Reprodutseeritav andmeanalüüs kasutades R keelt

Ülo Maiväli & Taavi Päll

2019-09-25

Contents

Haara kannel, Vanemuine! 5
0.1 Sissejuhatus 6
0.2 R keel 7
0.3 Andmeanalüüs 8
0.3.1 Mida andmeanalüütik peaks küsima, enne kui ta alustab. 9
0.4 Tarkvaratööriistad 12
0.4.1 Installeeri vajalikud programmid 12
0.4.2 Loo GitHubi konto 12
0.4.3 Loo uus R projekt 13
0.4.4 Git *Merge* konfliktid 13
0.4.5 R projekti kataloogi soovitatav minimaalne struktuur 15
0.4.6 Pakettide installeerimine 16
0.4.7 R repositooriumid 18
0.5 R on kalkulaator 20
0.6 R-i tööpõhimõte 22
0.6.1 Funktsioon 25
0.6.2 Sama koodi saab kirjutada neljal erineval viisil 27
0.6.3 objekt 29
0.6.4 Andmete tüübid 31
0.6.5 Objektide klassid 32
0.6.6 Indekseerimine 39
0.7 Lihtne töö Andmeraamidega 44
0.7.1 Võrdleme andmeraame kahel viisil ja summeerime andmeraami. 44
0.7.2 Põhitehted andmeraamidega 45
0.7.3 Rekodeerime andmeraami väärtusi 48
0.7.4 Ühendame kaks andmeraami rea kaupa . . . 49

0.7.5 ühendame kaks andmeraami veeru kaupa . . 50
0.7.6 andmeraamide ühendamine join()-ga 51
0.7.7 Nii saab raamist kätte vektori, millega tehteid teha. 52
0.7.8 Andmeraamide salvestamine (eksport-import) 53
0.7.9 NA-d 56
0.8 map() - kordame sama operatsiooni igale listi liikmele 62
0.9 Regular expression ja find & replace 71
0.9.1 Common operations with regular expressions 74
0.9.2 Find and replace 75
0.10 Funktsioonid on R keele verbid 77
0.10.1 Kirjutame R funktsiooni 80
0.11 Graafilised lahendused 84
0.11.1 Baasgraafika 84
0.11.2 ggplot2 94
0.11.3 *Facet* – pisigraafik 109
0.11.4 Mitu graafikut paneelidena ühel joonisel . . 114
0.11.5 Teljed 114
0.11.6 Graafiku pealkiri, alapeakiri ja allkiri 125
0.11.7 Graafiku legend 126
0.11.8 Värviskaalad 127
0.11.9 A complex ggplot 136
0.12 Kuraditosin graafikut, mida sa peaksid enne surma joonistama 138
0.12.1 Cleveland plot 138
0.12.2 Andmepunktid mediaani või aritmeetilise keskmisega 144
0.12.3 Histogramm 148
0.12.4 Tihedusplot 155
0.12.5 Boxplot 159
0.12.6 Joongraafikud 162
0.12.7 Scatter plot 168
0.12.8 Tulpdiagramm 176
0.12.9 Residuaalide plot 181
0.12.10 Tukey summa-erinevuse graafik 187

0.12.11 QQ-plot 195
0.12.12 Heat map 207
0.12.13 Biplot ja peakomponentanalüüs 217
0.13 Üldised jooniste printsiibid 220
0.14 Tidyverse 221
0.14.1 Tidy tabeli struktuur 222
0.14.2 dplyr ja selle viis verbi 225
0.14.3 `separate()` üks veerg mitmeks 238
0.14.4 Faktorid 241
0.15 lisa: funktsioonid 250
0.15.1 raamatukogud 250
0.15.2 failide sisselugemine 251
0.15.3 matemaatika 251
0.15.4 andmeraamid 251

Haara kannel, Vanemuine!

Kas oled tundnud, et sul tekib andmeid rohkem kui sa neid "käistsi" analüüsida jõuad? Sa oled sunnitud analüüsiks valima oma multidimensionaalsetest "suurtest" andmetest ainult pisikese osa. See ei pruugi olla iseenesest halb, sest vähendab oluliselt testitavaid hüpoteese ja keskendub ainult kõige selgemini interpreteeritavatele efektidele. Teisalt, kas sa oled tundnud frustratsiooni algandmete, transformatsioonide, jooniste ja statistikute paigutamisel *workbook*-i. Kas sa oled tundnud frustratsiooni sellest *workbook*-ist kuu-kaks hiljem aru saamisel. Kas keegi teine saab aru mis sa oma andmetega teinud oled?

Kui sul tekivad eelmainitud probleemid, oled sa ilmselt valmis järgmiseks elu muutvaks sammuks andmeanalüüsi ja statistika vallas – skaleerimaks need protsessid ülesse võttes kasutusele skriptid ja muutes oma töövoo reprodutseeritavaks.

Skriptid ja koodid võimaldavad sul hoida lahus algandmed (mis on püha ja puutumatu) andmete töötlusest ja töödeldud andmetest

ning genereerida eraldiseisvad andmeanalüüsi produktid – joonised ja raportid.

Selline reprodutseeritav töövoog on tänapäeval võimalik organiseerida kasutades erinevaid andmeanalüüsi programmeerimiskeeli, eelkõige näiteks *Python* ja R. *Python*-il ja R-il on loomulikult mitmeid erinevusi ja paralleelseid omadusi, esimene on nö täielik keel, võimaldades luua ka iseseisva graafilise kasutajaliidesega programme. R on seevastu mõeldud eelkõige andmeanalüüsiks ja selle tulemuste visualiseerimiseks.

Antud raamat keskendub sissejuhatusele **R statistilise programmeerimiskeelde**, mille jaoks on praeguseks hetkeks välja töötatud ka suurepärased kasutajaliidesed, nii et R kasutamine ei eelda 100% tööd käsurealt-konsoolist. Lisaks R-ile annab antud raamat ka mõned soovitused oma **töövoo reprodutseeritavaks organiseerimiseks**.

Jõudu ja entusiasmi sellel teel!

0.1 Sissejuhatus

See õpik on kirjutatud inimestele, kes kasutavad, mitte ei uuri, statistikat. Õpiku kasutaja peaks olema võimeline töötama R keskkonnas. Meie lähenemised statistika õpetamisele on arvutuslikud, mis tähendab, et me eelistame meetodi matemaatilise aluse asemel õpetada selle kasutamist ja tulemuste tõlgendamist. See õpik on bayesiaanlik ja ei õpeta sageduslikku statistikat. Me usume, et nii on lihtsam ja tulusam statistikat õppida ja et Bayesi statistikat kasutades saab rahuldada 99% teie tegelikest statistilistest vajadustest paremini, kui see on võimalik klassikaliste sageduslike meetoditega. Me usume ka, et kuigi praegused kiired arengud bayesi statistikas on tänaseks juba viinud selle suurel määral tavakasutajale kättesaadavasse vormi, toovad lähiaastad selles vallas veel suuri muutusi. Nende muutustega koos peab arenema ka bayesi õpetamine.

Me kasutame järgmisi R-i pakette, mis on kõik loodud bayesi mudelite rakendamise lihtsustamiseks: “rethinking” (McElreath 2016), “brms” (Bürkner 2017), “rstanarm” (Stan Development Team 2016), “BayesianFirstAid” (Bååth 2013) ja “bayesplot” (Gabry and Mahr 2017). Lisaks veel “bayesboot” bootstrapimiseks (Bååth 2016). Bayesi arvutusteks kasutavad need paketid Stan ja JAGS mcmc sämplereid (viimast küll ainult ‘BayesianFirstAid pakett). Selle õpiku valmimisel on kasutatud McElreathi (McElreath 2015), Kruschke (Kruschke 2015) ja Gelmani (Gelman et al. 2014) õpikuid.

0.2 R keel

R on andmeanalüüsile spetsialiseeritud vabavaraline programmeerimiskeel. See on nn kõrge tasemega keel, mis tähendab, et selle selle keele “kõnelemine” ei ole liiga erinev inglise keele kõnelemisest. Erinevus on selles, et kui inglise keeles saab rääkida peaaegu kõigest, ja näiteks Pythoni programmeerimiskeeles sobib kõneleda kõigest, mida saab programmeerida, siis R-s on otstarbekas ja efektiivne teha andmeanalüüsi ja statistikat. R-i kasutusala on kitsam. Teine erinevus inglise keelest on, et R keele sõnad e funktsioonid, on pea alati tegusõnad. Näit sum() tähendab “take the sum”, filter() tähendab “filter data” jne. Seega saab R koodi tõlkida kui “tee seda, siis tee toda”, ja nii edasi.

R-i põhiline eelis näiteks Pythoni ees (tema suurim konkurent andmeanalüüsil) on, et R-s on olemas suurem valik abifunktsioone, mis võimaldavad sujuvamaid, paremini inimloetavaid ja lihtsamini õpitavaid andmeanalüüsi töövooge, eriti mis puudutab jooniste tegemist. Valdav enamik professionaalseid statistikuid kasutab R-i, samas kui inseneride ja keemikute seas on pigem populaarne Python ja näiteks pildianalüüsil ruulib Matlab. Kõike, mida saab teha R-is, saab teha ka Pythonis ja Matlabis (ja enamasti ka vastupidi).

R-i arendavad selle keele igapäevased kasutajad, kelle hulgas on

juhtivad statistikud/ andmeanalüütikuid. Seega (1) jõuavad uued meetodid sageli enne R-i kui kuhugi mujale, (2) sa tead täpselt, kes tegi selle meetodi, mida sa täna kasutasid (ja kus on selle viide kirjanduses), (3) sa võid kirjutada meetodi autorile ja abi küsida. Kuigi R-i abifailid võivad olla rudimentaarsed, keerulised või lihtsalt kasutud, annab guugeldamine vastuse enamustele küsimustele.

0.3 Andmeanalüüs

Andmeanalüüs on see osa teadusprojektist, mis enamasti algab siis, kui katse/vaatlus lõpeb. Andmeanalüüsi motiveerivad teaduslikud küsimused, kuid selle formaalsed väljundid on hoopistükis kujul (1) summaarsed statistikud, (2) joonised ja (3) statistilised mudelid. Need andmeanalüüsi tulemused konverteeritakse omakorda mitteformaalsel moel teaduslikeks hüpoteesideks, tulemusteks ja järeldusteks. Seega on andmeanalüüsi vahetuks sisendiks enamasti tabelid, mis sisaldavad arve (ja muud), ning vahetuks väljundiks joonised ja tabelid. Aga sellest hoolimata ei tegele andmeanalüüs primaarselt arvudega vaid teaduslike hüpoteesidega. Seega ei ole andmeanalüüs mitte matemaatika osa, vaid osa teaduslikust protsessist (mis kasutab matemaatikat tööriistana).

Me nimetame seda, mida me teeme andmeanalüüsiks, mitte statistikaks, selle pärast, et juhtida tähelepanu asjaolule, et valdav osa andmeanalüüsi töömahust ei kulu praktikas mitte statistilisete mudelitega töötamisele, ega isegi mitte jooniste tegemisele, vaid andmete puhastamisele, transformeerimisele ning analüüsiks sobivale kujule viimisele.

Andmeanalüüsi etapiks on katseskeem juba lõplikult valmis - me teame, mis on kontroll- ja mis on katsetingimused, mitu korda ja millisel tasemel on katset korratud jne - katsed lõpetatud ja tulemused üles tähendatud. Kui läbi viidud katse või selle dokumenteerimine ei ole analüüsiks optimaalne, on enamasti juba hilja (loe: kulukas) midagi muuta. Seega oleks hea, kui andmeanalüüsi

plaan valmiks paralleelselt katse plaaniga ja me teaksime juba enne katse alustamist, kuidas selle tulemusi analüüsima hakkame. Ideaalis mängime me juba enne katse alustamist analüüsi läbi simuleeritud andmetel, mis võimaldab meil näiteks otsustada, mitu korda on mõtekas päris katset korrata.

Kahjuks on eelnevad soovid enamasti vaid utoopia: inimesed, kes katset planeerivad ei mõtle sageli üldse sellele, kuidas hiljem tulemusi analüüsida. Mida keerukam katseskeem või suurem andmehulk, seda suurema tõenäosusega antakse andmed analüüsida inimesele, kes ei osalenud katse planeerimisel ja kes sageli ei ole ka spetsialist teaduses, millest antud katseskeem võrsus.

0.3.1 Mida andmeanalüütik peaks küsima, enne kui ta alustab.

1. Teaduslik taust - millistele teaduslikele küsimustele tahetakse vastust leida, ja millistel küsimustel on juba vastus olemas (neile andmete põhjal vastuse otsimine oleks ajaraiskamine).
2. Katse taust - miks on tehtud just selline katse/vaatlus ja mitte teistsugune. Kas katse on väga kallis/ajakulukas või saab seda vajadusel modifitseerida ja üle teha?
3. Katse skeem - mis on kontroll- ja mis on katsetingimused. Kas me soovime võrrelda erinevaid katsetingimusi omavahel või piirdume katse-kontroll võrdlustega? Mitu korda on katset korratud. Kas tegu on tehniliste ja/või teaduslike korduskatsetega? [Milline on selle katse üldistusjõud?] Kas katseskeem on hierarhilise struktuuriga või mitte? [Kas sama mõõtmisobjekti on mõõdetud mitu korda? Kas katses on võimalikke batch-i-efekte, mida saab tuvastada?]
4. Mida on mõõdetud ja mida me teame mõõtmisaparatuuri, mõõtmistäpsuse ja -hajuvuse kohta.
5. Varieeruvus - Kas mõõtmisviga on suur või väike võrreldes mõõtmisobjektide loomuliku ja teaduslikult huvi-

tava varieeruvusega. Kas andmete üldine varieeruvus on ootuspärane või üllatavalt väike/suur? Kas meil on põhjust arvata, et katse ja kontrolltingimustel võiksid olla tulemused erineva varieeruvuse määraga (ja/või erineva jaotusega)?

6. Kas me mõõdame teaduslikus mõttes õiget asja või tuleb mõõtmistulemusi kuidagi transformeerida, et need muutuks teaduslikult huvitavaks (näiteks kaal ja pikkus transformeerida kehamassiindeksiks).

7. Millisel kujul on meie poolt analüüsitav mõõtmistulemus (pidev suurus, diskreetne suurus, kahe arvu suhe, faktor, suunaline faktor jms) ja milline võiks olla selle jaotus (Poisson, binoom, normaal, lognormaal jms).

8. Mis on analüüsi vahetu eesmärk. Kas me soovime saada sissevaadet bioloogilisse mehhanismi, mis meie andmed genereeris, või soovime hoopis ennustada mingi muutuja väärtusi, teiste muutujate väätruste põhjal? Kas rõhuasetus on formaalsetel mudelitel, mis testivad olemasolevaid hüpoteese, või eksploratoorsel graafilisel analüüsil, eesmärgiga genereerida uusi hüpoteese? Kas tegemist on pigem eelkatsega?

9. Milline on andmete struktuur ja kvaliteet? Kas meil on muutujaid, mida tasuks välja visata või ühendada (seesama kehamassiindeksi näide)? Kas me peame erinevaid andmetabeleid ühendama või vastupidi lahku ajama? Kui palju on meil puuduvaid andmeid, kuidas on need kodeeritud ja miks nad puuduvad? Kas me peaksime proovima puuduvaid andmeid imputeerida? Kas andmetes on selgeid vigu (absurdseid väärtusi)? Kui kõrge või madal on andmestiku üldine kvaliteet?

10. Kas me saame aru, millised andmed on igas veerus - mida tähendab kodeering “sugu: 3” või “vanus: 99”?

11. Kes on andmete omanik (inimene, kes teeb teie analüüsi põhjal teaduslikke järeldusi)? Kas talle on võimalik sel-

> gitada, millised on analüüsi tulemused ja kas ta suudab neid kasutada teaduslike järelduste formuleerimisel? Teie analüüsi tarbija ei ole mitte niivõrd teadusartikli lugeja vaid inimene, kes selle artikli kirjutab. Ja kui ta ei suuda teie tööd kasutada, ei ole teil põhjust sellele oma aega raisata. Paljud teadlased usuvad pikale elukogemusele toetudes, et statistika on retooriline vahend, mis toodab p väärtusi, mis avavad ajakirjade uksed. Nende jaoks on sisuline andmeanalüüs nagu jõuluvana, maagiline tegelane, kellest küll räägime lastele, aga kellele täiskasvanud meeleldi oma aega ei kuluta.

Kokkuvõtteks: andmeanalüüs ei ole lihtne ega automaatne protsess, millel oleks üks õige viis, kuidas seda teha. Vähegi keerulisema ülesande korral võite olla üsna kindel, et leidub sama palju mõistlikke andmeanalüüsi töövooge, kui palju on leida mõistlikke andmeanalüütikuid.

Andmeanalüüsis on lihtne teha ausaid vigu koodikirjutamise näpukatest sobimatute mudelite rakendamiseni. Selle pärast on tähtis kontrollida, et iga koodirida ikka teeks seda, mida te arvasite seda tegevat. Teine oluline punkt on, et teie kood peaks olema raprodutseeritav, mis tähendab, et iga soovija peaks suutma algse andmetabeli olemaolul teie koodi jooksutada algusest lõpuni ilma veateateid saamata ja saama teiega identsed (või väga sarnased) tulemused. Siis on suurem võimalus, et vead avastatakse ja parandatakse. Pane tähele, et on olemas BioArxiv, kuhu saab oma käsikirja ja koodi talletada ammu enne avaldamist mõnes eelretsenseeritavas ajakirjas. See võimaldab fikseerida oma avastuste prioriteet, aga ka anda teadusüldsusele võimalus leida ja parandada vead, enne kui need suurt piinlikust valmistama hakkavad.

`Analüüsi reprodutseeritavuseks on vaja kolme asja. Algne andmetabel näit .csv laiendiga, R-i kood .Rmd või .R laiendiga ja ReadMe dokument, kus on kirjas mis on mis andmetabelis, pluss katseskeemi kirjeldus.`

0.4 Tarkvaratööriistad

0.4.1 Installeeri vajalikud programmid

Praktiline kursus eeldab töötavate R, RStudio ja Git programmide olemasolu sinu arvutist. Kõik on väga lihtsad installid.

1. Googelda "install R" või mine otse R allalaadimise veebilehele, laadi alla ja installi sobiv versioon.
2. Googelda "install RStudio" või mine otse RStudio allalaadimise veebilehele, laadi alla ja installi sobiv versioon.
3. Googelda "install git" või mine otse Git allalaadimise veebilehele, laadi alla ja installi sobiv versioon.

0.4.2 Loo GitHubi konto

GitHub on veebipõhine versioonikontrolli repositoorium ja veebimajutuse teenus.

- konto loomiseks mine lehele https://github.com. Loo endale oma nimega seotud avalik konto. Tulevikule mõeldes vali kasutajanimi hoolikalt. Ära muretse detailide pärast, need on võimalik täita hiljem.
- Loo repo nimega `intro_demo`.
- Lisa repole lühike ja informatiivne kirjeldus.
- Vali "Public".
- Pane linnuke kasti "Initialize this repository with a README".
- Klikka "Create Repository".

0.4.3 Loo uus R projekt

NB! Loo kataloogide nimed ilma tühikuteta. Tühikute asemel kasuta alakriipsu "_".

4. Ava RStudio (R ise töötab taustal ja sa ei pea seda kunagi ise avama)
5. Ava RStudio akna (Joonis 1) paremalt ülevalt nurgast "Project" menüüst "New Project" dialoog.
6. Ava "New Directory" > "Empty Project" > vali projekti_nimi ja oma failisüsteemi alamkataloog kus see projekti kataloog asuma hakkab. Meie kursusel pane projekti/kataloogi nimeks "rstats2017".

Rohkem infot R projekti loomise kohta leiad RStudio infoleheküljelt: Using Projects.

0.4.4 Git *Merge* konfliktid

Kollaboreerides üle GitHubi tekivad varem või hiljem konfliktid projekti failide versioonide vahel nn. "merge conflicts", mille lahendamine on väga oluline.

- Oma repo GitHubi veebilehel muuda/paranda README.md dokumenti ja "Commit"-i seda lühisõnumiga mis sa muutsid/parandasid.
- Seejärel, muuda oma arvutis olevat README.md faili RStudio-s viies sinna sisse mingi teistsuguse muudatuse. Tee "Commit" oma muudatustele.
- Proovi "push"-ida – sa saad veateate!
- Proovi "pull".
- Lahenda "merge" konflikt ja seejärel "commit" + "push".

Githubi veateadete lugemine ja Google otsing aitavad sind.

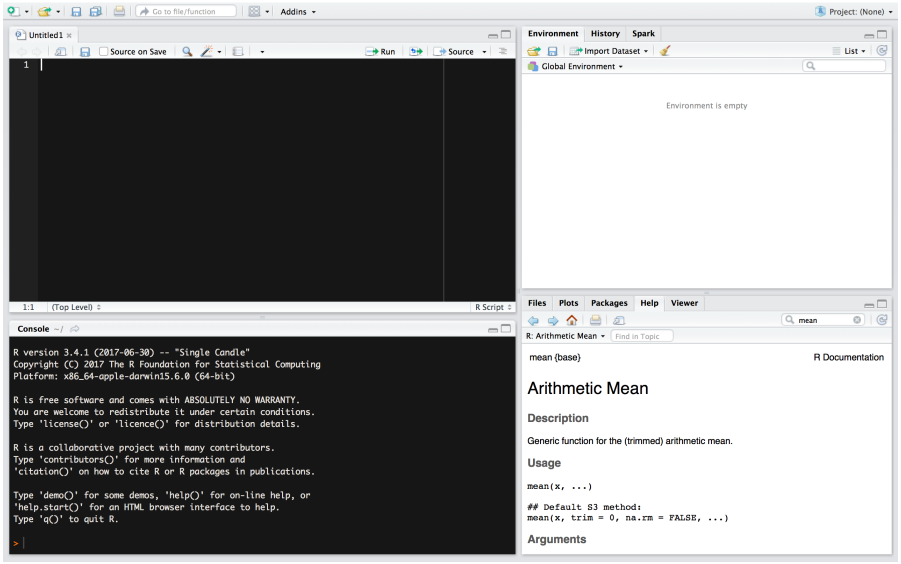


FIGURE 1 RStudio konsoolis on neli akent. Üleval vasakul on sinu poolt nimega varustatud koodi ja teksti editor kuhu kirjutad R skripti. Sinna kirjutad oma koodi ja kommentaarid sellele. All vasakul on konsool. Sinna sisestatakse käivitamisel sinu R kood ja sinna trükitakse väljund. Üleval paremal on Environment aken olulise sakiga . Seal on näha R-i objektid, mis on sulle töökeskkonnas kättesaadavad ja millega sa saad töötada. menüüs on võimalik muutusi vaadata ja 'commit'ida ja -ga suhelda. All paremal on paneel mitme sakiga. Files tab töötab nagu failihaldur. Kui sa lood või avad R projekti, siis näidatakse seal vaikimisi sinu töökataloogi. Kui kasutad R projekti, siis ei ole vaja töökataloogi eraldi seadistada. Plots paneelile ilmuvad joonised, mille sa teed. Packages näitab sulle sinu arvutis olevaid R-i pakette ehk raamatukogusid. Help paneeli avanevad help failid (ka need, mida konsooli kaudu otsitakse).

0.4.5 R projekti kataloogi soovitatav minimaalne struktuur

Iga R projekt peab olema täiesti iseseisev (*selfcontained*) ja sisaldama andmeid ja koodi, mis on koos piisavad projektiga seotud arvutuste läbi viimiseks ja raporti genereerimiseks. Kõik faili *path*-id peavad olema suhtelised. -TAAVI, SEE TULEKS LAHTI KIRJUTADA.

R projekti kataloog peaks sisaldama projekti kirjeldavaid faile, mis nimetatakse DESCRIPTION ja README.md. **DESCRIPTION** on tavaline tekstifail ja sisaldab projekti metainfot ja infot projekti sõltuvuste kohta, nagu väliste andmesettide asukoht, vajalik tarkvara jne. **README.md** on markdown formaadis projekti info, sisaldab juhendeid kasutajatele. Igale GitHubi repole on soovitav koostada README.md, esialgu kasvõi projekti pealkiri ja üks kirjeldav lause. README.md ja DESCRIPTION asuvad projekti juurkataloogis.

> Projekti juurkataloogi jäävad ka kõik .Rmd laiendiga teksti ja analüüsi tulemusi sisaldavad failid, millest genereeritakse lõplik raport/dokument.

Suuremad projektid, nagu näiteks teadusartikkel või raamat, võivad sisaldada mitmeid Rmd faile ja võib tekkida kange kisatus need mõnda alamkataloogi tõsta. Aga `knitr::knit()`, mis Rmarkdowni markdowniks konverteerib, arvestab, et Rmd fail asub juurkataloogis ja arvestab juurkataloogi suhtes ka failis olevaid *path*-e teistele failidele (näiteks "data/my_data.csv").

data/ kataloog sisaldab faile toorandmetega. Need failid peavad olema R-i poolt loetavad ja soovitavalt tekstipõhised, laienditega TXT, CSV, TSV jne. Neid faile ei muudeta, ainult loetakse. Kogu algandmete töötlus toimub programmaatiliselt. Suured failid muudavad versioonikontrolli aeglaseks, samuti on suheliselt mõttetu versioonikontroll binaarsete failide korral (MS näiteks), sest diffid pole lihtsalt inimkeeles. Github ütleb suurte failide kohta nii: *"GitHub will warn you when pushing files larger than 50 MB. You will not be allowed to push files larger than 100 MB."*

R/ kataloog sisaldab R skripte.

output/ kataloogis on kasutaja poolt genereeritud andmed ja puhastatud andmesetid.

```
project/
|- DESCRIPTION          # project metadata and dependencies
|- README.md            # description of contents and guide to users
|- my_analysis.Rmd      # markdown file containing analysis
|                       # writeup together with R code chunks
|
|- data/                # raw data, not changed once created
|  +-my_rawdata.csv     # data files in open formats,
|                       # such as TXT, CSV, TSV etc.
|
|- R/                   # any programmatic code
|  +-my_scripts.R       # R code used to analyse and
|                       # visualise data
|
|- output/              # generated data
|  +-my_clean_data.rds  # saved R object or cleaned up datasets
```

Kui selles kataloogis olevad skriptid on annoteeritud kasutades Roxygen-i (Wickham, Danenberg, and Eugster 2017), siis genereeritakse automaatselt funktsioonide dokumentatsioon kataloogi **man/**. Rohkem projekti pakkimise kohta loe värskest preprindist "Packaging data analytical work reproducibly using R" (Marwick, Boettiger, and Mullen 2017).

0.4.6 Pakettide installeerimine

R *library*-d ehk paketid sisaldavad ühte või enamat mingit kindlat operatsiooni läbi viivat funktsiooni. **R baaspakett sisaldab juba mitmeid funktsioone.** Kõige esimene sõnum `sum()` help lehel on "sum {base}", mis tähendab, et see funktsioon kuulub nn. baasfunktsioonide hulka. Need funktsioonid on alati kättesaadavad sest neid sisaldavad raamatukogud laetakse vaikimisi teie töökeskkonda. Näiteks "base" raamatukogu versioon 3.6.1 sisaldab 457 funktsiooni. Enamasti on sarnaseid asju tegevad funktsioonid koondatud kokku

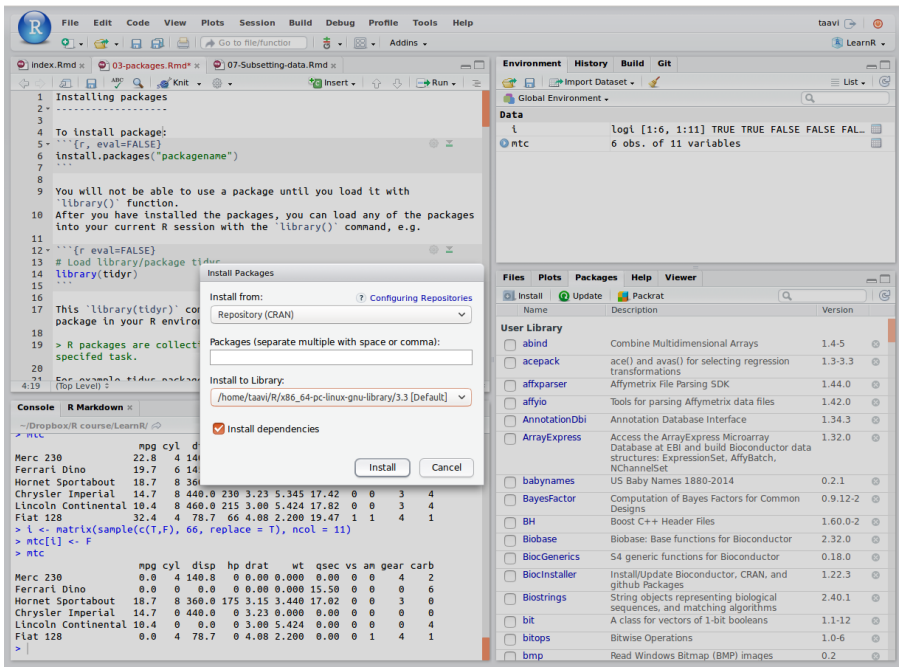


FIGURE 2 RStudio 'Install Packages' dialoogiaken.

raamatukogudesse ehk pakettidesse, mis tuleb eraldi R kesksest repositooriumist CRAN alla laadida ja installeerida.

Selleks, et installeerida pakett, sisesta järgnev käsurida R konsooli:

```
## eg use "ggplot2" as packagename
install.packages("packagename")
```

NB! Kui mõni raamatukogu sel viisil alla ei tule, siis guugeldage selle nime + R ja vaadake instruktsioone installeerimiseks. Suure tõenäosusega on tegemist mõnes teises repos (näiteks Bioconductor) või ainult GitHubis asuva paketiga.

RStudio võimaldab ka *point-and-click* stiilis pakettide installeerimist:

> Sa ei saa installeeritud pakette enne kasutada, kui laadid nad töökeskkonda kasutades `library()` funktsiooni.

Peale installeerimist lae pakett oma R sessiooni kasutades `library()` käsku, näiteks:

```
## Load library/package dplyr
library(dplyr)
```

`library(dplyr)` käsk teeb R sessioonis kasutatavaks kõik "dplyr" paketi funktsioonid.

Näiteks "dplyr" pakett sisaldab 267 funktsiooni:

```
library(dplyr)
## let's look at the head of package list
head(ls("package:dplyr"), 20)
#>  [1] "%>%"           "add_count"     "add_count_"
#>  [4] "add_row"       "add_rownames"  "add_tally"
#>  [7] "add_tally_"    "all_equal"     "all_vars"
#> [10] "anti_join"     "any_vars"      "arrange"
#> [13] "arrange_"      "arrange_all"   "arrange_at"
#> [16] "arrange_if"    "as_data_frame" "as_label"
#> [19] "as_tibble"     "as.tbl"
```

Konfliktide korral eri pakettide sama nimega funktsioonide vahel saab `::` operaatorit kasutades kutsuda välja/importida funktsiooni spetsiifilisest paketist:

```
dplyr::select(df, my_var)
```

Sellisel kujul funktsioonide kasutamisel pole vaja imporditavat funktsiooni sisaldavat raamatukogu töökeskkonda laadida.

Funktsioonide-pakettide help failid RStudio kasutajaliidesest: Kui te lähete RStudios paremal all olevale "Packages" tabile, siis on võimalik klikkida raamatukogu nimele ja näha selle help-faile, tutooriale ja kõiki selle raamatukogu funktsioone koos nende help failidega.

0.4.7 R repositooriumid

R pakid on saadaval kolmest põhilisest repositooriumist:

1. **CRAN** https://cran.r-project.org

```
install.packages("ggplot2")
```

2. **Bioconductor** https://www.bioconductor.org

```
# First run biocLite script fron bioconductor.org
source("https://bioconductor.org/biocLite.R")
# use 'http' in url if 'https' is unavailable.
biocLite("edgeR")
```

3. **GitHub** https://github.com

```
# following command installs xaringan (presentation ninja) package from Git
devtools::install_github("yihui/xaringan")
```

NB! antud praktilise kursuse raames tutvume ja kasutame 'tidyverse' metapaketi funktsioone, laadides need iga sessiooni alguses:

```
# install.packages("tidyverse")
library(tidyverse)
```

Nüüd on teil `tidyverse` pakett arvutis. Tegelikult kuuluvad siia raamatukokku omakorda tosinkond raamatukogu — tidyverse on pisut meta. Igal juhul muutuvad selle funktsioonid kättesaadavaks peale seda, kui te need töökeskkonda sisse loete

Veel üks tehniline detail. `library(tidyverse)` käsk ei loe sisse kõiki alam-raamatukogusid, mis selle nime all CRAN-ist alla laaditi. Need tuleb vajadusel eraldi ükshaaval sisse lugeda.

Enamasti kirjutatakse kõik sisse loetavad raamatukogud kohe R scripti algusesse. Siis on ilusti näha, mida hiljem vaja läheb.

0.4.7.1 R-i versiooni uuendamine

mac-is

1. installeeri uus Ri versioon
2. Kopeeri folderid oma vanast R-i versioonist uude versiooni (tegelikult need folderid, mida uues kataloogis ei ole - sest mõned paketid lastakse peale juba baas installatsiooniga).

/Library/Frameworks/R.framework/Versions/x.xx/Resources/library

(Asenda x.xx vana ja uue R versiooni numbriga)

installed.packages() aitab leida kataloogi, kus asuvad sinu paketid (näit kui eelnev rada ei tööta)

3. Uuenda paketid uues kataloogis

update.packages(checkBuilt=TRUE)

R konsoolis pead vastama y igale küsimusele, mida seal küsitakse

4. kontrolli, et kõik töötas:

packageStatus()

windowsis peaks töötama

install.packages("installr")

library(installr)

updateR()

0.5 R on kalkulaator

Liidame `2 + 2`.

```
2 + 2
#> [1] 4
```

Nüüd trükiti see vastus konsooli kujul `[1] 4`. See tähendab, et `2 + 2 = 4`.

Kontrollime seda:

```
## liidame 2 ja 2 ning vaatame kas vastus võrdub 4
answer <- (2 + 2) == 4
## Trükime vastuse välja
answer
#> [1] TRUE
```

Vastus on TRUE, (logical).

Pane tähele, et aritmeetiline võrdusmärk on `==` (sest = tähendab hoopis väärtuse määramist objektile/argumendile).

Veel mõned näidisarvutused:

```
## 3 astmes 2; Please read Note ?'**'
3 ^ 2 # 3**2 also works
## Ruutjuur 3st
sqrt(3)
## Naturaallogaritm sajast
log(100)
```

Arvule π on määratud oma objekt `pi`. Seega on soovitav enda poolt loodavatele objektidele mitte panna nimeks “pi”.

```
## Ümarda pi neljale komakohale
round(pi, 4)
#> [1] 3.14
```

Ümardamine on oluline tulemuste väljaprintimisel.

```
library(tidyverse)
library(VIM)
library(readxl)
library(skimr)
```

0.6 R-i tööpõhimõte

Iga kord, kui avate R studio, alustate R-i sessiooni, mis seisneb funktsioonide (väikeste programmijuppide) rakendamises andmetele eesmärgiga muuta tabelite struktuuri, transformeerida andmeid, genereerida jooniseid ja/või arvutada statistikuid. Kõik see toimub R-i töökeskkonnas. Tulemusi saab töökeskkonnast eksportida arvuti kõvakettale näiteks .pdf (joonised) või .csv (tabelid) formaadis. Teie sessiooni põhiline tulemus ei ole siiski eksporditavad asjad, vaid R-i kood (script), mida jooksutades on võimalik korrataval viisil algset andmetabelit manipuleerida. Kõik, mida te töökeskonnas teete, kajastub koodis ja on korratav.

R-i sessioon näeb üldjoontes välja niimoodi:

(1) Andmed ja funktsioonid (raamatukogude kujul) loetakse R-i töökeskkonda, (2) andmeid töödeldakse funktsioonide abil ja (3) tööproduktid eksporditakse/salvestatakse teistele programmidele kättesaadavasse vormi.

- töökeskkond (workspace) - sisaldab üles laetud objekte
- keskkond (environment) - sisaldab nimega seotud objekte
- pakett e raamatukogu - sisaldab funktsioone, andmeid ja seletavaid faile
- funktsioon - R-i alamprogramm
- argument - funktsiooni tööd reguleeriv parameeter
- objekt - funktsioon, andmestruktuur ja mida iganes saab töökeskkonda viia
- nimi - objekt seotakse nimega <- abil, et see keskkonnas nähtavaks/taaskasutatavaks muuta

Andmetüübid:

- character - tähemärgid

- numeric (double) - ratsionaalarvud
- integer - täisarvud
- factor - muutuja, millel on loetud hulk nimedega "tasemeid" (levels), kus iga tase on sisemiselt kodeeritud täisarvuga alates 1st. On olemas nii järjestatud kui järjestamata tasemetega faktorid. Näit "mees"-naine on tüüpiliselt järjestamata tasemed, aga vähe-keskmiselt-palju peaks olema järjestatud.
- logical - TRUE/FALSE. TRUE on sisemiselt kodeeritud kui 1 ja FALSE kui 0.

relatsioonilised operaatorid

- , >= (on suurem või võrdne), <, <=,
- == (võrdub),
- != (ei võrdu),
- %in% (sisaldub),
- & (and),
- (or)

```
1 == 2
#> [1] FALSE
1 != 2
#> [1] TRUE

1 %in% 1:5
#> [1] TRUE
1&4 %in% 1:5 #1 ja 4 sisalduvad vektoris c(1, 2, 3, 4, 5) --- TRUE
#> [1] TRUE
c(1, 8) %in% 1:5
#> [1]  TRUE FALSE
1|8 %in% 1:5 #1 või 8 sisalduvad vektoris c(1, 2, 3, 4, 5) --- TRUE
#> [1] TRUE
6 %in% 1:5 #6 sisaldub vektoris c(1, 2, 3, 4, 5) --- see lause on FALSE
#> [1] FALSE
char.vector <- c("apple", "banana", "cantaloupe", "dragonfruit")
```

```
"apple" %in% char.vector
#> [1] TRUE
```

andmeklassid: (idee on andmed struktureerida hõlbustamaks edasisi tehteid nendega)

- vektor - 1D järjestatud sama tüüpi andmed
- list - kokku kogutud, järjestatud ja nimetatud objektid (vektorid, andmeraamid, mudelid, teised listid jms)
- maatriks - 2D neljakandiline andmestruktuur, mis koosneb sama tüüpi andmetest
- andmeraam (data frame, tibble) - 2D neljakandiline andmestruktuur, mis koosneb veeru kaupa kõrvuti asetatud ühepikkustest vektoritest (erinevad vektorid võivad sisaldada erinevaid tüüpi andmeid). Lähim asi nelinurksele tabelile (aga Exceli tabelis – erinevalt R-st – saab samasse veergu panna ka erinevat tüüpi andmeid).

Avades R Studio peaksite nägema tühja keskkonda (Enviroment tab, ülal paremas aknas). R-i tööpõhimõte on järgmine.

