Tilastolliset Ohjelmistot: R

Anton Klåvus (2020), Juho Kopra ja Santtu Tikka (2021–2024)

2024-08-14

Sisällysluettelo

Johdanto

Tämä materiaali on suunniteltu käytettäväksi Jyväskylän yliopiston kurssilla Tilastolliset Ohjelmistot sekä Itä-Suomen yliopiston R-kurssilla. Materiaali toimii R-ohjelmoinnin harjoittelun tukena.

Anton Klåvusin vuonna 2020 kirjoittamaa ansiokasta materiaalia on kehitetty lukukauden 2021-22 R-kielen kurssia ajatellen. Materiaalia on täydennetty tarvittavin osin ja amalla materiaali on muunnettu verkkokirjamuotoon. Tämä opiskelumateriaali on luotu Quarto-julkaisujärjestelmän avulla . Kirjaa voi lukea verkkoselaimella ja se toimii myös puhelimella.

Lunttilappu

Tämän materiaalin ohessa kannattaa käyttää apuna nk. Cheat Sheetiä eli "lunttilappua". Lunttilapusta on helppo tarkastaa miten jokin jo oppimasi asia tehdään R:ssä, jos et vielä muista kunnolla kyseistä asiaa. Internetistä löytyy Cheat Sheetejä useisiin R-paketteihin ja muihin kokonaisuuksiin, mutta tässä käytetään Base R Cheat Sheetiä. Lataa Base R Cheat Sheet itsellesi painamalla tästä.

Verkkolähteitä

Tämä materiaali on tarkoitettu riittäväksi materiaaliksi kurssille. Tässä kuitenkin joitakin verkosta löytyviä lähteitä, joista voi olla apua.

- Tutorialspoint Soveltuu R:n opiskeluun englannin kielellä, jos osaa entuudestaan jo vähän ohjelmoida.
- R for Data Science Laaja verkkokirja R-ohjelmointiin datatieteen näkökulmasta.

Alkuvalmistelut

RStudio

RStudio on ohjelmointiympäristö eli IDE (Integrated Development Environment), joka tekee ohjelmoinnista huomattavasti mukavampaa. RStudio on saatavilla useille käyttöjärjestelmille

ja se on ilmainen ohjelma. Tässä kirjassa oletetaan, että käytössä on RStudio, mutta muutkin ympäristöt ovat sopivia.

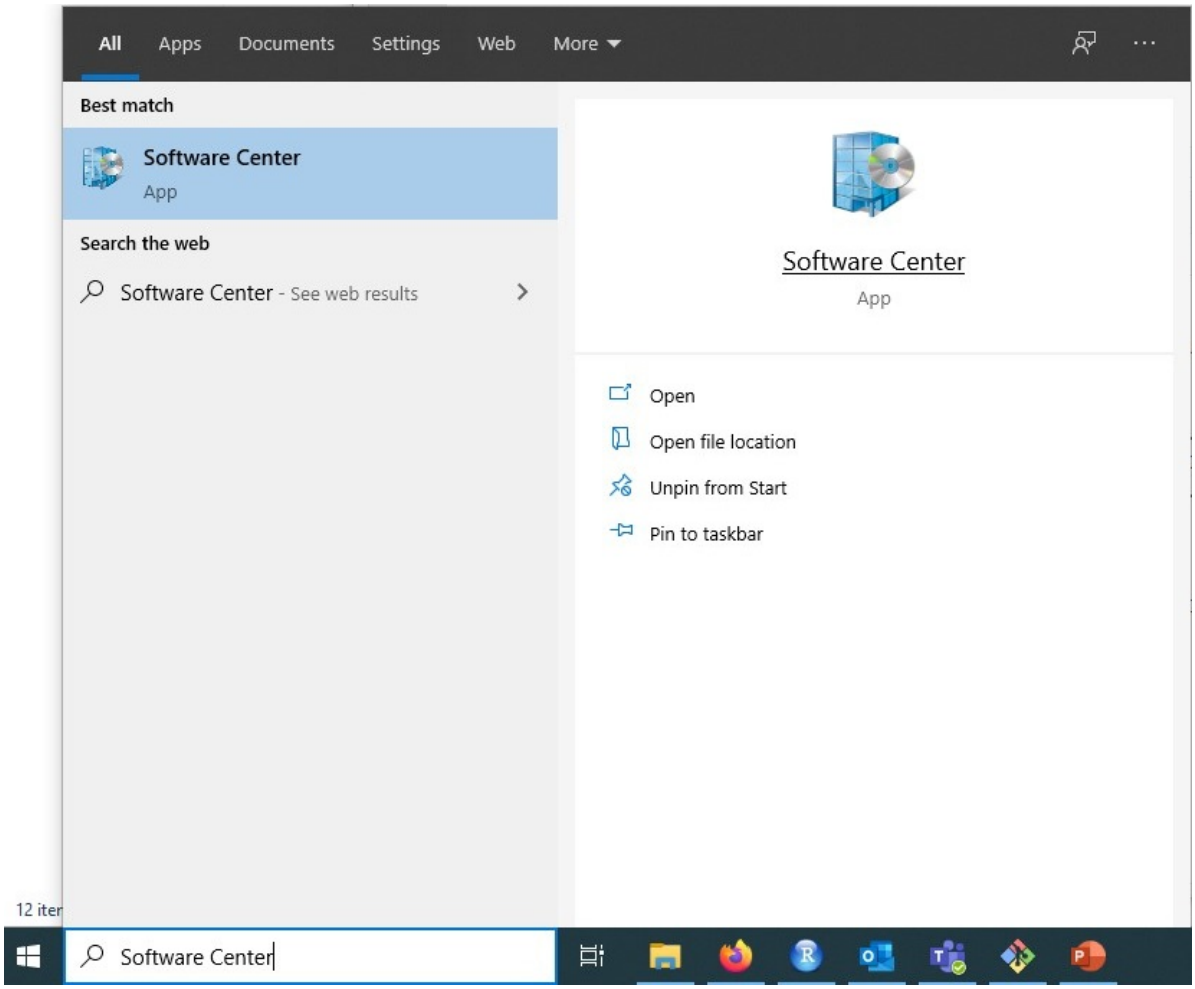
R:n ja RStudion asentaminen omalle tietokoneelle

Mene seuraavalle sivulle, josta asennat ensin R:n (1. vaihe) ja sitten RStudio Desktop omalle käyttöjärjestelmällesi. Ellet tiedä käyttöjärjestelmääsi, on se luultavimmin Windows 10.

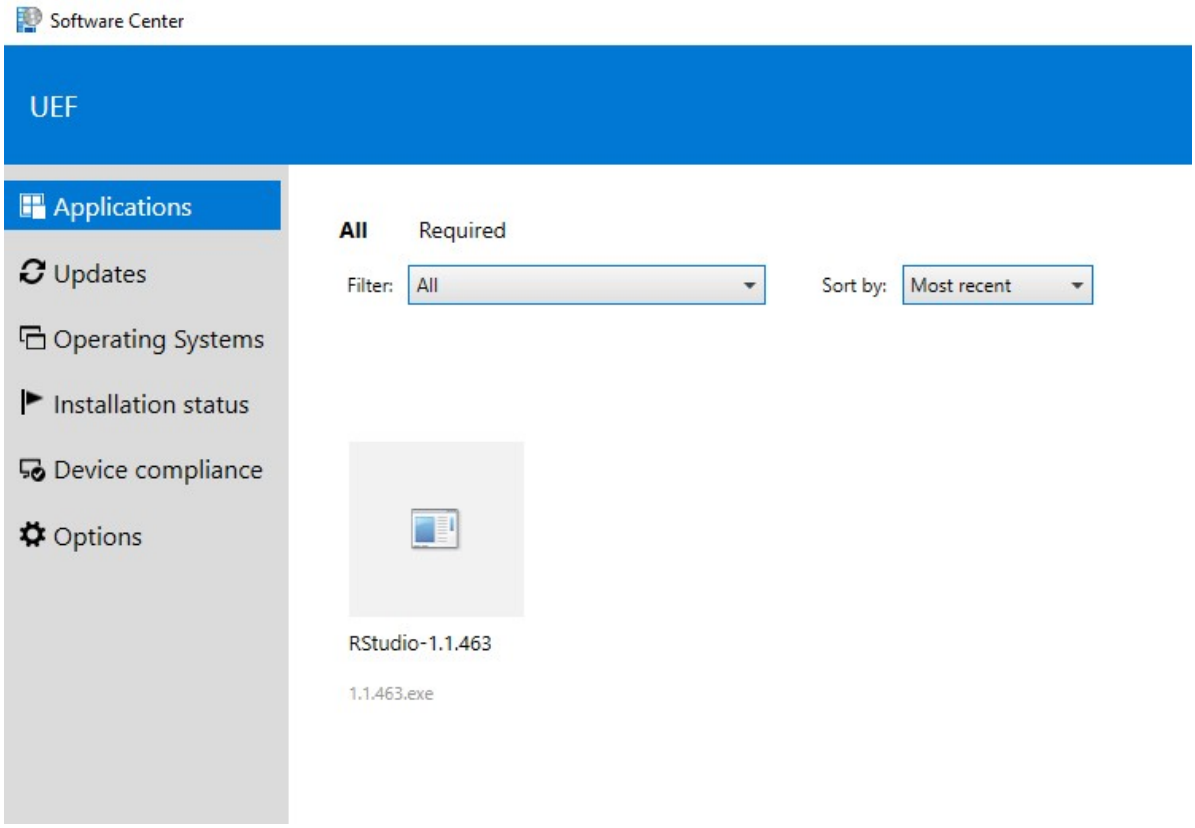
https://www.rstudio.com/products/rstudio/download/#download (avautuu uuteen ikkunaan)

RStudion asennus yliopiston koneelle

Mikäli et halua käyttää omaa tietokonettasi kurssin suoritamiseen, niin RStudion saa asennettua yliopiston koneille Sofware Centerin kautta. Software Center löytyy Windowsin omalla haulla.

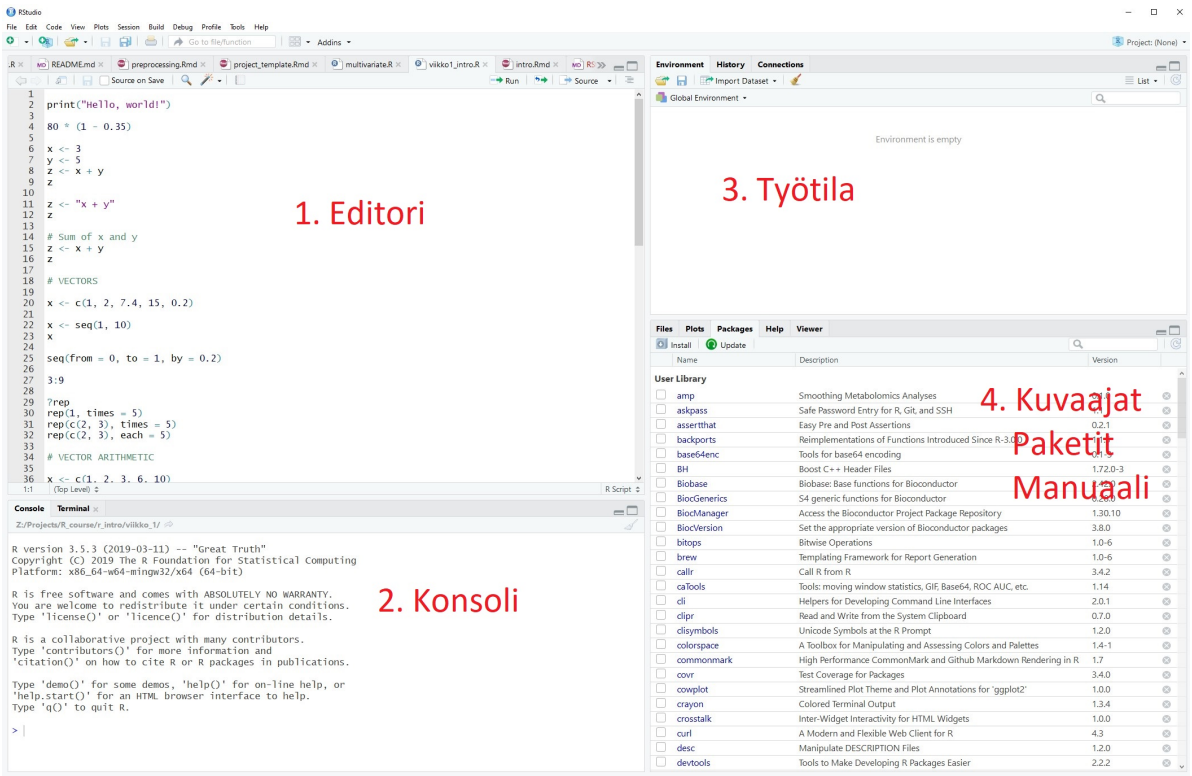


RStudio:n voi asentaa Software Centeristä, ja RStudion pitäisi sen jälkeen olla käytettävissä. Tyypillisesti R ja RStudio ovat kuitenkin jo valmiiksi asennettuna



RStudion käyttö

RStudion näkymässä on neljä osaa:



1. Editori.

Editorilla kirjoitetaan R-koodia sisältäviä tiedostoja, eli R-skriptejä. Uuden skriptin saa auki painamalla File -> New File -> R Script (tai Ctrl + Shift + N). Skripteihin tutustutaan myöhemmin kurssilla, mutta ne ovat yksinkertaisuudessaan kokoelma R-komentoja, jotka yhdessä tekevät jotain, esimerkiksi analysoivat jonkin tutkimusprojektin datan tai piirtävät valmiista tuloksista kuvaajia.

Editoriin kirjoitettua koodia voi ajaa rivi kerrallaan painamalla rivin kohdalla Ctrl + Enter. Useamman rivin voi myös maalata ja suorittaa kerrallaan. Yläreunassa oleva “Source”-nappi ajaa kaiken nykyisen tiedoston koodin.

R-skriptejä voi tallentaa ihan kuin muitakin tiedostoja. R-skpriptien tiedostopääte on .R. Kaikki harjoitustehtävissä ja loppukokeessa käyttämäsi koodi on syytä kirjoittaa skripteihin. Kun tehtävät tallentaa tällä tavalla, voi ensi kerralla vain yksinkertaisesti ajaa skriptin haluamaansa tehtävään asti.

2. Konsoli.

Konsolissa "ajetaan" eli suoritetaan R-komentoja. Jos editoriin kirjoitettua koodia ajetaan, RStudio ajaa komennot automaattisesti konsolissa. Konsolissa pelkkä Enter riittää koodirivin suorittamiseen. Voit kokeilla kirjoittaa konsoliin jonkun laskutoimituksen, kuten `2 * 3` ja painaa Enter, jolloin tuloksen pitäisi tulostua konsoliin. Voit myös kokeilla kirjoittaa laskuja editoriin, ja painaa Ctrl + Enter, jolloin pitäisi tapahtua sama asia. Konsoliin tulostuvat myös mahdolliset viestit, varoitukset ja virheilmoitukset.

Suurin ero konsolin ja editorin välillä on se, että **konsoliin kirjoitetut komennot eivät tallennu mihinkään tiedostoon**. Jos siis haluat säilyttää koodisi, se tulee kirjoittaa editoriin ja tallentaa .R-tiedostoon. Saman istunnon aikana tehtyjä komentoja voi konsolissa selata ylös- ja alas-nuolila.

Moodlen ohjeissa ja videoissa käytetään R:ää puhtaasta R-konsolista. Voit siis kuvitella, että kurssin videoissa näkyy vain RStudion tämä osa, ja muut osat ovat vain helpottamassa työtäsi.

3. Työtila

Työtilassa näkyvät R-istunnon aikana luodut muuttujat.

4. Tiedostot / Kuvaajat / Paketit / Manuaali

Tässä osassa on monta käytännöllistä välilehteä:

- Files: Näyttää käyttöjärjestelmän hakemistorakenteen, oletusarvoitesti työhakemiston.
- Plots: Tänne ilmestyvät R:llä piirretyt kuvaajat.
- Packages: Täältä voi hallita asennettuja paketteja (alla ohjeet tällä kurssilla tarvittavien pakettien asennukseen).
- Help: Täällä voi selata R:n manuaalia, jossa on ohjeet jokaiselle R-komennolla. Voit kokeilla ajaa editorissa tai konsolissa komennon `?print`, joka avaa `print`-funktion ohjesivun.

Rcourse-paketin asentaminen

R-paketit

R-ohjelmoinnissa asennetaan usein R-paketteja. Paketit ovat kokonaisuuksia, jotka lisäävät R:ään ominaisuuksia. Esimerkiksi tällä kurssille tarvittava paketti `Rcourse` sisältää

harjoitustehtäviä kurssin aihepiireistä sekä loppukokeen, jonka perusteella kurssin suoritus arvioidaan.

Asentaminen

Rcourse-paketti asennetaan suorittamalla seuraava koodi R:ssä. Kopioi koodi joko R-skriptiin ja aja se tai kopioi se suoraan Console-ikkunaan ja paina Enter-näppäintä.

```
install.packages("remotes")
remotes::install_github("santikka/R-course")
```

Tämän jälkeen paketti tulee ottaa käyttöön

```
library("Rcourse")
```

Komento `info()` tulostaa paketin ohjeet (ensimmäisellä käyttökerralla kieli on englanti). Voit vaihtaa kielen suomeksi näin:

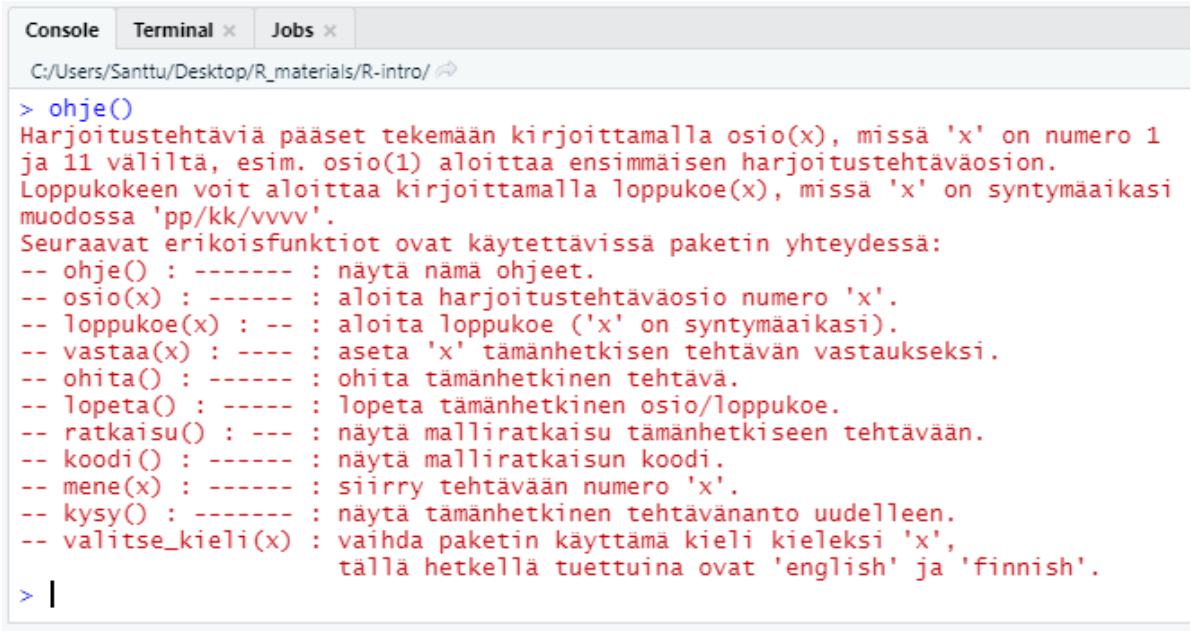
```
select_language("finnish")
```

Jos haluat, että kielivalinta säilyy R-istunnosta toiseen, tulee asettaa seuraava argumentti:

```
select_language("finnish", save_selection = TRUE)
```

Huomaa, että kielen vaihtuessa myös joidenkin pakettiin liittyvien funktioiden nimet vaihtuvat. Tarkastele vielä suomenkielisiä komentoja:

```
ohje()
```



Aloita sitten osion 1 harjoitustehtävien suorittaminen komennolla

```
osio(1)
```

Kun olet suorittanut harjoitusosion 1, voit jatkaa seuraavaan osioon. Osiot 1-7 ovat pakollisia (tentit kysyvät näiden osioiden sisältöjä) ja osiot 8-11 ovat lisämateriaalia kiinnostuneille (ei kysytä tentissä).

Opiskelu ja tenttiminen Rcourse-paketin avulla

Kurssin harjoitustehtävät suoritetaan käyttäen Rcourse-pakettia, eli 1. osion voi aloittaa komennolla

```
osio(1)
```

Lisäksi tenttiminen onnistuu vastaavasti funktiolla `loppukoe(x)`, mutta tällöin merkin `x` tilalle on annettava oma syntymäaika muodossa “dd/mm/yyyy”. Esim. henkilö joka on syntynyt 1. tammikuuta 1990 antaisi

```
loppukoe("01/01/1990")
```

Huomaa, että loppukokeen kysymykset vaihtuvat joka suorituskerralla. Lisää tietoa loppukokeesta löydät kurssin verkkosivulta.

Tehtävien tallentaminen skripteihin RStudiolla

Suurin osa kurssin tehtävistä on melko lyhyitä, joten ne voi tarvittaessa tehdä suoraan konsoliin. On kuitenkin suositeltavaa kirjoittaa varsinkin pidemmät ja monimutkaisemmat tehtävät muistiin skriptitiedostoon. Jokaista osiota varten kannattaa tehdä erillinen R-skripti, joka sisältää tehtävien tarvitseman koodin sekä palautuskomennot. Tällainen skripti näyttää jotakuinkin tältä:

```
# Teht 1
vast <- 1
vastaa(vast)

# Teht 2
vast <- c(1, 2, 3)
vastaa(vast)

# Teht 3
vast <- "jotain"
vastaa(vast)
```

Mikäli käytät nimen `vast` sijasta jotain muuta nimeä, niin sinun on käytettävä samaa nimä myös vastaa-funktion argumenttina! Huomaa, että tehtäviin vastataan aina syöttämällä R-objekti, paitsi kuvien piirtämistä käsittelevässä osiossa.

1 R

1.1 Mikä R on ja mitä sillä tehdään?

R on tehty ensisijaisesti tilastotiedettä ja data-analyysiä varten. R:llä kirjoitetaan yleensä lyhyitä ohjelmia, joita kutsutaan skripteiksi. R:llä ei siis ole tarkoitus kehittää esimerkiksi pelejä, tai muita ohjelmia joissa on graafinen käyttöliittymä, kuten vaikkapa Photoshop. R ei myöskään ole web-ohjelmointiin tarkoitettu kieli (vaikka oikeilla paketeilla R:lläkin pystyy tekemään web-sovelluksia).

R on korkean tason ohjelmointikieli. Tämä tarkoittaa sitä, että R:ssä on paljon valmiita komentoja, joiden "alta" löytyy paljon lisää koodia, johon R-ohjelmoijan ei kuitenkaan tarvitse itse koskea. Esimerkiksi tilastollinen t-testi vaatii useita matemaattisia välivaiheita, mutta R-ohjelmoija voi suorittaa testin yhdellä komennolla (`t.test`), joka antaa kaikki tarvittavat tiedot testistä.

R:n käyttöä ja ohjelmointia oppii parhaiten tekemällä. Tässä dokumentaatiossa on tekstin väliin upotettu R-koodia harmaissa laatikoissa, kuten alla olevassa esimerkissä. Koodin alla esiintyy usein myös koodin ajamisen aiheuttamia tulosteita (output) tekstin seassa. Otetaan ensimmäiseksi esimerkiksi klassinen "Hello, world!"-komento:

```
print("Hello, world!")
```

```
[1] "Hello, world!"
```

`print`-funktio tulostaa sille annetun tekstin konsoliin ja se on kätevä funktio mm. ohjelman toiminnan testaamiseen ja pidemmän ohjelman etenemisen seurantaan.

R:ää voi käyttää myös laskimen sijaan. Alla olevassa esimerkissä lasketaan kuinka paljon jää hintaa 80 euron hintaiselle tuotteelle 35% alennuksen jälkeen.

```
80 * (1 - 0.35)
```

```
[1] 52
```

Yksittäisten komentojen ajamisesta ei kuitenkaan ole yleensä hyötyä, ellei tuloksia voi tallentaa johonkin. Ohjelmointikielissä tietoja tallennetaan muuttujiin, joita käsitellään seuraavaksi.

2 Muuttujat ja vektorit

2.1 Muuttujat

Muuttujat (*variables*) ovat yksi tärkeimmistä ohjelmointikielien rakenteista. Muuttujien tehtävä on säilyttää tietoa ja tuloksia edellisistä laskutoimituksista. Alla on yksinkertainen esimerkki muuttujien käytöstä R:ssä.

```
x <- 3
y <- 5
z <- x + y
z
```

```
[1] 8
```

Edellisessä esimerkissä **sijoitetaan** (*assign*) eli tallennetaan muuttujaan `x` arvo 3 ja muuttujaan `y` arvo 5. Sen jälkeen muuttujien `x` ja `y` summa sijoitetaan muuttujaan `z`, jonka jälkeen tulostetaan muuttujan `z` arvo. Symboli `<-` on R:n **sijoitusoperaattori** (*assignment operator*) (myös yhtä kuin-merkki `=` toimii melkein aina, mutta `<-` merkin käyttöä suositellaan vahvasti). Sijoitusoperaattori kertoo R:lle, että symbolin `<-` vasemmalle puolelle sijoitetaan sen oikean puolen laskutoimituksen tulos. Vasen puoli määrittää muuttujan nimen

Mutta miten muuttujan `z` arvo tulostui konsoliin, vaikka koodissa ei käytetty funktiota `print`? R:n erikoisominaisuus moneen muuhun ohjelmointikieleen verrattuna on se, että print-käskyä ei tarvitse aina kirjoittaa, vaan pelkästään muuttujan (tai laskutoimituksen) kirjoittaminen tulostaa arvon konsoliin, kuten alla oleva koodi havainnollistaa:

```
z
```

```
[1] 8
```

```
print(z)
```

```
[1] 8
```

```
x + y
```

```
[1] 8
```

```
print(x + y)
```

```
[1] 8
```

```
3 + 5
```

```
[1] 8
```

```
print(3 + 5)
```

```
[1] 8
```

Muuttujiin voi sijoittaa muutakin kuin yksittäisiä lukuja, kuten merkkijonoja (strings), vektoreita, tai paljon monimutkaisempiakin rakenteita.

```
x <- "Hello world"
x
```

```
[1] "Hello world"
```

2.2 Kommentit

Myöhemmin vastaan tulevassa koodissa käytetään kommentteja. Kommentit ovat koodin oheen kirjoitettua tekstiä, joka ei ole ohjelmointikieltä, ja joka ohitetaan koodia ajettaessa. Kommenttien tarkoitus on kuvailla koodin toimintaa. Oman koodin kommentointia on hyvä harjoitella alusta lähtien, vaikka ensimmäisten tehtävien koodi onkin hyvin yksinkertaista. R:ssä kommentit merkataan #-symbolilla. Edellinen esimerkki kommentoituna voisi näyttää jotakuinkin tältä:

```
# Assign arbitrary numbers to two variables
x <- 3
y <- 5
# Sum of two variables
z <- x + y
# Print the results
z
```

```
[1] 8
```

2.3 Vektorit

Nyt kun muuttujat ovat tuttuja, voimme siirtyä käsittelemään vektoreita (*vector*). R:n vektorit ovat yksinkertaisia järjestettyjä tietorakenteita, jotka koostuvat alkioista (*elements*), esimerkiksi reaaliluvuista. Alla oleva esimerkki sijoittaa muuttujaan `x` vektorin, joka sisältää 5 lukua ja tulostaa vektorin `x` sisällön konsoliin.

```
x <- c(1, 2, 7.4, 15, 0.2)
x
```

```
[1]  1.0  2.0  7.4 15.0  0.2
```

Yksinkertaisin tapa tehdä vektori R:ssä on käyttää `c`-funktiota (`c` tulee sanasta *combine*), joka luo vektorin, joka sisältää sille annetut arvot annetussa järjestyksessä. Monet R-kielen komennot ja funktiot luovat vektoreita, alla muutama esimerkki:

```
# Sequence from 1 to 10
seq(1, 10)
```

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

```
# Sequence from 0 to 1 with 0.2 intervals
seq(0, 1, by = 0.2)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

```
# A sequence of length 6 (starting from 1 with an interval of 1)
seq_len(6)
```

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

```
# A sequence from 3 to 9 with an interval of 1
3:9
```

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

```r
# Repeat the number 1 five times
rep(1, 5)
```

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

```r
# Repeat vector c(1, 2) 3 times
rep(c(1, 2), 3)
```

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

```r
# Repeat all values in vector c(1, 2, 3) 3 times
rep(c(1, 2, 3), 3)
```

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

Erityisesti usein hyödyllisiä funktiota ovat `seq`, jolla voidaan luoda lukujonoja halutulla tiheydellä, sekä `rep`, joka toistaa sille annettun luvun tai vektorin halutun monta kertaa.

2.3.1 Vektorilaskentaa

Vektoreilla laskeminen on usein hyvin intuitiivista (lisää vaaranpaikoista myöhemmin). Kun vektoriin kohdistetaan laskutoimintoja, sama operaatio tehdään kaikille vektorin alkioille. Kyseessä on ns. vektorisaatio (*vectorization*).

```r
x <- c(1, 2, 3, 6, 10)
x * 2
```

```
[1]  2  4  6 12 20
```

```r
x / 2 + 1
```

```
[1] 1.5 2.0 2.5 4.0 6.0
```

Entä jos vektoreita lisää toisiinsa, tai kertoo keskenään? Jos vektorit ovat samanpituisia, operaatio toteutetaan alkio kerrallaan. Jos vektorit ovat eripituisia, R yrittää kierrättää (*recycle*) lyhyempää vektoria niin, että siitä tulee yhtä pitkä kuin pidempi vektori. Tämän jälkeen operaatio suoritetaan alkio kerrallaan (itse asiassa näin tapahtui myös aiemmissa esimerkeissä, kun vektori kerrottiin yksittäisellä luvulla. R:ssä yksittäiset luvut ovat vektoreita, joiden pituus on 1). Jos kierrätys ei onnistu, eli pidemmän vektorin pituus ei ole jaollinen lyhyemmän pituudella, R antaa virheilmoituksen.

```
x <- c(1, 2, 3, 6, 10, 2)
y <- c(1, 1, 1, 3, 3, 3) # or rep(c(1, 3), each = 3)
z <- c(2, 4)

x + y # Element-wise sum
```

```
[1]  2  3  4  9 13  5
```

```
x * y # Element-wise multiplication
```

```
[1]  1  2  3 18 30  6
```

```
x + z
```

```
[1]  3  6  5 10 12  6
```

R:ssä on myös paljon funktioita, joilla voi laskea vektoreista erilaisia tunnuslukuja, kuten keskiarvon, mediaanin, keskihajonnan, pituuden, ym.

```
x <- c(1, 2, 3, 6, 10, 2)
# Sample mean (average)
mean(x)
```

```
[1] 4
```

```
# Standard deviation
sd(x)
```

```
[1] 3.405877
```

```r
# Sum
sum(x)
```

```
[1] 24
```

```r
# length
length(x)
```

```
[1] 6
```

2.3.2 Ei-numeeriset vektorit

2.3.2.1 Merkkijonovektorit

Vektorien ei ole pakko sisältää lukuja. Vektorit voivat sisältää esimerkiksi merkkijonoja, kuten alussa nähty "Hello, world!". Merkkijonotyypin nimi R:ssä on `character`.

```r
x <- c("Hello, world!", "R is the best", "I", "like", "programming", "!")
x
```

```
[1] "Hello, world!" "R is the best" "I"             "like"
[5] "programming"   "!"
```

Merkkijonovektoreiden muokkausta varten on omia funktiota, tärkeimpinä `paste` ja `paste0`, jotka yhdistävät merkkijonoja toisiinsa. Myös numeerisia vektoreita voi antaa näille funktioille, ja ne muutetaan merkkijonoiksi.

```r
first_names <- c("Diana", "Peter", "Bruce")
last_names <- c("Prince", "Parker", "Wayne")
paste(first_names, last_names)
```

```
[1] "Diana Prince" "Peter Parker" "Bruce Wayne"
```

```r
students <- paste0("Student_", 1:5)
```

2.3.2.2 Loogiset vektorit

Kolmas yleinen vektorityyppi on looginen vektori, joka sisältää arvoja `TRUE` eli tosi tai `FALSE` eli epätosi. Loogisia vektoreita käytetään yleensä joko merkitsemään binäärisiä muuttuja (esimerkiksi paastosiko koehenkilö ennen näytteenottoa) tai vektorien ja matriisien indeksoinnissa (tästä lisää pian). Tyypillinen käyttötarkoitus loogisille vektoreille on poimia aineistosta havainnot, jotka täyttävät tietyt ehdot. Tällöin loogisia vektoreita syntyy erilaisten loogisten operaattorien avulla:

```
x <- c(1, 2, 3, 6, 10, 2)

x > 3 # Is the element of x greater than 3?
```

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

```
x >= 3 # Greater or equal to three=
```

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

```
x == 6 # Equal to 6?
```

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

```
x != 2 # Not equal to 2?
```

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

2.3.2.3 Loogiset vektorit ja matematiikka

Jos loogiselle vektorille tekee operaation, joka odottaa numeerista vektoria, R muuttaa automaattisesti arvot `TRUE` ykkösiksi ja arvot `FALSE` nolliksi. Tämä on erityisen hyödyllistä käytettäessä funktiota `sum`. Tällä tavalla saadaan helposti tietää esim. kuinka moni vektorin alkio täyttää tietyn ehdon:

```
x <- c(1, 3, 5, 2, 19)
above_3 <- x > 3

# Logical vector automatically converted to numeric
x + 1
```

```
[1]  2  4  6  3 20
```

```r
# how many elements of x are smaller than 10?
sum(x < 10)
```

```
[1] 4
```

2.3.3 Vektorien indeksointi ja osajoukon valinta

Usein vektorista halutaan poimia vain tietyt arvot, esimerkiksi vain ensimmäiset 5 arvoa, tai vain arvot, jotka täyttävät tietyt ehdot. R:ssä vektorin indeksointiin käytetään hakasulkeita `[]`. Yleisin indeksointitapa on antaa hakasulkeiden sisään vektori kokonaislukuja, jotka vastaavat niiden alkioiden järjestyslukuja, jotka vektorista halutaan poimia (HUOM R:ssä indeksointi alkaa ykkösestä, ei nollasta!). Toinen vaihtoehto on käyttää loogista vektoria, jolloin vektorista poimitaan ne alkiot, joiden kohdalla loogisen vektorin arvo on `TRUE`. Tämä on yksinkertaisempaa kuin miltä se kuulostaa:

```r
x <- c(1, 2, 3, 6, 10, 2)

# Picking exact elements
x[2:3] # Second and third values
```

```
[1] 2 3
```

```r
x[c(4, 5, 1)] # Note that the order does not have to be increasing
```

```
[1]  6 10  1
```

```r
# Using logical vector as condition
x[x > 3]
```

```
[1]  6 10
```

```r
# The condition can be based on another vector
characters <- c("Yoda", "C-3PO", "Rey", "R2-D2", "Anakin", "Baby Yoda")
heights <- c(66, 175, 170, 109, 183, 40.5)
# Only characters shorter than 120 cm
characters[heights < 120]
```

```
[1] "Yoda"      "R2-D2"     "Baby Yoda"
```

2.3.4 Puuttuvat arvot

Monessa tutkimusprojektissa törmätään syystä tai toisesta jossain vaiheessa puuttuviin arvoihin. Hyvä esimerkki ovat seurantatutkimukset, jossa usein seurannan lopussa on jäljellä vähemmän koehenkilöitä kuin alussa.

Puuttuvia arvoja merkitään R:ssä symbolilla `NA` (not available). Puuttuvat arvot noudattavat yksinkertaista logiikkaa: mikä tahansa operaatio `NA`:lle antaa tulokseksi `NA`. Funktiot, jotka operoivat vektoreilla, kuten `sum` tai `mean` voidaan erikseen asettaa poistamaan puuttuvat arvot ennen summan, keskiarvon tms. laskemista.

