R tutuksi

Ville Hyvönen, Henri Karttunen, Toni Lehtonen, Aku Leivonen
& Savi Virolainen
Matematiikan ja tilastotieteen laitos
Helsingin yliopisto

Kevät 2018

Sisältö

Saatteeksi

Materiaali on jaettu useaan kappaleeseen: jokaisen osan materiaali sisältää lyhyet esittelyt kappaleen aiheista ja niihin liittyviä esimerkkejä. Kappaleiden loppuihin on listattuna keskeisimpiä aiheisiin liittyviä R:n funktioita, mutta näiden funktioiden tarkempaan toimintaan tutustuminen jätetään lukijan tehtäväksi. Moniste on alunperin luotu kurssin Tilastollinen päättely R-ohjelmistolla tueksi, ja sitä on myöhemmin jalostettu yleisemmäksi avuksi R:n alkeiden opetteluun erityisesti kursseilla Tilastotiede ja R tutuksi I ja II.

Moniste ei ole läheskään kaiken kattava esitys R:n käytön perusteista, mutta aiheesta löytyy paljon tasokkaita kirjoja ja nettimateriaaleja, esimerkiksi:

- Peter Dalgaard: Introductory Statistics with R (Springer, 2002 ja 2008).

Jos R ohjelmointikielenä kiinnostaa, niin hyviä teoksia syvällisempään tutustumiseen ovat ainakin:

- Norman Matloff: Art of R Programming (No Starch Press 2011)
- Hadley Wickham: Advanced R: `http://adv-r.had.co.nz/`.

Monisteessa ei käsitellä R:n asentamiseen, erilaisiin käyttöliittymiin tai RTMC:hen liittyviä asioita. Jos R-ohjelmisto on täysin uusi käsite, on syytä tutustua aluksi RStudion ja RTMC:n perusasioihin esimerkiksi näihin erikseen laadittujen videosarjojen avulla.

Huomautuksia virheistä voi lähettää sähköpostitse osoitteeseen aku.leivonen at helsinki.fi.

Kappale 1

Tietotyypit ja skalaarimuuttujat

Tässä kappaleessa tutustutaan joihinkin R:n yleisimmin käyttetyihin tietotyyppeihin. Lisäksi käydään läpi niiden peruslaskutoimituksia ja tarkastellaan myös loogisia operaattoreita.

R:n käyttämiä tietotyyppejä ovat esimerkiksi kokonaisluku (integer), liukuluku (double tai numeric[1]) eli käytännössä reaaliluku, totuusarvo (logical), merkkijono (character) ja lista (list). Muuttujan tietotyypin tunteminen on tärkeää muun muassa siksi, että se määrittelee miten kyseistä muuttujaa voidaan käsitellä ja minkälaisia operaatioita sille tai sen avulla voidaan tehdä. Lisäksi kappaleessa 2 tarkasteltavat ns. atomiset tietorakenteet, eli vektorit, matriisit ja arrayt, voivat käsittää elementteinään vain saman tietotyypin alkioita.

1.1 Skalaarimuuttujat ja lukujen tulostaminen

Skalaarit ovat yksipaikkaisia vektoreita, ja niitä voidaan käsitellä R-ohjelmistolla sellaisenaan, syöttämällä haluttu luku konsoliin. Desimaalierottimena toimii tavallinen piste [.].

Esimerkki 1.1. Tulostetaan lukuja:

```
> 15

[1] 15
```

[1] numeric on formaalisti reaalimuuttujan ”mode”, ja sillä viitataan doublea yleisemmin numeeriseen muuttujaan. Esimerkiksi integer-tietotyypin muuttujat eivät ole doubleja, mutta numeerisina muuttujina ne ovat numericceja.

```
> 3.14159

[1] 3.14159
```

Tehdään heti selväksi, että komentorivin alussa näkyvä ”suurempi kuin” (>) -merkki näkyy ainoastaan konsolissa eikä sitä kirjoiteta koskaan ohjelmakoodiin. Lisäksi tulosterivin alussa näkyvä hakasuluissa oleva luku tarkoittaa vain, monesko vektorin arvo on tulosterivin ensimmäinen arvo. Vektoreita käsitellään seuraavassa luvussa joten toistaiseksi tähän hakasuluissa olevaan ykköseen ei tarvitse kiinnittää huomiota.

Skalaari (tai jokin muu muuttuja) voidaan sijoittaa lähes minkä tahansa nimiseen muuttujaan käyttämällä sijoitusoperaattoria <-. Nimen on syytä kuitenkin alkaa kirjaimella, eikä olemassa olevien funktioiden nimiä ole järkevää tai mahdollista ylikirjoittaa. Tallennetun muuttujan sisällön voi tulostaa antamalla muuttujan nimen komentona.

Esimerkki 1.2. Sijoitetaan muuttujiin arvoja ja tulostetaan ne:

```
> a <- 42
> a

[1] 42

> b <- 0.001
> b

[1] 0.001
```

1.2 Muuttujan tyyppi

Muuttujan tietotyyppiä voi tarkastella `typeof()`-komennolla. Tässä esimerkkeinä käytettyihin tietotyyppeihin tutustutaan tarkemmin niitä käsittelevissä osioissa.

Esimerkki 1.3. Tallennetaan erilaisia skalaarimuuttujia ja selvitetään niiden tietotyypit:

```
> a <- 5
> b <- TRUE
> c <- 13.34
> d <- "Nainen"

> typeof(a)
```

```
[1] "double"

> typeof(b)

[1] "logical"

> typeof(c)

[1] "double"

> typeof(d)

[1] "character"
```

Muuttujien tyyppiä voi testata `is`- ja muuttaa `as`-alkuisilla funktioilla. Päätteeksi funktioon laitetaan pisteen jälkeen halutun tietotyypin nimi. Muuttujan tyyppiä testaava `is`-alkuinen funktio palauttaa totuusarvon `TRUE` tai `FALSE` sen mukaan onko muuttuja kysyttyä tyyppiä.

Esimerkki 1.4. Tallennetaan luku 42 merkkijonona muuttujaan `a`, selvitetään sen tietotyyppi ja muutetaan se numeeriseksi muuttujaksi:

```
> a <- "42"
> a

[1] "42"

> is.double(a)

[1] FALSE

> is.character(a)

[1] TRUE

> a <- as.double(a)
> a

[1] 42
```

```
> is.double(a)

[1] TRUE

> is.character(a)

[1] FALSE
```

1.3 Muuttujan luokka

R:ssä muuttujille tai olioille, kuten vektoreille tai matriiseille, voi antaa erilaisia ominaisuuksia eli attribuutteja. Yksi yleisistä muuttujalle määriteltävistä attribuuteista on luokka (class), joka määrittelee olion käyttäytymistä ja suhdetta muun luokan omaaviin olioihin. Jos oliolla ei ole luokka-attribuuttia, on sillä kuitenkin ns. implisiittinen luokka, joka on esimerkiksi monien skalaarimuuttujien tapauksessa sama kuin muuttujan tyyppi (tai ”mode”).

Luokkiin ei tässä monisteessa tutustuta sen tarkemmin, mutta niiden olemassaolo on hyvä tiedostaa. Esimerkiksi kappaleessa 4.1 tarkasteltavat luokittelumuuttujat ovat tietotyypiltään kokonaislukuja ja luokaltaan faktoreita (factor), ja kappaleessa 3.2 tarkasteltavat taulukot ovat tietotyypiltään listoja ja luokaltaan taulukoita (data frame).

Muuttujan luokka voidaan selvittää komennon `class()`-avulla.

Esimerkki 1.5. Tallennetaan luku 42 kokonaislukuna muuttujaan `a` ja tarkastetaan sen tietotyyppi ja luokka. Muutetaan sitten muuttuja `a` luokaltaan faktoriksi ja tarkastetaan sen tietotyyppi ja luokka uudelleen.

```
> a <- as.integer(42)
> a

[1] 42

> typeof(a)

[1] "integer"

> class(a)

[1] "integer"
```

```
> a <- as.factor(a)
> a

[1] 42
Levels: 42

> typeof(a)

[1] "integer"

> class(a)

[1] "factor"
```

1.4 Numeeristen skalaarimuuttujien laskuoperaatiot

Reaaliluvuilla voidaan suorittaa helposti kaikki peruslaskutoimitukset: yhteen- ja vähennyslasku sekä kerto- ja jakolasku. Laskuoperaatiot voidaan tehdä myös tallennetuille muuttujille, joita käytetään laskun osana aivan kuin numeroita.

Esimerkki 1.6. Yhteen- ja vähennyslasku

```
> a <- 5
> b <- 3
> a - b

[1] 2

> a + b

[1] 8

> a - 6

[1] -1
```

Esimerkki 1.7. Kerto- ja jakolasku

```
> a*b
```

```
[1] 15

> a/b

[1] 1.666667

> a*3/b

[1] 5
```

Esimerkki 1.8. Logaritmi, exponentti, potenssi ja modulo

```
> log(a)

[1] 1.609438

> exp(a)

[1] 148.4132

> a^b

[1] 125

> a %% b

[1] 2
```

1.5 Loogiset operaattorit ja totuusarvot

Totuusarvoja (logical) ovat itseselitteiset `TRUE` ja `FALSE` (nämä voi myös lyhentää kirjaimilla `T` ja `F`), joita käytetään, kun jokin asia on joko totta tai epätotta. Kuten myöhemmin nähdään, totuusarvoja voidaan myös käyttää alkioiden valitsemiseen tietorakenteista.

R:ssä on käytössä normaalit vertailuoperaatiot suurempi kuin (`>`), pienempi kuin (`<`), suurempi tai yhtä suuri kuin (`>=`), ja pienempi tai yhtä suuri kuin (`<=`). Yhtäsuuruusvertailu saadaan operaattorilla `==` ja erisuuruusvertailu operaattorilla `!=`. **Huom.** Pelkkä tavallinen yhtäsuuruusmerkki = toimii useimmiten kuten sijoitusoperaattori, joten sitä ei voi käyttää vertailussa. Vertailuoperaatiot antavat tuloksena totuusarvon `TRUE` tai `FALSE`. Vertailun tuloksen voi tallentaa muuttujaan.

Esimerkki 1.9. Vertailuoperaattorien käyttöä

```
> 1 < 0

[1] FALSE

> x <- 1
> x < 2

[1] TRUE

> 2 == 2

[1] TRUE

> y <- 2 != 2
> y

[1] FALSE

> y == TRUE

[1] FALSE
```

Lisäksi käytössä ovat looginen JA (`&`), TAI (`|`) ja negaatio (`!`), joiden avulla voidaan kirjoittaa pidempiä ehtolauseita.

Esimerkki 1.10.

```
> (1 > 0) & (1 < 0)

[1] FALSE

> (1 > 0) | (1 < 0)

[1] TRUE

> !(1 < 0)

[1] TRUE
```

```
> TRUE & FALSE

[1] FALSE

> TRUE | FALSE

[1] TRUE

> !FALSE

[1] TRUE
```

1.6 Kokonaisluvut

Kokonaisluvut (integer) ovat nimensä mukaisesti R:n tietotyyppi kokonaisluvuille. Tavanomaiset laskuoperaatiot tehdään tyypillisesti reaalilukujen (double) avulla, mutta kokonaislukuja käytetään esimerkiksi myöhemmin tarkasteltavien luokittelumuuttujien eli faktoreiden määrittelemisessä. Huomaa, että vaikka jokin muuttuja olisi numeeriselta arvoltaan kokonaisluku, sen tietotyyppi ei välttämättä ole ”integer”.

Esimerkki 1.11.

```
> a <- 42
> b <- as.integer(42)
> typeof(a)

[1] "double"

> typeof(b)

[1] "integer"

> is.numeric(a)

[1] TRUE

> is.numeric(b)

[1] TRUE
```

1.7 Merkkijonot

Numeeristen muuttujien ja totuusarvojen lisäksi R:ssä on käytössä oma tietotyyppi merkkijonoille (character). Merkkijonot kirjoitetaan joko yksin- tai kaksinkertaisten lainausmerkkien sisään, ja niitä voidaan sijoittaa muuttujaan aivan kuten numeroita ja totuusarvojakin. Yksittäisille merkeille ei ole omaa tietotyyppiä, vaan ne tallennetaan merkkijonoina, joiden pituus on yksi.

Esimerkki 1.12.

```
> "R"

[1] "R"

> merkkijono <- "data-analyysi"
> merkkijono

[1] "data-analyysi"

> typeof(merkkijono)

[1] "character"
```

1.8 Puuttuvat arvot

R:ssä puuttuvaa arvoa merkitään `NA`:lla. Tietotyypiltään ja luokaltaan puuttuva arvo on itse asiassa totuusarvo (logical), mutta se käyttäytyy eri tavalla kuin `TRUE` ja `FALSE`. Puuttuvaa arvoa voi ajatella muuttujan tyhjänä arvona, joiden merkitys korostuu data-analyysissä, koska suurissa aineistoissa on luonnostaan lähes aina puuttuvia arvoja. Jos puuttuviin arvoihin soveltaa äsken esiteltyjä skalaarimuuttujien tai totuusarvojen operaatioita, on lopputulos myös puuttuva arvo eli `NA`.

Esimerkki 1.13.

```
> NA+2

[1] NA

> NA == 3

[1] NA
```

```
> NA == NA

[1] NA

> !NA

[1] NA
```

Koska loogiset operaatiot eivät toimi puuttuville arvoille samalla tavalla kuin esimerkiksi skalaarimuuttujille, tiedon siitä, onko muuttujan arvo puuttuva voi tarkistaa `is.na()`-funktiolla.

Esimerkki 1.14.

```
> a <- NA
> # Ei näin
> a == NA

[1] NA

> # Vaan näin
> is.na(a)

[1] TRUE
```

1.9 Sulkujen käytöstä

R-ohjelmoinnissa erilaisten sulkujen käyttäminen on keskeisessä osassa, koska R-ohjelmointi perustuu pitkälti valmiiden sekä itse rakennettavien funktioiden käyttöön. Tavallisia sulkuja `()` käytetään kaikkien R:n funktioiden kanssa siten, että kaikki funktioille annettavat argumentit syötetään sulkujen sisään. Tästä nähtiin esimerkkeinä jo muun muassa logaritmifunktio `log()` sekä muuttujan tietotyypin tarkastava funktio `is.numeric()`. Sulkuja voidaan käyttää kuitenkin myös esimerkiksi muuttamaan laskuoperaatioiden järjestystä, sillä R suorittaa laskuoperaatiot yleisen laskujärjestyssopimuksen mukaan.

Esimerkki 1.15.

```
> 2+10/2
```

```
[1] 7

> (2+10)/2

[1] 6
```

Turhien sulkujen lisäämisellä ei ole vaikutusta, kunhan avaavien ja sulkevien sulkujen määrä on sama.

Esimerkki 1.16.

```
> (((2+10)/2))

[1] 6
```

Jos sulkevia sulkuja on liikaa, komento päättyy virheilmoitukseen. Jos taas avaavia sulkuja on liikaa, rivin alkuun ilmestyy plus-merkki, koska R tulkitsee tämän rivinvaihdoksi ja keskeneräiseksi komennoksi. Tällöin R odottaa puuttuvaa sulkevaa sulkua.

Esimerkki 1.17.

```
> (2+10)/2)

Error: unexpected ')' in "(2+10)/2)"

> ((2+10)/2
+
```

Jos siis rivin alkuun jää plus-merkki eikä mitään tapahdu, komento tulee päättää lisäämällä riittävä määrä sulkevia sulkuja perään.

```
> ((2+10)/2
+ )

[1] 6
```

1.10 Keskeisiä operaattoreita ja funktioita

Oletetaan, että `a` ja `b` ovat skalaareja, joilla voidaan laskea seuraavat laskutoimitukset ja operaatiot.
Yhteenlasku: `a+b`
Vähennyslasku: `a-b`

Kertolasku: `a*b`
Jakolasku: `a/b`
Potenssi: `a^b`

Itseisarvo: `abs(a)`
Neliöjuuri: `sqrt(a)`
Logaritmi (luonnollinen): `log(a)`
Logaritmi (10-kantainen): `log10(a)`
Logaritmi (b-kantainen): `log(a,base=b)`
Eksponenttifunktio: `exp(a)`
Trigonometriset funktiot: `sin(a)`, `cos(a)`, `tan(a)`, `asin(a)`, `acos(a)`, `atan(a)`

Kertoma: `factorial(a)`
Binomikerroin (a yli b): `choose(a,b)`
Kymmenen potenssit: esim. `1e5` (100 000), `9e-4` (0.0009)
Kattofunktio: `ceiling(a)`
Lattiafunktio: `floor(a)`
Pyöristys b:n desimaalin tarkkuuteen: `round(a, digits=b)`
Jakojäännös: `a%%b`

Suurempi kuin: `a > b`
Pienempi kuin: `a < b`
Vähintään: `a >= b`
Korkeintaan: `a <= b`
Yhtäsuuruusvertailu: `a == b`
Erisuuruusvertailu: `a != b`
TAI: `|` (pystyviiva)
JA: `&`

`typeof(a)`: Muuttujan tietotyyppi
`class(a)`: Muuttujan luokka

Muuttujan tyypin tarkistus (palauttaa `TRUE`, jos haluttua tyyppiä)
`is.numeric(a)`
`is.integer(a)`
`is.double(a)`
`is.character(a)`
`is.logical(a)`
`is.na(a)`: Tarkistaa, onko muuttuja puuttuva arvo

Muuttujan tyypin vaihtaminen
`as.numeric(a)`
`as.character(a)`
jne.

Kappale 2

Atomiset tietorakenteet

Atomisilla tietorakenteilla tarkoitetaan sellaisia datan säilömismuotoja, jotka voivat käsittää vain saman tietotyypin alkioita, esimerkiksi vain numeerisia arvoja. Data-analyysissä yleisimmin käytettyjä atomisia tietorakenteita ovat vektorit, matriisit ja "moniulotteiset matriisit" eli arrayt. Muita kuin atomisia tietorakenteita ovat esimerkiksi kappaleessa 3.1 tarkasteltavat listat, joihin voi säilöä myös eri tietotyypin alkioita[1].

2.1 Vektorit

2.1.1 Vektorien luominen

Skalaareita, kuten vektoreitakin, voidaan yhdistää uusiksi vektoreiksi käyttäen yhdistämisfunktiota `c()`. Saatu vektoriarvoinen muuttuja voidaan tallentaa muuttujaan aivan kuten skalaaritkin.

Esimerkki 2.1. Luodaan vektorit $(0, 1, 2, 3)$ ja $(0, 1, 2, 3, 4, 5, 6)$:

```
> a <- c(0, 1, 2, 3)
> b <- c(a, 4, 5, 6)
> a

[1] 0 1 2 3

> b
```

[1] Itse asiassa listat ovat ei-atomisia vektoreita, mutta tässä monisteessa vektorilla tarkoitetaan atomista vektoria ilman eri mainintaa

```
[1] 0 1 2 3 4 5 6
```

Esimerkki 2.2. Edellisen esimerkin vektori voidaan tehdä myös kaksoispisteoperaattorilla tai funktiolla `seq`, jotka luovat säännöllisiä jonoja.

```
> 0:6

[1] 0 1 2 3 4 5 6

> seq(0, 6, by=1)

[1] 0 1 2 3 4 5 6
```

Esimerkki 2.3. Nämä toimivat myös toiseen suuntaan ja erilaisilla väleillä:

```
> 6:0

[1] 6 5 4 3 2 1 0

> seq(0, 100, by=10)

 [1]   0  10  20  30  40  50  60  70  80  90 100

> seq(0, 6, by=0.5)

 [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

Jos samaan vektoriin (tai mihin tahansa atomiseen tietorakenteeseen) yrittää tallentaa usean eri tietotyypin alkioita, R muuttaa koko vektorin ”yleisempää” muotoa olevaan tietotyyppiin. Esimerkiksi liitettäessä merkkijonoja ja numeroita samaan vektoriin R muuttaa numerot merkkijoiksi.

Esimerkki 2.4.

```
> a <- c("Pearson", "Gosset", "Fisher", 5)
> a

[1] "Pearson" "Gosset"  "Fisher"  "5"

> typeof(a)

[1] "character"
```

Katsotaan seuraavaksi, mitä tulosterivien alussa olevat hakasuluissa olevat luvut tarkoittavat.

Esimerkki 2.5. Luodaan pitkä vektori.

```
> seq(0,200,by=2)
  [1]   0   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72
 [38]  74  76  78  80  82  84  86  88  90  92  94  96  98 100 102 104 106 108 110 112 114 116 118 120 122 124 126 128 130 132 134 136 138 140 142 144 146
 [75] 148 150 152 154 156 158 160 162 164 166 168 170 172 174 176 178 180 182 184 186 188 190 192 194 196 198 200
```

Tulosterivin hakasuluissa oleva luku kertoo siis, monesko vektorin arvo kyseisen tulosterivin ensimmäinen arvo on, eli rivin ensimmäisen arvon indeksin. Esimerkiksi nähdään, että vektorissa, jossa on luvut 0 kahteensataan kahden välein, 38. arvo on 74 ja 75. arvo on 148.

2.1.2 Vektorien käsittely

Vektorin alkioita voidaan valita antamalla halutut indeksit vektorin nimen perässä olevissa hakasuluissa. Huom. R:ssä indeksointi alkaa aina 1:stä eikä 0:sta. Jos viitataan indeksiin, jota ei ole vektorissa, tuloksena on puuttuva arvo, eli `NA`.

Esimerkki 2.6. Luodaan vektori `a` ja valitaan sen alkioita:

```
> a <- c(3, 6, 30, 3, 0)
> a[1]

[1] 3

> a[5]

[1] 0

> a[6]

[1] NA
```

Vektorista voidaan valita myös useita eri alkoita kerralla antamalla halutut indeksit vektorina. Tuloksena on tällöin osavektori.

Esimerkki 2.7. (Jatkoa edelliseen) valitaan vektorin `a` kolme viimeistä alkiota eri tavoilla:

```
> a[c(3, 4, 5)]
```

```
[1] 30  3  0

> a[3:5]

[1] 30  3  0

> a[3:length(a)]

[1] 30  3  0

> a[c(-1, -2)]

[1] 30  3  0

> a[c(F, F, T, T, T)]

[1] 30  3  0
```

Funktio `length()` palauttaa vektorin pituuden. Negatiiviset indeksit taas palauttavat koko vektorin lukuunottamatta näitä indeksejä, esimerkiksi `a[c(-1,-2)]` antaa vektorin `a` lukuunottamatta sen ensimmäistä ja toista alkoita. Vektoria voidaan indeksoida loogisella vektorilla, jolloin valitaan alkiot, joiden kohdalla on indeksivektorin arvo on `TRUE`.

Edellisen esimerkin viimeisestä tapaa valita vektorin alkoita voidaan käyttää vektorin alkoiden valitsemiseen ehtolauseiden avulla, mikä tulee jatkossa olemaan erittäin kätevää osa-aineistojen valitsemisessa. Ehtolause `a > 3` vertaa jokaista `a`:n alkiota lukuun kolme ja palauttaa vertailun tuloksen totuusarvovektorina:

Esimerkki 2.8.

```
> a > 3

[1] FALSE  TRUE  TRUE FALSE FALSE
```

Nyt käytettäessä ehtolausetta `a > 3` `a`:n indeksinä, sen pitäisi palauttaa `a`:n toinen ja kolmas alkio, eli juuri ne `a`:n alkiot, jotka ovat suurempia kuin 3.

Esimerkki 2.9.

```
> a[a > 3]

[1]  6 30
```

Jos vektorissa on puuttuvia arvoja, lopputulos ei kuitenkaan aina ole toivottu. Yri-

tetään esimerkiksi selvittää kuinka moni vektorin arvoista on alle kuusi `length()`-funktion [2] avulla, kun vektori sisältää puuttuvia arvoja.

Esimerkki 2.10.

```
> a <- c(3,NA,6,54,NA,5)
> a < 6

[1]  TRUE    NA FALSE FALSE    NA  TRUE

> a[a<6]

[1]  3 NA NA  5

> length(a[a<6])

[1] 4
```

Huomataan, että vertailu `a<6` palauttaa myös puuttuvan arvon niille `a`:n alkioille, joiden arvo puuttuu. Käytettäessä ehtoa edelleen `a`:n indeksinä, mukaan otetaan niiden `a`:n arvojen lisäksi, jotka ovat pienempiä kuin kuusi, myös puuttuvat arvot. Siten lopputulos on neljän pituinen vektori, joka sisältää kaksi puuttuvaa arvoa halutun kahden pituisen vektorin sijasta. Yksi ratkaisu tähän ongelmaan on soveltaa `which()`-funktiota, joka antaa ne vektorin indeksit, joille argumenttina annettu ehto on tosi, ja jättää automaattisesti puuttuvat arvot huomioimatta.

Esimerkki 2.11.

```
> which(a<6)

[1] 1 6

> a[which(a<6)]

[1] 3 5

> length(a[which(a<6)])

[1] 2
```

[2] Komento `sum(a<6, na.rm=T)` olisi yksinkertaisempi tapa selvittää asia, koska tämä summaa `TRUE`-totuusarvoja yhteen.

Tässä siis `which()`-funktio palauttaa ensin indeksit, joille `a`:n arvo on pienempää kuin kuusi, ja sijoitettaessa nämä `a`:n indeksiksi saadaan kyseiset arvot, eli 3 ja 5.

Vieläkin suorempi tapa on käyttää `subset()`-funktioita, joka valitsee ensimmäisenä argumenttina annetusta vektorista ne alkiot, joille toisena argumenttina annettu ehto on tosi. Myös `subset()` jättää automaattisesti puuttuvat arvot huomioimatta.

Esimerkki 2.12.

```
> subset(a, a<6)

[1] 3 5

> length(subset(a, a<6))

[1] 2
```

2.1.3 Numeeristen vektorien laskutoimitukset

Numeerisilla vektoreilla voidaan laskea kuten yksittäisillä skalaarimuuttujilla. Laskuoperaatiot tapahtuvat alkioittain. Yleensä sovelluksissa, esim. aineistoissa, operoidaan samanpituisten vektorien kanssa, jolloin **samanpituisten vektorien alkiot vastaavat toisiaan**. Jos pidemmän vektorin pituus ei ole lyhyemmän vektorin pituuden monikerta, R antaa varoituksen, mutta ei virhettä.

Esimerkki 2.13.

```
> a <- 1:8
> b <- 11:18
> a

[1] 1 2 3 4 5 6 7 8

> b

[1] 11 12 13 14 15 16 17 18

> a+b

[1] 12 14 16 18 20 22 24 26

> a*b
```

```
[1]  11  24  39  56  75  96 119 144

> a + c(1,2)

[1]  2  4  4  6  6  8  8 10

> a + c(1,2,3)

[1]  2  4  6  5  7  9  8 10
Warning message:
In a + c(1, 2, 3) :
  longer object length is not a multiple of shorter object length
```

Vektorilla voidaan myös operoida skalaarin kanssa. Myös tällöin laskuoperaatiot tapahtuvat alkioittain.

Esimerkki 2.14.

```
> a

[1] 1 2 3 4 5 6 7 8

> a+3

[1]  4  5  6  7  8  9 10 11

> 3*a

[1]  3  6  9 12 15 18 21 24

> a^2

[1]  1  4  9 16 25 36 49 64
```

Skalaarimuuttujien funktiot toimivat vektoreilla niin ikään alkioittain.

Esimerkki 2.15.

```
> a

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

```
> log(a)

[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101 2.0794415

> a%%3

[1] 1 2 0 1 2 0 1 2

> cos(a)

[1]  0.5403023 -0.4161468 -0.9899925 -0.6536436  0.2836622  0.9601703  0.7539023
-0.1455000
```

Vektoreille on myös omia hyödyllisiä funktioita. Aiemmin jo nähtiin, että funktio `length()` antaa vektorin pituuden, eli alkioiden lukumäärän. Funktiolla `sum()` voidaan laskea vektorin arvot yhteen. Myös totuusarvovektoriin voidaan soveltaa funktiota `sum()` (harjoitustehtävä). Funktio `prod()` kertoo kaikki vektorin arvot keskenään.

Esimerkki 2.16.

```
> x <- c(3,7,2,7)
> length(x)

[1] 4

> sum(x)

[1] 19

> prod(x)

[1] 294

> max(x)

[1] 7
```

2.1.4 Lisää merkkijonovektoreista

Merkkijonoja voidaan liittää toisiinsa `paste`-funktiolla. Funktio palauttaa merkkijonovektorin, jossa argumentteina annettujen vektorien alkiot on liitetty toisiinsa. Funktio käyttää oletuksena merkkijonojen erottimena välilyöntiä, mutta erottimen voi vaihtaa `sep`-argumentilla. Funktio `paste0` liittää merkkijonot toisiinsa ilman erotinta. Liittäminen tapahtuu alkioittain, eli jos vektorit ovat samanpituiset, vastaavien alkioiden merkkijonot liitetään toisiinsa.

Esimerkki 2.17. Sovelletaan `paste`-funktiota samanpituisiin merkkijonovektoreihin.

```
> x <- c("a","e","i","o")
> y <- c("c","u","b","j")
> paste(x,y)

[1] "a c" "e u" "i b" "o j"

> paste0(x,y)

[1] "ac" "eu" "ib" "oj"

> paste(x,y,sep="-")

[1] "a-c" "e-u" "i-b" "o-j"
```

Esimerkki 2.18. Liitetään sama merkkijono jokaiseen merkkijonovektorin `x` alkioon.

```
> paste(x,1,sep="-")

[1] "a-1" "e-1" "i-1" "o-1"
```

R:ssä on valmiina esimerkiksi merkkijonovektorit `letters` (kirjaimet a-z), `month.name` (kuukaudet englanniksi) ja `month.abb` (kuukausien lyhenteet englanniksi.

Merkkijonoista voidaan poimia osamerkkijonoja funktion `substr` avulla. Funktio ottaa argumentteinaan käsiteltävän merkkijonovektorin sekä kaksi kokonaislukua, jotka määräävät, mistä kohtaa merkkijonoa leikataan.

