Pikakurssi R-ympäristön käyttöön

Tuomas A. Rajala

Versio: 9. syyskuuta 2011

Tämän dokumentin tarkoitus on toimia lyhyenä interaktiivisena esittelynä R-ympäristön käyttöön.

Se on suunniteltu rungoksi Jyväskylän yliopiston Matematiikan ja tilastotieteen laitoksen "R-kurssi" kurssille. Kurssin pituus on 4x4 tuntia. Tämä syksyn 2011 versio on pienin muutoksin sama kuin kevään 2011 versio.

1 Osa 1: Johdanto

1.1 Alkuaskeleet: Mikä R?

R on ilmainen ja yhteisövetoinen tietokoneohjelma. R on kaupallisen esikuvansa S-tietokoneohjelman mukaisesti kehitetty kvantitatiivisten aineistojen ja ilmiöiden käsittelyyn, analysointiin ja mallintamiseen. Se toimii kaikilla yleisillä käyttöjärjestelmillä, kehittyy aktiivisesti, ja on toiminnallisuuksiltaan lähes rajattomasti laajennettavissa.

R:ää käytetään pääasiassa tilastollisiin tarkoituksiin, mutta se sisältää toiminnallisuuksia jotka mahdollistavat käytön mm. matriisilaskimena, data-louhimena, kuva-analysaattorina ja dna-sekvensserinä. R:n ilmaisuus, yhteisövetoinen kehitysmalli ja korkealuokkaiset grafiikkaominaisuudet ovat johtanut siihen että yksittäisten akateemikkojen ja insinöörien lisäksi suuret yritykset kuten IBM, Google, Pfizer ja Shell tukevat ja kehittävät R:ää aktiivisesti.

Tässä oppaassa ei ole tarkoitus listata kaikkia R:n ominaisuuksia ja hätäisesti esitellä sen syvempiä kykyjä, vaan antaa perustuntuma R:n käytöstä ja mikä tärkeintä, mahdollistaa

itsenäinen R:n opettelu jatkossa. Suurin osa oppimisesta tapahtuu itse konetta naputtamalla, ja uusia ominaisuuksia löytyy lähes päivittäin kokeneellekin käyttäjälle.

1.1.1 Käskytettävä laskukone

R on ennenkaikkea interaktiivinen laskukone, ja se toimii yksinkertaisella komento-toiminta -periaatteella. R on käynnistämisen jälkeen valmis ottamaan vastaan komentoja kun näet komentokehotteen ”>”. Komentokehotteeseen syötetään komento, esimerkiksi `3+2*(5/4)`, ja komento käynnistetään painamalla Enter-näppäintä:

```
> 2+2*(5/4)
[1] 4.5
```

Lasku suoritetaan tavallisin laskusäännöin sulut huomioiden ja tulos palautetaan ruudulle. `[1]` tarkoittaa rivin ensimmäisen alkion järjestyslukua, hyödyllinen tieto jatkossa kun ruudulle tulostuu monirivinen vektori tai matriisi.

R:ssä on sisäänrakennettuna kaikki tavallisimmat matemaattiset funktiot:

```
> exp(-2)
[1] 0.1353353

> cos(pi)
[1] -1
```

Moni asia on suoraviivainen:

```
> x<-rnorm(1000)
> plot(x)
```

Funktio `rnorm` (“random normal”) luo 1000 satunnaislukua simuloimalla normaalijakaumaa, sijoitusoperaatio `<-` asettaa ne talteen muuttujaan `x`, ja funktio `plot` piirtää niistä kuvan.

R:n saa sammutettua komennolla

```
> q()
```

ja tallennus-kysymykseen vastataan kieltävästi.

1.2 Perusominaisuuksia

1.2.1 Oliot ja muuttujat

R:ää voidaan pitää korkeamman asteen ohjelmointikielenä. Tämä tarkoittaa sitä, että jokainen koneen muistissa oleva asia kuten numero `2`, muuttuja `x` tai funktio `plot`, on niin sanottu *olio* (eng. *object*): Monipuolinen ja 'älykäs' tietorakenne. Tästä johtuen on R:n käyttö varsin joustavaa verrattuna matalamman tason ohjelmointikieliin kuten C tai FORTRAN. Voimme suorittaa komentoja sekoittaen erityyppisiä olioita ja mikäli oliot ovat tarpeeksi älykkäitä, on lopputulos järkevä. Esimerkiksi edellä komensimme `plot(x)`, jossa hyvin monikäyttöinen `plot`-funktio tietää automaattisesti mitä pitää tehdä koska `x` osaa kertoa sille olevansa vektori reaalilukuja. Olio-ajattelu ei ole pakollinen, mutta se selittää paljon R:n käyttömukavuudesta ja toiminnallisuuksista.

Komennot palauttavat usein jonkin arvon, ja mikäli emme tallenna arvoa muistiin se tulostuu ruudulle eikä ole enää muistissa. Tallentaminen muistiin eli *muuttujaan* tapahtuu sijoitusoperaattorin `<-` (kaksi merkkiä) avulla. Tallennuksen jälkeen muuttujissa olevia arvoja voidaan jatkokäyttää edelleen:

```
> 2+2
[1] 4

> y<-2+2
> y
[1] 4

> luku2<-y*(y+1)
> luku2
[1] 20
```

Toinen rivi voitaisiin lausua "y olkoon 2 plus 2": Kuten lukiomatematiikassa tapana, sijoitusoperaattorin `<-` oikealla puolella suoritetaan lauseke `2+2` ja tulos sijoitetaan vasemmalle puolelle muuttujaan `y`. Sitten kutsumme muuttujaa `y` nimeltä jolloin sen sisältö tulostuu ruudulle. Neljännellä rivillä `luku2` saa arvokseen tuloksen laskusta `y*(y+1)`. Viidennellä rivillä on komentona taas vain muuttujan nimi jolloin sen arvo tulostuu ruudulle.

Muuttujat ovat R:n käytössä erittäin tärkeitä, sillä kaikki ainestot ja välilaskut tallennetaan muuttujiin. Niiden nimeämisessä kannattaa käyttää selkeyttä (`pituus.1v` vai `p1`?). Muuttuja ei voi alkaa numerolla, mutta muuten kirjaimet ja numerot ovat vapaasti käytössä. Isot ja pienet kirjaimet eroavat: `abc` ei ole sama kuin `Abc`.

Huomaa että R on jo varannut (muunmuassa) nimet `c, q, t, C, D, F, I, T`, ja niiden huolimattomasta käytöstä seuraa ongelmia.

Harjoituksia 1.2.1:

1. Laske lukujen $1, 2, 3, 2, 2.5, 5, 3$ keskiarvo:
   - Tee summamuuttuja `S<- 1+2+3+2+2.5+5+3`. Tee myös lukumäärää kuvaava muuttuja `n<-7`.
   - Jaa summa lukumäärällä, `S/n`.
2. Kokeile erilaisia alkeisoperaatioita: `2*2`, `12/5`, `(2*6)/(3+2)`, `4^2`, `sqrt(16)`.

1.2.2 Funktiot, parametrit ja lausekkeet

Komennoista osa annetaan *funktiokutsuina*. Funktio on, kuten matematiikassakin, jonkinlainen suoritus tai suoritusten joukko joka (usein) palauttaa jonkin arvon, mahdollisesti käyttäen annettuja parametreja. Funktiokutsun muoto R:ssä on

```
nimi(parametri1, parametri2, ..., parametriN)
```

Sulkujen käyttö on tärkeää jotta R tunnistaa komentosi funktiokutsuksi. Esimerkiksi `sin(pi/2)`, parametrinä `pi/2`, laskee sinifunktion arvon kyseisessä kohdassa ja palauttaa tuloksen eli numeron 1. Toisena esimerkkinä `plot(x)`, parametrinä `x`, piirtää annetusta oliosta kuvan ja palauttaa tyhjän. Ja niin edelleen.

Kaikki funktiot eivät tarvitse parametrejä. Esimerkiksi funktio `ls` (`list`)

```
> ls()
[1] "luku2" "x"     "y"
```

palauttaa käyttäjän luomien muuttujien nimet. Toistamiseen, on tärkeää muistaa sulut `()`: jos ne puuttuvat, ei funktiota suoriteta. Sen sijaan sen rakenne tulostetaan kuin se olisi tavallinen muuttuja.

Yhdistämällä muuttujia, funktiokutsuja ja operaattoreita (`+`, `/`, `<-`,...) saadaan monimutkaisempia komentojonona eli *lausekkeita*:

```
> y.exp <- (-1)*log(1-runif(1000))
```

Ensin `runif` luo 1000 satunnaislukua väliltä $[0, 1]$ (`r`andom `unif`orm), vähennetään jokainen niistä ykkösestä, otetaan jokaisesta logaritmi ja sitten vielä vastaluku. Lopuksi saatu vektoriarvoinen tulos sijoitetaan muuttujaan `y.exp`. Huomaa kuinka R käsittelee yksittäisen luvun ja 1000 lukua pitkän vektorin summaamista. Tämä niin kutsuttu kierrätys on yksi R:n kätevistä aritmeettisista perusominaisuuksista, ja törmäämme siihen jatkossa.

Voimme piirtää arvoista histogrammin siihen tarkoitetulla funktiolla:

```
> hist(y.exp)
```

Tässä annamme parametrinä luomamme arvot. `hist` on funktio jolle voi antaa muitakin parametrejä: Nimetään parametri joka säätelee väriä:

```
> hist(y.exp, col=2)
```

R on täynnä erilaisia pikkuasioita tekeviä funktiota, ja yleensä kaikkeen mitä haluaa tehdä löytyy jo valmis ratkaisu.

Harjoituksia 1.2.2:

1. Laske seuraavat lausekkeet (kirjoitusharjoittelua)
   - $\sqrt{2\pi}\cos(\frac{3\pi}{4})$
   - $\log(e^{1+acos(1/\sqrt{2})}+1)$
2. Mitä käy kun kutsut funktiota ilman sulkuja, esim. `exp`? Entäs ilman parametrejä, `exp()`? Entä `hist()`? Entä `x()`?

1.3 Tietorakenteet

R:ssä on monenlaisia tietorakenteita, eli erilaisten tietojen, arvojen ja taulukoiden tallentamismuotoja. Yksinkertaisimmat ja yleisimmät niistä ovat yksittäiset numerot (kuten `0`,`-12.9`), merkkijonot (`"a4"`, `"red"`) ja totuusarvot (`TRUE`, `FALSE` lyh. `T, F`).

Numeroita ovat kaikki yksiulotteiset luvut. Ne voidaan jakaa kokonaislukuihin, (`integer`), liukulukuihin eli reaalilukuihin (`numeric`), ja imaginäärilukuihin (`complex`).

Merkkijonot (`character`) ovat kirjainmerkkejä sisältäviä muuttujia. Ne laitetaan aina lainausmerkkeihin, `"tähän tapaan"` ja voivat sisältää sekä välilyötejä että numeroita.

```
> nimi<-"Oma nimi"
> print(nimi)
[1] "Oma nimi"
```

R:ssä on lisäksi erityinen puuttuvaa tietoa kuvaava alkio `NA` (Not Available). Tällöin tietoa käsittelevälle funktioille on erikseen sanottavissa miten haluamme tiedoissamme olevat puuttuvat alkiot huomioida.

1.3.1 Vektorit

Monimutkaisemmista tietorakenteista tärkein on *vektori*, eli järjestetty jono alkioita. Yksinkertaisin tapa luoda vektoreita on käyttää funktiota `c` (combine):

```
> paino1<-c(60, 72, 57)
> paino2<-c(90, 95, 72)
```

Voimme yhdistää myös vektoreita toisiinsa `c`:n avulla:

```
> paino<-c(paino1, paino2)
> paino
[1] 60 72 57 90 95 72

> pituus<-c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

Kun vektorille tehdään aritmeettinen operaatio yksittäisellä luvulla (esim. neliöinti `^2`), kierrätetään operaatio kaikkien vektorin alkioiden yli:

```
> pituus2<-pituus^2
```

Tämä erona komentoon jossa kaksi *samanmittaista* vektoria osallistuvat operaatioon, jolloin lasku lasketaan vastinalkio kerrallaan:

```
> BMI<-paino/pituus2
```

Vektoreiden käsittelyyn on paljon funktiota. Esimerkiksi `length` palauttaa vektorin pituuden, `sum` laskee vektorin alkiot yhteen, `mean` laskeen niiden keskiarvon:

```
> sum(pituus)
[1] 10.75

> length(pituus)
[1] 6

> sum(pituus)/length(pituus)
[1] 1.791667

> mean(pituus)
[1] 1.791667
```

Yhdistetään useampi toiminta samalle riville

```
> sum( (pituus-mean(pituus))^2 )/(length(pituus)-1)
[1] 0.01005667
```

Eli varianssi,

```
> var(pituus)
[1] 0.01005667
```

Harjoituksia 1.3.1:

1. Tee vektori `lukuja` johon laitat 10 satunnaisesti arpomaasi kokonaislukua joukosta 0,...,9. Esimerkiksi `lukuja<-c(4, 2, 6, 9, 3, 8, 4, 4, 2, 0)`,
2. Laske `lukuja` vektorin keskiarvo, `mean(lukuja)`. Saitko 4.5? Entäs varianssi: saitko 8.5?
3. Arvotaan vielä 3 lukua. Lisää ne yksi kerrallaan `lukuja` vektorin loppuun toistaen komentoa `lukuja<-c(lukuja, uusiluku)`. Nuoliylös -näppäin on avuksi. Miten on nyt keskiarvo ja varianssi?

1.3.2 Vektorin osajoukot

Vektorin osajoukon poimiminen tehdään sijalukuihin eli *indekseihin* viittamalla. Viittaminen tapahtuu hakasulkuja käyttäen: `vektori[indeksi]`. Indeksointi alkaa 1:stä ja päättyy vektorin pituuteen. Indeksin on aina oltava kokonaisluku. Esimerkiksi:

```
> pituus[4]
[1] 1.9

> pituus[1]
[1] 1.75
```

Voimme poimia useita alkioita antamalla indeksinä vektorin. Poimitaan ensimmäiset kolme alkiota kerralla:

```
> pituus[c(1,2,3)]
[1] 1.75 1.80 1.65
```

Huomaa kuinka yhdistämme kaksi komentoa sisäkkäin: Luomme vektorin `c(1,2,3)`, ja käytämme sitä heti indeksijoukkona. Samaan alkioon voidaan viitata tarvittaessa useammin kin:

```
> pituus[c(2,2,5,5,5)]
[1] 1.80 1.80 1.74 1.74 1.74
```

R:ssä voidaan poimia myös negaation kautta: Mikäli indeksi on negatiivinen, poimitaan kaikki paitsi indeksijoukon osoittamat arvot:

```
> pituus[-1]
[1] 1.80 1.65 1.90 1.74 1.91

> pituus[-c(1,2,3)]
[1] 1.90 1.74 1.91
```

Säännöllisten lukujonojen luomiseen on hyvä tietää `seq`uence, joka toimii kahdella eri tavalla:

```
> seq(0, 3, by=0.25)
 [1] 0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00

> seq(1, 3, length=3)
[1] 1 2 3
```

Tärkeä on myös R:n kätevä `mistä:mihin` komento:

```
> 1:3
[1] 1 2 3

> 2:length(pituus)
[1] 2 3 4 5 6
```

Jos haluaa vaikkapa vaihtaa järjestystä niin takaperinkin onnistuu:

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

Askellus on aina kokonainen 1.

Ison aineiston haluttuja osajoukkoja selvitellessä kannattaa ne usein laittaa talteen jotta niitä voi käyttää uudelleen:

```
> ekat<-1:3
> pituus[ekat]
[1] 1.75 1.80 1.65

> paino[ekat]
[1] 60 72 57
```

Indeksinumeroiden lisäksi voidaan käyttää totuusarvovektoria. Se on aineistovektorin pituinen totuusarvovektori, joka jokaisen alkion kohdalla vastaa kysymykseen: Otetaanko alkio mukaan vai ei? Vaikkapa näin:

```
> BMI
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630

> yli25<-c(F, F, F, F, T, F)
> BMI[yli25]
[1] 31.37799

> paino[yli25]
[1] 95
```

Tämä voi tuntua hankalalta, mutta totuusarvoilla poimiminen on tehokasta kun yhdistämme sen *suhteellisiin* operaatioihin, eli alkioiden vertailuun johonkin kynnysarvoon. Onko totta että

```
> BMI > 25
[1] FALSE FALSE FALSE FALSE  TRUE FALSE
```

(Huomaa että komentokehote ei ole osa komentoa) `R` tekee jokaisen `BMI`-vektorin alkion kohdalla vertailun: Olenko isompi kuin 25, tosi/epätosi? Eli totuusarvovektoria ei tarvitse kirjoittaa käsin, vaan

```
> yli25<- BMI>25
> paino[yli25]
[1] 95
```

Loogiset operaatiot JA (`&`-merkki) sekä TAI (`|`-merkki) ovat myös käytössä. Niillä voimme yhdistellä suhteellisia operaatioita:

```
> TRUE | FALSE
[1] TRUE

> c(TRUE, TRUE) & c(TRUE, FALSE)
[1]  TRUE FALSE
```

Taas alkio kerrallaan kun on kaksi samanpituista vektoria. Saamme näin tehokkaasti valittua vektorista kiinnostavia osajoukkoja vaikkapa näin: “BMI pienempi kuin 25“ JA “pituus suurempi kuin 1.8“,

```
> (BMI < 25) & (pituus > 1.8)
[1] FALSE FALSE FALSE  TRUE FALSE  TRUE
```

”Pienempi tai yhtäsuuri” on merkkiyhdistelmä `<=` (kaksi merkkiä) ja vastaavasti toiseensuuntaan `>=`. Yhtäsuurutta testataan ”==” merkillä (tupla=), ja erisuuruutta ”!=” merkillä.

Harjoituksia 1.3.2:

1. Mitä eroa on `1:5-1` ja `1:(5-1)`?
2. Mitä tapahtuu : `pituus[0]`? `pituus[29]`? `pituus[5:10]`?
3. Selvitä totuusarvoja käyttäen mitkä ovat painot joilla vastaava BMI on alle 20.
4. Entäs ne joilla BMI ei ole välillä 20-25? Totuudesta saa epätotuuden huutomerkillä: `!TRUE==FALSE`.
5. Totuusvektorista saa helposti indeksivektorin komennolla `which(vek)` (’mitkä?’): Se palauttaa vektorin `vek` indeksit joiden kohdalla totuusarvo on TRUE. Mitkä ovat BMI 20-25 havaintojen indeksit? Montako niitä on (käytä `length`-funktiota).
6. Äskeinen oli helppo laskea silmäilemällä. Entäs kun lukujoukko on: `paljon<-runif(1000000)` (tee). Montako niistä on korkeintaan 0.5? Entäs mikä on välillä 0.2-0.8 olevien lukujen varianssi?

1.3.3 Matriisi

Samanpituiset vektorit voi yhdistää rinnakkain tai allekkain, jolloin saadaan *matriisi*. Tähän voidaan käyttää esimerkiksi `cbind` tai `rbind` (column tai row bind) funktioita:

```
> A<-cbind(pituus, paino, BMI)
```

Tuloksena on 6×3-matriisi, jonka ensimmäinen sarakkeessa on pituudet, toisessa sarakkeessa painot ja kolmannessa BMI-arvot.

Matriisin alkioihin viitataan kaksiulotteisesti "[rivi,sarake]" periaatteella. Esim. 5. rivi, 2. sarake:

```
> A[5, 2]
paino
   95
```

3. rivi, 1. sarake:

```
> A[3, 1]
pituus
  1.65
```

Tyhjä tarkoittaa koko riviä tai saraketta: koko 5. rivi tai koko 1. sarake,

```
> A[5, ]
  pituus    paino      BMI
 1.74000 95.00000 31.37799

> A[,1]
[1] 1.75 1.80 1.65 1.90 1.74 1.91
```

Myös osajoukkoja on helppo poimia. Poimitaan 5. rivin 1. ja 2. alkio:

```
> A[5, c(1,2)]
pituus  paino
  1.74  95.00
```

Myös osamatriisin voi poimia, eli monta alkioita yhtäaikaa sekä riveistä että sarakkeista. Poimitaan ne rivit mille BMI yli 20, ja vain sarakkeet 1 ja 3:

```
> yli20<-BMI>20
> A[yli20, c(1,3)]
     pituus      BMI
[1,]   1.80 22.22222
[2,]   1.65 20.93664
[3,]   1.90 24.93075
[4,]   1.74 31.37799
```

Tai negaation kautta

```
> A[yli20, -2]
```

Harjoituksia 1.3.3:

1. Luo matriisi `B` johon tulee riveiksi pituus, paino ja BMI.
2. Transponoi matriisi `B` komennolla `t(B)`. Kokeile monitestausta: `A==t(B)` ja toisaalta funktiokutsua `all.equal(A,t(B))`. Mitä eroa?
3. Mitä käy matriisille `A` jos teet sijoituksen `A[1,1]<-"argh"`? Mites nyt `A==t(B)`? (Vertailua tehtäessä numerot muutetaan merkkijonoiksi!)

1.3.4 Tietokehikko ja lista

Matriisissa kaikkien alkioiden on siis oltava samantyyppisiä tai jokainen alkio muutetaan vähiten rajoittavaan tyyppiin (yleensä merkkijonoksi). Tämän rajoituksen poistaa *tietokehikko* (data frame), joka on kuin matriisi siinä että kaikkien sarakkeiden (ja rivien) pitää olla yhtä pitkät, mutta lisäksi se mahdollistaa erilaisten vektorityyppien yhdistelyn. Lisätään aineistoomme sukupuolta kuvaava muuttuja, ja luodaan tietokehikko komennolla `data.frame`:

```
> sukupuoli<-c("mies","nainen","mies","mies","nainen","mies")
> tiedot<-data.frame(sukupuoli, pituus, paino, BMI, yli25)
> tiedot
  sukupuoli pituus paino      BMI yli25
1      mies   1.75    60 19.59184 FALSE
2    nainen   1.80    72 22.22222 FALSE
3      mies   1.65    57 20.93664 FALSE
4      mies   1.90    90 24.93075 FALSE
5    nainen   1.74    95 31.37799  TRUE
6      mies   1.91    72 19.73630 FALSE
```

Tietokehikon lisäetuna on myös se että sarakkeisiin voi nyt viitata niiden nimellä

```
> tiedot[ , "pituus"]
[1] 1.75 1.80 1.65 1.90 1.74 1.91
```

tai alijoukko-merkkiä $ käyttäen:

```
> tiedot$pituus
[1] 1.75 1.80 1.65 1.90 1.74 1.91

> tiedot$sukupuoli
[1] mies   nainen mies   mies   nainen mies
Levels: mies nainen
```

Tietokehikko muutti sukupuoli-vektorin automaattisesti taso-vektoriksi, josta myöhemmin lisää.

Viimeisenä tietorakenteena vielä *lista*: Se voi periaattessa koostua mistä vain, vaikkapa toisista listoista. Lista on kätevä rakenne kun haluaa esimerkiksi kerätä tietoa yhteen nippuun:

```
> tiedot.l<-list(tiedot=tiedot, vaarassa=tiedot[yli25,], pvm=date(), huom="Esim 1")
> tiedot.l
$tiedot
  sukupuoli pituus paino      BMI yli25
1      mies    1.75    60 19.59184 FALSE
2    nainen    1.80    72 22.22222 FALSE
3      mies    1.65    57 20.93664 FALSE
4      mies    1.90    90 24.93075 FALSE
5    nainen    1.74    95 31.37799  TRUE
6      mies    1.91    72 19.73630 FALSE

$vaarassa
  sukupuoli pituus paino      BMI yli25
5    nainen    1.74    95 31.37799  TRUE

$pvm
[1] "Fri Sep  9 15:34:16 2011"

$huom
[1] "Esim 1"
```

Funktio names palauttaa osien nimet (mikäli nimetty), ja niihin voi viitata $-merkin avulla:

```
> names(tiedot.l)
[1] "tiedot"   "vaarassa" "pvm"      "huom"

> tiedot.l$huom
[1] "Esim 1"
```

Lista on järjestetty, eli sen alkioihin voi viitata myös indeksillä. Silloin tulos on lista: Mikäli haluamme tavarat listan sisältä on käytettävä tuplahakasulkuja, näin:

```
> tiedot.l[[2]]
  sukupuoli pituus paino      BMI yli25
5    nainen   1.74    95 31.37799  TRUE
```

Tietorakenteista jatketaan myöhemmin. Pistetään vielä rakennettu tietokehikko ja lista talteen,

```
> save(tiedot, tiedot.l, file="tiedot.rda")
```

ja harjoitellaan tässä esittelyosion lopuksi hieman grafiikkaa. Piirretään pituuden suhde painoon ja korostetaan värein painoindeksiä. Painotasot olkoon

```
> tasonimet<-c("Alipaino", "Normaali", "Ylipaino")
```

ja BMI:n mukaiset tasot olkoon

```
> tasot<-c(2,2,2,3,3,2)
```

Ja sitten piirretään:

```
> plot(pituus, paino, col=tasot, pch=17, cex=1.2)
> legend("topleft", tasonimet, col=1:3, pch=17)
```

Piirtämisessä `pch=17` tuottaa täytettyjä kolmioita, ja kuvaselite-funktiossa `legend` parametri `cex=0.8` pienentää hieman kirjasinta. Yksityiskohdat sivuun; Nyt kannattaa kiinnittää huomita miten kuva tuotettiin. Piirrettiin ensin kuvaaja jota säädeltiin parametrein, ja sitten lisättiin kuvaselitteet omalla funktiollaan, omin parametrein. Näistä tulee jatkossa enemmän, mutta tällainen toiminta on R:n käytön sydän. Rajattuja asioita askel kerrallaan.

Kotitehtäviä 1:

1. Luo kaksi vektoria joissa on naisten enegiansaanti ennen ja jälkeen kuukautisten: Ennen 5260, 5640, 6390, 6805, 7515, 8770), jälkeen 3910, 3885, 5645, 5265, 6790, 7335.
2. Luo vektori, jossa on havaintojen muutos. Mitä huomaat? Luo muutoksen prosentuaalinen vektori. Mikä on keskimääräinen muutos prosenteissa?
3. Mikä on tyypillinen energiamuutos alle 7000 yksikköä ennen kuluttavissa?
4. Rakenna tiedoista tietokehikko. Voit pyöristää prosentit käyttäen funktiota `round(arvot, digits=1)`.

2 Osa 2: R ympäristö ja I/O

Pureudutaan hieman tarkemmin R-ohjelman toimintaan. Kaikki ohjeet tässä monisteessa toimivat kaikilla käyttöjärjestelmillä, niin kotona kuin konttorillakin. R:n käytössä on vain pieniä eroja riippuen käyttöjärjestelmästä, esimerkiksi osa toiminnoista on Windows- ja Mac OS X-alustoilla käytettävissä myös valikoista ja työkaluriveiltä; Yleensä ne tosin ovat vain pikanappuloita varsinaisille R-lausekkeille.

2.1 Komentojen kirjoittamisesta komentokehoteeseen

Heti ensimmäisestä komennosta asti R pitää kirjaa käyttäjän toiminnoista. Edellisiin komentoihin voi aina palata painamalla nuoli ylös-näppäintä, ja takaisinpäin nuoli alas-näppäintä. Rivillä siirrytään normaalisti vasemmalle ja oikealle nuolilla.

Tärkeä nappula mikä kannattaa opetella heti alussa on tabulaattori (tab, caps lockin yläpuolella). Se saa R:n täydentämään eli ehdottamaan kirjoitettavalle sanalle loppua. Kokeile: Kirjoita `data.` ja paina tabulaattoria, tarvittaessa kahdesti. Saat listan kaikista R:n tuntemista asioista joiden alku on `data.`. lisää merkki `f` ja kokeile uudestaan. Näin löytyy usein tarvittava funktio nopeasti (esim. Wilcoxonin testi, alkaen `wilco` ja tab).

2.2 Istunto

R:n käynnistäminen avaa niin sanotun *istunnon*, oman pienen universumin jossa käyttäjä antaa komentojaan ja luo olioitaan. Istunnon aikaiset muuttujat ovat muistissa vain niin kauan kuin istunto on käynnissä. Tai kunnes ne poistetaan esimerkiksi funktiolla `rm` (remove):

```
> x<-seq(2,10, by=2)
> exists("x")
[1] TRUE
```

x on siis olemassa muistissa,

```
> x
[1]  2  4  6  8 10

> rm(x)
> exists("x")
[1] FALSE
```

Mutta se poistettiin ja ei enää ole olemassa. Istunnon sulkeminen poistaa kaikki muuttujat automaattisesti.

2.2.1 Työhakemisto

R:n istunnolla on fyysinen sijainti tietokoneen kovalevyllä, ns. *työkansio*, josta oletuksena etsitään ja johon oletuksena tallennetaan tietoa. getwd-funktio (get working directory) kertoo nykyisen työhakemiston, ja se voidaan muuttaa funktiolla setwd(polku). Windowsissa tämä toiminta löytyy valikosta (file→change dir). Windows-koneessa voi näkyä esim.

