

Data-analyysi R-ohjelmistolla

Henri Karttunen ja Ville Hyvönen
Matematiikan ja tilastotieteen laitos
Helsingin yliopisto

Kevät 2015

Sisältö

Saatteeksi	3
1 Vektorit, matriisit ja peruslaskutoimitukset	4
1.1 Skalaarimuuttujat	4
1.1.1 Määrittely	4
1.1.2 Laskuoperaatiot	5
1.1.3 Loogiset operaattorit ja totuusarvot	5
1.2 Vektorit ja matriisit	6
1.2.1 Vektorien luominen	6
1.2.2 Vektorien käsittely	7
1.2.3 Matriisien luominen	9
1.2.4 Matriisien käsittely	9
1.2.5 Vektorien ja matriisien laskutoimitukset	11
2 Taulukot, tilastolliset funktiot ja grafiikka	14
2.1 R:n tietotyypit	14
2.1.1 Muuttujan tyyppi	14
2.1.2 Merkkijonot	15
2.1.3 Faktorit	16
2.2 Taulukko	17
2.2.1 Taulukon luominen	17
2.2.2 Aineiston lataaminen tiedostosta	18
2.2.3 Alkioihin, sarakkeisiin ja riveihin viittäminen	19
2.2.4 Osa-aineistojen valinta	20
2.3 Tilastolliset funktiot	21
2.3.1 Tunnusluvut	21
2.3.2 Simulointi ja jakaumafunktiot	23
2.4 Aineiston visuaalinen tarkastelu	24

3	Funktiot, faktorit ja ristiintaulukointi	26
3.1	Omat funktiot	26
3.2	Koodin osien toistaminen	28
3.3	Faktorit	30
3.3.1	Faktorien luominen	31
3.4	Osa-aineistojen valinta, osa 2	32
3.4.1	Alkioiden valinta vektorista	32
3.4.2	Osa-aineistojen valinta taulukosta	33
3.4.3	Tunnuslukujen laskeminen osa-aineistoittain	34
3.5	Ristiintaulukointi	35
3.5.1	Frekvenssitaulu	35
3.5.2	Ristiintaulukko	36
4	Aineiston luokittelu ja luottamusvälit	38
4.1	Osa-aineistojen valinta, osa 3	38
4.1.1	Sarakkeiden valinta nimellä	38
4.2	Aineiston luokittelu	40
4.3	T-luottamusväli	41
4.4	Kuvien piirtämisestä	42
5	Listat ja tilastollinen testaaminen	44
5.1	Lista	44
5.1.1	Listan luominen	44
5.1.2	Listan komponenttien nimeäminen	46
5.1.3	Listan läpikäyminen	48
5.2	Lisää funktioista	50
5.2.1	Funktion palautusarvon kirjoittaminen näkyviin	50
5.2.2	Koodiblokit ja koodin sisennys	51
5.2.3	replicate	52
5.3	Yhden otoksen T-testi	53
5.4	Riippumattomuustesti	55
6	Lineaarinen regressio ja Bayes-päätely	57
6.1	Yhden selittäjän lineaarinen regressio	57
6.2	Useamman selittäjän lineaarinen regressio	62
6.3	If else - rakenne	63
6.4	Bayes-päätelyä	66
6.4.1	Doping-testi-esimerkki	66
6.4.2	Doping-testi-esimerkki vektorisoituna	67

Saatteeksi

Tähän on kerätty keväällä 2015 pitämämme Data-analyysi R-ohjelmistolla-kurssin materiaali. Materiaali on jaettu kuuteen viikkoon: jokaisen osan materiaali sisältää lyhyet esittelyt viikon aiheista ja niihin liittyviä esimerkkejä. Kurssi on tarkoitettu käytäväksi samaan aikaan kurssin Johdatus tilastolliseen päättelyyn kanssa, joten matemaattisen tilastotieteen peruskäsitteet oletetaan tunnetuiksi.

Tämän monisteen tarkoitus ei ole olla kattava esitys R:n käytön perusteista, vaan tarkoitus on tarjota materiaalia itseopiskeluun kurssin tehtävien tueksi. Kurssilla ei ole myöskään erillistä kurssikirjaa, mutta R:n perusteista löytyy paljon tasokkaita kirjoja ja nettimateriaaleja. Hyviä ovat esimerkiksi:

- Petri Koistisen R-kurssi kevät 2007: <http://www.rni.helsinki.fi/pek/r-johdatus/kmat.html>
- Peter Dalgaard: *Introductory Statistics with R* (Springer, 2002 ja 2008).

Vaikka perusteet ehditään käydä läpi kuudessa viikossa, paljon R:n ominaisuuksista jää myös käymättä. Monilla tilastotieteen, tietojenkäsittelytieteen sekä sovellusalojen kursseilla R:ää käytetään oikeiden aineistojen käsittelyyn, joten näitä kursseja voi ajatella eräänlaisina 'R:n jatkokursseina'. Erityisesti kurssilla *Computational statistics* perehdytään R:n käyttöön simuloinnissa ja C++:n ja R:n yhdistämiseen.

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

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

Huomautuksia virheistä voi lähettää sähköpostitse kirjoittajille: henri.c.karttunen@helsinki.fi tai ville.o.hyvonen@helsinki.fi.

Työn iloa!
t. Ville & Henkka

Viikko 1

Vektorit, matriisit ja peruslaskutoimitukset

Ensimmäisen viikon tavoitteena on saada riittävät taidot käsitellä toisella viikolla käyttöön tulevia aineistoja. Aineistot luetaan R:ään matriiseina tai matriisin tapaisina tietorakenteina, joita kutsutaan taulukoiksi (data frame). Näiden käsittelyä varten peruslaskutoimitukset on oltava hyvin hallussa.

1.1 Skalaarimuuttujat

1.1.1 Määrittely

Skalaareja voidaan käsitellä R-ohjelmistolla sellaisenaan, syöttämällä haluttu luku konsoliin. Desimaalierottimenä toimii tavallinen piste [.]

Esimerkki 1.1. Tulostetaan ensin lukuja

```
> 15
[1] 15
> 3.14159
[1] 3.14159
```

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 sisälön voi tulostaa antamalla muuttujan nimen komentona.

Esimerkki 1.2. Sijoitetaan muuttujiin arvoja

```
> a <- 42
> a
[1] 42
> b <- 0.001
> b
[1] 0.001
```

1.1.2 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 1.3. Yhteen- ja vähennyslasku

```
> a <- 5
> b <- 3
> a-b
[1] 2
> a+b
[1] 8
> a - 6
[1] -1
```

Esimerkki 1.4. Kerto- ja jakolasku

```
> a*b
[1] 15
> a/b
[1] 1.666667
> a %% b
[1] 2
```

1.1.3 Loogiset operaattorit ja totuusarvot

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 (nämä voi myös lyhentää kirjaimilla T ja F). Vertailun tuloksen voi tallentaa muuttujaan.

Esimerkki 1.5. 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 1.6.

```
> (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
```

1.2 Vektorit ja matriisit

1.2.1 Vektorien luominen

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

kuten skalaaritkin. Itse asiassa R:ssä skalaaritkin ovat oikeasti yksipaikkaisia vektoreita.

Monesti yksinkertaisia aineistoja on helppo käsitellä vektorimuodossa, jolloin vektorin arvot olisivat esimerkiksi mittaustuloksia jostakin kokeesta.

Esimerkki 1.7. 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] 0 1 2 3 4 5 6
```

Esimerkki 1.8. Edellisen esimerkin vektori saadaan myös seuraavilla tavoilla:

```
> 0:6
[1] 0 1 2 3 4 5 6
> seq(0,6,by=1)
[1] 0 1 2 3 4 5 6
```

Esimerkki 1.9. 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
```

1.2.2 Vektorien käsittely

Vektorin alkioihin voidaan viitata antamalla halutut indeksit hakasuluissa. Huom. R:ssä indeksointi alkaa 1:stä eikä 0:sta. Jos viitataan indeksiin, jota ei ole vektorissa, tuloksena on puuttuva arvo, eli NA.

Esimerkki 1.10. 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
```

Voidaan valita myös useita eri alkoita kerralla antamalla indeksiksi vektori.

Esimerkki 1.11. (Jatkoa edelliseen) valitaan vektorin `a` kolme viimeistä alkioita 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 alkio, 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 alkioita lukuun kolme ja palauttaa vertailun tuloksen totuusarvovektorina:

Esimerkki 1.12.

```
> a > 3
[1] FALSE TRUE TRUE FALSE FALSE
```

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

Esimerkki 1.13.

```
> a[a > 3]
[1] 6 30
```

Juuri kuten halusimmekin. Voimme siis kätevästi valita alkoita vektorista ehtolauseiden avulla.

1.2.3 Matriisien luominen

R:ssä matriisi luodaan vektorista `matrix()` -komennolla. Funktio tarvitsee myös tiedon siitä, onko annettava data järjestetty riveittäin vai sarakkeittain (`byrow`).

Esimerkki 1.14. 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), ncol=3, nrow=3, byrow=TRUE)
     [,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 tehdään. Kokeile! Vastaavasti argumentin arvoksi voidaan antaa `FALSE` tai `F`. Riveittäin ja sarakkeittain järjestämisen eroa voit kokeilla antamalla dataksi `c(1,4,7,2,5,8,3,6,9)`.

1.2.4 Matriisien käsittely

Matriiseja voidaan käsitellä kuten vektoreitakin: hakasulkuja käyttämällä voidaan valita alkioita ja tehdä vertailuja. Matriiseista voidaan yksittäisten solujen lisäksi valita myös kokonaisia rivejä tai sarakkeita.

Esimerkki 1.15. Solujen, rivien ja sarakkeiden valitseminen:

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

Esimerkki 1.16. Arvojen sijoittaminen matriisiin:

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

Esimerkki 1.17. Rivien ja sarakkaiden lisääminen

```
> a <- matrix(c(1,2,3,4,5,6,7,8,9), ncol=3, nrow=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 käy helpoiten käyttämällä totuusarvoja vektorioperaatioiden tapaan.

Esimerkki 1.18. Rivien ja sarakkaiden poistaminen

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

```

1.2.5 Vektorien ja matriisien laskutoimitukset

Matriiseille saadaan laskettua matriisien tulo operaattorin `%*%` avulla. Pelkkä kertomerkki tuottaa tässä hieman erilaisen tuloksen, kokeile! Matriisien yhteen- ja vähennyslaskut toimivat tavallisesti käyttäen operaattoreita `+` ja `-`. Transpoosi saadaan komennolla `t()`, jolle argumentiksi annetaan transponoitava matriisi.

Esimerkki 1.19. 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), ncol=3, nrow=3, byrow=T)
> B <- matrix(c(9,8,7,6,5,4,3,2,1), ncol=3, nrow=3, byrow=T)
> A%*%B
      [,1] [,2] [,3]
[1,]   30   24   18
[2,]   84   69   54
[3,]  138  114   90

```

Esimerkki 1.20. Transponoidaan edellisen esimerkin matriisi A:

```

> t(A)
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

```

Esimerkki 1.21. Vektorien ja matriisien tulo onnistuu täysin samalla tavalla:

```

> a<-c(0,1,2,3)
> a %*% diag(4)
      [,1] [,2] [,3] [,4]
[1,]    0    1    2    3

```

Matriisin kääntäminen onnistuu funktiolla `solve()`.

Esimerkki 1.22. 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

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

missä

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

ja

$$y = [15 \ 14 \ 12 \ 11 \ 10 \ 8 \ 6]'$$

```
> 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

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

Viikko 2

Taulukot, tilastolliset funktiot ja grafiikka

Ensimmäisen viikon tehtävissä harjoiteltiin matriisien ja vektorien käsittelyä, sekä niiden alkoiden valintaa. Nyt näitä taitoja sovelletaan oikeiden aineistojen käsittelyyn.

Tämän viikon tärkeimpiä teemoja ovat datan alustava tarkastelu kuvien ja tunnuslukujen avulla, simulaatioiden käyttö, tallennetun aineiston käyttöönotto ja tietotyypit.

2.1 R:n tietotyypit

2.1.1 Muuttujan tyyppi

R:n tietotyypeistä viime viikolla esiteltiin numeeriset muuttujat ja totuusarvot. Muuttujan tyyppiä voi tarkastella `class()`-funktioilla ¹.

Esimerkki 2.1.

