klasea`	×
`nireklasea`	×

Fitxategien izenak jartzeko arauak ere gida gutxitan jasotzen dira, eta izena bera baino, hedapena zein izan behar den arautzen da. Edonola ere, fitxategien izenak erabakitzean sistema eragile ezberdinek dituzten berezitasunak ere aintzat hartu behar dira. Esate baterako, Unix sistematan fitxategien izenetan tartea erabiltzea ez da oso gomendagarria, arazoak sor ditzakelako.

Hona hemen fitxategien izenak sortzeko poposaturiko arauak:

R4: Fitxategien izenak. *Script* fitxategien izenek letra xehez, zenbakiz eta arazorik ematen ez duten karaterez –hala nola, marratxoa edo azpirra– osatuko dira. Hitzak banatzeko, azpimarra erabili eta, beste sinboloentzat –dezimalak banatzeko sinboloa, esate baterako– berriz, marratxoa. **A7** Gomendioan arau honen salbuespen bat jasotzen da.

R5: Fitxategien hedapenak. *Script* fitxategien hedapena `.R` da, eta ez `.r`. Datu eta `workspace`n kasuetan, berriz, fitxategien hedapena `.RData` izan behar da.

`nire_fitxategia.R`	✓
`nire.fitxategia.R`	×
`nireFitxategia.R`	×
`NireFitxategia.R`	×
`nirefitxategia.R`	×

A.2 Sintaxia

Kodea idaztean, ondoko arau hauek jarraitu behar dira.

R6: Lerroen luzeera. Lerroek 80 karaktereko luzeera izan behar dute, gehienez.

R7: Koskatzea. Kodea koskatzean –begiztetan, esate baterako– bi tarte erabili behar dira, eta inoiz ez tabuladorea.

R8: Esleipenak. Exekutatzen den kodean, *beti* `<-p` sintaxia erabili balioak esleitzean. Funtzioak definitzean eta deitzean, berriz, balioak esleitzeko `=` ikurra erabili.

Beste hainbat lengoaitan gertatzen den legez, R lengoaian kode-blokeak giltza ikurra erabiliz mugatzen dira. Jarraian kode-blokeen inguruko arau batzuko jasotzen dira.

R9: Kode blokeak. Kode-bloke guztiak giltzen artean idatzi behar dira, lerro bakarrekoak ere.

R10: Hasierako giltza I. Ez idatzi hasierako giltza bera bakarrik lerro batean.

R11: Hasierako giltza II. Ez idatzi ezer hasierako giltzaren ostean, iruzkin bat ez bada.

R12: Hasierako giltza III. Ondo koskatu kode-bloke guztiak, kode-blokea erabiltzen duen aginduaren hasiera erreferentziatzat hartuz.

R13: Hasierako giltza IV. Hasierako giltzaren aurretik, tarte bat utzi.

R14: Amaierako giltza. Amaierako giltza bera bakarrik lerro batean idatzi behar da. Arau honek bi salbuespen ditu. Lehenegoa `else` hitz erreserbatua da, `if` egituraren lehenengo blokea ixten duen amaierako giltzaren ostean idatziko dena. Bigarren salbuespena funtzio barruan dauden kode-blokeak dira. Kasu honetan, amaierako giltza eta gero funtzioaren deia ixten duen parantesia(k) idatz daite(z)ke.

A.3 Tarteak

Tarte edo hutsuneen erabileraren inguruan konsensu handia egon arren, badaude giden artean zenbait ezberdintasun. Hauen artean `=` ikurra dago. Googleko gidaren (eta beste zenbait giden) arabera, ikurra horren bi aldeetan tarteak utzi behar dira. Bioconductorreko gidan, berriz, tarterik ez ustea da gomendioa. Guk, kode lerroak albait motzen egitearren, Bioconductorreko gidan proposatutakoari jarrituko diogu.

R15: Eragile bitarrak. Eragile bitar guztiak (hala nola, `+`, `==`, `/`, etab.) hutsunez inguratu behar dira. Salbuespenak `:`, `::` eta `:::` eragileak eta `=` eragilea funtzio-deietan.

R16: Parentesiak. Funtzio-deietan izan ezik, hasierako parentesiaren aurretik tarte bat utzi.

R17: Komak. Koma baten aurrean, tarterik ez; Koma baten ostean, beti tarte bat utzi.

R18: Lerrokatzeko tarteak. Kodea lerrokatzeko (esleipenak, esate baterako), tarte gehiago sartzea badago. Izan ere, tarte hauen erabilera gomendatzen da, kodearen irakurgarritasuna hobetzen baitute.

R19: Iruzkinak I. Iruzkinetan, beti tarte bat utzi `#` sinboloa eta gero.

R20: Iruzkinak II. Kodea eta gero idazen diren iruzkinetan, utzi bi tarte `#` sinboloa baino lehenago.

A.4 Dokumentazioa

R funtzioen dokumentazioa idazteko `*.Rd` fitxategiak erabiltzen dira, baina sistema hori paketeetan dauden funtzioekin bakarrik erabil daiteke. Guk funtzio bat idazten dugunean, haren dokumentazioa idazteko bi aukera ditugu: **roxygen2** paketearen sintaxia erabili (**?**) edo, Googleko gidan gomendatzen den moduan, funtzioaren kodean bertan dokumentazioa txertatu, iruzkinak erabiliz.