```
> getwd()
[1] "D:/users/tarajala/My Documents"
```

getwd():n tulos on siis nykyin työhakemisto. Kaikki kovalevyä käyttävät toiminnot kuten tiedostoluku, -kirjoitus ja työtilan tallennus tapahtuu oletuksena työhakemistossa.

Jos tulossa on pitempikin R istunto on sitä varten hyvä luoda oma hakemistonsa jotta istunnon aikaiset aineistot ja tallennuksen säilyvät selkeässä paikassa. Esimerkkinä luon C-asemalle hakemistoon ”R-kurssi11” uuden alihakemiston ”harj1”, ja vaihdan sen työhakemistokseni:

```
> setwd("c:/R-kurssi11/harj1")
```

Tekstipohjaista R:ää käytettäessä nopein tapa on yksinkertaisesti käynnistää R haluamassaan hakemistossa jolloin se on automaattisesti kyseinen työhakemisto. Graafisten ympäristöjen käytössä on aina pikanappula käytössä.

Harjoituksia 2.2.1:

1. Tee itsellesi hakemisto `R-kurssi`, johon keräät tämä kurssin tiedostoja.
2. Tee kurssihakemistoon alihakemisto ”osa2”, ja muuta se työhakemistoksi.
3. Seuraavilla kerroilla aina kun aloitat uuden R-istunnon tee sopiva alihakemisto ja vaihda se työhakemistoksi.

2.2.2 Tiedot talteen ennen lopetusta

Koska jokainen istunto alkaa tyhjänä, on tietoa pystyttävä tuomaan istunnon sisälle ja sitä on myös saatava levylle talteen istunnosta ulos. Järein tapa tehdä tämä on tallentaa koko istunto funktiolla `save.image` joka luo työhakemistoon tiedoston nimeltä **.RData**. Se sisältää komentohistorian, muistissa olevat muuttujat, käyttäjän luomat funktiot ja muut istunnon alkutilasta poikkeavat asiat.

Yleensä kuitenkin riittää tallentaa vain yksittäisiä, tarpeellisia palasia talteen, jolloin esimerkiksi aineistomuunnosten välivaiheet ja muut roskat eivät jää muistiin viemään tilaa. `save` funktiolla tallennetaan R:n omassa formaatissa tietoa levylle. Sille annetaan parametri `file="tiednimi"`, loput ovat tallennettavien muuttujien nimiä.

```
> x<-runif(10)
> y<-seq(0,5,length=10)
> save(x, y, file="kaksivektoria.rda")
> rm(x, y)
> x
Error: object 'x' not found
```

`save`tettuja tiedostoja luetaan takaisin `load`-komennolla. Sille annetaan parametriksi vain tiedoston nimi:

```
> load("kaksivektoria.rda")
> cbind(x, y)
```

Näin oleellista tietoa saadaan talteen, voidaan kuljettaa vaikkapa muistitikulla, ja ladata käyttöön taas toisaalla.

Kun on aika lopettaa istunto, suljetaan R joko antamalla komento `q()` tai suljetaan graafinen ikkuna. R kysyy vielä varmistuksen lopettamiselle; Jos hyväksyt tallentamisen,

ajaa R komennon `save.image()` sinun puolestasi ennen lopullista poistumista, eli tallentaa koko istunnon tilan työhakemistoon `.RData` tiedostoon.

Harjoituksia 2.2.2:

1. Kopio ensimmäisessä osassa tallentamasi tiedosto **tiedot.rda** työhakemistoon ja lataa sen sisältö.
2. Tarkista tietojen latautuminen listaamalla kaikki R:n muistissa olevat muuttujat komennolla `ls()`.

2.2.3 Ongelmatilanteissa stop-nappia

Jos tuntuu siltä että R jää jumiin eli viimeisin komento ei suoriudu loppuun, on komennon suoritus mahdollista lopettaa väkisin. Graafisissa versioissa on erikseen työkaluikoni tätä varten, mutta Esc-näppäin ajaa saman asian. Linux-versiossa stop-nappina on yleinen unix-hätäyhdistelmä Ctrl-c (Ctrl pohjaan ja c näppäintä).

Kokeillaan jumitusta: Luodaan silmukka (komentoja toistava rakenne) joka loputtomiin kirjoittaa ruudulle ”argh!”:

```
> while(TRUE) print("argh!")
```

Jos kaikki kuitenkin menee niin jumiin että stop-nappi ei toimi, joudut sammuttamaan R:n väkisin vaikkapa Windowsin palkista. Tällöin menetetään kaikki tallentamaton tieto. Siksi onkin hyvä tallennella tietoa aika-ajoin, ja varsinkin raskaan aineistonmuokkauksen tai laskennan jälkeen.

2.2.4 Pitkien lausekkeiden syöttäminen

Ruudulla `>` merkki tarkoittaa siis R:n komentokehoitetta, ”anna komento”. Mikäli lausekkeemme on yli rivin pituinen voimme jatkaa toiselle riville rivivaihdolla (enter) jolloin komentokehote muuttuu symboliksi `+`. R jää odottamaa komennon viimeistelyä ennen kuin se suoritetaan:

```
> pituus<-c(1.75, 1.80,
+ 1.65, 1.90, 1.74, 1.91)
```

Useimmiten kuitenkin törmäämme +-merkkiin kun kirjoitamme komennon väärin, esimerkiksi kun emme muista kirjoittaa tarpeeksi sulkuja, tai toinen lainausmerkki unohtuu merkkijonosta:

```
> hist( exp(1-rnorm(100) )
+
+ )
> abc<-"abc
+ "
```

Komennon syötön voi aina keskeyttää stop-napilla.

2.2.5 Lyhyet ohjelmat eli skriptit: lausekkeiden niputus

R:n käyttö on paljolti komentojen peräkkäistä kirjoittamista interaktiivisesti, eli teemme jotain, sijoitamme tuloksen johonkin, teemme sille jotain, ja niin edelleen. Komentojen kirjoittaminen komentokehotteelle on tärkeänä osana toimintaa, mutta näin toimiessa muutoksen tekeminen aikaisempaan välivaiheeseen on hankalaa. Siksi paria riviä pidemmät komentosarjat kirjoitetaankin erilliseksi pikku ohjelmaksi (ei Isoksi Ohjelmaksi kuten Firefox tai Windows, vaan pieneksi ohjelmaksi kuten pullaresepti tai ajo-ohjeet).

Näitä pieniä ohjelmia voi pitää toimintamme käsikirjoituksena (eng. script): Kirjoitamme siihen rivi riviltä mitä haluamme R:n tekevän, ja syötämme sen sitten R:lle joko kokonaan tai haluttuina palasina, jonka jälkeen R tulkitsee orjallisesti komentomme, kuten tarkoitettu, rivi riviltä.

Skriptien kirjoittamiseen käy tavallinen tekstieditori kuten Notepad. R:n Windows/Mac versiossa on sisäänrakennettuna valmis editori jota kannattaa alkuvaiheessa hyödyntää. Myöhemmin voi siirtyä käyttämään tekstinkorostus- ja automaattitäydennyksillä varustettuja editoreita, jotka tuovat sujuvuutta pidempiin analyyseihin. Kirjoitti sitten millä tahansa on toimintakuvio usein seuraava:

1. Kirjoitetaan haluttuja komentoja:

   Tiedostoon:

   ```
   paino<-c(60,72,57,90,95,72)
   pituus<-c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
   BMI<-paino/pituus^2
   yli25<- BMI>25
   A<-cbind(pituus, paino, BMI)
   sukupuoli<-c("mies","nainen")[c(1,2,1,1,2,1)]
   tiedot<-data.frame(sukupuoli, pituus, paino, BMI, yli25)
   ```

2. Tallennetaan tiedosto työhakemistoon esimerkiksi nimellä **tiedot.R**.
3. Luetaan tiedosto R:ään:

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

4. Tarkastellaan tulosta:

```
> tiedot
```

Kun haluamme muuttaa jotain, toistetaan kohtia 1-3.

Tässä `source` komento lukee tiedoston rivi riviltä, ja toteuttaa komennot kuin ne syötettäisiin käsin. Graafisissa systeemeissä voidaan myös valita vain osa riveistä ja syöttää ne R-ympäristöön tarpeen mukaan. Esimerkiksi Windows R:n tekstieditorissa Ctrl-r syöttää valitun rivin tai osatekstin komentokehoitteelle, kuten tekee esimerkiksi KDE:n yleiseditori Kate. Varsin kätevää.

Kun komentoja alkaa olla useampia pötkössä on hyvä lisäksi kommentoida käsikirjoitusta, eli lisätä komentojen sekaan selventäviä huomautuksia. Siihen käytetään merkkiä `#`. Mikäli komentorivillä on merkki `#`, R ohittaa kaiken sen jälkeisen tekstin:

```
> x<-1
> # Nämä ovat vain kommentteja, R ei tee niille mitään.
> # x<-2
> x  # tänne peräänkin voi kirjoittaa
[1] 1
```

Niiden lisääminen käsikirjoitukseen on tärkeää sekä sinulle että muille: Muistat itse mitä olit tekemässä ja muut ymmärtävät mitä yrität tehdä. Tämä on tärkeä osa toistettavan tieteen periaatetta.

Tavallisen istunnon aikana ajotiedostomme voisi alkaa näyttää vaikkapa tältä:

Tiedostossa **tiedot.R**

```
# Aineistossa 6 henkilöä, miehiä naisia. Tutkitaan ylipainoa.

paino<-c(60,72,57,90,95,72)                    # painot kiloina
pituus<-c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91) # pituudet metreinä
sukupuoli<-c("mies","nainen")[c(1,2,1,1,2,1)] # sukupuolet

# Lasketaan painoindeksi
BMI<-paino/pituus^2
# Kenellä menee yli 25?
yli25<- BMI>25
# kerätään taulukkoon
tiedot<-data.frame(sukupuoli, pituus, paino, BMI, yli25)
```

Luettavuuden takia tyhjiä rivejä kannattaa jättää, ne eivät tee ajettuna mitään.

Jatkossa on hyvä aina aloittaa istunto luomalla uusi käsikirjoitustiedosto jota kommentoi usein. Siihen on siten helppo palata myöhemmin, ja kirjoitusvirheiden takia ei mene ylimäärin aikaa hukkaan.

Harjoituksia 2.2.5:

1. Tee käsikirjoitustiedosto `luvut1.R` johon kirjoitat ensimmäisessä osassa lukujen arvontaan liittyvät komennot.
2. Kommentoi tarpeen mukaan, esimerkiksi uusien oppimiesi komentojen kohdalla. Muista tallentaa.
3. Tarkista toimivuus `source`-komennolla.

2.2.6 Sisäinen apujärjestelmä ja tekniset manuaalit

Koska R:n käyttö on hyvin pitkälti komentojen ja funktiokutsujen antamista, rivi kerrallaan koodausta, seuraa siitä se että erilaisten funktionimien ja niiden lukemattomien parametrinimien muistaminen on tärkeä mutta vaativa urakka (lue: käytännössä mahdoton urakka). Tämän takia on R:ään rakennettu varsin kattava ja nopea apujärjestelmä, josta löytyy jokaisen funktion kuvaus sekä käyttöohjeet.

Apujärjestelmään päästään käsiksi seuraavasti: Kysytään vaikkapa ”mitenkäs sitä `cbind`-komentoa käytettiin?”:

```
> help(cbind)
```

Funktio `help` avaa apujärjestelmän ohjekirjasivut kohdasta `cbind`. Sille on synonyymi `?`:

```
> ?cbind     # tai ?"cbind"
```

Ohjekirjan voi myös avata sisällysluettelon kohdalta, eli ”otetaan manuaali käteen”:

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

Manuaalisivuilla on määrätty sisältörakenne:

1. Otsikko (topic)

2. Lyhyt kuvaus (Description)
3. Käyttäminen (Usage)
4. Parametrien kuvaus (Arguments)
5. Kuvaus (Details)

Lisänä voi olla yksi tai useampi seuraavista: Palautetun tuloksen kuvaus (Value), Aiheeseen liittyviä muita R-manuaalin sivuja (See Also), Kirjallisuusviitteitä (References), Esimerkkkejä käytöstä (Examples). Manuaali voi olla varsin tekninen, mutta selkeän rakenteen ansioista tarvittava tieto löytyy helposti.

Avun etsiminen on nopeaa:

```
> help.search("mean")
```

tai lyhyemmin `??`

```
> ??mean
```

Hakutulokset tulevat listattuna *paketeittain*, `paketti::olio`. Paketteja voi ajatella vähän niinkuin kaupunginkirjaston eri osastoja: Kaunokirjallisuus, historia, koirakirjat, jne. Eri paketit sisältävät eri asioita, vaikka kaikki kuitenkin sisältävät muuttujia ja funktiota. Tavallisimmat komennot löytyvät paketeista `base`, `utils` tai `stats` (”yleistieto”), joten niiden sisältä osuneet hakutulokset kannattaa tarkistaa ensimmäisenä.

Hyvä R:n käytön perussääntö on että kun törmätään uuteen funktioon niin luetaan edes lyhyt kuvaus siitä mitä se tekee. Näin sen takia ettei vahingossa tee mitään mitä ei ymmärrä. Joten kirjoitappa

