R HASIBERRIENTZAT

Emmanuel Paradis
paradis@isem.univ-montp2.fr

Itzultzaileak:
Gorka Azkune Galparsoro
gorkaazkune@yahoo.es
Yosu Yurramendi Mendizabal
yosu.yurramendi@ehu.es

Udako **Euskal Unibertsitatea**
Bilbo, 2005

Bizkaiko Foru Aldundia
Kultura Saila

ISBN: 84-8438-077-7
Argitaratzailea: UEU. Erribera 14, 1. D, 48005 Bilbo. http://www.ueu.org
URL: http://www.ueu.org/ueu/buruxkak.htm
Hizkuntza-zuzenketen arduraduna: Ander Altuna Gabiola

Edizio honen ontzailea:
Udako Euskal Unibertsitatea
http://www.ueu.org
argitalpenak@ueu.org

Julien Claude, Christophe Declerq, Élodie Gazave, Friedrich Leisch eta Mathieu Ros eskertu nahi ditut dokumentu honen aurreko bertsioetan egindako iruzkin eta emandako gomendioengatik. R programatu duen programatzaile-taldea ere bereziki eskertu nahiko nuke, programa horren garapenean zehar eginiko esfortzu handiengatik eta “rhelp” eztabaida-gunean erakutsiriko gogoengatik. Eskerrik asko R erabili ondoren galderak eta iruzkinak bidali dizkidaten guztiei, beraiei esker idatzi ahal izan baitut “R hasiberrientzat”.

Aurkibidea

1. Hitzaurrea

Dokumentu honen helburua R erabiltzen hasi nahi duten pertsonentzat nolabaiteko hasiera-puntu bilakatzea da. R-ren oinarrizko erabilera lantzeko asmoz, programa horren funtzionamenduan sakontzea erabaki dut. R-k eskaintzen dituen aukera-piloak ikusita, komeni da hasiberriek kontzeptu eta nozio batzuk bereganatzea eta pixkanaka beren ezagueran aurrera egitea. Bestalde, azalpenak ahalik eta sinpleenak izan daitezen saiatu naiz, hori bai, xehetasun erabilgarrienak ahaztu gabe; xehetasun horiek, askotan taulen bidez azalduko ditut.

R analisi estatistiko eta grafikoetara bideraturiko sistema bat da, Ross Ihaka-k eta Robert Gentleman-ek[1] sortua. Sistema honek nolabaiteko izaera bikoitza du: alde batetik programa bat da, baina bestetik programazio-lengoaia ere bada. Oro har, R, AT&T Bell laboratorioetan sortu zen S lengoaiaren dialektotzat hartzen da. Interesa duenarentzat, S, Insightful[2] etxeak plazaraturiko S-PLUS programaren bidez eskura daiteke. R-ren eta S-ren diseinuan ezberdintasun garrantzitsuak topa daitezke: gai horretan sakondu nahi dutenei Ihaka-k eta Gentleman-ek 1996an argitaraturiko artikulua irakur dezatela gomendatzen diet, edo bestela R-ri buruzko Galdera Ohikoenak[3] izeneko atala kontsulta dezaten. Azken hori, R programarekin batera banatzen da.

R doan banatzen da *GNU*k (*General Public Licence*[4]) ezarritako irizpideen arabera; *R-ren Garapenerako Talde Nuklear* izenarekin ezagutzen diren estatistiko batzuk arduratzen dira R-ren banaketaz eta garapenaz.

R hainbat formatan aurki daiteke: alde batetik, ia oso-osorik C lengoaiaz idatzitako iturburu-kodearen forman (errutina batzuk Fortran lengoaian daude), batez ere Unix eta Linux sistemetarako, eta bestetik, Windows, Linux (Debian, Mandrake, RedHat, SuSe), Macintosh eta Alpha Unix sistemetarako aurrekonpilaturiko artxibo bitarren forman.

1. Ihaka R. eta Gentleman R. (1996): "R: a language for data analysis and graphics", *Journal of Computational and Graphical Statistics,* **5**, 299–314.

2. Ikusi http://www.insightful.com/products/splus/default.html informazio gehiago eskuratzeko

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

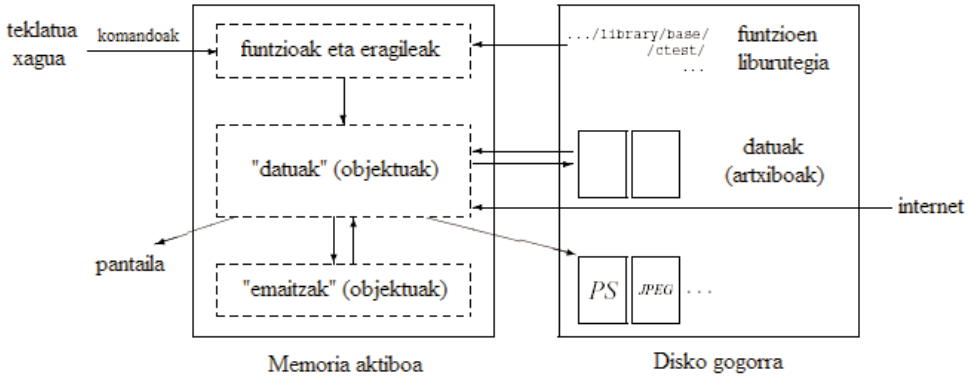
4. Informazio gehiago http://www.gnu.org/ helbidean

R instalatzeko beharrezkoak diren artxiboak, bai iturburu-kodeak zein aurrekonpilaturiko bitarrak erabiliz, Interneteko *Comprehensive R Archive Network* (CRAN)[5] kokalekuan aurki daitezke. Bertan, egokiro instalatzeko argibideak ere badaude. Linux-en bertsioetarako (Debian, …), bitarrak eskuarki R-ren zein Linux-en bertsio eguneratuenetarako topa daitezke; beharrezko baderitzozu, bisita ezazu CRAN gunea.

R-k analisi estatistiko eta grafikoetarako funtzio asko dauzka; grafikoak egin bezain laster ikus daitezke dagokien leihoan, baita gorde ere hainbat formatutan (jpg, png, bmp, ps, pdf, emf, pictex, xfig; eskuragarri dauden formatuak sistema eragileen arabera zehazten dira). Analisi estatistikoen emaitzak pantailan ikus daitezke, eta tarteko emaitza batzuk (*P*- balioak, erregresio-koefizienteak, hondarrak…) gorde, beste artxibo batera esportatu, edo beste analisi batzuetan erabil daitezke.

R lengoaiak, adibidez, begiztak (“loops” ingelesez) programatzeko aukera ematen dio erabiltzaileari, horrek datu-multzoak era jarraian azter ditzan. Programa bakar batean hainbat funtzio estatistiko konbinatzea ere posible da analisi konplexuagoak egiteko. R-ren erabiltzaileek S-rako idatzita dauden hamaikatxo programa dituzte eskura, guztiak Interneten[6]; programa gehienak zuzenean R-n erabil daitezke inongo arazorik gabe.

Hasiera batean R-k oso zaila dirudi aditua ez denarentzat. Hori, ordea, ez da horrela. R-ren ezaugarri nabarmenena bere malgutasuna da, hain zuzen ere. Programa klasikoek analisi baten emaitza zuzenean erakusten badute ere, R-k, “objektu” moduan gordetzen ditu emaitza horiek. Horrela analisi bat egin daiteke, emaitzak berehala erakusteko beharrik gabe. Azaldu berri duguna nahiko arraroa gerta dakioke erabiltzaileari, baina ezaugarri hori oso interesgarria izan ohi da, zeren



1. irudia. R-ren funtzionamenduaren ikuspegi eskematikoa

5. http://cran.r-project.org/
6. Adibidez: http://stat.cmu.edu/S/

erabiltzaileak emaitza orokorretik komeni zaizkion zatiak soilik eskura baititzake. Adibidez, 20 erregresioz osaturiko segida bat korritzen badugu eta erregresio-koefizienteak konparatu nahi baditugu, R-k, estimaturiko koefizienteak bakarrik erakutsi diezazkiguke. Era horretara, emaitzak ilara bakar batean erakuts daitezke. Kontu egin programa klasiko batek 20 leiho irekiko lituzkeela. Aurrerago R-ren malgutasuna beste programa estatistiko ohikoenekin alderatzen dituzten adibideak ikusiko ditugu.

2. Zenbait kontzepturen azalpena hasi aurretik

R behar bezala instalatu ondoren, nahikoa da dagokion artxibo exekutagarria martxan jartzea R erabiltzen hasteko. Kurtsoreak (defektuz ">" ikurra da) adierazten du programa komando bat hartzeko zain dagoela. Windows-en komando batzuk menuen bidez exekuta daitezke (adib. laguntza bilatu, artxiboak ireki…). Baliteke, behin hona iritsita, R lehendabizikoz darabilen batek "eta orain zer egin behar dut?" galdetzea. R lehenengo aldiz erabili aurretik komeni da bere funtzionamenduaren inguruko ideia bat edukitzea, eta horixe da, hain zuzen ere, guk orain egingo duguna. Lehenik eta behin R-k nola funtzionatzen duen ikusiko dugu. Ondoren, R-n objektuak sortzeko balio duen "esleitu" eragilea deskribatuko dut, baita memorian sortu ditugun objektu horiek nola maneiatu ere. Azkenik, laguntza erabiltzen ikasiko dugu. R-n, beste programa askotan ez bezala, laguntza nahiko erabilgarri eta intuitiboa da.

2.1. R-REN FUNTZIONAMENDUA

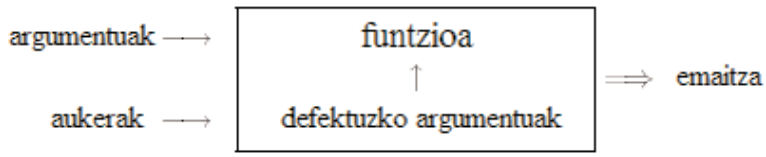
R *Objektuei Zuzendutako* lengoaia da: hitz konplexu horren atzean ezkutatzen da R-ren sinpletasun eta malgutasun guztia. Baliteke R programazio-lengoaia bat izateak bere burua programatzailetzat ez daukan jende askorengan atzerakada sorraraztea. Horrek ez luke horrela izan behar bi arrazoi nagusirengatik. Alde batetik, R lengoaia interpretatu bat da (Java bezalaxe) eta ez konpilatua (C, C++, Fortran, Pascal…). Horrek esan nahi du, teklatuan idazten ditugun komandoak zuzenean exekutatuko direla inongo exekutagarririk sortzeko beharrik gabe.

Bigarren arrazoia R-ren sintaxi sinple eta intuitiboa da. Esaterako, erregresio lineal bat `lm(y ~x)`komandoa idatziz exekuta daiteke. Funtzio bat exekutatu nahi dugun bakoitzean, komandoaren izenaren atzetik *beti* parentesiak erabili behar ditugu, nahiz eta barruan ezer ez idatzi (adib. `ls()`). Funtzioaren izena parentesirik gabe idazten bada, R-k funtzio horren edukia (kodea) erakutsiko digu.

Dokumentu honetan, testuan zehar bestelakorik adierazten ez den bitartean, funtzioen izenak parentesiekin idatziko dira beste objektuetatik bereizteko.

Objektuei Zuzendua izateak ondoriotzat dakar aldagaiak, funtzioak, emaitzak, etab. ordenagailuaren memoria aktiboan *izen* konkretu bat duten *objektu* moduan gordetzea. Erabiltzaileak objektuok aldatu edo manipula ditzake eragileak (aritmetikoak, logikoak eta aldarapenezkoak) zein funtzioak erabiliz. Ez ahaztu funtzioak beraiek ere objektuak direla.

Eragileen erabilera eta funtzionamendua nahiko intuitiboak dira; horien inguruko xehetasunak aurrerago ikusiko ditugu (32. orria). Funtzio bat R-n ondoren agertzen den bezala irudika daiteke:



Argumentuak objektuak izan daitezke ("datuak", formulak, espresioak…). Batzuk funtzioan bertan definituta egon daitezke defektuz, hala ere, argumentu horiek erabiltzaileak alda ditzake aukera batzuen bitartez. R-n funtzio batek argumenturik ez izatea posible da, denak defektuz definituta daudelako (eta beren balioak aukeren bitartez aldatuta), edo funtzioak berak, berez ez duelako argumenturik. Aurrerago ikusiko dugu funtzioak nola erabili eta eraiki (76. orria). Oraingoz, eman berri dugun deskripzioa nahikoa da R-ren oinarrizko funtzionamendua ulertzeko.

R-n ekintza guztiak ordenagailuaren memorian gordetzen diren objektuen bidez gauzatzen dira, inongo aldi baterako artxiboen beharrik gabe (1. irudia). Artxiboak bakarrik irakurri eta idazten dira datuen eta emaitzen (grafikoak…) sarrera-irteeretarako. Funtzioak komando definitu batzuk erabiliz exekuta ditzake erabiltzaileak. Emaitzak zuzenean pantailan ikus daitezke, objektu gisa gorde edo diskoan idatz daitezke zuzenean (grafikoen kasuan bereziki). Emaitzak objektuak direnez azken batean, datutzat jo eta datuak balira bezala azter ditzakegu. Datu-artxiboak disko gogorretik zuzenean edo sarearen bitartez urrutiko zerbitzari batetik irakur daitezke.

Eskuragarri dauden funtzio guztiak R_HOME/library liburutegian daude gordeta (R_HOME da R instalatuta dagoen direktorioa). Direktorio horretan funtzio-*paketeak* aurki ditzakegu. Pakete horiek, aldi berean, direktoriotan egituratuta daude. Bestalde, base izeneko paketea R-ren nukleoa da. Bertan topa daitezke datuak irakurri eta erabiltzeko lengoaiaren oinarrizko funtzioak, funtzio grafiko batzuk eta funtzio estatistiko batzuk (erregresio-lineala eta bariantza-analisia). Pakete bakoitzean R izeneko direktorio bat dago. Direktorio horretan paketearen izen bera duen artxibo bat ere gordetzen da (esaterako, base paketean R_HOME/library/base/R/base izeneko artxibo bat dago). Artxibo hori ASCII formatoan dago eta pakete horrek dituen funtzio guztiak ditu.

Komandorik sinpleena objektu baten izena idaztea da, horrela bere edukia ikus dezagun. Adibidez, `n` izeneko objektu batek 10 balioa badu:

```
> n
[1] 10
```

1 digituak adierazten du `n`-ren bistaratzea bere lehen elementutik hasita egiten dela. Komando horrek `print` funtzioaren erabilera inplizitua du. Aurreko adibidea `print(n)` komandoaren parekoa da (batzuetan `print` funtzioa esplizituki erabili behar da, funtzio baten barnean edo begizta batean kasu).

Objektu baten izenak letra batekin hasi behar du (A-Z eta a-z) eta letrak, digituak (0-9) zein puntuak (.) izan ditzake. R-k letra larrien eta xeheen artean bereitzen du, beraz `x` eta `X` objektu ezberdinak dira (baita Windows sisteman lanean diharduenean ere).

2.2. OBJEKTUEN SORKUNTZA, LISTATUA ETA DEUSEZTATZEA MEMORIAN

Objektu bat sortzeko "esleitu" eragilea erabili behar dugu. Eragile hori gezi baten bidez adierazten da, ken ikurra eta ">" edo "<" ikurrak erabiliz, objektuaren esleipenaren norabidearen arabera:

```
> n <- 15
> n
[1] 15
> 5 -> n
> n
[1] 5
> x <- 1
> X <- 10
> x
[1] 1
> X
[1] 10
```

Objektua jadanik memorian badago, bere balio zaharra desagertu egingo da esleipenaren ondoren (aldaketa hori memoriako objektuetan soilik gertatzen da, eta ez diskoko datuetan). Horrela, esleituriko balioa eragiketa baten edota funtzio baten emaitza izan daiteke:

```
> n <- 10 + 2
> n
[1] 12
> n <- 3 + rnorm(1)
> n
[1] 2.208807
```

`rnorm(1)` funtzioak ausazko datu bat sortzen du, batezbestekoa 0 eta bariantza 1 dituen banaketa normal batean oinarrituz (25. orria). Kontuan hartu espresio bat idatz dezakegula bere balioa inongo objekturi esleitu gabe; kasu horretan emaitza pantailaratu egiten da, baina ez da gordetzen memorian:

```
> (10 + 2) * 5
[1] 60
```

Hemendik aurrera ez ditugu esleipenak idatziko adibideak ulertzeko beharrezkoak ez badira.

`ls` funtzioak memorian dauden objektuak listatu besterik ez du egiten: beraien izenak bakarrik erakusten ditu.

```
> name <- "Ainhoa"; n1 <- 10; n2 <- 100; m <- 0.5
> ls()
[1] "m" "n1" "n2" "name"
```

Ohartu puntu eta koma erabiltzen dela ilara berean komando ezberdinak bereizteko. Karaktere konkretu bat duten objektuak listatu nahi badira `pattern` aukera erabiltzen da (`pat` bezala labur daiteke):

```
> ls(pat = "m")
[1] "m" "name"
```

Zerrendan karaktere konkretu batekin hasten diren objektuak soilik agertzeko:

```
> ls(pat = "^m")
[1] "m"
```

`ls.str()` funtzioak memorian dauden objektuen hainbat xehetasun erakusten ditu:

```
> ls.str()
m : num 0.5
n1 : num 10
n2 : num 100
name : chr "Ainhoa"
```

`pattern` aukera era berean erabil daiteke `ls.str()` funtzioarekin. Funtzio horrek eskaintzen duen beste aukera interesgarri bat `max.level` da. Horren bitartez objektu konposatuen bistaratze-xehetasunak zehatz daitezke. `ls.str()` funtzioak, defektuz, memoriako objektuen xehetasun guztiak erakusten ditu, datu-esparruen ("data frames") zutabeak, matrizeak eta zerrendak barne. Jokamolde horrek sor dezakeen informazio-piloa ekiditeko, eta beraz xehetasun horiek guztiak ez erakusteko, `max.level = -1` aukera erabil dezakegu:

```
> M <- data.frame(n1, n2, m)
> ls.str(pat = "M")
M : 'data.frame': 1 obs. of 3 variables:
$ n1: num 10
$ n2: num 100
$ m : num 0.5
> ls.str(pat="M", max.level=-1)
M : 'data.frame': 1 obs. of 3 variables:
```

Memoriako objektuak ezabatzeko `rm()` funtzioa darabilgu: `rm(x)`-k `x` objektua ezabatzen du, `rm(x,y)`-k bai `x` eta bai `y` ezabatzen ditu eta `rm(list=ls())` idatziz gero, memorian dauden objektu guztiak ezabatzen dira; `ls()` funtzioarekin aipatu ditugun aukera berak erabil daitezke ezabatze hautakor bat egiteko: **`rm(list=ls(pat="^m"))`**.

2.3. LAGUNTZA

R-ren laguntzak oso informazio erabilgarria ematen du funtzioen erabileraren inguruan. Funtzio bati buruzko laguntza zuzenean lor dezakegu. Adibidez:

```
> ?lm
```

idazten badugu, `lm()` funtzioaren laguntza azalduko zaigu R barruan (*eredu lineala*). `help(lm)` edo `help("lm")` komandoek eragin bera dute. Azken funtzio hori ezohiko karaktereekin laguntza eskatzen dugunean erabili behar da:

```
> ?*
Error: syntax error
> help("*")
Arithmetic package:base R Documentation
Arithmetic Operators
...
```

Laguntzari dei egindakoan, leiho edo orri bat agertzen da (sistema eragilearen arabera). Bertan, lehen lerroan funtzioari buruzko informazio orokorra dago, funtzio edo eragile hori dagoen paketea kasu. Ondoren izenburua agertuko da eta bere atzetik, funtzioaren inguruko informazio zehatzagoa ematen duten atal batzuk.

Description: deskripzio motza.

Usage: funtzio baten kasuan, bere izena eta onartzen dituen argumentu guztiak erakusten ditu, baita defektuzko balio posibleak ere (aukerak); eragile baten kasuan berriz, bere erabilera tipikoa deskribatzen du.

Arguments: funtzio baten argumentu guztiak deskribatzen ditu zehatz-mehatz.

Details: xehetasunez josita deskripzioa.

Value: eragile edo funtzioak itzuliko lukeen objektu-mota.

See Also: antzeko funtzio edo eragileak agertzen diren laguntza-orriak.

Examples: eskuarki `examples()` (laguntza ireki gabe) funtzioa erabiliz exekuta daitezkeen adibideak.

Examples atala aztertzea oso egokia da R erabiltzen hasi berri direnentzat. **Arguments** atala ongi irakurtzea ere oso erabilgarria da. Laguntzan ager daitezkeen beste atal batzuk **Note** (ohar gehigarriak), **References** (bibliografia erabilgarria) edo **Author(s)** (egile(ar)en izena(k)) dira.

`help` funtzioak, defektuz, memorian kargatuta dauden paketeetan soilik bilatzen du. `try.all.packages` aukerak (bere defektuzko balioa `FALSE` (faltsua) da) eskura dauden pakete guztietan begiratzeko aukera ematen du bere balioa `TRUE` (egiazkoa) jarriz gero:

```
> help("bs")
Error in help("bs") : No documentation for 'bs' in specified
packages and libraries:
you could try 'help.search("bs")'
> help("bs", try.all.packages=TRUE)
topic 'bs' is not in any loaded package
but can be found in package 'splines' in library
'D:/rw1041/library'
```

Laguntza html formatoan ikusteko (Netscape erabiliz esaterako) ondorengo komandoa idatzi:

```
> help.start()
```

Laguntza-mota horrekin bilaketetan gako-hitzak erabil ditzakegu. **See Also** atalak beste laguntza-orri batzuetara eramango gaituzten loturak ditu. Gako-hitzen bidezko bilaketak egiteko `help.search` funtzioa ere erabil daiteke, baina hori oraindik azterkizun dago (R 1.5.0).

