Õppematerjal on valminud
Maateaduste ja ökoloogia
doktorikooli raames
EL struktuuritoetuse toel

Euroopa Liit
Euroopa Sotsiaalfond

Eesti tuleviku heaks

Antud kursusel vaatleme statistikatarkvara R kasutamist Windowsi kasutaja vaatenurgast. Samas ei erine teiste operatsioonisüsteemide korral kasutamine märkimisväärselt. Näidete tegemiseks on kasutatud Ri versiooni 2.12.

Me ei vaatle tarkvara installeerimist, ent sellest saab ülevaate näiteks punktide 2.1 ja 2.2 abil lehelt
`http://cran.r-project.org/bin/windows/base/rw-FAQ.html`

Windowsi mittekasutajad saavad abi lehelt
`http://cran.r-project.org/doc/manuals/R-admin.html`

Eestikeelne juhend Windowsi kasutajale on näiteks aadressil
`http://dspace.utlib.ee/dspace/html/10062/14885/installeerimine.html`

Kuivõrd R on vabavara siis on igati mõistlik see *oma* arvutisse paigaldada. Samas tasub mainimist, et R töötab väga edukalt ka mälupulgalt.

Tere, R!

Peale programmi käivitamist avaneb lisaks põhiaknale (RGui) selle sees ka konsooliaken (R Console), kus muuhulgas on ära toodud mõned kasulikud käsud, mida konsooliaknasse kirjutada võib[1]. Näiteks

```
> citation()
```

annab teada, kuidas Ri kasutamist artiklis tsiteerida.

Alustame esmalt lihtsate näidetega, et hiljem nende põhjal selgitada natuke Ri üldisi põhimõtteid, mille mõistmine tagab tõrgeteta töö ka keerukamates olukordades.

```
> 3 + 5 * 9 ** 0.5
[1] 18
> (3 + 5) * 9 ** 0.5
[1] 24
```

[1] Käsku asub R täitma peale klahvi 'Enter' vajutamist. Kui käsk oli vigane siis kuvatakse veateade. Kui käsk oli poolik, siis liigutakse konsooliaknas uuele reale ja sümbol `>` rea alguses asendub sümboliga `+`. Meil on siis võimalik käsk lõpetada või vajutada 'Esc' eelneva tühistamiseks.

Tore! R oskab arvutada ning tunneb tehete järjekorda. Sulgude abil saame tähistada need tehted, mis tuleb teha esmalt. Muidugi saame ka tulemusi salvestada

```
> a = 3 + 5 * 9 ** 0.5
> a + 2
[1] 20
```

Tekitasime muutuja `a`. Asi, mida siin märkida: muutujale `a` omistatakse väärtus kõige lõpuks ehk siis kui kõik muu on juba tehtud. Pärast liitsime muutujale `a` arvu `2` juurde. Aga R vaatab lihtsalt järgi `a` väärtuse, muutuja ise jääb muutmata:

```
> a
[1] 18
```

Nüüd on arusaadav ka, miks võime kirjutada

```
> a = a + 2
```

Omistamine sooritatakse viimasena. Esmalt asub R interpreteerima osa `a + 2`. Selleks vaadatakse, mis peitub muutuja `a` nime taga. See on arv `18` ja see võetakse teadmiseks. Tehe oli aga `a + 2` ja seega R arvutab `18 + 2` ja saab tulemuseks `20`. Et käsk oli `a = a + 2` siis saab see tulemus muutuja `a` uueks väärtuseks. Tõepoolest

```
> a
[1] 20
```

Tuleme korraks veel tagasi eelmise näite juurde. Kui kirjutasime `a = a + 2` siis R ei kuvanud muutuja `a` uut väärtust. Antud näite korral ei ole see ilmselt kuigi oluline (sest me teame `a` uut väärtust peast), ent see ei pea alati nii olema. Sel juhul võime anda käsu

```
> (a = a + 2)
[1] 22
```

Kuivõrd nüüd oli juba `a = 20`, siis muidugi on `a` uus väärtus `22`, ent nüüd see väärtus lisaks ka kuvati.

Vaatleme järgnevalt funktsioonide klassikalisemat kuju Ris. Tegelikult oleme seda juba näinud käsu `citation()` juures. Vaatleme järgmist näidet

```
> b = c(1, 5, 9)
> mean(b)
[1] 5
```

Mängus oli lausa kaks funktsiooni. Esmalt `c()`[2], mis tekitab argumentidest vektori ja pärast `mean()`, mis leiab argumendi aritmeetilise keskmise. Mis põhiline, nägime ära Ri funktsiooni rakendamise üldkuju:

[2]Selline imelik nimi tuleb sõnast *combine* – funktsioon kombineerib argumendid kokku. Kuivõrd seda funktsiooni läheb vaja erakordselt sageli siis on funktsiooni nime lühidus tõeline voorus.

```
funktsiooninimi(väärtus1, väärtus2, ...)
```

Nii oli funktsioonile `c()` antud argumente kokku kolm ja funktsioonile `mean()` vaid üks. Nüüd on arusaadav ka käsk `citation()` – me rakendasime sellenimelist funktsiooni, ent ei andnud lihtsalt ühtegi argumenti[3].

Kuidas on lood siis liitmisega? Seal me ju ei kirjutanud `sum(3, 5)`. Aga see viimane tegelikult töötab![4]

```
> sum(3, 5)
[1] 8
```

Asi mida meelde jätta on, et tegelikkuses on Ris kõik käsud funktsiooni(de) rakendamised. Nii näiteks omistamist interpreteerib R kui

```
> '='(d, 4)
```

ja isegi lihtsalt objekti välja kutsumine toob kaasa selle, et sellele objektile rakendatakse funktsiooni `print()`.

Ülesanded

I Olgu meil teada huntide rünnakute arv kuues piirkonnas. Seejuures eristatud on marutõbiste huntide rünnakud ja huntide kisklusrünnakud.

Kubermang	Marutõbi	Kisklus
Mogiljev	39	27
Peterburg	6	18
Podolsk	27	17
Smolensk	9	3
Vilnius	23	0
Vladimirsk	17	7

Tekita üks vektor marutõbiste rünnakute arvudest ja teine kisklusünnakute arvudest. Leia keskmine marutõbiste rünnakute arv kubermangu kohta ja keskmine kisklusrünnakute arv kubermangu kohta. Samuti leia rünnakute koguarv kubermangude kaupa ning ka rünnakute koguarv kõigi kubermangude peale kokku.[5]

II Ehk oleks mugavam toimetada kui meil oleksid kõik andmed üheskoos. Seda on lihtne saavutada funktsiooni `matrix()` abiga. Esimese argumendina tuleb ette anda arvud, mis maatriksisse paigutada, teisena maatriksi ridade arv, kolmandana veergude arv. Vaata, mida teeb `mean()` selle maatriksiga.

[3] Ja me ei saa kirjutada lihtsalt `citation` ilma sulgudeta, sest see võiks olla hoopis mõne muutuja nimi. Kui me seda siiski teeme ja sellise nimega muutujat ei eksisteeri siis kuvatakse meile funktsiooni sisu (kood), ent see ei pruugi olla alati informatiivne.

[4] Lihtsalt `3+5` on mugavam kirjutada ja see kood tõlgitakse automaatselt Ri jaoks ümber. Tõsi küll, tegelikult summeerimist operaatori `+` abil ei tõlgita funktsiooniks `sum()` vaid hoopis funktsiooniks `'+'()`

[5] Pane tähele, et võimalusel toimetab R suuremate üksustega (nagu näiteks vektor) elementhaaval. See võimaldab väga mahukaid operatsioone väga lihtsalt teostada.

Kui me kasutasime funktsiooni `matrix()` maatriksi loomiseks siis paigutas R andmed nõnda, et esimesena täideti maatriksi esimene veerg, siis teine veerg. Täpsemalt – maatriks täideti veergepidi. Samas võib ette tulla olukord, kui maatriks oleks vaja täita ridupidi[6]. Osutub, et piisab anda maatriksi loomisel veel neljanda argumendina väärtus `TRUE`. See toob meid funktsiooni argumendi vaikeväärtuste juurde. Enamikel funktsioonidel on ühel või mitmel argumendil olemas vaikeväärtus. See tähendab, et kui me selle funktsiooni vastavat argumenti ära ei määra siis kasutatakse vaikeväärtust. Niisiis on funktsiooni `matrix()` neljanda argumendi, mis määrab, kas maatriks täidetakse elementidega ridupidi vaikeväärtus `FALSE`, mis tähendabki, et kui me selle argumendi väärtust ei muuda (ette ei anna) siis täidetakse loodav maatriks arvudega veergepidi.

Praeguseks peab juba olema tekkinud küsimus: „kuidas ma tean, millised on funktsiooni argumendid, mis järjekorras need on ning millised on vaikeväärtused?“. Selleks on funktsioonidel olemas abifailid[7], mida saab välja kutsuda funktsiooniga `help()`, ent seda funktsiooni võimaldab kasutada ka lühikuju:

?“funktsiooninimi“

Anname siis käsu

```
> ?"matrix"
```

ja tutvume selle funktsiooni abifailiga. Abifail algab pealkirjaga. Abifaili esimene osa kannab nime *Description*. Seal on toodud funktsiooni lühikirjeldus. Väga sage on, et mitu funktsiooni on sarnase temaatika alusel kirjeldatud ühises abifailis: ka siin on lisaks ära kirjeldatud funktsioonid `as.matrix()` ja `is.matrix()`. Järgneb osa nimega *Usage*, kus on üles loetletud funktsiooni argumendid ja nende vaikeväärtused. Näeme, et funktsioonil `matrix()` on tegelikult lausa viis argumenti ja neil kõigil on ka vaikeväärtused. Mis aga veelgi olulisem – pea igal argumendil on ka nimi! See tähendab, et me saame üldjuhul funktsiooni välja kutsuda ka kujul

funktsiooninimi(argument1=väärtus1, argument2=väärtus2, ...)

Asja iva on, et selline väljakutse lubab argumentide järjekorda muuta. Nii näiteks

```
> matrix(c(1, 2, 5, 6, 11, 15), 3, 2, TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    5    6
[3,]   11   15

> matrix(c(1, 2, 5, 6, 11, 15), byrow=TRUE, nrow=3, ncol=2)
```

[6] Muidugi võib siin kasutada kavalust – teha alguses maatriks äravahetatud mõõtmetega, täita see veergepidi ja siis lõpuks transponeerida

[7] Ri abifailid, mis on HTML kujul, avanevad veebibrauseris. Vahel võib see funktsionaalsus töötada puudulikult. Näiteks esimese käsu peale avaneb vaid veebibrauser, ent mitte leht abifailiga. Sel juhul on abi käsu kordamisest. Vahel võib probleeme põhjustada ka brauseri proxy serveri määrang. HTML formaadis abifailid on väga mugavad, sest võimaldavad hüperlinkide abil kiirelt erinevate abifailide vahel liikuda. Tegelikult on abifailidel olemas ka pealeht, mis avaneb käsuga `help.start()`

```
     [,1] [,2]
[1,]    1    2
[2,]    5    6
[3,]   11   15
```

Järgneb osa abifailis kannab nime *Arguments* ja siin seletatakse lahti, millised on funktsiooni argumentide võimalikud väärtused. Järgnevad täpsustusi sisaldav osa *Details*, viited osas *References*, sarnast tulemit võimaldavad funktsioonid osas *See Also* ning lõpuks näited osas *Examples*. Viimased võime ise konsooliaknasse kopeerida, ent võime ka kirjutada

```
> example("matrix")
```

mispeale kõik näited sinna kopeeritakse[8]. Üks asi mida veel märkida: argumentidele võime väärtused anda vahetult, muutujate kaudu või funktsiooni väljundite kaudu, ent argumentide nimed on fikseeritud. Hea on ka teada, et funktsioon `args()` võimaldab kuvada funktsiooni argumendid koos vaikeväärtustega. Kui me funktsiooni üldist tööpõhimõtet teame on see tunduvalt kiirem ja mugavam meeldetuletus kui funktsiooni abifaili avamine.

Järgmine küsimus. Praegu me vaatasime meile teadaoleva nimega funktsiooni abifaili. Aga kuidas leida üles see funktsioon, mis teeb mida vaja? Ühe lahenduse pakub funktsioon `apropos()`, millele saame ette anda sõne (teksti jutumärkide vahel), mis peaks meie arvates funktsiooni nimes esinema. Seepeale kuvatakse kõik seda sõnet sisaldavad funktsioonid. Näiteks hii-ruut testi otsides võime proovida

```
> apropos("chi")
 [1] ".Machine"    "ChickWeight" "chickwts"    "chisq.test"  "dchisq"
 [6] "machine"     "Machine"     "pchisq"      "qchisq"      "rchisq"
```

Loetelust on nüüd lihtne välja valida funktsioon `chisq.test()`. Siinkohal on ka hea tähelepanu juhtida kandilistes sulgudes olevatele numbritele, mida R väljundi juures kuvab. Tegu on lihtsalt elemendi järjekorranumbriga. Sageli on väljundis vaid üks element ja seetõttu ka vaid `[1]` esimese (ja ainsa) rea ees.

Teine võimalus on funktsioon `help.search()`, mis otsib vastet abifailide pealkirjadest. Funktsioonil on ka lühikuju `??` ehk näiteks regressiooniteemalisi funktsioone on mõtet otsida käsuga

```
> ??"regression"
```

Ri konsooli on sisse ehitatud ka nn iselõpetaja (*autocomplete*), mis aitab leida sobiva nimega funktsiooni. Kui meil on juba osa funktsiooninimest kirjutatud ning sellest piisab, et määrata üheselt, millist funktsiooni me silmas peame, siis vajutades klahvi 'tab' lõpetab R funktsiooninime kirjutamise meie eest. Kui kirjutatud funktsiooninime osast ei piisa, et funktsioon üheselt ära määrata, siis Ri poolset tegevust klahvi 'tab' vajutamisel ei järgne, ent kui vajutame seda klahvi veel teist korda siis kuvatakse kõigi selliselt algavate funktsioonide nimed. Kui funktsiooninimi ja esimene sulg on juba kirjas (või ka mõni argument, selle väärtus ja järgnev koma), siis kuvab kahekordne klahvi 'tab' vajutus antud funktsiooni võimalikud argumendid.

[8] Lisaks abifailides leiduvatele näidetele on mõned mahukamad näited vormistatud demodena. Vaata lähemalt `?"demo"`

Ülesanded

III Leia funktsioon, mis võimaldab leida kisklusrünnakute ja marutõbiste rünnakute vahelist korrelatsiooni. Leia see korrelatsioon kasutades esmalt lineaarset (Pearsoni) korrelatsioonikordajat ja siis Spearmani astakkorrelatsioonikordajat. Leia vahend, et testida saadud tulemuste statistilist olulisust. Kas saame väita, et kisklusrünnakute ja marutõbiste rünnakute arvukuse vahel on seos?

IV Kas oskad ära tõestada, et rünnakute proportsioonid (kisklus vs marutõbi) kubermangudes ei ole ühesugused?

Me ei pea (tegelikult ei ole isegi soovitav) kirjutama koodi otse konsooliaknasse. Võime selle kirjutada näiteks mõnesse tekstieditori ja sealt hiljem Ri konsooliaknasse kopeerida. Ril (tegelikult Ri Windowsi versioonil) on tekstieditor ka kaasas. See nn skriptiaken avaneb valides menüüst *File > New Script*. Sisuliselt on tegu tavalise tekstieditoriga, suuremaks erisuseks on ainult selle integreeritus R-i konsooliaknaga. Nimelt saame skriptiaknas käsu, mis asub kursoriga samal real (või ka mitu väljavalitud rida) otse R-i konsooliaknasse täitmiseks saata vajutades lihtsalt 'ctrl+r'. Skriptiaknas olevate käskude salvestamiseks saab valida *File > Save* (skriptiaken peab olema aktiivne), mispeale vaikimisi tekitatakse .R lõpuga fail. Selline fail on tegelikult siiski tavaline nn *plain text* fail, millisel tavaliselt laiendiks .txt. Seega saab seda tüüpi faile edukalt mistahes tekstieditoriga avada ja muuta.

Siin tuleb ära mainida ka nn kommentaarisümbol `#`, millele (samal real) järgnevat teksti ignoreeritakse. Nii saab oma programmidele märkusi juurde kirjutada ilma, et R neist segadusse satuks[9]. Oma kirjutatud koodi kommenteerimine on praktikas väga oluline. Ris on sage mitme erineva funktsiooni järjest rakendamine (nii nagu maatriksit moodustades rakendasime esmalt funktsiooni `c()` ja siis selle tulemil funktsiooni `matrix()`), mis on väga mugav. Kui me oleme oma mõtetega koodi kirjutamise juures siis on kerge kirjutada koodi, kus ühel real on kasutatud näiteks viit funktsiooni. Mõni kuu hiljem sellele koodireale peale vaadates aga tekib hämmeldus. Sellest aitavad üle saada selgitavad märkused kommentaaride näol.

Võib tunduda, et töö Riga on kohmakas – iga väiksemgi asi eeldab üht või mitut rida koodi. Samas võiks hiirega juhitavas nn osuta-ja-vajuta töökeskkonnas neid operatsioone mugavamalt ja intuitiivsemalt sooritada. See on siiski vaid esmamulje. See, et meil jääb kogu analüüsi igast sammust „märk maha“ on tegelikult asendamatu funktsionaalsus. Kujutage ette, et kolleeg saadab teile andmestiku, mida põhjalikult analüüsite. Hiljem selgub, et andmestikus olid osad andmeväljad vigased. Nüüd on tarvis kogu analüüs uuesti teha. Ühel juhul tähendab see uut osutamist ja vajutamist. Teisel juhul aga lihtsalt valmis koodi uut läbijooksutamist[10].

Lisaks veel kaks asja, mis ehk praeguseks juba avastatud. Esiteks, R eristab suuri ja väikesi tähti. Teiseks, kui soovime mõnd eelnevat käsku (muutmata või veidi muudetud kujul) korrata siis saame „sisestatud käskude ajaloos liikuda“ klahvidega 'nool üles' ja 'nool alla' [11].

[9] Lisaks võib olla huvitav teada, et samale reale võib mahutada ka mitu käsku. Need tuleb lihtsalt omavahel sümboliga `;` eraldada.

[10] Ilmselt ei ole kuigi tore ka see kui näiteks teeme andmetes mingeid muudatusi ja näiteks mõne kuu pärast sama probleemi juurde tagasi tulles ei mäleta me enam, mida muutsime. Ühesõnaga dokumenteerimine on kriitilise tähtsusega. See vaatenurk on tuntud kui *Reproducible Research*.

[11] Ehkki üldjuhul väga mugav, põhjustab see funktsionaalsus mõnikord ka meelehärmi – käsurea peal saame edasi tagasi liikuda klahvidega 'nool vasakule' ja 'nool paremale' ning neid kasutades on kerge kogemata vajutada 'nool üles' või 'nool alla', mispeale senikirjutatu käsurealt kaob. See on samuti hea põhjus, miks peaksime kasutama eraldi tekstieditori, mitte kirjutama koodi otse konsooliaknasse.

Paketid

Praeguseks oleme õppinud, kuidas leida üles meile tööks vajalikud funktsioonid. Funktsioonide abifaile vaadates võis ehk silma jääda, et enne pealkirja kuvati ka funktsiooni nimi ning lisaks veel loogilistes sulgudes näiteks *base* (funktsiooni `matrix()` puhul) või *stats* (funktsiooni `cor()` puhul). Need on pakettide (*packages*) nimed. R on üles ehitatud nõnda, et sarnase temaatikaga funktsioonid on koondatud ühte paketti. Üksjagu pakette paigaldatakse arvutisse kohe Ri installeerimisel. Mõned neist omakorda on Ri käivitamisel kohe kasutatavad (ehk mõned paketid on juba tööle laetud). Kui pakett veel tööle laetud ei ole siis saab seda teha käsuga kujul

library("paketinimi")

Seepeale laetakse pakett tööle ning lisaks laetakse tööle ka kõik need paketid, mille funktsioonidest antud paketi funktsioonid sõltuvad[12]. Ri taaskäivitamisel tuleb vajalikud paketid (mida vaikimisi tööle ei laeta) uuesti tööle laadida.

Saamaks ülevaadet näiteks paketis `stats` sisalduvatest funktsioonidest (tegelikult objektidest) võime anda käsu

```
> library(help="stats")
```

mispeale kuvatakse esmalt info paketi kohta (autor, versioon jms) ja seejärel kõigi paketis leiduvate funktsioonide nimed ja pealkirjad[13].

Käsu

```
> .packages(all.available=TRUE)
```

järel kuvatakse nimekiri kõigist arvutis leiduvatest pakettidest. Mitte kõiki neid ei laeta Ri käivitades tööle. Tööle laetud pakettide nimistu avaneb käsu

```
> (.packages())
[1] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
[7] "base"
```

abil.

Miks mitte laadida juba Ri käivitades tööle suurem hulk pakette? Lisaks ressursside kasutamise efektiivsusele on kasutaja jaoks oluline ka see, et paketid võivad sisaldada sama nimega funktsioone[14] (sest mugav on funktsiooninimed hoida lühikesed). See tähendab aga, et käsk mingit funktsiooni rakendada on ilma taustinfota ambiguoosne – ei ole selge kumba paketi vastava nimega funktsiooni mõeldakse. Ris on see lahendatud otsingujärjekorra abil – R kasutab vaikimisi seda funktsiooni, mille pakett on otsingujärjekorras eespool. Otsingujärjekorda näeme käsu

[12] Kuivõrd paketid koosnevad funktsioonidest siis on eriti graafikalahenduste puhul küllalt tavaline, et antud paketi funktsioonid kasutavad juba mõne teise paketi funktsioonide tööd

[13] Vahel on paketiga kaasas ka üks või mitu paketi tööd tutvustavat kirjutist, mida kutsutakse vinjettideks. Vinjettide loetelu annab käsk `vignette()`. Sama funktsioon (argumendiks vinjeti nimi) võimaldab ka vinjettide lugemist.

[14] R annab selle kohta ka hoiatusteate kui mõni funktsioon „kinni kaetakse“

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

abil. Esimesel kohal on üldkeskkond (*Global Environment*), kuhu tekivad vaikimisi meie poolt loodud objektid. Ri baaspakett on alati viimasel kohal. Uue paketi laadimisel saame funktsiooni `library()` rakendades öelda, mitmendale kohale otsingujärjekorras uus pakett asetada tuleks. Siiski saab ka „kinni kaetud“ funktsioonile üldjuhul ligi, andes käsu stiilis paketinimi:::funktsiooninimi[15]

Praeguseks on selge, et mitte kõik paketid ei tule kaasa Ri installeerimisega. Pigem vastupidi. *The Comprehensive R Archive Network* ehk CRAN, kust Ri kasutaja kerge vaevaga pakette oma arvutisse installeerida saab[16], hõlmab 2010. aasta lõpu seisuga enam kui 2500 paketti[17]. Pakettide installeerimine käib käsuga

```
install.packages("paketinimi")
```

kusjuures tegelikult võib nimistus olla ka mitu paketti (siis tuleb need funktsiooni `c()` abil kokku panna). Installeerimine on (erinevalt pakettide töölelaadimisest) ühekordne protsess. Uue Ri versiooni installeerimisel saab küll installeeritud paketid alamkataloogist *library*[18] ümber tõsta, ent üldjuhul on ka pakette vaja aeg-ajalt uuendada. Seda saab teha, andes käsu

```
> update.packages(ask=FALSE)
```

Ülesanded

V Tööks segamudelitega on Ris kaks peamist paketti *nlme* ja *lme4*. Neist esimene on automaatselt kaasas ja tuleb vaid tööle laadida, teine tuleb esmalt ka installeerida. Lae mõlemad paketid tööle. Mis juhtus?