```
missing <- c(1, 2, NA, 4, NA, 6)
full <- seq(1, 6)

# Addition with NA returns NA
missing + full
```

```
[1]  2  4 NA  8 NA 12
```

```
# Sum of vector with NAs returns NA
sum(missing)
```

```
[1] NA
```

```
# Removing NAs before summation
sum(missing, na.rm = TRUE)
```

```
[1] 13
```

HUOM! Funktio `is.na` tarkistaa, onko jokin arvo puuttuva sille annetussa vektorissa. Perinteinen yhtäsuuruuden testaaminen ei siis toimi. Funktio `complete.cases` muistuttaa `is.na` funktiota, mutta sitä voidaan käyttää myös kokonaisille aineistoille, jolloin se palauttaa totuusarvon `TRUE` niiden rivien kohdalla, jotka eivät sisällä lainkaan puuttuvaa tietoa yhdessäkään muuttujassa.

```
# Just returns NA
NA == NA
```

```
[1] NA
```

```r
# Returns a logical value as expected
is.na(NA)
```

```
[1] TRUE
```

```r
is.na(1)
```

```
[1] FALSE
```

```r
# is.na operates element-wise on a vector
missing <- c(1, 2, NA, 4, NA, 6)
is.na(missing)
```

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

```r
# complete.cases gives the data elements which do not have missing data.
# It can be used with data frames also.
complete.cases(missing)
```

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

2.4 Extra: Alkeistietotyypit ja erikoisarvot

Tässä lisätieto-osiossa käsitellään asioita, joita et välttämättä tarvitse kurssista suoriutuaksesi. Mikäli käytät R:ää enemmän, niin vastaan tulee enemmin tai myöhemmin ongelmia, joissa tarvitsee näitä taitoja. Voit palata näihin myöhemmin koska tahansa, kun haluat syventää ymmärrystäsi R:stä.

R on rakennettu sisäisesti siten, että vektorin kukin elementti on jotain alkeistietotyyppiä. R:ssä on valmiina kuusi alkeistietotyyppiä:

- looginen (`logical`)
- kokonaisluku (`integer`)
- numeerinen (eli reaaliluku, `numeric`)
- kompleksiluku (`complex`)
- merkkijono (`character`)
- bitti (`raw`)

Näistä tarpeellisimpia ovat numeerinen, looginen, ja merkkijono. Kompleksilukua tarvitsee vain joissain erityistapauksissa ja kokonaisluvut ovat nykyään lähes aina tallennettu numeerisina. Bittejä, eli `raw`-tyypin vektoreita käytetään harvoin.

```
# Integers are usually stored as reals
x <- c(1, 2, 3)
class(x)
```

```
[1] "numeric"
```

```
# You can create integers by adding capital L behind the number
x_int <- c(1L, 2L, 3L)
class(x_int)
```

```
[1] "integer"
```

```
# Character strings (or just strings)
x_char <- c("I", "have", "a", "cat.")
class(x_char)
```

```
[1] "character"
```

```
# Complex numbers
x_comp <- c(0i, 2 + 1i, 1 - 3i)
class(x_comp)
```

```
[1] "complex"
```

```
# Logical vector
x_logi <- c(TRUE, FALSE)
class(x_logi)
```

```
[1] "logical"
```

```
# Raw vector
x_raw <- as.raw(c(0, 1, 2))
class(x_raw)
```

```
[1] "raw"
```

Joskus vektorin tiedot ovat väärässä muodossa, esim. merkkijonoina, mutta niitä haluttaisiin käsitellä numeroarvoina. Näihin operaatioihin on omat funktionsa. Tällöin voi kuitenkin tulla ongelmia, jos muutettava vektori ei ole helposti muutettavissa haluttuun muotoon.

```
# Chacter string to numeric
class(as.numeric(x_char))
```

```
Warning: NAs introduced by coercion
```

```
[1] "numeric"
```

```
# If character contains only values it is easy
x_char2 <- c("0", "5", "6.5")
as.numeric(x_char2)
```

```
[1] 0.0 5.0 6.5
```

```
# Integer to numeric
x_int_to_num <- as.numeric(x_int)
x_int_to_num
```

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

```
class(x_int_to_num)
```

```
[1] "numeric"
```

```
# Numeric to integer
x_num_to_int <- as.integer(x)
x_num_to_int
```

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

```
class(x_num_to_int)
```

```
[1] "integer"
```

Vielä pari lisähuomiota puuttuvista arvoista (näitä ei tarvita usein) koskien niiden tietotyyppejä. Puuttuvalla arvolla on myös alkeistietotyyppi. Mikäli `NA`-arvon tietotyyppi tulee määritellä, niin sen voi tehdä seuraavasti. Jos luodaan muuttuja tai vektori, jossa on vain yksi arvo, joka on `NA`, niin se oletusarvoisesti looginen.

```
# Specify a numeric NA value
NA_real_
```

```
[1] NA
```

```
# Specify a complex number NA value
NA_complex_
```

```
[1] NA
```

```
# Specify a integer number NA value
NA_integer_
```

```
[1] NA
```

```
# Specify a character NA value
NA_character_
```

```
[1] NA
```

```
# NA gives a logical type when evaluated alone
class(NA)
```

```
[1] "logical"
```

```
# NA_real_ is numeric
class(NA_real_)
```

```
[1] "numeric"
```

2.4.1 Ääretön ja miinus ääretön

R:ssä on myös ääretön ja miinus ääretön. Ne on toteutettu samaan tapaan kuin puuttuva arvo, mutta niiden tarkasteluun on omat funktionsa. Ääretön ja miinus ääretön arvot syntyvät esimerkiksi silloin, kun nollasta poikkeavia lukuja jaetaan nollalla.

```
# You can type in infinity or minus infinity if needed
x <- c(1, 2, Inf, 5, -Inf)
# Use is.finite to determine if numbers are finite or not
is.finite(x)
```

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

```
# Division by zero makes Inf or -Inf (unless 0/0)
x_div_zero <- c(1, 2, -3) / c(3, 0, 0)
x_div_zero
```

```
[1] 0.3333333       Inf      -Inf
```

```
is.finite(x_div_zero)
```

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

2.4.2 Ei-numero

Mikäli R:ssä sattuu tekemään jonkin matemaattisen toimenpiteen, joka ei ole sallittu, esimerkiksi nollan jaon nollalla tai luvun -1 logaritmin, niin tämä tuottaa R:ssä tietotyypin, joka on `NaN` (lyhenne sanoista Not a Number). Mikäli `NaN`-arvoa tutkii funktiolla `is.finite` tai `is.na`, niin huomaa että `NaN` ei ole äärellinen ja `NaN` tulkitaan `NA`:ksi.

```
x_div_zero_by_zero <- 0/0

# Tests for NaN
is.nan(x_div_zero_by_zero)
```

```
[1] TRUE
```

```r
# Tests if it is finite
is.finite(x_div_zero_by_zero)
```

```
[1] FALSE
```

```r
# Tests if it is NA (missing)
is.na(x_div_zero_by_zero)
```

```
[1] TRUE
```

3 Tietotyypit

Tässä osassa tutustutaan neljään uuteen tietorakenteeseen:

- Datakehikko (data.frame)
- Matriisi (matrix)
- Ristitaulukko (table)
- Taulukko (array)
- Lista (list)

Datakehikko on R:n objekti, jossa voidaan säilyttää aineistoa. Aineiston muuttujat ovat datakehikon sarakkeita ja havaintoyksiköt rivejä. Datakehikossa jokaisella muuttujilla tulee aina olla nimi. Datakehikko on tyypillisin tietotyyppi erilaisten aineistojen käsittelyyn, jotka sisältävät useamman kuin yhden muuttujan.

Matriisi voi olla entuudestaan tuttu käsite myös tilastotieteen tai matematiikan kursseilta, ja R:n matriisi vastaakin matemaattista matriisia. Tästä syystä matriisi on hyvin yleinen tietorakenne, johon ei voi olla törmäämättä jos käyttää R:ää tutkimuksessa. Matriisilla tehdään yleensä kuitenkin matemaattisia operaatioita, eikä se ensisijaisesti ole aineiston säilytyspaikka.

Taulukko on juuri sitä miltä se kuulostaa: vektorintapainen tietorakenne, johon tallennetaan alkioita (elements), joilla on kaikilla sama luokka (class), eli esimerkiksi lukuja. Ero vektoriin on se, että taulukolla on useampi ulottuvuus. Matriisi on erikoistapaus taulukosta, sillä matriisi on kaksiulotteinen taulukko. Matriisi vastaa siis oikeastaan paremmin sitä mielikuvaa, joka monelle tulee mieleen suomen kielen sanasta taulukko, ja matriisit ovatkin paljon yleisempiä kuin moniulotteiset taulukot. Taulukoita käytetään yleensä frekvenssijakaumien tai suhteellisten osuuksien tarkasteluun ja testaamiseen.

Lista on järjestetty kokoelma alkioita, jotka voivat olla eri tyyppisiä objekteja.

Koska datakehikko on kaikista tärkein ja eniten käytetty tietotyyppi, niin aloitetaan siitä.

3.1 Datakehikko (data.frame)

Datakehikko (*data frame*) on erittäin yleinen tietorakenne tiedon tallentamiseen R:ssä. Datakehikko on kaksiulotteinen tietorakenne, eli sillä on rivejä ja sarakkeita. Datakehikon sarakkeet muodostuvat vektoreista. Sarakevektorit voivat olla eri luokan vektoreita, mutta

datakehikko asettaa lisärajoitteen: vektoreiden on oltava yhtä pitkiä. Yhden rivin sarakkeilla olevien arvojen ajatellaan koskevan yhtä havaintoa. Sarakkeet voivat sisältää myös puuttuvaa tietoa (eli `NA` arvoja).

Luodaan datakehikko, jossa on kaksi muuttujaa, **`height`** ja **`weight`**, ja sijoitetaan niihin kahdeksan mittauksen tiedot. Huomionarvoista on se, että komennon **`data.frame`** sulkujen sisällä on käytettävä yhtäsuuruusmerkkiä (=) eikä sijoitusoperaattoria (<-). Tämä johtuu siitä, että teknisesti ottaen **`data.frame`** on funktio (funktioista lisää myöhemmin).

```
study_data <- data.frame(
  ID = 1:8,
  height = c(189.8, 184.0, 173.8, 175.9, 169.0, 183.7, 181.8, 16.9),
  gender = c("male", "female", "male", "male", "female", "male", "male", "female")
)
study_data
```

```
  ID height gender
1  1  189.8   male
2  2  184.0 female
3  3  173.8   male
4  4  175.9   male
5  5  169.0 female
6  6  183.7   male
7  7  181.8   male
8  8   16.9 female
```

Huomaa, että jos sarakkeita ei itse nimeä, niin **`data.frame`** nimeää ne automaattisesti, mutta näin luodut nimet eivät välttämättä ole ollenkaan kuvaavia.

```
no_names_data <- data.frame(
  1:8,
  c(189.8, 184.0, 173.8, 175.9, 169.0, 183.7, 181.8, 16.9),
  c("male", "female", "male", "male", "female", "male", "male", "female")
)
no_names_data
```

```
  X1.8 c.189.8..184..173.8..175.9..169..183.7..181.8..16.9.
1    1                                                189.8
2    2                                                184.0
3    3                                                173.8
4    4                                                175.9
5    5                                                169.0
```

```
6    6                                                    183.7
7    7                                                    181.8
8    8                                                     16.9
  c..male....female....male....male....female....male....male...
1                                                                 male
2                                                               female
3                                                                 male
4                                                                 male
5                                                               female
6                                                                 male
7                                                                 male
8                                                               female
```

Datakehikkojen käsittelystä kerrotaan tarkemmin luvussa Datan muokkaaminen.

3.2 Matriisi

3.2.1 Matriisin luominen

Matriisin luominen on yksinkertaista ja se tapahtuu funktiolla `matrix`.

```
matrix(1:9, nrow = 3, ncol = 3)
```

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

Funktiolle annetaan siis matriisiin tallennettavat luvut vektorina, sekä matriisin rivien ja sarakkeiden määrä (argumentit `ncol` ja `nrow`). Arvot sijoitetaan matriisiin sarakkeittain. Matriisi voi koostua myös kokonaan tietystä arvosta:

```
matrix(0, nrow = 2, ncol = 5)
```

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

Jos matriisin tallennettavat luvut annetaan vektorina, niin tällöin riittää antaa vain joko rivien tai sarakkein lukumäärä, ja R osaa päätellä puuttuvan dimension annetun vektorin perusteella.

```
matrix(1:9, nrow = 3)
```

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

Useimmiten matriisien data luetaan R:ään jostain tiedostosta, joka on tuotettu Excelillä tai jollain muulla ohjelmalla (tutkimustulosten kirjaus suoraan R:ään on raskasta). Matriisien luonti käsin on kuitenkin hyvä osata, sillä pienillä matriiseilla on kätevää testata omaa koodia ja tehdä matriisilaskutoimituksia. Myös yllä olevan kaltaisia, esim. nollalla täytettyjä matriiseja on joskus kätevää käyttää "alustana", kun lasketaan omasta datasta tuloksia rivi tai sarake kerrallaan. Tämä johtuu siitä, että olemassa olevan matriisin rivin arvojen muuttaminen on nopeampi operaatio kuin rivin lisääminen matriisiin.

3.2.2 Matriisin koko

Joskus voi törmätä matriiseihin, joiden kokoa ei tiedetä, tai ei haluta olettaa. Tällöin tarvitaan funktioita, jotka kertovat matriisin koosta. Esimerkiksi, kun luetaan dataa R:ään tiedostoista, on hyvä tarkistaa, että kaikki rivit ja sarakkeet ovat mukana. Funktiot `nrow` ja `ncol` palauttavat rivien ja sarakkeiden määrän, `dim` palauttaa matriisin rivien ja sarakkeiden määrän vektorina, jossa rivien määrä on ensimmäinen alkio.

```
X <- matrix(1:12, ncol = 4)
# Number of rows
nrow(X)
```

```
[1] 3
```

```
# Number of columns
ncol(X)
```

```
[1] 4
```

```r
# Dimensions
dim(X)
```

```
[1] 3 4
```

Nämä funktiot toimivat myös datakehikoille.

3.2.3 Matriisin indeksointi

Matriisin indeksointi on hyvin samantapainen operaatio kuin vektorin indeksointi, eli matriisin perään laitetaan hakasulkeet ja niihin määritellään halutut indeksit. Matriisin indeksoinnissa pitää kuitenkin antaa erikseen indeksit riveille ja sarakkeille, pilkulla erotettuna. Jos hakasulkeisiin antaa vain yhden luvun ilman pilkkua, niin R käsittelee matriisia vektorina, jolloin indeksointi tapahtuu kuten vektoreiden tapauksessa. Datakehikoita voidaan indeksoida useimmissa tapauksissa kuten matriiseja.

```r
# Only nrow is enough, since the number of columns must be 3
X <- matrix(1:9, nrow = 3)
X
```

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

```r
# Element on second row, third column
X[2, 3]
```

```
[1] 8
```

```r
# The complete first row
X[1, ]
```

```
[1] 1 4 7
```

```r
# The second and third values of the second column
X[2:3, 3]
```

```
[1] 8 9
```

```r
# Get rows where the values of the first column are > 1
X[X[, 1] > 1, ]
```

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

HUOM: jos matriisia indeksoidessa tuloksessa sarakkeiden tai rivien määrä on tasan yksi, kuten yllä olevissa esimerkeissä viimeistä lukuun ottamatta, tuloksena on vektori, ei matriisi. Jos haluaa tuloksen olevan matriisi, tulee hakasulkeisiin lisätä argumentti `drop = FALSE`

```r
# The complete first row
X[1, , drop = FALSE]
```

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

```r
# The second and third values of the second column
X[2:3, 3, drop = FALSE]
```

```
     [,1]
[1,]    8
[2,]    9
```

Matriiseja voi myös muokata sijoittamalla haluttuihin paikkoihin uusia arvoja:

```r
# Copy of X
X_new <- X
# Replace first row with new values
X_new[1, ] <- c(10, 13, 15)
X_new
```

```
     [,1] [,2] [,3]
[1,]   10   13   15
[2,]    2    5    8
[3,]    3    6    9
```

```
# Replacement can also be a single value, and will be recycled
X_new[2:3, 1] <- 0
X_new
```

```
     [,1] [,2] [,3]
[1,]   10   13   15
[2,]    0    5    8
[3,]    0    6    9
```

Matriisista voi myös poimia tietyt rivit tai sarakkeet jättämällä tiettyjä rivejä tai sarakkeita pois. Tämä tapahtuu antamalla indeksi miinusmerkkisenä:

```
# Without first row
X[-1, ]
```

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

```
# Without second column
X[, -2]
```

```
     [,1] [,2]
[1,]    1    7
[2,]    2    8
[3,]    3    9
```

Huomaa kuitenkin, että positiivisia ja negatiivisia indeksejä ei voi käyttää samanaikaisesti tietyssä dimensiossa:

```
# Trying to mix positive and negative indices
X[c(-1, 1), ]
```

```
Error in X[c(-1, 1), ]: only 0's may be mixed with negative subscripts
```

3.2.4 Indeksimatriisi (index matrix)

Jos halutaan poimia useampi yksittäinen arvo matriisista, tulee käyttää indeksimatriisia (index matrix).

Esimerkiksi, jos haluttaisiin poimia äskeisestä matriisista `x` arvot indekseissä [1, 2], [1, 3] ja [2, 2], niin seuraava koodi ei toimi:

```
X[c(1, 1, 2), c(2, 3, 2)]
```

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

vaan tulee käyttää indeksimatriisia, jonka jokainen rivi antaa yhden halutun alkion rivi- ja sarakeindeksin tässä järjestyksessä. Indeksimatriiseja tehdessä kannattaa asettaa argumentti `byrow = TRUE`, jolloin alkiot laitetaan matriisiin rivi kerrallaan, ei sarake kerrallaan kuten oletusarvoisesti tehtäisiin.

```
i <- matrix(c(1, 2, 1, 3, 2, 2), nrow = 3, byrow = TRUE)
i
```

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

```
X[i]
```

```
[1] 4 7 5
```

3.2.5 Matriisien rakentaminen vektoreista

Matriisi koostuu usein useammasta muuttujasta ja havainnoista. Yleensä jokainen rivi vastaa yhtä havaintoa, ja sarake muuttujaa. Tämän takia on hyvä tietää, miten yksittäisistä vektoreista saa koottua matriiseja. Alla olevassa esimerkissä on koottu yhteen matriisiin Star Wars -hahmojen pituuksia ja painoja. Tämä tapahtuu `cbind` funktiolla (column bind), joka nimensä mukaisesti yhdistää vektorit matriisin sarakkeiksi. `cbind` voi yhdistää myös valmiita matriiseja yhteen, niin että matriisit ovat "vierekkäin" eli yhdistetyssä matriisissa on kummankin matriisin sarakkeet (rivien määrän tulee olla sama). Vastaavasti `rbind` (row bind) yhdistää matriiseja "allekkain" (sarakkeiden määrän tulee olla sama).

```r
heights <- c(172, 167, 96, 202, 150, 178)
masses <- c(77, 75, 32, 136, 49, 120)

starwars <- cbind(heights, masses)
starwars
```

```
     heights masses
[1,]     172     77
[2,]     167     75
[3,]      96     32
[4,]     202    136
[5,]     150     49
[6,]     178    120
```

3.2.6 Rivien ja sarakkeiden nimeäminen

Matriisien rivit ja sarakkeet voi nimetä, ja usein tässä onkin järkeä. Yllä olevassa esimerkissä **starwars**-matriisin sarakkeet on nimetty alkuperäisten vektorien mukaan. Alla olevassa esimerkissä on lisää tapoja nimetä rivejä ja sarakkeita

```r
# Set column names by naming arguments while building matrix from vectors
cbind(Height = heights, Mass = masses)
```

```
     Height Mass
[1,]    172   77
[2,]    167   75
[3,]     96   32
[4,]    202  136
[5,]    150   49
[6,]    178  120
```

```r
# Set column and row names explicitly
colnames(starwars) <- c("Height", "Mass")
rownames(starwars) <- c(
  "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", "Leia Organa", "Owen Lars"
)
starwars
```

```
               Height Mass
Luke Skywalker    172   77
```

```
C-3PO             167   75
R2-D2              96   32
Darth Vader       202  136
Leia Organa       150   49
Owen Lars         178  120
```

Nimettyjä matriiseja voi indeksoida myös nimien perusteella:

```
starwars[c("Luke Skywalker", "R2-D2"), ]
```

```
               Height Mass
Luke Skywalker    172   77
R2-D2              96   32
```

Matriisiin voi myös lisätä uusia sarakkeita `cbind` funktiolla. Alla lisätään matriisiin starwars uusi sarake, jossa on hahmojen BMI:

```
# Create a vector for BMI and add to matrix with cbind
bmi <- starwars[, "Mass"] / (starwars[, "Height"] / 100)^2
cbind(starwars, "BMI" = bmi)
```

```
               Height Mass      BMI
Luke Skywalker    172   77 26.02758
C-3PO             167   75 26.89232
R2-D2              96   32 34.72222
Darth Vader       202  136 33.33007
Leia Organa       150   49 21.77778
Owen Lars         178  120 37.87401
```

3.2.7 Matriiseilla laskeminen

Matriiseilla laskeminen on hyvin samankaltaista kuin vektoreilla laskeminen. Matriisin ja yksittäisen luvun välisessä operaatiossa matriisin alkiot käsitellään yksitellen. Samoin samankokoiset matriisit voi esim. lisätä yhteen, jolloin lisäys tapahtuu alkio kerrallaan.

```
X <- matrix(1:9, nrow = 3)
Y <- matrix(3:11, nrow = 3, ncol = 3)
# Element-wise multiplication
X * 2
```

```
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
```

```
# Element-wise sum
X + Y
```

```
     [,1] [,2] [,3]
[1,]    4   10   16
[2,]    6   12   18
[3,]    8   14   20
```

Matriiseille on lisäksi määritelty paljon matriisien omia laskutoimituksia, joita ei käsitellä tarkemmin tässä materiaalissa. Matriisilaskentaa opiskelleille huomio: R:ssä oletuksena kertolasku tehdään alkioittain, matriisitulo tapahtuu operaattorilla `%*%` ja matriisin transpoosin voi määrittää funktiolla `t`.

3.3 Ristitaulukko

Ristitaulukko (kontingenssitaulukko, *contingency table*) on matriisia muistuttava kaksi- tai useampiulotteinen tietorakenne frekvenssiaineistojen käsittelyyn. Tässä materiaalissa käsittelemme vain kaksiulotteisia ristitaulukoita yksinkertaisuuden vuoksi. Ristitaulukko kuvaa kahden luokittelu- tai järjestysasteikollisen muuttujan havaintojakaumaa: jokaisessa taulukon solussa on tietyn muuttujien tasojen yhdistelmän havaintojen lukumäärä aineistossa. Huomaa, että ristitaulukko ei ole taulukko (*array*)!

Tarkastellaan esimerkkinä lämpötilan ja kuukauden ristitaulokkoa R:n sisäisessä ilmanlaatuaineistossa `airquality`. Taulukko luodaan funktiolla `table`, joka ottaa argumentteinaan kaksi muuttujaa, joista ristitaulukko muodostetaan. Tässä esimerkissä lisäksi funktio `cut` muodostaa lämpötilamuuttujasta `Temp` luokitteluasteikollisen muuttujan sen kvartiilien perusteella.

```
crosstab <- table(
  cut(airquality$Temp, quantile(airquality$Temp)),
  airquality$Month
)
crosstab
```

```
         5  6  7  8  9
(56,72] 24  3  0  1 10
(72,79]  5 15  2  9 10
(79,85]  1  7 19  7  5
(85,97]  0  5 10 14  5
```

Tässä tapauksessa ristitaulukon rivimuuttuja on lämpötila ja sarakemuuttuja on kuukausi. Frekvenssien sijaan voimme myös tarkastella suhteellisia osuuksia `prop.table` funktiolla.

```
prop.table(crosstab)
```

```
                  5           6           7           8           9
(56,72] 0.157894737 0.019736842 0.000000000 0.006578947 0.065789474
(72,79] 0.032894737 0.098684211 0.013157895 0.059210526 0.065789474
(79,85] 0.006578947 0.046052632 0.125000000 0.046052632 0.032894737
(85,97] 0.000000000 0.032894737 0.065789474 0.092105263 0.032894737
```

Usein kiinnostavampaa on kuitenkin tarkastella ehdollisia suhteellisia osuuksia, eli osuuksia joko rivi- tai sarakemuuttujan eri tasojen sisällä. Tämän mahdollistaa `prop.table`-funktion argumentti `margin`, joka määrittää, tehdäänkö tarkastelu rivimuuttujan (`margin = 1`) vai sarakemuuttujan (`margin = 2`) suhteen.

```
prop.table(crosstab, margin = 1)
```

```
                 5          6          7          8          9
(56,72] 0.63157895 0.07894737 0.00000000 0.02631579 0.26315789
(72,79] 0.12195122 0.36585366 0.04878049 0.21951220 0.24390244
(79,85] 0.02564103 0.17948718 0.48717949 0.17948718 0.12820513
(85,97] 0.00000000 0.14705882 0.29411765 0.41176471 0.14705882
```

```
prop.table(crosstab, margin = 2)
```

```
                 5          6          7          8          9
(56,72] 0.80000000 0.10000000 0.00000000 0.03225806 0.33333333
(72,79] 0.16666667 0.50000000 0.06451613 0.29032258 0.33333333
(79,85] 0.03333333 0.23333333 0.61290323 0.22580645 0.16666667
(85,97] 0.00000000 0.16666667 0.32258065 0.45161290 0.16666667
```

Ristitaulukkoa voidaan myös käyttää rivi- ja sarakemuuttujan riippuvuuden testaamiseen Khiin neliö -testillä, johon palataan myöhemmin luvussa 8.3.

3.4 Tietotyyppien tarkastelu

Kaikkia objekteja voi tulostaa Console-ikkunassa kutsumalla objektin nimen. Joskus tarvitaan kuitenkin apufunktioita.

3.4.1 View()

Mikäli käytät RStudiota, niin tarkempaa tarkastelua varten kannattaa kuitenkin käyttää `View`-funktiota. `View` avaa ikkunan, jossa voi selata data framen tai matriisin rivejä ja sarakkeita, sekä järjestää arvoja halutun sarakkeen mukaan (tämä järjestys säilyy vain `View`-näkymässä, itse muuttujan rakenne ei muutu). Mikäli aineistossasi on satoja tuhansia tai miljoonia rivejä, niin `View` saattaa olla liian hidas.

3.4.2 str()

Perineinen tapa tarkastella objekteja R:ssä on funktio `str`, joka toimii kaikissa R-ympäristöissä. Funktio `str` tulostaa tiivistetyssä muodossa kaiken, mitä sille annettu objekti sisältää. Esimerkiksi datakehikon tapauksessa sen avulla saadaan sekä muuttujien nimet, niitä vastaavien vektoreiden tyypit että ruudulle mahtuvan osan vektoreiden alkioista.

```
# Examine the structure of data frame
str(study_data)
```

```
'data.frame':   8 obs. of  3 variables:
 $ ID    : int  1 2 3 4 5 6 7 8
 $ height: num  190 184 174 176 169 ...
 $ gender: chr  "male" "female" "male" "male" ...
```

3.4.3 head()

Jos aineistossa on todella paljon rivejä, on sen tulostaminen Console-ikkunaan ikävää. Ladataan esimerkiksi `iris`-data, jossa on 150 havaintoa. Tulostettaessa rivejä on niin monta, että muuttujien nimet eivät näy, mikä on epämiellyttävää. Parempi tapa saada käsitys aineistosta on kutsua sitä `head`-funktion avulla.

```r
# Load data for this example
data(iris)

# Try to print iris-data directly
iris
```

Console | Terminal | R Markdown | Jobs

R 4.1.0 · ~/Research/Projektit/r_intro_jukop/

138	6.4	3.1	5.5	1.8	virginica
139	6.0	3.0	4.8	1.8	virginica
140	6.9	3.1	5.4	2.1	virginica
141	6.7	3.1	5.6	2.4	virginica
142	6.9	3.1	5.1	2.3	virginica
143	5.8	2.7	5.1	1.9	virginica
144	6.8	3.2	5.9	2.3	virginica
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

```
>
```

```r
# Print 6 first rows of iris data
head(iris)
```


```
# You can also define the number of rows to print
head(iris, 2)
```

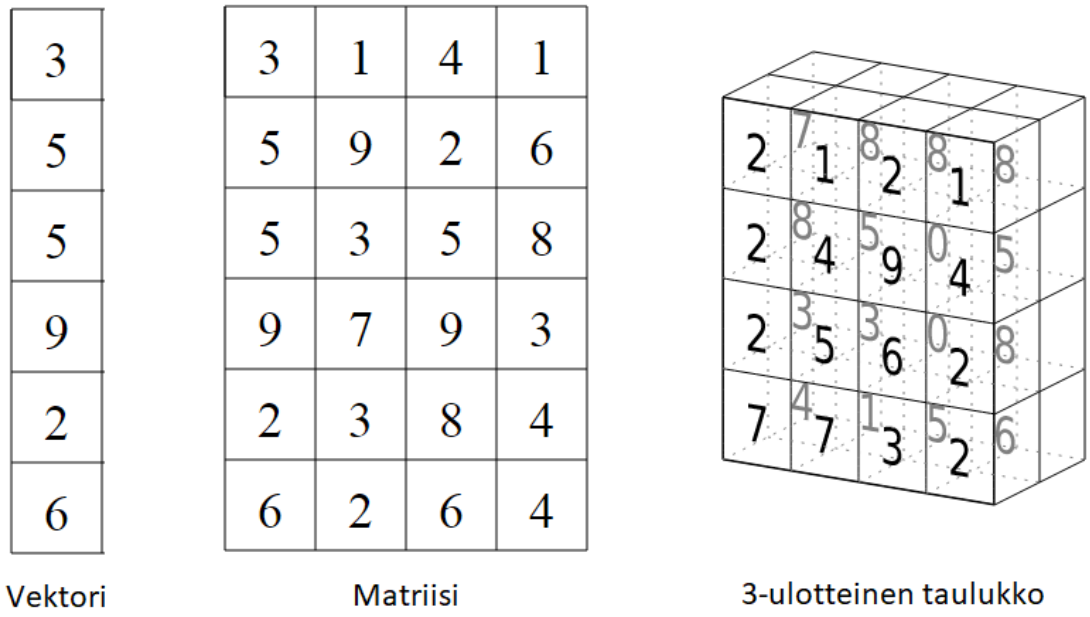
3.5 Extra: Taulukko ja lista

Taulukoita (*array*) ja listoja (*list*) ei tavallista data-analyysiä toteutettaessa yleensä tarvita. Lue kuitenkin seuraava, jotta saat yleiskäsityksen mihin niitä tarvitaan. Voit myös palata perehtymään taulukoihin ja listoihin myöhemmin koska tahansa.

3.5.1 Taulukko

Kuten alussa todettiin, taulukot (array) ovat hyvin harvinaisia, joten niihin ei kannata tällä kurssilla keskittyä. Niitä kuitenkin tarvitaan joidenkin tehtävien tekemiseen, joten tässä on hyvin lyhyt oppimäärä taulukoista.

Taulukot ovat matriisien kaltaisia, mutta taulukossa voi olla yli kaksi ulottuvuutta. Oikeastaan matriisit ovat kaksiulotteisia taulukoita. Alla on esimerkki 3-ulotteisesta taulukosta, jota voi ajatella “peräkkäin” olevina matriiseina. Alla on kuva 1-ulotteisesta taulukosta eli vektorista, 2-ulotteisesta taulukosta eli matriisista ja 3-ulotteisesta taulukosta.



Taulukkoja luodaan matriisien tapaan funktiolla `array`. Toisin kuin matriisien tapauksessa, `array`-funktiolle pitää luetella sen kaikki ulottuvuudet vektorina. Alla oleva esimerkki luo 3-ulotteisen taulukon, jonka voi ajatella koostuvan kolmesta 4×2 matriisista.

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

```
, , 1

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

, , 2

     [,1] [,2]
[1,]    9   13
[2,]   10   14
[3,]   11   15
[4,]   12   16
```

```
, , 3

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

Taulukoita indeksoidaan aivan kuten matriiseja, mutta jokaiselle ulottuvuudelle on annettava oma indeksi:

```
# The first 2 rows of each "layer"
my_array[1:2, , ]
```

```
, , 1

     [,1] [,2]
[1,]    1    5
[2,]    2    6

, , 2

     [,1] [,2]
[1,]    9   13
[2,]   10   14

, , 3

     [,1] [,2]
[1,]   17   21
[2,]   18   22
```

```
# Second column from last two layers
my_array[, 2, 2:3]
```

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

3.5.2 Lista

Listat ovat tärkeitä erityisesti silloin, kun aletaan toteuttamaan uusia toimintoja R-kieleen omien funktioiden avulla. Niin kauan kun valmiit R-funktiot riittävät, ei listoilla ole juuri käyttöä.