`apropos` funtzioak memorian kargatuta dauden paketeetan argumentutzat emandako hitza duten funtzio guztiak aurkitzen ditu:

```
> apropos(help)
[1] "help" "help.search" "help.start"
[4] "link.html.help"
```

3. Datuen erabilera R-n

3.1. OBJEKTUAK

R-k izena eta edukia dituzten objektuekin lan egiten duela ikusi dugu. Baina objektuek errepresentatzen duten datu-mota zehazten duten *atributuak* ere badituzte. Atributu horien erabilera ulertzeko, demagun 1, 2 edo 3 balioak har ditzakeen aldagai bat dugula: aldagai hori zenbaki oso bat (adibidez, habi batean dauden arrautza-kopurua) edo aldagai kategoriko bat (esaterako, oskoldun-talde bateko banakoen sexua: arra, emea edo hermafrodita) izan daiteke.

Agerikoa da aldagai horren azterketa estatistikoaren emaitzak ez direla berdinak izango kasu bietan: R-n objektuaren atributuek beharrezko informazio guztia ematen digute. Oro har, eta hitz teknikoagoak erabiliz, objektu bati aplikatzen zaion funtzio baten eragina objektuaren atributuen menpekoa da.

Objektu orok *berezko* bi atributu ditu: *mota* eta *luzera*. Mota erabiltzen da adierazteko objektu horren elementuak zein oinarrizko klasetakoak diren; lau mota dira: zenbakizkoa, karakterea, konplexua[7] eta logikoa (`FALSE` [faltsua] edo `TRUE` [egiazkoa]). Badaude bestelako mota batzuk ere, baina ez dituzte datuak errepresentatzen (funtzioak eta espresioak, adibidez). Luzerak adierazten du objektuak duen elementu-kopurua. Objektu baten mota eta luzera ikusteko nahikoa da erabiltzea `mode` eta `length` funtzioak:

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
```

7. Dokumentu honetan ez dugu mota konplexua askotan aipatuko.

Datu bat ezin denean eskuratu, `NA` moduan irudikatzen da (ingelesezko '*not available*'), datuaren mota edozein delarik ere. Oso handiak diren zenbakizko datuak idazkera esponentzialean idatz daitezke:

```
> N <- 2.1e23
> N
[1] 2.1e+23
```

R-k egokiro irudikatzen ditu zenbakizko balio ez-finituak: adibidez, $\pm\infty$ `Inf` eta $-$`Inf` idazten ditu, edo zenbakizkoak ez diren balioak `NaN` (ingelesezko '*not a number*') bezala irudikatzen ditu.

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

Karaktere motako aldagaiak komatxoen (`"`) bidez mugatzen dira. Komatxoa bera aldagaiaren barnean sar daiteke \ ikurra jartzen bada aurretik. Bai pantailan bistaratzeko erabiltzen den `cat` funtzioak, zein artxiboetan idazteko balio duen `write.table` funtzioak bi karaktereok \" batera erabil ditzakete (22. orria, ikus funtzio horren `qmethod` aukera).

```
> cit <- "Hark esan zuen: \"Komatxoak R-ko testuetan jar
daitezke.\""
> cit
[1] "Hark esan zuen: \"Komatxoak R-ko testuetan jar
daitezke.\""
> cat(cit)
Hark esan zuen: "Komatxoak R-ko testuetan jar daitezke."
```

Ondorengo taulan laburtu ditugu objektu-motak eta berauek irudikatzen dituzten datuak.

objektua	**motak**	**mota bat baino gehiago objektu berean?**
bektorea	zenbakia, karakterea, konplexua *edo* logikoa	Ez
faktorea	zenbakia *edo* karakterea	Ez
array-a	zenbakia, karakterea, konplexua *edo* logikoa	Ez
matrizea	zenbakia, karakterea, konplexua *edo* logikoa	Ez
data.frame	zenbakia, karakterea, konplexua *edo* logikoa	Bai
ts	zenbakia, karakterea, konplexua *edo* logikoa	Bai
zerrenda	zenbakia, karakterea, konplexua, logikoa, funtzioa, espresioa, …	Bai

Bektorea aldagai bat da, bere esanahirik ohikoenean. Faktorea aldagai kategoriko bat da. Array bat *k* dimentsioko taula bat da eta matrizea, berriz, $k = 2$ duen array bat besterik ez da. Ohartu array edo matrize batean elementu guztiak mota berekoak direla. 'data.frame' bat (datu-base edo datu-esparrua) luzera berekoak baina mota ezberdinekoak izan daitezkeen bektore edota faktore bat edo gehiagoz osaturiko taula bat da. 'ts' bat denbora-segida bat da eta beraz, atributu gehigarriak ditu hala nola data eta maiztasuna. Azkenik, zerrenda batek edozein motatako objektuak izan ditzake, zerrendak ere barne!

3.2. ARTXIBO BATETIK DATUAK IRAKURRI

R-k lan-direktorioa darabil artxiboak irakurri zein idazteko. Direktorio hori zein den jakiteko, `getwd()` (*get working directory*) komandoa erabil daiteke. Lan-direktorioa aldatzeko `setwd()` funtzioa darabilgu; adibidez, `setwd("C:/data")` edo `setwd("/home/paradis/R")`. Nahitaezkoa da artxiboaren helbide ('path') osoa idaztea artxiboa lan-direktorioan ez badago[8].

R-k testu-artxibo (ASCII) gisa gordetako datuak irakur ditzake ondoko funtzioekin: `read.table` (bere aldaerekin, ikus beherago), `scan` eta `read.fwf`. Beste formatu batzuetan dauden artxiboak ere irakur ditzake R-k (Excel, SAS, SPSS…), baita SQL datu-baseak atzitu ere. Hala ere, horretarako behar diren funtzioak ez daude `base` paketean. Erabiltzaile aurreratuentzat funtzionalitate hori oso erabilgarria den arren, gu ASCII formatuan dauden artxiboak irakurtzeko funtzioetara mugatuko gara.

`read.table` funtzioak datu-esparru ('data.frame') bat sortzen du eta berau da datuak era tabulatu batean irakurtzeko aukerarik onena. Adibidez, data.dat izeneko artxibo bat badugu:

```
> niredatuak <- read.table("data.dat")
```

komandoa idatziz gero, `niredatuak` izeneko datu-esparru bat sortuko da eta aldagai bakoitzak `V1`, `V2`,... izena hartuko du defektuz. Aldagai horiek `niredatuak$V1`, `niredatuak$V2`... edo `niredatuak["V1"]`, `niredatuak["V2"]`... edo baita `niredatuak[,1]`, `niredatuak[,2]`...[9] idatzita banan-banan atzi daitezke. Ondorengo taulan defektuzko balioak dituzten hainbat aukera (R-k darabiltzanak erabiltzaileak ezer adierazten ez badio) ikus daitezke:

8. Windows-en, erabilgarria da Rgui.exe izeneko aliasa sortzea, bere propietateak editatzea eta direktorioa aldatzea "Comenzar en:" eremuan "Alias" gingilaren azpian: hori izango da hortik aurrera lan-direktorio berria R alias hori erabiliz exekutatuz gero.

9. Bada ezberdintasun bat: `niredatuak$V1` eta `niredatuak[,1]` bektoreak dira eta `niredatuak["V1"]`, aldiz, datu-esparru bat. Aurrerago ikusiko ditugu objektuak erabiltzeko xehetasun gehiago (17. orria).

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec
           = ".", row.names, col.names, as.is = FALSE,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill =
           !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#")
```

`file`	artxiboaren izena ("" artean edo karaktere motako aldagai gisa), ziur aski bere helbidearekin lan-direktorioan ez badago (\ ikurra onartzen ez denez / ikurra erabili behar da, Windowsen ere), edo artxiborantz daraman helbide arrotz batekin, URL (http://...) erako zerbait
`header`	artxiboak aldagaien izenak lehen ilaran dituenentz zehazten duen aldagai logikoa (FALSE edo TRUE)
`sep`	artxiboan eremu-bereizle gisa erabiltzen den bereizlea; adibidez, `sep="\t"` tabulazio bat bada
`quote`	aldagaiak karaktere moduan aipatzeko erabili behar diren karaktereak
`dec`	puntu dezimala irudikatzeko erabili behar den karakterea
`row.names`	karaktere edo zenbaki motako izenak dituzten ilarez osaturiko bektorea (defektuz: 1, 2, 3, …)
`col.names`	aldagaien izenak agertzen diren bektorea (defektuz: V1, V2, V3, …)
`as.is`	karaktere motako aldagaiak faktore bihurtzea kontrolatzen du (FALSE bada) edo karaktere gisa mantentzen ditu (TRUE bada); `as.is` karaktere gisa mantendu behar diren aldagaiak markatzen dituen bektore logiko edo zenbakizkoa izan daiteke
`na.strings`	eskura ez dauden balioak kodetzeko darabilgun balioa (NA defektuz)
`colClasses`	zutabeentzat klaseak sortzen dituen karaktere-bektore bat
`nrows`	irakurri behar den ilara-kopuru maximoa (balio negatiboak ez dira kontuan hartzen)
`skip`	datuak irakurri aurretik kontuan hartu ez diren ilaren kopurua
`check.names`	TRUE bada, aldagaien izenak R-rentzat egokiak diren aztertzen du
`fill`	TRUE bada eta ilara guztiek aldagai-kopuru bera ez badute, "zuriuneak" gehitzen ditu
`strip.white`	(`sep`-en menpe) TRUE bada, soberan dagoen espazioa ezabatzen du karaktere motako aldagaien aurretik eta ondoren
`blank.lines.skip`	TRUE bada, zuriz dauden ilarak ez ditu kontuan hartzen
`comment.char`	datu-artxiboan iruzkinak definitzen dituen karakterea; karaktere horrekin hasten diren ilarak ez dira kontuan hartuko irakurketan (aukera hori baliogabetzeko `comment.char =""` idatzi)

`read.table`-en aldaerak interesgarriak dira, defektuzko aukera ezberdinak baitituzte:

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, ...)
read.csv2(file, header = TRUE, sep = ";", quote="\"",
dec=",",
         fill = TRUE, ...)
read.delim(file, header = TRUE, sep = "\t", quote="\"",
dec=".",
         fill = TRUE, ...)
read.delim2(file, header = TRUE, sep = "\t", quote="\"",
dec=",",
         fill = TRUE, ...)
```

`scan` funtzioa `read.table` baino malguagoa da. Azken horretan ez bezala, aldagaien mota zehaztea posible da:

```
> niredatuak <- scan("data.dat", what = list("", 0, 0))
```

Adibide horretan, `scan`-ek data.dat artxiboko hiru aldagai irakurtzen ditu; lehena karaktere bat da eta hurrengo biak zenbakizkoak. Bada aipatzeko moduko beste ezberdintasun bat: `scan()` funtzioak bektore, matrize, datu-esparru, zerrenda… moduko objektuak sor ditzake. Aurreko adibidean `niredatuak` hiru bektorez osaturiko zerrenda bat da. Defektuz, hots, `what` argumentua baztertzen bada, `scan()` funtzioak zenbakizko bektore bat sortzen du. Irakurritako datuak ez badatoz espero z(ir)en mot(ar)ekin bat (bai defektuz zein `what` erabiliz zehaztuta), errore-mezu bat igortzen da. Aukerak ondoren zehazten ditugu:

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
      quote = if (sep=="\n") "" else "'\"", dec = ".", skip =
      0, nlines = 0, na.strings = "NA",flush = FALSE, fill =
      FALSE, strip.white = FALSE, quiet = FALSE,
      blank.lines.skip = TRUE, multi.line = TRUE,
      comment.char = "#")
```

`file`	artxiboaren izena ("" artean), ziur aski bere helbidearekin lan-direktorioan ez badago (\ ikurra onartzen ez denez / ikurra erabili behar da, Windowsen ere) edo artxiborantz daraman helbide arrotz batekin, URL (http://...) erako zerbait; `file` ="" bada, datuak teklatutik sartu behar dira (sarrera ilara zuri batekin amaitzen da)
`what`	datuak zein motatakoak diren zehazten du (defektuz, zenbakizkoa)
`nmax`	irakurri behar diren datuen kopuru maximoa, edo `what` zerrenda motakoa bada, irakurri behar diren ilaren kopuru maximoa (defektuz, `scan`-ek amaiera-marka topatu arte datu guztiak irakurtzen ditu)
`n`	irakurtzeko dauden datuen kopurua (defektuz ez dago mugarik)
`sep`	artxiboan erabiltzen den eremu bereizlea
`quote`	karaktere motako aldagaiak adierazteko erabiltzen den karakterea
`dec`	puntu dezimala irudikatzeko erabili behar den karakterea
`skip`	datuak irakurri aurretik kontuan hartu ez diren ilaren kopurua
`nlines`	irakurri behar den ilara-kopurua
`na.string`	eskura ez dauden balioak kodetzeko darabilgun balioa (NA defektuz)
`flush`	`TRUE` bada, `scan` hurrengo ilarara joango da zutabe guztiak irakurri ondoren (erabiltzaileak iruzkinak gehi ditzake datu-artxiboan)
`fill`	TRUE bada eta ilara guztiek aldagai-kopuru bera ez badute, "zuriuneak" gehitzen ditu
`strip.white`	(`sep`-en menpe) TRUE bada, soberan dagoen espazioa ezabatzen du karaktere motako aldagaien aurretik eta ondoren
`quiet`	`FALSE` bada, `scan`-ek irakurri diren eremuak erakusten ditu ilara batean
`blank.lines.skip`	TRUE bada, zuriz dauden ilarak ez ditu kontuan hartzen
`multi.line`	`what` zerrenda bat bada, artxiboan banako baten aldagaiak ilara bakar batean dauden adierazten du (`FALSE`)
`comment.char`	datu-artxiboan iruzkinak definitzen dituen karaktera; karaktere horrekin hasten diren ilarak ez dira kontuan hartuko irakurketan

`read.fwf` funtzioa *formatu finko zabal*ean dauden artxiboak irakurtzeko erabil daiteke:

```
read.fwf(file, widths, sep="\t", as.is = FALSE,
         skip = 0, row.names, col.names, n = -1)
```

Eremuen zabalera zehazten duen `widths` aukera izan ezik, beste guztiak `read.table()` funtzioarenak berak dira. Adibidez, datuak.txt izeneko artxibo batek eskuinean erakusten ditugun datuak baditu, ondoko komandoarekin irakur daiteke:

```
A1.501.2
A1.551.3
B1.601.4
B1.651.5
C1.701.6
C1.751.7
```

```
> niredatuak <- read.fwf("datuak.txt", widths=c(1, 4, 3))
> niredatuak
  V1   V2  V3
1  A 1.50 1.2
2  A 1.55 1.3
3  B 1.60 1.4
4  B 1.65 1.5
5  C 1.70 1.6
6  C 1.75 1.7
```

3.3. DATUAK GORDE

`write.table` funtzioak objektu baten edukia artxibo batean gordetzen du. Objektua, eskuarki, datu-esparru bat da ('data.frame'), baina beste edozein motatakoa ere izan daiteke (bektorea, matrizea…). Argumentuak eta aukerak honokoak dira:

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
          eol = "\n", na = "NA", dec = ".", row.names = TRUE,
          col.names = TRUE, qmethod = c("escape", "double"))
```

`x`	esportatu behar den objektuaren izena
`file`	artxiboaren izena (defektuz objektua pantailan azaltzen da)
`append`	`TRUE` bada, datuak artxibora gehitzen ditu aurretik zeuden datuak zanpatu gabe
`quote`	bektore logiko edo zenbakizkoa: TRUE bada, karaktere eta faktore motako aldagaiak `""` artean idazten ditu; bestela, zenbakizko bektoreak `""` artean idatzi beharreko aldagaien kopurua adierazten du (bi kasuetan aldagaien izenak `""` artean idazten dira, baina ez `quote=FALSE` bada)
`sep`	artxiboan erabiltzen den eremu bereizlea
`eol`	ilara amaiera adierazten duen karakterea (`"\n"` 'itzulera' da)
`na`	falta diren datuentzat erabili behar den karakterea
`dec`	puntu dezimala irudikatzeko erabili behar den karakterea
`row.names`	artxiboan ilaren izenak idatzi behar diren adierazten duen aukera logikoa
`col.names`	zutabeen izenentzat identifikatzailea
`qmethod`	`quote=TRUE` bada, komatxo bikoitzak tratatu behar diren era zehazten du karaktere motako aldagaietan: "`escape`" (edo "e", defektuz) bada, `"` bakoitza `\"` batez ordezkatzen da; `"d"` bada, `"` bakoitza `""` -rekin ordezkatzen da

Objektu baten edukia artxibo batean gordetzeko era erraz bat `write(x, file="data.txt")` erabiltzea da. Bertan, x, objektuaren izena da (bektorea, matrizea edo array bat izan daiteke). Funtzio horrek bi aukera ditu: `nc` (edo `ncol`) aukerak artxiboan dagoen zutabe-kopurua definitzen du (defektuz, `nc=1` da, `x`

karaktere motakoa bada, eta `nc=5` bestela), eta `append` (logikoa) aukerak, berriz, artxiboan datuak gehitzen ditu bertan zeuden datuak zanpatu gabe (`TRUE`) edo denak ezabatzen ditu (`FALSE`, defektuzko balioa).

Edozein motatako objektu-multzo bat gordetzeko `save(x, y, z, file="xyz.RData")` komandoa erabil daiteke. Makina ezberdinen arteko datu-transferentzia errazteko `ascii = TRUE` aukera erabilgarria da. Datuak (R-ren terminologian *workspace* edo *"lan-espazioa"* izenpean) beranduago karga daitezke memorian `load("xyz.RData")` komandoa idatziz. Bestalde, aipatu baita ere `save.image()` funtzioa `save(list=ls(all=TRUE), file=".RData")` komandoa idazteko modu motzagoa dela (memoriako objektu guztiak .RData artxiboan gordetzen ditu).

3.4. DATUAK SORTU

3.4.1. Sekuentzia erregularrak

Zenbaki osoen sekuentzia erregular bat, 1etik 30era esaterako, ondorengo eran sor daiteke:

```
> x <- 1:30
```

Sortu dugun x bektoreak 30 elementu ditu. Espresio batean, ':' eragileak beste eragile aritmetikoekiko lehentasuna du:

```
> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
```

`seq` funtzioak zenbaki errealen sekuentziak sor ditzake:

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

bertan, lehen zenbakiak sekuentziaren hasiera zehazten du, bigarrenak amaiera eta hirugarrenak sekuentziak izan behar duen hazkundea. Ondorengoa ere erabil daiteke:

```
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Balioak zuzenean ere idatz daitezke `c` funtzioa erabilita:

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Nahi izanez gero, balioak zuzenean teklatutik sar daitezke. Horretarako `scan` funtzioa erabili behar da, aukerak alde batera utzita:

```
> z <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
10:
Read 9 items
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

`rep` funtzioak elementu berdin-berdinak dituen bektore bat sortzen du:

```
> rep(1, 30)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

`sequence` funtzioak zenbaki osoen sekuentzia-segida bat sortzen du. Bertan, sekuentzia bakoitzak argumentu gisa zehaztutako zenbakia(k) d(it)u amaieran.

```
> sequence(4:5)
[1] 1 2 3 4 1 2 3 4 5
> sequence(c(10,5))
[1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
```

`gl` funtzioa (*maila-sortzailea*) oso erabilgarria da, faktoreen segida erregularrak sortzen baititu. Funtzioak `gl(k, n)` forma du; bertan, `k` maila-zenbakia (edo klase-zenbakia) da eta `n` maila bakoitzean nahi dugun kopia-kopurua. Bi aukera ditu funtzioak: `length` aukerak sortu behar den datu-kopurua zehazten du eta `labels` faktoreen izenak zehazteko erabiltzen da. Adibideak:

```
> gl(3, 5)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> gl(3, 5, length=30)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3
3
Levels: 1 2 3
> gl(2, 6, label=c("Arra", "Emea"))
[1] Arra Arra Arra Arra Arra Arra
[7] Emea Emea Emea Emea Emea Emea
Levels: Arra Emea
> gl(2, 10)
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
> gl(2, 1, length=20)
[1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels: 1 2
> gl(2, 2, length=20)
[1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels: 1 2
```

Azkenik, `expand.grid()` funtzioak argumentu gisa emaniko bektoreen edo faktoreen konbinazio posible guztiak dituen datu-esparru bat sortzen du:

```
> expand.grid(a=c(60,80), p=c(100, 300), sexua=c("Arra",
"Emea"))
   a   p sexua
1 60 100 Arra
2 80 100 Arra
3 60 300 Arra
4 80 300 Arra
5 60 100 Emea
6 80 100 Emea
7 60 300 Emea
8 80 300 Emea
```

3.4.2. Ausazko sekuentziak

Banaketa/funtzioa	**Funtzioa**
Gauss (normala)	`rnorm(n, mean=0, sd=1)`
esponentziala	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (*F*)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomiala	`rbinom(n, size, prob)`
geometrikoa	`rgeom(n, prob)`
hipergeometrikoa	`rhyper(nn, m, n, k)`
logistikoa	`rlogis(n, location=0, scale=1)`
lognormala	`rlnorm(n, meanlog=0, sdlog=1)`
binomial negatiboa	`rnbinom(n, size, prob)`
uniformea	`runif(n, min=0, max=1)`
Wilcoxon-en estatistikoa	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

Ausazko datuak sortzeko aukera oso interesgarria da estatistikan eta R-k horretarako gaitasuna du funtzio eta banaketa askorentzat. Funtzio horiek `rfunc(n, p1, p2, ...)` forma izan ohi dute; bertan, `func`-ek banaketa adierazten du, `n` sortu nahi dugun datu-kopurua da eta `p1`, `p2`… banaketaren parametroek hartuko dituzten balioak dira. Aurreko taulan banaketa bakoitzaren xehetasunak agertzen dira, baita parametroek dituzten defektuzko balioak ere (azaltzen ez bada, parametroa erabiltzaileak zehaztu behar duela adierazi nahi da).