Esimerkki 2.19.

```
> substr(month.name,start=1,stop=4)

 [1] "Janu" "Febr" "Marc" "Apri" "May"  "June" "July" "Augu" "Sept" "Octo"
"Nove" "Dece"
```

Funktion `substr` avulla voidaan myös korvata osa merkkijonosta toisella merkkijonolla

Esimerkki 2.20.

```
> substr(month.name,1,2)

 [1] "Ja" "Fe" "Ma" "Ap" "Ma" "Ju" "Ju" "Au" "Se" "Oc" "No" "De"

> substr(month.name,1,2) <- letters[1:12]
> month.name

 [1] "aanuary"   "bebruary"  "carch"     "dpril"     "eay"       "fune"
 "guly"      "hugust"    "ieptember" "jctober"   "kovember"  "lecember"
```

2.2 Matriisit

2.2.1 Matriisien luominen

R:ssä matriisi luodaan vektorista `matrix()` -komennolla. Matriisin alkiot sisältävän vektorin lisäksi funktiolle tulee kertoa haluttu rivien ja sarakkeiden määrä. Funktio tarvitsee myös tiedon siitä, onko annettava data järjestetty riveittäin (byrow) vai sarakkeittain.

Esimerkki 2.21. Luodaan matriisi

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

käyttäen komentoa `matrix()`

```
> matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3, byrow=TRUE)

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

Tulosteessa näkyvät rivien ja sarakkeiden alussa olevat hakasuluissa olevat luvut kuvaavat indeksiä, eli rivin tai sarakkeen järjestyslukua.

Vaihtoehtoisesti sama matriisi voidaan luoda sarakkeittain järjestetyllä vektorilla:

```
> matrix(c(1, 4, 7, 2, 5, 8, 3, 6, 9), nrow=3, ncol=3, byrow=FALSE)
```

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

Edellisessä siis yhdistettiin luvut 1-9 komennolla `c()` vektoriksi ja kutsuttiin komentoa `matrix()`. Argumentteina funktiolle `matrix()` annettiin matriisin dimensiot 3x3 ja tieto, että data on järjestetty riveittäin. `TRUE` voidaan myös lyhentää `T`, kuten jatkossa usein tehdään. Vastaavasti argumentin arvoksi voidaan antaa `FALSE` tai `F`.

Erilaisia diagonaalimatriiseja voidaan luoda kätevästi funktion `diag()` avulla. Sen ensimmäiseksi argumentiksi asetetaan halutut diagonaalialkiot ja toisena argumenttina määrätään haluttu rivien määrä. Halutessaan funktiolle voi myös asettaa toivotun sarakkaiden lukumäärän.

Esimerkki 2.22. Luodaan erilaisia diagonaalimatriiseja

```
> diag(1, nrow=3)

     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1

> diag(c(2, 3), nrow=2)

     [,1] [,2]
[1,]    2    0
[2,]    0    3
```

2.2.2 Matriisien käsittely

Matriiseja voidaan käsitellä kuten vektoreitakin: hakasulkuja käyttämällä voidaan valita alkioita ja tehdä vertailuja. Matriisin alkion sijainnin määräävät sen rivi ja sarake yhdessä. Alkioon viitatessa tulee haluttu rivi ja sarake erottaa hakasulkujen sisällä toisistaan pilkulla siten, että rivi ilmoitetetaan pilkun vasemmalla ja sarake oikealla puolella.

Matriiseista voidaan yksittäisten solujen lisäksi valita myös kokonaisia rivejä tai sarakkeita. Riviin voidaan viitata ilmoittamalla haluttu rivi hakasulkujen sisällä pilkun vasemmalla puolella, mutta jättämällä pilkun oikea puoli tyhjäksi. Vastaavasti sarakkeeseen

viitataan ilmoittamalla haluttu sarake pilkun oikealla puolella ja jättämällä pilkun vasen puoli tyhjäksi.

Jos matriisista valitsee vain yhden rivin, muuntaa R sen automaattisesti yksinkertaisimmaksi mahdolliseksi tietorakenteeksi eli vektoriksi. Mikäli haluaa säilyttää muuttujan tietorakenteen matriisina, tulee osa-aineistoa valitessa asettaa hakasulkujen sisään argumentiksi `drop=FALSE`.[3]

Esimerkki 2.23. Solujen, rivien ja sarakkeiden valitseminen

```
> a <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3, byrow=TRUE)
> a

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> a[,1]

[1] 1 4 7

> a[,1, drop=FALSE]

     [,1]
[1,]    1
[2,]    4
[3,]    7

> a[1,]

[1] 1 2 3

> a[1, , drop=FALSE]

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

[3]R yksinkertaistaa osa-aineistoja myös muiden tietorakenteiden tapauksissa, ja myös muiden tietorakenteiden tapauksissa tämä voidaan välttää argumentilla `drop=FALSE`.

```
> a[1, 2]

[1] 2

> a[a>5]

[1] 7 8 6 9
```

Esimerkki 2.24. Arvojen sijoittaminen matriisiin

```
> a <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3, byrow=TRUE)
> a[2, 1] <- 9001
> a

     [,1] [,2] [,3]
[1,]    1    2    3
[2,] 9001    5    6
[3,]    7    8    9

> a[,1] <- c(1, 2, 3)
> a

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    5    6
[3,]    3    8    9
```

Matriisiin voidaan lisätä rivejä ja sarakkeita käyttämällä komentoja `rbind()` ja `cbind()`

Esimerkki 2.25. Rivien ja sarakkaiden lisääminen

```
> a <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3, byrow=TRUE)
> rbind(a, c(1, 1, 1))

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

```
> cbind(rbind(a, c(1, 1, 1)), c(2, 2, 2, 2))

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

Rivien ja sarakkeiden poisto onnistuu helposti käyttämällä totuusarvoja vektorioperaatioiden tapaan.

Esimerkki 2.26. Rivien ja sarakkaiden poistaminen

```
> a <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3, byrow=TRUE)
> a[,c(F, T, T)]

     [,1] [,2]
[1,]    2    3
[2,]    5    6
[3,]    8    9

> a[c(T, T, F),]

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

2.2.3 Vektorien ja matriisien laskutoimitukset

Matriiseille saadaan laskettua matriisien tulo operaattorin `%*%` avulla. Huomaa, että operaatio toimii vain, mikäli matriisitulo on määritelty, eli mikäli toisessa matriisissa on yhtä monta riviä kuin ensimmäisessä matriisissa on sarakkeita. Pelkällä kertomerkillä R laskee tavanomaiset kertolaskut samadimensioisten matriisien (tai vektoreiden) alkioiden välillä. Matriisien yhteen- ja vähennyslaskut toimivat tavallisesti käyttäen operaattoreita `+` ja `-`. Transpoosi saadaan komennolla `t()`, jolle argumentiksi annetaan transponoitava matriisi.

Esimerkki 2.27. Merkitään

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

ja

$$B := \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}.$$

Lasketaan matriisitulo AB:

```
> A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3, byrow=T)
> B <- matrix(c(9, 8, 7, 6, 5, 4, 3, 2, 1), nrow=3, ncol=3, byrow=T)
> A%*%B

     [,1] [,2] [,3]
[1,]   30   24   18
[2,]   84   69   54
[3,]  138  114   90
```

Esimerkki 2.28. Transponoidaan edellisen esimerkin matriisi A:

```
> t(A)

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

Esimerkki 2.29. Vektorien ja matriisien tulo onnistuu täysin samalla tavalla:

```
> a<-c(0, 1, 2, 3)
> a %*% diag(1, nrow=4)

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

Matriisin voi kääntää funktiolla `solve()` ja ominaisarvot ja -vektorit voi laskea funktiolla `eigen()`.

Esimerkki 2.30. Lasketaan erään diagonaalimatriisin käänteismatriisi ja ominaisarvot sekä -vektorit:

```
> a <- diag(1:3, nrow=3)
> solve(a)

     [,1] [,2]      [,3]
[1,]    1  0.0 0.0000000
[2,]    0  0.5 0.0000000
[3,]    0  0.0 0.3333333

> eigen(a)

$values
[1] 3 2 1

$vectors
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
[3,]    1    0    0
```

Esimerkki 2.31. Esimerkki matriisien soveltamisesta lineaarisen regressiomallin estimoinnissa. Tutkitaan aineistoa

x	y
4	15
6	14
7	12
11	11
14	10
21	8
32	6

ja oletetaan, että

$$y_i = \alpha + \beta x_i + \varepsilon_i,$$

jollain $\alpha, \beta \in \mathbb{R}$, kun $\varepsilon_i \sim N(0, \sigma^2)$ kaikilla i. Sovitetaan nyt aineistoon suora käyttäen kaavaa[4]

$$(\hat{\beta}, \hat{\alpha}) = (X'X)^{-1}X'y,$$

[4]Todellisuudessa lineaarisen mallin estimoinnissa mallimatriisiin X sovelletaan usein QR-hajotelmaa, koska matriisin $X'X$ suora kääntäminen voi johtaa numeerisiin epätarkkuuksiin.

missä

$$X = \begin{bmatrix} 4 & 1 \\ 6 & 1 \\ 7 & 1 \\ 11 & 1 \\ 14 & 1 \\ 21 & 1 \\ 32 & 1 \end{bmatrix}$$

ja

$$y = \begin{bmatrix} 15 & 14 & 12 & 11 & 10 & 8 & 6 \end{bmatrix}'.$$

```
> X <- matrix(c(4, 6, 7, 11, 14, 21, 32, rep(1, 7)), ncol=2)
> Y <- matrix(c(15, 14, 12, 11, 10, 8, 6), ncol=1)
> solve(t(X)%*%X)%*%t(X)%*%Y

           [,1]
[1,] -0.3072666
[2,] 15.0271896
```

Nyt aineistoon sovitettu suora on siis

$$y(x) \approx -0.307x + 15.027.$$

2.3 Moniulotteiset matriisit eli arrayt

Tavanomaisten kaksiulotteisten matriisien lisäksi R:ssä on mahdollista käsitellä myös ns. moniulotteisia matriiseja eli ”array”:ta. Kun matriisin ensimmäisenä dimensiona voidaan ajatella olevan rivien määrä ja toisena sararakkeiden määrä, voidaan arraylle näiden lisäksi asettaa haluttu määrä lisädimensioita. Esimerkiksi kolmiulotteisen arrayn voi ajatella koostuvan peräkkäisistä tai päällekäisistä matriiseista, ja kaksiulotteinen array on vain tavanomainen matriisi. Neliulotteisen arrayn voi puolestaan ajatella eräänlaiseksi kokoelmaksi kolmiulotteisia arrayta, jne.

2.3.1 Arrayn luominen

Matriisin tapaan array luodaan vektorista, mutta `matrix()`-komennon sijasta käytetään komentoa `array()`. Arrayn alkiot sisältävän vektorin lisäksi funktio tarvitsee tiedon ha-

lutuista dimensioista, eli esimerkiksi kolmiulotteisen arrayn tapauksessa rivien, sarakkeiden ja ”päällekäisten matriisien” määrän. Arrayn alkiot täytetään annetusta vektorista dimensioittain järjestyksessä.

Esimerkki 2.32. Luodaan kolmiulotteinen array, jossa on neljä riviä, kolme saraketta ja kaksi ”matriisia päällekäin”:

```
> a <- array(1:24, dim=c(4, 3, 2))
> a

, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

2.3.2 Arrayn käsittely

Arrayn alkioihin viittaaminen tai niiden valitseminen tapahtuu hakasulkujen avulla samalla tavalla kuin matriiseillakin. Hakasulkujen sisällä eri dimensiot erotellaan pilkuilla siten, että laitimmainen elementti vasemmalla viittaa riveihin, seuraava oikealla sarakkeisiin, siitä seuraava kolmanteen dimensioon jne. Kokonaisiin dimensioihin voidaan viitata tai niitä voidaan valita samoin kuin matriiseilla kokonaisia rivejä tai sarakkeita, eli määrämällä hakasulkujen sisällä haluttu dimensio sitä vastaavassa elementissä ja jättämällä muiden dimensioiden kohdalle pelkät pilkut.

Huomaa, että esimerkiksi kolmiulotteisen arrayn tapauksessa mihin tahansa yksittäiseen dimensioon viittaminen tuottaa aina (kaksiulotteisen) matriisin, koska kyse on kolmiulotteisesta rakenteesta. Mikäli yksittäisiin dimensioihin viittaessa haluaa kuitenkin säilyttää tietorakenteen alkuperäisdimensioisena, tulee käyttää argumenttia `drop=FALSE`.

Esimerkki 2.33. Valitaan kolmiulotteisen arrayn alkioita ja dimensioita:

```
> a[4, 2, 1]

[1] 8

> a[1, 1, 2]

[1] 13

> a[, 3, 2]

[1] 21 22 23 24

> a[, , 2]

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24

> a[1, , ]

     [,1] [,2]
[1,]    1   13
[2,]    5   17
[3,]    9   21

> a[1, , , drop=FALSE]

, , 1

     [,1] [,2] [,3]
[1,]    1    5    9

, , 2

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

```
[1,]   13   17   21
```

Esimerkki 2.34. Sijoitetaan arvoja kolmiulotteiseen arrayhyn:

```
> a[1, , ] <- matrix(101:106, nrow=3, ncol=2)
> a[4, 2, 1] <- 42
> a

, , 1

     [,1] [,2] [,3]
[1,]  101  102  103
[2,]    2    6   10
[3,]    3    7   11
[4,]    4   42   12

, , 2

     [,1] [,2] [,3]
[1,]  104  105  106
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

2.3.3 Arrayn hyödyntäminen

Mitään varsinaista yli kaksiulotteisten ”matriisien” algebraa eivät R:n peruslaskutoimitukset tue. Sen sijaan motiivina arrayn käyttöön on usein sen laskennallinen tehokkuus ja kätevyys datan säilömisessä ja käsittelemisessä. Arrayn sijasta samadimensioisia matriiseja voi tuntua houkuttelevalta säilöä esimerkiksi kappaleessa 3.1 tarkasteltavaan listaan, mutta lista on tietorakenteena arrayhin verrattuna raskas, ja sen operoiminen on hidasta.

Mikäli arraysta valitsee dimension tai dimensiot sopivasti siten, että tuloksena on matriisi, voidaan valittuun dimensioon soveltaa tavanomaista matriisilaskentaa.

Esimerkki 2.35. Matriisikertolaskuja kolmiulotteisen arrayn dimensioille

```
> b <- array(1:8, dim=c(2, 2, 2))
> b
```

```
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

> c <- diag(1:2, nrow=2)
> c

     [,1] [,2]
[1,]    1    0
[2,]    0    2

> b[, , 2]%*%c

     [,1] [,2]
[1,]    5   14
[2,]    6   16

> b[1, , ]%*%c

     [,1] [,2]
[1,]    1   10
[2,]    3   14

> b[, 1, ]%*%c

     [,1] [,2]
[1,]    1   10
[2,]    2   12
```

2.4 Keskeisiä operaattoreita ja funktioita

Funktioiden toimintaan ja funktioille annettaviin argumentteihin voit tutustua tarkemmin R:n ohjekirjasta.
Vektorien yhdistäminen: `c()`
Säännöllisen jonovektorin luominen: `seq()`
Toinen tapa luoda säännöllinen jonovektori: Esim. `1:10`
Vektorin alkioiden määrä: `length()`

Merkkijonojen liittäminen: `paste()`, `paste0()`
Osamerkkijonon poiminta merkkijonosta: `substr()`
Uppercase: `toupper()`
Lowercase: `tolower()`

Ehdot täyttävien alkioiden valinta vektorista `x`: `x[LOOGINEN EHTO]`
Toinen tapa valita ehdot täyttäviä alkioita: `subset(x, LOOGINEN EHTO)`
Ehdot täyttävien indeksien valinta: `which(LOOGINEN EHTO)`

Matriisin luominen: `matrix()`
Diagonaalimatriisin luominen: `diag()`
Rivien yhdistäminen: `rbind()`
Sarakkeiden yhdistäminen: `cbind()`
Rivien määrä: `nrow()`
Sarakkeiden määrä: `ncol()`
Dimensio (rivit ja sarakkeet): `dim()`

Ehdot täyttävien rivien valinta matriisista `X`: `X[LOOGINEN EHTO,]`
Ehdot täyttävien sarakkeiden valinta matriisista: `X[,LOOGINEN EHTO]`
Ehdot täyttävien alkioiden valinta matriisista: `X[RIVIEHTO, SARAKE-EHTO]`

Matriisien `X` ja `Y` matriisitulo: `X%*%Y`
Transpoosi: `t()`
Käänteismatriisi: `solve()`
Ominaisarvot ja -vektorit: `eigen()`

Kappale 3

Ei-atomiset tietorakenteet

Ei-atomisilla tietorakenteilla tarkoitetaan sellaisia datan säilömismuotoja, jotka voivat käsittää eri tietotyyppien alkioita. Yksi yleisesti data-analyysissä käytetty ei-atominen tietorakenne on lista (joka on samalla myös tietotyyppi), jonka elementteihin voi säiliöä kaikenlaisia muuttujia sekaisin, esimerkiksi merkkijonoja, vektoreita, matriiseja ja toisia listoja.

Toinen yleisesti käytetty ei-atominen tietorakenne on taulukko (data frame), joka on todellisuudessa vain lista samanpituisista vierekkäin järjestetyistä (tyypillisesti atomisista[1]) vektoreista. Kaksiulotteisen rakenteensa vuoksi taulukolla on listan ominaisuuksien lisäksi myös monia matriisin ominaisuuksia. Kyseessä on siitä erityinen tietorakenne, että data luetaan R:ssä tyypillisesti taulukoksi (ks. kappale 4.2).

3.1 Lista

3.1.1 Listan luominen

Lista luodaan funktiolla `list()` siten, että argumentteina sille listataan pilkulla erotellen alkiot, jotka sen halutaan sisältävän. Tyhjän listan voi luoda jättämällä argumentit tyhjäksi.

Esimerkki 3.1. Luodaan lista, joka sisältää merkkijonon, skalaarin (eli yksialkioisen numeerisen vektorin), numeerisen vektorin ja totuusarvovektorin.

```
> lista <- list("moi", 3.14159, 1:10, c(TRUE, TRUE, FALSE))
> lista
```

[1] taulukon sarakkeeksi voi halutessaan asettaa myös ei-atomisen vektorin eli listan, mutta tämä on harvinaisempaa, ja se ohitetaan tässä monisteessa.

```
[[1]]
[1] "moi"

[[2]]
[1] 3.14159

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

[[4]]
[1]  TRUE  TRUE FALSE
```

3.1.2 Listan komponenttien valitseminen

Listasta voidaan valita alilistoja hakasulkujen avulla samalla tavalla kuin vektorista alkioita.

Esimerkki 3.2.

```
> lista[1]

[[1]]
[1] "moi"

> lista[1:3]

[[1]]
[1] "moi"

[[2]]
[1] 3.14159

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

Huomaa, että alilistat ovat edelleen listoja. Esimerkiksi kun valitaan listan kolmas komponentti ja tallennetaan se muuttujaan `kolmas`, tuloksena on yhden pituinen lista, jonka ainoa komponentti on alkuperäinen vektori.

Esimerkki 3.3.

```
> kolmas <- lista[3]
> kolmas

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

> typeof(kolmas)

[1] "list"
```

Jos halutaan päästä käsiksi suoraan listan komponentteihin, kirjoitetaan komponentin numero tuplahakasulkuihin, jolloin saadaan kyseinen komponentti sen alkuperäisessä muodossa.

Esimerkki 3.4.

```
> kolmas <- lista[[3]]
> kolmas

 [1]  1  2  3  4  5  6  7  8  9 10
> typeof(kolmas)

[1] "integer"

> kolmas[1]

[1] 1
```

3.1.3 Listan komponenttien nimeäminen

Listan komponentit voi ja on monesti myös suositeltavaa nimetä, mikä helpottaa niihin viittamista. Luodaan nimetty lista, johon talletetaan tietoja opiskelijasta.

Esimerkki 3.5.

```
> opiskelija <- list(nimi="Fisher", opiskelijanro=1234567, aloitusvuosi=2018,
+                    kurssit=c(57045, 57046, 57703))
> opiskelija

$nimi
[1] "Fisher"
```

```
$opiskelijanro
[1] 1234567

$aloitusvuosi
[1] 2018

$kurssit
[1] 57045 57046 57703
```

Nimetyn listan komponentteihin voidaan viitata $-operaattorilla kirjoittamalla sen vasemmalle puolelle halutun listan nimi ja oikealle puolelle halutun komponentin nimi. Nimella viittaaminen antaa komponentin alkuperäisessä muodossaan samalla tavalla kuin tuplahakasulut. Nimellä viittaaminen on monesti suositeltavampaa kuin järjestysnumerolla viittaminen, sillä se toimii edelleen, vaikka listaan lisäisi tai poistaisi komponentteja, tai vaihtaisi niiden järjestystä.

Esimerkki 3.6.

```
> opiskelija$nimi

[1] "Fisher"

> opiskelija$kurssit[2]

[1] 57046
```

3.1.4 Listan komponenttien lisääminen, muuttaminen ja poistaminen

Listaan voi lisätä komponentteja osoittamalla käsiteltävästä listasta sen järjestysnumeron tai sen nimen (jonka ei tarvitse olla vielä olemassa), mihin alkioon uusi komponentti halutaan asettaa, ja asettamalla halutun komponentin sijoitusoperaattorin avulla.

Listan komponentteja voidaan vastaavasti muuttaa asettamalla sijoitusoperaattorin avulla halutuksi komponentiksi sen muutettu versio.

Esimerkki 3.7. Lisätään Fisherin suorittamiin kursseihin kurssi 57798, ja lisätään uudet `paa_aine`- sekä `sivuaine`-komponentit:

```
> opiskelija$kurssit[4] <- 57798
> opiskelija$paa_aine <- "tilastotiede"
> opiskelija$sivuaine <- "matematiikka"
```

```
> opiskelija

$nimi
[1] "Fisher"

$opiskelijanro
[1] 1234567

$aloitusvuosi
[1] 2018

$kurssit
[1] 57045 57046 57703 57798

$paa_aine
[1] "tilastotiede"

$sivuaine
[1] "matematiikka"
```

Listasta voidaan poistaa komponentteja asettamalla niiden arvoksi `NULL` sijoitusoperaattorin avulla.

Esimerkki 3.8. Poistetaan `sivuaine`-komponentti sijoittamalla `NULL`:

```
> opiskelija$sivuaine <- NULL
> opiskelija

$nimi
[1] "Fisher"

$opiskelijanro
[1] 1234567

$aloitusvuosi
[1] 2018

$kurssit
[1] 57045 57046 57703 57798
```

```
$paa_aine
[1] "tilastotiede"
```

3.2 Taulukko

Taulukko eli data frame on aiemmin todetun mukaisesti kaksiulotteisen rakenteen omaava lista, jossa samanpituisia (tyypillisesti) atomisia vektoreita on järjestetty vierekkäin sarakkeiksi. Taulukon kukin sarake vastaa siis kutakin taustalla olevan listan alkiota. Taulukkoon voi näin ollen säiliöä eri sarakkeisiin eri tietotyypin alkioita, mutta kunkin sarakkeen sisällä alkioiden tietotyypin tulee olla sama (ellei taulukon sarakkeeksi aseta listaa). Data onkin tyypillisesti järjestetty taulukkoon siten, että kutakin muuttujaa vastaa yksi sarake, ja kutakin havaintoyksikköä (esimerkiksi henkilöä, jonka ominaisuuksia on ilmoitettu sarakkeissa) vastaa yksi rivi.

3.2.1 Taulukon luominen

Taulukko luodaan komennolla `data.frame()` samaan tapaan kuin lista tai nimetty lista. Sen argumentteina luetellaan pilkulla erotellen samanpituisia vektoreita, joista taulukon sarakkeet muodostetaan. Taulukossa sarakkeet on tapana nimetä, mutta myös sen riveille voi antaa nimet argumentin `row.names` avulla. Jos rivit jättää nimeämättä, ne nimetään automaattisesti juoksevalla numeroinnilla.

Esimerkki 3.9. Luodaan taulukko, jossa kuvitteellisiin henkilöihin liitetään tunnus, pisteet ja lempiruoka.

```
> rivinimet <- c("Pearson", "Gosset", "Fisher")
> tunnus <- c(42, 85, 12)
> pisteet <- c(100, 151, 122)
> lempiruoka <- c("Pizza", "Spagetti", "Gulashi")
> taulukko <- data.frame(tunnus=tunnus, pisteet=pisteet,
+                        lempiruoka=lempiruoka, row.names=rivinimet)
> taulukko

        tunnus pisteet lempiruoka
Pearson     42     100      Pizza
Gosset      85     151   Spagetti
Fisher      12     122    Gulashi

> typeof(taulukko)
```

```
[1] "list"

> class(taulukko)

[1] "data.frame"
```

Taulukkoa luodessa (ja erityisesti ladatessa dataa R:än) on syytä kiinnittää huomiota `data.frame()`-funktion argumenttiin `stringsAsFactors`, joka määrittää muutetaanko merkkijono-tyyppiset sarakkeet faktoreiksi (ks. kappale 4.1). Argumentin oletusarvo on `TRUE`, ellei sitä ole erikseen muutettu, joten halutessasi pitää merkkijonot merkkijonoina, tulee taulukkoa luodessa asetta funktion `data.frame()` argumenttina `stringsAsFactors=FALSE`.

Funktion `str()` avulla voidaan tarkastella taulukon (ja myös muiden olioiden) rakennetta. Alla esitetyssä esimerkissä nähdään, että edellisessä esimerkissä luodun taulukon ”lempiruoka”-sarake on luokaltaan faktori.

Esimerkki 3.10. Tarkastellaan edellisessä esimerkissä luodun taulukon rakennetta

```
> str(taulukko)

'data.frame': 3 obs. of  3 variables:
 $ tunnus    : num  42 85 12
 $ pisteet   : num  100 151 122
 $ lempiruoka: Factor w/ 3 levels "Gulashi","Pizza",..: 2 3 1
```

Suuria taulukoita on usein taulukon tulostamisen sijasta kätevämpää tarkastella `View()`-funktion avulla.

Esimerkki 3.11.

```
> View(taulukko)
```

Tämä komento avaa taulukon RStudiossa taulukkolaskentatyyppiselle välilehdelle.

3.2.2 Rivien ja sarakkeiden lisääminen taulukkoon

Uuden nimetyn sarakkeen lisääminen taulukkoon onnistuu helpoiten `$`-operaattorin ja sijoituksen avulla.

Esimerkki 3.12.

```
> taulukko$kengannumero <- c(42,41,45)
> taulukko
```

```
        tunnus pisteet lempiruoka kengannumero
Pearson     42     100      Pizza           42
Gosset      85     151   Spagetti           41
Fisher      12     122    Gulashi           45
```

Jos taulukossa on jo samanniminen sarake, se korvautuu automaattisesti uudella sarakkeella.

```
> taulukko$kengannumero <- c(NA,NA,45)
> taulukko

        tunnus pisteet lempiruoka kengannumero
Pearson     42     100      Pizza           NA
Gosset      85     151   Spagetti           NA
Fisher      12     122    Gulashi           45
```

Uuden rivin lisääminen voidaan tehdä esimerkiksi funktiolla `rbind`. On kuitenkin syytä huomata, että tällöin lisättävän rivin tulee olla myös taulukko, jolla on samannimiset sarakkeet kuin taulukolla, johon rivi lisätään.

Esimerkki 3.13.

```
> new <- data.frame(tunnus=21,pisteet=140,lempiruoka="Pizza",kengannumero=44,
row.names="Bayes")
> new

      tunnus pisteet lempiruoka kengannumero
Bayes     21     140      Pizza           44

> rbind(taulukko,new)

        tunnus pisteet lempiruoka kengannumero
Pearson     42     100      Pizza           NA
Gosset      85     151   Spagetti           NA
Fisher      12     122    Gulashi           45
Bayes       21     140      Pizza           44
```

3.2.3 Osa-aineistojen valitseminen taulukosta

Taulukosta voidaan valita osa-aineistoja, komponentteja ja alkioita ainakin kolmella eri tavalla:

- Samalla tavalla kuin listasta kohtelemalla taulukon sarakkeita kuin listan komponentteja.
- Samalla tavalla kuin matriisista kohtelemalla taulukon sarakkeita kuin matriisin sarakkeita ja taulukon rivejä kuin matriisin rivejä.
- Käyttämällä osa-aineistojen valintaan tarkoitettua funktiota, kuten `subset()`.

Se mitä tapaa kannattaa milloinkin käyttää on tilannekohtaista. Esimerkiksi yksittäisten sarakkeiden poimiminen sarakkeen nimeen viitaten voi monesti olla kätevämpää ja tuntua luontevammalta, kuin ensin selvittää halutun sarakkeen järjestysnumero ja käyttää sitten sitä viitaten sarakkeeseen kuin matriisin sarakkeeseen. Toisaalta jos taulukon sarakkeita halutaan käydä läpi jonkin silmukan (kuten for-loopin) avulla, tai jos sarakkeiden nimet eivät ole merkityksellisiä, voi järjestysnumeroiden käyttäminen tuntua luontevammalta.

Taulukon rivejä valitaan tyypillisesti järjestysnumeroiden tai ehtolauseiden avulla. Rivien valitseminen on tietysti mahdollista tehdä myös rivinimiin viitaten, mutta tämän asian käsittely sivuutetaan.[2]

Jos osa-aineiston muodostamisessa tarvitaan useita erilaisia ehtoja ja rajauksia, on puolestaan funktion `subset()` käyttö usein kätevintä ja koodin luettavuuden kannalta suotavaa. Käytössä `subset()` vain ”wrappaa” listojen ja atomisten tietorakenteiden osa-aineistojen valintaan käytettävät tekniikat käyttäjäystävälliseen muotoon, ja sitä voidaan käyttää myös vektoreille ja matriiseille.

Esimerkki 3.14. Valitaan esimerkin 3.9 taulukosta sarakkeita ja alkioita kohtelemalla taulukkoa kuin listaa:

```
> taulukko[1]

        tunnus
Pearson     42
Gosset      85
Fisher      12

> taulukko[[2]]
```

[2]Myös atomisen matriisin rivejä ja sarakkeita voidaan nimetä ja niihin voidaan viitata, mutta tätäkään ei tässä käsitellä enempää.