VI Milline on uute pakettide laadimise järgselt otsingujärjekord? Kuidas saame kasutada funktsiooni `AIC()` paketist *stats*?

Et kuidas nüüd lõpuks Rist välja pääseda? Windowsis on üleval nurgas rist, kust ikka programme sulgeda saab. Seepeale küsib R, kas peaks salvestama tehtud töö (tekitatud objektid). Aga stiilne on lahkuda funktsiooni abil. Anname käsu

```
> q("no")
```

ja R paneb ise asjad (ilma midagi salvestamata) kokku.

[15] Aga see ei pruugi alati nii lihtne olla – kui paketil puudub *namespace* siis see lähenemine ei tööta. Üldiselt selline probleem siiski väga sage ei ole.

[16] Sarnaselt Ri installeerimisega eeldame internetiühenduse olemasolu. Loomulikult on ka pakettide installeerimine võimalik näiteks mõnelt kohalikult andmekandjalt, ent seda siinkohal ei kirjelda.

[17] Pakettide nimekiri on aadressil `http://cran.r-project.org/web/packages/`

[18] Näiteks `C:\Program Files\R\R-2.12.0\library`

Objektid

Eelnevalt rääkisime, et iga Ri käsk on oma olemuselt funktsiooni rakendamine. Ent iga funktsioon, muutuja, konstant jms Ris on objekt. Seega

iga Ri käsk on funktsiooni(de) rakendamine objekti(de)le [1]

Üldkeskkonnas alguses objekte ei ole, küll aga tööle laetud pakettides (peamiselt funktsioonid). Nendest saame ülevaate funktsiooni `objects()` abil. Sellele funktsioonile võime argumendiks ette anda numbri ning otsingujärjekorras antud numbrile vastava paketi[2] objektid kuvatakse.

Enda loodud objekte saame eemaldada funktsiooni `rm()` abil.

Ülesanded

I Kuva kõik Ri baaspaketis leiduvad objektid. Vaata funktsiooni `objects()` abifailist, kuidas kuvada see objektide nimistu (ühe käsuga) siis kui me ei tea küll paketi numbrit otsingujärjekorras, ent teame paketi nime.

II Võrdle paketis *graphics* leiduvaid objekte funktsiooni `objects()` abil saadava loendi ja funktsiooni `library()` abil tekitatava loendi abil. Kas märkad mõnd erinevust? Kuidas saaksime näha funktsiooni `plot.factor()` koodi?

Igal Ri objektil on klass, mida saab vaadata funktsiooni `class()` abil[3]. Objekti klassi põhjal määratakse, milliseid funktsioone saab objektile rakendada[4]. Järgnevalt vaatleme olulisemaid objekte Ris.

Objektid: vektor

Teame juba, et vektoreid luuakse funktsiooni `c()` abil. Sellleks, et vektoreid eristada näitab funktsioon `class()` nende puhul elementide väärtusetüüpi. Võimalikud variandid on

```
> class(c(1, 2)) #arvuline vektor
[1] "numeric"
> class(1i) #kompleksarvuline vektor
[1] "complex"
> class(TRUE) #tõeväärtusvektor
[1] "logical"
> class("roomaja") #sõnevektor
```

[1] Öelda, et „iga käsk on objekti(de) rakendamine objekti(de)le“ ei oleks väga informatiivne
[2] Hiljem näeme, et otsingujärjekorda võib tegelikult lisada ka andmemaatrikseid
[3] See ei ole 100% tõene, tegelikult on veel olemas funktsioon `typeof()`, ent nii on seda teemat kõige lihtsam käsitleda
[4] Täpsemalt geneeriliste funktsioonide juures

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

Kuivõrd vektoris on elementidel järjekorranumbrid siis on lihtne neid välja kutsuda. Ris toimub see kandiliste sulgude abil. Siin on aga päris mitu erinevat võimalust:

```
> maru = c(39, 6, 27, 9, 23, 17)
> kisk = c(27, 18, 17, 3, 0, 7)
> maru[5] #üks element järjekorranumbri alusel
[1] 23
> maru[c(3, 1)] #mitu elementi järjekorranumbrite alusel
[1] 27 39
> maru[-c(3, 4)] #milliseid elemente me ei soovi
[1] 39  6 23 17
> maru[kisk > 5] #tõeväärtuste vektori abil
[1] 39  6 27 17
```

Vektori elementidele võib lisaks järjekorranumbritele (mis on olemas vaikimisi) panna ka nimed. Seda võimaldab funktsioon `names()`. Sellisel juhul saab elemente ka nimepidi välja kutsuda.

```
> kubermang = c("Mogiljev", "Peterburg", "Podolsk", "Smolensk",
          "Vilnius", "Vladimirsk")
> names(maru) = kubermang
> maru
  Mogiljev  Peterburg    Podolsk   Smolensk    Vilnius Vladimirsk
        39          6         27          9         23         17
> maru["Podolsk"]
Podolsk
     27
```

Objektid: list ja tuletatud objektid

List on väga üldine objekt, mis võib endas sisaldada kõiki muud tüüpi objekte. Seega võib listis esimesel positsioonil olla näiteks tõeväärtusvektor ja teisel positsioonil funktsioon. Võime ette kujutada, et kuitahes keerulise objekti võime kirja panna listina. Küllalt sageli ongi funktsioonide väljundiks list. Listi loomine toimub funktsiooni `list()` abil.

```
> tulemus = cor.test(maru, kisk, method="spearman")
> analyys = list(maru, kisk, tulemus)
> class(analyys)
[1] "list"
```

Listist elementide välja kutsumine toimub topelt kandiliste sulgude abil. Ent kui elementidel on nimi siis töötab ka järjekorranumbri asemel nime andmine. Lisaks on kasutatav sümbol `$`. Isegi kui objekti klass ei ole list vaid midagi spetsiifilisemat on need kolm viisi reeglina elementide väljakutsumiseks kasutatavad. Listi elementidele nime andmine käib samuti funktsiooni `names()` abil[5].

[5] Järgmiste näidete juures pane veel tähele ka, et arvu täis- ja murdosa eraldamiseks on Ris kasutusel punkt. Koma eraldab teadupärast funktsiooni argumente.

```
> analyys[[2]] #listil analyys ei ole elementidel nimesid
[1] 27 18 17  3  0  7
> class(tulemus) #ei ole list
[1] "htest"
> tulemus #funktsioon print meile objekti struktuuri ei paljasta
        Spearman's rank correlation rho

data:  maru and kisk
S = 26, p-value = 0.6583
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.2571429
> tulemus[[3]]
[1] 0.6583333
> tulemus[["p.value"]]
[1] 0.6583333
> tulemus$p.value
[1] 0.6583333
> analyys[[3]][[3]] #mis juhtus?
[1] 0.6583333
```

Ülesanded

III Millise tulemuse ja miks annab käsk `class(names(maru))`?

IV Kuidas saame objekti `maru` teise elemendi nimeks seada „St. Peterburg“?

V R ei keela meil omada üldkeskkonnas objekte, mis ühtivad mõne funktsiooni nimega töölelaetud paketis. Seega kui esmalt defineerime `mean=mean(maru)` siis `class(mean)` annab meile tulemuseks `numeric`. Miks ei võiks kirjutada näiteks `class(mean())`? Kas mäletad veel, kuidas nüüd funktsioonile `mean()` ligi saada?

VI Listi näite juures teadsin, et objekti `tulemus` kolmandal positsioonil asub element nimega *p.value*. Kuidas võisin selle teada saada?

Objektid: maatriks

Nagu teame saab maatrikseid luua funktsiooni `matrix()` abil. Maatriksite korral on elementide väljakutsumiseks eriline viis, mis laieneb ka näiteks massiividele ja andmemaatriksitele. Kasutusel on ikka kandilised sulud (nagu vektorite korral), ent rea ja veerunumbrid on komaga eraldatud. Esmalt näitame seega ära, milliseid ridu me soovime/ei soovi ja seejärel teeme sama veergudega. Välja tühjaks jätmine tähendab kõige valimist[6].

[6]Tegelikult võib väita ka, et selline viis on olemas juba vektoritel – ka seal saame kandilised sulud tühjaks jättes kogu vektori

```
> hundid = matrix(c(maru, kisk), nrow=6)
> class(hundid)
[1] "matrix"
> hundid[1, ]
[1] 39 27
> hundid[c(4, 3), 2]
[1]  3 17
```

Maatriksi ridadele ja veergudele nimesid andes peame kasutama funktsiooni `dimnames()`. Argumendina tuleb ette anda list. Esimesest elemendist saavad reanimed, teisest veerunimed (kui listil on ainult üks element siis saavad sellest reanimed). Kui omakorda listi elementidel on nimed, siis saavad ka maatriksi dimensioonid endale nimed. Kui listil on nimed olemas siis on ka nimedega väljakutsed võimalikud.

```
> dimnames(hundid) = list(kubermang, c("kisklus", "marutaud"))
> hundid["Mogiljev", "kisklus"]
[1] 39
```

Objektid: massiiv

Nii nagu maatriks on vektori laiend kahele dimensioonile nii on massiiv (*array*) viis esitada sama tüüpi andmeid kolme või enama dimensiooni korral. Näitena võime ette kujutada kolmemõõtmelist sagedustabelit. Muidu on massiiv analoogiline maatriksile. Loomine toimub funktsiooni `array()` abil.

Objektid: faktor

Faktor (*factor*) on oma olemuselt küll nagu sõnevektor, ent muuhulgas võivad faktori tasemed omada järjestust (saame luua järjestustunnuse). Faktori loomiseks kasutatakse funktsiooni `factor()`. Järjestatud tasemetega faktori korral tuleb seada argument `order=TRUE`. Vaikimisi tekitatakse faktori tasemed tähestiku järjekorras. Kui me seda ei soovi, siis peame funktsiooni `levels()` abil ise järjekorra määrama.

```
> ohvrid = c("naine", "hobune", "mees", "mees", "mees", "lammas", "hobune")
> ohver = factor(ohvrid)
> ohver #tasemed on tähestiku järjekorras
[1] naine  hobune mees   mees   mees   lammas hobune
Levels: hobune lammas mees naine
> levels(ohver) #võimalikud faktori tasemete väärtused
[1] "hobune" "lammas" "mees"   "naine"
> ohver = factor(ohvrid, levels=c("mees", "naine", "hobune", "lammas"))
> ohver #andsime ise tasemete järjekorra
[1] naine  hobune mees   mees   mees   lammas hobune
Levels: mees naine hobune lammas
```

Faktorist mingit alamosa eraldades võib võimalike tasemete info olla liiga lai. Sellisel juhul faktorit „uuesti luues“ see info korrigeeritakse (muidugi ainult juhul kui uuesti loodud faktor ka objektina salvestatakse).

```
> ohver2 = ohver[c(1,3)]
> levels(ohver2) #ülearused tasemed
[1] "mees"   "naine"  "hobune" "lammas"
> factor(ohver2) #kaotame ülearused tasemed ära
[1] naine mees
Levels: mees naine
> ohvrivanus = factor(c("laps", "täiskasvanu", "vanur",
                  "laps", "laps", "täiskasvanu"),ordered=T)
> ohvrivanus #tasemed on nüüd järjestatud
[1] laps        täiskasvanu vanur       laps        laps        täiskasvanu
Levels: laps < täiskasvanu < vanur
```

Vahel võib esineda olukord, kus meil on faktori tasemed numbrilisel (või ka näiteks tähelisel) kujul ja me soovime nendele tähistele alles sisu anda. Selleks on funktsioonil `factor()` argument `labels`. Vähim (tähestikus eespool asuv) väärtus saab esimese väärtuse, järgmine teise ja nii edasi.

```
> factor(c(1, 1, 1, 2, 2, 1, 2), labels=c("mees", "naine"))
[1] mees  mees  mees  naine naine mees  naine
Levels: mees naine
```

Objektid: andmemaatriks

Andmemaatriks (*data frame*) on maatriksi üldistus ja listi kitsendus. See tähendab, et veerud (andmemaatriksi elemendid) võivad olla erinevat tüüpi vektorid (siia hulka loeme ka faktorid) aga need peavad kõik olema ühesuguse elementide arvuga. Seega on andmemaatriks tavaline viis andmeid esitada: ridades on vaatlused ja veergudes tunnused. Andmemaatriksi loob funktsioon `data.frame()`, kusjuures sõnevektorid muudetakse vaikimisi faktoriteks.

```
> riik=c("Valgevene", "Vene", "Vene", "Vene", "Leedu", "Vene")
> hundiandmed = data.frame(maru, kisk, riik)
> hundiandmed #kuna 'maru' omas nimesid siis kanti need edasi
           maru kisk      riik
Mogiljev     39   27 Valgevene
Peterburg     6   18      Vene
Podolsk      27   17      Vene
Smolensk      9    3      Vene
Vilnius      23    0     Leedu
Vladimirsk   17    7      Vene
```

Andmemaatriksi puhul on andmete väljakutsumiseks kasutatavad viisid nii maatriksitelt (kaks indeksit kandilistes sulgudes) kui ka listidelt (topelt kandilised sulud või `$` veeru valimiseks). Muidugi

saab kasutada ka rea- ja veerunimesid. Nimede seadmiseks või lugemiseks on kasutatavad funktsioonid `dimnames()`, `colnames()`, `rownames()`[7].

Ülesanded

VII Kuidas muuta objekti `ohver` nii, et oleks vaid kaks võimalikku väärtust: loom ja inimene?

VIII Miks tulid faktormuutuja `ohvrivanus` korral tasemed ise õiges järjekorras? Kuidas peame üldjuhul talitama järjestatud tasemetega faktori loomisel?

IX Näita vähemalt nelja erinevat viisi, kuidas valida andmemaatriksist `hundiandmed` väljaspool praegust Vene territooriumi toimunud kisklusrünnakute arvud.

Objektid: valem

Valemitega (objekt *formula*) tegeleme statistiliste analüüside juures veel põhjalikumalt, ent kuna valemisüntaksit saab kasutada Ris ka mujal (näiteks graafikute joonistamisel) siis teeme siinkohal väikese sissejuhatuse.

Valemi aluseks on sümbol `~`. Sellest vasakul paiknevad sõltuvad tunnused ja paremal seletavad tunnused. Seega kui meil on tunnus `maru`, mida me soovime kirjelda tunnuse `pindala` abil siis märgime seda

```
maru ~ pindala
```

Mitu tunnust eraldatakse omavahel sümboliga `+`. Tunnuste koosmõjusid väljendame sümboliga `:`. Sümbol `*` tähistab "peamõjud ja koosmõjud". Seega mudelid

```
maru ~ pindala + tihedus + pindala:tihedus
maru ~ pindala * tihedus
```

on samaväärsed. Üldjuhul võime valemisse sisse kirjutada ka tavalisi matemaatilisi funktsioone nagu logaritm või siinus. Sümbolid (koos tunnustega), millel on aga valemisüntaksis eritähendus tuleb kindlasti kirja panna funktsiooni `I()` abil. Nii näiteks valem

```
maru ~ I(pindala * tihedus) + tihedus
```

tähistab, et me modelleerime tunnust `maru` kahe tunnuse abil, kus esimene on saadud kahe tunnuse väärtuste korrutamisel. Kirjutades mudeli paremale poolele juurde `-1` jätame ära vabaliikme (mis pannakse mudelisse vaikimisi). Valemi loomiseks saab kasutada funktsiooni `formula()`.

Valemis esinevate tunnuste kohta me eraldi midagi ütlema ei pea. Nende klass määrab selle, kuidas neid mudelis tõlgendatakse. See tähendab, et kui `tihedus` on arvuline siis teda nii ka käsitletakse, kui aga faktor siis ka käsitletakse seda tunnust faktorina.

[7] Tegelikult on kõik need funktsioonid kasutatavad ka mistahes muu maatriksilaadse objekti korral, `dimnames()` ka üldisemal juhul

Objektiteisendus ja geneerilised funktsioonid

Sageli on võimalik objekti klassi muuta (seda protsessi nimetatakse *coercion*). Näiteks üheveerulist maatriksit on kerge vaadelda vektorina (mida see ka ju on). Kõikide seninähtud klasside jaoks on olemas funktsioonid kujul

as.klass(objekt) ja is.klass(objekt)

mis võimaldavad vastavalt muuta ja kontrollida objekti klassi. Seega näiteks käsk `as.matrix(maru)` muudab vektori `maru` maatriksiks. Muidugi on objekti klassi võimalik (mõistlikult) muuta ainult teataval juhul. Näiteks listi ei saa muuta vektoriks. Lihtsamat objekti saab alati muuta keerukamaks. Üks võimalus sellist keerukust (jämedalt) väljendada on

tõeväärtuseline<arvuline<kompleksarvuline<sõneline<list

Kui me muudame vektoris, maatriksis või massiivis mõnd elementi eeltoodud pingerea kohaselt keerukamaks, siis teisendab R keerukamaks ka kõik teised elemendid.

```
> maru[3] = "22 kuni 26"
> maru #kõik numbrid muudeti sõnedeks
    Mogiljev    Peterburg      Podolsk     Smolensk      Vilnius   Vladimirsk
        "39"          "6" "22 kuni 26"          "9"         "23"         "17"
```

Sageli võimaldavad ka Ri funktsioonid mitmesuguseid sisendeid. Nii võime funktsioonile `data.frame()` ette anda väga erinevat tüüpi objekte (näiteks maatriks, vektor, faktor). Esimese sammuna viib funktsioon kõigi nendega läbi objektiteisenduse. Selleks kasutatakse geneerilist funktsiooni `as.data.frame()`.

Geneeriliste funktsioonide põhimõtte seletamiseks vaatleme funktsiooni `print()`. Me näeme, et näiteks rakendades seda kas faktorile või andmemaatriksile saame üsna erinevad tulemused. See tuleneb asjaolust, et funktsioon `print()` on geneeriline – sõltuvalt objekti klassist teeb see erinevaid asju. Praktikas näeb geneerilise funktsiooni töö (funktsiooni `print()` näitel) välja järgmiselt:

I. Saadakse käsk funktsiooni `print()` rakendamiseks objektile

II. Vaadatakse järgi objekti klass (klassil võib olla ka mitu väärtust)

III. Otsitakse funktsiooni nimega `print.objektiklass()` (iga klassi väärtuse jaoks)

IV. Kui selline funktsioon on olemas siis seda rakendatakse objektile

V. Kui sellist funktsiooni pole siis rakendatakse objektile funktsiooni `print.default()`

VI. Kui ka sellist funktsiooni ei oleks siis väljastataks veateade

Ülesanded

X Loo üks tõeväärtusvektor ja vaata kuidas see teisendatakse numbriliseks. Rakenda (teisendamata) tõeväärtusvektorile funktsiooni `mean()` ja vaata, mis juhtub.

XI Vahel andmeid sisse lugedes juhtub, et arvuline tunnus loetakse sisse faktorina. Olgu meil näiteks `mune=factor(c(6, 2, 3))`. Meil oleks vaja see tunnus arvuliseks teisendada. Proovi, mida teeb faktoriga funktsioon `as.numeric()`. Miks nii? Mida siis teha?

Teades objekti klassi on meil kerge üles leida meetodid (funktsioonid), mis oskavad seda tüüpi objektiga mõistlikult ümber käia. Appi tuleb funktsioon `methods()`. Tegelikult saab seda funktsiooni kasutada lausa kahte moodi. Andes käsu

```
> methods(summary)
 [1] summary.aov            summary.aovlist        summary.aspell*
 [4] summary.connection     summary.data.frame     summary.Date
 [7] summary.default        summary.ecdf*          summary.factor
[10] summary.glm            summary.infl           summary.lm
[13] summary.loess*         summary.manova         summary.matrix
[16] summary.mlm            summary.nls*           summary.packageStatus*
[19] summary.POSIXct        summary.POSIXlt        summary.ppr*
[22] summary.prcomp*        summary.princomp*      summary.srcfile
[25] summary.srcref         summary.stepfun        summary.stl*
[28] summary.table          summary.tukeysmooth*

   Non-visible functions are asterisked
```

kuvatakse kõik funktsioonid nimega `summary.objektiklass()`, ent käsu

```
> methods(class="matrix")
 [1] anyDuplicated.matrix as.data.frame.matrix as.raster.matrix*
 [4] boxplot.matrix       determinant.matrix   duplicated.matrix
 [7] edit.matrix*         head.matrix          isSymmetric.matrix
[10] relist.matrix*       subset.matrix        summary.matrix
[13] tail.matrix          unique.matrix

   Non-visible functions are asterisked
```

peale kuvatakse funktsioonid nimega `funktsiooninimi.matrix()`.

Erisümbolid

Lisaks seninähtule on Ris veel mõned erisümbolid, mis omavad kindlat tähendust. Mõnega neist ei pruugi kasutaja üldse kokku puutudagi, samas näiteks puuduvate väärtustega toimetulek on andmeanalüüsis kriitiline (ja võib alguses Ris problemaatiline tunduda). Erisümboleid ei saa me üle kirjutada (neid muutujatena kasutusele võtta). Üldse on segaduse vältimiseks mõistlik hoolitseda, et näiteks meie esialgses andmemaatriksis ei leidu välju, mis ühtiks mõne Ri erisümboliga, ent tähistavad tegelikkuses hoopis midagi muud.

Sümbol `NA` (*not available*) on Ris puuduva väärtuse tähiseks. Kuigi mitmed mudeli sobitamise funktsioonid omavad argumenti, millega saab näidata, mida puuduva väärtusega ette võtta, võib vahel olla vajalik ka ise puuduvate väärtustega vaatlused andmestikust eraldada. Siin on kaks head abimeest: funktsioonid `is.na()` ja `complete.cases()`, millest esimene kontrollib väärtuse puudumist eraldi iga elemendi jaoks, teine aga kõigi vaatluste olemasolu terve andmerea (vaatluse) jaoks. Tuleb tähelepanu pöörata, et isegi kui funktsioonil on olemas argument, mis võimaldab puuduvaid väärtusi eemaldada, võib selle vaikeväärtus olla selline, et eemaldamist ei toimu. Kui aga R üritab rakendada funktsiooni (näiteks kokku liita) väärtustele, millest mõni on puuduv, siis on ka tulemuseks `NA`[1].

Sümbolid `Inf` ja `-Inf` märgivad, et tulemus on vastavalt „lõpmata suur“ või „lõpmata väike“. Sellise tulemuse võime saada näiteks nulliga jagamisel. Samas ka siis kui arv on lihtsalt liiga suur Ri jaoks – näiteks `2**1024` on liiga suur. Sümbol `NaN` (*not a number*) tuleb mängu siis kui meie poolt nõutud (numbrilisele) tehtele ei ole võimalik anda vastust isegi mitte `Inf` ja `-Inf` abil. Tegu on seega tõelise määramatusega. Muuhulgas näiteks

```
> 1/0-1/0
[1] NaN
```

Lisaks on Ris veel niinimetatud nullobjekt, mida tähistataksegi `NULL`. Vahel võib see olla näiteks funktsiooni argumendi väärtuseks märkimaks et me ei väärtusta vastavat argumenti. Samuti võib funktsioon „tagastada nullobjekti“. See tähendab sisuliselt, et funktsioon ei tagasta midagi. Ka uue objekti loomisel võib vahel sümboliga `NULL` kokku puutuda.