(1) te laete võrgust alla ja installeerite oma kõvakettale vajalikud R-i alamprogrammid, mida kutsutakse pakettideks ehk raamatukogudeks. Paketid on funktsioonide (kuigi mitte ainult) kogud. Need funktsioonid ei ole aga veel R-s kasutamiseks kättesaadavad.

(2) te loete R-i töökeskkonda (workspace) sisse vajalikud paketid (oma kõvakettalt) ja andmetabeli(d), mida soovite töödelda (oma kõvakettalt või otse võrgust), muutes need seega R-s kasutatavaks. Seda tuleb teha igal R-i sessioonil uuesti. Igat asja, mis on töökeskkonnas, kutsutakse objektiks. Objektid, mis on omistatud nimele, ilmuvad nähtavale Enviroment tab-i, koos oma struktuuri lühikirjeldsega.

(3) Te sisestate töökeskkonnas olevatesse funtksioonidesse andmed ja argumendid. Sageli teeb üks funktsioon ühte toimingut, mistõttu koosneb teie töövoog funktsioonide

rakendamisest üksteise järel sellisel moel, et eelmise funktsiooni väljund on sisend järgmisele funktsioonile.

(4) Te ekspordite/salvestate oma kõvakettale need objektid (tabelid, joonised), mida soovite tulevikus avada teiste programmidega. Samuti salvestate oma põhilise töötulemuse - R-i scripti (näit .Rmd või .R laiendiga failina).

Järgmise sessiooni saate alustada juba oma salvestatud koodi baasilt — jooksutades algandmete peal olemasoleva koodi ning seejärel lisades uut koodi (andmetabeli ja raamatukogud tuleb iga sessiooni jaoks uuesti keskkonda sisse lugeda).

0.6.1 Funktsioon

- Hea funktsioon teeb ühte asja. Näiteks funktsioon `t()` (*t* tähendab *transpose*) transponeerib maatriksi nii, et ridadest saavad veerud ja vastupidi, aga funktsioon `c()` (*c* tähendab *combine* v *concatenate*) moodustab sisestatud objektidest vektori.
- Funktsiooni nime taha käivad sulud. Ilma sulgudeta funktsiooni nime jooksutamine annab väljundina selle funktsiooni koodi.
- Enamustel funktsioonidel on argumendid, mis käivad sulgude sisse ja on üksteisest komadega eraldatud. Kasutaja saab argumentidele anda väärtused, mis määravad andmed, millel funktsioon töötab, ja selle, mida funktsioon nende andmetega täpselt teeb. Funktsiooni iga argument täpsustab funktsiooni jaoks, mida teha.
- Osadel argumentidel on vaikeväärtused, mida saab käsitsi muuta. Vaikeväärtused, nagu ka funktsiooni argumentide nimekirja ja kirjelduse, leiab `?funktsiooni_nimi` (ilma sulgudeta) abil. NB! Ära kasuta funktsioone, mille argumente sa ei tunne.
- Argumendid võivad olla kas kohustuslikud (ilma argumendi väärtust sisestamata funktsioon ei tööta), või mitte. Näiteks funktsioon `plot(x, y, ...)` argumendid on objekt nimega x, mis annab x teljele plotitud andmete koordinaadid, objekt nimega

y, mis annab sama y teljele, ning lisaargumendid, mis võivad sõltuda x-i ja või y-i vormist. x on kohustuslik argument, aga y ei ole tingimata vajalik (kas y on vajalik või mitte, sõltub sisestatud x-i struktuurist).

- Kui argumendi väärtus sisestatakse teksti kujul, siis enamasti jutumärkides. Jutumärgid muudavad R-i jaoks teksti tähemärkide jadaks e stringiks, mille sees R numbreid ei tõlgenda arvudena.
- Argumendid on järjestatud ja neil on nimed. Nimi trumpab järjekorra üle selles mõttes, et me võime argumentide nimed funktsiooni kirjutada suvalises järjekorras ilma, et funktsiooni töö sellest muutuks. Samas, kui me sisestame funktsiooni argumendid ilma nimedeta, siis on argumentide järjekord tähtis, sest need seostatakse vaikimisi nimedega vastavalt oma järjekorranumbrile. Oletame, et meil on vektorid `kaal <- c(2.3, 4.3, 3)` ja `pikkus <- c(7, 5, 9)`. Me võime need funktsiooni sisestada nii: `plot(x = kaal, y = pikkus)`, `plot(y = pikkus, x = kaal)` ja `plot(kaal, pikkus)` teevad kõik identse scatterploti (aga `plot(pikkus, kaal)` ei tee).
- Funktsiooni esimese argumendi saab enamasti sisestada ka alternatiivsel viisil, %>% pipe operaatori abil. Niimoodi jooksevad `fun(arg1, arg2)` ja `arg1 %>% fun(arg2)` koodid enamasti identselt. Kumba koodi eelistada on seega "vaid" koodi loetavuse küsimus. `arg1 %>% fun(arg2)` on sama, mis `arg1 %>% fun(., arg2)`, kus punkt "." näitab vaikimisi, mitmenda argumendi kohale me oma arg1 sisse torutame. Seda teades on võimalik ka vorm `arg2 %>% fun(arg1,.)`, kus toru kaudu anname sisse 2. või ükskõik millise muu argumendi. Siin ei ole muud saladust kui, et peame punkti asukoha funktsioonis eksplitsiitselt ära näitama.

Ülesanne: uuri välja, mida määravad järgneva funktsiooni argumendid.

```
plot(table(rpois(100, 5)), type = "h", col = "red",
lwd = 10, main = "rpois(100, lambda = 5)")
```

Pane tähele, et funktsiooni "plot" argumendid "table" ja "rpois" on ka ise funktsioonid, millel on kummagil oma argumendid.

0.6.2 Sama koodi saab kirjutada neljal erineval viisil

Idee on sooritada järjest operatsioone nii, et eelmise operatsiooni väljund (R-i objekt) oleks sisendiks järgmisele operatsioonile (funktsioonile). See on lihtne hargnemisteta analüüsiskeem.

Kui me muudame olemasolevat objekti, siis me kas jätame muudetud objektile vana objekti nime või me anname talle uue nime. Esimesel juhul läheb eelmine muutmata objekt töökeskkonnast kaduma, aga nimesid ei tule juurde ja säilib töövoo sujuvus. Teisel juhul jäävad analüüsi vaheobjektid meile alles ja nende juurde saab alati tagasi tulla. Aga samas tekib palju sarnaste nimedega objekte.

0.6.2.1 Esimene võimalus - anname järjest tekkinud objektid samale nimele.

```
a <- c(2, 3)
a <- sum(a)
a <- sqrt(a)
a <- round(a, 2)
a
#> [1] 2.24
```

0.6.2.2 Teine võimalus - uued nimed.

Nii saab tekkinud objekte hiljem kasutada.

```
a <- c(2, 3)
a1 <- sum(a)
a2 <- sqrt(a1)
a3 <- round(a2, 2)
a3
#> [1] 2.24
```

0.6.2.3 Kolmas võimalus on lühem variant esimesest.

Me nimelt ühendame etapid toru `%>%` kaudu. Toru operaator ei ole siiski baas R-is kohe kättesaadav, vaid tuleb laadida kas **magrittr** või **dplyr** paketist (viimatinimetatu laadib selle funktsiooni ka vaikimisi esimesena nimetatud raamatukogust). Siin me võtame objekti "a" (nö. andmed), suuname selle funktsiooni `sum()`, võtame selle funktsiooni väljundi ja suuname selle omakorda funktsiooni `sqrt()`. Seejärel võtame selle funktsiooni outputi ja määrame selle nimele "result" (aga võime selle ka mõne teise nimega siduda). Kui mõni funktsioon võtab ainult ühe parameetri, mille me talle toru kaudu sisse sõõdame, siis pole selle funktsiooni taga isegi sulge vaja (R hea stiili juhised soovitavad siiski alati kasutada funktsiooni koos sulgudega).

See on hea lühike ja inimloetav viis koodi kirjutada, mis on masina jaoks identne esimese koodiga.

```
library(dplyr)
a <- c(2, 3)
result <- a %>% sum() %>% sqrt() %>% round(2)
result
#> [1] 2.24
```

0.6.2.4 Neljas võimalus, klassikaline baas R lahendus:

```
a <- c(2, 3)
result <- round(sqrt(sum(a)), 2)
result
#> [1] 2.24
```

Sellist koodi loetakse keskelt väljappoole ja kirjutatakse alates viimasest operatsioonist, mida soovitakse, et kood teeks. Masina jaoks pole vahet. Inimese jaoks on küll: 4. variant nõuab hästi pestud ajusid.

Koodi lühidus 4 –> 3 –> 1 –> 2 (pikem) Lollikindlus 2 –> 1 –>

3 –> 4 (vähem lollikindel) Loetavus 3 –> 2 –> 1 –> 4 (halvemini loetav)

See on teie otsustada, millist koodivormi te millal kasutate, aga te peaksite oskama lugeda neid kõiki.

0.6.3 objekt

R-i töökeskkonnas "workspace" asuvad **objektid**, millega me töötame. Igal objektil on nimi, mille abil saab selle objektiga opereerida (teda argumendina funktsioonidesse sisestada). Tüüpilised objektid on:

- Vektorid, maatriksid, listid ja andmeraamid.
- Statistiliste analüüside väljundid (mudeliobjektid, S3, S4 klass).
- Funktsioonid.

Funktsioon `ls()` annab objektide nimed teie workspace-s.

`rm(a)` eemaldab objekti nimega a töökeskkonnast.

Selleks, et salvestada töökeskkond faili, kasuta "Save" nuppu "Environment" akna servast või menüüst "Session" -> "Save Workspace As".

Projekti sulgemisel salvestab RStudio vaikimisi töökeskkonna. **Parema reprodutseeritavuse huvides pole siiski soovitav töökeskkonda peale töö lõppu projekti sulgemisel salvestada!**. Lülitame automaatse salvestamise välja:

- Selleks mine "Tools" > "Global Options" > kõige ülemine, "R General" menüüs vali "Save workspace to .RData on exit" > "Never" ever!
- Võta ära linnuke "Restore .RData to workspace at startup" eest.

Kui on mingid kaua aega võtvad kalkulatsioonid või allalaadimised, salvesta need eraldi .rds faili ja laadi koodis vastavalt vajadusele: `write_rds()`, `read_rds()`.

0.6.3.1 Objekt ja nimi

Kui teil sünnib laps, annate talle nime. R-s on vastupidi: nimele antakse objekt

```
babe <- "beebi"
babe
#> [1] "beebi"
```

Siin on kõigepealt nimi (babe), siis assigneerimise sümbol `<-` ja lõpuks objekt, mis on nimele antud (string "beebi").

NB! Stringid on jutumärkides, nimed mitte. Nimi üksi evalueeritakse kui käsk: "print object". Antud juhul trükitakse konsooli string "beebi"

Nüüd muudame objekti nime taga:

```
babe <- c("saatan", "inglike")
babe
#> [1] "saatan"  "inglike"
```

Tulemuseks on sama nimi, mis tähistab nüüd midagi muud (vektorit, mis koosneb 2st stringist). Objekt "beebi" kaotas oma nime ja on nüüd workspacest kadunud. `class()` annab meile objekti klassi.

```
class(babe)
#> [1] "character"
```

Antud juhul character.

> Ainult need objektid, mis on assigneeritud nimele, lähevad workspace ja on sellistena kasutatvad edasises analüüsis.

```
apples <- 2
bananas <- 3
apples + bananas
#> [1] 5
```

Selle ekspressiooni tulemus trükitakse ainult R konsooli. Kuna teda ei määrata nimele, siis ei ilmu see ka workspace.

```
a <- 2
b <- 3
a <- a + b
# objekti nimega 'a' struktuur
str(a)
#>  num 5
```

Nüüd on nimega a seostatud uus objekt, mis sisaldab numbrit 5 (olles ühe elemendiga vektor). Ja nimega a eelnevalt seostatud objekt, mis koosnes numbrist 2, on workspacest lahkunud.

0.6.3.1.1 Nimede vorm

- Nimed algavad ingliskeelse tähemärgiga, mitte numbriga ega $€%&/?~ˇöõüä
- Nimed ei sisalda tühikuid
- Tühiku asemel kasuta alakriipsu: näiteks eriti_pikk_nimi
- SUURED ja väiksed tähed on nimes erinevad
- Nimed peaksid kirjeldama objekti, mis on sellele nimele assigneeritud ja nad võivad olla pikad sest TAB klahv annab auto-complete.
- alt + - on otsetee <- jaoks

0.6.4 Andmete tüübid

- numeric / integer
- logical – 2 väärtust TRUE/FALSE
- character
- factor (ordered and unordered) - 2+ diskreetset väärtust, mis võivad olla järjestatud suuremast väiksemani (aga ei asu üksteisest võrdsel kaugusel). Faktoreid käsitleme põhjalikumalt hiljem.

Faktoritel on tasemed (level) ja sisemiselt on iga faktori tase tähistatud täisarvulise numbriga.

Andmete tüüpe saab üksteiseks konverteerida `as.numeric()`, `as.character()`, `as.factor()`.

```
a <- 5:10
#vektor, mis koosneb 6st täisarvust 5st 10-ni
class(a)
#> [1] "integer"

a_char <- c("5", "6", "7")
#jutumärgid tähistavad tähemärki, mitte arvu.
class(a_char)
#> [1] "character"

a1 <- as.factor(a)
a1
#> [1] 5  6  7  8  9  10
#> Levels: 5 6 7 8 9 10
a2 <- as.numeric(a1)
#see ei tööta, sest faktori tasemed
#rekodeeritakse sisemiselt numbritena alates 1st.
a2
#> [1] 1 2 3 4 5 6
a3 <- as.numeric(as.character(a1))
#see töötab, taastab numbrid 5st 10-ni
#kõigepealt konverteerime faktori tasemed tähemärkideks
#(ignoreerides sisemisi rekodeeringuid).
#Seejärel konverteerime tähemärgid numbriteks.
a3
#> [1]  5  6  7  8  9 10
```

0.6.5 Objektide klassid

0.6.5.1 Vektor

Vektor on rida kindlas järjekorras arve, tähemärkide stringe või TRUE/FALSE loogilisi väärtusi. Iga vektor ja maatriks (mis on 2D vektor) sisaldab ainult ühte tüüpi andmeid. Vektor on elementaarüksus, millega me teeme tehteid. Andmetabelis ripuvad kõrvuti ühepikad vektorid (üks vektor = üks tulp) ja R-le meeldib arvutada

vektori kaupa vasakult paremale (mis tabelis on ülevalt alla sest vektori algus on üleval tabeli peas). Pikema kui üheelemendise vektori loomiseks kasuta funktsiooni `c()` – combine

Loome numbrilise vektori ja vaatame ta struktuuri:

```
minu_vektor <- c(1, 3, 4)
str(minu_vektor)
#>  num [1:3] 1 3 4
```

Loome vektori puuduva väärtusega, vaatame vektori klassi:

```
minu_vektor <- c(1, NA, 4)
minu_vektor
#> [1]  1 NA  4
class(minu_vektor)
#> [1] "numeric"
```

Klass jääb *numeric*-uks.

Kui vektoris on segamini numbrid ja stringid, siis muudetakse numbrid ka stringideks:

```
minu_vektor <- c(1, "2", 2, 4, "joe")
minu_vektor
#> [1] "1"   "2"   "2"   "4"   "joe"
class(minu_vektor)
#> [1] "character"
```

Piisab ühest "tõrvatilgast meepotis", et teie vektor ei sisaldaks enam numbreid.

Eelnevast segavektorist on võimalik numbrid päästa kasutades käsku `as.numeric()`:

```
as.numeric(minu_vektor)
#> Warning: NAs introduced by coercion
#> [1]  1  2  2  4 NA
```

Väärtus "joe" muudeti NA-ks, kuna seda ei olnud võimalik numbriks muuta. Samuti peab olema tähelepanelik faktorite muutmisel numbriteks:

```
minu_vektor <- factor(c(9, "12", 12, 1.4, "joe"))
minu_vektor
#> [1] 9   12  12  1.4 joe
#> Levels: 1.4 12 9 joe
class(minu_vektor)
#> [1] "factor"
## Kui muudame faktori otse numbriks, saame faktori taseme numbri
as.numeric(minu_vektor)
#> [1] 3 2 2 1 4
```

Faktorite muutmisel numbriteks tuleb need kõigepealt stringideks muuta:

```
as.numeric(as.character(minu_vektor))
#> Warning: NAs introduced by coercion
#> [1]  9.0 12.0 12.0  1.4   NA
```

Järgneva trikiga saab stringidest kätte numbrid:

```
minu_vektor <- c(1, "A2", "$2", "joe")
## parse_number() is imported from tidyverse 'readr'
minu_vektor <- parse_number(minu_vektor) %>% as.vector()
#> Warning: 1 parsing failure.
#> row col expected actual
#>   4  -- a number    joe
str(minu_vektor)
#>  num [1:4] 1 2 2 NA
```

R säilitab vektori algse järjekorra. Sageli on aga vaja tulemusi näiteks vaatamiseks ja presenteerimiseks sorteerida suuruse või tähestiku järjekorras:

```
## sorts vector in ascending order
sort(x, decreasing = FALSE, ...)
```

Vektori unikaalsed väärtused saab kätte käsuga `unique()`:

```
## returns a vector or data frame, but with duplicate elements/rows removed
unique(c(1,1,1,2,2,2,2,2,3,3,4,5,5))
#> [1] 1 2 3 4 5
```

Uus vektor automaatselt: `seq()` ja `rep()`

seq annab kasvava või kahaneva rea. rep kordab väärtusi.

```
seq(2, 3, by = 0.5)
#> [1] 2.0 2.5 3.0
seq(2, 3, length.out = 5)
#> [1] 2.00 2.25 2.50 2.75 3.00
rep(1:2, times = 3)
#> [1] 1 2 1 2 1 2
rep(1:2, each = 3)
#> [1] 1 1 1 2 2 2
rep(c("a", "b"), each = 3, times = 2)
#>  [1] "a" "a" "a" "b" "b" "b" "a" "a" "a" "b" "b" "b"
```

0.6.5.1.1 Tehted arvuliste vektoritega

Vektoreid saab liita, lahutada, korrutada ja jagada.

```
a <- c(1, 2, 3)
b <- 4
a + b
#> [1] 5 6 7
```

Kõik vektor a liikmed liideti arvuga 3 (kuna vektor b koosnes ühest liikmest, läks see kordusesse)

```
a <- c(1, 2, 3)
b <- c(4, 5)
a + b
#> Warning in a + b: longer object length is not a
#> multiple of shorter object length
#> [1] 5 7 7
```

Aga see töötab veateatega, sest vektorite pikkused ei ole üksteise kordajad 1 + 4; 2 + 5, 3 + 4

```
a <- c(1, 2, 3, 4)
b <- c(5, 6)
a + b
#> [1]  6  8  8 10
```

See töötab: 1 + 5; 2 + 6; 3 + 5; 4 + 6

```
a <- c(1, 2, 3, 4)
b <- c(5, 6, 7, 8)
a + b
#> [1]  6  8 10 12
```

Samuti see (ühepikkused vektorid — igat liiget kasutatakse üks kord)

```
a <- c(TRUE, FALSE, TRUE)
sum(a)
#> [1] 2
mean(a)
#> [1] 0.667
```

Mis siin juhtus? R kodeerib sisemiselt TRUE kui 1 ja FALSE kui 0-i. summa 1 + 0 + 1 = 2. Mean seevastu võtab ühtede summa (TRUE elementide arvu) suhte vektori elementide arvust ja annab seega TRUE väärtuste suhtarvu. Seda loogiliste väärtuste omadust õpime varsti praktikas kasutama.

0.6.5.2 List

List on objektitüüp, kuhu saab koondada kõiki teisi objekte, kaasa arvatud listid. R-i jaoks on list lihtsalt vektor, mille elemendid ei pean olema sama andmetüüpi (nagu tavalistel nn lihtsatel vektoritel).
Praktikas kasutatakse listi enamasti lihtsalt erinevate R-i objektide koos hoidmiseks ühes suuremas meta-objektis. List on nagu jõuluvana kingikott, kus kommid, sokipaarid ja muud kingid segamini kolisevad. Listidega töötamist vaatame lähemalt veidi hiljem.

Näiteks list, kus on 1 vektor nimega a, 1 tibble nimega b ja 1

list nimega c, mis omakorda sisaldab vektorit nimega d ja tibblet nimega e. Seega on meil tegu rekursiivse listiga.

```
# numeric vector a
a <- runif(5)
# data.frame
ab <- data.frame(a, b = rnorm(5))
# linear model
model <- lm(mpg ~ hp, data = mtcars)
# your grandma on bongos
grandma <- "your grandma on bongos"
# let's creat list
happy_list <- list(a, ab, model, grandma)
happy_list
#> [[1]]
#> [1] 0.8904 0.0833 0.0699 0.3242 0.6959
#>
#> [[2]]
#>        a       b
#> 1 0.8904  0.0513
#> 2 0.0833 -0.0790
#> 3 0.0699  0.7334
#> 4 0.3242  1.4441
#> 5 0.6959  0.4122
#>
#> [[3]]
#>
#> Call:
#> lm(formula = mpg ~ hp, data = mtcars)
#>
#> Coefficients:
#> (Intercept)           hp
#>     30.0989      -0.0682
#>
#>
#> [[4]]
#> [1] "your grandma on bongos"
```

Võtame listist välja elemndi “ab”:

```
happy_list$ab
#> NULL
```

0.6.5.3 data frame ja tibble

Andmeraam on eriline list, mis koosneb ühepikkustest ja sama tüüpi vektoritest (listi iga element on vektor). Iga vektor on df-i veerg ja igas veerus on ainult ühte tüüpi andmed. Need vektorid ripuvad andmeraamis kõrvuti nagu tuulehaugid suitsuahjus, kusjuures vektori algus vastab tuulehaugi peale, mis on konksu otsas (konks vastab andmeraamis veeru nimele). Iga vektori nimi muutub sellises tabelis veeru nimeks.

R-s on 2 andmeraami tüüpi: data frame ja tibble, mis on väga sarnased. Tibble on uuem, veidi kaunima väljatrükiga ja pisut mugavam kasutada.

> Erinevalt data frame-st saab tibblesse lisada ka list tulpasid, mis võimaldab sisuliselt suvalisi R objekte tibblesse paigutada. Põhimõtteliselt piisab ainult ühest andmestruktuurist – tibble, et R-is töötada. Kõik, mis juhtub tibbles, jääb tibblesse.

“Tidyverse” töötab tibblega veidi paremini kui data frame-ga, aga see vahe ei ole suur.

Siin on meil 3 vektorit: shop, apples ja oranges, millest me paneme kokku tibble nimega fruits

```
## loome kolm vektorit
shop <- c("maxima", "tesco", "lidl")
apples <- c(1, 4, 43)
oranges <- c(2, 32, NA)
vabakava <- list(letters, runif(10), lm(mpg ~ cyl, mtcars))
## paneme need vektorid kokku tibble-sse
fruits <- tibble(shop, apples, oranges, vabakava)
fruits
#> # A tibble: 3 x 4
#>   shop   apples oranges vabakava
```

```
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima      1       2 <chr [26]>
#> 2 tesco       4      32 <dbl [10]>
#> 3 lidl       43      NA <lm>
```

Siin ta on, ilusti meie workspace-s. Pange tähele viimast tulpa "vabakava", mis sisaldab *character* vectorit, numbrilist vektorit ja lineaarse mudeli objekti.

Listi juba nii lihtsalt data.frame-i ei pane:

```
dfs <- try(data.frame(shop, apples, oranges, vabakava))
#> Error in as.data.frame.default(x[[i]], optional = TRUE, stringsAsFactors
#>   cannot coerce class '"lm"' to a data.frame
dfs
#> [1] "Error in as.data.frame.default(x[[i]], optional = TRUE, stringsAsFa
#> attr(,"class")
#> [1] "try-error"
#> attr(,"condition")
#> <simpleError in as.data.frame.default(x[[i]], optional = TRUE, stringsAs
```

0.6.5.4 Matrix

Maatriks on 2-dimensionaalne vektor, sisaldab ainult ühte tüüpi andmeid – numbrid, stringid, faktorid.

Tip: me saame sageli andmeraami otse maatriksina kasutada kui me viskame sealt välja mitte-numbrilised tulbad. Aga saame ka andmeraame konverteerida maatriksiks, ja tagasi.

```
fruits <- as.matrix(fruits)
class(fruits)
```

0.6.6 Indekseerimine

Igale vektori, listi, andmeraami ja maatriksi elemendile vastab unikaalne postiindeks, mille abil saame just selle elemendi unikaalselt indentifitseerida, välja võtta ja töödelda. Seega on in-

deksi mõte väga lühikese käsuga välja võtta R-i objektide üksikuid elemente. R-s algab indeksi numeratsioon 1-st (mitte 0-st, nagu näiteks Pythonis).

0.6.6.1 Vektorid ja nende indeksid on ühedimensionaalsed

```
my_vector <- 2:5
my_vector
#> [1] 2 3 4 5
my_vector[1] #1. element ehk number 2
#> [1] 2
my_vector[c(1,3)] #1. ja 3. element
#> [1] 2 4
my_vector[-1] #kõik elemendid, v.a. 1. element
#> [1] 3 4 5
my_vector[c(-1, -3)] #kõik elemendid, v.a. 1. ja 3. element
#> [1] 3 5
my_vector[3:5] #elemendid 3, 4 ja 5 (element 5 on määramata, seega NA)
#> [1]  4  5 NA
my_vector[-(3:length(my_vector))] #1. ja 2. element
#> [1] 2 3
```

0.6.6.2 Andmeraamid ja maatriksid on kahedimensionaalsed, nagu ka nende indeksid

2D indeksi kuju on [rea_indeks, veeru_indeks].

```
dat <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
dat
# üks andmepunkt: 1. rida, 2. veerg
dat[1, 2]

# 1. rida, kõik veerud
dat[1, ]

# 2. veerg, kõik read
dat[, 2]
```

```
# kõik read peale 1.
dat[-1, ]

# viskab välja 2. veeru
dat[, -2]

# 2 andmepunkti: 2. rida, 1. ja 2. veerg
dat[2, 1:2]

# 2 andmepunkti: 2. rida, 3. ja 4. veerg
dat[2, c(1, 2)]

#viskab välja 1. ja 2. rea
dat[-c(1, 2), ]

#veerg nimega colB, output on erandina vektor!
dat$colB
```

Kui me indekseerimisega tibblest veeru ehk vektori välja võtame, on output class: tibble. Kui me teeme sama data frame-st, siis on output class: vector.

Nüüd veidi keerulisemad konstruktsioonid, mis võimaldavad tabeli ühe kindla veeru väärtusi välja tõmmata teise veeru väärtuste järgi filteerides. Püüdke sellest koodist aru saada, et te hiljem ära tunneksite, kui midagi sellist vastu tuleb. Õnneks ei ole teil endil vaja sellist koodi kirjutada, me õpetame teile varsti lihtsama filtri meetodi.

```
dat <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
dat$colB[dat$colA != "a" ]
#> [1] 2 3
#jätab sisse kõik vektori colB väärtused,
#kus samas tabeli reas olev colA väärtus ei
#ole "a". output on vektor!
dat$colA[dat$colB > 1]
#> [1] "b" "c"
```

```
#jätab sisse kõik vektori colA väärtused,
#kus samas tabeli reas olev colB väärtus >1.
#output on vektor.
```

0.6.6.3 Listide indekseerimine

Listi indekseerimisel kasutame kahte sorti nurksulge, "[ ]" ja "[[ ]]", mis töötavad erinevalt.

Kui listi vaadata nagu objektide vanglat, siis kaksiksulgude `[[ ]]` abil on võimalik üksikuid objekte vanglast välja päästa nii, et taastub nende algne kuju ehk class. Seevastu üksiksulud `[ ]` tekitavad uue listi, kus on säilinud osad algse listi elemendid, ehk uue vangla vähemate vangidega.

> Kaksiksulud "[[ ]]" päästavad listist välja ühe elemendi ja taastavad selle algse class-i (data.frame, vektor, list jms). Üksiksulud "[ ]" võtavad algsest listist välja teie poolt valitud elemendid aga jätavad uue objekti ikka listi kujule.

```
my_list <- list(a = tibble(colA = c("A", "B"), colB = c(1, 2)), b = c(1, NA,
## this list has two elements, a data frame called "a" and a character vecto
str(my_list)
#> List of 2
#>  $ a:Classes 'tbl_df', 'tbl' and 'data.frame':   2 obs. of  2 variables:
#>   ..$ colA: chr [1:2] "A" "B"
#>   ..$ colB: num [1:2] 1 2
#>  $ b: chr [1:3] "1" NA "s"
```

Tõmbame listist välja tibble:

```
my_tibble <- my_list[[1]]
my_tibble
#> # A tibble: 2 x 2
#>   colA   colB
#>   <chr> <dbl>
#> 1 A         1
#> 2 B         2
```

See ei ole enam list.

Nüüd võtame üksiksuluga listist välja 1. elemendi, mis on tibble, aga output ei ole mitte tibble, vaid ikka list. Seekord ühe elemendiga, mis on tibble.

```
aa <- my_list[1]
str(aa)
#> List of 1
#>  $ a:Classes 'tbl_df', 'tbl' and 'data.frame':   2 obs. of  2 variables:
#>   ..$ colA: chr [1:2] "A" "B"
#>   ..$ colB: num [1:2] 1 2
```

```
aa1 <- my_list$a[2,] #class is df
aa1
#> # A tibble: 1 x 2
#>   colA   colB
#>   <chr> <dbl>
#> 1 B         2
```

```
aa3 <- my_list[[1]][1,]
aa3
#> # A tibble: 1 x 2
#>   colA   colB
#>   <chr> <dbl>
#> 1 A         1
```

Kõigepealt läksime kaksiksulgudega listi taseme võrra sisse ja võtsime välja objekti my_list 1. elemendi, tema algses tibble formaadis, (indeksi 1. dimensioon). Seejärel korjame sealt välja 1. rea, tibble formaati muutmata ja seega üksiksulgudes (indeksi 2. ja 3. dimensioon).

Pane tähele, et `[[ ]]` lubab ainult ühe elemendi korraga listist välja päästa.

0.7 Lihtne töö Andmeraamidega

0.7.1 Võrdleme andmeraame kahel viisil ja summeerime andmeraami.

1. all_equal

df1 on märklaud ja df2 on see, mida võrreldakse. convert = TRUE ühtlustab kahe tabeli vahel sarnased andmetüübid (n. factor ja character).

```
all_equal(df1, df2, convert = FALSE)
```

2. diffdf raamatukogu annab detailsema väljundi

```
diffdf::diffdf(df1, df2)
```

Andmetabeli Summary saab mitmel viisil, skimr::skim() funktsioon on üks paremaid

```
skimr::skim(iris)
#> Skim summary statistics
#>  n obs: 150
#>  n variables: 5
#>
#> -- Variable type:factor -------------------------------------------------
#>  variable missing complete   n n_unique
#>   Species       0      150 150        3
#>                       top_counts ordered
#>  set: 50, ver: 50, vir: 50, NA: 0   FALSE
#>
#> -- Variable type:numeric ------------------------------------------------
#>      variable missing complete   n mean   sd  p0 p25
#>  Petal.Length       0      150 150 3.76 1.77 1   1.6
#>   Petal.Width       0      150 150 1.2  0.76 0.1 0.3
#>  Sepal.Length       0      150 150 5.84 0.83 4.3 5.1
#>   Sepal.Width       0      150 150 3.06 0.44 2   2.8
```

```
#>   p50 p75 p100
#>  4.35 5.1  6.9
#>  1.3  1.8  2.5
#>  5.8  6.4  7.9
#>  3    3.3  4.4
```

BaasR kasutab `summary(df)` vormi.

0.7.2 Põhitehted andmeraamidega

```
count(iris, Species) #loeb üles, mitu korda igat näitu veerus Species esine
#> # A tibble: 3 x 2
#>   Species        n
#>   <fct>      <int>
#> 1 setosa        50
#> 2 versicolor    50
#> 3 virginica     50
summary(iris)
#>   Sepal.Length   Sepal.Width    Petal.Length
#>  Min.   :4.30   Min.   :2.00   Min.   :1.00
#>  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60
#>  Median :5.80   Median :3.00   Median :4.35
#>  Mean   :5.84   Mean   :3.06   Mean   :3.76
#>  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10
#>  Max.   :7.90   Max.   :4.40   Max.   :6.90
#>   Petal.Width        Species
#>  Min.   :0.1   setosa    :50
#>  1st Qu.:0.3   versicolor:50
#>  Median :1.3   virginica :50
#>  Mean   :1.2
#>  3rd Qu.:1.8
#>  Max.   :2.5
names(iris) #annab veerunimed
#> [1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
#> [4] "Petal.Width"  "Species"
nrow(iris) #mitu rida?
```

```
#> [1] 150
ncol(iris) #mitu veergu?
#> [1] 5
arrange(iris, desc(Sepal.Length)) %>% head(3)
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1          7.9         3.8          6.4         2.0
#> 2          7.7         3.8          6.7         2.2
#> 3          7.7         2.6          6.9         2.3
#>     Species
#> 1 virginica
#> 2 virginica
#> 3 virginica
#sorteerib tabeli veeru "Sepal.Length" väärtuste järgi
#langevalt (default on tõusev sorteerimine).
#Võib argumendina anda mitu veergu.
top_n(iris, 2, Sepal.Length)
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1          7.7         3.8          6.7         2.2
#> 2          7.7         2.6          6.9         2.3
#> 3          7.7         2.8          6.7         2.0
#> 4          7.9         3.8          6.4         2.0
#> 5          7.7         3.0          6.1         2.3
#>     Species
#> 1 virginica
#> 2 virginica
#> 3 virginica
#> 4 virginica
#> 5 virginica
#saab 2 või rohkem rida, milles on kõige suuremad S.L. väärtused
top_n(iris, -2, Sepal.Length) #saab 2 rida, milles on kõige väiksemad väärt
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1          4.4         2.9          1.4         0.2
#> 2          4.3         3.0          1.1         0.1
#> 3          4.4         3.0          1.3         0.2
#> 4          4.4         3.2          1.3         0.2
#>   Species
#> 1  setosa
```

```
#> 2  setosa
#> 3  setosa
#> 4  setosa
```

Tibblega saab teha maatriksarvutusi, kui kasutada ainult arvudega ridu. `apply()` arvutab maatriksi rea (1) või veeru (2) kaupa, vastavalt funktsioonile, mille sa ette annad.

```
colSums(fruits[ , 2:3])
#>  apples oranges
#>      48      NA
rowSums(fruits[ , 2:3])
#> [1]  3 36 NA
rowMeans(fruits[ , 2:3])
#> [1]  1.5 18.0   NA
colMeans(fruits[ , 2:3])
#>  apples oranges
#>      16      NA
fruits_subset <- fruits[ , 2:3]

# 1 tähendab, et arvuta sd rea kaupa
apply(fruits_subset, 1, sd)
#> [1]  0.707 19.799     NA
# 2 tähendab, et arvuta sd veeru kaupa
apply(fruits_subset, 2, sd)
#>  apples oranges
#>    23.4      NA
```

Lisame käsitsi tabelile rea:

```
fruits <- add_row(fruits,
                  shop = "konsum",
                  apples = 132,
                  oranges = -5,
                  .before = 3)
fruits
#> # A tibble: 4 x 4
#>   shop   apples oranges vabakava
```

```
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima      1       2 <chr [26]>
#> 2 tesco       4      32 <dbl [10]>
#> 3 konsum    132      -5 <NULL>
#> 4 lidl       43      NA <lm>
```

Proovi ise:

```
add_column()
```

Eelnevaid verbe ei kasuta me just sageli, sest tavaliselt loeme andmed sisse väljaspoolt R-i. Aga väga kasulikud on järgmised käsud:

0.7.3 Rekodeerime andmeraami väärtusi

```
fruits$apples[fruits$apples==43] <- 333
fruits
#> # A tibble: 4 x 4
#>   shop   apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima      1       2 <chr [26]>
#> 2 tesco       4      32 <dbl [10]>
#> 3 konsum    132      -5 <NULL>
#> 4 lidl      333      NA <lm>
fruits$shop[fruits$shop=="tesco"] <- "TESCO"
fruits
#> # A tibble: 4 x 4
#>   shop   apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima      1       2 <chr [26]>
#> 2 TESCO       4      32 <dbl [10]>
#> 3 konsum    132      -5 <NULL>
#> 4 lidl      333      NA <lm>
fruits$apples[fruits$apples>100] <- NA
fruits
#> # A tibble: 4 x 4
```

```
#>   shop   apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima      1       2 <chr [26]>
#> 2 TESCO       4      32 <dbl [10]>
#> 3 konsum     NA      -5 <NULL>
#> 4 lidl       NA      NA <lm>
```

Viskame välja duplikaatread, aga ainult need kus veerg nimega col1 sisaldab identseid väärtusi (mitmest identse väärtusega reast jääb alles ainult esimene)

```
distinct(dat, col1, .keep_all = TRUE)
# kõikide col vastu
distinct(dat)
```

Rekodeerime `Inf` ja `NA` väärtused nulliks (mis küll tavaliselt on halb mõte):

```
# inf to 0
x[is.infinite(x)] <- 0
# NA to 0
x[is.na(x)] <- 0
```

0.7.4 Ühendame kaks andmeraami rea kaupa

Tabeli veergude arv ei muutu, ridade arv kasvab.

```
dfs <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
dfs1 <- tibble(colA = "d", colB =  4)
#id teeb veel ühe veeru, mis näitab, kummast algtabelist iga uue tabeli rid
bind_rows(dfs, dfs1, .id = "id")
#> # A tibble: 4 x 3
#>   id    colA   colB
#>   <chr> <chr> <dbl>
#> 1 1     a         1
#> 2 1     b         2
#> 3 1     c         3
#> 4 2     d         4
```

Vaata Environmentist need tabelid üle ja mõtle järgi, mis juhtus.

Kui `bind_rows()` miskipärast ei tööta, proovi `do.call(rbind, dfs)`, mis on väga sarnane.

NB! Alati kontrollige, et ühendatud tabel oleks selline, nagu te tahtsite!