Funtzio horiek guztiak `r` letra `d`, `p` edo `q` letrez ordezkatuta erabil daitezke. Horrela, probabilitate-dentsitatea (`dfunc(x, ...)`), metatutako probabilitate-dentsitatea (`pfunc(x, ...)`) eta kuartilaren balioa (`qfunc(p, …)`, $0<p<1$) kalkula daitezke oso era errazean.

3.5. OBJEKTUEN ERABILERA

3.5.1. Objektuak sortu

Aurreko ataletan esleitu eragilea erabiliz objektuak sortzeko era batzuk ikusi ditugu; era horretara sortutako objektuen mota eta klasea, oro har, era inplizitu batean finkatzen dira. Hala ere, posible da objektu bat sortzea bere klase, mota, luzera, etab. zehaztuz. Hurbilketa hori oso interesgarria da objektuen erabileraren ikuspuntutik. Adibidez, objektu 'huts' bat sor daiteke eta ondoren bere elementuak banan-banan alda daitezke; hori `c()` funtzioa erabiliz balio guztiak batera kokatzea baino askoz eraginkorragoa izan daiteke. Baldintza horietan sistema indexatua erabil daiteke, aurrerago ikusiko dugun bezala (33. orria).

Jadanik sorturik dauden objektuetan oinarrituta beste objektu berri batzuk sortzea ere oso interesgarria izan daiteke. Adibidez, ereduen segida bat egokitu nahi bada, formulak zerrenda batean sartu eta ondoren banan-banan atera ditzakegu `lm` funtzio batean sartzeko.

R-ren ikaskuntza fase honetan, honako funtzio hauek ikusteak balio praktikoa ez ezik, didaktikoa ere badu. Objektu baten sorrera esplizituak bere egitura hobeto ulertzea dakarkigu eta aldez aurretik aipatutako ideietan sakontzen lagun diezaguke.

Bektorea. `vector` funtzioak (`mode` eta `length` argumentuak ditu) zenbaki, logiko edo karaktere motako elementuak dituen bektore bat sortzen du, `mode` argumentuaren balioaren arabera (0, `FALSE` edo "" hurrenez hurren). Ondorengo funtzioek gauza bera egiten dute, baina argumentu bakarrarekin (bektorearen luzera): `numeric()`, `logical()` eta `character()`.

Faktorea. Faktore batek aldagai kategoriko baten balioak ez ezik, aldagai horren maila posibleak ere gordetzen ditu (baita datuetan agertzen badira ere). `factor` funtzioak hurrengo aukerak dituen faktore bat sortzen du:

```
factor(x, levels = sort(unique(x), na.last = TRUE),
   labels = levels, exclude = NA, ordered =
   is.ordered(x))
```

`levels`-ek faktorearen maila posibleak zehazten ditu (defektuz, `x`-ren balio bakarrak), `labels`-ek mailen izenak definitzen ditu, `exclude`-k maila bakoitzetik kanporatu beharreko `x` balioak zein diren zehazten du eta `ordered` faktorearen mailak ordenaturik daudenetz adierazten digun argumentu logiko bat da. Gogoan izan `x` dela zenbaki edo karaktere motakoa. Adibideak:

```
> factor(1:3)
[1] 1 2 3
Levels: 1 2 3
> factor(1:3, levels=1:5)
[1] 1 2 3
Levels: 1 2 3 4 5
> factor(1:3, labels=c("A", "B", "C"))
[1] A B C
Levels: A B C
> factor(1:5, exclude=4)
[1] 1 2 3 NA 5
Levels: 1 2 3 5
```

`levels` funtzioak faktore baten maila posibleak ateratzen ditu:

```
> ff <- factor(c(2, 4), levels=2:5)
> ff
[1] 2 4
Levels: 2 3 4 5
> levels(ff)
[1] "2" "3" "4" "5"
```

Matrizea. Matrize bat, azken batean, atributu gehigarri bat (dim) duen bektorea besterik ez da. Atributu hori, aldi berean, 2 luzerako zenbakizko bektore bat da eta matrizearen errenkada- eta zutabe-kopurua definitzen du. Matrize bat `matrix` funtzioa erabiliz sor daiteke:

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
dimnames = NULL)
```

`byrow` aukerak adierazten du ea `data`-n dauden balioek zutabeak (defektuz) edo errenkadak (`TRUE` bada) bete behar dituzten. `dimnames` aukerak, berriz, errenkada eta zutabeei izenak jartzeko aukera ematen digu.

```
> matrix(data=5, nr=2, nc=2)
     [,1] [,2]
[1,]  5    5
[2,]  5    5
> matrix(1:6, 2, 3)
     [,1] [,2] [,3]
[1,]  1    3    5
[2,]  2    4    6
> matrix(1:6, 2, 3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]  1    2    3
[2,]  4    5    6
```

`dim` atributuari balio egokiak emanez matrize bat sortzeko beste era bat dugu (hasiera batean `NULL` balioa du):

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

Datu-esparrua. Jada ikusi dugu `read.table` funtzioarekin datu-esparru ('data.frame') bat sortzen dela era inplizituan; `data.frame` funtzioarekin ere gauza bera egin dezakegu. Argumentu gisa jarritako bektoreek luzera bera izan behar dute, edo baten bat besteak baino motzagoa bada, hainbat aldiz "birziklatzen" da:

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x, n)
  x  n
1 1 10
2 2 10
3 3 10
4 4 10
> data.frame(x, M)
  x  M
1 1 10
2 2 35
3 3 10
4 4 35
> data.frame(x, y)
Error in data.frame(x, y) :
arguments imply differing number of rows: 4, 3
```

Datu-esparruan faktore bat sartu nahi bada, bektore(ar)en luzera bera izan behar du. Zutabeen izenak aldatzea posible da `data.frame(A1=x, A2=n)` idatziz. Errenkaden izenak ere zehatz daitezke `row.names` aukerarekin. Hori, noski, datu-esparruak dituen errenkada adinako luzera duen karaktere motako bektore bat izango da. Azkenik, kontuan izan datu-esparruek matrizeen `dim` atributuaren antzeko atributu bat dutela.

Zerrenda. Zerrenda bat datu-esparru baten antzera sortzen da, `list` funtzioa erabiliz. Edozein objektu-mota sar daiteke zerrendetan. `data.frame()` funtzioarekin ez bezala, objektuen izenak ez dira defektuz hartzen; aurreko adibidetik `x` eta `y` bektoreak hartuko ditugu:

```
> L1 <- list(x, y); L2 <- list(A=x, B=y)
> L1
[[1]]
[1] 1 2 3 4
[[2]]
[1] 2 3 4
> L2
$A
[1] 1 2 3 4
$B
[1] 2 3 4
> names(L1)
NULL
> names(L2)
[1] "A" "B"
```

Denbora-segidak. `ts` funtzioak "`ts`" motako (denbora-segida) objektu bat sortzen du, bektore batean (aldagai bakarreko denbora-segida) edo matrize batean (aldagai anitzeko segida) oinarrituta. Era honetako objektu batek dituen aukerak ondorengoak dira:

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
   deltat = 1, ts.eps = getOption("ts.eps"), class, names)
```

`data`	bektore edo matrize bat
`start`	lehen behaketaren denbora, zeina zenbaki bat zein bi osokoz osaturiko bektore bat izan daitekeen (ikus beheko adibidea)
`end`	azken behaketaren denbora. `start` bezalaxe zehazten da
`frecuency`	denbora unitateko eginiko behaketa-kopurua
`deltat`	ondoz-ondoko behaketen arteko laginketa-denboraren alderantzizkoa (adb. 1/12 hileko datuetarako); `deltat` ala `frecuency` zehaztu behar da soilik
`ts.eps`	segidak konparatzeko erabili behar den tolerantzia. Maiztasunak berdintzat hartzen dira, beraien arteko kendura `ts.eps` baino txikiagoa bada
`class`	objektuari esleitu behar zaion klasea; oro har, "`ts`" izango da aldagai bakarreko denbora-segida bada eta `c("mts", "ts")` aldagai anitzeko segida bada

`names` aldagai anitzeko segida baten kasuan, segida bakoitzaren izenak dituen karaktere motako bektore bat; defektuz, `data`-ren zutabeen izenak, edo `Serie1`, `Serie2`,…

`ts()` funtzioarekin sortutako denbora-segida batzuen adibideak:

```
> ts(1:10, start = 1959)
Time Series:
Start = 1959
End = 1968
Frequency = 1
[1] 1 2 3 4 5 6 7 8 9 10
> ts(1:47, frequency = 12, start = c(1959, 2))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1959   1   2   3   4   5   6   7   8   9  10  11
1960  12  13  14  15  16  17  18  19  20  21  22  23
1961  24  25  26  27  28  29  30  31  32  33  34  35
1962  36  37  38  39  40  41  42  43  44  45  46  47
> ts(1:10, frequency = 4, start = c(1959, 2))
     Qtr1 Qtr2 Qtr3 Qtr4
1959       1    2    3
1960  4    5    6    7
1961  8    9   10
> ts(matrix(rpois(36, 5), 12, 3), start=c(1961, 1),
frequency=12)
          Series 1 Series 2 Series 3
Jan 1961        8        5        4
Feb 1961        6        6        9
Mar 1961        2        3        3
Apr 1961        8        5        4
May 1961        4        9        3
Jun 1961        4        6       13
Jul 1961        4        2        6
Aug 1961       11        6        4
Sep 1961        6        5        7
Oct 1961        6        5        7
Nov 1961        5        5        7
Dec 1961        8        5        2
```

Espresioa. Espresio motako objektuek garrantzia handia dute R-n. Espresio bat R-k ulertzen duen karaktere segida bat da. Komando baliagarri bakoitza espresio bat da. Komando bat teklatuan idazten denean, R-k *ebaluatu* eta baliagarria bada, exekutatu egiten du. Ebaluatu gabeko espresioak sortzea oso erabilgarria izan daiteke askotan: horixe bera egiten du `expression` funtzioak. Noski, posible da espresio hori ondoren ebaluatzea `eval()` funtzioarekin.

```
> x <- 3; y <- 2.5; z <- 1
> exp1 <- expression(x / (y + exp(z)))
> exp1
expression(x/(y + exp(z)))
> eval(exp1)
[1] 0.5749019
```

Espresioak, esaterako, grafikoetan ekuazioak sartzeko erabil daitezke (47. orria). Espresio bat karaktere motako aldagai batean oinarrituz sor daiteke. Funtzio batzuek espresioak darabiltzate argumentutzat; adibidez, deribatu partzialak kalkulatzen dituen `D()` funtzioa:

```
> D(exp1, "x")
1/(y + exp(z))
> D(exp1, "y")
-x/(y + exp(z))^2
> D(exp1, "z")
-x * exp(z)/(y + exp(z))^2
```

3.5.2. Objektuen bihurketa

Irakurlea honezkero ohartu da objektu batzuen arteko ezberdintasuna oso txikia dela; beraz, objektu baten mota aldatzea nahiko erraza izan beharko litzateke, esaterako, atributu batzuk aldatuta. Bihurketa horiek `as.`*`zerbait`* funtzioen bidez egin daitezke. R-k (1.5.1 bertsioa) mota honetako 77 funtzio ditu base paketean, beraz, ez dugu askoz gehiago landuko gai hau.

Bihurketa horren emaitza, noski, bihurtu dugun objektuaren atributuen menpe dago. Bihurketek, eskuarki, oso arau intuitiboak erabiltzen dituzte. Hurrengo taulak moten bihurketak jasotzen ditu.

Bihurketa	**Funtzioa**	**Arauak**
Zenbakizko motara	`as.numeric`	FALSE → 0 TRUE → 1 "1", "2",... → "1", "2", … "A", … → NA

Mota logikora	`as.logic`	0 → `FALSE`
		beste zenbakiak → `TRUE`
		`"FALSE"`, `"F"` → `FALSE`
		`"TRUE"`, `"T"` → `TRUE`
		beste karaktereak → `NA`

Karaktere motara	`as.character`	1, 2, … → `"1"`, `"2"`, …
		`FALSE` → `"FALSE"`
		`TRUE` → `"TRUE"`

Klase ezberdineko objektuen arteko bihurketak egiten dituzten funtzioak ere badaude (`as.matrix`, `as.data.frame`, `as.ts`, `as.expression`...). Funtzio horiek, *mota* ez ezik beste atributu batzuk ere alda ditzakete bihurketa gauzatzean. Beste behin, bihurketaren emaitzak intuitiboak dira oro har. Faktoreak zenbakizko balio bihurtzea, adibidez, ohiko eragiketa bat da. Kasu horretan, R-k faktorearen mailen *zenbakizko kodifikazioa* darabil (eta ez faktorearen beraren balio literalak) bihurketa egiteko:

```
> fac <- factor(c(1, 10))
> fac
[1] 1 10
Levels: 1 10
> as.numeric(fac)
[1] 1 2
```

Bihurketa egitean faktorearen balio literalak mantentzea nahi bada, lehenbizi faktorea karakterera bihurtu behar da, eta ondoren zenbakizkora.

```
> as.numeric(as.character(fac))
[1] 1 10
```

Jokabide hori oso erabilgarria izan daiteke artxibo batean balio ez-zenbakizkoak dituen zenbakizko aldagai bat badugu. Lehenago ikusi dugun bezala, `read.table()` funtzioak, defektuz, zutabea faktore bezala irakurriko du.

3.5.3. Eragileak

Jada esan dugun bezala, R-n[10] hiru motatako eragileak daude. Hona hemen zerrenda.

Eragileak					
Aritmetikoak		Konparaziozkoak		Logikoak	
`+`	batuketa	`<`	txikiagoa	`!x`	EZ logikoa
`-`	kenketa	`>`	handiagoa	`x&y`	ETA logikoa
`*`	biderketa	`<=`	txikiagoa edo berdin	`x&&y`	ETA logikoa
`/`	zatiketa	`>=`	handiagoa edo berdin	`x|y`	EDO logikoa
`^`	potentzia	`==`	berdin	`x || y`	EDO logikoa
`%%`	modulua	`!=`	ezberdin	`xor(x, y)`	EDO esklusiboa
`%/%`	osoen zatiketa				

Konparaziozko eragileek eta eragile aritmetikoek bi elementurekin egiten dute lan (`x+y`, `a<b`). Eragile aritmetikoek zenbakizko aldagaietan eta konplexuetan dute eragina, baina baita aldagai logikoetan ere; kasu horretan, balio logikoak zenbakizko baliotzat hartzen dira. Konparaziozko eragileak, berriz, edozein motatako aldagaietan aplika daitezke eta balio logiko bat edo gehiago itzuliko dizkigute.

10. Karaktere hauek ere eragileak dira R-n: `$`, `[`, `[[`, `:`, `?`, `<-`.

Eragile logikoek objektu logiko batean (`!`) edo bitan eragin dezakete eta balio logiko bat edo gehiago itzul ditzakete. “ETA” eta “EDO” eragileak bi formatan aurkezten dira: alde batetik, forma sinplea, non eragileak objektuaren elementu guztiak konparatzen dituen eta egin dituen konparazioak adina balio logiko itzultzen dituen. Bestetik forma bikoitza dugu; hor, eragileak objektuen lehen elementuak bakarrik konparatuko ditu.

0<x<1 moduko ezberdintzak idazteko nahitaezkoa da “ETA” eragilea erabiltzea. Aurreko ezberdintza horrela idatziko genuke R-n: `0<x & x<1`. `0<x<1` espresioak ere balio du, baina ez du itzuliko espero dugun emaitza, bi eragileak mota berdinekoak direnez, ezkerretik eskuinera hurrenez hurren exekutatuko baitira. Lehendabizi `0<x` alderaketa egingo du eta itzulitako balioa 1ekin konparatuko du (`TRUE` edo `FALSE` `<1`): kasu horretan, balio logikoa zenbakizko balio bihurtuko da inplizituki (`1` edo `0` `<1`).

Konparaziozko eragileek eragigaitzat dituzten objektuen elementu guztiak konparatzen dituzte (txikienen balioak birziklatuz behar izanez gero) eta tamaina bereko objektu bat itzultzen dute. Bi objektu “guztiz” konparatu ahal izateko, `identical` funtzioa erabili behar da nahitaez:

```
> x <- 1:3; y <- 1:3
> x == y
[1] TRUE TRUE TRUE
> identical(x, y)
[1] TRUE
```

3.5.4. Objektu baten balioak atzitu: sistema indexatua

Sistema indexatua objektu baten elementuak nahi bezala atzitzeko era malgu eta eraginkorra da. Adibidez, `x` bektore baten hirugarren elementua atzitu nahi badugu, `x[ 3]` idatzi besterik ez dugu. `x` matrize edo datu-esparru bat bada, i. errenkada eta j. zutabea atzitzeko `x[ i, j]` idaztea nahikoa da. Hirugarren zutabeko balio guztiak aldatzeko, berriz:

```
> x[, 3] <- 10.2
```

Errenkada adierazten ez dugunean, guztiak hartzen ditugu. Sistema indexatua bi dimentsio baino gehiagoko matrizeentzat oso erraz heda daiteke (esaterako hiru dimentsioko matrize bat: `x[ i, j, k]`, `x[ , , 3]`, …). Ez ahaztu indexazioa egiteko kortxete errektangeluarrak erabili behar direla eta parentesiak funtzio baten argumentuak zehazteko ditugula:

```
> x(1)
Error: couldn’t find function “x”
```

Indexazioa errenkada edo zutabe bat edo gehiago ezabatzeko erabil daiteke. Adibidez, `x[ -1, ]` komandoak lehen errenkada ezabatzen du, eta `x[ -c(1, 15), ]` komandoak gauza bera egingo du lehen eta hamabosgarren errenkadekin.

Bektore, matrize edo array baten balioak atzitzeko, konparaziozko espresio bat erabil daiteke indize gisa:

```
> x <- 1:10
> x[x >= 5] <- 20
> x
[1] 1 2 3 4 20 20 20 20 20 20
> x[x == 1] <- 25
> x
[1] 25 2 3 4 20 20 20 20 20 20
```

Indexazio logikoaren aplikazio praktiko bat ondorengoa da: demagun osoen bektore baten zenbaki bikoitiak soilik aukeratu nahi ditugula:

```
> x <- rpois(40, lambda=5)
> x
[1] 5 9 4 7 7 6 4 5 11 3 5 7 1 5 3 9 2 2 5 2
[21] 4 6 6 5 4 5 3 4 3 3 3 7 7 3 8 1 4 2 1 4
> x[x %% 2 == 0]
[1] 4 6 4 2 2 2 4 6 6 4 4 8 4 2 4
```

Beraz, sistema indexatuak konparaziozko eragileek itzulitako balioak darabiltza. Balio horiek lehenago kalkula daitezke eta behar izanez gero, birziklatu ere egin daitezke:

```
> x <- 1:40
> s <- c(FALSE, TRUE)
> x[s]
[1] 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

Aurreko kasuan, `x` bektoreari `s` baldintza aplikatu diogunez, lehen elementua ez da aukeratzen (`FALSE`), bigarrena bai (`TRUE`), hirugarrena ez (`FALSE`), etab. Hori, hautazko indexazioa egiteko era trinko eta boteretsu bat da, begiztarik erabili gabe.

Indexazio logikoa datu-esparruekin ere erabil daiteke, hori bai, esparruaren zutabe bakoitza mota ezberdinekoa izateak gehitzen duen zailtasuna kontuan hartu behar da.

Zerrendetan, berriz, elementuak atzitzea oso erraza da (elementuak edozein motatakoak izan daitezke). Horretarako kortxete errektangeluar bikoitzak erabili behar ditugu; adibidez, `my.list[[ 3]]` idatziz gero `my.list`-eko hirugarren

elementua atzituko dugu. Emaitza bera ere indexa dezakegu bektore, matrize, eta abarrentzat azaldu dugun eran. Hirugarren objektu hori bektore bat bada, bere balioak `my.list[[3]][i]` idatzita alda ditzakegu, hiru dimentsioko matrize bat bada `my.list[[3]][i, j, k]`, ...

3.5.5. Izendun objektuen balioak atzitu

izen kontzeptua behin baino gehiagotan azaldu zaigu jada. Izenak, atributuak dira eta mota askotakoak daude (*names*, *colnames*, *rownames*, *dimnames*). Izenen inguruko ideia oso sinpleen inguruan jardungo dugu orain, batez ere, objektuen elementuak atzitzeko duten erabilerarekin erlazionatutako ideien inguruan.

Objektu baten elementuek izena badute, izen horiek indizetzat erabilita atera daitezke. Elementuak era horretara ateraz gero, jatorrizko objektuaren atributuek bere horretan jarraituko dute. Adibidez, `DF` datu-esparru batek `x`, `y` eta `z` aldagaiak baditu, `DF["x"]` komandoak `x` soilik duen datu-esparru bat aterako du; **`DF[c("x", "y")]`** komandoak bi aldagai horiek dituen datu-esparru bat aterako du. Honek zerrendekin ere funtzionatzen du elementuek izenak badituzte.

Irakurlea ohartu den bezala, hemen erabili dugun indizea karaktere motako bektore bat da. Lehenago deskribatu ditugun zenbakizko bektore edo bektore logikoekin bezala, bektore hori bere baitan defini daiteke eta geroago erabili elementuak ateratzeko.

Datu-esparru batetik bektore edo faktore bat ateratzeko `$` ikurra erabil daiteke (ad. `DF$x`). Prozedura hori zerrendetan ere aplika daiteke.

3.5.6. Datu-editorea

Zenbakizko objektu bat editatzeko, kalkulu-orri baten antzeko editore grafiko bat erabil daiteke. Adibidez, `X` matrize bat bada, `data.entry(X)` komandoak editore grafiko bat irekiko du; bertan, matrizearen balioak aldatu edo zutabe zein errenkada berriak gehi ditzakegu.