Ülesanded

I Täiendame veidi hundirünnakute andmeid. Olgu

```
> maru=c(39, 6, 27, 9, 23, 17, 15, 12)
> kisk=c(27, 18, 17, 3, 0, 7, NA, 1)
> kubermang = c("Mogiljev", "Peterburg", "Podolsk", "Smolensk",
         "Vilnius", "Vladimirsk","Harkov","Minsk")
```

Tekita andmemaatriks, mille ridadel ja veergudel on nimed. Rakenda funktsiooni `mean()`. Mis juhtus? Proovi funktsioonide `is.na()` ja `complete.cases()` toimimist andmemaatriksil. Kuidas kasutada viimast neist ainult täielike kirjete valimiseks andmestikust? Vali need ja leia nüüd tunnuste keskmised. Uuri funktsiooni `mean()` abifaili, et saada teada, kuidas oleks tegelikult mõistlik talitada. Talita mõistlikult.

[1] Mõne loogilise tingimuse korral võime saada ka olukorra, kus väärtus tõepoolest tähtsust ei oma. Sel juhul ei ole muidugi tulemuseks puuduv väärtus.

Vaata, mis oleks juhtunud kui oleksime loonud tavalise maatriksi ja rakendanud sellele funktsiooni `mean()`. Kas oskad leida funktsiooni, mis teeks ka tavalise maatriksi korral „õiget asja“?

II Soovime hinnata keskmist marutõverünnakute ja kisklusrünnakute suhet. Ilmselt on see Harkovi kubermangus teadmata ja seetõttu tuleks see kubermang eemaldada. Tee seda tekitades uus andmemaatriks nimega `mat`. Kas nüüd võime kasutada käsku

```
> mean(mat$maru / mat$kisk)
```

või tekib probleem?

Lihtsad funktsioonid: operaatorid

Ehkki me oleme juba näinud mõnesid lihtsaid funktsioone töös (näiteks `mean()` ja `cor()`) on siinkohal mõtet üles lugeda mõned lihtsad funktsioonid, mida sageli vaja läheb.

Operaatorid `+`, `-`, `*`, `/` ning `**` või `^` astendamise jaoks on traditsioonilised. Lisaks veel täisarvulist jagamist teostav `%/%` ja täisarvulise jagamise jäägi leidev `%%`.

Lihtsad funktsioonid: trigonomeetria ja logaritm

Igati ootuspäraselt töötavad `sin()`, `cos()`, `tan()`. Vastavad pöördfunktsioonid on `asin()`, `acos()`, ja `atan()`. Konstant π on defineeritud muutujana `pi`. Funktsioon `\log()` leiab vaikimisi naturaallogaritmi, ent argument `base` võimaldab valida ka teise aluse. Eksponentfunktsioon peitub käsu `\exp()` taga.

Lihtsad funktsioonid: ümardamine

Funktsioon `floor()` ümardab murdosa allapoole, `ceiling()` ülespoole. Meile „tavalist“ ümardamist võimaldab funktsioon `round()`, mille argumendiga `digits` saame määrata soovitava komakohtade arvu (vaikimisi ümardatakse täisarvuks). Analoogiliselt võimaldab ümardada ka funktsioon `signif()`, ent siin määrab argument `digits` soovitava tüvenumbrite arvu. Lisaks eksisteerib veel funktsioon `trunc()`, mis ümardab alati nulli suunas (negatiivseid arve suuremaks, positiivseid väiksemaks).

Lihtsad funktsioonid: jadade tekitamine

Jadade moodustamisel on abiks `:`, `seq()` ja `rep()`, mille tööd vaatleme järgmiste näidete abil.

```
> -3 : 7
 [1] -3 -2 -1  0  1  2  3  4  5  6  7
> 6 : 1
[1] 6 5 4 3 2 1
> seq(from=3, to=6, by=0.5)
```

```
[1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0
> seq(3, 6, length=5)
[1] 3.00 3.75 4.50 5.25 6.00
> rep(3, times=6)
[1] 3 3 3 3 3 3
> rep(c(2, 5), times=c(1, 3))
[1] 2 5 5 5
> rep(c(2, 5), each=4)
[1] 2 2 2 2 5 5 5 5
```

Lihtsad funktsioonid: objektide mõõtmed

Jadade pikkuse leidmiseks on funktsioon `length()`, funktsioon `dim()` annab vähemalt kahemõõtmelise objekti (näiteks maatriks või massiiv) „mõõdud“. Kahemõõtmeliste objektide korral saab kasutada ka `ncol()` ja `nrow()`.

Ülesanded

III Vaata, millise tulemuse annavad `1 : 3 + 2`, `1 : 3 * 2` ja millise `1 : 3 ^ 2`. MIda järeldad tehete järjekorra kohta? Mõtle ka, millised olid näiteks esimese avaldise puhul liidetavate mõõtmed. Mis juhtub aga `1 : 3 + c(2, 2)` korral?

IV Kuidas on kõige lihtsam moodustada jada ühest kümneni, kus elementidel on vahelduvad märgid (vaheldumisi pluss ja miinus)?

Lihtsad funktsioonid: arvkarakteristikud ja järkstatistikud

Keskmist saab leida funktsiooni `mean()` abil. Argument `trim` võimaldab määrata murdosa vähimatest ja suurimatest vaatlustest, mis keskmise arvutamisel välja jäetakse. Mediaani leiab `median()`, kvantiile üldiselt `quantile()`. Dispersiooni leiab `var()`, standardhälbe `sd()`. Valimi haarde (vähim ja suurim element) leiab `range()` ja kvartiilide vahe `IQR()` kovariatsiooni `cov()` ja korrelatsiooni `cor()`. Kui kõigil pea kõigil eelnimetatutel jäetakse puuduvad väärtused välja argumendi `na.rm` abil siis viimase kahe funktsiooni jaoks tuleb kasutada argumenti `use`, millel ka rohkem võimalikke väärtusi. Lisaks eksisteerib veel funktsioon `rank()`, mille korral tagastatakse elementide järjekorranumbrid variatsioonreas (suurimal elemendil on see üks, teiseks suurel kaks jne, võrdsete vaatluste korral on väärtuse määramiseks erinevaid võimalusi).

```
> mean(maru / kisk, trim=0.2, na.rm=T)
[1] 4.09225
> median(maru)
[1] 16
> quantile(1 : 5, c(0.25, 0.5, 0.99))
 25%  50%  99%
2.00 3.00 4.96
```

Lihtsad funktsioonid: veel huvitavat

Vektori elementide kasvavalt või kahanevalt järjestamiseks on kasutatav funktsioon `sort()`[2]. Funktsioon `rev()` pöörab vektori järjestuse ümber (esimesest elemendist saab viimane, teisest eelviimane jne).

Funktsioonid `cumsum()`, `cumprod()`, `cummin()` ja `cummax()` võimaldavad vektoril kumulatiivseid operatsioone aga puuduvad väärtused tuleks esmalt eemaldada. Näiteks

```
> cumsum(maru)
[1]  39  45  72  81 104 121 136 148
> cummax(kisk)
[1] 27 27 27 27 27 27 NA NA
```

Funktsioonid `min()` ja `max()` on minimaalse ja maksimaalse elemendi leidmiseks, `which.min()` ja `which.max()` aga leiavad vastavate elementide järjekorranumbri. Funktsioonid `pmax()` ja `pmin()` leiavad maksimumi ja miinimumi komponenthaaval.

```
> max(maru)
[1] 39
> which.max(maru)
[1] 1
> pmax(kisk, maru)
[1] 39 18 27  9 23 17 NA 12
```

Maatriksite jaoks on otstarbekas maatrikskorrutamist võimaldav `%*%`, maatriksi transponeerib (vahetab ära read ja veerud) funktsioon `t()` ning pöördmaatriksi leiab `solve()`. Diagonaalmaatriksit saab luua (või maatriksilt peadiagonaali eraldada) funktsiooniga `diag()`. Maatriksi determinandi leiab funktsioon `det()`. Kaks väga kasulikku funktsiooni on veel `rbind()` ja `cbind()`, mis kleebivad objekte kokku vastavalt ridupidi ja veergepidi.

Tuntud funktsioonid on veel näiteks `choose()` kombinatsioonide arvu leidmiseks ja `sample()` juhusliku valimi võtmiseks vektorist.

Ülesanded

V Vaata, millisel kujul võivad andmed olla funktsiooni `cor()` rakendamisel. Proovi, mis juhtub kui tunnuseid on rohkem kui kaks?

VI Pane hundi marutaudi- ja kisklusrünnakute andmed kokku `cbind()` abiga. Leia selle põhjal tunnustevaheline Spearmani korrelatsioonikordaja, ümarda see täpsuseni kaks kohta peale koma ja salvesta tulemus uue muutujana. Kuva see muutuja. Kus on teine komakoht?

[2] Aga see on kasutatav vaid vektorite jaoks. Seda, kuidas sorteerida mitmedimensionaalseid objekte nagu andmemaatriks vaatleme edaspidi.

Loogiliste tingimuste konstrueerimine

Me oleme juba näinud, et üks viis objektist valikuliselt andmeid välja kutsuda on loogiliste tingimuste kasutamine. Praktikas erinevaid analüüse läbi viies on ilmselt just see viis enim kasutust leidev. Meenutame, et idee oli lihtne – tõeväärtusvektoris peab `TRUE` seisma nendel positsioonidel, millistelt andmeid soovime. Ülejäänud kohtadel peab paiknema `FALSE`.

Kõige lihtsamad tingimused on konstrueeritavad operaatorite `==`, `>=`, `>`, `<`, `<=`, `!=` abil. Eriti tasub rõhutada operaatorit `==`, sest ikka kipub juhtuma, et võrdustingimuse kirjutamise asemel kasutatakse hoopis omistamise sümbolit `=`.

```
> kubermang == "Vilnius"
[1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
> kisk > 20
[1]  TRUE FALSE FALSE FALSE FALSE FALSE    NA FALSE
> maru != 9
[1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

Lihtsaid tingimusi saab omavahel kokku panna operaatorite `&` ja `|` abil. Esimene (JA) tagastab `TRUE` vaid siis kui mõlemad tingimused on tõesed, teine (VÕI) aga ka siis kui vaid üks tingimustest on tõene. Lisaks veel `!` (EI), mis muudab tõeväärtused vastupidiseks.

```
> kisk > 20 & maru != 9
[1]  TRUE FALSE FALSE FALSE FALSE FALSE    NA FALSE
> kisk > 20 | maru != 9 #miks ei ole siin puuduvat väärtust?
[1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
> !(kisk > 20 | maru != 9)
[1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

Kui meil on palju ühte tüüpi tingimusi siis on abiks ka funktsioonid `any()` ja `all()`. Esimene tagastab `TRUE` kui vähemalt üks tingimus kehtib. Teine vaid siis kui kehtivad kõik tingimused.

```
> any(is.na(kisk)) #kas on puuduvaid väärtusi?
[1] TRUE
> all(!is.na(kisk)) #kas kõik väärtused on olemas?
[1] FALSE
```

Funktsioon `which()` ei tagasta tõeväärtusi vaid hoopis nende elementide positsiooninumbrid, mille korral tingimus on täidetud.

```
> which(is.na(kisk)) #millised väärtused on puudu?
[1] 7
```

Väga kasulik on ka operaator `%in%`, mis tagastab `TRUE` nende väärtuste jaoks mis eksisteerivad etteantud võrdlushulgas. Näiteks

```
> maru %in% kisk
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
```

Operatsioone hulkadega saab teostada funktsioonide `union()` (ühend ehk kõik erineva väärtused, mis esinevad kahe hulga peale kokku), `intersect()` (ühisosa ehk kõik väärtused, mis esinevad mõlemas väärtustehulgas) ja `setdiff()` (vahe ehk kõik need väärtused, mis esinevad ainult esimeses hulgas) abil. Lisaks eksisteerib veel funktsioon `setequal()`, mis kontrollib, kas väärtushulgad kattuvad.

```
> union(maru,kisk)
 [1] 39  6 27  9 23 17 15 12 18  3  0  7 NA  1
> intersect(maru,kisk)
[1] 27 17
> setdiff(maru,kisk)
[1] 39  6  9 23 15 12
> setequal(rep(7,3),c(7,7))
[1] TRUE
```

Ülesanded

VII Kuidas valida andmemaatriksist nimega `andmed` välja need tunnuse `tiivapikkus` väärtused, mille korral tunnuse `vaatluspäev` väärtused on vahemikus 11 kuni 20 või 41 kuni 50? Kuidas valida need kirjed, mille korral `tiivapikkus` ei jää vahemikku 30 kuni 50?

VIII Olgu meil andmemaatriks `leiud`, mis sisaldab lisaks taimeliigi nimele (tunnus `liik`) ka leiukoha koordinaate (tunnused `pikkus` ja `laius`). Lisaks on meil eraldi kaks vektorit `katI` ja `katII`, mis sisaldavad kaitse all olevate liikide nimetusi. Looduskaitse eesmärkidel on meil vaja välja selgitada nende leiukohad, kus kasvavad kaitse alla kuuluvad liigid. Eralda need asukohad andmemaatriksist `leiud` (ja salvesta objektina `kaitstud`).

IX Olgu meil nüüd veel üks teine andmemaatriks `planeeringud`, kus samuti objektide koordinaadid (tunnused samuti vastavalt `pikkus` ja `laius`). Kuidas kontrollida, kas mõne kaitsealuse liigi leiukoht kattub mõne planeeringu asukohaga?

Andmete import ja eksport

Seni oleme andmed Ri käsitsi sisestanud. Sagedasem on aga olukord, kus meil on andmestik juba eelnevalt digitaalsel kujul (failina) olemas. Andmeid on võimalik hoida erinevates formaatides, Windowsi kasutaja jaoks on kõige tüüpilisem ilmselt nn Exceli tabel ehk .xls formaat. R on võimeline otse .xls failidest (või ka andmebaasidest) andmeid importima (selleks saab kasutada näiteks paketi `RODBC` abi), ent kasutajale lihtsaim[1] viis andmete impordiks on (nt Exceli või OpenOffice Calci abiga) fail esmalt .csv formaati teisendada ja alles seejärel Ri sisse lugeda. Sama põhimõte (ainult vastupidises järjekorras) kehtib ka andmete Rist välja viimisel. Vajaduse korral tuleb appi võtta pakett `foreign`, mille funktsioonid võimaldavad andmestike importi enamikest tuntud andmetöötlussüsteemide failiformaatidest.

Failiformaadi .csv korral on andmed esitatud lihtteksti kujul ehk nii, et need on loetavad igasuguse tekstieditoriga (ja ka veebibrauseriga). Andmevälju eraldab mingi kindel eraldussümbol[2] (*delimiter*) ja andmerea lõppu märgib 'Enter'.

Andmete sisselugemiseks on Ris funktsioon `read.table()`, millele tuleks ette anda

I. andmefaili asukoht (argument `file`)

II. kas andmefaili esimeses reas paiknevad veergude nimed (argument `header` vaikeväärtusega `FALSE`)

III. eraldajana kasutatav sümbol (argument `sep`)

IV. arvu täisosa murdosast eristav sümbol ehk kümnendkohtade alguse tähis (argument `dec`)

V. puuduvate väärtuste tähistus (argument `na.strings`)

Faili asukoht antakse jutumärkide vahel ning see koht võib olla nt kas asukoht kõvakettal (näiteks `"C:\\hp\\andmed.csv"`) või ka internetiaadress (`"http://www.ut.ee/~keiser/andmed.csv"`). Kui veebiaadressi üleskirjutus on standardne, siis nn kohalikke aadresse on soovitav märkida just topeltkaldkriipsu abil (teine võimalus on kasutada ka lokaalsetel teedel selliseid kaldkriipse nagu veebiaadressidel).

Eraldajaks on sageli kas semikoolon, koma, tühik või tabulaator (ehk 'tab'). Vastavad väärtused on `";"`, `","`, `" "` ja `"\t"`. Vaikeväärtuseks on hoopis `""`, mis loeb eraldajaks ühe või ka mitu järjestikust tühikut või tabulaatorit. Puuduvate väärtuste korral tuleb `sep` kindlasti ise väärtustada.

Täisosa ja murdosa piiri vaikeväärtus on `"."`, ent eestikeelse Windowsi korral on levinud selle piiri tähistamine koma abil ning sel juhul tuleks määrata `dec=","`.

Puuduvate väärtuste tähistamise vaikeväärtus on `"NA"`. Ise võib anda ka mitu puuduva väärtuse tähist nagu näiteks `na.strings=c("NA", "")` (mõistagi ei tohi need väärtused kokku langeda argumentidele `sep` ja `dec` antud väärtustega).

[1] Erand on see, kui andmestik on tõesti väga-väga(-väga) suur. Sellisel juhul ei ole võimalik kõiki andmeid sisse lugeda ja nt SQL päring on siis vägagi otstarbekas.

[2] kaks järjestikust eraldussümbolit tähendab seega, et antud andmeväljal asuv väärtus on puuduv ehk teadmata

Argumente on funktsioonil `read.table()` muidugi veel, ent enamustel juhtudel ülejäänute vaikeväärtusi põhjust muuta ei ole. Keeruka andmefaili korral[3] võib vaja olla hoopis funktsiooni `scan()`.

Väärtustamist vajavate argumentide hulga vähendamiseks on olemas veel neli eelseadistatud andmete sisselugemisfunktsiooni. Nende kohta saad lugeda funktsiooni `read.table()` abifailist. Eesti seadistusega Windowsi korral on küllalt tüüpiline, et saame kasutada funktsiooni `read.csv2()`.

Tasub märkida, et kui andmetabelis ei esine puuduvaid väärtusi siis on lihtsaimaks variandiks hoopis andmete otse Windowsi puhvrist sisse lugemine. Selleks tuleb Excelis või Calcis andmed märgistada ja 'ctrl+c' abil puhvrisse võtta. Nüüd saab Rile andmete asukohaks määrata `file="clipboard"` ja lisaks tuleks seada ka `sep="\t"`.

Üks asjaolu veel, mis kipub andmete importimisel ununema – sisseloetud andmed (R teeb neist andmemaatriksi) tuleks kindlasti ka omistada!

Andmete ekspordi kohta käiv on paljuski sarnane impordiga – lihtsaim on produtseerida fail lihtteksti kujul. Selleks saab kasutata funktsiooni `write.table()`, millele tuleks ette anda

I. andmemaatriksi nimi, mida eksportida

II. asukoht, kuhu andmefail luua tuleks (argument `file`)

III. kas väljastatakse ka tunnuste nimed (argument `col.names` vaikeväärtusega `TRUE`)

IV. millist sümbolit tuleks kasutada väärtuste eraldamiseks (argument `sep`)

V. millist sümbolit tuleks kasutada täisosa eraldamiseks murdosast (argument `dec`)

VI. kuidas tähistada puuduvad väärtused (argument `na`)

Funktsiooni `write.table()` vaikeväärtused on igati mõistlikud. Erandiks on vaid ehk `row.names=TRUE`, mis üldjuhul tähendab, et väljastatava faili esimese veeru moodustab objektide järjekorranumber. Muuhulgas võib see valik olla problemaatiline seetõttu, et tekib probleem veerunimede paiknemisega – sellest saab aga üle kui valida ka `col.names=NA`, mis sel juhul lisab reanimede ette ühe eraldaja[4] .

Windowsis on ka selle funktsiooni puhul kasutatav valik `file="clipboard"` (analoogiliset impordiga). Välises programmis saab andmed puhvrist kätte nagu ikka 'ctrl+v' abil.

Ülesanded

I Vaata brauseriga faili `"http://www.ut.ee/~keiser/andmed.csv"` ning pane kirja käsk, mis võimaldab selle andmestiku koos veerunimedega Ri importida. Impordi see andmestik Ri objektina nimega `andmed1`.

[3] see tähendab siiski lihtteksti kuju, lihtsalt andmeread ei pea olema korrapärased

[4] Huvi võib pakkuda veel Ri numbrite esitamine. Paljukohalisi arve väljendab R reeglina nn teaduslikus formaadis ehk kümne astmete abil (näiteks `1e+06` tähistab arvu 1 000 000). R teeb seda siiski ühtlaselt selles mõttes, et kui tunnuse üks väärtus teaduslikus formaadis esitatakse, siis esitatakse nii kõik tunnuse väärtused. See ei pruugi aga nii jääda tunnuste ekspordil, mistõttu võib tekkida soov Ri numbrite esitamise viisi ise seadistada. Selleks saab kasutada funktsiooni `options()` argumenti `scipen`. Argument `digits` võimaldab seada Ri tulemuste esitamise täpsust.)

II Väljasta see andmestik otse mõnesse tabelarvutusprogrammi.

Töökataloog

Töötades failidega arvuti kõvakettal on küllalt ebamugav iga kord pikki failide asukohti sisestada. Veelgi enam tekib probleeme kui liigutame andmeid ühelt andmekandjalt teisele (näiteks vahel töötame kõvakettalt vahel aga mälupulgalt), sest siis tuleb ka Ri skriptis leiduvaid linke pidevalt uuendada. Selle probleemi lahenduseks on Ris töökataloog, mis Windowsis on vaikimisi kasutaja *My documents*. Töökataloogi kuvamiseks on funktsioon `getwd()` ja määramiseks `setwd()`. Viimasele tuleb asukoht edastada nii nagu nägime seda näiteks funktsiooni `read.table()` puhul[5].

Seega Ri sessiooni on sageli mõistlik alustada töökataloogi seadistamisega käsuga stiilis

```
setwd("kataloog kõvakettal koos täieliku teega")
```

ja hiljem kasutame impordiks käsku kujul

```
read.table(failinimi, header=tõeväärtus, sep=sõne, dec=sõne, na.strings=sõne(d))
```

ja ekspordiks kujul

```
write.table(andmestik, failinimi, col.names=väärtus, sep=sõne, dec=sõne, na=sõne)
```

Seejuures võib failinimele eelneda alamkataloog, sest küllalt sageli on meil mitu projekti ja need asuvad eri kataloogides.

Saades import- või eksportprotseduuri käigus veateate on levinuimaks põhjuseks vigaselt esitatud faili asukoht (näiteks vastavat kataloogi ei eksisteerigi). Sellisel juhul on veateates muuhulgas kirjas *cannot open the connection*.

Funktsioon `source()` on samuti mõtet siinkohal ära mainida. Sellega on võimalik Ri sööta kõik (et-teantud) failis leiduvad Ri käsud. Seega kui kaks teadlast töötavad sama probleemi erinevate osade kallal ja neile ei paku huvi teise programmikoodi põhjalikumalt vaadelda siis võib üks teisele saata oma programmikoodi ja see teine mugavalt koodi läbi lasta, et siis ise probleemi lahendamisega edasi minna. Muidugi võib ka lihtsalt olla, et meil endal on koodi kirjutamine pooleli ja me tahame kiirelt pooleli jäänud kohast edasi minna.

Ri salvestusformaat

Oletame, et meil on juba andmed Ris ja me ei plaani neid enam eksportida andmestikuna (vaid näiteks produtseerida nende andmete põhjal graafikuid) või siis soovime lihtsalt ajutiselt töö katkestada. Siis

[5] aga töökataloog ei saa muidugi olla veebiaadress

on hea teada, et Ril on ka oma andmete salvestamise formaat. Salvestamisel toimub funktsiooni `save()` abil. Ette tuleb anda salvestatav objekt ja argument `file`, mille väärtuseks asukoht koos failinimega (traditsiooniliselt märgitakse Ri tehtud failide laiendiks .rda aga selle saame ise määrata).