Näiteks, võib-olla te tahtsite järgnevat tabelit saada, aga võib-olla ka mitte:

```
df2 <- tibble(ColC = "d", ColD = 4)
## works by guessing your true intention
bind_rows(dfs1, df2)
#> # A tibble: 2 x 4
#>   colA   colB ColC   ColD
#>   <chr> <dbl> <chr> <dbl>
#> 1 d         4 <NA>     NA
#> 2 <NA>     NA d         4
```

0.7.5 ühendame kaks andmeraami veeru kaupa

Meil on 2 verbi: bind_cols ja cbind, millest esimene on konservatiivsem. Proovige eelkõige bind_col-ga läbi saada, aga kui muidu ei saa, siis cbind ühendab vahest asju, mida bind_cols keeldub puutumast. NB! Alati kontrollige, et ühendatud tabel oleks selline, nagu te tahtsite!

```
dfx <- tibble(colC = c(4, 5, 6))
bind_cols(dfs, dfx)
#> # A tibble: 3 x 3
#>   colA   colB  colC
#>   <chr> <dbl> <dbl>
#> 1 a         1     4
#> 2 b         2     5
#> 3 c         3     6
```

0.7.6 andmeraamide ühendamine join()-ga

Kõigepealt 2 tabelit: df1 ja df2.

```
df1 <- tribble(
  ~ Member,          ~ yr_of_birth,
  "John Lennon",     1940,
  "Paul McCartney", 1942
)

df1
#> # A tibble: 2 x 2
#>   Member         yr_of_birth
#>   <chr>                <dbl>
#> 1 John Lennon           1940
#> 2 Paul McCartney        1942
```

```
df2 <- tribble(
  ~ Member,            ~ instrument,    ~ yr_of_birth,
  "John Lennon",      "guitar",         1940,
  "Ringo Starr",      "drums",          1940,
  "George Harrisson", "guitar",         1942
)
df2
#> # A tibble: 3 x 3
#>   Member           instrument yr_of_birth
#>   <chr>            <chr>            <dbl>
#> 1 John Lennon      guitar            1940
#> 2 Ringo Starr      drums             1940
#> 3 George Harrisson guitar            1942
```

Ühendan 2 tabelit nii, et mõlema tabeli kõik read ilmuvad uude tabelisse.

```
full_join(df1, df2)
#> # A tibble: 4 x 3
#>   Member           yr_of_birth instrument
#>   <chr>                  <dbl> <chr>
#> 1 John Lennon             1940 guitar
```

```
#> 2 Paul McCartney          1942 <NA>
#> 3 Ringo Starr             1940 drums
#> 4 George Harrisson        1942 guitar
```

Ühendan esimese tabeliga df2 nii, et ainult df1 read säilivad, aga df2-lt võetakse sisse veerud, mis df1-s puuduvad. See on hea join, kui on vaja algtabelile lisada infot teistest tabelitest.

```
left_join(df1, df2)
#> # A tibble: 2 x 3
#>   Member         yr_of_birth instrument
#>   <chr>                <dbl> <chr>
#> 1 John Lennon           1940 guitar
#> 2 Paul McCartney        1942 <NA>
```

Jätan alles ainult need df1 read, millele vastab mõni df2 rida.

```
semi_join(df1, df2)
#> # A tibble: 1 x 2
#>   Member      yr_of_birth
#>   <chr>             <dbl>
#> 1 John Lennon        1940
```

Jätan alles ainult need df1 read, millele ei vasta ükski df2 rida.

```
anti_join(df1, df2)
#> # A tibble: 1 x 2
#>   Member         yr_of_birth
#>   <chr>                <dbl>
#> 1 Paul McCartney        1942
```

0.7.7 Nii saab raamist kätte vektori, millega tehteid teha.

Tibble jääb muidugi endisel kujul alles.

```
ubinad <- fruits$apples
ubinad <- ubinad + 2
ubinad
#> [1]  3  6 NA NA
```

```
## see on jälle vektor
str(ubinad)
#>  num [1:4] 3 6 NA NA
```

0.7.8 Andmeraamide salvestamine (eksport-import)

Andmeraami saame salvestada näiteks csv-na (comma separated file) oma kõvakettale, kasutame "tidyverse" analooge paketist "readr", mille nimed on baas R funktsioonidest eristatavad alakriipsu "_" kasutamisega. "readr" laaditakse automaatselt koos "tidyverse" laadimisega.

```
## loome uuesti fruits data tibble
shop <- c("maxima", "tesco", "lidl")
apples <- c(1, 4, 43)
oranges <- c(2, 32, NA)
fruits <- tibble(shop, apples, oranges, vabakava)
## kirjutame fruits tabeli csv faili fruits.csv kataloogi data
write_csv(fruits, "data/fruits.csv")
```

Kuhu see fail läks? See läks meie projekti juurkataloogi kausta "data/", juurkataloogi asukoha oma arvuti kõvakettal leiame käsuga:

```
getwd()
#> [1] "/Users/ulomaivali/Dropbox/loengud/R course/2017- R course/loengu-rm
```

Andmete sisselugemine töökataloogist:

```
fruits <-  read_csv("data/fruits.csv")
```

Andmeraamide sisselugemiseks on kaks paralleelset süsteemi: baas-R-i read.table() ja selle mugavusfunktsioonid (read.csv(), read.csv2() jne) ning readr paketti, mis laaditakse koos tidyversiga, funktsioon read_delim() ja selle mugavusfunktsioonid (read_csv jne). Tavaliselt soovitame eelistada alakriipsuga variante (http://r4ds.had.co.nz/data-import.html).

read_delim()-l ja selle poegade argument `col_types`

`= cols(col_name_1 = col_double(), col_name_2 = col_date(format = ""))` võimaldab spetsifitseerida kindlatele veergudele, mis tüübiga need sisse loetakse. Töötab ka `cols_only(a = col_integer())`, samuti standardsed lühendid andmetüüpidele: `cols(a = "i", b = "d")`. Vaikimisi otsustab programm andmetüübi iga veeru esimese 1000 elemendi põhjal. Vahest tasub kõik veerud sisse lugeda character-idena, et oleks parem probleeme tuvastada: `df1 <- read_csv("my_data_frame_name.csv"), col_types = cols(.default = col_character()))`. .default - kõik nimega veerud, mille kohta ei ole eksplitsiitselt teisiti öeldud, lähevad sisse lugemisel selle alla. .

Seda, milline sümbol kodeerib sisseloetavas failis koma ja milline on "grouping mark", mis eraldab tuhandeid, saab sisestada `locale = locale(decimal_mark = ",", grouping_mark = ".")` abil. Või näit: `locale("et", decimal_mark = ";")`. Vt ka https://cran.r-project.org/web/packages/readr/vignettes/locales.html.

https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes annab nimekirja riikide lokaalitähistest.

Argument `skip = n` jätab esimesed n rida sisse lugemata. Argument `comment = "#"` jätab sisse lugemata read, mis algavad #-ga.

Argument `col_names = FALSE` ei loe esimest rida sisse veerunimedena (see on vaikekäitumine) ja selle asemel nimetatakse veerud X1 ... Xn. `col_names = c("x", "y", "z"))` loeb tabeli sisse uute veerunimedega x, y ja z.

`na = "."` argument ütleb, et tabeli kirjed, mis on punktid, tuleb sisse lugeda NA-dena.

Kui teil on kataloogitäis faile (näit .csv lõpuga), mida soovite kõiki korraga sisse lugeda, siis tehke nii:

```
library(fs)
#järgnev loeb sisse iga faili eraldi kataloogist nimega data_dir
fs::dir_ls(data_dir, regexp = "\\.csv$") %>% map(read_csv)

#Kui meil on mitu faili samade tulbanimedega ja tahame
```

```
#need sisse lugeda ühte faili üksteise järel, siis
dir_ls(data_dir, regexp = "\\.csv$") %>% map_dfr(read_csv, .id = "source")
#.id on optsionaalne argument, mis lisab uude faili lisaveeru,
#kus on unikaalsed viited igale algtabelile, et oleks näha, millisest
#tabelist iga uue tabeli rida pärit on.
```

MS exceli failist saab tabeleid importida "readxl" raamatukogu abil.

```
library(readxl)
## kõigepealt vaatame kui palju sheete failis on
sheets <- excel_sheets("data/excelfile.xlsx")
## siis impordime näiteks esimese sheeti
dfs <- read_excel("data/excelfile.xlsx", sheet = sheets[1])
```

Excelist csv-na eksporditud failid tuleks sisse lugeda käsuga `read_csv2` või `read.csv2` (need on erinevad funktsioonid; read.csv2 loeb selle sisse data frame-na ja read_csv2 tibble-na).

R-i saab sisse lugeda palju erinevaid andmeformaate. Näiteks, installi RStudio addin: "Gotta read em all R", vaata eespool. See läheb ülesse tab-i Addins. Sealt saab selle avada ja selle abil tabeleid oma workspace üles laadida.

Alternatiiv: mine alla paremake Files tab-le, navigeeri sinna kuhu vaja ja kliki faili nimele, mida tahad R-i importida.

Mõlemal juhul ilmub alla konsooli (all vasakul) koodijupp, mille jooksutamine peaks asja ära tegema. Te võite tahta selle koodi kopeerida üles vasakusse aknasse kus teie ülejäänud kood tulevastele põlvedele säilub.

> Tüüpiliselt töötate R-s oma algse andmestikuga. Reprodutseeruvaks projektiks on vaja 2 asja: algandmeid ja koodi, millega neid manipuleerida. R ei muuda algandmeid, mille te näiteks csv-na sisse loete.

Andmetabelite salvestamine töö vaheproduktidena ei ole sageli vajalik, sest te jooksutate iga kord, kui te oma projekti juurde naasete, kogu analüüsi uuesti kuni kohani, kuhu te pooleli jäite.

See tagab, et teie kood töötab tervikuna. Erandiks on tabelid, mille arvutamine võtab palju aega.

Tibble konverteerimine data frame-ks ja tagasi tibbleks:

```
class(fruits) #näitab ojekti klassi
#> [1] "tbl_df"     "tbl"        "data.frame"
fruits <- as.data.frame(fruits)
class(fruits)
#> [1] "data.frame"
fruits <- as_tibble(fruits)
class(fruits)
#> [1] "tbl_df"     "tbl"        "data.frame"
```

0.7.9 NA-d

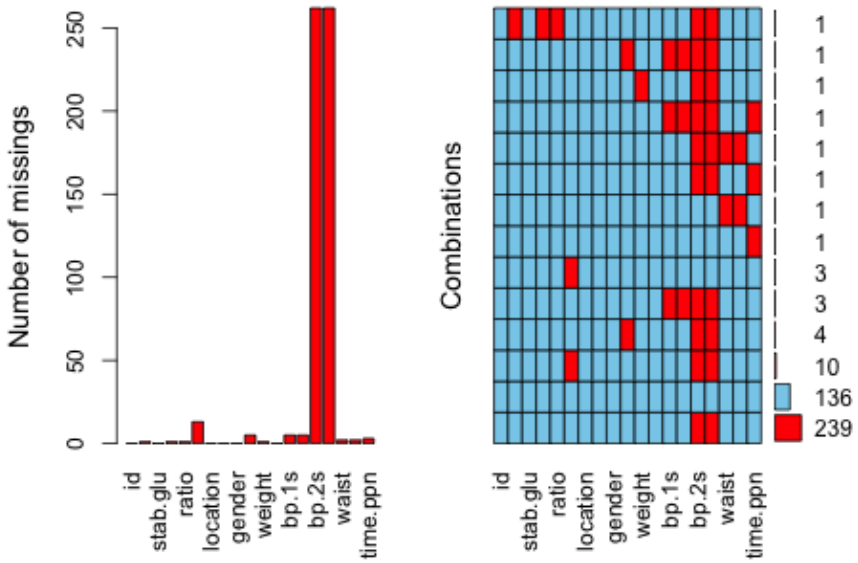
Loe üles NA-d, 0-d, inf-id ja unikaalsed väärtused.

```
library(funModeling)
diabetes <- read_delim("data/diabetes.csv",
    ";", escape_double = FALSE, trim_ws = TRUE)

diabetes %>% status() %>%
  mutate_if(is.numeric, round, 2) %>%
  kableExtra::kable()
```

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
id	0	0	0	0.00	0	0	numeric	403
chol	0	0	1	0.00	0	0	numeric	154
stab.glu	0	0	0	0.00	0	0	numeric	116
hdl	0	0	1	0.00	0	0	numeric	77
ratio	0	0	1	0.00	0	0	numeric	69
glyhb	0	0	13	0.03	0	0	numeric	239
location	0	0	0	0.00	0	0	character	2
age	0	0	0	0.00	0	0	numeric	68
gender	0	0	0	0.00	0	0	character	2
height	0	0	5	0.01	0	0	numeric	22
weight	0	0	1	0.00	0	0	numeric	140
frame	0	0	12	0.03	0	0	character	3
bp.1s	0	0	5	0.01	0	0	numeric	71
bp.1d	0	0	5	0.01	0	0	numeric	57
bp.2s	0	0	262	0.65	0	0	numeric	48
bp.2d	0	0	262	0.65	0	0	numeric	36
waist	0	0	2	0.00	0	0	numeric	30
hip	0	0	2	0.00	0	0	numeric	32
time.ppn	0	0	3	0.01	0	0	numeric	60

```
diabetes <- read.table(file = "data/diabetes.csv", sep = ";", dec = ",", hea
str(diabetes)
#> 'data.frame':    403 obs. of  19 variables:
#>  $ id      : int  1000 1001 1002 1003 1005 1008 1011 1015 1016 1022 ...
#>  $ chol    : int  203 165 228 78 249 248 195 227 177 263 ...
#>  $ stab.glu: int  82 97 92 93 90 94 92 75 87 89 ...
#>  $ hdl     : int  56 24 37 12 28 69 41 44 49 40 ...
#>  $ ratio   : num  3.6 6.9 6.2 6.5 8.9 ...
#>  $ glyhb   : num  4.31 4.44 4.64 4.63 7.72 ...
#>  $ location: Factor w/ 2 levels "Buckingham","Louisa": 1 1 1 1 1 1 1 1 1
#>  $ age     : int  46 29 58 67 64 34 30 37 45 55 ...
#>  $ gender  : Factor w/ 2 levels "female","male": 1 1 1 2 2 2 2 2 2 1 ...
#>  $ height  : int  62 64 61 67 68 71 69 59 69 63 ...
#>  $ weight  : int  121 218 256 119 183 190 191 170 166 202 ...
#>  $ frame   : Factor w/ 4 levels "","large","medium",..: 3 2 2 2 3 2 3 3
#>  $ bp.1s   : int  118 112 190 110 138 132 161 NA 160 108 ...
```

```
#>  $ bp.1d   : int  59 68 92 50 80 86 112 NA 80 72 ...
#>  $ bp.2s   : int  NA NA 185 NA NA NA 161 NA 128 NA ...
#>  $ bp.2d   : int  NA NA 92 NA NA NA 112 NA 86 NA ...
#>  $ waist   : int  29 46 49 33 44 36 46 34 34 45 ...
#>  $ hip     : int  38 48 57 38 41 42 49 39 40 50 ...
#>  $ time.ppn: int  720 360 180 480 300 195 720 1020 300 240 ...
VIM::aggr(diabetes, prop = FALSE, numbers = TRUE)
```



Siit on näha, et kui me viskame välja 2 tulpa ja seejärel kõik read, mis sisaldavad NA-sid, kaotame me umbes 20 rida 380-st, mis ei ole suur kaotus.

Kui palju ridu, milles on 0 NA-d? Mitu % kõikidest ridadest?

```
nrows <- nrow(diabetes)
  ncomplete <- sum(complete.cases(diabetes))
  ncomplete #136
#> [1] 136
  ncomplete/nrows #34%
#> [1] 0.337
```

0.7.9.1 Mitu NA-d on igas tulbas?

```
diabetes %>% map_df(~sum(is.na(.))) %>% t()
#>           [,1]
#> id           0
#> chol         1
#> stab.glu     0
#> hdl          1
#> ratio        1
#> glyhb       13
#> location     0
#> age          0
#> gender       0
#> height       5
#> weight       1
#> frame        0
#> bp.1s        5
#> bp.1d        5
#> bp.2s      262
#> bp.2d      262
#> waist        2
#> hip          2
#> time.ppn     3
```

väljund on uus tabel NA-de arvuga igale algse tabeli veerule

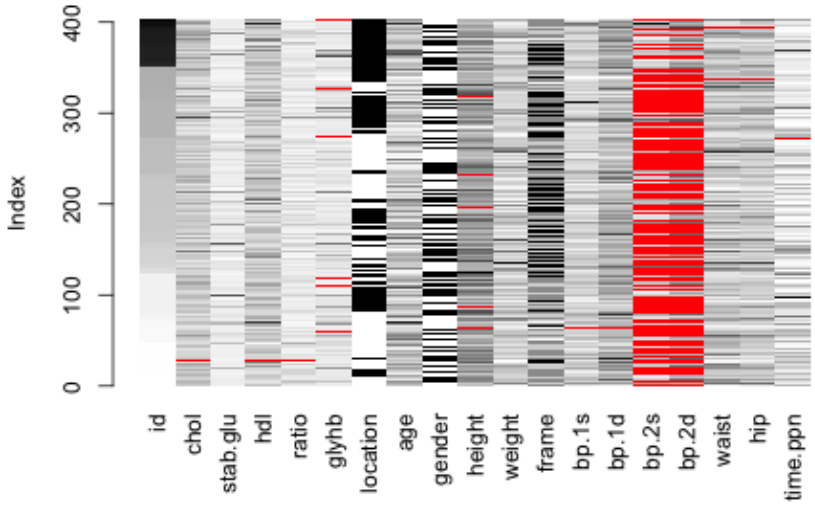
Eelnev ekspressioon töötab nii: map_df() loeb kokku (summeerib) diabetes tabeli igale veerule, mitu elementi selles veerus andis ekspressioonile is.na() vastuseks TRUE. is.na() on funktsioon, mis annab väljundiks TRUE v FALSE, sõltuvalt vektori elemendi NA-staatusest.

```
is.na(c(NA, "3", "sd", "NA"))
#> [1]  TRUE FALSE FALSE FALSE
```

Pane tähele, et string “NA” ei ole sama asi, mis loogiline konstant NA.

Ploti NAd punasega igale tabeli reale ja tulbale mida tumedam halli toon seda suurem number selle tulba kontekstis:

```
VIM::matrixplot(diabetes)
#>
#> Click in a column to sort by the corresponding variable.
#> To regain use of the VIM GUI and the R console, click outside the plot r
```



0.7.9.2 Kuidas rekodeerida NA-d näiteks 0-ks:

```
dfs[is.na(dfs)] <- 0
dfs[is.na(dfs)] <- "other"
dfs[dfs == 0] <- NA # teeb vastupidi 0-d NA-deks
```

Pane tähele, et NA tähistamine ei käi character vectorina vaid dedikeeritud `is.na()` funktsiooniga.

coalesce teeb seda peenemalt. kõigepealt kõik

```
x <- c(1:5, NA, NA, NA)
```

```
coalesce(x, 0L)
#> [1] 1 2 3 4 5 0 0 0
```

Nii saab 2 vektori põhjal kolmanda nii, et NA-d asendatakse vastava väärtusega:

```
y <- c(1, 2, NA, NA, 5)
z <- c(NA, NA, 3, 4, 5)
coalesce(y, z)
#> [1] 1 2 3 4 5
```

`filter_all(weather, any_vars(is.na(.)))` näitab ridu, mis sisaldavad NA-sid

`filter_at(weather, vars(starts_with("wind")), all_vars(is.na(.)))` read, kus veerg, mis sisaldab wind, on NA.

0.7.9.3 Rekodeerime NA-ks

```
na_if(x, y)
#x - vektor ehk tabeli veerg, mida modifitseerime
#y - väärtus, mida soovime NA-ga asendada

na_if(dfs, "") #teeb dfs tabelis tühjad lahtrid NA-deks
na_if(dfs, "other") #teeb lahtrid, kus on "other" NA-deks
na_if(dfs, 0) #teeb 0-d NA-deks.
```

0.7.9.4 drop_na() viskab tabelist välja NA-dega read

`drop_na(data, c(column1, column2))` - argument variable võimaldab visata välja read, mis on NA-d ainult kindlates veergudes (column1 ja column2 meie näites - aga ära unusta column1 asemele kirjutada oma veeru nime). Ära unusta ka kasutamast vektori vormi c(), et veerge määrata. column1:column4 vorm töötab samuti ja võtab NAd veergudest 1-4 (kui veeru nr 1 nimi on column1 jne).

0.7.9.5 viska tabelist välja veerud, milles on liiga palju NA-sid

Meil on lai tabel sadade numbriliste veegudega. Neist paljud on NA-rikkad (andmevaesed) ja tuleks tabelist eemaldada. Aga kuidas seda teha?

Selleks (1) koostame vektori (vekt), milles on tabeli df-i iga numbrilise veeru NA-de suhtarv, (2) viskame sellest vektorist välja nende veergude nimed, milles on liiga palju NA-sid ja (3) subsetime df-i saadud vektori (vekt1) vastu.

NB! vekt ja vekt1 on nn named vektorid, milles vektori iga element (NAde suhtarv mingis tabeli veerus) on seotud selle elemendi nimega, mis on identne selle veeru nimega tabelis df.

```
vekt <- sapply(df, function(x) mean(is.na(x)))
#NA-de protsent igas veerus
vekt
vekt1 <- vekt[vekt < 0.8]
#subsettisin vektori elemendid, mis on < 0.8 (NA-sid alla 80%). 216 tk.
#vekt1 is a named vector
vekt1n <- names(vekt1) #vektor named vektori vekt1 nimedest
df_with_fewer_cols <- subset(df, select = vekt1n)
#subsetime (jätame alles) ainult need df-i veerud,
#mille nimele vastab mõni vektori nad1n element
```

0.8 map() - kordame sama operatsiooni igale listi liikmele

Järgnevad meetodid töötavad nii listidel, data frame-del kui vektoritel. Seda sellepärast, et formaalselt on list vektori tüüp (rekursiivne vektor), kuhu on võimalik elementidena panna mida iganes, k.a. teisi vektoreid. Enamus R-i funktsioone, mis töötavad lihtsate mitte-rekursiivsete vektoritega (ja df-dega), ei tööta listide peal.

purrr::map() perekonna funktsioonid töötavad nii lihtsate vektorite

kui listide peal. Need funktsioonid rakendavad kasutaja poolt ette antud funktsiooni järjest igale vektori elemendile. map() vajab 2 argumenti: vektor ja funktsioon, mida selle vektori elementidele rakendada. map() võtab sisse listi (vektori, data frame) ja väljastab listi (vektori, data frame). Seega saab seda hästi pipe-s rakendada.

Kui tahad, map() anda funktsiooni lisaargumentidega, siis need eraldatakse komadega `map(list1, round, digits = 2)`.

Kui sa ei taha väljundina listi, vaid lihtsat numbrilist vektorit, siis kasuta `map_dbl()`.

```
1:10 %>%
  map(rnorm, n = 10) %>%
  map_dbl(mean)
#>  [1]  1.05  2.10  3.11  3.83  4.93  5.62  7.09  7.81
#>  [9]  8.83 10.27
```

map()-l on kokku 8 versiooni erinevate väljunditega.

- map() - list
- map_dbl() - floating point number vektor
- map_chr() - character vektor
- map_dfc() - data frame column binded
- map_dfr() - data frame row binded (lisab iga elemendi df-i reana)
- map_int() - integer vektor
- map_lgl() - logical vektor
- walk() - nähtamatu väljund (kasutatakse funktsioonide puhul, mis ei anna *command line*-le väljundit, nagu plot() või failide salvestamine).

sisestame map-i funktsiooni asemel ekspressiooni `max(df1$col1) - min(df1$col1)`:

ekspressiooni juhatab sisse ~ (tilde) ja seal asendatakse see, millega opereeritakse, .x -ga: `~ max(.x) - min(.x)`.

Kasutades funktsiooni `pluck()` nopime järgnevas koodis igast "params" listi alam-listist mu ja sd ning kasutame neid, et genereerida 3 portsu juhuslikke arve, millest igas on 5 juhuslikku arvu, mis on genereeritud vastavalt selles alam-listis spetsifitseeritud mu-le ja sigmale.

```
params <- list(
  "norm1" = list("mu" = 0, "sd" = 1),
  "norm2" = list("mu" = 1, "sd" = 1),
  "norm3" = list("mu" = 2, "scale" = 1)
)
params %>% map(~rnorm(5, mean = pluck(.x, 1), sd = pluck(.x, 2)))
#> $norm1
#> [1] -0.501 -1.293 -0.750  1.314  0.827
#>
#> $norm2
#> [1]  1.9188  3.0938  1.4370  0.0903 -0.6402
#>
#> $norm3
#> [1] 2.676 1.785 1.819 3.002 0.544
```

`enframe()` konverteerib nimedega vektori df-ks, millel on 2 veergu (name, value).

0.8.0.1 map2()

Itereerib üle kahe vektori - map2(.x, .y, .f).

.x ja .y on sama pikad vektorid (ühe-elemendine vektor kõlbab ka - seda lihtsalt retsükleeritakse nii mitu korda, kui teises vektoris on liikmeid).

.f on funktsioon või ekspressioon (valem)

Ekspressioon `map2()`-le algab ikka tildega; esimese vektori elemendid on .x teise vektori elemendid on .y

Näiteks `map2(x, y, ~ .x + .y)` liidab vektorid x ja y

0.8.0.2 pmap()

itereerib üle 3+ vektori. Näiteks `pmap(list(x, y, z), sum)` liidab 3 vektorit (x, y ja z on ise vektorid)

Järgnevas koodis anname ette listi long_numbers kolme vektoriga (pi, exp(1) ja sqrt(2)) ning vektori digits kolme liikmega, mida kasutab funktsiooni round() argument digits. Andes sellele argumendile 3 erinevat väärtust saame me kolm erinevat ümardamist kolmele listi long_numbers liikmele.

```
long_numbers <- list(pi, exp(1), sqrt(2))
digits <- list(2, 3, 4)
pmap(list(x = long_numbers, digits = digits), round)
#> [[1]]
#> [1] 3.14
#>
#> [[2]]
#> [1] 2.72
#>
#> [[3]]
#> [1] 1.41
```

pmap() ekspressioonide sisesed elemndid on ..1, ..2, ..3 jne, mitte .x ja .y nagu map2-l.

NB! pmap-i saab sisetada data frame, mille peal see töötab rea kaupa.

```
parameters <- data.frame(
  n = c(1, 2, 3),
  min = c(0, 5, 10),
  max = c(1, 6, 11)
)
parameters %>% pmap(runif)
#> [[1]]
#> [1] 0.722
#>
#> [[2]]
```

```
#> [1] 5.46 5.00
#>
#> [[3]]
#> [1] 11.0 10.1 10.5
```

See töötab sest runif() võtab 3 argumenti ja df-l parameters on 3 veergu.

Järgmine funktsioon rakendub suvalisele df-le rea kaupa ja arvutab igale reale näit sd. Aga selleks transponeerime read veergudeks ja rakendame tavalist map()-i.

```
rmap <- function (.x, .f, ...) {
    if(is.null(dim(.x))) stop("dim(X) must have a positive length")
    .x <- t(.x) %>% as.data.frame(.,stringsAsFactors=F)
    purrr::map_dfr(.x=.x,.f=.f,...)
}
parameters %>% rmap(sd)
#> # A tibble: 1 x 3
#>       V1    V2    V3
#>    <dbl> <dbl> <dbl>
#> 1 0.577  2.08  4.36
```

apply teeb sama lihtsamini.

```
apply(parameters, 1, sd)
#> [1] 0.577 2.082 4.359
```

0.8.0.3 invoke_map()

itereerib üle funktsioonide vektori, millele järgneb argumentide vektor. 1. funktsioon esimese argumendiga jne.

```
functions <- list(rnorm, rlnorm, rcauchy)
n <- list(c(5, 2, 3), 2, 3)
invoke_map(functions, n)
#> [[1]]
#> [1]  5.59  1.28 -2.64 -2.08  1.34
#>
```

```
#> [[2]]
#> [1] 0.588 4.051
#>
#> [[3]]
#> [1] -0.0491 -0.6352  3.5326
```

anname sisse esimese argumendi (100) igasse funktsiooni

```
functions <- list(rnorm, rlnorm, rcauchy)
n <- c(5, 2, 3)
invoke_map(functions, n, 100)
#> [[1]]
#> [1] 100.1 101.0 100.8  98.7 101.0
#>
#> [[2]]
#> [1] 7.71e+42 1.10e+43
#>
#> [[3]]
#> [1]  96.7 100.0 113.4
```

mitu argumenti igale funktsioonile:

```
args <- list(norm = c(3, mean = 0, sd = 1),
             lnorm = c(2, meanlog = 1, sdlog = 2),
             cauchy = c(1, location = 10, scale = 100))

invoke_map(functions, args)
#> [[1]]
#> [1] -1.011  0.403  0.577
#>
#> [[2]]
#> [1] 4.20 1.39
#>
#> [[3]]
#> [1] -1819
```

0.8.0.3.1 map shortcuts

`pluck()` võtab listist välja elemendi (vektori, data frame jms) nii nagu see on (mitte listina).

```
list1 <- list(
  numbers = 1:3,
  letters = c("a", "b", "c"),
  logicals = c(TRUE, FALSE)
)

pluck(list1, 1) # list1 %>% pluck(1)
#> [1] 1 2 3
pluck(list1, "numbers") # list1 %>% pluck("numbers")
#> [1] 1 2 3
```

Andes map()-le ette character stringi (elemendi nime), saame tagasi elemendi igast alam-listist, mille nimi vastab sellele stringile. See on shotcut pluck-ile.

```
params <- list(
  "norm1" = list("mu" = 0, "sd" = 1),
  "norm2" = list("mu" = 1, "sd" = 1),
  "norm3" = list("mu" = 2, "scale" = 1)
)
map_dbl(params, "mu")
#> norm1 norm2 norm3
#>     0     1     2
```

Sama teeb, kui map-le ette anda elemendi positsioon listis

```
map_dbl(params, 1)
#> norm1 norm2 norm3
#>     0     1     2
```

Nii saab kätte samad tulbad (vektorid) mitmest data frame-st (kui list sisaldab data frame-sid).

veel mõned abifunktsioonid:

lmap() works exclusively with functions that take lists imap() applies a function to each element of a vector, and its index map_at() and map_if() only map a function to specific elements of a list.

0.8.0.3.2 List column

Df-i veerg, mille andmetüüp on list. Näiteks mudeliobjektid, funktsioonid ja teised df-d võivad minna list columnisse! List columnid on ise listid, mitte andmevektorid.

```
List veerud võimaldavad panna samasse tabelisse erinevaid asju -
andmeid, mudeleid, mudeli koefitsiente jms.
```

nest() teeb uue df-i, kus on 1. veerg grupeeriva muutuja tasemetega, millele järgneb list column, mille iga element on tibble. Iga tibble sisaldab relevantset infot grupeeriva muutuja vastava taseme kohta.

```
library(gapminder)
(nested_gapminder <- gapminder %>% group_by(country) %>% nest())
#> # A tibble: 142 x 2
#> # Groups:   country [142]
#>   country               data
#>   <fct>       <list<df[,5]>>
#> 1 Afghanistan       [12 x 5]
#> 2 Albania           [12 x 5]
#> 3 Algeria           [12 x 5]
#> 4 Angola            [12 x 5]
#> 5 Argentina         [12 x 5]
#> 6 Australia         [12 x 5]
#> # ... with 136 more rows
```

unnest() teeb algse df-i tagasi.

iga nested_gapminder$data element on ise df:

```
nested_gapminder %>%
  pluck("data") %>%
  pluck(1) # %>% lm(lifeExp ~ year, data = .)
```

```
#> # A tibble: 12 x 5
#>   continent  year lifeExp      pop gdpPercap
#>   <fct>     <int>   <dbl>    <int>     <dbl>
#> 1 Asia       1952    28.8  8425333      779.
#> 2 Asia       1957    30.3  9240934      821.
#> 3 Asia       1962    32.0 10267083      853.
#> 4 Asia       1967    34.0 11537966      836.
#> 5 Asia       1972    36.1 13079460      740.
#> 6 Asia       1977    38.4 14880372      786.
#> # ... with 6 more rows
#fitime ühe mudeli 1. elemendile (1. riik)
```

fit a model to each tibble nested within nested_gapminder and then store those models as a list column

fitime mudeli igale listi veerule (igale riigile). väljund on ilge list.

```
model1 <- nested_gapminder %>%
  pluck("data") %>%
  map(~ lm(lifeExp ~ year, data = .x))
```

arvutame mudelid igale riigile ja pistame väljundi (mudeliobjekti) nested_gapminder uude list columnisse:

```
models1 <- nested_gapminder %>%
  mutate(models = map(data, ~ lm(lifeExp ~ year, data = .x)))
```

võtame välja mudeli koefitsiendi year ja paneme uude veergu nimega coefficient:

```
models1 <-  models1 %>% mutate( coefficient = map_dbl(models, ~coef(.x) %>%
```

df-i veerg models on ühtlasi list, millele saame map_dbl() rakendada.

järgnevad 3 koodirida teevad sama asja - võtavad välja 1. mudeli:

```
models1 %>%
  pluck("models") %>%
  pluck(1)
```

```
models1[[1, 3]]

models1$models[[1]]
```

0.9 Regular expression ja find & replace

Regular expression annab võimaluse lühidalt kirjeldada mitte-üheseid otsinguparameetreid.

regular expression on string, mis kirjeldab mitut stringi

A regular expression Regular Expressions as used in R

string on märkide järjestus, mis on jutumärkide vahel (“” või ”). Osad märgid ei ole R stringis otse representeeritavad. Neid representeerivad nn special characters ehk erimärgid. Iga kord kui te regular expressionis näete peate seda stringis, mis representeerib seda rexexp-i, kirjutama kui \.

`writeLines()` näitab kuidas R näeb su stringi peale seda, kui erimärgid on välja loetud (parsitud).

```
writeLines("\\.") # \.
writeLines("\\ is a backslash") # \ is a backslash
```

Enamus märke (k.a. tähed ja numbrid) tähistavad ainult iseennast.

- . tähistab igat märki.

Märgiklass on märkide nimekiri nurksulgude vahel, nagu näiteks `[[:alnum:]]`, mis on sama kui `[A-z0-9]`. Enamasti tuleb need stringi kirjutada topelt nurksulgudes: [[:siia_märgiklass:]]. Aga näiteks [0-9] on üksik nurksulgudes.

- tavalised märgiklassid:
- `[:alnum:]` numbrid ja tähed: AacF123
- `[:digit:]` numbrid: 123
- `[:alpha:]` tähed: asdf

- `[:upper:]` suured tähed: ASDF
- `[:lower:]` väiksed tähed: asdf
- `[:punct:]` ! " # $ % & ' ( ) * + , - / : ; < = > ? @ [ ] ^ _ ‘ ‘ { | } ~.
- `[:space:]` space, tab ja newline.
- `[:blank:]` tab ja newline.

Metamärgid on `. \ | ( ) [ { ^ $ * + ?`. Nende tähendus sõltub kontekstist. `\\` teeb metamärgist tavalise märgi.

```
trüki see   regex
\\n         \n   new line (return)
\\t         \t   tab
\\s         \s   any whitespace (\S - any non-whitespace)
\\d         \d   any digit (\D - any non-digit)
\\w         \w   any word character (\W - non-word char)
\\b         \b   word boundaries

Selleks, et trükkida erimärk tavalise märgina:

trüki  selleks
\\.    .
\\!    !
\\?    ?
\\\\   \
\\(    (
\\{    {
```

- Repetition quantifiers put after regex specify how many times regex is matched: `?`, zero or one; `*`, zero or more times; `+`, one or more times; `{n}`, n times; `{n,}`, n or more times; `{n,m}`, n to m times.
- ^ anchors the regular expression to the start of the string.
- $ anchors the the regular expression to end of the string.
- ab|d tähendab ab või d
- [abe] tähendab ühte kolmest (kas a või b või e)

- [^abe] tähendab kõike, mis ei ole a või b või e
- [a-c] tähendab a või b või c

Sulud annavad eelistuse

- (ab|d)e tähendab abe või de

Leia string, millele järgneb või eelneb mingi string

- a(?=c) annab need a-d, millele järgneb c
- a(?!c) annab need a-d, millele ei järgne c
- (?<=b)a annab need a-d, millele eelneb b
- (?<!b)a annab need a-d, millele ei eelne b

patterns that match more than one character:

```
. (dot): any character apart from a newline.

\\d: any digit.

\\s: any whitespace (space, tab, newline).

\[abc]: match a, b, or c.

\[!abc]: match anything except a, b, or c.

To create a regular expression containing \d or \s, you???ll need to escape t

abc|d..f will match either "abc", or "deaf".
```

Et mitte interpreteerida stringi tavalise regex-ina: `regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)` ignore cases, match end of lines and end of strings, allow R comments within regex's , and/or to have `.` match everything including `\n`. Näiteks `str_detect("I", regex("i", TRUE))`

0.9.1 Common operations with regular expressions

- Locate a pattern match (positions)
- str_detect() annab TRUE/FALSE
- str_which() annab stringide, mis sisaldavad otsingumustrit, indeksinumbrid
- str_count() annab esinemiste arvu stringis
- str_locate_all() annab otsingumustri positsiooninumbri (indeksi) stringis
- Extract a matched pattern
- str_sub() võtab välja otsitud alamstringi; otsing indeksinumbrite järgi
- str_subset() võtab välja terve stringi; regex otsing
- str_extract_all() võtab välja mustri (alamstringi); regex otsing
- str_match_all() annab maatriksi, millel on veerg igale grupile regeximustris
- Replace a matched pattern
- str_replace_all()
- str_to_lower()
- str_to_upper()
- str_to_title()
- stringi pikkus
- str_length() annab märkide arvu stringis
- str_trim() võtab maha whitespace stringi algusest/lõpust
- ühenda ja eralda stringe
- str_c() ühendab, k.a. kollapseerib mitu stringi üheks (arg collapse=)

- str_dup() kordab stringi n korda
- str_split_fixed() jagab stringi alamstringide maatriksiks
- glue::glue_data() teeb stringi df-st, listist v environmentist
- järjesta stringe
- str_sort() annab sorditud character vectori

0.9.2 Find and replace

```
library(stringr)
x<- c("apple", "ananas", "banana")

#replaces all a-s at the beginning of strings with e-s
str_replace(x, "^a", "e")
#> [1] "epple"  "enanas" "banana"

# str_replace only replaces at the first occurence at each string
str_replace(x, "a", "e")
#> [1] "epple"  "enanas" "benana"

#str_replace_all replaces all a-s anywhere in the strings
str_replace_all(x, "a", "e")
#> [1] "epple"  "enenes" "benene"

#replaces a and the following character at the end of string with nothing (
str_replace(x, "a.$", "")
#> [1] "apple"  "anan"   "banana"

#replaces a-s or s-s at the end of string with e-s
str_replace(x, "(a|s)$", "e")
#> [1] "apple"  "ananae" "banane"

#replaces a-s or s-s anywhere in the string with e-s
str_replace_all(x, "a|s", "e")
#> [1] "epple"  "enenee" "benene"
```

```
#remove all numbers.
y<-c("as1", "2we3w", "3e")
str_replace_all(y, "\\d", "")
#> [1] "as"  "wew" "e"

#remove everything, except numbers.
str_replace_all(y, "[A-Za-z_]", "")
#> [1] "1"  "23" "3"
```

```
x<- c("apple", "apple pie")
str_replace_all(x, "^apple$","m") #To force to only match a complete string
#> [1] "m"         "apple pie"
str_replace_all(x, "\\s","_") #space to _
#> [1] "apple"     "apple_pie"
str_replace_all(x, "[apl]","_") #a or p or l to _
#> [1] "____e"     "____e _ie"
str_replace_all(x, "[ap|p.e]","_") # ap or p.e to _
#> [1] "___l_"     "___l_ _i_"
```

näide: meil on vector v, milles täht tähistab katse tüüpi, number, mis on tähe ees, tähistab mõõtmisobjekti identiteeti ja tähe järel asuv number tähistab ajapunkti tundides (h). F ja f tähistavad sama asja. Kõigepealt võtame välja F-i mõõtmisojbekti ehk subjekti koodid

```
library(stringr)
v <- c("1F1", "12F2h", "13f1", "2S")



v_f <- str_subset(v, "[Ff]")
#filtreerime F ja f sisaldavad stringid
v_f
#> [1] "1F1"   "12F2h" "13f1"
v_f_subject <- str_replace_all(v_f, "[Ff][0-9]+h?", "")
#string "F või f, number üks või enam korda, h 0 või enam korda" asendada t
v_f_subject
#> [1] "1"  "12" "13"
```

Ja nääd võtame välja ajapunktide koodid. Kõigepealt asendame stringid, mis sisaldavad vähemalt üht numbrit, millele järgneb F v f tühja stringiga. Seejärel asendame tühja stringiga h-d. Ja lõpuks avaldame iga ajapunkti numbrina (mitte enam stringina).

```
library(tidyverse)
str_replace_all(v_f, "[0-9]+[Ff]", "") %>% str_replace_all("h", "") %>% as.i
#> [1] 1 2 1
```

0.10 Funktsioonid on R keele verbid

Kasutaja ütleb nii täpselt kui oskab, mida ta tahab ja R-s elab kratt, kes püüab ära arvata, mida on vaja teha. Vahest teeb kah. Vahest isegi seda, mida kasutaja tahtis. Mõni arvab, et R-i puudus on veateadete puudumine või krüptilised veateated. Sama kehtib ka R-i helpi kohta. Seega tasub alati kontrollida, kas R ikka tegi seda, mida sina talle enda arust ette kirjutasid.