R21: Funtzioen dokumentazioa. roxygen2 paketea erabili ezean, gehitu, funtzioaren

hasieran, dokumentazio-iruzkinak. Dokumentazioak, gutxienez, eremu hauek izan behar ditu: Azalpen motz bat (lerro batekoa), argumentuen deskripzioa, funtzioak itzultzen duenaren deskripzioa eta, beharrezkoa balitz, azalpen gehigarriak. Hona hemen egituraren adibide bat.

```
new.function <- function(arg1, arg2) {
  # Description of what the function does
  #
  # Args:
  #   arg1: Description
  #   arg2: Description
  #
  # Returns:
  #   Description of the result
  #
  # Details:
  #   Any relevant information
  #
  ...
}
```

A.5 Fitxategien egitura

Estilo-giden xedea kodea ulergarriagoa izatea da. Helburu hori lortzeko, sintaxia ez ezik, *script* fitxategien antolaketa ere garrantzitsua da. Estiloaren aspektu hau Googleko gidan jasota dago. Jarraian dagoen araua bertan dauden irizpideetan oinarrituta dago.

R22: Fitxategien egitura. *Script* fitxategiak egitura honen arabaera antolatu behar dira (aukeratu behar diren puntuak kasu bakoitzaren arabera):

- *Copyright* eko informazioa.
- Egile(ar)en informazioa. Fitxategia banatu behar bada, gehitu kontaktu-informazioa.
- Fitxategiaren deskripzioa.
- Data eta bertsioari buruzko informazioa.
- Behar diren `source` eta `library` agindu guztiak.
- Funtzioen definizioa.
- Konfigurazio-informazioa (bide-izenak, konstanteak, etab.).
- Exekutatu behar den kodea.

Fitxategien elementuak banatzeko, Wickham-ek iruzkin mota berezia erabiltzea gomendatzen du, kodea zatitzeko.

R23: Kodearen zatiketa. Kodea zatitzeko, `-` edo `=` sekuentzia batekin amaitzen duten iruzkinak erabili

A.6 Funtzioak

Atal honek funtzioak idazterakoan aintzat hartu behar diren arau pare bat jasotzen ditu.

R24: Argumentuak. Funtzio-definizioetan, balio lehenetsirik ez duten argumentuak hasieran idatzi behar dira; Balio lehenetsia dutenak argumentuen zerrendaren amaierara gehitu behar dira.

R25: `return` agindua. Funtzio batek zer edo zer itzultzen badu, `return` agindua erabili beti.

A.7 Beste arau batzuk

R26: Puntu eta koma. Ez erabili `;` ikurra; lerro bat, agindu bat.

R27: Konstante logikoak. Konstante logikoak adierazteko `TRUE` eta `FALSE` erabili beti, eta ez `T`, `F`, `1` edo `0`.

R28: Erabiltzaileari zuzendutako mezuak. Kodearen exekuzioan erabiltzaileari mezuak bidaltzeko `message` funtzioa erabili. Kodeak konpondu dituen arazoen berri emateko `warning` funtzioa erabili eta konpondu ez dituen akatsak, `error` funtzioaren bidez adierazi. `cat` eta `print` funtzioak objektuak bistaratzeko soilik erabili.

A.8 Beste gomendio batzuk

Atal honek beste hainbat gomendio jasotzen du. Gomendio hauek ez daude kodearekin zuzenean lotuta eta, hortaz, bere irakurgarritasunean eragin handirik ez dute. Hori dela eta, gomendioak arau maila ez daukate baina, halere, hemen jasotzen direnak kodea idazteko praktika onak dira.

A1: Izenak. Izen motzak eta ezkerretik eskuinera irakurtzen direnak erabili. Ahal den neurrian, funtzioen izenerako aditzak erabili. Saiatu existitzen diren funtzioen izenak ez erabiltzen.

A2: Objektuak. Ez erabili objektuak beharrezkoak ez badira. Egin behar duzuna `S3` objektuekin egin ahal bada, ez erabili `S4` motako objektuak.

A3: Hizkuntza. Izenak eta iruzkinak idazteko ingelesa erabili beti.

A4: Kopiatu eta itsatsi. Ez kopiatu eta itsatsi kodea. Kode zati bat toki batean baino gehiagotan erabili behar baduzu, funtzio bat sortu.

A5: Paketeak. Askotan erabiliko duzun funtzio-sorta bat baduzu, pakete batean sartzea baloratu.

A6: Fitxategien egitura. *Script* fitxategi batek lerro asko baditu, kodea fitxategitan zatitu (esate baterako, banatu funtzioen definizioa eta exekutatzen den kodea).

A7: Fitxategien izenak. Fitxategi batean klase bakar baten definizioa badago, jarri fitxategiari klasearen izena. Kasu honetan, **R4** araua ez da aplikagarria.

A8: Kode-probak. Kodea probatzen duten funtzioak inplementatu. Helburu hau betetzeko diseinatuta dagoen paketeren bat erabiltzen ez baduzu, jarri proba-funtzioak beste fitxategi batean. Fitxategi hauei izena emateko jatorrizko fitxategien izenari `_test.R` atzizkia gehitu.

A9: Argumentuak. Funtzio-deietan, erabili beti `argumentu=balioa` sintaxia, ez bakarrik balioa.