```
> help(help)
```

Harjoituksia 2.2.6:

1. Etsi seuraaviin kysymyksiin vastaus manuaalista:
   - Mikä parametri muuttaa `read.table`-funktion erotusmerkkiä, eli merkkiä mikä erottaa taulukon solut toisistaan luettavassa tiedostossa?
   - Mitä kertoo `?Arithmetic`? Entä `sin`?

- Sinulla on kaksi vektoria, `A<-c(1,2,3)` ja `B<-c(2,3,4)`. Haluat yhdistää ne joukko-yhdisteenä eli $A \cup B$. Tekisit sen varmaan käsin kun ovat niin lyhyitä. Mutta entä jos ne olisivatkin paljon pidemmät? (hakusana 'union', kohta 'Set operations')
- Sinulla on vektori `pit<-c(1.6, 1.9, 2.0, NA, 1.5)`. Laske sen keskiarvo käyttäen `mean`-funktiota. Miten huomioit puuttuvan arvon?

2.3 Aineisto-tiedostojen lukeminen

Ennen pitkää olisi hyvä saada se oma aineisto istunnon sisään. Tähän R:ssä on monenlaisia tapoja. Niistä yksinkertaisin on tietenkin käsin syöttäminen kuten edellisessä kappaleessa tehtiin, mutta tämä käy hyvin nopeasti työlääksi.

R:ssä on monta tapaa tehdä asia helpoksi, kunhan vaan tiedetään missä muodossa aineistoa ollaan lukemassa. Yksinkertaisimmillaan näin: Olkoon meillä työhakemistossa tiedosto nimeltä **kesamokit1.txt**, jossa on tekstieditorilla kirjoitettu pötköön vuosittaisia kesämökkien lukumääriä Suomessa.

Tiedosto **kesamokit1.txt:**

```
176104 251744 367686 416236 450569 474277 485118
```

Se voidaan lukea R:ään komennolla `scan`:

```
> kesamokit<-scan("kesamokit1.txt", sep=" ")
> kesamokit
[1] 176104 251744 367686 416236 450569 474277 485118
```

`scan` lukee tiedostosta lukuja pötkössä, ja parametri `sep=" "` sanoo sille että luvut on erotettu välilyönnillä.

Pötkö numeroita ei ole kovin informatiivinen. Tiedosto **kesamokit2.txt** sisältääkin jokaiselle numerolle myös otsikoksi vuosiluvun, ja vielä kuvauksen tiedostosta. Lisäksi tiedostossa on toinen rivillinen numeroita:

Tiedostossa **kesamokit2.txt**

```
# Vuotuiset kesämökkien määrät koko Suomessa
          "v1970" "v1980" "v1990" "v1995" "v2000" "v2005" "v2009"
Suomi       176104 251744 367686 416236 450569 474277 485118
Keski-Suomi  11021  17638  25607  29792  33121  35369  34860
```

Käytetään sen lukemiseen monipuolisempaa `read.table`-funktiota:

```
> kesamokit<-read.table("kesamokit2.txt", header=TRUE, skip=1, row.names=1)
```

Huomaa `skip=1`: Se käskee ohittamaan tiedoston ylimmän rivin. Koska `header=TRUE`, tulkitaan ensimmäinen luettava rivi sarake-otsikoiksi, `row.names=1` määrittää ensimmäinen sarakkeen rivinimiksi. Vakioerottimena toimii yksi tai useampi tyhjä merkki.

R:ssä tiedostoon viittaaminen ei rajoitu vain kovalevylle: Voimme ihan hyvin käskyttää vaikkapa

```
> kesamokit<-read.table("http://users.jyu.fi/~tarajala/R/Aineistot/kesamokit2.txt",
+ header=TRUE, skip=1, row.names=1)
```

Tiedostoihin tulostaminen onnistuu myös. Muistellaan, että pelkkä muuttujannimi komentona tulostaa sisällön ruudulle:

```
> kesamokit
             v1970  v1980  v1990  v1995  v2000  v2005  v2009
Suomi       176104 251744 367686 416236 450569 474277 485118
Keski-Suomi  11021  17638  25607  29792  33121  35369  34860
```

”Ruutu” eli näyttö on R:lle vain yksi laite muiden muassa. Tässä kannattaa ajatella kuin tosiaan printtaisi: Mihin printteriin haluat printata? Vakioprintteri on ruutu. Voimme ohjata printtauksen tiedostoon `sink`-funktiolla (upota):

```
> sink(file="kesamokit3.txt")
```

Nyt kun printataan, eli

```
> kesamokit
```

mitään ei näy. Miksi? Koska printtasit tiedostoon! Kaikki mitä printtailet menee pötkössä tiedostoon. `sink()` ilman parametriä palauttaa tulostuksen ruudulle:

```
> sink()
```

Tarkista että tiedoston sisältö on nyt sama mitä olisi näkynyt ruudulla.

`sink` toimii hyvin kun haluat kirjoittaa tekstitiedostoon taulukon tai vektorin.

Kotitehtäviä 2:

1. Lataa maailman populaatioita kuvaava aineisto tiedostosta **populaatio.txt** kurssin nettisivuilta joko tallentamalla se työkansioon tai suoraan nettiosoitetta käyttäen. (huomioi parametrit `skip` ja `header`).
2. Lisää aineistoon rivi jossa on sarakkeiden totaalipopulaatiot. Tämän suorittaminen onnistuu kahdella tavalla:
   - Summaa ensimmäinen sarake, sijoita vektoriin `yht`. Summaa toinen sarake, lisää vektorin `yht` perään käyttäen, esim. `yht<-c(yht, summa)`.
   - Työlästä eikö totta? Siksipä tälle onkin olemassa pikatoiminto: `colSums(aineisto)`. Käytä sitä. (toki myös `rowSums` löytyy)
3. Lisää `yht` vektori taulukkoon käyttäen `rbind`-komentoa (row bind): `rbind(aineisto, Yhteensä=yht)`. Tässä siis määritellään riville nimikin. Mitä käy jos et anna nimeä?
4. Tallenna uusi aineisto tiedostoon (joko `sink` tai `save`, tai molemmat)
5. Piirretään vielä kuva: `plot(x=seq(1950,2050, by=10), y=aineisto["Yhteensä",], type="b")`, ”b” on lyhennys sanasta ”both”, tarkoittaen että piirrettään pallot sekä viiva. Kokeile ”l” ja ”p”.

3 Osa 3: Vielä tietorakenteista

Tutustutaan nyt tarkemmin tietorakenteisiin ja niiden käsittelyyn. Tarvitsemme `tiedot`-taulukkoa, joten lataa se käyttöön.

3.1 Tietorakenteiden käsittely

3.1.1 Järjestely

Vektorin järjestäminen suuruksien mukaan:

```
> x<-c(3.5, 5.1, 15, 1.9, 10, 23)
> sort(x)
[1]  1.9  3.5  5.1 10.0 15.0 23.0
```

Laskeva järjestys saadaan aikaan `sort(x, decreasing=T)`. Vektorin alkioiden järjestysluvut saapi funktiolla

```
> order(x)
[1] 4 1 2 5 3 6
```

Tästä saadaan indeksivektori, jonka mukaan poimittuina alkiot ovat järjestyksessä. Tämän avulla hoituu vaikkapa tietokehikon järjestäminen:

```
> jarj<-order(tiedot$pituus)
> tiedot[jarj,]
```

Harjoituksia 3.1.1:

1. Järjestä tiedot-taulukko BMI:n mukaan käyttäen `order`-funktiota. Koeta myös laskevaa järjestystä.

3.1.2 Kasvattaminen, kutistaminen, korvaaminen

Mikäli haluamme kasvattaa vektoria esim. uusilla havainnoilla, voimme tehdä sen monella tapaa. Esimerkiksi `c`-funktiolla voi lisätä havaintoja nopeasti vektorin alkuun tai loppuun: Olkoon havainnot

```
> hav<-c("e", "g", "h")
```

Lisätään alkuun ja loppuun:

```
> hav<-c("d",hav,"i")
```

Väliin lisääminen onnistuu `append`-funktiolla:

```
> append(hav, "f", after=2)
[1] "d" "e" "f" "g" "h" "i"
```

jossa `after=2` ilmoittaa sijainnin minkä jälkeen alkio asetetaan.

Kutistaminen onnistuu yksinkertaisesti päällesijoittamisella:

```
> hav[-2]
[1] "d" "g" "h" "i"

> hav<-hav[-2]
```

Vektorin ja taulukon alkioihin viitattiin indekseillä, esim.

```
> tiedot[1,3]
[1] 60

> tiedot$pituus[6]
[1] 1.91
```

Alkioihin viittamisen kautta voimme myös sijoittaa niihin tietoa, eli korvata alkioita yksittäin:

```
> tiedot[1,3]<-82
> tiedot$pituus[6]<-1.94
```

Harjoituksia 3.1.2:

1. Lisää havainto (mies, 1.89, 81) tiedot-taulukkoon, alkaen `tiedot[7,1]<-"mies"` jne.
2. Kun muutimme yksittäisiä pituus-arvoja ovat BMI ja yli25 nyt väärin. Korjaa taulu kuntoon eli aseta tarvittavat BMI ja yli25 arvot kohdalleen yksitellen, tai laske k.o. sarakkeet kokonaan uudestaan.
3. Mitä käy kun koetat `rbind`-funktion avulla lisätä tiedot-taulukkoon esimerkiksi vektorin `c("mies", 1.89, 81, 0, F)`?
4. Lisätään koko rivi: 1. Luo `uusirivi` tietokehikko jossa uudet tiedot (1 rivi). 2. Kopioi sille sarakenimet tiedot-taulukosta, eli `names(uusirivi)<-names(tiedot)`. 3. Koeta nyt `rbind`-funktiota.

3.1.3 Matriiseista

Matriiseilla ja taulukoilla on ns. dimensio-ominaisuus. Siihen pääsee käsiksi `dim`-funktiolla:

```
> dim(tiedot)
[1] 6 5
```

Luodaan kaksi matriisia kahdella eri tavalla: `cbind`-komennolla

```
> rivi1<-c(1,2,3)
> rivi2<-c(4,5,6)
> A<-rbind(rivi1, rivi2)
```

joka on 2×3 matriisi. Tehdään toinen `matrix`-komennolla:

```
> alkiot<-1:12
> B<-matrix(alkiot, byrow=FALSE, nrow=3)
```

Josta tulee 3×2 matriisi (latoo alkiot `byrow=T` eli riveittäin, `nrow=3` kolmeen riviin). Matriisien tulo, AB, tehdään R:ssä omalla tulomerkillään:

```
> A%*%B
      [,1] [,2] [,3] [,4]
rivi1   14   32   50   68
rivi2   32   77  122  167
```

Harjoituksia 3.1.3:

1. Mitä käy jos käytät tavallista tuloa, eli vaikkapa `A*A`? (A kuten tekstissä)
2. Laske $(AA^T)^{-1}$. Käänteismatriisin ratkaisee `solve`-funktio, transpoosin sai `t`-funktiolla.

3.1.4 Monistaminen

Säännöllistä toistoa sisältävien vektoreiden luominen onnistuu `rep` (`rep`eat) funktiolla:

```
> x<-rep(1, 5)
> x
[1] 1 1 1 1 1
```

Voimme toistaa myös pitempää sarjaa:

```
> y<-rep(1:3, 3)
> y
[1] 1 2 3 1 2 3 1 2 3
```

Toisenlainen kuvio, nyt jokaista vektorin alkiota tietty määrä:

```
> z<-rep( c("a","b","c"), c(2,4,6))
> z
 [1] "a" "a" "b" "b" "b" "b" "c" "c" "c" "c" "c" "c"
```

3.1.5 Pilkkominen

Aineiston pilkkominen osajoukoiksi on usein tarpeen, jos ei muuta niin jatkokäsittelyn takia tai siksi että kiinnostaa vain tietty osajoukko isosta aineistosta.

Otetaan tässä välissä hieman isompi aineisto käyttöön R:n sisäisestä kirjastosta käyttäen komentoa `data`:

```
> data(morley)
> names(morley)
[1] "Expt"  "Run"   "Speed"
```

`morley`-aineisto sisältää tietokehikossa mittauksia valonnopeudesta, 20 mittausta per viisi koetilannetta eli yhteensä 100 havaintoa.

Pilkotaan `morley`-aineiston havaitut nopeudet kokeiden suhteen käyttäen funktiota `split`:

```
> morley.l<-split(morley$Speed, morley$Expt)
> morley.l
```

Ensimmäiseksi parametriksi annetaan ”mikä” pilkotaan ja toiseksi ”minkä mukaan”. Tuloksena on lista jossa osavektoreina ovat kokeita vastaavat havainnot.

Harjoituksia 3.1.5:

1. Pilko tiedot-taulukko miehiin ja naisiin.
2. Pilko tiedot-taulukko niihin joilla BMI yli 20 ja muihin.

3.1.6 Toistaminen

Usein halutaan laskea samoja laskuja osajoukoille, esimerkiksi `morley`-aineisto jokaisesta kokeesta haluttaisiin keskiarvonopeus. Voimme tehdä sen koe kerrallaan, vaikkapa käyttäen totuusindeksointi:

```
> expt1<- ( morley$Expt==1 )
> koe1arvot<-morley$Speed[expt1]
> mean(koe1arvot)
[1] 909
```

tai listarakennetta hyödyntäen

```
> mean(morley.l$"1")
[1] 909

> mean(morley.l$"2")
[1] 856
```

ja niin edelleen. R:ssä tämä *toistaminen*, ”tee joukon jokaiselle alkiolle”, on yleistä puuhaa ja voidaan toteuttaa erinäisin `apply`-komennoin.

Listoille ja tietokehikoille toistamisen voi toteuttaa funktiolla `lapply` sekä sen yksinkertaistuksella `sapply`:

```
> lapply(morley.l, mean)
```

`lapply` palauttaa tuloksen listana, kun taas

```
> sapply(morley.l, mean)
    1     2     3     4     5
909.0 856.0 845.0 820.5 831.5
```

yksinkertaistaa tulosta palauttaen vektorin tai matriisin.

Taulukkorakenteille (matriisi, tietokehikko) toistamisen toteuttaa `apply`. Esimerkki: Luodaan koekohtaisesta listasta 20×5-matriisi ja lasketaan sarakkeiden keskiarvot. Matriisin luonti onnistuu esimerkiksi `matrix`-komennolla:

```
> U<-matrix(morley$Speed, byrow=FALSE, nrow=20)
> dim(U)
[1] 20  5
```

`byrow=FALSE` sanoo että täytetään matriisia ensin sarake täyteen, `nrow=20` kertoo että rivejä per sarake halutaan 20. Tämä toimii koska aineisto on sopivasti kokeittain järjestyksessä, ja koe on tasapainossa eli havaintoja per koe on yhtämonta (5×20).

Lasketaan sitten sarakekeskiarvot:

```
> apply(U, MARGIN=2, mean)
[1] 909.0 856.0 845.0 820.5 831.5
```

`MARGIN=2` tarkoittaa sarakkeita (`MARGIN=1` rivejä). `apply` kutsuu funktiota `mean` jokaiselle sarakevektorille erikseen. Tämä on siis sama kuin edellä, nyt vain matriisirakenteella laskettuna.

Lopuksi vielä ryhmittelyn itse tekevä `tapply`, jolle annetaan aineistovektori sekä vektori joka kuvaa miten aineisto ryhmitellään:

```
> tapply(morley$Speed, morley$Expt, mean)
    1     2     3     4     5
909.0 856.0 845.0 820.5 831.5
```

Harjoituksia 3.1.6:

1. Käyttäen `split`- ja `sapply`-funktiota laske `tiedot`-taulukon miesten ja naisten pituuksien keskiarvot.

3.2 Tasomuuttujat eli faktorit

Usein käsittelemme faktori- eli taso- tai kategoriamuuttujia. Ne ovat muuttujia jotka kuvaavat esimerkiksi havainnon koe-erää, sukupuolta, sosiaalista luokitusta tai kuntoluokitusta. Yleensä tason arvot koodataan kokonaisluvuiksi tai kirjaimiksi.

Kokonaisluvun tai kirjaimen sijaan tasomuuttujana on R:ssä hyvä käyttää omaa tietotyyppiänsä `factor`, jolloin moni funktio osaa käsitellä sitä oikein eikä esimerkiksi tulkitse sitä lukuarvoksi. Lisäksi tasojen järkevä nimeäminen auttaa tuloksien tulkintaa (”terve”, ”sairas”vai ”A”,”B”?).

3.2.1 Faktoreiden luominen

`factor`-muuttuja voidaan luoda mistä vaan vektorista `factor`-funktiolla:

```
> koe<-morley$Expt
> fkoe<-factor( koe, levels=1:5)
> fkoe
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [38] 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4
 [75] 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
Levels: 1 2 3 4 5

> levels(fkoe)
[1] "1" "2" "3" "4" "5"
```

`levels`funktio palauttaa tasomuuttujan mahdolliset tasot.

Jotta `R` ymmärtää kokeiden numerot vain koodeiksi, luodaan tasovektori komennolla `factor`, jolle annetaan parametrina kaikki mahdolliset tasot: Tämä on tärkeä antaa jos haluamme varmistaa että myös havaitsemattomat tasot ovat tiedossa (esim. koe 6 data ei vielä saatavilla, mutta tullaan saamaan).

Annetaan tasoille nimet:

```
> levels(fkoe)<-c("koe1", "koe2", "koe3", "koe4", "koe5")
> fkoe
```

Huomaa sijoitusoperaattorin toiminta. Jotkin funktiot, kuten `levels` ja `names`, voivat esiintyä sijoitusoperaattorin vasemmalla puolella.

Moni `R`:n funktio osaa käsitellä tasomuuttujia:

```
> morley$Expt<-fkoe
> boxplot(Speed ~ Expt, data=morley)
```

Annettu termi `Speed` $\sim$ `Expt` on eräänlainen yhtälö: ”Speed:iä selittää Expt”. `boxplot` ymmärtää sen ryhmittelynä koska `morley$Expt` on tasomuuttuja. `data=morley` ohjaa etsimään muuttujia tietokehikon sisältä.

Harjoituksia 3.2.1:

1. Luo tasomuuttuja `BMIluokat` joka kuvaa tiedot-taulukon BMI-arvoa ryhmissä Alle 20, välillä 20-25, yli 25. Lisää muuttuja taulukkoon.

3.2.2 Tyyppimuunnoksista

Joskus on tärkeä muuttaa arvoja tyypistä toiseksi, kuten esimerkiksi edellä koodinumerovektorimme oikeaksi tasomuuttujaksi. Voimme muuttaa tietotyypistä toiseen `as.`*tyyppi*-funktiolla,

esimerkiksi `as.numeric`, `as.vector`, `as.factor` ja `as.matrix`, jolloin `R` tekee parhaansa muuttaakseen muuttujan haluttuun muotoon:

```
> is(123)
[1] "numeric" "vector"
```

luku 123 on numero sekä yhden mittainen vektori,

```
> c123<-as.character(123)
> is(c123)
[1] "character"           "vector"              "data.frameRowLabels"
[4] "SuperClassMethod"
```

muunnettiin se tekstiksi. Toisinpäin ei ole niin helppoa:

```
> as.integer("abc")
[1] NA
```

nyt tulos on puuttuva arvo (`NA`) koska tekstin ”abc” muuttamista numeroksi ei ole yksikäsitteisesti määritelty.

Hyödyllinen on myös

```
> as.factor(1:5)
[1] 1 2 3 4 5
Levels: 1 2 3 4 5
```

Jolla voi nopeasti muuttaa esim. tekstistä faktoriksi. Tämä on osana mm. tietokehikkoa jos annetaan merkkijono-vektori.

3.2.3 Tulosten rakenteesta

Listat ovat kokoelmia erilaisia tietorakenteita, ja usein `R`:n komennot palauttavatkin tuloksensa monipuolisena listana jossa on tietoa annetuista parametreistä, laskennon tuloksista, tarkkuuksista jne.

Huomaa että se mitä tulostuu ruudulle ja se mitä itseasiassa palautetaan, ei aina ole sama asia: Usein funktion tuloksella on ikioma tulostustapansa, riippumatta tuloksen tietotyypistä:

```
> fit1<-lm(Speed ~ Expt, data=morley)
> fit1
```

```
Call:
lm(formula = Speed ~ Expt, data = morley)

Coefficients:
(Intercept)     Exptkoe2     Exptkoe3     Exptkoe4     Exptkoe5
      909.0        -53.0        -64.0        -88.5        -77.5

> names(fit1)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

eli `fit1`-muuttujassa oleva olio sisältää useita elementtejä joita ei vakiona tulostu ruudulle.

`lm`-funktion tulos on lista, ja voimme poimia siitä osia kuten totuttua,

```
> fit1$coefficients
(Intercept)    Exptkoe2    Exptkoe3    Exptkoe4    Exptkoe5
      909.0       -53.0       -64.0       -88.5       -77.5

> plot(fit1$residuals, col=morley$Expt, pch=19, ylab="residuaalit")
> abline(h=0)
```

Residuualit tulostuvat ruudulle. `abline` lisää suoran, tässä tapauksessa $h = 0$ eli horisontaalisuora kohdasta 0.

Joskus tuloksilla on vielä oma `summary`:säkin, jolloin saadaan ruutu täyteen tietoa ja tulostusta:

```
> summary(fit1)
```

Tuloste riippuu aina tilanteesta.

---

4 Osa 4: Grafiikkaa

Tutkitaan ensiksi miten peruskuvaajat saadaan tehtyä, ja sen jälkeen katsotaan miten niiden tuottamaa jälkeä saadaan viriteltyä.

4.1 Kuvien rakentaminen

Kuvien saattaminen haluttuun muotoon muodostuu paloittain: Tyypillisesti piirrämme ensin runkokuvan ja lisäämme siihen sitten osia, kuten lisää kuvaajia, havaintopisteitä tai kuvatekstejä.

Kuvien piirtämiseen käytettävät funktiot voidaan jakaa kahteen ryhmään: Korkean ja matalan tason funktioihin. Korkean tason funktioita ovat sellaiset funktiot, jotka kutsuttuina luovat uuden kuvan (esim. ikkunan), visualisoivat aineiston ja lisäävät kuvaan vielä tarpeelliset akselit ja tunnisteet. Matalan tason funktiot taas suorittavat näistä tehtävistä vain osan, esimerkiksi lisäävät valmiiseen kuvaan x-akselin, tekstiä tai viivoja.

Tärkein korkean tason funktio on `plot`. Se osaa käsitellä monenlaisia eri tietotyyppejä, ja jos ei muista erityistä piirtokomentoa kannattaa kokeilla `plot`tia:

```
> data(morley)
> plot(morley)
```

Mitä tapahtuu? Koska `morley` on taulukko, eli käytännössä annamme nipun (sarake)vektoreita, piirtää `plot` meille niistä pareittaiset hajontakuviot (itseasiassa `plot` kutsuu edelleen funktiota `pairs`). Hajontakuvio on usein vakiotoiminto kun annetaan lukuvektoreita. Plotataan hajontakuvio antamalla halutut kaksi vektoria:

```
> plot(x=morley$Expt, y=morley$Speed)
> plot(morley$Expt, morley$Speed)
```

Tästä voidaan jättää parametrinimet `x=, y=` pois sillä kyseisiksi parametreiksi tulkitaan automaattisesti kaksi ensimmäistä annettua parametriä.

Matalan tason funktioilla voi lisätä korkeamman tason kuvaan lisää yksityiskohtia. Esimerkkeinä

- pisteiden lisääminen, `points`, tai viivojen piirtäminen, `lines` tai `curve`,
- Suoran lisääminen käyttäen kulmakerrointa, `abline`,
- tekstin lisääminen, `text`,
- Symboli- ja viivaselitteet, `legend`.

Harjoituksia 4.1.0:

1. Kokeile seuraavaa:
    - `xvec<-seq(0, 1, by=0.01)`

- `plot(xvec, 2*exp(xvec), type="l")`

2. Tee nyt näin: `curve( 2*exp(x), from=0, to=1)`.
3. Piirrä $\frac{1}{\sqrt{2\pi}} \exp(-0.5x^2)$ kuvaaja välillä [-3,3].

4.1.1 Kuviin liittyviä parametrejä

Kuvia rakentaessa hyödyllisiä parametrejä ovat muun muassa `main="Otsikko"` eli kuvan otsikko, `xlim,ylim` x- ja y-akselin alueen määräävät (alku, loppu)-vektorit, akselien tekstit `xlab="x akseli"` ja `ylab="y akseli"`, erilaiset väriparametrit kuten `col`, `bg` ja `fg` jotka muuttavat yleistä piirtoväriä ja riippuen funktiosta taustan ja edustan väriä.

Näiden lisäksi eri funktioilla on myös omia parametrejä joista voi katsoa tietoa manuaalista kyseisen funktion kohdalta. Yleisiä graafisia parametrejä voidaan määrittää kahdella tavalla: Väliaikaisesti piirtofunktion kutsussa (muutetaan piirtomerkki, `pch`)

```
> x<-runif(100)
> plot(x, pch="+", main="Tasajakauma")
```

tai kiinteästi `par`-funktion avulla

```
> par(pch="+")
> plot(x, main="Tasajakauma")
```

Jälkimmäisessä asetus säilyy myös seuraavaan `plot`-komentoon, kun ensimmäisessä asetus palautuu vakioarvoon (`pch=1`).

`par`-funktiolla voidaan määrittää myös itse laitteen asetuksia, kuten marginaalien tai piirtokankaan kokoja, tai jakaa kuva-alue osiin:

```
> par(mfrow=c(1,2), pin=c(2,2))
> plot(x)
> hist(x)
```

Tässä `mfrow=c(1,2)` (multiple figure, rows) määrittää kuvalaitteen jaon 1×2 eli yksi rivi, kaksi saraketta. `pin=c(2,2)` säätää kuvaajan kooksi tuumina 2×2. Yleisistä grafiikkaparametreistä voi katsoa lisää `par`-funktion manuaalisivulta.

Kuvat rakennetaan paloissa: Korkean asteen funktiolla piirretään pohjakuva, matalan tason funktiolla lisätään yksityiskohtia. Säädetään parametrejä, toistetaan kunnes hyvä. Esimerkiksi seuraavasti, missä aineistossa auton nopeuksia ja jarrutusmatkoja:

```
> data(cars)
> plot(cars$speed, cars$dist)
```

Sovitetaan lineaarinen regressio, eli $dist = a + b \cdot speed + e$, jossa virhe e oletetaan normaaliksi. Sovitus onnistuu `lm`-funktiolla (`l`inear `model`):

```
> fit1<-lm(dist ~ speed, data=cars)
```

Lisätään sovitettu käyrä $y = a + bx$ kuvaan käyttäen `abline`- (`a`-`b`-line, vakio-kulmakerroin-suora):

```
> abline(fit1)       # abline-funktio osaa poimia kertoimet lm-tuloksen sisältä
```

Katsotaan jäännöksiä erikseen

```
> hist(fit1$residuals, freq=F)
```

Piirretään vielä lopuksi kuvat vierekkäin ja viimeistellään:

```
> par(mfrow=c(1,2), ps=10)
> # kuva 1
> plot(cars$speed, cars$dist, xlab="Nopeus", ylab="Jarrutusmatka",
+      main="Autojen jarrutusmatkoja 1920-luvulla", pch=19)
> abline(fit1, col=2)
> legend("topleft", paste("b=",round(fit1$coefficients[2],3), sep=""))
> # kuva 2
> hist(residuals(fit1), freq=F, col="gray50", xlab="", main="Jäännökset")
```

Tulokseksi saadaan kuva joka selvittää aineiston lineaarista riippuvuusrakennetta. Jäännöskuvaajasta nähdään että malli ei välttämättä ole hyvä sillä jäännösten kuvaaja ei ole aivan symmetrinen.

4.1.2 Väreistä

Yleinen väriparametri on nimeltään `col`. Se säätää yleensä merkin/viivan väriä. Se voidaan antaa jonakin seuraavista:

- Kokonaisluku väliltä 1-8: Ne viittaavat hyvin erottuviin vakioväreihin (musta, punainen, vihreä,...).
- R:n sisäinen värinimike, esim. `col="blue"`. Värit voi listata funktiolla `colors()`.
- Liukuvärinä: Puna-viher-sini-arvoilla määritelty väri, funktio `rgb`, tai HSV-arvo `hsv`, tai harmaasävyarvo, funktio `gray(arvo)` missä `arvo` välillä 0-1.