Paljudel juhtudel ütleb (hea) funktsiooni nimi mida see teeb:

```
# create two test vectors
x <- c(6, 3, 3, 4, 5)
y <- c(1, 3, 4, 2, 7)
```

```
# calculate correlation
cor(x, y)
#> [1] -0.117
# calculate sum
sum(x)
#> [1] 21
# calculate sum of two vectors
sum(x, y)
#> [1] 38
# calculate average
mean(x)
#> [1] 4.2
```

```r
# calculate median
median(x)
#> [1] 4
# calculate standard deviation
sd(x)
#> [1] 1.3
# return quantiles
quantile(x)
#>   0%  25%  50%  75% 100%
#>    3    3    4    5    6
# return maximum value
max(x)
#> [1] 6
# return minimum value
min(x)
#> [1] 3
```

R-is teevad asju programmikesed, mida kutsutakse **funktsioonideks**. Te võite mõelda funktsioonist nagu verbist. Näiteks funktsiooni `sum()` korral loe: "võta summa". Iga funktsiooni nime järel on sulud. Nende sulgude sees asuvad selle funktsiooni **argumendid**. Argumendid määravad ära funktsiooni käitumise. Et näha, millised argumendid on funktsiooni käivitamiseks vajalikud ja milliseid on üldse võimalik seadistada, kasuta 'help' käsku.

```r
?sum
```

Help paneelis paremal all ilmub nüüd selle funktsiooni R dokumentatsioon. Vaata seal peatükki Usage: `sum(..., na.rm = FALSE)` ja edasi peatükki Arguments, mis ütleb, et `...` (ellipsis) tähistab vektoreid.

```
sum {base}  R Documentation
Sum of Vector Elements

Description:

sum returns the sum of all the values present in its arguments.
```

```
Usage

sum(..., na.rm = FALSE)

Arguments

... - numeric or complex or logical vectors.

na.rm   - logical. Should missing values (including NaN) be removed?
```

Seega võtab funktsioon `sum()` kaks argumenti: vektori arvudest (või loogilise vektori, mis koosneb TRUE ja FALSE määrangutest), ning "na.rm" argumendi, millele saab anda väärtuseks kas, TRUE või FALSE. Usage ütleb ka, et vaikimisi on `na.rm = FALSE`, mis tähendab, et sellele argumendile on antud vaikeväärtus – kui me seda ise ei muuda, siis jäävad NA-d arvutusse sisse. Kuna NA tähendab "tundmatu arv" siis iga tehe NA-dega annab vastuseks "tundmatu arv" ehk NA (tundmatu arv + 2 = tundmatu arv). Seega NA tulemus annab märku, et teie andmetes võib olla midagi valesti.

```
## moodustame vektori
apples <- c(1, 34, 43, NA)
## arvutame summa
sum(apples, na.rm = TRUE)
#> [1] 78
```

Niimoodi saab arvutada summat vektorile nimega "apples".

Sisestades R käsureale funktsiooni ilma selle sulgudeta saab masinast selle funktsiooni koodi. Näiteks:

```
sum
#> function (..., na.rm = FALSE)  .Primitive("sum")
```

Tulemus näitab, et `sum()` on `Primitive` funktsioon, mis põhimõtteliselt tähendab, et ta põhineb C koodil ja ei kasuta R koodi.

0.10.1 Kirjutame R funktsiooni

Võib ju väita, et funktsiooni ainus mõte on peita teie eest korduvad vajalikud koodiread kood funktsiooni nime taha. Põhjus, miks R-s on funktsioonid, on korduse vähendamine, koodi loetavaks muutmine ja seega ka ruumi kokkuhoid. Koodi funktsioonidena kasutamine suurendab analüüside reprodutseeritavust, kuna funktsioonis olev kood pärineb ühest allikast, mitte ei ole paljude koopiatena igal pool laiali. See muudab pikad koodilõigud hõlpsalt taaskasutatavaks sest lihtsam on kirjutada lühike funktsiooni nimi ja sisestada selle funktsiooni argumendid. Koodi funktsioonidesse kokku surumine vähendab võimalusi lollideks vigadeks, mida te võite teha pikkade koodijuppidega manipuleerides. Seega tasub teil õppida ka oma korduvaid koodiridu funktsioonidena vormistama.

Kõige pealt kirjutame natuke koodi.

```
# two apples
apples <- 2
# three oranges
oranges <- 3
# parentheses around expression assigning result to an object
# ensure that result is also printed to R console
(inventory <- apples + oranges)
#> [1] 5
```

Ja nüüd pakendame selle tehte funktsiooni `add2()`. Funktsiooni defineerimiseks kasutame järgmist r ekspressiooni `function( arglist ) expr`, kus "arglist" on tühi või ühe või rohkema nimega argumenti kujul `name=expression`; "expr" on R-i ekspressioon st. kood mida see funktsioon käivitab. Funktsiooni viimane evlueeritav koodirida on see, mis tuleb välja selle funktsiooni outputina.

All toodud näites on selleks `x + y` tehte vastus.

```
add2 <- function(x, y) {
    x + y
}
```

Seda koodi jooksutades näeme, et meie funktsioon ilmub R-i En-

vironmenti, kuhu tekib Functions lahter. Seal on näha ka selle funktsiooni kaks argumenti, apples ja oranges.

Antud funktsiooni käivitamine annab veateate, sest funktsiooni argumentidel pole väärtusi:

```
## run function in failsafe mode
inventory <- try(add2())
#> Error in add2() : argument "x" is missing, with no default
## when function fails, error message is returned
class(inventory)
#> [1] "try-error"
## print error message
cat(inventory)
#> Error in add2() : argument "x" is missing, with no default
```

Andes funktsiooni argumentidele väärtused, saab väljundi:

```
## run function with proper arguments
inventory <- add2(x = apples, y = oranges)
## numeric vector is returned
class(inventory)
#> [1] "numeric"
## result
inventory
#> [1] 5
```

Nüüd midagi kasulikumat!

Funktsioon standrardvea arvutamiseks (baas R-s sellist funktsiooni ei ole): `sd()` funktsioon arvutab standardhälbe. Sellel on kaks argumenti: x and na.rm. Me teame, et SEM=SD/sqrt(N) kus N = length(x)

```
calc_sem <- function(x) {
  stdev <- sd(x)
  n <- length(x)
  stdev / sqrt(n)
}
```

x hoiab lihtsalt kohta andmetele, mida me tahame sinna funkt-

siooni suunata. `sd()`, `sqrt()` ja `length()` on olemasolevad baas R funktsioonid, mille me oma funktsiooni hõlmame.

```
## create numeric vector
numbers <- c(2, 3.4, 54, NA, 3)
calc_sem(numbers)
#> [1] NA
```

No jah, kui meil on andmetes tundmatu arv (`NA`) siis on ka tulemuseks tundmatu arv.

Sellisel juhul tuleb NA väärtused vektorist enne selle funktsiooni kasutamist välja visata:

```
numbers_filtered <- na.omit(numbers)
calc_sem(numbers_filtered)
#> [1] 12.8
```

On ka võimalus funktsiooni sisse kirjutada **NA väärtuste käsitlemine**. Näiteks, üks võimalus on **anda viga** ja funktsioon katkestada, et kasutaja saaks ise ühemõtteliselt oma andmetest NA väärtused eemaldada. Teine võimalus on funktsioonis **NA-d vaikimisi eemaldada** ja anda selle kohta näiteks teade.

NA-de vaikimisi eemaldamiseks on hetkel mitu võimalust, kasutame kõigepealt nö. valet lahendust:

```
calc_sem <- function(x) {
  ## kasutame sd funktsiooni argumenti na.rm
  stdev <- sd(x, na.rm = TRUE)
  n <- length(x)
  stdev / sqrt(n)
}

calc_sem(numbers)
#> [1] 11.5
```

See annab meile vale tulemuse sest `na.rm = TRUE` viskab küll NA-d välja meie vektorist aga jätab vektori pikkuse muutmata (`length(x)` rida).

Teeme uue versiooni oma funktsioonist, mis viskab vaikimisi välja puuduvad väärtused, kui need on olemas ja annab siis ka selle kohta hoiatuse.

```
## x on numbriline vektor
calc_sem <- function(x) {

  ## viskame NA väärtused vektorist välja
  x <- na.omit(x)

  ## kui vektoris on NA väärtusi, siis hoiatame kasutajat
  if(inherits(na.action(x), "omit")) {
    warning("Removed NAs from vector.\n")
  }

  ## arvutame standardvea kasutades filtreeritud vektorit
  stdev <- sd(x)
  n <- length(x)
  stdev / sqrt(n)
}

calc_sem(numbers)
#> Warning in calc_sem(numbers): Removed NAs from vector.
#> [1] 12.8
length(numbers)
#> [1] 5
```

Missugune funktsiooni käitumine valida, sõltub kasutaja vajadusest. Rohkem infot NA käsitlemise funktsioonide kohta saab `?na.omit` abifailist.

Olgu see õpetuseks, et funktsioonide kirjutamine on järk-järguline protsess ja sellele, et alati saab paremini teha.

0.11 Graafilised lahendused

R-s on kaks olulisemat graafikasüsteemi, mida võib vaadata nagu kaht eraldi dialekti, mis mõlemad elavad R keele sees.

- **Baasgraafika** võimaldab lihtsate vahenditega teha kiireid ja suhteliselt ilusaid graafikuid. Seda kasutame sageli enda tarbeks kiirete plottide tegemiseks. Baasgraafika abil saab teha ka väga keerukaid ja komplekssseid publitseerimiskavaliteedis graafikuid.
- **"ggplot2"** raamatukogu on hea ilupiltide vormistamiseks ja keskmiselt keeruliste visualiseeringute tegemiseks.

Kuigi "ggplot2" ja tema sateliit-raamatukogud on meie põhilised huviobjekid, alustame siiski baasgraafikast. Ehki me piirdume vaid väga lihtsate näidetega tasub teada, et baasgraafikas saab teha ka komplekseid visualiseeringuid: http://shinyapps.org/apps/RGraphCompendium/index.php

Laadime peatükis edaspidi vajalikud libraryd:

```
library(tidyverse)
library(ggthemes)
library(ggrepel)
library(wesanderson)
library(ggridges)
library(viridis)
library(zoo)
library(graphics)
```

0.11.1 Baasgraafika

Kõigepealt laadime tabeli, mida me visuaalselt analüüsima hakkame:

```
iris <- as_tibble(iris)
iris
#> # A tibble: 150 x 5
```

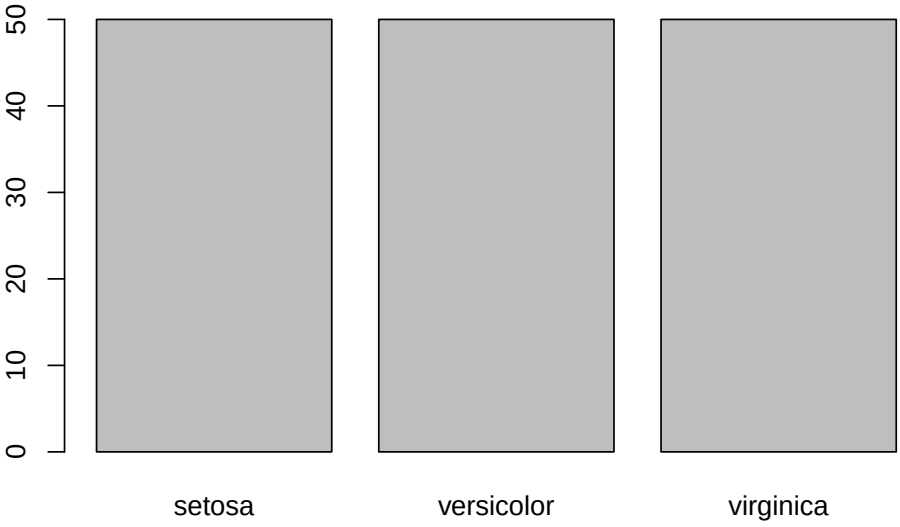
```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
#> 1          5.1         3.5          1.4         0.2
#> 2          4.9         3            1.4         0.2
#> 3          4.7         3.2          1.3         0.2
#> 4          4.6         3.1          1.5         0.2
#> 5          5           3.6          1.4         0.2
#> 6          5.4         3.9          1.7         0.4
#> # ... with 144 more rows, and 1 more variable:
#> #   Species <fct>
```

See sisaldab mõõtmistulemusi sentimeetrites kolme iirise perekonna liigi kohta. Esimest korda avaldati need andmed 1936. aastal R.A. Fisheri poolt.

Baasgraafika põhiverb on `plot()`. See püüab teie poolt ette antud andmete pealt ära arvata, millist graafikut te soovite. `plot()` põhiargumendid on x ja y, mis määravad selle, mis väärtused asetatakse x-teljele ja mis läheb y-teljele. Esimene argument on vaikimisi x ja teine y.

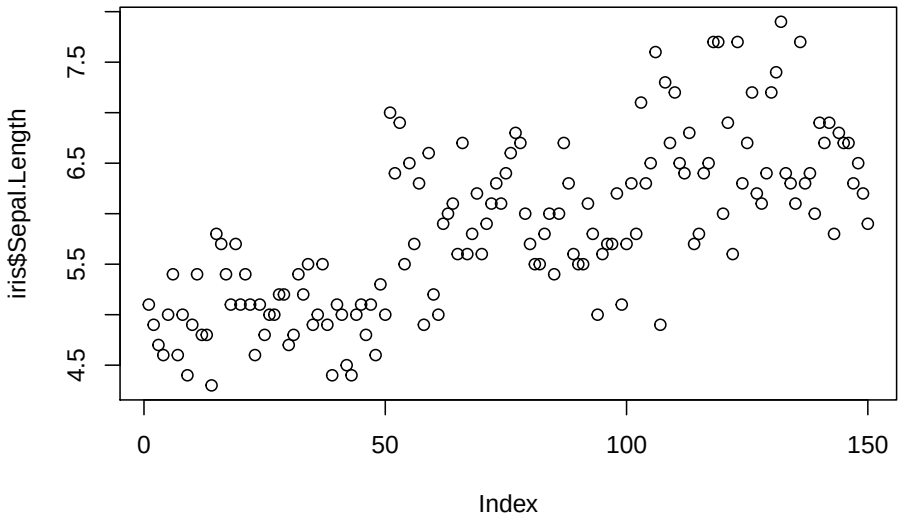
Kui te annate ette faktorandmed, on vastuseks tulpdiagramm, kus tulbad loevad üles selle faktori kõigi tasemete esinemiste arvu. Antud juhul on meil igast liigist mõõdetud 50 isendit.

```
plot(iris$Species)
```



Kui te annate ette ühe pideva muutuja:

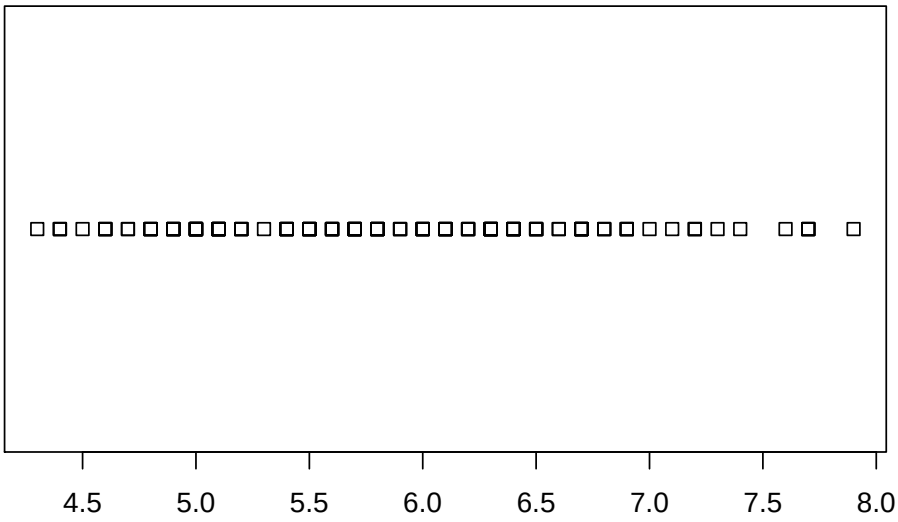
```
plot(iris$Sepal.Length)
```



Nüüd on tulemuseks graafik, kus on näha mõõtmisete rea (ehk tabeli) iga järgmise liikme (tabeli rea) väärtus. Siin on meil kokku 150 mõõtmist muutujale `Sepal.Length`.

Alternatiiv sellele vaatele on `stripchart()`

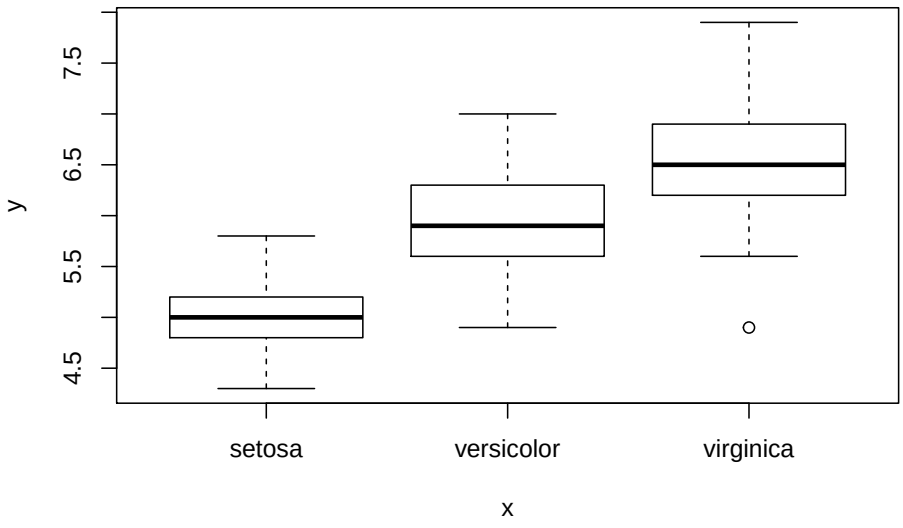
```
stripchart(iris$Sepal.Length)
```



Enam lihtsamaks üks joonis ei lähe!

Mis juhtub, kui me x-teljele paneme faktortunnuse ja y-teljele pideva tunnuse?

```
plot(iris$Species, iris$Sepal.Length)
```



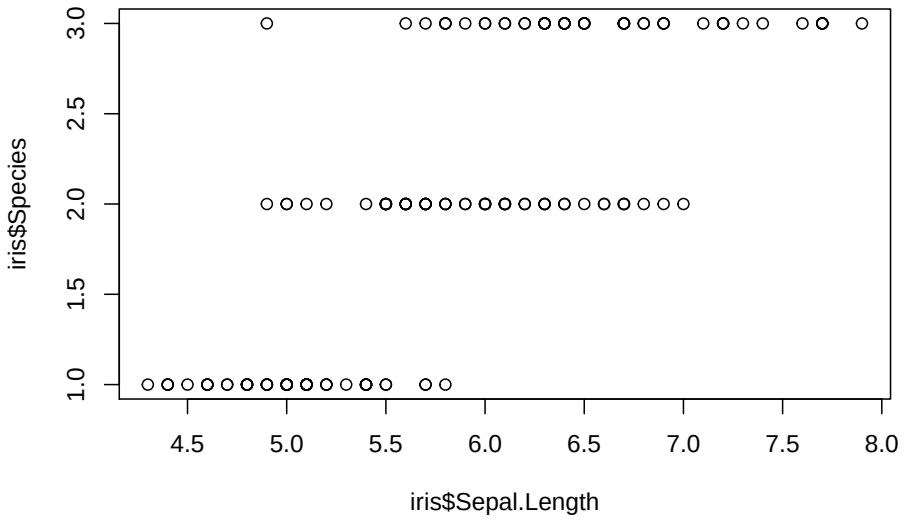
Vastuseks on boxplot. Sama graafiku saame ka nii:

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

Siin on tegu R-i mudeli notatsiooniga: y-telje muutuja, tilde, x-telje muutuja. Tilde näitab, et y sõltub x-st stohhastiliselt, mitte deterministlikult. Deterministliku seost tähistatakse võrdusmärgiga (=).

Aga vastupidi?

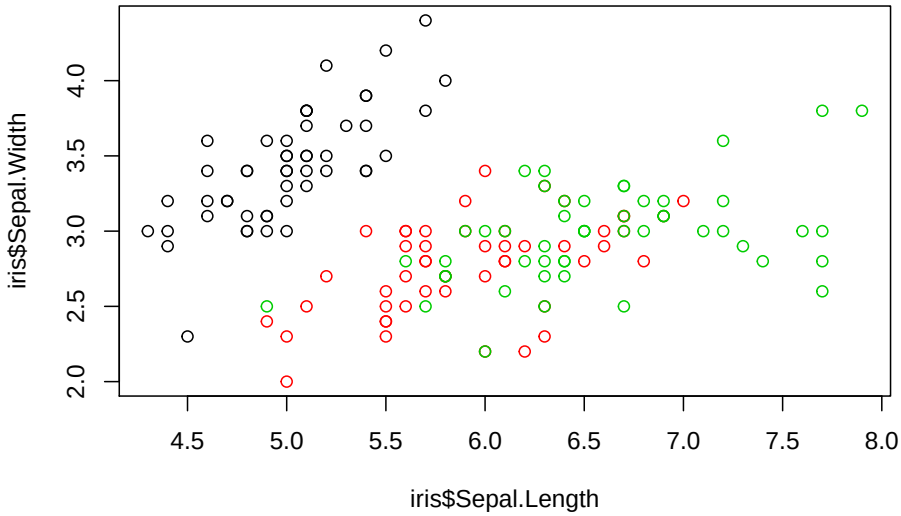
```
plot(iris$Sepal.Length, iris$Species)
```



Pole paha, see on üsna informatiivne scatterplot.

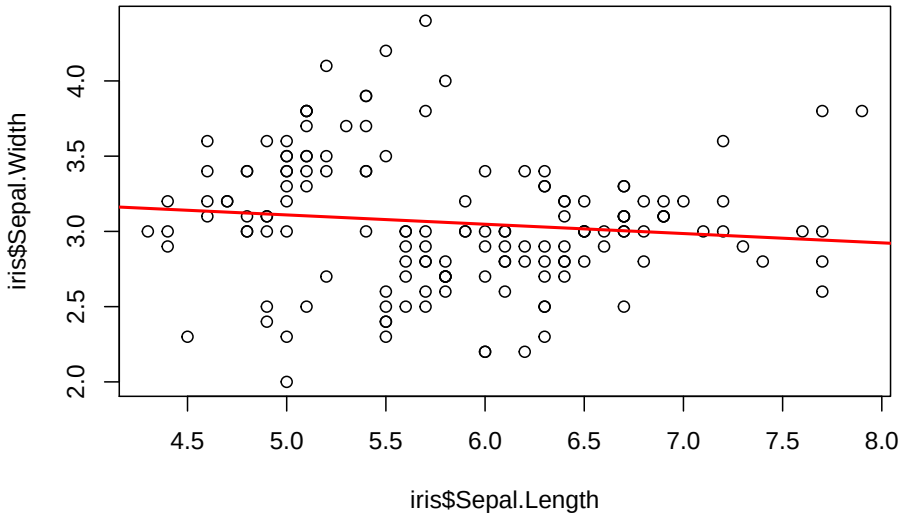
Järgmiseks kahe pideva muutuja scatterplot, kus me veel lisaks värvime punktid liikide järgi.

```
plot(iris$Sepal.Length, iris$Sepal.Width, col = iris$Species)
```



Ja lõpuks tõmbame läbi punktide punase regressioonijoone:

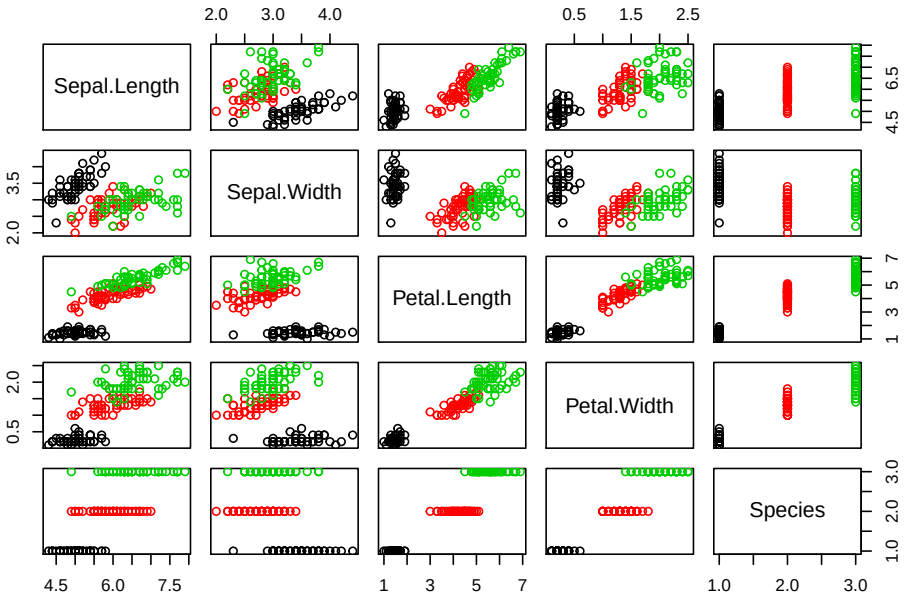
```
plot(iris$Sepal.Length, iris$Sepal.Width)
model <- lm(iris$Sepal.Width ~ iris$Sepal.Length)
abline(model, col = "red", lwd = 2)
```



"lwd" parameeter reguleerib joone laiust. `lm()` on funktsioon, mis fitib sirge vähimruutude meetodil.

Mis juhtub, kui me anname `plot()` funktsioonile sisse kogu irise tibble?

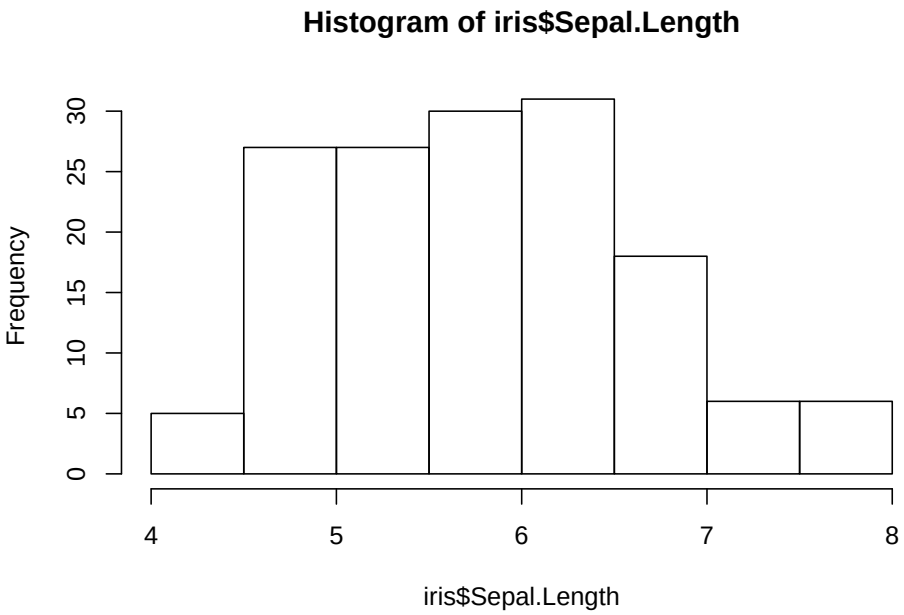
```
plot(iris, col = iris$Species)
```



Juhhei, tulemus on paariviisiline graafik kõigist muutujate kombinatsioonidest.

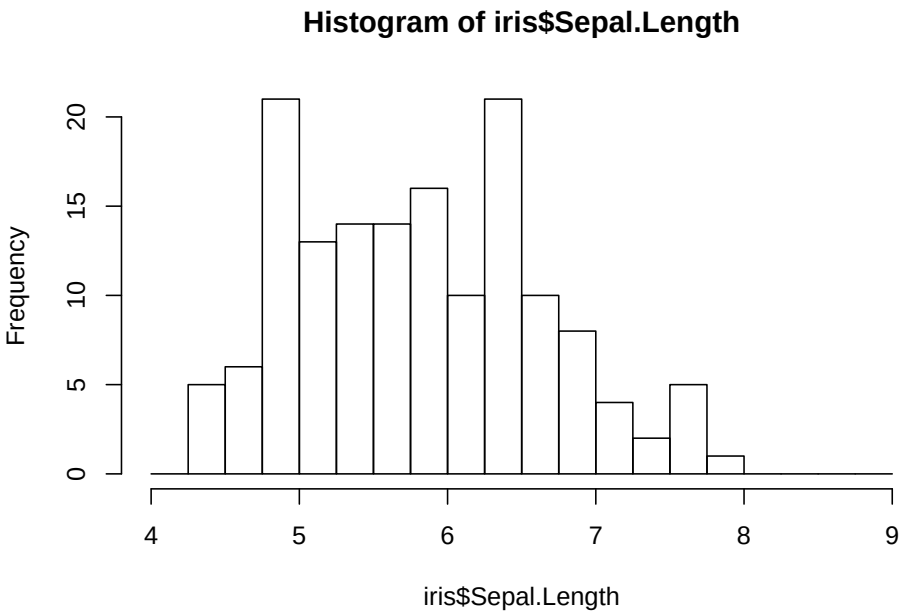
Ainus mitte-plot verb, mida baasgraafikas vajame, on `hist()`, mis joonistab histogrammi.

```
hist(iris$Sepal.Length)
```



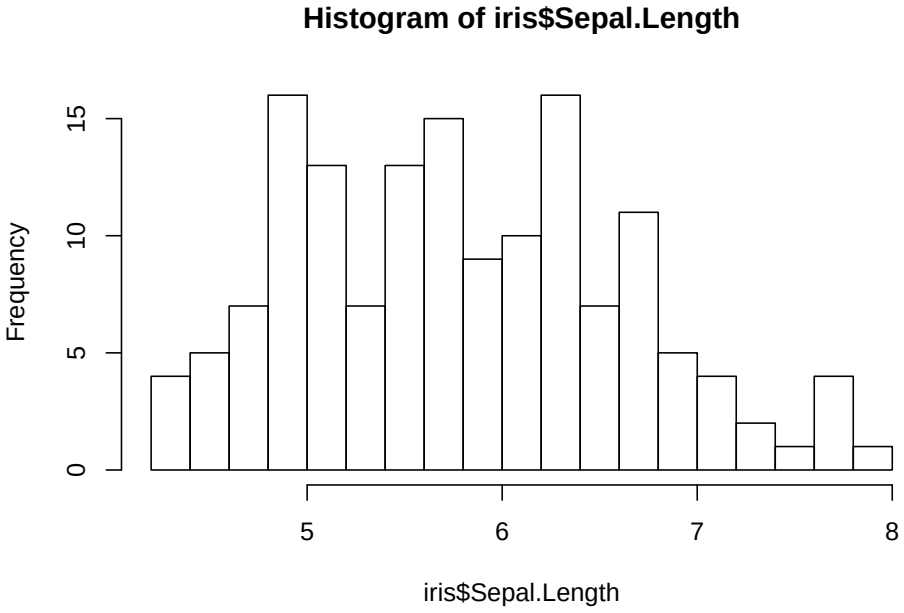
Histogrammi tegemiseks jagatakse andmepunktid nende väärtuste järgi bin-idesse ja plotitakse igasse bin-i sattunud andmepunktide arv. Näiteks esimeses bin-is on "Sepal.Length" muutuja väärtused, mis jäävad 4 ja 4.5 cm vahele ja selliseid väärtusi on kokku viis. Histogrammi puhul on oluline teada, et selle kuju sõltub bin-ide laiusest. Bini laiust saab muuta kahel viisil, andes ette bin-ide piirid või arvu:

```
hist(iris$Sepal.Length, breaks = seq(4, 9, by = 0.25))
```



với

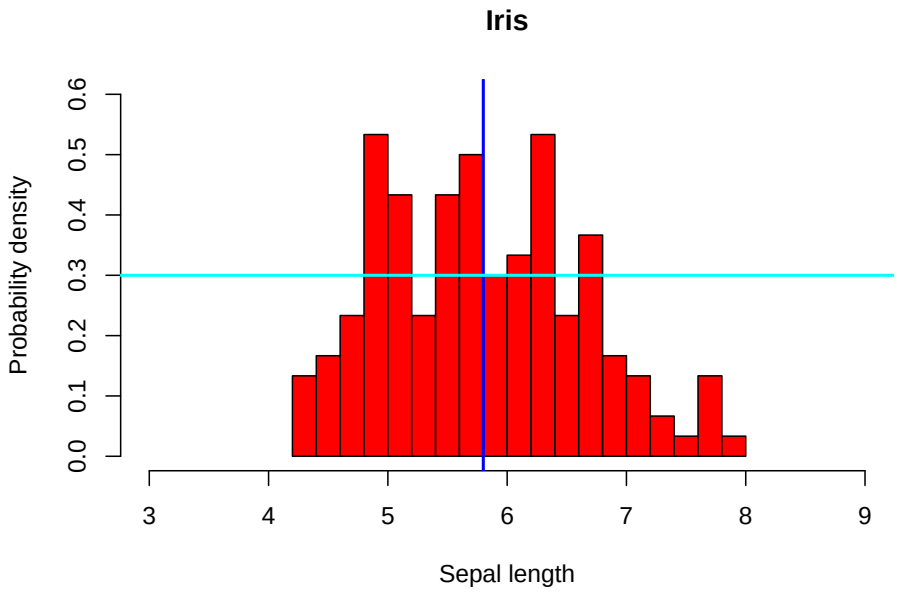
```
hist(iris$Sepal.Length, breaks = 15)
```



See viimane on kiire viis bin-i laiust reguleerida, aga arvestage, et sõltuvalt andmetest ei pruugi "breaks = 15" tähendada, et teie histogrammil on 15 bin-i.

Ja lõpuks veel üks histogramm, et demonstreerida baas R-i võimalusi (samad argumendid töötavad ka `plot()` funktsioonis):

```
hist(iris$Sepal.Length,
     freq = FALSE,
     col="red",
     breaks = 15,
     xlim = c(3, 9),
     ylim = c(0, 0.6),
     main = "Iris",
     xlab = "Sepal length",
     ylab = "Probability density")

abline(v = median(iris$Sepal.Length), col = "blue", lwd = 2)
abline(h = 0.3, col = "cyan", lwd = 2)
```



0.11.2 ggplot2

Ggplot on avaldamiseks sobiva tasemega lihtne aga võimas graafikasüsteem. Näiteid selle abil tehtud visualiseeringutest leiab näiteks järgnevatelt linkidelt:

- http://ggplot2.tidyverse.org/reference/
- http://www.r-graph-gallery.com
- http://www.ggplot2-exts.org
- http://www.cookbook-r.com

"ggplot2" paketi põhiverb on `ggplot()`. See graafikasüsteem töötab kiht-kihi-haaval ja uusi kihte lisatakse pluss-märgi abil. See annab modulaarsuse kaudu lihtsuse ja võimaluse luua ka keerulisi taieseid. Tõenäoliselt on ggplot hetkel kättesaadavatest graafikasüsteemidest parim (kaasa arvatud tasulised programmid!).

0.11.2.1 graafika "keel"

Millised elemendid on igal endast lugupidaval graafikul?

Esiteks teljestik ehk graafiku ruum. Isegi siis, kui telgi pole välja joonistatud, on nad tegelikult alati olemas ja määravad akna, milles olevaid andmeid graafikul kuvatakse. Teljed võivad olla andmetega samas "ruumis" või transformeeritud (näit. logaritmitud). Lisaks on teljedel on suund (vasakult paremale, ülevalt alla, ringiratast, jms). Vastavalt teljestikule võib graafiku kuju olla 1D sirge, 2D ruut, 3D kuup, kera, ristkülik, trapets vms.

Teiseks graafikule kaardistatud muutujad. Näit võib x teljele kaardistada pideva muutuja "Sepal.Width" ja y teljele pideva muutuja Sepal.Length. Lisaks võime igale andmepunktile kaardistada näiteks värvi, mis vastab faktormuutuja Species tasemele. Aga võib ka x teljele kaardistada muutuja Sepal.Length ja y telje kaardistamata jätta. Esimesel juhul on võimalik joonistada näiteks scatter plot või line plot, teisel juhul aga histogramm või tihedusplot. Seega on graafiku tüüp veel lahtine.

Kolmandaks graafiku tüüp. Osad tüübid kasutavad andmeid

otse (n scatter plot), teised arvutavad andmete pealt statistiku, ja plotivad selle (n histogramm). Graafikul on see nn data ink.

Neljandaks graafiku esteeetika. Siia kuuluvad telgede paksused, värvid, tähistused, abijooned, taustavärvid ja kõik muu, mis otseselt ei kajasta andmeid. Graafikul on see nn non-data ink. Oluline on suhe data ink/non-data ink. Kui see suhe on väga madal, siis on teie graafiku teaduslik sisu raskesti leitav ja ilmselt peaks graafikut muutma. Samas, kui see suhe on väga kõrge, tekib oht, et puudu on tähelepanu õigesse kohta juhtivad abijooned ja muu selline, mistõttu andmed ripuvad nagu õhus.