Lista (list) on vektorinkaltainen tietorakenne, jossa on järjestyksessä alkioita, jotka on mahdollisesti nimetty. Tärkeä ero vektoriin verrattuna on, että listan alkiot voivat olla erityyppisiä. Listoja luodaan `list`-funktiolla:

```
example_list <- list(
  c(1, 2, 3),
  matrix(0, nrow = 3, ncol = 4),
  "list can include anything"
)
example_list
```

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

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

[[3]]
[1] "list can include anything"
```

```
subject_ids <- c("ANKL", "PEPA", "DIPR")
measurements <- matrix(
  c(
    1, 2.5, 3,
    3.5, 5, 3,
    2.3, 3, 1.6
  ),
  nrow = 3
)
colnames(measurements) <- c("CRP", "HDL", "LDL")
rownames(measurements) <- subject_ids
# List names can be given with or without quotes
study <- list(
```

```
  Subject_ID = subject_ids,
  "Measurements" = measurements,
  Study_name = "Blood tests"
)
study
```

```
$Subject_ID
[1] "ANKL" "PEPA" "DIPR"

$Measurements
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6

$Study_name
[1] "Blood tests"
```

Listoja ja niiden kaltaisia olioita käytetään R:ssä paljon. Listoihin on kätevä tallentaa erityyppistä tietoa, joka kuitenkin halutaan säilyttää yhtenä kokonaisuutena. Esimerkiksi yksinkertaisetkin tilastolliset mallit tuottavat paljon erilaista tietoa, joka tallennetaan listaan (tarkemmin listan kaltaiseen olioon, tästä lisää myöhemmin).

3.5.2.1 Listojen alkioiden käsittely

Listan alkioihin pääsee käsiksi kahdella eri tavalla: kaksoishakasulkeilla `[[]]` tai, jos lista on nimetty, dollarimerkillä `$`:

```
# By position
study[[2]]
```

```
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6
```

```
# By name
study[["Subject_ID"]]
```

```
[1] "ANKL" "PEPA" "DIPR"
```

```r
# Using dollar sign
study$Study_name
```

```
[1] "Blood tests"
```

Listaa voi indeksoida myös yksinkertaisilla hakasulkeilla. Tällöin palautetaan aina lista, eikä yksittäistä alkiota kuten aiemmin. Palautetaan ensiksi mieleen funktio `class`, joka palauttaa argumenttinsa luokan (class). Vektorin luokka vaihtelee vektorin sisällön mukaan: numeric = lukuja, character = merkkijonoja, logical = loogisia arvoja, jne. Listojen luokka on luonnollisesti list. R:ssä kaikki muuttujiin tallennettavat tiedot ovat olioita (object). R-olioilla on aina luokka, joka määrittää sen ominaisuudet. Esimerkiksi `print` ja `plot`-komennot toimivat eri tavalla riippuen niiden argumentin luokasta.

Tarkastellaan alla, mikä ero yksinkertaisilla ja kaksinkertaisilla hakasulkeilla on listan indeksoinnissa:

```r
# Returns a list of length one with the matrix as the only element
study[2]
```

```
$Measurements
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6
```

```r
class(study[2])
```

```
[1] "list"
```

```r
# Returns the actual matrix
study[[2]]
```

```
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6
```

```r
class(study[[2]])
```

```
[1] "matrix" "array"
```

```r
# Dollar sign also returns the matrix
class(study$Measurements)
```

```
[1] "matrix" "array"
```

```r
# Single brackets works as subscripting just like with vectors
study[2:3]
```

```
$Measurements
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6

$Study_name
[1] "Blood tests"
```

3.5.2.2 Alkion lisäys listaan ja listojen yhdistäminen

Yksittäisen alkion voi lisätä listaan sijoittamalla listan johonkin indeksiin tai nimeen uusi arvo (indeksin pitää olla yhtä suurempi kuin listan pituus). HUOM! Listan alkio voi myös itse olla lista (sisäkkäinen lista = nested list).

```r
# Add a character matrix as the fourth element of study
study[[4]] <- matrix(
  c(
    "CPR", "HDL", "LDL",
    "C-reactive protein", "High-density lipoprotein", "Low-density lipoprotein"
  ),
  ncol = 2
)
# An element of a list can also be a list
study[["professional"]] <- list(
  name = c("John H. Watson"),
  position = "Medical doctor",
```

```
  age = 45
)
study
```

```
$Subject_ID
[1] "ANKL" "PEPA" "DIPR"

$Measurements
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6

$Study_name
[1] "Blood tests"

[[4]]
     [,1]  [,2]
[1,] "CPR" "C-reactive protein"
[2,] "HDL" "High-density lipoprotein"
[3,] "LDL" "Low-density lipoprotein"

$professional
$professional$name
[1] "John H. Watson"

$professional$position
[1] "Medical doctor"

$professional$age
[1] 45
```

```
# Note that the fourth element has no name
names(study)
```

```
[1] "Subject_ID"   "Measurements" "Study_name"   ""             "professional"
```

Listoja voi yhdistää vektorien tapaan c-funktiolla:

```r
# Concatenate two vectors
vector1 <- c(3, 6, 5)
vector2 <- c(1, 2, 3)
c(vector1, vector2)
```

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

```r
list1 <- list(vector = vector1, name = "list1")
list2 <- study[1:2]
# Concatenate three lists, names stay the same
c(list1, list2, list(first_element = "A", second = "B"))
```

```
$vector
[1] 3 6 5

$name
[1] "list1"

$Subject_ID
[1] "ANKL" "PEPA" "DIPR"

$Measurements
     CRP HDL LDL
ANKL 1.0 3.5 2.3
PEPA 2.5 5.0 3.0
DIPR 3.0 3.0 1.6

$first_element
[1] "A"

$second
[1] "B"
```

4 Tunnusluvut

Tunnusluvut (statistics) ovat keskeinen osa tilastotiedettä. Tunnuslukujen avulla voidaan tiivistää ja tarkastella aineistoa. Tässä luvussa käsitellään tyypillisimpien tunnuslukujen laskemista aineistosta. Näitä tunnuslukuja voi sanoa myös empiirisiksi, koska ne on laskettu aineistosta.

4.1 Sijaintia kuvaavat tunnusluvut

4.1.1 Minimi ja maksimi

Minimi tarkoittaa aineiston pienintä arvoa kyseiselle muuttujalle. Maksimi on vastaavasti suurin arvo. Minimi ja maksimi ovat periaatteessa helppo laskea funktioiden `min` ja `max` avulla, mutta niihinkin liittyy pari pientä sudenkuoppaa. Funktiot `min` ja `max` hyväksyvät argumenteikseen vain numeerisia vektoreita.

```
dat_for_loc <- c(-1.25, -4.1, 1.16, -3.05, 4.17, 0.73, -3.14, 3.39, -2.55, 0.4)
min(dat_for_loc)
```

```
[1] -4.1
```

```
max(dat_for_loc)
```

```
[1] 4.17
```

Joskus minimiä ja maksimia tarvitaan tilanteessa, jossa halutaan vaikkapa muuttaa kaikki negatiiviset arvot nolliksi (tai positiiviset, jos maksimi). Tämä onnistuu helpoiten funktioiden `pmin` ja `pmax` avulla. Samalla tapaa, jos halutaan kaikki lukua 1 pienemmät luvut muutettua luvuksi 1, niin tämä onnistuu vaihtamalla toinen argumentti luvuksi 1.

```
# We want to get rid of all values below 0 and make them 0
pmin(dat_for_loc, 0)
```

```
 [1] -1.25 -4.10  0.00 -3.05  0.00  0.00 -3.14  0.00 -2.55  0.00
```

```
# Similar, but get rid of all values over 0
pmax(dat_for_loc, 0)
```

```
 [1] 0.00 0.00 1.16 0.00 4.17 0.73 0.00 3.39 0.00 0.40
```

```
# We can do similar things to any limit, e.g. 1
pmin(dat_for_loc, 1)
```

```
 [1] -1.25 -4.10  1.00 -3.05  1.00  0.73 -3.14  1.00 -2.55  0.40
```

Funktiota `pmin` ja `pmax` voi käyttää vieläkin yleisemmässä muodossa antamalla yksittäisen lukuarvon sijasta vektorin. Näitä emme käsittele tässä, mutta kiinnostuneet voivat kokeilla lisää itse.

4.1.2 Keskiarvo

Keskiarvo saadaan laskemalla muuttujan kaikki havainnot yhteen ja jakamalla summa havaintojen määrällä. Esimerkiksi aineiston $1, 2, 3, 4$ keskiarvo on $(1 + 2 + 3 + 4)/4 = 2.5$. Keskiarvoa satunnaismuuttujan X havainnoille voidaan merkitä matemaattisesti seuraavasti:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n},$$

missä merkintä $\overline{x}$ tarkoittaa keskiarvoa, $x_1, \ldots, x_n$ ovat havaintoja ja n on havaintojen lukumäärä.

Keskiarvo voidaan laskea helposti funktiolla `mean`.

```
tooth_length <- ToothGrowth$len
mean(tooth_length)
```

```
[1] 18.81333
```

Mikäli muuttujassa on puuttuvia arvoja (`NA`) niin keskiarvoksi tulee oletusarvoisesti `NA`. Puuttuvat arvot voi jättää pois keskiarvon laskennasta antamalla funktiolle lisäargumentiksi `na.rm = TRUE`.

```r
# Create some data
dat_for_mean <- c(1, 2, NA, 4)
# Data with NA results mean with NA
mean(dat_for_mean)
```

```
[1] NA
```

```r
# Leave NA-values out and calculate mean from the remaining ones
mean(dat_for_mean, na.rm = TRUE)
```

```
[1] 2.333333
```

4.1.3 Mediaani

Mediaani ilmaisee aineiston keskimmäisen havainnon. Toisin sanoen puolet havainnoista on mediaania suurempia ja puolet mediaania pienempiä. Esimerkiksi aineiston $1, 1, 2, 3, 5$ mediaani on 2. Jos aineistossa on parillinen määrä lukuja, otetaan kaksi keskimmäistä ja lasketaan ne yhteen ja jaetaan kahdella (keskiarvo). Aineiston $3, 3, 5, 6, 7, 17$ mediaani on $(5 + 6)/2 = 5.5$. Mediaani on helppoa laskea funktiolla `median`.

```r
# Let's think about median
dat_for_median <- c(7, 2, 3, 4, 1, 7, 0, 4, 3, 3, 2, 6)
dat_for_median
```

```
 [1] 7 2 3 4 1 7 0 4 3 3 2 6
```

```r
sort(dat_for_median) # Median would be the middle value in the arranged data, thus 3
```

```
 [1] 0 1 2 2 3 3 3 4 4 6 7 7
```

```r
# Getting median in R
median(dat_for_median)
```

```
[1] 3
```

4.1.4 Kvantiilit

Mediaani siis kertoi kohdan, jossa 50 % aineistosta on pienenmpiä kuin kyseinen arvo. Entä jos haluamme luvun, jota pienempiä ovat vaikkapa 10 % aineiston havainnoista tai mikä tahansa muu osuus? Tällainen yleistys on nimeltään kvantiili. Joillakin kvantiileilla on erityisnimet. Ne ovat

- mediaani (50 % aineistosta on tätä pienempiä)
- alakvartiili (25 %)
- yläkvartiili (75 %)
- desiilit (10% välein)
  - 10 %:n desiili, 20 %:n desiili jne.

Haluamansa kvantiilin voi laskea funktiolla `quantile`. Jos haluat laskea 30 %:n kvantiilin, niin anna argumentille `probs` tätä vastaava suhteellinen osuus eli 0.30.

```
quantile(dat_for_median, probs = 0.30)
```

```
30%
2.3
```

`quantile`-funktiolle voi antaa useita kvantiileita laskettavaksi kerralla. Tällöin argumentille `probs` on annettava vektori. Esimerkiksi kvartiilit ja mediaanin voi laskea samanaikaisesti näin:

```
quantile(dat_for_median, probs = c(0.25, 0.5, 0.75))
```

```
25% 50% 75%
2.0 3.0 4.5
```

Ääritapauksena voidaan havaita, että laskemalla 0 %:n ja 100 %:n kvantiilit saadaan tulokseksi minimi ja maksimi (yksikään arvo ei ole minimiä pienempi eikä yksikään maksimia suurempi. Samaan lopputulokseen pääsee myös funktiolla `range`. Kokeillaan tätä

```
# 0 % and 100 % quantile gives a range of the data
quantile(dat_for_median, probs = c(0.00, 1.00))
```

```
  0% 100%
   0    7
```

```r
# Let's compare with min and max
min(dat_for_median)
```

```
[1] 0
```

```r
max(dat_for_median)
```

```
[1] 7
```

```r
# There is also function called range
range(dat_for_median)
```

```
[1] 0 7
```

Esimerkiksi viiksilaatikkokuvaa vastaavat lukuarvot eli minimin, alakvartiilin, mediaanin, yläkvartiilin ja maksimin saa kätevästi quantile-funktiolla antamalla `probs`-argumentille vektorin `c(0, 0.25, 0.5, 0.75, 1)`. Tätä sanotaan joskus viiden numeron yhteenvedoksi.

```r
quantile(dat_for_median, probs = c(0, 0.25, 0.5, 0.75, 1))
```

```
  0%  25%  50%  75% 100% 
 0.0  2.0  3.0  4.5  7.0 
```

4.1.5 Moodi

Moodi ilmaisee muuttujan yleisimmän arvon. Valitettavasti R:ssä ei ole valmista funktiota moodin laskemiseen. Sen sijaan funktio nimeltään `mode` antaa objektin tyypin, eikä laske moodia. Jos moodin haluaa laskea R:ssä, on ensin muodostettava aineistosta frekvenssitaulukko ja sitten etsittävä taulokosta se arvo, josta on eniten havaintoja, eli suurin frekvenssi

```r
# Find out the mode
dat_for_mode <- c(
  7, 2, 3, 4, 1, 7, 0, 4, 3, 3, 2, 6, 1, 3, 3, 1, 6, 0, 1, 3,
  0, 6, 4, 2, 3, 2, 2, 7, 3, 1, 5, 3, 4, 3, 3, 2, 2, 4, 2, 1,
  5, 3, 2, 2, 2, 3, 4, 2, 5, 3, 4, 2, 1, 4, 2, 3, 1, 1, 4, 3,
  2, 3, 5, 4, 4, 4, 1, 3, 1, 3, 5, 2, 3, 1, 4, 2, 4, 2, 1, 0,
  3, 3, 3, 3, 4, 4, 4, 3, 4, 4, 2, 1, 2, 4, 4, 4, 6, 2, 3, 2
```

```
)
tab <- table(dat_for_mode)
# Let's see how is tab
tab
```

```
dat_for_mode
 0  1  2  3  4  5  6  7
 4 14 22 26 22  5  4  3
```

```
# Let's pick up the largest frequency using which.max function
tab[which.max(tab)]
```

```
 3
26
```

```
# Mode is 3 and the frequency is 26
# Let's pick up only the value 3
names(tab)[which.max(tab)]
```

```
[1] "3"
```

```
# That is character so let's convert it to numeric
as.numeric(names(tab)[which.max(tab)])
```

```
[1] 3
```

Ylläoleva on hyvä esimerkki tilanteesta, jossa moodin laskemiseen käytetty koodi on kätevä kirjoittaa funktioksi, jolloin moodi on helppo laskea jatkossa toisilla aineistoilla. Funktiohin palataan osiossa Funktiot. Alla on joka tapauksessa esimerkki moodi-funktiosta ilman suurempia selityksiä. **Huom!** tämä moodi-funktio toimii vain numeerisille vektoreille!

```
# Write a function for mode
moodi <- function(x) {
  tab <- table(x)
  as.numeric(names(tab)[which.max(tab)])
}
# Use that function
moodi(dat_for_mode)
```

```
[1] 3
```

```r
# NOTE! This only works for numeric data!
dat_for_mode_character <- c("a", "a", "a", "b", "c", "c")
# This gives NA as output and a warning!
moodi(dat_for_mode_character)
```

```
Warning in moodi(dat_for_mode_character): NAs introduced by coercion
```

```
[1] NA
```

4.2 Vaihtelua kuvaavat tunnusluvut

4.2.1 Varianssi ja keskihajonta

Yksittäiselle numeeriselle muuttujalle voidaan laskea varianssi (*variance*) funktiolla `var`. Varianssia tulkittaessa kannattaa muistaa, että varianssin mittayksikkö ei ole sama kuin alkuperäisen muuttujan, vaan mittayksikkö tulee korottaa toiseen potenssiin. Esim. jos pituuden yksikkö on cm, niin pituuden varianssin yksikkö on cm^2. Käytännössä tulkintaa kannattaa yrittää keskihajonnan avulla.

```r
# pull the variable from data frame and use it directly in function var
var(ToothGrowth$len)
```

```
[1] 58.51202
```

```r
# calculate the variance-covariance matrix for entire data frame
# (gives NA to any pairs with categorical variables)
# variances are obtained from the diagonal (58.51, NA, 0.3954)
var(ToothGrowth)
```

```
Warning in var(ToothGrowth): NAs introduced by coercion
```

```
           len supp      dose
len  58.512023   NA 3.8612994
supp        NA   NA        NA
dose  3.861299   NA 0.3954802
```

Keskihajonta (*standard deviation*) saadaan vastaavasti funktiolla `sd`. Keskihajonta on varianssin neliöjuuri.

```
# standard deviation
sd(ToothGrowth$len)
```

```
[1] 7.649315
```

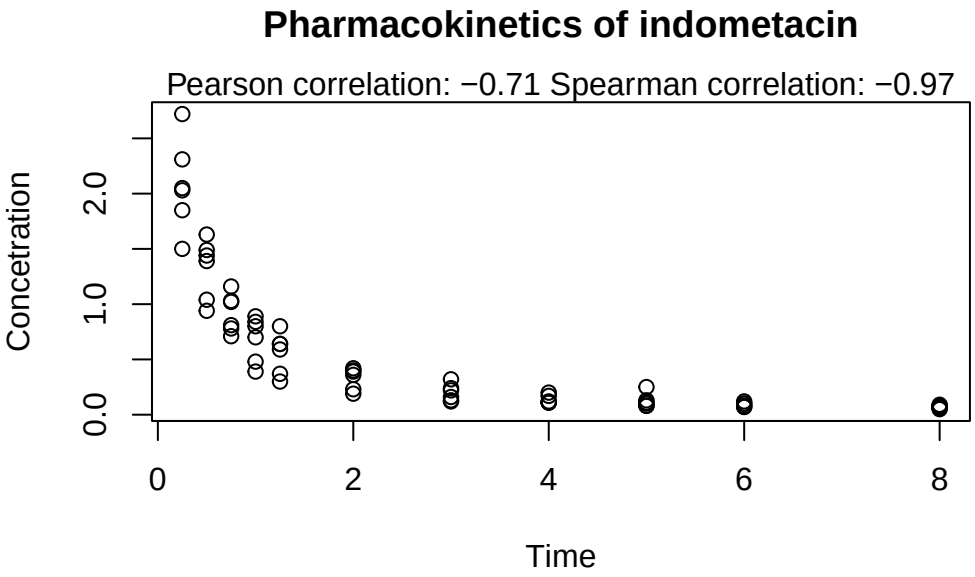
4.2.2 Korrelaatio

Korrelaatio (*correlation*) on suure jolla voidaan mitata kahden muuttujan välistä riippuvuutta. Korrelaatiolle on monia erilaisia mittareita, joista yleisimmät ovat Pearsonin korrelaatiokerroin, joka mittaa kahden muuttujan välistä lineaarista riippuvuutta, ja Spearmanin järjestyskorrelaatiokerroin, joka mittaa kahden muuttujan välistä riippuvuutta ilman lineaarisuusoletusta, mutta olettaa kuitenkin monotonisen riippuvuuden. HUOM: korrelaatio ei ota kantaa siihen, kuinka vahva riippuvuus on (käyrän jyrkkyys), vaan pelkästään siihen, kuinka systemaattinen riippuvuus on. Kummatkin korrelaatiokertoimet saavat arvoja väliltä [-1, 1], jossa -1 on täydellinen negatiivinen korrelaatio (toisen muuttujan kasvaessa toinen aina pienenee) ja 1 on täydellinen positiivinen korrelaatio.

Korrelaation kahden vektorin välillä voi R:ssä laskea funktiolla `cor`. Otetaan esimerkiksi R:n sisäinen aineisto `Indometh`, jossa on mitattu indometasiinin farmakokinetiikkaa, ja selvitetään ajan ja indometasiinin konsentraation väliselle riippuvuudelle Pearsonin ja Spearmanin korrelaatiokertoimet. Piirretään sen jälkeen hajontakuvio mittaustuloksista ja lisätään kuvaajaan alaotsikoksi korrelaatiokertoimet. Tutustumme samalla funktioon `round`, jolla voi pyöristää lukuja halutulle desimaalitarkkuudelle. Huomaa, että `round`-funktio pyöristää aina lähimpään parilliseen lukuun, esim. luku 0.5 pyöristyy lukuun 0, mutta 1.5 pyöristyy lukuun 2. Funktio `mtext` lisää tekstin kuvaajan marginaaliin.

```
# Pearson correlation
pearson <- cor(Indometh$time, Indometh$conc, method = "pearson")
# Spearman correlation
spearman <- cor(Indometh$time, Indometh$conc, method = "spearman")
# Scatter plot
plot(
  Indometh$time,
  Indometh$conc,
  xlab = "Time",
  ylab = "Concetration",
  main = "Pharmacokinetics of indometacin"
)
```

```r
# Paste concatenates strings
subtitle <- paste(
  "Pearson correlation:", round(pearson, digits = 2),
  "Spearman correlation:", round(spearman, digits = 2)
)
# Add subtitle to plot
mtext(subtitle)
```



Tässä esimerkissä nähdään hyvin Pearsonin ja Spearmanin korrelaatiokertoimien ero. Koska Indometasiinin konsentraatio laskee eksponentiaalisesti, ei lineaarisesti, Pearsonin korrelaatiokerroin on "vain" -0.7, kun taas Spearmanin korrelaatiokerroin -0.97 vastaa lähes täydellistä negatiivista korrelaatiota.

4.3 Yhteenveto aineistosta (summary)

Kätevä tapa saada nopea yhteenveto datakehikon kaikista muuttujista on soveltaa `summary`-funktiota datakehikkoon.

```r
# Calculate summary for ToothGrowth data
summary(ToothGrowth)
```

```
      len        supp         dose
 Min.   : 4.20   OJ:30   Min.   :0.500
 1st Qu.:13.07   VC:30   1st Qu.:0.500
```

```
 Median :19.25           Median :1.000
 Mean   :18.81           Mean   :1.167
 3rd Qu.:25.27           3rd Qu.:2.000
 Max.   :33.90           Max.   :2.000
```

`summary` huolii myös yksittäisen vektorin, jolloin yhteenveto tulostuu vaakasuuntaisena.

```
tooth_length <- ToothGrowth$len
summary(tooth_length)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   4.20   13.07   19.25   18.81   25.27   33.90
```

4.4 Uniikit arvot

Usein on tarpeen tietää jonkin tarkasteltavan muuttujan saamat uniikit arvot aineistossa, esimerkiksi kun halutaan määrittää faktorin tasot. Tätä varten R:ssä on funktio `unique`, joka kertoo vektorin uniikit arvot.

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

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

Uniikkien arvojen lukumäärän saa helposti käyttämällä lisäksi `length` funktiota

```
length(unique(x))
```

```
[1] 5
```

Vektorissamme `x` oli siis 5 uniikkia arvoa.

4.5 Tunnuslukujen laskeminen ryhmittäin

Jos kiinnostuksen kohteena on vertailla ryhmiä toisiinsa esimerkiksi keskiarvon suhteen, on keskiarvot laskettava joka ryhmälle. Tämä voidaan tehdä esimerkiksi `tapply` funktiolla:

```
tapply(ToothGrowth$len, ToothGrowth$dose, mean)
```

```
   0.5      1      2
10.605 19.735 26.100
```

Funktion ensimmäinen argumentti vektori vastemuuttujan arvoista, joista olemme kiinnostuneita (tässä tapauksessa R:n sisäisen aineiston `ToothGrow` hampaiden pituuskasvumittaukset `len`). Toinen argumentti on vektori, joka kertoo mihin ryhmään kukin ensimmäisen argumentin arvoista kuuluu (tässä tapauksessa C-vitamiiniannos `dose`, annoskoot: 0.5, 1, ja 2 mg/vuorokausi). Kolmas argumentti määrittää funktion, jota sovelletaan joka ryhmässä erikseen (tässä tapauksessa keskiarvofunktio `mean`, mutta tämä voi olla mikä tahansa tunnusluku). Lasketut tunnusluvut palautetaan vektorina ryhmiä vastaavassa järjestyksessä.

5 Datan lukeminen

Tässä luvussa tutustutaan datan sisään lukemiseen ja sisäänluetun datan tarkistamiseen. Tähän mennessä kaikki kurssilla käsitelty data on luotu R:ssä. Useimmiten R:llä käsiteltävä data on kuitenkin tallennettu tiedostoon, joka on luotu jollain ohjelmalla tai kirjattu esim. Excelissä.

Tässä luvussa esitellyt funktiot lukevat erilaisia tiedostoja, mutta kaikki palauttavat datakehikon. Datakehikko sopii aineiston käsittelyyn hyvin, sillä siihen voi tallentaa niin numeerisia kuin tekstimuotoisia muuttujia. Voit tarvittessa kerrata datakehikon toimintaa datakehikko-kappaleesta.

Lopussa käydään myös läpi tapoja lukea Excel-, SPSS- ja SAS-tiedostoja. Näitä tiedostoja ei käsitellä kurssin tehtävissä, mutta on hyvä tietää, että niitä voi lukea R:ään suoraan muuttamatta niitä ensin johonkin toiseen muotoon.

5.1 Hakemistopolut ja tiedostopäätteet

5.1.1 Hakemistopolut

Jotta aineiston lataus tiedostosta onnistuu, tulee käyttäjän olla tietoinen siitä, missä hakemistopolussa eli kansiossa R työskentelee lataushetkellä. R:llä on siis koko ajan jokin hakemistopolku, johon se viittaa. R:n käyttämän hakemistopolun saat selville komennolla `getwd()`.

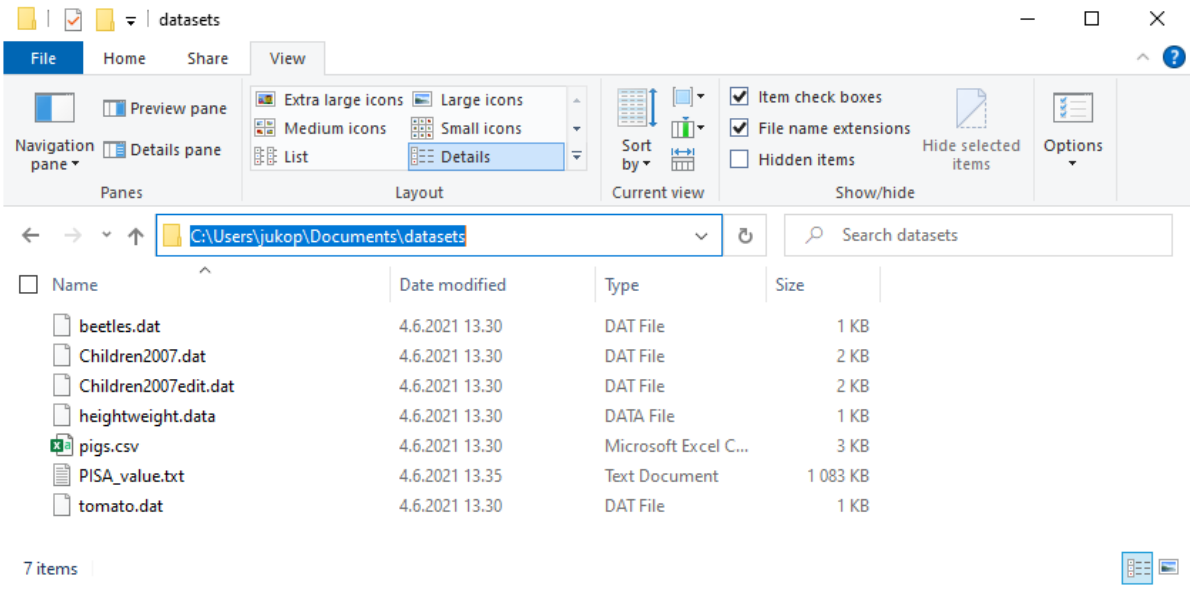
```
getwd()
```

```
[1] "C:/Users/jukop/Documents"
```

Tämän esimerkin tapauksessa R käyttää siis hakemistoa `C:/Users/jukop/Documents`. Jos kurssilla tarvittavan datasets.zip -tiedoston aineistot olisi purettu kansioon `C:/Users/jukop/Documents/datase` niin hakemistopolkju kannattaa vaihtaa juuri tähän hakemistoon. Se tapahtuu näin:

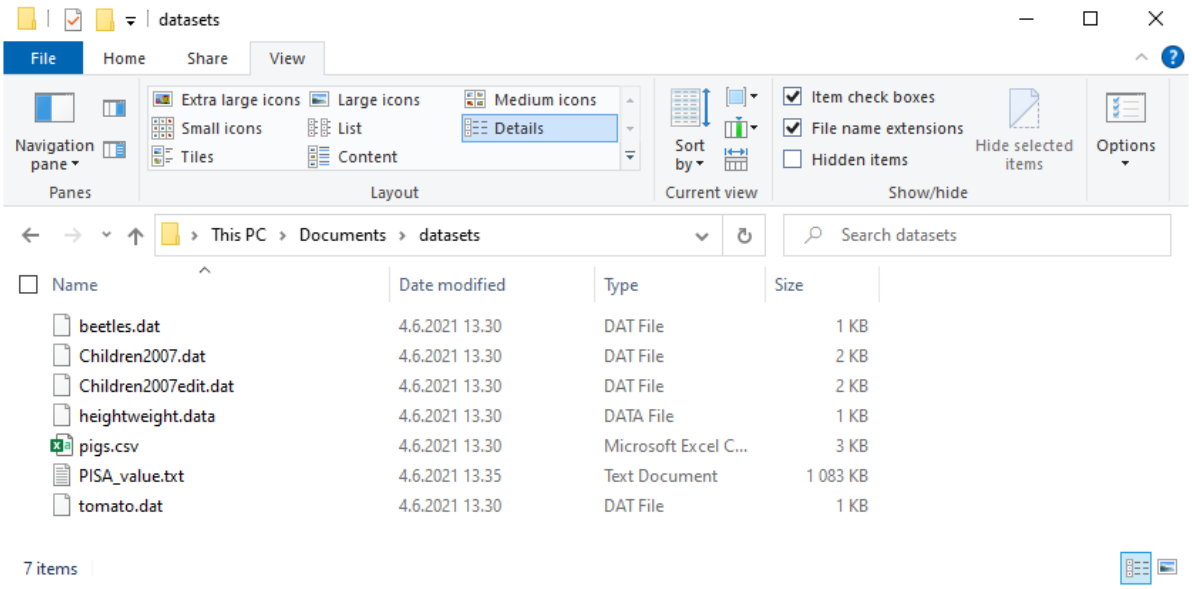
```
setwd("C:/Users/jukop/Documents/datasets")
```

`setwd` ei tulosta mitään, jos kansion vaihtaminen onnistuu. Komennolla `getwd()` voidaan uudelleen tarkastaa, että hakemisto todella vaihtui. ***Vinkki!*** Ellet tiedä mikä on tarkka hakemistopolku, johon olet purkanut tiedostot, niin se onnistuu klikkaamalla Windowsissa tiedostoselaimen osoiteriviä. Voit kopioida hakemistopolun siitä, mutta vaihda kuitenkin kenoviivat (\) kauttaviivoiksi (/). Kenoviivoilla on R:ssä erityismerkitys merkkijonoissa, joten ne eivät kelpaa sellaisenaan. Kaksinkertainen kenoviiva (\\) toimisi myös.



5.1.2 Tiedostopäätteet

Windows ei oletuksena nykyisin näytä tiedostopäätteitä. Ne kannattaakin asettaa näkymään tiedostoselaimen avulla. Kyseinen asetus löytyy tiedostoselaimen View-välilehdeltä kohdasta Show/Hide valinta File extensions. Merkitse kyseinen kohta valituksi, jolloin näet tiedostopäätteet, kuten kuvassa. Nyt on helppoa käsittää, kun opettaja puhuu CSV-tiedostoista, että niiden tiedostopääte on .csv.



5.2 Tekstitiedostot

Tekstitiedosto tarkoittaa tässä tapauksessa tiedostoa, joka ei sisällä tekstin lisäksi mitään muuta, kuten erilaisia muotoilutietoja. Tekstitiedostojen yleisimmät tiedostopäätteet ovat .txt ja .csv (comma separated value). Esim. Excelin .xlsx-tiedostot tai Wordin .docx-tiedostot eivät ole tekstitiedostoja, koska niissä on paljon muutakin tietoa tekstin lisäksi.

5.2.1 read.table

Kun dataa tallennetaan tekstitiedostoon, tiedoston ensimmäisellä rivillä ovat usein sarakkeiden nimet, ja seuraavilla riveillä mahdollisesti rivin nimi, ja sitten sarakkeiden arvot. Jokaisen kentän tulee olla erotettu samalla merkillä (field separator character). Yleisiä erotinmerkkejä ovat sarkain eli tab, välilyönti ja pilkku. Alla olevassa esimerkissä on neljältä kuvitteelliselta koehenkilöltä mitattu puna-vihervärisokeuteen liitettyjen geenien OPN1LW ja OPN1MW ilmentymistasot (lukuarvot ovat allekirjoittaneen hihasta). Tässä eri arvot on erotettu sarkaimella.