Salvestatud faili laadimine toimub funktsiooni `load()` abil. Kõik failis leiduvad objektid on peale laadimist Ris kättesaadavad, mingit eraldi omistamist enam teha vaja ei ole.

Kui soovime salvestada kõik loodud andmestikud ja muud objektid siis on mõistlik kasutada funktsiooni `save.image()`, mis käivitatakse automaatselt ka Ri sessiooni lõpetamisel kui lubame tehtud töö salvestamise. Kui me lubame tehtud töö automaatsalvestamise sessiooni lõpus (ja me ei ole ära muutnud vaikimisi kasutatavat töökataloogi), siis see salvestus järgmisel Ri käivitamisel ka laetakse. Kui me seda ei soovi siis võime *.Rdata* nimelise faili töökataloogist kustutada.

Ka sisestatud käsud saab niimoodi salvestada. Selleks on funktsioon `savehistory()`. Seegi funktsioon käivitatakse automaatselt Ri sessiooni lõpetamisel kui lubame tehtud töö salvestamise. Käskude ajaloo laadimiseks saab kasutada funktsiooni `loadhistory()` ja kuvamiseks funktsiooni `history()`[6]. Kui me lubame tehtud töö automaatsalvestamise sessiooni lõpus (ja me ei ole ära muutnud vaikimisi kasutatavat töökataloogi), siis laetakse ka käskude ajalugu. Kui me seda ei soovi siis võime *.Rhistory* nimelise faili töökataloogist kustutada[7].

Ülesanded

III Vaata oma praeguse R sessiooni töökataloogi ja proovi seda muuta. Salvesta uude töökataloogi üle-eelmises ülesandes sisseloetud andmestik.

IV Taaskäivta R ja proovi tekitatud fail nüüd uuesti Ri sisse lugeda. Kuva käskude ajalugu. Mitu käsku selles sisaldub?

Esmane ülevaade andmetest

Peale andmestiku sisselugemist on meie esmaseks huviks tuvastada, kas R on andmed õigesti vormindanud. Hea ülevaate annavad funktsioonid `str()` ja `summary()`. Mõlemad on geneerilised funktsioonid ja on kasulikud ka muud tüüpi objektide puhul. Funtsioon `str()` (*structure*) annab meile teada objekti klassi ja üritab arusaadavalt väljendada objekti struktuuri. Andmemaatriksi puhul kuvatakse tunnused ja nende tüübid koos esimeste väärtustega. Funktsioon `summary()` annab ülevaate tunnuste jaotusest. Arvuliste tunnuste puhul tähendab see muuhulgas kvartiile ja keskmist, mittearvuliste tunnuste puhul tunnuse väärtuste sagedust.

Kasulik võib olla ka funktsioon `head()`, mis kuvab andmestiku esimesed kirjed. Eksisteerib ka funktsioon `tail()`.

[6] Seda funktsiooni saab kasutada ka niisama sessiooni käigus ilma midagi salvestamata või laadimata

[7] Tegelikult saab automaatsalvestamist ja -laadimist reguleerida ka otse Ri käivitades. Vaata näiteks lisa B.1 manuaalist *An Introduction to R*

Ülesanded

V Kasuta eelmises alapunktis nimetatud funktsioone sisseloetud andmestiku peal ja selgita välja, kas andmestikus esineb puuduvaid väärtusi. Milline funktsioonidest võimaldas selle jaoks parimat ülevaadet?

VI Kas märkasid eelneva punkti käigus, et andmestikus on midagi paigast ära? Kui ei siis otsi veel! Mis on paigast ära? Kuidas seda parandada?

Mugavam töö andmemaatriksitega

Eelmise näite juures nägime, mis võib olla andmestike korral tülikas – ühe tunnuse eraldamiseks peame alati kaasama ka andmestiku nime. Kui me töötame ainult ühe andmestikuga korraga, siis tundub, et andmestiku nime võiks ju „eest ära jätta“ ehk kasutada lühinimesid. Seda on võimalik realiseerida kahel viisil.

Kasutades funktsiooni `attach()` luuakse otsingujärjekorda lisakoht, kuhu kopeeritakse kõik objektis leiduvad elemendid (näiteks andmemaatriksi veerud). Nüüd saame neid veerge küll otse välja kutsuda, ent neid muutes või uusi objekte tekitades ei kajastu need muutused esialgses andmestikus. Uued objektid tekivad vaikimisi hoopis üldkeskkonda, mis on aga otsingujärjekorras igal juhul eespool. Nii on kerged tekkima erinevad segadused ja seetõttu pole funktsiooni `attach()` kasutamine andmestiku lühinimede kasutamise võimaldamiseks soovitav. Andmestiku eemaldamine otsingujärjekorrast toimub funktsiooniga `detach()`.

```
> attach(andmed1)
> grupp = factor(grupp, labels=LETTERS[1 : 10])
> grupp
 [1] A A A B B B C C C D D D E E E F F F G G G H H H I I I J J J
Levels: A B C D E F G H I J
> andmed2 = andmed1[kaal > 30, ]
> andmed2 = andmed2[complete.cases(andmed2), ]
> detach(andmed1)
> attach(andmed2) #tähelepanu!
The following object(s) are masked _by_ '.GlobalEnv':

    grupp
> pikkus[grupp == "A"] #kas need raskete grupi A isendite pikkused?
[1] 13.99 13.65 11.92
> detach(andmed2)
> rm(grupp)
```

Mõistlikum on kasutada funktsiooni `attach()` asemel funktsioone `with()` ja `within()`. Neist esimene võimaldab lühinimede kasutamist ja tagastab etteantud käsu või käskude tulemuse[8]. Teine võimaldab

[8] kuid kui käske on mitu tuleb need paigutada sümbolite `{ }` vahele

teha muutusi andmestikus endas (näiteks lisada uusi tunnuseid) ja tagastab muudetud andmestiku, ent omistamiseks tuleb kasutada `=` asemel `<-`[9].

```
> andmed1 = within(andmed1, grupp <- factor(grupp, labels=LETTERS[1 : 10]))
> with(andmed1, pikkus[grupp == "A"])
[1] 5.45 7.95 5.23
```

Väga mugav on andmestikega koos kasutada funktsiooni `subset()`, mis eraldab andmemaatriksist loogilisele tingimusele vastavad kirjed. Me nägime eelmises praktikumis, kuidas seda ise teha, ent siin on üheks plussiks asjaolu, et juhul kui tulenevalt puuduvatest väärtustest ei ole võimalik teada saada ka tõeväärtust, jäetakse kirje lihtsalt välja.

```
> subset(andmed1, kaal > 35)
   toit grupp pikkus  kaal
28    a     J  17.82 47.58
29    b     J  15.25 38.69
```

Andmestike sorteerimine ja ühendamine

Andmestiku sorteerimine etteantud tunnus(t)e järgi on Ris mõneti ebatraditsionaalne – funktsioon `order()` tagastab ridade järjestuse etteantud tunnus(t)e järgi sorteerituna. Seega tuleb read nüüd selle järjestuse alusel välja kutsuda. Näiteks käsk

```
> andmed1[order(andmed1$kaal, andmed1$pikkus), ]
```

sorteerib andmestiku `andmed1` tunnuse `kaal` järgi kasvavalt. Kui esineb võrdseid kaale, siis järjestatakse viigilised vaatlused tunnuse `pikkus` järgi kasvavalt.

Põhimõtteliselt saab andmemaatrikseid kokku panna varem tutvustatud funktsioonide `cbind()` ja `rbind()` abiga. Kui meil on lihtsalt vaja andmestikku lisada kirjeid (ja tunnused on mõlemas andmestikus samad ja ka samas järjekorras) siis võib viimane neist olla isegi mõistlik. Küll aga ei saa me tüüpiliselt lisada andmestikke uusi tunnuseid funktsiooni `cbind()` abil, sest reeglina ei ole vaatlused andmestikes samas järjekorras või on ühes andmestikus ka mõned sellised vaatlused, mida teises pole. Siin tuleb appi funktsioon `merge()`, mis vaikimisi leiab andmemaatriksite ühised tunnusenimed ja üritab nende tunnuste põhjal andmestikke liita. Kui me soovime liita mingite mittekattuvate nimedega tunnuste abil või mitte kasutada kõiki kattuvate nimedega tunnuseid siis tuleb ise ära määrata tunnused, mida kasutada soovime väärtustades argumendid `by.x` ja `by.y`. Argumendid `all.x` ja `all.y` reguleerivad, kuidas käitutakse kui vastavalt esimeses või teises andmemaatriksis leidub vastuseta kirje. Väärtus `TRUE` tähendab, et antud rida ühendmaatriksisse siiski tekitatakse ja puuduvate tunnuste kohale kirjutatakse `NA`. Vaikeväärtus on aga `FALSE`. Lõpptulemus sorteeritakse vaikimisi ühiste veergude järgi.

[9] Tegelikult on see Ris üldse soovitav, ent meie kasutame lühiduse mõttes operaatorit `=`. Reeglina sellest midagi halba ei sünni.

Ülesanded

VII Vaata andmestikku `http://www.ut.ee/~keiser/andmed2.csv` ja loe see Ri. Kas selle andmestiku saab objektiga `andmed1` liita?

VIII Teosta andmestike liitmine ja salvesta see objektina `uus`. Mitu tunnust on liitandmestikus? Kas liitmine toimus ühe või mitme tunnuse abil? Salvesta liitandmestik endale hilisemaks kasutamiseks.

Korduvad kirjed

Vahel on andmestik mitme inimese ühistöö ja võib esineda korduvaid kirjeid. Vahel soovime lihtsalt leida mingite tunnustekombinatsiooni korral erinevad võimalikud väärtused andmestikus. Funktsioon `unique()` jätab alles ainult ühe koopia igast korduvast kirjest.

```
> unique(uus$mootja)
[1] E B A
Levels: A B E
> unique(uus$toit)
[1] c a b
Levels: a b c
> m.toit = data.frame(uus$mootja, uus$toit)
> unique(m.toit) #üsna vähe kombinatsioone
   uus.mootja uus.toit
1           E        c
2           B        c
3           A        a
5           B        b
11          E        b
```

Funktsioon `duplicated()` tagastab tõeväärtusvektori, kus `TRUE` seisab vaid nendel positsioonidel, mis sisaldavad juba eelmistel positsioonidel esinenud kirjeid.

```
> duplicated(uus$mootja)
 [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> duplicated(uus$toit, fromLast=TRUE) #tagantpoolt ettepoole
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE FALSE FALSE FALSE
> duplicated(m.toit)
 [1] FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Riga kaasasolevad andmestikud

Riga on kaasas hulgaliselt näidisandmestikke (enamus neist sisaldavad päris andmeid ja on paketis `datasets`, mis vaikimisi tööle laetud) ning töölelaetud pakettide andmestikest saab ülevaate käsu `data()` abil.

Kõik installeeritud pakettide andmestike nimed ja kirjeldused kuvab käsk

```
> data(package = .packages(all.available = TRUE))
```

Mitte kõikide pakettide andmestikud ei ole kohe peale paketi töölelaadimist kättesaadavad. Kui näeme, et paketis sisalduvad andmestikud, ent neid ei ole võimalik nime järgi välja kutsuda, siis tuleb need esmalt kättesaadavaks teha andes andmestiku nime argumendina (sõne kujul) funktsioonile `data()`

Ülesanded

IX Anna käsk `data()` ja vaata nimistut. Vaata, mida kujutab endast andmestik `ChickWeight`. Uuri seda ka funktsiooni `str()` abil. Millist infot andmestik sisaldab? Kuidas saada andmestiku kohta veel infot?

X Võrdle andmestikku `ChickWeight` andmestikuga `BodyWeight` paketist `nlme`. Millised on sarnasused ja millised erinevused?

Nagu nägime, võib andmestik lisaks andmetele sisaldada veel mitmesugust infot. See on üldiselt andmestiku külge (või mingi muu Ri objekti külge) pandud nn atribuutide abil. Muuhulgas võib näiteks klass olla objekti atribuut. Objekti kõik atribuudid annab meile funktsioon `attributes()`. Konkreetse nimega atribuuti aitab kätte saada funktsioon `attr()`.

Tabelid

Mitmemõõtmelise sagedustabeli joonistamine on samuti üks esimesi samme andmete analüüsil. See võimaldab näiteks saada ülevaadet valimi tasakaalulisusest. Ehkki Ris on olemas ka funktsioon `table()` on tunduvalt mugavam[1] kasutada funktsiooni `xtabs()`, mis võimaldab kasutada mudelisüntaksit (ehk väljendada soovitavat tulemit valemi abil). See tähendab ka, et funktsioonil on argument `data`, millele anname väärtuseks oma andmemaatriksi ja mujal (valemi kirjapanekul) saame kasutada lühinimesid. Sagedustabeli korral meil seletatavat tunnust ei ole ja seetõttu jäetakse valemi vasak pool tühjaks. Paremal poolel on ääretunnused. Kasutades eelmisel korral loodud liitandmestikku.

```
> xtabs( ~ mootja + toit, data=uus)
      toit
mootja  a  b  c
     A 10  0  0
     B  0  6  4
     E  0  4  6
```

Kuivõrd kolme tunnuse korral ei ole tulemus enam kuigi hästi loetav siis tuleks `xtabs()` väljundile rakendada omakorda funktsiooni `ftable()`.

```
> ftable(xtabs( ~ mootja + toit + grupp, data=uus))
            grupp A B C D E F G H I J
mootja toit
A      a          1 1 1 1 1 1 1 1 1 1
       b          0 0 0 0 0 0 0 0 0 0
       c          0 0 0 0 0 0 0 0 0 0
B      a          0 0 0 0 0 0 0 0 0 0
       b          0 1 1 1 0 1 0 1 0 1
       c          0 0 0 1 1 0 0 1 1 0
E      a          0 0 0 0 0 0 0 0 0 0
       b          1 0 0 0 1 0 1 0 1 0
       c          1 1 1 0 0 1 1 0 0 1
```

Samuti on `xtabs()` tulemile võimalik rakendada funktsiooni `prop.table()` saamaks jaotustabeleid. Selle argumendile `margin` võime anda väärtuseks `1` kui soovime osakaale reasagedustest, `2` kui veerusagedustest ja jätta väärtustamata (vaikeväärtus) kui soovime üldsagedusi.

```
> prop.table(xtabs( ~ mootja + toit, data=uus))
      toit
mootja         a         b         c
     A 0.3333333 0.0000000 0.0000000
     B 0.0000000 0.2000000 0.1333333
     E 0.0000000 0.1333333 0.2000000
> prop.table(xtabs( ~ mootja + toit, data=uus), margin=1)
```

[1] väljaarvatud juhul kui soovime ühe tunnuse jaotust

```
       toit
mootja   a   b   c
     A 1.0 0.0 0.0
     B 0.0 0.6 0.4
     E 0.0 0.4 0.6
```

Kui me ei soovi osakaale vaid rea- või veerusummasid siis saame `prop.table()` asemel kasutada funktsiooni `margin.table()`, mille korral argument `margin` on saamoodi väärtustatav. Sageli on sellist tulemit võimalik saada ka lihtsamini.

Funktsiooni `xtabs()` saab kasutada ka olukorras, kus ühes veerus on juba sagedused. Selleks tuleb sümbolist `~` vasakule märkida sagedusi näitav tunnus.

Vahel soovime klasse moodustada ka pidevast tunnusest. Sel juhul saame kasutada funktsiooni `cut()`, millele anname ette vektori pideva tunnusega ja vahemike otspunktid või lihtsalt vahemike arvu argumendi `breaks` abil. Tulemuseks on faktor, millele saame juba mõistlikult[2] rakendada tabeldusfunktsioone.

Ülesanded

I Leia tunnuse `toit` jaotus andmestikus `uus`. Kuidas näeks välja lühim lahendus? Kuidas oleks see jaotus tuletatav selle alapunkti esimese näite väljundist?

II Leia tunnuse `kaal` haare. Tükelda tunnus viieks klassiks. Kas vaikimisi (`breaks=5`) tekivad normaalsete otspunktidega klassid? Tee kümne ühiku laiused klassid nii, et esimese alguspunkt on 0 ja viimase lõpp-punkt 50, ning salvesta see uue tunnusena andmestikku. Kuva uue tunnuse ja tunnuse `toit` ühisjaotus. Konstrueeri selle põhjal mõistlik jaotustabel. Interpreteeri saadud tulemust.

Graafikute tüübid

Nii eelduste kontrollis kui hüpoteeside genereerimisel on graafilised meetodid sageli kesksel kohal. R on tuntud oma külluslike graafikavõimaluste poolest, ent graafikute peenhäälestamine ei pruugi olla kuigi lihtne.

Histogrammi saame joonistada funktsiooni `hist()` abiga. Kui klassid (saab ette anda argumendi `breaks` abil nii nagu funktsiooni `cut()` korral, ent ka vaikimisi kasutatav meetod käitub üldjuhul mõistlikult) on defineeritud võrdsete laiustega siis kujutatakse vaikimisi sagedusi, muidu teisendatakse graafikut nii, et iga tulba pindala väljendab tõenäosust[3].

Niisiis näiteks käsu

```
> hist(uus$kaal)
```

[2] Näiteks `xtabs()` on nõus tabeldama ka pidevaid tunnuseid. Aga sellises tabelis on reeglina arvukalt nulle.
[3] ehk lähendatakse tihedusfunktsiooni

andmisel joonistab R tunnuse `kaal` histogrammi. Esimene vormindamist puudutav aspekt, mille reguleerimine käib Ri põhigraafikute puhul ühteoodi on telgede nimetused ja graafiku pealkiri. Horisontaaltelge reguleerib argument `xlab`, vertikaaltelge `ylab` ja pealkirja `main`[4]. Universaalne on ka argumentide `xlim` ja `ylim` kasutamine, mis määravad vastavalt horisontaal- ja vertikaaltelje algus- ja lõpppunkti[5].

Põhiline graafikute joonistamise tööriist on siiski geneeriline funktsioon `plot()`, mis omab ka argumenti `data` ja lubab kasutada mudeli süntaksit[6]. Võime joonistada tavalise hajuvusdiagrammi

```
> plot(kaal ~ pikkus, data=uus)
```

ent vajadusel saame kujutada ka ajaliselt seotud vaatlusi

```
> plot(weight ~ Time, data=ChickWeight, subset=(Chick == 1), type = "l")
```

Vaikimisi on andmete kujutamistüüpi määrav argument `type` väärtusega `"p"`, ent lisaks võivad olla kasulikud veel eelnähtud `"l"` või ka `"b"` (nii punktid kui ühendusjooned). Valikud `"s"` ja `"S"` annavad treppfunktsiooni ja `"h"` vertikaalsed jooned (tulbad). Logaritmilist skaalat saab tekitada argumendiga `log` väärtustega `"x"`,`"y"` või `"xy"`.

Kui me kirjutame pärast sümbolit `~` kategoorilise tunnuse joonistatakse karpdiagramm, kui ka enne sümbolit olev tunnus on kategooriline siis mosaiikdiagramm.

```
> plot(kaal ~ toit, data=uus)
> plot(toit ~ mootja, data=uus)
```

Juhul kui me soovime aga lihtsalt ühe tunnuse abil karpdiagrammi joonistada siis tuleb ikkagi kasutada funktsiooni `boxplot()` ning kui me ei soovi horisontaalteljel kujutada aega vaid lihtsalt vaatluse järjekorranumbrit, siis ei ole otstarbekas kasutada mudeli süntaksit[7].

Seega näiteks

```
> boxplot(uus$kaal)
> plot(ChickWeight$weight[1:12], type="l")
```

Andmetest algülevaate saamisel võib kasulik olla veel näiteks `interaction.plot()`, millele tuleb ette anda kaks kategoorilist ja üks pidev tunnus, mispeale kuvatakse pideva tunnuse keskmiste graafikud kategooriliste tunnuste väärtuste kombinatsioonide kaupa. Argumendi `fun` abil saab määrata ka, et kuvataks mitte keskmised vaid mõne muu funktsiooni väärtused (näiteks mediaan)

```
> interaction.plot(uus$toit, uus$grupp, uus$pikkus)
```

Otse andmemaatriksile võib rakendada funktsiooni `plot.design()`, ent see funktsioon võimaldab kasutada ka mudelisüntaksit ning analoogiliselt eelmisega saab funktsiooni, mida rühmadele rakendatakse ise määrata.

```
> plot.design(kaal ~ toit + mootja, data=uus, fun=median)
```

[4] Väärtustamise korral tuleb kõigi kolme argumendile anda väärtuseks sõne nagu näiteks `"x-telg"`

[5] Need kaks argumenti ootavad väärtusena kaheelemendilist arvulist vektorit nagu näiteks `c(3, 6)`

[6] Funktsioonil `hist()` on lisaks graafilisele ka tavaline väljund (tüüpi list). Seega on võimalik histogramm ka esmalt lihtsalt objektina salvestada ja hoopis hiljem funktsiooni `plot()` abil kuvada.

[7] probleemid tekivad kui kasutame lisaks argumenti `subset`

Tulpdiagramme saab joonistada funktsiooni `barplot()` abil. Kui me soovime lihtsalt ühe kategoorilise tunnuse sagedusi visualiseerida siis piisab sisendiks ühest sageduste vektorist, ent see võib tulla ka tabelist.

```
> barplot(xtabs( ~ toit, data=uus))
```

Kui sisend on kahemõõtmeline sagedustabel või maatriks, siis kuvatakse tulbad ühe tunnuse väärtuste põhjal grupeerituna. Kui lisaks määrata `beside=TRUE` siis paigutatakse tulbad üksteise kõrvale. Argument `horiz` vaikeväärtusega `FALSE` võimaldab määrata, mis suunalised tulbad olema peaksid.

```
> barplot(xtabs( ~ toit + grupp, data=uus), horiz=TRUE)
> barplot(xtabs( ~ toit + grupp, data=uus), beside=TRUE)
```

Huvitavad graafikute kuvamise funktsioonid on veel näiteks `pairs()`, mis joonistab andemaatriksi kõikvõimalike tunnusepaaride hajuvusgraafikud. Mudelisüntaksit võimaldab funktsioon `cdplot()`, mis kuvab kategoorilise tunnuse jaotuse muutumist pideva tunnuse muutudes[8]. Funktsioon `sunflowerplot()` tekitab hajuvusgraafiku, ent annab kattuvatest punktidest parema ülevaate, kuvades iga kattuvuse kohta ühe täiendava päevalille õielehe.

```
> pairs(uus)
> cdplot(toit ~ kaal, data=uus)
```

Ülesanded

III Mis juhtub joonega ühendatud punktide graafiku korral siis kui punktid ei ole järjestatud? Proovi näiteks

```
> plot(kaal ~ pikkus, type="l", data=uus)
```

Kuidas seda probleemi vältida?

IV Võta jadast `1 : 10` (tagasipanekuga) valim mahuga 100 elementi. Nüüd võta sama mahuga valim jadast `1 : 5`. Kujuta ette, et esimene valim annab punkti esimese koordinaadi ja teine teise koordinaadi. Kuva tekkinud punktid.

Graafikute lihtsam kohandamine

Hajuvusgraafikul on sageli vaja illustreerida hoopis kolme tunnust (kahte pidevat ja üht kategoorilist). Selleks peaks kategoorilise tunnuse erinevatele väärtustele vastavatel punktidel olema kas erinev värv, erinev sümbol või mõlemat. Erineva värvi saab anda argumendi `col` abil ning sümboli saab valida argumendiga `pch`. Kui me anname neile argumentidele ette lihtsalt ühe numbri, siis kasutatakse vastavat väärtust iga punkti jaoks.