Ongi kõik. Teljestik sinna kaardistatud muutujatega, graafiku tüüp ja teema on kõik, mida me vajame, et teha ükskõik milline joonis

ggploti töövoog kajastab seda kiht-kihi haaval ideoloogiat. Kihid eraldatakse omavahel "+" märgiga. Minimaalselt pead ggplot() funktsioonile andma kolm asja:

1. **andmed**, mida visualiseeritakse,
2. `aes()` funktsiooni, mis määrab, milline muutuja läheb x-teljele ja milline y-teljele, ning
3. **geom**, mis määrab, mis tüüpi visualiseeringut sa tahad.

Lisaks määrad sa `aes()`-is, kas ja kuidas sa tahad grupeerida pidevaid muutujaid faktori tasemete järgi.

Lisakihtides saab dikteerida,

4. kuidas joonise elemendid jagada erinevate paneelide (facet, small multiple) vahel, kasutades facet_wrap() ja facet_grid() funktsioone.
5. määrata joonise teema theme() funktsiooni abil ning manipuleerida värve, telgi jms erinevate abifunktsioonidega.

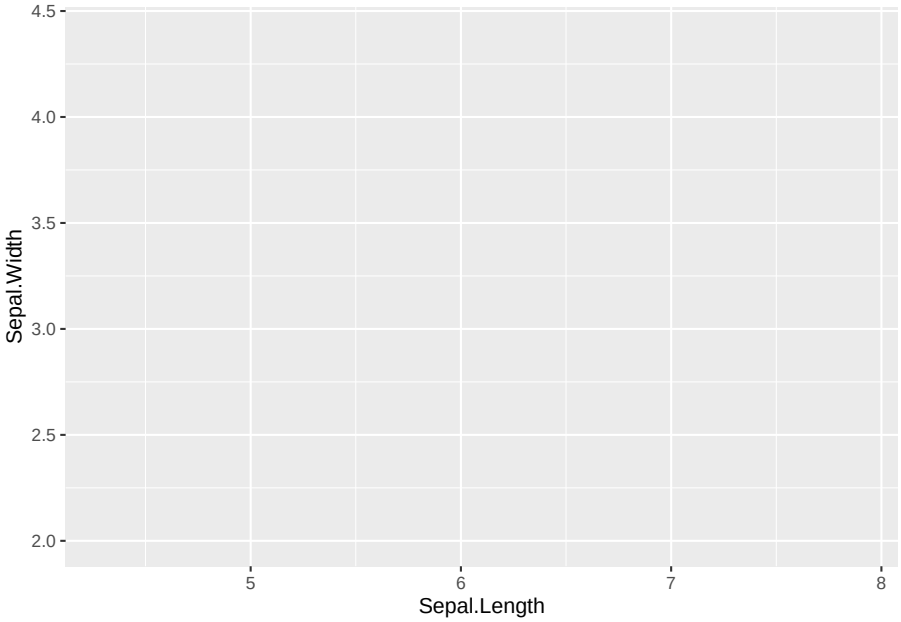
Kõigepealt suuname oma andmed `ggplot()` funktsiooni:

```
ggplot(iris)
```


Saime tühja ploti. ggplot() loob vaid koordinaatsüsteemi, millele saab kihte lisada. Erinevalt baasgraafikast, ggplot-i puhul ainult andmetest ei piisa, et graafik valmis joonistataks. Vaja on lisada kiht-kihilt instruktsioonid, kuidas andmed graafikule paigutada ja missugust graafikutüüpi visualiseerimiseks kasutada.

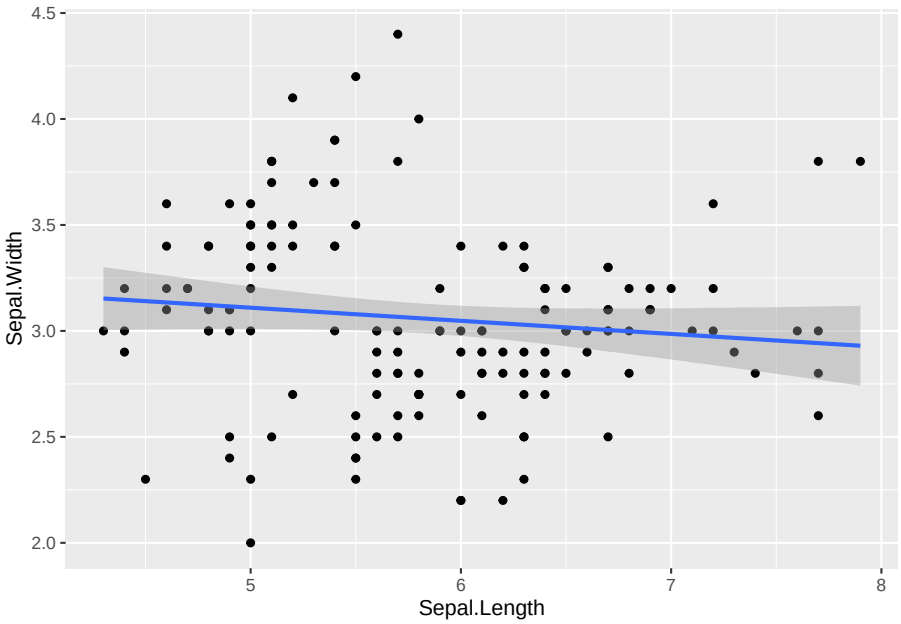
Nüüd ütleme, et x-teljele pannakse "Sepal.Length" ja y-teljele "Sepal.Width" andmed. Pane siin tähele, et me suuname kõigepealt selle ploti objekti p ja alles siis trükime selle ggplot objekti välja. Meie näites lisame edaspidi kihte sellele ggplot objektile.

```
p <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))
p
```



Graafik on ikka tühi sest me pole ggplotile öelnud, millist visualiseeringut me tahame. Teeme seda nüüd ja lisame andmepunktid kasutades `geom_point`-i ja lineaarse regressioonijoone kasutades `geom_smooth` funktsiooni koos argumendiga `method = lm`. Ka nüüd täiendame ggplot objekti p uute kihtidega:

```
p <- p + geom_point() + geom_smooth(method = lm)
p
```



Veelkord, me lisasime kaks kihti: esimene kiht `geom_point()` visualiseerib andmepunktid ja teine `geom_smooth(method = "lm")` joonistab regressioonisirge koos usaldusintervalliga (standardviga).

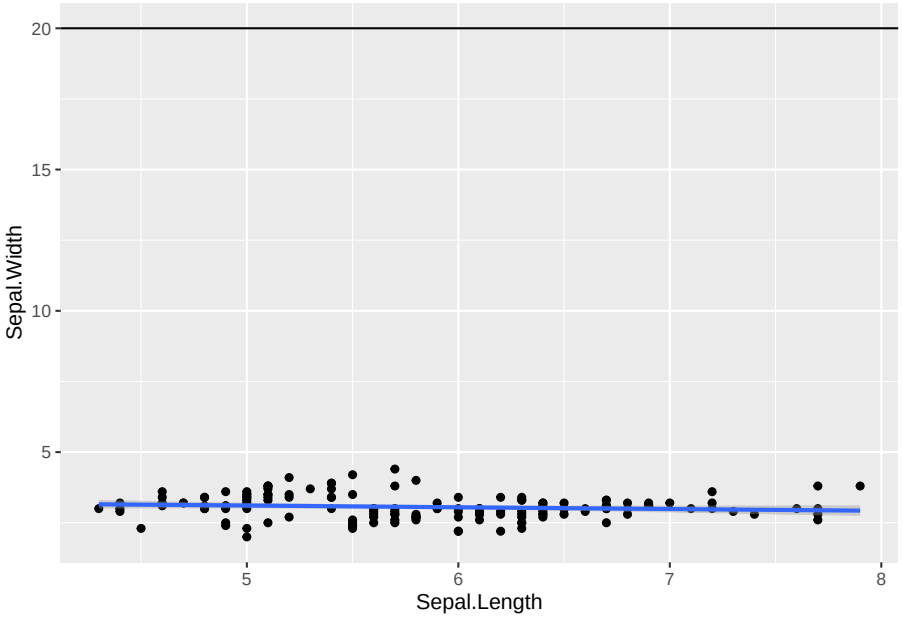
> Plussmärk peab ggplot-i koodireas olema vana rea lõpus, mitte uue rea (kihi) alguses

0.11.2.2 Lisame plotile sirgjooni

Horisontaalsed sirged saab graafikule lisada `geom_hline()` abil. Pane tähele, et eelnevalt me andsime oma ggplot-i põhikihtidele nime "p" ja seega panime selle alusploti oma töökeskkonda, et saaksime seda korduvkasutada.

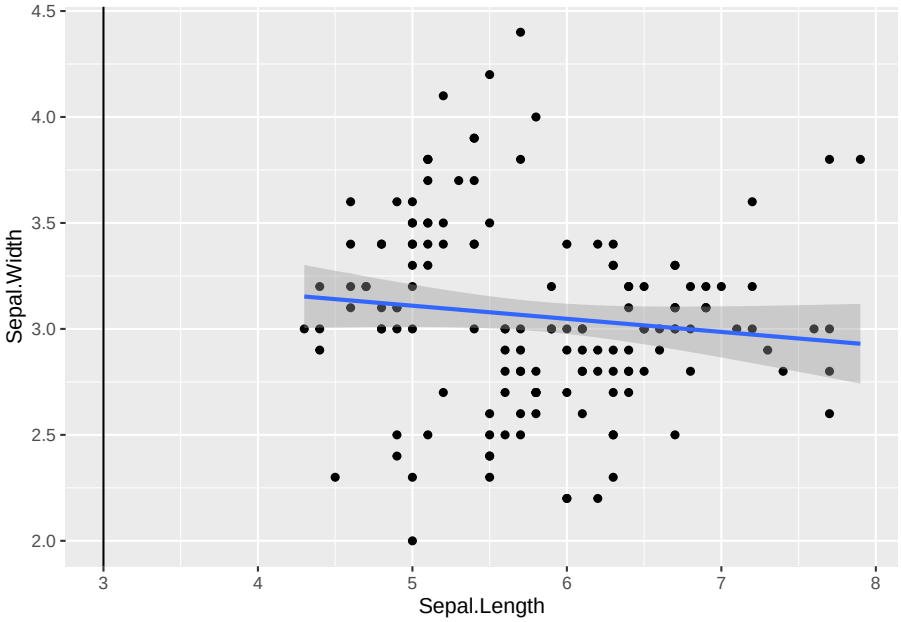
Lisame graafikule p horisontaaljoone y = 20:

```
# Add horizontal line at y = 20
p + geom_hline(yintercept = 20)
```



Vertikaalseid sirgeid saab lisada `geom_vline()` abil, näiteks vertikaalne sirge asukohas x = 3:

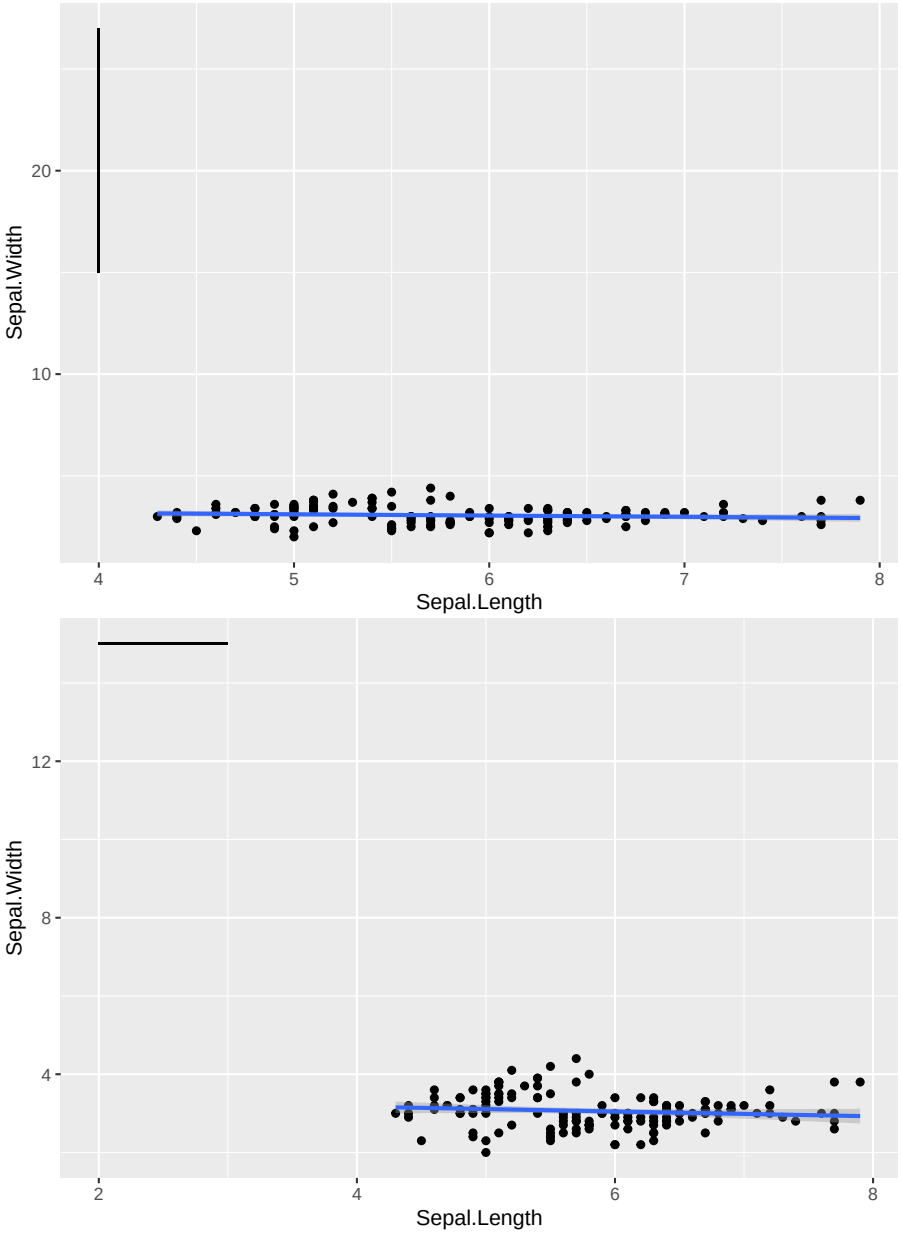
```
# Add a vertical line at x = 3
p + geom_vline(xintercept = 3)
```



0.11.2.3 Segmendid ja nooled

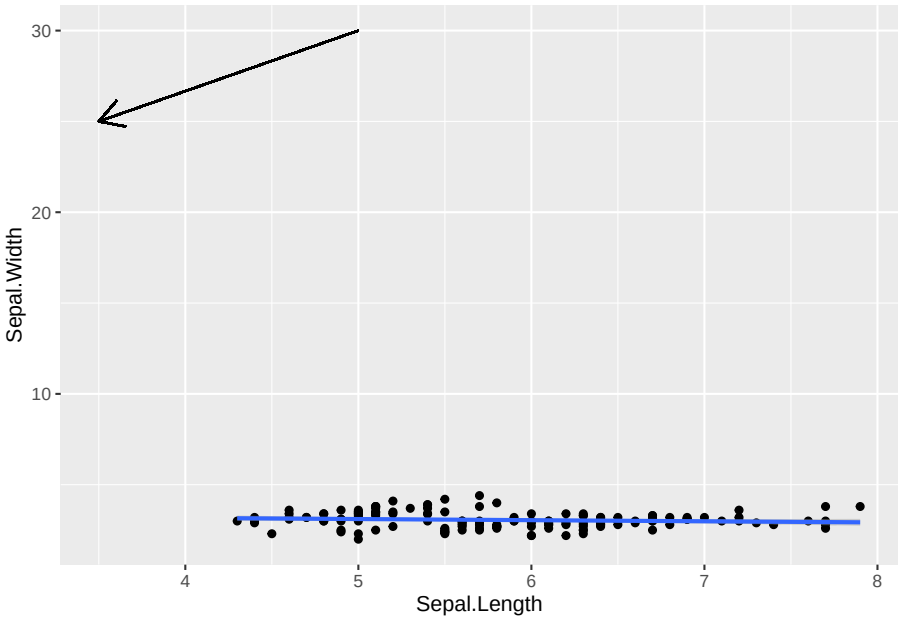
"ggplot2" funktsioon `geom_segment()` lisab joonejupi, mille algus ja lõpp on ette antud.

```
# Add a vertical line segment
p + geom_segment(aes(x = 4, y = 15, xend = 4, yend = 27))

# Add horizontal line segment
p + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15))
```



Saab joonistada ka **nooli**, kasutades arumenti "arrow" funktsioonis `geom_segment()`

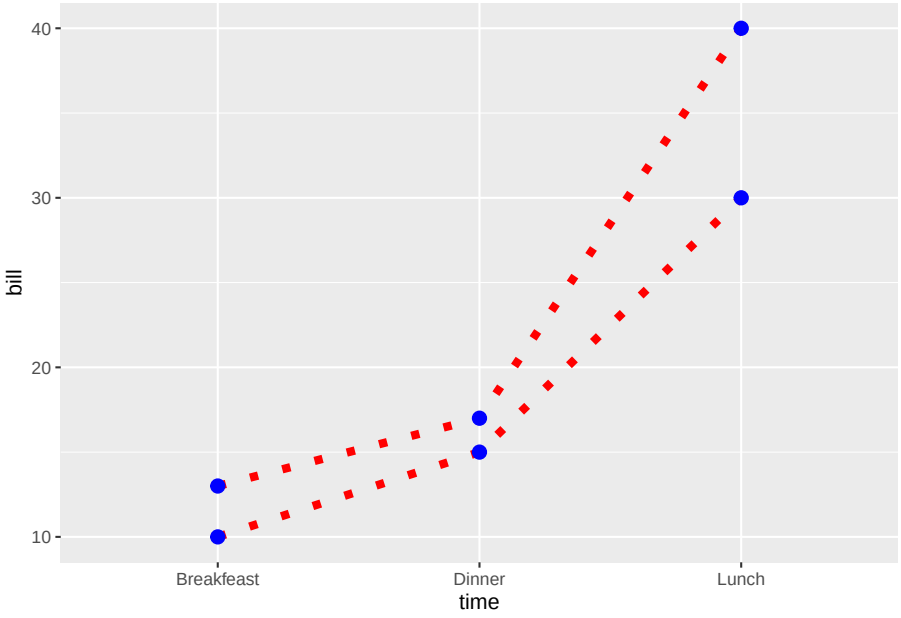
```
p + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
                 arrow = arrow(length = unit(0.5, "cm")))
```



0.11.2.4 Joongraafikud

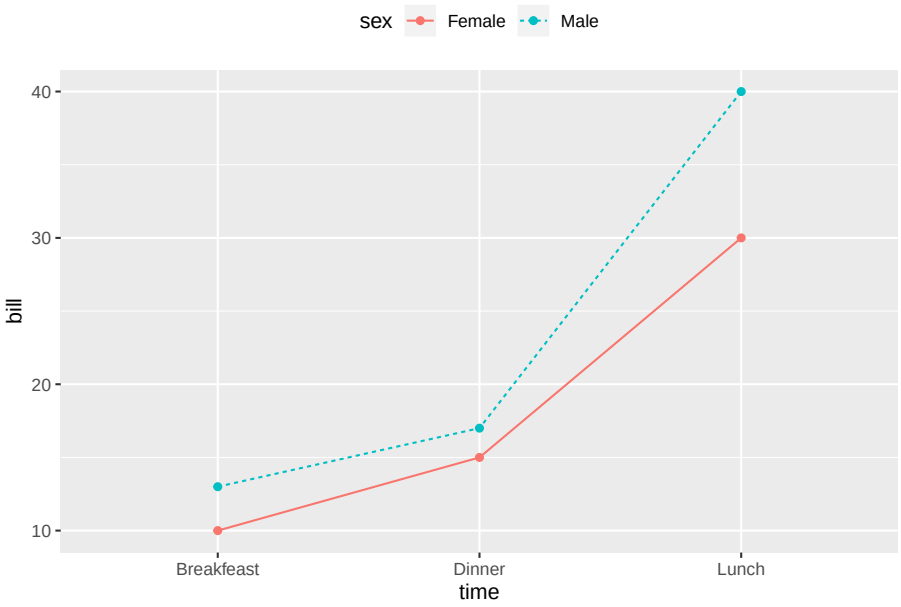
"ggplot2"-s on näiteks joonetüübid on "blank", "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

```
meals <- data.frame(sex = rep(c("Female", "Male"), each = 3),
                    time = c("Breakfeast", "Lunch", "Dinner"),
                    bill = c(10, 30, 15, 13, 40, 17) )

# Change line colors and sizes
ggplot(data = meals, aes(x = time, y = bill, group = sex)) +
  geom_line(linetype = "dotted", color = "red", size = 2) +
  geom_point(color = "blue", size = 3)
```



Järgneval graafikul muudame joonetüüpi automaatselt muutuja sex taseme järgi:

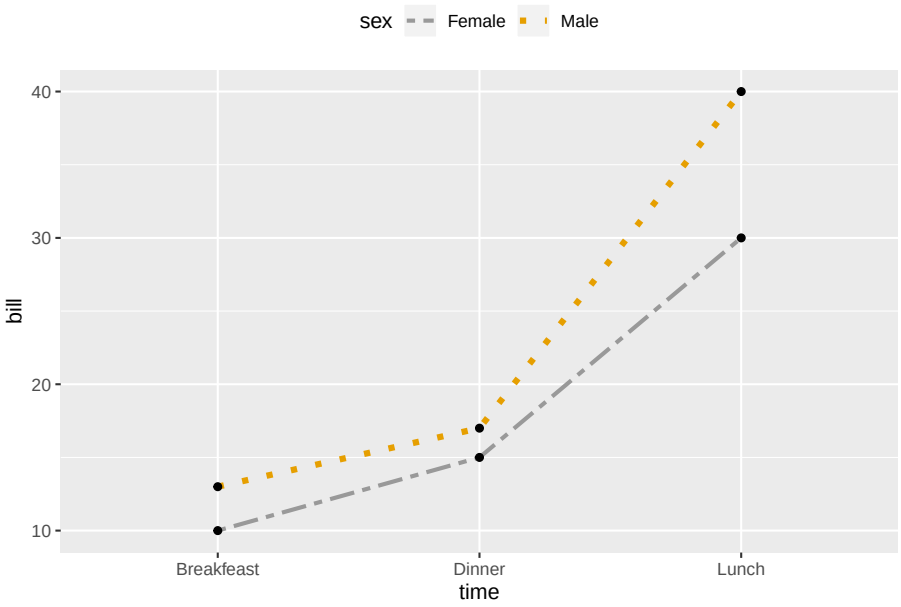
```
# Change line types + colors
ggplot(meals, aes(x = time, y = bill, group = sex)) +
  geom_line(aes(linetype = sex, color = sex)) +
  geom_point(aes(color = sex)) +
  theme(legend.position = "top")
```



Muuda jooni käsitsi:

- `scale_linetype_manual()`: joone tüüp
- `scale_color_manual()`: joone värv
- `scale_size_manual()`: joone laius

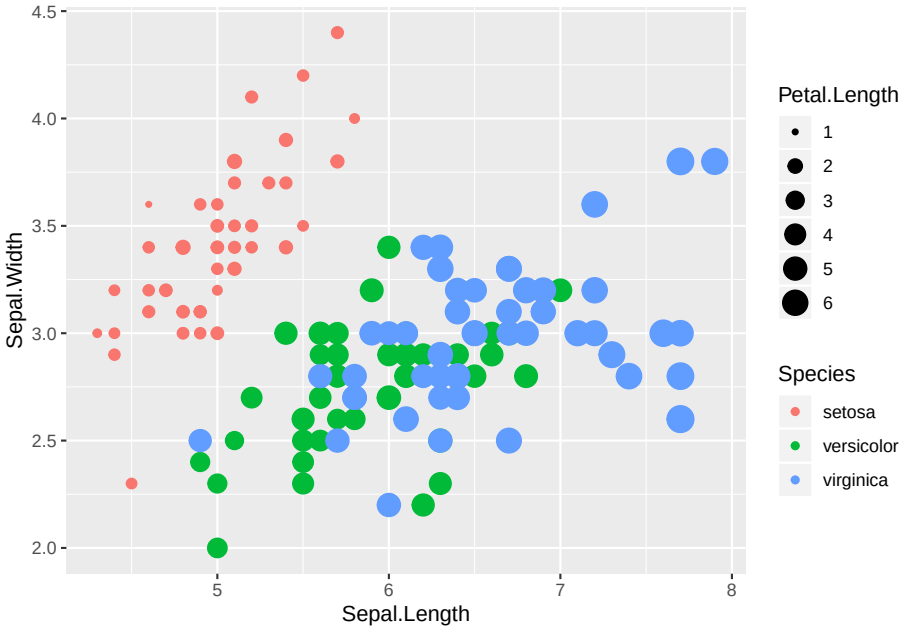
```
ggplot(meals, aes(x = time, y = bill, group = sex)) +
  geom_line(aes(linetype = sex, color = sex, size = sex)) +
  geom_point() +
  scale_linetype_manual(values = c("twodash", "dotted")) +
  scale_color_manual(values = c('#999999', '#E69F00')) +
  scale_size_manual(values = c(1, 1.5)) +
  theme(legend.position = "top")
```



0.11.2.5 Punktide tähistamise trikid

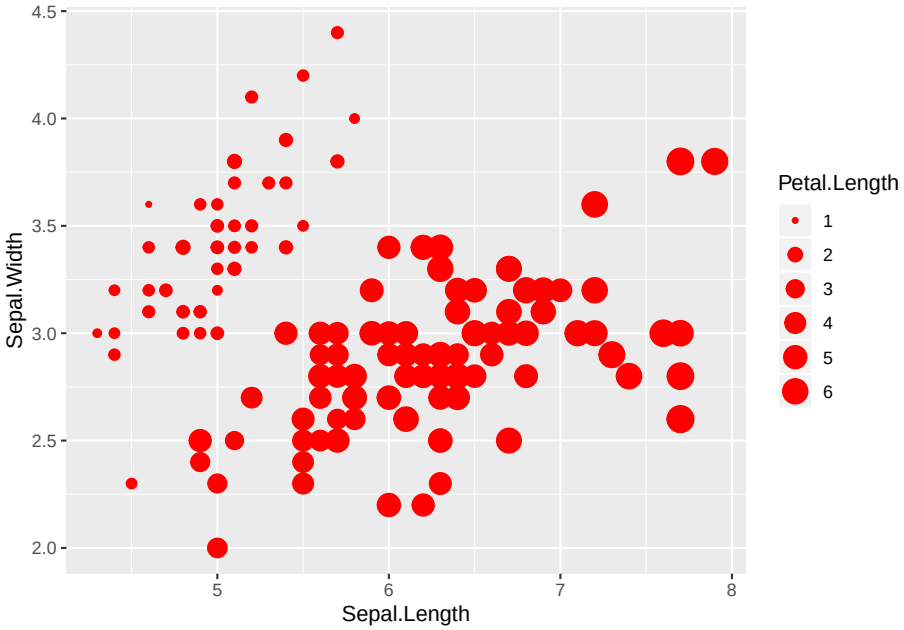
`aes()` töötab nii `ggplot()` kui `geom_` funktsioonides.

```
ggplot(iris) +
  geom_point(aes(x = Sepal.Length, y = Sepal.Width, size = Petal.Length, col
```



Kui me kasutame color argumenti `aes()`-st väljaspool, siis värvime kõik punktid sama värvi.

```
ggplot(iris) +
  geom_point(aes(x = Sepal.Length, y = Sepal.Width, size = Petal.Length), co
```



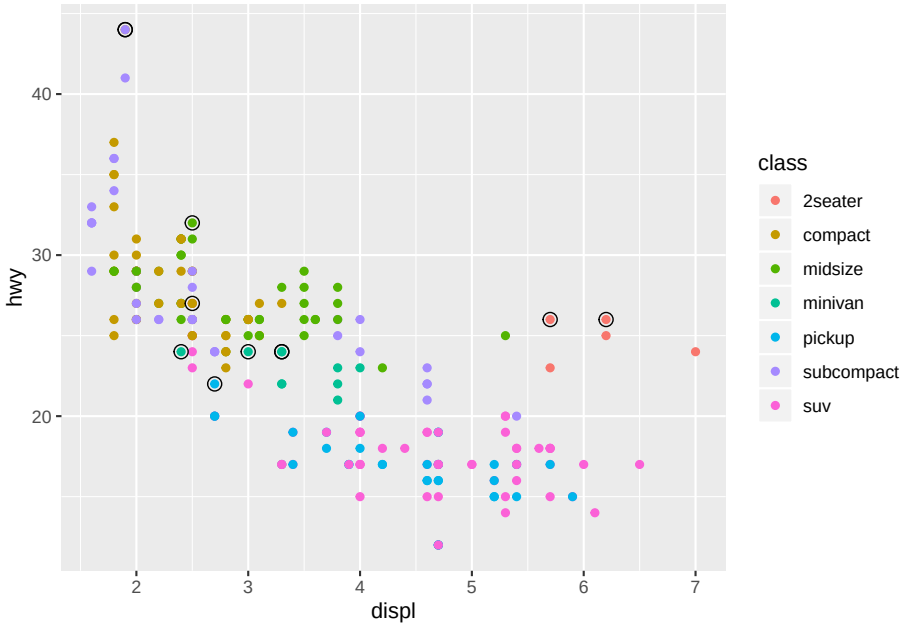
Kasulik trikk on kasutada mitut andmesetti sama ploti tegemiseks. Uus andmestik – "mpg" – on autode kütusekulu kohta.

```
head(mpg, 2)
#> # A tibble: 2 x 11
#>   manufacturer model displ  year   cyl trans drv
#>   <chr>        <chr> <dbl> <int> <int> <chr> <chr>
#> 1 audi         a4      1.8  1999     4 auto~ f
#> 2 audi         a4      1.8  1999     4 manu~ f
#> # ... with 4 more variables: cty <int>, hwy <int>,
#> #   fl <chr>, class <chr>

best_in_class <- mpg %>%
  group_by(class) %>%
  top_n(1, hwy)

head(best_in_class)
#> # A tibble: 6 x 11
#> # Groups:   class [2]
#>   manufacturer model displ  year   cyl trans drv
```

```
#>   <chr>        <chr> <dbl> <int> <int> <chr> <chr>
#> 1 chevrolet    corv~   5.7  1999     8 manu~ r
#> 2 chevrolet    corv~   6.2  2008     8 manu~ r
#> 3 dodge        cara~   2.4  1999     4 auto~ f
#> 4 dodge        cara~   3    1999     6 auto~ f
#> 5 dodge        cara~   3.3  2008     6 auto~ f
#> 6 dodge        cara~   3.3  2008     6 auto~ f
#> # ... with 4 more variables: cty <int>, hwy <int>,
#> #   fl <chr>, class <chr>
```

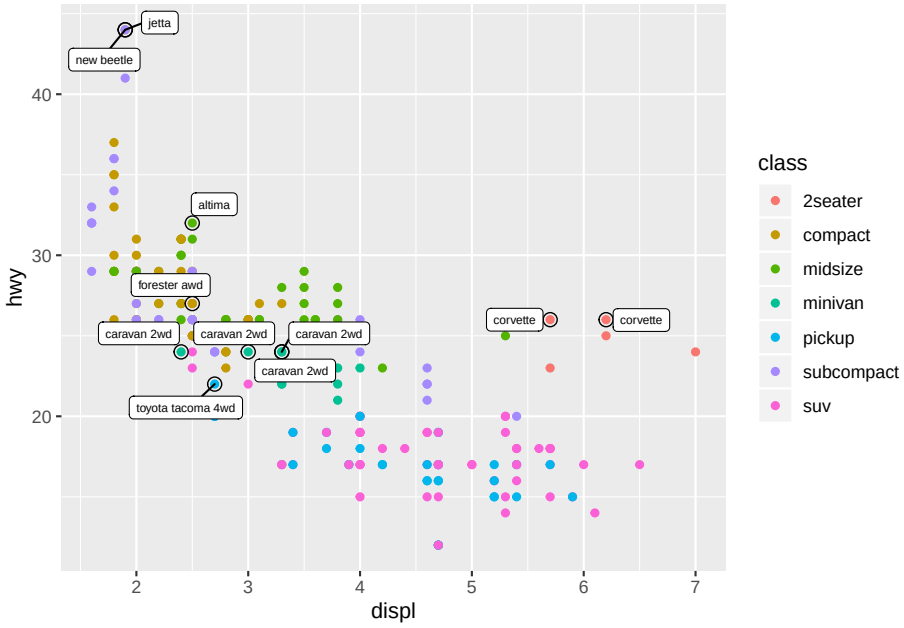
Siin läheb kitsam andmeset uude `geom_point()` kihti ja teeb osad punktid teistsuguseks. Need on oma klassi parimad autod.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class))+
  geom_point(size = 3, shape = 1, data = best_in_class)
```



Lõpuks toome graafikul eraldi välja nende parimate autode mudelite nimed. Selleks kasutame “ggrepel” raamatukogu funktsiooni `geom_label_repel()`.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class))+
  geom_point(size = 3, shape = 1, data = best_in_class) +
  geom_label_repel(aes(label = model), data = best_in_class, cex = 2)
```



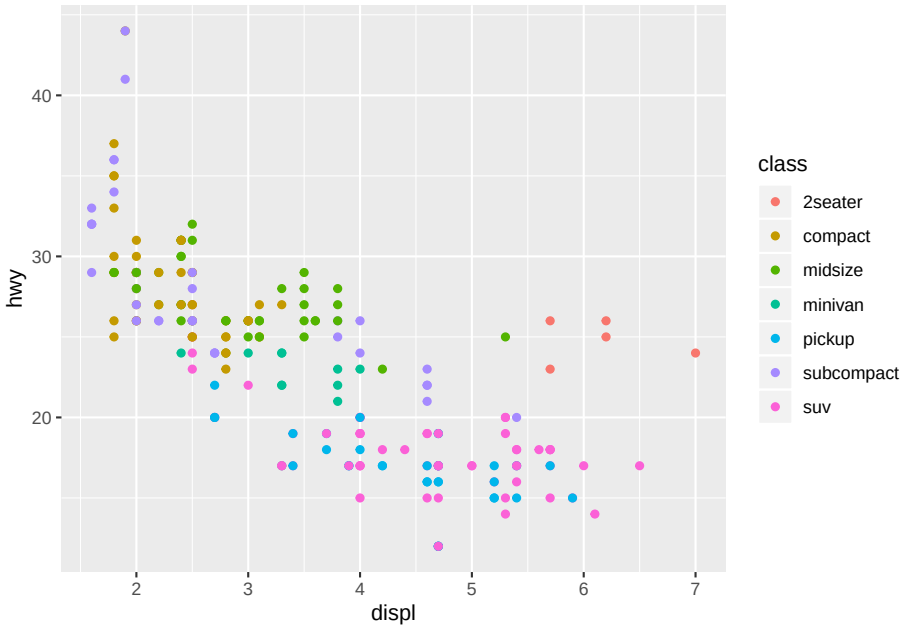
0.11.3 *Facet* – pisigraafik

Kui teil on mitmeid muutujaid või nende alamhulki, on teil kaks võimalust.

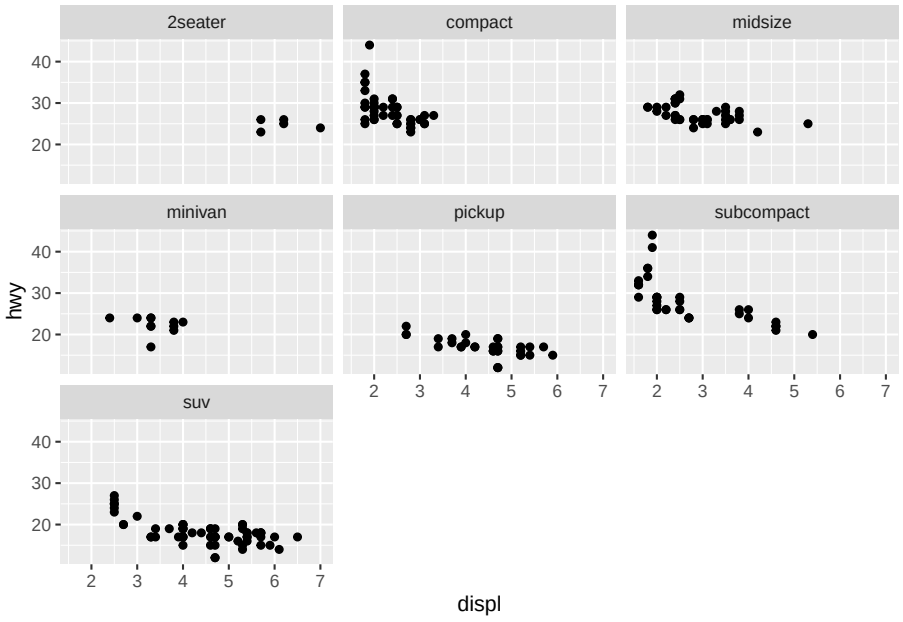
1. grupeeri pidevad muutujad faktormuutujate tasemete järgi ja kasuta color, fill, shape, size alpha parameetreid, et erinevatel gruppidel vahet teha.
2. grupeeri samamoodi ja kasuta facet-it, et iga grupp omaenda paneelile panna.

```
# here we separate different classes of cars into different colors
```

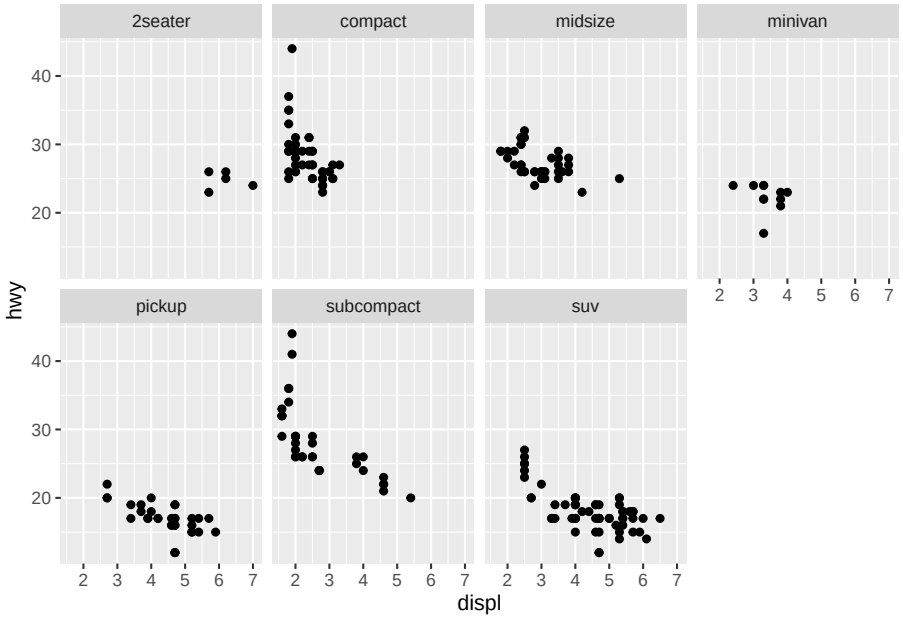
```
p <- ggplot(mpg, aes(displ, hwy))
p + geom_point(aes(colour = class))
```




```
p + geom_point() +
  facet_wrap(~ class)
```

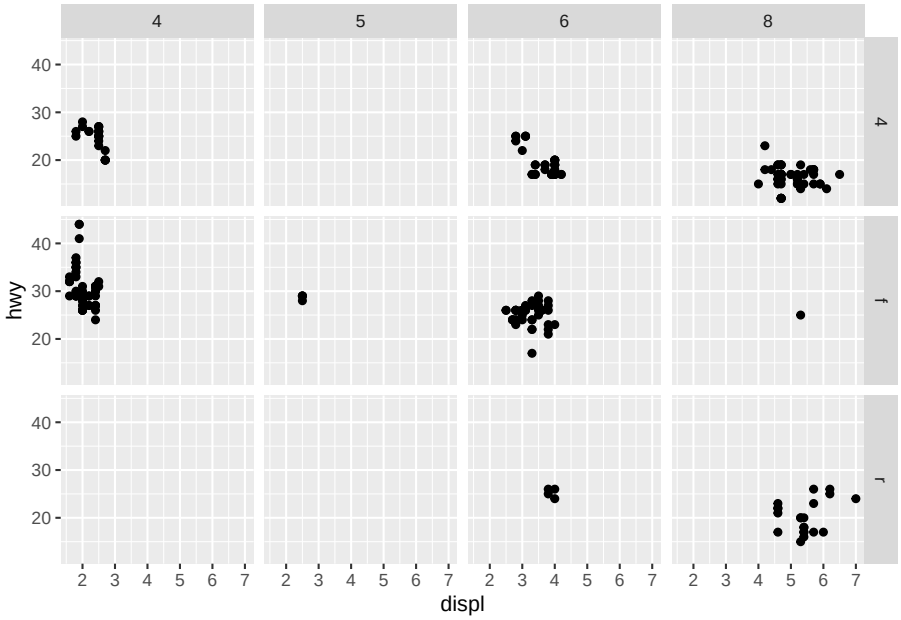



```
p + geom_point() +
  facet_wrap(~ class, nrow = 2)
```



Kui me tahame kahe muutuja kõigi kombinatsioonide vastu paneele, siis kasuta `facet_grid()` funktsiooni.