```
> a <- 5
> b <- TRUE
> c <- c(3,76,43,5)
> d <- c("Mies", "Nainen")
> e <- as.factor(d)
>
> class(a)
[1] "numeric"
> class(b)
```

¹Itse asiassa `class()` kertoo R:n objektin luokan, mutta yksinkertaisilla objekteilla, kuten vektoreilla luokka on sama kuin niiden tietotyyppi, ellei muuta ole asetettu.

```
[1] "logical"
> class(c)
[1] "numeric"
> class(d)
[1] "character"
> class(e)
[1] "factor"
>
```

Muuttujien tyyppiä voi testata `is-` ja muuttaa, `as-`alkuisilla funktioilla. Päätteeksi funktioon laitetaan pisteen jälkeen halutun tietotyypin nimi.

Esimerkki 2.2.

```
> a <- c("4", "47", "-7")
> a
[1] "4" "47" "-7"
>
> is.numeric(a)
[1] FALSE
> is.character(a)
[1] TRUE
>
> a <- as.numeric(a)
> a
[1] 4 47 -7
>
> is.numeric(a)
[1] TRUE
> is.character(a)
[1] FALSE
```

2.1.2 Merkkijonot

Numeeristen muuttujien ja totuusarvojen lisäksi R:ssä on käytössä oma tietotyyppi merkkijonoille. 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. Merkkijonoja voi myös sijoittaa peräkkäin vektoriin.

Esimerkki 2.3.

```
> "R"
[1] "R"
> merkkijono <- 'data-analyysi'
> merkkijono
[1] "data-analyysi"
> ohjaajat <- c('Ville', 'Henkka', 'Oskari')
> ohjaajat
[1] "Ville" "Henkka" "Oskari"
> class(ohjaajat)
[1] "character"
>
```

2.1.3 Faktorit

Faktori on R:n tietotyyppi, joka on tarkoitettu luokitteluasteikollisten muuttujien tallentamiseen ja käsittelyyn. Jos meillä on esimerkiksi muuttuja johon on tallennettu vastaajan sukupuoli, voimme muuttaa sen faktoriksi funktiolla `as.factor()`.

Esimerkki 2.4.

```
> 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
```

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. Tasoja on yhtä monta kuin alkuperäisessä muuttuja saa erilaisia arvoja. Faktoreista on hyötyä tehtäessä analyysijä osa-aineistoittain, esimerkiksi juuri sukupuolen mukaan jaoteltuna.

2.2 Taulukko

Vektoreihin ja matriiseihin voi tallentaa vain yhden tietotyypin alkoita kerrallaan. Yritettäessä tallentaa useaa eri tietotyyppiä olevia alkioita samaan vektoriin tai matriisiin, R muuttaa koko vektorin tai matriisin ”yleisempää” muotoa olevaan tietotyyppiin. Esimerkiksi liitettäessä merkkijonoja ja numeroita samaan vektoriin R muuttaa numerot merkkijonoiksi.

Esimerkki 2.5.

```
> a <- c(ohjaajat, 5)
> a
[1] "Ville" "Henkka" "Oskari" "5"
> class(a)
[1] "character"
```

2.2.1 Taulukon luominen

Tilastollisia aineistoja käsitellessä haluamme kuitenkin monesti sekä numeerisia että merkkijonomuotoisia muuttujia samaan tietorakenteeseen. Yleensä käsiteltävät aineistot ovat taulukoita, joissa rivit edustavat havaintoyksiköitä ja, seuraavassa esimerkissämme kurssin ohjaajat, ja sarakkeet tarkasteltavia muuttujia, esimerkissämme ohjaajien pyöräilemät tunnit vuoden aikana, ja ohjaajien kengännumero.

Data frame eli taulukko on R:n tietotyyppi, joka on tarkoitettu juuri tällaisten tilastollisten aineistojen tallentamiseen. Esimerkkiaineistostamme luodaan taulukko funktiolla `data.frame`. Argumentti `stringAsFactors = FALSE` määrittää, että funktio pitää aineiston merkkijonot merkkijonoina, eikä muuta niitä tyyppiltään faktoreiksi.

Esimerkki 2.6.

```
> nimi <- c('Ville', 'Henkka', 'Oskari')
> pyorailytunnit <- c(3, 130, 200)
> kengannumero <- c(42, 47, 42)
>
> ohjaajat <- data.frame(nimi, pyorailytunnit, kengannumero,
stringsAsFactors=FALSE)
>
> ohjaajat
  nimi pyorailytunnit kengannumero
1 Ville              3             42
2 Henkka            130             47
3 Oskari            200             42
```

```
>
> class(ohjaaajat)
[1] "data.frame"
>
```

Funktiolla `str()` voidaan tutkia tarkemmin taulukon sarakkeiden tietotyyppejä. Funktio myös tulostaa rivien ensimmäiset arvot.

Esimerkki 2.7.

```
> str(ohjaaajat)
'data.frame': 3 obs. of 3 variables:
 $ nimi          : chr  "Ville" "Henkka" "Oskari"
 $ pyorailytunnit: num  3 130 200
 $ kengannumero  : num  42 47 42
>
```

2.2.2 Aineiston lataaminen tiedostosta

Luettaessa tiedostoa R:n työhakemisto tulee ensin asettaa siihen hakemistoon, jossa tiedosto sijaitsee. Pieniä tiedostoja kannattaa 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 2.8.

```
> 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_2015")
> getwd()
[1] "C:/Users/Ville/Desktop/DA_R/Vuosi_2015"
```

Aineistot, kuten esimerkkiaineistonamme käyttämä pyöräilybarometri 2014 - aineisto,

tulevat usein Excel-taulukkona. Tällöin kannattaa avata tiedosto Excelissä tai vastaavassa taulukkolaskentaohjelmassa ja tallentaa tiedostosta 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). Harjoituksissa käytettävä aineisto on jo valmiiksi tallennettu CSV-muotoon.

CSV-tiedostoja luetaan taulukkoon funktiolla `read.csv`. Argumenteiksi annetaan tässä tapauksessa tiedoston nimi (tiedoston on sijaittava työhakemistossa), tiedostossa käytettävä sarakkeiden erotin (puolipiste) ja `stringsAsFactors`, joka määrittää muutetaanko aineiston merkkijono-sarakkeet faktoreiksi (oletusarvo on `TRUE`). Tarkastetaan että aineiston lataus onnistui tulostamalla viisi 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 tulkinat löytyvät aineiston mukana tulevasta koodikirjasta.

Esimerkki 2.9.

```
> pb <- read.csv(file='PB_2014.csv', sep=';', stringsAsFactors=FALSE)
>
> pb[1:5,1:10]
```

	RespondentID	Aidnr.1	ADate.1	Aalue	AA	AB.1	AC	pnro	Aq1	Aq2
1	1	1005	20140822	2	2	38	78	920	1	1
2	2	1010	20140822	2	2	24	59	730	1	5
3	3	1016	20140822	2	2	63	12	210	1	5
4	4	1032	20140822	1	2	61	8	170	1	5
5	5	1034	20140822	2	2	60	24	330	2	4

2.2.3 Alkioihin, sarakkeisiin ja riveihin viittäminen

Taulukkoa voi ajatella matriisina ², jonka sarakkeet on nimetty, ja jonka sarakkeet voivat olla keskenään eri tietotyyppisiä. Taulukon alkioihin, riveihin ja sarakkeisiin voi viitata indekseillä samalla tavalla kuin matriisien alkioihin, riveihin ja sarakkeisiin.

Esimerkki 2.10.

```
> ohjaajat[3,1]
[1] "Oskari"
>
> ohjaajat[2,]
```

²Oikeasti taulukko on pohjimmiltaan tyyppiltään lista eikä matriisi, mutta tästä lisää listojen yhteydessä.

```

      nimi pyorailytunnit kengannumero
2 Henkka          130          47
>
> ohjaajat[,3]
[1] 42 47 42

```

Yleensä taulukon sarakkeisiin kannattaa kuitenkin viitata niiden nimellä. Se toisaalta tekee koodista helpommin luettavaa, ja jos taulukkoon tulee uusia sarakkeita tai sarakkeiden järjestys vaihtuu, nimillä viitattaessa koodia ei tarvitse muuttaa. Taulukon sarakkeita voi valita $\$$ -operaattorilla. Taulukon sarakkeet ovat vektoreita, ja niitä voidaan käsitellä kaikilla ensimmäisellä viikolla opituilla vektorioperaatioilla.

Esimerkki 2.11.

```

> ohjaajat$nimi
[1] "Ville" "Henkka" "Oskari"
>
> ohjaajat$kengannumero
[1] 42 47 42
>
> class(ohjaajat$nimi)
[1] "character"
>
> class(ohjaajat$kengannumero)
[1] "numeric"
>
> 5 * ohjaajat$kengannumero
[1] 210 235 210
>
> ohjaajat$kengannumero[ohjaajat$kengannumero < 45]
[1] 42 42

```

2.2.4 Osa-aineistojen valinta

Aivan kuten matriiseistakin, taulukoista voidaan valita rivejä ehtolauseiden avulla. Jos halutaan esimerkiksi tarkastella kaikkia ohjaajia, joiden nimi on Oskari, tai kaikkia ohjaajia, joiden kengännumero on pienempää kuin 45, sijoitetaan vain haluttu ehto rivin indeksin paikalle (huomaa ehdon jälkeinen pilkku, joka erottaa rivin ja sarakkeen indeksin). Osa-aineistot ovat myös taulukoita, ja ne voidaan tallentaa myöhempää käyttöä varten.

Esimerkki 2.12.

```

> ohjaajat[ohjaajat$nimi == 'Oskari', ]
      nimi pyorailytunnit kengannumero
3 Oskari           200           42
>
> ohjaajat2 <- ohjaajat[ohjaajat$kengannumero < 45, ]
>
> ohjaajat2
      nimi pyorailytunnit kengannumero
1  Ville           3           42
3 Oskari           200           42
>
> class(ohjaajat2)
[1] "data.frame"

```

Valmiiseen taulukkoon voidaan lisätä sarakkeita kirjoittamalla haluttu sarakkeen nimi \$-operaattorilla, ja sijoittamalla arvot siihen vektorina. Sarakkeita voi poistaa sijoittamalla tyhjäärvon NULL poistettavaan sarakkeeseen.

Esimerkki 2.13.

```

> ohjaajat$viikkotunnit <- c(8,8,2)
> ohjaajat
      nimi pyorailytunnit kengannumero viikkotunnit
1  Ville           3           42           8
2  Henkka          130           47           8
3 Oskari           200           42           2
>
> ohjaajat$viikkotunnit <- NULL
> ohjaajat
      nimi pyorailytunnit kengannumero
1  Ville           3           42
2  Henkka          130           47
3 Oskari           200           42

```

2.3 Tilastolliset funktiot

2.3.1 Tunnusluvut

Lasketaan tunnuslukuja esimerkkiaineistostamme. Funktio `length()` palauttaa argumentina annetun vektorin pituuden, ja `sum()` palauttaa argumentin alkioden summan, joten

näiden avulla saadaan laskettua pyöräilytuntien keskiarvo.

Esimerkki 2.14.

```
> sum(ohjaajat$pyorailytunnit) / length(ohjaajat$pyorailytunnit)
[1] 111
```

R:ssä on valmiina laaja valikoima tilastollisia funktioita, joten keskiarvo voidaan laskea helpommin käyttäen funktiota `mean()`.

Esimerkki 2.15.

```
> mean(ohjaajat$pyorailytunnit)
[1] 111
```

R:ssä puuttuvaa arvoa merkitään `NA`:lla. Jos aineistoa luettaessa solu on tyhjä, R sijoittaa sen paikalle `NA`:n. Laskettaessa tunnuslukuja, kuten keskiarvoa vektorista jossa on yksikin puuttuva arvo, R palauttaa puuttuvan arvon. Jos halutaan laskea keskiarvo niistä alkioista joilla on arvo, on `mean()`-funktiolle annettava argumentiksi `na.rm=TRUE`. Sama argumentti toimii myös monen muun funktion kanssa.

Esimerkki 2.16.