```
Subject_ID  OPN1LW  OPN1MW
ANKL    11264   12365
DIPR    10636   12725
PEPA    5630    13248
BRWA    8294    13060
```

Tämä data löytyy myös oheisesta tiedostosta `gene_data.txt`. Tekstitiedostot voi lukea sisään funktiolla `read.table`, jolla on tiedoston polun (file path) lisäksi monta muutakin argumenttia, joista tärkeimmät ovat:

- `header`: looginen arvo (`TRUE`/`FALSE`), jolla kerrotaan funktiolle, onko ensimmäisellä rivillä sarakkeiden nimet vai ei.
- `sep`: erotinmerkki, jolla muuttujien arvot on eroteltu.
- `dec`: desimaalierotin eli desimaalilukujen merkki, jolla desimaalit on eroteltu. Tämä on tärkeä lähinnä suomalaisille, koska Suomessa desimaalierotin on jostain syystä pilkku, eikä piste kuten useimmissa muissa maissa.

Luetaan edellisen esimerkin data R:ään datakehikoksi hakemistosta `data`:

```
gene_data <- read.table("data/gene_data.txt", header = TRUE)
gene_data
```

```
  Subject_ID OPN1LW OPN1MW
1       ANKL  11264  12365
2       DIPR  10636  12725
3       PEPA   5630  13248
4       BRWA   8294  13060
```

Yllä olevassa esimerkissä ei määritelty erikseen erotinmerkkiä, jolloin erotinmerkiksi tulkitaan kaikki tyhjä tila (white space) eli välilyönnit, sarkaimet jne. Halutessaan erotinmerkin voi myös asettaa. Jos erotinmerkki on sarkain, tulee asettaa `sep = "\t"`

```
gene_data <- read.table("data/gene_data.txt", sep = "\t", header = TRUE)
gene_data
```

```
  Subject_ID OPN1LW OPN1MW
1       ANKL  11264  12365
2       DIPR  10636  12725
3       PEPA   5630  13248
4       BRWA   8294  13060
```

Kuten yllä huomattiin, sarkain erotinmerkkinä merkataan `"\t"`, eikä lainausmerkeillä, joiden sisään laitettaisiin tyhjää tilaa sarkainnäppäimellä. Tämä on yksi esimerkki koodinvaihtomerkin (escape character) `\` käytöstä. R:ssä ja ohjelmointikielissä ylipäätään kenoviiva toimii koodinvaihtomerkkinä, eli sitä ei käsitellä kuin muita merkkejä, vaan se muuttaa seuraavan merkin toimintaa. Usein tämä tarkoittaa sitä, että kenoviivan avulla merkataan sarkainta, rivinvaihtoa (newline, `\n`) ja muita erikoismerkkejä. Koodinvaihtomerkin

käyttöä ei tarvitse osata tämän enempää, mutta se esitellään tässä, koska se aiheuttaa ongelmia Windowsin käyttäjille.

Windowsin tiedostopoluissa kansioiden välissä on kenoviiva, kun taas Mac- ja Linux-käyttöjärjestelmissä käytetään kauttaviivaa `/`. Koska R:ssä kenoviiva on koodinvaihtomerkki, niin helpoin tapa on käyttää tiedostopoluissa Macin ja Linuxien tyyliä. Jos taas halutaan lukea tiedosto R:ään käyttäen Windowsin tapaisia tiedostopolkuja, kenoviivat `\` pitää kirjoittaa kahteen kertaan eli `\\`, jotta R tulkitsee polun oikein. Tällöin ensimmäinen kenoviiva kertoo, että toinen kenoviiva on aito kenoviiva, eikä koodinvaihtomerkki.

Luetaan seuraavaksi sisään data-hakemistossa oleva tiedosto `tooth_growth.csv`, joka sisältää dataa tutkimuksesta C-vitamiinin vaikutuksesta hampaiden kasvuun marsuilla. .csv-tiedostopääte tulee sanoista comma separated value, eli tiedostossa arvot ovat eroteltu pilkulla. Asetetaan siis `sep`-argumentiksi `","`. Tämä tiedosto sisältää myös rivien nimet ensimmäisessä sarakkeessa. Tämä voidaan kertoa `read.table`-funktiolle argumentilla `row.names`, jonka arvoksi voi asettaa sarakkeen numeron, josta rivien nimet napataan.

```
tooth <- read.table("data/tooth_growth.csv", header = TRUE, sep = ",", row.names = 1)
tooth
```

```
    len supp dose
34  9.7   OJ  0.5
16 17.3   VC  1.0
55 24.8   OJ  2.0
44 26.4   OJ  1.0
58 27.3   OJ  2.0
26 32.5   VC  2.0
14 17.3   VC  1.0
60 23.0   OJ  2.0
15 22.5   VC  1.0
9   5.2   VC  0.5
```

Tutkimuksessa marsuille annettiin C-vitamiinia eri annoksina (dose, mitattu milligrammoina), joko appelsiinimehussa (OJ) tai askorbiinihappona (VC), ja mitattiin odontoblastien (hammasluun emosolu) pituus (len).

5.2.2 read.csv

Tiedostot, joissa arvot ovat pilkulla eroteltuina ovat niin yleisiä, että niiden lukemiseen on oma funktio: `read.csv`, joka on käytännössä sama funktio kuin `read.table`, mutta parametrien oletusarvot ovat erilaiset, niin että `read.csv(file)` ~ `read.table(file, header = TRUE, sep = ","))`.

```r
tooth <- read.csv("data/tooth_growth.csv", row.names = 1)
tooth
```

```
    len supp dose
34  9.7   OJ  0.5
16 17.3   VC  1.0
55 24.8   OJ  2.0
44 26.4   OJ  1.0
58 27.3   OJ  2.0
26 32.5   VC  2.0
14 17.3   VC  1.0
60 23.0   OJ  2.0
15 22.5   VC  1.0
9   5.2   VC  0.5
```

5.2.2.1 read.csv2

HUOM: Koska Suomessa pilkkua käytetään desimaalierottimena, kenttien rajaaminen pilkulla ei toimi. Käytännössä tämä näkyy siten, että suomenkielinen Excel tallentaa .csv-tiedosto oletuksena muodossa, jossa desimaalierottimena on pilkku ja kenttien välissä puolipiste `;`. Jos siis olet tallentanut Excelistä taulukon .csv-muotoon ja sen lukeminen R:ään aiheuttaa hankaluuksia, kyse on todennäköisesti erotinmerkistä. Onneksi R:ssä on valmiina funktio `read.csv2`, joka osaa lukea puolipisteelliset .csv-tiedostot oikein.

5.3 Datakehikon tarkastelu

Kun data on luettu sisään R:ään, kannattaa aina tarkistaa, että kaikki data on luettu oikein. Tässä muutama vinkki datakehikon tutkimiseen, joista osaa käsiteltiin jo datakehikko-kappaleessa:

`dim` antaa datakehikon dimensiot, eli rivien ja sarakkeiden määrän.
`View` avaa datakehikon erilliseen ikkunaan, jossa sitä voi tarkastella. Suositellaan vain pienemmille datakehikoille `str` kertoo rivien ja sarakkeiden määrät sekä kaikkien sarakkeiden luokat. Kätevä tapa tarkistaa mm. että lukuja sisältävät sarakkeet eivät ole vahingossa muuttuneet merkkijonoiksi. `table` on kätevä kategoristen sarakkeiden tutkimiseen. Se kertoo, kuinka monta havaintoa muuttujan arvoilla on. `table` voi ottaa vastaan myös kaksi kategorista muuttujaa, ja laskee jokaiselle muuttujien arvojen yhdistelmälle havaintojen lukumäärän.

Katsotaan, mitä `str` kertoo juuri lukemastamme tooth-datasta.

```
str(tooth)
```

```
'data.frame':   10 obs. of  3 variables:
 $ len : num  9.7 17.3 24.8 26.4 27.3 32.5 17.3 23 22.5 5.2
 $ supp: chr  "OJ" "VC" "OJ" "OJ" ...
 $ dose: num  0.5 1 2 1 2 2 1 2 1 0.5
```

Kuten näimme aiemmin, mukana on 10 havaintoa ja 3 muuttujaa. `len` ja `dose` ovat luokkaa `numeric` eli desimaalilukuja, ja `supp` on luokkaa `factor`, eli faktori. Faktoritietotyyppiä käsitellään enemmän lineaaristen mallien yhteydessä, mutta sillä merkitään usein kategorisia muuttujia.

Lasketaan seuraavaksi, kuinka monelle marsulle annettiin appelsiinimehua ja kuinka monelle askorbiinihappoa.

```
table(tooth$supp)
```

```
OJ VC
 5  5
```

Kumpaakin annostelutapaa käytettiin siis viisi kertaa. Voimme myös selvittää, miten eri annokset jakautuvat annostelutavan suhteen:

```
table(tooth$supp, tooth$dose)
```

```
    0.5 1 2
  OJ   1 1 3
  VC   1 3 1
```

Appelsiinimehuna annettiin siis 0.5 mg ja 1 mg annoksia kumpaakin 1 kappale, ja 2 mg annoksia 3 kappaletta.

5.3.1 R:n sisäänrakennetut aineistot

R:ssä on monta sisäänrakennettua aineistoa. Näitä on kätevää käyttää nopeaan testaamiseen ja ne vilahtelevatkin usein R-oppaissa. Esimerkiksi aikaisempi odontoblastien pituuksia sisältävä aineistomme on oikeastaan pieni otos R:n sisäisestä aineistotsa `ToothGrowth`.

R:n sisäiset aineistot ovat koko ajan käytettävissä, vaikka ne eivät näy RStudion ympäristössä (Environment). Voimme esimerkiksi katsoa, millainen rakenne kokonaisella `ToothGroth`-datalla on:

```
str(ToothGrowth)
```

```
'data.frame':   60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
 $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

R:n aineistoja voi käyttää moneen eri tarkoitukseen, kuten datan visualisoinnin tai tilastollisten toimenpiteiden testaamiseen. Listan kaikista R:n sisäisitä aineistoista saa komennolla `data()`. Tarkempia tietoja yksittäisistä aineistoista saa help-sivulta kuten funktioden tapauksessa, esimerkiksi `?ToothGrowth`

5.4 Muut tiedostot

5.4.1 Excel

Excelin käyttämiä .xlsx-tiedostoja voi lukea suoraan R:ään, vaikka yleensä ne suositellaan muuntamaan ensin .csv-muotoon. Suoraa lukemista varten pitää asentaa **readxl**-paketti, minkä voi tehdä RStudion Packages-valikoksta tai suoraan komennolla `install.packages("readxl")`. Paketin funktiolla `read_xlsx()` voi lukea sisään .xlsx-tiedostoja, tai yksikkäitisä taulukon sivuja. Excel-tiedostojen kirjoittamiseen löytyy myös vastaava paketti **writexl**.

Vaihtoehtoinen paketti Excel-tiedostojen lukemiseen on **openxlsx**, jolla voi sekä lukea että kirjoittaa .xlsx-tiedostoja, mutta se on tyypillisesti hitaampi *readxl* ja *writexl* paketteihin verattuna.

5.4.2 SPSS

Eri tutkimusryhmissä dataa säilytetään usein SPSS-tiedostoissa (.sav). SPSS-tiedostojen käsittelyyn voi käyttää **haven**-paketin funktioita `read_sav` ja `write_sav`. **haven**-paketti sisältää myös funktiot Stata- ja SAS-tiedostoille.

6 Datan muokkaaminen

Aineisto ei tyypillisesti ole valmiiksi oikeassa muodossa. Voi olla, että halutaan esimerkiksi käyttää vain jotain osajoukkoa aineistosta tai muodostaa uusia muuttujia analyysia varten. Tällöin tarvitaan komentoja aineiston muokkaamiseksi.

Yleinen käytännön vinkki: Aineiston muokkaaminen (data wrangling) on isojen tutkimusaineistojen kohdalla todella työlästä. Tällöin saatetaan joutua yhdistelemään aineistoja useista lähteistä, etsimään virheellisiä arvoja, muokkaamaan tekstimuotoisia (character) muuttujia eri muotoon ym. Mikäli halutaan muokata tekstimuotoisia vektoreita eri muotoon, niin ne kannattaa muuttaa faktoriksi vasta lopuksi, sillä muuttujan ei ole yleensä tarpeellista olla faktorimuodossa aineistoja muokatessa. Faktorit ovat tyypillisesti tarpeen vasta kun aineistoa aletaan todella analysoimaan.

6.1 Uuden muuttujan tai rivin luonti datakehikkoon

Uusi muuttuja voidaan luoda R:ssä joko perustuen aineiston muihin muuttujiin, tai muuttujan arvot voidaan syöttää vektorina aineistoon. Mikäli uusi muuttuja syötetään lukuina R-koodiin, tulee varmistua siitä, että havaintoja on sama määrä kuin aineistossa on rivejä. Muutoin aineisto tulee syötettyä virheellisesti ja tulokset eivät pidä paikkaansa.

Uuden sarakkeen luonti tapahtuu samalla tavalla kuin jo olemassa olevan sarakkeen muokkaaminen eli dollarisymbolilla, jossa dollarin jälkeen annetaan ensin uuden sarakkeen nimi ja tähän sijoitetaan halutut uuden muuttujan arvot.

```
study_data <- read.table("data/study_data.txt")

# evaluate the number of rows and columns
dim(study_data)
```

```
[1] 8 3
```

```r
# there are 8 rows

# initiate a new variable called weight (imput data) with correct number of rows
study_data$weight <- c(78.2, 65.8, 49.2, 71.2, 58.3, 54.1, 74.2, 62.8)

# calculate a new variable based on existing variables
study_data$height_m <- study_data$height / 100 # height as meters
study_data$BMI <- study_data$weight / (study_data$height_m^2)
study_data
```

```
  ID height gender weight height_m        BMI
1  1  189.8   male   78.2    1.898   21.70773
2  2  184.0 female   65.8    1.840   19.43526
3  3  173.8   male   49.2    1.738   16.28792
4  4  175.9   male   71.2    1.759   23.01168
5  5  169.0 female   58.3    1.690   20.41245
6  6  183.7   male   54.1    1.837   16.03168
7  7  181.8   male   74.2    1.818   22.44999
8  8   16.9 female   62.8    0.169 2198.80256
```

6.2 Datakehikon käsittely

Datakehikosta voidaan poimia sarakkeita joko niiden nimien tai niitä vastaavien indeksien perusteella, kuten matriisin tapauksessa. Yksittäisiä sarakkeita voidaan poimia ja muokata myös dollarisymbolin $ kautta.

```r
# Subscripting with variable names
study_data[, c("height", "gender")]
```

```
  height gender
1  189.8   male
2  184.0 female
3  173.8   male
4  175.9   male
5  169.0 female
6  183.7   male
7  181.8   male
8   16.9 female
```

```r
# Subscripting with brackets - as matrix (but I do not recommend this style!)
study_data[, 1:2]
```

```
  ID height
1  1  189.8
2  2  184.0
3  3  173.8
4  4  175.9
5  5  169.0
6  6  183.7
7  7  181.8
8  8   16.9
```

```r
# Rownames and colnames
colnames(study_data)
```

```
[1] "ID"       "height"   "gender"   "weight"   "height_m" "BMI"
```

```r
names(study_data)
```

```
[1] "ID"       "height"   "gender"   "weight"   "height_m" "BMI"
```

```r
# Individual columns can be accessed and added with dollar sign
# Let's say that we find out that the ID number 8 was typed in incorrectly. We can fix the er
study_data$height <- c(189.8, 184.0, 173.8, 175.9, 169.0, 183.7, NA, 160.9)
study_data
```

```
  ID height gender weight height_m        BMI
1  1  189.8   male   78.2    1.898   21.70773
2  2  184.0 female   65.8    1.840   19.43526
3  3  173.8   male   49.2    1.738   16.28792
4  4  175.9   male   71.2    1.759   23.01168
5  5  169.0 female   58.3    1.690   20.41245
6  6  183.7   male   54.1    1.837   16.03168
7  7     NA   male   74.2    1.818   22.44999
8  8  160.9 female   62.8    0.169 2198.80256
```

```r
# It would have been possible to change value of only one cell e.g. like this
study_data$height[8] <- 161.9
study_data
```

```
  ID height gender weight height_m        BMI
1  1  189.8   male   78.2    1.898   21.70773
2  2  184.0 female   65.8    1.840   19.43526
3  3  173.8   male   49.2    1.738   16.28792
4  4  175.9   male   71.2    1.759   23.01168
5  5  169.0 female   58.3    1.690   20.41245
6  6  183.7   male   54.1    1.837   16.03168
7  7     NA   male   74.2    1.818   22.44999
8  8  161.9 female   62.8    0.169 2198.80256
```

Uuden rivin lisäys datakehikkoon on hieman monimutkaisempaa kuin uuden rivin lisääminen matriisiin, sillä ensin pitää tehdä uusi datakehikko, jolla on samat sarakkeet kuin alkuperäisellä (samassa järjestyksessä), ja vasta sitten liittää se komennolla `rbind`. Käyttäjän tulee myös huolehtia siitä, että sarakkeet ovat samaa tyyppiä kuin alkuperäisessä datakehikossa.

```r
new_row <- data.frame(
  ID = 11,
  height = 182,
  gender = "male",
  weight = 81.2,
  height_m = 1.82,
  BMI = 81.2 / 1.82^2
)
rbind(study_data, new_row)
```

```
   ID height gender weight height_m        BMI
1   1  189.8   male   78.2    1.898   21.70773
2   2  184.0 female   65.8    1.840   19.43526
3   3  173.8   male   49.2    1.738   16.28792
4   4  175.9   male   71.2    1.759   23.01168
5   5  169.0 female   58.3    1.690   20.41245
6   6  183.7   male   54.1    1.837   16.03168
7   7     NA   male   74.2    1.818   22.44999
8   8  161.9 female   62.8    0.169 2198.80256
11 11  182.0   male   81.2    1.820   24.51395
```

6.3 Osajoukon valinta

Aineistosta voi poimia osajoukon hakasulkujen avulla indeksoimalla. Osajoukon poimintaan tarvitaan usein vertailuoperattoreita, ja jos kriteerejä on useita, niin tarvitaan myös useita loogisia operaattoreita. Tarkemmin operaattoreita käsitellään luvussa Loogiset operaattorit. Voit käyttää kyseisen osion taulukkoa apuna jo tässä osiossa. Osajoukkoja voidaan poimia myös suoraan antamalla halutut indeksit esimerkiksi indeksivektorin avulla.

```
# Filter only females
study_data[study_data$gender == "female", ]
```

```
  ID height gender weight height_m        BMI
2  2  184.0 female   65.8    1.840   19.43526
5  5  169.0 female   58.3    1.690   20.41245
8  8  161.9 female   62.8    0.169 2198.80256
```

```
# Filter individuals whose height is less than or equal to 175
study_data[study_data$height <= 175, ]
```

```
   ID height gender weight height_m        BMI
3   3  173.8   male   49.2    1.738   16.28792
5   5  169.0 female   58.3    1.690   20.41245
NA NA     NA   <NA>     NA       NA         NA
8   8  161.9 female   62.8    0.169 2198.80256
```

```
# Filter individuals whose height is not missing and is less than or equal to 175
study_data[!is.na(study_data$height) & study_data$height <= 175, ]
```

```
  ID height gender weight height_m        BMI
3  3  173.8   male   49.2    1.738   16.28792
5  5  169.0 female   58.3    1.690   20.41245
8  8  161.9 female   62.8    0.169 2198.80256
```

```
# Use multiple filter criteria
study_data[study_data$height <= 175 & study_data$gender == "female", ]
```

```
  ID height gender weight height_m        BMI
5  5  169.0 female   58.3    1.690   20.41245
8  8  161.9 female   62.8    0.169 2198.80256
```

```r
# Select individuals (rows) 1,3, and 7 directly with a vector of indices
ind <- c(1, 3, 7)
study_data[ind,]
```

```
  ID height gender weight height_m      BMI
1  1  189.8   male   78.2    1.898 21.70773
3  3  173.8   male   49.2    1.738 16.28792
7  7     NA   male   74.2    1.818 22.44999
```

6.4 Datakehikon ja vektorin järjestäminen

Yhden vektorin arvot voidaan asettaa nousevaan tai laskevaan järjestykseen funktiolla `sort`. Funktiota voidaan soveltaa niin numeerisiin kuin merkkitietoa sisältäviin vektoreihinkin. Oletusarvoisesti järjestys on nouseva, eli numeeriset arvot järjestetään pienimmästä suurimpaan ja merkkitieto aakkosjärjestykseen. Järjestyksen voi kääntää laskevaksi argumentilla `decreasing = TRUE`. Huomaa, että ääkkösten tapauksessa `sort` ei välttämättä aina järjestä alkioita oikein merkistöstä riippuen.

```r
nums <- c(3, 1, 7, 8, 5, 4)
chars <- c("ab", "ca", "ac", "bb", "ba", "cb")

# Ascending order
sort(nums)
```

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

```r
sort(chars)
```

```
[1] "ab" "ac" "ba" "bb" "ca" "cb"
```

```r
# Descending order
sort(nums, decreasing = TRUE)
```

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

```r
sort(chars, decreasing = TRUE)
```

```
[1] "cb" "ca" "bb" "ba" "ac" "ab"
```

Jossain tilanteissa on haluttavaa järjestää datakehikon rivit jonkin muuttujan tai muuttujien suhteen. Tähän tarkoitukseen voi käyttää funktiota `order`, joka palauttaa yhden tai useamman argumentin alkioiden järjestysluvut (rank). Seuraavassa esimerkissä aineiston rivit järjestetään koehenkilöiden pituuden suhteen nouseevaan suuruusjärjestykseen.

```
study_data[order(study_data$height), ]
```

```
  ID height gender weight height_m        BMI
8  8  161.9 female   62.8    0.169 2198.80256
5  5  169.0 female   58.3    1.690   20.41245
3  3  173.8   male   49.2    1.738   16.28792
4  4  175.9   male   71.2    1.759   23.01168
6  6  183.7   male   54.1    1.837   16.03168
2  2  184.0 female   65.8    1.840   19.43526
1  1  189.8   male   78.2    1.898   21.70773
7  7     NA   male   74.2    1.818   22.44999
```

Järjestäminen voidaan tehdä usean muuttujan suhteen, esimerkiksi pituuden ja painon. Tämä tarkoittaa nousevassa järjestyksessä sitä, että jos kahdella koehenkilöllä on täsmälleen sama pituus, valitaan heidän keskinäinen järjestyksensä painon perusteella. Huomaa `sort`- ja `order`-funktioiden ero: `sort` palauttaa suoraan järjestetyn vektorin kun taas `order` alkioiden järjestysluvut.

6.5 Faktorit

R:n numeeriset vektorit ovat lähtökohtaisesti välimatka- tai suhdeasteikollisia. Olet ehkä ihmetellytkin, miten luokitteluasteikollinen (kategorinen) tai järjestysasteikollinen muuttuja määritellään. Kategorista muuttujaa sanotaan R:ssä **faktoriksi**. Numeerisen tai tekstimuotoisen muuttujan tai vektorin voi muuttaa faktori-muotoiseksi muuttujaksi `factor` -funktiolla.

```
# Let's change gender from character string to a factor and rename it as fgender
study_data$fgender <- factor(study_data$gender)

# Let's now compare the printing of gender and fgender
study_data$gender
```

```
[1] "male"   "female" "male"   "male"   "female" "male"   "male"   "female"
```

```
study_data$fgender
```

```
[1] male   female male   male   female male   male   female
Levels: female male
```

Huomaa, että faktori tulostaa faktorin tasot eli kaikkien mahdollisten luokkien nimet faktorin perässä: `Levels: female male`.

Usein vastaan tulee myös tilanne, jossa faktorin eri tasoja vastaavat kokonaislukuarvot, kuten tässä esimerkissä luvut 1, 2 ja 3. Tällaisessa tilanteessa faktorin tasojen merkitys on usein annettu jossain dokumenttitiedostossa. Tällöin faktorin tasot ja niiden kuvaukset (labels) tulee määrittää käsin.

```
# Create a data for this example
wall_dat <- data.frame(
  building_ID = c(1, 2, 3, 4, 5, 6),
  building_material = c(1, 1, 2, 2, 3, 3)
)

# Name is 'building_material' very long, I want to rename it
names(wall_dat) <- c("building_ID", "build_mat")

# We know from some kind of documentation that 1 stands for "wood", 2 is "steel" and 3 is "br
wall_dat$fbuild_mat <- factor(
  wall_dat$build_mat,levels = c(1, 2, 3),
  labels = c("wood", "steel", "brick")
)

str(wall_dat)
```

```
'data.frame':   6 obs. of  3 variables:
 $ building_ID: num  1 2 3 4 5 6
 $ build_mat  : num  1 1 2 2 3 3
 $ fbuild_mat : Factor w/ 3 levels "wood","steel",..: 1 1 2 2 3 3
```

Faktorimuuttujan kuvaukset siis kertovat, mitä varsinaiset tasoarvot tarkoittavat.

6.6 Extra: Lääketutkimusesimerkki

R:ssä on aiemmin nähtyjen `numeric`-, `character`- ja `logical`-tyyppien lisäksi muitakin vektoriluokkia, joista tärkein on `factor` eli faktori. Faktoreihin tallennetaan kategorisia muuttujia, kuten tutkimuksessa määrättyjä ryhmiä, aikapisteitä tms. Luodaan esimerkiksi faktori, jossa on kuvitteellisen lääketutkimuksen osallistujien ryhmätiedot:

```
groups <- as.factor(
  c(
    "drug1", "drug2", "control", "drug1", "control",
    "drug2", "drug2", "control", "control", "drug1"
  )
)
groups
```

```
 [1] drug1   drug2   control drug1   control drug2   drug2   control control
[10] drug1
Levels: control drug1 drug2
```

Factoreita voi luoda muista vektoreista funktioilla `factor` tai `as.factor`. `as.factor` muuntaa vektorin automaattisesti ja nopeasti factoriksi, ja säilyttää myös jo valmiiksi faktoriluokan vektorien tasojen järjestyksen (tästä lisää pian).

Kuten tulosteesta nähdään, faktorin tulostus tulostaa faktorin alkiot (HUOM: ei lainausmerkkejä) sekä faktorin tasot. Faktorit ovat pinnan alla kokonaisluku- eli `integer`-vektoreita, joissa on päällä "kerros", joka määrittää factorin tasot. Edellä nähty vektori `groups` näyttää siis tältä:

Levels	drug1	drug2	control	drug1	control	drug2	drug2	control	control	drug1
Integers	2	3	1	2	1	3	3	1	1	2

Faktorien tasoille annetaan siis lukuarvot ykkösestä eteenpäin. Oletuksena ensimmäinen taso eli taso 1 on aakkosissa ensimmäinen arvo, tai pienin lukuarvo jos faktori tehdään numeerisista muuttujista. Lukuarvot saa näkyville muuntamalla factorin numeeriseksi vektoriksi:

```
as.numeric(groups)
```

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

Tasojen järjestyksen voi myös päättää itse. Tämä on tärkeää, sillä kuten pian nähdään, faktorin ensimmäinen taso on monissa tilastollisissa testeissä ns. referenssitaso, johon muita tasoja verrataan. Usein esiintyvä tapaus ovat tutkimukset, joissa ovat ryhmät nimeltä case ja control. Koska case on aakkosissa ennen controllia, R käyttää oletuksen case-ryhmää referenssitasona, ja testaa miten control-ryhmä poikkeaa tästä tasosta, vaikka haluaisimme päinvastaisen määrittelyn. Tasot voi itse määrittää näin:

```
study_groups <- factor(
  c("case", "control", "control", "case", "case"),
  levels = c("control", "case")
)
study_groups
```

```
[1] case    control control case    case
Levels: control case
```

Nyt tasot ovat oikeassa järjestyksessä!

Kuten aiemmin mainittiin, faktoreita voi tehdä myös numeerisista vektoreista. HUOM: muista, että `as.numeric` palauttaa faktorin kokonaislukuarvot, ei alkuperäisiä lukuja. Alkuperäiset luvut saa käyttämällä ensin `as.character`-funktiota, joka muuttaa faktorin tasot merkkijonovektoriksi.

```
time_points <- as.factor(c(0, 0, 1, 1, 5, 5, 1, 0, 5))
time_points
```

```
[1] 0 0 1 1 5 5 1 0 5
Levels: 0 1 5
```

```
# Probably not what you expect
as.numeric(time_points)
```

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

```
# First to character, then to numeric
as.numeric(as.character(time_points))
```

```
[1] 0 0 1 1 5 5 1 0 5
```

7 Kuvaajien piirtäminen

Tässä luvussa tutustutaan kuvaajien piirtämiseen. R:n piirtokomennot voidaan jakaa kolmeen ryhmään:

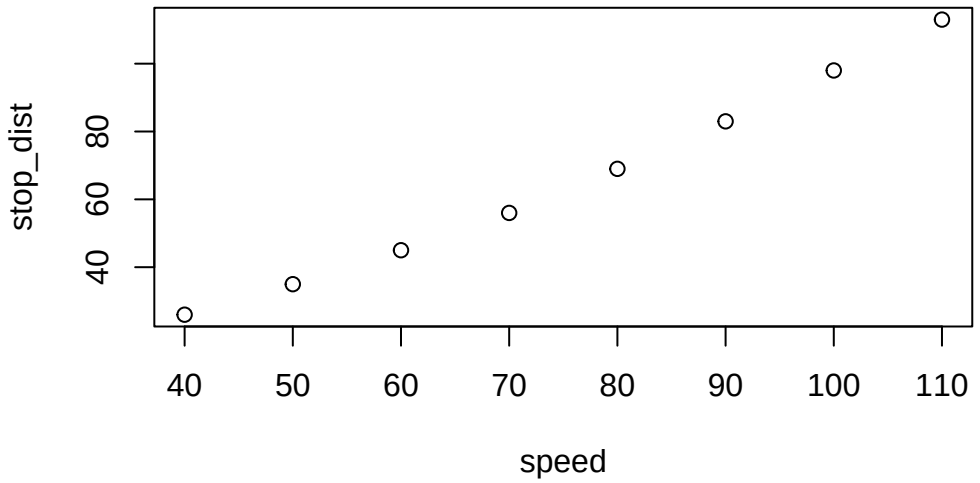
- Korkean tason grafiikkatoiminnot piirtävät aina uuden kuvan.
- Alemman tason grafiikkatoiminnot lisäävät olemassa olevaan kuvaan uusia osia.
- Interaktiiviset grafiikkatoiminnot mahdollistavat vuorovaikutuksen kuvan kanssa. (Näiden käyttö on helpompaa opettaa videolla, joten niitä ei käsitellä tässä).

7.1 Korkean tason piirtofunktiot

7.1.1 plot

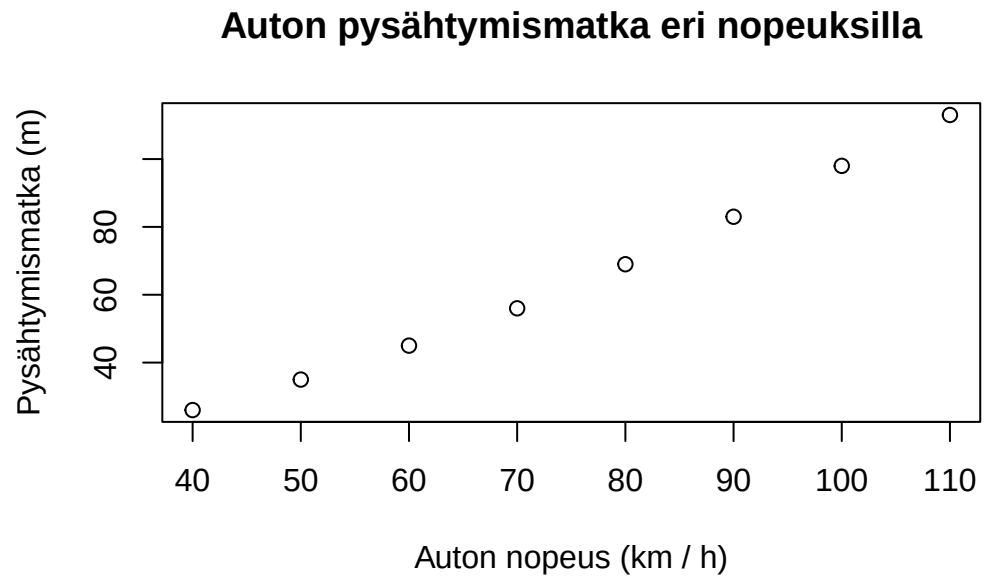
Korkean tason piirtofunktioista ylivoimaisesti yleisin on `plot`. `plot`-funktio on hyvin monipuolinen, mutta sen yleisin käyttötarkoitus on piirtää hajontakuvio (scatter plot) yhdestä tai kahdesta vektorista. Alla on hajontakuvio auton jarrutusmatkoista eri nopeuksilla:

```
# Car speeds (km/h)
speed <- seq(40, 110, by = 10)
# Stopping distances (m)
stop_dist <- c(26, 35, 45, 56, 69, 83, 98, 113)
# Draw the plot
plot(x = speed, y = stop_dist)
```



`plot`-funktiolle annetaan siis kaksi yhtä pitkää vektoria, joissa ovat pisteiden x- ja y-koordinaatit. Halutessaan kuvalle voi antaa otsikon (title) ja nimetä uudestaan kuvan akselit (axis labels). Tämä onkin usein hyvä idea, sillä R:n muuttujien nimissä ei saa olla välilyöntejä tai erikoismerkkejä, mutta usein näiden käyttö akselien nimissä on hyvin informatiivista.

```
plot(
  x = speed,
  y = stop_dist,
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
```

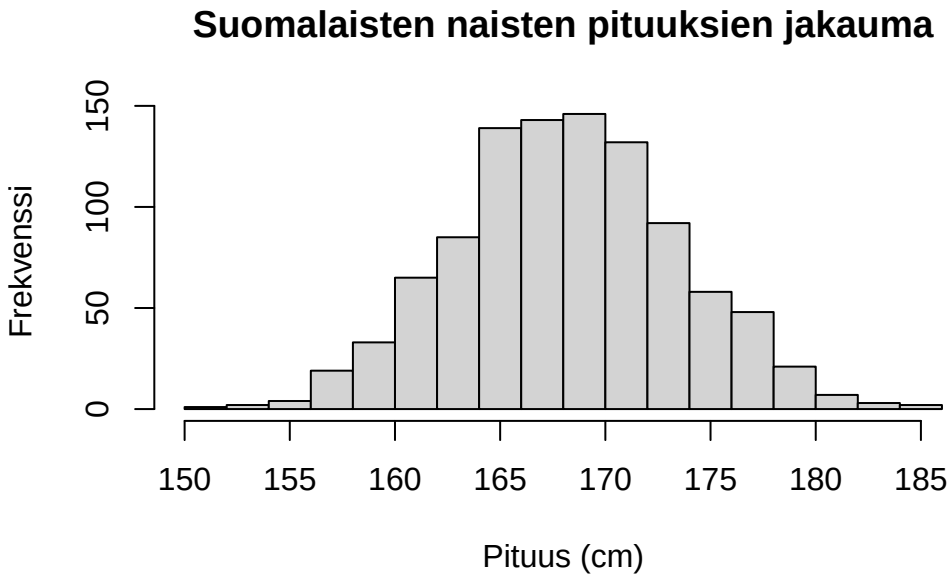


`plot`-funktiolle voi antaa muitakin argumentteja, jotka säätävät mm. pisteiden väriä, kokoa ja muotoa, akselien rajoja jne. Yleisiä kuvaajien parametreja voi säätää funktiolla `par` (graphical parameters).

7.1.2 hist

`hist` piirtää histogrammeja. Histogrammit kuvaavat jatkuvan muuttujan jakaumaa.