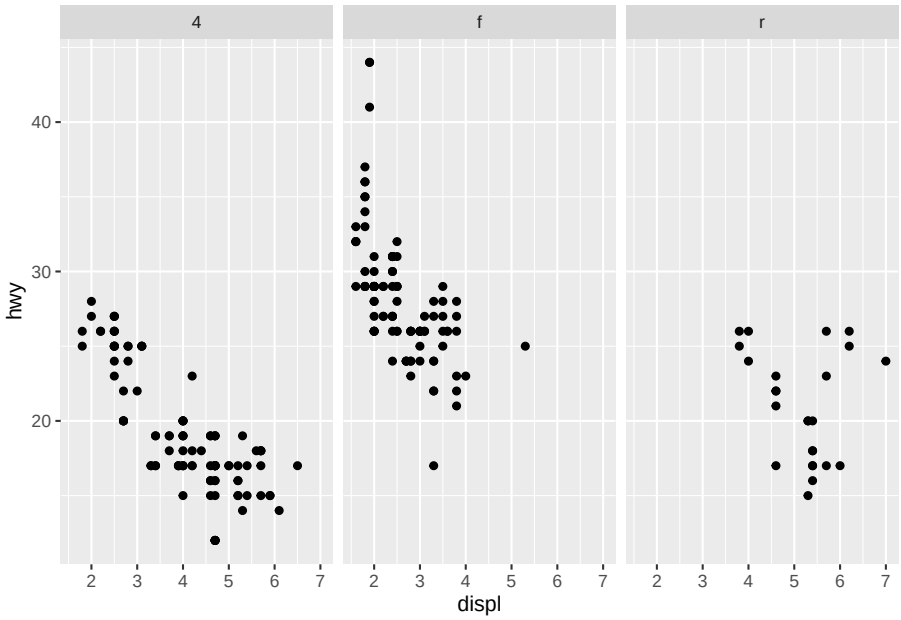
```
p + geom_point() +
  facet_grid(drv ~ cyl)
```



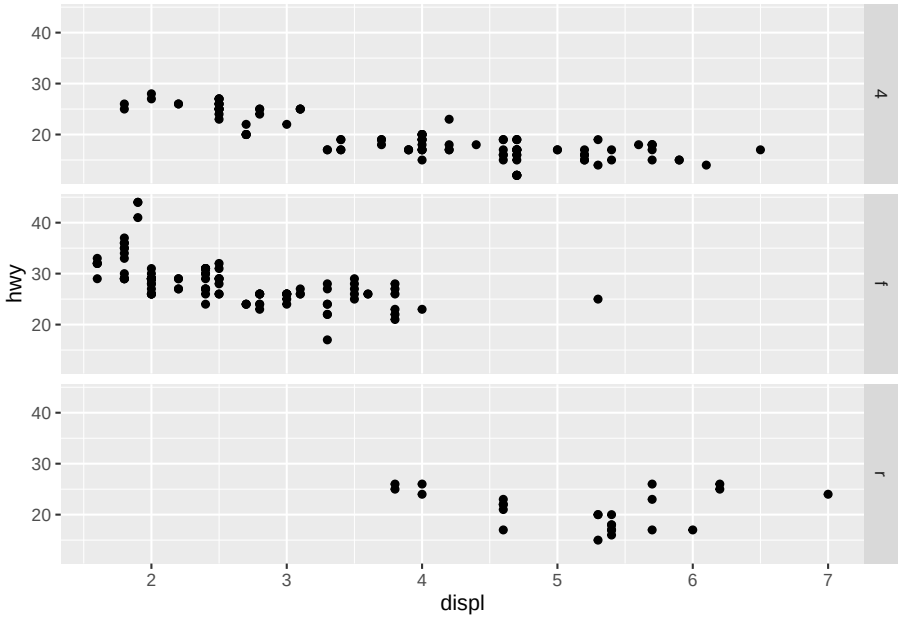
- "drv" – drive - 4(-wheel), f(orward), r(ear).
- "cyl" – cylinders - 4, 5, 6, or 8.

Kasutades punkti `.` on võimalik asetada kõik alamgraafikud kõrvuti `(. ~ var)` või üksteise peale `(var ~ .)`.

```
p + geom_point() +
  facet_grid(. ~ drv)
```




```
p + geom_point() +
  facet_grid(drv ~ .)
```



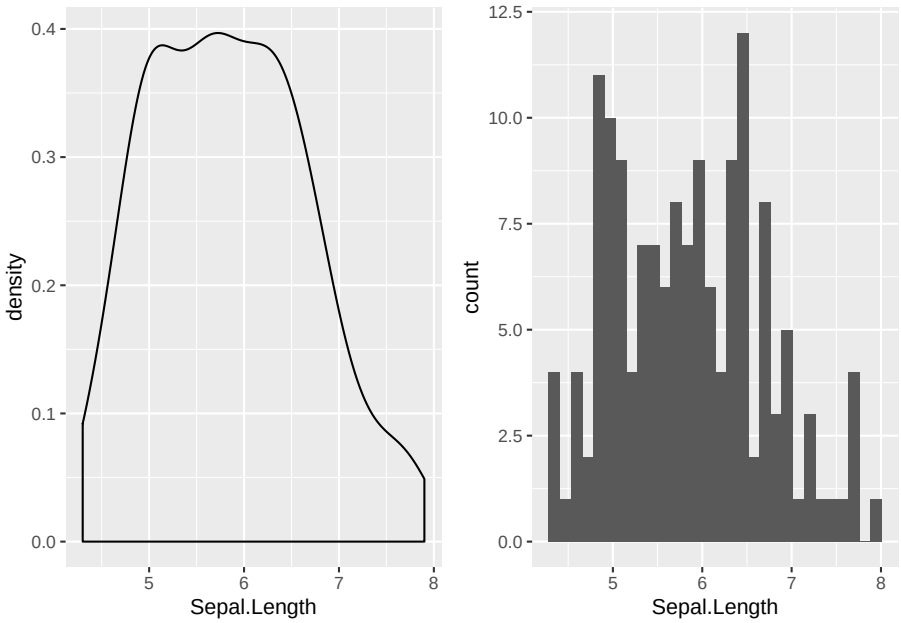
0.11.4 Mitu graafikut paneelidena ühel joonisel

Kõigepealt tooda komponentgraafikud ggplot() abil ja tee nendest graafilised objektid. Näiteks nii:

```
library(tidyverse)
i1 <- ggplot(data= iris, aes(x=Sepal.Length)) + geom_histogram()
i2 <- ggplot(data= iris, aes(x=Sepal.Length)) + geom_density()
```

Seejäral, kasuta funktsioon `gridExtra::grid.arrange()` et graafikud kõrvuti panna

```
library(gridExtra)
grid.arrange(i2, i1, nrow = 1) # ncol = 2 also works
```



0.11.5 Teljed

0.11.5.1 Telgede ulatus

Telgede ulatust saab määrata kolmel erineval viisil

1. filtreeri andmeid, mida plotid
2. pane x- ja y-teljele piirangud `xlim()`, `ylim()`
3. kasuta `coord_cartesian()` ja xlim, ylim parameetritena selle sees: `coord_cartesian(xlim = c(5, 7), ylim = c(10, 30))`

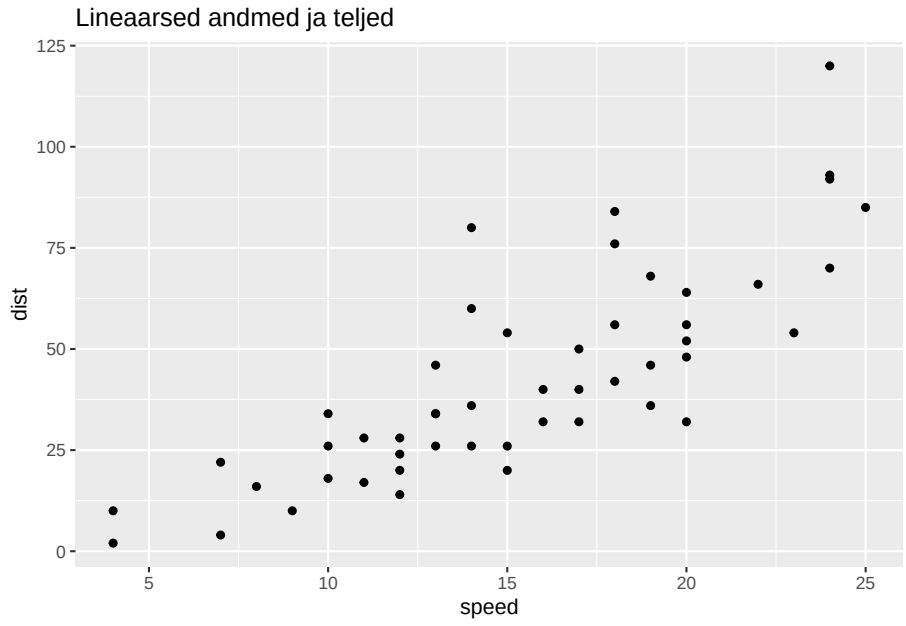
Telgede ulatust saab muuta ka x- ja y-teljele eraldi:

- `scale_x_continuous(limits = range(mpg$displ))`
- `scale_y_continuous(limits = range(mpg$hwy))`

0.11.5.2 Log skaalas teljed

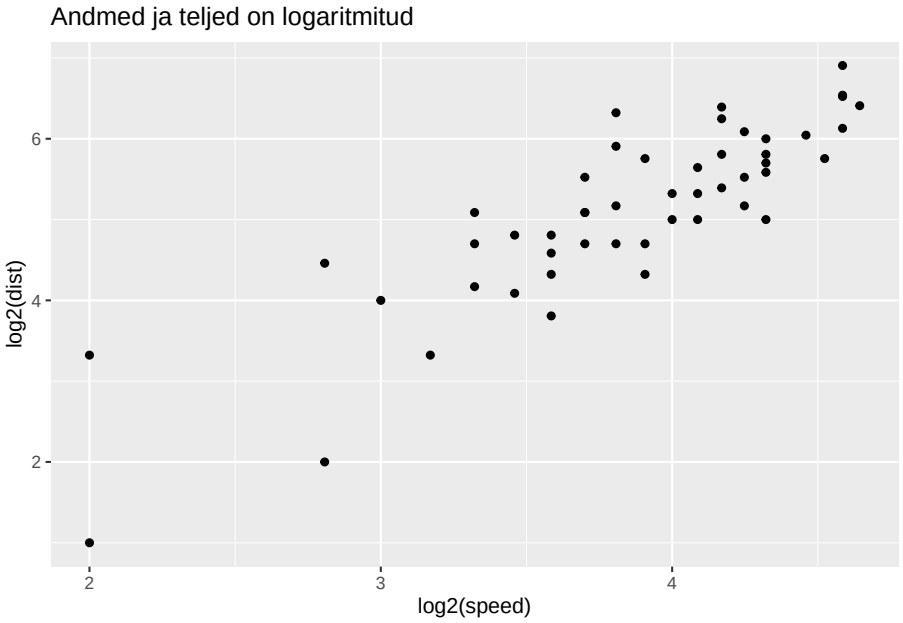
1. Lineaarsed andmed lineaarsetel telgedel.

```
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point() +
  ggtitle("Lineaarsed andmed ja teljed")
```



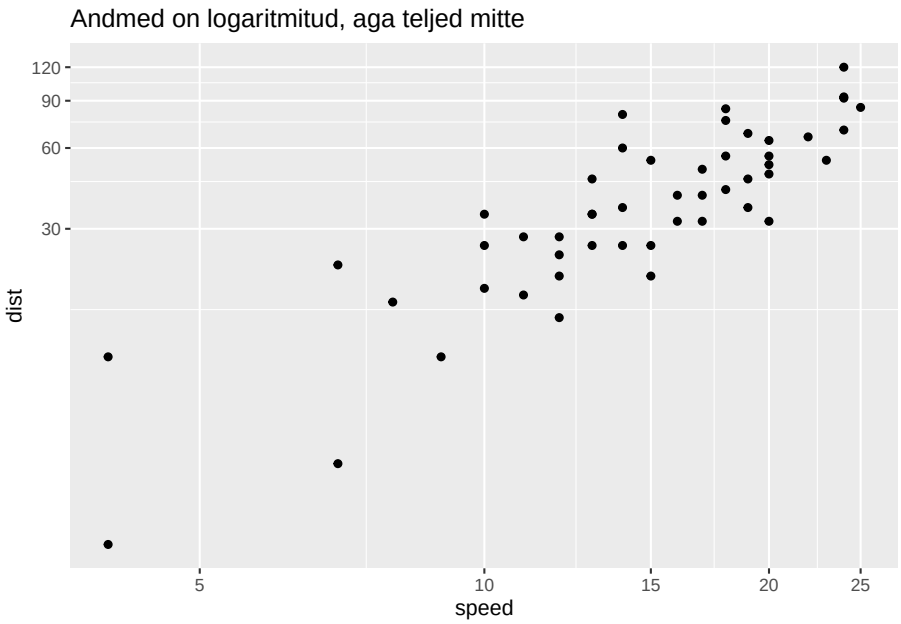
2. Logaritmi andmed `aes()`-s.

```
ggplot(cars, aes(x = log2(speed), y = log2(dist))) +
  geom_point() +
  ggtitle("Andmed ja teljed on logaritmitud")
```



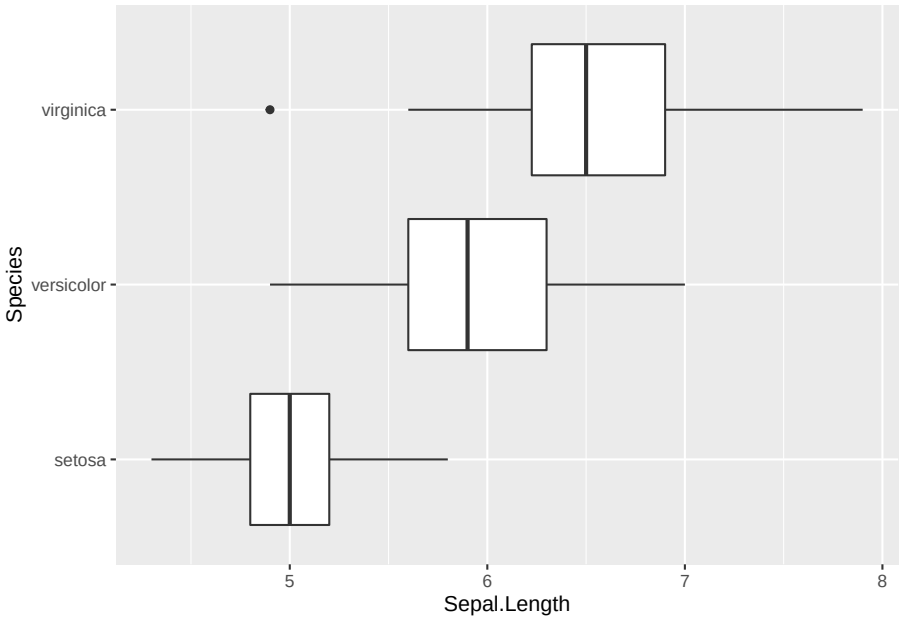
3. Andmed on logaritmitud, aga teljed mitte.

```
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point() +
  coord_trans(x = "log2", y = "log2") +
  ggtitle("Andmed on logaritmitud, aga teljed mitte")
```



0.11.5.3 Pöörame graafikut 90 kraadi

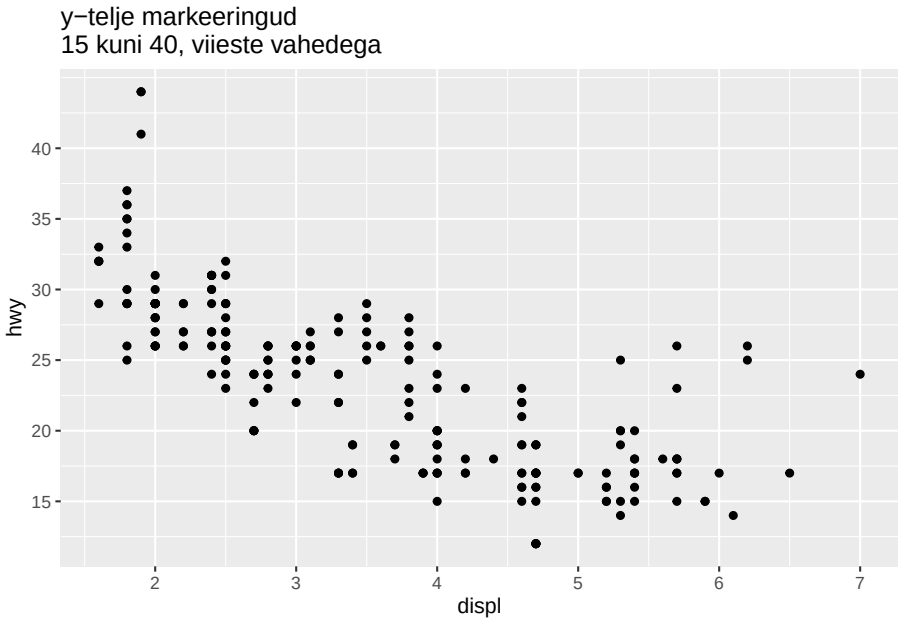
```
ggplot(iris, mapping = aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  coord_flip()
```



0.11.5.4 Muudame telgede markeeringuid

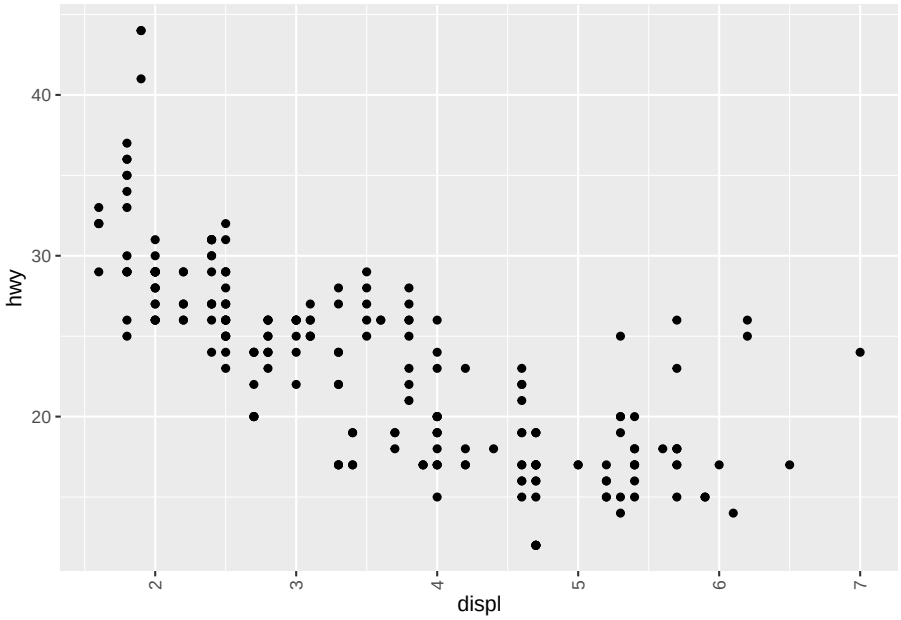
Muudame y-telje markeeringut:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  scale_y_continuous(breaks = seq(15, 40, by = 5)) +
  ggtitle("y-telje markeeringud\n15 kuni 40, viieste vahedega")
```



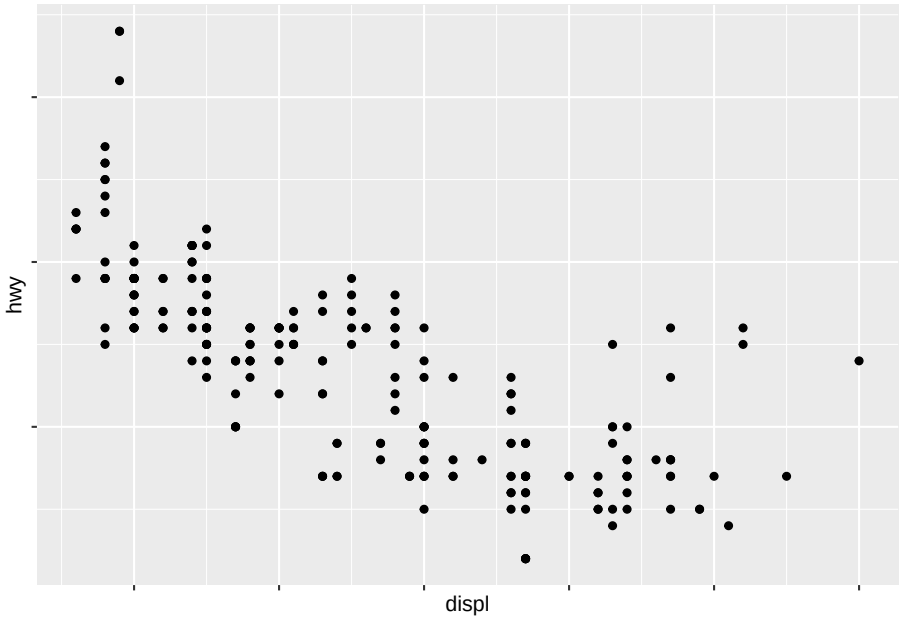
Muudame x-telje markeeringute nurka muutes `theme()` funktsiooni argumenti "axis.text.x":

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



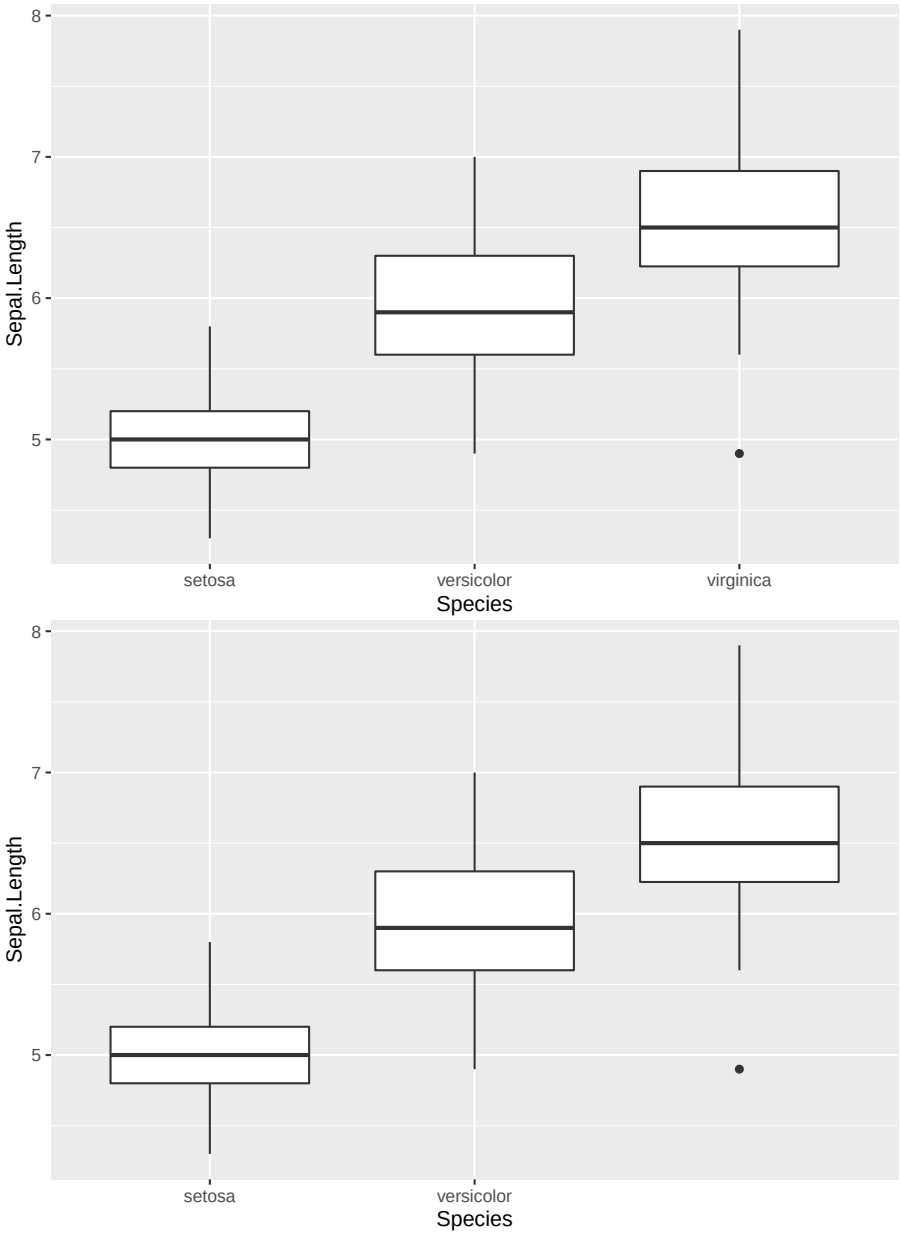
Eemaldame telgede markeeringud, ka läbi `theme()` funktsiooni:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  theme(axis.text = element_blank())
```



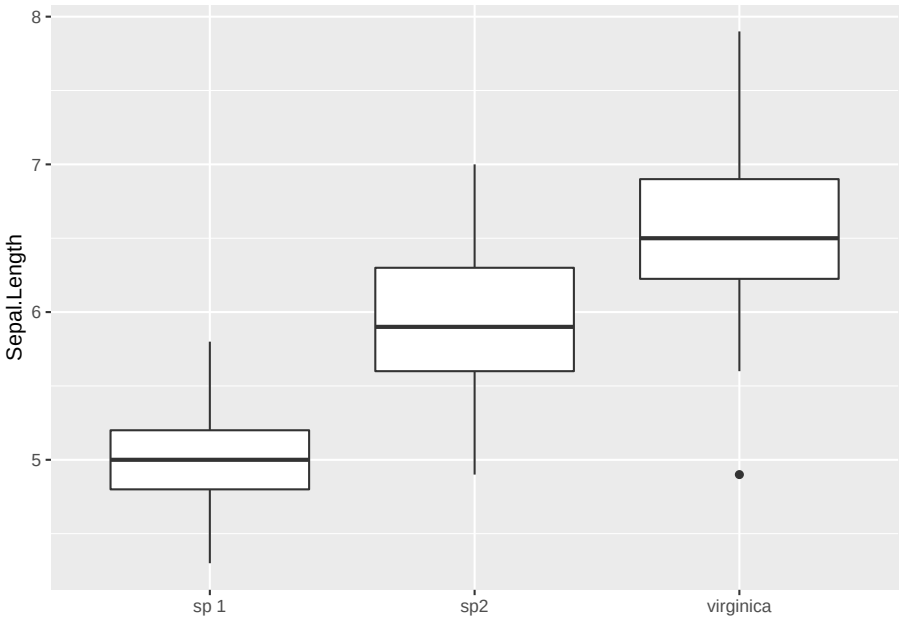
Muudame teljemarkeeringute järjekorda

```
p <- ggplot(iris, aes(Species, Sepal.Length)) + geom_boxplot()
p
p + scale_x_discrete(breaks=c("versicolor", "setosa"))
```



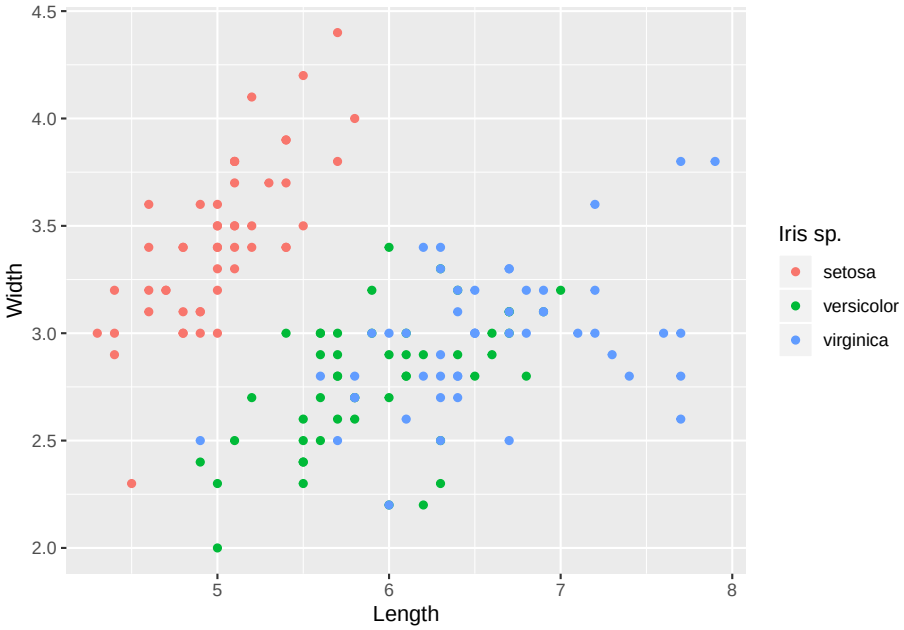
Muuda teljemarkeeringuid ja kustuta telje nimi.

```
p + scale_x_discrete(labels=c("setosa" = "sp 1", "versicolor" = "sp2"), name
```



0.11.5.5 telgede ja legendi nimed

```
p <- ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point()
p + labs(
     x = "Length",
     y = "Width",
     color = "Iris sp."
     )
```

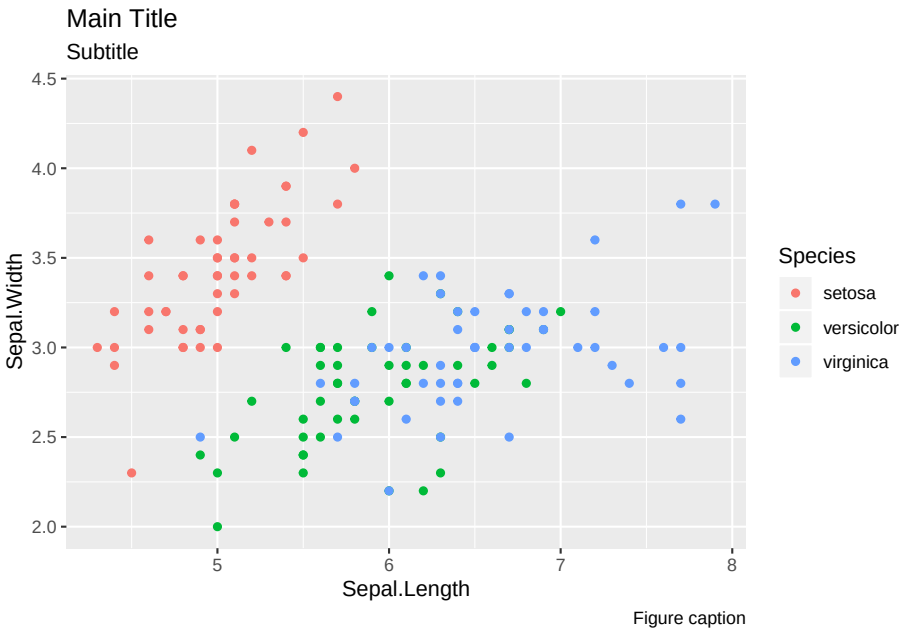


Eemaldame telgede nimed:

```
p + theme(axis.title = element_blank())
```

0.11.6 Graafiku pealkiri, alapeakiri ja allkiri

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point() +
  labs(
     title = "Main Title",
     subtitle = "Subtitle",
     caption = "Figure caption"
      )
```



`ggtitle()` annab graafikule pealkirja

0.11.6.1 Täpitähed jms graafikutel

R-s on sümbolid tavapäraselt UTF-8 kodeeringus. Mitte-inglise tähestikku kuuluvaid sümboleid saab lisada, andes ette nende kodeeringu, millele eelneb backslash. Täisnimekirja UTF-8 kodeeringutest leiab https://www.fileformat.info/info/charset/UTF-8/list.htm

0.11.7 Graafiku legend

Legend erinevalt graafikust endast ei ole pool-läbipaistev.

```
norm <- tibble(x = rnorm(1000), y = rnorm(1000))
norm$z <- cut(norm$x, 3, labels = c( "a" ,  "b" ,  "c" )) #creates a new col

ggplot(norm, aes(x, y)) +
```

```
  geom_point(aes(colour = z), alpha = 0.3) +
  guides(colour = guide_legend(override.aes = list(alpha = 1)))
```

legend graafiku sisse

```
df <- data.frame(x = 1:3, y = 1:3, z = c( "a" ,  "b" ,  "c" ))
base <- ggplot(df, aes(x, y)) +
  geom_point(aes(colour = z), size = 3) +
  xlab(NULL) +
  ylab(NULL)

base + theme(legend.position = c(0, 1), legend.justification = c(0, 1))
base + theme(legend.position = c(0.5, 0.5), legend.justification = c(0.5, 0.
base + theme(legend.position = c(1, 0), legend.justification = c(1, 0))
```

legendi asukoht graafiku ümber:

```
base + theme(legend.position = "left")
base + theme(legend.position = "top")
base + theme(legend.position = "bottom")
base + theme(legend.position = "right") # the default
```

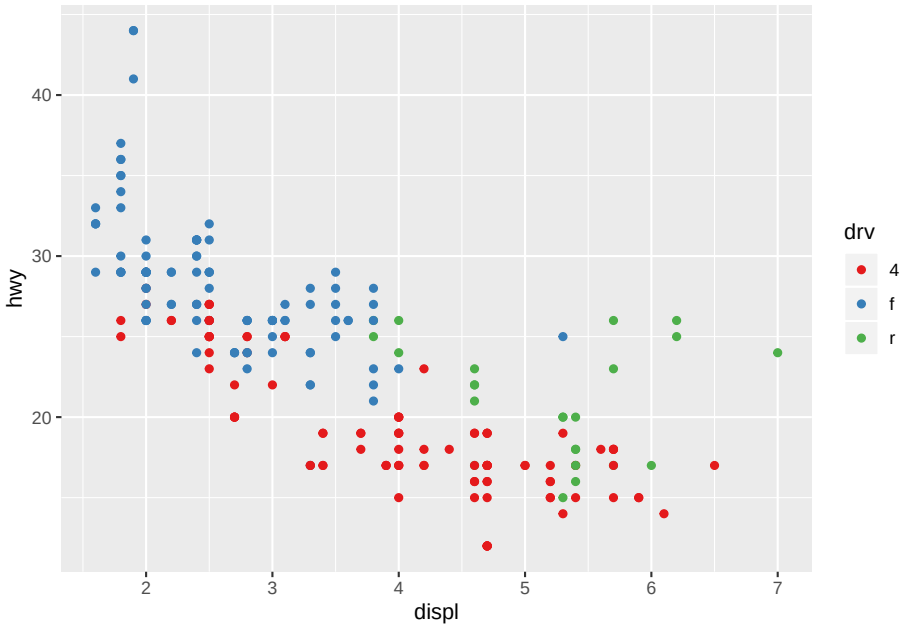
eemalda legend

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class))+
  theme(legend.position = "none")
```

0.11.8 Värviskaalad

ColorBreweri skaala "Set1" on hästi nähtav värvipimedatele. colour_brewer skaalad loodi diskreetsetele muutujatele, aga nad näevad sageli head välja ka pidevate muutujate korral.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv)) +
  scale_colour_brewer(palette = "Set1")
```



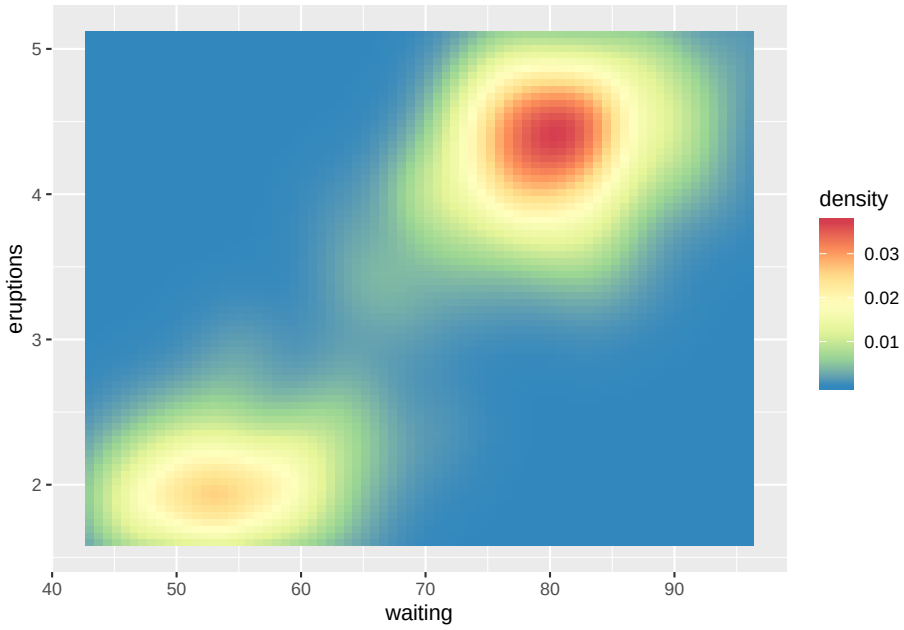
0.11.8.1 Värviskaalad pidevatele muutujatele

Pidevatele muutujatele töötab scale_colour_gradient() or scale_fill_gradient(). scale_colour_gradient2() võimaldab eristada näiteks positiivseid ja negatiivseid väärtusi erinevate värviskaaladega.