```
> ika <- c(30, 30, NA, 44)
> mean(ika)
[1] NA
> mean(ika, na.rm=TRUE)
[1] 34.66667
```

R:ssä on funktiot mm. myös keskihajonnalle, mediaanille, minimille ja maksimille.

Esimerkki 2.17.

```
> sd(ohjaajat$pyorailytunnit)
[1] 99.86491
>
> median(ohjaajat$pyorailytunnit)
[1] 130
>
> min(ohjaajat$pyorailytunnit)
[1] 3
>
> max(ohjaajat$pyorailytunnit)
[1] 200
```

2.3.2 Simulointi ja jakaumafunktiot

Tällä kurssilla keskitytään jakaumien osalta lähinnä normaali- ja binomijakaumien käsittelyyn. Näiden käsittelyä varten R:ssä on käteviä funktiota, joihin perehdytään seuraavaksi.

Seuraavissa esimerkeissä tutkitaan jakaumia $N(0, 1)$ ja $\text{Bin}(1/3, 13)$. Vastaavat funktiot löytyy myös muille keskeisille jakaumille.

Esimerkki 2.18. Kvantiilifunktiot: Haetaan piste, jonka vasemmalla puolella on $1/4$ jakauman todennäköisyysmassasta

```
# Normaalijakauma
> qnorm(mean=0, sd = 1, p = 1/4, lower=T)
[1] -0.6744898

# Binomijakauma
> qbinom(p = 1/4, size=13, prob=1/3, lower.tail = T)
[1] 3
```

Esimerkki 2.19. Tiheysfunktiot: Tiheys- ja pistetodennäköisyysfunktion $f(x)$ arvo kohdassa $x = 4$.

```
# Normaalijakauma
> dnorm(x=4, mean=0, sd = 1)
[1] 0.0001338302

# Binomijakauma
> dbinom(x = 4, size = 13, prob = 1/3)
[1] 0.2296147
```

Esimerkki 2.20. Kertymäfunktiot: Arvo kohdassa $q = 4$.

```
# Normaalijakauma
> pnorm(q = 4, mean=0, sd = 1)
[1] 0.9999683

# Binomijakauma
> pbinom(q = 4, size = 13, prob = 1/3)
[1] 0.5520387
```

Esimerkki 2.21. Jakauman simulointi: Satunnaisotos


```

# 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

```

2.4 Aineiston visuaalinen tarkastelu

Aineistoa voidaan R:ssä visualisoida monilla eri tavoilla, joista käsitellään nyt alkeiden kannalta oleelliset:

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

Tutkitaan näiden käyttöä seuraavaksi esimerkein. Piirrä kuvat R:llä nähdäksesi miltä ne näyttävät.

Esimerkki 2.22. Piirretään funktion x^2 kuvaaja välillä $[0, 1]$ käyttäen `curve()`-funktioita

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

Esimerkki 2.23. Piirretään funktion x^2 kuvaaja välillä $[0, 1]$ käyttäen `plot()`-funktioita

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

Huomaa, että tässä valinta `type='l'` käskää piirtämään kuvaajan viivoina. Kokeile myös, mitä tapahtuu ilman tätä.

Käytetään seuraavaksi R:n mukana valmiiksi tulevaa esimerkkiaineistoa Iris, johon on kerätty mittaustuloksia kolmesta erilaisesta kurjenmiekkalajista. Lisätietoja kyseisestä aineistosta saat komennolla `?iris`.

Esimerkki 2.24. Tarkastellaan kaunokurjenmiekköjen (iris setosa) terälehtien pituuksia visuaalisesti histogrammin avulla:

```
hist(iris[iris$Species=="setosa",]$Petal.Length)
```

Esimerkki 2.25. Tarkastellaan vielä kirjokurjenmiekkojen (*iris versicolor*) terälehtien pituuksia boxplotilla:

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

Esimerkki 2.26. Tarkastellaan vielä koko aineiston terälehtien pituuksia boxplotilla, lajitteluperusteena kasvin laji :

```
boxplot(iris$Petal.Length ~ iris$Species)
```

Tutkitaan seuraavaksi eri kurjenmiekkalajien terälehtien pituutta suhteessa terälehtien leveyteen. Aloitetaan ensin valitsemalla yksi laji.

Esimerkki 2.27. Tutkitaan nyt lajia *Iris setosa* ja valitaan nyt x-akselille terälehtien havaittu pituus ja y-akselille leveys ja piirretään koko komeus `plot()`-funktiolla. Huomaa, että aineiston voi antaa `plot()`-funktiolle parametrilla `data`.

```
plot(Petal.Width ~ Petal.Length,  
     data=iris[iris$Species == "setosa",])
```

Kuvaajiin saa väriä antamalla `plot()`-funktiolle parametrin `col` arvoksi värin joko tekstina (esim "red", "blue"jne) tai jonkin numeron. Myös monia muita vaihtoehtoja löytyy.

Esimerkki 2.28. Esitetään seuraavaksi *Iris*-aineiston kaikkien lajien terälehtien pituus suhteessa niiden leveyteen ja merkitään eri lajeja eri väreillä.

```
> # Katsotaan lajien järjestys  
> levels(iris$Species)  
[1] "setosa"      "versicolor" "virginica"  
  
> # Luodaan värivektori tälle järjestykselle  
> iris_colors <- c("red","green","blue")  
  
> # Piirretään kuvaaja koko aineistosta väreillä  
> plot(Petal.Width~Petal.Length, data=iris,  
      col=iris_colors[Species])
```

Viikko 3

Funktiot, faktorit ja ristiintaulukointi

Ensimmäisen ja toisen viikon harjoituksissa keskityttiin matriisien ja vektorien käsittelyyn sekä aineiston pintapuoliseen tarkasteluun. Nyt kolmannella viikolla syvennetään otetta aineistoon ja tutkitaan ohjelmoinnin näkökulmasta hieman mutkikkaampia rakenteita.

Tähän asti kaikkea on tehty vain kerran ja nyt tutustutaankin siihen, kuinka jotakin operaatioita saadaan toistettua ja koodia uudelleenkäytettyä. Samalla tutustutaan omien funktioiden kirjoittamiseen ja for-silmukkaan.

Aineiston analysoinnissa jatketaan tunnuslukujen tarkastelua ja opetellaan uusia työkaluja, joiden avulla perusanalyysi saadaan joustavammaksi ja nopeammaksi.

3.1 Omat funktiot

R:ssä on valmiina funktiot lähes kaikkeen peruskäyttöön, esimerkiksi otoskeskiarvon, -varianssin ja vastaavien laskemiseen. Joskus näitä funktiota ja niiden tuloksia on kuitenkin tarve yhdistellä uusiksi funktioiksi. Hyvin kirjoitettu funktio voidaan myös helposti siirtää uusiin koodeihin ja näin päästään käyttämään jo luotua koodia nopeasti uudestaan.

Katsotaan seuraavaksi miten voidaan tehdä omia funktioita ja miten niitä käytetään.

Esimerkki 3.1. Luodaan funktio `zeros()`, joka palauttaa N kappaletta nollia.

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

Kerrataan vielä pala kerrallaan, mitä edellisessä koodissa tapahtui.

- Ensin määritellään funktion nimi `zeros`.

- Tähän sijoitetaan sijoitusnuolella funktio, jolla on yksi argumentti, N .
- Funktion 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 3.2. Käytetään edellisen esimerkin funktiota `zeros` luomaan nollavektoreita.

```
> zeros(1)
[1] 0
> zeros(5)
[1] 0 0 0 0 0
> zeros(20)
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Tehdään seuraavaksi hieman monimutkaisempi funktio

Esimerkki 3.3. Käytetään toisen viikon tehtävissä esiteltyä funktiota `runif()` laskemaan N satunnaislukua väliltä $[a, b]$. Pyöristetään saadut arvot käyttäen funktiota `round()`, tallennetaan tulos vektoriin `int_values` ja palautetaan se:

```
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:

```
> 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
```

Huomaa: Funktion argumentit on annettava siinä järjestyksessä, missä ne on funktion määrittelyssä annettu. Tässä tapauksessa järjestyksessä on N , a , b .

Huomaa: Poikkeus edelliseen huomioon: Mikäli argumentit nimetään, voidaan niitä kutsua eri järjestyksessäkin:

Esimerkiksi `random_integers(a=0, N=2, b=2)`

3.2 Koodin osien toistaminen

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 kahteen hieman erilaiseen vaihtoehtoon.

Esimerkki 3.4. Tehdään ensin yksinkertainen tulostussilmukka ja tutkitaan sen toimintaa:

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

Mitä for-silmukka tekee? Se toistaa aaltosulkeiden sisälle kirjoitettua koodia. Katsotaan hieman tarkemmin rakennetta:

- `for(i in 1:10)` määrittelee for-silmukan ja sen sisälle muuttujan `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 kuitenkin oikeasti halutaan tulostaa.

Tutkitaan nyt mitä käy, kun edellä annetun koodin ajaa:

```
> 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
```

Samat temput voidaan usein tehdä myös komennolla `sapply`, joka palauttaa tuloksen vektorina. Tehdään nyt edellisen esimerkin koodi käyttäen `sapply()`-funktiota.

Esimerkki 3.5. `sapply()` vaatii funktion kirjoittamisen, mutta funktiot voivat olla usein hyvinkin yksinkertaisia. Annettua funktiota toistetaan jokaisella annetun vektorin arvolla, aivan kuten for-silmukassa. Lopuksi palautetuista arvoista muodostetaan vektori:

```
> sapply(1:10, function(x) { return(x) })
[1] 1 2 3 4 5 6 7 8 9 10
```

Esimerkki 3.6. Luodaan matriisi, kirjoitetaan funktio laskemaan sen sarakkeista maksimiarvoja ja lopuksi toistetaan koodi kaikille sarakkeille käyttäen sekä for-silmukkaa että `sapply()`-funktiota.

Luodaan 10x10 matriisi A:

```
A <- matrix(1:100, nrow=10)
```

Nyt tehdään funktio, joka valitsee matriisin A sarakkeen numero i ja palauttaa sen maksimiarvon:

```
max_value <- function(i) {
  return(max(A[,i]))
}
```

Nyt tehdään vektori kaikista maksimiarvoista ensin for-silmukkaa käyttäen:

```
# Tehdään ensin nollavektori tuloksia varten
values <- rep(0,10)

for(j in 1:10){
  # Sijoitetaan saatu arvo values-vektoriin
  values[j] <- max_value(j)
}
```

Nyt values-vektori voidaan tulostaa:

```
> values
[1] 10 20 30 40 50 60 70 80 90 100
```

Tehdään sama vielä käyttäen `sapply()`-funktiota:

```
> sapply(1:10, function(x) max_value(x))
[1] 10 20 30 40 50 60 70 80 90 100
```

Huomio: Vertaa tätä tulosta viikon 2 harjoitukseen 4d.

Esimerkki 3.7. Tutustutaan vielä funktioon `apply()`. Tämän funktion avulla voidaan tehdä matriisille operaatioita joko riveittäin tai sarakkeittain.

Luodaan nyt ensin matriisi `A`, otetaan siitä sarakkeet 4,5,6,7,8 ja tehdään näille laskuja. Funktion `apply()` toisena annettava argumentti (eli `MARGIN=`) määrittelee käytetäänkö rivejä vai sarakkeita: 1 = rivi, 2 = sarake.