```
[1] 100 151 122

> taulukko$lempiruoka

[1] Pizza    Spagetti Gulashi
Levels: Gulashi Pizza Spagetti

> taulukko$lempiruoka[3]

[1] Gulashi
Levels: Gulashi Pizza Spagetti

> taulukko$tunnus[1:2]

[1] 42 85
```

Esimerkki 3.15. Valitaan esimerkin 3.9 taulukosta rivejä, sarakkeita ja alkioita kohtelemalla taulukkoa kuin matriisia:

```
> taulukko[,1]

[1] 42 85 12

> taulukko[,3]

[1] Pizza    Spagetti Gulashi
Levels: Gulashi Pizza Spagetti

> taulukko[2,]

       tunnus pisteet lempiruoka
Gosset     85     151   Spagetti

> taulukko[2, 2]

[1] 151
```

Samaan tapaan kuin vektoreista ja matriiseista, voidaan myös taulukosta valita alkioita ja osa-aineistoja ehtolauseiden avulla. Monesti taulukosta halutaan valita sellaiset

havaintoyksiköt eli rivit, joilla jonkin tai joidenkin muuttujien osalta jokin ehto toteutuu. Tällöin haluttu ehto kirjoitetaan loogisten operaattoreiden avulla rivin indeksin paikalle (huomaa ehdon jälkeinen pilkku, joka erottaa rivin ja sarakkeen indeksin).

Esimerkki 3.16. Valitaan esimerkin 3.9 taulukosta ne rivit, joita vastaavien henkilöiden pistemäärä on suurempi kuin 100:

```
> taulukko$pisteet > 100

[1] FALSE  TRUE  TRUE

> taulukko[taulukko$pisteet > 100,]

       tunnus pisteet lempiruoka
Gosset     85     151   Spagetti
Fisher     12     122    Gulashi
```

Esimerkki 3.17. Valitaan esimerkin 3.9 taulukosta ne rivit, joita vastaavien henkilöiden pistemäärä on suurempi kuin 100 ja joiden lempiruoka on Gulashi:

```
> taulukko[taulukko$pisteet > 100 & taulukko$lempiruoka == "Gulashi",]

       tunnus pisteet lempiruoka
Fisher     12     122    Gulashi
```

Valitessa taulukosta tietyn ehdon täyttävät rivit, on tuloksena uusi taulukko, joka voidaan tallentaa ja jota voidaan käsitellä kuten mitä tahansa muutakin taulukkoa.

Esimerkki 3.18.

```
> taulukko2 <- taulukko[taulukko$pisteet > 100,]
> taulukko2$tunnus

[1] 85 12

> taulukko2[2, 3]

[1] Gulashi
Levels: Gulashi Pizza Spagetti
```

Kuten todettu, useita ehtoja ja rajauksia toteuttavan osa-aineiston valitseminen on usein kätevintä tehdä `subset()`-funktiolla. Ensimmäisenä argumenttina sille annetaan

taulukko, josta osa-aineisto valitaan, toisena argumenttina haluttu ehtolause loogisten operaattoreiden avulla ja kolmantena argumenttina valitaan halutut sarakkeet. Tuloksena on (oletusasetuksilla) uusi taulukko.

Esimerkki 3.19. Valitaan esimerkin 3.9 taulukosta osa-aineisto, joka sisältää täsmälleen sarakkeet ”tunnus” ja ”pisteet”, ja ne henkilöt, joiden pisteet on vähintään 130 tai lempiruoka on ”Pizza” tai ”Rusina”. Esitellään myös operaattori `%in%`, joka palauttaa totuusarvon `TRUE`, mikäli sen vasemmalle puolelle kirjoitettu alkio kuuluu sen oikealle puolelle kirjoitettuun vektoriin. Tässä tapauksessa siis mikäli lempiruoka on joko ”Pizza” tai ”Rusina”.

```
> subset(taulukko,
+        pisteet>=130 | lempiruoka %in% c("Pizza", "Rusina"),
+        select=c("tunnus", "pisteet"))

        tunnus pisteet
Pearson     42     100
Gosset      85     151
```

Taulukon sarakkeita voidaan poistaa samoin kuin listasta alkioita, eli asettamalla sarakkeen arvoksi `NULL`. Vaihtoehtoisesti voi valita osataulukon, joka ei sisällä poistettavaa saraketta.

Esimerkki 3.20. Poistetaan esimerkin 3.9 taulukosta ”lempiruoka-sarake asettamalla sen arvoksi `NULL`:

```
> taulukko$lempiruoka <- NULL
> taulukko

        tunnus pisteet
Pearson     42     100
Gosset      85     151
Fisher      12     122
```

Esimerkki 3.21. Muodostetaan edellisen esimerkin taulukosta (josta sarake ”lempiruoka” on poistettu) osataulukko, jossa on vain sarake nimeltä ”tunnus” kolmella eri tavalla: (1) Valitsemalla pelkästään haluttu sarake viittaamalla sen nimeen, (2) valitsemalla kaikki sarakkeet paitsi sarake nimeltä ”pisteet” joukko-operaattorin `setdiff()` avulla ja (3) valitsemalla haluttu osa-aineisto `subset()`-funktiolla:

```
> # (1)
```

```
> taulukko[c("tunnus")]

        tunnus
Pearson     42
Gosset      85
Fisher      12

> # (2)
> taulukko[setdiff(names(taulukko), "pisteet")]

        tunnus
Pearson     42
Gosset      85
Fisher      12

> # (3)
> subset(taulukko, select="tunnus")

        tunnus
Pearson     42
Gosset      85
Fisher      12
```

3.3 Keskeisiä operaattoreita ja funktioita

Funktioiden toimintaan ja funktioille annettaviin argumentteihin voit tutustua tarkemmin R:n ohjekirjasta.

Listan luominen nimetyillä komponenteilla: `list(nimi1 = ..., nimi2 = ...)`
Listan purkaminen atomiseksi/atomisiksi tietorakenteeksi `unlist()`
Taulukon luominen nimetyillä sarakkeilla: `data.frame(sarake1= ..., sarake2 = ...)`

Komponenttien tai sarakkeiden valinta nimien perusteella: `lista$komponentti`, `taulukko$sarake`
Sarakkeen tai komponentin lisääminen: `lista$komponentti <-`
`taulukko$sarake <-`

Ehdot täyttävien rivien valinta taulukosta: `taulukko[LOOGINEN EHTO,]`
Ehdot täyttävien sarakkeiden valinta taulukosta: `taulukko[,LOOGINEN EHTO]`

Toinen tapa: `subset(taulukko, select= SARAKKEET)`
Ehdot täyttävän osa-aineiston valinta taulukosta: `taulukko[RIVIEHTO,SARAKE-EHTO]`
Toinen tapa: `subset(taulukko, EHTO, select= SARAKKEET)`

Objektin rakenne: `str()`

Rivien nimeäminen: `row.names() <- c(...)`
Sarakkeiden nimeäminen: `colnames() <- c(...)`
Rivien määrä: `nrow()`
Sarakkeiden määrä: `ncol()`

Kappale 4

Faktorit ja aineiston lataaminen tiedostosta

4.1 Faktorit

Faktori (factor) on tietotyypin integer vektoreille määriteltävä luokka, joka on tarkoitettu täsmentämään, että kyseessä on luokittelu- tai järjestysasteikollinen muuttuja. Luokitteluasteikollisella muuttujalla tarkoitetaan muuttujaa, jonka arvoilla ei ole numeerista tulkintaa. Tällaisia ovat esimerkiksi sukupuoli ja kotikunta. Järjestysasteikollisella muuttujalla tarkoitetaan muuttujaa, jonka arvoilla on järjestys. Esimerkiksi mielipidemittauksen vastaukset asteikolla (1) Huono, (2) Ei huono eikä hyvä, (3) Hyvä, ovat siis järjestysasteikollinen muuttuja. Luokittelu- tai järjestysasteikollisen muuttujan täsmentäminen faktoriksi on tärkeää, koska silloin R osaa käsitellä sitä oikealla tavalla aineistoja analysoitaessa, esimerkiksi regressioanalyysissä.

4.1.1 Faktorin luominen ja sen tasot

Vektorin voi muuttaa faktoriksi käyttämällä komentoa `as.factor()` tai `factor()` ja asettamalla argumentiksi faktoriksi muutettavan vektorin. Jälkimmäistä funkiota käytettäessä faktorin tasot ja tasojen nimet voi määrätä argumenttien `levels` ja `labels` avulla. Muuttujan eri tasot voi nimetä ja uudelleenjärjestää jälkikäteen funktioiden `levels()` ja `relevel()` avulla.

Kun kokonaislukutyyppisen vektorin muuttaa faktoriksi, määräytyvät faktorin eri tasot eli luokittelu- tai järjestysmuuttujan asteikko oletusasetuksilla siten, että kutakin kokonaislukua vastaa yksi taso. Kokonaislukuvektorin lisäksi myös esimerkiksi merkkijonovektorin voi muuttaa faktoriksi. Tällöin tuloksena olevan luokittelumuuttujan tasoina toimivat vektorin sisältämät erilaiset merkkijonot, mutta taustalla oleva tietotyyppi

muuttuu kokonaisluvuksi.

Esimerkki 4.1. Luodaan vektori, johon on tallennettu sukupuoli merkkijonona, muutetaan se faktoriksi ja tarkastellaan sen rakennetta funktion `str()` avulla. Vaihdetaan sitten tasojen järjestystä siten, että "N" on ensimmäisenä. Luodaan lopuksi sama faktori, jossa "N" on järjestyksessä ensimmäisenä, suoraan funktion `factor()` avulla.

```
> sukupuoli <- c("N", "N", "M", "N", "M", "N")
> sukupuoli

[1] "N" "N" "M" "N" "M" "N"

> str(sukupuoli)

 chr [1:6] "N" "N" "M" "N" "M" "N"

> sukupuoli <- as.factor(sukupuoli)
> sukupuoli

[1] N N M N M N
Levels: M N

> str(sukupuoli)

 Factor w/ 2 levels "M","N": 2 2 1 2 1 2

> sukupuoli <- relevel(sukupuoli, ref="N")
> sukupuoli

[1] N N M N M N
Levels: N M

> str(sukupuoli)

 Factor w/ 2 levels "N","M": 1 1 2 1 2 1

> factor(c("N","N","M","N","M","N"), levels=c("N", "M"))

[1] N N M N M N
Levels: N M
```

Tarkasteltaessa faktoriksi muutettua muuttujaa `str()`-funktiolla huomataan, että sen arvot on koodattu uudelleen luvuiksi 1 ja 2; alkuperäiset arvot `M` ja `N` ovat faktorin tasoja.

Esimerkki 4.2. Luodaan vektori, johon on tallennettu kokonaislukuja, ja muutetaan se faktoriksi. Nimetään sitten tuloksena saadun luokittelumuuttujan tasot merkityksellisesti. Luodaan lopuksi sama faktori, jossa tasot on nimetty merkityksellisesti, suoraan funktion `factor()` avulla.

```
> lukuja <- c(1, 2, 1, 1, 3, 2)
> lukuja

[1] 1 2 1 1 3 2

> lukuja <- as.factor(lukuja)
> lukuja

[1] 1 2 1 1 3 2
Levels: 1 2 3

> levels(lukuja) <- c("Huono", "Hyvä", "En_tiedä")
> lukuja

[1] Huono    Hyvä     Huono    Huono    En_tiedä Hyvä
Levels: Huono Hyvä En_tiedä

> factor(c(1, 2, 1, 1, 3, 2), labels=c("Huono", "Hyvä", "En_tiedä"))

[1] Huono    Hyvä     Huono    Huono    En_tiedä Hyvä
Levels: Huono Hyvä En_tiedä
```

Funktiota `levels()` voi tasojen uudelleennimeämisen lisäksi käyttää myös faktorin tasojen tarkastelemiseen, muokkaamiseen ja tallentamiseen.

Esimerkki 4.3.

```
> levels(sukupuoli)

[1] "N" "M"

> levels(lukuja)

[1] "punainen"  "sininen"   "keltainen"
```

4.1.2 Käyttämättömien tasojen poistaminen ja lisääminen

Faktorit toimivat oletusarvoisesti niin, että ne säilyttävät jokaisen tason kuvaukset, vaikka muuttujasta rajattaisiin pois kaikki tiettyjen tasojen arvot. Tilanteesta riippuen tällaisia tyhjiä tasoja ei välttämättä haluta kuitenkaan säilyttää. Niiden poistaminen onnistuu funktiolla `droplevels()`, joka poistaa muuttujan jokaisen käyttämättömän tason. Vaihtoehtoisesti muuttujaa rajattaessa voi käyttää argumenttia `drop=TRUE`. Huomaa, että faktoreista voidaan valita osa-aineistoja ehtolauseiden avulla samalla tavalla kuin merkkijonovektoreista.

Esimerkki 4.4. Muodostetaan esimerkin 4.2 faktorista ”lukuja” osa-aineisto, jossa tasoa ”En_tiedä” osoittavat arvot on poistettu, ja poistetaan tuloksena olevasta faktorista käyttämättömät tasot.

```
> lukuja2 <- lukuja[lukuja!="En_tiedä"]
> lukuja2

[1] Huono Hyvä  Huono Huono Hyvä
Levels: Huono Hyvä En_tiedä

> lukuja2 <- droplevels(lukuja2)
> lukuja2

[1] Huono Hyvä  Huono Huono Hyvä
Levels: Huono Hyvä

> lukuja[lukuja!="En_tiedä", drop=TRUE]

[1] Huono Hyvä  Huono Huono Hyvä
Levels: Huono Hyvä
```

Faktoreille voidaan myös vastaavasti antaa kuvauksia tasoille, joihin kuuluvia arvoja ei kuitenkaan esiinny faktoriksi määritettävässä muuttujassa. Lisäksi osa muuttujassa esiintyvistä arvoista voidaan jättää kuvaamatta miksikään tasoksi. Edelliset onnistuvat määrämäällä kaikki mahdolliset tasot `levels()`-funktiolla valmiille faktorille, tai vaihtoehtoisesti faktoria luotaessa funktion `factor()` argumentilla `levels`. Tästä voi olla hyötyä tilanteissa, joissa samaa koodia halutaan esimerkiksi hyödyntää myöhemmin päivitetyllä aineistolla. Lisäksi sen avulla varmistetaan, että faktorin tasoihin kuulumattomat arvot merkitään puuttuviksi varsinaisessa faktori-muuttujassa.

Esimerkki 4.5. Lisätään edellisen esimerkin rajattuun `lukuja2`-muuttujaan taso ”En_tiedä”

takaisin:

```
> levels(lukuja2) <- c("Huono", "Hyvä", "En_tiedä")
> lukuja2

[1] Huono Hyvä  Huono Huono Hyvä
Levels: Huono Hyvä En_tiedä
```

Esimerkki 4.6. Määritellään esimerkin 4.2 kolmea eri lukua sisältävästä vektorista faktori, jossa on vain kaksi tasoa:

```
> factor(c(1, 2, 1, 1, 3, 2), levels=1:2, labels=c("Huono", "Hyvä"))

[1] Huono Hyvä  Huono Huono <NA>  Hyvä
Levels: Huono Hyvä
```

4.2 Aineiston lataaminen tiedostosta

Luettaessa tiedostoa R:n työhakemisto tulee ensin asettaa siihen hakemistoon, jossa tiedosto sijaitsee, tai vaihtoehtoisesti tiedosto voidaan lukea viittaamalla siihen sen koko polulla. Pieniä tiedostoja kannattaa monesti säilyttää samassa hakemistossa jonne R-koodi, jossa ne luetaan, tallennetaan, niin ne on helppo löytää. R:n nykyisen työhakemiston näkee funktiolla `getwd()` ja uuden työhakemiston pystyy asettamaan funktiolla `setwd()`. Huomaa kenoviivojen suunta, ne ovat toiseen suuntaan kuin Windows-järjestelmissä, joten hakemiston nimeä kopioitaessa ne pitää kääntää. Myös toisen ”vääränsuuntaisen” kenoviivan lisääminen ensimmäisen perään toimii ja voi olla helpompaa näppäillä.

Esimerkki 4.7.

```
> getwd()

[1] "C:/Users/Ville/Documents/R/win-library/3.0/muste"

> setwd("C:/Users/Ville/Desktop")
> getwd()

[1] "C:/Users/Ville/Desktop"

> setwd("C:\\Users\\Ville\\Desktop\\DA_R\\Vuosi_2017")
> getwd()
```

```
[1] "C:/Users/Ville/Desktop/DA_R/Vuosi_2017"
```

Taulukkomuotoisen aineistojen lataamista varten R:stä löytyvät `read.table()` ja siitä johdetut erikoistuneemmat funktiot. Monentyyppisten taulukoitten lukemiseen soveltuvan `read.table()`-funktion tärkeimpiä argumentteja ovat taulukon sisältävän tiedoston nimi (tiedoston on sijaittava työhakemistossa tai sitten nimen sijasta tulee antaa tiedoston polku), tiedostossa rivien, tekstin ja desimaalien erottamiseen käytetyt merkit sekä `header` ja `stringsAsFactors`. Näistä `header` määrittää, tulkitaanko taulukon ensimmäisen rivin arvot sarakkeiden nimiksi. `StringsAsFactors` vuorostaan kertoo, muutetaanko aineiston merkkijono-sarakkeet faktoreiksi.

Monesti aineisto tulee Excel-taulukkona. Tällöin se kannattaa avata Excelissä tai vastaavassa taulukkolaskentaohjelmassa ja tallentaa siitä uusi versio CSV-muodossa. CSV on lyhenne sanoista comma separated values; CSV-tiedostossa taulukko on tallennettu selväkielisenä ja taulukon sarakkeet on erotettu pilkulla tai muulla vastaavalla merkillä (tässä tapauksessa puolipisteellä).

CSV-tiedostojen lukemista varten on olemassa `read.table()`:a hieman kätevämpi funktio, `read.csv2()`. Se toimii aivan samalla tavalla kuin `read.table()`, mutta `stringAsFactors`:ia lukuunottamatta sen argumentit on jo oletuksena määritelty monille CSV-tiedostoille sopiviksi[1].

Alla esitetyssä esimerkissä ladataan esimerkkiaineisto kummallakin funktiolla ja tarkastetaan, että lataus onnistui tulostamalla kolme ensimmäistä riviä kymmenestä ensimmäisestä sarakkeesta. Taulukon tuhatta ensimmäistä riviä voi tarkastella Excel-tyyppisessä taulukossa klikkaamalla taulukon nimeä R-studiossa tai konsolissa `View()`-funktiolla. Sarakkeiden nimien tulkinnat löytyvät monesti aineiston mukana tulevasta koodikirjasta.

Esimerkki 4.8. Ladataan aineisto OT2014.csv R:ssä taulukoksi, jonka nimi on `ot1`. Käytetään funktiota `read.table`.

```
> ot1 <- read.table(file = "OT2014.csv", dec = ",",
+ sep = ";", header = T, quote = "\"",
+ stringsAsFactors = FALSE)
> ot1[1:3,1:10]

  fsd_no fsd_vr fsd_id kohdenro q1a q1b q1c q1_1 q1_2 q1_3
1   2978      1      1    10002   2   2  NA    3    1    2
2   2978      1      2    10003   2   2  NA    3    1    1
```

[1] Tällaisia ovat sellaiset tiedostot, joissa desimaaleja erotetaan pilkuilla ja sarakkeita puolipisteillä. Toisissa CSV-tiedostoissa vastaavat erottimet voivat olla myös piste desimaaleille ja pilkku sarakkeille. Funktio `read.csv()` soveltuu jälkimmäisten lataamiseen.

```
3   2978      1      3    10005   2   2  NA    3    1    2
```

Ladataan aineisto OT2014.csv R:ssä taulukoksi, jonka nimi on `ot2`. Käytetään funktiota `read.csv2`.

```
> ot2 <- read.csv2(file = "OT2014.csv", stringsAsFactors = FALSE)
> ot2[1:3,1:10]

  fsd_no fsd_vr fsd_id kohdenro q1a q1b q1c q1_1 q1_2 q1_3
1   2978      1      1    10002   2   2  NA    3    1    2
2   2978      1      2    10003   2   2  NA    3    1    1
3   2978      1      3    10005   2   2  NA    3    1    2
```

4.3 Aineiston kirjoittaminen tiedostoon

Taulukko tai matriisi voidaan myös viedä R:stä muualle, eli kirjoittaa se esimerkiksi tekstitiedostoksi tai Excel-taulukoksi. Kuten tiedoston lukemisessa, myös tiedostoon kirjoittaessa työhakemisto tulee ensin asettaa siihen hakemistoon, johon tiedosto halutaan kirjoittaa. Vaihtoehtoisesti voidaan kirjoittaa koko tiedostopolku viitatessa kirjoitettavan tiedoston nimeen.

Tiedostoon kirjoittamista varten R:stä löytyvät esimerkiksi funktiot `write.table()` ja `write.csv`. Kummallekin tulee antaa ensimmäisenä argumenttina `x` R-objekti (taulukko), joka kirjoitetaan tiedostoksi. Toisena argumenttina annetaan tiedoston nimi merkkijonona, eli lainausmerkkien sisällä. Lisäksi funktiolle voi antaa tarkentavia lisäargumentteja, joihin voi tutustua tarkemmin R:n ohjekirjassa komennolla `?write.table`.

Esimerkki 4.9. Tallennetaan aineisto `iris` tekstitiedostona työhakemistoon.

```
> setwd("D:/Datat")
> write.table(iris,"iris.txt")
```

Nyt työhakemistosta tulisi löytyä tekstitiedosto `iris.txt`.

4.4 Keskeisiä funktioita

Faktoriksi muuttaminen: `factor()`, `as.factor()`
Faktorin tasot: `levels()`

Työkansion asettaminen: `setwd("TIEDOSTOPOLKU")`
Työkansion selvittäminen: `getwd()`

Aineiston lukeminen tiedostosta: `read.table()`, `read.csv()`, `read.csv2()`, `read.delim()`

Kappale 5

Tilastolliset funktiot ja grafiikka

5.1 Tilastolliset funktiot

Tilastolliset funktiot käsittävät tässä erilaisten tunnuslukujen laskemiseen käytettävät sekä todennäköisyysjakaumia koskevat funktiot.

5.1.1 Tunnusluvut

R:ssä on valmiit funktiot yleisimpien tilastollisten tunnuslukujen, kuten summan (`sum()`), mediaanin (`median()`), keskiarvon (`mean()`), minimin (`min()`), maksimin (`max()`), keskihajonnan (`sd()`), varianssin (`var()`) ja kovarianssin (`cov()`), laskemiseen.

Edellä luetelluille funktiolle tulee ensimmäisenä argumenttina antaa olio tai olioita, eli esimerkiksi vektori tai matriisi, jonka tai joiden alkioista haluttu tunnusluku tai tunnusluvut lasketaan. Se minkälaisia olioita funktio osaa käsitellä, riippuu funktiosta, ja se mitä funktio palauttaa, riippuu sille syötetyistä olioista. Toisaalta argumentin ollessa vain numeerinen vektori, palauttaa kukin näistä tunnuslukufunktioista odotetulla tavalla lasketun tunnusluvun.

Esimerkki 5.1. Luodaan taulukko, joka sisältää tietoja kuvitteellisista henkilöistä ja lasketaan taulukon sarakkeille tunnuslukuja:

```
> taulukko <- data.frame(tunnus=c(42, 85, 12),
+                        pisteet=c(100, 151, 122),
+                        kasvis=c(FALSE, TRUE, FALSE),
+                        row.names=c("Pearson", "Gosset", "Fisher"))
> taulukko

        tunnus pisteet kasvis
```

```
Pearson     42     100  FALSE
Gosset      85     151   TRUE
Fisher      12     122  FALSE

> min(taulukko$tunnus)

[1] 12

> mean(taulukko$pisteet)

[1] 124.3333

> sd(taulukko$pisteet)

[1] 25.57994

> sum(taulukko$kasvis)

[1] 1
```

Muu tunnuslukufunktiolle annettava tärkeä (looginen) argumentti on `na.rm`, jolla määrätään, miten funktio toimii kohdatessaan puuttuvia arvoja. Oletusasetuksia käytettäessä tunnuslukufunktiot palauttavat puuttuvan arvon, jos vektori (tai muu olio) jonka alkioista tunnusluku lasketaan, sisältää yhdenkin puuttuvan arvon. Jos tunnusluku halutaan laskea kaikista paitsi puuttuvista arvoista, tulee funktion argumenteissa asettaa `na.rm=TRUE`. Komennolla `anyNA()` voi tarkistaa sisältääkö annettu vektori puuttuvia arvoja.

Esimerkki 5.2.

```
> a <- c(5:2, NA, 42)
> a

[1]  5  4  3  2 NA 42

> anyNA(a)

[1] TRUE

> median(a)

[1] NA
```

```
> median(a, na.rm=TRUE)

[1] 4
```

5.1.2 Jakaumafunktiot ja simulointi

Monien yleisten todennäköisyysjakaumien käsittelyyn on R:ssä olemassa valmiit funktiot. Jakaumafunktiot on nimetty kaksiosaisesti siten, että nimen ensimmäinen kirjain täsmentää onko kyseessä halutun jakauman tiheys- (`d`), kertymä- (`p`) vai kvantiilifunktio (`q`), tai halutaanko jakaumasta simuloida havaintoja (`r`). Ensimmäisen kirjaimen jälkeen tuleva osaa puolestaan täsmentää mistä jakaumasta on kyse.

Tuettuja jakaumia ovat muun muassa normaalijakauma (`norm`), t-jakauma (`t`), binomijakauma (`binom`) ja khi-toiseen jakauma (`chisq`). Täyden listan ja lisätietoja saa näkyviin komennolla `?Distributions`. Esimerkiksi normaalijakauman kertymäfunktion arvoja voidaan siis laskea kommennolla `pnorm()`, ja t-jakaumasta voidaan simuloida havaintoja komennolla `rt()`.

Seuraavissa esimerkeissä tutkitaan satunnaismuuttujia X ja Y, missä $X \sim \mathrm{N}(0, 1)$ ja $Y \sim \mathrm{Bin}(13, 1/3)$.

Esimerkki 5.3. Tiheys- ja pistetodennäköisyysfunktiot: Tiheys- tai pistetodennäköisyysfunktion $f(x)$ arvo kohdassa $x = 4$. Muista, että tiheysfunktiolla tämä ei ole todennäköisyys, mutta pistetodennäköisyysfunktiolla on!

```
# Normaalijakauma (tf)
> dnorm(x=4, mean=0, sd=1)

[1] 0.0001338302

# Binomijakauma (ptnf)
> dbinom(x=4, size=13, prob=1/3)

[1] 0.2296147
```

Näillä funktioilla voidaan laskea myös useampi pistetodennäköisyys- tai tiheysfunktion arvo antamalla argumentiksi `x` vektori. Tulos palautetaan myös vektorina.

Esimerkki 5.4.

```
> dbinom(x=c(4,5,6),size=13,prob=1/3)
```

```
[1] 0.2296147 0.2066532 0.1377688
```

Esimerkki 5.5. Kertymäfunktiot: Arvo kohdassa $q = 4$, eli vasen häntätodennäköisyys $P(X \leq 4)$ ja vasen häntätodennäköisyys $P(Y \leq 4)$. Argumentilla `lower.tail` voidaan täsmentää, jos halutaan laskea oikea häntätodennäköisyys. Oikealla häntätodennäköisyydellä tarkoitetaan tässä todennäköisyyttä, että satunnaismuuttuja on aidosti suurempaa kuin argumenttina annettu luku `q`. Huomaa, että jatkuvilla jakaumilla (esim. normaalijakauma) todennäköisyys on sama riippumatta siitä, onko yhtäsuuruus mukana vai ei.

```
# Normaalijakauma
# Vasen häntätodennäköisyys
> pnorm(q=4, mean=0, sd=1)

[1] 0.9999683

# Oikea häntätodennäköisyys
> pnorm(q=2,mean=0,sd=1,lower.tail=FALSE)

[1] 0.02275013

# Binomijakauma
> pbinom(q=4, size=13, prob=1/3)

[1] 0.5520387
```

Esimerkki 5.6. Kvantiilifunktiot: Haetaan piste, jonka vasemmalla puolella on 1/4 jakauman todennäköisyysmassasta.

```
# Normaalijakauma
> qnorm(mean=0, sd=1, p=1/4, lower.tail=TRUE)

[1] -0.6744898

# Binomijakauma
> qbinom(p=1/4, size=13, prob=1/3, lower.tail=TRUE)

[1] 3
```

Myös näillä funktioilla voidaan laskea useampi arvo samanaikaisesti antamalla argumentiksi `p` tai `q` vektori. Tulos on tällöin samanpituinen vektori (vrt. esimerkki 5.4).

Esimerkki 5.7. Jakauman simulointi, eli satunnaisotos halutusta jakaumasta. Nämä funktiot palauttavat satunnaiset luvut vektorina.