`data.entry` funtzioak argumentutzat emaniko objektua zuzenean aldatzen du bere emaitza esleitzeko inongo beharrik gabe. `de` funtzioak, berriz, argumentutzat emaniko objektuak zerrenda batean itzultzen ditu, gehienetan aldatuta. Emaitza hori, defektuz, pantailan erakusten da, baina beste funtzio askorekin bezala, beste objektu bati eslei dakioke.

Datu-editorearen xehetasunak sistema eragilearen menpe daude (editorea oraindik ez da inplementatu sistema guztientzat).

3.5.7. Funtzio aritmetiko sinpleak

R-n, datuak manipulatzeko funtzio-piloa dago. Errazena jada ikusi dugu: `c`. Horrek, parentesi artean idatzitako objektuak kateatzen ditu. Adibidez:

```
> c(1:5, seq(10, 11, 0.2))
[1] 1.0 2.0 3.0 4.0 5.0 10.0 10.2 10.4 10.6 10.8 11.0
```

Bektoreak espresio aritmetiko klasikoekin manipula daitezke:

```
> x <- 1:4
> y <- rep(1, 4)
> z <- x + y
> z
[1] 2 3 4 5
```

Luzera ezberdineko bi bektore gehi ditzakegu. Kasu horretan bektore motzena birziklatu egiten da. Adibideak:

```
> x <- 1:4
> y <- 1:2
> z <- x + y
> z
[1] 2 4 4 6
> x <- 1:3
> y <- 1:2
> z <- x + y
Warning message:
longer object length
is not a multiple of shorter object length in: x + y
> z
[1] 2 4 4
```

Ikus daitekeenez, R-k prebentzio-mezu bat igorri du eta ez errore bat; beraz, eragiketa gauzatu du. Bektore bateko elementu guztiei balio bat gehitu (edo biderkatu) nahi badiegu:

```
> x <- 1:4
> a <- 10
> z <- a * x
> z
[1] 10 20 30 40
```

R-n eskuragarri ditugun funtzioak hainbeste dira, ezen ezinezkoa zaigun guztiak zerrenda batean aurkeztea. Funtzio matematiko sinple guztiak aurki daitezke (`log`, `exp`, `log10`, `log2`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `abs`, `sqrt`, . . .), funtzio bereziak (`gamma`, `digamma`, `beta`, `besselI`, . . .), eta estatistikan oso erabilgarriak diren hainbat funtzio. Funtzio horietariko batzuk hurrengo taulan zehazten ditugu:

`sum(x)`	`x`-ren elementuen batura
`prod(x)`	`x`-ren elementuen biderketa
`max(x)`	`x` objektuaren balio maximoa
`min(x)`	`x` objektuaren balio minimoa
`which.max(x)`	`x`-ren elementu maximoaren indizea itzultzen du
`which.min(x)`	`x`-ren elementu minimoaren indizea itzultzen du
`range(x)`	`x`-ren heina edo `c(min(x), max(x))`
`length(x)`	`x`-ren elementu-kopurua
`mean(x)`	`x`-ren elementuen arteko batezbestekoa
`median(x)`	`x`-ren elementuen arteko mediana
`var(x)` o `cov(x)`	`x`-ren elementuen bariantza (*n-1* erabiliz kalkulatzen da); `x` matrize edo datu-esparru bat bada, bariantza-kobariantza matrizea kalkulatzen da
`cor(x)`	`x`-ren korrelazio-matrizea, `x` matrize edo datu-esparrua bada (1 `x` bektorea bada)
`var(x, y)` o `cov(x, y)`	`x` eta `y`-ren arteko kobariantza, edo `x` eta `y`-ren zutabeen arteko kobariantza `x` eta `y` matrizeak edo datu-esparruak badira
`cor(x, y)`	`x` eta `y`-ren arteko korrelazio lineala, edo korrelazio-matrizea `x` eta `y` matrizeak edo datu-esparruak badira

Funtzio horiek guztiek balio bakarra itzultzen dute (1 luzerako bektore bat). Badira salbuespenak: `range()` funtzioak 2 luzerako bektore bat itzultzen du, eta `var()`, `cov()` eta `cor()` funtzioek, berriz, matrizeak itzul ditzakete. Hurrengo funtzioek bektore konplexuagoak itzul ditzakete:

`round(x, n)`	`x`-ren elementuak `n` hamartarreko zenbakietara biribiltzen ditu
`rev(x)`	`x`-ren elementuak atzekoz aurrera jartzen ditu
`sort(x)`	`x`-ren elementuak goranzko ordenan jartzen ditu; beheranzko ordenan jartzeko: `rev(sort(x))`
`rank(x)`	`x`-ren elementuak lerrokatzen ditu
`log(x, base)`	"base" oinarriko `x`-ren logaritmoa kalkulatzen du
`scale(x)`	`x` matrize bat bada, datuak zentratu eta gutxitu egiten ditu; zentratzea bakarrik nahi bada `sacale=FALSE` erabili, gutxitzeko soilik `center=FALSE` (defektuz, `center=TRUE`, `sacale=TRUE`)
`pmin(x,y,...)`	i. elementua `x[ i]` , `y[ i]` , … elementuen arteko minimoa duen bektore bat
`pmax(x,y,...)`	aurrekoa bezalakoa baina maximoarekin
`cumsum(x)`	i. elementua `x[ 1]` -etik `x[ i]` -rako elementuen batura duen bektore bat
`cumprod(x)`	aurrekoa bezalakoa baina biderkadurarentzat
`cummin(x)`	aurrekoa bezalakoa baina minimoarentzat
`cummax(x)`	arrekoa bezalakoa baina maximoarentzat
`match(x, y)`	`x` bektorearen luzera bera duen bektore bat itzultzen du; bertan, `x`-n dauden elementuetatik `y`-n ere agertzen direnak egongo dira (`NA` `y`-n ez badago)
`which(x == a)`	`x`-ren indizeek osaturiko bektore bat itzultzen du eragiketaren emaitza `TRUE` bada (adibide honetan, `x[ i] == a` egiten duten `i`-ren balioak). Funtzio honen argumentuak aldagai logiko bat izan behar du
`choose(n, k)`	*n* errepikapenetan *k* gertaeren konbinazio guztiak kalkulatzen ditu = $n!/[(n-k)!k!]$
`na.omit(x)`	`NA` balioa duten datuak ezabatzen ditu (`x` matrize edo datu-esparru bat bada, errenkada osoa ezabatzen du)
`na.fail(x)`	errore-mezu bat igortzen du `x`-k gutxienez `NA` bat badu
`unique(x)`	`x` bektore edo datu-esparru bat bada, antzeko objektu bat itzultzen du, baina errepikatzen ziren elementuak ezabatuta
`table(x)`	`x`-ren balio ezberdinen kopuruaz osaturiko taula bat itzultzen du (tipikoki osoentzat eta faktoreentzat)
`subset(x, ...)`	Zehazten zaion irizpidearen araberako `x`-ren aukeraketa bat itzultzen du (… oro har, konparazio bat izaten da irizpidea: `x$V1 <10`); `x` datu-esparru bat bada, `select` aukerak zein aldagai mantendu nahi ditugun adieraziko du (- jarriz gero, zein aldagai arbuiatu nahi ditugun)
`sample(x, size)`	`x`-ren `size` elementuak ausaz lagintzen ditu, baina ordezkapenak egin gabe; `replace=TRUE` aukera jarriz gero, laginketa ordezkapenekin egiten du

3.5.8. Kalkuluak matrizeekin

R-k erraztasunak ematen ditu matrizeak manipulatu eta beraiekin eragiketak egiteko. `rbind()` eta `cbind()` funtzioek matrizeak elkartzen dituzte beraien errenkada edo zutabeekiko, hurrenez hurren:

```
> m1 <- matrix(1, nr = 2, nc = 2)
> m2 <- matrix(2, nr = 2, nc = 2)
> rbind(m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
> cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    2
[2,]    1    1    2    2
```

Bi matrizeren biderketa egiteko '`%*%`' eragigaia erabiltzen da. Adibidez, `m1` eta `m2` bi matrize hartuta:

```
> rbind(m1, m2) %*% cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    2    2    4    4
[2,]    2    2    4    4
[3,]    4    4    8    8
[4,]    4    4    8    8
> cbind(m1, m2) %*% rbind(m1, m2)
     [,1] [,2]
[1,]   10   10
[2,]   10   10
```

Matrize baten transposizioa `t` funtzioarekin egiten da; funtzio horrek datu-esparruekin ere funtzionatzen du.

`diag` funtzioa matrize baten diagonala atera edo aldatzeko, edo matrize-diagonal bat sortzeko erabil daiteke:

```
> diag(m1)
[1] 1 1
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
     [,1] [,2]
[1,]   10    1
[2,]    1   10
> diag(3)
     [,1] [,2] [,3]
```

```
[1,]  1    0    0
[2,]  0    1    0
[3,]  0    0    1
> v <- c(10, 20, 30)
> diag(v)
     [,1] [,2] [,3]
[1,] 10    0    0
[2,]  0   20    0
[3,]  0    0   30
> diag(2.1, nr = 3, nc = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,] 2.1  0.0  0.0   0    0
[2,] 0.0  2.1  0.0   0    0
[3,] 0.0  0.0  2.1   0    0
```

R-k matrizeekin kalkuluak egiteko funtzio batzuk ere baditu. Aipatzeko modukoak dira honoko hauek: `solve()` matrize bat alderantziz jartzeko, `qr()` deskonposaketarako, `eigen()` berezko balio eta bektoreak kalkulatzeko eta `svd()` deskonposaketa singularrak egiteko.

4. Grafikoak R-n

R-k mota askotako grafikoak eskaintzen ditu. Ideia bat izatearren, idatz ezazu `demo(graphics)` komandoa. Ezinezkoa zaigu, benetan, R-k grafikoak egiteko orduan eskaintzen dituen aukera guztiak hemen zehaztea. Funtzio grafiko bakoitzak milaka aukera ditu R-n. Horrek izugarrizko malgutasuna ematen du grafikoak sortzeko, eta beste edozein pakete grafikori alde handia ateratzen dio.

Funtzio grafikoen erabilera dokumentu honen hasieran azaldu ditugun trazen nahiko ezberdina da. Konkretuki, funtzio grafiko baten emaitza ezin zaio objektu bati esleitu[11], gailu grafiko batera eramaten baita. Gailu grafiko bat leiho grafiko edo artxibo bat da.

Bi motatako funtzio grafikoak ikus ditzakegu: alde batetik grafiko berriak sortzeko balio duten *goi-mailako funtzio grafikoak*, eta bestetik, jada eginda dauden grafikoei elementu berriak gehitzeko erabiltzen diren *behe-mailako funtzio grafikoak*. Grafikoak defektuz definituta dauden parametro grafiko batzuen arabera sortzen dira. Parametro horiek `par` funtzioarekin alda daitezke.

Lehenik eta behin, grafiko eta gailu grafikoak erabiltzen ikasiko dugu; ondoren, funtzio grafiko batzuk eta beraien parametroak ikusiko ditugu xehetasun handiz. Azkenik, grid eta lattice paketeak azalduko zaizkigu. Horien funtzionamendua R-ko funtzio grafiko 'arruntena' bezalakoa da.

4.1. GRAFIKOEN ERABILERA

4.1.1. Gailu grafiko anitz ireki

Funtzio grafiko bat exekutatzean, R-k leiho berri bat irekitzen du grafikoa erakusteko, aurretik beste gailu bat ireki ez badugu. Gailu grafiko bat funtzio egokia erabilita irekitzen da. Eskuragarri dauden gailu grafikoen mota sistema

11. Badira salbuespenak: `hist()` eta `barplot()` funtzioek zerrenda eta matrize motako emaitzak ere sortzen dituzte.

eragilearen araberakoa da. Leiho grafikoek `x11` izena dute Unix/Linux-en, `windows` Windows-en eta `macintosh` Mac-en. Unix/Linux-en eta Windows-en `x11()` komandoa idatziz leiho grafiko bat ireki daiteke, Windows-en `windows()` komandora jotzen duen alias bat definitzen baita. Ireki nahi den gailu grafikoa artxibo bat bada, artxibo-mota horren araberako funtzioak erabili behar dira: `postscript()`, `pdf()`, `png()`, … Eskura dauden gailu grafikoen zerrenda `?device` komandoarekin lor daiteke.

Irekitako azken gailua gailu aktibo bihurtuko da. Horrek esan nahi du sortzen ditugun grafikoak gailu horretan marraztuko (edo idatziko) direla. `dev.list()` funtzioak irekitako gailuen zerrenda bat erakutsiko digu:

```
> x11(); x11(); pdf()
> dev.list()
X11 X11 pdf
  2   3   4
```

Zenbakiek gailuen zenbakiari egiten diote erreferentzia. Zenbaki horiek gailu aktiboa aldatzeko erabil daitezke. Gailu aktiboa zein den jakiteko:

```
> dev.cur()
pdf
  4
```

eta gailu aktiboa aldatzeko:

```
> dev.set(3)
X11
  3
```

Merezi du Windows-erako eginiko R-ren bertsioaren bi ezaugarri aipatzea: 1) `win.metafile` funtzioak Windows-en meta-artxiboa den gailu bat irekitzen du, eta 2) “History” menua aukeratuz gero, leiho grafikoa aktibo dagoenean, sesio batean egindako grafiko guztiak ‘grabatzen’ uzten du (aukera hori, defektuz, ez dago martxan, baina erabiltzaileak aktiba dezake menu horretako “Recording”-en klik eginez).

4.1.2. Grafiko baten antolatzea

`split.screen` funtzioak gailu grafiko aktiboa zatitzea ahalbidetzen du. Adibidez:

```
> split.screen(c(1, 2))
```

idatziz gero, gailua bi zatitan banatuko dugu. Zati bakoitza `screen(1)` edo `screen(2)` idatzita aukera dezakegu; `erase.screen()` funtzioak marraztu dugun azken grafikoa ezabatzen du. Gailu baten zati bat, aldi berean, beste zati

txikiago batzuetan bana daiteke `split.screen()` funtzioa erabiliz. Horrela, egitura konplexuak sor ditzakegu.

Funtzio horiek antzeko beste batzuekin bateraezinak dira (`layout()` eta `coplot()` esaterako) eta ezin dira gailu grafiko anitzekin erabili. Berauen erabilera, adibidez, datuen azterketa grafikora mugatu behar da.

`layout` funtzioak gailu aktiboa zenbait zatitan banatzen du. Grafikoak zati horietan marraztuko dira hurrenez hurren. Funtzio horrek azpileihoen kopurua zehazten duen zenbaki osoen matrize bat du argumentu nagusitzat. Adibidez, gailua lau zati berdinetan banatzeko:

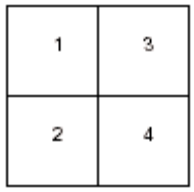
```
> layout(matrix(1:4, 2, 2))
```

Posible da, noski, matrize hori aurrez sortzea. Era horretara, hobeto ikusiko dugu gailua nola zatituko dugun:

```
> mat <- matrix(1:4, 2, 2)
> mat
     [,1] [,2]
[1,]  1    3
[2,]  2    4
> layout(mat)
```

Sortu berri dugun zatiketa ikusteko `layout.show` funtzioa erabil dezakegu. Bertan, azpileihoen kopurua idatzi beharko dugu argumentu moduan. Adibide honetan:

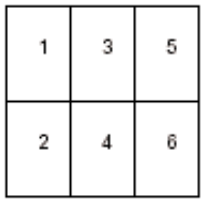
```
>layout.show(4)
```



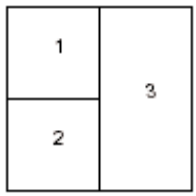
Ondorengo adibideetan `layout()` funtzioak eskaintzen dituen hainbat aukera azaltzen dira.

```
>layout(matrix(1:6, 3, 2))
>layout.show(6)
```

```
>layout(matrix(1:6, 2, 3))
>layout.show(6)
```



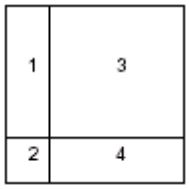

```
>m <- matrix(c(1:3, 3), 2, 2)
>layout(m)
>layout.show(3)
```



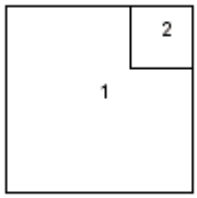
Azaldu zaizkigun adibideetan, inon ere ez dugu `matrix()` funtzioaren `byrow` aukera erabili (errenkadaka irakurri) eta beraz, azpileihoak zutabeen arabera zenbakitzen dira; nahi izanez gero, `matrix(..., byrow=TRUE)` ere idatz daiteke. Era horretara azpileihoak errenkaden arabera zenbakituko dira. Matrizean, zenbakiak nahi den ordenan eman daitezke, adibidez `matrix(c(2, 1, 4, 3), 2, 2)`.

Defektuz, `layout()` funtzioak gailua dimentsio erregularretan banatzen du: hori `widths` eta `heights` aukerekin alda daiteke. Dimentsio horiek era erlatiboan[12] eman behar dira. Adibideak:

```
>m <- matrix(1:4, 2, 2)
>layout(m, widths=c(1, 3),
heights=c(3, 1))
>layout.show(4)
```



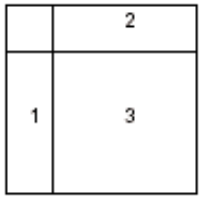

```
>m <- matrix(c(1,1,2,1),2,2)
>layout(m, widths=c(2, 1),
heights=c(1, 2))
>layout.show(2)
```



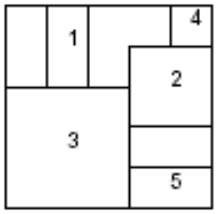
12. Zentimetrotan ere eman daitezke, ikus `? layout`.

Matrizeko zenbakiak zeroak ere izan daitezke. Horrela array konplexuak sortzeko aukera ematen zaigu (baita esoterikoak ere askotan).

```
>m <- matrix(0:3, 2, 2)
>layout(m, c(1, 3), c(1, 3))
>layout.show(3)
```




```
>m <- matrix(scan(), 5, 5)
1: 0 0 3 3 3 1 1 3 3 3
11: 0 0 3 3 3 0 2 2 0 5
21: 4 2 2 0 5
26:
Read 25 items
>layout(m)
>layout.show(5)
```



4.2. FUNTZIO GRAFIKOAK

Ondorengo taulak R-n aurki daitezkeen funtzio grafiko batzuk laburbiltzen ditu.

`plot(x)`	`x`-ren balioak irudikatzen ditu (*y* ardatzean) *x* ardatzean ordenatuta
`plot(x, y)`	bi aldagaiko grafikoa; bertan `x` (*x* ardatzean) eta `y` (*y* ardatzean) irudikatzen dira
`sunflowerplot(x, y)`	`plot()` bezalakoa, baina koordenatu bera duten puntuak lore gisa irudikatzen dira; bertan, puntu-kopurua petalo-kopuruarekin adierazten da
`piechart(x)`	'pie' (tarta) motako grafiko zirkularra
`boxplot(x)`	'box-and-whiskers' motako grafikoa
`stripplot(x)`	`x`-ren balioak zuzen batean irudikatzen dituen grafikoa (lagintamaina txikietan `boxplot()`-en alternatiba gisa erabiltzen da)
`coplot(x~y |z)`	`z`-ren balio edo balio-tarte bakoitzarentzat sortutako `x`-ren eta `y`-ren bi aldagaiko grafikoa
`interaction.plot (f1, f2, y)`	`f1` eta `f2` faktoreak badira, `y`-ren batezbestekoa irudikatzen du (*y* ardatzean) `f1`-en (*x* ardatzean) eta `f2`-ren balioekiko (kurba ezberdinak); `fun` aukerak `y`-ren estatistiko bat aukeratzeko balio du (defektuz, batezbestekoa: `fun=mean`)
`matplot(x,y)`	bi aldagaiko grafikoa; bertan `x`-ren lehen zutabea *vs*. `y`-ren lehen zutabea, `x`-ren bigarren zutabea *vs*. `y`-ren bigarren zutabea, etab.
`dotplot(x)`	`x` datu-esparru bat bada, Cleveland motako puntuzko grafiko bat egiten du (errenkadaz errenkada eta zutabez zutabe pilaturiko grafikoak)
`Fourfoldplot(x)`	zirkulu laurdenak erabiltzen ditu populazio ezberdinentzat bi aldagai dikotomikoren arteko erlazioa irudikatzeko (`x`-k `dim=c(2, 2, k)` duen array bat edo `dim=c(2, 2)` duen matrize bat (`k=1` izanez gero) izan behar du)
`assocplot(x)`	Cohen-en grafiko 'lagunkoi' bat; bertan, bi dimentsioko kontingentzia-taula batean errenkaden eta zutabeen arteko independentzia-desbideraketak erakusten dira
`mosaicplot(x)`	mosaiko erako grafiko bat; bertan kontingentzia-taula bateko erregresio log-lineal baten hondarrak erakusten dira