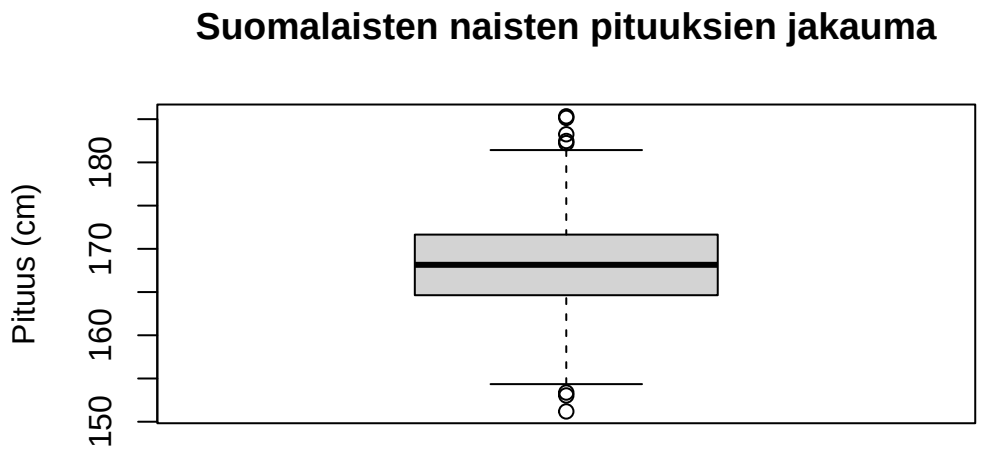
```
# A vector of 1000 observations from a normal distribution of heights of Finnish women
heights <- rnorm(n = 1000, mean = 168, sd = 5.4)
hist(
  heights,
  breaks = 20,
  main = "Suomalaisten naisten pituuksien jakauma",
  xlab = "Pituus (cm)",
  ylab = "Frekvenssi"
)
```



7.1.3 boxplot

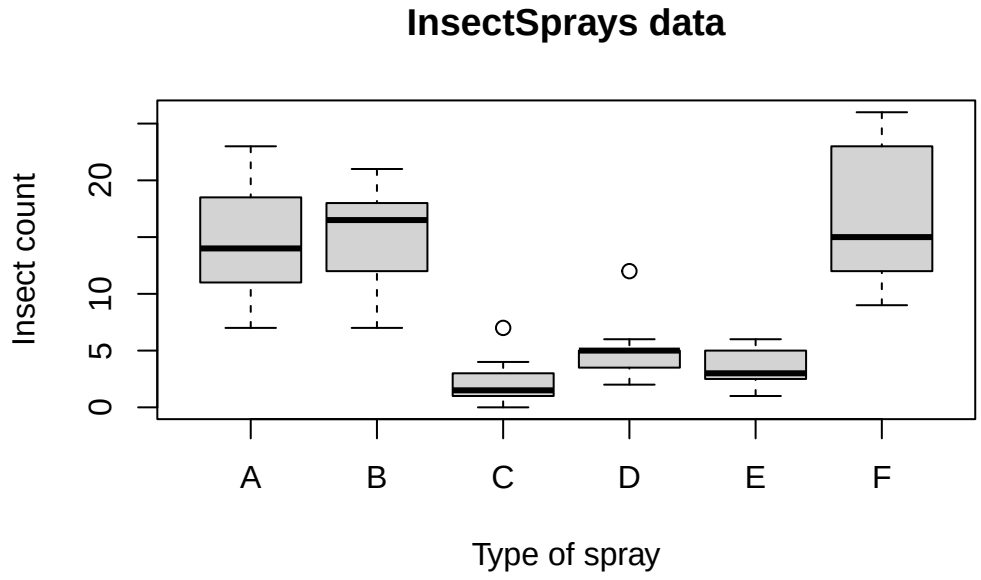
Toinen tapa kuvata jatkuvan muuttujan jakaumaa on viiksilaatikko (joskus myös laatikko-viikset -kuvaaja), joita piirretään `boxplot`-funktiolla:

```r
boxplot(
  heights,
  breaks = 20,
  main = "Suomalaisten naisten pituuksien jakauma",
  ylab = "Pituus (cm)"
)
```



Usein on hyödyllistä piirtää viiksilaatikko usealle ryhmälle samaan kuvaan. Tämä voidaan myös tehdä suoraan `boxplot`-funktiolla. Alla esimerkki R:n sisäisestä aineistosta `InsectSprays`, joka sisältää hyönteisten lukumääriä eri hyönteismyrkkykäsittelyjen jälkeen. Piirretään lukumäärien viiksilaatikot joka käsittelylle:

```r
boxplot(
  count ~ spray,
  data = InsectSprays,
  xlab = "Type of spray",
  ylab = "Insect count",
  main = "InsectSprays data"
)
```



Tällaisessa tilanteessa voimme siis käyttää samaa kaavasyntaksia (`formula`) kuin lineaarisen mallin tapauksessa. Kaavan vasen puoli kertoo vastemuuttujan (`count`) ja kaavan oikea puoli kertoo muuttujan, joka määrittää tarkasteltavat ryhmät (`spray`)

7.1.4 barplot

Vastaavasti diskreetin muuttujan jakaumaa voi kuvata pylväsdiagrammilla käyttäen `barplot`-funktiota. Alla on esimerkki opiskelijoiden kotipaikkakuntien jakaumasta. Tässä tulee myös tutuksi tärkeä vektorien ominaisuus: nimeäminen. Nimettyjen vektorien (named vectors) alkioilla on järjestyslukujen lisäksi nimet. Nimet annetaan olla olevaan tyyliin `nimi = alkio`. Nimetyt vektori käyttäytyvät aivan kuin tavalliset vektorit, mutta niitä voi indeksoida myös nimien avulla, ja jotkut funktiot, kuten `barplot`, käyttävät hyödyksi alkioiden nimiä. Nimettyjen vektorien käyttö ei ole kurssin ydinasioita, mutta tämä on hyödyllistä osata.

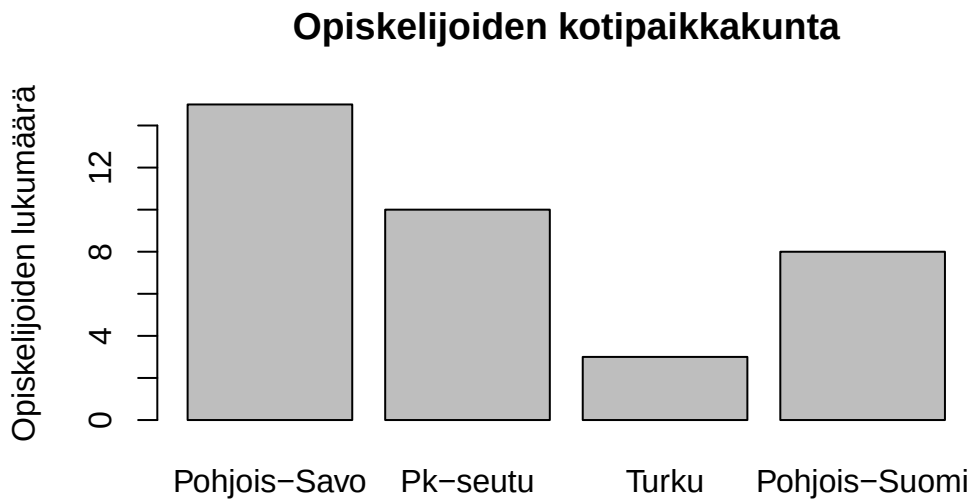
```
origin <- c(
  "Pohjois-Savo" = 15,
  "Pk-seutu" = 10,
  "Turku" = 3,
  "Pohjois-Suomi" = 8
)
origin
```

```
 Pohjois-Savo       Pk-seutu          Turku Pohjois-Suomi
           15             10              3              8
```

```r
origin["Turku"]
```

```
Turku
    3
```

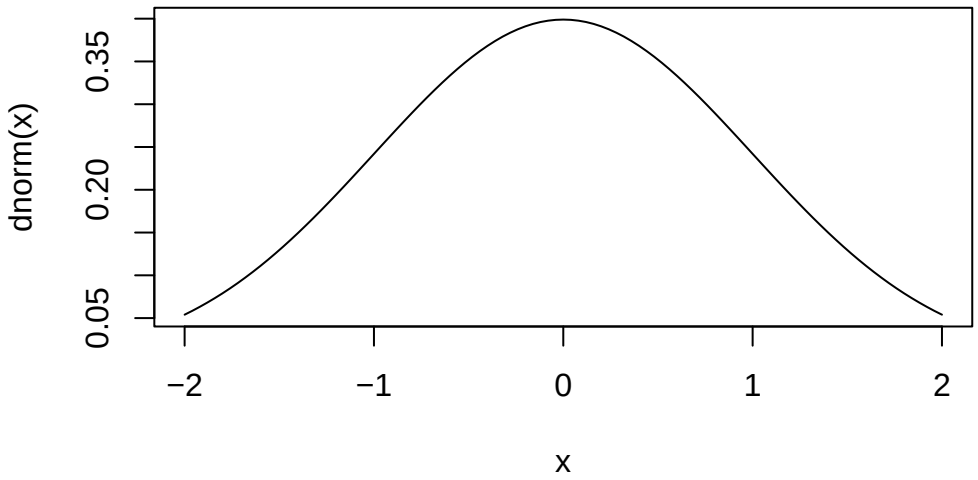
```r
barplot(
  origin,
  main = "Opiskelijoiden kotipaikkakunta",
  ylab = "Opiskelijoiden lukumäärä"
)
```



7.1.5 curve

Funktio `curve` on hyödyllinen matemaattisten funktioiden graafien piirtämiseen. Funktio ottaa syötteenään piirrettävän funktion, sekä tarkasteluvälin, jolla graafi piirretään. Piirretään esimerkiksi standardinormaalijakauman tiheysfunktio välillä (-2, 2):

```r
curve(expr = dnorm(x), from = -2, to = 2)
```



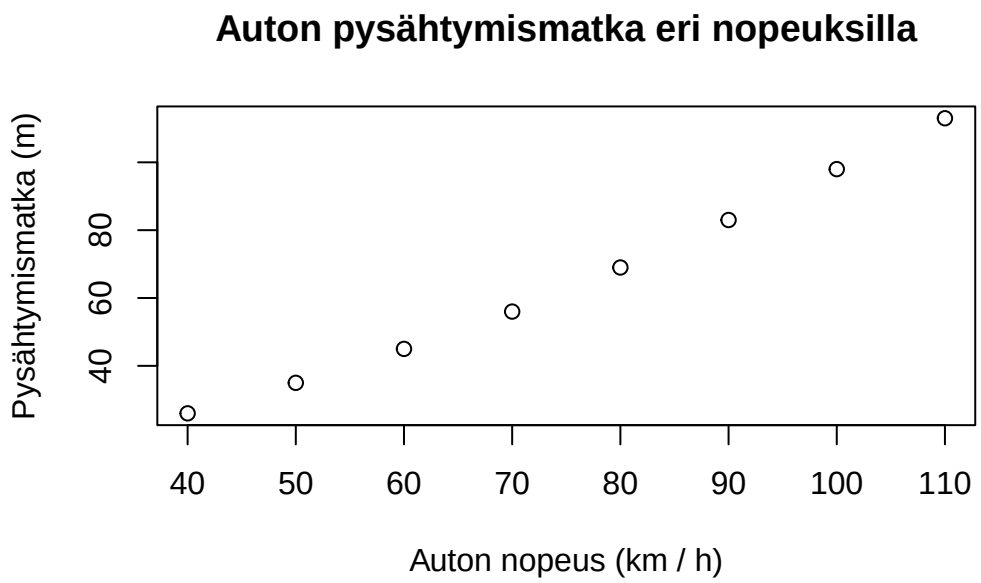
Funktio `curve` olettaa, että tarkasteltavan funktion tai lausekkeen argumentti on nimeltään `x`, mutta tätä voi vaihtaa tarvittaessa argumentilla `xname`.

7.2 Alemman tason grafiikkatoiminnot

Alemman tason grafiikkatoiminnoilla voi lisätä olemassa olevaan kuvaan lisää osia, kuten tekstiä, pisteitä tai selitteen (legend).

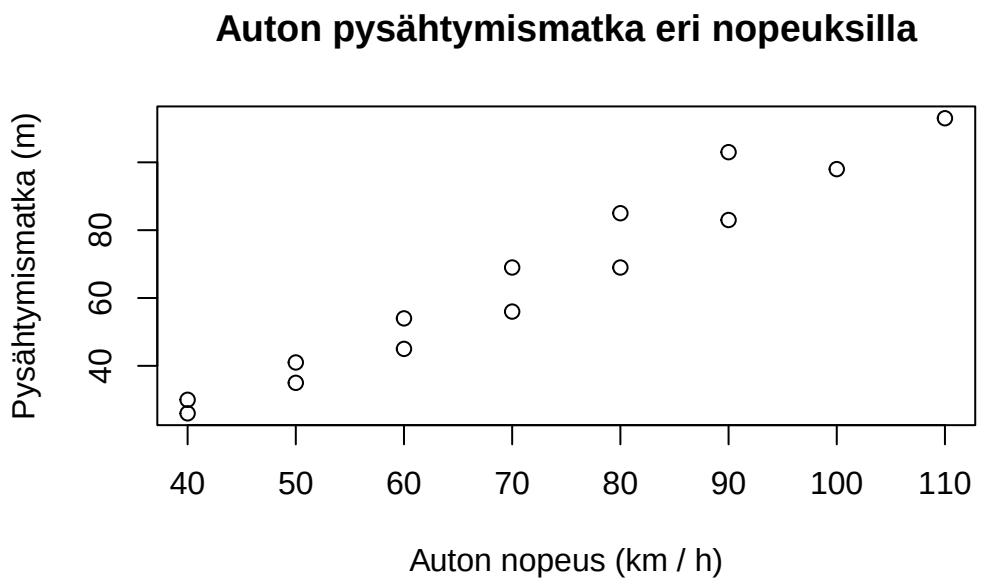
Otetaan esimerkiksi alussa nähty kuvaaja autojen pysähtymismatkoista ja lisätään siihen uusia osia. Tässä vielä alkuperäinen kuva:

```
plot(
  x = speed,
  y = stop_dist,
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
```



Lisätään kuvaajan jarrutusmatkat liukkaalla kelillä. Uusia pisteitä voi piirtää `points`-funktiolla, jolle annetaan x- ja y-koordinaatit vektoreina ihan kuin `plot`-funktiollekin.

```
stop_dist_wet <- c(30, 41, 54, 69, 85, 103, 122, 143)
plot(
  x = speed,
  y = stop_dist,
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
points(x = speed, y = stop_dist_wet)
```

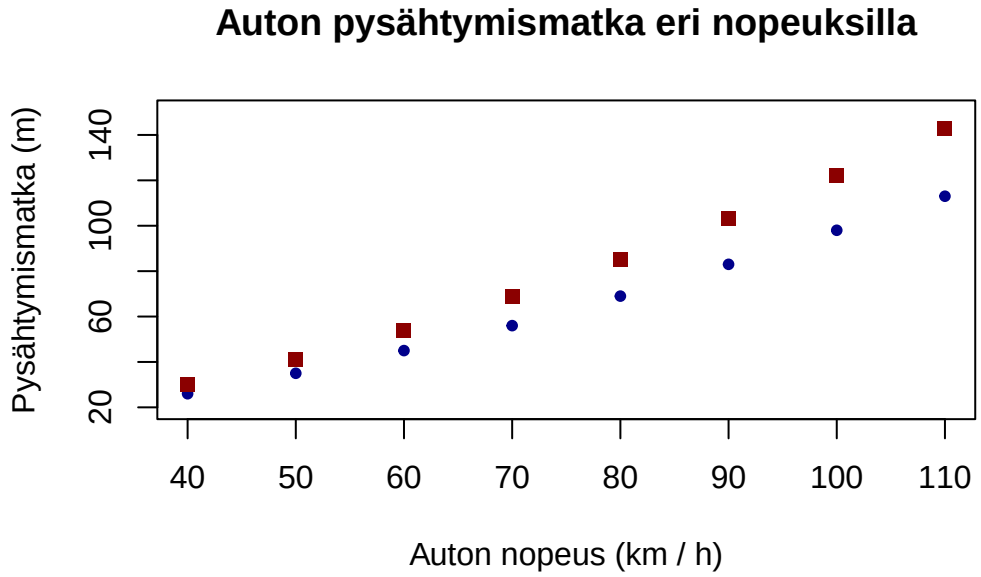


Ylläolevassa kuvaajassa on kaksi ongelmaa: ylimmät pisteet eivät näy, koska kuvaajan y-akseli loppuu kesken. y-akseli on piirretty alkuperäisten jarrutusmatkojen pohjalta, ja koska liukkaalla kelillä jarrutus kestää pidempään, uudet pisteet eivät mahdu kuvaajaan. Toinen ongelma on se, että pisteitä ei voi erottaa toisistaan.

Ensimmäinen ongelma ratkeaa säätämällä käsin y-akselin rajat. Tämä tapahtuu argumentilla `ylim`, jolle annetaan vektorissa ylä- ja alaraja (vastaavasti `xlim` säätää x-akselin rajat).

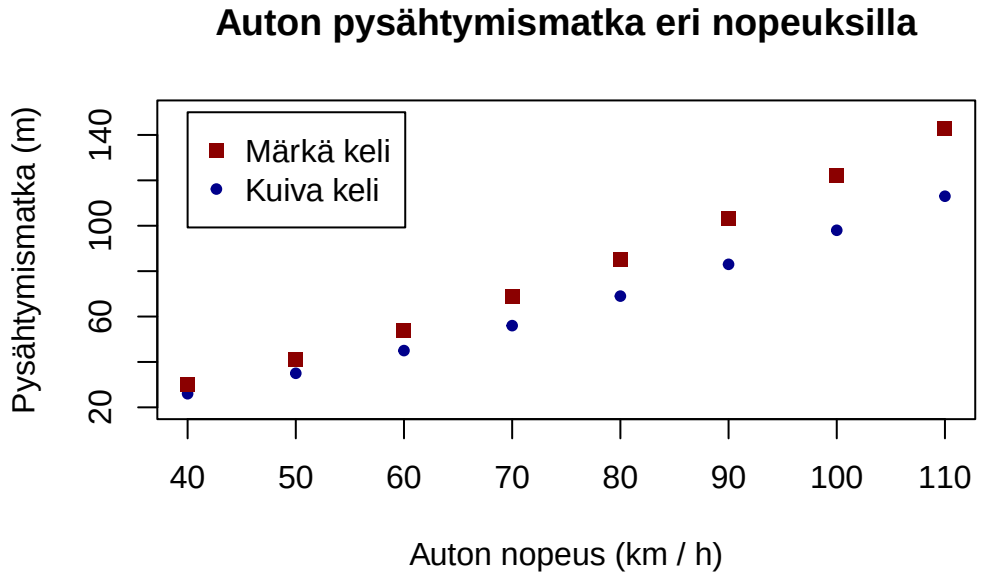
Lisäksi piirretään selvyyden vuoksi pisteet eri värisinä ja eri kuvioilla. Argumentti `col` säätää pisteiden värin ja `pch` pisteiden muodon. Eri väri- ja muotovaihtoehdot löytyvät googlaamalla.

```
plot(
  x = speed,
  y = stop_dist,
  col = "darkblue",
  pch = 20,
  ylim = c(20, 150),
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
points(x = speed, y = stop_dist_wet, pch = 15, col = "darkred")
```



Nyt kuvaaja alkaa jo näyttää paremmalta, mutta kuvaajasta ei vielä voi päätellä, mitä eri väriset pisteet tarkoittavat. Lisätään siis kuvaajaan selite `legend`-komennolla. Selitteelle määritetään paikka kuvaajassa `x` ja `y` argumenteilla (vasemman yläkulman koordinaatit). Sen jälkeen annetaan selitetekstit (`legend`), sekä selitteen muodot ja värit (`pch` ja `col`, kuten aiemmin). HUOM! Selitteen symbolit ja värit on itse osattava laittaa oikeaan järjestykseen. Selitteen tekstit annetaan järjestyksessä ylhäältä alas, ja piirtomerkit tulee antaa samassa järjestyksessä.

```
plot(
  x = speed,
  y = stop_dist,
  col = "darkblue",
  pch = 20,
  ylim = c(20, 150),
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
points(x = speed, y = stop_dist_wet, pch = 15, col = "darkred")
legend(
  x = 40,
  y = 150,
  legend = c("Märkä keli", "Kuiva keli"),
  pch = c(15, 20),
  col = c("darkred", "darkblue")
)
```



Säädetään kuvaajaa vielä hiukan, ja lisätään siihen käyrä kuvaamaan jarrutusmatkan ennustetta `lines`-funktiolla.

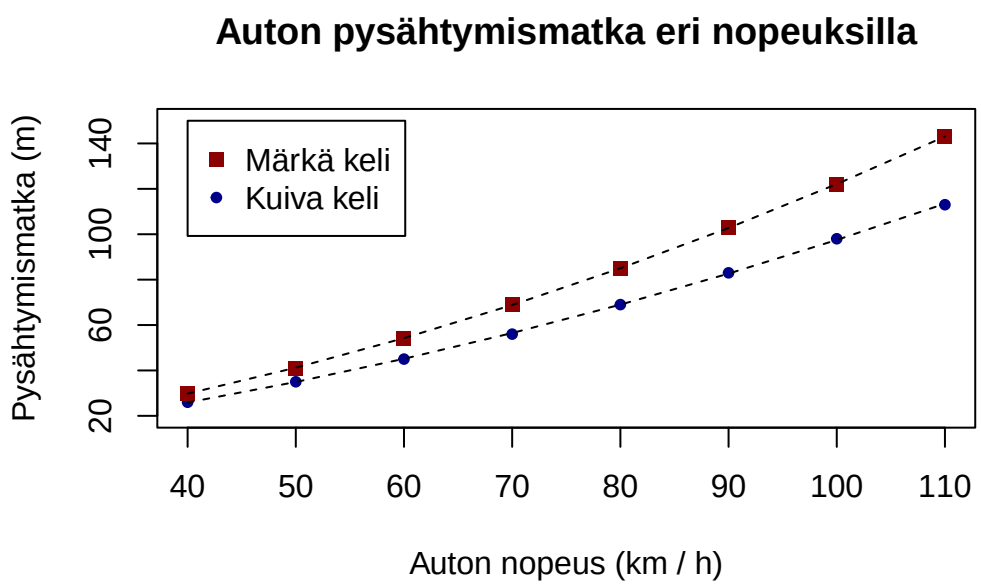
Alla olevassa koodissa lasketaan ensin `lm`-funktion avulla sopivat parametrit käyrälle. Lineaarisia malleja käsitellään vasta kappaleessa lineaariset mallit, joten tässä vaiheessa niistä ei tarvitse vielä ymmärtää muuta kuin se, että `lm`-funktio sovittaa lineaarisen mallin (tässä tapauksessa muotoa matka $= a + b \cdot \text{nopeus} + c \cdot \text{nopeus}^2$), jonka perusteella voidaan ennustaa pysähtymismatkaa myös muille kuin mitatuille nopeuksille.

```
# Create vector of squared speeds to fit second order polynomial
speed_squared <- speed^2

# Model for dry weather
model_dry <- lm(stop_dist ~ speed + speed_squared)
prediction_dry <- model_dry$fitted.values

# Model for rainy weather
model_wet <- lm(stop_dist_wet ~ speed + speed_squared)
prediction_wet <- model_wet$fitted.values
```

`lines` tarvitsee `x` ja `y` argumentit kuten `points`, mutta piirtää viivan, ei pisteitä. Käytetään äsken laskettuja mallien antamia ennusteita (`prediction`-vektoreita) y-koordinaatteina. Tehdään viivoista katkoviivoja argumentilla `lty = "dashed"` (lty eli line type).

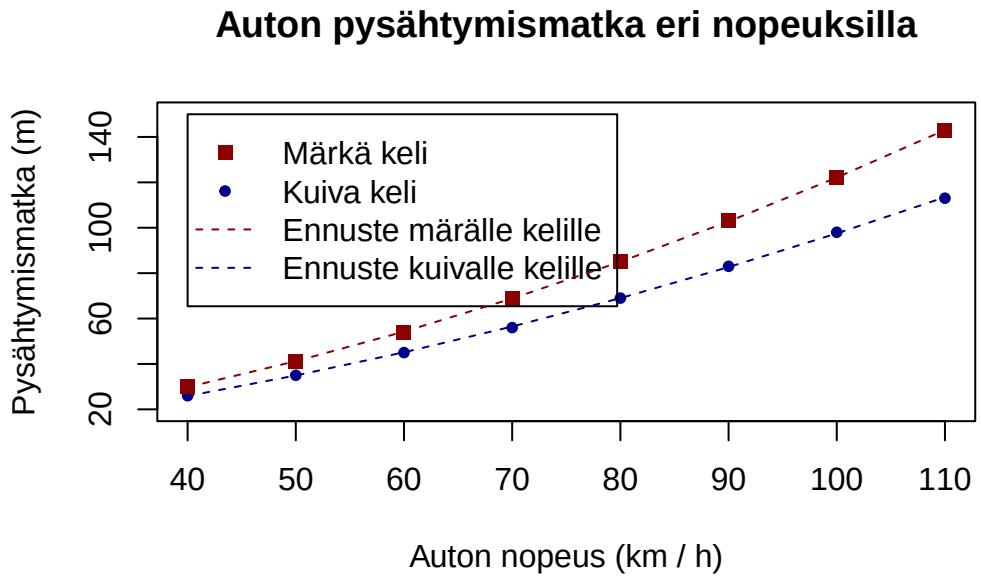
```
plot(
  x = speed,
```

```
  y = stop_dist,
  col = "darkblue",
  pch = 20,
  ylim = c(20, 150),
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
points(x = speed, y = stop_dist_wet, pch = 15, col = "darkred")
legend(
  x = 40,
  y = 150,
  legend = c("Märkä keli", "Kuiva keli"),
  pch = c(15, 20),
  col = c("darkred", "darkblue")
)
lines(speed, prediction_dry, lty = "dashed")
lines(speed, prediction_wet, lty = "dashed")
```



Seuraavaksi voidaan värittää käyrät samoilla väreillä kuin pisteet, ja lisätä niille omat selitteet. Tässä vaiheessa selitteen tekemisestä tulee jo melko monimutkaista, sillä selitteessä on mukana pisteitä ja käyriä. Tästä syystä selitteen argumentteihin pitää laittaa puuttuvia arvoja `pch` ja `lty`-argumenteille, koska selitteen ensimmäiset rivit eivät viittaa mihinkään käyrään, vaan pelkästään pisteisiin ja vastaavasti kaksi alinta riviä viittaavat vain käyriin.

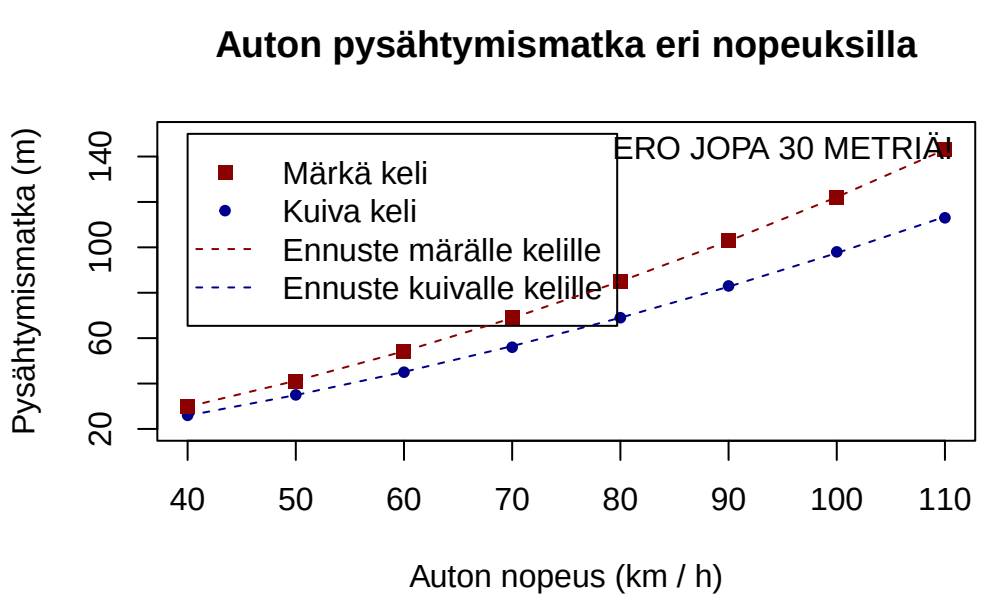
```r
plot(
  x = speed,
  y = stop_dist,
  col = "darkblue",
  pch = 20,
  ylim = c(20, 150),
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)",
  ylab = "Pysähtymismatka (m)"
)
points(x = speed, y = stop_dist_wet, pch = 15, col = "darkred")
legend(
  x = 40,
  y = 150,
  legend = c(
    "Märkä keli",
    "Kuiva keli",
    "Ennuste märälle kelille",
    "Ennuste kuivalle kelille"
  ),
  pch = c(15, 20, NA, NA),
  lty = c(NA, NA, "dashed", "dashed"),
  col = c("darkred", "darkblue", "darkred", "darkblue")
)
lines(speed, prediction_dry, lty = "dashed", col = "darkblue")
lines(speed, prediction_wet, lty = "dashed", col = "darkred")
```



Kuvaajamme on melkein valmis iltapäivälehteen muistuttamaan liukkaiden kelien vaaroista, mutta jotta siitä tulisi oikein säväyttävä, siinä pitää toki olla tekstiä! Lisätään siis vielä pieni tekstin pätkä, joka korostaa eroa liukkaan ja kuivan kelin välillä. Tekstiä voi lisätä `text`-funktiolla, jolle annetaan tuttuun tapaan `x` ja `y`-argumentit, joilla määritetään tekstin paikka ja `labels` määrittää itse tekstin (kaikki argumentit voivat olla myös pidempiä vektoreita, jolloin tulee useampi teksti eri paikkoihin). Lisäksi parametrillä `adj` (adjust) voi hienosäätää tekstin paikkaa. `adj` on vektori, jossa on hienosäätöarvot x- ja y-suunnissa.

```
plot(
  x = speed,
  y = stop_dist,
  col = "darkblue",
  pch = 20,
  ylim = c(20, 150),
  main = "Auton pysähtymismatka eri nopeuksilla",
  xlab = "Auton nopeus (km / h)", ylab = "Pysähtymismatka (m)"
)
points(x = speed, y = stop_dist_wet, pch = 15, col = "darkred")
legend(
  x = 40,
  y = 150,
  legend = c(
    "Märkä keli",
    "Kuiva keli",
    "Ennuste märälle kelille",
    "Ennuste kuivalle kelille"
```

```
  ),
  pch = c(15, 20, NA, NA),
  lty = c(NA, NA, "dashed", "dashed"),
  col = c("darkred", "darkblue", "darkred", "darkblue")
)
lines(speed, prediction_dry, lty = "dashed", col = "darkblue")
lines(speed, prediction_wet, lty = "dashed", col = "darkred")
text(x = 95, y = 145, labels = "ERO JOPA 30 METRIÄ!")
```



Kuvaajamme on nyt valmis!

7.3 Kuvaajien piirtäminen käytännössä

Jos äskeisen esimerkin aikana tuntui siltä, että näimme paljon vaivaa ja saimme lopputulokseksi kuvaajan, joka ei oikeastaan edes näytä kovin hyvältä, olet aivan oikeassa. Kuvaajien rakentaminen itse R:n peruskomennoilla on raskasta, ja usein perusgrafiikkatoimintoja käytetään lähinnä omaan käyttöön tulevien kuvaajien piirtämiseen nopeasti. Peruskomennot on kuitenkin hyvä hallita, sillä niitä saattaa tarvita valmiilla työkaluilla tehtyjen kuvaajien muokkaamiseen. Varsinkin tekstin lisääminen, sekä akselien nimeäminen ja otsikon muuttaminen ovat hyviä taitoja osata.

R tarjoaa paljon valmiita työkaluja erilaisten kuvaajien piirtämiseen. Valitettavasti tällä kurssilla ei ole aikaa sukeltaa näiden työkalujen käyttöön, sillä ennen niiden käyttöä pitää ymmärtää enemmän R:n monimutkaisemmista tietorakenteista. Inspiraatiota ja motivaatiota

voi kuitenkin hakea esimerkiksi R Graph Gallery-sivulta tai ggpubr-paketin ohjeista. R:n ehdottomasti monipuolisin ja kehitetyin työkalu kuvien piirtämiseen on `ggplot2` paketti.

8 Tilastollinen testaaminen

8.1 Testaamisen periaatteita

Tilastollisilla testeillä pyritään arvioimaan perusjoukkoa koskevien väiteiden paikkansapitävyyttä todennäköisyyslaskennan keinoin. Lähtökohtana on niin sanottu **nollahypoteesi** (H_0), joka yleensä vastaa tilannetta, jossa mahdolliset väitettä tukevat haivainnot ovat vain sattuman seurausta. Esimerkiksi jos tutkitaan onko jokin lääkeaine tehokas hoitokeino, voisi nollahypoteesi olla muotoa "lääkeaineella ei ole vaikutusta". Nollahypoteesiin liittyy aina **vastahypoteesi** (H_1), joka yleensä nollahypoteesin vastakohta, ja vastaa mielenkiinnon kohteena olevaa väitettä (esim. "lääkeaineella on vaikutusta").

Tilastolliset testit olettava nollahypoteesin olevan totta, jolloin nollahypoteesin mielessä erittäin harvinaiset tulokset antavat aihetta epäillä nollahypoteesin mielekkyyttä. Testiin liittyy yleensä **testisuure**, joka on jokin aineistosta laskettu tunnusluku. Testisuureen jakauman perusteella voidaan arvioida todennäköisyyttä, että havaittu tulos olisi vain sattuman seurausta. Tätä todennäköisyyttä kutsutaan **p-arvoksi**. Perinteisesti tilastotieteessä asetetaan etukäteen jokin merkitsevyystaso (significance level, α), ja jos saatu p-arvo on merkitsevyystasoa pienempi niin nollahypoteesi hylätään (yleensä $\alpha = 0.05$). Jos p-arvo on merkitsevyystaso pienempi, niin havaintoa kutsutaan tilastollisesti merkitseväksi (nykyisin suositellaan myös termiä "tilastollisesti erottuva", sillä sana "merkitsevä" menee usein sekaisin sanan "merkittävä" kanssa).

8.2 t-testi

Studentin t-testi on yksi tunnetuimmista tilastollisista testeistä. Se testaa yhden tai kahden ryhmän odotusarvoja tietylle muuttujalle.

Tarkastellaan R:n sisäistä dataa `sleep`, joka sisältää mittauksia muutoksista oppilaiden unen määrässä (muuttuja `extra`, muutos unen määrässä tunneissa) kahdella eri lääkkeellä (muuttuja `group`). Jokainen oppilas kokeili kumpaakin lääkettä, muuttuja `ID` yksilöi oppilaat.