```
# Normaalijakauma
> rnorm(mean=0, sd=1, n=10)

 [1] -0.8876916 -1.3342456  0.2967970 -0.0250188  0.8236606  1.0947668 -0.3756786
 [8] -0.2220601 -1.2274948 -0.4169028

# Binomijakauma
> rbinom(size=13, prob=1/3, n=10)

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

5.2 Aineiston visuaalinen tarkastelu

5.2.1 Piirtofunktiot

Aineistoa voidaan R:ssä visualisoida monilla eri tavoilla. Alkeiden kannalta oleellisimmat funktiot ovat:

- `curve()`: Funktion kuvaaja
- `plot()`: Monikäyttöinen piirtofunktio
- `hist()`: Histogrammi
- `boxplot()`: Boxplot (laatikko ja viikset)

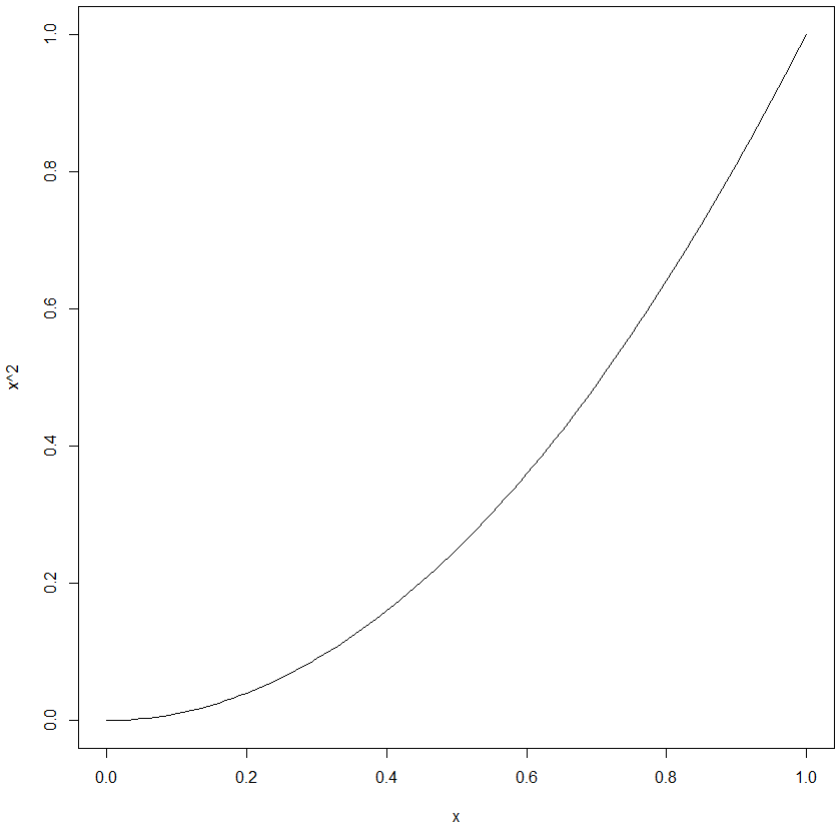
Edellä luetelluista funktioista, kuten muistakin funktioista, ja niiden käytöstä saa lisätietoja komennolla `?funktion_nimi`. Tutkitaan piirtofunktioiden käyttöä seuraavaksi esimerkein. Piirrä kuvat R:llä nähdäksesi miltä ne näyttävät.

Esimerkki 5.8. Piirretään funktion x^2 kuvaaja välillä $[0, 1]$ käyttäen `curve()`-funktiota:

```
> curve(x^2, from=0, to=1)
```

Esimerkki 5.9. Piirretään funktion x^2 kuvaaja välillä $[0, 1]$ käyttäen `plot()`-funktiota siten, että pisteparit (x, x^2), $x = 0.00, 0.01, ..., 0.99, 1.00$ yhdistetään toisiinsa viivoilla:

```
> x <- seq(0, 1, by=0.01)
> plot(x=x, y=x^2, type='l')
```



Kuva 5.1: Funktion x^2 kuvaaja välillä $[0, 1]$

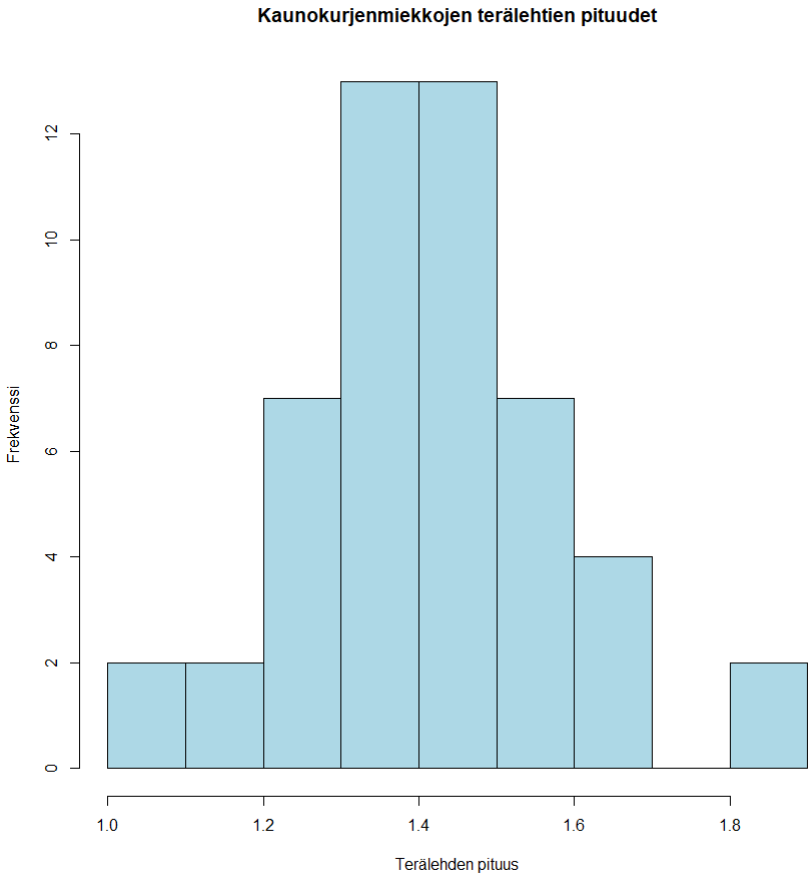
Huomaa, että tässä valinta `type='l'` käskee piirtämään kuvaajan viivoina. Oletusasetusta `type='p'` käyttäessä R piirtää kuvan pisteinä.

Seuraavassa käytetään R:n mukana valmiiksi tulevaa esimerkkiaineistoa `iris`, johon on kerätty mittaustuloksia kolmesta erilaisesta kurjenmiekkalajista. Lisätietoja kyseisestä aineistosta saa komennolla `?iris`.

Esimerkki 5.10. Tarkastellaan kaunokurjenmiekkojen (iris setosa) terälehden pituuksia visuaalisesti histogrammin avulla antamalla funktion `hist()` argumentiksi terälehtien pituudet sisältävän vektorin. Käytetään piirtofunktion lisäargumentteja, joilla voidaan määrittää esimerkiksi otsikoita ja värejä. Näistä kerrotaan lisää kappaleessa 5.2.2

```
> hist(iris[iris$Species=="setosa",]$Petal.Length,col="lightblue",
```

```
main="Kaunokurjenmiekkojen terälehtien pituudet",
xlab="Terälehden pituus",ylab="Frekvenssi")
```

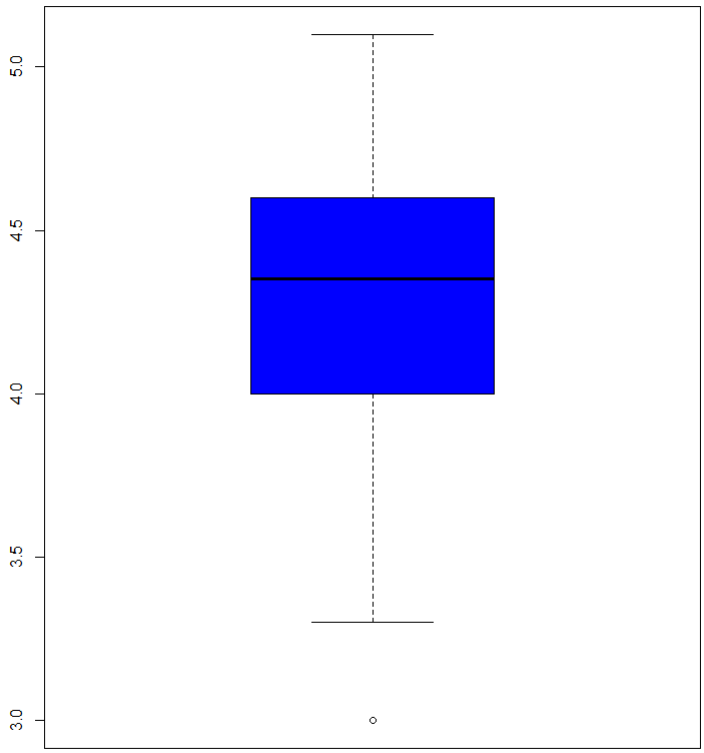


Kuva 5.2: Esimerkin 5.10 histogrammi

Esimerkki 5.11. Tarkastellaan kirjokurjenmiekkojen (iris versicolor) terälehden pituuksia boxplotilla antamalla funktion `boxplot()` argumentiksi terälehtien pituudet sisältävä vektori:

```
boxplot(iris[iris$Species=="versicolor",]$Petal.Length,col="blue")
```

Esimerkki 5.12. Tarkastellaan kaikkien ainoston sisältämien kukkien terälehden pituuksia boxplotilla, lajitteluperusteena kasvin laji:



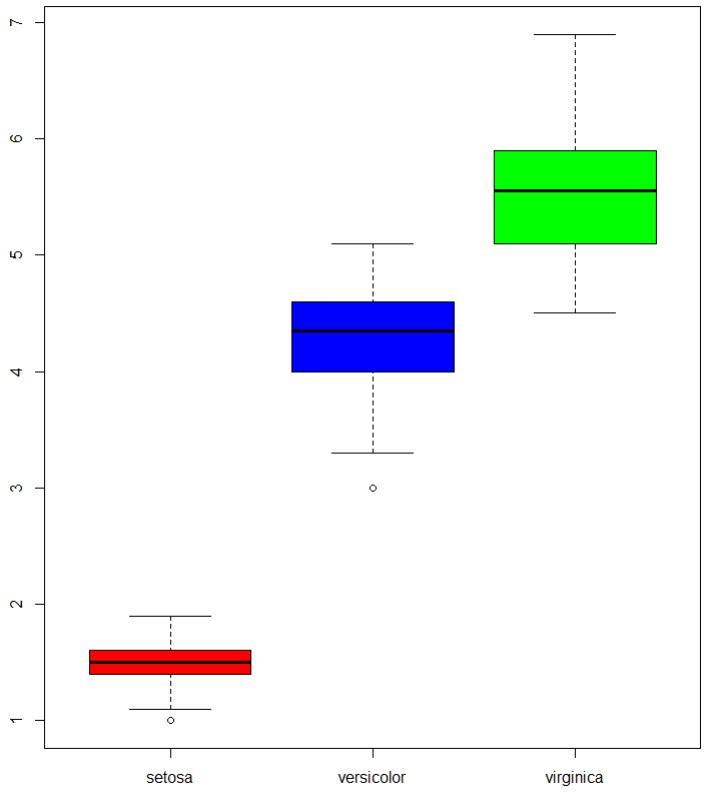
Kuva 5.3: Esimerkin 5.11 viiksilaatikkokuva

```
boxplot(iris$Petal.Length~iris$Species,col=c("red","blue","yellow"))
```

Seuraavissa esimerkeissä tutkitaan eri kurjenmiekkalajien terälehtien pituuksien suhdetta terälehtien leveyteen.

Esimerkki 5.13. Tutkitaan terälehden pituuden ja leveyden välistä yhteyttä `iris`-aineistossa. Piirretään hajontakuva `plot()`-funktiolla. Hajontakuvassa ensimmäisenä argumenttina annetaan x-pisteet ja toisen argumenttina y-pisteet.

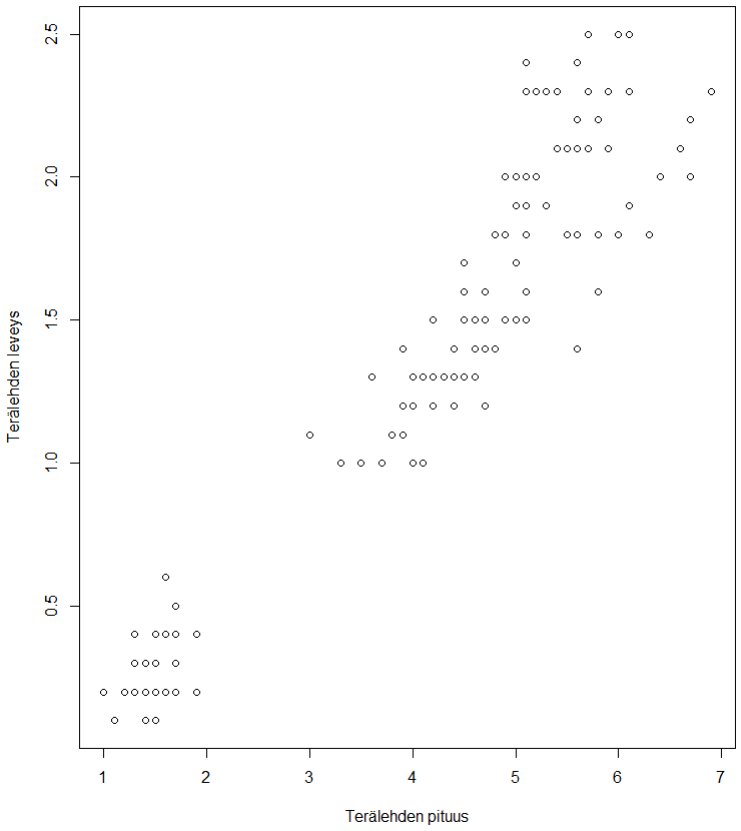
```
> plot(iris$Petal.Length,iris$Petal.Width,xlab="Terälehden pituus",
ylab="Terälehden leveys")
```



Kuva 5.4: Esimerkin 5.12 viiksilaatikkokuva

Piirroksiin voi asettaa eri värejä argumentin `col` avulla, antamalla sille värin esimerkiksi merkkijonona (esim. ”red”, ”blue” jne.) tai numerona. Muita vaihtoehtoja löytyy googlaamalla ”R color cheatsheet”. Yksittäisen värin sijasta argumentiksi `col` voi myös asettaa värit sisältävän vektorin. Jos annetun vektorin pituus on yhtä suuri kuin piirrettävien pisteiden lukumäärä, määrää vektorin kukin elementti kunkin järjestysnumeroltaan vastaavan pisteen värin.

Esimerkki 5.14. Piirretään hajontakuva `iris`-aineiston kaikkien lajien terälehtien pituuksien ja leveyksien välillä ja erotaan lajit toisistaan merkitsemällä eri lajeja eri väreillä. Piirretään hajontakuva nyt käyttäen ensimmäisenä argumenttina kaavaa (formula). Tällöin y-akselia vastaava muuttuja tulee ennen matomerkkiä ja x-akselia vastaava muuttuja matomerkin jälkeen. Kun kuvaaja piirretään kaavan avulla, voidaan aineisto antaa argu-

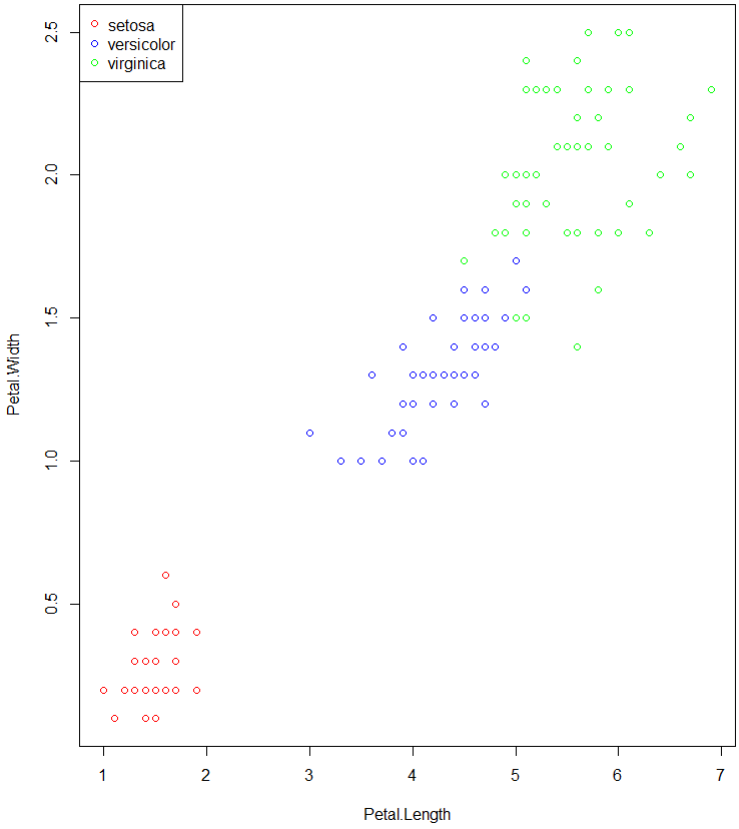


Kuva 5.5: Esimerkin 5.13 hajontakuva

menttina `data`, jolloin `$`-operaattoria ei tarvita.

```
> # Katsotaan lajien järjestys
> levels(iris$Species)

[1] "setosa"     "versicolor" "virginica"

> # Luodaan värivektori tälle järjestykselle
> iris_colors <- c("red", "blue", "green")

> # Piirretään kuvaaja koko aineistosta väreillä
> plot(Petal.Width ~ Petal.Length, data=iris, col=iris_colors[Species])
```

```
> # Lisätään selite komennolla legend():
> legend("topleft", legend=levels(iris$Species), col=iris_colors, pch=1)
```



Kuva 5.6: Esimerkin 5.14 hajontakuva, jossa lajit eroteltuina värien avulla.

5.2.2 Graafiset parametrit

Edellisessä kappaleessa esitetyille piirtofunktioille voi antaa suuren määrän erilaisia argumentteja tai parametreja, joiden avulla piirrettävä kuva voidaan säätää halutunlaiseksi. Listan käytettävissä olevista parametreista kuvauksineen saa näkyviin komennolla `?‘graphical parameters‘`. Alla on listattuna keskeisiä argumentteja, joita voi antaa suoraan piirtofunktiolle.

- `main` Kuvan pääotsikko merkkijonona.
- `sub` Kuvan alaotsikko merkkijonona.
- `xlab` ja `ylab` Akselien otsikot merkkijonoina.
- `col` Kuvaajan väri merkkijonona tai numerona. Esim. `col="red"`.
- `xlim` ja `ylim` Akselien rajat vektorina, jonka komponentteina ovat alaraja ja yläraja. Esim. `xlim=c(0,100)`.
- `type` Kuvaajan tyyppi, joka kertoo millainen kuvaaja halutaan piirtää (`plot`-funktiolla). Mahdollisia tyyppejä ovat `"p"` (pisteinä), `"l"` (viivoina/viivadiagrammina), `"b"` (pisteinä ja viivoina), `"c"` (katkonaisina viivoina pisteiden välillä), `"o"` (pisteinä ja viivoina päällekkäin), `"h"` (pystyviivoina), `"s"` ja `"S"` (”portaina”), `"n"` (tyhjänä).

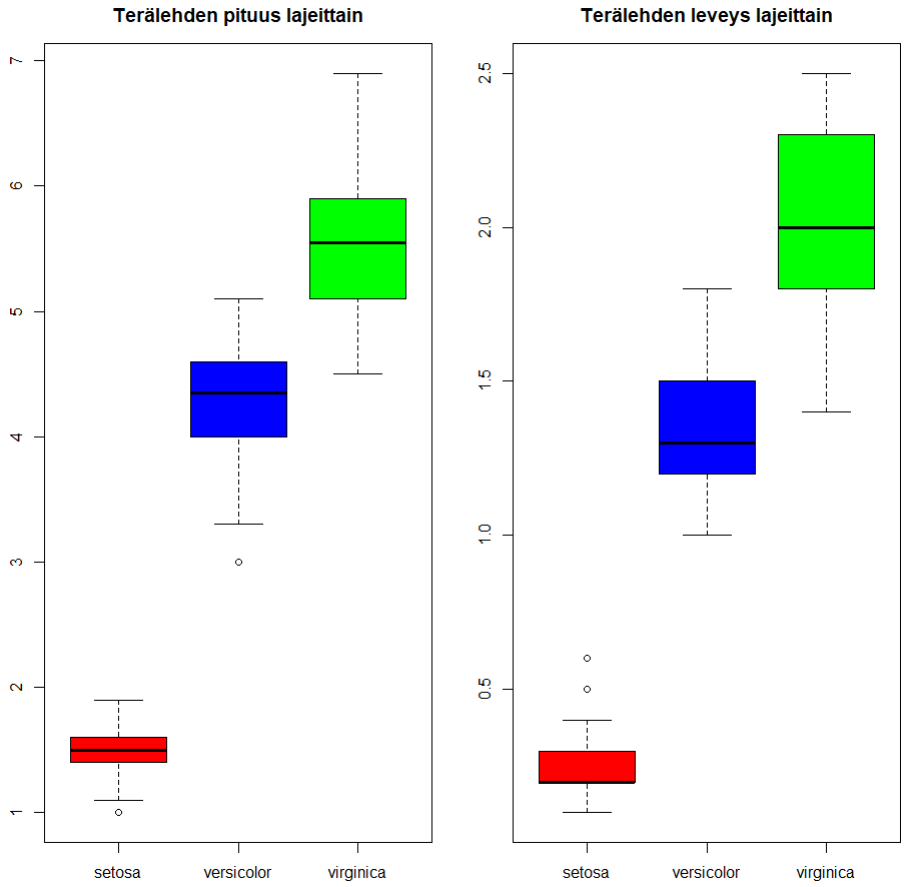
Joitakin parametreja, kuten kuvan ympärille jätettäviä marginaaleja tai samaan kuvioon piirrettävien kuvien asettelua, ei kuitenkaan voida säätää piirtofunktion argumenteissa, vaan ne tulee asettaa kommennon `par()` avulla.

Kuvan ympärille jätettävät marginaalit voi määrätä `par()`-funktion argumentilla `mar` antamalla sen arvoksi nelipaikkaisen numeerisen vektorin. Vektorin ensimmäinen elementti määrää alamarginaalin, toinen vasemman-, kolmas ylä- ja neljäs oikeanpuoleisen marginaalin koon tekstirivien määrällä mitattuna. Vaihtoehtoisesti marginaalit voi säätää argumentilla `mai`, jolloin yksikkönä on tuumat.

Samaan kuvioon voidaan piirtää useita kuvia ns. matriisimuotoon järjestettynä `par()`-funktion argumentin `mfrow` tai `mfcol` avulla. Argumentiksi asetetaan kaksipaikkainen vektori, jonka ensimmäinen elementti määrää haluttujen ”rivien” määrän ja toinen ”sarakkeiden” määrän. Näistä `mfrow` täyttää ”kuvamatriisin” riveittäin ja `mfcol` sarakkeittain järjestyksessä.

Esimerkki 5.15. Piirretään samaan kuvioon kaksi kuvaa vierekkäin, ja asetetaan kuvion jokaisen kuvan oikean sivun marginaaliksi 1.1 ja muiden sivujen marginaaliksi 3.1 riviä. Vasemman puoleiseen kuvaan piirretään kuva 5.4 uudestaan ja oikealle puolelle vastaava kuva, jossa tarkasteltavana muuttujana on `Petal.Width`

```
> par(mar=c(3.1,3.1,3.1,1.1),mfrow=c(1,2))
> boxplot(iris$Petal.Length~iris$Species,col=iris_colors,
main="Terälehden pituus lajeittain")
> boxplot(iris$Petal.Width~iris$Species,col=iris_colors,
main="Terälehden leveys lajeittain")
```



Kuva 5.7: Esimerkin 5.15 kuva, jossa kaksi viiksilaatikkokuvaa piirretty vierekkäin.

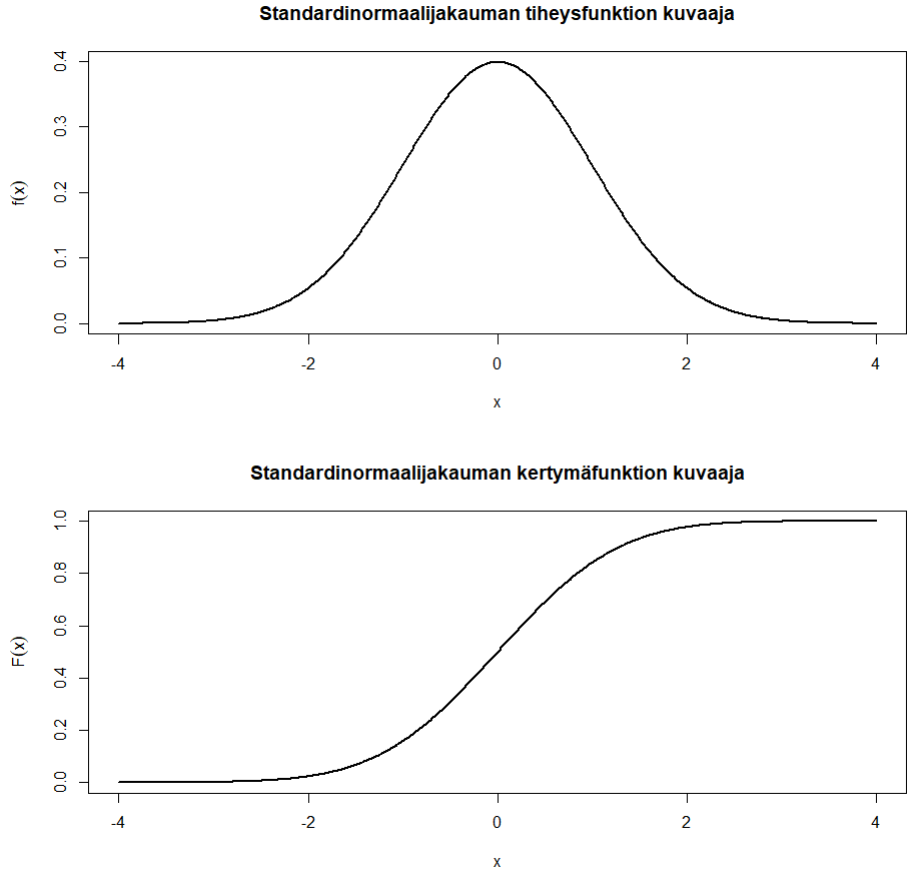
Jos `par()`-funktiolle ei anna argumentteja, palauttaa se käytössä olevat parametrit listana.

Esimerkki 5.16.

```
> pars <- par()
> pars$mar

[1] 3.1 3.1 3.1 1.1

> pars$mfrow

[1] 1 2
```

Esimerkki 5.17. Piirretään standardinormaalijakauman tiheys- ja kertymäfunktioiden kuvaajat allekkain. Piirretään kuvaajat välillä [-4,4]. Argumentti `lwd` määrittää piirrettävän viivan paksuuden: mitä suurempi numero, sitä paksumpi viiva.

```
> par(mfrow=c(2,1))
> x <- seq(-4,4,by=0.01)
> plot(x,dnorm(x,0,1),main="Standardinormaalijakauman tiheysfunktion kuvaaja",
xlab=expression(x),ylab=expression(f(x)),type="l",lwd=2)
> plot(x,pnorm(x,0,1),main="Standardinormaalijakauman kertymäfunktion kuvaaja",
xlab=expression(x),ylab=expression(F(x)),type="l",lwd=2)
```



Kuva 5.8: Esimerkin 5.17 tiheys- ja kertymäfunktioiden kuvaajat.

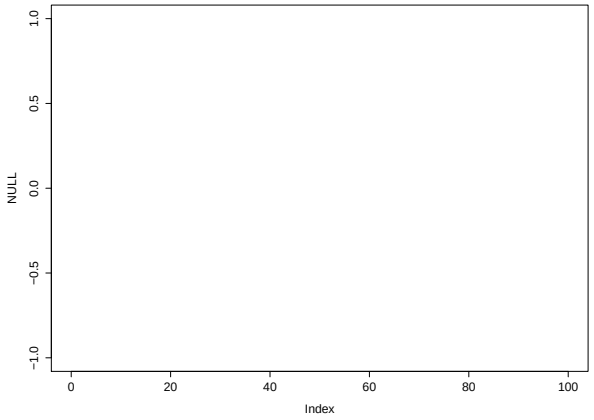
5.2.3 Asioiden lisääminen olemassa olevaan kuvaan

Joskus kuvaa ei voida piirtää kokonaisuudessaan `plot()`-funktiolla, vaan kuva joudutaan piirtämään osissa. Tällaisia tilanteita tulee esimerkiksi silloin, kun halutaan piirtää useita asioita samaan kuvaan.

Tutkitaan seuraavaksi miten `plot()`-funktiota voidaan käyttää piirtämään tyhjä kuva ja miten siihen voidaan lisätä asioita jälkikäteen käyttäen funktiota `lines()`.

Esimerkki 5.18. Luodaan tyhjä kuvaaja. On kuitenkin syytä kertoa `plot()`-funktiolle x- ja y-akselien minimit ja maksimit. Tämä voidaan tehdä käyttäen argumentteja `xlim` ja `ylim`:

```
> plot(NULL, xlim=c(0,100), ylim=c(-1,1))
```

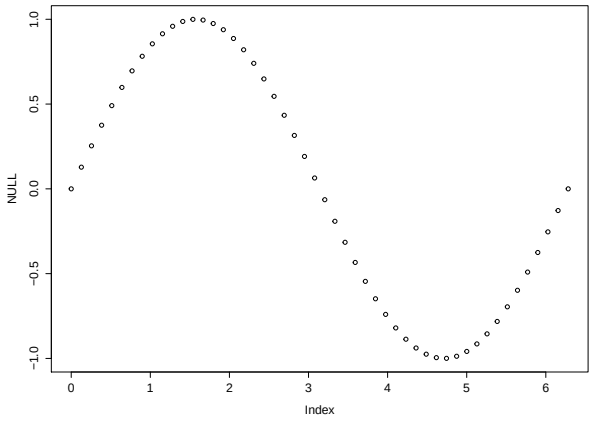


Tällä saadaan kuvaaja, joka on muuten tyhjä, mutta siihen on piirretty x-akseli välille [0,100] ja y-akseli välille [-1,1].

