Data-analyysi R-ohjelmistolla, kevät 2015

Viikko 5

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

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

1 Lista

1.1 Listan luominen

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

Esimerkki 1.

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

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

Esimerkki 2.

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

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

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

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

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

Listasta voidaan valita alilistoja samalla tavalla kuin vektorista alkioita.

Esimerkki 3.

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

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

[[2]]
[1] 3.14159

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

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

Esimerkki 4.

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

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

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

Esimerkki 5.

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

```
> class(kolmas)
[1] "integer"
> kolmas[1]
[1] 1
```

1.2 Listan komponenttien nimeäminen

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

Esimerkki 6.

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

$opiskelijanro
[1] 1468267

$aloitusvuosi
[1] 2012

$kurssit
[1] 57045 57046 57703
```

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

Esimerkki 7.

```
> opiskelija$nimi
[1] "Aslak"
> opiskelija[[1]]
[1] "Aslak"
> opiskelija$kurssit[1]
[1] 57045
```

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

Esimerkki 8.

```
> opiskelija$kurssit <- c(opiskelija$kurssit, 57798)
> opiskelija$paa_aine <- "tilastotiede"
> opiskelija
$nimi
[1] "Aslak"
```

```
$opiskelijanro
[1] 1468267

$aloitusvuosi
[1] 2012

$kurssit
[1] 57045 57046 57703 57798

$paa_aine
[1] "tilastotiede"
```

Poistetaan `paa_aine`-komponentti sijoittamalla siihen tyhjäarvo `NULL`.

Esimerkki 9.

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

$opiskelijanro
[1] 1102544

$aloitusvuosi
[1] 2012

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

1.3 Listan läpikäyminen

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

Esimerkki 10.

```
> numerolista <- list(1:10, c(2,4,6), 15)
> numerolista
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] 2 4 6

[[3]]
[1] 15
```

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

[[2]]
[1] 12

[[3]]
[1] 15
```

Tarkka lukija saattoi tässä vaiheessa huomata, että edellisen esimerkin olisi voinut yhtä hyvin tehdä aiemmin esitellyllä `sapply()`-funktiolla. Tämä on totta, itse asiassa `sapply()`, eli *s*implified *apply* on vain ns. wrapper `lapply()`:lle, eli se tekee saman asian kuin `lapply()`, eli soveltaa toisena argumenttina annettua funktiota jokaiseen ensimmäisena argumenttina annetun listan komponenttiin[2]. Erona on vain se, että `sapply()` muuttaa tuloksen automaattisesti vektoriksi tai matriisiksi jos se on mahdollista.

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

Esimerkki 11.

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

[[2]]
[1]  4 16 36

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

2 Lisää funktioista

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

Esimerkki 12.

```
> toinen_potenssi <- function(x) {
    return(x^2)
  }
>
```

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

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

2.1 Funktion palautusarvon kirjoittaminen näkyviin

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

Esimerkki 13.

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

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

2.2 Koodiblokit ja koodin sisennys

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

Esimerkki 14.

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

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

2.3 replicate

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

Esimerkki 15.

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

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

Esimerkki 16.

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

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

Esimerkki 17.

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

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

3 Yhden otoksen T-testi

Tutkitaan jälleen R:n mukana tulevaa klassista Iris-esimerkkiaineistoa, ja erityisesti eri kurjenmiekkalajien terälehtien pituuksia. Testataan eroaako kaunokurjenmiekkojen (*iris setosa*) terälehtien pituus 5:stä, eli testataan t-testillä[3]. nollahypoteesia $H_0 : \mu = 5$ kaksisuuntaista vastahypoteesia $H_1 : \mu \neq 5$ vastaan, missä μ on kaunokurjenmiekkojen terälehtien pituuden odotusarvo. Yhden otoksen t-testi tehdään luonnollisesti jo viime viikolla luottamusvälien laskemiseen käytetyllä `t.test`-funktiolla. Ensimmäiseksi argumentiksi annetaan vektori, jossa on testattava aineisto, ja argumentti `mu` määrittelee nollahypoteesiarvon μ_0. Argumentti `alternative` määrittelee, onko vastahypoteesi kaksi- vai yksisuuntainen, ja jos se on yksisuuntainen, niin kumpaan suuntaan. Sitä ei kuitenkaan tarvitse tässä määritellä, sillä oletusarvo on kaksisuuntainen.