8.2.1 Yhden otoksen t-testi

Testaamme aluksi hypoteesia, että muutos unen määrässä lääkkeen käytön jälkeen on 0 (H_0 : $\mu = 0$). Funktiota `t.test` voi käyttää monella tapaa. Tässä esimerkissä annamme funktiolle kaavan `extra ~ 1`, eli ns. `formula`-objektin ensimmäisenä argumenttina, joka on osa R:n syntaksia tilastollisten mallien ja riippuvuusrakenteiden määrittellyyn. Kaava määrittelee, että ~-merkin vasen puoli on vastemuuttuja, ja oikea puoli sisältää selittävät muuttujat. Koska emme tee testiä minkään toisen muuttujan suhteen, niin kaavan oikean puoli on vain luku 1, joka R:n syntaksissa tarkoittaa, että se on vakio. Tämä ei siis tarkoita esimerkiksi sitä, että nollahypoteesimme olisi, että muutos unen määrässä olisi 1 tunti. Nollahypoteesin mukainen odotusarvo määritellään argumentilla `mu`, joka yhden otoksen testissä saa oletusarvon 0.

```
# One sample test
tt1 <- t.test(extra ~ 1, data = sleep)
tt1
```

```

    One Sample t-test

data:  extra
t = 3.413, df = 19, p-value = 0.002918
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.5955845 2.4844155
sample estimates:
mean of x
     1.54
```

Tuloksena saamme t-testisuureen arvon, vapausasteet sekä testin p-arvon. Koska p-arvo on pieni (perinteisesti rajana käytetään lukua 0.05, mutta tämä vaihtelee tieteenalasta riippuen) niin nollahypoteesi hylätään, eli testin mukaan muutos unen määrässä poikkeaa tilastollisesti merkitsevästi nollasta kumpaa tahansa lääkettä käytettäessä. Tuloste kertoo myös testin vastahypoteesin H_1 kohdassa "alternative hypothesis".

Testiin liittyvät tunnusluvut (testisuure, vapausasteet ja p-arvo) saamme eriteltyä tulosobjektista `tt1` seuraavasti:

```
# Test statistic
tt1$statistic
```

```
       t
3.412965
```

```r
# Degrees of freedom
tt1$parameter
```

```
df
19
```

```r
# p-value
tt1$p.value
```

```
[1] 0.00291762
```

8.2.2 Kahden otoksen t-testi

Entäpä jos haluammekin testata hypoteesia, että kumpikin lääke vaikuttaa samalla tavalla unen määrään ($H_0 : \mu_1 = \mu_2$)? Voimme tässäkin tapauksessa käyttää `formula`-syntaksia hyödyksi. Vakion 1 sijaan sijoitamme nyt lääkettä vastaavan muuttujan `group` kaavassa `~`-merkin oikealle puolelle.

```r
tt2 <- t.test(extra ~ group, data = sleep)
tt2
```

```
    Welch Two Sample t-test

data:  extra by group
t = -1.8608, df = 17.776, p-value = 0.07939
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean in group 1 mean in group 2
           0.75            2.33
```

Testiobjektin sisältö vastaa yhden otoksen testiä suurimmilta osin. Näämme, että testin tulos ei tällä kertaa ollut tilastollisesti merkitsevä (merkitsevyystasolla 0.05) eli testin mukaan ei ole näyttöä siitä, että lääkkeet vaikuttaisivat eri tavalla unen määrään, jolloin nollahypoteesia ei hylätä.

Tarkkasilmäinen lukija saattoi kuitenkin huomata, että tämä testi ei aivan vastaa tarkoitusta, sillä `sleep`-aineistossa jokainen koehenkilö kokeili kumpaakin lääkettä, jolloin oikea tapa olisi testata lääkkeiden vaikutuksen erotusta, sillä mittaukset ovat toisistaan riippuvia. Tehdään tämä seuraavaksi.

8.2.3 Riippuvien (parittaisten) otosten t-testi

Jotta mittausparit tulevat otettua huomioon testissä, on `t.test`-funktiolle annettava argumentti `paired = TRUE`. Tässä testissä nollahypoteesi on, että lääkkeiden vaikutuksen erotuksen odotusarvo on 0 ($H_0 : \mu_d = 0$).

```
tt3 <- t.test(extra ~ group, data = sleep, paired = TRUE)
tt3
```

```
    Paired t-test

data:  extra by group
t = -4.0621, df = 9, p-value = 0.002833
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -2.4598858 -0.7001142
sample estimates:
mean difference
          -1.58
```

Tällä kertaa tulos on taas tilastollisesti merkitsevä, eli lääkkeiden vaikutuksessa unen määrään on tilastollisesti merkitsevä ero, jolloin nollahypoteesi hylätään.

8.3 Khiin neliö -testi

Kahden kategorisen muuttujan riippuvuuden tutkimiseen voidaan käyttää khiin neliö -testiä (χ^2 test). Tyypillisesti halutaan verrata jonkin muuttujan ryhmien välisiä eroja, kuten puoluekannatusta alueittain tai sukupuolten suhteen. Testin ideana on verrata havaittua ristitaulukkoa nollahypoteesin mukaiseen ristitaulukkoon, jossa muuttujien välillä ei ole lainkaan riippuvuutta. Khiin neliö -testin testisuure perustuu näiden kahden taulukon eroihin.

Tarkastellaan Yhdysvaltalaista kyselytutkimusaineistoa, joka sisältää tiedon henkilön puoluekannatuksesta ja sukupuolesta. Tutkitaan khiin neliö -testin avulla, riippuuko puoluekannatus sukupuolesta. R:ssä tämä voidaan tehdä funktiolla `chisq.test`.

```
## From Agresti(2007) p.39
M <- as.table(rbind(c(762L, 327L, 468L), c(484L, 239L, 477L)))
dimnames(M) <- list(
  gender = c("F", "M"),
```

```
  party = c("Democrat", "Independent", "Republican")
)
(Xsq <- chisq.test(M))  # Prints test summary
```

```
    Pearson's Chi-squared test

data:  M
X-squared = 30.07, df = 2, p-value = 2.954e-07
```

```
Xsq$observed   # observed counts (same as M)
```

```
      party
gender Democrat Independent Republican
     F      762         327        468
     M      484         239        477
```

```
Xsq$expected   # expected counts under the null
```

```
      party
gender Democrat Independent Republican
     F 703.6714    319.6453   533.6834
     M 542.3286    246.3547   411.3166
```

```
Xsq$residuals  # Pearson residuals
```

```
      party
gender   Democrat Independent Republican
     F  2.1988558   0.4113702 -2.8432397
     M -2.5046695  -0.4685829  3.2386734
```

```
Xsq$stdres     # standardized residuals
```

```
      party
gender   Democrat Independent Republican
     F  4.5020535   0.6994517 -5.3159455
     M -4.5020535  -0.6994517  5.3159455
```

Koska testin p-arvo on pieni, niin nollahypoteesi hylätään ja todetaan, että puoluekannatus riippuu tilastollisesti merkitsevästi sukupuolesta. Testin luotettavuuden kannalta on kuitenkin hyvä huomioida, että testiin liittyy oletuksia, jotka koskevat odotettuja frekvenssejä (eli nollahypoteesin mukaisen ristitaulukon frekvenssejä). Tyypillisesti vaaditaan, että odotetun frekvenssin on oltava vähintään 5 vähintään 80%:ssa taulukon soluista, eikä yhdenkään solun odotettu frekvenssi ole alle 1. Tarkistetaan oletukset edellisen esimerkin tapauksessa, odotetut frekvenssit löytyvät testiobjektin alkiosta `expected`:

```
all(Xsq$expected >= 1)
```

```
[1] TRUE
```

```
mean(Xsq$expected >= 5) >= 0.80
```

```
[1] TRUE
```

Oletukset ovat tältä osin kunnossa. Edellä funktio `all` ottaa syötteenään loogisen vektorin ja palauttaa `TRUE` jos syötteen kaikki alkiot ovat `TRUE`. Muutoin funktio palauttaa `FALSE`. Voisimme myös laskea odotetut frekvenssit seuraavalla tavalla suoraan taulukosta ennen testin tekemistä:

```
rowSums(M) %*% t(colSums(M)) / sum(M)
```

```
     Democrat Independent Republican
[1,] 703.6714    319.6453   533.6834
[2,] 542.3286    246.3547   411.3166
```

Toisin sanoen, kerromme jokaisen rivisumman vastaavalla sarakesummalla ja jaamme lopputuloksen havaintojen määrällä.

Edellisessä esimerkissä ristitaulukko oli valmiiksi rakennettu annetuista frekvensseistä. Ristitaulukko voitaisiin myös rakentaa yksilötason aineistosta, esimerkiksi datakehikosta, joka sisältää rivin jokaista kyselyyn vastannutta henkilöä kohden ja tiedon vastaajan ilmoittamasta puolueesta ja sukupuolesta. Seuraavassa esimerkissä kyselytutkimusaineisto on muuttujassa `poll_data`, joka voidaan muuntaa ristitauloksi funktiolla `table`. Huomataan, että tällä tavalla muodostettu ristitaulukko on sama kuin suoraan frekvensseistä koottu taulukko.

```
head(poll_data)
```

```
  gender    party
1      F Democrat
2      F Democrat
3      F Democrat
4      F Democrat
5      F Democrat
6      F Democrat
```

```
nrow(poll_data)
```

```
[1] 2757
```

```
# The cross tabulations are the same
identical(M, table(poll_data))
```

```
[1] TRUE
```

8.4 Varianssianalyysi

Varianssianalyysin voidaan ajatella olevan t-testin yleistys, jossa yhden tai kahden odotusarvon sijaan verrataankin kerralla useamman ryhmän odotusarvoja keskenään. Menetelmä saa nimensä siitä, että sen testisuure perustuu kiinnostuksen kohteena olevan muuttujan kokonaisvaihtelun (varianssin) jakamiseen verratavien ryhmien sisäiseen vaihteluun ja niiden väliseen vaihteluun. Koska harjoitusaineistossa `study_data` ei vielä ole kategorista muuttujaa, jossa on vähintään kolme kategoriaa, niin luodaan sellainen.

```
study_data <- read.table("data/study_data.txt")

# create a new categorical variable called age_group
# 3,5,6 => ryhmä 1
# 2,4,8 => ryhmä 2
# 1,7   => ryhmä 3
# vaste: weight
study_data$age_group <- c("3", "2", "1", "2", "1", "1", "3", "2")
study_data$fage_group <- factor(study_data$age_group)
```

Hypoteesit ovat:

- H_0: Ryhmien odotusarvot ovat samat tarkasteltavan muuttujan suhteen ($\mu_1 = \mu_2 = \cdots = \mu_n$),

- H_1: Ryhmien odotusarvoissa on eroa tarkasteltavan muuttujan suhteen ($\mu_i \neq \mu_j$ ainakin joillekin $i \neq j$).

```
# conduct Analysis of Variance (ANOVA) for study_data
# we test if averages of height differ between age groups
summary(aov(height ~ fage_group, data = study_data))
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
fage_group   2   5599    2799   0.782  0.506
Residuals    5  17901    3580
```

Varianssianalyysin `summary` sisältää seuraavat sarakkeet: `Df` kertoo testiin liittyvän F-jakauman vapausasteet, `Sum Sq` kertoo ryhmiin liittyvän neliösumman ja jäännösneliösumman, `Mean Sq` kertoo vastaavan keskineliösumman, `F value` kertoo testisuureen arvon ja `Pr(>F)` kertoo testin P-arvon. Tässä tapauksessa testin p-arvo on 0.417, joten nollahypoteesia ei hylätä.

8.5 Levenen testi

Levenen testillä tutkitaan ovatko jonkin muuttujan varianssit samat kahdessa tai useammassa ryhmässä. Testiä ei ole toteutettu valmiiksi R:ssä, mutta se on saatavilla Rcourse-paketin kautta funktiossa `leveneTest`. Testin nollahypoteesi on, että muuttujan varianssit ovat samat joka ryhmässä.

Selvitetään onko harjoitusaineiston `study_data` muuttujan `height` varianssissa eroa eri ikäryhmien välillä (`fage_group`) Levenen testillä.

```
leveneTest(height ~ fage_group, data = study_data)
```

```
Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group  2  0.7627 0.5139
       5
```

Testin p-arvo löytyy sarakkeesta `Pr(>F)`, ja se on 0.6105. Testin mukaan muuttujan varianssit ovat siis samat joka ryhmässä, ja nollahypoteesi jää voimaan.

8.6 Shapiro-Wilk -testi

Shapiro-Wilk -testillä tutkitaan onko jokin muuttuja normaalijakautunut. Testi löytyy funktiosta `shapiro.test`, ja se ottaa argumenttinaan yhden muuttujan havainnot vektorina. Funktiolla ei voi siis suoraan testata esimerkiksi sitä, onko muuttuja normaalijakautunut joissakin osaryhmissä, vaan aineisto on ensin jaettava sopiviin osiin. Testin nollahypoteesi on, että muuttuja on normaalijakautunut.

Tarkastellaan jälleen harjoitusaineistoa `study_data` ja testataan muuttujan `height` normaalisuutta.

```
shapiro.test(study_data$height)
```

```
    Shapiro-Wilk normality test

data:  study_data$height
W = 0.52834, p-value = 2.26e-05
```

Testin p-arvon voi lukea kohdasta `p-value` ja se on aineistolle 0.8973, eli `height`-muuttuja on testin mukaan normaalijakautunut, ja nollahypoteesi jää voimaan.

9 Lineaariset mallit

Lineaarisessa mallissa eli lineaarisessa regressiossa tavoite on arvioida vastemuuttujan lineaarista riippuvuutta selittävistä muuttujista. Käytetään esimerkkinä R:n sisäistä dataa `cars`, joka sisältää 50 auton nopeudet ja pysähtymismatkat. Tavoitteena on tutkia, miten auton pysähtymismatka riippuu auton nopeudesta.

9.1 Teoria

Yksinkertaisin mahdollinen lineaarinen regressiomalli on:

$$y = \beta_0 + \beta_1 x_1 + \epsilon$$

- y on **vastemuuttuja**, eli tässä auton pysähtymismatka
- β_0 on ns. **vakiotermi** eli **regressiosuoran** y-akselin leikkauskohta
- β_1 on selittävän muuttujan eli auton nopeuden **regressiokerroin** eli **regressiosuoran** kulmakerroin
- x_1 on selittävä muuttuja eli auton nopeus (km/h)
- ϵ on **jäännös**, joka oletetaan normaalijakautuneeksi

Mallissa siis oletetaan, että auton pysähtymismatka nopeudella 0 km/h on β_0 ja kasvaa β_1 verran, kun nopeus kasvaa 1 km/h. Lisäksi mukana on virhetermi ϵ, joka selittää satunnaisen vaihtelun tuloksissa lineaarisen käyrän ympärillä.

Jos malliin halutaan lisätä selittäviä muuttujia, kuten esimerkiksi auton jarrujen kunto (x_2) tai sääolosuhteet (x_3), malli näyttää tältä:

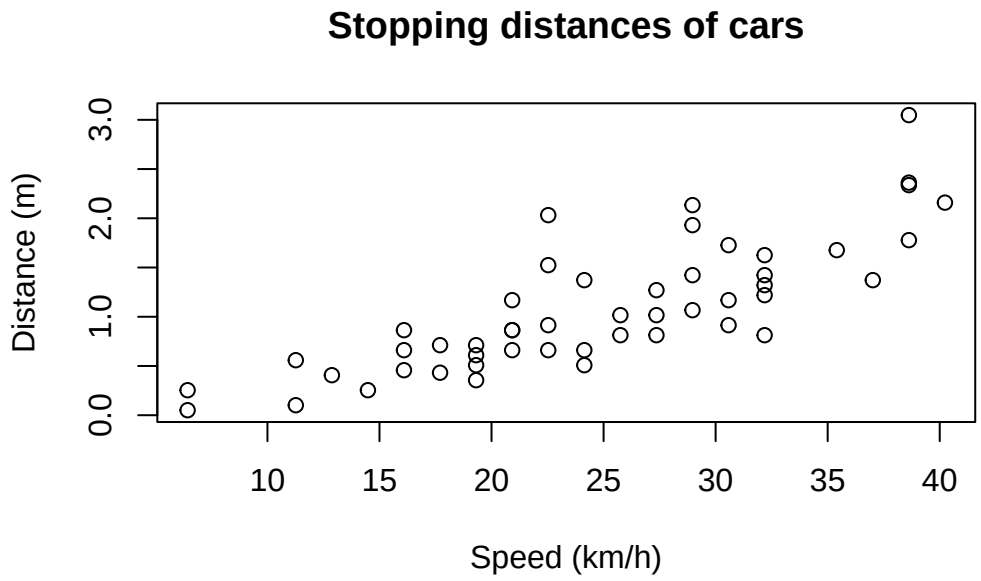
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... + \epsilon$$

Eli jokaiselle selittävälle muuttujalle annetaan oma regressiokerroin.

9.2 Esimerkki

Muutetaan ensin `cars`-aineiston muuttujat meille tuttuihin yksiköihin, ja piirretään hajontakuvio havainnoista:

```
# Change to SI units
cars$speed <- cars$speed * 1.60934
cars$dist <- cars$dist * 0.0254
# Scatter plot
plot(
  cars$speed,
  cars$dist,
  xlab = "Speed (km/h)",
  ylab = "Distance (m)",
  main = "Stopping distances of cars"
)
```



Autojen välillä on eroja, mutta kuten voi odottaa, suuremmilla nopeuksilla auton pysähtymismatka kasvaa. Käytetään seuraavaksi R:n funktiota `lm`, jolla voidaan sovittaa dataan lineaarinen malli:

```
model <- lm(dist ~ speed, data = cars)
```

`lm`-funktiolle annetaan ensimmäiseksi argumentiksi lineaarisen mallin kaava (`formula`), jossa `~` korvaa yllä nähdyn yhtä kuin -merkin. HUOM: vakiotermi on automaattisesti mukana,

eli sitä ei tarvitse kirjata erikseen. Lisäksi täytyy antaa argumentti `data`, jonka tulee olla datakehikko, josta kaavassa olevat muuttujat löytyvät. Jos malliin haluttaisiin useampia selittäjiä, lisättäisiin ne `~` merkin oikealle puolelle `+` merkein eroteltuna, esim. mallissa jossa vaste olisi muuttuja `y` ja selittäjät `x1`, `x2` ja `x3`, tulisi kaava kirjoittaa muodossa `y ~ x1 + x2 + x3`.

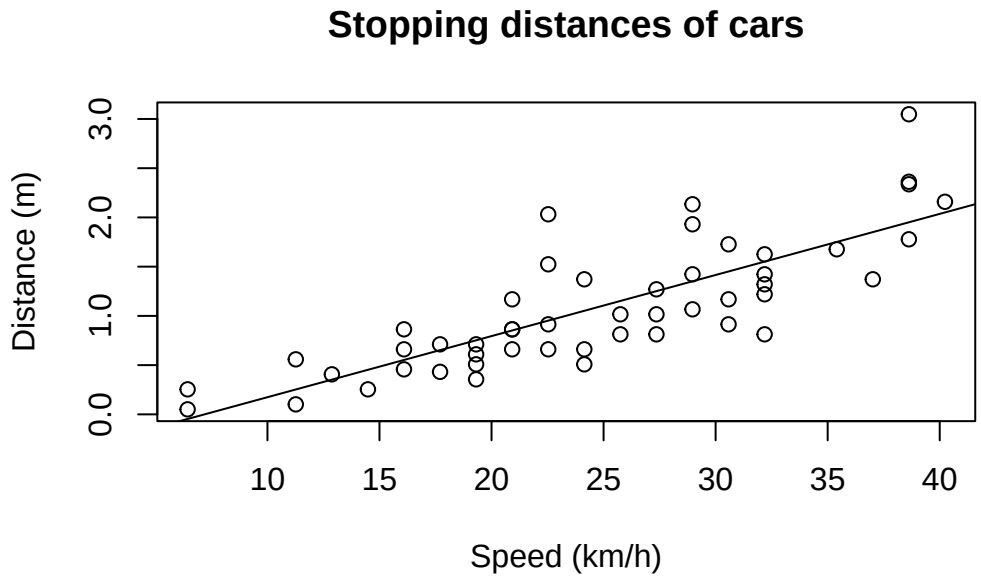
Lineaarisesta mallista saadaan irti paljon tietoa, tärkeimpinä mallin regressiokertoimet (*regression coefficients*), jotka saadaan näkyviin funktiolla `coef`, jolle annetaan argumenttina lineaarisen mallin sisältävä objekti `model`.

```
coef(model)
```

```
(Intercept)       speed
-0.44650901  0.06206469
```

Yllä olevista kertoimista voidaan päätellä, että kun auton nopeus kasvaa 1 km/h niin sen pysähtymismatka kasvaa keskimäärin noin 0.06 m, ja odotettu kasvukäyrä leikkaa y-akselin -0.4 m kohdalla. Voimme piirtää tämän käyrän kuvaajaan `abline`-funktion avulla, antamalla sille mallin kertoimet:

```
cf <- coef(model)
plot(
  cars$speed,
  cars$dist,
  xlab = "Speed (km/h)",
  ylab = "Distance (m)",
  main = "Stopping distances of cars"
)
abline(a = cf[1], b = cf[2])
```



9.3 Tarkempia tietoja mallista

Muihin mallin tietoihin pääsee käsiksi `summary`-funktion avulla, joko tulostamalla tuloksen konsoliin, tai sijoittamalla sen muuttujaan, josta voi etsiä mallin tietoja.

```
# Print summary information
summary(model)
```

```
Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73835 -0.24194 -0.05771  0.23405  1.09731

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.446509   0.171664  -2.601   0.0123 *
speed        0.062065   0.006558   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3906 on 48 degrees of freedom
```

```
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

```r
# Save summary and access specific information
s <- summary(model)
s$adj.r.squared
```

```
[1] 0.6438102
```

`summary` kertoo mm. regressiokertoimien estimaattien lisäksi niihin liittyvät p-arvot kohdassa `Pr(>|t|)`, sekä mallin selitysasteen (merkintätapa johtuu siitä, että p-arvot tulevat t-testeistä). Tässä tapauksessa muuttujan `speed` p-arvo on hyvin pieni, joten voimme todeta suurella varmuudella, että autojen pysähtymismatka riippuu (lineaarisesti) auton nopeudesta. R^2 eli selitysaste (*coefficient of determination*) kertoo, kuinka suuren osuuden pysähtymismatkojen varianssista auton nopeus selittää.

Mallin regressiokerrointen estimoitu kovarianssimatriisi saadaan funktiolla `vcov` (variance-covariance matrix). Kertoimien keskivirheet saadaan tästä edelleen helposti matriisin diagonaalin neliöjuurina (funktiot `diag` ja `sqrt`):

```r
# Covariance matrix of the regression coefficients
vcov(model)
```

```
             (Intercept)         speed
(Intercept)  0.029468659 -1.065882e-03
speed       -0.001065882  4.300714e-05
```

```r
# Standard errors only
sqrt(diag(vcov(model)))
```

```
(Intercept)       speed
0.171664380 0.006557983
```

Luottamusvälit (*confidence interval*) mallin parametreille saadaan funktiolla `confint`. Oletuksena lasketaan 95 %:n luottamusvälit.

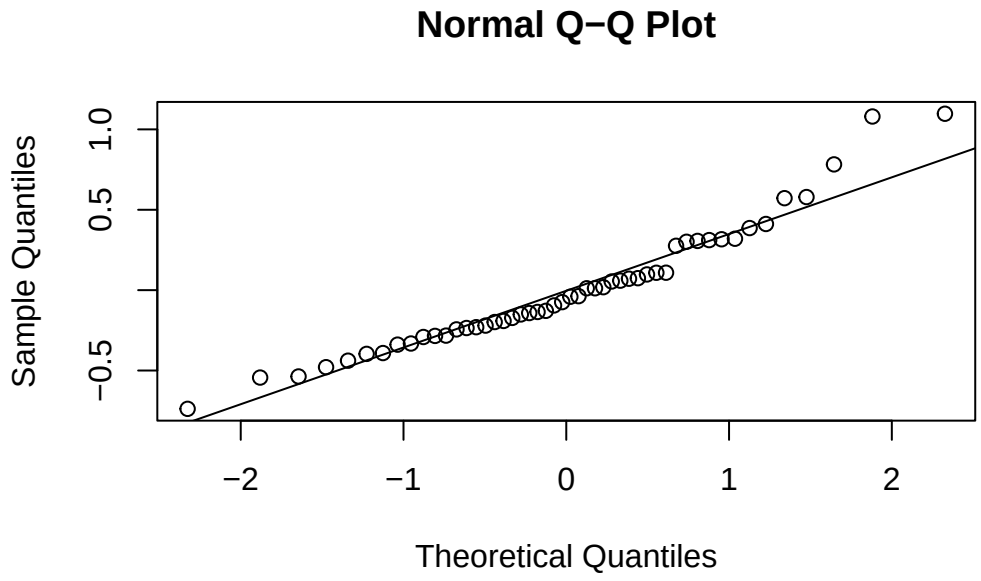
```r
confint(model)
```

```
                 2.5 %     97.5 %
(Intercept) -0.79166338 -0.1013546
speed        0.04887898  0.0752504
```

9.4 Jäännökset

Lineaarisen regressiomallin oletus on jäännösten normaalijakautuneisuus. Voimme tarkastella tätä oletusta esimerkiksi ns. kvantiili-kvantiili-kuvion avulla (*quantile-quantile plot, qq-plot*). Mallin jäännökset löytyvät malliobjektin alkiosta `residuals`. Piirretään regressiomallin kvantiili-kvantiili-kuvio funktiolla `qqnorm`. Voimme myös lisätä kuvaan suoran `qqline`-funktiolla, joka kuvaa täydellistä mallin sopivuutta: jäännösten tulisi asettua mahdollisimman hyvin tälle suoralle. Suuret poikkeamat suorasta kertovat siitä, että oletus jäännösten normaalijakautuneisuudesta ei välttämättä pidä paikkansa.

```
qqnorm(s$residuals)
qqline(s$residuals)
```

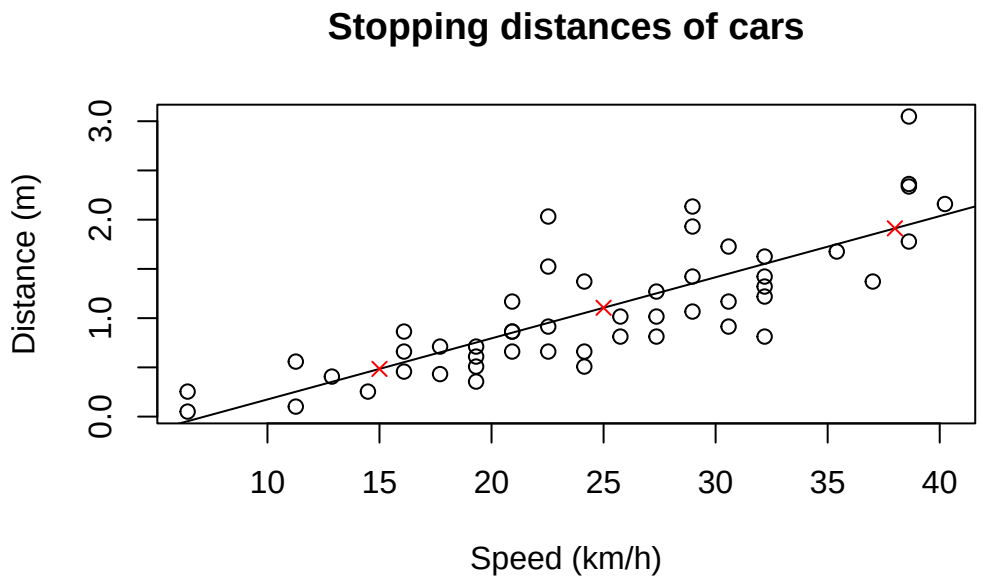


Tässä tapauksessa jäännökset näyttävät asettuvan melko hyvin suoralle eikä suuria poikkeamia ole havaittavissa, joten normaalisuusoletusta ei ole syytä epäillä kuvan perusteella.

9.5 Ennustaminen

Kun lineaarinen malli on estimoitu, sen perusteella voidaan myös ennustaa arvoja uusille havainnoille. Tämä tapahtuu `predict`-funktiolla, jolle annetaan malli, sekä datakehikko, joka sisältää ne selittäjien arvot, joille halutaan laskea ennusteet. Tämä datakehikko voi sisältää useita rivejä, jolloin ennuste lasketaan joka riville. Ennustetaan edellisen mallin perusteella pysähtymismatka autolle kolmella uudella nopeudella ja lisätään ne edelliseen kuvaajaan punaisilla rukseilla:

```r
# Create data frame with new speed values
new_data <- data.frame(speed = c(25, 15, 38))
# Create dist column by predicting from linear model
new_data$dist <- predict(model, newdata = new_data)

# Add points to previous plot
plot(
  cars$speed,
  cars$dist,
  xlab = "Speed (km/h)",
  ylab = "Distance (m)",
  main = "Stopping distances of cars"
)
abline(a = model$coefficients[1], b = model$coefficients[2])
points(new_data$speed, new_data$dist, pch = 4, col = "red")
```



Kuten huomataan, ennustetut arvot ovat täsmälleen käyrän päällä.

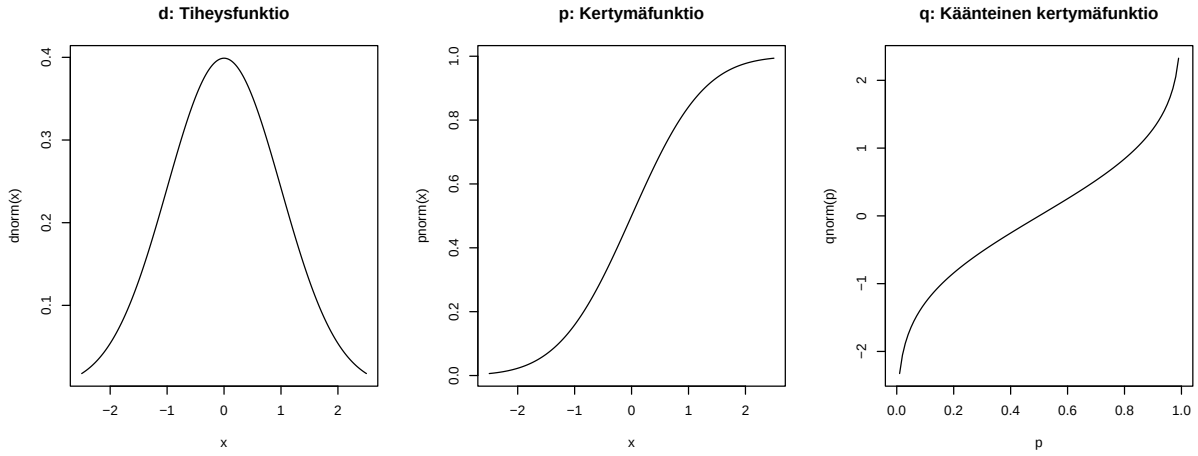
10 Todennäköisyysjakaumat

Monille yleisimmistä tilastollisista jakaumista eli todennäköisyysjakaumista on valmiita funktiota R:ssä. Funktioita on neljää eri tyyppiä, jotka merkitään funktion nimen ensimmäisellä kirjaimella.

- `d`: Tiheysfunktio: mikä on tiheysfunktion arvo pisteessä x?
- `p`: Kertymäfunktio: millä todennäköisyydellä jakaumasta poimittu arvo on pienempi/suurempi kuin q?
- `q`: Käänteinen kertymäfunktio (eli kvantiilifunktio): mille arvolle kertymäfunktio palauttaa todennäköisyyden p?
- `r`: Satunnaislukugeneraattori: simuloi satunnaisia havaintoja jakaumasta.

Näitä neljää kirjainta seuraa varsinaisen jakauman määrittävä pääte. Esim. normaalijakaumalle pääte on `norm`, jolloin tätä vastaavat R:n jakaumafunktiot ovat `dnorm`, `pnorm`, `qnorm`, ja `rnorm`. Vastaavasti t-jakaumalle pääte on `t`, jolloin funktiot ovat `dt`, `pt`, `qt` ja `rt`.

Alla ovat kuvaajat ensimmäisestä kolmesta funktiosta standardinormaalijakaumalle (pääte `norm`):

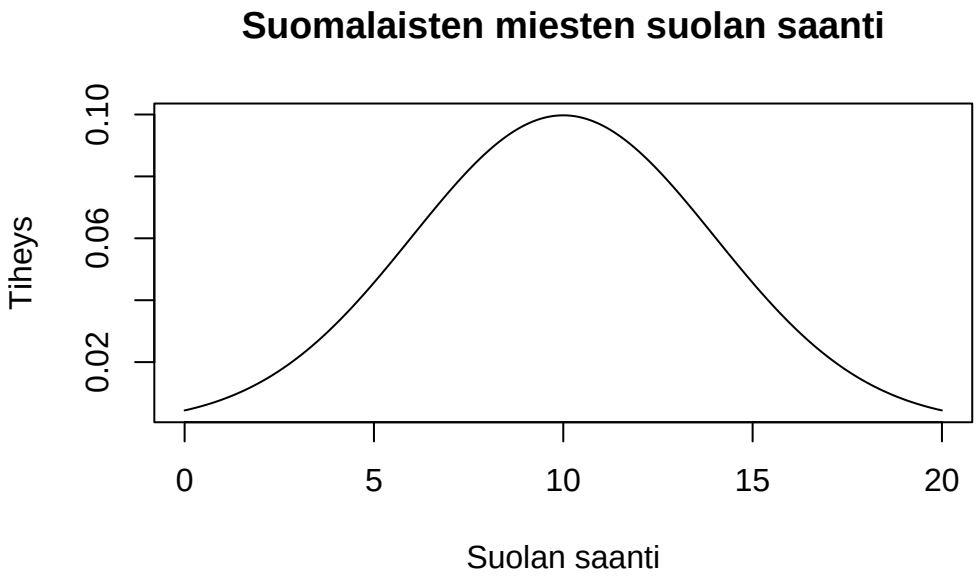


10.1 Esimerkki: normaalijakauma

Oletetaan, että suomalaisten miesten suolan saanti on normaalijakautunut odotusarvolla 10 grammaa päivässä ja keskihajonta on 4 grammaa päivässä (odotusarvo on totta, keskihajonta

allekirjoittaneen hihasta). Piirretään ensin kuva jakaumasta välillä $[0, 20]$ grammaa päivässä. Jakauman muoto saadaan funktiolla `dnorm`, eli yllä olevan ohjeen mukaan `d`-alkuinen funktio antaa tiheysfunktion, ja `norm`-pääte viittaa normaalijakaumaan. Normaalijakauman funktiolle tulee kertoa jakauman odotusarvo (`mean`) ja keskihajonta (`sd`).

```
# Sequential vector  of salt consumption
salt <- seq(0, 20, by = 0.1)
# Density function
density <- dnorm(salt, mean = 10, sd = 4)
# Line plot
plot(
  salt, density, type = "l",
  xlab = "Suolan saanti",
  ylab = "Tiheys",
  main = "Suomalaisten miesten suolan saanti"
)
```



Aikuisten saantisuositus on enintään 5 grammaa suolaa päivässä. Kuinka moni suomalainen mies syö tämän jakauman mukaan sopivasti suolaa? Vastaus saadaan kertymäfunktiosta todennäköisyytenä $P(X \leq 5)$ `pnorm`-funktion avulla.