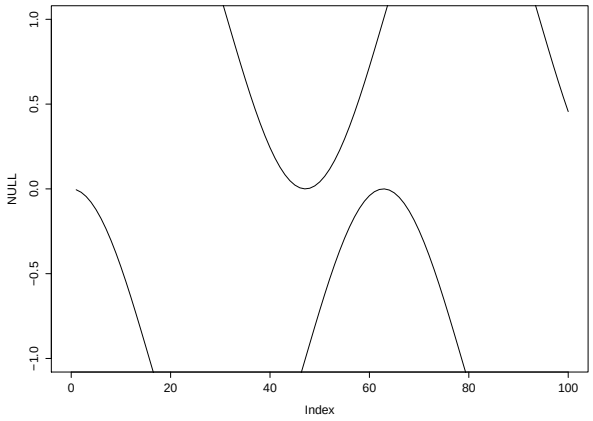
Esimerkki 5.19. Luodaan nyt tyhjä kuvaajaikkuna ja piirretään siihen pisteitä sini-funktiosta:

```
> plot(NULL, xlim=c(0,2*pi), ylim=c(-1,1))
> x_points <- seq(0,2*pi, length.out=50)
> points(x=x_points, y=sin(x_points))
```

Esimerkki 5.20. Piirretään nyt esimerkin 5.18 koodilla luotavaan tyhjään kuvaajaan sini- ja kosini -käyrät:



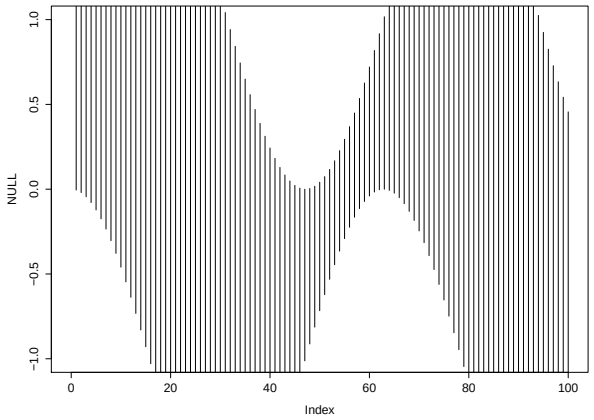

```
> x <- sapply(1:100, function(x) c(sin(x/10)+1, cos(x/10)-1))
> plot(NULL, xlim=c(0,100), ylim=c(-1,1))
> lines(x[1,], type="l")
> lines(x[2,], type="l")
```



Katsotaan sitten, miten voidaan piirtää viivoja pisteiden välille. Seuraava esimerkki on mielenkiintoinen esimerkiksi luottamusvälin toiminnan havainnollistamisessa. Soveltaminen luottamusväliin jätetään kuitenkin tehtäväksi, joten sovelletaan tätä nyt edellisen esimerkin sini- ja kosini- funktioihin.

Esimerkki 5.21. Piirretään pystyviivoja sini-funktiosta kosini-funktioon:

```
> x <- sapply(1:100, function(x) c(sin(x/10)+1, cos(x/10)-1))
> plot(NULL, xlim=c(0,100), ylim=c(-1,1))
> segments(x0 = 1:100, y0 = x[1,], y1 = x[2,])
```



Tässä `segments()`-funktion argumentti `x0` on viivan paikka x-akselilla ja `y0, y1` ovat viivan päätepisteet y-akselilla.

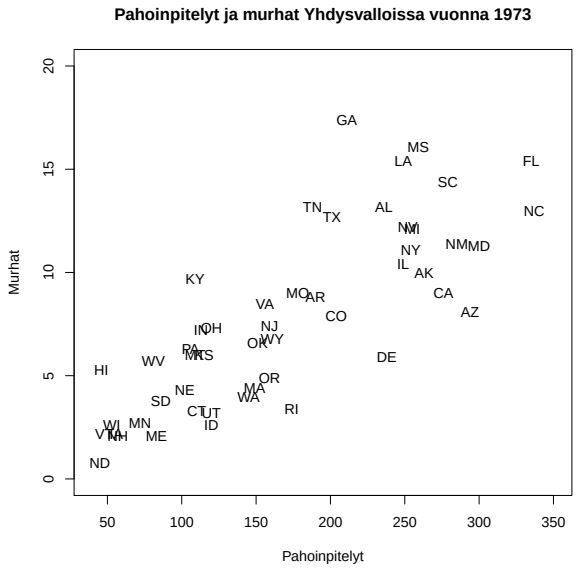
Kokeillaan vielä, kuinka kuvaajaan voidaan lisätä tekstiä `text`-funktion avulla. Tätä voidaan hyödyntää esimerkiksi, kun kuvaajasta halutaan korostaa vaikkapa poikkeuksellisia havaintoja tai kun hajontakuvan symboleina halutaan käyttää kahden tai useamman merkin mittaisia merkkijonoja.

Esimerkki 5.22. Tarkastellaan R:stä löytyvää esimerkkiaineistoa `USArrests`, joka sisältää osavaltiotasoisia rikostilastoja (per 100000 asukasta) Yhdysvalloista vuodelta 1973. Havainnollistetaan siinä havaittavaa pahoinpitelyjen ja murhien välistä yhteyttä hajontakuvalla, jossa kukin havainto merkitään sitä vastaavan osavaltion lyhenteellä. Osavaltioiden lyhenteet löytyvät R:ssä oletuksena objektista `state.abb`.

```
> plot(NULL, xlim = c(40, 350), ylim = c(0, 20),
+    xlab = 'Pahoinpitelyt', ylab = 'Murhat',
+    main = 'Pahoinpitelyt ja murhat Yhdysvalloissa vuonna 1973')
> text(x = USArrests$Assault, y = USArrests$Murder, labels = state.abb)
```

5.3 Keskeisiä funktioita

Funktioiden toimintaan ja funktioille annettaviin argumentteihin voit tutustua tarkemmin R:n ohjekirjasta.



Vektorin x alkioiden summa: `sum(x)`
Vektorin x alkioiden otoskeskiarvo: `mean(x)`
Vektorin x alkioiden otosvarianssi: `var(x)`
Vektorin x alkioiden otoskeskihajonta: `sd(x)`
Kahden vektorin x ja y otoskovarianssi: `cov(x,y)`
Kahden vektorin x ja y otoskorrelaatio: `cor(x,y)`
Suurin vektorin x alkio: `max(x)`
Pienin vektorin x alkio: `min(x)`
Vektorin x keskimmäinen alkio: `median(x)`
Vektorin x kvantiilit: `quantile(x)`
Yleiskatsaus vektorin x arvojen jakautumisesta: `summary(x)`

Vektorin x alkioiden järjestäminen suuruusjärjestykseen: `sort(x)`
Vektorin alkioiden järjestysluvut: `rank(x)`

Jakaumafunktiot (funktion ensimmäinen kirjain määrää mitä jakaumasta lasketaan `d`: tiheysfunktio/pistetodennäköisyysfunktio, `p`: kertymäfunktio, `q`: kvantiili, `r`: havaintojen simulointi jakaumasta)
Binomijakauma: `dbinom/pbinom/qbinom/rbinom`
Geometrinen jakauma: `dgeom/pgeom/qgeom/rgeom`
Poisson-jakauma: `dpois/ppois/qpois/rpois`

Hypergeometrinen jakauma: `dhyper/phyper/qhyper/rhyper`
Negatiivinen binomijakauma: `dnbinom/pnbinom/qnbinom/rnbinom`
Normaalijakauma: `dnorm/pnorm/qnorm/rnorm`
Eksponenttijakauma: `dexp/pexp/qexp/rexp`
Tasajakauma: `dunif/punif/qunif/runif`
Studentin t-jakauma: `dt/pt/qt/rt`
Khii toiseen-jakauma: `dchisq/pchisq/qchisq/rchisq`
Gammajakauma: `dgamma/pgamma/qgamma/rgamma`
Beta-jakauma: `dbeta/pbeta/qbeta/rbeta`
F-jakauma: `df/pf/qf/rf`

Satunnaisotos vektorista: `sample()`

Piirtofunktiot
Yleinen kuva, jonka tyyppi riippuu siitä, minkälaisia argumentteja sille annetaan: `plot()`
Funktion kuvaaja: `curve()`
Histogrammi: `hist()`
Pylväskuva: `barplot()`
Viiksilaatikkokuva: `boxplot()`

Olemassaolevaan kuvaan asioiden lisääminen
Pisteiden lisääminen: `points()`
Käyrän lisääminen: `lines()`
Pysty-/vaakaviivan lisääminen koko kuvan läpi: `abline()`
Viivojen piirtäminen valituista pisteistä valittuihin pisteisiin: `segments()`

Katso pitkä lista graafisia parametreja piirtofunktioille: `?par`
Akselien asetukset: `axis()`
Kuvaan lisättävät selitykset: `legend()`

Kappale 6

Funktiot ja silmukat

6.1 Funktiot

Kaikki operaatiot R:ssä ovat itse asiassa funktiokutsuja - eivät siis vain komennot kuten `mean()`, `plot()` tai `par()`, vaan myös komennot kuten `+`, `(`, `$` ja `[]`. Funktiot eivät siten ole vain keskeinen osa tehokasta R-ohjelmointia, vaan ne ovat hyvin oleellisia myös pyrkimyksessä ymmärtää R:ää yleisellä tasolla. Tässä kappaleessa tutustutaan kuitenkin vain lyhyesti omien funktioiden rakentamisen alkeisiin sekä aiempaa tarkemmin siihen, kuinka valmiiden funktioiden argumentteja voidaan asettaa.

6.1.1 Omat funktiot

Monenlaiseen peruskäyttöön, esimerkiksi otoskeskiarvon, -varianssin ja vastaavien laskemiseen, löytyy valmiit funktiot. Joskus valmiita funktioita on kuitenkin tarve yhdistellä uusiksi funktioiksi omiin tarpeisiin sopivaksi. Hyvin kirjoitettu funktio voidaan helposti siirtää uusiin koodeihin ja näin päästään käyttämään jo luotua koodia nopeasti uudestaan.

Omia funktioita voi luoda komennolla `function()`, jonka argumentteina tulee pilkulla erotellen nimetä luotavassa funktiossa käytettävät sellaiset argumentit, joihin käyttäjän halutaan voivan vaikuttaa[1]. Samalla nimetyille argumenteille voi asettaa oletusarvot (jotka voi asettaa tai laskea myös itse funktion sisällä). Funktion toiminta voidaan koodata

[1]Kaikkia tällaisia argumentteja ei käytännössä tarvitse nimetä, koska käyttäjälle voi antaa mahdollisuuden määrätä myös muita argumentteja kolmen pisteen osoittaman argumentin `...` avulla. Tämä voi olla kätevää esimerkiksi silloin, kun funktio mahdollisesti kutsuu jotakin muuta funktiota, jonka argumentteihin halutaan antaa käyttäjälle mahdollisuus vaikuttaa, mutta ei haluta erikseen nimetä valittavissa olevia argumentteja. Tällainen `...`-argumentti esiintyy esimerkiksi `plot()`-funktiossa, ja se mahdollistaa muun muassa graafisten parametrien säätämisen listaamatta niitä kaikkia erikseen.

funktio-komennon perään aaltosulkeiden sisään ja funktion palauttaman olion voi määrä `return()`-komennolla.

Esimerkki 6.1. Luodaan yksinkertainen funktio `zeros()`, jolle annetaan argumenttina luonnollinen luku `N`, ja joka palauttaa `N`-pituisen nollavektorin:

```
zeros <- function(N) {
  z <- rep(0, N)
  return(z)
}
```

Kerrataan vielä pala kerrallaan, mitä edellisessä koodissa tapahtuu.

- Ensin määritellään funktion nimi `zeros`.
- Tähän sijoitetaan sijoitusnuolella komento `function(N)`. Sulkujen sisälle kirjoitetaan funktiota kutsuttaessa sille annettavat arvot, eli argumentit[2]. Tällä funktiolla on yksi argumentti: `N`.
- Varsinainen funktio, eli funktion suorittamat toiminnot kirjoitetaan aaltosulkeiden {} sisään:
  - Muuttujaan `z` sijoitetaan `N` kappaletta nollia
  - muuttuja `z` palautetaan komennolla `return(z)`

Huomaa: Funktion määrittelevä koodi täytyy ajaa, jotta funktio tallentuu R:n työtilaan. Vasta tämän jälkeen funktiota voi kutsua.

Esimerkki 6.2. Käytetään edellisen esimerkin funktiota `zeros()` luomaan nollavektoreita.

```
> zeros(1)

[1] 0

> zeros(5)

[1] 0 0 0 0 0
```

[2]Tarkemmin sanottuna funktion määritelmässä määriteltyjä funktion argumentteja (tässä `N`) kutsutaan *formaaleiksi argumenteiksi*, ja funktiota kutsuttaessa niille annettavia arvoja varsinaisiksi argumenteiksi.

```
> zeros(20)

 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

R:ssä, toisin kuin esimerkiksi C:ssä tai Javassa, `return()`-komento ei ole välttämätön funktion lopussa, vaan funktio palauttaa automaattisesti viimeiseksi käsitellyn arvon. Siten esimerkin 6.1 funktio voidaan kirjoittaa kompaktimmin seuraavasti.

Esimerkki 6.3.
```
zeros <- function(N) {
  rep(0, N)
}
```

Vain yhden rivin sisältävissä funktioissa myöskään aaltosulut eivät ole välttämättömiä, vaan funktion ainoan rivin voi kirjoittaa suoraan `function()`-komennon perään, joko seuraavalle, tai samalle riville (näistä ensimmäistä ei kuitenkaan kannata käyttää koodin luettavuuden heikentymisen takia). Siten myös seuraavat versiot ovat yhtäpitäviä esimerkin 6.1 funktion kanssa:

Esimerkki 6.4.
```
zeros <- function(N)
  rep(0, N)
```

Esimerkki 6.5.
```
zeros <- function(N) rep(0, N)
```

Seuraavassa esimerkissä luodaan hieman monimutkaisempi funktio. Käytetään jakaumafunktiota `runif()` arpomaan `N` satunnaislukua väliltä $[a, b]$. Pyöristetään saadut arvot käyttäen funktiota `round()`, tallennetaan tulos vektoriin `int_values` ja palautetaan se:

Esimerkki 6.6.
```
random_integers <- function(N, a, b) {
  values <- runif(n=N, min=a, max=b)
  int_values <- round(values)
  return(int_values)
}
```

Kutsumalla tätä funktiota saadaan satunnaisia kokonaislukuja (jotka eivät kuitenkaan ole otos välin $[a, b]$ diskreetistä tasajakaumasta):

```
> random_integers(10, 0, 5)
```

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

> random_integers(4, 10, 12)

[1] 11 12 11 12
```

Esimerkki 6.7. Tehdään funktio, joka laskee binomikertoimen luvuista n ja k, eli $\binom{n}{k}$. Binomikerroin luvuista n ja k on määritelty olemaan

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

olettaen, että n ja k ovat ei-negatiivisia kokonaislukuja ja $n \geq k$. Tähän on olemassa R:n valmis funktio `choose()`, mutta toteutetaan samalla tavalla toimiva funktio itse.

```
binomikerroin <- function(n,k) {
  osoittaja <- factorial(n)
  nimittaja <- factorial(k)*factorial(n-k)
  osoittaja/nimittaja
}

> binomikerroin(10,2)

[1] 45

> choose(10,2)

[1] 45
```

6.1.2 Nimetyt argumentit ja oletusarvot argumenteille

Omien funktioiden määrittelyssä voidaan käyttää myös kahta R:n funktioiden erityispiirrettä: *nimettyjä argumentteja* ja *oletusarvoja* argumenteille (esimerkiksi C:ssä tai Javassa ei ole kumpaakaan näistä ominaisuuksista, vaan funktioita kutsuttaessa niille täytyy antaa kaikki sen määritelmässä annetut argumentit siinä järjestyksessä kuin ne on määritelmään kirjoitettu.

Esimerkki 6.8.

```
random_integers <- function(N=1, a=0, b=1) {
  values <- runif(n=N, min=a, max=b)
  int_values <- round(values)
```

```
  return(int_values)
}
```

Nämä mahdollistavat esimerkiksi seuraavanlaiset tavat kutsua `random_integers()`-funktiota:

```
> random_integers(10, 15, 20)

 [1] 18 18 20 20 19 18 18 20 19 17

> random_integers(b=7, a=2, N=10)

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

> random_integers(10, -1)

 [1]  1 -1  0  0  0  1 -1  0  0  1

> random_integers(10, b=5)

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

> random_integers(10)

 [1] 1 1 1 1 1 0 0 1 0 0

> random_integers()

[1] 1
```

Ensimmäisessä funktiokutsussa, jossa yhtäkään argumenttia ei ole nimetty, R yhdistää funktiolle sitä kutsuttaessa annetut varsinaiset argumentit 10, 15, 20 sen *formaaleihin argumentteihin* `N, a, b` siinä järjestyksessä kuin formaalit argumentit on kirjoitettu funktiokutsussa. Siten `N = 10, a = 15` ja `b = 20`.

Seuraavassa kutsussa kaikki argumentit on nimetty, jolloin järjestyksellä ei ole väliä, vaan kaikki argumentit yhdistetään nimen perusteella.

Kolmannessa kutsussa taas kaksi ensimmäistä argumenttia yhdistetään järjestyksen mukaan, eli `N = 10` ja `a = -1`. Sen sijaan formaalille argumentille `b` ei ole määritelty

arvoa funktiokutsussa, joten sille käytetään funktion määrittelyssä asetettua oletusarvoa `b = 1`.

Neljännessä kutsussa yhdistetään ensin `b = 5` nimen mukaan, ja sitten `N = 10` järjestyksen mukaan. Formaalille argumentille `a` ei ole annettu funktiokutsussa arvoa, joten sille käytetään oletusarvoa `a = 0`

Viidennessä kutsussa yhdistetään ensin `N = 10` järjestyksen mukaan, ja sen jälkeen käytetään lopuille argumenteille oletusarvoja `a = 0` ja `b = 1`.

Viimeisessä funktiokutsussa taas ei ole annettu yhtään arvoa, joten kaikille formaaleille argumenteille käytetään niiden oletusarvoja `N = 1`, `a = 0` ja `b = 1`.

R yhdistää siis formaalit argumentit varsinaisiin argumentteihin ensisijaisesti nimen perusteella, ja sen jälkeen alkaa yhdistelemään jäljellejääneitä nimeämättömiä argumentteja järjestyksen perusteella. Tämä mahdollistaa esimerkiksi seuraavan varsin hämäävän funktiokutsun:

Esimerkki 6.9.

```
> # Älä tee näin!
> random_integers(a = 100, b = 200, 10)

 [1] 189 105 132 143 143 131 175 168 188 101
```

Tätä funktiokutsua tulkitessaan R yhdistää ensin `a = 100` ja `b = 200` nimen perusteella, ja sen jälkeen alkaa yhdistelemään jäljellejääviä (nimeämättömiä) argumentteja järjestyksen perusteella, jolloin se yhdistää `N = 10`. Tällaiset funktiokutsut ovat koodin lukijalle erittäin hankalia tulkita, joten jos samaan funktiokutsuun yhdistetään sekä nimeämättömiä että nimettyjä argumentteja, hyvään ohjelmointitapaan kuuluu **määritellä kaikki nimeämättömät argumentit ennen nimettyjä**, esimerkiksi seuraavasti:

Esimerkki 6.10.

```
> random_integers(10, a = 100, b = 200)

 [1] 182 131 112 193 123 183 107 168 142 122
```

Nimettyjä argumentteja kannattaa käyttää funktiokutsussa yleensä selkeyden ja luettavuuden vuoksi, jos funktiolle annetaan paljon argumentteja. Esimerkiksi seuraava funktiokutsu, jota voi käyttää lukemaan kappaleen 4.2 esimerkkiaineisto, kyllä toimii, mutta ei ole erityisen helposti tulkittava:

Esimerkki 6.11.

```
> ot <- read.csv('OT2014.csv', TRUE, ';', '\"', ',')
```

Sen sijaan seuraava funktiokutsu, joka tekee täsmälleen saman asian (mikä varmistetaan `identical()`-funktiolla, joka testaa, ovatko kaksi oliota samat) on huomattavasti luettavampi:

Esimerkki 6.12.

```
> ot2 <- read.csv('OT2014.csv', sep = ';', dec = ',')
> identical(ot, ot2)

[1] TRUE
```

Tästä versiosta harjaantunut silmä näkee heti, että tiedostoa luettaessa sarakkeiden erottimena (separator) halutaan käyttää puolipistettä, ja desimaalierottimena pilkkua. Huomaa myös, että koska argumentit `sep` ja `dec` nimettiin, argumentit `header` ja `quote` voidaan jättää kirjoittamatta, jolloin niille käytetään oletusarvoja. R:n valmiiden funktioiden formaalien argumenttien nimen, järjestyksen ja oletusarvot näkee R:n help-komennolla. Esimerkiksi `?read.csv`-komennolla löydetään seuraava kuvaus:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)
```

6.2 Silmukat

Joitakin koodin osia on tarve toistaa useita kertoja, eikä saman koodin kirjoittaminen useaan kertaan peräkkäin ole järkevää. Tällöin tarvitaan for -silmukkaa tai jotakin vastaavaa rakennetta. Tutustutaan seuraavaksi erilaisiin tapoihin luoda koodiin toistorakenteita.

Esimerkki 6.13. *Motivoiva esimerkki silmukoiden käyttöön.* Luodaan matriisi, jossa on viisi riviä ja kymmenen saraketta ja oletetaan, että halutaan laskea erikseen matriisin jokaiselle sarakkeelle saraketulot, eli kertoa yksittäisten sarakkeiden arvot keskenään.

```
> A <- matrix(1:50, nrow=5)
> A

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    6   11   16   21   26   31   36   41    46
[2,]    2    7   12   17   22   27   32   37   42    47
[3,]    3    8   13   18   23   28   33   38   43    48
[4,]    4    9   14   19   24   29   34   39   44    49
[5,]    5   10   15   20   25   30   35   40   45    50
```

Saraketuloja voitaisiin laskea sarake kerrallaan kirjoittamalla samaa koodia putkeen vain sarakenumeron vaihdellessa:

```
> prod(A[,1])

[1] 120

> prod(A[,2])

[1] 30240

> prod(A[,3])

[1] 360360
```

Tällainen koodi on kuitenkin paitsi työlästä, myös kömpelön näköistä. Tyylikkäämpi ja vaivattomampi tapa on käyttää silmukoita, eli koodia toistavia rakenteita, joihin kuuluvat seuraavaksi esiteltävä for-silmukka sekä `apply`-perheen funktiot.

6.2.1 For-silmukka

Tehdään ensin yksinkertainen tulostussilmukka ja tutkitaan sen toimintaa:

Esimerkki 6.14.

```
> for(i in 1:10) {
   print(i)
  }

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

Mitä for-silmukka tekee? Se toistaa aaltosulkeiden sisälle kirjoitettua koodia indeksiksi annetun muuttujan `i` eri arvoilla. Katsotaan hieman tarkemmin rakennetta:

- `for(i in 1:10)` määrittelee for-silmukan ja sen sisälle indeksimuuttujan `i`.
- Aaltosulkeiden sisällä oleva koodi toistetaan siis jokaisella vektorin `1:10` luvulla. Jokaisella näistä toistoista muuttuja `i` saa järjestyksessä yhden tämän vektorin arvoista.
- Silmukan sisällä ei voi tulostaa antamalla pelkän muuttujan nimen komentona, vaan joudutaan käyttämään komentoa `print()`. Harvoin silmukan sisällä kuitenkaan oikeasti halutaan tulostaa, joten siinä mielessä tämä esimerkki on hieman hölmö.

Katsotaan seuraavaksi, miten aiemman esimerkin matriisin saraketulot voitaisiin laskea for-silmukan avulla.

Esimerkki 6.15.

```
> saraketulot <- numeric(ncol(A))
> for(i in 1:ncol(A)) {
   saraketulot[i] <- prod(A[,i])
  }
> saraketulot

 [1]       120     30240    360360   1860480   6375600  17100720
 [7]  38955840  78960960 146611080 254251200
```

Ensin alustetaan numeerinen vektori `saraketulot`, johon tullaan tallentamaan kaikki 10 saraketuloa. Funktio `numeric` luo nollavektorin, jossa on niin monta alkiota kuin sille annetaan argumenttina. Tässä alkioiden määräksi asetetaan matriisin A sarakkeiden määrä funktiolla `ncol`, jolloin vektorissa `saraketulot` on kymmenen nollaa ennen for-silmukan suorittamista.

For-silmukassa indeksimuuttuja `i` käy läpi kaikki kokonaisluvut yhdestä sarakkeiden määrään eli kymmeneen. Jokaisella indeksin `i` arvolla vektorin `saraketulot` `i`:nnenteen alkioon sijoitetaan matriisin `A` `i`:nnen sarakkeen arvojen tulo.

Vaikka for-silmukkaa käyttämällä vältettiin saman koodin toistaminen kymmenen kertaa, voidaan toistorakenne hoitaa vielä vähemmälläkin koodin kirjoittamisella `apply`-perheen funktioiden avulla.

6.2.2 apply()

R-ohjelmoijilla on monesti tapana suosia `apply`-perheen funktoita `for`-silmukoiden sijaan, kun operaatioita halutaan toistaa useille vektorien matriisien, tai taulukoiden alkioille. Tämä tekee monesti koodista kompaktimpaa, selkeämpää, ja vähemmän herkkää ohjelmointivirheille, eli bugeille. Sen sijaan se, että `apply`-perheen funktiot olisivat nopeampia

kuin `for`-silmukat, on urbaani legenda. Pitää kyllä paikkansa, että `for`-silmukat ovat (enimmäkseen) hitaita R:ssä, mutta niin ovat myös `apply`-perheen funktiot. Sen sijaan nopeaa on *vektorisoida* operaatiot, eli esittää ne vektorien ja matriisien laskutoimituksina. Aina tämä ei kuitenkaan ole mahdollista.

Tutustutaan ensimmäisenä funktioon `apply()`. Sen ensimmäinen argumentti on matriisi tai taulukko, johon kolmantena argumenttina annettua funktiota sovelletaan. Toiselle argumentille (`MARGIN=`) annetaan arvo 1 tai 2 riippuen siitä halutaanko funktiota soveltaa matriisin riveihin (1) vai sarakkeisiin (2).

Saraketuloesimerkissä sovellettiin samaa funktiota matriisin `A` jokaiseen sarakkeeseen for-silmukan sisällä. Sama voidaan tehdä nyt funktiolla `apply` antamalla argumenteiksi matriisi ja sopiva funktio. Vain yhden argumentin tarvitsevan funktion yhteydessä `apply()`:lle riittää antaa pelkkä sovellettavan funktion nimi, tässä tapauksessa `prod`

Esimerkki 6.16.

```
> apply(A, 2, prod)

 [1]       120     30240    360360   1860480   6375600  17100720
 [7]  38955840  78960960 146611080 254251200
```

Luodaan nyt ensin matriisi `B`, valitaan siitä sarakkeet 4,5,6,7,8 matriisiin `B_1` ja lasketaan sen riveittäiset keskiarvot ja sarakesummat:

Esimerkki 6.17.

```
> B <- matrix(seq(5,9, length.out=100), nrow=10)
> B_1 <- B[,4:8]
> B_1

          [,1]     [,2]     [,3]     [,4]     [,5]
 [1,] 6.212121 6.616162 7.020202 7.424242 7.828283
 [2,] 6.252525 6.656566 7.060606 7.464646 7.868687
 [3,] 6.292929 6.696970 7.101010 7.505051 7.909091
 [4,] 6.333333 6.737374 7.141414 7.545455 7.949495
 [5,] 6.373737 6.777778 7.181818 7.585859 7.989899
 [6,] 6.414141 6.818182 7.222222 7.626263 8.030303
 [7,] 6.454545 6.858586 7.262626 7.666667 8.070707
 [8,] 6.494949 6.898990 7.303030 7.707071 8.111111
 [9,] 6.535354 6.939394 7.343434 7.747475 8.151515
[10,] 6.575758 6.979798 7.383838 7.787879 8.191919

> # Riveittäiset keskiarvot
```

```
> apply(B_1, 1, mean)

 [1] 7.020202 7.060606 7.101010 7.141414 7.181818 7.222222 7.262626
 [8] 7.303030 7.343434 7.383838

> # Sarakesummat
> apply(B_1, 2, sum)

[1] 63.93939 67.97980 72.02020 76.06061 80.10101
```

Lasketaan seuraavaksi matriisin `B_1` jokaisen rivin pienimmän ja suurimman alkion tulo.

Esimerkki 6.18.

```
> apply(B_1,1,function(x) max(x)*min(x))

 [1] 48.63024 49.19916 49.77135 50.34680 50.92552 51.50750 52.09275
 [8] 52.68126 53.27303 53.86807
```

Mikäli kolmantena argumenttina annetulla funktiolla on omia argumentteja, ne annetaan `apply`-funktiolle lisäargumenteiksi. Esimerkiksi, jos puuttuvia arvoja ei haluta laskea mukaan sarakekeskiarvoihin, huomioidaan tämä antamalla `mean`-funktion argumentti `na.rm=TRUE` `apply`:n lisäargumentiksi.

Esimerkki 6.19.

```
> B_1[c(2,3),c(1,2,5)] <- NA
> B_1

          [,1]     [,2]     [,3]     [,4]     [,5]
 [1,] 6.212121 6.616162 7.020202 7.424242 7.828283
 [2,]       NA       NA 7.060606 7.464646       NA
 [3,]       NA       NA 7.101010 7.505051       NA
 [4,] 6.333333 6.737374 7.141414 7.545455 7.949495
 [5,] 6.373737 6.777778 7.181818 7.585859 7.989899
 [6,] 6.414141 6.818182 7.222222 7.626263 8.030303
 [7,] 6.454545 6.858586 7.262626 7.666667 8.070707
 [8,] 6.494949 6.898990 7.303030 7.707071 8.111111
 [9,] 6.535354 6.939394 7.343434 7.747475 8.151515
[10,] 6.575758 6.979798 7.383838 7.787879 8.191919
```

```
> apply(B_1,2,mean)

[1]       NA       NA 7.202020 7.606061       NA

> apply(B_1,2,mean,na.rm=TRUE)

[1] 6.424242 6.828283 7.202020 7.606061 8.040404
```

6.2.3 lapply(), sapply() ja vapply()

Kun samaa operaatiota halutaan toistaa erilaisilla arvoilla ja tulos halutaan palauttaa listana, voidaan käyttää funktiota `lapply`, joka ottaa ensimmäisenä argumenttinaan objektin (usein vektori tai lista) ja toisena argumenttina funktion, jota sovelletaan jokaisella objektin arvolla.

Esimerkki 6.20. Tarkastellaan R:stä valmiiksi löytyvää aineistoa `iris`. Tuotetaan neljän ensimmäisen sarakkeen yhteenveto `summary`-funktiolla ja palautetaan ne listana.

```
> lapply(1:4,function(x) summary(iris[,x]))

[[1]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900

[[2]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.800   3.000   3.057   3.300   4.400

[[3]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.600   4.350   3.758   5.100   6.900

[[4]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.100   0.300   1.300   1.199   1.800   2.500
```

Huomataan, että `lapply` palauttaa listan, jossa on yhtä monta komponenttia kuin argumenttina annetussa vektorissa `1:4` on arvoja.