```
> A <- matrix(
+   seq(5,9, length.out=100),
+   nrow=10)
> A_1 <- A[,4:8]
> # Riveittäiset keskiarvot
> apply(A_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
>
> # Sarakkeittaiset summat
> apply(A_1, 2, sum)
 [1] 63.93939 67.97980 72.02020 76.06061 80.10101
```

3.3 Faktorit

Edellisellä viikolla esittelimme lyhyesti R:n tietotyypeistä faktorin. Faktoreiden tarkoitus on R:ssä kuvata luokittelu- tai järjestysasteikollisia muuttujia. Luokitteluasteikollisella muuttujalla tarkoitetaan muuttujaa, jonka arvoilla ei ole selkeää järjestystä. Tällaisia ovat olla esimerkiksi sukupuoli tai kotikunta. Järjestysasteikollisella muuttujalla tarkoitetaan muuttujaa, jonka arvoilla on järjestys, mutta joiden välimatkaa ei voida yksiselitteisesti mitata. Tällaisia muuttujia ovat esimerkiksi pyöräilybarometrin muuttujat `Aq6-Aq15`, jotka mittaavat vastaajien tyytyväisyyttä erilaisiin Helsingin pyöräilyolosuhteisiin asteikolla Tyytyväinen (1), Melko tyytyväinen (2), Melko tyytymätön (3), Tyytymätön (4). Arvoilla on selkeä järjestys, mutta niillä ei voida laskea, esimerkiksi ei tiedetä onko ero "tyytyväisen" ja "melko tyytyväisen" välillä yhtä suuri kuin ero "melko tyytyväisen" ja "melko tyytymättömän" välillä.

Muuttujan tyyppin muuttaminen faktorille kertoo R:lle, että kyseessä on luokittelu- tai järjestysasteikollinen muuttuja, joten se osaa käsitellä sitä oikealla tavalla aineistoja analysoidessa, esimerkiksi regressioanalyysissä. Faktorit ovat hyödyllisiä myös piirrettäessä kuvia ja tehtäessä ristiintaulukoita; tällöin R kirjoittaa luokkien nimet automaattisesti, eikä niitä tarvitse lisätä käsin tai katsoa koodikirjasta.

3.3.1 Faktorien luominen

Kerrataan faktorin luominen. Luettaessa aineistoa tiedostosta R tekee merkkijonoista automaattisesti faktoreita, jos tiedoston taulukkoon lukevalle funktiolle ei anneta argumentiksi `stringsAsFactors=FALSE`. Aina, kuten esimerkkiaineistomme tapauksessa tämä ei ole toivottua, joten lisäsimme tiedostoa lukiessamme funktioon `read.csv()` argumentin `stringsAsFactors=TRUE`, jolloin merkkijonot (tässä tapauksessa vapaat tekstikentät) säilyivät merkkijoinoina. Jos esimerkiksi halutaan luoda faktori merkkijonotyypisistä muuttujasta, johon on tallennettu opiskelijan pääaine (Matematiikka, Taloustiede tai Tilastotiede), se tapahtuu `factor`-komennolla.

Esimerkki 3.8.

```
> paa_aine <- c("Mat", "Tal", "Tal", "Til", "Til", "Mat", "Til")
> paa_aine <- factor(paa_aine)
> str(paa_aine)
Factor w/ 3 levels "Mat","Tal","Til": 1 2 2 3 3 1 3
```

Funktiolla `str()` nähdään, että R muutti muuttujan `paa_aine` koodauksen merkkijonosta:

```
"Mat", "Tal", "Tal", "Til", "Til", "Mat", "Til"
```

numeeriseksi:

```
1 2 2 3 3 1 3
```

Luodulla faktorilla on siis kolme tasoa, "Mat", "Tal" ja "Til", eli matematiikan opiskelijat on koodattu uudelleen ykkösiksi, taloustieteen opiskelijat kakkosiksi ja tilastotieteen opiskelijat kolmosiksi.

Aineistossa on usein myös numeerisesti koodattuja luokitteluaineistollisia muuttujia, jotka halutaan muuttaa analyyseja varten faktoreiksi. Otetaan esimerkiksi opiskelijan tiedekuntaa kuvaava muuttuja `tiedekunta`, jonka arvot ovat Matemaattis-luonnontieteellinen tiedekunta (ML), joka on koodattu ykköseksi, ja Valtiotieteellinen tiedekunta (VT), joka on koodattu kakkoseksi. Myös numeerinen muuttuja muutetaan faktoriksi funktiolla `factor()`, mutta nyt funktiolle annetaan argumenttiin `labels` halutut faktorin tasojen kuvaukset merkkijonovektorina.

Esimerkki 3.9.

```
> tiedekunta <- c(1,2,2,1,1,1,2)
> tiedekunta <- factor(tiedekunta, labels=c("ML", "VT"))
> str(tiedekunta)
Factor w/ 2 levels "ML","VT": 1 2 2 1 1 1 2
```


Tarkasteltaessa faktoriksi muutettua muuttujaa funktiolla `str()` huomataan, että muuttujan numeeriset arvot pysyivät ennallaan, mutta niihin liitettiin `factor()`-funktiolle annetut tasojen kuvaukset ML ja VT.

3.4 Osa-aineistojen valinta, osa 2

3.4.1 Alkioiden valinta vektorista

Ensimmäisellä viikolla tarkastelimme vektorin alkioiden valintaa. Yksinkertaisimmillaan tämä tapahtuu sijoittamalla valintaehto vektorin indeksiksi. Jos vektorissa on puuttuvia arvoja, lopputulos ei kuitenkaan aina ole toivottu. Yritetään esimerkiksi selvittää kuinka moni vektorin `a` arvoista on alle kuusi `length()`-funktion ¹ avulla.

Esimerkki 3.10.

```
> 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` alkiuille, 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 3.11.

```
> which(a<6)
[1] 1 6
> a[which(a<6)]
[1] 3 5
> length(a[which(a<6)])
[1] 2
```

¹Komento `sum(a<6, na.rm=T)` olisi tietenkin yksinkertaisempi tapa selvittää asia.

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 suurempi 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 3.12.

```
> subset(a, a<6)
[1] 3 5
> length(subset(a, a<6))
[1] 2
```

3.4.2 Osa-aineistojen valinta taulukosta

Tehdään taulukko (kuvitteellisista) R-kurssin osallistujista. Lisätään esimerkkiaineistomme muuttuja `tutkinto`, joka kuvaa sitä onko opiskelija kandi- vai maisterivaiheessa. Arvotaan vielä opiskelijanumerot ja liitetään muuttujat yhteen `opiskelijat`-taulukoksi.

Esimerkki 3.13.

```
> tutkinto <- c(1,1,1,NA,2,NA,1)
> tutkinto <- factor(tutkinto, labels=c("kandi", "maisteri"))
> opiskelijaNro <- runif(length(paa_aine), min=1000000, max=1500000)
> opiskelijat <- data.frame(opiskelijaNro, paa_aine, tiedekunta, tutkinto)
> opiskelijat
```

	opiskelijaNro	paa_aine	tiedekunta	tutkinto
1	1289002	Mat	ML	kandi
2	1404864	Tal	VT	kandi
3	1058537	Tal	VT	kandi
4	1153128	Til	ML	<NA>
5	1315334	Til	ML	maisteri
6	1074686	Mat	ML	<NA>
7	1180208	Til	VT	kandi

Osasta opiskelijoista ei ole tietoa heidän opiskelujensa vaiheesta, joten `tutkinto`-muuttuja saa puuttuvia arvoja. Oletetaan, että halutaan tarkastella lähemmin kurssin kandivaiheessa olevia opiskelijoita. Niihin lähestymistapa, jossa valintaehto sijoitetaan `opiskelijat`-taulukon rivi-indeksin paikalle, tuottaa hieman ikävän näköisen lopputuloksen.

Esimerkki 3.14.

```

> opiskelijat[opiskelijat$tutkinto == 'kandi', ]
  opiskelijaNro paa_aine tiedekunta tutkinto
1      1289002      Mat          ML   kandi
2      1404864      Tal          VT   kandi
3      1058537      Tal          VT   kandi
NA              NA      <NA>      <NA>   <NA>
NA.1           NA      <NA>      <NA>   <NA>
7      1180208      Til          VT   kandi
>
> nrow(opiskelijat[opiskelijat$tutkinto == 'kandi', ])
[1] 6

```

Valittuun osa-aineistoon tulivat mukaan myös rivit, joissa `tutkinto`-muuttujan arvo puuttui, ja kaikki muutkin näiden rivien arvot muuttuivat puuttuviksi ². Tämä ei liene haluttu tulos.

Siistimpi lopputulos saadaan joko jälleen `which()`-funktion avulla, tai sitten soveltamalla suoraan `subset()`-funktiota taulukoille. Huomaa, että käytettäessä `subset()`:iä taulukkoon sarakkeen nimeä ei tarvitse kirjoittaa muodossa `opiskelijat$tutkinto`, vaan pelkkä sarakkeen nimi riittää.

Esimerkki 3.15.

```

> subset(opiskelijat, tutkinto == 'kandi')
  opiskelijaNro paa_aine tiedekunta tutkinto
1      1289002      Mat          ML   kandi
2      1404864      Tal          VT   kandi
3      1058537      Tal          VT   kandi
7      1180208      Til          VT   kandi
>
> nrow(subset(opiskelijat, tutkinto == 'kandi'))
[1] 4

```

3.4.3 Tunnuslukujen laskeminen osa-aineistoittain

Liitetään esimerkkiaineistoamme R-kurssin osallistujista vielä muuttuja opintopisteet, joka kertoo opiskelijan kurssin alkuun mennessä suorittamat opintopisteet.

Esimerkki 3.16.

²Funktio `nrow()` palauttaa argumentiksi annetun matriisin tai taulukon rivien määrän ja `ncol()` taas palauttaa sarakkeiden määrän.

```

> opiskelijat$opintopisteet <- c(20, 33, 55, 120, 230, 172, 50)
> opiskelijat
  opiskelijaNro paa_aine tiedekunta tutkinto opintopisteet
1      1196156      Mat      ML      kandi           20
2      1154825      Tal      VT      kandi           33
3      1072480      Tal      VT      kandi           55
4      1468267      Til      ML      <NA>          120
5      1102544      Til      ML maisteri          230
6      1448363      Mat      ML      <NA>          172
7      1345642      Til      VT      kandi           50

```

Edellisellä viikolla laskimme tunnuslukuja osa-aineistoittain, esimerkiksi keski-iat miehille ja naisille erikseen pyöräilybarometri-aineistosta. Tämä onnistuu kätevästi myös R:stä vakiona löytyvällä `apply`-perheen funktiolla `tapply()`. Käytettäessä `tapply()`-funktioita osa-aineistoja ei valita erikseen käsin, vaan `tapply()` tekee sen automaattisesti. Lasketaan esimerkiksi opintopisteiden keskiarvot pääaineittain.

Esimerkki 3.17.

```

> tapply(opiskelijat$opintopisteet, opiskelijat$paa_aine, mean)
      Mat      Tal      Til
96.0000  44.0000 133.3333

```

Funktio `tapply()` hajottaa ensin ensimmäisenä argumenttina annetun vektorin, tässä tapauksessa `opintopisteet`, ryhmiin toisena argumenttina annetun faktorin, tässä tapauksessa `paa_aine`, mukaan. Sen jälkeen `tapply()` suorittaa kolmantena argumenttina annetun funktion, tässä tapauksessa `mean()`:in, kuhunkin osa-aineistoon ja palauttaa tuloksen nimettynä vektorina ³.

3.5 Ristiintaulukointi

3.5.1 Frekvenssitaulu

Muuttujan jakaumaa voidaan tutkia funktiolla `table()`, joka palauttaa kutakin vektorin alkioden saamaa arvoa kohti, kuinka moni vektorin alkioista saa tämän arvon.

Esimerkki 3.18.

```
a <- c(10,22,438,7,22,22,10,438,438,22,22,22)
```

³Nimetty vektori on muuten kuten tavallinen vektori, mutta sen alkioilla on nimet joilla niihin voidaan viitata.

```
table(a)
a
  7  10  22 438
  1   2   6   3
```

Voidaan esimerkiksi selvittää kuinka moni kurssin osallistujista suorittaa kandidaatin- ja kuinka moni maisterintutkintoa. Tällaista muuttujan eri arvojen määriä kuvaavaa taulua kutsutaan frekvenssitauluksi. Huomaa, että `table()` jättää puuttuvat arvot automaattisesti huomioimatta. Nähdään myös, että R tulostaa taulun sarakkeiden otsikoiksi automaattisesti faktorin tasot, mikä helpottaa tulkintaa.