```
df <- data.frame(x = 1, y = 1:5, z = c(1, 3, 2, NA, 5))
p <- ggplot(df, aes(x, y)) + geom_tile(aes(fill = z), size = 5)
p
# Make missing colours invisible
p + scale_fill_gradient(na.value = NA)
# Customise on a black and white scale
p + scale_fill_gradient(low =  "black" , high =  "white" , na.value =  "red"

#gradient between n colours
p+scale_color_gradientn(colours = rainbow(5))
```

```
# Use distiller variant with continous data
ggplot(faithfuld) +
```

```
  geom_tile(aes(waiting, eruptions, fill = density)) +
  scale_fill_distiller(palette = "Spectral")
```



0.11.8.2 Värviskaalad faktormuutujatele

Tavaline värviskaala on scale_colour_hue() ja scale_fill_hue(), mis valivad värve HCL värvirattast. Töötavad hästi kuni u 8 värvini.

```
ToothGrowth <- ToothGrowth
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
mtcars <- mtcars
mtcars$cyl <- as.factor(mtcars$cyl)

#bp for discrete color scales
bp<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()
bp
#sp for continuous scales
sp<-ggplot(mtcars, aes(x=wt, y=mpg, color=cyl)) + geom_point()
```

```
sp

#You can control the default chroma and luminance, and the range
#of hues, with the h, c and l arguments
bp + scale_fill_hue(l=40, c=35, h = c(180, 300)) #boxplot
sp + scale_color_hue(l=40, c=35) #scatterplot
```

Halli varjunditega töötab scale_fill_grey().

```
bp + scale_fill_grey(start = 0.5, end = 1)
```

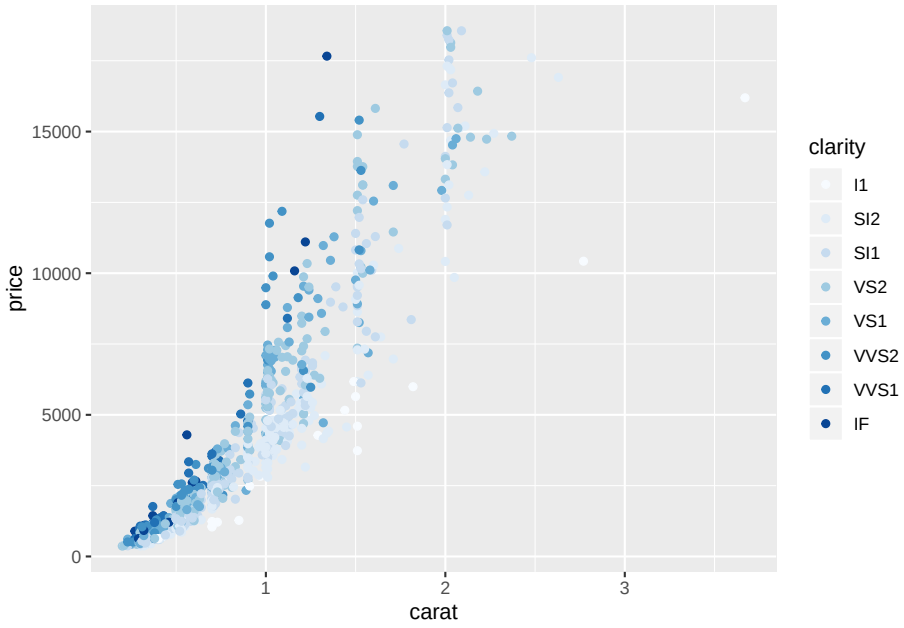
Järgmine võimalus on käsitsi värve sättida

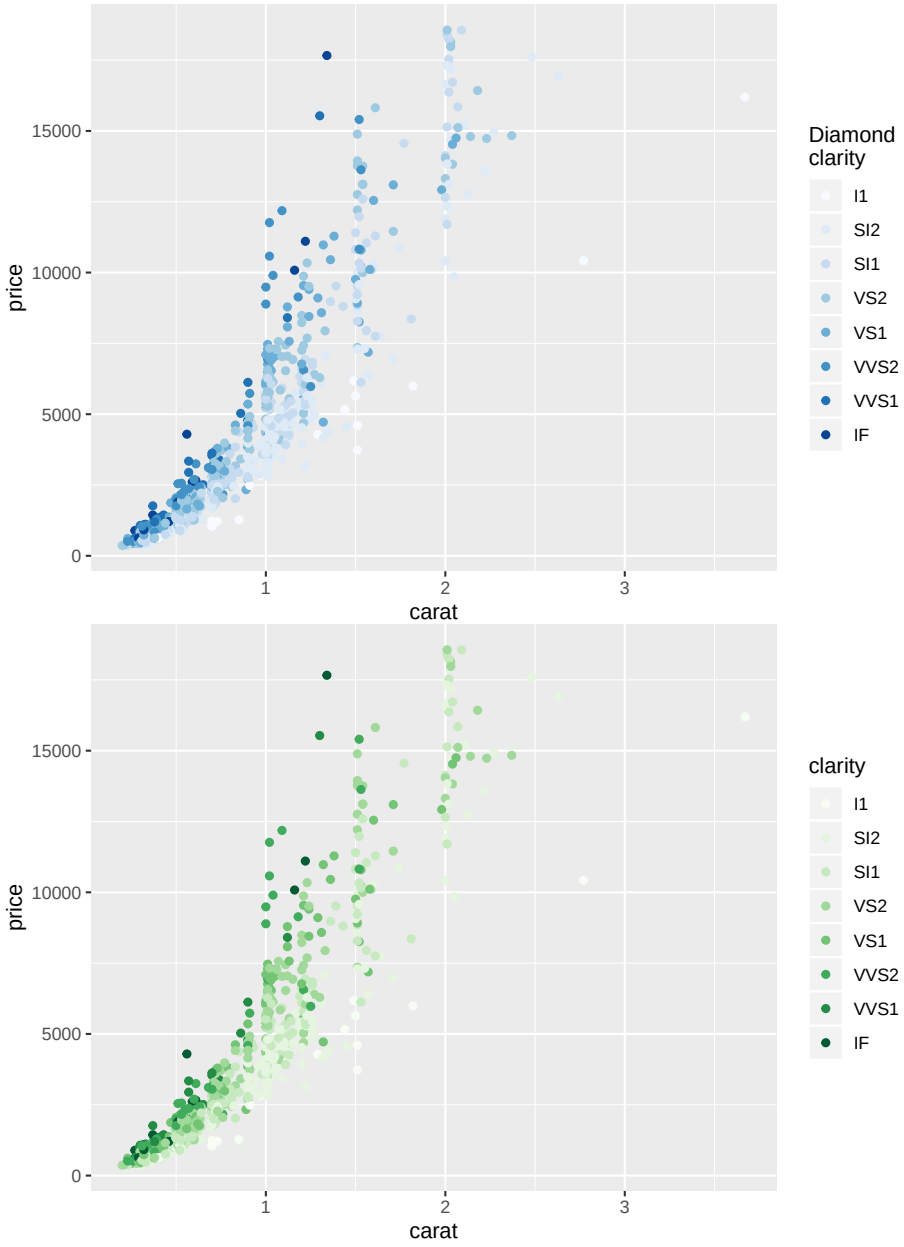
```
#bp for discrete color scales
bp<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()
bp
#sp for continuous scales
sp<-ggplot(mtcars, aes(x=wt, y=mpg, color=cyl)) + geom_point()
sp
bp + scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))
sp + scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```

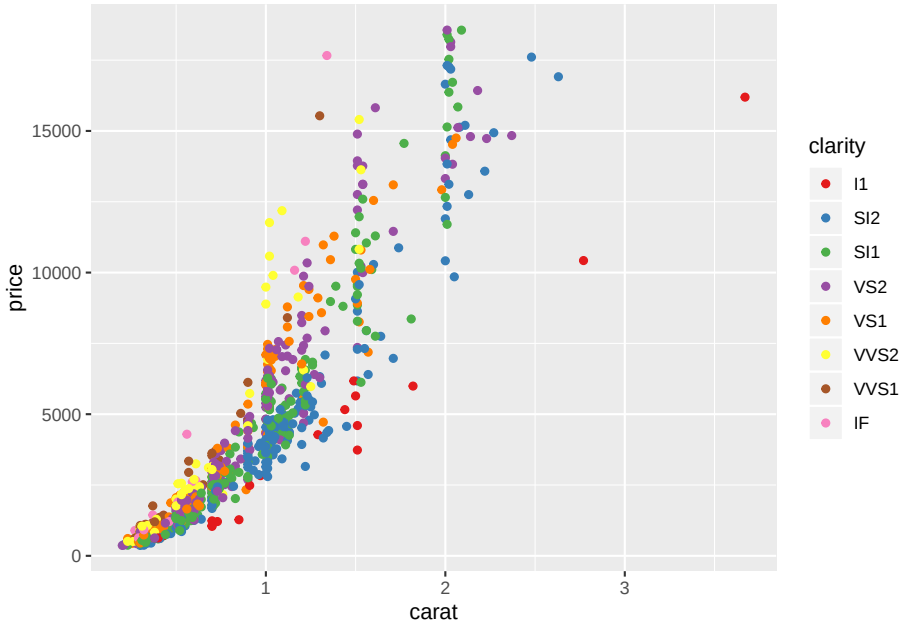
Colour_brewer-i skaalad on loodud faktormuutujaid silmas pidades.

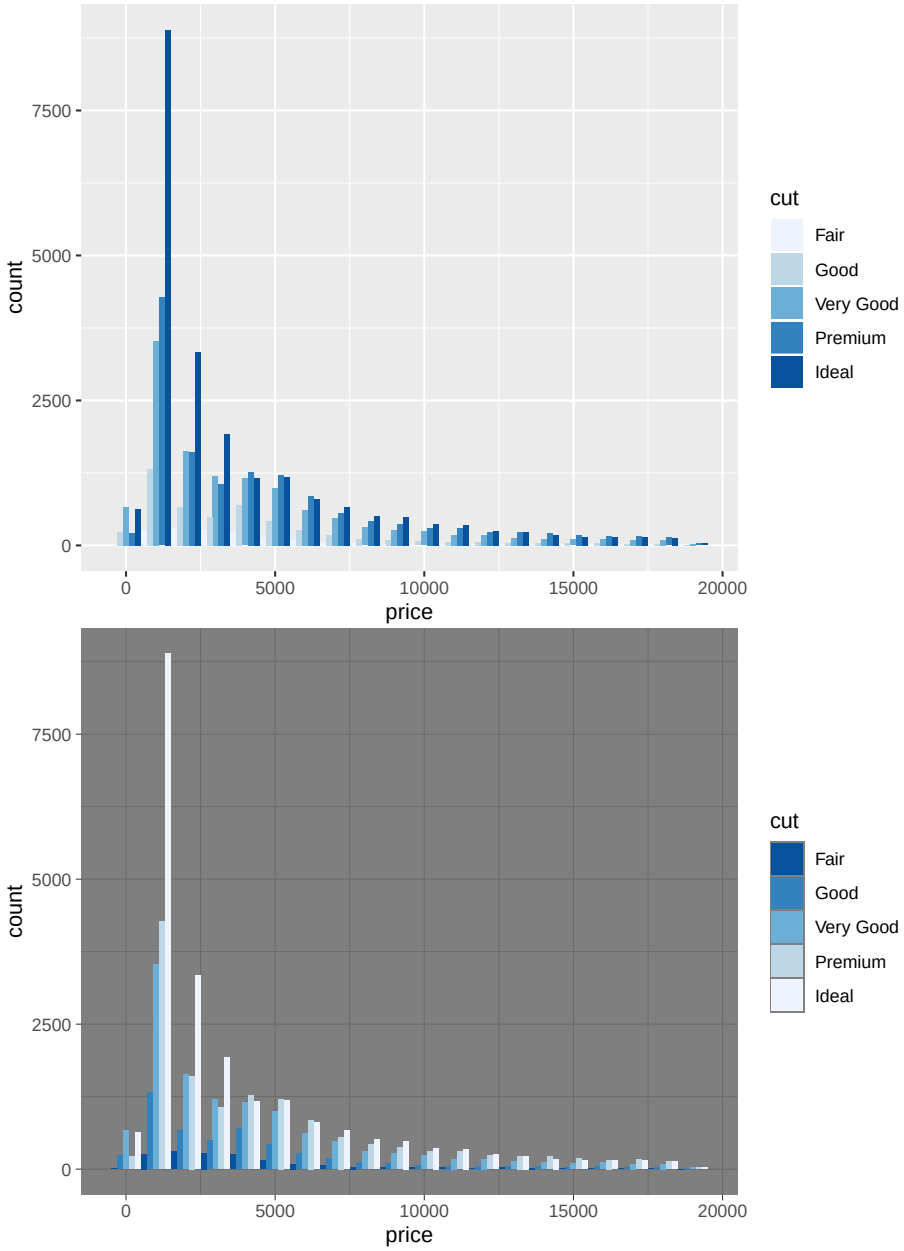
```
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
d <- ggplot(dsamp, aes(carat, price)) +
  geom_point(aes(colour = clarity))
d + scale_colour_brewer()

# Change scale label
d + scale_colour_brewer("Diamond\nclarity")

# Select brewer palette to use, see ?scales::brewer_pal for more details
d + scale_colour_brewer(palette = "Greens")
d + scale_colour_brewer(palette = "Set1")

# scale_fill_brewer works just the same as
```

```
# scale_colour_brewer but for fill colours
p <- ggplot(diamonds, aes(x = price, fill = cut)) +
  geom_histogram(position = "dodge", binwidth = 1000)
p + scale_fill_brewer()
# the order of colour can be reversed
# the brewer scales look better on a darker background
p + scale_fill_brewer(direction = -1) + theme_dark()
```




price
15000
10000
5000
0
1
2
3
carat
Diamond clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF
price
15000
10000
5000
0
1
2
3
carat
clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF


price
15000
10000
5000
0
1
2
3
carat
clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF



Väga lahedad värviskaalad, mis eriti hästi sobivad diskreetsetele muutujatele, on wesanderson paketis. Enamus skaalasid on küll ainult 3-5 värviga. Sealt saab siiski ekstrapoleerida rohkematele värvidele (?wes_palette; ?wes_palettes).

```
#install.packages("wesanderson")
#library(wesanderson)

#bp for discrete color scales
bp<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()
bp

#wes_palette(name, n, type = c("discrete", "continuous"))
#n - the nr of colors desired, type - do you want a continious scalle?
bp+scale_fill_manual(values=wes_palette(n=3, name="GrandBudapest"))

wes_palette("Royal1")
wes_palette("GrandBudapest")
wes_palette("Cavalcanti")
wes_palette("BottleRocket")
wes_palette("Darjeeling")

wes_palettes #gives the complete list of palettes
```

Argument **breaks** kontrollib legendi. Sama kehtib ka teiste scale_xx() funktsioonide kohta.

```
bp <- ToothGrowth %>%
  ggplot(aes(x = dose, y = len, fill = dose)) +
  geom_boxplot()
bp
# Box plot
bp + scale_fill_manual(breaks = c("2", "1", "0.5"),
                       values = c("red", "blue", "green"))

# color palettes
bp + scale_fill_brewer(palette = "Dark2")
#sp + scale_color_brewer(palette="Dark2")

# use graysacle
```

```
#Change the gray value at the low and the high ends of the palette :
bp + scale_fill_grey(start = 0.8, end = 0.2) + theme_classic()
```

The ColorBrewer scales are documented online at http://colorbrewer2.org/ and made available in R via the RColorBrewer package. When you have a predefined mapping between values and colours, use scale_colour_manual().

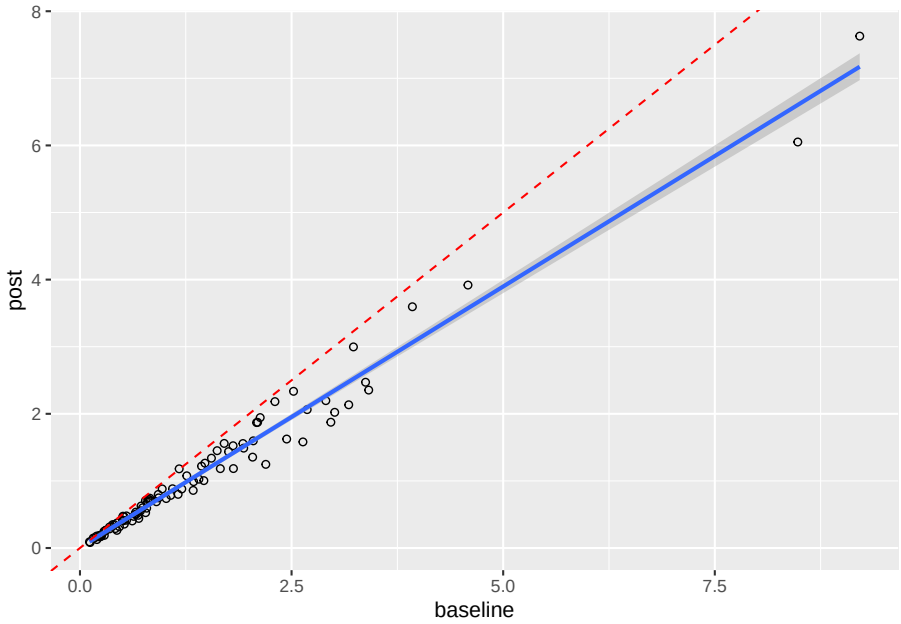
`scale_colour_manual(values = c(factor_level_1 = "red", factor_level_2 = "blue")`

`scale_colour_viridis()` provided by the viridis package is a continuous analog of the categorical ColorBrewer scales.

0.11.9 A complex ggplot

Let's pretend that we are measuring the same quantity by immunoassay at baseline and after 1 year of storage at -80 degrees. We'll add some heteroscedastic error and create some apparent degradation of about 20%:

```
set.seed(10)
baseline <- rlnorm(100, 0, 1)
post <- 0.8 * baseline + rnorm(100, 0, 0.10 * baseline)
my_data <- tibble(baseline, post)
my_data %>%
  ggplot(aes(baseline, post)) +
  geom_point(shape = 1) + # Use hollow circles
  geom_smooth(method = "lm") + # Add linear regression line
  geom_abline(slope = 1, intercept = 0, linetype = 2, colour = "red")
```



Now we will prepare the difference data:

```
diff <- (post - baseline)
diffp <- (post - baseline) / baseline * 100
sd.diff <- sd(diff)
sd.diffp <- sd(diffp)
my.data <- data.frame(baseline, post, diff, diffp)
```

In standard Bland Altman plots, one plots the difference between methods against the average of the methods, but in this case, the x-axis should be the baseline result, because that is the closest thing we have to the truth.

```
library(ggExtra)
diffplot <- ggplot(my.data, aes(baseline, diff)) +
  geom_point(size=2, colour = rgb(0,0,0, alpha = 0.5)) +
  theme_bw() +
  #when the +/- 2SD lines will fall outside the default plot limits
  #they need to be pre-stated explicitly to make the histogram line up prop
  ylim(mean(my.data$diff) - 3*sd.diff, mean(my.data$diff) + 3*sd.diff) +
  geom_hline(yintercept = 0, linetype = 3) +
```

```
  geom_hline(yintercept = mean(my.data$diff)) +
  geom_hline(yintercept = mean(my.data$diff) + 2*sd.diff, linetype = 2) +
  geom_hline(yintercept = mean(my.data$diff) - 2*sd.diff, linetype = 2) +
  ylab("Difference pre and post Storage (mg/L)") +
  xlab("Baseline Concentration (mg/L)")

#And now for the magic - we'll use 25 bins
ggMarginal(diffplot, type = "histogram", bins = 25)
```

0.12 Kuraditosin graafikut, mida sa peaksid enne surma joonistama

Andmete plottimisel otsib analüütik tasakaalu informatsioonikao ja trendide/mustrite/kovarieeruvuste nähtavaks tegemise vahel. Idee on siin, et teie andmed võivad sisaldada a) juhuslikku müra, b) trende/mustreid, mis teile suurt huvi ei paku ja c) teid huvitavaid varjatud mustreid. Kui andmeid on palju ja need on mürarikkad ja kui igavad trendid/mustrid varjavad huvitavaid trende/mustreid, siis aitab vahest andmete graafiline redutseerimine üldisemale kujule ja nende modelleerimine. Kui andmeid ei ole väga palju, siis tasuks siiski vältida infot kaotavaid graafikuid ning joonistada algsed või ümber arvutatud andmepunktid. Järgnevalt esitame valiku graafikutüüpe erinevat tüüpi andmetele.

0.12.1 Cleveland plot

x- pidev muutuja; y - faktormuutuja

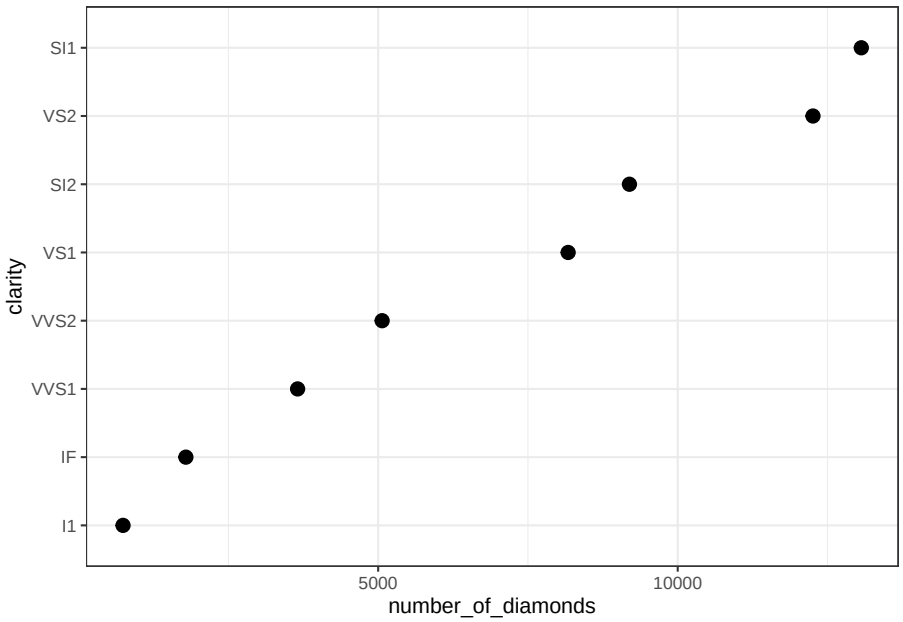
Seda plotti kasuta a) kui iga muutja kohta on üks andmepunkt või b) kui soovid avaldada keskmise koos usalduspiiridega.

> Sageli lahendatakse sarnased ülesanded tulpdiagrammidega, mis ei ole aga üldiselt hea mõte, sest tulpdiagrammid juhivad asjatult tähelepanu tulpadele endile, pigem kui nende otstele,

mis tegelikult andmete keskmist kajastavad. Kuna inimese aju tahab võrrelda tulpade kõrgusi suhtelistes, mitte absoluutsetes ühikutes (kui tulp A on 30% kõrgem kui tulp B, siis me näeme efekti suurust, mis on u 1/3), peavad tulbad algama mingilt oodatavalt baastasemelt (tavaliselt nullist). See aga võib muuta raskeks huvitavate efektide märkamise, kui need on protsentuaalselt väikesed. Näiteks 5%-ne CO2 taseme tõus atmosfääris on teaduslikult väga oluline, aga tulpdiagrammi korrektselt kasutades tuleb vaevu graafikult välja.

Kõigepealt plotime, mitu korda esinevad *diamonds* tabelis erinevate faktormuutuja *clarity* tasemetega teemandid (*clarity* igale tasemele vastab üks number – selle *clarity*-ga teemantite arv).

```
dd <- diamonds %>%
  group_by(clarity) %>%
  summarise(number_of_diamonds = n())

dd %>%
  ggplot(aes(x = number_of_diamonds,
             y = reorder(clarity, number_of_diamonds))) +
  geom_point(size = 3) +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_line(colour = "grey60", linetype = "das
  labs(y = "clarity") +
  theme_bw()
```



Graafiku loetavuse huvides on mõistlik Y- telg sorteerida väärtuste järgi.

Järgmisel joonisel on näha *iris* tabeli *Sepal length* veeru keskmised koos 50% ja 95% usaldusintervallidega. Usaldusintervallid annavad hinnangu meie ebakindlusele keskväärtuse (mitte näiteks algandmete) paiknemise kohta, arvestades meie valimi suurust ja sellest tulenevat valimiviga. 50% CI tähendab, et me oleme täpselt sama vähe üllatunud leides tõese väärtuse väljaspoolt intervalli, kui leides selle intervalli seest. 95% CI tähendab, et me oleme mõõdukalt veendunud, et tõene väärtus asub intervallis (aga me arvestame siiski, et ühel juhul 20-st ta ei tee seda). **NB! Need tõlgendused eeldavad, et meie andmetes esinev juhuslik varieeruvus on palju suurem kui seal leiduv suunatud varieeruvus (ehk bias).**

Kasutame Publish::ci.mean(), et arvutada usaldusintervallid (antud juhul 10% CI)

```
library(Publish) #siit ci.mean()
a <- rnorm(10)
```

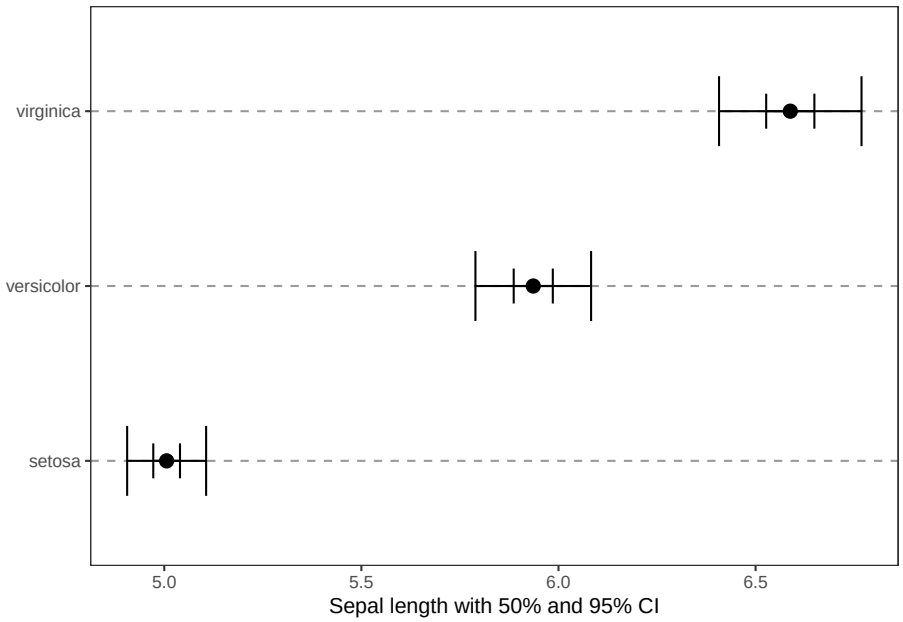
```
a1 <- ci.mean(a, alpha = 0.9)
str(a1)
#> List of 6
#>  $ mean     : num 0.151
#>  $ se       : num 0.332
#>  $ lower    : num 0.108
#>  $ upper    : num 0.194
#>  $ level    : num 0.9
#>  $ statistic: chr "arithmetic"
#>  - attr(*, "class")= chr [1:2] "ci" "list"
```

kisume listi elemendi nimega lower (usaldusintervalli alumine piir) välja 3-l alternatiivsel viisil

```
a1$lower
#> [1] 0.108
a1[[3]]
#> [1] 0.108
a1 %>% pluck("lower")
#> [1] 0.108
```

```
iris1 <- iris %>%
  group_by(Species) %>%
  summarise(Mean = mean(Sepal.Length),
            CI_low_0.5 = ci.mean(Sepal.Length, alpha=0.5) %>% pluck("lower"),
            CI_high_0.5 = ci.mean(Sepal.Length, alpha=0.5) %>% pluck("upper"),
            CI_low_0.95 = ci.mean(Sepal.Length) %>% pluck("lower"),
            CI_high_0.95 = ci.mean(Sepal.Length) %>% pluck("upper")
            )
#pluck() takes a named element out of a list
#ci.mean() output is a list of 6 elements

ggplot(data = iris1, aes(x = Mean, y = Species)) +
  geom_point(size = 3) +
  geom_errorbarh(aes(xmin = CI_low_0.5,
                     xmax = CI_high_0.5),
                 height = 0.2) +
  geom_errorbarh(aes(xmin = CI_low_0.95,
```

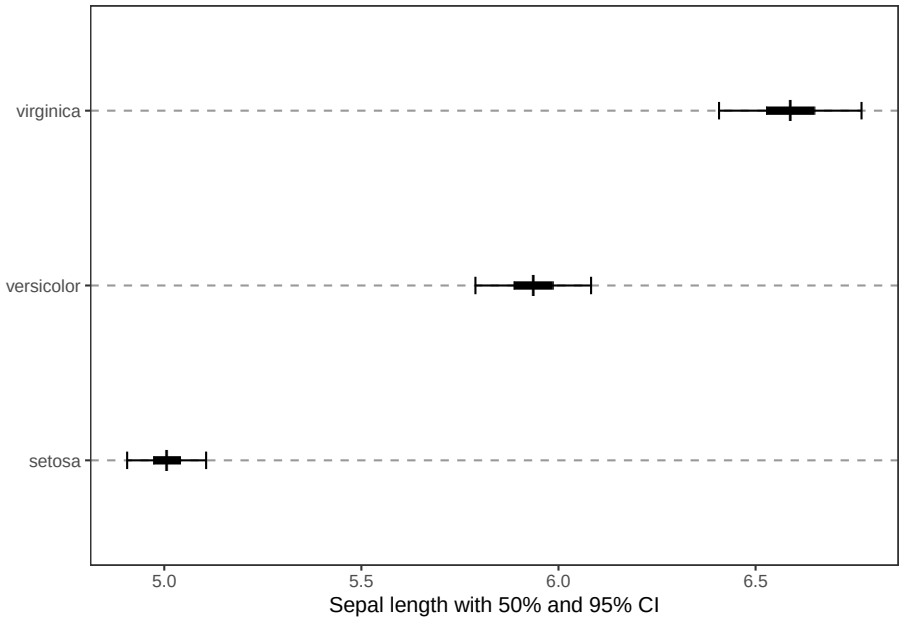
```
                    xmax = CI_high_0.95),
                height = 0.4) +
  theme_bw()+
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_line(colour = "grey60", linetype = "das
  labs(x = "Sepal length with 50% and 95% CI",
       y = NULL)
```



Alternatiivne graafiku kuju (muudetud on ainult geom_point size ja shape parameetreid):

```
ggplot(data = iris1, aes(x = Mean, y = Species)) +
  geom_point(size = 5, shape = 108) +
  geom_errorbarh(aes(xmin = CI_low_0.5,
                     xmax = CI_high_0.5),
                 height = 0,
                 size = 2) +
  geom_errorbarh(aes(xmin = CI_low_0.95,
                     xmax = CI_high_0.95),
```

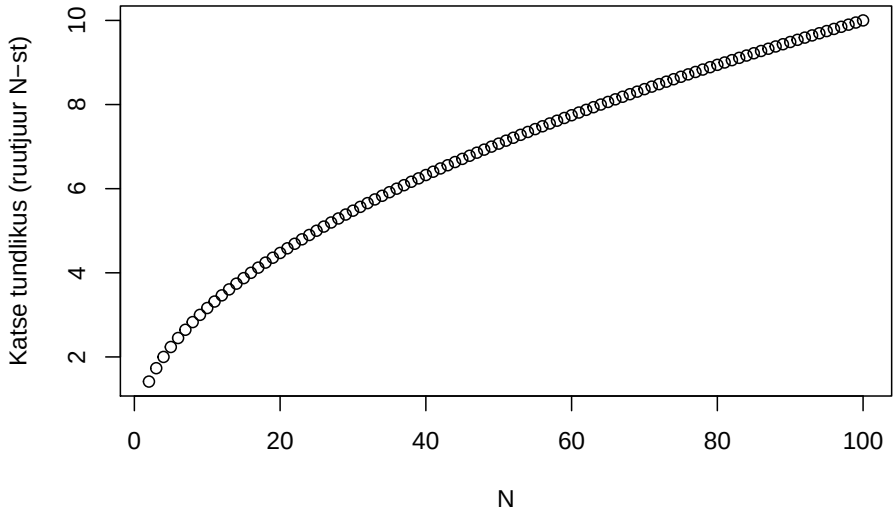
```
                height = 0.1) +
theme_bw()+
theme(panel.grid.major.x = element_blank(),
      panel.grid.minor.x = element_blank(),
      panel.grid.major.y = element_line(colour = "grey60", linetype = "das
labs(x = "Sepal length with 50% and 95% CI",
     y = NULL)
```



CI-d saab arvutada ka käsitsi. Kui valimi suurus on piisav ja normaaljaotus pole meie andmetest liiga kaugel, siis saame CI arvutamiseks kasutada järgmisi heuristikuid:

CI_percentage	nr_of_SEMs
50.0	0.675
75.0	1.150
90.0	1.645
95.0	1.960
97.0	2.170
99.0	2.575
99.9	3.291

SEM on standardviga, mille arvutame jagades valimi standardhälbe ruutjuurega valimi suurusest N. Kuna CI sõltub SEM-ist, sõltub see muidugi ka N-st, aga mitte lineaarselt, vaid üle ruutjuure. See tähendab, et uuringu usaldusväärsuse tõstmine, tõstes N-i kipub olema progressiivselt kulukas protsess. Analoogiana võib siin tuua sportliku vormi tõstmine, kus trennis käimisega alustades on suhteliselt lihtne tõsta oma sooritust näiteks 20% võrra, aga peale aastast usinat rassimist tuleb juba teha väga tõsine pingutus, et saavutada veel 1% tõusu.



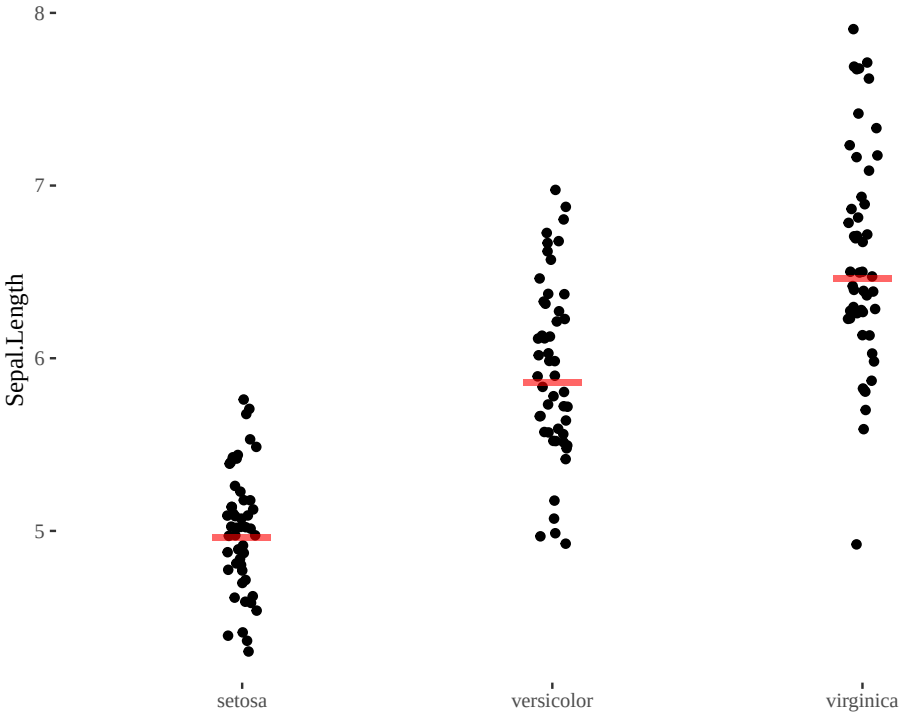
Nagu näha jooniselt, on meil tegu progresiivselt kallineva ülesandega: mida rohkem tahame usalduspiire kitsamaks muuta **suhteliselt** (mis on sama, mis öelda, et me tahame tõsta katse tundlikust), seda suurema tõusu peame tagama kogutud andmete hulgas **absoluutarvuna**.

0.12.2 Andmepunktid mediaani või aritmeetilise keskmisega

x - faktormuutuja; y - pidev muutuja

Kui N $<$ 20, siis on see tavaliselt parim valik sest säilitab maksimaalselt andmetes leiduvat informatsiooni.

```
ggplot(iris, aes(x=Species, y=Sepal.Length)) +
  geom_jitter(width = 0.05)+
  stat_summary(fun.y = median, geom = "point", shape = 95,
               color = "red", size = 15, alpha = 0.6) +
  labs(x = NULL) +
  theme_tufte()
```

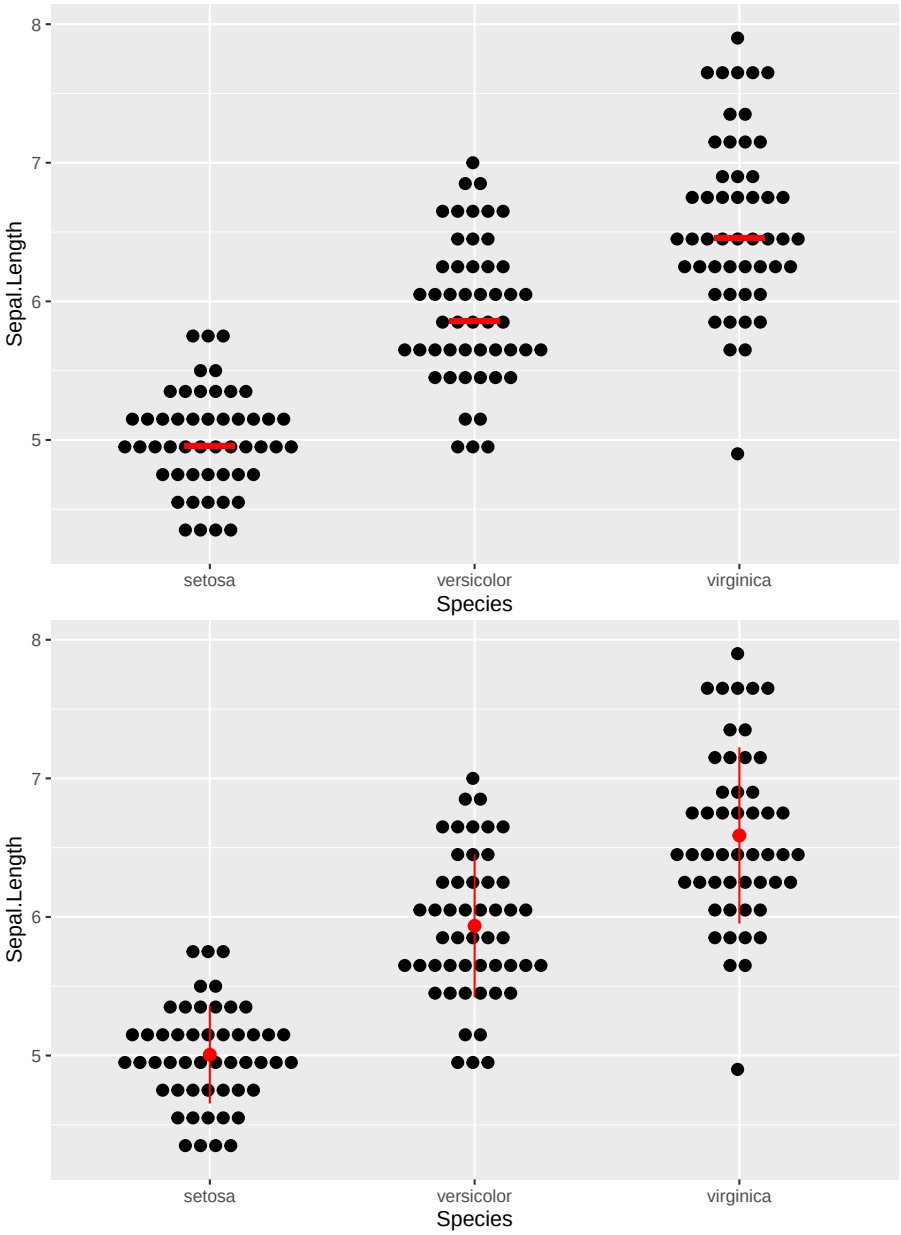


Siin on meil lausa 50 andmepunkti iga Irise liigi kohta ja graafik on ikkagi täitsa hästi loetav.