`pairs(x)`	`x` matrize edo datu-esparru bat baldin bada, `x`-ren zutabe guztien arteko bi aldagaiko grafiko posible guztiak marrazten ditu
`plot.ts(x)`	`x` "`ts`" motako objektua bada, `x` denborarekiko nola aldatzen den erakusten digun grafikoa marraztuko du. `x` aldagai anitzekoa izan daiteke, baina segidek maiztasun eta data berak izan behar dituzte
`ts.plot(x)`	aurrekoa bezalakoa, baina `x` aldagai anitzekoa denean, segidek data ezberdinak eduki ditzakete, ez ordea maiztasunak
`hist(x)`	`x`-ren maiztasunen histograma
`barplot(x)`	`x`-ren balioen histograma
`qqnorm(x)`	`x`-ren kuartilak banaketa normal batetik espero daitekeenarekiko
`qqplot(x, y)`	`y`-ren kuartilak `x`-ren kuartilekiko
`contour(x, y, z)`	ingeraden grafiko bat ('gràfico de contornos' gaztelaniaz; datuak interpolatu egiten dira kurbak marrazteko); bertan, x-k eta y-k bektoreak izan behar dute, eta z-k `dim(z)=c(length(x), length(y))` betetzen duen matrize bat izan behar du (`x` eta `y` alde batera utz daitezke)
`filled.contour (x, y, z)`	aurrekoa bezalakoa, baina ingeraden arteko gainazalak kolorez-tatuak agertzen dira, eta koloreen legenda bat marrazten da
`image(x, y, z)`	aurrekoa bezalakoa baina koloreekin (datuak bere horretan marrazten dira)
`persp(x, y, z)`	aurrekoa bezalakoa baina perspektiban (datuak bere horretan marrazten dira)
`stars(x)`	`x` matrize edo datu-esparru bat bada, segmentudun grafiko edo izar bat marrazten du; bertan, `x`-ren errenkada bakoitza izar batekin irudikatzen da eta zutabeak segmentuen luzerarekin adierazten dira
`symbols(x, y, ...)`	`x`-k eta `y`-k adierazitako koordenatuetan ikurrak marrazten ditu (biribilak, karratuak, laukizuzenak, izarrak, termometroak edo kutxak); horien tamaina, kolorea eta bestelakoak argumentu gehiagorekin zehazten dira
`termplot(mod.obj)`	erregresio-eredu baten (`mod.obj`) eraginen (partzialak) grafikoa

Aurkeztu berri ditugun funtzio horien aukera eta argumentuak R-ren laguntzan topa daitezke. Aukera horietariko batzuk berberak dira funtzio grafiko batzuentzat; hona hemen garrantzitsuenak (defektuzko balioekin):

`add=FALSE` `TRUE` bada, grafikoa aurrez egindakoaren gainean marrazten du (aurrez eginda badago)

`axes=TRUE` `FALSE` bada, ez ditu ardatzak ezta grafikoaren kaxa ere marrazten

`type="p"` grafiko-mota zehazten du; "`p`": puntuzka, "`l`": lerroak, "`b`": lerroz elkarturiko puntuak, "`h`": lerro bertikalak, "`s`": eskailera, datuak lerro bertikalen gaineko zatiarekin irudikatzen dira, "`S`": aurrekoa bezala, baina datuak lerro bertikalen azpialdearekin irudikatzen dira

`xlim=, ylim=` ardatzen goi-bornea eta behe-bornea zehazten ditu; adibidez, `xlim=c(1, 10)` edo `xlim=range(x)`

`xlab=, ylab=` ardatzak izendatzeko; karaktere motako aldagaiak izan behar dira

`main=` izenburua; karaktere motakoa

`sub=` azpititulua (hizki txikiagoz idatzita)

4.3. BEHE-MAILAKO KOMANDO GRAFIKOAK

R-k baditu jada marraztuta dauden grafikoetan eragiten duten hainbat funtzio grafiko: *behe-mailako komando grafikoak*. Hona hemen garrantzitsuenak:

`points(x, y)`	puntuak gehitzen ditu (`type=` aukera erabil daiteke)
`lines(x, y)`	aurrekoa bezalakoa, baina lerroekin
`text(x, y, labels, ...)`	`labels`-ek emaniko testua gehitzen du koordenatuetan (`x`, `y`); adibide tipiko bat: `plot(x, y, type="n"); text(x, y, names)`
`mtext(text, side=3, line=0, ...)`	`text`-ek emaniko testua gehitzen du `side`-k zehazten duen lekuan (ikus `axis()` beherago); `line`-k grafiko-eremuaren lerroa zehazten du
`segments(x0, y0, x1, y1)`	(`x0`, `y0`) puntutik (`x1`, `y1`) puntura lerro bat marrazten du
`arrows(x0, y0, x1, y1, angle= 30, code=2)`	aurrekoa bezalakoa, baina geziak marrazten ditu (`x0`, `y0`) puntutik (`x1`, `y1`) puntura `code=2` bada, (`x1`, `y1`) puntutik (`x0`, `y0`) puntura `code=1` bada, edo bietatik `code=3` bada; `angle`-k geziaren buruak izan behar duen angelua zehazten du
`abline(a,b)`	`a` koordenatutik irten eta `b` malda duen lerro bat marrazten du
`abline(h=y)`	`y` ordenatuan lerro horizontal ba marrazten du
`abline(v=x)`	`x` abzisan lerro bertikal bat marrazten du
`abline(lm.obj)`	`lm.obj`-ek emaniko erregresio-lerroa marrazten du (ikus 5. atala)
`rect(x1, y1, x2, y2)`	laukizuzen bat marrazten du; bertan, ezker, eskuin, goi eta behe aldeak `x1`, `x2`, `x3` eta `x4` zenbakiek zehazten dituzte hurrenez hurren
`polygon(x, y)`	`x`-k eta `y`-k zehaztutako erpinak dituen poligono bat marrazten du
`legend(x, y, legend)`	(`x`, `y`) puntuan `legend`-ek zehaztutako ikurrak dituen legenda gehitzen du
`title()`	izenburu bat gehitzen du, eta aukeran azpititulu bat ere bai
`axis(side, vect)`	ardatz bat gehitzen du azpialdean (`side=1`), ezkerraldean (`2`), goialdean (`3`), edo eskuinaldean (`4`); `vect`-ek (aukerakoa) ardatz-markagailuak ('tick marks') marraztu behar diren abzisa (edo ordenatua) adierazten du
`rug(x)`	`x` datuak marrazten ditu *x* ardatzean, lerro bertikal txiki eran
`locator(n, type="n", ...)`	erabiltzaileak grafikoaren gainean `n` klik egin ondoren, klik horien (*x*, *y*) koordenatuak itzultzen ditu; hautazko parametro grafikoekiko (`...`) ikurrak (`type="p"`) edo lerroak (`type="l"`) ere marrazten ditu; defektuz ez da ezer marrazten (`type="n"`)
`identify(x, ...)`	`locator()`-en antzekoa, baina kasu honetan klik egin den puntutik gertuen aurkitzen den `x`-ren balioa grafikoan bertan inprimatzen du (edo nahi izanez gero `labels=` aukeran zehaztutako legenda batena); izenekin erlazionatuta dauden puntuak grafikoan identifikatzeko oso egokia

Ohartu espresio matematikoak grafikoan bertan idatz daitezkeela `text(x, y, expression(...))`; bertan `expression` funtzioak bere argumentua espresio matematiko bilakatzen du. Adibidez,

```
> text(x, y, expression(p == over(1, 1+e^-(beta*x+alpha))))
```

idatziz gero, espresioa grafikoaren (x, y) puntuan era honetara ikusiko da:

$$p = \frac{1}{1 + e^{-(\beta X + \alpha)}}$$

Aldagai bat jarri nahi bada espresio batean, `substitute` eta `as.expression` funtzioak erabil daitezke; adibidez, R^2-ren balioa jartzeko (aurrez kalkulatu eta `Rsquared` objektuan gorde da):

```
> text(x, y, as.expression(substitute(R^2==r, list(r=Rsquared))))
```

grafikoan, (x, y) puntuan honakoa ikusiko da:

$$R^2 = 0{,}9856298$$

Hiru hamartar soilik ikusteko, kodea honako eran alda dezakegu:

```
> text(x, y, as.expression(substitute(R^2==r,
+           list(r=round(Rsquared, 3)))))
```

eta honela ikusiko da:

$$R^2 = 0{,}986$$

Azkenik, R etzanez idazteko:

```
> text(x, y, as.expression(substitute(italic(R)^2==r,
+           list(r=round(Rsquared, 3)))))
```

$$R^2 = 0{,}986$$

4.4. PARAMETRO GRAFIKOAK

Behe-mailako komando grafikoak erabiltzeaz gain, grafikoen aurkezpena hobe daiteke bestelako parametro grafikoak erabilita. Horiek, funtzio grafikoen aukera bezala (ez du guztientzat balio), edo `par` funtzioarekin parametro grafikoak aldioro aldatuz erabil daitezke; hau da, hurrengo grafikoak erabiltzaileak zehaztu dituen parametroen arabera marraztuko dira. Adibidez,

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

idatziz gero, hurrengo grafiko guztiek atzealde horia izango dute. 68 parametro grafiko ezberdin daude R-n eta batzuek oso antzeko funtzioak betetzen dituzte. Parametro grafiko guztien zerrenda osoa `?par` idatziz ikus daiteke; honako taulan gehien erabiltzen direnak agertzen dira bakarrik.

adj	testuaren justifikazioa kontrolatzen du (0 ezkerrera justifikatzeko, 0.5 zentratzeko, 1 eskuinera justifikatzeko)
bg	atzealdearen kolorea zehazten du (ad.: bg="red", bg="blue", ... Aukeran dauden 657 koloreen zerrenda colors() aukerarekin ikus daiteke)
bty	grafikoaren inguruan marrazten den kaxa-mota kontrolatzen du: "o", "l", "7", "c" edo "]" (kaxak egokitu zaion karakterearen itxura izango du); bty="n" bada, ez da kaxarik marrazten
cex	testu eta ikurrek defektuzko balioarekiko duten tamaina kontrolatzen duen balio bat da; hurrengo parametroek kontrol bera dute ardatzetako zenbakietarako: cex.axis, ardatzetako izenburuak, cex.lab, izenburu nagusia, cex.main, eta azpitituluak, cex.sub
col	ikurren kolorea kontrolatzen du; cex-en bezala, hauexek dira: col.axis, col.lab, col.main, eta col.sub
font	testuaren estiloa kontrolatzen duen oso bat (1: normal, 2: etzana, 3: letra lodia, 4: etzana eta lodia); cex-ek beste aukera dauzkagu orain ere: font.axis, font.lab, font.main eta font.sub
las	karaktereek ardatzetan duten orientazioa kontrolatzen duen osoa (0: ardatzekiko paralelo, 1: horizontal, 2: ardatzekiko elkarzut, 3: bertikal)
lty	lerro-mota kontrolatzen duen osoko edo karaktere bat; (1: solidoa, 2: hautsia, 3: puntuzkatua, 4: lerro-puntua, 5: lerro luze-motza, 6: bi lerro motz), edo gehienez 8 karaktere dituen ("0" eta "9" artean) eta marraztu diren elementuen eta zuriuneen luzera puntutan edo pixeletan txandaka zehazten duen sekuentzia bat; adibidez lty="44" eta lty=2 idazteak eragin bera izango du
lwd	lerroen zabalera kontrolatzen duen zenbaki bat
mar	4 zenbakizko balioak dituen bektore bat da, eta bere lana ardatzen arteko espazioa eta grafikoaren ertzak kontrolatzea da honako forman: c(azpialdea, ezkerraldea, goialdea, eskuinaldea); defektuzko balioak c(5.1, 4.1, 4.1, 2.1)
mfcol	c(nr, nc) formako bektore bat da eta grafikoen leihoa nr errenkada eta nc zutabe dituen matrize batean zatitzeko erabiltzen da; grafikoak zutabeetan marrazten dira (ikus 4.1.2. atala)
mfrow	aurrekoa bezala, baina grafikoak errenkadetan marrazten dira (ikus 4.1.2. atala)
pch	ikur-mota kontrolatzen du, bai 1 eta 25 bitartean dagoen oso bat, baita "" artean doan karaktere bat izanda ere (2. irudia)
ps	testuen eta ikurren tamaina (puntutan) kontrolatzen duen oso bat
pty	grafikoa marraztu behar den eremua zehazten duen karaktere bat, "s": karratua, "m": maximoa
tck	ardatz-markagailuen luzera grafikoaren altuera edo zabaleraren zatiki eran zehazten duen balio bat; tck=1 bada, saretxo bat marrazten da
tcl	ardatz-markagailuen luzera testu-lerro baten zatiki eran zehazten duen balio bat (defektuz tcl=0.5)
xaxt	xaxt="n" bada, *x* ardatza kokatzen da, baina ez da erakusten (axis(side=1, ...)-rekin batera erabiltzeko aproposa)
yaxt	yaxt="n" bada, *y* ardatza kokatzen da, baina ez da erakusten (axis(side=2, ...)-rekin batera erabiltzeko aproposa)

4.5. ADIBIDE PRAKTIKO BAT

R-ren funtzionalitate grafiko batzuk hobeto ikusteko, har dezagun ausaz sorturiko 10 koordenatu-pare dituen bi aldagaiko grafiko bat. Balio horiek sortzeko hau egin behar da:

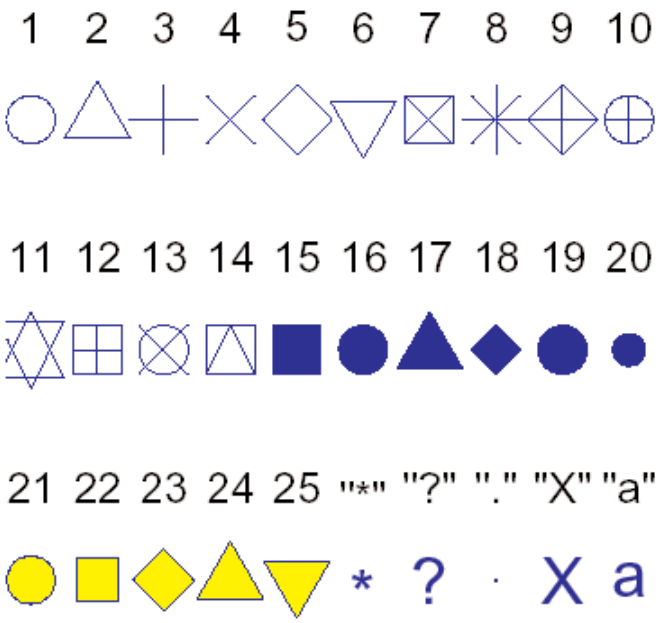
```
> x <- rnorm(10)
> y <- rnorm(10)
```

Nahi dugun grafikoa lortzeko `plot()` erabiliko dugu; komando hau idaztea nahikoa da:

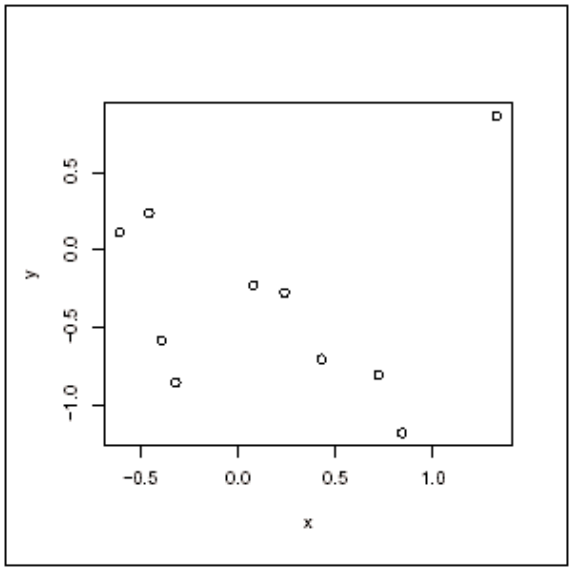
```
> plot(x, y)
```

Horrela, grafikoa gailu grafiko aktiboan ikus ahal izango dugu. Emaitza 3. irudian ikus daiteke. Defektuz, R-k grafikoak era "burutsu" batean marrazten ditu: ardatz-markagailuen arteko espazioa, ardatzetan jarri beharreko etiketen kokagunea, etab. grafikoa ahalik eta irakurterrazen izan dadin kalkulatzen ditu R-k.

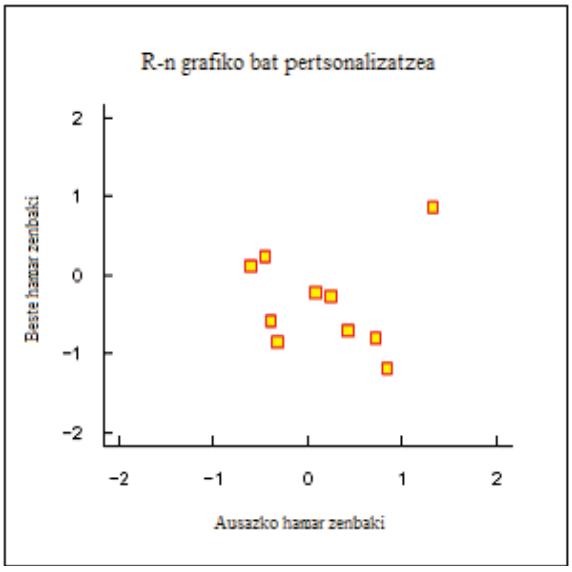
Hala ere, erabiltzaileak grafikoaren aurkezpena alda dezake, esaterako, aurrez ezarritako estilo batera egokitzeko edo hitzaldi baterako ukitu pertsonal bat emateko asmoz.



2. irudia. R-n dauden ikur grafikoak (pch=1:25). Koloreak lortzeko col="blue", bg="yellow" aukerak erabili dira; bigarren aukerak 21-25 ikurrentzat bakarrik du eragina. Edozein karaktere erabil daiteke (`pch="*"`, `"?"`, `"."`, …).



3. irudia. `plot` **funtzioa inongo aukerarik erabili gabe.**



4. irudia. `plot` **funtzioa aukerak erabilita.**

Grafiko bat aldatzeko erarik sinpleena defektuzko balioak aldatzea da aukerak erabiliz. Gure adibidean, irudia asko alda dezakegu honela:

```
plot(x, y, xlab="Ausazko hamar zenbaki", ylab="Beste hamar
      zenbaki", xlim=c(-2, 2), ylim=c(-2, 2), pch=22,
      col="red",
      bg="yellow", bty="l", tcl=0.4,
      main="R-n grafiko bat pertsonalizatzea", las=1,
      cex=1.5)
```

Emaitza 4. irudian ikus daiteke. Erabili ditugun aukerak banan-banan aztertuko ditugu. Lehendabizi, `xlab`-ek eta `ylab`-ek ardatzen izenburuak aldatzen dituzte. Defektuz, izenburuok, aldagaien izenak dira. `xlim`-en eta `ylim`-en bitartez bi ardatzen mugak[13] definitzen ditugu. `pch` parametro grafikoa, aukera bat balitz bezala erabiltzen dugu: `pch=22`-k kolore ezberdindun ingurune eta atzealdea dituen lauki bat zehazten du. Koloreok `col`-ek eta `bg`-k zehazten dituzte hurrenez hurren. Parametro grafikoen taulan azaldu dugu `bty`, `tcl`, `las` eta `cex` parametroek duten eragina. Azkenik, `main` erabilita izenburu bat idatzi dugu.

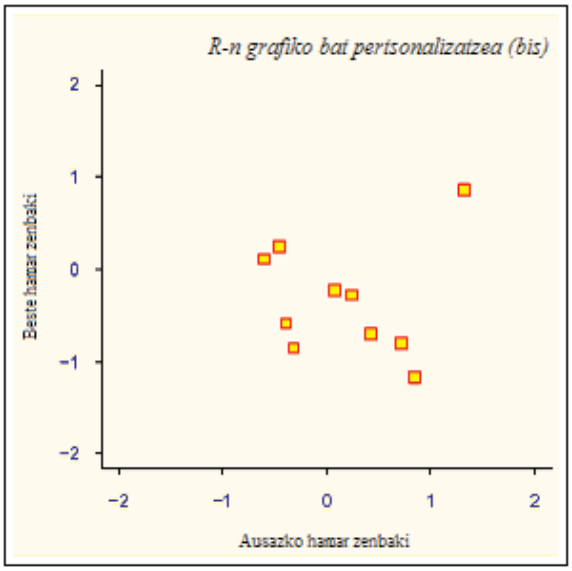
Parametro grafikoek eta behe-mailako funtzio grafikoek grafiko baten itxura oraindik ere gehiago aldatzen uzten digute. Aurretik ikusi dugun bezala, parametro grafiko batzuk ezin dira argumentutzat erabili `plot` eta antzeko funtzioekin. Parametro horietariko batzuk aldatuko ditugu `par()` funtzioarekin, eta ondorioz, komando batzuk idatzi beharko ditugu. Parametro grafikoak aldatzen direnean, beren aurreko balioak gordetzea komeni da ondoren berriro ere balio horiek ezartzeko. Hona hemen 5. irudia lortzeko erabili ditugun komandoak:

```
opar <- par()
par(bg="lightyellow", col.axis="blue", mar=c(4, 4, 2.5,
0.25))
plot(x, y, xlab="Ausazko hamar zenbaki", ylab="Beste hamar
      zenbaki", xlim=c(-2, 2), ylim=c(-2, 2), pch=22,
      col="red", bg="yellow", bty="l", tcl=-.25, las=1,
      cex=1.5)
title("R-n grafiko bat pertsonalizatzea (bis)", font.main=3,
      adj=1)
par(opar)
```

Azter ditzagun komando horiek izan duten eragina. Lehenik eta behin, defektuzko parametro grafikoak `opar` izeneko zerrendan kopiatu dira. Hiru parametro aldatuko ditugu: `bg` atzealdearen kolorea, `col.axis` ardatzetako zenbakien kolorea, eta `mar` grafikoaren ertzetako marjinen tamaina. Grafikoa 4. irudian marraztu zen oso antzera marrazten da oraingo honetan ere. Marjinen aldaketak

13. Defektuz, R-k % 4 gehitzen die ardatz bakoitzaren mugei. Portaera hori xaxz="i" eta yaxs="i" parametro grafikoak aldatuz alda daiteke (`plot()` funtzioari aukera gisa pasa diezazkiokegu).

grafiko-eremuaren inguruko espazioa erabilgarri bihurtzen du. Izenburua behe-mailako `title` funtzioarekin gehitu da. Era horretara, parametro batzuk gehi daitezke argumentu gisa, grafikoaren beste ezaugarriak aldi berean aldatu gabe. Azkenik, hasierako parametro grafikoak azken komandoarekin berrezartzen dira.