Esimerkki 3.19.

```
> table(opiskelijat$tutkinto)
```

```
  kandi maisteri
      4         1
```

3.5.2 Ristiintaulukko

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*). Taulukoidaan kurssin opiskelijoiden pääaine ja tiedekunta vastakkain.

Esimerkki 3.20.

```
> table(opiskelijat$paa_aine, opiskelijat$tiedekunta)
```

```
      ML VT
Mat   2  0
Tal   0  2
Til   2  1
```

Nähdään että kurssin osallistujista kaikki matemaatikot opiskelevat matemaattis-luonnontieteellisessä tiedekunnassa ja taloustieteilijät valtiotieteellisessä. Tilastotieteilijöitä taas opiskelee kummassakin tiedekunnassa.

Ristiintaulukoista pystyy valitsemaan alkoita, rivejä ja sarakkeita kuten matriisista. Valitaan edellisen esimerkin taulun ensimmäisen rivin ensimmäisen sarakkeen alkio, ja sen jälkeen koko ensimmäinen rivi.

Esimerkki 3.21.

```

> paa_aine_X_tiedekunta <- table(opiskelijat$paa_aine, opiskelijat$tiedekunta)
> paa_aine_X_tiedekunta[1,1]
[1] 2
> paa_aine_X_tiedekunta[1,]
ML VT
 2  0

```

Tauluihin pystyy myös käyttämään funktioita riveittäin ja sarakkeittain `apply()`-funktioilla. Lasketaan esimerkkitaulun sarakesummat, eli kurssin opiskelijoiden kokonaismäärä tiedekunnittain.

Esimerkki 3.22.

```

> apply(paa_aine_X_tiedekunta,2,sum)
ML VT
 4  3

```

Rivi- ja sarakeprosentit saadaan funktiolla `prop.table()`, jolle annetaan argumentiksi alkuperäinen taulu, josta prosentit halutaan laskea ja 1 tai 2 sen mukaan, halutaanko rivi- vai sarakeprosentit. Lasketaan esimerkkitaulustamme sarakeprosentit, eli kunkin pääaineen edustajien osuudet tiedekunnittain kurssin osallistujista.

Esimerkki 3.23.

```

prop.table(paa_aine_X_tiedekunta,2)

```

	ML	VT
Mat	0.5000000	0.0000000
Tal	0.0000000	0.6666667
Til	0.5000000	0.3333333

Viikko 4

Aineiston luokittelu ja luottamusvälit

Tällä viikolla tutkitaan aineistoa luokittelun, summamuuttujien ja luottamusvälien avulla. Koska esimerkit vaativat jo jonkinlaisen yksinkertaisen aineiston käyttöä, otetaan tarkasteluun taulukko A. Voit luoda kyseisen taulukon omia kokeilujasi varten seuraavalla koodilla:

```
A <- data.frame(matrix(1:100, nrow=20))
colnames(A) <- c("A1", "A2", "A3",
                "A4", "A5")
```

Taulukossa on siis viisi saraketta, nimeltään A1 - A5, ja 20 riviä.

Neljännän viikon tehtävät vaativat paikoin pohdintaa ja edellisten viikkojen osaamisen yhdistämistä.

4.1 Osa-aineistojen valinta, osa 3

4.1.1 Sarakkeiden valinta nimellä

Viikolla 3 tutustuttiin `subset()`-funktion käyttöön osa-aineistojen valinnassa. Seuraavaksi katsotaan miten `subset()` toimii osa-aineistojen valinnassa laajemmin.

Esimerkki 4.1. Valitaan taulukosta A sarakkeet A2 ja A3 ja tallennetaan valittu osa-aineisto muuttujaan A_23

```
A_23 <- subset(A, select=c("A2", "A3"))
```

Esimerkki 4.2. Valitaan seuraavaksi taulukosta A sarakkeista A2 ja A3 ne rivit, joissa $A4 > 70$. Tallennetaan valittu osa-aineisto muuttujaan A_23_yli70

```
A_23_yli70 <- subset(A, A4 > 70, select=c("A2", "A3"))
```

Monien sarakkeiden valinta kirjoittaen jokaisen valittavan sarakkeen nimi käsin on saattaa olla huomattavan työlästä. Usein aineistossa sarakkeet nimetään ja numeroidaan jollakin loogisella tavalla, joka noudattaa helposti toistettavaa kaavaa. Käydään seuraavaksi läpi yksi nopeahko tapa valita tällaisessa tapauksessa useita nimiä samalla kerralla.

Esimerkki 4.3. Katsotaan ensin miten toimii `paste()`-funktio:

```
> paste("Sarake",1:5)
[1] "Sarake 1" "Sarake 2" "Sarake 3" "Sarake 4" "Sarake 5"
> paste("Sarake",1:5, sep="")
[1] "Sarake1" "Sarake2" "Sarake3" "Sarake4" "Sarake5"
> paste("Sarake",1:5, sep="-")
[1] "Sarake-1" "Sarake-2" "Sarake-3" "Sarake-4" "Sarake-5"
```

Tässä siis argumentti `sep` kertoo mitä tekstin ja numeron väliin tulee. Voidaan myös muodostaa sarakenimiä, joissa on kirjaimia nimen ja numeron perässä.

Esimerkki 4.4. Katsotaan miten `paste()`-funktiolla voidaan muodostaa hivenen monimutkaisempia sarakenimiä:

```
> paste("Sarake", 1, letters[1:5], sep="")
[1] "Sarake1a" "Sarake1b" "Sarake1c" "Sarake1d" "Sarake1e"
> paste("Sarake", 1, LETTERS[1:5], sep="")
[1] "Sarake1A" "Sarake1B" "Sarake1C" "Sarake1D" "Sarake1E"
```

Esimerkki 4.5. Käytetään nyt `paste()`-funktiota muodostamaan sarja kiinnostavien sarakkeiden nimiä esimerkkitaulukosta A. Nyt esimerkkitaulukossa A on vain viisi saraketta, mutta valitaan ne kaikki:

```
# Valittavat sarakkeet:
> paste("A",1:5, sep="")
[1] "A1" "A2" "A3" "A4" "A5"
# Sitten valitaan:
subset(A, select=paste("A",1:5, sep=""))
```

Esimerkki 4.6. Jatketaan vielä edellistä esimerkkiä ja valitaan sarakkeista A1, A3, A5 ne rivit, joilla A2 on parillinen:


```
> subset(A, A2%%2 == 0, select=paste("A",seq(1,5,by=2), sep=""))
  A1 A3 A5
2   2 42 82
4   4 44 84
6   6 46 86
8   8 48 88
10  10 50 90
12  12 52 92
14  14 54 94
16  16 56 96
18  18 58 98
20  20 60 100
```

4.2 Aineiston luokittelu

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

Esimerkki 4.7. Luokitellaan nyt taulukon A sarake A1 kahteen luokkaan:

```
> cut(A$A1, breaks=c(0,11,20))
 [1] (0,11] (0,11] (0,11] (0,11] (0,11] (0,11] (0,11] (0,11] (0,11] (0,11]
[11] (0,11] (11,20] (11,20] (11,20] (11,20] (11,20] (11,20] (11,20] (11,20] (11,20]
Levels: (0,11] (11,20]
```

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

Esimerkki 4.8. Jatketaan edellisen esimerkin luokittelua ja jaetaan taulukon A sarake A1 tällä kertaa kolmeen ja viiteen luokkaan. Katsotaan tuloksena saatavaa vektoria selvytyden vuoksi `table()`-komennon avulla:

```
> table(cut(A$A1, breaks=c(0,11,15,20)))

(0,11] (11,15] (15,20]
      11      4      5
> table(cut(A$A1, breaks=c(0,6,10,15,18,20)))

(0,6] (6,10] (10,15] (15,18] (18,20]
   6     4     5     3     2
```

Esimerkki 4.9. Jatketaan edellistä esimerkkiä ja lasketaan kolmeen luokkaan jaetusta aineistosta luokkakohtaiset otoskeskihajonnat sarakkeelle A3 käyttäen `tapply()`-funktiota:

```
> luokittelu <- cut(A$A1, breaks=c(0,11,15,20))
> tapply(A$A3, luokittelu, sd)
 (0,11] (11,15] (15,20]
3.316625 1.290994 1.581139
```

4.3 T-luottamusväli

Oletetaan, että käytettävän aineiston havainnot ovat otos normaalijakaumasta tuntemattomien parametrein μ ja σ^2 . Tutustutaan estimaatin $\hat{\mu}$ luottamusvälin laskemiseen. Lisätietoa estimaateista ja luottamusväleistä löytyy samaan aikaan käynnissä olevan Johdatus tilastolliseen päättelyyn -kurssin luentomonisteesta.

Esimerkki 4.10. 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 komenolla `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 4.11. 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.

4.4 Kuvien piirtämisestä

Kuvien piirtäminen on aineiston analysoinnissa tärkeä vaihe. Kaikkia kuvia ei voi kuitenkaan piirtää `plot()`-funktioilla suoraan, vaan kuva joudutaan piirtämään osissa. Tällaisia tilanteita tulee esimerkiksi silloin, kun halutaan piirtää useita asioita samaan kuvaan.

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

Esimerkki 4.12. Luodaan tyhjä kuvaaja. On kuitenkin syytä kertoa `plot()`-funktioille x- ja y-akselien minimi- 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 4.13. 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 4.14. Piirretään nyt esimerkin 4.12 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")
```

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

Katsotaan vielä, 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 4.15. 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.

Viikko 5

Listat ja tilastollinen testaaminen

Tällä viikolla tutustutaan viimeiseen R:n tärkeistä tietorakenteista, eli listaan, joka mahdollistaa eri tietotyyppiä olevien ja eripituisten vektorien tallentamisen samaan tietorakenteeseen. Lista on kätevä esimerkiksi silloin kun halutaan että funktio palauttaa useampia erityyppisiä arvoja, mikä onkin tällä viikolla sen pääasiallinen käyttötarkoitus. Kuitenkin perehdyttäessä syvemmin ohjelmointiin R:llä listalle löytyy monia muitakin käyttötarkoituksia, esimerkiksi R:n luokkarakenne perustuu listoihin.

Tällä viikolla päästään lopultakin myös pureutumaan tilastolliseen testaamiseen. Testeistä käydään läpi yhden otoksen t-testi ja riippumattomuustesti ristiintaulukolle, eli ns. Khiin neliön testi.

5.1 Lista

5.1.1 Listan luominen

R:n tietotyypeistä vektoriin voi tallentaa vain yhden tyyppisiä alkioita kerrallaan. Jos yrittää yhdistää useamman eri tyyppin alkioita vektoriksi, R pakottaa muuttujat kaikki alkiot automaattisesti "alimman" tason tietotyyppiksi, esimerkkinä tapauksessa merkkijonoksi.

Esimerkki 5.1.

```
> vektori <- c("moi", 3.14159, 1:10, c(T,T,F))
> vektori
 [1] "moi"      "3.14159" "1"      "2"      "3"      "4"      "5"
 [8] "6"        "7"        "8"      "9"      "10"     "TRUE"   "TRUE"
[15] "FALSE"
> class(vektori)
[1] "character"
```

Taulukkoa luotaessa taas taulukon sarakkeiden tulee olla samanpituisia ¹. Haluttaessa tallentaa erityyppisiä ja eripituisia vektoreita samaan tietorakenteeseen, tarvitaan R:n tietotyyppiä `list`, eli lista. Lista voi koostua numeerisista, merkkijono- tai totuusarvo-vektoreista, toisista listoista, taulukoista, tai oikeastaan mistä tahansa R:n olioista. Luodaan lista, joka sisältää merkkijonon, skalaarin (eli yksialkioisen numeerisen vektorin, numeerisen vektorin ja merkkijonovektorin. Listan komponentit annetaan yksinkertaisesti `list()`-funktiolle, ja erotellaan pilkuilla.

Esimerkki 5.2.

```
lista <- list("moi", 3.14159, 1:10, c(T,T,F))
> lista
[[1]]
[1] "moi"

[[2]]
[1] 3.14159

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

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

Listasta voidaan valita alilistoja samalla tavalla kuin vektorista alkioita.