```
pnorm(5, mean = 10, sd = 4)
```

```
[1] 0.1056498
```

Tämän jakauman mukaan vain noin 11 % suomalaisista miehistä syö suolaa sopivasti!

Suomalaisten naiset syövät keskimäärin 7 grammaa suolaa päivässä. Kuinka moni mies syö tätä enemmän suolaa? `pnorm` antaa oletuksena arvon $P(X \leq 7)$. Nyt halutaan kuitenkin tietää $P(X > 7)$, joka saadaan asettamalla `lower.tail = FALSE`:

```
pnorm(7, mean = 10, sd = 4, lower.tail = FALSE)
```

```
[1] 0.7733726
```

Noin 77 % miehistä syö suolaa keskimääräistä naista enemmän.

Entä jos halutaan tietää, kuinka paljon suolaa eniten syövä 10 % vähintään saa? Tähän voidaan vastata funktiolla `qnorm`, joka on jakauman käänteinen kertymäfunktio, eli funktion `pnorm` käänteisfunktio. Samoin kuin `pnorm`, `qnorm`-funktion oletus on, että todennäköisyydet lasketaan jakauman vasemmasta hännästä alkaen. Vastaus tähän kysymykseen selviää siis näillä kahdella tavalla:

```
qnorm(0.1, mean = 10, sd = 4, lower.tail = FALSE)
```

```
[1] 15.12621
```

```
qnorm(0.9, mean = 10, sd = 4)
```

```
[1] 15.12621
```

Eli tämän jakauman mukaan eniten suolaa saava 10 % miehistä syö yli kolminkertaisen määrän suolaa suositukseen verrattuna.

10.2 Muita jakaumia

Vastaavat funktiot löytyvät myös muille jakaumille, kuten:

- Khiin neliö: `chisq`
- Eksponentiaalinen: `exp`
- Studentin t: `t`
- Tasajakauma: `unif`
- Poisson: `pois`
- Binomijakauma: `binom`

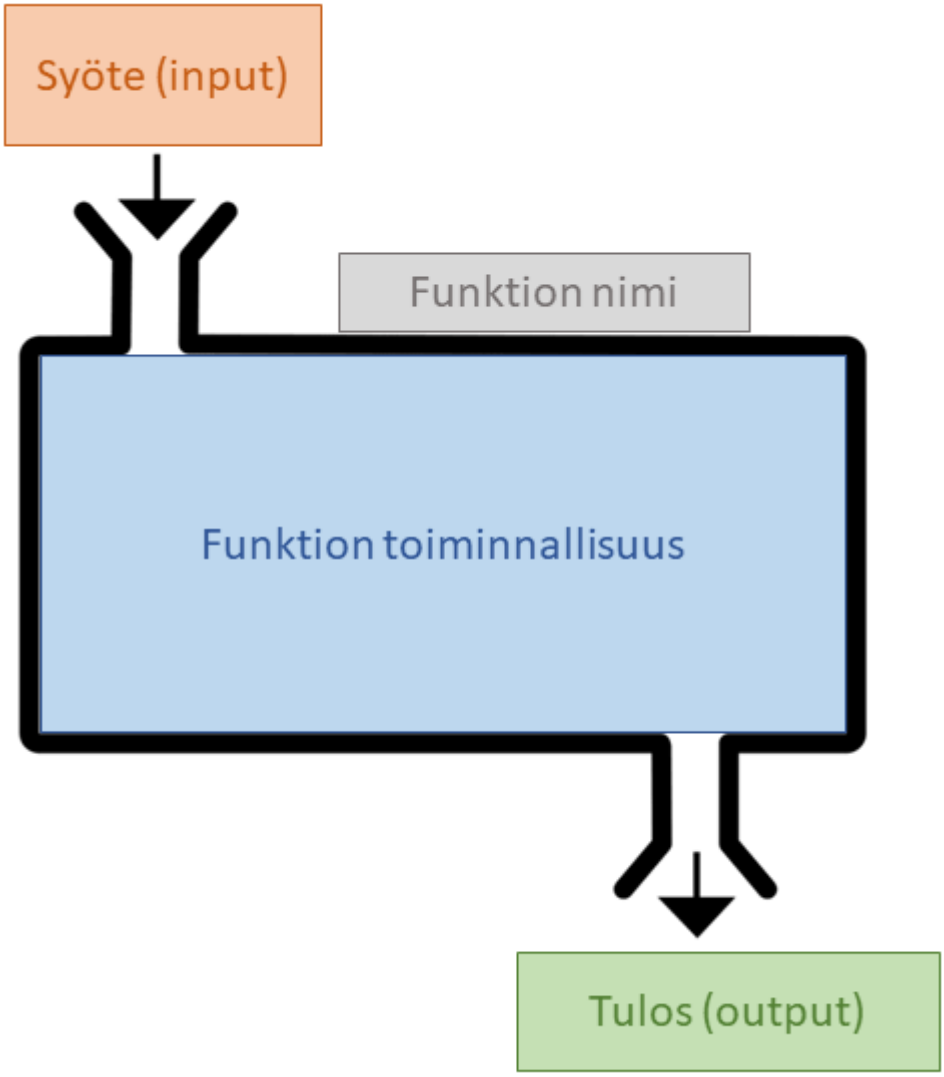
ja niin edelleen.

11 Funktiot

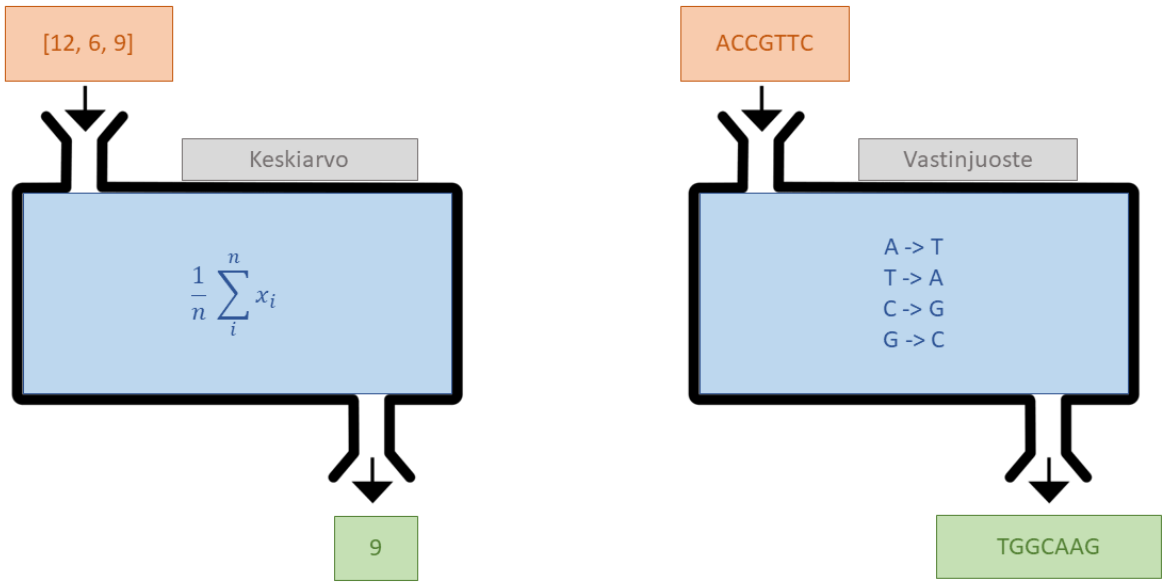
Tässä luvussa tutustutaan omien funktioiden kirjoittamiseen ja pureudutaan sitä kautta syvemmälle R-funktioiden toimintaan.

11.1 Funktion käsite

Funktio on nimetty kokonaisuus järjesteltyä ja uudelleenkäytettävää koodia, jonka tarkoitus on suorittaa yksi tarkkaan määrätty tehtävä. Funktioilla on syöte (input) ja tulos (output). Funktion tehtävä on palauttaa (return) syötteen perusteella laskettu tulos. Alla olevassa kuvassa näkyvät funktion neljä osaa: nimi, syöte, toiminnallisuus ja tulos.



Otetaan esimerkiksi kaksi funktiota: “Keskiarvo” ottaa syötteenä halutun määrän lukuja, ja laskee niiden keskiarvon. “Vastinjuoste” ottaa syötteenä DNA-juosteen ja palauttaa sen vastinjuosteen.



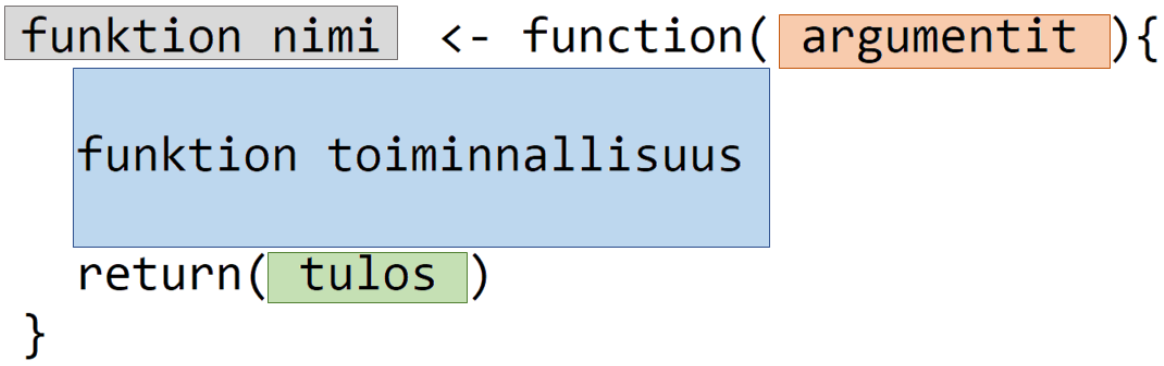
Funktioilla voi olla myös erityyppisiä syötteitä. Voitaisiin esimerkiksi määritellä funktio, jolle annettaisiin syötteenä henkilön ikä, pituus, paino, sekä elintapatietoja, ja funktio laskisi näiden pohjalta eliniänodotteen. Yksittäisestä funktion syötteestä käytetään tavallisesti nimitystä "argumentti".

11.2 R-funktiot

11.2.1 Funktioiden määrittely

Tähän mennessä olemme jo käyttäneet monia R-funktioita, eikä meidän ole tarvinnut miettiä niiden toimintaa kovin syvällisesti. Mahdolliset virhetilanteet on kuitenkin paljon helpompi ratkaista, kun ymmärtää miten funktiot toimivat R:ssä.

R-funktioita luodaan `function`-komennolla. Funktion luominen näyttää tältä:



R-funktiot siis koostuvat samoista osista kuin yllä esitellyt funktiot:

- Funktion nimi nimeää muuttujan, johon funktio tallennetaan.
- Funktion syöte koostuu argumenteista (funktio voi olla myös argumentiton).
- Funktion toiminnallisuus on R-koodia.
- Funktion tulos palautetaan komennolla `return`.

Tehdään esimerkiksi funktio BMI:n laskemiseen:

```
# Define function name and arguments
bmi <- function(height, mass) {
  # Compute BMI
  value <- mass / height^2
  rounded <- round(value, digits = 1)
  # Return computed value
  return(rounded)
}
```

Ensimmäisellä rivillä määritellään muuttuja, johon funktio tallennetaan, eli funktion nimi `bmi`. Lisäksi määritetään funktion argumentit, tässä tapauksessa `height` ja `mass`. Itse funktion koodi tulee aaltosulkeiden sisään seuraaville riveille. Ensimmäinen koodirivi laskee BMI:n ja toinen pyöristää tuloksen yhden desimaalin tarkkuuteen. Kolmas rivi palauttaa sen.

Voimme nyt kutsua (call) funktiotamme aivan kuin muitakin R-funktioita:

```
# Example
my_bmi <- bmi(height = 1.79, mass = 74)
my_bmi
```

```
[1] 23.1
```

HUOM: palautettava arvo on ainoa asia, joka välittyy funktion ulkopuolelle. Koska funktiomme palauttaa pyöristetyn arvon, alkuperäiseen arvoon ei pääse funktion ulkopuolelta käsiksi.

```
my_bmi <- bmi(height = 1.90, mass = 95)
# Throws error
value
```

Funktioiden sisällä luodut muuttujat ovat siis olemassa vain sen sisällä ja lakkaavat olemasta, kun funktion suoritus lakkaa.

11.2.2 Argumentit ja funktion kutsuminen

R:ssä funktioiden argumentteja voi määritellä eri tavoilla, mutta yleisimmässä tapauksessa funktiolla on tietty määrä nimettyjä argumentteja. Edellisen esimerkin `bmi`-funktiolla on kaksi argumenttia, `height` ja `mass`. R-kunktioita voi kutsua monella eri tavalla, ja tutustutaan tähän lisää tämän yksinkertaisen funktion avulla.

Yksi tapa on kutsua funktiota antamalla argumenttien arvot ilman niiden nimiä. HUOM: jos argumentteja ei nimeä, niiden tulee olla oikeassa järjestyksessä. Alla olevan esimerkin toisessa kohdassa argumentit menevät sekaisin.

```
# Call without argument names
bmi(1.65, 62)
```

```
[1] 22.8
```

```
# Arguments in wrong order -> weird results / error
bmi(62, 1.65)
```

```
[1] 0
```

Argumentit voi myös nimetä, kuten edellisissä esimerkeissä tehtiin. Tällöin järjestyksellä ei ole väliä, koska funktiolle on selvää, mitä argumenttia tarkoitetaan.

```
bmi(height = 1.65, mass = 62)
```

```
[1] 22.8
```

```
bmi(mass = 62, height = 1.65)
```

```
[1] 22.8
```

On myös mahdollista nimetä vain osa argumenteista. Tällöin nimeämättömät argumentit asetetaan argumenteiksi “tyhjiin kohtiin” vasemmalta oikealle.

```
bmi(1.65, mass = 62)
```

```
[1] 22.8
```

```
bmi(62, height = 1.65)
```

```
[1] 22.8
```

Jos funktioille yritetään antaa argumentteja, joita ei ole määritelty, seuraa virhe:

```
# Causes error
bmi(height = 1.65, weight = 62)
```

```
Error in bmi(height = 1.65, weight = 62): unused argument (weight = 62)
```

Samoin jos jokin argumentti puuttuu, seuraa virhe:

```
# Causes error
bmi(height = 1.65)
```

```
Error in bmi(height = 1.65): argument "mass" is missing, with no default
```

HUOM: vaikka argumentit saa antaa haluamassaan järjestyksessä ja nimettynä tai nimeämättömänä, kannattaa kuitenkin olla johdonmukainen. Yleisohjeena argumentit kannattaa aina nimetä ja pyrkiä antamaan siinä järjestyksessä, kuin ne on funktiossa määritelty. Näin koodin lukeminen ja ylläpito on paljon helpompaa. Poikkeuksena sääntöön ovat funktiot, joiden toiminta on yksinkertaista, tai joiden ensimmäiset argumentit ovat niin tunnettuja, että niitä ei ole syytä nimetä.

Otetaan esimerkiksi funktio `seq`. Jos avaat funktion help-sivun komennolla `?seq`, näet, että ensimmäiset argumentit ovat nimeltään `from` ja `to`. Koska `seq` on hyvin yleinen ja tunnettu, sitä kutsutaan yleensä niin, että `from` ja `to` jätetään nimeämättä. Muut argumentit, kuten `by` ja `length.out` yleensä nimetään, koska niitä ei aina käytetä, eikä voida olettaa koodin lukijan muistavan, mitä argumenttia tarkoitetaan, vaikka `seq` toimisi ilman nimiä, jos annettaisiin peräkkäin `from`, `to` ja `by`. Vastaavasti `plot`-komennon tapauksessa ei aina kirjoiteta nimiä `x` ja `y`-argumenteille, mutta väriä yms. ohjaavat argumentit nimetään.

11.2.2.1 Oletusarvot (default values)

Monilla R-funktioilla on paljon argumentteja, joista kaikkia ei kuitenkaan tarvitse määrittää erikseen, vaan niillä on oletusarvoja (default values). Esimerkiksi `seq` tekee oletuksena vektorin, joka sisältää kaikki kokonaisluvut `from`-argumentista `to`-argumenttiin. Tätä käyttäytymistä voi kuitenkin muuttaa `by`- ja `length.out`-argumentteja säätämällä.

Tehdään nyt omaan `bmi`-funktioomme uusi argumentti `height_multiplier`, joka saa oletuksena arvon 1. Jos halutaan antaa pituus senttimetreissä metrien sijaan, voidaan asettaa pituuden kertoimeksi 0.01.

```
bmi <- function(height, mass, height_multiplier = 1) {
  # Compute BMI
  value <- mass / (height * height_multiplier)^2
  rounded <- round(value, digits = 1)
  # Return computed value
  return(rounded)
}
bmi(height = 1.65, mass = 62)
```

```
[1] 22.8
```

```
bmi(height = 165, mass = 62, height_multiplier = 0.01)
```

```
[1] 22.8
```

Argumentin oletusarvo merkataan siis funktion määrittelyssä `=`-merkillä, kuten funktion argumenttien anto yleensä. Monilla valmiiden funktioiden argumenteilla on oletusarvona tyhjä arvo eli `NULL`. Tämä tarkoittaa usein, että argumentin voi jättää tyhjäksi, mutta oletusarvon valinta on niin monimutkainen prosessi, että sitä ei voi kirjoittaa funktion määrittelyssä yhdelle riville. Usein tämä tarkoittaa sitä, että oletusarvo riippuu muista argumenteista. HUOM: `NULL` on eri asia kuin `NA`, ja käyttäytyy eri tavoin. Aiheesta lisää täällä.

11.2.3 Funktio ilman argumentteja

Joillain funktioilla ei ole ollenkaan argumentteja. Esimerkiksi R:n sisäiset funktiot `Sys.time` ja `Sys.Date` palauttavat tämänhetkisen ajan ja päivämäärän, eivätkä tarvitse argumentteja.

```r
Sys.time()
```

```
[1] "2024-08-21 16:35:55 EEST"
```

Itse tehdyt funktiot voivat myös toimia ilman argumentteja. Niitä käytetään usein R-istunnon tilan, koodia ajavan tietokoneen ominaisuuksien tai ajan selvittämiseen. Tämä melko hyödytön esimerkkifunktio palauttaa tämän dokumentin kirjoittajan nimen:

```r
author <- function() {
  return("Anton Klåvus")
}
author()
```

```
[1] "Anton Klåvus"
```

11.2.4 Usean arvon palautus

R-funktiot palauttavat aina yhden objektin. Palautukseen käytetään funktiota `return`, kuten aiemmin on nähty. Jos funktiosta halutaan ulos useampi objekti, on muodostettava esimerkiksi lista, joka sisältää halutut objektit. Jos siis `bmi`-funktiosta haluttaisiin palauttaa sekä pyöristetty, että alkuperäinen BMI:n arvo, voidaan ne palauttaa listassa:

```r
bmi_list <- function(height, mass, height_multiplier = 1) {
  # Compute BMI
  value <- mass / (height * height_multiplier)^2
  rounded <- round(value, digits = 1)
  # Return computed value
  values <- list(
    original = value,
    rounded = rounded
  )
  return(values)
}
result <- bmi_list(1.65, 62)
result
```

```
$original
[1] 22.77319

$rounded
[1] 22.8
```

```
result$rounded
```

```
[1] 22.8
```

11.2.5 Palautus ilman return-käskyä

R on siitä erikoinen ohjelmointikieli, että R-funktiot voivat palauttaa arvoja myös ilman eksplisiittistä `return`-käskyä. Jos R-funktiossa ei ole `return`-käskyä, ja viimeinen rivi on vain muuttuja, tai sijoitus muuttujaan, tämän muuttujan arvo palautetaan automaattisesti. `bmi`-funktion voisi siis kirjoittaa myös näin:

```
bmi <- function(height, mass, height_multiplier = 1) {
  # Compute BMI
  value <- mass / (height * height_multiplier)^2
  rounded <- round(value, digits = 1)
  # Return computed value
  rounded
}
bmi(1.65, 62)
```

```
[1] 22.8
```

Yleensä on kuitenkin hyvä käyttää `return`-käskyä, niin pysyy paremmin perässä siitä, mitä koodi tekee, eikä sen kirjoittaminen ole kokeneellekaan ohjelmoijalle huono tapa.

11.2.6 Funktio ilman tulosta

Funktion tarkoitus ei ole aina palauttaa jotain. Yleisiä esimerkkejä ovat `cat` ja `plot`, jotka tulostavat ja piirtävät asioita. Jos näiden funktion paluuarvon yrittää sijoittaa muuttujaan, on tuloksena `NULL`, eli tyhjä arvo.

```
cat_return <- cat("What does cat return?\n")
```

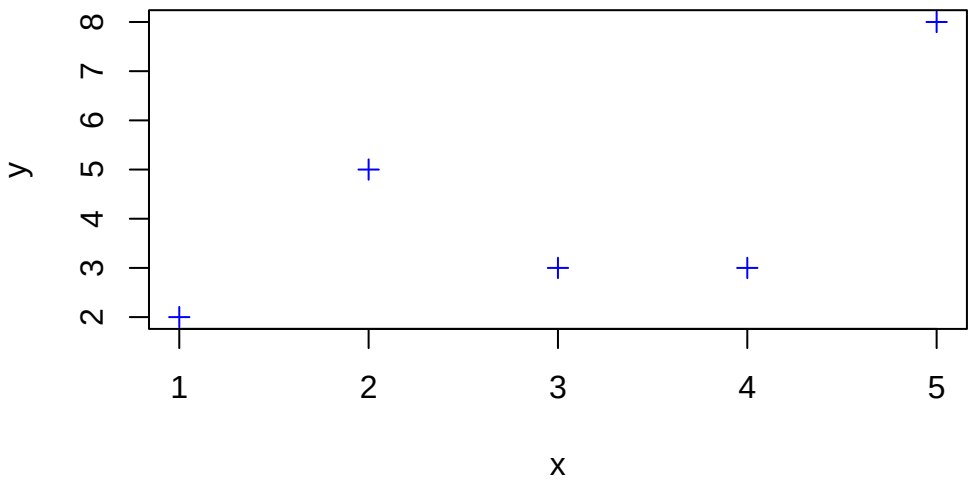
```
What does cat return?
```

```
cat_return
```

```
NULL
```

Itse tehty funktio palauttaa `NULL`, jos viimeinen komento palauttaa `NULL`:

```
# Function for plotting blue squares
blue_squares <- function(x, y) {
  plot(x, y, pch = 3, col = "blue")
}
value <- blue_squares(1:5, c(2, 5, 3, 3, 8))
```




```
value
```

```
NULL
```

11.2.7 Funktion lyhytmuoto

Yksinkertaisten (tyypillisesti yhden rivin) funktioiden tapauksessa aaltosulkuja ei ole pakollista käyttää funktion koodin rajaamiseen. Tarkastellaan esimerkkifunktiota, joka laskee puuttuvien havaintojen `NA` määrän vektorista `x`.

```
count_missing <- function(x) {
  mis <- is.na(x)
  count <- sum(mis)
  return(count)
}
```

Purkamalla välivaiheet auki ja jättämällä `return`-käsky pois, voitaisiin funktio kirjoittaa lyhytmuodossa seuraavasti:

```
count_missing <- function(x) sum(is.na(x))
```

11.2.8 Anonyymi funktio

Jos funktiota tarvitaan vain yksittäiseen laskutoimitukseen, voidaan se määritellä myös ilman muuttujaa, johon funktio tallennettaisiin. Tällöin kyse on ns. anonyymistä funktiosta (inline function, anonymous function, lambda function). Anonyymi funktio määritellään kirjoittamalla `\(x)` jonka perään kirjoitetaan funktion lauseke, esim. funktio joka laskee vektorin `x` neliöt voitaisiin kirjoittaa muodossa

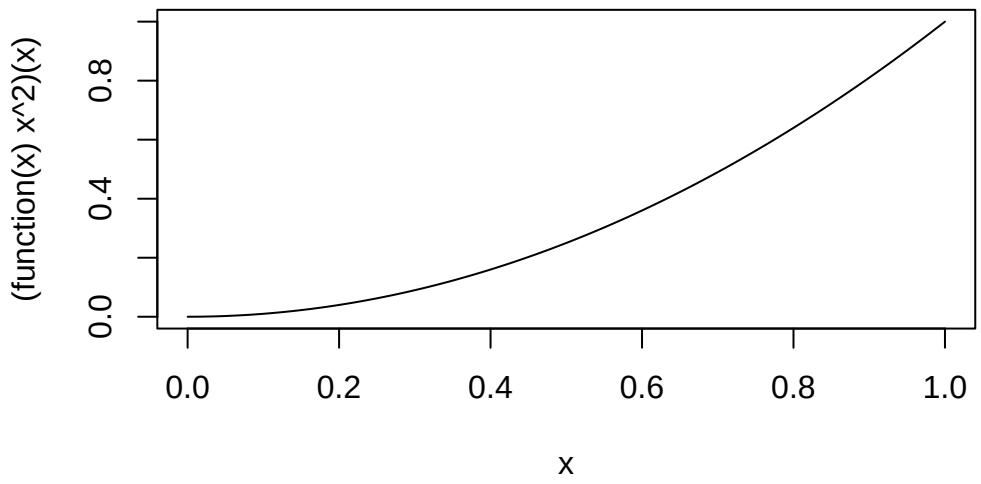
```
\(x) x^2
```

ja sitä voidaan käyttää kuten muitakin funktioita

```
(\(x) x^2)(3)
```

```
[1] 9
```

```
curve((\(x) x^2)(x))
```



Anonyymeillä funktioilla voi myös olla useampi argumentti

```
(\(x, y) x + y)(3, 4)
```

```
[1] 7
```

Huom! Anonyymit funktiot vaativat R-version 4.1 tai sitä uudemman.

12 Ehtorakenteet

Funktiot-luvun funktiot suorittavat aina samat komennot riippumatta syötteestä. Entä jos funktion toiminnassa pitäisi ottaa huomioon erilaisia tapauksia, eli suorittaa tiettyjä komentoja vain joissain tilanteissa? Tätä varten ohjelmointikielissä on ehtorakenteita, eli ns. if/else- tai kontrollirakenteita, jotka ohjaavat ohjelman toimintaa.

Tutustutaan ensin tarkemmin loogisiin operaattoreihin, joita tarvitaan ehdollisten tilanteiden määrittelyyn.

12.1 Loogiset operaattorit

Tässä on lyhyt lista loogisista operaattoreista R:ssä

Operaattori	Toiminto
<	pienempi kuin
<=	pienempi tai yhtä suuri kuin
>	suurempi kuin
>=	suurempi tai yhtä suuri kuin
==	yhtä kuin
!=	ei yhtä kuin
!	ei (negaatio)
|	looginen “tai” alkioittain
||	looginen “tai” yksittäisille arvoille
&	looginen “ja” alkioittain
&&	looginen “ja” yksittäisille arvoille
%in%	mitkä vasemman puolen alkiot ovat myös oikean puolen alkioita?

Kaikki loogiset operaattorit palauttavat joko arvon (tai vektorin arvoja) `TRUE`, `FALSE` tai `NA`. Vertailuoperaattorien käyttö on jo tullut tutuksi aikaisemmissa luvuissa, mutta tutustutaan vielä tarkemmin viimeisten rivien operaattoreihin.

12.1.0.1 Negaatio

Looginen negaatio palauttaa loogisen lauseen vastakohdan, eli muuttaa arvon `TRUE` arvoksi `FALSE` ja arvon `FALSE` arvoksi `TRUE`.

```
10 > 12
```

```
[1] FALSE
```

```
!(10 > 12)
```

```
[1] TRUE
```

```
# Also works without parentheses
!10 > 12
```

```
[1] TRUE
```

```
!is.na(NA)
```

```
[1] FALSE
```

12.1.0.2 Looginen "tai" (disjunktio)

Loogiselle tai-operaattorille annetaan kaksi loogista lausetta, ja tai-operaattori palauttaa `TRUE`, jos vähintään toinen lauseista on `TRUE`. R:ssä tai-operaattori merkitään pystyviivalla `|` tai kahdella pystyviivalla `||`. Näistä `|` käy läpi vektoreita alkioittain, `||` vertaa kahta loogista lausetta, ja toista lausetta ei suoriteta, jos ensimmäinen on `TRUE` (koska `||` palauttaa `TRUE` riippumatta toisen lauseen arvosta). Jos tämä tuntui monimutkaiselta, niin riittää muistaa, että ehtorakenteissa kannattaa käyttää muotoa `||`.

```
10 > 12 || "a" < "b"
```

```
[1] TRUE
```

```
2 > 1 || 4 > 2
```

```
[1] TRUE
```

```
"a" > "c" || 1 > 10
```

```
[1] FALSE
```

12.1.0.3 Looginen “ja” (konjunktio)

Loogiselle ja-operaattorille annetaan kaksi lausetta. Ja-operaattori palauttaa `TRUE`, jos kummatkin lauseet ovat `TRUE`. R:ssä ja-operaattorit ovat `&` ja `&&`, jotka käyttäytyvät kuten `|` ja `||`.

```
10 > 12 && "a" < "b"
```

```
[1] FALSE
```

```
2 > 1 && 4 > 2
```

```
[1] TRUE
```

```
"a" > "c" && 1 > 10
```

```
[1] FALSE
```

12.1.0.4 Osajoukko

`%in%`-operaattorilla voi tarkistaa, kuulvatko jotkin arvot johonkin toiseen joukkoon arvoja. Tämä voitaisiin toteuttaa myös usealla tai-operaattorilla, mutta `%in%` on usein paljon kätevämpi.

```
dna_bases <- c("A", "C", "G", "T")
rna_bases <- c("A", "C", "G", "U")

"T" %in% dna_bases
```

```
[1] TRUE
```

```
"T" %in% rna_bases
```

```
[1] FALSE
```

```
# With negation
!"A" %in% dna_bases
```

```
[1] FALSE
```

Operaattoria voi soveltaa myös vektoreihin, jolloin operaattori palauttaa loogisen vektorin, jonka jokainen alkio kertoo, kuuluiko vastaava operaation vasemman puolen alkio operaation oikeaan puoleen.

```
dna_bases %in% rna_bases
```

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

12.1.0.5 Monimutkaisemmat lauseet

Operaattoreita voidaan myös yhdistellä monimutkaisemmiksi lauseiksi. Tällöin lauseiden evaluointijärjestys määritetään tarvittaessa suluilla.

```
dog <- list(
  breed = "golden retriever",
  height = 45,
  weight = 27
)

dog$breed == "golden retriever" && dog$weight < 25 || dog$height < 50
```

```
[1] TRUE
```

12.1.0.6 a < x < b

usein tulee vastaan tilanteita, joissa halutaan tarkistaa, onko jokin luku halutulla välillä. Tämä kirjoitetaan matemaatiisesti esim. näin: $a < x < b$, jossa tarkastetaan, onko x välillä (a, b). Tämä ei kuitenkaan valitettavasti toimi R:ssä, vaan tarkistus pitää jakaa kahteen osaan:

```
# Are x and y between 0 and 1?
x <- 3
y <- 0.3
0 <= x && x <= 1
```

```
[1] FALSE
```

```
0 <= y && y <= 1
```

```
[1] TRUE
```

12.2 Ehtorakenteet

Aloitetaan esimerkistä: tehtävänä on kirjoittaa funktio, jolle annetaan syötteenä potilaan hemoglobiiniarvo. Funktion on tarkoitus hälyttää, jos hemoglobiini laskee alle viitearvojen alarajan 117. Kyseinen funktio voisi näyttää vaikka tältä:

```
hb_alert <- function(hb) {
  if (hb < 117) {
    return("Hemoglobin is low!")
  }
}
```

Funktiolla on siis yksi argumentti, `hb` eli hemoglobiiniarvo. Funktion sisällä on `if`-rakenne. Rakenteessa on kaksi osaa: ehto, ja rakenteen sisäinen koodi. Rakenteen sisäinen koodi ajetaan vain, jos ehto täyttyy. Ehto merkitään `if`-komennon jälkeen sulkeisiin, ja rakenteen sisäinen koodi kirjoitetaan sulkeiden jälkeen aaltosulkeiden sisään. (Jos aaltosulkeiden sisään tulisi vain yksi rivi koodia, aaltoasulkeet voi jättää pois, mutta näissä esimerkeissä käytetään aina aaltoasulkeita).

Kokeillaan, miten funktio toimii eri hemoglobiiniarvoilla:

```
# Nothing happens
hb_alert(130)
# returns alert
hb_alert(110)
```

```
[1] "Hemoglobin is low!"
```

Funktio siis toimii oletetusti, eli se hälyttää vain, jos hemoglobiinitaso on alle 117. Käyttäjän kannalta olisi kuitenkin kätevää saada jonkinlainen palaute myös silloin, kun hemoglobiinitaso on tarpeeksi korkea. Tätä varten voidaan käyttää `else`-komentoa:

```
hb_alert <- function(hb) {
  if (hb < 117) {
    return("Hemoglobin is low!")
  } else {
    return("Hemoglobin is OK")
  }
}

hb_alert(130)
```

```
[1] "Hemoglobin is OK"
```

`else`-komennon jälkeinen koodi siis ajetaan, jos ehto `hb < 117` ei täyty.

Tällä hetkellä funktiomme toimii oikein vain naispotilaille, sillä miehillä hemoglobiiniarvojen alaraja on 134. Lisätään siis funktioomme argumentti `sex` sukupuolta varten ja muokataan funktion toimintaa niin, että se osaa ottaa huomioon sukupuolen. Nyt `if`-rakenteen ehdosta tulee jo hieman monimutkaisempi:

```
hb_alert <- function(hb, sex) {
  if (sex == "female" && hb < 117 || sex == "male" && hb < 134) {
    return("Hemoglobin is low!")
  } else {
    return("Hemoglobin OK")
  }
}

hb_alert(hb = 120, sex = "female")
```

```
[1] "Hemoglobin OK"
```

```
hb_alert(hb = 120, sex = "male")
```

```
[1] "Hemoglobin is low!"
```

Entä jos haluaisimme tulostaa eri varoituksen mies- ja naispotilaille? Tähän tarvitaan `else if`-rakennetta:

```
hb_alert <- function(hb, sex) {
  if (sex == "female" && hb < 117) {
    return("Hemoglobin is low for a female!")
  } else if (sex == "male" && hb < 134) {
    return("Hemoglobin is low for a male!")
  } else {
    return("Hemoglobin OK")
  }
}

hb_alert(hb = 110, sex = "female")
```

```
[1] "Hemoglobin is low for a female!"
```

```
hb_alert(hb = 120, sex = "male")
```

```
[1] "Hemoglobin is low for a male!"
```

Nyt funktio tarkistaa ensin, onko potilas nainen ja onko hänen hemoglobiininsa alle 117. Jos ei, siirrytään eteenpäin ja tarkistetaan, onko potilas mies ja onko hänen hemoglobiininsa alle 130. Jos ei, siirrytään viimeiseen kohtaan, ja tulostetaan "Hemoglobin is OK".

`else if`-rakenteita voi olla rajoittamaton määrä ensimmäisen `if`-rakenteen jälkeen. Lisätään funktioon hälytys kriittisestä hemoglobiinin määrästä (hb $<$ 50) riippumatta sukupuolesta:

```
hb_alert <- function(hb, sex) {
  if (sex == "female" && hb < 117) {
    return("Hemoglobin is low for a female!")
  } else if (sex == "male" && hb < 134) {
    return("Hemoglobin is low for a male!")
  } else if (hb < 50) {
    return("Hemoglobin is critical")
  } else {
    return("Hemoglobin OK")
  }
}

hb_alert(hb = 32, sex = "female")
```

```
[1] "Hemoglobin is low for a female!"
```

Kuten huomataan, yllä oleva koodi ei toimikaan, kuten piti. Näin alhaisella hemoglobiinilla pitäisi tulla varoitus kriittisestä tilasta. Koodi suoritus ei kuitenkaan ikinä etene kriittisen tilan varoitukseen asti, sillä ensimmäinen ehto täyttyy. Korjataan tilanne siirtämällä kriittisen tilan ehto ensimmäiseksi:

```
hb_alert <- function(hb, sex) {
  if (hb < 50) {
    return("Hemoglobin is critical")
  } else if (sex == "male" && hb < 134) {
    return("Hemoglobin is low for a male!")
  } else if (sex == "female" && hb < 117) {
    return("Hemoglobin is low for a female!")
  } else {
    return("Hemoglobin OK")
  }
}

hb_alert(hb = 32, sex = "female")
```

```
[1] "Hemoglobin is critical"
```

```
hb_alert(hb = 120, sex = "female")
```

```
[1] "Hemoglobin OK"
```

```
hb_alert(hb = 120, sex = "male")
```

```
[1] "Hemoglobin is low for a male!"
```

Nyt funktio toimii haluamallamme tavalla!

Funktioissa voi myös olla useampi ehtorakenne. Ehtorakenteita käytetään usein tarkistamaan argumenttien arvoja. Lisätään ehtorakenteet argumenttien tarkistamiseksi:

```
hb_alert <- function(hb, sex) {
  # Hemoglobin should be numeric and positive
  if (!is.numeric(hb) || hb < 0) {
    return("Hemoglobin should be numeric and positive")
  }
  if (!sex %in% c("female", "male")) {
```

```
    return("This function can only deal with binary sex: female or male")
  }

  if (hb < 50) {
    return("Hemoglobin is critical")
  } else if (sex == "male" && hb < 134) {
    return("Hemoglobin is low for a male!")
  } else if (sex == "female" && hb < 117) {
    return("Hemoglobin is low for a female!")
  } else {
    return("Hemoglobin OK")
  }
}

hb_alert(hb = "120", sex = "female")
```

```
[1] "Hemoglobin should be numeric and positive"
```

```
hb_alert(hb = 120, sex = "FEMALE")
```

```
[1] "This function can only deal with binary sex: female or male"
```

12.3 Alkioiden poimiminen vektorista tietyn ehdon perusteella

Seuraava tilanne on melko tyypillinen: on käytävä läpi vektorin arvot, ja säilytettävä niistä ne, jotka täyttivät tietyn ehdon. Tätä ongelmaa voi lähestyä esimerkiksi seuraavalla tavalla:

- Luo apufunktio, joka ottaa syötteeksi yhden arvon, ja tarkistaa täyttyykö ehto. Tämän funktion tulee palauttaa `TRUE`, jos ehto täyttyy ja `FALSE`, jos ehto ei täyty.
- Käytä funktiota `Vectorize`, jolla voit muuttaa funktiosi vektoroiduksi funktioksi. Vektorointi tarkoittaa tässä yhteydessä sitä, että yhden alkion sijaan vektoroitua funktiota voidaankin kutsua vektoriargumentilla, ja jokaiselle argumentin alkiolle suoritetaan alkuperäisen funktion määrittelemä operaatio.
- Käytä vektoroitua apufunktiota vektorin indeksointiin.

Tässä on esimerkki, jossa käydään läpi vektori DNA:n emäksiä, joista poimitaan vain sytosiinit ja guaniinit.