Esimerkki 18.

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

        One Sample t-test

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

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

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

pituuden keskiarvolle. Viimeisellä rivillä on otoskeskiarvo 5.006, joka on tosiaan on huomattavan lähellä viittä.

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

Esimerkki 19.

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

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

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

Esimerkki 20.

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

Funktion `t.test` palautusarvo on siis lista [4]! Voimme poimia sen komponentteja niiden nimen avulla, kuten teimme jo viime viikolla luotamusvälien tapauksessa.

Esimerkki 21.

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

[4] Itse asiassa `t.test`:in palautusarvo on luokan `htest` olio, kuten tulosteen viimeiseltä riviltä nähdään. R:n ns. S3-tyypin (Toinen R:n luokkatyyppi on uudempi S4, joka muistuttaa määrittelyltään enemmän muiden olio-orientoituneiden kielten, kuten Javan, luokkia. Suurin osa R:n luokista on kuitenkin S3-tyypin luokkia.) olio on yksinkertaisesti lista, johon on liitetty luokan nimeävä attribuutti, tässä tapauksessa `htest`.

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

Esimerkki 22.

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

4 Riippumattomuustesti

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

Esimerkki 23. Tutustutaan nyt fiktiivisen leipomon toimintaan ja sen tuotekehityksessä ilmenneeseen ongelmaan. Leipomossa on nimittäin huomattu, että taikinan kohoaminen on ajoittain huteraa ja epäilee, ettei käytetty hiiva (Hiiva 1) ole parasta A-laatua. Tästä syystä leipuri päättää koittaa kilpailijan vastaavaa tuotetta (Hiiva 2) ja kirjaa testituloksensa ylös neljästä sadasta taikinaerästä.

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

Tutkitaan seuraavaksi, olisiko leipurilla tilastollisia perusteita vaihtaa hiivan toimittajaa. Valitaan nollahypoteesi H_0 : "Hiivan valinta ei vaikuta taikinan kohoamiseen"ja lasketaan testisuure funktion `chisq.test()`-funktion avulla:

```
> O <- matrix(c(120,80,173,27), ncol=2)
> chisq.test(O, corr=F)

        Pearson's Chi-squared test

data:  O
X-squared = 35.8394, df = 1, p-value = 2.143e-09
```

Tässä argumentti `corr=F` ilmoittaa, ettei haluta käyttää jatkuvuuskorjausta. Testin tuloksena saadaan hyvin pieni p-arvo, jonka voidaan tulkita viittaavan siihen, että taikinan kohoaminen todella riippuisi hiivan valinnasta.

Sama voitaisiin hyvinkin laskea käyttämättä funktiota `chisq.test`. Muistetaan, että testisuureen kaava on

$$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}},$$

missä O_{ij} ovat havaitut frekvenssit ja E_{ij} niitä vastaavat odotetut arvot. E_{ij} voidaan laskea havaituista frekvensseistä tulona rivin i ja sarakkeen j summista ja jakamalla sitten koko havaintomäärällä, eli

$$E_{ij} = n_{+j}n_{i+}/n.$$

Tästä voidaan laskea testisuure ja sen p-arvo käyttäen jakaumafunktiota `pchisq()`, jolle annetaan argumentteina testisuureen asymptoottisen jakauman vapausasteet ja tieto siitä, että halutaan käyttää yläkvantiileja:

```
> # Lasketaan odotetut frekvenssit
> E <- (apply(O, 1, sum) %*% t(apply(O,2,sum))) / sum(O)
> # Lasketaan testisuure
> X <- sum((O - E)^2 / E)
> X
[1] 35.83937
> pchisq(X, df=1, lower=F)
[1] 2.142742e-09
```