Esimerkki 5.3.

```
> lista[1]
[[1]]
[1] "moi"

> lista[1:3]
[[1]]
[1] "moi"

[[2]]
[1] 3.14159
```

¹Oikeastaan taulukko on pohjimmiltaan lista, jonka alkiot (eli taulukon sarakkeet) ovat samaa tietotyyppiä olevia samanpituisia vektoreita.

```
[[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 5.4.

```
> kolmas <- lista[3]
> kolmas
[[1]]
[1] 1 2 3 4 5 6 7 8 9 10

> class(kolmas)
[1] "list"
> kolmas[1]
[[1]]
[1] 1 2 3 4 5 6 7 8 9 10
```

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

Esimerkki 5.5.

```
> kolmas <- lista[[3]]
> kolmas
[1] 1 2 3 4 5 6 7 8 9 10
> class(kolmas)
[1] "integer"
> kolmas[1]
[1] 1
```

5.1.2 Listan komponenttien nimeäminen

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

Esimerkki 5.6.

```
> opiskelija <- list(nimi="Aslak", opiskelijanro=sample(x=1000000:1500000, size=1),
+                   aloitusvuosi=2012, kurssit=c(57045, 57046, 57703))
```

```
> opiskelija
```

```
$nimi
```

```
[1] "Aslak"
```

```
$opiskelijanro
```

```
[1] 1468267
```

```
$aloitusvuosi
```

```
[1] 2012
```

```
$kurssit
```

```
[1] 57045 57046 57703
```

Nimetyin listan komponentteihin voidaan viitata `$`-operaattorilla samalla tavalla kuin taulukon sarakkeisiin. Nimellä viittaaminen antaa komponentin alkuperäisessä muodossaan samalla tavalla kuin tuplahakasulut. Kuten taulukoiden sarakkeiden tapauksessa, nimellä viittaaminen on suositeltavampaa kuin järjestysnumerolla viittaminen, sillä se toimii edelleen, vaikka listaan lisäksi tai poistaisi komponentteja tai vaihtaisi niiden järjestystä.

Esimerkki 5.7.

```
> opiskelija$nimi
```

```
[1] "Aslak"
```

```
> opiskelija[[1]]
```

```
[1] "Aslak"
```

```
> opiskelija$kurssit[1]
```

```
[1] 57045
```

Listaan komponentteja voi muuttaa ja niitä voi lisätä ja poistaa samalla tavalla kuin taulukon sarakkeita. Lisätään Aslakin suorittamiin kursseihin kurssi 57798, "Tilastotieteen juuret", ja lisätään uusi `paa_aine`-komponentti.

Esimerkki 5.8.

```
> opiskelija$kurssit <- c(opiskelija$kurssit, 57798)
```

```
> opiskelija$paa_aine <- "tilastotiede"
```

```
> opiskelija
```

```
$nimi
```

```
[1] "Aslak"
```

```
$opiskelijanro
```

```
[1] 1468267
```



```
$aloitusvuosi  
[1] 2012
```

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

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

Poistetaan paa_aine-komponentti sijoittamalla siihen tyhjäarvo NULL.

Esimerkki 5.9.

```
> opiskelija$paa_aine <- NULL  
> opiskelija  
$nimi  
[1] "Aslak"
```

```
$opiskelijanro  
[1] 1102544
```

```
$aloitusvuosi  
[1] 2012
```

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

5.1.3 Listan läpikäyminen

Jos halutaan käydä kaikki listan komponentit läpi, voidaan tietenkin kirjoittaa for-silmukka, joka käy läpi kaikki listan indeksit. Kätevämpi tapa lienee kuitenkin käydä lista läpi `apply`-perheeseen kuuluvalla `lapply()`-funktioilla, joka soveltaa toisena argumenttina annettua funktiota ensimmäisenä argumenttina annetun listan jokaiseen komponenttiin. Lasketaan esimerkiksi summa kolme numeerista vektoria sisältävän listan jokaisesta komponentissa.

Esimerkki 5.10.

```
> numerolista <- list(1:10, c(2,4,6), 15)  
> numerolista  
[[1]]
```

```

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

[[2]]
[1] 2 4 6

[[3]]
[1] 15

>
> lapply(numerolista, sum)
[[1]]
[1] 55

[[2]]
[1] 12

[[3]]
[1] 15

```

Tarkka lukija saattoi tässä vaiheessa huomata, että edellisen esimerkin olisi voinut yhtä hyvin tehdä aiemmin esitellyllä `sapply()`-funktioilla. Tämä on totta, itse asiassa `sapply()`, eli *simplified apply* on vain ns. wrapper `lapply()`:lle, eli se tekee saman asian kuin `lapply()`, eli soveltaa toisena argumenttina annettua funktiota jokaiseen ensimmäisenä argumenttina annetun listan komponenttiin². Erona on vain se, että `sapply()` muuttaa tuloksen automaattisesti vektoriksi tai matriisiksi jos se on mahdollista.

Seuraavassa esimerkissä, jossa korotetaan listan jokainen komponentti toiseen potenssiin, `lapply()`:n käyttö on ehkä hieman perustellumpaa, sillä tulos halutaan todennäköisesti edelleen listan muodossa.

Esimerkki 5.11.

```

> lapply(numerolista, function(x) x^2)
[[1]]
[1] 1 4 9 16 25 36 49 64 81 100

[[2]]
[1] 4 16 36

```

²Jos `sapply()`:lle annetaan ensimmäiseksi argumentiksi vektori, R muuttaa sen automaattisesti listaksi.

```
[[3]]  
[1] 225
```

5.2 Lisää funktioista

Edellisen kappaleen viimeisen esimerkin funktiosta joka korotti listan komponentit toiseen potenssiin puuttuivat aaltosulut ja return-rivi. Jos aaltosulkujen sisälle tulisi vain yksi rivi, ne eivät ole välttämättömiä. Samoin jos funktiossa ei ole return-riviä, R palauttaa automaattisesti viimeiseksi käsitellyn arvon. Sama funktio voidaan siis kirjoittaa lyhyemmin käyttäen hyväksi näitä ominaisuuksia:

Esimerkki 5.12.

```
> toinen_potenssi <- function(x) {  
  return(x^2)  
}  
>  
> toinen_potenssi2 <- function(x) x^2  
>  
> toinen_potenssi(1:10)  
[1] 1 4 9 16 25 36 49 64 81 100  
> toinen_potenssi2(1:10)  
[1] 1 4 9 16 25 36 49 64 81 100
```

5.2.1 Funktion palautusarvon kirjoittaminen näkyviin

Usein R-koodissa funktion palautusarvo jätetään edellisen esimerkin tapaan kirjoittamatta jos se on viimeinen funktion käsittelemä arvo, varsinkin edellisen kaltaisissa yhden rivin funktioissa joissa se on ainoa funktion palauttama arvo. Tämän kanssa täytyy kuitenkin olla tarkkana. Seuraava esimerkki, jossa vektorin x n :nen potenssin laskeva funktio on toteutettu ensin kirjoittamalla palautusarvolla näkyviin, ja sen jälkeen ilman palautusarvoa, toimikoon varoittavana esimerkkinä.

Esimerkki 5.13.

```
> potenssi <- function(x,n) {  
  tulos <- 1  
  for(i in 1:n)  
    tulos <- tulos * x  
  return(tulos)  
}
```

```

>
> potenssi2 <- function(x,n) {
  tulos <- 1
  for(i in 1:n)
    tulos <- tulos * x
  }
>
>
> p <- potenssi(10, 3)
> p2 <- potenssi2(10, 3)
>
> p
[1] 1000
> p2
NULL

```

Nyt funktion toinen versio, jossa palautettava arvo on jätetty kirjoittamatta, palauttaakin odotetun tuloksen 1000 sijasta tyhjäärvon NULL. Tämä johtuu siitä, että viimeinen R:n käsittelemä arvo ei ole kolmannessa `for`-silmukan toistossa laskettava $100 \cdot 10 = 1000$. R:ssä nimittäin myös itse `for`-silmukka on funktio, joka palauttaa *näkymättömän* arvon (eli arvon joka ei tulostu näytölle komentoa suoritettaessa), tässä tapauksessa tyhjäärvon NULL. Tämä on viimeinen funktion `potenssi2` käsittelemä arvo, ja siten myös sen palautusarvo. Joten jos kyseessä on pidempi kuin yhden rivin funktio, funktion palautusarvo kannattaakin aina varmuuden vuoksi kirjoittaa näkyviin `return`:illa, ellei ole aivan varma siitä että tietää mitä tekee.

5.2.2 Koodiblokit ja koodin sisennys

Aaltosulut erottavat niin sanotun *koodiblokin*, joka voi olla esimerkiksi funktion tai `for`-silmukan sisältö. Hyvään ohjelmointityyliin kuuluu lähes kaikissa ohjelmointikielissä, myös R:ssä, sisentää koodiblokin sisältöä esimerkiksi yhden sarkaimen verran luettavuuden helpottamiseksi, kuten edellisessä esimerkissä on tehty. Jos koodiblokki sisältää vain yhden rivin, kuten edellisen esimerkin `for`-silmukka, joka sisältää vain rivin `tulos <- tulos * x`, aaltosulut voi jättää pois. Esimerkiksi seuraavat kaksi tapaa tulostaa lukujen 1-3 toiset potenssit toimivat aivan samalla tavalla.

Esimerkki 5.14.

```

> for(i in 1:3) {
  print(i^2)
}

```

```

[1] 1
[1] 4
[1] 9
> for(i in 1:3)
  print(i^2)
[1] 1
[1] 4
[1] 9
>

```

Huomaa että vaikka aaltosulut jätetään pois, koodiblokki sisennetään edelleen luettavuuden takia. Rstudiota tai muuta ohjelmointiympäristöä käytettäessä ohjelmointiympäristö pyrkii usein sisentämään koodia automaattisesti.

5.2.3 replicate

Viime viikon tehtävässä kuusi mallivastauksessa käytettiin `sapply`:a kahdensadan normaalijakaumasta otetun otoksen luottamusvälien laskemiseen.

Esimerkki 5.15.

```
> luottamus <- sapply(1:200, function(x) t.test(rnorm(n=100, mean=0, sd=1))$conf.int)
```

Koodissa ei kuitenkaan käytetty määritellyn funktion argumenttia `x` (joka siis saa arvot 1-200) mitenkään, vaan haluttiin ainostaan toistaa funktiokutsu `t.test(rnorm(n=100, mean=0, sd=1))$conf.int` 200 kertaa. Tämä onnistuu siistimmän näköisesti `replicate`-funktioilla, jolle annetaan ensimmäiseksi argumentiksi montako kertaan toisena argumenttina annettu funktio halutaan toistaa. Ylläoleva rivi voidaan kirjoittaa `replicate`:n avulla seuraavasti.

Esimerkki 5.16.

```
> luottamus2 <- replicate(200, t.test(rnorm(n=100, mean=0, sd=1))$conf.int)
```

Erityisen hyvin `replicate` sopii juuri ylläolevan kaltaisiin tilanteisiin, jossa halutaan simuloida useita otoksia samasta jakaumasta. Seuraavassa vielä yksinkertaisempi esimerkki, jossa simuloidaan ensin `sapply`:llä viisi kahden pituista otosta välin (0,1) tasajakau-
masta, ja sen jälkeen tehdään sama `replicate`:lla.

Esimerkki 5.17.

```
> sapply(1:5, function(x) runif(n=2))
      [,1]      [,2]      [,3]      [,4]      [,5]

```

```

[1,] 0.9707293 0.7077211 0.04963769 0.8592144 0.505355
[2,] 0.5519202 0.1321746 0.06600105 0.7088197 0.340548
>
> replicate(5, runif(n=2))
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.5331686 0.0314747 0.4623011 0.4429368 0.8823194
[2,] 0.9104867 0.2517356 0.9978928 0.4128540 0.2310118

```

Oikeastaan `replicate` onkin vain ns. "wrapper"`sapply`:lle, eli yllä olevan esimerkin alempi versio tekee R:n kannalta täsmälleen saman asian kuin ylempi.

5.3 Yhden otoksen T-testi

Tutkitaan jälleen R:n mukana tulevaa klassista Iris-esimerkkiaineistoa, ja erityisesti eri kurjenmiekkalajien terälehtien pituuksia. Testataan eroaako kaunokurjenmiekkojen (*iris setosa*) terälehtien pituus 5:stä, eli testataan t-testillä³. 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 luonnollisesti jo viime viikolla luottamusvälien laskemiseen käytetyllä `t.test`-funktioilla. Ensimmäiseksi argumentiksi annetaan vektori, jossa on testattava aineisto, ja argumentti `mu` määrittelee nollahypoteesiarvon μ_0 . Argumentti `alternative` määrittelee, onko vastahypoteesi kaksisuuntainen, ja jos se on yksisuuntainen, niin kumpaan suuntaan. Sitä ei kuitenkaan tarvitse tässä määritellä, sillä oletusarvo on kaksisuuntainen.

Esimerkki 5.18.