```
> plot(kaal ~ pikkus, col=2, pch=3, data=uus)
```

[8] Põhimõtteliselt üritab sama ka `plot()` kui seletatav tunnus on kategooriline ja seletav pidev, ent `cdplot()` annab „silutud“ tulemuse.

Selleks, et iga punkt võiks olla erinev, peab ka värvi- või sümbolivektor olema sama pikkusega kui tunnusvektorid.

```
> plot(kaal ~ pikkus, col=as.numeric(toit), pch=as.numeric(mootja), data=uus)
```

Milline numbriline väärtus vastab millisele värvile või sümbolile? Sümbolite numbrite ja väärtuste vastavus on fikseeritud. Näiteks võime kirjutada

```
> plot(1 : 20, pch=1 : 20) #kas saad aru, miks kuvatakse just selline graafik?
```

nägemaks 20 esimest sümbolit. Värvide numbrite vastamist toonidele reguleerib funktsioon `palette()`, mis ilma argumentideta kuvabki hetkevastavuse numbrite ja toonide vahel ehk värvipaleti. Selle vastavuse võime aga ka ise määrata andes ette värvitoonide nimetuste vektori. Seda tehes tagastab funktsioon endise värvipaleti, mille saame nii mugavalt salvestada (juhul kui soovime seda veel hiljem kasutada). Millised on võimalikud värvid? Funktsioon `colors()` kuvab kõik Rile teada olevad värvid[9]. Neid värve võib ka ise sõnena otse argumendile `col` ette anda nii nagu argumendile `pch` võib ise ette anda (ühekohalisi) sümboleid sõnekujul.

Vahel on meil vaja graafikule objekte lisada. Lihtsamad lisandused on ilmselt (murd)jooned ja punktid. Esimeste lisamine olemasolevale graafikule käib funktsiooni `lines()` ja teiste lisamine funktsiooni `points()` abil. Kasutada saab mudelisüntaksit ja muidu on käsu andmine analoogiline funktsiooni `plot()` kasutamisele[10]. Seega näiteks

```
> plot(kaal ~ pikkus, subset=(toit == "a"), data=uus, pch="a", col="blue")
> points(kaal ~ pikkus, subset=(mootja == "E"), data=uus, pch="E", col="red")
> #mis läks halvasti?
> subset(uus, mootja == "E")
```

Graafikule täienduste tegemiseks võivad kasulikud olla ka funktsioonid `segments()` (etteantud punktide vahele joone tõmbamine), `arrows()` (noole joonistamine), `polygon()` (hulknurga joonistamine), `text()` (teksti lisamine), `rug()` (punktide jaotuse märkimine). Joonte korral on alati[11] võimalik määrata joone tüüp argumendiga `lty` (näiteks kriipsjoon või kriips-punktjoon) ja jämedus argumendiga `lwd` (vaikeväärtusega 1).

Ülesanded

V Vaata Rile tuntud värvide jadast järgi, millistel positsioonidel asuvad hallid toonid. Tekita sajast teineteisele järgnevast hallist toonist uus palett, salvestades samaaegselt vana värvipaleti. Kuva tekitatud värvid. Taasta endine värvipalett.

VI Tekita kaks alamandmestikku: üks nende isendite jaoks, kes kaaluvad üle 25 ühiku ja teised, kes alla. Kuva samal graafikul mõlema andmestiku tunnuste `kaal` ja `pikkus` hajuvusdiagramm, kusjuures esimeses andmestikus toidule `a` vastavad punktid värvi punaseks ja teises andmestikus toidule `c` vastavad punktid siniseks.

[9] aga soovi korral võime funktsiooni `rgb` abil ise värvid kokku segada

[10] ainult telgede seadistus on nüüd muidugi paigas ja seda muuta ei saa

[11] ka näiteks tavagraafiku korral, kus valime `type="l"`

Sirgjooni on graafikule mugav lisada ka funktsiooni `abline()` abil. Sellel on neli olulisemat argumenti, mida siinkohal vaatleme. Saamaks horisontaaljoont kasutame argumenti `h`, millele anname ette joonel paiknevate punktide vertikaaltelje koordinaadi. Analoogiliselt saame vertikaaljoone argumendi `v` abil. Argumentidega `a` ja `b` saame aga määrata joone algordinaadi ja tõusu. Kui anname väärtuse kõigile neljale argumendile siis joonistatakse kokku kolm joont.

Kuigi Ri graafiku vaikekujundus on küllalt mõistlik on vahel siiski vajalik objektide suurust muuta. Graafikul kujutatud sümbolite suhtelist suurust saab reguleerida argumendiga `cex`, telgede ühikute kuvamise suurust argumendiga `cex.axis`, telgede nimetuste suurust argumendiga `cex.lab`, graafiku pealkirja suurust `cex.main`. Kõik need argumendid ootavad arvulisi väärtusi ja vaikeväärtus on 1. Tegelikult on analoogiliselt suuruse muutmisega võimalik muuta ka värve (näiteks argument `col.lab` reguleerib telgede nimetuste värvi) ja ka kirjastiili (näiteks argument `font.lab` reguleerib telgede nimetuste kirjastiili)

```
> plot(kaal ~ pikkus, subset=(toit == "a"), data=uus, pch="a", col="blue",
         cex=1.5, font.lab=3, ylab="peis", xlab="sublimitas")
> abline(a=-0.4383, b=2.5327, lty=2, lwd=1.5)
```

Telgede paigutuse mugandamiseks tuleb üldjuhul esmalt telje (või telgede) joonistamine keelata valides, kas `xaxt="n"` või `yaxt="n"`, mispeale vastavalt horisontaaltelg või vertikaaltelg jäetakse joonistamata[12]. Pärast seda tuleb telg nüüd ise funktsiooni `axis()` abil joonistada. Sellel funktsioonil on kolm peamist argumenti: `side`, mis määrab, kuhu telg joonistatakse (võimalikud väärtused 1, 2, 3 ja 4, mis vastavad neljale küljele alustades alt ja liikudes päripäeva), `at`, mis määrab kriipsukeste asukohad ning `labels`, mis määrab kuvatavad sümbolid kriipsukeste juurde (kui jätame selle argumendi määramata, siis kasutatakse telje vastavaid väärtusi).

```
> plot(kaal ~ pikkus, subset=(mootja == "E"), data=uus, pch="E", col="red",
         cex=1.2, axes=FALSE, cex.lab=1.8, xlim=c(3, 17))
> axis(side=1, at=c(5, 10, 15), cex.axis=1.7, lwd=3)
> axis(side=2, at=c(10, 20, 30), cex.axis=1.7, lwd=3)
```

Sageli on graafikule vaja lisada ka legend. See käibki funktsiooni `legend()` abil. Argumentide `x` ja `y` abil saame anda legendi ülemise vasaku nurga koordinaadi. Aga kui valime hoopis `x=locator(1)`, siis saame ülanurga asukoha ise hiireklikiga määrata. Argument `legend` ootab väärtusena sõnevektorit ja määrab legendi teksti(d), ning näiteks `pch` abil saab määrata vastavad sümbolid (võib aga joonistada hoopis jooned `lty` abil) ja `col` abil sümbolite (või joonte) värvid.

```
> legend(x=locator(1), legend=c("mõõtja E"), col="red", pch="E")
```

[12] Sobib ka `axes=FALSE`, mis jätab mõlemad teljed joonistamata, ent lisaks jääb siis joonistamata ka ümbriskast

Juhuslikud suurused ja jaotused

Enne statistiliste testide juurde minekut vaatleme põgusalt tuntud tõenäosusjaotuste kasutamist Ris. Süsteem on lihtne – kõigi tuntud jaotuste jaoks on olemas neli funktsiooni algustähega `p`, `q`, `d` või `r`, millele järgneb jaotuse nimi (või selle lühend). Esimene on jaotusfunktsioon, teine kvantiilfunktsioon, kolmas tihedusfunktsioon ja neljas jaotusest juhuslikke arve genereerida võimaldav funktsioon. Näiteks normaaljaotuse puhul on vastavad funktsioonid

- `pnorm()`, mis tagastab tõenäosuse saada (kui me parameetreid ei muuda siis standardsest) normaaljaotusest väiksem arv kui meie poolt ette antu
- `qnorm()`, mis on eelmise pöördfunktsioon ja seega tagastab meie poolt sisestatud tõenäosusele vastava (standardse) normaaljaotuse kvantiili ehk arvu, millest väiksema arvu saame (standardsest) normaaljaotusest meie etteantud tõenäosusega.
- `dnorm()`, mis annab (standardse) normaaljaotuse tihedusfunktsiooni väärtuse antud kohal ehk väljendab antud väärtuse tõepära
- `rnorm()`, mis genereerib meie poolt ette antud koguse juhuslikke arve (standardsest) normaaljaotusest

Jaotuste vastavust funktsioonidele kirjeldab järgmine tabel.

jaotus	funktsiooninime lõpp
normaaljaotus	norm
β-jaotus	beta
binoomjaotus	binom
hii-ruut jaotus	chisq
eksponentjaotus	exp
F-jaotus	f
Γ-jaotus	gamma
geomeetriline jaotus	geom
hüpergeomeetriline jaotus	hyper
log-normaalne jaotus	lnorm
multinomiaalne jaotus	multinom
negatiivne binoomjaotus	nbinom
Poissoni jaotus	pois
t-jaotus	t
ühtlane jaotus	unif
Weibulli jaotus	weibull

Ülesanded

I Hundi rünnakuid analüüsides leidsime Spearmani korrelatsioonikordaja väärtuse $S = 0.20482$, kusjuures $n = 7$. Leia korrelatsioonikordajale vastav p-väärtus kui teame, et statistik

$$S\sqrt{\frac{n-2}{1-S^2}}$$

on nullhüpoteesi kehtides ligikaudu t-jaotusega vabadusastmete arvuga $n-2$. Kas kisklusrünnakute ja marutaudirünnakute arvu vahel piirkonniti on seos?

II Üks rusikareegel on, et absoluutväärtuselt 2.5 ületav (normeeritud) jääk viitab võimalikule probleemile mudeli juures. Leia kui suur on tõenäosus, et mõni standardsest normaaljaotusest pärit jääk on absoluutväärtuselt suurem kui 2.5 juhul kui meil on valim mahuga 100. Aga kui meil on valim mahuga 300? Kuivõrd põhjendatud on see rusikareegel nende valimimahtude korral?

Keskmiste võrdlemine[1]

Lihtsamad viisid gruppide keskmiste võrdlemiseks on t-test või siis selle mitteparameetriline analoog Kruskal-Wallise test (kahe grupi jaoks eksisteerib ka Mann-Whitney-Wilcoxoni mitteparameetriline test, mille üldistus Kruskal-Wallise test on). Üpris intuitiivselt on neid teste võimaldavate funktsioonide nimetused Ris `t.test()`, `kruskal.test()` ja `wilcox.test()`. Kõigil juhtudel saab kasutada mudeli süntaksit: vasakule poole sümbolit `~` kirjutame uuritava tunnuse nime ja paremale poole grupeeriva tunnuse.

Olulisemad funktsiooni `t.test()` argumendid on veel ühe- ja kahepoolse sisuka hüpoteesi vahel valida võimaldav `alternative` võimalike väärtustega `"two.sided"`, `"less"`, `"greater"` (vaikeväärtus on liithüpotees ehk valik `"two.sided"`), keskmistevaheline erinevus `mu`, mis vastab nullhüpoteesile (vaikimisi `mu=0`). Samuti saame valida, kas tegu on kordusmõõtmistega (argument `paired` vaikeväärtusega `FALSE`), kas dispersioonid on mõlemas grupi samad (argument `var.equal` vaikeväärtusega `FALSE`) ning millise olulisusnivoo korral soovime leida usaldusvahemiku keskmisele või keskmiste vahele (argument `conf.level` vaikeväärtusega `0.95`). Samasugused argumendid (välja arvatud `var.equal`, sest eeldatakse dispersioonide võrdsust) on olemas ka funktsioonil `wilcox.test()`. Puuduvad väärtused jäetakse vaikimisi kõigi kolme testi korral analüüsist välja. Vahel, kui paarikaupa võrdlusi on vaja teha palju, on mõistlik kasutada funktsiooni `pairwise.t.test()` või `pariwise.wilcox.test()`. Need funktsioonid ei võimalda küll mudeli süntaksit ja tagastatakse ainult p-väärtused, ent samas saab p-väärtusi sobivalt korrigeerida[2] argumendi `p.adjust.method` abil.

Ülesanded

III Lae enda arvutisse Ri pakett, mis käib kaasas raamatuga *"Mixed Effects Models and Extensions in Ecology with R" (Zuur et al., 2009)* aadressilt

`http://www.highstat.com/Book2/AED_1.0.zip`

ja installeeri see. Lae pakett tööle ja lae paketist omakorda andmestik `Boar`. Vaata andmestikku. Andmestikus on neli metssigu kirjeldavat tunnust: `Tb`, mis näitab tuberkuloosi olemasolu isendil, `SEX`, mis näitab isendi sugu, `AgeClass`, mis näitab isendi vanusklassi ning `LengthCT`, mis on isendi pikkus ninast sabani (mõõdetuna mööda selga). Testi keskmise pikkuse sõltuvust soost.

IV Uuri ka pikkuse jaotumist vanusegrupiti. Kas siin paistab loogiline seos? Mida ütleb statistiline test? Vaata, milline on testi poolt tagastatud objekt ja salvesta testi tagastatud p-väärtus eraldi objektina.

[1] dispersioonanalüüsi vaatame lineaarse mudeli erijuhuna hiljem
[2] mis on vajalik tänu korduvale testimisele

Testid sagedustabelitele[3]

Hii-ruut testiga kontrollitakse mitmeid erinevaid hüpoteese. Kõige levinum on ilmselt kahemõõtmelise sagedustabeli põhjal tunnustevahelise seose olemasolu või puudumise kontrollimine. Seda ja ka lihtsalt diskreetse jaotuse vastavust oodatule saab kontrollida funktsiooni `chisq.test()` abil. Kahemõõtmelise sagedustabeli korral (sisend on maatriks või ka näiteks funktsiooni `xtabs()` poolt tagastatud sagedustabel) piisab ühest sisendist.

Näiteks eelnevalt vaadatud toidukatse andmete puhul

```
> chisq.test(xtabs( ~ mootja + toit, data=uus))
        Pearson's Chi-squared test

data:  xtabs(~mootja + toit, data = uus)
X-squared = 31.2, df = 4, p-value = 2.787e-06
```

R hoiatab, kui oodatav vaatluste arv mingis kategoorias on liialt väike. Märkimist väärib ka, et 2x2 sagedustabeli korral kasutatakse vaikimisi (nn Yates'i) pidevusparandust.

Funktsioonile `chisq.test()` vektorit ette andes soovime kontrollida empiirilise sagedusjaotuse vastavust oodatavale jaotusele. Kui me rohkem argumente ette ei anna, siis eeldatakse, et oodatav jaotus on ühtlane – kõigis klassides peaks keksmiselt olema samapalju vaatlusi. Üldjuhul me sellist testi teha ei soovi ja siis vajame lisaks mõõdetud sagedustele ka oodatavaid sagedusi, mille anname argumendi `p` abil (tegelikult oodatakse `p` väärtusena tõenäosuste vektorit, ent kui lisaks määrata `rescale.p=TRUE`, siis muudetakse oodatud sagedused Ri poolt ise tõenäosusteks). Väärib märkimist, et tegelikult saame nii testida hüpoteese ka mitmemõõtmelise diskreetse jaotuse kohta.

```
> sagedused = table(Boar$AgeClass)
> sum(sagedused)
[1] 650
> oodati = c(30, 150, 250, 220)
> chisq.test(sagedused, p=oodati, rescale.p=TRUE)
        Chi-squared test for given probabilities

data:  sagedused
X-squared = 17.763, df = 3, p-value = 0.0004922
```

Kui sagedustabeli äärejaotused on meie kontrolli all, siis on seose olemasolu kontrollil alternatiiviks näiteks sarnase kasutuspõhimõttega funktsioon `fisher.test()`. Näiteks G-testi ja Barnardi testi kood Ri jaoks on samuti olemas, ent mitte osa Ri standardpaketist. Küll aga saab Cochran-Mantel-Haenszeli[4] testi teha funktsiooni `mantelhaen.test()` abil. Sisendiks on siin kolmemõõtmeline sagedustabel, kusjuures viimane tunnus peaks olema see, mis meile huvi ei paku, ent mis tekitab sagedustabelite korduse.

[3] Nägime juba eelnevalt, et funktsioon `cor.test()` võimaldab testida Pearsoni, Spearmani või Kendalli korrelatsioonikordaja statistilist olulisust. Seetõttu vaatleme siin alapunktis vaid kategooriliste tunnuste vaheliste seoste olemasolu testimist.

[4] sarnaselt tavalisele hii-ruut testile pakub siin huvi shansside suhte (*odds ratio*) võrdumine ühega ehk tunnustevaheline sõltumatus, ent kuivõrd siin on „kahemõõtmelisi sagedustableid mitu" siis on selle testi juures eelduseks ka, et shansside suhe on kõigil kordadel võrdne

```
> mantelhaen.test(xtabs( ~ SEX + Tb + AgeClass, data=Boar))
        Mantel-Haenszel chi-squared test with continuity correction

data:  xtabs(~SEX + Tb + AgeClass, data = Boar)
Mantel-Haenszel X-squared = 2.4812, df = 1, p-value = 0.1152
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
 0.520205 1.053652
sample estimates:
common odds ratio
        0.7403481
```

Hii-ruut statistikul põhineb ka funktsiooni `prop.test()` töö, mis võimaldab kontrollida, kas kõikides gruppides jaotub binaarne tunnus samas suhtes (ehk siis kontrollida sedasama, mida võimaldab `chisq.test()` kitsendusega, et üks tunnus on binaarne), ent lisaks saab kontrollida ka jaotumise vastamist etteantud jaotumisskeemile (võib grupiti olla erinev), mida saab ette anda argumendi `p` abil (argument ootab väärtuseks vektorit tõenäosustega). Sellisel juhul kombineerime mitu küsimust ühte testi ehk teeme omnibustesti – nullhüpoteesi kummutamine näitab, et vähemalt ühes grupis on asjad (suure tõenäosusega) paigast ära. Sisendiks sobib siingi funktsiooni `xtabs()` poolt tagastatud sagedustabel, kusjuures grupid tekitav tunnus (mitte 0/1 tunnus) peaks olema antud esimesena (muidu tuleb sagedustabel esmalt transponeerida). Kui me soovime ikkagi proportsioone paarikaupa võrrelda, siis saab seda teha funktsiooni `pairwise.prop.test()` abil, mis lubab ka p-väärtuste mitmese võrdluse korrektsiooni.

Ülesanded

V Testi metssigade andmestiku põhjal, kas kõigi vanusklasside korral on tuberkuloosihaigeid sigu 50%.

VI Veel üks huvitav hii-ruut statistikul põhinev test on McNemari test, mida saab teha funktsiooni `mcnemar.test()` abil. Uuri abifailist, milleks see test mõeldud on. Uuri ka näidet. Kas suudad tuua näite mõnest bioloogiaga seotud probleemist, kus selline test kasulik võib olla?

Eelduste kontrollimine

Küllalt sageli on statistiliste analüüside eelduseks, et tunnused peavad olema (näiteks gruppides) normaaljaotusega või olema homoskedastilised (kõikides gruppides sama dispersioon).

Normaaljaotuse eeldust on võimalik kontrollida joonistades näiteks kvantiil-kvantiilgraafiku funktsiooni `qqnorm()` abil. Võib ka läbi viia Shapiro-Wilksi normaalsuse testi funktsiooni `shapiro.test()` abil. Levinuim meetod, mida saab kasutada pidevate jaotuste korral, on ilmselt siiski Kolmogorov-Smirnovi test, mille viib läbi funktsioon `ks.test()`. Selle testi korral saame ise määrata, millise jaotusega vastavust me kontrollime. Seejuures võime võrdlemiseks kasutatava jaotusfunktsiooni võtta ka esimeses alapunktis mainitud jaotusfunktsioonide hulgast. Jaotuse parameetrid tuleb muidugi ette anda[5]. Näi-

[5]Seda testi saab kasutada ka testimiseks, kas kaks valimit pärinevad samast jaotusest. Sellisel juhul ei ole vaja määratleda, millisest jaotusest konkreetselt.

teks võime kontrollida, kas pikkusmõõdu jaotus sõltub ka soost, andes käsu

```
> ks.test(Boar$LengthCT[Boar$SEX == 1], Boar$LengthCT[Boar$SEX == 2])
        Two-sample Kolmogorov-Smirnov test

data:  Boar$LengthCT[Boar$SEX == 1] and Boar$LengthCT[Boar$SEX == 2]
D = 0.0956, p-value = 0.1882
alternative hypothesis: two-sided
```

ning näeme, et antud juhul seda väita ei saa. Lisaks tuleb veel tähelepanu pöörata, et korduvate väärtuste korral valimis ei pruugi leitav p-väärtus olla täpne (pideva jaotuse korral meil tegelikult korduvaid väärtusi esineda ei saa). Ohtlik on see väikese valimi korral.

Kontrollimaks, kas pikkusmõõdu jaotus on normaaljaotus keskväärtusega 120 ja standardhälbega 30 anname käsu

```
> ks.test(Boar$LengthCT, pnorm, mean=120, sd=30)
        One-sample Kolmogorov-Smirnov test

data:  Boar$LengthCT
D = 0.2034, p-value < 2.2e-16
alternative hypothesis: two-sided
```

ja näeme, et sellise jaotusega antud juhul küll tegemist ei ole.

Põhimõtteliselt on võimalik kontrollida ka ühepoolset hüpoteesi (argument `alternative` vaikeväärtusega `"two.sided"`), näiteks `"greater"` korral baseerub teststatistik valimi ja nullhüpoteesile vastava jaotuse jaotusfunktsioonide vahel – seega on sellise olukorra üheks praktiliseks vasteks olukord, kus jaotus, kust pärineb valim, on nihutatud vasakule (ehk siis "väiksem").

Üks aspekt, mille vastu praktikas sageli eksitakse, on Kolmogorov-Smirnovi testi korral veel referentsjaotuse parameetrite määramine – ei ole aktsepteeritav, et leiame esmalt valimi põhjal aritmeetilise keskmise ja standardhälbe ning seejärel kontrollime, kas valim pärineb selliste parameetritega normaaljaotusest. Nii saame tüüpiliselt põhjendamatult suure p-väärtuse. Lahenduseks on siis valimi kaheks jagamine – esimese osa põhjal hindame parameetrid, teise põhjal testime.

Dispersioonide võrdsust gruppides võimaldab testida päris mitu funktsiooni. Neist `var.test()` (kahe grupi jaoks ja seetõttu ka ühepoolset hüpoteesi lubav) ja `bartlett.test()` (kahe või enama grupi jaoks) on kasutatavad normaaljaotuse eeldusel, samas kui astakutel põhinev `fligner.test()` (samuti kahe või enama grupi jaoks) sellist eeldust ei tee. Kõigi kolme puhul on kasutatav mudeli süntaks.