5 Tehtäviä

5.1 Tehtäviä

1. Puuttuvien arvojen huomioon ottaminen summamuuttujaa laskettaessa.

   a) Lataa Pyöräilybarometri-aineisto ja valitse osa-aineisto muuttujista `Aq6 - Aq15` samalla tavalla kuin viime viikon tehtävässä 1. Muuta ”En osaa sanoa”-vastaukset puuttuviksi.

   b) Laske kuinka monta puuttuvaa vastausta kullakin vastaajalla on yhteensä kysymyksiin `Aq6 - Aq15` (esimerkiksi jos vastaaja on jättänyt vastaamatta kolmeen näistä kymmennestä kysymyksestä, muuttujan arvoksi tulee tämän vastaajan kohdalla 3) ja tallenna tulos vektoriin `puuttuvat`

   c) Tulosta muuttujan `puuttuvat` frekvenssitaulu ja laske kuinka moni vastaaja on jättänyt vastaamatta viiteen tai useampaan kysymykseen. Vihje: yksiulotteinen taulu on nimetty vektori, joten voit summata sen alkioita. Taulun kaikkien alkioiden summan saa laskettua komennolla `sum(taulu[1:length(taulu)])`, jos taulu on tallennettu muuttujaan `taulu`. Ole kuitenkin tarkkana indeksien kanssa!

   d) Luo summamuuttuja `tyytyvaisyys` samalla tavalla kuin viime viikon tehtävässä yksi. Käytä tällä kertaa argumenttia `na.rm=TRUE`, jolloin muuttujan arvo lasketaan, jos vastaaja on vastannut yhteenkin kysymyksistä. Tallenna summamuuttuja sarakkeeksi alkuperäiseen taulukkoon `pb`. Laske kuinka monta puuttuvaa arvoa summamuuttuja saa.

   e) Vastaajien, jotka ovat jättäneet vastaamatta useaan kysymyksistä joista summamuuttuja lasketaan, vastauksia ei yleensä haluta ottaa huomioon summamuuttujaa laskettaessa. Muuta siis summamuuttujan `tyytyvaisyys` arvo puuttuvaksi, jos vastaaja on jättänyt vastaamatta viiteen tai useampaan kysymykseen (eli muuttujan `puuttuvat`

arvo on 5 tai suurempi. Laske kuinka monta puuttuvaa arvoa summamuuttuja `tyytyvaisyys` nyt saa, ja tarkasta että tulos on sama kuin c-kohdassa laskettu arvo.

2. Testataan poikkeaako edellisessä tehtävässä laskettu `tyytyvaisyys-muuttujan` keskiarvo tilastollisesti merkitsevästi 2.2:sta merkitsevyystasolla 0.05. Testataan siis t-testillä nollahypoteesia $H_0 : \mu = 2.2$ kaksisuuntaista vastahypoteesiä $H_1 : \mu \neq 2.2$ vastaan.

   a) Lasketaan ensin testisuure käsin. Laske t-testisuureen arvo kaavasta

   $$t = \frac{\overline{y} - \mu_0}{s/\sqrt{n}},$$

   missä $\overline{y}$ on muuttujan `tyytyvaisyys` otoskeskiarvo, s sen otoskeskihajonta ja n sen otoskoko (huom. n on todellinen otoskoko, eli puuttuvia arvoja ei lasketa mukaan) ja μ_0 nollahypoteesin mukainen parametrin arvo.

   b) Laske testin p-arvo kaavasta

   $$p = P(|T| \geq |t|) = 2(1 - F_{n-1}(|t|)),$$

   missä F_{n-1} on t:n jakauman vapausasteella $n - 1$ kertymäfunktio, joka saadaan R:ssa funktiolla `pt`.

   c) Tarkasta laskemasi t-testisuureen arvo ja p-arvo funktiolla `t.test`. Tulkitse testin tulosta.