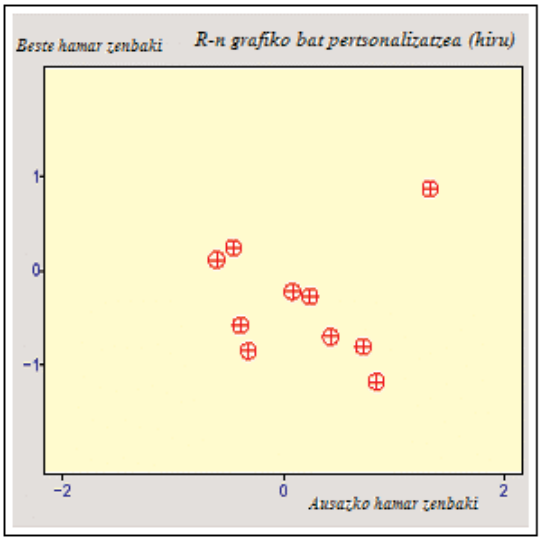


5. irudia. `plot, par` **eta** `title` **funtzioak.**

Eta orain, erabateko kontrola! 5. irudian, R-k oraindik ere, besteak beste, ardatz-markagailuen kopurua, edo izenburuaren eta grafiko-eremuaren arteko distantzia kontrolatzen ditu. Orain ikusiko dugu nola kontrola dezakegun guztiz grafiko baten aurkezpena. Erabiliko dugun estrategia hauxe da: lehenik eta behin, grafiko huts bat marraztuko dugu `plot(..., type="n")` komandoarekin, eta ondoren gehituko ditugu puntuak, ardatzak, etiketak, etab. behe-mailako funtzio grafikoekin. Gauza berri batzuk sartuko ditugu, hala nola grafiko-eremuaren kolorea aldatzea. Komandoak segidan daude eta emaitza 6. irudian ikus daiteke.

```
opar <- par()
par(bg="lightgray", mar=c(2.5, 1.5, 2.5, 0.25))
plot(x, y, type="n", xlab="", ylab="", xlim=c(-2, 2),
ylim=c(-2, 2), xaxt="n", yaxt="n")
rect(-3, -3, 3, 3, col="cornsilk")
points(x, y, pch=10, col="red", cex=2)
axis(side=1, c(-2, 0, 2), tcl=-0.2, labels=FALSE)
axis(side=2, -1:1, tcl=-0.2, labels=FALSE)
title("R-n grafiko bat pertsonalizatzea (hiru)",
font.main=4, adj=1, cex.main=1)
mtext("Ausazko hamar zenbaki", side=1, line=1, at=1, cex=0.9,
      font=3)
```

```
mtext("Beste hamar zenbaki", line=0.5, at=-1.8, cex=0.9,
font=3)
mtext(c(-2, 0, 2), side=1, las=1, at=c(-2, 0, 2), line=0.3,
col="blue", cex=0.9)
mtext(-1:1, side=2, las=1, at=-1:1, line=0.2, col="blue",
cex=0.9)
par(opar)
```



6. irudia. "Eskuz" egindako grafiko bat.

Lehen egin dugun bezala, defektuzko parametro grafikoak gorde egin ditugu eta atzealdearen kolorea zein marjinak aldatu egin ditugu. Grafikoa marrazteko hainbat aukera erabili ditugu: `type="n"` puntuak ez kokatzeko, `xlab=""` eta `ylab=""` ardatzetan izenbururik ez idazteko eta `xaxt="n"` eta `yaxt="n"` ardatzak ez marrazteko. Horrela, grafikoaren eremuaren inguruan kaxa bat soilik marrazten da eta ardatzen mugak definitzen dira `xlim` eta `ylim`-ekiko. `axes=FALSE` aukera ere erabil genezakeen, baina kasu horretan ez ardatzak, ezta kaxa ere ez ziratekeen marraztuko.

Grafiko-eremuan elementuak gehitzeko behe-mailako funtzioak behar ditugu. Puntuak gehitu aurretik grafikoaren eremu barruko kolorea aldatu dugu `rect()`-ekin: laukizuzenaren tamaina beti izango da grafiko-eremua baino dezente handiagoa.

Puntuak marrazteko `points()` erabiltzen da. Kasu honetan ikur berri bat erabili dugu. Ardatzak `axis()`-ekin gehitu dira: bigarren argumentu gisa pasa dugun bektoreak ardatz-markagailuen koordenatuak zehazten ditu. `labels=FALSE` aukerak markagailuetan oharrik ez dela idatzi behar adierazten du. Aukera horrek

karaktere motako bektore bat ere onartzen du, `labels=c("A", "B", "C")` esaterako.

Azkenik, izenburua jartzeko `title()` funtzioaz baliatzen gara. Ohartu letramota pixka bat aldatu dugula. Ardatzetako oharrak `mtext()`-ekin (*marjinetako testuen*) idazten dira. Funtzio horren lehen argumentua idatzi beharreko testua zein den zehazten duen karaktere motako bektore bat da. `line` aukerak grafikoeremuaren distantzia adierazten du (defektuz, `line=0`), eta `at`-k koordenatua. `mtext()`-i egiten zaion bigarren deiean `side`-n defektuzko balioa erabiltzen da (`3`). `mtext()`-i egindako hurrengo bi deietan, karaktere bihurtuko den zenbakizko bektore bat pasatzen zaio lehen argumentu gisa.

4.6. grid *ETA* lattice *PAKETEAK*

grid eta lattice paketeak, R-n egindako S-PLUSeko Trellis-en grafikoen inplementazioak dira. Trellis aldagai anitzeko datuak bistaratzeko metodo bat da eta aldagaien arteko erlazio eta elkarrekintzak aztertzeko oso erabilgarria suertatzen da[14].

lattice-ren (eta Trellis-en) atzean dagoen ideia baldintzazko grafiko anitzena da: bi aldagaiko grafiko bat hainbat grafikotan banatzen da hirugarren aldagai batekiko. `coplot` funtzioak antzeko zerbait egiten du, baina grid-ek, malguagoa izateaz gain, funtzionalitate askoz gehiago aurkezten ditu.

grid edo lattice paketeekin eratutako grafikoak ezin dira konbinatu edo nahastu aurretik ikusi ditugun funtzioek egindakoekin, pakete berri horiek modu grafiko berri bat erabiltzen baitute[15]. Modu berri horrek bere parametro grafikoen sistema berezia dauka, guk orain arte ikusi dugunaren guztiz ezberdina. Hala ere, posible da bi modu grafikook sesio eta gailu grafiko berean erabiltzea.

Ikuspuntu praktiko batetik begiratuta, grid-ek modu grafikoan behar diren funtzio guztiak ditu, eta lattice-k, berriz, gehien erabiltzen diren funtzioak.

lattice-n aurki ditzakegun funtzio gehienek formula bat hartzen dute argumentu nagusitzat; adibidez, `y ~x`[16]. `y ~x | z` formulak adierazten du `y`-ren `x`-rekiko grafikoa `z`-ren balioekiko marraztutako grafiko anitzen bidez marraztuko dela.

Hurrengo taulan laburbildu ditugu lattice paketean dauden funtzio garrantzitsuenak. Argumentu gisa emandako formula erabili ohi dena da, baina funtzio horiek guztiek baldintzazko formulak (`y ~x | z`) onartzen dituzte argumentu

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

15. Modu grafiko berri horrek oinarrizko paketearen moduak aurkezten dituen hainbat ahulgune gainditzen ditu, grafikoekiko elkarrekintzarik eza, esaterako.

16. `plot()`-ek ere formulak onartzen ditu argumentu nagusi gisa: `x` eta `y` luzera bereko bektoreak badira, `plot(x ~y)` eta `plot(x, y)` grafiko bera dira.

nagusi gisa; beherago jarritako adibideetan ikusiko den bezala, azken kasu horretan *z*-rekiko grafiko anitz marraztuko dira.

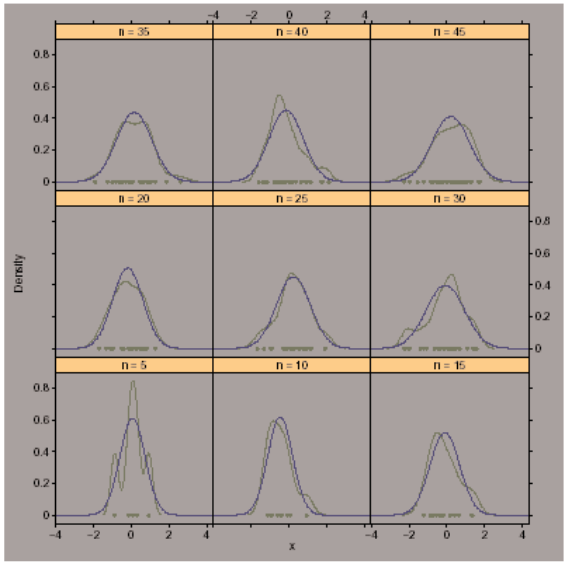
`barchart(y ~x)`	*y* -ren balioen histograma *x*-ren balioekiko
`bwplot(y ~x)`	"box-and-whiskers" motako grafikoa
`densityplot(~x)`	dentsitate-funtzioen grafikoa
`dotplot(y ~x)`	Cleveland motako puntuzko grafikoa (errenkadaz errenkada eta zutabez zutabe pilatutako grafikoak)
`histogram(~x)`	*x*-ren maiztasunen histograma
`qqmath(~x)`	*x*-ren kuartilak banaketa teoriko batetik espero daitekeenarekin ados
`stripplot(y ~x)`	dimentsio bakarreko grafikoa, *x* zenbakizkoa izan behar da, *y* faktorea izan daiteke
`qq(y ~x)`	bi banaketa konparatzeko kuartilak; *x* zenbakizkoa izan behar da, *y* zenbakizkoa, karaktere motakoa edo faktorea izan daiteke, baina gutxienez bi maila izan behar ditu
`xyplot(y ~x)`	bi aldagaiko grafikoak (funtzionalitate askorekin)
`levelplot(z ~x*y)`	*x*-k eta *y*-k emandako koordenatuetan marraztutako *z*-ren balioen koloretako grafikoa (*x*, *y* eta *z* luzera berdinekoak izan behar dira)
`splom(~x)`	bi aldagaiko grafikoen matrizea
`parallel(~x)`	koordenatu paraleloen grafikoa

lattice paketeko funtzio batzuek base paketeko funtzio grafiko batzuen izen berak dituzte. Azken horiek "ezkutatu" egiten dira lattice memorian kargatzen denean.

Ikus ditzagun, lattice-ren ezaugarri batzuk agerian jartzeko asmoz, adibide batzuk. Lehenik eta behin, pakete horretako funtzioak erabili ahal izateko memorian kargatu behar da `library(lattice)` komandoarekin.

Dentsitate-funtzioen grafikoekin has gaitezen. Funtzio horiek egiteko nahikoa da `densityplot(~x)` komandoa erabiltzea. Emaitza, dentsitate-funtzio enpirikoa erakusten duen kurba bat izango da; bertan, puntuek, *x* ardatzean egindako behakuntzak adierazten dizkigute (`rug()`-en antzekoa). Gure adibidea pixka bat konplexuagoa izango da, grafiko bakoitzean dentsitate-kurba enpirikoei banaketa normal batetik espero daitezkeen kurbak gainezarriko baitizkiegu. `panel` argumentua erabiltzea nahitaezko bilakatzen da grafiko bakoitzean zer marraztu behar den definitzen baitu. Komandoak honako hauek dira:

```
n <- seq(5, 45, 5)
x <- rnorm(sum(n))
y <- factor(rep(n, n), labels=paste("n =", n))
densityplot(~ x | y,
      panel = function(x, ...) {
            panel.densityplot(x, col="DarkOliveGreen", ...)
            panel.mathdensity(dmath=dnorm,
            args=list(mean=mean(x), sd=sd(x)),
            col="darkblue")
      })
```



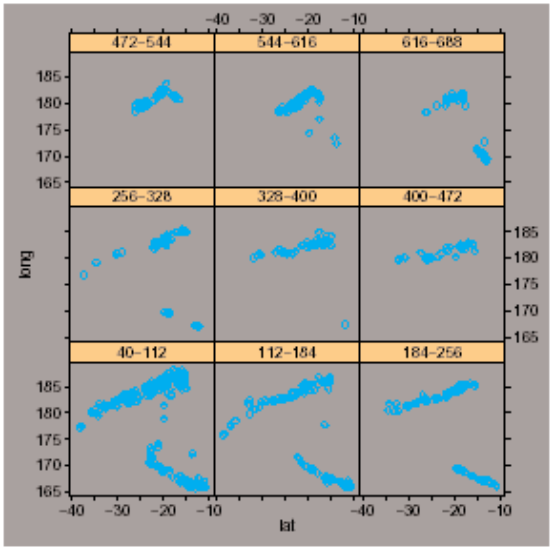
7. irudia. `densityplot` **funtzioa.**

Lehen hiru ilarek banaketa normal batetik hartutako ausazko lagin bat sortzen dute. Lagin hori 5, 10, 15, …, eta 45 tamainako azpilaginetan banatzen da. Ondoren `densityplot` funtzioari deitzen zaio. Horrek azpilagin bakoitzarentzat grafiko bat sortzen du. `panel`-ek funtzio bat hartzen du argumentu gisa. Gure adibidean lattice-n aurrez definitutako bi funtziori dei egiten dien funtzio bat definitu dugu: `panel.densityplot`, dentsitate-funtzio enpirikoa marrazteko, eta `panel.mathdensity`, banaketa normal batean oinarrituz lortuko genukeen dentsitate-funtzio teorikoa marrazteko. `panel`-en ez bada inongo argumenturik zehazten, defektuz dei egiten zaio `panel.densityplot` funtzioari: `densityplot (~x | y)` komandoak 7. irudia sortuko luke, baina kurba urdinik gabe.

Hurrengo adibideetan R-n aukeran dauzkagun datu-multzo batzuk erabiliko ditugu: Fiji uharteen inguruan gertatu diren 1000 gertaera sismikoren kokagunea eta iris edo ostargi-belar lorearen hiru espezieren zenbait neurri.

8. irudian gertaera sismikoen kokagunea ikus dezakegu beraien sakoneraren arabera. Grafiko hori sortzeko behar diren komandoak honako hauek dira:

```
data(quakes)
mini <- min(quakes$depth)
maxi <- max(quakes$depth)
int <- ceiling((maxi - mini)/9)
inf <- seq(mini, maxi, int)
quakes$depth.cat <- factor(floor(((quakes$depth - mini) / int)),
                    labels=paste(inf, inf + int, sep="-"))
xyplot(lat ~ long | depth.cat, data = quakes)
```



8. irudia. `xyplot` **funtzioa "quakes"-en dauden datuekin.**

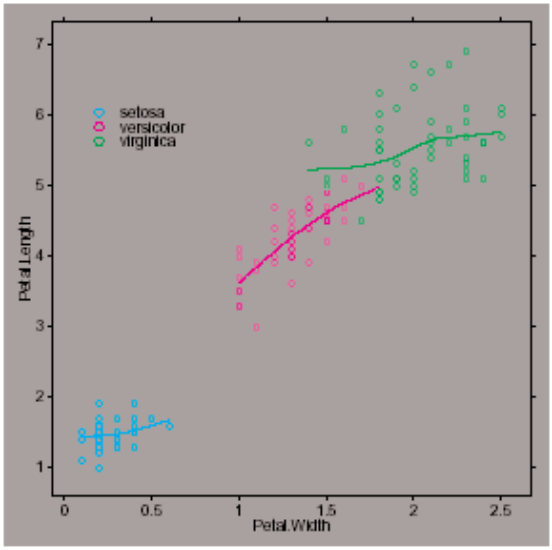
Lehen komandoak `quakes`-en datuak kargatzen ditu memorian. Hurrengo 5 komandoek faktore bat sortzen dute sakonera (`depth` aldagaia) 9 tarte berdinetan zatituz (maila berarekin): faktorearen mailak tarte horien goi- eta behe-mugekin zenbakitzen dira. Aldagaiak definitu ondoren, `xyplot` funtzioari dei egiten diogu formula egokiarekin eta aldagaiak non bilatu behar dituen adierazten dion `data` argumentu batekin[17].

Behin datuak `iris`-en ditugula, espezien arteko gainezarmena nahiko baxua denez, elkarrengandik ezberdin daitezke irudian (9. irudia). Komandoak honakoak dira:

```
data(iris)
xyplot(
      Petal.Length ~ Petal.Width, data = iris,
      groups=Species,
      panel = panel.superpose,
      type = c("p", "smooth"), span=.75,
      key = list(x=0.15, y=0.85,
            points=list(col=trellis.par.get()[["superpose.sym
                  bol"]]$col[1:3], pch = 1),
            text = list(levels(iris$Species)))
)
```

17. `plot()`-ek ezin ditu `data` moduko argumentuak jaso; aldagaien kokagunea esplizituki zehaztu behar da. Adibidez, `plot(quakes$long ~quakes$lat)`.

`xyplot` funtzioari adibide honetan egin zaion deia, besteetan egindakoak baino zertxobait konplexuagoa da eta aukera asko erabiltzen ditu. Guztiak ikusiko ditugu xehetasun handiz. `groups` aukerak, bere izenak adierazten duen moduan, beste aukera batzuek geroago erabiliko dituzten taldeak definitzen ditu. Gorago `panel` aukera ikusi dugu. Aukera hori talde ezberdinak grafikoan nola irudikatu zehazteko erabili dugu: hemen `panel.superpose` aurrez definitutako funtzioa darabilgu taldeak grafiko berean gainezar daitezen. Funtzio horri aukerarik eman ez diogunez, defektuzko koloreak erabiliko ditu taldeak ezberdintzeko. `type` aukerak, `plot()`-en bezala, datuak nola irudikatu zehazten du, baina orain bektore berean hainbat argumentu zehatz daitezke: `"p"` puntuak marrazteko eta `"smooth"` `span`-ek zehaztutako malgutasunez datuetara egokitutako kurba bat marrazteko. `key` aukerak legenda bat gehitzen dio grafikoari; kasu honetan sintaxia nahiko korapilatsua da, baina lattice-ren etorkizuneko bertsioetan R-ko grafiko estandarretan `legend` funtzioak duenaren antzeko sintaxi bat inplementatu nahi da, sinpletasunean irabazteko. `key`-k zerrenda bat hartzen du argumentutzat: `x`-k eta `y`-k legendaren koordenatuak ematen dizkigute (ez badira zehazten marrazki-eremutik at idatziko da); `points`-ek ikur-mota zehazten du (espresioa korapilatsu bihurtzen da ikurra defektuzko definizioetatik hartzen baita); eta `text`-ek legendaren testua ematen digu, gure kasuan, iris espeziearen izena.



9. irudia. `xyplot` **funtzioa "iris"-en datuekin.**

Orain splom funtzioa ikusiko dugu iris-en datu berdinekin. 10. irudia sortzeko ondorengo komandoak erabili ditugu:

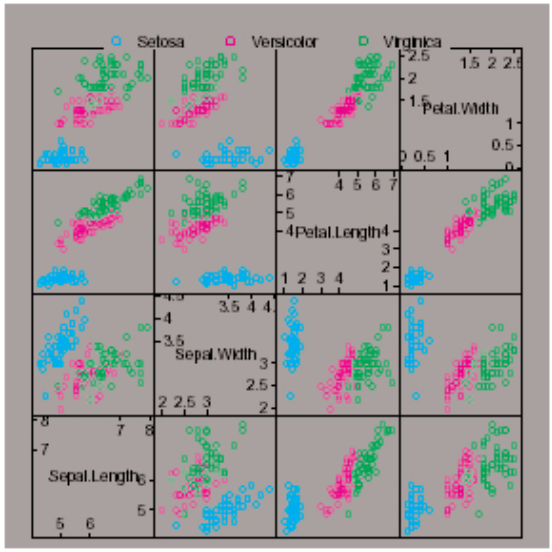
```
splom(
   ~iris[1:4], groups = Species, data = iris, xlab = "",
   panel = panel.superpose,
   key = list(columns = 3,
      points=list(col=trellis.par.get()[["superpose.symbol"]]
           $col[1:3],pch = 1),
      text = list(c("Setosa", "Versicolor", "Virginica")))
)
```

Oraingoan, argumentu nagusia matrize bat da (iris-en lehen lau zutabeak). Emaitza matrizearen zutabeen artean sor daitezkeen bi aldagaiko grafiko guztiak dira, pairs funtzio estandarrean bezala. splom-ek, defektuz, "Scatter Plot Matrix" testua gehitzen du *x* ardatzaren azpian: hori ekiditeko, xlab="" aukera darabilgu. Beste aukerak aurreko adibidean aurkeztu ditugunen antzekoak dira, key-n agertzen den columns=3 izan ezik. Aukera horrekin legenda hiru zutabetan aurkez dezakegu.