```
> fligner.test(LengthCT ~ AgeClass, data=Boar)
        Fligner-Killeen test of homogeneity of variances

data:  LengthCT by AgeClass
Fligner-Killeen:med chi-squared = 19.4108, df = 3, p-value = 0.0002248
```

Lisaks eksisteerivad veel jaotusvabad `ansari.test()` ja `mood.test()`, mis võimaldavad kahe grupi korral kontrollida hüpoteesi, et mõlemad valimid on pärit samast jaotusklassist, ent omavad erinevat

skaalaparameetrit[6]. Süntaks on analoogiline eelnevatega. Mõlemal juhul on võimalik kontrollida ka ühepoolset hüpoteesi.

Ülesanded

VII Leia andmestiku uus korral, kas faktortunnuste `toit` ja `grupp` tasemete lõikes on tunnus `kaal` sama normaaljaotusega. Kas see tähendab, et saame läbi viia kahefaktorilise dispersioonanalüüsi?

VIII Kontrolli tunnuse `kaal` normaalsust tunnuse `toit` põhjal moodustatud gruppides visuaalselt.

[6] sellele vastab näiteks olukord, kus mõlemad valimid on küll pärit sama keskväärtusega normaaljaotusest, ent hajuvus (dispersioon) on mõlemal erinev

Erinevate mudelite sobitamisel on Ris siiski üks ühine põhimõte – sobitatud mudel tuleks objektina salvestada. Sinna objekti salvestatakse kõik oluline, mis mudeli juurde kuulub, ning hiljem saab sealt abifunktsioonide abil infot eraldada[1].

Lineaarne mudel

Kuigi tavaline lineaarne mudel on mugavalt käsitletav ühe teemana, on Ris säilinud eraldi viis dispersioonanalüüsi ja regressioonanalüüsi tegemiseks. Meie kasutame siin siiski vaid funktsiooni `lm()`, mis võimaldab mõlemat edukalt läbi viia ning vajadusel saab kätte ka dispersioonanalüüsi tabeli.

Põhimõtteliselt on mudeli sobitamine väga lihtne[2]. Näiteks

```
> m1 = lm(kaal ~ pikkus + toit, data=uus)
> summary(m1)
Call:
lm(formula = kaal ~ pikkus + toit, data = uus)

Residuals:
    Min      1Q  Median      3Q     Max
-4.1079 -1.4891  0.0128  1.6849  3.9212

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.4690     2.0166  -0.233    0.818
pikkus        2.5354     0.1609  15.754 4.16e-13 ***
toitb         0.1046     1.1006   0.095    0.925
toitc        -0.3802     1.1832  -0.321    0.751
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.254 on 21 degrees of freedom
  (5 observations deleted due to missingness)
Multiple R-squared: 0.9301,        Adjusted R-squared: 0.9201
F-statistic: 93.13 on 3 and 21 DF,  p-value: 2.701e-12
```

kus geneeriline funktsioon `summary()` on tüüpiliselt see, mida pärast mudeli sobitamist, põgusa ülevaate saamiseks, objektile rakendada soovime. Antud ülevaade sisaldab muuhulgas olulisustõenäosusi, mis pidevate tunnuste korral langevad kokku nn kolmandat tüüpi summadel põhinevate olulisustõenäosustega[3].

Milliseid funktsioone veel sobitatud mudelile rakendada võib? Dispersioonanalüüsi tabeli tüüpi väljundi annab funktsioon `anova()`. Parameetrite punkthinnangud väljastab `coef()` ja usaldusvahemikud

[1] et salvestatud mudel on üldiselt ikka listi tüüpi objekt siis muidugi saame infot kätte ka listist ise õigest kohast elemente välja kutsudes

[2] kui me kirjutame paremale poole sümbolit ~ sümboli . siis on selle tähenduseks „kõik tunnused andmestikus välja arvatud need, mida modelleerime“. See võimaldab palju tunnuseid küllalt lühikese käsuga mudelisse haarata, sest sümboli - abil saab nimistust omakorda tunnuseid välja jätta

[3] seega väljendavad argumendi olulisust tingimusel, et kõik ülejäänud argumendid on mudelisse võetud

`confint()`. Sobitatud väärtused tagastab `fitted()` ja tegelike väärtuste ning sobitatud väärtuste vahed (ehk mudeli vead) `resid()`. Graafilise ülevaate mitmetest mudeli diagnostikaks kasutatavatest väärtustest annab funktsioon `plot()`[4]. Mudeli logaritmilise tõepära väljastab `logLik()` ning selle põhjal saame ise arvutada näiteks informatsioonikriteeriume.

Mudeldamisest üldiselt

Räägime siin veel paarist mudeldamise aspektist. Esmalt mudeli uuendamine, milleks kasutame funktsiooni `update()`. Esimeseks sisendiks on mudel, mida uuendama asume ja teiseks tehtavad muutused. Teist argumenti väärtustades saab kasutada sümbolit `.`, mille tähenduseks on siin „kõik, mis oli eelnevalt mudelis“. Seega kuju `.~.` tähendab eelnevat mudelit, ning sinna võime nüüd `+` abil uusi tunnuseid juurde panna või siis `-` abil ära võtta. Näiteks

```
> m2 = update(m1, . ~ . - toit)
> m3 = update(m1, . ~ . - 1) #jätame vabaliikme välja
> summary(m2)

Call:
lm(formula = kaal ~ pikkus, data = uus)

Residuals:
    Min      1Q  Median      3Q     Max
-4.4630 -1.6826  0.2374  1.8389  4.1029

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.7550     1.6081   -0.47    0.643
pikkus        2.5545     0.1468   17.41 9.77e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.164 on 23 degrees of freedom
  (5 observations deleted due to missingness)
Multiple R-squared: 0.9294,        Adjusted R-squared: 0.9264
F-statistic: 302.9 on 1 and 23 DF,  p-value: 9.767e-15
```

Teine kasulik rakendus on funktsiooni `anova()` kasutamine mitme (reeglina siiski kahe) argumendi korral. Sellisel juhul vaadeldakse mõne summaarse mudeli headust peegeldava statistiku (näiteks mudeli vigade ruutude summa või mudeli hälbimus) muutumist ja testitakse keerulisema mudeli täiendavate parameetrite olulisust. Seega saame nii kontrollida ka, kas „faktor tervikuna“ on oluline[5].

Asjaolu, et kõik mudelid on mingi keerulisema mudeli (mis samuti võrdlusesse kaasatakse) erijuht, on üks tingimus selleks, et mudeleid nii võrrelda saaksime. Teiseks tuleb muidugi tagada, et mudelid oleksid ikka sobitatud täpselt samadele andmetele (see on eriti oluline puuduvate väärtuste korral!). Reeglina peame siiski ise teadma, mis tingimustel me mudeleid võrrelda saame, sest `anova()` on teinekord nõus tegema ka selliseid võrdlusi, mis tegelikult mingit mõtet ei oma.

[4] nende ja muude mudeli diagnostikavõimaluste kohta vaata funktsiooni `influence.measures()` abifailist

[5] selleks saab kasutada ka funktsiooni `drop1()`

Antud juhul on kõigi kolme mudeli puhul kasutatavad vaatlused samad (kõigil juhtudel on samad 5 valimi vaatlust on välja jäetud). Näiteks võime siis testida

```
> anova(m1, m2)
Analysis of Variance Table

Model 1: kaal ~ pikkus + toit
Model 2: kaal ~ pikkus
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     21 106.69
2     23 107.69 -2  -0.99872 0.0983 0.9068
```

ja näeme, et faktor `toit` ei ole oluline.

Mudeli väljavõttest oleme juba näinud, et R kasutab faktorite puhul vaikimisi kontraste, kus baastasemeks on faktori (reeglina tähestikuliselt kui me pole ise tasemeid muutnud) esimene tase. See, milliseid kontraste kasutame omab olulist tähendust, sest sellest sõltub, mida parameeter, mille nullist erinemist me kontrollime, reaalselt väljendab. Nii näiteks on mudelis `m1` parameetri *(Intercept)* tähenduseks kaalu keskmine väärtus toidutüübi „a“ korral ehk siis vabaliige pluss toidu „a“ mõju. Selles mudelis väljendab *toitb* kordaja keskmiste tasemete erinemist toidutüübi „b“ ja toidutüübi „a“ vahel (sama tunnuse `pikkus` väärtuse korral). Mudelis `m3` aga vabaliiget pole ning *toitb* väljendab kaalu keskmist väärtust toidutüübi „b“ korral. Seda, mida enne väljendas *(Intercept)* väljendab nüüd *toita*. Kontrastide muutmiseks saab kasutada argumenti `contrasts`, mille väärtuseks tuleb anda list, kus elementide nimed vastavad faktorite nimedele ja võimalikud väärtused leiame funktsiooni `contrasts()` abifailist. Nii näiteks käsk

```
> update(m1, . ~ ., contrasts = list(toit="contr.SAS"))
Call:
lm(formula = kaal ~ pikkus + toit, data = uus, contrasts = list(toit = "contr.SAS"))

Coefficients:
(Intercept)       pikkus        toita        toitb
    -0.8492       2.5354       0.3802       0.4848
```

seab baastasemeks faktori viimase taseme nagu tarkvara SAS korral vaikimisi.

Ülesanded

I Sobita metssigade andmestikule lineaarne mudel ja selgita tulemusi. Leia mudeli parameetritele usaldusvahemikud. Kas kõigis vanuseklassides tundub sigadel olevat erinev pikkus? Kuidas selgitad, et tunnus `Tb` on üksinda mudelis oluline?

II Oletame nüüd, et meil on mudel sobitatud ning seletavate tunnustena kasutame sugu ja vanust. Me soovime sobitatud mudeli põhjal ennustusi teha. Vaata, kuidas kasutada funktsiooni `predict()` ja ennusta vanusklassi 3 kuuluva emase metssea pikkust küsides muuhulgas 99%-usaldusvahemikku. Mitu sellise tunnuste väärtustekombinatsiooniga siga on andmestikus. Kui paljudel neist kuulub pikkusmõõt ennustatud vahemikku?

Üldistatud lineaarne mudel

Põhifunktsiooniks on siin `glm()`, mille süntaks on analoogiline funktsiooniga `lm()`. Erinevuseks on, et argumendi `family` abil saame määrata ka vaatluste (tingliku) jaotuse ja seosefunktsiooni. Võimalikud jaotuse valikud on `binomial`, `Gamma`, `inverse.gaussian`, `poisson`, `quasibinomial` ja `quasipoisson`. Abifunktsioonid, mida mudelile rakendada võib on üldjuhul needsamad, mis tavalise lineaarse mudeli korral. Üheks erinevuseks on, et funktsioon `anova()` vajab kindlasti ka argumendi `test` väärtustamist.

```
> ylekaal = (uus$kaal > 30) * 1 #muudame kaalu binaarseks
> uus = cbind(uus, ylekaal)
> m3 = glm(ylekaal ~ pikkus + toit, data=uus, family=binomial)
> summary(m3)

Call:
glm(formula = ylekaal ~ pikkus + toit, family = binomial, data = uus)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.71998  -0.25337  -0.09343   0.02053   1.80861

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -19.0410     8.4155  -2.263   0.0237 *
pikkus        1.5435     0.6834   2.259   0.0239 *
toitb         0.8003     1.9544   0.410   0.6822
toitc        -0.7766     2.0888  -0.372   0.7100
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 29.6477  on 24  degrees of freedom
Residual deviance:  9.9779  on 21  degrees of freedom
  (5 observations deleted due to missingness)
AIC: 17.978

Number of Fisher Scoring iterations: 7
```

Ennustamisel on funktsioonile `predict()` reeglina põhjust määrata `type="response"`, sest siis ennustatakse tulemusi originaalskaalal.

Ülesanded

III Kas äsjasobitatud mudelis on faktor `toit` oluline?

IV Kui suur on mudeli kohaselt tõenäosus, et vanusklassi 3 kuuluva emase metsseal on tuberkuloos? Millise tunnustekombinatsiooni korral on see tõenäosus suurim? Miks?

Korreleeritud vaatlustega lineaarne mudel

Sageli oleme mõnd subjekti mõõtnud korduvalt, objektid on ruumiliselt lähedal (ja lähedasemad objektid on sarnasemad) või siis on näiteks erinevate vaatluste korral eeldatav mõõtetäpsus erinev. Kõigil neil juhtudel on selge, et pole põhjust eeldada, et vaatluste (tinglik) dispersioonimaatriks oleks lihtsalt ühikmaatriksi kordne. Sellisel juhul saame kasutada paketist `nlme`[6] pärit funktsiooni `gls()`, mille kaks olulist argumenti on `correlation` ja `weights` vastavalt korrelatsioonistruktuuri ja dispersiooni modelleerimiseks. Võimalikest valikutest ja täpsest süntaksist saab ülevaate vastavalt käskudega `?corClasses` ja `?varClasses`. Lisaks on võimalik valida ka mudeli parameetrite hindamise meetod argumendi `method` abil. Vaikeväärtus `"REML"` tagab küll reeglina väiksema nihkega parameetrite hinnangud, ent samas pole siis üldjuhul kasutatavad tõepäral põhinevad informatsioonikriteeriumid mudeli valikul. Alternatiiviks on valik `"ML"`.

Vaatame siinkohal argumentide `weights` ja `correlation` väärtustamist ainult õige põgusalt.

väärtuse kuju	dispersioon kasvab
`varFixed(~tunnus)`	lineaarselt koos tunnuse väärtusega
`varPower(form=~tunnus)`	polünomiaalselt koos tunnuse väärtusega
`varPower(form=~tunnus|grupeeriv tunnus)`	ja igas grupis erinevalt
`varExp(form=~tunnus)`	eksponentsiaalselt koos tunnuse väärtusega
`varExp(form=~tunnus|grupeeriv tunnus)`	ja igas grupis erinevalt
`varIdent(form=~1|grupeeriv tunnus)`	konstantne grupi sees

Kõigil juhtudel peale esimese ja viimase võib kasutada `~tunnus` asemel ka `~fitted(.)`, mis vastab olukorrale, kus dispersioon kasvab koos sobitatud väärtusega.

väärtuse kuju	korrelatsioon on
`corCompSymm(form=~1)`	konstantne üle kogu valimi
`corCompSymm(form=~1|grupeeriv tunnus)`	konstantne grupi sees
`corSymm(form=~tunnus|grupeeriv tunnus)`	suvaline
`corAR1(form=~tunnus|grupeeriv tunnus)`	absoluutväärtuselt eksponentsiaalselt kahanev
`corCAR1(form=~tunnus|grupeeriv tunnus)`	sama pideva aja jaoks

```
> m4 = gls(kaal ~ pikkus, data=uus, na.action="na.omit",
         correlation=corCompSymm(form=~1|toit))
> summary(m4)
Generalized least squares fit by REML
  Model: kaal ~ pikkus
  Data: uus
       AIC      BIC    logLik
```

[6] Ehkki väga suuri võimalusi pakkuv pakett, on selle funktsioone kasutades tarvis olla eriti ettevaatlik puuduvate väärtustega, mis kipuvad põhjustama veateateid ja muid probleeme. Kui on ette teada, millised tunnused kasutusse tulevad, siis on kõige õigem tekitada esmalt uus andmestik koos ainult vajalike tunnustega ning seejärel eemaldada uuest andmestikust need valimi elemendid, mis sisaldavad puuduvaid väärtusi. Selliselt puhastatud andmestikuga probleeme ei teki.

```
  114.6434 119.1854 -53.32172

Correlation Structure: Compound symmetry
 Formula: ~1 | toit
 Parameter estimate(s):
       Rho
-0.1218077

Coefficients:
                Value Std.Error   t-value p-value
(Intercept) -1.548736 1.2572318 -1.231862  0.2304
pikkus       2.639247 0.1167803 22.600097  0.0000

 Correlation:
       (Intr)
pikkus -0.997

Standardized residuals:
         Min           Q1          Med           Q3          Max
-2.282934314 -0.670050645 -0.004625792  0.829367821  2.098682017

Residual standard error: 2.113156
Degrees of freedom: 25 total; 23 residual
> m5 = update(m4, weights=varIdent(form=~1|toit))
> anova(m4, m5)
   Model df      AIC      BIC    logLik   Test  L.Ratio p-value
m4     1  4 114.6434 119.1854 -53.32172
m5     2  6 114.3414 121.1544 -51.17071 1 vs 2 4.302007  0.1164
```

Valiku `corAR1` korral peab `tunnus` omama täisarvulisi väärtusi ja selle väärtuste vahe väljendab vaatluste sammulist kaugust (grupi sees). Kui `~tunnus` asendada valikuga `~1` siis vastab see olukorrale, kus järjestuse määrab vaatluste järjekord valimis. Valiku `corCAR1` korral ei pea `tunnus` omama täisarvulisi väärtusi (ent võimalikud pole ka negatiivsed korrelatsioonid). Lisaks eksisteerivad veel nn ruumilise autokorrelatsiooni valikud `corExp`, `corGaus`, `corLin`, mille korral `form` võib sisaldada ka mitut tunnust, ning nende tunnuste väärtused määravad objektidevahelise kauguse (meetrika `metric` põhjal). Valiku `corExp` korral kirjeldab korrelatsiooni muutumist kauguse muutudes eksponentfunktsioon, `corGaus` korral eksponentfunktsioon argumendi ruudust ning `corLin` korral lineaarfunktsioon. Lisaks eksisteerib veel tõeväärtusega valik `nugget` (vaikeväärtus `FALSE`), mis määrab, kas korrelatsioon kasvab objektide järjestikusel lähenemisel üheni või ainult mingi piirini (mis hinnatakse).

Seega käsk

```
> gls(kaal ~ pikkus, data=andmed, na.action="na.omit", method="ML",
         correlation=corAR1(form=~as.numeric(grupp)|toit))
```

vastab mudelile, kus objektide järjestuse (kauguse teineteisest) defineerib tunnus `grupp` (täisarvuliseks muudetuna) ja käsk

```
> gls(kaal ~ pikkus, data=andmed, na.action="na.omit", method="ML",
         correlation=corExp(form=~x+y, nugget=TRUE))
```

mudelile, kus vaatlustevahelist korrelatsiooni kirjeldab funktsioon kujul

$$(1-n)\cdot\exp\left(-\frac{d}{r}\right),$$

kus n on hinnatav parameeter nimega ehe variatsioon (*nugget*) ja r hinnatav parameeter nimega ulatus (*range*) ning d on vaatlustevaheline (eukleidiline) kaugus, mis põhineb koordinaatidel x ja y.

Lineaarne segamudel

Siin käib töö samuti paketi `nlme` funktsiooniga, mis sedakorda kannab nime `lme()`. Saab kasutada ka eelmises punktis mainitud valikuid argumentide `weights` ja `correlation` abil, ent lisandub veel argument `random`, mille väärtustamiseks on veidi ebatavaline süntaks. Sümboli `|` järel väljendatakse grupeerivat tunnust[7]. Seega sama grupeeriva tunnuse väärtuse korral on objektidel ka samad juhuslike efektide väärtused. Sümbolile `|` eelnevad tunnused, millele kordajatele juhuslikud efektid lisanduvad[8]. Abifunktsioonidest, mida mudelile rakendada lisanduvad `fixef()` ja `ranef()`, funktsiooni `confint()` vahetab välja funktsioon `intervals()`.

```
> m6 = lme(kaal ~ pikkus, data=uus, na.action="na.omit",
         random=~1|grupp)
> summary(m6)
Linear mixed-effects model fit by REML
 Data: uus
       AIC      BIC    logLik
  117.3779 121.9199 -54.68897

Random effects:
 Formula: ~1 | grupp
         (Intercept) Residual
StdDev: 0.0001143537 2.163823

Fixed effects: kaal ~ pikkus
                 Value Std.Error DF   t-value p-value
(Intercept) -0.7550164 1.6080778 14 -0.469515  0.6459
pikkus       2.5545105 0.1467676 14 17.405135  0.0000
 Correlation:
       (Intr)
pikkus -0.963

Standardized Within-Group Residuals:
```

[7] või kui tegu on allutatud tunnustega siis ka tunnuseid, kusjuures allutamist tähistatakse sümboli `/` abil – vasakul pool ülemine ja paremal pool allutatud tunnus

[8] Kui vasakul pool sümbolit `|` on ainult `1` siis tähendab see juhuslikku vabaliiget, ent vabaliige lisatakse alati vaikimisi kui seda seal ei ole.

```
       Min         Q1        Med         Q3        Max
-2.0625352 -0.7775983  0.1097181  0.8498455  1.8961504

Number of Observations: 25
Number of Groups: 10
```

Kas saad aru miks järgnev mudel oleks antud andmestiku korral mõttetu?