3. Tarkastellaan edellisen tehtävän tulosta visuaalisesti.

   a) Piirrä t:n jakauman vapausasteella $n - 1$ (missä n on edellisen tehtävän `tyytvaisyys`-muuttujan otoskoko) jakauman tiheysfunktion kuvaaja välillä $[-4, 4]$. T:n jakauman tiheysfunktion arvot saadaan R:ssä funktiolla `dt`.

   b) Liitä kuvaan pystysuorat viivat kohtiin $t_{n-1}(0.025)$ ja $t_{n-1}(0.975)$, eli t:n jakauman vapausasteella $n - 1$ 2.5 prosentin ja 97.5:n prosentin kvantiilien kohtiin (vihje: funktio `qt`). Tee viivoista haluamasi väriset.

   c) Liitä kuvaan pystysuora viiva (erivärinen kuin edelliset viivat) edellisen tehtävän t-testisuureen arvon kohtaan.

   d) Miten tulkitset kuvaa: missä ovat merkitsevyystason 0.05 t-testin kriittiset alueet, ja sijoittuuko t-testisuureen arvo kriittiselle alueelle?

4. Jatkoa edelliseen tehtävään.

   a) Simuloi sata t-testisuureen arvoa, eli 100:n kokoinen otos t:n jakaumasta vapausasteella $n - 1$, missä n on edelleen `tyytyvaisyys`-muuttujan otoskoko.

   b) Lisää edellisen tehtävän kuvaan jälleen uudella värillä pystysuorat viivat a-kohdassa simuloitujen arvojen kohtiin.

c) Arvioi silmämääräisesti kuinka moni viivoista, eli simuloiduista t-testisuureen arvoista, on testin kriittisellä alueella? Entä onko yksikään simuloiduista t-testisuureen arvoista yhtä kaukana jakauman hännässä kuin havaittu (tehtävässä kolme laskettu) t-testisuureen arvo?

d) Tarkasta edellisen kohdan arvio laskemalla kuinka moni a-kohdassa simuloiduista testisuureen arvoista on testin kriittisellä alueella, eli pienempää kuin $t_{n-1}(0.975)$ tai suurempaa kuin $t_{n-1}(0.025)$ (huomaa että kvantiileille on tässä käytetty JTP:n merkintöjä, jotka ovat "väärinpäin": piste $t_{n-1}(0.975)$ on t:n jakauman 2.5 prosentin kvantiili ja piste $t_{n-1}(0.025)$ 97.5 prosentin kvantiili)?

5. Lisää simulointia

a) Simuloi tällä kertaa sata n:n kokoista (n on `tyytyvaisyys`-muuttujan otoskoko) otosta normaalijakaumasta, jonka keskiarvo on 2.2, keskihajonta `tyytyvaisyys`-muuttujan keskihajonta. Vihje: `replicate`.

b) Laske jokaiselle otokselle muuttuja z kaavasta

$$z = \frac{\overline{y} - \mu}{\sigma/\sqrt{n}},$$

missä $\overline{y}$ on kunkin otoksen otoskeskiarvo, $\mu = 2.2$, σ on `tyytyvaisyys`-muuttujan otoskeskihajonta, jota käytettiin simuloitaessa jakauman parametrina ja n on otoksien koko (eli `tyytyvaisyys`-muuttujan otoskoko). Tallenna tulokset vektoriin `z`. Vihje: `apply` (Voit myös tehdä a- ja b-kohdan saman `replicate`-lauseen sisällä viime viikon tehtävän 6 a) tyyliin, jos uskallat).

c) Piirrä standardinormaalijakauman tiheysfunktion kuvaaja välillä $[-4, 4]$.

d) Liitä kuvaan pystysuorat viivat kohtiin $z_{0.975}$ ja $z_{0.025}$, eli standardinormaalijakauman 2.5 prosentin ja 97.5 prosentin kvantiilien kohdalle mieleiselläsi värillä.

e) Lisää kuvaan pystysuorat viivat vektorin `z` arvojen kohtiin jollain toisella värillä.

f) Vertaa lopputulosta edellisen tehtävän kuvaan, ja selitä mitä havaitset (tätä ei tarvitse tietää, mutta mieti asiaa ja selitä näkemyksesi)?