```r
# Helper function
is_cg <- function(base) {
  if (base %in% c("C", "G")) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

# Vectorize
is_cg_vector <- Vectorize(is_cg)

# Main function
pick_cg <- function(bases) {
  only_cg <- bases[is_cg_vector(bases)]
  return(only_cg)
}

# NOTE: this only checks the first value of the vector
my_bases <- c("A", "C", "C", "T", "G", "T")
#is_cg(my_bases) # This produces error in 4.2.x

# This works as expected
is_cg_vector(my_bases)
```

```
    A     C     C     T     G     T
FALSE  TRUE  TRUE FALSE  TRUE FALSE
```

```r
# Pick only C and G
pick_cg(my_bases)
```

```
[1] "C" "C" "G"
```

13 Toistorakenteet (loops)

Toistorakenne toistaa annettua koodia. Toistorakenteet ovat ehtorakenteiden ohella ohjelmoinnin perusrakennuspalikoita. Tässä osiossa tutustutaan kahteen yleisimpään tapaukseen eli `for` ja `while` -silmukoihin. Mukana on myös maininta silmukoiden korvaamisesta R:n `apply`-funktioilla.

13.1 For-silmukka

For-silmukka toistaa koodia ennalta määrättyjen iteraatioiden verran. For-silmukalla voi esimerkiksi käydä läpi datakehikon tai matriisin sarakkeita tai rivejä, tai vektorin arvoja. For-silmukka iteroi aina jonkin järjestetyn rakenteen yli, esimerkiksi vektorin. For-silmukalle annetaan tyypillisesti vektori arvoja, ja ns. iteraatiomuuttuja, johon tallennetaan vuorotellen yksi alkio annetusta vektorista. Käytännössä tämä näyttää tältä:

```
for (i in seq(3, 7)) {
  print(i)
}
```

For-silmukassa määritetään siis ensin iteraatiomuuttuja eli `i` ja sen saamat arvot eli `seq(3, 7)` komennolla `in`. Sen jälkeen hakasulkeiden sisältämä koodi toistetaan jokaiselle `i`:n arvolle järjestyksessä. Tässä tapauksessa yksinkertaisesti tulostetaan muuttujan `i` arvo. Huomaa, että for-silmukan `in` ei ole sama asia kuin looginen operaattori `%in%`. `in` on R-kielen varattu symboli, jolloin esimerkiksi muuttuujaa nimeltä `in` ei ole mahdollista luoda.

Usein halutaan käydä läpi jonkin vektorin tai matriisin arvoja. Alla oleva koodi laskee matriisin `X` rivien summan (tähän voisi myös käyttää valmista funktiota `rowSums()`). Aluksi alustetaan tyhjä vektori, johon rivien summat tallennetaan. Sen jälkeen käydään läpi matriisin rivit ja tallennetaan rivin summa alussa alustettuun vektoriin.

```
# Create matrix X
X <- matrix(1:12, nrow = 4)
X

# Initialize vector for row sums
```

```
row_sums <- rep(0, nrow(X))
# Iterate over rows of X
for (i in seq(1, nrow(X))) {
  # Assign sum of the current row to the vector
  row_sums[i] <- sum(X[i, ])
}

# Compare results with the result from base R function
row_sums
rowSums(X)
```

For-silmukalla voi myös toteuttaa Ehtorakenteet-luvussa tehdyn funktion, joka poimii DNA:n emäksistä vain sytosiinit ja guaniinit. Tällä kertaa apufunktiota `is_cg()` ei tarvitse vektorisoida, koska for-silmukka käy läpi kaikki emäkset. Tämä silmukka voidaan toteuttaa kahdella tavalla. Ensimmäinen tapa on käyttää iteraatiomuuttujana `i`:tä, joka käy läpi iteraation ykkösestä emäsvektroin pituuteen:

```
# Helper function
is_cg <- function(base) {
  if (base %in% c("C", "G")) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

# Main function
pick_cg1 <- function(bases) {
  # Initialize empty vector
  only_cg <- c()
  for (i in seq(1, length(bases))) {
    # If the current base is C or G, add it to only_cg
    if (is_cg(bases[i])) {
      only_cg <- c(only_cg, bases[i])
    }
  }
  return(only_cg)
}

my_bases <- c("A", "C", "C", "T", "G", "T")
pick_cg1(my_bases)
```

Toinen vaihtoehto on iteroida suoraan vektorin `bases` yli, jolloin iteraatiomuuttujaan tallentuu suoraan kyseinen emäs:

```
pick_cg2 <- function(bases) {
  # Initialize empty vector
  only_cg <- c()
  for (base in bases) {
    # If the current base is C or G, add it to only_cg
    if (is_cg(base)) {
      only_cg <- c(only_cg, base)
    }
  }
  return(only_cg)
}

my_bases <- c("A", "C", "C", "T", "G", "T")
pick_cg2(my_bases)
```

Iteraatiomuuttujan voi siis nimetä haluamallaan tavalla, sen ei aina tarvitse olla `i`. Jos kuitenkin iteraatiomuuttujaan tallennetaan vain yksi luku, on suositeltavaa käyttää `i`:tä. Tämä on hyvin vakiintunut tapa ohjelmointikielestä ja ohjelmoijasta riippumatta, vaikka muutoin muuttujien nimeämiseen on erilaisia koulukuntia riippuen ohjelmoijan taustasta. Jos taas iteroidaan vektorin nimeltä `bases` yli, on luonnollinen valinta iteraatiomuuttujan nimeksi tässä tapauksessa `base`.

13.2 While-silmukka

While-silmukkaa käytetään, kun iteraatioiden määrä ei ole ennalta tiedossa, vaan while-silmukkaa toistetaan niin kauan, kuin tietty ehto on voimassa. Hyvä esimerkki while-silmukasta on proteiinisynteesi (yksinkertaistettuna): alla oleva funktio käy läpi RNA-molekyylin kodoneita, kunnes löytää aloituskodonin AUG. Sen jälkeen funktio rakentaa aminohappoketjua kodonien perusteella, kunnes vastaan tulee jokin lopetuskodoneista. Oikean proteiinin löytämiseen käytetään Biostrings-paketista löytyvää geneettistä koodia, joka on nimetty vektori, jossa on kodoneita vastaavien aminohappojen kirjainlyhenne, tai lopetuskodonien tapauksessa merkki “*“:

```
rna_code <- Biostrings::RNA_GENETIC_CODE
rna_code
```

```r
prot_synth <- function(codons) {
  # Initialize iterable as the first codon
  i <- 1
  # Initialize empty amino acid chain
  protein <- c()
  # Find starting codon
  while (codons[i] != "AUG") {
    i <- i + 1
  }
  # After starting codon, build protein until one of the stop codons is met
  while (rna_code[codons[i]] != "*") {
    protein <- c(protein, rna_code[codons[i]])
    i <- i + 1
  }
  return(protein)
}

prot_synth(
  codons = c("UUG", "GAA", "AUG", "UGU", "AGU", "AGA", "UCG", "UCG", "UGA", "GCA")
)
```

While-silmukalle annetaan siis ensin ehto, joka tarkistetaan ennen jokaista iteraatiota. Jos ehto täyttyy, suoritetaan yksi iteraatio, ja tarkistetaan ehto uudestaan. HUOM: while-silmukkaa koodatessa tulee huolehtia siitä, että silmukan ehdon on mahdollista olla lopulta epätosi, muuten silmukka saattaa jäädä pyörimään ikuisesti!

Käytännössä kaikki for-silmukat voisi korvata while-silmukoilla, mutta for-silmukoiden käyttö on kätevämpää, sillä niissä iteraatiomuuttujaa ei tarvitse kasvattaa erikseen.

```r
# A simple for loop
for (i in seq(1, 4)) {
  print(i * 2)
}

# The same as above
i <- 1
while (i <= 4) {
  print(i * 2)
  i <- i + 1
}
```

13.3 Sisäkkäiset silmukat (nested loops)

Silmukoita voi myös olla useampi sisäkkäin. Alla olevassa esimerkissä on taulukko tutkimuksesta, jossa on mitattu eri eliöiden β-globiinigeenin ensimmäisen eksonin samankaltaisuutta. Pienempi luku tarkoittaa enemmän samankaltaista geeniä.

	Human	Goat	Opossum	Lemur	Mouse	Rabbit	Gorilla
Human	0.0	4.7	4.6	2.7	3.2	3.2	1.6
Goat	4.7	0.0	7.2	5.9	7.8	3.7	5.5
Opossum	4.6	7.2	0.0	5.3	5.3	6.3	5.7
Lemur	2.7	5.9	5.3	0.0	4.3	2.7	3.2
Mouse	3.2	7.8	5.3	4.3	0.0	6.0	2.9
Rabbit	3.2	3.7	6.3	2.7	6.0	0.0	3.8
Gorilla	1.6	5.5	5.7	3.2	2.9	3.8	0.0

Tämä data on hakemiston `data` tiedostossa `exons.csv`, joten luetaan se R:ään:

```
exons <- read.csv("data/exons.csv", row.names = 1)
```

Etsitään seuravaksi kaikki eliöparit, joiden geenien etäisyys on alle 4 ja lisätään parit datakehikkoon, jossa on kaksi saraketta, ja jokainen rivi edustaa yhtä eliöparia. Käytetään tähän kahta sisäkkäistä for-silmukkaa. Toisen silmukan iteraatiomuuttujan nimi on yleensä `j`, seuraavan `k` ja niin edelleen. Käydään exons läpi niin, että `i` on rivin numero, ja `j` sarakkeen numero, ja etsitään sopivat parit.

```
# Initialize empty data frame for the pairs
close_pairs <- data.frame()

# Iterate over rows and columns
for (i in seq(1, nrow(exons))) {
  for (j in seq(1, ncol(exons))) {
    # Check if dissimilarity is below 4
    if (exons[i, j] < 4) {
      # Add the pair as a new row to close_pairs
      new_row <- data.frame(
        Species_1 = rownames(exons)[i],
        Species_2 = colnames(exons)[j]
      )
      close_pairs <- rbind(
        close_pairs,
        new_row
```

```
      )
    }
  }
}

close_pairs
```

Koodimme toimii jo ihan hyvin, mutta tuloksessa on hieman turhaa tavaraa: `exons` on symmetrinen, joten monet parit on esitetty tuloksessa kahdesti. Tämä voidaan ratkaista muuttamalla toista for-silmukkaa:

```
# Initialize empty data frame for the pairs
close_pairs <- data.frame()

# Iterate over rows and columns
for (i in seq(1, nrow(exons))) {
  # Only check upper diagonal
  for (j in seq(i, ncol(exons))) {
    # Check if dissimilarity is below 4
    if (exons[i, j] < 4) {
      # Add the pair as a new row to close_pairs
      new_row <- data.frame(
        Species_1 = rownames(exons)[i],
        Species_2 = colnames(exons)[j]
      )
      close_pairs <- rbind(
        close_pairs,
        new_row
      )
    }
  }
}

close_pairs
```

Nyt toisen silmukan läpi käymät `j`:n arvot riippuvat `i`:n arvosta. Tämä koodi käy läpi taulukon yläkolmion, eli diagonaalin yläpuolella olevat alkiot. Ensimmäisellä kierroksella `j` käy läpi arvot 1–7, seuraavalla kierroksella 2–7, sitten 3–7 jne. Vastaavasti voitaisiin myös käydä läpi alakolmio komennolla `for(j in seq(1, i))`.

Emme kuitenkaan voi olla vieläkään tyytyväisiä tulokseen, sillä mukana ovat “parit”, joissa kumpikin laji on sama. Näistä emme luonnollisesti ole kiinnostuneita. Nämä parit voidaan poistaa esimerkiksi `next`-komennolla.

13.4 Iterointiin puuttuminen: `next` ja `break`

Joskus silmukan toimintaan on hyvä puuttua kesken suorituksen. Joskus yksi iteraatio halutaan sivuuttaa kokonaan, toisinaan taas halutaan keskeyttää koko silmukka. Näihin tarkoituksiin R:ssä ovat komennot `next` ja `break`.

Lisätään edelliseen esimerkkiin toiminto, joka ohittaa diagonaalilla olevat rivit, eli hyppää iteraation yli, jos `i` ja `j` ovat yhtä suuret. Käytetään tähän `next`-komentoa, joka ohjaa ohjelman suoraan seuraavaan iteraatioon sen silmukan suhteen, jonka sisällä komento on:

```
# Initialize empty data frame for the pairs
close_pairs <- data.frame()

# Iterate over rows and columns
for (i in seq(1, nrow(exons))) {
  # Only check upper diagonal
  for (j in seq(i, ncol(exons))) {
    if (i == j) {
       next
    }
    # Check if dissimilarity is below 4
    if (exons[i, j] < 4) {
      # Add the pair as a new row to close_pairs
      new_row <- data.frame(
        Species_1 = rownames(exons)[i],
        Species_2 = colnames(exons)[j]
      )
      close_pairs <- rbind(
        close_pairs,
        new_row
      )
    }
  }
}

close_pairs
```

Nyt pääsimme eroon kaikista turhista pareista! Jos haluaisimme kaikkien parien sijaan etsiä vain ensimmäisen parin, jonka geenien etäisyys on alle 3, voisimme käyttää komentoa `break`, joka keskeyttää silmukan turhan suorittamisen haluamamme parin löydyttyä.

```
close_pair <- c()

# Iterate over rows and columns
for (i in seq(1, nrow(exons))) {
  # Only check upper diagonal
  for (j in seq(i, ncol(exons))) {
    if (i == j) {
       next
    }
    # Check if dissimilarity is below 3
    if (exons[i, j] < 3) {
      # Assign pair to close_pair and stop search
      close_pair <- c(
        Species_1 = rownames(exons)[i],
        Species_2 = colnames(exons)[j]
      )
      break
    }
  }
}

close_pair
```

HUOM: Tämä ei kuitenkaan ole oikea pari: Jos `exons` datakehikkoa käydään läpi rivi kerrallaan, ensimmäinen pari, jonka arvo on alle 3 on Human ja Lemur, ei Mouse ja Gorilla. Mikä siis meni väärin? Kun kyse on näin pienestä aineistosta, voidaan mahdollisia ongelmia tutkia lisäämällä silmukoiden sisään `print()`-komentoja, jotka kertovat meille silmukan etenemisestä. Lisätään siis edelliseen silmukkaan rivi, joka tulostaa iteraatiomuuttujat `i` ja `j` jokaisella iteraatiolla, sekä rivi, joka tulostaa uuden parin, kun sellainen löytyy:

```
close_pair <- c()

# Iterate over rows and columns
for (i in seq(1, nrow(exons))) {
  # Only check upper diagonal
  for (j in seq(i, ncol(exons))) {
    # Monitor loop
    print(c(i, j))
    if (i == j) {
      next
    }
    # Check if dissimilarity is below 3
```

```
    if (exons[i, j] < 3) {
      # Assign pair to close_pair and stop search
      close_pair <- c(
        Species_1 = rownames(exons)[i],
        Species_2 = colnames(exons)[j]
      )
      print(close_pair)
      break
    }
  }
}

close_pair
```

Nyt huomataan, että iteraatio etenee rivillä yksi neljänteen sarakkeeseen asti, ja löytää parin Human-Lemur, aivan kuten pitikin. Jostain syystä ohjelma siirtyy kuitenkin sen jälkeen toiselle riville. Tämä johtuu siitä, että `break`-komento katkaisee vain yhden for-silmukan kerrallaan. Jos haluamme katkaista myös ulomman silmukan, meidän tulee lisätä ulomman silmukan loppuun tarkastus, joka tarkastaa, onko pari jo löytynyt. Tämä voidaan testata esimerkiksi vektorin `close_pairs` pituuden avulla. Jos `if`-rakenteelle antaa pelkän luvun, luku tulkitaan arvoksi `TRUE`, jos se ei ole nolla.

```
close_pair <- c()

# Iterate over rows and columns
for (i in seq(1, nrow(exons))) {
  # Only check upper diagonal
  for (j in seq(i, ncol(exons))) {
    if (i == j) {
       next
    }
    # Check if dissimilarity is below 3
    if (exons[i, j] < 3) {
      # Assign pair to close_pair and stop search
      close_pair <- c(
        Species_1 = rownames(exons)[i],
        Species_2 = colnames(exons)[j]
      )
      break
    }
  }
  # Stop iterating if the pair has been found
```

```
  if (length(close_pair)) {
    break
  }
}

close_pair
```

Nyt koodimme toimii, kuten pitääkin!

13.5 Apply-funktiot

R:ssä käytetään silmukoiden lisäksi `apply`-funktioperheen funktioita, joilla voi käydä läpi datakehikkoja, matriiseja tai vektoreita ilman silmukoita. Joissain tapauksissa `apply`-funktiot ovat myös nopeampia kuin silmukat. Tästä syystä niitä näkee käytettävän paljon, ja varsinkin kokeneemmat R-ohjelmoijat käyttävät niitä usein silmukoiden sijaan. Tällä kurssilla näitä funktioita ei kuitenkaan tarvita. Tässä on annettu muutamia esimerkkejä, voit lukea lisää esimerkiksi DataCampin tutoriaalista

`apply` käy läpi matriisin/data framen rivit tai sarakkeet, ja ajaa jonkin funktion jokaiselle riville tai sarakkeelle. Alla oleva esimerkki standardoi kaikki R:n sisäisen datan `trees` sarakkeet siten, että sarakkeen arvoista vähennetään sarakkeen keskiarvo ja tulos jaetaan sarakkeen keskihajonnalla. Standardoinnin tarkoitus on, että kaikkien sarakkeiden keskiarvoksi saadaan 0, ja kaikilla on sama varianssi (ja keskihajonta) 1.

```
head(trees)

scaler <- function(x) {
  scaled <- (x - mean(x)) / sd(x)
  scaled
}

scaled_trees <- apply(X = trees, MARGIN = 2, FUN = scaler)
scaled_trees <- as.data.frame(scaled_trees)
head(scaled_trees)
```

`MARGIN`-argumentilla määritetään, käydäänkö läpi rivit vai sarakkeet (1 = rivit, 2 = sarakkeet, moniulotteisten taulujen tapauksissa myös muut dimensiot ovat mahdollisia). HUOM: `apply` palauttaa aina matriisin tai vektorin. Jos tulos halutaan muuntaa takaisin datakehikoksi, täytyy se tehdä erikseen.

Tarkistetaan tulos laskemalla sarakkeiden keskiarvot ja varianssit. Tämä voidaan tehdä `apply`-funktiolla, tai käyttää `sapply`-funktiota, joka käy automaattisesti datakehikon sarakkeet, ja ajaa saman funktion sarakkeille kuten `apply`.

```
apply(scaled_trees, 2, mean)
sapply(scaled_trees, var)
```

Huomaa, että sarakkeiden keskiarvot eivät ole täsmälleen 0. Tämä johtuu R:n rajallisesta numeerisesta tarkkuudesta. Käytännössä itseisarvoltaan tätä luokkaa olevat arvot ovat nollia.

14 Numeeriset menetelmät

HUOM! Tätä osiota ei tarvitse opiskella Itä-Suomen yliopiston kurssilla!

Monilla käytännön matemaattisilla ongelmilla ei ole suljetussa muodossa esitettävissä olevaa ratkaisua. Tällöin joudutaan tyypillisesti turvautumaan numeerisiin menetelmiin, joiden avulla pyritään tuottamaan likiarvoinen ratkaisu ongelmaan. R:stä löytyy useita valmiita funktioita erilaisiin numeerista laskentaa vaativiin ongelmiin. Tyypillisimpiä tapauksia ovat jonkin funktion minimin tai maksimin etsiminen, funktion juurten etsintä ja integrointi.

14.1 Optimointi

14.1.1 Yksi parametri

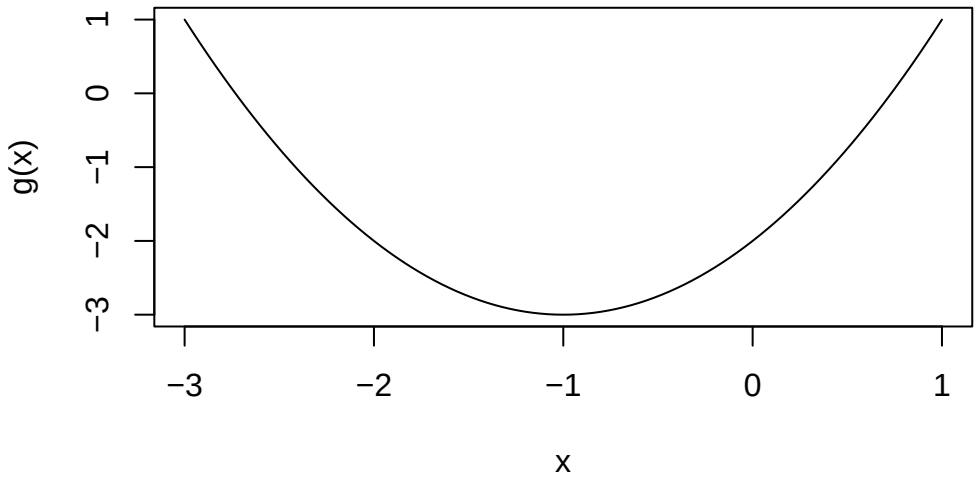
Aloitetaan yksinkertaisesta tapauksesta, jossa haluamme löytää funktion minimin yhden parametrin suhteen. Tällöin voidaan käyttää funktiota `optimize`.

```
optimize(f, interval, ..., lower = min(interval), upper = max(interval),
         maximum = FALSE, tol = .Machine$double.eps^0.25)
```

Ensimmäinen argumentti `f` on funktio, jota minimoidaan sen ensimmäisen argumentin suhteen (jos funktiolla on muita argumenttej, joita tarvitaan, tulee ne antaa mukana `optimize` funktiokutsussa). Välin, jolta minimipistettä haetaa, voi ilmoittaa joku argumentilla `interval`, joka on vektori sisältäen välin päätepisteet. Vaihtoehtoisesti välin ylä- ja alaraja voidaan ilmoittaa erikseen argumenteilla `lower` ja `upper`, vastaavasti. Mikäli etsittäisiinkin minimin sijaan maksimia, tulisi asettaa myös argumentti `maximum = TRUE`.

Etsitään nyt funktion $g(x) = x^2 + 2x + 2$ minimi. Hakuvälin `interval` valitsemiseksi voimme esimerkiksi piirtää ensin funktion kuvaajan, jotta saamme suurin piirtein selville, missä minimi mahdollisesti sijaitsee:

```
g <- function(x) x^2 + 2*x -2
curve(g, xlim = c(-3, 1))
```



Kuvan perusteella minimiarvo saavutetaan pisteessä $x = -1$. Käytetään nyt `optimize` funktiota:

```
optimize(g, interval = c(-3, 1))
```

```
$minimum
[1] -1

$objective
[1] -3
```

Funktion palauttmassa listassa alkio `minimum` ilmoittaa pisteen, jossa minimi saavutetaan. Alkio `objective` antaa tavoitefunktion (eli funktion `f`) arvon kyseisessä psiteessä.

14.1.2 Useampi parametri

Mikäli funktiota halutaan minimoida useamman kuin yhden parametrin suhteen, voidaan käyttää funktiota `optim`.

```
optim(par, fn, gr = NULL, ...,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

Ensimmäinen argumentti `par` on vektori, joka antaa alkuarvot jokaiselle parametrille, jonka suhteen minimointia halutaan tehdä. Seuraava argumentti `fn` on minimoivata funktio, jonka ensimmäisen argumentin tulee vastata argumenttia `par` (vektori, jossa on

yhtä monta alkiota). Vastaavasti kuten `optimize`-funktiossa, argumentit `lower` ja `upper` määrittävä alueen, jolta minimiä etsitään. Huomaa kuitenkin, että koska funktiolla `fn` on nyt useampi parametri, ovat `lower` ja `upper` myös vektoreita jotka ilmoittavat rajat jokaiselle parametrille erikseen. Alkuarvojen `par` on myös toteutettava mahdolliset rajoitteet. Argumentti `method` valitsee käytettävän optimointimenetelmän. Metelmistä riittää tietää tässä vaiheessa se, että jos optimointia halutaan tehdä käyttäen rajoitteita (`lower` ja `upper`), voidaan menetelmäksi valita "L-BFGS-B", muuten voidaan käyttää oletusarvoa. Muista `optim`-funktion argumenteista ei tämän kurssin puitteissa tarvitse välittää.

Etsitään funktion $f(x, y) = y^2 \exp(-0.5(y^2 + x^2))$ lokaali maksimi joukossa $-1 < x < 3$, $-1 < y < 3$. Annetaan alkuarvoiksi $x = 0.5$ ja $y = 0.5$. `optim`-funktio etsii oletusarvoisesti funktion minimiä, joten vaihtamalla funktion merkki etsitäänkin maksimia. Huomaa, että funktiolla `f` on vain yksi argumentti `x`, vaikka funktiolla f on kaksi argumenttia, x ja y. Tämä johtuu siitä, että `optim`-funktion tapauksessa parametrien `par` on esiinnyttävä funktion argumenteissa vektorina. Vektorin `x` ensimmäinen alkio `x[1]` vastaa siis muuttujaa x ja toinen alkio `x[2]` vastaa muuttujaa y. Tämä yleistyy useamman kuin kahden muuttujan funktioille, kun vektorin `x` pituutta kasvatetaan vastaavasti (esim. kolmas muuttuja z olisi `x[3]` jne.).

```
f <- function(x) -x[2]^2 * exp(-0.5 * (x[2]^2 + x[1]^2))
optim(c(0.5, 0.5), f, lower = c(-1, -1), upper = c(3, 3), method = "L-BFGS-B")
```

```
$par
[1] -7.582426e-10  1.414214e+00

$value
[1] -0.7357589

$counts
function gradient
       8        8

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Funktion palauttamassa tulosteessa `par` kertoo maksimipisteen koordinaatit. Ensimmäinen alkio kertoo maksimipisteen x-koordinaatin, ja toinen sen y-koordinaatin (huomioi erityisesti 1. alkion merkintätapa `-7.582426e-10` joka tarkoittaa samaa kuin $-7.582426 \cdot 10^{-10}$, eli noin 0.00000000076, eli x koordinaatti on siis käytännössä 0). `value` ilmoittaa löydettyä maksimipistettä vastaavan funktion arvon. Koska funktion merkki vaihdettiin maksimin

etsimiseksi, on todellinen maksimiarvo siis löydetyn optimin vastaluku, eli ≈ 0.7357589. Muut tulostukset ovat optimoinnin konvergenssiin liittyviä lisätietoja. Vaihtoehtoisesti maksimia voi etsiä suoraankin vaihtamatta funktion merkkiä antamalla `optim`-funktiolle lisäargumentti `control = list(fnscale = -1)`.

Etsitään vielä kolmen muuttujan funktion $h(x, y, z) = \exp(-x^2 - 3x - 7y^2 + 3y + z^3 - 2z - 3)$ lokaali maksimi joukossa $-2 < x < 2$, $-3 < y < 3$, $-3 < z < 0$.

```
h <- function(x) exp(-x[1]^2 - 3*x[1] - 7*x[2]^2 + 3*x[2] + x[3]^3 - 2*x[3] - 3)
optim(
  c(0.5, 0.5, -0.5), h, lower = c(-2, -3, -3), upper = c(2, 3, 0),
  method = "L-BFGS-B", control = list(fnscale = -1)
)
```

```
$par
[1] -1.5000009  0.2142866 -0.8164968

$value
[1] 1.934968

$counts
function gradient
      22       22

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Kuten edellä, `par` ilmoittaa maksimipisteen koordinaatit. Maksimi saavutetaan siis pisteessä $(x, y, z) \approx (-1.5000009, 0.2142866, -0.8164968)$ jolloin funktio h saa kohdan `value` ilmoittaman arvon ≈ 1.934968. Nyt koska käytettiin argumenttia `control = list(fnscale = -1)`, ei tuloksen merkkiä tarvitse vaihtaa.

Alkuarvot funktioille `optim` ja `optimize` tulee valita siten, että rajoitteet ovat voimassa. Alkuarvojen valintaan on vaikea antaa yleispätevää ohjetta, ja usein onkin hyvä kokeilla eri arvoja ja verrata niillä saatuja tuloksia. Yhden ja kahden muuttujan tapauksissa löydettyjen optimipisteiden mielekkyyttä voi tarkastella esimerkiksi piirtämällä funktion kuvaajan annetussa joukossa.

14.2 Funktion juurten etsintä

Funktion juuria voidaan etsiä funktiolla `uniroot`.

```
uniroot(f, interval, ...,
        lower = min(interval), upper = max(interval),
        f.lower = f(lower, ...), f.upper = f(upper, ...),
        extendInt = c("no", "yes", "downX", "upX"), check.conv = FALSE,
        tol = .Machine$double.eps^0.25, maxiter = 1000, trace = 0)
```

Funktion `f` juuria etsitään annetulta väliltä `interval`, sen ensimmäisen argumentin suhteen (jonka tulee olla skalaari). Halutun välin voi määrittää myös sen päätepisteinä käyttäen argumentteja `lower` ja `upper`. Muut `uniroot`-funktion argumentit eivät ole tämän kurssin kannalta oleellisia. Huomaa, että annetun välin tulee todella sisältää funktion juuri, muuten juurta ei luonnollisesti löydy.

Etsitään funktion $w(x) = x^3 - 2x - 5$ juurta väliltä $(-5, 5)$.

```
w <- function(x) { x^3 - 2*x - 5 }
uniroot(w, interval = c(-5, 5))
```

```
$root
[1] 2.094528

$f.root
[1] -0.0002653143

$iter
[1] 9

$init.it
[1] NA

$estim.prec
[1] 6.103516e-05
```

Funktion palauttamassa tulosteessa kohta `root` ilmoittaa löydetyn juuren. Mikäli juurta ei löydy annetulta väliltä, funktio antaa varoituksen. Kohta `froot` ilmoittaa funktion arvon löydetyssä pisteessä (funktion arvon ja nollan ero riippuu laskennan tarkkuudesta ja käytetystä menetelmästä).

14.3 Numeerinen integrointi

R:n optimointityökaluihin kuuluu myös funktio `integrate`, jolla voi laskea useimpien funktioiden määrättyjä integraaleja.

```
integrate(f, lower, upper, ..., subdivisions = 100L,
          rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol,
          stop.on.error = TRUE, keep.xy = FALSE, aux = NULL)
```

Ensimmäinen argumentti `f` on funktio, jota halutaan integroida. Argumentit `lower` ja `upper` määräävät integrointivälin, jonka päätepisteet voivat olla myös äärettömiä. Tällöin voidaan asettaa `lower = -Inf` tai vastaavasti `upper = Inf`.

Integroidaan funktiota $f(x) = x^2 + 3x - 2$ välin $[-2, 3]$ yli.

```
poly <- function(x) { x^2 + 3*x - 2 }
integrate(poly, -2, 3)
```

```
9.166667 with absolute error < 2.8e-13
```

Integraalin arvo on siis noin 9.17.