```
> lme(kaal ~ toit, data=uus, na.action="na.omit",
         random=~1|grupp/toit)
```

Nii saame sobitada juhuslike tõusudega mudeli[9]:

```
> m7 = lme(kaal ~ pikkus, random=~pikkus|toit, na.action="na.omit",
         data=uus, method="ML")
> ranef(m7)
    (Intercept)        pikkus
a  6.589263e-10  1.644651e-24
b  2.067618e-09  5.017978e-24
c -2.726544e-09 -6.660682e-24
```

Üldistatud lineaarne segamudel

Sedavõrd üldiste mudelite sobitamiseks on Ris erinevaid võimalusi, ent levinuimaks tööriistaks on paketist `lme4` pärit funktsioon `lmer`[10] Argumendi `family` kasutamine on sama nagu eelnevalt kirjeldatud. Tavalise lineaarse segamudeli korral reguleerib tõeväärtusega argument `REML` sama, mida eelnevalt `method`. Üldistatud lineaarse segamudeli korral määrab argument `nAGQ` kasutatava lähendi täpsuse[11] Mõneti intuitiivsem on juhuslike faktorite sisestamine mudelisse – need tuleb lihtsalt sulgude vahele paigutada. Kõik erinevates sulgudes paiknevad faktorid loetakse mittekorreleerituks.

Mudelile *m6* vastab seega käsk

```
> m8 = lmer(kaal ~ pikkus + (1|grupp), na.action="na.omit", data=uus)
> summary(m8)
Linear mixed model fit by REML
Formula: kaal ~ pikkus + (1 | grupp)
   Data: uus
   AIC   BIC logLik deviance REMLdev
 117.4 122.3 -54.69    107.5   109.4
Random effects:
 Groups   Name        Variance Std.Dev.
```

[9] juhuslike tõusude all mõtleme, et iga grupi (ehk `toit` väärtuse) korral on lisaks vabaliikmele ka tunnuse `pikkus` kordaja erinev

[10] Kuigi `lmer` on veel veidi lihvimata produkt (kogu planeeritud funktsionaalsus ei tööta) on olulisemad jaotusklassid kasutatavad ning parameetrite hindamise meetodid hetkel kasutatavaist täpseimad. Muuhulgas aga tähendab see, et juhuslikke efekte (mitte tasemeid) ei tohi olla väga palju – muidu lahendusmeetod ei koondu.

[11] väärtuseks sobivad positiivsed täisarvud, mida suurem väärtus, seda kvaliteetsem (ent aeglasem) on tõepära lähendamine

```
 grupp    (Intercept) 0.0000   0.0000
 Residual             4.6821   2.1638
Number of obs: 25, groups: grupp, 10

Fixed effects:
            Estimate Std. Error t value
(Intercept)  -0.7550     1.6081   -0.47
pikkus        2.5545     0.1468   17.41

Correlation of Fixed Effects:
       (Intr)
pikkus -0.963
```

ja eelnevalt demonstreeritud üleparametriseeritud mudelile

```
> lmer(kaal ~ toit + (1|grupp) + (1|grupp:toit), na.action="na.omit", data=uus)
```

mille korral funktsioon meid ka probleemi eest hoiatab.

Parameetrite usaldusvahemike leidmine (ja seega ka hüpoteeside testimine) käib `lme4` mudelite korral hetkel parameetrite järeljaotusest simuleerimise teel[12]

```
> HPDinterval(mcmcsamp(m8, n=2000))
$fixef
                lower    upper
(Intercept) -4.056210 2.566060
pikkus       2.259756 2.855854
attr(,"Probability")
[1] 0.95

$ST
     lower     upper
[1,]     0 0.5115746
attr(,"Probability")
[1] 0.95

$sigma
        lower    upper
[1,] 1.588451 2.873683
attr(,"Probability")
[1] 0.95
```

Ülesanded

V Kasutame andmestikku `Koalas` paketist `AED` ning modelleerime tunnust `pdens_2.5km`, mis väljendab koaaladele sobivate elupaikade tihedust 2.5 kilomeetri raadiuses vaatluspunktist. Kas

[12] aga see ei ole veel realiseeritud üldistatud segamudelite jaoks

tunnuse `Site` käsitlemine juhusliku faktorina võimaldab elupaikade tiheduse varieerumist modelleerida? Kuidas modellerida seda koordinaatide `easting` ja `northing` abil? Milline ruumilist autokorrelatsiooni arvestavatest mudelitest on parim?

VI Modelleeri koaalade kohalolu väljendavat tunnust `presence` kasutades seletavate tunnustena sobivate elupaikade tihedust `pdens_2.5km` ja tunnust `Site`. Paiguta mudeli jäägid mõistlikult maatriksisse ning vaata kui paljude `Site` väärtuste korral on jäägid sama väärtusega. Kas see võib viidata probleemile ruumilise autokorrelatsiooniga? Kuidas võiks välja näha permutatsioonitest kontrollimaks, milline on oodatav sama jääkide väärtusega kolmeste gruppide arv? Kas oleks ehk parem üldse juhuslikust faktorist loobuda?

Me vaatasime juba eelnevalt Ri baasgraafika kasutamist ning nägime selle võimalusi: küllaltki lihtsate käskudega saame produtseerida publitseerimiskõlbulikke graafikuid. Ehkki nt graafikupõhine hüpoteeside püstitamine või eelduste kontroll on võimalik, võiks see olla veelgi käepärasem. Siinkohal tuleb mängu pakett `lattice`, mida vaatleme selles praktikumis. Ehkki ka neid graafikuid on võimalik peenhäälestada ei ole see siinkohal meie eesmärk – soovime lihtsalt andmeid võimalikult ülevaatlikult visualiseerida. Muuhulgas üritame siin ära katta artiklis *A protocol for data exploration to avoid common statistical problems (Zuur et al., 2009)* soovitatud viisid andmete visualiseerimiseks andmetöötluse käigus.

Trellise graafika põhimõtted

Paketi kaugeks eellaseks võib lugeda nn Trellise graafikat, mis loodi üheksakümnendate alguses. Põhiuuenduseks on eraldi väljad eraldi graafikute jaoks, mis võimaldab andmeid gruppide kaupa paremini kõrvutada. Graafikud ise, mida joonistada saab, on põhimõtteliselt tuntud (histogramm, hajuvusdiagramm, karpdiagramm jm). Graafikute tellimiseks on siin aga erinevad käsud vastavalt graafikutüübile.

Tellitava graafiku võib kokku võtta järgnevalt:

vertikaaltelje tunnused ~ horisontaaltelje tunnused | grupeerivad tunnused

kusjuures tunnused eraldatakse nagu ikka sümboli `+` abil. Iga vertikaaltelje tunnus kombineeritakse läbi iga horisontaaltelje tunnusega ja iga grupeeritavate tunnuste tasemekombinatsioonile vastav graafik kuvatakse eraldi väljal. Sageli kasutame vaid üht vertikaaltelje ja üht horisontaaltelje tunnust.

Histogramm

Alustame histogrammist, mida saabki joonistada funktsiooni `histogram()` abil.

```
> histogram(~ LengthCT | AgeClass, data=Boar)
```

Oluline argument selle graafiku juures on `type`, mille abil saame määrata, kas soovime kujutada suhtelist sagedust (väärtus `"percent"`, vaikimisi), sagedust (väärtus `"count"`) või tihedusfunktsiooni lähendit (väärtus `"density"`). Pöörame ka tähelepanu, kuidas on kujutatud grupeeriva tunnuse väärtused. Punane triip märgib liikumist võimalikult väärtusteskaalal (vasakult paremale). Seega on väljade joonistuspõhimõte paketis `lattice` alt üles, vasakult paremale[1]. Kuivõrd tunnus `AgeClass` on numbriline, siis ei ole selle väärtust kuvatud. Kui me seda soovime, siis on valikuteks kas tunnuse faktoriks teisendamine või täiendava argumendi väärtustamine

```
> histogram(~ LengthCT | factor(AgeClass), data=Boar)
> histogram(~ LengthCT | AgeClass, data=Boar,
          strip=strip.custom(strip.levels=TRUE))
```

[1] seda reguleerib argument `as.table` vaikeväärtusega `FALSE`

Antud juhul tundub mõistlik paigutada kõik histogrammid ühte tulpa. Selleks saab kasutada argumenti `layout`, mis määrab väljade paigutuse lehel. Esimene number näitab, mitu välja panna igasse ritta ja teine mitu välja panna igasse veergu[2]. Niisiis

```
> histogram(~ LengthCT | factor(AgeClass), data=Boar, layout=c(1, 4))
```

ja tunnustevaheline sõltuvus (sisuliselt multikollineaarsus kui kujutaksime näiteks tunnust `Tb` ette modelleeritava tunnusena) on kohe hästi visualiseeritud. Põhimõtteliselt võime ka määrata veel

```
> histogram(~ LengthCT | factor(AgeClass), data=Boar, layout=c(1, 4),
          strip = FALSE, strip.left = TRUE)
```

Kui välju on sedavõrd palju, et nende ühele lehele paigutamine muudab tulemuse loetamatuks, siis on samuti põhjust argumendi `layout` abil ise sobiv paigutus määrata. Eri lehtede vahel liikumiseks on Windowsi kasutajal põhimõtteliselt võimalik lülitada sisse graafikute ajaloo salvestamine (graafikaakna aktiivses olekus valida *History* > *Recording*) ja seejärel saab graafikute ajaloos liikuda klahvidega 'page up' ja 'page down', ent selline meetod ei kipu paketi `lattice` funktsioonide puhul hästi töötama. Üheks lahenduseks on seada

```
> par(ask=TRUE)
```

mille tagajärjel toimub uuele graafikalehele minek alles klahvi 'Enter' vajutuse järel.

Ülesanded

I Visualiseeri tunnuse `kaal` jaotumist tunnuse `toit` järgi. Miks ei oma antud juhul erilist tähtsust, kas valime `type="count"` või `type="percent"`, ent see ei olnud nii andmestiku `Boar` korral?

II Kuidas oleks kõige parem kujutada jaotumist gruppides siis kui viimased moodustatakse tunnuste `mootja` ja `toit` tasemekombinatsioonide põhjal?

Kui meil on andmeid väga vähe ja histogrammi joonistamine tundub mõttetu, siis on üks võimalus ka lihtsalt ühemõõtmeline hajuvusdiagramm

```
> dotplot( ~ kaal | toit, data=uus, layout=c(1, 3))
```

Võimalike erindite leidmiseks on muidugi kasutatav karpdiagramm, ent ka eelnevaga analoogiline graafik võib meid aidata. Idee on siin, et kui meil on mingi tausttunnus, mille järgi vaatlused sorteerida ja mis oodatavalt peaks tunnusega, millest erindeid otsime, tugevalt seotud olema, siis saame kasutada vaatluste (tausttunnuse põhjal leitud) järjestust vertikaalteljel. Näiteks

```
> dotplot(kaal ~ pikkus, data=uus)
```

ning antud juhul näeme, et tunnuse `pikkus` jaotus on küllalt ilus. Juhul kui me soovime vertikaalteljel kujutada kategoorilist tunnust ning selle tasemed järjestada pideva tunnuse enda (summaarse statistiku nagu keskmine) järgi, siis saame kasutada funktsiooni `reorder()` abi. Näiteks

```
> dotplot(reorder(grupp, pikkus) ~ pikkus, data=uus)
```

viitab, et hajuvus gruppide sees on küllalt suur, ent osad grupid eristuvad küllalt selgelt ka omavahel võrrelduna.

[2] seega võib ka öelda, et esimene number näitab veergude arvu ja teine ridade arvu, ent see järjestus on vastupidine näiteks maatriksi mõõtmete määramisega ja seetõttu segadusttekitav

Veel sageduste kujutamisest

Muidugi saab kasutada ka tulpdiagramme, mille kujutamiseks on funktsioon `barchart()`. Mõistlik sisend on funktsiooni `xtabs()` poolt loodud (sagedus)tabel. Esimesena peaks tulema see tunnus, mille väärtused on kindlasti ühel väljal. Kui tunnuseid on kaks siis lüüakse graafikud eraldi väljadele (teise tunnuse alusel) vaid juhul kui valitakse ka `groups=FALSE`. Kui tunnuseid on kolm (või enam), siis toimub (viimas(t)e tunnus(t)e alusel) eri väljadele paigutamine igal juhul, ent valiku `groups=FALSE` korral on ühel väljal vaid ühe tunnuse (tingliku) jaotuse graafik.

Argument `horizontal`, millele saab ette anda tõeväärtuse, määrab tulpade suuna. Valik `auto.key=TRUE`, mis kasutatav ka teiste graafikute puhul, lisab lihtsa legendi.

```
> barchart(xtabs(~ AgeClass + SEX, data=Boar), auto.key=TRUE)
> barchart(xtabs(~ AgeClass + SEX, data=Boar), auto.key=TRUE,
         horizontal=FALSE, stack=FALSE)
> barchart(xtabs(~ Tb + AgeClass + SEX, data=Boar), groups=FALSE)
```

Ka funktsiooniga `dotplot()` saadav väljund võib olla mõistlik[3]

```
> dotplot(xtabs(~ AgeClass + SEX + Tb, data=Boar), auto.key=TRUE)
```

Nagu ikka on ebavõrdsete valimimahtude korral vahel otstarbekas kuvada hoopis suhtelisi sagedusi.

Karpdiagramm

Jaotuste võrdlemiseks (nii paiknemise kui kuju mõttes) on üks efektiivsemaid visuaalseid viise karpdiagramm, mille joonistamiseks on paketis `lattice` funktsioon `bwplot` ja selle süntaks on eelnevaga hästi kooskõlas. Oluline on vaid, et horisontaaltelje tunnus oleks kindlasti faktor.

```
> bwplot(LengthCT ~ factor(SEX) | AgeClass, data=Boar, layout=c(4, 1))
```

Samuti saame gruppide kaupa joonistada kvantiil-kvantiilgraafikuid. Näiteks

```
> qqmath(~ LengthCT | factor(SEX), data=Boar)
```

ütleb meile, et pikkuse jaotuseks ei ole kumbagi soo puhul normaaljaotus. Samas käsk

```
> qqmath(~ LengthCT | factor(SEX), data=Boar, groups=AgeClass)
```

annab veidi informatiivsema tulemi.

Hajuvusdiagramm

Kõige laiema kasutusega funktsioon paketis on ilmselt siiski `xyplot()`, mis joonistab hajuvusgraafikuid. Üheks oluliseks täienduseks on siin argumendi `type` väärtused `"r"` ja `"smooth"`, mis lisavad vastavalt regressioonijoone ja lokaalselt silutud kõvera. Kui soovime siiski ka punkte endid säilitada, siis tuleb lisaks anda ka väärtus `"p"`. Näiteks

[3] seda eriti juhul kui meil on väga palju gruppe ja tekkivate postide arv muudab graafiku lugemise keeruliseks

```
> xyplot(LengthCT ~ AgeClass | SEX, data=Boar, type=c("r", "p"))
```

viitab küllalt selgelt, et kui käsitleme tunnust `AgeClass` pidevana, siis põhjustab see noorimate kategoorias pikkusmõõdu ülehindamise.

Antud funktsiooni korral võib ka mitme tunnuse samaaegne kujutamine ühel ja samal teljel olla mõistlik.

```
> xyplot(kaal + pikkus ~ toit | grupp, data=uus, auto.key=TRUE)
```

Ülesanded

III Millise graafikuga oleks hea visualiseerida tunnuse `kaal` jaotust nägemaks, kas tunnusel `grupp` oleks juhusliku faktorina mõtet?

IV Kuidas oleks hea visualiseerida tunnuste `kaal` ja `pikkus` ühisjaotust nägemaks, kas tunnusel `grupp` oleks (tunnuse `pikkus` kordajat mõjutava) juhusliku faktorina mõtet?

Lõpetuseks märgime, et kuivõrd paketi `lattice` poolt pakutavate võimaluste hulk on tõesti suur siis kindlasti tekib vahel vajadus mõne täiendava valiku järele. Kui me põhimõtteliselt suudame graafikut ette kujutada, siis on võimalik seda ka õige[4] käsuga tellida. Siis on hea teada, et kõik olulisem info on koondatud funktsiooni `xyplot()` abifaili.

[4] aga võibolla küllalt keerulise

Siinkohal vaatleme veel teemasid, mis mujale ei mahtunud, ent tegelikult Ri kasutamisel väga olulised.

Apply funktsioonid

Ri baaspaketis eksisteerivad nn *apply* funktsioonid, mis võimaldavad läbi viia keerukaid operatsioone suurte andmehulkadega. Sageli on meil eesmärk andmed mingi reegli järgi grupeerida ja iga grupi korral midagi analüüsida. Selleks ongi *apply* funktsioonid[1].

Lihtsaim sarja funktsioonidest kannabki nime `apply()` ja ootab argumendiks maatriksit (või massiivi), millele (argumendi `MARGIN` väärtuse põhjal) ridupidi või veergepidi (või mõnd muud dimensioonipidi) etteantud funktsiooni (argumendi `FUN` väärtus) rakendada. Näiteks võime ette kujutada, et soovime leida iga tunnuse kvintiilid. Põhimõtteliselt on kvantiilide leidmist võimaldav funktsioon `quantile()` olemas ja seega saame seda tunnusekaupa teha, ent kui tunnuseid on palju, siis on see küllalt tülikas. Näiteks andmemaatriksis `uus` on meil kaks arvulist tunnust. Esmalt veendume, et lihtne lähenemine antud juhul ei tööta, sest tunnused pannakse kokku:

```
> quantile(uus[ , 1 : 2], c(0.2, 0.4, 0.6, 0.8), na.rm=TRUE)
   20%    40%    60%    80%
 9.592 12.352 21.294 26.582
```

Samas

```
> apply(uus[ , 1 : 2], 2, quantile, probs=c(0.2, 0.4, 0.6, 0.8), na.rm=TRUE)
    pikkus   kaal
20%  8.238 21.606
40%  9.672 23.678
60% 10.438 26.646
80% 12.594 31.830
```

töötab ilusti. Valisime antud juhul `MARGIN=2`, et funktsiooni rakendamine toimuks veergepidi ning rakendatava funktsiooni täiendavad argumendid andsime viimasena. Põhimõtteliselt võib argumendi `MARGIN` väärtus olla ka vektor, kus loetud üles mitu dimensiooni: näiteks `c(1,2)`. Selliselt juhul moodustavad eraldi üksused, millele funktsiooni rakendama hakatakse, kõikvõimalikud valitud dimensioonide väärtuste kombinatsioonid (maatriksi puhul on seega tegu üksikute väärtustega).

Teine väga kasulik funktsioon on `tapply()`, mille esimeseks sisendiks on tüüpiliselt vektor ja teiseks list faktoritest, mille tasemekombinatsioonid määravad ära grupid, millele funktsiooni (mis määratud argumendiga `FUN`) rakendama hakatakse.

Nii näiteks

```
> tapply(uus$kaal, list(uus$toit, uus$mootja), mean, na.rm=TRUE)
          A        B        E
a 28.71625       NA       NA
b       NA 26.50000 26.65500
c       NA 26.90333 21.15167
```

[1] Tegelikult võib mõnes mõttes paremini arusaadavad sama kasutusotstarbega funktsioonid leida paketist *plyr*, ent teatud (harvadel) juhtudel võib viimaste töös esineda probleeme.

Analoogiliselt töötab ka funktsioon `aggregate()`, ent siin on esimeseks sisendiks andmemaatriks.

```
> aggregate(uus[1 : 2], list(uus$toit, uus$mootja), mean, na.rm=TRUE)
  Group.1 Group.2   pikkus     kaal
1       a       A 11.36000 28.71625
2       b       B 10.55167 26.50000
3       c       B  9.41500 26.90333
4       b       E 11.41000 26.65500
5       c       E  8.85400 21.15167
```

Lisaks on põhjust veel teada funktsioone `lapply()`, mis rakendab etteantud funktsiooni listi (mis on esimeseks sisendiks) igale elemendile ja tagastab samuti listi, ja `sapply()`, mis käitub nagu `lapply()`, ent tagastab võimaluse korral lihtsama objekti (nagu nt vektor). Funktsioon `mapply()` kasutab etteantud funktsiooni esimese argumendina esimesest andmehulgast esimest elementi, teise argumendina teise andmehulga esimest elementi jne[2]. Seejärel rakendatakse funktsiooni teistele elementidele jne.

```
> s = mapply(rnorm, n=10 ** (1 : 7), mean=5, sd=2)
> abs(5 - sapply(s, mean))
[1] 1.224518e+00 2.241667e-01 4.105625e-02 1.016852e-02 4.717093e-03
[6] 7.308133e-04 3.170036e-05
```

Ülesanded

I Kuidas leida andmestiku `uus` igas tunnuse `mootja` poolt tekitatud klassis tunnuse `pikkus` maksimaalne element? Kasuta funktsiooni `with` abi lühinimede kasutamiseks.

II Olgu meil eesmärk leida andmestiku `uus` puuduvate väärtuste järjekorranumbrid iga tunnuse jaoks. Kuidas seda saavutada?

Funktsiooni kirjutamine

Vahel on meil vaja analoogilist operatsiooni sooritada korduvalt, ent funktsiooni selleks operatsiooniks ei eksisteeri. Kui me põhimõtteliselt oskame operatsiooni iga sammu kirja panna siis suudame ka ise kirjutada funktsiooni, mis just meile vajalikku operatsiooni sooritaks.

Oletame näiteks, et meil on vaja andmetest välja selekteerida ainult osad vaatlused. Täpsemalt, kujutame ette, et meil on andmestik vaatlustega, kus tunnus `pindala` väljendab lageala pindala (ruutkilomeetrites) ning tunnus `kood` vastava ala koodi. Andmestiku omapäraks on, et mida eespool asub vaatlus, seda uuem see on. Pindalad on andmestikus väga erinevad ning meie eesmärgiks on tekitada uus tunnus (sõnevektor), mis märgiks ära alad, mille pindala jääb vahemikku 2 kuni 3 ruutkilomeetrit. Seejuures soovime, et meie poolt välja valitud territoorium kataks ligikaudu 100 ruutkilomeetrit (ning seetõttu kasutame vaid kõige uuemaid vaatlusi). Liiga väikesed territooriumid markeerime sõnega `"väike"`, liiga suured territooriumid markeerime sõnega `"suur"`, meile sobivad sõnega `"sobib"` ja sobivad, ent väljapoole kumulatiivset 100 ruutkilomeetri piiri jäävad alad sõnega `"vana"`.

[2] Kui andmehulgad on õigesti nimetatud siis ei ole järjekord oluline

Kuidas sellist tunnust tekitada? Võimalusi on muidugi mitu, ent kui näiteks kõik senitoodud arvud (2, 3 ja 100) võivad hiljem ka muutuda (või näiteks hakkab meie loodut hiljem kasutama keegi teine), siis võib funktsiooni kirjutamine olla päris hea mõte.

Funktsioon ise võiks välja näha umbes järgmine[3]:

```
> kvalimine = function(x, kokku=100, min=2, max=3){
          summa = 0
          vaatlusi = dim(x)[2]
          uus = rep("vana", vaatlusi)
          for (i in 1 : vaatlusi){
                  if (x$pindala[i] > max){
                          uus[i] = "suur"
                          next
                  }
                  if (x$pindala[i] < min){
                          uus[i]="väike"
                          next
                  }
                  if (summa < kokku){
                          uus[i]="sobib"
                          summa=summa+x$pindala[i]
                          next
                  }
                  break
          }
          print(paste("Valitud alade pindala kokku:", summa))
          print(paste("Valitud alasid:", sum(uus == "sobib")))
          print(paste("Väikeseid alasid:", sum(uus == "väike")))
          print(paste("Suuri alasid:", sum(uus == "suur")))
          print(paste("Vanu alasid:", sum(uus == "vana")))
          return(data.frame(x, uus))
  }
```

Nagu näeme on Ri funktsiooni keha loogiliste sulgude vahel. Analoogiliselt tuleb kokku koondada ka näiteks tingimuslausele `if` või tsükli algust tähistavale lausele `for` järgnev tegevus. On kolm põhilist programmeerimise elementi, mida me seni pole käsitlenud. Tingimuslause

if (tingimus){tegevus(ed)} else {tegevus(ed)}

korral kontrollitakse mingi tingivuse kehtivust ning sõltuvalt tõeväärtusest valitakse edasine tegevus. Seejuures võib tegevuseks olla muidugi mingi uus tingimuse kontroll ja sellele vastav tegevus ehk teine tingimuslause esimese sees.

Tingimuslausega tsükli

[3] puuduvaid väärtusi antud juhul ei eelda

while (tingimus){tegevus(ed)}

korral toimub tegevus(t)e tegemine ainult siis kui tingimus on täidetud. Erinevus seisneb siin selles, et pärast tegevus(t)e lõppu kontrollitakse tingimust uuesti ning kui see on endiselt tõene siis korratakse ka tegevust/tegevusi ning see tsükkel kestab seni, kuni tingimus enam ei kehti.

Stabiilne tsükkel

for (tsüklimuutuja in väärtushulk){tegevus(ed)}

on põhimõttelt analoogiline eelnevaga, ent erinevus on siin asjaolus, et tsüklimuutuja võtab peale igakordset tegevus(t)e lõppu järgmise väärtuse väärtushulgas ning tsükkel lõpeb siis kui ka viimase väärtuse korral on tegevus(ed) sooritatud.

Eranditeks on laused `next`, mis lõpetab tegevuse(d) ja alustab tsüklit uuesti tsüklimuutuja järgmise väärtusega väärtushulgast ning `break`, mis lõpetab kogu protsessi enneaegselt.

Anonüümsed funktsioonid

Eelnevalt kirjutatud funtsioonil oli nimi ning seda oli võimalik hiljem välja kutsuda. Vahel vajame aga lihtsat funktsiooni, näiteks ainult mõne apply funktsiooni rakendamise jaoks. Näiteks kujutame ette, et soovime iga toidu ja mõõtja kombinatsiooni jaoks leida teiseks suure kaaluga isendi. Ei ole midagi lihtsamat kui kirjutada anonüümne funktsioon otse apply funktsiooni argumendi `FUN` väärtuseks[4]:

```
> tapply(uus$kaal, list(uus$toit, uus$mootja),
         function(x){sort(x, decreasing=TRUE)[2]})
     A     B     E
a 33.3    NA    NA
b   NA 26.71 30.63
c   NA 26.91 26.34
```

Tuleb lihtsalt teha nii, et funktsioon suudaks saadava sisendiga toime tulla ning funktsiooni väljund oleks õiget tüüpi.

Ülesanded

III Loo andmestikku `uus` täiendav tunnus `mitmes`, mis iga tunnuse `kaal` väärtuse jaoks väljendab (suhtelist) järjekorranumbrit toidu ja mõõtja kombinatsiooni jaoks. Seega iga kombinatsiooni suurem kaal saab väärtuseks 1, teiseks suur 2 jne.

IV Leia tunnuse `mitmes` alusel moodustatud gruppide jaoks keskväärtuse ja mediaani vahe tunnuse `kaal` jaoks. Kuidas see vahe käitub?

[4] Ris tagastatakse funktsiooni puhul alati vaikimisi viimase rea väärtus, mistõttu `return` ei ole kohustuslik

Sõnetöötlus

Vahel on meil sõnedest vaja kätte saada ainult mingi alamsõne, ent seni pole me vaadanud kuidas seda saavutada. Esimese asjana sõne pikkuse määramine funktsiooni `nchar()` abil.