```

> iris_setosa <- subset(iris, iris$Species == "setosa")
> t.test(iris_setosa$Sepal.Length, mu=5)

```

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

```

³T-testi esitellään kurssin Johdatus tilastolliseen päättelyyn monisteen luvussa 6

T-testin tulosteen toinen rivi kertoo t-testi-suureen arvon 0.12, t:n jakauman vapausasteen 49 (aineistossa oli 50 havaintoa, jolloin vapausaste on $50 - 1 = 49$) ja p-arvon 0.90. Nollahypoteesia siitä, että kaunokurjenmiekkojen terälehtien pituuden keskiarvo olisi 5, ei siis voida hylätä. Seuraava rivi kertoo testissä käytetyn vastahypoteesin (ja siten implisiit-tisesti myös nollahypoteesin). Seuraavalla rivillä on jo viime kerralta tuttu 95 prosentin luottamusväli pituuden keskiarvolle. Viimeisellä rivillä on otoskeskiarvo 5.006, joka on tosiaan 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 5.19.

```
> t.test(iris[iris$Species == "setosa", ]$Sepal.Length, mu=5)
```

Ainakin omasta mielestäni ylempi versio on huomattavasti luettavampi ja vähemmän herkempi virheille.

Tallennetaan `t.test`-funktion palauttama olio muuttujaan sen lähempää tarkastelua varten.

Esimerkki 5.20.

```
> str(t_setosa)
List of 9
 $ statistic   : Named num 0.12
  ..- attr(*, "names")= chr "t"
 $ parameter   : Named num 49
  ..- attr(*, "names")= chr "df"
 $ p.value     : num 0.905
 $ conf.int    : atomic [1:2] 4.91 5.11
  ..- attr(*, "conf.level")= num 0.95
 $ estimate    : Named num 5.01
  ..- attr(*, "names")= chr "mean of x"
 $ null.value  : Named num 5
  ..- attr(*, "names")= chr "mean"
 $ alternative : chr "two.sided"
 $ method      : chr "One Sample t-test"
 $ data.name   : chr "iris_setosa$Sepal.Length"
 - attr(*, "class")= chr "htest"
```

Funktion `t.test` palautusarvo on siis lista ⁴! Voimme poimia sen komponentteja niiden

⁴Itse asiassa `t.test`:in palautusarvo on luokan `htest` olio, kuten tulosteen viimeiseltä riviltä nähdään.

nimen avulla, kuten teimme jo viime viikolla luotamusvälien tapauksessa.

Esimerkki 5.21.

```
> t_setosa$conf.int
[1] 4.905824 5.106176
attr(,"conf.level")
[1] 0.95
>
> t_setosa$p.value
[1] 0.9046885
```

Koska kysymyksessä on lista, komponentteja voi poimia myös niiden indeksiä käyttäen.

Esimerkki 5.22.

```
> t_setosa[[4]]
[1] 4.905824 5.106176
attr(,"conf.level")
[1] 0.95
>
> t_setosa[[3]]
[1] 0.9046885
```

5.4 Riippumattomuustesti

Tilastollisten testien avulla voidaan tutkia aineistosta myös asioiden välisiä riippuvuus-suhteita. Tutustutaan seuraavaksi Karl Pearsonin kehittämään riippumattomuustestiin, jonka testisuure on asympotoottisesti χ^2 -jakautunut. Testin tarkempia yksityiskohtia voit tutkia esimerkiksi Johdatus tilastolliseen päättelyyn -kurssin kurssimonisteesta, tällä kursilla keskitymme testin soveltamiseen.

Esimerkki 5.23. Tutustutaan nyt fiktiivisen leipomon toimintaan ja sen tuotekehityksessä ilmenneeseen ongelmaan. Leipomossa on nimittäin huomattu, että taikinan kohoaminen on ajoittain huteraa ja 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.

R:n ns. S3-tyyppin (Toinen R:n luokkatyyppi on uudempi S4, joka muistuttaa määrittelyltään enemmän muiden olio-orientoituneiden kielten, kuten Javan, luokkia. Suurin osa R:n luokista on kuitenkin S3-tyyppin luokkia.) olio on yksinkertaisesti lista, johon on liitetty luokan nimeävä attribuutti, tässä tapauksessa `hstest`.

	Hiiva 1	Hiiva 2
Kohoaminen OK	120	173
Ei kohonnut	80	27

Tutkitaan seuraavaksi, olisiko leipurilla tilastollisia perusteita vaihtaa hiivan toimitajaa. 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
```

Tässä 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 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 testisuureen asymptoottisen jakauman vapausasteet ja tieto siitä, että halutaan käyttää yläkvanttileja:

```
> # 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
```

Viikko 6

Lineaarinen regressio ja Bayes-päätely

Viimeistä viedään! Kuudennen viikon teemana ovat lineaarinen malli ja Bayes-päätely.

6.1 Yhden selittäjän lineaarinen regressio

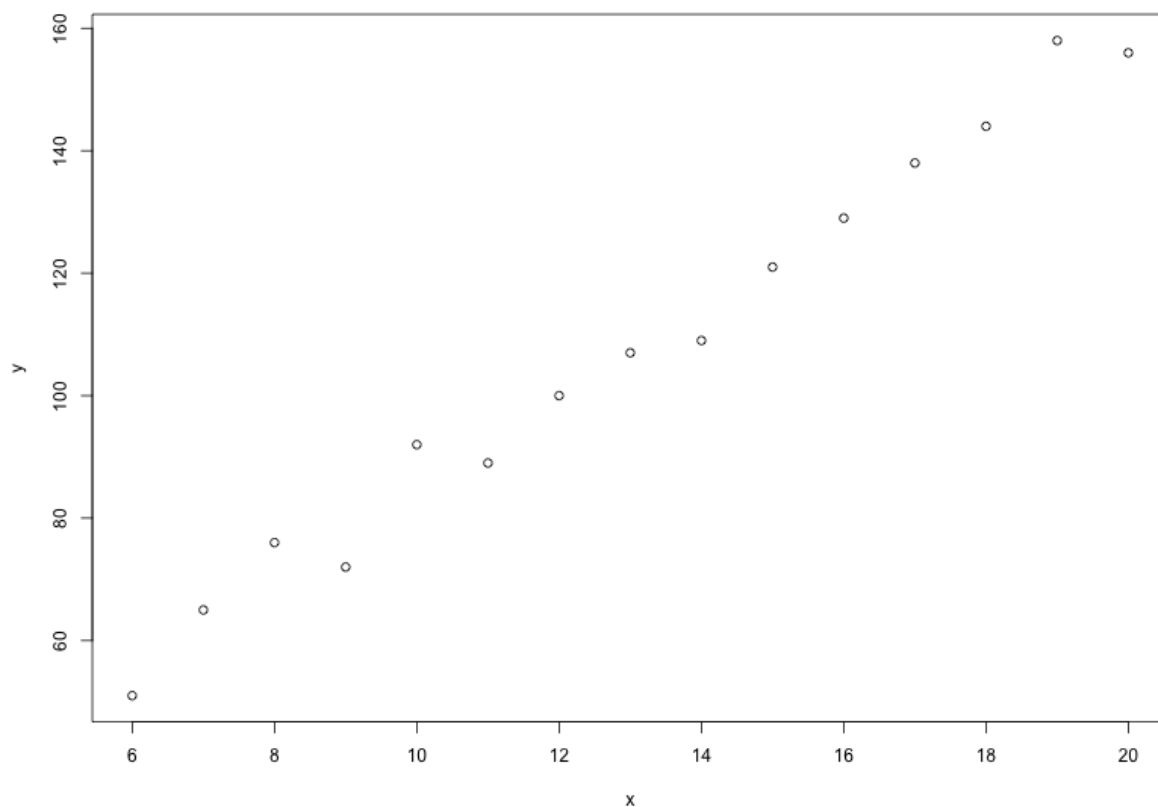
Esimerkki 6.1. Tutkitaan fiktiivistä aineistoa, jossa x on ajoneuvon tankkiin laitettun polttoaineen määrä litroissa ja y kertoo montako kilometria ajoneuvolla päästiin ennen polttoaineen loppumista. Havaitaan seuraava aineisto:

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

Nyt tuntuu hyvinkin luonnolliselta ajatella, että ajettujen kilometrien määrä riippuisi tankatun polttoaineen määrästä kutakuinkin lineaarisesti. Kuvasta 6.1 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 $\varepsilon \sim N(0, \sigma)$ on virhetermi. Estimoidaan nyt regressiosuoran vakio β_0 ja kulmakerroin β_1 R:llä.



Kuva 6.1: Ajettujen kilometrien ja tankatun polttoaineen yhteys esimerkissä

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

Call:

```
lm(formula = y ~ x)
```

Coefficients:

```
(Intercept)          x
      10.515         7.432
```

Esimerkki 6.2. Tarkastellaan edellisen esimerkin estimointitulosta. Saadut suurimman uskottavuuden estimaatit ovat $\hat{\beta}_0 = 10.515$ ja $\hat{\beta}_1 = 7.432$. Lisätään edellisessä esimerkissä piirrettyyn kuvaan punaisella regressiosuora käyttämällä kaavaa

$$y = 10.515 + 7.432x.$$

Tämän voi tehdä monella tavalla, mutta katsotaan nyt toteutus `lines()`-ja `curve()`-funktioilla:

```
# Viiva lines()-funktioilla
x <- c(0,50)
y <- 10.515 + 7.432*x
lines(x,y, col="red")
```

```
# Viiva curve() -funktioilla. Argumentti add=T lisää viivan
# olemassaolevaan kuvaan, eikä luo uutta ikkunaa.
curve(10.515 + 7.432*x, from=0, to=50, add=T, col="red")
```

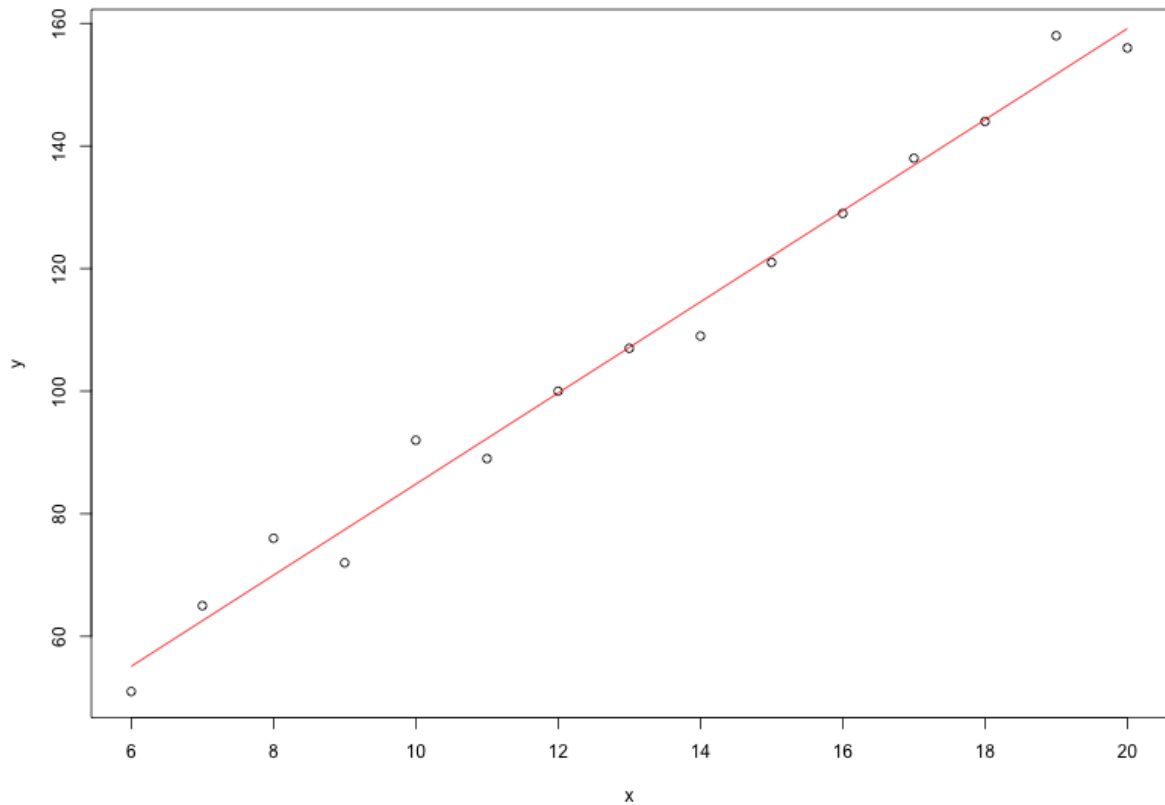
Esimerkki 6.3. Lisätään edellisissä esimerkeissä kasattuun kuvaan vielä ennustettujen arvojen luottamusvälit. Muodostetaan ensin muuttuja `x_values`, joka sisältää kaikki ne pisteet, joissa ennustetun arvon luottamusväli halutaan laskea. Tämän jälkeen 95% luottamusväli saadaan laskettua ja lisättyä kuvaan seuraavasti