Usein voidaan antaa myös vektori värejä, esimerkiksi kun piirretään useita pisteitä tai aineisto on ryhmitelty tasojen avulla. Toteutus riippuu piirtäjäfunktiosta. Usean värin antamiseen on helpotuksina kokonaislukuvektorien kuten `1:4` lisäksi liukuvia värivektoreita luovat funktiot kuten `rainbow, heat.colors, terrain.colors, topo.colors, cm.colors`, esimerkiksi

```
> x<-1:10
> xy<-expand.grid(x,x) # luo hilan
> plot(xy, cex=3, pch=15, col=heat.colors(100)) # 100 askelta
```

Mikäli värivektorin pituus ei vastaa piirrettävien osien määrää, kierrätetään sitä alusta tarpeellinen määrä. Sama käy yleisesti myös muille asetuksille, kuten vaikkapa piirtomerkille `pch` tai kokoa muuttavalle `cex`-parametrille:

```
> plot(1:10, rep(0,10), pch=1:5, col=1:5, cex=1:5)
```

4.2 Valmiit piirturit

Seuraavaksi katsotaan erityisiä kuvanpiirtofunktioita ja niihin liittyviä parametrejä.

4.2.1 Histogrammi ja muut 1-ulotteisen vektorin jakaumakuvaajat

Histogrammi, eli lukujen esiintymiskertojen määrää esittävä pylväskuvaaja, saadaan komennolla `hist`:

```
> lukuja<-morley$Speed
> hist(lukuja)
```

Histogrammi ottaa aineiston vastaan vektorina.

Histogrammille ominaisina parametreinä voidaan esimerkiksi antaa niputusvälien lukumäärä, määrätä näytetäänkö frekvenssi vai suhteellinen osuus ja lisätäänkö lukumäärät kuvaan. Lisäksi käytössä ovat yleiset otsikko-, väri- ja kokoasetukset kuten edellä mainittiin:

```
> hist(lukuja, breaks=20, col="black", border="white", freq=F, main="Nopeuksia")
```

`freq=F` pyytää absoluuttisten frekvenssien sijaan suhteelliset osuudet, `breaks=50` määrittää arvojaon lukumäärän (pylväiden lukumäärän), `col=1` sanoo että väri 1 (joka on musta), `border="white"` asettaa pylväänreunan valkoiseksi ja `main` muuttaa kuvan otsikon.

Funktio `density` tuottaa ydinestimoidun tiheysfunktion. Lisätään se kuvaan `lines`-komennolla:

```
> lines(density(lukuja, bw=0.3), col=4, lwd=2)
```

Ydintasoituksen pehmeyden `bw` arvoa on hyvä vaihdella ja kokeilla mikä sopii parhaiten.

Kuvaan voi lisätä havainnot vaikkapa komennolla

```
> rug(lukuja, col="red")
```

Aineiston empiiristä jakaumaa voi tarkastella myös kumulatiivisella kertymäfunktiolla,

```
> kkf<-ecdf(lukuja)
> plot(kkf, do.points=FALSE, verticals=TRUE)
```

`do.points=FALSE` kieltää piirtämästä pallukoita kuvaajan solmukohtiin, ja `verticals=TRUE` lisää porrasfunktion solmukohtien pystyviivat.

4.2.2 Luokiteltujen aineistojen kuvaajia

Olkoon nyt aineistossa mitattuna reaalilukuja, joiden lisäksi on vähintää yksi tasomuuttuja kuvaamassa havaintojen ryhmittelyä. Tällaista aineistoa voi kuvata helposti viiksilankakuvaajalla eli `boxplot`illa (box-and-whiskers):

```
> data(morley)
> boxplot(Speed ~ Expt, data=morley)
```

Tässä käytetään yleensä R:n mallisyntaksia $y \sim x$ joka luetaan "y:n määrittää x". `boxplot` haluaa x:n tasomuuttujana ja muuttaa muut tietotyypit tasomuuttujaksi `as.factor`-funktion avulla.

Käytössä on yleisten väri- ja kokoasetusten lisäksi erikoisparametrejä, kuten poikkeavuuksien (outlier) määrittämiseen liittyvä `range` ja osajoukkoja valitseva `subset`:

```
> expt4<- morley$Expt < 4
> boxplot(Speed ~ Expt, data=morley, subset=expt4, range=0, col="gold",
+ main="Valonnopeuksia")
```

Lisää tietoa manuaalista `?boxplot`.

`coplot` piirtää ryhmittäiset hajontakuviot: Sille annetaan `boxplot`in tapaan kaava, missä jaottelevat ehto annetaan muodossa $y \sim x|ehto$. Katsotaan aineistoa, missä on mitattu viideltä eri puulta ympärysmitta usessa eri iässä:

```
> data(Orange)
> coplot(circumference ~ age | Tree, data=Orange, show.given=FALSE)
```

`show.given=FALSE` poistaa kuvasta luokkien frekvenssit -osion.

4.2.3 Taulukkoaineiston kuvaajat

Taulukoitujen aineistoja voi havainnoillistaa vaikkapa pylväsdiagrammilla, funktio `barplot`. Se ottaa pylvään korkeudet vektorina:

```
> data(Orange)
> karvot<-tapply(Orange$circumference, Orange$age, mean)
> barplot(karvot, xlab="Ikä päivissä")
```

Säätöjä ovat mm. usean pylvään pinoamisen sijaan vierekkäin asettaminen, pylväiden piirto vaakatasoon, välistyksien säätö, nimikkeiden säätö:

```
> arvot<-matrix(Orange$circumference, ncol=7, byrow=T)
> colnames(arvot)<-unique(Orange$age)
> rownames(arvot)<-paste("puu", 1:5,sep="")
> barplot(arvot, main="Puiden koko eri iässä", xlab="Ikä päivinä", ylab="cm",
+ beside=T, legend.text=T, args.legend=list(x="topleft"))
```

Jaoteltua aineistoa voi kuvata myös `dotchart`-funktiolla. Se on hyvä kuvaamaan kahden tason keskiarvoja. Esimerkiksi aineisto, jossa on 150 kasvia, kolmea eri lajia, jokaiselta lajilta mitattu terälehden (petal) sekä verholehden (sepal) keskimääräinen pituus ja leveys:

```
> data(iris)
> # lasketaan keskiarvot käyttäen apply-funktioita
> irism<-apply(iris[,1:4], MARGIN=2, tapply, iris[,5], mean)
> dotchart(irism, col=2:4, pch=2:4, xlab="cm", main="Keskimääräiset koot, n=50/ryhmä")
```

4.2.4 Muita kuvia tarpeen mukaan

R:stä löytyy lisäksi paljon muita kuvantuottoon sopivia funktiota. Otetaan tässä useimmin käytetyt esimerkit:

Kaksiulotteisia hiloja voidaan kuvata `image`-funktiolla: Matriisiaineistossa `volcano` on 870mx610m neliöalueesta 10m välein tallennettu korkeus meren pinnasta:

```
> data(volcano)
> image(volcano, col=topo.colors(125), axes=F, zlim=range(0, volcano))
```

Tähän voidaan lisätä reunaviiva-kuvaaja eli `contour`-kuvaaja,

```
> contour(volcano, add=TRUE)
```

Kolmiulotteisia kuvia saa myös aikaan, esim.

```
> persp(z=volcano, theta=30, box=T)
```

Lisäkirjastoista saa potkua kuviin, mm. kirjastot `lattice`, `rgl` ja `ggplot2` ovat suosittuja laajennuksia R:n perusgrafiikkaominaisuuksiin.

4.3 Grafiikkalaitteet eli mihin piirretään

R osaa tallentaa grafiikkaa niin vektori-grafiikkana (kuvan osat vapaasti suurennettavissa) kuin rasterigrafiikkaa (pikselikuvia).

R:ssä grafiikan tuottamiseen käytetään 'laitteita' (devices), jotka voi tulkita vaikkapa piirtoalustoiksi. Kun haluamme piirtää kuvan, ensin valitsemme 'laitteen', esimerkiksi uuden ikkunan tai tiedoston kovalevyltä, ja sen jälkeen piirrämme sille valitsemallamme piirtotyökalulla. Vertaa `sink`-funktion toimintaan.

Oletuksena R piirtää grafiikka-ikkunaan. Mikäli grafiikka-ikkunaa ei ole vielä luotu kun piirrämme, se luodaan ja kuva piirretään siihen. Esimerkkinä piirretään auto-aineisto:

```
> plot(x=cars$speed, y=cars$dist, type="b")
```

`plot`-funktion kutsuminen avaa uuden grafiikka-ikkunan mikäli sellaista ei vielä ole auki, ja kuvaaja piirretään sille. `type="b"` (both points and lines) saa aikaan sen että pisteiden väliin tulee viiva.

Ohjaus tiedostoon onnistuu erillisillä laite-funktioilla. Piirretään edellinen kuva pdf-tiedostoon: Ensin avataan 'tiedostolaite', sen jälkeen säädetään laitteen parametrit `par`-funktiolla, sitten piirretään, ja lopuksi suljetaan tiedosto.

```
> pdf(file="kesamokit.pdf", width=5, height=5)
> par(cex=1.2, col=2)
> plot(x=cars$speed, y=cars$dist, type="b")
> dev.off()
```

Huomaa että nyt on myös muistettava sulkea laite komennolla `dev.off()`: Muuten laite, eli tässä tapauksessa tiedosto, jää 'auki' eli jää vastaanottamaa lisää grafiikkaa, ja yleensä menee sekaisin kun piirrämme seuraavan kuvan sinne vahingossa.

Sama kuva saataisiin postscript-tiedostoon

```
> postscript(file="kesamokit.eps", width=5, height=5, horizontal=F)
> par(cex=1.2)
> plot(x=cars$speed, y=cars$dist, col=2, type="b")
> dev.off()
```

Jossa annetaan nyt laitteelle ominainen `horizontal`-parametri (jotka on tarkistettava ohjeista). Ja niin edelleen esimerkiksi png-kuvaksi `png`-funktioilla.

Kotitehtäviä 4:

1. Piirrä morley-datan Speed-havainnoista hajontakuvio (`plot(morley$speed)`) siten että 1) Nimeät sekä x- ja y-akselin että otsikon (xlab, ylab, main), 2) lisäät vaakasuoran keskiarvoviivan (`abline(h=keskiarvo)`), 3) Eri kokeiden havainnoilla on oma piirtosymboli (pch).
2. Kuinka saat `curve`-funktion lisäämään viivan jo piirrettyyn kuvaan? Piirrä sillä polynomien x^k, missä $k = 1, 2, 3$, kuvaajat samaan kuvaan. Vaihda viivatyyppiä ja väriä jotta kuva on selkeä, säädä y- ja x-akselia akselia tarpeen mukaan. Lisää `legend`-komennolla kuvaselite.
3. Katso kuvalaitteista Devices-manuaalisivulta, `?Devices`.
4. Rakenna ja koristele morley-aineistosta kuva jossa
   - on vierekkäin kaksi ruutua,
   - Ensimmäisessä on morley-aineiston boxplot,
   - Toisessa on kaikkien nopeuksien histogrammi.
   - Lisää histogrammiin keskiarvon sekä 95%:n luottamusvälin päiden kohdille pystyviivat. Välin pisteiden laskemisessa käytä kaavaa $mean \pm 2 * \sqrt{Var/n}$. (pystyviivan saat esim. `abline(v=keskiarvo)`).
   - Piirrä kuva tiedostoon valitsemassasi tiedostotyypissä eli valitse Devices-osiosta mieleisesi piirtolaite ja käytä sitä.
5. Valitse Osa 4:n esimerkkikuvaaja ja tutustu sen parametreihin tarkemmin ohjeista. Muuttele parametrejä ja katso mitä tapahtuu.

5 Osa 5: Ohjelmointikielen perusteita

5.1 Omat funktiot

`R` on lausekekieli: Kaikki komennot kuten funktiokutsut ja sijoitusoperaatiot ovat *lausekkeita*. Lausekkeet palauttavat jonkin arvon.

Lausekkeita voidaan ryhmitellä käyttäen aaltosulkuja `{lauseke1; lauseke2; ...}`, joko puolipisteellä tai rivinvaihdolla erotettuna. Tällaista *lausekeryhmää* kutsuttaessa lausekkeet toteutetaan järjestyksessä ja lausekeryhmän viimeisen lausekkeen arvo palautetaan `R`-ympäristöön. Tärkeä huomio tässä on se että lausekeryhmä on edelleen myös yksikössä lauseke. Se voi olla siis osaa suurempaa ryhmää, ja niin edelleen.

Luodaan lausekeryhmä ja sijoitetaan se muistiin muuttujaan:

```
> lau1<-{x<-1; y<-cos(x); y}
```

Sen sisältö suoritetaan nimellä kutsumalla,

```
> lau1
[1] 0.5403023
```

Luettavuuden kannalta voimme kirjoittaa, jo totuttuun tapaan, lausekeryhmän lausekkeet myös omille riveilleen:

```
> lau1<-{
+   x<-1
+   y<-cos(x)
+   y
+ }
```

Kun kutsumme `lau1`, suoritetaan järjestyksessä lausekeryhmän lausekkeet, eli ensin `x<-1`, sitten `y<-cos(x)` ja lopuksi pelkkä `y`, joka ollessaan ryhmän viimeinen lauseke on myös koko lausekeryhmän palautusarvo.

Lausekeryhmä on jäykkä rakenne siinä mielessä että kun se on kerran määritelty, niin muutokset sen toimintaan vaativat uuden määrittelyn. Usein tarvitsemme lausekeryhmiä, joiden toimintaa voimme kontrolloida kutsun aikana. Se onnistuu lausekeryhmän yleistyksellä, jo tutuksi tulleella *funktio*-rakenteella. Funktio on siis lausekeryhmä, jonka sisäistä ajonaikaista muuttuja-avaruutta voimme säädellä kutsun aikana annettavilla *parametreillä*.