```
> nimed = names(islands) #maismaaosade nimed
> nchar(nimed)
 [1]  6 10  4  9 12  6  5  6  7  7  5  4  5  9  6  9  6 10  8  6  7  7  4  6  5
[26] 10  8  8  8 11 10 15 15 12 13 13 15  8 13 11 11  7  6  8 16  5  9  8
```

Funktsioon `paste()`, mida oleme juba töös näinud võimaldab sõnesid[5] kokku kleepida, kusjuures argument `sep` määrab loodavate sõnede sees kasutatava eraldaja. Lisaks on funktsioonil veel argument `collapse`, mille väärtustamise korral kleebitakse tulemuseks saadav sõnevektor kokku üheks (pikaks) sõneks ning eraldajana kasutatakse antud väärtust.

```
> (koos = paste(nimed, islands))[1 : 10]
 [1] "Africa 11506"    "Antarctica 5500" "Asia 16988"      "Australia 2968"
 [5] "Axel Heiberg 16" "Baffin 184"      "Banks 23"        "Borneo 280"
 [9] "Britain 84"      "Celebes 73"
```

Pöördoperatsiooni ehk sõnede tükeldamist teostab `strsplit()`, millele tuleb lisaks sõnevektorile (argumendi `split` abil) ette anda ka sümbol, mis määrab tükelduskohad[6]. Tagastatakse list[7], kus esialgse vektori igale elemendile vastab üks listi element.

```
> strsplit(koos, " ")[1 : 5]
[[1]]
[1] "Africa" "11506"

[[2]]
[1] "Antarctica" "5500"

[[3]]
[1] "Asia"  "16988"

[[4]]
[1] "Australia" "2968"

[[5]]
[1] "Axel"    "Heiberg" "16"
```

Sõne fikseeritud osa eraldamiseks või asendamiseks on funktsioon `substr()`, millele tuleb lisaks sõnevektorile ette anda ka esimese ja viimase valitava sümboli asukoht. Valitud sümbolid võib ka asendada.

```
> (algus3 = substr(nimed, 1, 3))[1 : 5]
```

[5] tegelikult ka näiteks arve ja kõike muud, mida saab sõne(de)ks teisendada

[6] tegelikult ei pea eraldajat/eraldajaid nii lihtsalt määrama – kasutada saab ka nn regulaaravaldisi, millega saame lühidalt kirja panna väga keerukaid mustreid. Regulaaravaldiste kohta vaata täpsemalt `?regex`

[7] ja siis võib kasulik olla funktsioon `unlist()`, mis kõik ühte vektorisse kokku paneb

```
[1] "Afr" "Ant" "Asi" "Aus" "Axe"
> substr(algus3, 1, 1) = "@"
> algus3[1 : 5]
[1] "@fr" "@nt" "@si" "@us" "@xe"
```

Võimsamat otsingut ja asendamist (regulaaravaldiste kaasabil) võimaldavad vastavalt `grep()` ja `gsub()`[8]. Mõlema korral tuleb muster anda esimesele argumendile (nimega `pattern`), sõnevektor ise tuleb anda vastavalt teise või kolmandana (sest funktsiooni `gsub()` korral on teine argument `replacement` ehk asendus). Funktsioon `grep()` tagastab leitud vastete positsioonid sõnevektoris ja `gsub()` muudetud sõnevektori.

```
> nimed[grep("B", nimed)]
[1] "Baffin"      "Banks"       "Borneo"      "Britain"     "New Britain"
> gsub("New", "Uus", nimed)[30 : 35]
[1] "Uus Britain"     "Uus Guinea"      "Uus Zealand (N)" "Uus Zealand (S)"
[5] "Uusfoundland"    "North America"
```

Ülesanded

V Oletame, et meile pakuvad huvi ainult need maismaaosad, mille nimi algab (mitte lihtsalt ei sisalda) sümboliga *B*. Kuidas neid leida kavalusega? Regulaaravaldisega?

VI Funktsiooni `strsplit()` näite juures märkasime, et tunnuste eraldamine ei olnud enam lihtne, sest sama eraldaja oli juba eelnevalt kasutuses. Kuidas mõne apply funktsiooni abil see asi korda ajada nii, et saaksime pindalad uuesti eraldi tunnusena kätte.

Kuupäevad

Nagu ikka on kuupäevad täiesti eraldiseisvad nii lihtsalt arvudest kui sõnedest. Põhimõtteliselt võime muidugi kuupäevad jätta sõnedeks, ent see tähendab, et näiteks kahe kuupäeva ajalist vahet ei ole võimalik leida. Seetõttu on ka Rile mõistlik arusaadavaks teha kui tunnus väljendab just nimelt kuupäevi.

Kui meie kuupäevatunnus on sõne vormis, siis tuleb Rile ette näidata lihtsalt kuupäeva formaat. Näiteks

```
> as.Date("01.12.2010", format="%d.%m.%Y")
[1] "2010-12-01"
```

kus *%d* tähendab päeva numbrit, *%m* kuu numbrit ning *%Y* täispikka aastanumbrit[9]. Tulemuseks ongi objekt, mille klassiks kuupäev.

Vahel võib aeg olla tähistatud numbriga, mis väljendab päevi mingist fikseeritud kuupäevast[10]. Sellisel juhul tuleb sellest fikseeritud kuupäevast ka Ri informeerida.

[8] nende funktsioonide abifailist võib lugeda veel teistegi huvitavate analoogiliste funktsioonide kohta, mis tagastavad veidi teistsugust infot

[9] rohkem väärtusi leiab funktsiooni `strptime()` abifailist

[10] üldiselt niimoodi kuupäevi talletataksegi

```
> as.Date(200, origin="2010-05-15")
[1] "2010-12-01"
```

Juhul kui meil on vaja kasutada ka kellaaegu, siis võib olla otstarbekas kasutada üldisemat klassi võimaldamaks näiteks mugavamat ajavahede arvutamist. Siin tuleb mängu funktsioon `strptime()`, mille kasutamine on analoogiline eelnevale.

```
> (aeg1 = strptime("2010-09-12 kell 11", format="%Y-%m-%d kell %H"))
[1] "2010-09-12 11:00:00"
> (aeg2 = strptime("2010/08/12 @11:34", format="%Y/%m/%d @%H:%M"))
[1] "2010-08-12 11:34:00"
> aeg1 - aeg2
Time difference of 30.97639 days
> as.numeric(aeg1 - aeg2, units="mins")
[1] 44606
```

Lõpetuseks

Ehkki kursusel käsitletu peaks olema piisav, et küllalt edukalt ilma kõrvalise abita Riga töötada, võib siiski arvata, et pikemal kasutamisel kerkib esile mõni probleem, millest enda jõud üle ei käi. Kui probleem on tõepolest lihtne siis on see kindlasti tekkinud veel paljudel teistelgi kasutajatel ja suure tõenäosusega on see ka kusagil internetifoorumites käsitlemist leidnud. Selle ülesleidmine sõltub siis lihtsalt meie oskusest mõnesse interneti otsingumootorisse mõistlikud märksõnad sisse kirjutada. Üks ainult Rile keskenduv otsingumootor on leitav aadressilt `www.rseek.org` ja mitmed muud on leitavad Ri kodulehelt. Siiski on põhjust üles lugeda mõned täiendavad abivahendid.

Raamatud

See nimistu kasvab kiiresti. Eriti valdkonnaspetsiifiliste teoste nimekiri. Mõned klassikud on

M. J. Crawley – The R book (2007)
`http://www.bio.ic.ac.uk/research/mjcraw/therbook/index.htm`
entsüklopeediamõõtu teos, mis katab Ri baasteadmised (sealhulgas lihtsam statistiline andmetöötlus)

P. Murrell – R Graphics (2005)
`http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html`
käsitlus katab lihtsad ja mõned keerulisemad Ri graafikud

D. Sarkar – Lattice: Multivariate Data Visualization with R (2008)
`http://lmdvr.r-forge.r-project.org`
põhjalikum ülevaade paketist *lattice* selle autorilt

Pikk (ja uuenev) nimistu Riga seotud raamatutest
`http://www.r-project.org/doc/bib/R-books.html` (`http://tinyurl.com/ffw8t`)
Huvi võivad pakkuda näiteks Bolker (2008) ja Zuur et al. (2009).

TÜ raamatukogu märksõnaks on „R (programmeerimiskeel)“. Nagu ikka võib raamatutest esmase ülevaate saada Google Booksi abiga.

Internet

Ri meililistid on leitavad aadressilt
`http://www.r-project.org/mail.html`
Kõiki neid katab ka näiteks Google'i otsing

Ri paketid rühmitatud temaatikapõhiselt (*R taskview*)
`http://cran.r-project.org/web/views/`

Ri käsiraamatud
`http://cran.r-project.org/manuals.html`

Tom Short'i tähtsamate Ri funktsioonide lühikokkuvõte kasutaja lauale
`http://cran.r-project.org/doc/contrib/Short-refcard.pdf` (`http://tinyurl.com/hqs4z`)

R-temaatiline teadusajakiri, kus tutvustatakse uusi pakette
`http://journal.r-project.org/index.html`

Ri graafikagalerii koos koodidega
`http://addictedtor.free.fr/graphiques/thumbs.php` (`http://tinyurl.com/ggbcm`)

Midagi filosoofilisemat: Ri minevikust, olevikust ja tulevikust kirjutab John Fox
`http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Fox.pdf`
(`http://tinyurl.com/2w3gpeh`)

Graafikute keerulisem kohandamine*

Võimalused graafikute kohandamiseks on tegelikult väga laiad ning kui me soovime Ri abil publitseerimiskõlblikku graafikut produtseerida, siis võib vahel[1] vaja minna täiendavat seadistamisoskust. Räägime siin sel teemal põhjalikumalt.

Esimene asi, millel pikemalt peatuda, on graafikaparameetreid reguleeriv funktsioon `par()`, mille kasutamist oleme enne ka näinud ja mis oma käitumiselt sarnaneb funktsioonile `options()` või `palette()`. See tähendab, et kui me kutsume funktsiooni välja ilma argumentideta kuvatakse graafikaparameetrite hetkeväärtused. Muidugi saame sama funktsiooni abil parameetreid ka muuta, andes neile uued väärtused, kusjuures `par()` tagastab muudetud parameetrite endised väärtused[2].

```
> par() #vaatame parameetrite nimesid ja väärtusi
> op = par(cex=2, cex.lab=2) #muudame parameetrite väärtusi, salvestame endised
> par(op) #taastame endised väärtused
```

Oluline on teada, et graafikaakna sulgemise korral taastatakse kõigi graafikaparameetrite vaikeväärtused. Nagu me parameetrite nimistust näeme on paljud neist väärtustatavad ka vastava graafikut joonistava funktsiooni (nt `plot()`) väljakutsumisel. Sellisel juhul kehtivad väärtustused muidugi ainult selle graafiku jaoks ning järgmise graafiku korral neid seadeid enam ei arvestata.

Üks lihtsamaid graafikaparameetreid on `las`, mis muudab telgede ühikute paigutuse suunda ja mille vaikeväärtus on 0.

`las` väärtus	telgede ühikud on
0	telgedega samasuunalised
1	horisontaalsed
2	telgedega risti
3	vertikaalsed

Teksti joondamist seab parameeter `adj`, mille vaikeväärtuseks on 0.5, mis vastab tsentreeritud paigutusele. Vasakule joondatud teksti saame väärtuse 0 ning paremale joondatud teksti väärtuse 1 korral. Kasutatavad on ka vahepealsed väärtused.

Teljekriipsukeste pikkust saab määrata argumendi `tck` abiga, mille absoluutväärtus on kriipsukese pikkuse suhe ümbriskasti lühema külje pikkusesse. Märk määrab kriipsukeste suuna.

[1] pigem vist sageli
[2] ehk me saame ka vanad väärtused meelde jätta ja need hiljem ennistada

```
> plot(weight ~ Time, col=as.numeric(Chick), data=ChickWeight,
         las=1, adj=0, tck=0.02)
```

Kuidas määrata ümbriskastist väljaspoole jääva ruumi (ehk ääriste) suurust? See käib parameetri `mar` abil, kus väärtuseks on neljaelemendiline vektor. Esimene element vastab alumisele äärisele ja nii edasi päripäeva. Ühikuks on seejuures tekstiridade arv. See on üks parameetritest, mis tuleb ära muuta juba enne graafiku joonistamist.

```
> op = par(mar=c(2.2, 2.2, 0.2, 0.2))
> plot(weight ~ Time, col=as.numeric(Chick), data=ChickWeight)
> par(op)
```

Vahel on vaja kõrvuti paigutada mitu graafikut ning siin vajame funktsiooni `layout()` abi[3]. Esimene argument `mat` ootab väärtuseks maatriksit positiivsete täisarvudega. Maatriksi mõõtmed näitavad graafikute arvu ja asetust ning väärtused joonistamise järjekorda. Funktsioon `layout()` tagastab joonistatavate graafikute arvu ning selle funktsioonile `layout.show()` ette andes[4] näeme alade paigutust. Kujutame ette, et soovime esmalt kujutada hajuvusdiagrammi all vasakul nurgas, seejärel marginaaljaotusi, vastavalt hajuvusdiagrammist ülal ja paremal ning ülemise parema nurga jätta tühjaks. Kasutame andmestikuna geisri Old Faithful purskamise ja mittepurskamise kestusandmeid. Selguse huvides teeme graafiku vormindamist sammhaaval.

```
> paigutus = matrix(c(2,1,4,3), nrow=2)
> paigutus
     [,1] [,2]
[1,]    2    4
[2,]    1    3

> hp=layout(paigutus)
> layout.show(hp)
> plot(eruptions ~ waiting, data=faithful)
> tykeldus1=cut(faithful$waiting, breaks=12)
> tykeldus2=cut(faithful$eruptions, breaks=12)
> barplot(table(tykeldus1))
> barplot(table(tykeldus2), horiz=TRUE)
```

Nii saadav tulemus ei ole kiita. Tunduks mõistlik hajuvusdiagrammi ala suurendada. Funktsiooni `layout()` argumendid `widths` ja `heights` lubavad seada vastavalt alade suhtelised laiused ja kõrgused. Seega kui meil on antud juhul graafikud kahes reas ja veerus ja me soovime, et esimene veerg oleks kolm korda laiem kui teine ja esimene rida kolm korda kitsam kui teine, siis peaksime valima

```
> layout(paigutus, widths=c(3, 1), heights=c(1, 3))
```

mispeale graafikuid uuesti joonistades saame aga veateated, sest tulpdiagrammid ei mahu oma aladele ära. Põhjuseks on nende ääriste liigne suurus. Katsume seda parandada neid vähendades. Lisaks jätame ära tulpdiagrammide teljed ja surume tulbad üksteise kõrvale.

[3] Lihtsamal juhul aitavad ka graafikaparameetrid `mfrow` ja `mfcol`

[4] arvestades, et tegu on lihtsalt graafikute arvuga võime selle ka lihtsalt ise ette anda ilma midagi eelnevalt salvestamata

```
> par("mar") #vaatame järgi hetkeväärtused
> plot(eruptions ~ waiting, data=faithful)
> op = par(mar=c(0, 4.1, 0.2, 2.1))
> barplot(table(tykeldus1), axes=FALSE, space=0, names.arg="")
> par(mar=c(5.1, 0, 4.1, 0.2))
> barplot(table(tykeldus1), axes=FALSE, horiz=TRUE, space=0, names.arg="")
> par(op)
```

Nii hakkab asi ilmet võtma, ent tegelikult võiks graafikutevahelised vahed veelgi koomale tõmmata. Seega on vaja ka hajuvusgraafiku ääriseid muuta. Et asukohad klapiksid tuleb siis muuta ka teisi ääriseid. Lisaks suurendame veidi kirjade ja vähendame punktide suurust.

```
> layout(paigutus, widths=c(3, 1), heights=c(1, 3))
> op = par(mar=c(4.1, 4.1, 0.2, 0.2))
> plot(eruptions ~ waiting, data=faithful, cex=0.7, cex.lab=1.5, cex.axis=1.5)
> par(mar=c(0, 4.1, 0.2, 0.2))
> barplot(table(tykeldus1), axes=FALSE, space=0, names.arg="")
> par(mar=c(4.1, 0, 0.2, 0.2))
> barplot(table(tykeldus1), axes=FALSE, horiz=TRUE, space=0, names.arg="")
> par(op)
```

Tulemusena ei taha telgede pealkirjad hästi ära mahtuda. Üks võimalus oleks vastavaid ääriseid suurendada, ent teine võimalus on tekitada kogu graafikute poolt kaetud alale ääris. Seda reguleerib parameeter `oma`. Hetkel

```
> par("oma")
[1] 0 0 0 0
```

Väärtuste mõjujärjekord on ikka sama (alustades alt ja liikudes päripäeva) ning mõõtühikuks ikka tekstiridade arv.

```
> op = par(oma=c(1, 1, 1, 1))
> layout(paigutus, widths=c(3, 1), heights=c(1, 3))
> layout.show(4)
> par(mar=c(4.1, 4.1, 0.2, 0.2))
> plot(eruptions ~ waiting, data=faithful, cex=0.7, cex.lab=1.5, cex.axis=1.5)
> par(mar=c(0, 4.1, 0.2, 0.2))
> barplot(table(tykeldus1), axes=FALSE, space=0, names.arg="")
> par(mar=c(4.1, 0, 0.2, 0.2))
> barplot(table(tykeldus1), axes=FALSE, horiz=TRUE, space=0, names.arg="")
> par(op)
```

Tulemuseks on küllalt professionaalse välimusega graafik, kusjuures nagu näeme, ei ole vajalike koodiridade arv kuigi suur[5].

Lisaks eksisteerib veel parameeter `mgp`, mida saab väärtustada kolmeelemendilise vektoriga. Esimene määrab telje pealkirja, teine telgede ühikute ning kolmas telje enda kauguse (vastavatest ümbriskasti äärtest).

[5] iseasi, kas oskame need kohe nii kirja panna, et lõpptulemusega rahule jääme

Vahel on meil telgedele vaja anda nimed, kus sisalduvad nt kreeka tähed, ala- või ülaindeksid. Seda aitab saavutada funktsioon `expression()`. Ülaindeksid saame sümboli `^`, ning alaindeksid kandiliste sulgude abil. Kreeka tähed tuleb kirjutada inglise keeles. Kui meil on lisaks vaja samas kasutada ka tavalist teksti siis tuleb tulemus esmalt ise kokku kleepida. Näiteks

```
> plot(1 : 10, qchisq(0.95, 1 : 10), xlab="vabadusastmete arv",
         ylab=expression(paste(chi^2, " jaotuse 0.05 täiendkvantiil")))
```

Vahel võib olla vajadus, et graafiku enda mõõtmed oleksid proportsioonis telgede kajastatud väärtustega. Graafikuakna mõõtmeid saame muuta hiire abil, ent seda võib ka teha funktsiooni `x11()` abil, mis tegelikkuses graafikuakna avab. Vaikimisi on akna suuruseks 7 tolli korda 7 tolli. Saamaks näiteks laia ja madalat akent võime tellida

```
> x11(12,3)
```

Graafikute joonistamine otse faili*

Peale graafiku valmimist saame (aktiivse graafikuakna korral) valida *File* > *Save as* ning seejärel määrata sobiva failiformaadi, millena graafik salvestada tuleks. Sellise praktika korral on küll lihtsasti võimalik muuta graafikuakna suurust, ent see võib põhjustada probleeme graafiku loetavuses. Sageli on soovitav tekitada graafik hoopis otse faili. Selleks saame kasutada nt funktsioone `pdf()` ja `png()`[6]. Windowsis annab formaat .pdf parima tulemi[7], ent paraku ei saa seda tüüpi faile üldiselt otse näiteks Wordi ümber tõsta. Üheks lahenduseks on faili avamine kasutades vabavaralist programmi Adobe Reader ning seal *Tools* > *Select & Zoom* > *Snapshot Tool* abil pildi puhvrisse võtmine ning seejärel otse Wordi kopeerimine. See lähenemine ei pruugi anda häid tulemusi OpenOffice Writeri korral. Sel juhul on kasulikum produtseerida Ri abil .emf fail, milleks võime kasutada funktsiooni `win.metafile()`. Peale joonistamise lõppu tuleb failid ka salvestada, mille teostab funktsioon `dev.off()`. Failid produtseeritakse töökataloogi. Näiteks

```
> pdf("testfail.pdf")
> plot(weight ~ Time, col=as.numeric(Chick), cex=0.5, data=ChickWeight)
> dev.off()
```

Kui me soovime joonistada mitu graafikut, siis tuleb kas eelnevat tegevust (iga kord veidi muudetud kujul) korrata (sest kui me vahepeal käsku `dev.off()` ei anna, siis salvestatakse vaid viimane graafik[8]) või siis võime ka failinimesse märkida sõneosa kujul `%0xd`, kus `x` on näiteks 1 või 2, mispeale iga graafik salvestatakse eraldi failina, kusjuures `%0xd` failinimes asendatakse graafiku järjekorranumbriga, mille kohtade arvu määrab `x`. Kui me kasutame väljastamiseks funktsiooni `pdf()`, siis tuleb lisaks määrata `onefile=FALSE`, mispeale iga graafik salvestatakse eraldi faili.

```
> pdf("testfail%02d.pdf", onefile=FALSE)
> plot(weight ~ Time, col=as.numeric(Chick), cex=0.5, data=ChickWeight)
> plot(weight ~ Time, col=as.numeric(Chick), cex=0.8, data=ChickWeight)
> plot(weight ~ Time, col=as.numeric(Chick), cex=1, data=ChickWeight)
> dev.off()
```

[6] muuhulgas saab neile ette anda ka graafiku mõõtmed nagu funktsioonile `x11()`

[7] Tegelikult saab ka Windowsis nii mõndagi ära teha, et saada nt ilusaid .png graafikuid, ent see eeldab olulisel määral lisatarkvara paigaldamist. Vaata näiteks `http://www.rforge.net/Cairo/index.html`

[8] või siis pannakse kõik graafikud ühte faili kokku