For-silmukalla tehtävät toistorakenteet voidaan usein tehdä myös seuraavaksi esiteltävien `vapply()` ja `sapply()`-funktioiden avulla. Nämä funktiot ovat äsken esitellyn `lapply`-funktion wrappereita.

Ensimmäisena argumenttina `sapply()`:lle annetaan vektori, ja toisena funktio. Toisena argumenttina annettua funktiota sovelletaan jokaiseen ensimmäisenä argumenttina annettuun vektorin arvoon, aivan kuten for-silmukassa. Lopuksi palautetuista arvoista muodostetaan vektori, jonka `sapply()` palauttaa:

Esimerkki 6.21.

```
> sapply(1:10, function(x) { return(x^2) })

 [1]   1   4   9  16  25  36  49  64  81 100
```

Tässä esimerkissä siis `sapply()` sijoitti luvut 1:stä 10:een funktioon, joka nostaa sen argumentin toiseen potenssiin.

Tämä voidaan esittää myös kompaktimmassa muodossa:

Esimerkki 6.22.

```
> sapply(1:10, function(x) x^2)

 [1]   1   4   9  16  25  36  49  64  81 100
```

Jos toiseksi argumentiksi annettava funktio on määritelty etukäteen, ja tarvitsee vain yhden argumentin (eli sillä on korkeintaan yksi formaali argumentti, jolle ei ole määritelty oletusarvoa), niin `sapply()`:lle voidaan antaa toiseksi argumentiksi pelkkä funktion nimi (aivan kuten `apply`:llekin. Esimerkikiksi seuraava komento, joka ottaa luvuista yhdestä viiteen luonnollisen logaritmin:

Esimerkki 6.23.

```
> sapply(1:5, function(x) log(x))

[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

voidaan kirjoittaa kompaktimmin seuraavasti:

Esimerkki 6.24.

```
> sapply(1:5, log)

[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

Jos taas valmiiksi määritellylle funktiolle halutaan antaa ylimääräisiä argumentteja, sekin onnistuu lisäämällä ne ylimääräisiksi argumenteiksi `sapply()`:lle, joka välittää ne edelleen sovellettavalle funktiolle. Jos halutaankin ottaa 2-kantainen logaritmi luonnollisen logaritmin sijaan (funktion `log()` kannan määrittävän `base`-argumentin oletusarvo on

`exp(1)`, eli Neperin luku e), kyseinen komento:

Esimerkki 6.25.
```
> sapply(1:5, function(x) log(x, base = 2))

[1] 0.000000 1.000000 1.584963 2.000000 2.321928
```

voidaan lyhentää seuraavasti:

Esimerkki 6.26.
```
> sapply(1:5, log, base = 2)

[1] 0.000000 1.000000 1.584963 2.000000 2.321928
```

Esimerkki 6.27. Lasketaan nyt aiemman esimerkin saraketulot funktion `sapply()` avulla.

```
> sapply(1:ncol(A),function(x) prod(A[,x]))

 [1]       120     30240    360360   1860480   6375600  17100720  38955840
 [8]  78960960 146611080 254251200
```

Funktio `vapply` on hyvin samanlainen kuin `sapply`, mutta sille tulee määrittää jo aluksi miten se palauttaa tulokset. Näin ollen `vapply` voi olla `sapply`:a turvallisempi (ja joskus myös nopeampi).

Esimerkki 6.28. Lasketaan aiemman esimerkin saraketulot funktion `vapply()` avulla. Oletetaan, että tulokset halutaan numeerisena vektorina (double).

```
> vapply(1:ncol(A),function(x) prod(A[,x]),FUN.VALUE=numeric(1))

 [1]       120     30240    360360   1860480   6375600  17100720  38955840
 [8]  78960960 146611080 254251200
```

Oletetaan sitten, että tulokset halutaankin kokonaislukuina (tietotyyppi integer).

```
> vapply(1:ncol(A),function(x) prod(A[,x]),FUN.VALUE=integer(1))

Error in vapply(1:ncol(A), function(x) prod(A[, x]), FUN.VALUE = integer(1)) :
  values must be type 'integer',
 but FUN(X[[1]]) result is type 'double'
```

Jos siis halutaan varmistua siitä, että tulos on haluttua tyyppiä, esim. kokonaislukuja, niin funktiolla `vapply()` tämä voidaan aina varmistaa. Ylläoleva virheviesti on hyödyllinen, sillä se kertoo heti, että palautusarvo ei ole kokonaislukutyyppiä. Esimerkiksi funktiolla `sapply()` laskiessa tämä olisi voinut jäädä huomaamatta. Nyt koodi voidaan virheviestin ansiosta korjata halutuksi:

```
> vapply(1:ncol(A),function(x) as.integer(prod(A[,x])),FUN.VALUE=integer(1))

 [1]       120     30240    360360   1860480   6375600  17100720  38955840
 [8]  78960960 146611080 254251200
```

6.2.4 tapply()

Joskus on tarvetta laskea tunnuslukuja osa-aineistoittain. Tämä onnistuu kätevästi myös R:stä vakiona löytyvällä `apply`-perheen funktiolla `tapply()`. Käytettäessä `tapply()`-funktiota osa-aineistoja ei valita erikseen käsin, vaan `tapply()` tekee sen automaattisesti. Seuraavassa esimerkissä sovelletaan funktiota `tapply()` R:stä valmiiksi löytyvään aineistoon `warpbreaks`.

Esimerkki 6.29. Tarkastellaan R:stä valmiiksi löytyvää aineistoa `warpbreaks`, jossa tutkitaan kangaspuita ja loimilankoja. Lasketaan loimilankojen määrien keskiarvot langan pingottuneisuuden mukaan: `L` (matala), `M` (keskitaso) ja `H` (korkea).

```
> tapply(warpbreaks$breaks,warpbreaks$tension,mean)

       L        M        H
36.38889 26.38889 21.66667
```

Funktio `tapply()` hajoittaa ensin ensimmäisenä argumenttina annetun vektorin, tässä tapauksessa aineiston sarakkeen `breaks`, ryhmiin toisena argumenttina annetun faktorin, tässä tapauksessa aineiston sarakkeen `tension`, mukaan. Sen jälkeen `tapply()` suorittaa kolmantena argumenttina annetun funktion, tässä tapauksessa `mean()`:in, kuhunkin osa-aineistoon ja palauttaa tuloksen nimettynä arrayna. Vaikka tuloste näyttää nimetyltä vektorilta, niin kyseessä on todellisuudessa array. Funktion palauttama olio voidaan kuitenkin muuttaa haluttuun luokkaan esimerkiksi `as.vector` tai `as.data.frame` -funktioilla. Jos käytettävä funktio ei ole sellainen, että se palauttaa atomisen arvon, tulos palautetaan listana. Kokeile esimerkiksi soveltaa samaan aineistoon funktiota `summary()` funktion `mean()` sijaan.

6.3 If else - rakenne

Ehtolauseita voi ohjelmoida R:ssä `if else`-rakenteella. Tulostetaan esimerkiksi näytölle, kumpi muuttujien `a` ja `b` arvoista on suurempi. Samalla esitellään myös funktio `cat`, joka tulostaa näytölle kaikki sille argumentteina annetut muuttujat, tässä tapauksessa muuttujan `a` arvon, sen jälkeen merkkijonon `on pienempi tai yhtä suuri kuin`, ja sen jälkeen muuttujan `b` arvon.

Esimerkki 6.30.

```
> a <- 5
> b <- 10
> if(a > b) {
    cat(a,"on suurempi kuin",b)
  } else {
    cat(a,"on pienempi tai yhtä suuri kuin", b)
  }

5 on pienempi tai yhtä suuri kuin 10
```

Jos `if`-lauseen ehto, tässä tapauksessa `a > b`, on totta, suoritetaan `if`-osan jälkeen aaltosuluissa oleva osa. Jos taas se on epätosi, suoritetaan `else`-osan jälkeen aaltosuluissa oleva osa, eli tulostetaan että `a`:n arvo on pienempi tai yhtäs suuri kuin `b`:n arvo.

`Else`-osa ei ole pakollinen, vaan voidaan käyttää pelkästään `if`-osaa. Tällöin jos ehto on epätosi, mitään ei tapahdu; esimerkiksi seuraava komento ei tulosta mitään.

Esimerkki 6.31.

```
> a <- 5
> b <- 10
> if(a > b) {
    cat(a,"on suurempi kuin",b)
  }
```

Jos halutaan testata useampaa ehtoa peräkkäin, voidaan lisätä `if`-lauseita. Esimerkiksi seuraavassa testataan ensin onko `a` suurempaa kuin `b`, ja jos ei ole, testataan onko se suurempaa kuin `b`. Jos tämäkään ei pidä paikkaansa, toteutetaan lopulta `else`-osa, eli tulostetaan että luvut ovat yhtä suuret.

Esimerkki 6.32.

```
> a <- 10
> b <- 10
```

```
> if(a > b) {
    cat(a,"on suurempi kuin",b)
  } else if(a < b){
    cat(a,"on pienempi kuin", b)
  } else {
    cat(a,"on yhtä suuri kuin", b)
  }

10 on yhtä suuri kuin 10
```

Kuten funktioiden ja `for`-silmukoiden tapauksessa, aaltosuluissa oleva osa on yleensä tapana sisentää, kuten yllä olevissa esimerkeissä. Sen sijaan aaltosulkujen poisjättäminen ei onnistu samalla tavalla: jos `if`-osan aaltosulut jättää kirjoittamatta, niin R ei osaa arvata, että tulossa on vielä `else`-osa, ja antaa virheilmoituksen. Esimerkiksi seuraava koodi ei ajettuna toimi, vaan antaa virheilmoituksen `Error: unexpected 'else' in "else"`.

Esimerkki 6.33.

```
# Huom. ei toimi!
if(a > b)
  cat(a,"on suurempi kuin",b)
else
  cat(a,"on pienempi tai yhtä suuri kuin", b)
```

Jos aaltosulut haluaa jättää pois, koko `if else`-rakenne on kirjoitettava yhdelle rivil-le seuraavaan tapaan. Koska välissä ei ole rivinvaihtoa, R ei katkaise rakennetta ennen `else`:ä.

Esimerkki 6.34.

```
> if(a > b) cat(a,">",b) else cat(a,"<=", b)

10 > 5
```

R:ssä myös `if else`-rakenne on funktio, joten se palauttaa arvon. Tämä arvo on sen aaltosuluissa olevan (tai sen osan, joka kirjoitettaisiin aaltosulkuihin, jos ne kirjoitettaisiin näkyviin) osan, joka toteteutaan, viimeinen käsittelemä arvo. Tätä voidaan hyödyntää esimerkiksi valitsemalla suurempi luvuista `a` ja `b` ja sijoittamalla se muuttujaan `suurempi`.

Esimerkki 6.35.

```
> a <- 10
> b <- 5
> suurempi <- if(a > b) a else b
```

```
> suurempi

[1] 10
```

Monet aloittelevat R-ohjelmoijat, joilla on taustaa muista kielistä, käyttävät usein turhan paljon `for`-silmukoita ja `if else`-rakenteita, kun usein samat operaatiot ovat totetettavissa helpommin ja nopeammin R:n omien vektorisoitujen operaatioiden tai `apply`-perheen funktioiden avulla. Aina kuitenkaan tämä ei ole mahdollista, esimerkiksi monimutkaisempia simulaatioita voi olla hankala vektorisoida, ja ne voi olla helpompaa toteuttaa silmukoilla ja `if else`-valintarakenteilla.

6.4 Lisää esimerkkejä omista funktioista ja silmukoista

Esimerkki 6.36. Oletetaan, että erään älykkyystestin keskiarvo on 100 pistettä ja että tulokset ovat normaalijakautuneet väestössä. Tehdään funktio, joka ottaa argumentteinaan älykkyystestin tuloksen sekä väestön keskihajonnan. Funktion tulee kertoa yhden desimaalin tarkkuudella, kuinka monella prosentilla väestöstä älykkyystestin tulos on huonompi kuin argumenttina annettu pistemäärä.

Hyödynnetään prosenttiosuuden laskemisessa normaalijakauman kertymäfunktiota `pnorm` ja asetetaan keskihajonnan oletusarvoksi 15.

```
alykkyystesti <- function(tulos,keskihajonta=15) {
  prosentti <- round(100*pnorm(tulos,mean=100,sd=keskihajonta),digits = 1)
  paste0("Tulos on parempi kuin ", prosentti, " prosentilla väestöstä")
}

> alykkyystesti(tulos=129)

[1] "Tulos on parempi kuin 97.3 prosentilla väestöstä"

alykkyystesti(tulos=129,keskihajonta=14)

[1] "Tulos on parempi kuin 98.1 prosentilla väestöstä"
```

Esimerkki 6.37. Eurojackpot-pelissä yksi pelirivi koostuu 5:stä päänumerosta, jotka on valittu luvuista 1-50 ilman takaisinpanoa, sekä kahdesta tähtinumerosta, jotka on valittu lukujen 1-10 väliltä ilman takaisinpanoa. Lisäksi pelin yhteydessä on Jokeri-peli, jossa

yhteen peliriviin valitaan 7 numeroa väliltä 0-9 takaisinpanolla. Luodaan oma funktio, joka arpoo pelaajalle pelirivejä hänen haluamansa määrän n sekä lisäksi Jokeri-rivejä hänen haluamansa määrän m.

Arvotaan funktiossa ensin päänumerosarjoja ja tähtinumerosarjoja n kappaletta käyttäen funktiota `sapply`, joka palauttaa matriisin, jonka sarakkeet vastaavat pelirivien numerosarjoja. Jokeri-peli on vapaaehtoinen lisäpeli, joten jos Jokeri-rivien määrää ei anneta, niitä ei arvota lainkaan. Annetaan niiden määrälle siis oletusarvoksi 0.

Koska päänumerosarjat, tähtinumerosarjat ja Jokeri-pelin numerosarjat ovat eri pituisia vektoreita, niitä ei voi yhdistää järkevästi matriisiksi. Niinpä funktion kannattaa palauttaa tulokset listana, jonka ensimmäisessä komponentissa ovat päänumerosarjat, toisessa tähtinumerosarjat ja kolmannessa Jokeri-pelin numerosarjat matriisina. Palautetaan tulokset siten, että yksittäisten pelirivien numerosarjat vastaavat kunkin matriisin rivejä, eli transponoidaan matriisit.

```
eurojackpot <- function(n,m=0) {
  paanumerot <- sapply(1:n, function(x) sort(sample(1:50,5,replace=F)))
  tahtinumerot <- sapply(1:n, function(x) sort(sample(1:10,2,replace=F)))
  if(m>0) {
    jokerit <- sapply(1:m,function(x) sample(0:9,7,replace=T))
    return(list(Paanumerot=t(paanumerot),Tahtinumerot=t(tahtinumerot),
                Jokeri=t(jokerit)))
  }
  return(list(Paanumerot=t(paanumerot),Tahtinumerot=t(tahtinumerot)))
}
```

Kokeillaan arpoa pelaajalle 5 riviä Eurojackpot-peliä ja kaksi Jokeri-riviä.

```
> eurojackpot(5,2)

$Paanumerot
     [,1] [,2] [,3] [,4] [,5]
[1,]    6   14   15   37   50
[2,]   29   32   36   43   50
[3,]    8   19   21   30   42
[4,]    2    6   28   43   46
[5,]   34   35   41   48   49

$Tahtinumerot
```

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

$Jokeri
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2    3    5    5    0    4
[2,]    7    3    5    2    0    5    5
```

Jos Jokeri-rivien määrää ei anneta, niitä ei arvota:

```
> eurojackpot(5)

$Paanumerot
     [,1] [,2] [,3] [,4] [,5]
[1,]    7   15   21   32   36
[2,]   19   21   37   41   43
[3,]   18   24   34   35   46
[4,]   16   18   19   35   47
[5,]    7   25   34   44   45

$Tahtinumerot
     [,1] [,2]
[1,]    1    2
[2,]    7    9
[3,]    8    9
[4,]    8    9
[5,]    9   10
```

Esimerkki 6.38. Jatkoa edelliseen esimerkkiin. Tehdään funktio, joka tarkistaa, montako oikein osunutta numeroa kussakin Jokeri-rivissä on. Oletetaan, että oikea Jokeri-pelin rivi on $5, 1, 2, 0, 4, 9, 9$. Jokeri-pelissä numeroiden järjestyksellä on merkitystä, eli esim. rivillä $9, 9, 0, 4, 5, 6, 9$ oikeita numeroita olisi yksi (viimeinen). Funktio saa argumenttina edellisen esimerkin funktion palauttaman listan sekä oikean Jokeri-pelin rivin.

```
> tarkista_jokeri <- function(lista,oikea) {
  apply(lista$Jokeri,1,function(x) sum(x==oikea))
}
> oma <- eurojackpot(10,2)
> oma

$Paanumerot
      [,1] [,2] [,3] [,4] [,5]
 [1,]    1    4    6   22   46
 [2,]    4   26   34   41   45
 [3,]    3   11   23   25   27
 [4,]    2    6    7   17   32
 [5,]    1   17   35   47   50
 [6,]    8    9   10   23   49
 [7,]   11   17   37   46   48
 [8,]   10   27   28   35   49
 [9,]   19   27   38   42   50
[10,]    5   17   19   38   40

$Tahtinumerot
      [,1] [,2]
 [1,]    2    3
 [2,]    8   10
 [3,]    5    6
 [4,]    1    2
 [5,]    6    9
 [6,]    6    9
 [7,]    4    7
 [8,]    2    9
 [9,]    1    9
[10,]    8   10

$Jokeri
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    7    8    2    0    1    1    5
[2,]    5    1    2    3    5    7    2


> oikea_jokeri <- c(5,1,2,0,4,9,9)
```

```
> tarkista_jokeri(oma,oikea_jokeri)

[1] 2 3
```

Funktio palauttaa siis vektorin, jossa on kaikkien Jokeri-rivien oikein osuneiden numeroiden määrät. Tässä tapauksessa 2 ja 3.

6.5 Keskeisiä funktioita ja toimintoja

Funktioiden toimintaan ja funktioille annettaviin argumentteihin voit tutustua tarkemmin R:n ohjekirjasta.

Oman funktion luominen:

```
funktion_nimi <- function(arg1,arg2,...) {
  ... # Kaikki funktion tekemät asiat aaltosulkujen väliin
  palautusarvo # Funktion palauttama arvo viimeiselle riville
}
```

For-silmukka:

```
for(i in vektori) {
  ... # Toistettava operaatio aaltosulkujen väliin.
  # Jokaisen toiston jälkeen i saa seuraavan arvon vektorista
}
```

Ehtolauseet: `ifelse(ehto,arvoJosTosi,arvoJosEpätosi)`

```
if(ehto) {
  arvoJosTosi
} else {
  arvoJosEpätosi
}
```

Saman funktion soveltaminen matriisin tai taulukon riveille tai sarakkeille: `apply()`
Saman operaation toistaminen kaikilla objektin arvoilla (for-silmukka kompaktimmin): `lapply()`, `sapply()`, `vapply()`

Funktion soveltaminen vektorin arvoihin erikseen faktorin mukaan:
`tapply()`

Kappale 7

Aineiston tarkastelu

Tässä kappaleessa käsitellään erityisesti taulukkomuotoisen aineiston käsittelyä. Usein on tarpeellista tutkia aineiston muuttujien tyyppejä ja jakaumia ennen varsinaisia analyysejä.

7.1 Aineiston kuvailu

Taulukkomuotoisessa aineistossa taulukon yksittäinen rivi vastaa usein yhtä havaintoyksikköä ja yksittäinen sarake muuttujaa, joka voi olla esimerkiksi numeerinen vektori, merkkijono tai faktori. Jatkuvat ja järjestysasteikolliset muuttujat ovat yleensä numeerisia, kun taas luokitteluasteikolliset muuttujat on mielekästä esittää faktoreina. Tietyissä tapauksissa myös järjestysasteikolliset muuttujat on kätevämpää esittää faktoreina.

Melko hyvän yleiskatsauksen aineistosta saa `str`-funktiolla. Jos aineisto on taulukkomuotoinen, `str`-funktio listaa havaintojen ja muuttujien määrän sekä muuttujien nimet ja luokat. Tutkitaan esimerkeissä R:n `MASS`-kirjastosta valmiiksi löytyvää aineistoa `Aids2`.

Esimerkki 7.1.

```
> library(MASS)
> str(Aids2)

'data.frame': 2843 obs. of  7 variables:
 $ state  : Factor w/ 4 levels "NSW","Other",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ sex    : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ diag   : int  10905 11029 9551 9577 10015 9971 10746 10042 10464 10439 ...
 $ death  : int  11081 11096 9983 9654 10290 10344 11135 11069 10956 10873 ...
 $ status : Factor w/ 2 levels "A","D": 2 2 2 2 2 2 2 2 2 2 ...
 $ T.categ: Factor w/ 8 levels "hs","hsid","id",..: 1 1 1 5 1 1 8 1 1 2 ...
 $ age    : int  35 53 42 44 39 36 36 31 26 27 ...
```

Lyhyen katsauksen muuttujien jakaumista saa `summary`-funktiolla. Funktio ymmärtää tehdä katsauksen muuttujasta sen luokan mukaan. Esimerkiksi faktoriluokkaisista muuttujista, jotka ovat luokitteluasteikollisia, `summary`-funktio tekee frekvenssitaulun, kun taas numeeristen muuttujien jakaumia se tarkastelee kvantiilien ja keskiarvon avulla, jotka antavat usein hyvän perusnäkemyksen jatkuvien ja järjestysasteikollisten muuttujien jakaumista.

Esimerkki 7.2.

```
> summary(Aids2)
```

```
   state       sex            diag           death        status      T.categ
 NSW  :1780   F:  89   Min.   : 8302   Min.   : 8469   A:1082   hs     :2465
 Other: 249   M:2754   1st Qu.:10163   1st Qu.:10672   D:1761   blood  :  94
 QLD  : 226            Median :10665   Median :11235            hsid   :  72
 VIC  : 588            Mean   :10584   Mean   :10990            other  :  70
                       3rd Qu.:11103   3rd Qu.:11504            id     :  48
                       Max.   :11503   Max.   :11504            haem   :  46
                                                                (Other):  48
      age
 Min.   : 0.00
 1st Qu.:30.00
 Median :37.00
 Mean   :37.41
 3rd Qu.:43.00
 Max.   :82.00
```

Funktiota `summary` voi soveltaa myös suoraan yksittäisiin sarakkeisiin.

Esimerkki 7.3.

```
> summary(Aids2$diag)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8302   10163   10665   10584   11103   11503
```

Funktiolla `head` voidaan tulostaa aineiston muutama ensimmäinen rivi, jolloin myös saadaan jonkinlainen käsitys siitä, millainen aineisto on kyseessä.

Esimerkki 7.4.

```
> head(Aids2)
```

```
  state sex  diag death status T.categ age
1   NSW   M 10905 11081      D      hs  35
2   NSW   M 11029 11096      D      hs  53
3   NSW   M  9551  9983      D      hs  42
4   NSW   M  9577  9654      D    haem  44
5   NSW   M 10015 10290      D      hs  39
6   NSW   M  9971 10344      D      hs  36
```

7.2 Frekvenssitaulu ja ristiintaulukointi

Luokitteluasteikollisia muuttujia on usein mielekästä tarkastella frekvenssitaulun avulla. R:ssä tämä voidaan tehdä funktiolla `table`. Funktiolle annetaan argumentiksi vektori, jonka arvoista halutaan laskea frekvenssit.

Esimerkki 7.5.

```
> arvosanat <- c(5,4,3,2,1,2,4,3,2,2,5,5,5,3,2,2,1,NA,4,2)
> table(arvosanat)

arvosanat
1 2 3 4 5
2 7 3 3 4
```

Mikäli frekvenssitauluun halutaan laskea myös puuttuvien arvojen määrät mukaan, käytetään lisäargumenttia `useNA`. Argumentille voidaan antaa arvoksi merkkijono `"ifany"`, jos halutaan sen laskevan puuttuvien määrä taulukkoon, mikäli niitä on vektorissa yksikin. Jos argumentin arvoksi annetaan `"always"`, niin frekvenssitauluun tulee kohta puuttuville myös, vaikka niitä olisi 0.

Esimerkki 7.6.

```
> table(arvosanat,useNA="ifany")

arvosanat
   1    2    3    4    5 <NA>
   2    7    3    3    4    1
```

Esimerkki 7.7.

```
> table(Aids2$T.categ,useNA="always")

    hs   hsid     id    het   haem  blood mother  other   <NA>
```

```
  2465     72     48     41     46     94      7     70      0
```

Usein frekvenssien sijasta on mielekkäämpää tarkastella suhteellisia osuuksia koko joukosta. Tähän voidaan käyttää funktiota `prop.table`, joka ottaa argumenttinaan frekvenssitaulun, josta lasketaan osuudet.

Esimerkki 7.8.

```
> prop.table(table(Aids2$sex))

         F          M
0.03130496 0.96869504

> prop.table(table(arvosanat))

arvosanat
        1         2         3         4         5
0.1052632 0.3684211 0.1578947 0.1578947 0.2105263

> prop.table(table(arvosanat,useNA="ifany"))

arvosanat
   1    2    3    4    5 <NA>
0.10 0.35 0.15 0.15 0.20 0.05
```

Myös kahden muuttujan yhteisjakauman tarkastelu onnistuu `table()`-funktiolla. Taulua, joka sisältää kahden muuttujan arvojen yhdistelmien aineistossa saamat määrät, kutsutaan *ristiintaulukoksi* (jatkossa myös välillä lyhyemmin *tauluksi*). Ristiintaulukoinnissa `table`-funktiolle annetaan argumentteina kaksi samanpituista vektoria, joista ensimmäinen tulee taulun riveille ja toinen sarakkeille.

Esimerkki 7.9.

```
> table(Aids2$sex,Aids2$T.categ)

     hs hsid   id  het haem blood mother other
  F   1    0   20   20    0    37      4     7
  M 2464   72   28   21   46    57      3    63
```

Funktiota `prop.table` voidaan soveltaa myös ristiintaulukoinnissa.

Esimerkki 7.10.

```
> prop.table(table(Aids2$sex,Aids2$status))
```

```
            A          D
F 0.01266268 0.01864228
M 0.36792121 0.60077383
```

Jos halutaan tutkia osuuksia vain riveittäin tai sarakkeittain, annetaan `prop.table` funktiolle `margin`-argumentiksi joko 1 tai 2 . Luku 1 viittaa riveihin ja luku 2 sarakkeisiin.

Esimerkki 7.11. Lasketaan edellisen esimerkin riviprosentit.

```
> prop.table(table(Aids2$sex,Aids2$status),margin=1)

           A         D
F 0.4044944 0.5955056
M 0.3798112 0.6201888
```

Nämä voisi hyvin muuttaa myös oikeasti prosenteiksi kertomalla luvut sadalla.

```
> prop.table(table(Aids2$sex,Aids2$status),margin=1)*100

          A        D
F 40.44944 59.55056
M 37.98112 62.01888
```

Moniulotteisen taulukon eli arrayn reunajakaumien tarkastelu onnistuu funktiolla `margin.table()`. Funktiolle annetaan ensimmäisenä argumentteina array ja toisena argumenttina `margin`, jossa määritellään numerolla, minkä suhteen reunajakaumaa tarkastellaan. Luku 1 vastaa rivejä, luku 2 sarakkeita, luku 3 kolmatta ulottuvuutta jne. Esimerkiksi rivien reunajakauman saa antamalla `margin`-argumentiksi 1 tai rivien ja sarakkeiden yhteisreunajakauman antamalla argumentiksi vektorin `c(1,2)`.

Esimerkki 7.12. Tarkastellaan R:stä valmiiksi löytyvää aineistoa `HairEyeColor`, jossa on ristiintaulukoituna 592 opiskelijan hiusten värit, silmien värit ja sukupuolet kolmiulotteisessa muodossa eli arrayna.

```
> HairEyeColor

, , Sex = Male
```

```
       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

Tarkastellaan ensin silmien värin reunajakaumaa, eli taulukoidaan silmien värit välittämättä hiusten väristä ja sukupuolesta.

```
> margin.table(HairEyeColor,2)

Eye
Brown  Blue Hazel Green
  220   215    93    64
```

Tarkastellaan sitten hiusten ja silmien värin reunayhteisjakaumaa, eli ristiintaulukoidaan hiusten väri ja silmien väri välittämättä sukupuolesta.

```
> margin.table(HairEyeColor,c(1,2))

       Eye
Hair    Brown Blue Hazel Green
  Black    68   20    15     5
  Brown   119   84    54    29
  Red      26   17    14    14
  Blond     7   94    10    16
```

Kumpikin edellisistä olisi voitu tehdä myös `apply`-funktion avulla summaamalla kyseisiä indeksejä vastaavat arvot. Funktiolle `apply` voi nimittäin antaa indeksiksi myös vektorin, jos taulukko on moniulotteinen eli array. Funktio `margin.table` ei siis oikeastaan tee mitään sellaista, mitä funktiolla `apply` ei voisi tehdä.

7.3 Aineiston luokittelu

Aineiston luokittelu tulee tarpeen erityisesti jatkuvien muuttujien (ikä, pituus, paino...) tapauksessa. Luokittelu voidaan tehdä käyttäen funktiota `cut()`, joka palauttaa annetun aineiston faktorina. Tarkastellaan seuraavaksi `cut()`-funktion toimintaa esimerkein:

Esimerkki 7.13. Luokitellaan aineiston `Aids2` muuttuja `age` viiteen kategoriaan.

```
> age <- Aids2$age
> age2 <- cut(age,breaks=c(0,18,24,35,64,82))
> summary(age2)

(0,18] (18,24] (24,35] (35,64] (64,82]    NA's
     33     120    1151    1492      43       4

> str(age2)

 Factor w/ 5 levels "[0,18]","(18,24]",..: 3 4 4 4 4 4 4 3 3 3 ...
```

Huomaa: Mikäli jonkin solun arvo ei ole luokitteluun annetulla välillä, solun arvoksi tulee `NA`.