```
> x_values <- seq(6,20, length.out=100)
> pred <- predict(fit, interval = "conf",
+ level = 0.95, newdata = list(x=x_values))
> lines(x=x_values, y=pred[,2], col="green")
> lines(x=x_values, y=pred[,3], col="green")
```

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

```
> summary(fit)
```

Call:



Kuva 6.2: Ajettujen kilometrien ja tankatun polttoaineen aineisto ja siihen sovitettu regressiosuora

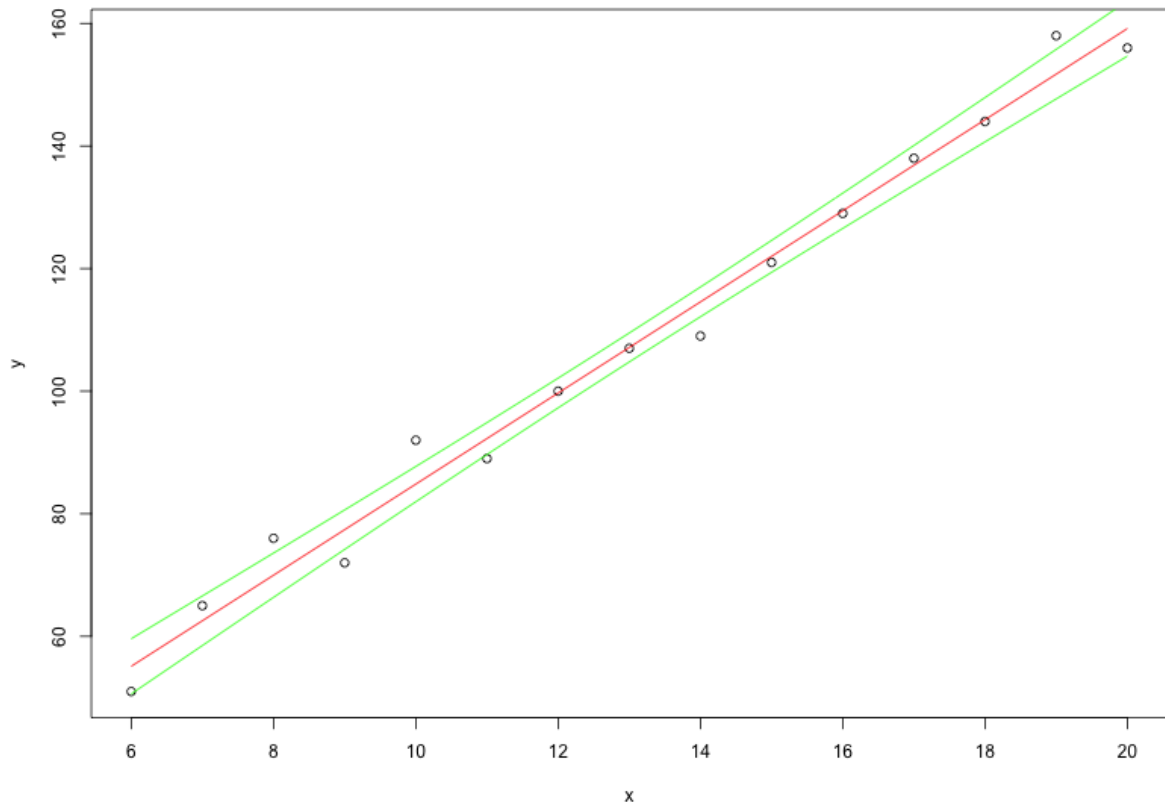
```
lm(formula = y ~ x)
```

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 ***



Kuva 6.3: Ajettujen kilometrien ja tankatun polttoaineen aineisto ja siihen sovitettu regressiosuora luottamusväleineen

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 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

Esimerkki 6.4. 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

Residuaaleja voidaan tarkastella esimerkiksi seuraavasti komennolla `plot(predict(fit), residuals(fit))`, joka piirtää mallin ennustamat selitettävän muuttujan arvot ja residuaalit vastakkain tavalliseen `plot()`-kuvaajaan. Mallioletuksen pätiessä residuaalien pitäisi olla likimain normaalijakautuneita, eikä residuaalikuviossa pitäisi olla nähtävissä mitään systemaattista vaihtelua.

6.2 Useamman selittäjän lineaarinen regressio

Esimerkki 6.5. 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.

```
> 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 ' ' 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

```

Voidaan havaita, että muuttujaa z vastaava kerroinparametri β_2 ei ole merkitsevästi poikkeava nolasta (p-arvo 0.850).

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 6.6.

```
> 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 6.7.

```

> 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 6.8.

```

> 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 6.9.

```
# 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 riville seuraavaan tapaan. Koska välissä ei ole rivinvaihtoa, R ei katkaise rakennetta ennen `else:`ä.

Esimerkki 6.10.

```
> 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 toteutetaan, viimeinen käsittelemä arvo. Tätä voidaan hyödyntää esimerkiksi valitsemalla suurempi luvuista `a` ja `b` ja sijoittamalla se muuttujaan `suurempi`.

Esimerkki 6.11.

```
> 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 toteutettavissa helpommin ja nopeammin R:n omien vektorisoitujen operaatioiden tai `apply`-perheen funktioiden avulla.

Aina kuitenkin tämä ei ole mahdollista, esimerkiksi monimutkaisempia simulaatioita voi olla hankala vektorisoida, ja ne voi olla helpompaa toteuttaa silmukoilla ja `if else`-valintarakenteilla.

6.4 Bayes-päätelyä

6.4.1 Doping-testi-esimerkki

(Bayes-päätelyn kurssin 2015 viikon 1 tehtävä 3) Merkitään satunnaismuuttujalla D sitä, käyttääkö urheilija ainetta, eli $D = 1$, jos urheilija on dopattu ja $D = 0$, jos urheilija on puhdas. Merkitään testin tulosta satunnaismuuttujalla T , eli $T = 1$, jos testitulokset on positiivinen, ja $T = 0$, jos testitulokset on negatiivinen. Testin sensitiivisyys, eli todennäköisyys että testitulokset on positiivinen jos urheilija käyttää ainetta, eli $P(T = 1|D = 1) = 0.98$. Testin spesifisyys, eli todennäköisyys että testitulokset on negatiivinen jos urheilija on puhdas, eli $P(T = 0|D = 0) = 0.95$.

Oletetaan että testattavista urheilijoista 1 % käyttää ainetta, eli $P(D = 1) = 0.01$. Havaitaan positiivinen testitulokset. Mikä nyt on todennäköisyys, että kyseinen positiivisen testitulokset saanut urheilija käyttää ainetta?

Tämä on helppo ratkaista kynällä ja paperilla (ks. JTP:n luku 10), mutta lasketaan harjoituksen vuoksi approksimaatio kyseiselle todennäköisyydelle simuloimalla. Käytetään simulaation otoskokona $n = 1000000$, eli simuloidaan miljoonan urheilijan otos, ja katsotaan mikä on käyttäjien osuus niistä urheilijoista jotka saavat positiivisen testitulokset.

Esimerkki 6.12.

```
> n <- 1000000
> doupatut <- rbinom(n=n, size=1, prob=0.01)
> positiiviset <- numeric(n)
> for(i in 1:n) {
  if(doupatut[i] == 1) {
    positiiviset[i] <- rbinom(n=1, size=1, prob=0.98)
  } else {
    positiiviset[i] <- rbinom(n=1, size=1, prob=1-0.95)
  }
}
> sum(doupatut * positiiviset) / sum(positiiviset)
[1] 0.1634877
```

Arvotaan ensin, onko urheilija dopattu vai ei, ja tallennetaan kunkin urheilijan doping-status vektoriin `doupatut` (1=käyttää, 0=ei käytä). Sen jälkeen arvotaan kunkin urheilijan testitulokset ja tallennetaan tulos vektoriin `positiiviset` (1=positiivinen, 0=negatiivinen): jos urheilija käyttää ainetta, positiivisen testitulokset todennäköisyys on 0.98, ja jos ei käytä, positiivisen testitulokset todennäköisyys on $1 - 0.95 = 0.05$.

Lopuksi vain lasketaan niiden urheilijoiden määrä, jotka sekä käyttävät ainetta että saivat positiivisen testituloksen (`doupatut * positiiviset` saa arvon 1 vain niille urheilijoille, joille sekä vektorin `doupatut` että `positiiviset` arvo on yksi, eli jotka sekä käyttävät ainetta että saavat positiivisen testituloksen) ja jaetaan se kaikkien positiivisen testituloksen saaneiden urheilijoiden määrällä.

Näin saadaan laskettua miljoonan kokoisesta simuloidusta otoksestamme käyttäjien osuus positiivisen testituloksen saaneista. Se on noin 0.163, eli 16.3% (voit tarkastaa miten lähellä tulos on tarkkaa arvoa ratkaisemalla laskun Bayesin kaavan avulla), mikä on yllättävän pieni ottaen huomioon testin hyvän tarkkuuden (sekä sensitiivisyys että spesifisyys ovat vähintään 0.95).

6.4.2 Doping-testi-esimerkki vektorisoituna

Edellä todettiin, että R:ssä `for`-silmukat ja `if else`-rakenteet voidaan monesti korvata R:n omilla vektorisoiduilla operaatioilla. Niin myös tässä tapauksessa. Seuraavassa täsmälleen sama simulaatio on toteutettu hieman erilaisella ja ehkä R:lle ominaisemmalla tavalla.

Esimerkki 6.13.

```
> n <- 1000000
> doupatut <- rbinom(n=n, size=1, prob=0.01)
> positiiviset <- numeric(n)
> n_doupatut <- sum(doupatut)
> positiiviset[which(doupatut == 1)] <- rbinom(n=n_doupatut, size=1, prob=0.98)
> positiiviset[which(doupatut == 0)] <- rbinom(n=n-n_doupatut, size=1, prob=1-0.95)
> sum(doupatut * positiiviset) / sum(positiiviset)
[1] 0.1668753
```

Koodin alku ja loppu ovat täsmälleen samoja kuin aiemmin, mutta `for`-silmukka on korvattu suorilla sijoituksilla vektoriin. Huomaa, että sekä vektori johon sijoitetaan ja sijoitettava vektori ovat samanpituisia, minkä takia sijoitus toimii:

Esimerkki 6.14.

```
> length(positiiviset[which(doupatut == 1)])
[1] 10064
> length(rbinom(n=n_doupatut, size=1, prob=0.98))
[1] 10064
```