Näin muutamme lausekeryhmän `lau1` funktioksi:

```
> fun1<-function()
+ {
+   x<-1
+   y<-cos(x)
+   y
+ }
> fun1()
[1] 0.5403023
```

Mikä muuttui? `fun1`:ksi sijoitetaan lausekeryhmän sijaan funktio käyttäen `function(){ ... }`-rakennetta. Sitä täytyy nyt kutsua sulkujen `()` kera, muuten ei oikeastaan ole eroa.

Funktio-rakenteen voima on parametrityksessä: Voimme määrittää lausekeryhmän sisäiseen muistiavaruuteen omia muuttujia:

```
> fun2<-function(x=1)
+ {
+   y<-cos(x)
+   y
+ }
> fun2(1)
[1] 0.5403023

> fun2(pi)
[1] -1
```

Harjoituksia 5.1.0:

1. Kirjoita funktio, jolle annetaan parametri `x` ja joka palauttaa arvon $\exp(-0.5x^2)$.

5.2 Parametrit alias argumentit, nimetyt parametrit sekä '...'

Funktio määrittämisen yleinen rakenne on muotoa

```
nimi<-function(parametri1, parametri2, ...)
    lausekeryhmä
```

missä `parametri1` on ensimmäinen parametri (argumentti), `parametrit2` on toinen parametri, ja niin edelleen. Esimerkiksi

```
> fun3<-function(x, y, z) {
+     (x+y)/z
+ }
> fun3(1, 2, 3)
[1] 1
```

Argumenteille voidaan antaa vakioarvot määrittelyvaiheessa:

```
> fun3<-function(x, y=2, z=3) {
+   (x+y)/z
+ }
> fun3(1)
[1] 1
```

Tässä siis numero `1` menee järjestyksessä ensimmäisen argumentin `x` arvoksi, ja `y,z` pitävät vakioarvonsa 2 ja 3. Argumenttien nimeäminen mahdollistaa järjestyksen muuttamisen:

```
> fun3(y=2, 1)
[1] 1
```

Tässä siis `x` saa ensimmäisen nimeämättömän arvon, ja `z` pitää vakioarvonsa. Tämä mahdollistaa sen että esimerkiksi komennot `plot(x, col=2, pch=2)` ja `plot(x, pch=2, col=2)` eivät eroa toiminnaltaan.

Funktion (eli lausekeryhmän) sisällä on aina oma muistiavaruutensa:

```
> x<-5
> fun3(x=1)
[1] 1

> x
[1] 5
```

Funktion sisäinen muuttuja `x` ja R-ympäristön muuttuja `x` elävät omaa elämää. Mutta: Jos lausekeryhmän sisällä käytetään muuttujaa jota ei ole lausekeryhmässä määritelty, etsitään R-ympäristön muuttujaa mikäli sellainen löytyy.

R:ssä on lisäksi erityinen 'parametri' nimeltään '`...`' (kolme pistettä). Se on vapaasti vaihteleva määrä ennalta määräämättömiä parametrejä, siis parametrijoukko. Sen hyöty on siinä että se voidaan antaa funktion sisällä eteenpäin toisille funktioille. Esimerkiksi näin:

```
> piirra<-function(x, ...) {
+   z<-exp(x)
+   hist(z, main="Exp-muunnos", ...)
+   mean(z)
+ }
> lukuja<-rnorm(100)
> piirra(lukuja)
[1] 1.33701

> piirra(lukuja, col="black", border="white")
[1] 1.33701
```

Tässä kaikki parametrin `x` lisäksi annetut lasketaan `...`-parametrijoukoksi, ja ne välitetään sellaisenaan `hist`-funktiolle.

Harjoituksia 5.2.0:

1. Mitä käy jos kutsut funktiota `piirra` ilman parametrejä, `piirra()`? Entä ilman sulkuja, `piirra`?

2. Luo funktio joka tekee seuraavan: 1) Ottaa kaksi nimettyä parametriä `x, lambda`, sekä vapaat parametrit '...', 2) Laskee muunnoksen `y<-x^lambda`, 3) piirtää histogrammin luvuista `y` käyttäen vapaita parametrejä '...' omina parametreinään. 4) palauttaa muunnetut arvot.

3. Testaa funktiota `cars`-aineiston jarrusmatkoilla (`cars$dist`, muista `data`-funktio). Millä vakion `lambda` arvolla, eli millä potenssimuunnoksella, saat mielestäsi 'normaalisoitua' datan, eli saat histogrammin joka näyttää kellokäyrältä?

5.3 Ehdollinen suorittaminen

Ehdollinen suorittaminen on tärkeä osa monipuolisten funktioiden rakentamista. Ehdollinen suorittaminen on muotoa

```
if( ehto )
      lausekeryhmä1
else
      lausekeryhmä2
```

Tästä on paljon hyötyä kun rakennamme omia funktioita, ja haluamme sisällyttää päätöksentekoa jota tehdään vasta ajon aikana. Lyhyt esimerkki:

```
> kumpi<-function(a) {
+   if( a < 0.5) {
+     print("Alle puoli")
+   }
+   else {
+     print("Yli puoli")
+   }
+ }
> kumpi(0.2)
[1] "Alle puoli"
```

Hieman isompi esimerkki, jossa funktio piirtää laskutoimituksen kuvan vain jos käyttäjä niin haluaa:

```
> f<-function(x, kuva=FALSE)
+ {
+   z<-exp(x)
+   if(kuva==TRUE) {
+     hist(z, main="Exp-muunnos")
+   }
```

```
+   else {
+     print("Ei piirretä")
+   }
+   cbind(ka=mean(x), sd=sd(x))
+ }
> f(lukuja)
[1] "Ei piirretä"
              ka        sd
[1,] 0.002277102 0.9739944

> f(lukuja, kuva=TRUE)
              ka        sd
[1,] 0.002277102 0.9739944
```

Ehdollisen lauseen jälkeen jatketaan ryhmän suorittamista rivi kerrallaan. Huomaa sisentäminen: Jokainen lausekeryhmä-taso sisennetään hieman edellistä enemmän jotta luettavuus paranee.

Harjoituksia 5.3.0:

1. Lisää harjoituksessa 5.2.0 tekemääsi funktioon parametri `p=FALSE` ja lisää testi: Jos `p` on tosi, piirretään histogrammi, muutoin tulostetaan viestiä että kuvaa ei piirretty. Testaa toimivuutta.
2. Voit jättää `else`-sanan ja sitä lausekeryhmän väliin jos sille ei ole tarvetta. Poista ne omasta funktiostasi ja testaa toimivuutta.

5.4 Silmukat

`if`-lauseen rinnalla toinen tärkeä suorituksen järjestykseen vaikuttava rakenne on *silmukka*. Silmukkaa käytetään kun halutaan toistaa tietty lauseke tai lausekeryhmä useaan otteeseen. Yleisin silmukka, `for`-silmukka, on muotoa

```
for(iteraattori in joukko)
  lausekeryhmä
```

joka menee seuraavasti: `iteraattori` käy läpi joukon `joukko`-arvot, ja jokaisella arvolla suoritetaan `lausekeryhmä`. Lähes aina on niin että `lausekeryhmä` hyödyntää jollain tapaa `iteraattori`-muuttujan arvoa. Esimerkiksi vektorin alkioiden summan laskeminen, nyt suoraan `R`-ympäristöön eikä funktion sisään (toki toimii myös funktiossa):

```
> lukuja<-runif(10)
> s<-0
```

```
> n<-length(lukuja)
> joukko<-1:n
> for(i in joukko) {
+   s<-s+lukuja[i]       # i käy läpi 1,2,...,n
+ }
> s
[1] 4.650943
```

Tai lyhyemmin

```
> s<-0
> for(luku in lukuja) { s<-s+luku }
```

Silmukan läpikäyntijoukko voi olla myös monimutkaisempi joukko, vaikkapa lista:

```
> data(morley)
> speed.l<-split(morley$Speed, morley$Expt)
> meanvec<-c()
> for(speeds1 in speed.l) {
+   meanvec<-c(meanvec, mean(speeds1))
+ }
```

Tässä siis `speeds1` käy läpi listan `morley.l` alkiot (jotka ovat vektoreita), ja joka kierros `speeds1` annetaan `mean`-funktiolle parametriksi, ja sen tulos lisätään aluksi tyhjän tulosvektorin loppuun.

Kotitehtäviä 5:

1. Aikaisemmin rakensimme kuvaajan jossa tarkastelimme lineaarista riippuvuutta (osan 4.1.1. lopussa). Tehdään lausekeryhmästä nyt funktio, joka
   - Ottaa parametreiksi kaksi vektoria nimeltä `x, y`
   - Sovittaa lineaarisen mallin $y = \alpha + \beta x + \epsilon$, (eli `fit1<-lm(y ~ x)`),
   - jakaa kuva-alueen kahtia,
   - piirtää aineistosta hajontakuvion,
   - lisää sovitetun regressiosuoran (`abline`),
   - piirtää residuaalien histogrammin, ja
   - palauttaa sovituksen yhteenvedon (`summary`).

2. Kokeile funktion toimivuutta `cars` aineistolla, eli kutsu funktiotasi siten että `x=cars$speed` ja `y=cars$dist`.
3. Kokeile funktiota vielä R:n sisäiseen aineistoon `women`, jossa muuttujat `women$weight, women$height` (katso `help(women)`).
4. Lisää funktioosi ainakin yksi parametri joka muuttaa funktion toimintaa `if`-lausekkeella. Esimerkiksi otsikon lisäys pyynnöstä tai värien käyttö, tai lisää välitulostuksia siitä mitä funktion ajon aikana tapahtuu (`print`-funktio on hyvä tekstin tulostukseen).

---

6 Osa 6: R ja satunnaismuuttujat sekä mallinnus

Tässä osassa käymme lyhyesti läpi R:n todennäköisyyslaskento-ominaisuuksia sekä hieman R:n mallikieltä. Lisää asiasta voi lukea esimerkiksi kirjastosta löytyvästä kirjasta Dalgaard, P: "Introductory statistics with R".

6.1 Satunnaisluvut ja satunnaismuuttujien jakaumat

R:ssä on sisäänrakennettuna tietoa lukuisista eri reaaliarvoisista todennäköisyysjakaumista. Mm. seuraavat jakaumat ovat heti käytössä: `beta, binom, chisq, exp, f, gamma, geom, hyper, logis, nbinom, norm, pois, t, unif`. Jokaisella jakaumalla on omat parametrinsä (keskiarvo, shape, df, yms.) joista enemmän manuaalissa kyseisen jakauman kohdalta.

Kaikista jakaumista on saatavilla seuraavat funktiot, esimerkkinä normaalijakauma jonka lyhenne on R:ssä `norm`:

- `rnorm(n=1)`: Satunnaislukujen poiminta,

```
> x<-rnorm(n=7, m=15, sd=3)
> x
[1] 16.13057 12.85171 17.58609 20.93636 13.20365 14.46995 15.22900
```

- `dnorm(x=0)`: Tiheysfunktion arvo kohdassa `x`,
- `pnorm(q=0)`: Kertymäfunktio , $P(X < x)$
- `qnorm(q=0)`: Kvantiilifunktio, palauttaa x s.e. $q = P(X < x)$

Siis selkeä kaava: Alussa `r,d,p,q` määrää tyypin, loput määrää jakauman. Siten esimerkiksi χ^2-jakauman kertymäfunktion arvo kohdassa 5, kun vapausasteita on 4, saadaan

```
> pchisq(5, df=4)
[1] 0.7127025
```

Harjoituksia 6.1.0:

1. Simuloi 50 Gamma-jakautunutta satunnaislukua parametrein (3,7). Piirrä histogrammi, ja lisää kuvaan Gamma(3,7) tiheysfunktio (esim. `curve`-funktiolla).

6.2 Tunnusluvut

Lukuarvoisen vektorin perustunnukset saadaan helposti: Keskiarvo, mediaani, varianssi, keskihajonta

```
> mean(x)
> median(x)
> var(x)
> sd(x)
```

q-kvantiilit, eli arvot x siten että $q = P(X < x)$

```
> quantile(x, probs=c(0.33, 0.5))
```

Tämä funktio on osanan esimerkiksi lukuvektorin `summary`-funktiossa:

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12.85   13.84   15.23   15.77   16.86   20.94
```

6.3 Satunnaisotanta

Satunnaisotanta onnistuu funktiolla `sample`:

```
> x<-1:12
> sample(x, size=10)
 [1]  3  7 12  4  6 10 11  9  5  1
```

Vakiona otanta on takaisinsijoittamatta, jonka saa muutettua

```
> sample(x, 10, replace=TRUE)
 [1] 5 9 5 3 5 1 5 9 6 3
```

jolloin samat alkiot voivat toistua. Indeksoinnin avulla tällä saadaan esimerkiksi satunnaisotos havainnoista. Tehdään 20 kokoinen satunnaisotos morley-havainnoista:

```
> data(morley)
> valit<-sample(1:length(morley$Speed), 20)
> morley[valit, ]
```

6.4 R:n tilastolliset mallit

R:ssä mallien kirjoittamiseen on oma syntaksinsa, ja tavallisimmin malleja sovitetaan funktioilla `lm` (linear model) ja `glm` (generalised linear model). Lineaarinen malli $y = a + bx + error$ kirjoitetaan R:ssä `y~x`, "y:tä selittää x". Vakiotermi a on automaattisesti mukana. Jos haluamme että vakiotermiä ei tule mukaan, kirjoitetaan $y \sim 0 + x$. Mallin sovitus, eli parametrien a, b sekä *error*-termin varianssin estimointi hoituu `lm`-funktiolla:

```
> data(cars)
> fit1<-lm(dist ~ speed, data=cars)
> fit2<-lm(dist ~ 0+speed, data=cars)
> summary(fit1)
```

`summary`-funktio käyttää hyväkseen `lm`-funktion tulosta ja laskee mm. mallin sopivuuteen liittyviä asioita käyttäen tuloksen arvoja. Sovitteesta saa myös `plot`-funktiolla kuvia. Lisäksi joskus voi tarvita esim. `anova`-, `residuals`- (jäännökset) ja `coef`-funktioita (sovitetut kertoimet), jotka palauttavat mallin sovituksesta haluttuja osia.

Jos selittäviä tekijöitä on enemmän kuin yksi on malli täydessä asussaan $y = a + b_1 x + b_2 z + b_{12} xz + error$. Sen R-versio on `y~x*z`. Mikäli emme halua ristitermiä mukaan kirjoitamme vain `y~x+z`. Pelkän ristitermin tarkastelu onnistuu `y~x:z`, joten käytännössä siis `x*z` on sama kuin `x+z+x:z`. Muuttujia voi lisätä, jolloin termien lukumäärä kasvaa, ja onkin katsottava tilannekohtaisesti mitkä kaikki termit ovat olennaisia.

Näistä ja yleistettyjen lineaaristen mallien sovituksesta voi lukea lisää tarvittaessa esimerkiksi R:n sisäisestä Introduction to R -oppaasta osioista "11: Statistical models in R" tai `formula`- ja `lm`-funktion manuaalisivulta.

7 Osa 7: Pidempiä esimerkkejä R:n käytöstä

R:n pääasiallinen käyttö monelle on tilastollisten menetelmien suorittaminen. Käydään nyt läpi joitain esimerkkitilanteita, alkaen aineiston luvusta ja päättyen tulkintoihin.

7.1 Makuasia?

On tehty oluen maisteluun liittyvä koe: 8 eri olutlajia, 7 eri maistajaa, jokainen maistaja arvostelee oluen arvosanalla 1-5.

Aineistotiedostossa on vain arvosanat, olut-maistaja järjestyksessä:

```
> arvosana<-scan("http://users.jyu.fi/~tarajala/R/Aineistot/oluet.txt", skip=4)
```

Luodaan tasomuuttujat jotka kuvaavat maistajia ja oluita. Tehdään maistaja käsin

```
> maistaja<-factor( rep(1:7, 8))
> maistaja
```

Olutlajille pitää tehdä ”1, 1,..., 1, 2, 2,...2, 3,...,8, 8” tasomuuttuja. Sen voi tehdä joko

```
> olut<-factor(rep(1:8, each=7))
> olut
```

tai koska tällaista tarvitsee tehdä usein on sitä varten tehty R:n oma funktio `gl` (generate levels):

```
> olut<-gl(8, 7)
> olut
```

Yhdistetään aineisto tietokehikoksi:

```
> oluet<-data.frame(olut, maistaja, arvosana)
> head(oluet)   # sama kuin oluet[1:6,]
```

Piirretään aineistosta kuvaajat:

```
> par(mfrow=c(1,2))
> boxplot( arvosana~maistaja, data=oluet, xlab="maistaja")
> boxplot( arvosana~olut, data=oluet, xlab="olut")
```

Mitä nähdään? Ainakin se että maistajat 5-7 eivät anna täysiä pisteitä millekkään oluelle. Toisaalta oluissakin näyttäisi olevan eroa: Olut 2 saa max. 3 pistettä.

Ettei jäätäisi kuva-asteelle tehdään tilastotiedettä: Tutkitaan ensin onko oluiden tuloksissa merkittävästi eroa:

```
> fit1<-lm(arvosana ~ olut, data=oluet)
```

summary-funktio palauttaa vaikka vallan mitä tuloksia sovituksesta. Käytetään nyt kuitenkin anova-funktiota, joka palauttaa vain ANOVA-taulukon eli jäännösneliösummat:

```
> anova(fit1)
```

Näyttäisi olevan lievästi eroa.

Jos olutta 2 ei oteta huomioon

```
> anova(lm(arvosana~olut, data=oluet, subset=(olut!=2)))
```

ei merkitsevää eroa enää löydy. Voimme myös tarkastella arvosanakeskiarvoja per olut pareittaisella t-testillä:

```
> pairwise.t.test(arvosana, olut)
```

Näemme että olut 2 tosiaan eroaa eniten muista.

Tutkitaan vielä onko maistajien välillä eroa:

```
> fit3<-lm(arvosana~maistaja, data=oluet)
> anova(fit3)
```

Miten käy jos huomioidaan molemmat tasot, eli soluina on maistaja×olut:

```
> fit4<-lm(arvosana~maistaja+olut, data=oluet)
> anova(fit4)
```

Samat päätelmät pätevät: Maistajien välillä ei löydy eroa, mutta oluissa löytyy.

7.2 Bootstrap eli epävarmuuden tarkastelu keinotekoisella otannalla

Meillä on aineistossa erään kurssin oppilaiden demopistehyvitykset sekä tenttitulos. Kiinnostaa tietää miten nämä kaksi asiaa korreloivat keskenään.

```
> x<-read.table("http://users.jyu.fi/~tarajala/R/Aineistot/demotulos.txt",
+               header=T)
> plot(x)
```

Korrelaation voi laskea suoraan:

```
> cor(x$tul, x$dem)
[1] 0.5028381
```

Mutta miten arvioida korrelaatiokertoimen tarkkuutta? Eli kuinka luotettava on 0.5, heiluuko se todellisuudessa -0.3 ja 0.6 välillä ja satuimme vaan osumaan tälläkertaa numeroon 0.5? Tutkitaan tätä bootstrap-menetelmällä.

Eli tarkoitus on simuloida havaintoja, laskea korrelaatiokerroin, ja toistaa monta kertaa jotta nähdään miten sen jakauma käyttäytyy. Luodaan ensin bootstrap-funktio:

```
> cor.boot<-function(x, y, iter=1000){
+   n<-length(x)
+   corvec<-numeric(iter)               # laitetaan kertoimet tähän talteen
+   for(i in 1:iter){
+     ord<-sample(1:n, n, replace=TRUE) # valitaan "otos" populaatiosta
+     xnew<-x[ord]                      # uusi aineisto
+     ynew<-y[ord]                      # uusi aineisto
+     corvec[i]<-cor(xnew, ynew)
+   }
+   corvec
+ }
```

Eli poimitaan uusi otos havaintojen joukosta takaisin sijoittaen, lasketaan korrelaatio, toistetaan. Katsotaan mitä tulee:

```
> tulos<-cor.boot(x$tul, x$dem, iter=1000)
> hist(tulos)
```

Tämä on nyt korrelaatiokertoimen jakauman estimaatti. Voimme laskea sille luottamusvälin

```
> quantile(tulos, prob=c(0.025, 0.5, 0.975))
```

Korrelaatio on siis positiivista: Kannattaa tehdä demot niin todennäköisesti pärjää paremmin tentissä.

7.3 Rapulajien välisestä kokoerosta

Aineistossa on suomalaisten järvi- ja jokirapujen pituuksia ja painoja, ja kiinnosta tietää miten paino riippuu pituudesta. Aineisto käyttöön:

```
> ravut<-read.table("http://users.jyu.fi/~tarajala/R/Aineistot/ravut.txt",
+        header=TRUE, skip=3)
```

Ravut on koodattu: 1=Jokirapu (JR), 2 = Täplärapu (TR). Muutetaan koodit R:n faktoreiksi:

```
> rapuf<-factor(ravut$num)
> levels(rapuf)<-c("JR","TR")
> ravut$laji<-rapuf
```

Voimme nyt kirjoittamisen vähentämiseksi ”kiinnittää” (`attach`) tietokehikko `ravut`. Tämä tarkoittaa sitä että tietokehikon sisäiset muuttujat, eli `ravut$pit, ravut$paino, ravut$num, ravut$laji` tulevat käyttöön muuttujanimillä `pit, paino, num, laji`. Tämä on käytännön helpotusta, mutta on tärkeä muistaa ”irrottaa” (`detach`) tietokehikko käytön jälkeen.

```
> attach(ravut)
```

Piirretään kuva, josta selvitetään riippuvuuden luonnetta:

```
> plot(pit, paino, col=num, pch=num)
> legend("topleft", c("JR","TR"), col=1:2, pch=1:2)
```

Riippuvuus ei näytä lineaariselta, vaan enemmänkin eksponentiaalisesti kasvavalta. Tehdään log-muunnos:

```
> lpaino<-log(paino)
> plot(pit, lpaino, col=num, pch=num)
```

Vieläkään ei näytä suoralta. Koetetaan vielä log-muunnos pituuteen:

```
> lpit<-log(pit)
> plot(lpit, lpaino, col=num, pch=num)
```

Nyt näyttää lineaariselta. Log-log-muunnos muutti aineiston lineaariseksi. Tähän siis sopisi malli

$$\log(paino) = a + b\log(pit)$$

eli ilman logaritmejä paino noudattaa potenssimallia

$$paino = e^a pit^b = d \cdot pit^b$$

Voidaanko lajit yhdistää? : Tutkitaan voiko lajien havainnot yhdistää eli riittääkö yksi d ja yksi b kuvaamaan molempia aineistoja tarpeeksi hyvin. Tämä helpottaa rapukannan arviointia, sillä tarvitsee käyttää vain yhtä lajista riippumatonta mittatikkua.

Kannattaa ajatella seuraavasti: Luodaan havainnoille tasomuuttuja z. Se saa arvon $z = 1$ kun yksilö on jokirapu ja arvon $z = 0$ kun yksilö on täplärapu. Täysi mallimme on log-log-asteikolla seuraava

$$\log(paino) = a + b\log(pit) + gz + hz\log(pit),$$

missä kerroin g kertoo lajien tasoerosta, ja h selittää lajien kasvueroa. Tämän näkee siitä että kun $z = 0$ on malli perus potenssimalli, mutta kun $z = 1$ saadaan

$$log(paino) = (a + g) + (b + h)\log(pit).$$

Nyt kiinnostaa onko $g = 0$ ja $h = 0$. Jos näin on, voimme unohtaa lajimerkin kokonaan ja yhdistää aineistot: Rapulajit eivät eroa mallin mielessä toisistaan.

Sovitetaan malli koko aineistoon:

```
> fit1both<-lm( lpaino ~ lpit*laji )
> summary(fit1both)
```

Nähtävästi koko malli ei tuo mitään uutta. Pienennetään mallia, eli koetetaan ilman ristitermiä h:

```
> fit2both<-lm( lpaino ~ lpit+laji )
> summary(fit2both)
```

Lajimuuttuja on siis merkitsevä vakiotason määrittämisessä. Siksipä emme voi yhdistää aineistoja, vaan meidän on huomioitava laji.

Summa summarum: Pituuden ja painon suhde noudattaa mallia

$$pituus = e^{-12.547} \cdot paino^{3.494} \cdot 0.969^z$$

missä jokiravulle $z = 1$ ja täpläravulle $z = 0$.

Ja muistetaan vielä irrottaa tietokehikko, `detach(ravut)`.

7.4 Valinnainen osio : Numeerisia pulmia

7.4.1 Newtonin algoritmi funktion nollakohdan selvittämiseksi

Sir Isaac Newtonin muinoin kehittelemä numeerinen funktion nollakohdan ratkaiseva algoritmi on vielä tänäkin päivänä voimissaan. Yksinkertaisuudessaan se menee funktiolle $f(x)$ näin:

1. Arvataan että nollakohta on x_0.
2. Korjataan arvausta: $x_1 = x_0 - f(x_0)/f'(x_0)$, missä f' on funktion f ensimmäinen derivaatta.
3. Mikäli muutos $|x_1 - x_0|$ on hyvin pieni, olemme lähellä oikeaa vastausta. Voidaan lopettaa.
4. Muuten sijoita $x_0 = x_1$ ja palaa kohtaa 2.

Miten tätä voi käyttää? No esimerkiksi seuraavasti: Arvioidaan lukua

$$a = \sqrt[7]{29}$$

Esitetään se funktion nollakohtana: $f(x) = x^7 - 29$ on funktio jolle $f(a) = 0$. Ratkaistaan a numeerisesti:

```
> newton<-function(f, fprime, x0){
+   loop<-TRUE
+   while(loop){
+     x1<- x0 - f(x0)/fprime(x0)
+     if(abs(x1-x0)<10^(-5)) loop<-F     # lopetusehto
+     x0<-x1
+   }
+   x0
+ }
```

Parametreinä annetaan funktio, funktion derivaatta sekä alkuarvaus. Kirjoitetaan funktio ja sen derivaatta:

```
> f<-function(x) x^7-29
> fprime<-function(x) 6*x^6
```

Ratkaistaan ongelmamme näillä eväillä:

```
> newton(f, fprime, 1)
```

7.4.2 Funktion numeerinen maksimointi

Funktion numeerinen maksimointi suoriutuu varsin kivuttomasti R:ssä. Ratkaistaan logaritmisen uskottavuusfunktion maksimikohta, eli tehdään suurimman uskottavuuden päättelyä parametrien selvittämiseksi havainnoista: Luodaan $n = 64$ mittainen aineisto jakaumasta Gamma(2,4):

```
> z<-rgamma(64, shape=2, rate=4)
```

Koetetaan ratkaista parametrit suurimman uskottavuuden päättelyllä: Oletetaan että havainnot ovat riippumattomia, ja tulevasta Gamma(a,b) mallista (siis jos vain saisimme havainnot). Uskottavuusfunktio on tällöin

$$L = \prod \frac{b^a}{\Gamma(a)} x_i^{a-1} e^{-bx_i}$$

ja sen logaritminen muoto

$$l = n \cdot a \cdot \log(b) - n \log(\Gamma(a)) + (a-1) \sum \log(x_i) - b \sum (x_i)$$

Ratkaistaan sen maksimikohta numeerisesti käyttäen R:n valmista optimoijaa. Se minimimoi annetun funktion, joten kirjoitetaan $-l$ funktioksi:

```
> negl<-function(a,b){
+   n<-length(z)
+   -(n*a*log(b)-n*log(gamma(a))+(a-1)*sum(log(z))-b*sum(z))
+ }
```

tämä syötetään optimoijalle, alkuarvauksemme kera:

```
> tulos<-nlm(negl, p=c(1,2))
```

Jaahas. Optimoija haluaa funktio jolle annetaan vektoriparametri. Tehdään apufunktio:

```
> negl.v<-function(ab) negl(ab[1], ab[2])
```

ja koetetaan uudestaan

```
> tulos<-nlm(negl.v, p=c(1,2))
> tulos$estimate
[1] 2.645368 5.336827
```

Vertaa näitä analyyttisiin suurimman uskottavuuden ratkaisuihin:

```
> cbind(a=mean(z)^2/var(z) , b=mean(z)/var(z))
            a       b
[1,] 2.733005 5.51363
```