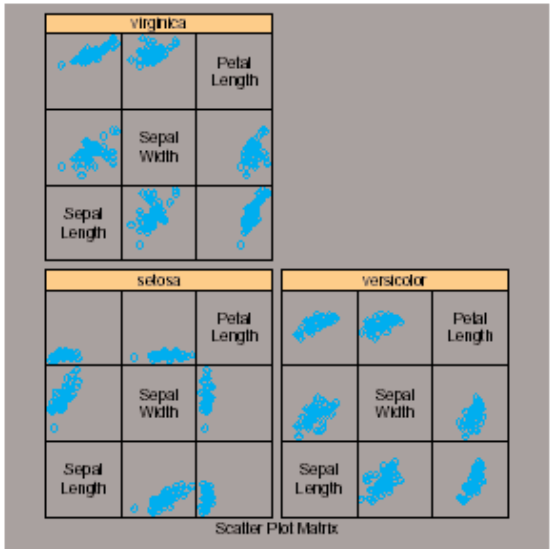
10. irudia pairs() funtzioarekin ere egin zitekeen, baina funtzio horrek ezin ditu 11. irudiko baldintzazko grafikoak egin. Kodea nahiko sinplea da:

```
splom(~iris[1:3] | Species, data = iris, pscales = 0,
   varnames = c("Sepal\nLength", "Sepal\nWidth",
"Petal\nLength"))
```

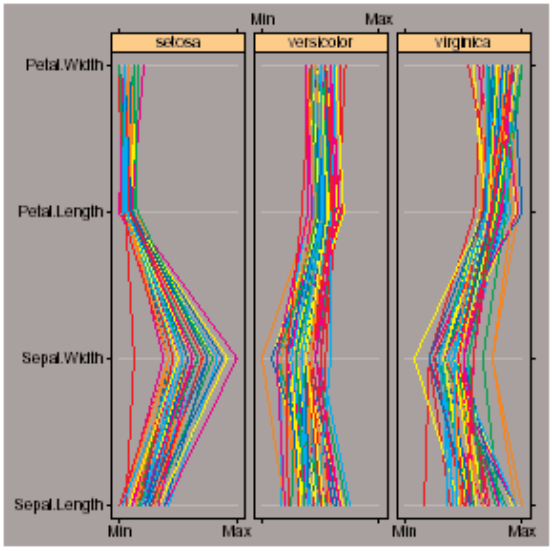
Azpigrafikoak nahiko txikiak direnez, bi aukera gehituko ditugu irudiaren irakurgarritasuna hobetzeko asmoz: pscales=0 ardatz-markagailuak kentzeko (azpigrafiko guztiak eskala berean marrazten dira), eta aldagaien izenak birdefinitu egiten ditugu bi ilaretan erakusteko ("\n" -k ilara berri batera joatea kodetzen du).



10. irudia. `splom` **funtzioa "iris"-en datuekin (1).**



11. irudia. `splom` **funtzioa "iris"-en datuekin (2).**



12. irudia. `parallel` **funtzioa "iris"-en datuekin.**

Azken adibideak koordenatu paraleloen teknika darabil aldagai anitzeko datuak aztertzeko. Aldagaiak ardatz batean jartzen dira (adibidez, *y* ardatzean) eta behatutako balioak beste ardatzean kokatzen dira (estandarizatu egiten dira konparagarriak izan daitezen). Banako bakoitzaren balio ezberdinak lerro batez elkartzen dira. `iris`-en datuekin, 12. irudia ondorengo kodea idatzita lortzen da:

```
parallel(~iris[, 1:4] | Species, data = iris, layout = c(3,
1))
```

5. Analisi estatistikoa R-rekin

Grafikoekin gertatzen zen bezala, ezinezkoa zaigu aztertzea R-k analisi estatistikoak egiteko eskaintzen dituen aukera guztiak. Atal honen helburua ikuspegi orokor bat eskaintzea da irakurleak R-ren ezaugarriak ezagut ditzan era guztietako analisiak egiteko orduan.

grid eta lattice paketeetan zeuden funtzioak izan ezik, orain arte ikusi ditugun funtzio guztiak base paketean daude. Datuen analisirako funtzio batzuk base paketean topa ditzakegu, baina R-n eskuragarri dauden metodo estatistiko gehienak paketeetan (`packages`) banatuta daude. Pakete horietariko batzuk base-rekin instalatuta datoz, beste batzuk `recommended` paketean daude estatistikan eskuarki erabiltzen diren metodoak baitarabiltzate, eta azkenik, beste asko `contributed` multzoaren barnean aurki daitezke eta erabiltzaileak berak instalatu behar ditu.

R-n datu-analisirako erabiltzen den metodo orokorra aurkezteko asmoz, base paketea soilik erabiltzen duen adibide sinple bat ikusiko dugu. Ondoren, *formulak* eta *funtzio generikoak* kontzeptuak azalduko ditugu, edozein analisi-motatarako erabilgarriak baitira. Beste paketeei eskainitako gainbegirada azkar batekin amaituko dugu.

5.1. BARIANTZA-ANALISI BATEN ADIBIDE SINPLEA

Hiru dira base paketean aurki ditzakegun funtzio estatistikorik garrantzitsuenak: `lm`, `qlm` eta `aov`, erregresio lineala, eredu lineal orokortuak eta bariantza-analisiak egiteko, hurrenez hurren. `loglin` ere aipatuko dugu eredu log-linealetarako, baina funtzio horrek kontingentzia-taula bat hartzen du argumentutzat formula baten ordez[18]. Bariantza-analisi bat nola egiten den ikusteko har ditzagun R-n datozen datu batzuk: `InsectSprays` (intsektizidak). 6 intsektizida probatu ziren intsektuen kopurua erantzun-aldagai hartuta. Intsektizida bakoitza 12 aldiz probatu zen,

18. MASS paketeak `loglm` funtzioa dauka; funtzio horrek `loglin`-ekin formula-interfaze bat ezartzea ahalbidetzen du.

hots, 72 behakuntza gauzatu ziren. Hemen ez dugu datuen esplorazio grafiko bat egingo, erabilitako intsektizidaren funtzioan erantzun-aldagaiaren bariantza-analisi sinple bat baizik. Datuak memorian `data` funtzioarekin kargatu ondoren, analisia `aov` funtzioarekin egiten da (erantzuna behar bezala itxuraldatu ondoren):

```
> data(InsectSprays)
> aov.spray <- aov(sqrt(count) ~ spray, data = InsectSprays)
```

`aov()`-ren argumentu nagusia (eta beharrezkoa) formula bat da, zeinak ~ ikurraren ezkerraldean erantzuna eta eskuinaldean azalpen-aldagaia zehazten dituen. `data=InsectSprays` aukerak adierazten du aldagaiak `InsectSprays` datu-esparruan bilatu behar direla. Aurkeztu berri dugun sintaxia honako honen baliokidea da:

```
> aov.spray <- aov(sqrt(InsectSprays$count) ~
InsectSprays$spray)
```

edo aldagaien zutabeen kopurua ezagutzen badugu:

```
> aov.spray <- aov(sqrt(InsectSprays[, 1]) ~ InsectSprays[, 2])
```

lehen sintaxia, ordea, garbiagoa da eta nahasketak ekiditen ditu.

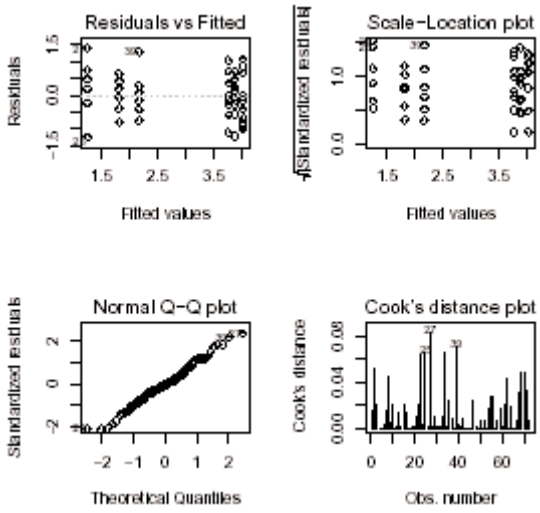
Analisiaren emaitzak ez dira erakusten, `aov.spray` izeneko objektu bati esleitzen baitzaizkio. Emaitzak lortzeko beste funtzio batzuk erabiliko ditugu, esaterako, `print()` analisiaren laburpen bat ikusteko (parametroak batik bat) eta `summary()` xehetasun gehiago ikusteko (proba estatistikoak barne):

```
> aov.spray
Call:
      aov(formula = sqrt(count) ~ spray, data = InsectSprays)

Terms:
                        spray  Residuals
Sum of Squares          88.43787     26.05798
Deg. of Freedom                5           66

Residual standard error: 0.6283453
Estimated effects may be unbalanced
> summary(aov.spray)
            Df     Sum Sq      Mean Sq     F value     Pr(>F)
spray        5     88.438      17.688      44.799 < 2.2e-16
***
Residuals  66    26.058        0.395
—-
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

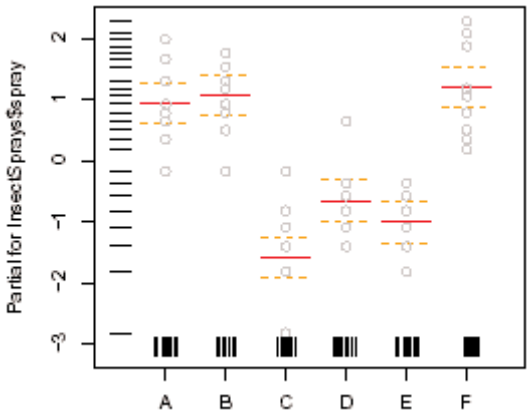
Gogora dezagun objektuaren izena komando gisa idaztea `print(aov.spray)` idaztearen baliokidea dela. Emaitzen irudikapen grafiko bat ikusi nahi badugu `plot()` edo `termplot()` funtzioak erabil ditzakegu. `plot(aov.spray)` idatzi aurretik leiho grafikoa lau zatitan banatuko dugu lau diagnostiko-grafikoak leiho berean marraz ditzan. Komandoak hauek dira:

```
> opar <- par()
> par(mfcol = c(2, 2))
> plot(aov.spray)
> par(opar)
> termplot(aov.spray, se=TRUE, partial.resid=TRUE, rug=TRUE)
```

eta emaitzak 13. eta 14. irudietan dauzkagu.



13. irudia. `aov`**-ren emaitzen irudikapen grafikoa** `plot()` **funtzioarekin**



14. irudia. `aov`**-ren emaitzen irudikapen grafikoa** `termplot()` **funtzioarekin.**

5.2. FORMULAK

Formulen erabilera zeharo garrantzitsua da R-rekin analisi estatistikoak egiteko: erabiltzen den nomenklatura funtzio (ia) guztientzat bera da. Formula bat `y ~eredua` gisa idazten da oro har; bertan, `y` menpeko (edo erantzun) aldagaia eta `eredua` parametro segida bat pasa behar zaion termino-multzo bat da. Termino horiek ikur aritmetikoen bidez irudikatzen dira, baina oso esanahi berezia dute R-n.

`a+b`	`a`-ren eta `b`-ren efektuak
`X`	`X` matrize bat bada, zutabe bakoitzarentzat efektu batukor bat zehazten du; adibidez, `X[ ,1] +X[ ,2] +...+X[ ,ncol(X)]`; zutabe batzuk aukera daitezke zenbakizko indizeekin (adib. `X[ ,2:4]`)
`a:b`	`a`-ren eta `b`-ren arteko elkarrekintza-efektua
`a*b`	efektu batukor eta elkarrekintzazkoa `a` eta `b`-rentzat (`a+b+a:b` espresioaren berdina)
`poly(a, n)`	`n` gradura arteko `a`-ren polinomioak
`^n`	`n` mailara arteko elkarrekintza guztiak zehazten ditu; adib. `(a+b+c)^2` idatzi ala `a+b+c+a:b+a:c+b:c` idaztea gauza bera da
`b%in% a`	`b`-ren efektuak `a`-n habiratuta daude (`a+a:b` edo `a/b` idaztearen baliokidea)
`a-b`	`b`-ren efektua eraisten du, adibidez: `(a+b+c)^2:a-b` eta `a+b+c+a:c+b:c` berdinak dira

`-1`	`y~x-1` jatorriarekiko erregresioa (berdin `y~x+0` edo `0+y~x`-rentzako ere)
`1`	`y~1` idatziz gero, efektu gabeko eredu bat atontzen du (interzeptua bakarrik)
`offset(...)`	efektu bat gehitzen dio ereduari parametroak estimatu gabe (adib. `offset(3*x)`)

Eragile aritmetikoek espresio arruntetan duten esanahiaren oso bestelakoa hartzen dute formuletan. Adibidez, `y~x1+x2` espresioak $y = \beta_1 x_1 + \beta_2 x_2 + \alpha$ eredua definitzen du, eta ez $y = \beta(x_1 + x_2) + \alpha$ (+ eragileak bere betiko esanahia eduki izan balu). Eragiketa aritmetikoak formuletan idazteko `I()` funtzioa erabil dezakegu: `y~I(x1+x2)` formulak $y = \beta(x_1 + x_2) + \alpha$ eredua definitzen du. Antzeko eran, $y = \beta_1 x + \beta_2 x^2 + \alpha$ eredua definitzeko `y ~poly(x, 2)` formula erabiliko dugu (eta ez `y ~x + x^2`).

Bariantza-analisietarako `aov()` funtzioak sintaxi berezia onartzen du ausazko efektuak definitzeko. Adibidez, `y ~a + Error(b)` formulak termino finko (`a`) baten efektu batukorrak eta ausazko bat (`b`) zehazten ditu.

5.3. FUNTZIO GENERIKOAK

Gogora dezagun R-n funtzioek argumentu gisa pasatzen zaizkien objektuen atributuen gainean egiten dutela lan. Analisiaren ondorioz sortzen diren objektuek `clase` izeneko atributu berezi bat izaten dute, analisia egiteko erabili den funtzioaren nolabaiteko "sinadura". Analisiaren emaitzak lortzeko erabiliko diren funtzioek objektuaren klasearen gainean egingo dute lan. Funtzio horiek generiko izena hartzen dute.

Adibidez, analisien emaitzak ateratzeko gehien erabiltzen den funtzioa analisien emaitzak xehetasun guztiz erakusten dituen `summary` da. Argumentu gisa pasatzen zaion objektua "`lm`" (eredu lineala) edo "`aov`" (bariantza-analisia) klasekoa bada, nahiko argi dago erakusten den informazioa ez dela bera izango. Funtzio generikoek analisi-mota guztietarako sintaxi bera edukitzearen abantaila aurkezten dute.

Analisi baten emaitzak gordetzen dituen objektua, oro har, zerrenda bat izaten da eta bere bistaratzea bere klasearen baitan dago. Ikusi dugu jada funtzio baten ekintza argumentu gisa pasatzen zaion objektuaren arabera dagoela. Hori R-ren ezaugarri orokor bat da[19]. Hurrengo taulan analisi baten informazioa lortzeko gehien erabiltzen diren funtzio generikoak agertzen dira. Funtzio horien erabilera tipikoa hau da:

19. R-n 100 funtzio generiko baino gehiago daude.

```
> mod <- lm(y ~ x)
> df.residual(mod)
[1] 8
```

`print`	laburpen motz bat itzultzen du
`summary`	xehetasunez betetako laburpen bat itzultzen du
`df.residual`	askatasun-gradua itzultzen du
`coef`	estimatutako koefizienteak itzultzen ditu (askotan beren errore estandarrekin)
`residuals`	hondarrak itzultzen ditu
`deviance`	desbideratzea itzultzen du
`fitted`	egokitutako balioak itzulzen ditu
`logLik`	probabilitatearen logaritmoa kalkulatzen du eta parametroen kopurua
`AIC`	Akaike-ren informazio-irizpidea edo AIC kalkulatzen du (`logLik`-en arabera)

`aov` edo `lm` funtzioek zerrenda bat itzultzen dute; bertan, elementu bakoitza analisiaren emaitzekin lotuta dagoen balioren bat da. Gure aurreko adibidea hartzen badugu (`InsectSprays`-en datuekin egin dugun bariantza-analisia), `aov()`-k itzulitako objektuaren egitura ikus dezakegu:

```
> str(aov.spray, max.level = -1)
List of 13
 - attr(*, "class")= chr [1:2] "aov" "lm"
```

Egitura hori ikusteko beste era bat objektuaren izenak bistaratzea da:

```
> names(aov.spray)
 [1] "coefficients"     "residuals"       "effects"
 [4] "rank"             "fitted.values"   "assign"
 [7] "qr"               "df.residual"     "contrasts"
 [10] "xlevels"   "call"            "terms"
 [13] "model"
```

Beste edozein zerrendatan bezala, elementuak bertatik atera daitezke:

```
> aov.spray$coefficients
(Intercept)       sprayB      sprayC      sprayD
3.7606784         0.1159530 -2.5158217 -1.5963245
    sprayE  sprayF
-1.9512174  0.2579388
```

`summary()`-k ere zerrenda bat sortzen du; hori, `aov()`-ren kasuan, saioen taula bat da:

```
> str(summary(aov.spray))
List of 1
 $ :Classes anova and 'data.frame': 2 obs. of 5 variables:
  ..$ Df         : num [1:2] 5 66
  ..$ Sum Sq     : num [1:2] 88.4 26.1
  ..$ Mean Sq    : num [1:2] 17.688 0.395
  ..$ F value    : num [1:2] 44.8 NA
```

```
   ..$ Pr(>F)     : num [1:2] 0 NA
  - attr(*, "class")= chr [1:2] "summary.aov" "listof"
> names(summary(aov.spray))
NULL
```

Funtzio generikoek *metodo* izena ere hartzen dute. Eskematikoki, `method.foo` gisa eratzen dira, non, `foo` analisi-funtzioa den. `summary`-ren kasuan, metodo hori darabilten funtzioak ikus ditzakegu:

```
> apropos("^summary")
[1] "summary"                  "summary.aov"
[3] "summary.aovlist"   "summary.connection"
[5] "summary.data.frame"       "summary.default"
[7] "summary.factor"           "summary.glm"
[9] "summary.glm.null"  "summary.infl"
[11] "summary.lm"              "summary.lm.null"
[13] "summary.manova"   "summary.matrix"
[15] "summary.mlm"             "summary.packageStatus"
[17] "summary.POSIXct"  "summary.POSIXlt"
[19] "summary.table"
```

Metodo horrek erregresio lineal baten kasuan nola jokatzen duen ikusita, analisi-bariantzaren kasuarekin konpara dezakegu:

```
> x <- y <- rnorm(5);
> mod <- lm(y ~ x)
> names(mod)
  [1] "coefficients"     "residuals"       "effects"
  [4] "rank"             "fitted.values" "assign"
  [7] "qr"               "df.residual"     "xlevels"
  [10] "call"            "terms"           "model"
> names(summary(mod))
  [1] "call"             "terms"           "residuals"
  [4] "coefficients"     "sigma"           "df"
  [7] "r.squared"  "adj.r.squared" "fstatistic"
  [10] "cov.unscaled"
```

`aov()`, `lm()`, `summary()`... funtzioek zerrendak itzultzen dituzten arren, ez dira arestian ikusi ditugun zerrenda "arruntak" bezala bistaratzen. Egia esan, objektu horien `print` metodoak dira (gogora dezagun objektu baten izena komando gisa idaztea `print()` erabiltzearen baliokidea dela):

```
> apropos("^print")
  [1] "print.pairwise.htest"         "print.power.htest"
  [3] "print"                        "print.anova"
  [5] "print.aov"                    "print.aovlist"
  [7] "print.atomic"                 "print.by"
  [9] "print.coefmat"                "print.connection"
  [11] "print.data.frame"            "print.default"
  [13] "print.density"               "print.difftime"
```

```
[15] "print.dummy.coef"          "print.dummy.coef.list"
[17] "print.factor"              "print.family"
[19] "print.formula"             "print.ftable"
[21] "print.glm"                 "print.glm.null"
[23] "print.hsearch"             "print.htest"
[25] "print.infl"                "print.integrate"
[27] "print.libraryIQR"          "print.listof"
[29] "print.lm"                  "print.lm.null"
[31] "print.logLik"              "print.matrix"
[33] "print.mtable"              "print.noquote"
[35] "print.octmode"             "print.ordered"
[37] "print.packageIQR"          "print.packageStatus"
[39] "print.POSIXct"             "print.POSIXlt"
[41] "print.recordedplot"        "print.rle"
[43] "print.SavedPlots"          "print.simple.list"
[45] "print.socket"              "print.summary.aov"
[47] "print.summary.aovlist"     "print.summary.glm"
[49] "print.summary.glm.null"    "print.summary.lm"
[51] "print.summary.lm.null"     "print.summary.manova"
[53] "print.summary.table"       "print.table"
[55] "print.tables.aov"          "print.terms"
[57] "print.ts"                  "print.xtabs"
```

`print`-en metodo horiek guztiek analisiaren araberako bistaratzeak egiten uzten digute.

Hurrengo taulak analisi baten emaitza diren objektuei analisi gehigarriak egiten dizkieten funtzio generikoak erakusten ditu. Argumentu nagusia aurreko analisiaren emaitza gordetzen duen objektua den arren, askotan, argumentu gehiago ere behar izaten dira (`predict` edo `update`-rentzat adibidez).

`add1`	eredu bati gehi dakizkiokeen termino guztiak probatzen ditu
`drop1`	eredu batetik ezaba daitezkeen termino guztiak probatzen ditu
`step`	AICrekin eredu bat aukeratzen du (`add1` eta `drop1`-i dei egiten die)
`anova`	eredu bat edo gehiagoren bariantza- edo desbideratze-analisien taula bat kalkulatzen du
`predict`	aurrez egokitutako eredu bati gehitzen zazikion datu berrien aurreikusitako balioak kalkulatzen ditu
`update`	datu edo formula berri batekin eredu bat berregokitzen du

Hainbat funtzio erabilgarri ere badaude objektu, eredu edo formula batetik informazioa atera nahi izanez gero. Adibide gisa, `alias()` dugu. Funtzio horrek formula batek zehazten duen eredu linealaren linealki menpe dauden terminoak topatzen ditu.

Azkenik, diagnostiko-grafikoak egiten dituzten funtzio grafikoak ere badaude, hala nola `plot` funtzioa, edo `termplot` (ikus aurreko adibidea), nahiz eta azken hori ez den guztiz generikoa, `predict`-i hots egiten baitio.

5.4. PAKETEAK

Hurrengo taulak base paketearekin banatzen diren paketeak erakusten ditu. Pakete bakoitza behin kargatuz gero erabil daiteke, ctest izan ezik. Azken hori bera bakarrik kargatzen da memorian R hasten denean:

```
> library(eda)
```

Pakete batean eskuragarri ditugun funtzioak ikusteko hauxe idatzi behar da:

```
> library(help=eda)
```

edo html formatuan daukagun laguntza erabiliz. Funtzio bati dagokion informazioa atzitzen ikasi dugu jada (7. orria).

Paketea	Deskripzioa
ctest	Proba klasikoak (Fisher, 'Student', Wilcoxon, Pearson, Bartlett, Kolmogorov-Smirnov,...
eda	"Exploratory Data Analysis"-en Tukey-k deskribitutako metodoak (doikuntza lineala eta medianen doikuntza)
lqs	erregresio erresistentea eta kobariantzaren estimazioa
methods	objektuentzako klaseak eta metodoak definitzea eta programazio-tresneria
modreg	erregresio modernoa (leunketa eta erregresio lokala)
mva	aldagai anitzeko analisia
nls	erregresio ez-lineala
splines	irudikapen polinomikoak
stepfun	banaketa-funtzio enpirikoak
tcltk	Tcl/Tk-ren interfaze grafikoaren elementuentzat R-tik interfazeak sortzeko erabiltzen diren funtzioak
tools	paketeen garapen eta administraziorako tresneria
ts	denbora-segiden analisiak

Contributions (ekarpenak) atalean aurki daitezkeen pakete-piloek R-n dauden metodo estatistikoen zerrenda handitzen dute. Pakete horiek bananduta banatzen dira eta R-n instalatu eta kargatu egin behar dira erabili ahal izateko. Ekarpenen atal horretan dauden paketeen zerrenda eta deskripzioa ikusteko CRANen web orria bisita dezan gonbidatzen dugu irakurlea[20]. Pakete horietariko asko *recommended* (aholkatuak) dira, datu-analisietan eskuarki erabiltzen diren metodo estatistikoak baitituzte (Windows-en, pakete aholkatuak R-ren instalazio arruntarekin batera banatzen dira SetupR.exe artxiboan). Aholkatutako paketeak ondorengo taulan deskribitzen dira laburki:

20. http://cran.r-project.org/src/contrib/PACKAGES.html

Paketea	Deskripzioa
boot	birlaginketa eta "bootstraping" erako metodoak
class	sailkatze-metodoak
cluster	taldekatze-metodoak
foreign	hainbat formatutako datuak irakurtzeko funtzioak (S3, Stata, SAS, Minitab, SPSS, Epi Info)
KernSmooth	leunketa nuklear eta dentsitate-estimazioetarako metodoak (bi aldagaiko nukleoak barne)
MASS	"Modern Applied Statistics with S-PLUS" (Venables & Ripley) liburutegiko hainbat funtzio, tresna eta datu ditu
mgcv	eredu gehigarri orokortuak
nlme	eredu lineal eta ez-linealak, efektu mistoekin
nnet	sare neuronalak eta eredu multinomial log-linealak
rpart	zatiketa errekurtsiboak
spatial	espazio-analisiak ("kriging", espazio-kobariantza,...)
survival	biziraupen-analisiak

Pakete bat instalatzeko prozedura sistema eragilearen eta R-ren instalazioaren menpekoa da: iturburu-kodea erabiltzeak edo artxibo bitarrak erabiltzeak R instalatzeko orduan, paketeen instalazioan zerikusi handia du. Azken hori bada R instalatzekoan erabilitakoa, CRANen topa daitezkeen aurrekonpilaturiko paketeak erabiltzea gomendatzen da. Windows-en, Rgui.exe artxibo bitarrak "Packages" izeneko menu bat du; horrek Internet bidez instalatzen ditu paketeak disko gogorrean CRAN web orritik zuzenean.

R lokalki konpilatu bada, paketeak iturburu-kodetik ('.tar.gz`moduko artxibo gisa banatzen da) instala daitezke. Adibidez, gee paketea instalatu nahi badugu, lehenik eta behin Internetetik gee 4.13-6.tar.gz artxiboa jaitsi behar dugu (4.13-6 zenbakiak paketearen bertsioa adierazten du; oro har, CRANen bertsio bakarra dago eskuragarri). Ondoren hau idatzi behar da sisteman (eta ez R-n):

```
R INSTALL gee_4.13-6.tar.gz
```

Paketeak maneiatzeko hainbat funtzio daude, `installed.packages()`, `CRAN.packages()` edo `download.packages()` esaterako. Honako komando hau erregularki idaztea ere gomendagarria da:

```
> update.packages()
```

Horrek sisteman dauden paketeen bertsioei begiratu eta CRANen eskuragarri daudenekin konparatzen ditu (komando horri, Windows-en, "Packages" menutik dei dakioke). Era horretara, erabiltzaileak beti azken bertsioko paketeak eduki ditzake.

6. Programazio praktikoa R-n

R-ren funtzionalitateak aztertu ditugunez, sar gaitezen programazio-lengoaia gisa duen erabilera ikustera. Praktikan erraz inplementa daitezkeen ideia sinpleak ikusiko ditugu.

6.1. BEGIZTAK ETA BEKTORIZAZIOA

R-k badu "menuak eta botoiak" erabiltzen dituzten beste programa estatistikoekin aldaratuz abantaila bat: elkarren atzetik exekutatzen diren analisi-segidak era errazean programatzeko aukera. Ezaugarri hori beste programazio-lengoaia guztiek dute, baina R-k baditu hainbat mekanismo programazioaren munduan esperientzia eta ezaguera gutxi dutenek ere nahiko erraz programa dezaten.

Beste lengoaiek bezala, R-k *kontrol-egiturak* ditu, C bezalako goi-mailako lengoaia batek dituenen antzekoak. Demagun `x` bektore bat dugula; `b`-ren balio bera duen `x`-ko elementu bakoitzeko, 0 balioa esleitu nahi diogu beste aldagai bati (`y`); baldintza ez bada betetzen 1 esleituko diogu. Lehenbizi `x`-ren luzera bera duen `y` bektore bat sortuko dugu:

```
y <- numeric(length(x))
for (i in 1:length(x)) if (x[i] == b) y[i] <- 0 else y[i] <-
1
```

Kortxeteak erabil daitezke agindu bat baino gehiago exekutatzeko:

```
for (i in 1:length(x)) {
     y[i] <- 0
     ...
}
if (x[i] == b) {
     y[i] <- 0
     ...
}
```

Beste aukera bat da aginduak exekutatzea soilik baldintza bat betetzen denean:

```
while (nirefun > minimoa) {
...
}
```

Hala ere, begizta eta egitura horiek ekidin daitezke R-k duen ezaugarri bati esker: *bektorizazioa*. Bektorizazioak begiztak espresioetan inplizituki sartzen ditu eta jada ikusi dugu kasu askotan. Adibidez, har dezagun bi bektoreren batuketa:

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

Batuketa hori begizta eran idatz zitekeen beste lengoaia askotan egiten den eran:

```
> z <- numeric(length(x))
> for (i in 1:length(z)) z[i] <- x[i] + y[i]
```

Kasu horretan, `z` bektorea aurrez sortu behar da elementuak indexatu behar baitira. Erraz ikusten da begizta esplizitu horrek bakarrik funtzionatuko duela `x`-ren eta `y`-ren luzerak berdinak diren kasuetan: programa aldatu beharko litzateke luzerak ezberdinak balira; lehen aukerak, ordea, edozein kasutan funtzionatuko luke.

Baldintzapeko exekuzioa (`if ... else`) indexazio logikoarekin ekidin daiteke; aurreko adibidean:

```
> y[x == b] <- 0
> y[x != b] <- 1
```

Begizten erabilera ekiditen duten hainbat funtzio ere badaude, esaterako, `apply()`. Horrek matrize baten errenkada edo zutabeekin egiten du lan, eta bere sintaxia honakoa da: `apply(X, MARGIN, FUN, ...)`; bertan, `X` matrize bat da, `MARGIN`ek errenkadak (`1`), zutabeak (`2`) edo biak (`c(1, 2)`) erabiliko diren adierazten du, `FUN` aplikatu behar den funtzio bat da (edo eragile bat, baina kasu horretan kortxeteetan zehaztu behar da), eta `...` `FUN`en aukerazko argumentuak dira. Ikus dezagun adibide sinple bat.

```
> x <- rnorm(10, -5, 0.1)
> y <- rnorm(10, 5, 2)
> X <- cbind(x, y) # X-ren zutabeek “x” eta “y” izenak
                     mantentzen dituzte
> apply(X, 2, mean)
         x         y
-4.975132 4.932979
> apply(X, 2, sd)
         x         y
0.0755153 2.1388071
```

`lapply()` funtzioak zerrenda batekin egiten du lan: bere sintaxia `apply`-renaren oso antzekoa da eta zerrenda bat itzultzen du.

```
> forms <- list(y ~ x, y ~ poly(x, 2))
> lapply(forms, lm)
[[1]]
Call:
FUN(formula = X[[1]])
Coefficients:
(Intercept)             x
    31.683       5.377
[[2]]
Call:
FUN(formula = X[[2]])
Coefficients:
(Intercept)       poly(x, 2)1       poly(x, 2)2
      4.9330            1.2181           -0.6037
```

`sapply()` funtzioa `lapply()`-ren bertsio malguago bat da. Argumentu nagusitzat bektore bat zein matrize bat har ditzake eta emaitzak era erosoago baten itzultzen ditu, oro har taula batean.

6.2. NOLA IDATZI PROGRAMAK R-N

Oro har, R-n idazten diren programak ASCII formatuan dauden eta '.R' bukaera duten artxiboetan gordetzen dira. Programa bat, eskuarki, erabiltzen da norbaitek eragiketa bat edo gehiago askotan egin nahi dituenean. Gure lehen adibidean, hiru hegazti-espezierentzat grafiko-mota bera egin nahi dugu eta datuak hiru artxibotan dauzkagu. Pausoz pauso joango gara eta arazo hori programatzeko hainbat era ikusiko ditugu.

Lehendabizi, egin dezagun gure programa erarik intuitiboenean, hots, behar diren komandoak bata bestearen atzetik exekutatuz. Hori bai, gailu grafikoa aurretik zatitu beharko dugu.

```
layout(matrix(1:3, 3, 1))           # leiho grafikoa zatitzen
du
data <- read.table("Swal.dat")      # datuak irakurtzen ditu
plot(data$V1, data$V2, type="l")
title("swallow")              # izenburu bat gehitzen du
data <- read.table("Wren.dat")
plot(data$V1, data$V2, type="l")
title("wren")
data <- read.table("Dunn.dat")
plot(data$V1, data$V2, type="l")
title("dunnock")
```

'#' karakterea iruzkinak gehitzeko erabiltzen da eta R-k alde batera uzten du exekuzioan.

Lehen programa horren arazoa da espezie gehiago gehitu nahi ditugunean oso luzea bilakatzea. Bestalde, komando batzuk askotan exekutatzen dira, eta beraz, multzokatu eta batera exekuta daitezke argumentu batzuk aldatuta. Hemen darabilgun estrategia argumentu horiek karaktere motako bektore batean sartu eta indexazioaren bidez balioak atzitzea da.

```
layout(matrix(1:3, 3, 1))           # leiho grafikoa zatitu
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
for(i in 1:length(species)) {
     data <- read.table(file[i])    # datuak irakurri
     plot(data$V1, data$V2, type="l")
     title(species[i])              # izenburua gehitu
}
```

Ohartu `file[ i]` argumentua ez dela komatxoen artean idazten `read.table()`-ren barnean, jada karaktere motakoa baita.

Orain gure programa askoz ere trinkoagoa da. Espezie berriak gehitzea askoz errazagoa da, beraien izenak programaren hasieran dauden bektore batzuetan baitaude.

Aurreko programek soilik funtzionatuko dute '.dat' datu-artxiboak R-ren lan-direktorioan badaude; horrela ez bada, erabiltzaileak lan-direktorioa aldatu edo programan helbide osoa idatzi beharko du (adibidez: `file <- "C:/data/Swal.dat"`). Programa Mybirds.R artxiboan badago, lehenik eta behin memoriara kargatu beharko da:

```
> source("Mybirds.R")
```

Aurreko adibidean bezala, horrek soilik funtzionatuko du `Mybirds.R` artxiboa R-ren lan-direktorioan badago; bestela, helbide osoa zehaztu behar da.

6.3. NOLA ERAIKI GEURE FUNTZIO PROPIOAK

Ikusi dugunez, R-n parentesien artean argumentuak dituzten funtzioen bitartez egiten da lan. R-k erabiltzaileari nahi dituen funtzioak eraikitzeko aukera ematen dio, eta noski, funtzio berri horiek beste guztien ezaugarri berak izango dituzte.

Norberaren funtzioak idazteak R-ren erabilera malgu, eraginkor eta arrazoizko bat dakar ondoriotzat. Har dezagun betiko adibidea: datu batzuk irakurri eta beraien grafikoa marraztea. Eragiketa hori hainbat egoeratan egin nahi badugu, funtzio bat idaztea ideia ona izan liteke:

```
nirefun <- function(S, F)
{
     data <- read.table(F)
     plot(data$V1, data$V2, type="l")
     title(S)
}
```

Funtzio hori exekutatzeko, lehenik eta behin, memorian kargatu behar dugu eta horretarako era asko ditugu. Funtzioaren lerroak teklatutik zuzenean idatz daitezke, beste edozein komando bezalaxe, edo testu-editore batetik kopiatu eta itsatsi. Funtzioa ASCII artxibo batean gordeta badago, `source()` funtzioarekin karga daiteke, beste programak bezala. Erabiltzaileak R hasten den bakoitzean bere funtzioa kargatu nahi badu, hori `.RData` artxibo batean gorde beharko du; artxibo-mota horri "lan-espazio" (ingelesezko 'workspace' hitzetik) izena ematen zaio eta lan-direktorioan egon behar du automatikoki memoriara kargatzeko. Beste aukera bat da '.Rprofile' edo 'Rprofile' artxiboa aldatzea (ikus `?Startup` xehetasun gehiago izateko). Azkenik, pakete bat sortzea ere posible da, baina aukera hori ez dugu hemen azalduko (ikus "Writing R extensions" gidaliburua).

Behin funtzioa memoriara kargatu dugunean, komando bakar bat erabilita exekuta daiteke. Adibidez, `nirefun("swallow", "Swal.dat")`. Beraz, orain gure programaren hirugarren bertsio bat dugu:

```
layout(matrix(1:3, 3, 1))
nirefun("swallow", "Swal.dat")
nirefun("wren", "Wrenn.dat")
nirefun("dunnock", "Dunn.dat")
```

`sapply()` ere erabil dezakegu. Era horretara programaren laugarren bertsio bat izango dugu:

```
layout(matrix(1:3, 3, 1))
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
sapply(species, nirefun, file)
```

R-n funtzio baten barnean erabili behar diren aldagaiak erazagutzea ez da beharrezkoa (C edo Fortran lengoaietan ez bezala). Funtzio bat exekutatzen denean, R-k *esparru lexikografikoa* deitzen den arau bat erabiltzen du aldagai bat globala ala funtzio baten aldagai lokala den erabakitzeko. Mekanismo hori hobeto ulertzeko, har dezagun hurrengo adibidea:

```
> foo <- function() print(x)
> x <- 1
> foo()
[1] 1
```

`x` izena ez dago `foo()`-ren barnean definituta, beraz R-k `x`-ren *inguruko* esparruan bilatuko du eta bere balioa inprimatuko du (bestela, errore-mezu bat igorri eta exekuzioa gelditu ko da).

`x` funtzio baten barne-objektu baten izentzat erabiltzen bada, `x`-ren balioa ingurune globalean (funtziotik at) ez da aldatuko.

```
> x <- 1
> foo2 <- function() { x <- 2; print(x) }
> foo2()
[1] 2
> x
[1] 1
```

Kasu honetan `print()`-ek bere ingurunean definitutako `x` objektua darabil, hots, `foo2`-ren ingurunean.

Arestian erabili dugun "*inguruko*" hitza oso garrantzitsua da. Gure bi adibide-funtzioetan *bi* ingurune daude: bat globala eta bestea funtzio bakoitzarekiko (`foo` eta `foo2`) lokala. Hiru ingurune edo gehiago badaude habiratuta, bilaketa, ingurune horretatik hasten da eta inguruko inguruneetara zabaltzen joango da ingurune globalera iritsi arte.

Funtzio bati argumentuak zehazteko bi era daude: beraien posizioarekin edo izenarekin (*argumentu markatuak* ere deitzen zaie). Adibidez, har dezagun hiru argumentudun funtzio bat:

```
foo <- function(arg1, arg2, arg3) {...}
```

`foo()` exekuta daiteke `arg1`, ... izenak erabili gabe, objektu bakoitzaren posizioa egokia bada; adibidez: `foo(x, y, z)`. Hala ere, posizioa ez da kontuan hartzen argumentuen izenak erabiltzen badira, adibidez, `foo(arg3=z, arg2=y, arg1=x)`. Funtzioen definizioan defektuzko balioak erabiltzea da R-ren funtzioen beste ezaugarri garrantzitsu bat. Adibidez:

```
foo <- function(arg1, arg2 = 5, arg3 = FALSE) {...}
```

`foo(x)` eta `foo(x, 5, FALSE)` komandoek emaitza bera izango dute. Funtzio baten definizioan defektuzko balioak zehaztea nahiko erabilgarria da eta malgutasun handiagoa ematen du.

Malgutasunaren beste adibide bat Ricker-en ereduari jarraitzen dion populazio baten portaera simulatzen duen honako funtzio honek dakarkigu:

$$N_{t+1} = N_t \exp\left[r\left(1 - \frac{N_t}{K}\right)\right]$$

Eredu hori populazioen dinamiketan asko erabiltzen da, batez ere arrainen ikerketa demografikoetan. Funtzio baten bitartez, eredu hori simulatu nahi dugu r hazkunde-tasarekiko eta N_0 populazioaren hasierako balioarekiko (K karga-kapazitatea 1 izan ohi da eta balio hori erabiliko dugu bere defektuzko baliotzat); emaitzak denborarekiko banakoen kopurua nola aldatzen den erakusten duen grafiko batean aurkeztuko dira. Aukera bat gehituko dugu erabiltzaileak simulazioaren azken pausuen emaitzak ikus ditzan (defektuz emaitza guztiak azaltzen dira grafikoan). Beheko funtzioak gauzatzen du Ricker-en ereduaren zenbakizko analisia.

```
ricker <- function(nzero, r, K=1, denbora=100, nondik=0,
                     nora=denbora)
{
        N <- numeric(denbora+1)
        N[1] <- nzero
        for (i in 1:denbora) N[i+1] <- N[i]*exp(r*(1 -
N[i]/K))
        Denbora <- 0:denbora
        plot(Denbora, N, type="l", xlim=c(nondik, nora))
}
```

Erabil dezagun funtzioa ereduaren propietateak aztertzeko:

```
> layout(matrix(1:3, 3, 1))
> ricker(0.1, 1); title("r = 1")
> ricker(0.1, 2); title("r = 2")
> ricker(0.1, 3); title("r = 3")
```

7. R-ri buruz aurki daitezkeen bestelako dokumentu eta testuak

Gidaliburuak. R-k defektuz R_HOME/doc/manual/ (R_HOME R instalatuta dagoen lekua izanik) helbidean instalatzen diren hainbat gidaliburu dakartza. Guztiak ingelesez daude:

- "An Introduction to R" [R-intro.pdf]
- "R Installation and Administration" [R-admin.pdf]
- "R Data Import/Export" [R-data.pdf]
- "Writing R Extensions" [R-exts.pdf]
- "R Language Definition" [R-lang.pdf]

Artxiboak hainbat formatutan (pdf, html, texi ...) egon daitezke instalazioaren arabera.

FAQ. R bere FAQ (*galdera ohikoenak*) propioarekin datorkigu. Hori R_HOME/doc/html/ direktorioan topa dezakegu. R-FAQ horren bertsioa erregularki berritzen da CRAN web orrian: http://cran.r-project.org/doc/FAQ/R-FAQ.html.

On line baliabideak. CRANen eta R-ren web orriek hainbat dokumentu, baliabide bibliografiko eta beste orri batzuetarako loturak dituzte. R-ren inguruko eta, oro har, metodo estatistikoei buruzko argitalpen-zerrenda (liburuak eta artikuluak) ere topa daiteke[21], baita R-ren erabiltzaileek idatzitako hainbat dokumentu eta tutoretza ere[22].

21. http://www.R-project.org/doc/bib/R-publications.html

22. http://cran.r-project.org/other-docs.html. Hemen gaztelaniaz idatzitako beste bi gidaliburu topa daitezke.

Posta-zerrendak. R-n hiru eztabaida-zerrenda daude; harpidetzea nahi izanez gero, mezu bat bidali edo http://www.R-project.org/mail.html helbideko artxiboak irakurri behar dira.

'r-help' eztabaida-zerrenda R-ren erabiltzaileentzat oso leku egokia izan liteke informazioa lortzeko (beste bi zerrendak bertsio berriak iragartzera, pakete berriak jartzera... eta programatzaileei zuzenduta daude). Erabiltzaile askok bidali dituzte hainbat funtzio eta programa 'r-help'-era. Horiek guztiak bertako artxiboetan topa daitezke. R-rekin arazorik balego, ondorengo pausuak jarraitzea oso garrantzitsua da 'r-help'-era mezu bat bidali aurretik:

1. on line laguntza irakurri kontu handiz (bilaketa-makina erabil daiteke)
2. galdera ohikoenak irakurri (R-FAQ)
3. 'r-help'-eko artxiboetan bilatu, eman berri dugun helbidean edo hainbat web orritan topa daitezkeen bilaketa-makinak erabilita[23].

R News. *R News* aldizkari elektronikoak eztabaida-zerrenda elektronikoen eta ohiko argitalpen zientifikoen artean dagoen hutsunea betetzea du helburutzat. Lehen zenbakia 2001eko urtarrilean argitaratu zen eta urteko hiru zenbaki ekoizten dira. Kurt Hornik eta Friedrich Leisch dira editoreak[24].

Argitalpen batean R aipatzeko. Azkenik, argitalpenen batean R aipatu nahi bada, jatorrizko artikulua aipatzea beharrezkoa da:

Ihaka R. & Gentleman R. 1996. R: a language for data analysis and graphics.

Journal of Computational and Graphical Statistics 5: 299–314.

23. Web orri horien helbideak http://cran.r-project.org/search.html helbidean topa daitezke
24. http://cran.r-project.org/doc/Rnews/