6. Lista funktion palautusarvona.

a) Kirjoita funktio joka laskee kaksisuuntaisen yhden otoksen t-testin t-testisuureen arvon ja p-arvon (tehtävän 2 kaavojen avulla, käyttämättä t.test-funktiota). Funktiolle annetaan parametreina tarkasteltavan muuutujan otoskeskiarvo $\overline{y}$, otoskeskihajonta s, otoskoko n ja nollahypoteesin mukainen parametrin arvo μ_0.

Funktion tulee palauttaa lista. jonka ensimmäinen komponentti on `t_testisuure`, joka sisältää t-testisuureen arvon, toinen komponentti `p_arvo`, joka sisältää testin p-arvon ja kolmas komponentti on `luottamusvali_95`, joka on vektori joka sisältää keskiarvon 95 prosentin luottamusvälin ala- ja ylärajan (laske tämäkin käyttämättä `t.test`-funktiota, kaava löytyy viime viikon harjoitusten tehtävästä kolme).

b) Testaa että funktiosi toimii oikein syöttämällä siihen tehtävän 2 arvot ja tarkastamalla että saat samat tulokset.

7. JTP:n harjoitusten 4 tehtävä 4 a). Testaa t-testillä edellisessä tehtävässä luomaasi funktiota käyttäen nollahypoteesia $H_0 : \mu = 3$ kaksisuuntaista vastahypoteesia $H_1 : \mu \neq 3$ vastaan merkitsevyystasolla 0.05, kun

   a)
   $$n = 10, \quad \overline{y} = 3.952, \quad s^2 = 1.981$$

   b)
   $$n = 15, \quad \overline{y} = 3.411, \quad s^2 = 1.321,$$

   ja tulkitse testien tulokset sanallisesti. Huomaa, että tehtävässä on annettu otosvarianssit s^2, eikä otoskeskihajontoja s.

8. Tutkitaan esimerkin 23 tapausta. Muuttuisiko päättelyn tulos, jos Hiivalle 2 olisikin havaittu 140 kohonnutta ja 60 kohoamatonta taikinaa?

9. Esimerkin 23 fiktiivinen leipomo päättää tehdä uuden aluevaltauksen ja värjätä taikinansa punaiseksi, vihreäksi ja siniseksi. Jokaista leipälajia valmistetaan päivittäin, värikohtaiset määrät saattavat vaihdella. Värikohtainen leivottujen ja myytyjen leipien lukumäärä kirjataan ylös.

   Tutki riippumattomuustestin avulla nollahypoteesia, jonka mukaan leivän väri ei vaikuta sen myyntiin. Viikon tuotanto ja myynti ovat taulukoituna alla. *Vihje: Pohdi ensin mitä tässä taulukossa on ja mitä pitää vielä laskea. Ehkä kannattaa taulukoida väreittäin viikon aikana myydyt ja myymättömät leivät.*

	Punainen		Vihreä		Sininen	
	Leivottu	Myyty	Leivottu	Myyty	Leivottu	Myyty
Maanantai	15	12	23	20	4	4
Tiistai	24	10	13	8	16	10
Keskiviikko	20	19	12	11	8	7
Torstai	10	10	14	12	22	18
Perjantai	25	22	24	20	24	22
Yhteensä	**94**	**73**	**86**	**71**	**74**	**61**

10. Tarkastellaan seuraavaksi Pyöräilybarometria. Muuta muuttujan `Aq3` ”En osaa sanoa”-vastaukset puuttyviksi arvoiksi. Tutki sen jälkeen riippumattomuustestin avulla riippuuko tyytyväisyys Helsinkiin pyöräilykaupunkina (sarake Aq3)

    a) Sukupuolesta (sarake AA)?

    b) Ikäryhmästä (sarake Aikalk1)?

    c) Asuinalueesta (sarake Aalue2)?

    d) Tulotasosta (sarake AH)?