Esimerkki 7.14. Jos luokitteluvälien halutaan olevan suljettuja vasemmalta ja avoimia oikealta (tai päin vastoin), voidaan käyttää argumenttia `right`.

```
> age3 <- cut(age,breaks=c(0,18,24,35,64,82),right = FALSE)
> summary(age3)

[0,18) [18,24) [24,35) [35,64) [64,82)    NA's
     34      71    1086    1602      49       1
```

7.4 Päivämäärien käsittely

Päivämäärämuuttujille on R:ssä oma luokka `Date`. Aineistossa päivämäärä saattaa olla merkkijonomuodossa ja jos halutaan hyödyntää esimerkiksi päivämääriin liittyviä ominaisuuksia, kuten laskea aikaeroja päivämäärien välillä, voidaan muuttujan luokka muuttaa `Date`-tyyppiseksi. Tämä onnistuu funktiolla `as.Date()`. R olettaa muutettavan päivämäärän olevan merkkijonona muodossa `"vuosi-kuukausi-päivä"` tai `"vuosi/kuukausi/päivä`. Muussa tapauksessa muoto tulee kertoa funktiolle `format`-argumentilla.

Esimerkki 7.15. Luodaan merkkijonovektori, jonka alkiot ovat päivämääriä ja muutetaan se R:ssä päivämääräksi siten, että R osaa käsitellä sitä päivämääränä.

```
> a <- c("2017-06-02","2018-04-01","2018-01-10","2017-11-18")
> summary(a)

   Length     Class      Mode
        4 character character

> a <- as.Date(a)
> summary(a)

        Min.      1st Qu.       Median         Mean      3rd Qu.         Max.
"2017-06-02" "2017-10-06" "2017-12-14" "2017-11-22" "2018-01-30" "2018-04-01"

# Lasketaan ero kolmannen ja neljännen päivämäärän välillä
> a[3]-a[4]

Time difference of 53 days

# Lasketaan päivämäärät vuodelta 2018
> sum(a >= "2018-01-01")

[1] 2
```

Esimerkki 7.16. Muutetaan suomalaisessa muodossa oleva päivämäärä R:n ymmärtämäksi päivämääräksi.

```
> b <- c("2.6.2017","1.4.2018","10.1.2018","18.11.2017")
> b <- as.Date(b,format="%d.%m.%Y")
> summary(b)

        Min.      1st Qu.       Median         Mean      3rd Qu.         Max.
"2017-06-02" "2017-10-06" "2017-12-14" "2017-11-22" "2018-01-30" "2018-04-01"
```

Funktiosta `as.Date` ja siitä, miten `format`-argumentti tulee sille syöttää, voi lukea lisää R:n manuaalista komennolla `?as.Date`.

7.5 Tunnuslukujen laskeminen aineistosta

Usein aineistosta halutaan vertailla eri ryhmiin liittyviä tilastollisia tunnuslukuja. Tähän tarkoitukseen sopii hyvin edellisessä luvussa esitelty `tapply()`-funktio. Tämän lisäksi tunnuslukujen laskemiseen osa-aineistoittain löytyy R:n vakiofunktioista myös `aggregate()`. Lasketaan nyt esimerkki 6.29 uudelleen funktion `aggregate()` avulla.

Esimerkki 7.17.

```
> keskiarvot <- aggregate(breaks~tension,FUN=mean,data=warpbreaks)
> keskiarvot

 tension   breaks
1       L 36.38889
2       M 26.38889
3       H 21.66667

> str(keskiarvot)

'data.frame': 3 obs. of  2 variables:
 $ tension: Factor w/ 3 levels "L","M","H": 1 2 3
 $ breaks : num  36.4 26.4 21.7
```

Toimintatavaltaan se vastaa hyvin läheisesti `tapply()`:ä; funktio jakaa ensimmäisessä argumentissa ~-merkin vasemmanpuoleiset vektorit ryhmiin oikeanpuolisten vektorien mukaisesti, ja suorittaa sitten kullekin näin saadulle osa-aineistolle argumenttina `FUN` annetun funktion. Lopulta `aggregate()` palauttaa saadut tulokset yhtenä data frame -taulukkona. Tästä johtuen `aggregate()` soveltuu `tapply()`:ä paremmin tilanteisiin, jossa näin saatuja osajoukkojen tunnuslukuja halutaan vielä käsitellä lisää. Erityisesti silloin, kun aineistoa halutaan ryhmitellä vähintään kolmen muuttujan perusteella, saattavat `tapply()`:n tuottamat nimetyt arrayt vaatia jonkun verran työstämistä ennen kuin niitä voidaan syöttää järkevästi argumentteina tiettyihin funktioihin.

7.6 Hyödyllisiä funktioita

Yleiskatsaus aineistoon tai yksittäisiin muuttujiin: `summary()`, `head()`, `fivenum(muuttuja)`

Jatkuvan muuttujan jakauman tarkastelu graafisesti: `hist()`, `boxplot()`
Kahden muuttujan yhteisjakauman tarkastelu hajontakuvan avulla:
`plot(Aineisto$muuttuja1,Aineisto$muuttuja2)`

Hajontakuvamatriisi: `pairs()`

Frekvenssitaulu/Ristiintaulukointi: `table()`
Reunafrekvenssit/reunayhteisjakaumat arraysta: `margin.table()`
Taulu, jossa näkyy suhteelliset osuudet: `prop.table()`

Tunnuslukujen (esim. keskiarvon) laskeminen ryhmittäin: `aggregate()`, `tapply()`
Rivikeskiarvot: `rowMeans()`
Sarakekeskiarvot: `colMeans()`

(Jatkuvan) muuttujan luokittelu: `cut()`

Päivämääräksi muuttaminen (jotta R ymmärtää käsitellä muuttujaa päivämääränä): `as.Date()`

Kappale 8

Tilastollisia työkaluja

Tässä kappaleessa esitellään yleisimpiä valmiita työkaluja tilastollisiin testeihin, luottamusväleihin ja lineaariseen malliin.

8.1 t-luottamusväli

Oletetaan, että käytettävän aineiston havainnot ovat otos normaalijakaumasta tuntemattomin parametrein μ ja σ^2. Tutustutaan estimaatin $\hat{\mu}$ luottamusvälin laskemiseen. Lisätietoa estimaateista ja luottamusväleistä löytyy Tilastotiede ja R tutuksi I ja II sekä Tilastollinen päättely I-kurssien materiaaleista.

Esimerkki 8.1. Olkoon nyt havaintovektori:

```
havainnot <- c(4,5,6,5,4,3,4,5,7,6,3,4,5,3)
```

Kaksisuuntainen t-luottamusväli, monen muun asian lisäksi, saadaan laskettua komennolla `t.test()` seuraavasti:

```
> t.test(havainnot, conf.level = 0.99)

One Sample t-test

data:  havainnot
t = 13.9916, df = 13, p-value = 3.247e-09
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 3.587237 5.555620
sample estimates:
```

```
mean of x
 4.571429
```

Tässä käytettiin luottamustasoa 0.99 (`conf.level=0.99`). Luottamusvälin ylä- ja alarajat voidaan lukea kohdasta confidence interval. Tässä tapauksessa luottamusväli on siis pyöristettynä kahteen desimaaliin $[3.59, 5.56]$.

Muihin tämän funktion antamiin tuloksiin palataan myöhemmin.

Esimerkki 8.2. Edellisen esimerkin luottamusväli saadaan myös seuraavasti:

```
> a <- t.test(havainnot, conf.level = 0.99)
> a$conf.int

[1]  3.587237 5.555620
attr(,"conf.level")
[1] 0.99
```

Tämä tapa on huomattavasti kätevämpi silloin, kun ollaan kiinnostuttu yksinomaan luottamusvälistä tai halutaan päästä käsittelemään luottamusvälin ylä- ja alarajoja.

8.2 Yhden otoksen t-testi

Tutkitaan jälleen R:n mukana tulevaa klassista Iris-esimerkkiaineistoa, ja erityisesti eri kurjenmiekkalajien terälehtien pituuksia. Testataan käyttäen merkitsevyystasoa $\alpha = 0.05$, eroaako kaunokurjenmiekkojen (*iris setosa*) terälehtien keskimääräinen pituus 5:stä, eli testataan t-testillä[1]. nollahypoteesia $H_0 : \mu = 5$ kaksisuuntaista vastahypoteesia $H_1 : \mu \neq 5$ vastaan, missä μ on kaunokurjenmiekkojen terälehtien pituuden odotusarvo. Yhden otoksen t-testi tehdään edellisessä kappaleessa luottamusvälien laskemiseen käytetyllä `t.test`-funktiolla. Ensimmäiseksi argumentiksi annetaan vektori, jossa on testattava aineisto, ja argumentti `mu` määrittelee nollahypoteesiarvon μ_0. Argumentti `alternative` määrittelee, onko vastahypoteesi kaksi- vai yksisuuntainen, ja jos se on yksisuuntainen, niin kumpaan suuntaan (`"less"` pienet kriittisiä, `"greater"` suuret kriittisiä). Testi on oletusarvoisesti kaksisuuntainen, eli jos tätä argumenttia ei anneta, testi suoritetaan kaksisuuntaisella vastahypoteesilla.

Esimerkki 8.3.

```
> iris_setosa <- subset(iris, Species == 'setosa')
> t.test(iris_setosa$Sepal.Length, mu=5)
```

[1] Yhden otoksen t-testi esitellään muun muassa kurssin Tilastollinen päättely I -monisteen jaksossa 6.7 sekä Tilastotiede ja R tutuksi I -kurssin materiaalin luvussa 9

```
One Sample t-test

data:  iris_setosa$Sepal.Length
t = 0.1204, df = 49, p-value = 0.9047
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
 4.905824 5.106176
sample estimates:
mean of x
    5.006
```

Tulosteen ensimmäinen rivi kertoo, että kyseessä on yhden otoksen t-testi, jonka R suorittaa automaattisesti, jos argumentiksi annetaan vain yksi vektori. Toinen rivi kertoo t-testisuureen arvon 0.12, t-jakauman vapausasteen 49 (aineistossa oli 50 havaintoa, jolloin vapausaste on 50 - 1 = 49) ja p-arvon 0.90. Havaittu p-arvo on suurempi kuin etukäteen määritelty merkitsevyystaso $\alpha = 0.05$, joten nollahypoteesia siitä, että kaunokurjenmiekkojen terälehtien pituuden keskiarvo olisi 5, ei voida hylätä. Seuraava rivi kertoo testissä käytetyn vastahypoteesin (ja siten implisiittisesti myös nollahypoteesin). Seuraavalla rivillä on 95 prosentin luottamusväli pituuden keskiarvolle. Viimeisellä rivillä on otoskeskiarvo 5.006, joka tosiaan on huomattavan lähellä viittä.

Esimerkissä valittiin ensin kaunokurjenmiekat omaksi aineistokseen. Tämä ei tietenkään ole välttämätöntä, vaan osa-aineiston valinnan ja testin voi tehdä myös samalla rivillä. Seuraava koodi tuottaa täsmälleen saman tuloksen kuin ylläoleva.

Esimerkki 8.4.

```
> t.test(iris$Sepal.Length[iris$Species == 'setosa'], mu=5)

One Sample t-test

data:  iris$Sepal.Length[iris$Species == "setosa"]
t = 0.12036, df = 49, p-value = 0.9047
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
 4.905824 5.106176
sample estimates:
mean of x
    5.006
```

Esimerkki 8.5. Yhden otoksen t-testi, kun vastahypoteesi on yksisuuntainen. Testataan merkitsevyystasoa $\alpha = 0.05$ käyttäen, onko terälehtien keskimääräinen leveys 3.5, kun vastahypoteesi on, että terälehtien leveys on vähemmän kuin 3.5.

```
> t.test(iris$Sepal.Width[iris$Species=="setosa"],mu=3.5,alternative="less")

One Sample t-test

data:  iris$Sepal.Width[iris$Species == "setosa"]
t = -1.3431, df = 49, p-value = 0.09272
alternative hypothesis: true mean is less than 3.5
95 percent confidence interval:
     -Inf 3.517876
sample estimates:
mean of x
    3.428
```

Havaittu p-arvo on suurempaa kuin α, joten nollahypoteesia ei hylätä tällä merkitsevyystasolla.

Esimerkki 8.6. Esimerkissä 8.2 saatiin poimittua luottamusväli funktion `t.test()` palauttamasta listasta. Sama onnistuu myös esimerkiksi testisuureelle ja p-arvolle.

```
> b <- t.test(iris_setosa$Sepal.Length, mu=5)
> b$statistic

        t
0.1203621

> b$p.value

[1] 0.9046885
```

8.3 Kahden otoksen t-testi

8.3.1 Riippumattomien otosten testi

Kahden otoksen t-testiä käytetään, kun halutaan tutkia, eroavatko normaalijakauman odotusarvot toisistaan kahden eri ryhmän välillä. Tutkitaan esimerkiksi, eroaako kau-

nokurjenmiekkojen (*iris setosa*) terälehden pituuden odotusarvoparametri μ_s kirjokurjenmiekkojen (*iris versicolor*) terälehden pituuden odotusarvoparametrista μ_v. Testataan siis nollahypoteesiä $H_0 : \mu_s = \mu_v$ kaksisuuntaista vastahypoteesiä $H_1 : \mu_s \neq \mu_v$ vastaan. Määritellään etukäteen testin merkitsevyystasoksi $\alpha = 0.05$.

Tämä onnistuu jälleen `t.test`-funktiolla. Sen ensimmäiseksi argumentiksi annetaan ensimmäisen testattavan muuttujan arvot sisältävä vektori, ja toiseksi argumentiksi toisen testattavan muuttujan arvot sisältävä vektori. Koska kahden otosken t-testi testaa, poikkeaako odotusarvoparametrien erotus $\mu_s - \mu_v$ tilastollisesti merkitsevästi nollasta, testattavan nollahypoteesin määrittävä parametri `mu` voidaan jättää sen oletusarvoon `mu = 0`.

Esimerkki 8.7.

```
> t.test(iris$Sepal.Length[iris$Species == 'setosa'],
         iris$Sepal.Length[iris$Species == 'versicolor'])

Welch Two Sample t-test

data:  iris$Sepal.Length[iris$Species == "setosa"] and
       iris$Sepal.Length[iris$Species == "versicolor"]

t = -10.521, df = 86.538, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.1057074 -0.7542926
sample estimates:
mean of x mean of y
    5.006     5.936
```

Tulosteesta nähdään ensimmäiseksi, että R:ssä oletusarvona on Welchin kahden otoksen testi, eli versio jossa verrattavien ryhmien variansseja ei oleteta samaksi. Studentin t-testin, jossa ryhmien varianssit oletetaan samoiksi, saa asettamalla `t.test`-funktiolle argumentin `var.equal = TRUE`.

Seuraavaksi tulosteessa on t-testisuureen arvo $t = -10.52$. Welchin t-testin tapauksessa t:n jakauman, jonka kvantiilien arvoon testisuureen arvoa verrataan, vapausasteet lasketaan ns. Satterthwaiten approksimaatiosta, joka tässä tapauksessa antaa vapausasteeksi n. 86.537. Näiden perusteella laskettu testin p-arvo on pienempää kuin $2.2 \cdot 10^{-16}$, eli hyvin pieni [2]. Koska havaittu p-arvo on pienempää kuin etukäteen määritelty testin merkitsevyystaso $\alpha = 0.05$, nollahypoteesi odotusarvojen yhtäsuuruudesta voidaan hylätä

[2] Jos testin p-arvo on todella lähellä nollaa, R ilmoittaa että p-arvo on pienempää kuin $2.2 \cdot 10^{-16}$.

tällä merkitsevyystasolla (ja oltaisiin voitu hylätä kaikilla muillakin yleisesti käytetyillä merkitsevyystasoilla).

Lisäksi `t.test` tulostaa 95 prosentin luottamusvälin testattavien parametrien erotukselle, tässä tapauksessa noin $[-1.11, -0.75]$, ja testattavien muuttujien otoskeskiarvot.

Toinen tapa suorittaa kahden otoksen t-testi käyttämällä `t.test`-funktion avulla hyödyntää R:n *kaavoja* (formula), jotka ovat tapa esittää tutkittava tilastollinen malli kompaktisti. Kaavoissa käytetään matoviivaa erottamaan selitettävä ja selitettävä muuttuja toisistaan, siten että selitettävä muuttuja on matoviivan vasemmalla, ja selittävät muuttujat matoviivan oikealla puolella. Tässä tapauksessa vastemuuttuja on terälehden pituus, eli `Sepal.length`, ja luokitteleva muuttuja on laji, eli `Species`, joten haluttu kaava on

```
Sepal.length ~ Species
```

Huomaa, että kirjoitimme kaavaan pelkät muuttujien nimet ilman taulukon nimeä `iris`. Yleensä kaavan argumentiksi ottavat funktiot sisältävät myös argumentin `data`, joka määrittää taulukon, johon kaavan muuttujien nimet viittaavat.

Aineistossa on havaintoja kolmesta eri kurjenmiekkalajista, joten faktorilla `Species` on vielä kolmaskin taso, jota ei haluta mukaan analyysiimme[3]. Tätä muotoa funktion kutsusta käytettäessä voimme rajata tarkasteltavan osa-aineiston kauno- ja kirjokurjenmiekkoihin `t.test`-funktion `subset`-argumentilla:

Esimerkki 8.8.

```
> t.test(Sepal.Length ~ Species, data = iris,
    subset = Species == "setosa" | Species == "versicolor")

Welch Two Sample t-test

data:  Sepal.Length by Species
t = -10.521, df = 86.538, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.1057074 -0.7542926
sample estimates:
    mean in group setosa mean in group versicolor
                   5.006                    5.936
```

[3]Jos halutaan testata, poikkeavatko kolmen tai useamman ryhmän keskiarvot toisistaan, käytetään varianssianalyysiä, joka on t-testin yleistys.

Tämä tapa kutsua `t.test`-funktiota tuottaa siis täsmälleen saman analyysin kuin esimerkin 8.7 funktiokutsu.

8.3.2 Parittainen kahden otoksen t-testi

Mikäli kahdessa otoksessa havainnot ovat parittaisia ja erityisesti toisen otoksen havainnon arvo riippuu toisen otoksen havainnon arvosta, käytetään parittaista kahden otoksen t-testiä. Tämä määrätään argumentilla `paired=TRUE`.

Esimerkki 8.9. Määritellään parittaiset otokset `x` ja `y` ja testataan nollahypoteesia, jonka mukaan näitä vastaavien normaalijakautuneiden populaatioiden odotusarvot ovat samat. Vastahypoteesi on kaksisuuntainen (odotusarvojen erotus poikkeaa nollasta).

```
> x <- c(13.5,14.2,19.3,16.3,15.0,11.9)
> y <- c(13.9,14.3,20.4,16.1,15.9,11.8)
> t.test(x,y,paired=TRUE)

Paired t-test

data:  x and y
t = -1.6775, df = 5, p-value = 0.1543
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.9285477  0.1952144
sample estimates:
mean of the differences
             -0.3666667
```

8.4 Lineaarisia malleja

8.4.1 Yhden selittäjän lineaarinen regressio

Esimerkki 8.10. Tutkitaan fiktiivistä aineistoa, jossa x on ajoneuvon tankkiin laitetun polttoaineen määrä litroissa ja y kertoo montako kilometria ajoneuvolla päästiin ennen polttoaineen loppumista. Tutkitaan seuraavaa aineistoa hajontakuvan avulla.

Ajetut kilometrit (y)	Polttoaineen määrä (x)
76	8
72	9
89	11
144	18
158	19
92	10
156	20
109	14
138	17
100	12
51	6
107	13
129	16
65	7
121	15

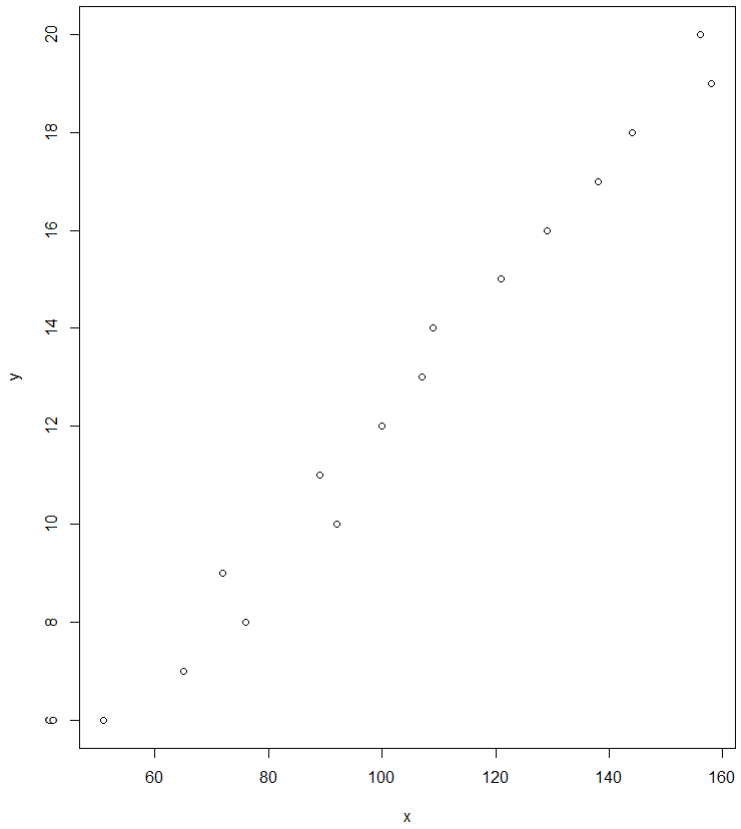
```
> y <- c(76,72,89,144,158,92,156,109,138,100,51,107,129,65,121)
> x <- c(8,9,11,18,19,10,20,14,17,12,6,13,16,7,15)
> plot(x,y)
```

Nyt tuntuu hyvin luonnolliselta ajatella, että ajettujen kilometrien määrä riippuisi tankatun polttoaineen määrästä kutakuinkin lineaarisesti. Muuttujien välisestä hajontakuvasta voidaan saada myös vahvistusta tälle intuitiolle. Kuvataan suhdetta kaavalla

$$y = \beta_0 + \beta_1 x + \varepsilon,$$

missä $\beta_0, \beta_1 \in \mathbb{R}$ ja $\varepsilon \sim N(0, \sigma^2)$ on virhetermi. Estimoidaan nyt regressiosuoran vakio β_0 ja kulmakerroin β_1 R:llä käyttäen funktiota `lm()`, jolle selitettävä ja selittävä muuttuja annetaan $\sim$-kaavamerkinnällä. Kaavamerkinnässä selitettävä muuttuja (y) tulee ennen matomerkkiä ja selittäjä (x) matomerkin jälkeen.

```
> fit <- lm(y~x)
> fit

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     10.515        7.432
```



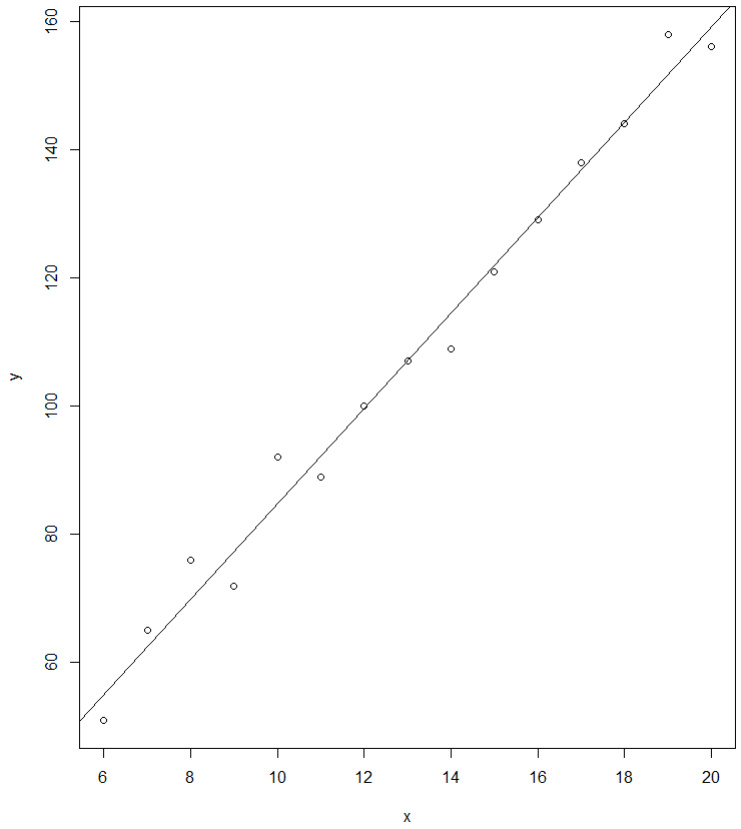
Kuva 8.1: Ajettujen kilometrien ja tankatun polttoaineen yhteys esimerkissä 8.10

Esimerkki 8.11. Tarkastellaan edellisen esimerkin estimointitulosta. Saadut suurimman uskottavuuden estimaatit ovat $\hat{\beta}_0 = 10.515$ ja $\hat{\beta}_1 = 7.432$. Piirretään hajontakuva ja lisätään edellisessä esimerkissä piirrettyyn kuvaan punaisella regressiosuora, jonka yhtälö on tässä tapauksessa

$$y = 10.515 + 7.432x.$$

Tämän voi tehdä monella tavalla ja kätevin tapa on funktion `abline` käyttäminen. Kun malli on tallennettu johonkin muuttujaan, voidaan tämä antaa `abline`-funktiolle argumentiksi, jolloin regressiosuora tulee piirretyksi olemassa olevaan kuvaan.

```
plot(x,y)
abline(fit)
```



Kuva 8.2: Ajettujen kilometrien ja tankatun polttoaineen aineisto ja siihen sovitettu regressiosuora

Tarkempaa tietoa mallista ja sen onnistumisesta saadaan käyttäen funktiota `summary()`:

```
> summary(fit)

Call:
lm(formula = y ~ x)

Residuals:
   Min     1Q Median     3Q    Max
-5.566 -3.214 -0.294  1.799  7.163
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.5155     3.4692   3.031  0.00965 **
x             7.4321     0.2532  29.348 2.88e-13 ***
---
Signif. codes:  0 ***, 0.001 **, 0.01 *, 0.05 ., 0.1

Residual standard error: 4.238 on 13 degrees of freedom
Multiple R-squared:  0.9851,Adjusted R-squared:  0.984
F-statistic: 861.3 on 1 and 13 DF,  p-value: 2.88e-13
```

Mitä kaikkea `summary()`-tuloste kertoo?

- Tiivistelmä residuaalien arvoista. Antaa hiukan kuvaa niiden jakaumasta.
- Yhden otoksen t-testit sille, ovatko regressiosuoran kulmakerroin β_1 ja vakioselittäjä β_0 nollasta poikkeava. Jos kulmakerroin β_1 on tilastollisesti merkitsevästi nollasta poikkeava, voidaan selittäjällä ajatella olevan yhteys selitettävään muuttujaan.

  Parametrin β_1 t-testissä nollahypoteesi on siis $H_0 : \beta_1 = 0$. Tämän oletuksen pohjalta parametrille lasketaan t-testisuure ja tätä vastaava p-arvo (`summary`-tulosteessa `t-value` ja `Pr(>|t|)`). p-arvon pienet arvot antavat näyttöä nollahypoteesia vastaan, ja riittävän pienellä p-arvolla nollahypoteesi voidaan hylätä. Estimaatin keskivirhe löytyy kohdasta `Std. Error`
- Mallin selitysaste (R-squared), eli kuinka paljon malli selittää selitettävän muuttujan havaintojen vaihtelua.

Esimerkki 8.12. Lasketaan vielä esimerkkitapauksen parametrien luottamusvälit. Funktio `summary()` ei näitä suoraan palauta, vaan ne saadaan laskettua funktiolla `confint()`:

```
> confint(fit)

                2.5 %     97.5 %
(Intercept) -2.2924922 -0.1085592
y            0.1227926  0.1423074
```

Funktio palauttaa oletuksena mallin parametreille saatuihin t-arvoihin pohjautuvat 95% t-luottamusvälit. Näillä väleillä on suora yhteys parametrien t-testisuureiden tulkintaan – testin p-arvo on pienempi kuin merkitsevyystaso $\alpha = 0.05$ niillä parametreilla, joiden luottamusvälit eivät sisällä arvoa 0.

8.4.2 Useamman selittäjän lineaarinen regressio

Useamman selittäjän regressiomalli sovitetaan samaan tapaan kuin yhden selittäjän malli, mutta selittäjät erotetaan kaavassa toisistaan plus-merkeillä.

Esimerkki 8.13. Kahden selittäjän malli. Tutkitaan edellisen luvun esimerkkiä polttoaineen määrän ja ajettujen kilometrien välillä, mutta lisätään aineistoon kulloisenkin ajokerran aikana mitattu ulkolämpötila.

Ajetut kilometrit (y)	Polttoaineen määrä (x)	Ulkolämpötila (z)
76	8	21.6
72	9	24.2
89	11	23.1
144	18	23.6
158	19	27.1
92	10	22.2
156	20	26.0
109	14	23.3
138	17	25.8
100	12	24.4
51	6	21.8
107	13	23.8
129	16	22.9
65	7	21.8
121	15	25.2

Muodostetaan malli nyt käyttäen funktiota `lm()`, aivan kuten yhdenkin selittäjän tapauksessa. Selittäjät tulevat matomerkin jälkeen plus-merkeillä eroteltuina.

```
> fit <- lm(y~x+z)
> fit

Call:
lm(formula = y ~ x + z)

Coefficients:
(Intercept)            x            z
    14.9273       7.4976      -0.2212

> summary(fit)
```

```
Call:
lm(formula = y ~ x + z)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7386 -3.2084 -0.6624  1.8672  7.0083

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  14.9273    23.1786   0.644    0.532
x             7.4976     0.4296  17.453  6.8e-10 ***
z            -0.2212     1.1481  -0.193    0.850
---
Signif. codes:  0 ***, 0.001 **, 0.01 *, 0.05 ., 0.1

Residual standard error: 4.404 on 12 degrees of freedom
Multiple R-squared:  0.9852,Adjusted R-squared:  0.9827
F-statistic: 398.8 on 2 and 12 DF,  p-value: 1.061e-11
```

Funktion `summary()` antamaa tulostetta tulkitaan kuten yhden selittäjän mallissa. Voidaan havaita, että muuttujaa z vastaava kerroinparametri β_2 ei ole merkitsevästi poikkeava nollasta (p-arvo 0.850). Nähdään myös, että selitysaste ei juurikaan noussut selittäjää lisäämällä. Toisaalta selitysaste oli jo ennestään erittäin korkea.

8.4.3 Mallien diagnostiikka

Lineaarisen mallin oletusten täyttyminen on aina syytä tarkistaa erityisesti, jos mallilla halutaan ennustaa uusia havaintoja tai mallin perusteella yritetään tehdä minkäänlaisia johtopäätöksiä. Tärkeintä on tutkia residuaalien jakaumaa ja niiden riippumattomuutta mallin sovittamista arvoista.

Esimerkki 8.14. Tutkitaan kahden selittäjän mallia, jossa R:n kirjastosta valmiiksi löytyvän `survey`-aineiston muuttujaa `Wr.Hnd` selitetään pituudella ja sukupuolella (dummy-muuttuja, joka saa arvon 0 tai 1 sen mukaan onko vastaaja nainen vai mies).

```
> fit <- lm(Wr.Hnd~Height+Sex,data=survey)
> summary(fit)
```

```
Call:
lm(formula = Wr.Hnd ~ Height + Sex, data = survey)

Residuals:
    Min      1Q  Median      3Q     Max
-5.5417 -0.8292  0.0513  0.9519  3.8627

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.46896    2.25101   2.874  0.00449 **
Height       0.06694    0.01356   4.937 1.64e-06 ***
SexMale      1.44573    0.26772   5.400 1.84e-07 ***
```

Selittäjien t-testeissä regressiokertoimet ovat tilastollisesti merkitsevästi nollasta poikkeavia. Tämä lupaa tietysti hyvää, mutta tutkitaan tarkemmin residuaaleja. Oletuksen mukaan residuaalit eli virhetermit ovat normaalijakautuneita odotusarvolla 0 ja vakiovarianssilla σ^2. Residuaalien ja mallin sovittamien arvojen välillä ei myöskään saisi olla mitään systemaattista riippuvuutta.

```
# Residuaaliarvojen histogrammi ja hajontakuva sovitettujen arvojen kanssa.
> hist(fit$residuals)
> plot(fit$residuals,fit$fitted.values)

# Neljä erilaista diagnostiikkakuvaa
> plot(fit)

Hit <Return> to see next plot:
```

Erilaisia diagnostiikkatarkasteluja on useita eikä niihin mennä tässä materiaalissa sen tarkemmin. Asiat ovat yleensä hyvin, jos residuaalien jakauma on lähellä nollakeskeistä normaalijakaumaa (paljon nollan lähellä olevia virhetermejä ja hyvin vähän havaintoja jakauman hännissä). Muussa tapauksessa mallin t-testien tulokset ovat kyseenalaisia. Myös jos residuaalien ja sovitteen välisessä hajontakuvassa pisteet näyttäisivät olevan sijoittuneen melko satunnaisesti ympäri kuvaa, niin mallin oletukset jäännösvaihtelun osalta täyttyvät todennäköisesti hyvin. Ei saisi olla niin, että residuaalin ja sovitteen arvot riippuisivat voimakkaasti toisistaan, sillä tämä viittaa heteroskedastisuuteen, jota lineaarisessa mallissa ei saisi olla. Mikäli mallissa havaitaan heteroskedastisuutta, voidaan siitä päästä eroon käyttämällä selittäjille tai selitettävälle muuttujalle (tai molemmille) erilaisia muunnoksia, kuten logaritmia.

8.4.4 Malleilla ennustaminen

Yksittäiselle havainnolle on helppo laskea ennuste suoraan regressiokertoimien estimaattien perusteella. Ennusteen laskemiselle on R:ssä kuitenkin myös oma `predict`-funktio, joka on kätevä erityisesti silloin, kun ennustettavia havaintoja on useampi tai halutaan laskea ennusteelle luottamus- tai ennustevälejä.

Esimerkki 8.15. Tutkitaan esimerkissä 8.14 sovitettua lineaarista mallia. Lasketaan ennuste muuttujan `Wr.Hnd` arvolle, kun uutena havaintona on 165 senttimetriä pitkä nainen. Ennusteen voisi laskea suoraan regressiokertoimien estimaateista, jotka ovat $\hat{\beta}_0 = 6.46896, \hat{\beta}_1 = 0.06694$ ja $\hat{\beta}_2 = 1.44573$. Malli antaa ennusteeksi

$$y = 6.46896 + 0.06694 \cdot 165 + 1.44573 \cdot 0 \approx 17.51.$$

Tehdään nyt sama suoraan käyttäen `predict`-funktiota.

```
> predict(fit,newdata=data.frame(Height=165,Sex="Female"))

       1
17.51482
```

Funktio `predict` saa ensimmäisenä argumenttinaan mallin (`fit`). Toinen argumentti `newdata` on taulukko, jossa on uudet havainnot. Taulukon muuttujien nimien tulee olla samat kuin selittäjien nimet mallissa, eli tässä tapauksessa `Height` ja `Sex`.

Tehdään seuraavaksi ennuste useammalle havainnolle. Otetaan mukaan 188 senttimetriä pitkä mies ja 178 senttimetriä pitkä nainen.

```
> predict(fit,newdata=data.frame(Height=c(165,188,178),
Sex=c("Female","Male","Female")))

       1        2        3
17.51482 20.50027 18.38510
```

Funktio `predict` palauttaa siis nimetyn vektorin, jossa on kullekin uudelle havainnolle tehdyt ennusteet.

`predict`-funktiolla voidaan myös laskea ennusteille luottamus- ja ennustevälejä käyttäen `interval`-argumenttia. Jos halutaan laskea luottamusvälejä, annetaan argumentille arvo `"conf"`. Jos halutaan laskea ennustevälejä, annetaan argumentille arvo `"predict"`. Argumentti `level` määrää luottamus- tai ennustetason.

```
> predict(fit,newdata=data.frame(Height=c(165,188,178),
Sex=c("Female","Male","Female")),interval="conf",level=0.99)

       fit      lwr      upr
1 17.51482 17.14543 17.88420
2 20.50027 20.01539 20.98515
3 18.38510 17.81561 18.95458

> predict(fit,newdata=data.frame(Height=c(165,188,178),
Sex=c("Female","Male","Female")),interval="predict",level=0.90)

       fit      lwr      upr
1 17.51482 15.13753 19.89210
2 20.50027 18.11462 22.88592
3 18.38510 15.99191 20.77828
```

Nyt funktio palauttaa matriisin, jonka rivit vastaavat jokaista uutta havaintoa. Ensimmäisessä sarakkeessa on jo äsken lasketut ennusteet. Toisessa ja kolmannessa sarakkeessa on lasketun luottamus- tai ennustevälin ala- ja ylärajat.

8.4.5 Yksisuuntainen varianssianalyysi

Kun halutaan testata odotusarvojen yhtäsuuruutta usemman ryhmän välillä, voidaan käyttää yksisuuntaista varianssianalyysiä. Testissä oletetaan, että havainnot ovat riippumattomia ja jokaisessa ryhmässä peräisin normaalijakaumasta samalla varianssiparametrilla. Nollahypoteesi on, että ryhmien odotusarvoparametrit ovat samat. Vastahypoteesi puolestaan on, että ainakin yksi odotusarvo poikkeaa muista.

R:ssä varianssianalyysia voidaan soveltaa muun muassa funktioilla `oneway.test`, `aov` tai `lm`.

Esimerkki 8.16. Tutkitaan `iris`-aineiston muuttujaa `Sepal.Width`. Halutaan testata, onko muuttujan odotusarvo sama jokaisella lajilla (`Species`). Tilannetta voi ensin havainnollistaa graafisesti esim. viiksilaatikkokuvan avulla. Sen avulla voidaan myös yrittää tutkia oletusta varianssien yhtäsuuruudesta, eli onko vaihtelu ryhmissä suurin piirtein samansuuruista (yksityiskohtiin ei tässä mennä).

```
> boxplot(iris$Sepal.Width~iris$Species)
```

Yksisuuntaiseen varianssianalyysiin sopii hyvin funktio `oneway.test`. Se ottaa argumentikseen kaavan (formula), jossa selitettävä muuttuja tulee ennen matomerkkiä $\sim$ ja ryh-

mittelevä faktorimuuttuja matomerkin jälkeen. Oletus varianssien yhtäsuuruudesta voidaan määritellä argumentin `var.equal` avulla. Jos tälle argumentille annetaan totuusarvo `FALSE`, funktio käyttää Welchin testisuuretta, joka olettaa erisuuret varianssit ryhmien välillä. Koska muuttujat ovat taulukosta, annetaan taulukon nimi `iris` argumentiksi `data`.

```
> oneway.test(Sepal.Width~Species,data=iris,var.equal=FALSE)

One-way analysis of means (not assuming equal variances)

data:  Sepal.Width and Species
F = 45.012, num df = 2.000, denom df = 97.402, p-value = 1.433e-14
```

Tässä esimerkissä funktion palauttama F-testisuureen arvo on 45.012 ja p-arvo hyvin pieni, joten nollahypoteesi voidaan hylätä pienelläkin merkitsevyystasolla.

Toinen vaihtoehto on käyttää funktiota `aov`. Tällä funktiolla ei voida kuitenkaan luopua oletuksesta varianssien yhtäsuuruudesta. Funktio `aov` ottaa argumentteinaan myös kaavan kuten `oneway.test`. Sen palauttama lista on kuitenkin hieman erilainen ja muun muassa hypoteesintestauksen tuloksiin pääsee käsiksi `summary`-funktion avulla.

```
> iris_aov <- aov(Sepal.Width~Species,data=iris)
> summary(iris_aov)

             Df Sum Sq Mean Sq F value Pr(>F)
Species       2  11.35   5.672   49.16 <2e-16 ***
Residuals   147  16.96   0.115
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Ryhmien välisiä eroja voidaan tutkia tarkemmin funktion `TukeyHSD` avulla. Se ottaa argumenttinaan funktion `aov` palauttaman listan.

```
> TukeyHSD(iris_aov)

  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Sepal.Width ~ Species, data = iris)

$Species
```

```
                       diff         lwr        upr     p adj
versicolor-setosa    -0.658 -0.81885528 -0.4971447 0.0000000
virginica-setosa     -0.454 -0.61485528 -0.2931447 0.0000000
virginica-versicolor  0.204  0.04314472  0.3648553 0.0087802
```

Funktio vertaa ryhmiä pareittain ja antaa ryhmien otoskeskiarvojen erotuksen sekä tämän luottamusvälin ylä- ja alarajan ja p-arvon.

Varianssianalyysin oletusten täyttymistä voi tutkia diagnostiikakuvien avulla.

```
# Neljä erilaista diagnostiikkakuvaa
> plot(iris_aov)
```

Kuten lineaarisessa regressiomallissa, residuaalien tulisi olla likimain normaalijakautuneita eikä residuaalien ja sovitteen välisessä hajontakuvassa saisi näkyä mitään systemaattista riippuvuutta.

8.4.6 Kaksisuuntainen varianssianalyysi

Kaksisuuntaisessa varianssianalyysimallissa tutkitaan kahden eri kategorisen tekijän vaikutusta normaalijakauman odotusarvoon. Kiinnostuksen kohteena on se, poikkeavatko eri kategoriakombinaatioita vastaavat vastemuuttujan odotusarvot toisistaan ja johtuvatko poikkeamat jommastakummasta tekijästä tai näiden yhteisvaikutuksesta. Mikäli selittävät muuttujat oletetaan toisistaan riippumattomiksi, yhteisvaikutusta ei estimoida ja `aov`-funktiolle annetaan kaavaksi (formula) `y` $\sim$ `a+b`, missä `y` on selitettävä muuttuja ja `a` ja `b` ryhmittelevät faktorimuuttujat. Mikäli selittäjiä `a` ja `b` ei oleteta toisistaan riippumattomiksi, annetaan kaavaksi (formula) `y` $\sim$ `a*b`, jolloin estimoidaan myös näiden yhteisvaikutus. Vaihtoehtoisesti voidaan estimoida myös pelkkä yhteisvaikutus antamalla kaavaksi `y` $\sim$ `a:b`

Kurssilla Tilastotiede ja R tutuksi II ei käydä varianssianalyysin osalta läpi selittäjien yhteisvaikutusta eikä myöskään tällaisia tehtäviä tule tällä kurssilla vastaan. Alla on kuitenkin esimerkki tapauksesta, jossa estimoidaan myös yhteisvaikutus. Tämän kurssin tehtävissä selittäjät oletetaan kuitenkin usein toisistaan riippumattomiksi, eli on turvallisempaa käyttää kaavaa `y` $\sim$ `a+b`

Esimerkki 8.17. Tarkastellaan R:stä valmiiksi löytyvää aineistoa `warpbreaks`, jossa tutkitaan kangaspuita ja loimilankoja. Halutaan selittää loimilankojen määrää tietyn mittaisessa loimessa, kun selittäjinä käytetään langan tyyppiä (A ja B) sekä langan pingottuneisuutta (low, medium, high). Tutkitaan tätä kaksisuuntaisella varianssianalyysilla ottaen huomioon myös langan tyypin ja pingottuneisuuden yhteisvaikutus.

```
> fit <- aov(breaks~wool*tension,data=warpbreaks)
> summary(fit)

             Df Sum Sq Mean Sq F value   Pr(>F)
wool          1    451   450.7   3.765 0.058213 .
tension       2   2034  1017.1   8.498 0.000693 ***
wool:tension  2   1003   501.4   4.189 0.021044 *
Residuals    48   5745   119.7
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Huomataan, että langan pingottuneisuus on tilastollisesti erittäin merkitsevä selittäjä, sillä p-arvo on 0.000693. Yhteisvaikutusta vastaava p-arvo alittaa viiden prosentin merkitsevyystason ja yksin lankatyyppiä vastaava p-arvo on 0.058213.

Eroja yksittäisten ryhmien välillä voidaan jälleen tarkastella funktiolla `TukeyHSD()`.

```
> TukeyHSD(fit)

  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = breaks ~ wool * tension, data = warpbreaks)

$wool
         diff       lwr       upr    p adj
B-A -5.777778 -11.76458 0.2090243 0.058213

$tension
          diff       lwr       upr     p adj
M-L -10.000000 -18.81965 -1.180353 0.0228554
H-L -14.722222 -23.54187 -5.902575 0.0005595
H-M  -4.722222 -13.54187  4.097425 0.4049442

$‘wool:tension‘
               diff       lwr        upr     p adj
B:L-A:L -16.3333333 -31.63966  -1.027012 0.0302143
A:M-A:L -20.5555556 -35.86188  -5.249234 0.0029580
B:M-A:L -15.7777778 -31.08410  -0.471456 0.0398172
A:H-A:L -20.0000000 -35.30632  -4.693678 0.0040955
B:H-A:L -25.7777778 -41.08410 -10.471456 0.0001136
```

```
A:M-B:L  -4.2222222 -19.52854  11.084100 0.9626541
B:M-B:L   0.5555556 -14.75077  15.861877 0.9999978
A:H-B:L  -3.6666667 -18.97299  11.639655 0.9797123
B:H-B:L  -9.4444444 -24.75077   5.861877 0.4560950
B:M-A:M   4.7777778 -10.52854  20.084100 0.9377205
A:H-A:M   0.5555556 -14.75077  15.861877 0.9999978
B:H-A:M  -5.2222222 -20.52854  10.084100 0.9114780
A:H-B:M  -4.2222222 -19.52854  11.084100 0.9626541
B:H-B:M -10.0000000 -25.30632   5.306322 0.3918767
B:H-A:H  -5.7777778 -21.08410   9.528544 0.8705572
```

Tulosteessa voidaan tarkastella eroja lankatyyppien välillä (komponentti `$wool`), pingottuneisuusryhmien välillä (komponentti `$tension`) tai näiden yhteiskombinaatioiden välillä (komponentti `$wool:tension`) Yhteiskombinaatio on esimerkiksi lankatyyppi B, jolla alhainen pingottuneisuus vs. lankatyyppi A, jolla alhainen pingottuneisuus (`B:L-A:L`).

8.5 Yhteensopivuus- ja riippumattomuustestit

Tutustutaan seuraavaksi yhteensopivuus- ja riippumattomuustesteihin, joissa testisuure on asymptoottisesti χ^2-jakautunut. Näihin testeihin tutustutaan tarkemmin kurssilla Tilastotiede ja R tutuksi II. R:ssä näille testeille löytyy funktio `chisq.test`.

Esimerkki 8.18. Tutkitaan Sheldon Rossin kirjan [4] esimerkkiä 13.1. Tiedetään, että 41 prosentilla amerikkalaisista on veriryhmä A, 9 prosentilla on veriryhmä B, 4 prosentilla on veriryhmä AB ja 46 prosentilla on veriryhmä O. 200 vatsasyöpäpotilaan otoksessa 92 henkilöllä oli veriryhmä A, 20 henkilöllä veriryhmä B, 4 henkilöllä veriryhmä AB ja 84 henkilöllä veriryhmä O. Testataan viiden prosentin merkitsevyystasolla ovatko vatsasyöpäpotilaiden veriryhmien osuudet samat kuin koko väestöllä.

Kyseessä on yhteensopivuustesti, jossa funktiolle `chisq.test()` annetaan argumenteiksi havaitut frekvenssit sekä odotetut osuudet (`p`).

```
> otos <- c(92,20,4,84)
> osuudet <- c(0.41,0.09,0.04,0.46)
> chisq.test(otos,p=osuudet)

Chi-squared test for given probabilities
```

[4] Sheldon Ross, Introductory Statistics 4th edition, 2017

```
data:  otos
X-squared = 4.1374, df = 3, p-value = 0.247
```

Funktio palauttaa χ^2-testisuureen arvon 4.1374, χ^2-jakauman vapausasteet $(4-1=3)$ sekä testisuureesta lasketun p-arvon. Tässä esimerkissä nollahypoteesi jää voimaan viiden prosentin merkitsevyystasolla.

Esimerkki 8.19. Tutustutaan nyt fiktiivisen leipomon toimintaan ja sen tuotekehityksessä ilmenneeseen ongelmaan. Leipomossa on nimittäin huomattu, että taikinan kohoaminen on ajoittain huteraa ja leipuri epäilee, ettei käytetty hiiva (Hiiva 1) ole parasta A-laatua. Tästä syystä leipuri päättää koittaa kilpailijan vastaavaa tuotetta (Hiiva 2) ja kirjaa testituloksensa ylös neljästä sadasta taikinaerästä.

	Hiiva 1	Hiiva 2
Kohoaminen OK	120	173
Ei kohonnut	80	27

Tutkitaan seuraavaksi riippumattomuustestillä, olisiko leipurilla tilastollisia perusteita vaihtaa hiivan toimittajaa. Valitaan nollahypoteesi H_0 : "Hiivan valinta ei vaikuta taikinan kohoamiseen"ja lasketaan testisuure funktion `chisq.test()`-funktion avulla:

```
> O <- matrix(c(120,80,173,27), ncol=2)
> chisq.test(O, corr=F)

Pearson's Chi-squared test

data:  O
X-squared = 35.8394, df = 1, p-value = 2.143e-09
```

Riippumattomuustestissä funktiolle `chisq.test` annetaan argumentiksi matriisi, jossa näkyvät frekvenssit. Argumentti `corr=F` ilmoittaa, ettei haluta käyttää jatkuvuuskorjausta. Testin tuloksena saadaan hyvin pieni p-arvo, jonka voidaan tulkita viittaavan siihen, että taikinan kohoaminen todella riippuisi hiivan valinnasta.

Sama voitaisiin hyvinkin laskea myös käyttämättä funktiota `chisq.test`. Muistetaan, että testisuureen kaava on

$$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}},$$

missä O_{ij} ovat havaitut frekvenssit ja E_{ij} niitä vastaavat odotetut arvot. E_{ij} voidaan laskea havaituista frekvensseistä tulona rivin i ja sarakkeen j summista ja jakamalla sitten koko havaintomäärällä, eli

$$E_{ij} = n_{+j}n_{i+}/n.$$

Tästä voidaan laskea testisuure ja sen p-arvo käyttäen jakaumafunktiota `pchisq()`, jolle annetaan argumentteina havaittu testisuureen arvo sekä testisuureen asymptoottisen jakauman vapausasteet ja tieto siitä, että halutaan käyttää yläkvantiileja:

```
> # Lasketaan odotetut frekvenssit
> E <- (apply(O, 1, sum) %*% t(apply(O,2,sum))) / sum(O)
> # Lasketaan testisuure
> X <- sum((O - E)^2 / E)
> X

[1] 35.83937

> pchisq(X, df=1, lower=F)

[1] 2.142742e-09
```

8.6 Binomikokeen testit ja luottamusvälit

Eksaktia yhden otoksen binomitestiä varten on R:ssä funktio `binom.test`. Tällä funktiolla tehtävä testi ei siis perustu normaaliapproksimaatioon, kuten monet testit binomikokeessa. Funktio saa ensimmäisenä argumenttinaan `x` joko onnistumisten määrän kokonaislukuna tai vektorin, jonka ensimmäinen alkio on onnistumisten määrä ja toinen epäonnistumisten määrä. Mikäli annetaan vain onnistumisten määrä, tulee toisena argumenttina `n` antaa kaikkien yritysten lukumäärä. Lisäksi funktiolle tulee antaa argumentti `p`, joka kertoo nollahypoteesin mukaisen onnistumistodennäköisyyden. Yksisuuntainen vastahypoteesi voidaan tarvittaessa määritellä argumentilla `alternative`, joka toimii samalla tavalla kuin `t.test`-funktiossa. Katso esimerkki 8.5.

Esimerkki 8.20. Havaitaan binomikokeessa 682 onnistumista ja 243 epäonnistumista. Testataan nollahypoteesia $p = \frac{7}{10}$ kaksisuuntaista vastahypoteesia $p \neq \frac{7}{10}$ vastaan.

```
> binom.test(c(682, 243), p = 7/10)

Exact binomial test

data:  c(682, 243)
```

```
number of successes = 682, number of trials = 925, p-value = 0.01327
alternative hypothesis: true probability of success is not equal to 0.7
95 percent confidence interval:
 0.7076683 0.7654066
sample estimates:
probability of success
             0.7372973
```

Esimerkki 8.21. Havaitaan binomikokeessa 682 onnistumista ja 243 epäonnistumista. Testataan nollahypoteesia $p = \frac{7}{10}$ yksisuuntaista vastahypoteesia $p > \frac{7}{10}$ vastaan.

```
> binom.test(x=682,n=682+243, p = 7/10,alternative="greater")

Exact binomial test

data:  682 and 682 + 243
number of successes = 682, number of trials = 925, p-value = 0.006848
alternative hypothesis: true probability of success is greater than 0.7
95 percent confidence interval:
 0.7124129 1.0000000
sample estimates:
probability of success
             0.7372973
```

Funktion `binom.test` palauttamassa tulosteessa näkyvät siis onnistumisten määrä ja yritysten määrän lisäksi testin p-arvo, jonka laskeminen ei perustu normaaliapproksimaatioon. Lisäksi funktio palauttaa estimaatin onnistumistodennäköisyydelle ja tälle lasketun 95 prosentin luottamusvälin. Luottamustason voi myös määritellä joksikin muuksi argumentilla `conf.int` aivan kuten `t.test`-funktiossakin.

8.7 Valmiita funktioita testeille ja tilastollisille malleille

Yhden ja kahden otoksen t-testi ja t-luottamusväli: `t.test()`
Binomitesti ja binomikokeen luottamusväli (eksakti): `binom.test()`
Suhteellisten osuuksien vertailutesti: `prop.test()`
Yhteensopivuus- ja riippumattomuustestit: `chisq.test()`
Poisson-testi: `poisson.test()`

Shapiro-Wilk-testi normaalisuudelle: `shapiro.test()`
F-testi varianssien vertailuun: `var.test()`

Yksisuuntainen varianssianalyysi: `oneway.test()`

Ei-parametrisia testejä
Friedmanin testi: `friedman.test()`
Kruskal-Wallisin testi: `kruskal.test()`
Wilcoxonin merkittyjen sijalukujen testi ja järjestyssummatesti:
`wilcox.test()`, parittainen: `pairwise.wilcox.test()`

Tilastolliset mallit
Kaava (formula), kun vastemuuttuja y ja riippumattomat selittäjät:
`y ~ x1+x2+x3+...`

Kaava, kun vastemuuttuja y ja selittäjät, joilla myös interaktiota (yhteisvaikutusta):
`y ~ x1+x2+x1:x2` tai suoraan `y ~ x1*x2`

Lineaarisen mallin sovittaminen: `malli <- lm(kaava)`
Yleistetyn lineaarisen mallin sovittaminen: `glm(kaava)`
Varianssianalyysi: `malli <- aov(kaava)`
Yksittäisten ryhmien väliset erot varianssianalyysissa: `TukeyHSD(malli)`
Mallilla ennustaminen: `predict(malli)`
Mallin parametrien estimaatit:
`coefficients(malli)` tai `malli$coefficients` Mallin parametrien estimaattien luottamusvälit: `confint(malli)`
Mallin residuaalit: `residuals(malli)` tai `malli$residuals`

Kappale 9

Paketit ja kirjastot

Monisteessa on pääasiassa käyty läpi R:n perusteita valmiita funktioita käyttäen. Edistyneempää käyttöä varten näiden ohella on hyvä opetella hyödyntämään myös netissä jaettavia R:n paketteja.

Paketit ovat funktioita, niiden dokumentaatiota ja usein esimerkkiaineistoja sisältäviä kokoelmia, jotka keskittyvät tyypillisesti tietyn aihealueen ongelmiin. Nykyisin saatavilla oleva pakettivalikoima on erittäin laaja, ja erilaisia työkaluja löytyykin mitä erikoistuneempiin aiheisiin. Käydään lyhyesti läpi esimerkin avulla, kuinka pakettien käyttäminen onnistuu.

9.1 Pakettien asentaminen

Tarkastellaan vaikkapa MASS-pakettia, joka sisältää R:n harjoittelun kannalta monia havainnollistavia aineistoja ja hyödyllisiä funktioita.

Pakettien asentamista varten on olemassa funktio `install.packages()`. Se ottaa argumentikseen ladattavan paketin nimen, jonka avulla se lähtee oletuksena etsimään pakettia CRAN:sta (Comprehensive R Archive Network). Mikäli paketti on ladattavissa, ja latauspalvelimelle ei ole asetettu oletusarvoa, pyytää funktio käyttäjää valitsemaan, mistä osoitteesta paketit halutaan ladata. Suurta käytännön merkitystä tällä valinnalla ei ole, joten tarjotuista vaihtoehdoista voit valita mieleisesi. Tämän jälkeen funktio lataa ja asentaa paketin sekä tarvittaessa sen vaatimat muut paketit koneelle.

Esimerkki 9.1.

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

Installing package into ‘D:/Toni/Documents/R/win-library/3.2’
(as ‘lib’ is unspecified)
```

```
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cloud.r-project.org/bin/windows/contrib/3.2/MASS_7.3-45.zip'
Content type 'application/zip' length 1085983 bytes (1.0 MB)
downloaded 1.0 MB

package 'MASS' successfully unpacked and MD5 sums checked
```

Paketteja voidaan asentaa `install.packages()`:n avulla myös muualtakin kuin CRAN:sta antamalla sille argumentiksi pelkän nimen lisäksi myös URL-osoitteen tai tiedostosijainnin, josta paketti löytyy. Asennuksen funktio tekee kaikissa tapauksissa oletuksena R:ään liittyvän ympäristömuuttujan R_LIBS_USER määrittämään kansioon. Sen sijaintia voi tarkastella ja muuttaa R-istunnon ajaksi funktiolla `.libPaths()`.

Esimerkki 9.2.

```
> .libPaths()

[1] "D:/Toni/Documents/R/win-library/3.2"

> .libPaths("D:/kirjasto")
> .libPaths()

[1] "D:/kirjasto"
```

Tästä voi olla hyötyä tilanteissa, joissa esimerkiksi käyttäjäkohtaiset rajoitukset estävät pakettien tallentamisen oletuskansioon. Vaihtoehtoisesti R:n saa vaihtamaan oletuskansion automaattisesti tekemällä R:n työkansioon .Renviron-tiedoston, jossa on argumentti `R_LIBS_USER=sijainti`, missä `sijainti` paikalle tulee halutun kansion sijainti. Jos pakettien asentamisessa ei kuitenkaan ole mitään ongelmia, ei näistä asetuksista tarvitse välittää.

9.2 Pakettien käyttäminen

Kun paketti on saatu asennettua, voidaan sen sisältö ottaa käyttöön R:ssä funktiolla `library()`.

Esimerkki 9.3.

```
# Kokeillaan tulostaa kaksi ensimmäistä riviä
# MASS-paketin geyser-aineistosta.
```

```
# Ilman pakettia oliota geyser ei ole määritelty
> geyser[1:2,]

Error: object 'geyser' not found

# Jos paketin koko sisältöä ei haluta avata R:ään, voidaan
# sen yhtä funktiota/aineistoa kutsua merkinnällä paketti::funktio
> MASS::geyser[1:2,]

  waiting duration
1      80 4.016667
2      71 2.150000

> library(MASS)
> geyser[1:2,]

  waiting duration
1      80 4.016667
2      71 2.150000
```

Funktion `install.packages()` tavoin `library()` etsii paketteja niiden oletuskansiosta. Jos paketti ei siis ole tallennettuna oletuskansiossa, on sen sijainti ilmoitettava `library()`:n argumenttina tai oletuskansio on vaihdettava.

Kun paketti on ladattu R:n työtilaan, voidaan sen funktioita käyttää kuten mitä tahansa R:n valmiita funktioita. Tyypillisesti paketteihin sisältyvät kuvaukset niiden sisällöstä, joita voi lukea help-komennolla, jos paketti on jo ladattu R:n työtilaan `library()`-funktiolla (**esimerkin 9.3** tapauksessa `?geyser`). Jos paketti on asennettu, muttei ladattu, helpin löytää käyttämällä paketin nimeä etuliitteenä: `?MASS::geyser`, tai etsiä kaikista asennetuista paketeista komennolla `??`, esimerkiksi `??geyser`.

9.3 Hyödyllisiä paketteja

`MASS`: Tilastollisia funktioita ja aineistoja
`dplyr`: Funktioita datan käsittelyyn
`Hmisc`: Funktioita mm. data-analyysiin, imputointiin ja grafiikkaan
`ggplot2`: Erittäin monipuolinen paketti kuvien piirtämiseen