Meil on võimalik teha sellest graafikust versioon, mis ei pane andmepunkti y skaalal täpselt õigesse kohta, vaid tekitab histogrammilaadsed andmebinnid, kus siiski iga punkt on eraldi näidatud. See lihtsustab veidi "kirjude" kompleksete andmete esitust, kuid kaotab informatsiooni andmepunktide täpse asukoha kohta. Eesmärk on muuta erinevused gruppide vahel paremini võrreldavaks.

```
p<-ggplot(iris, aes(x=Species, y=Sepal.Length)) +
  geom_dotplot(binaxis='y', stackdir='center', stackratio=1.3, dotsize=0.7)
```

```
p + stat_summary(fun.y = median, geom = "point", shape = 95,
                 color = "red", size = 15)
#try using shape=18, size=5.

#add mean and SD, use pointrange:
p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1), geom="pointrang

#use errorbar instead of pointrange:
#p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1), geom="errorba
```



Muuda punktide värvi nii:

scale_fill_manual() : to use custom colors

scale_fill_brewer() : to use color palettes from RColorBrewer package

scale_fill_grey() : to use grey color palettes

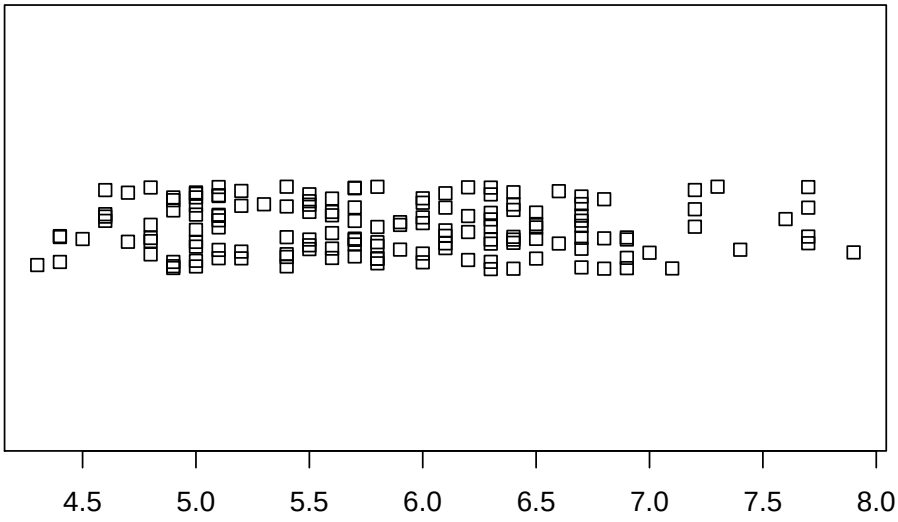
0.12.3 Histogramm

x - pidev muutuja

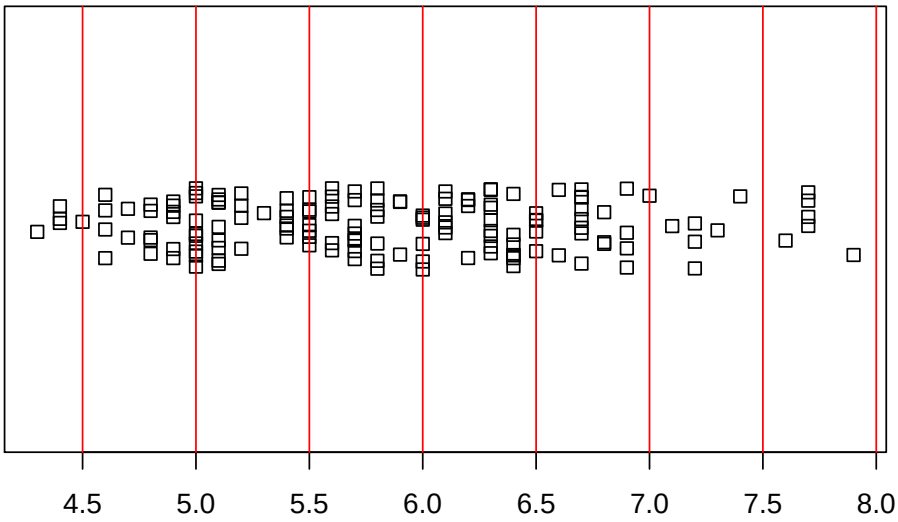
Kui teil on palju andmepunkte (>50) ning soovite uurida nende jaotust (ja/või võrrelda mitme andmestiku jaotust) siis tasub kindlasti alustada histogrammist. Histogrammi koostamine näeb välja järgmine:

1. ploti andmepunktid x - teljele (järgnev on põhimõtteliselt ühedimensionaalne plot, kuigi andmepunktid on üksteise suhtes veidi nihutatud, et nad üksteist ei varjutaks).

```
stripchart(iris$Sepal.Length, method = "jitter")
```

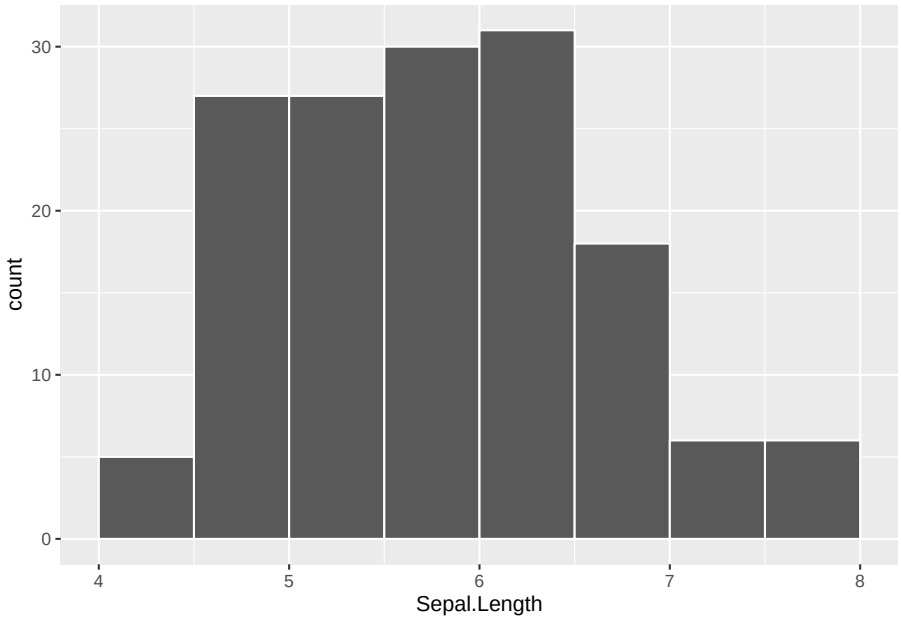


2. jaga andmestik x-teljel võrdse laiusega vahemikesse (binnidesse)



3. loe kokku, mitu andmepunkti sattus igasse binni. Näiteks on meil viimases binnis (7.5 . . . 8) kuus anmdepunkti
4. ploti iga bin tulpdiagrammina (y- teljel on tüüpiliselt andmepunktide arv)

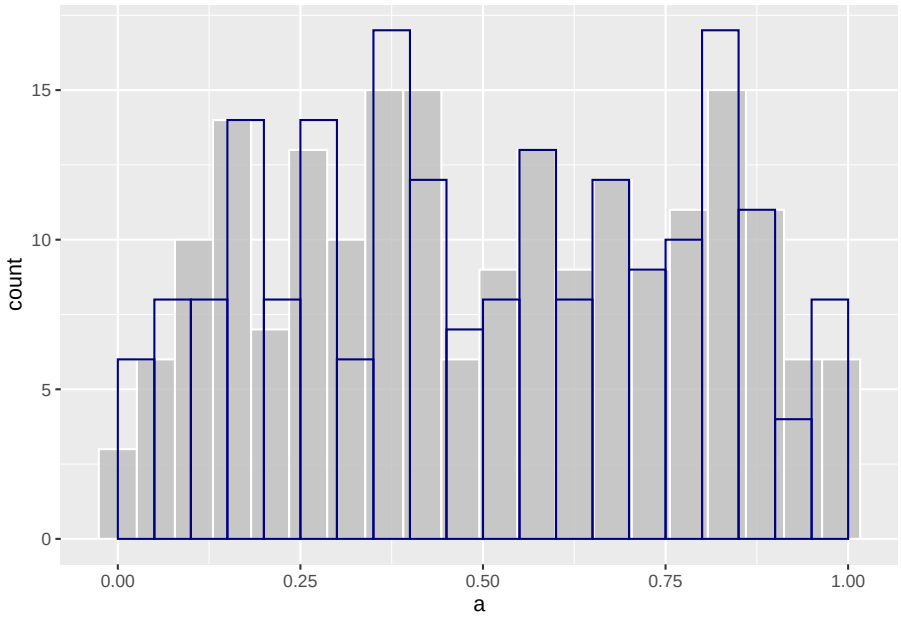
```
ggplot(iris, aes(x=Sepal.Length)) + geom_histogram(breaks= seq(4, 8, by=0.5)
```



Tavaliselt on mõistlik määrata histogrammi binnide laius ja asukoht `breaks` argumeniga. On olemas ka alternatiivsed argumendid `bins`, mis annab binnide arvu, ja `binwidth`, mis annab binni laiuse, aga ohutum on kasutada `breaks`-i. Vt ka geom_boxplot() funktsiooni helpi.

Järnevalt genereerime ühtlase jaotuse 0 ja 1 vahel ning plotime selle kahel histogrammil, millest esimene (halli taustaga) kasutab `bins` argumenti ja teine (sinine) kasutab `breaks` argumenti.

```
set.seed(12)
a1 <- tibble(a=runif(200))
ggplot(a1, aes(a)) +
  geom_histogram(bins = 20, color="white", alpha=0.8, fill="grey")+
  geom_histogram(breaks= seq(0,1, by=0.05), color= "navyblue", fill=NA)
```

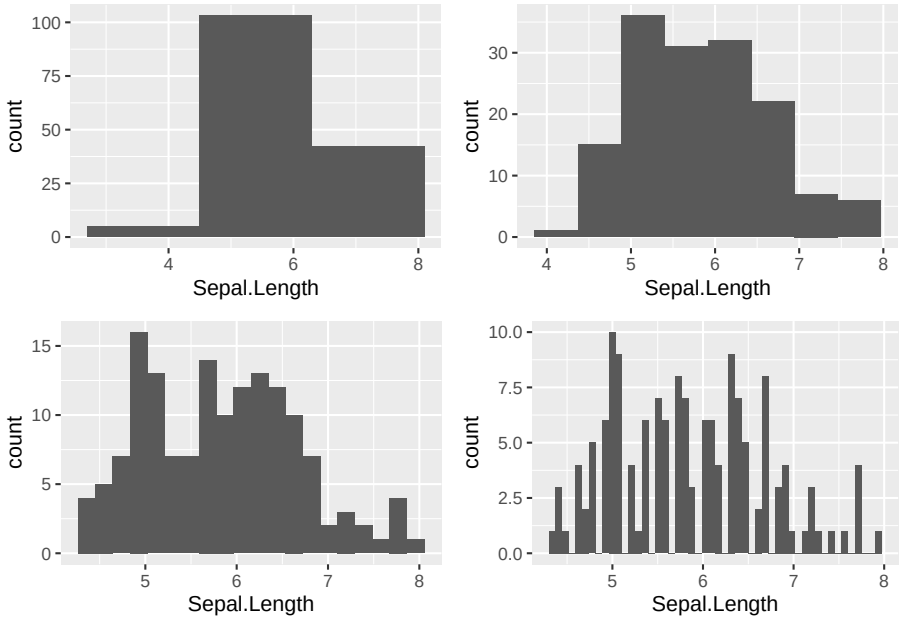


Pane tähele, et tulemus on küllaltki erinev ja et `breaks` argument töötab korrektselt. Nagu järgnev koodijupp näitab, on meil on 6 väärtust alla 0.05 (1. bin) ja 8 väärtust üle 0.95 (20. bin), mis on korrektselt kajastatud ainult `breaks` argumentdiga histogrammil.

```
a1 %>% filter(a < 0.05) %>% nrow()
#> [1] 6
a1 %>% filter(a > 0.95) %>% nrow()
#> [1] 8
```

NB! Väga tähtis on mõista, et binnide laius on meie suva järgi määratud. Samade andmete põhjal joonistatud erineva binilaiusega histogrammid võivad anda lugejale väga erinevaid signaale.

```
library(gridExtra)
g1 <- ggplot(iris, aes(Sepal.Length)) + geom_histogram(bins = 3)
g2 <- ggplot(iris, aes(Sepal.Length)) + geom_histogram(bins = 8)
g3 <- ggplot(iris, aes(Sepal.Length)) + geom_histogram(bins = 20)
g4 <- ggplot(iris, aes(Sepal.Length)) + geom_histogram(bins = 50)
grid.arrange(g1, g2, g3, g4, nrow = 2)
```

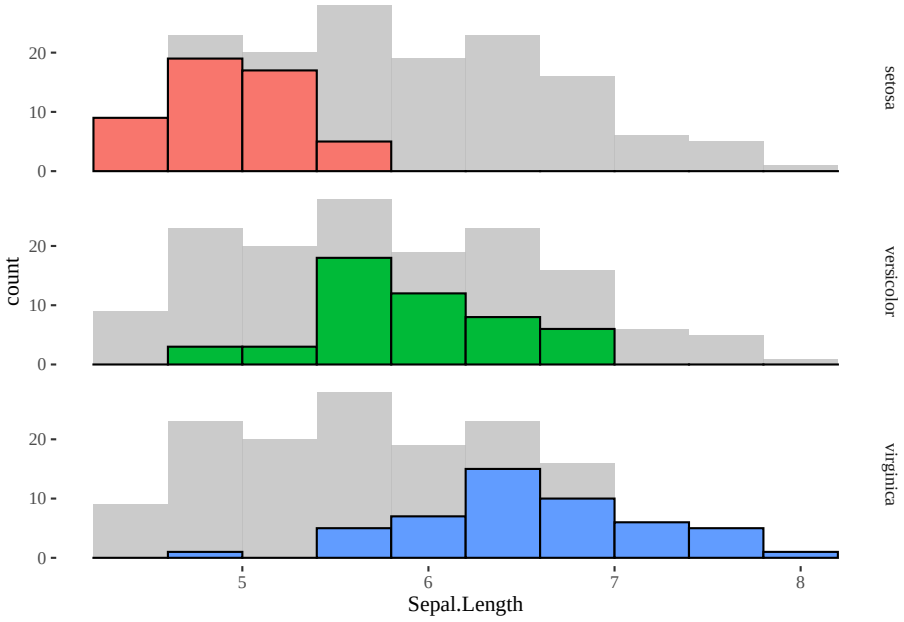


Seega on tasub joonistada samadest andmetest mitu erineva binnilaiusega histogrammi, et oma andmeid vaadata mitme nurga alt.

Kui tahame võrrelda mitmeid jaotusi, siis on meil järgmised variandid:

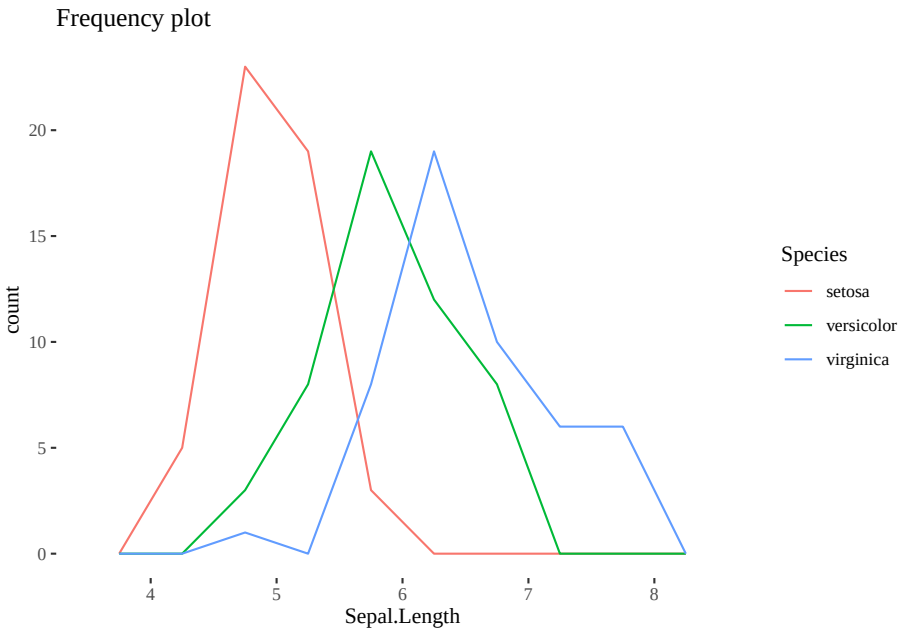
Kõigepealt, me võime panna mitu histogrammi üksteise alla. Selleks kasutame facet_grid funktsiooni ja paneme joonisele ka hallilt summaarsete andmete histogrammi. Selle funktsioon on pakkuda joonise lugejale ühtset võrdlusskaalat üle kolme paneeli.

```
d_bg <- iris[, -5]  # Background Data - full without the 5th column (Specie

ggplot(data = iris, aes(x = Sepal.Length, fill = Species)) +
  geom_histogram(data = d_bg, fill = "grey", alpha=0.8, bins=10) +
  geom_histogram(colour = "black", bins=10) +
  facet_grid(Species~.) +
  guides(fill = FALSE) +  # to remove the legend
  theme_tufte()          # for clean look overall
```



Teine võimalus on näidata kõiki koos ühel paneelil kasutades histogrammi asemel sageduspolügoni. See töötab täpselt nagu histogramm, ainult et tulpade asemel joonistatakse binnitippude vahele jooned. Neid on lihtsam samale paneelile üksteise otsa laduda.

```
ggplot(iris, aes(Sepal.Length, color=Species)) + geom_freqpoly(breaks= seq(4
```



Selle "histogrammi" binne saab ja tuleb manipuleerida täpselt samamoodi nagu geom_histogrammis.

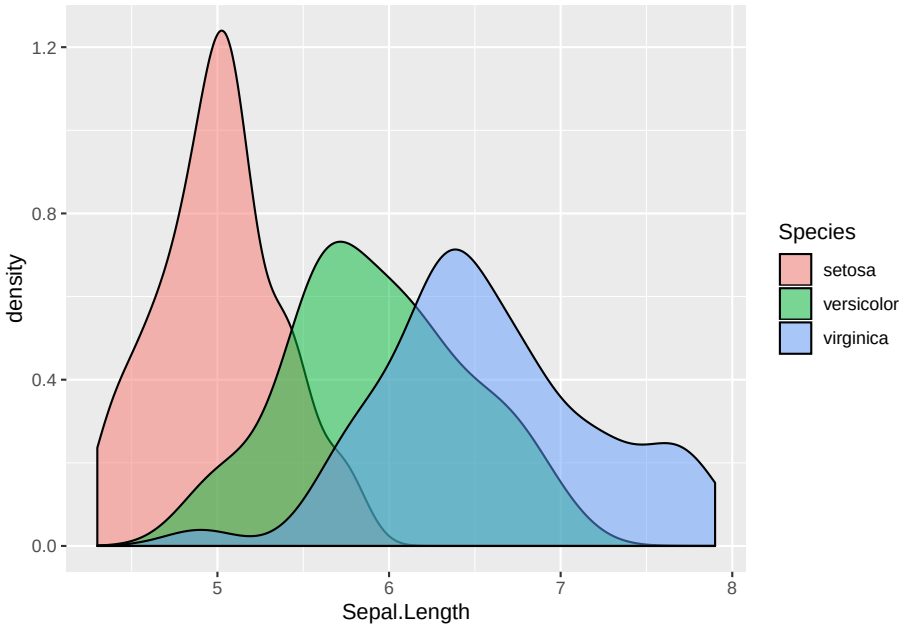
Veel üks hea meetod histogrammide võrdlemiseks on joonistada nn viiuliplot. See asendab sakilise histogrammi silutud joonega ja muudab seega võrdlemise kergemaks. Viiulile on ka kerge lisada algsed andmepunktid

```
ggplot(iris, aes(Species, Sepal.Length)) + geom_violin(aes(color=Species))+
  geom_jitter(size=0.2, width=0.1) + labs(title="Violin plot", x=NULL)
```

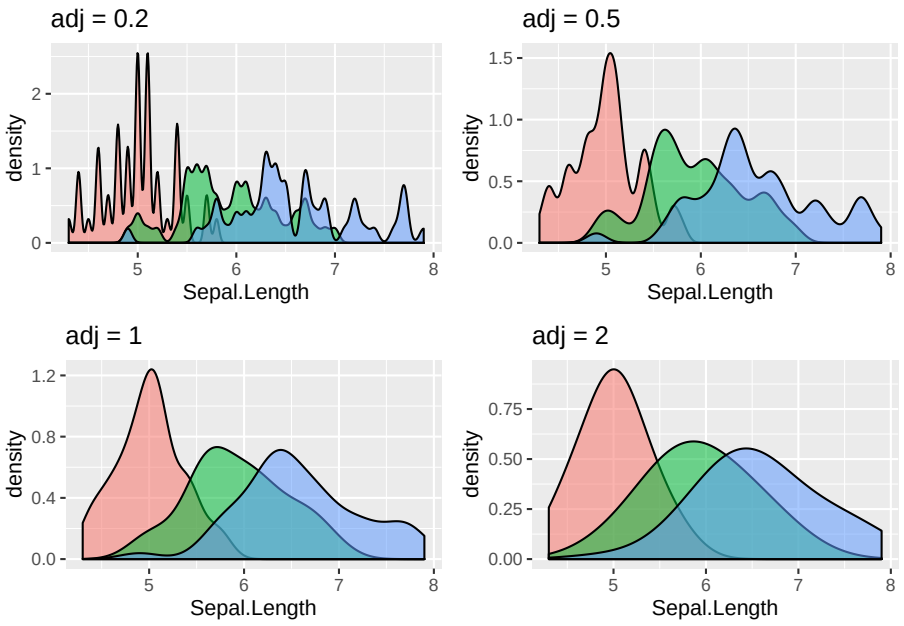
0.12.4 Tihedusplot

Hea alternatiiv histogrammile on joonistada silutud andmejaotus ehk tihedusplot. Tihedusplotil kajastab iga andmejaotust sujuv funktsioon ehk andmete tihedusjoon, ning kõik jaotused on normaliseeritud samale joonealusele pindalale (iga pindlala = 1, mis muudab selle sisuliselt tõenäosusfunktsiooniks). Seega saame hästi võrrelda erinevate jaotuste kujusid, aga mitte kõrgusi (ehk seda, mitu andmepunkti on mingis bin-is – selline võrdlus töötab muidugi histogrammi korral).

```
ggplot(iris, aes(Sepal.Length, fill=Species)) + geom_density(alpha=0.5)
```



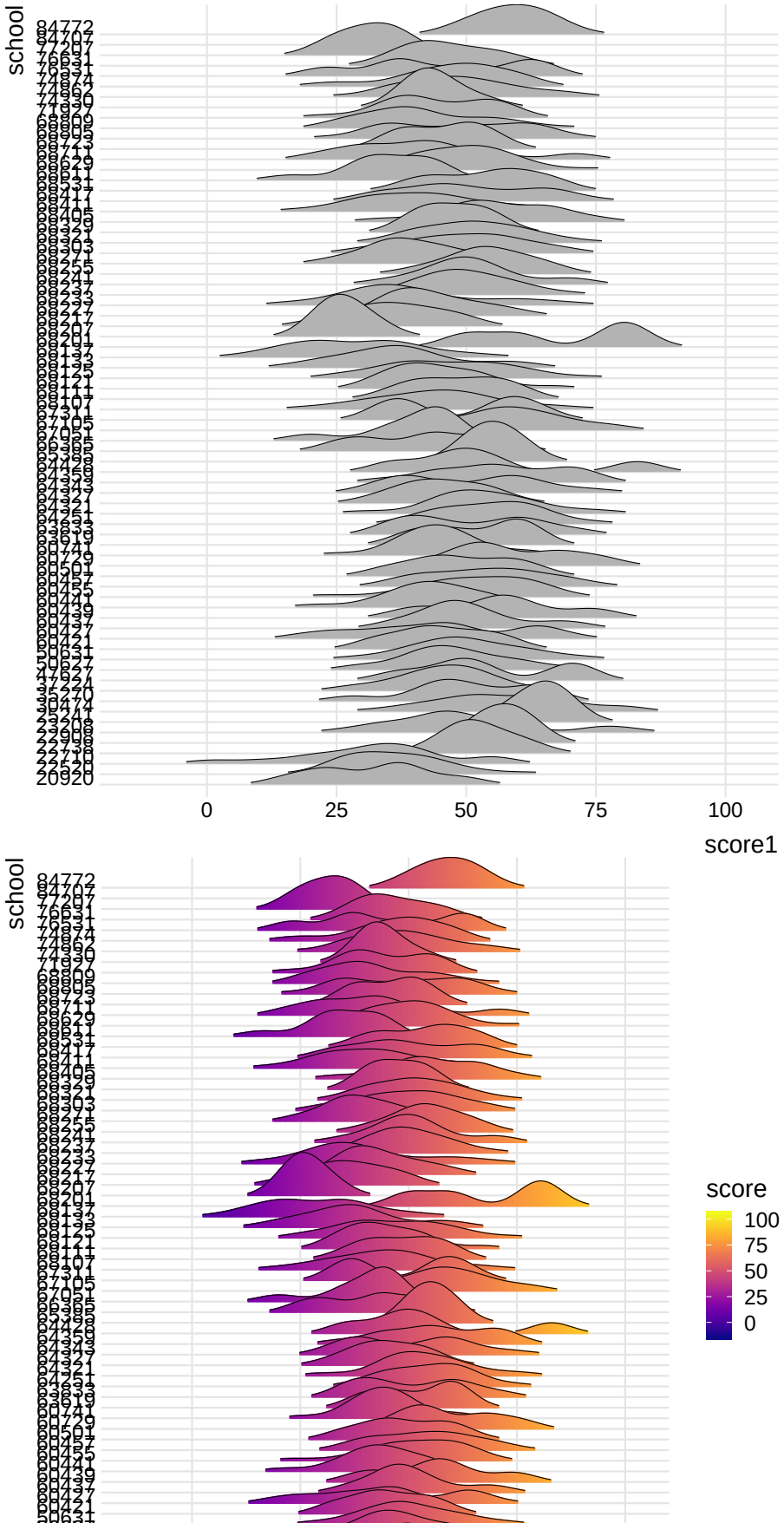
Adjust parameeter reguleerib funktsiooni silumise määra.



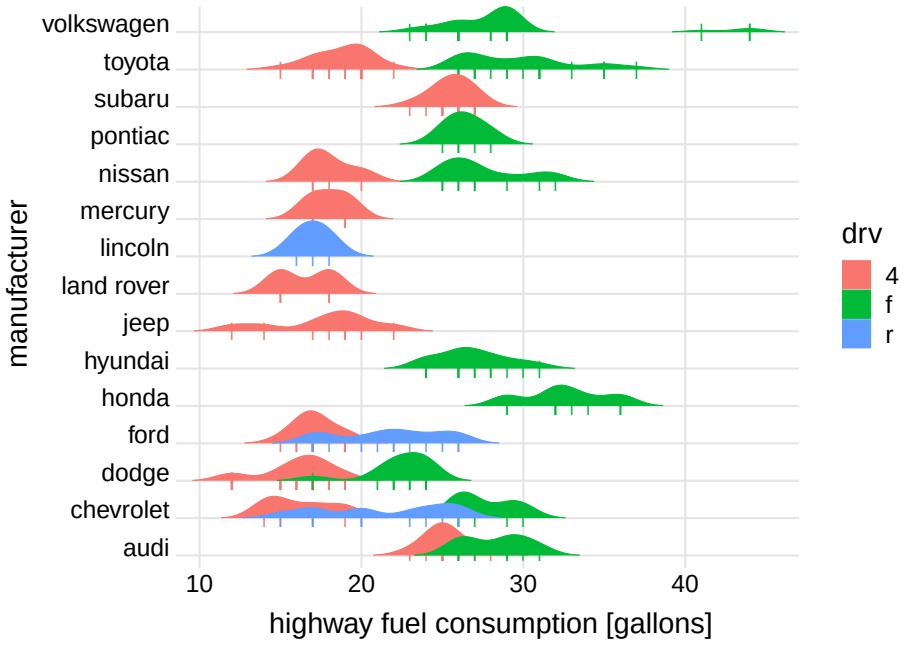
Veel üks võimalus jaotusi kõrvuti vaadata on joyplot, mis paneb samale paneelile kasvõi sada tihedusjaotust. Näiteid vaata ka aadressilt https://cran.r-project.org/web/packages/ggridges/vignettes/gallery.html

```
library(ggridges)
library(viridis)
```

```
sch <- read.csv("data/schools.csv")
sch$school <- as.factor(sch$school)

ggplot(sch, aes(x = score1, y = school, group = school)) +
  scale_fill_viridis(name = "score", option = "C") +
  geom_density_ridges(scale = 4, size = 0.25, rel_min_height = 0.05) +
  theme_ridges()
#> Warning: Removed 202 rows containing non-finite values
#> (stat_density_ridges).

ggplot(sch, aes(x = score1, y = school, group = school, fill = ..x..)) +
  scale_fill_viridis(name = "score", option = "C") +
  geom_density_ridges_gradient(scale = 4, size = 0.25, rel_min_height = 0.05
  theme_ridges()
```


school
0
25
50
75
100
score1
score
100
75
50
25
0

```
ggplot(mpg, aes(x=hwy, y=manufacturer, color=drv, point_color=drv, fill=drv)
  geom_density_ridges(jittered_points=TRUE, scale = .95, rel_min_height = .0
                      point_shape = "|", point_size = 3, size = 0.25,
                      position = position_points_jitter(height = 0)) +
  scale_y_discrete(expand = c(.01, 0)) +
  scale_x_continuous(expand = c(0, 0), name = "highway fuel consumption [gal
  theme_ridges(center = TRUE)
```

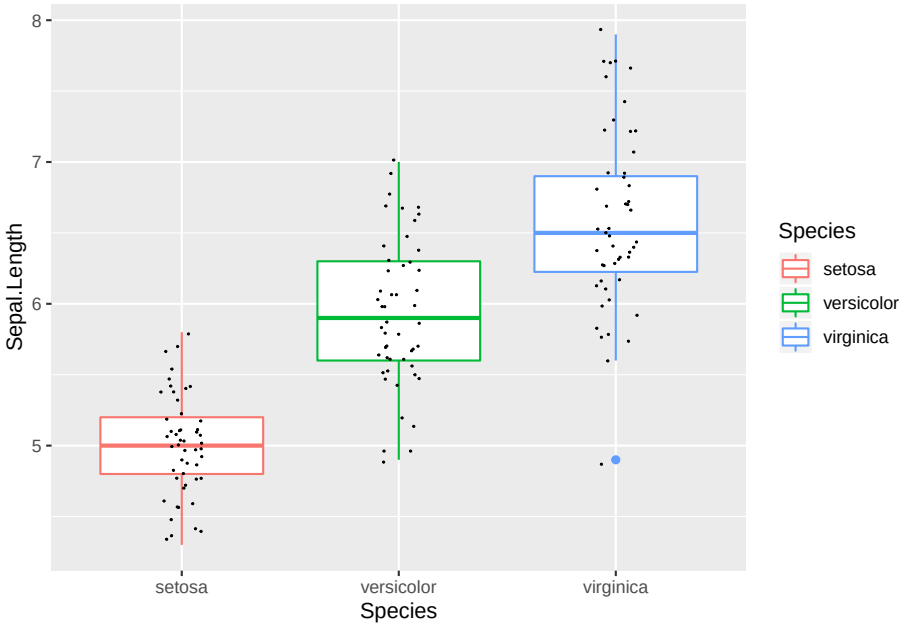


0.12.5 Boxplot

See plot mõeldi välja John Tukey poolt arvutieelsel ajastul (1969), ja see võimaldab millimeeterpaberi ja joonlaua abil võrrelda erinevaid jaotusi. Biomeditsiinis sai boxplot ülipopulaarseks veidi hilinenult, ca. 2010-2015. Inimese jaoks, kes oskab arvutit kasutada, võib viiulite joonistamine tunduda atraktiivsem (ja informatiivsem), aga kui võrreldavaid jaotusi on päris palju, siis võib ka boxploti kandiliselt lihtsusel eeliseid leida. Igal juhul käib klassikalise boxploti konstrueerimine järgevalt.

1. joonista andmepunktid 1D-s välja (nagu me tegime histogrammi puhul)
2. keskmine andmepunkt on mediaan. Selle kohale tuleb boxplotil keskmine kriips
3. ümbritse kastiga pooled andmepunktid (mõlemal pool mediaani). Nii määrad nn. interkvartiilse vahemiku (IQR).
4. pooleteistkordne IQR (y-telje suunas) annab meile vurrude maksimaalse pikkuse. Vurrud joonistatakse siiski ainult kuni viimase andmepunktini (aga kunagi mitte pikemad kui 1.5 IQR)
5. andmepunktid, mis jäävad väljaspoole 1.5 x IQR-i joonistatakse eraldi välja kui outlierid.

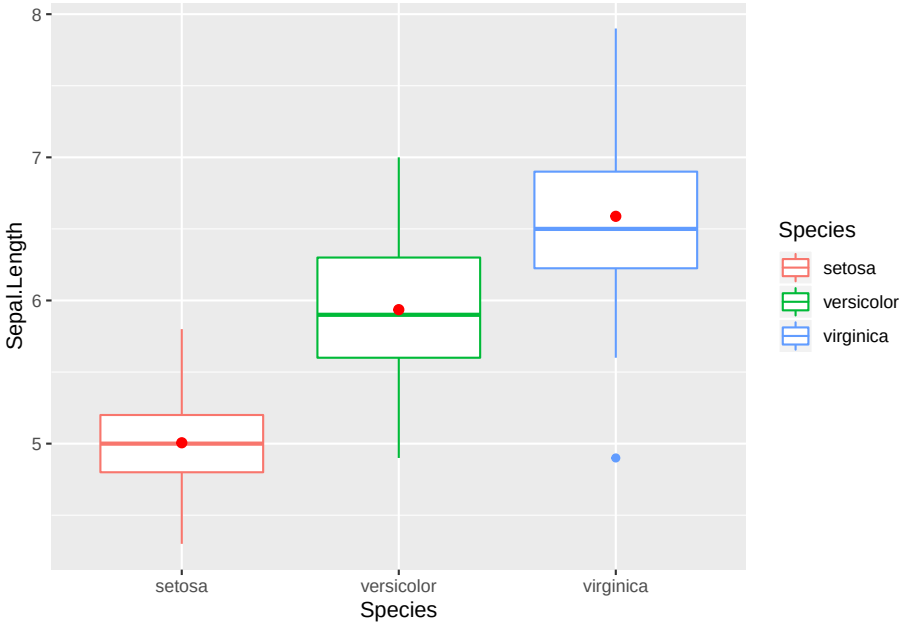
```
ggplot(iris, aes(Species, Sepal.Length, color = Species)) +
  geom_boxplot()+
  geom_jitter(width = 0.1, size=0.1, color="black")
```

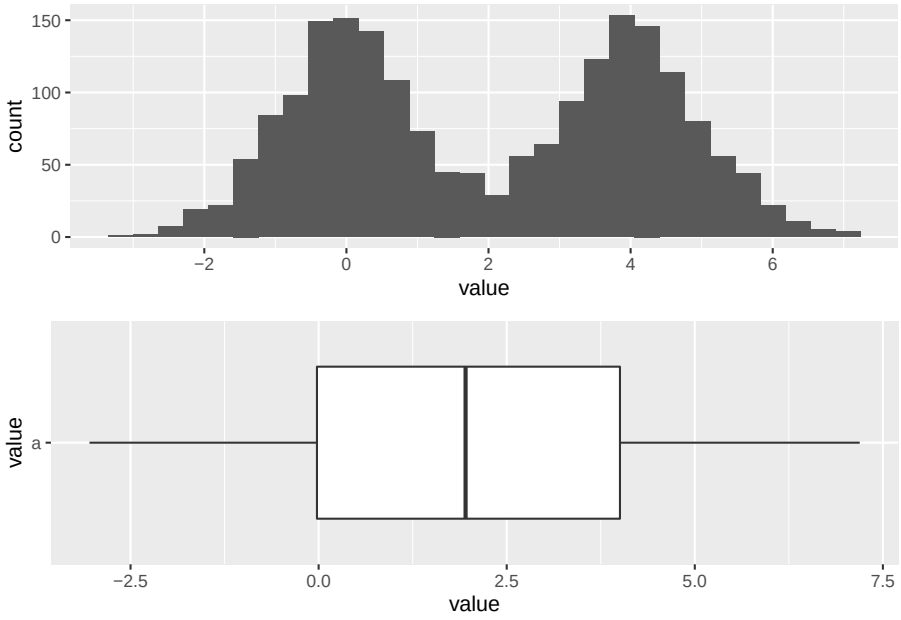


Boxplotile saab lisada ka aritmeetilise keskmise (järgnevas punase

täpina), aga pea meeles, et boxploti põhiline kasu tuleb sellest, et see ei eelda sümmeetrilist andmejaotust. Seega on mediaani lisamine üldiselt parem lahendus.

```
ggplot(iris, aes(Species, Sepal.Length, color = Species)) +
  geom_boxplot()+ stat_summary(fun.y=mean,col='red', geom='point', size=2)
```





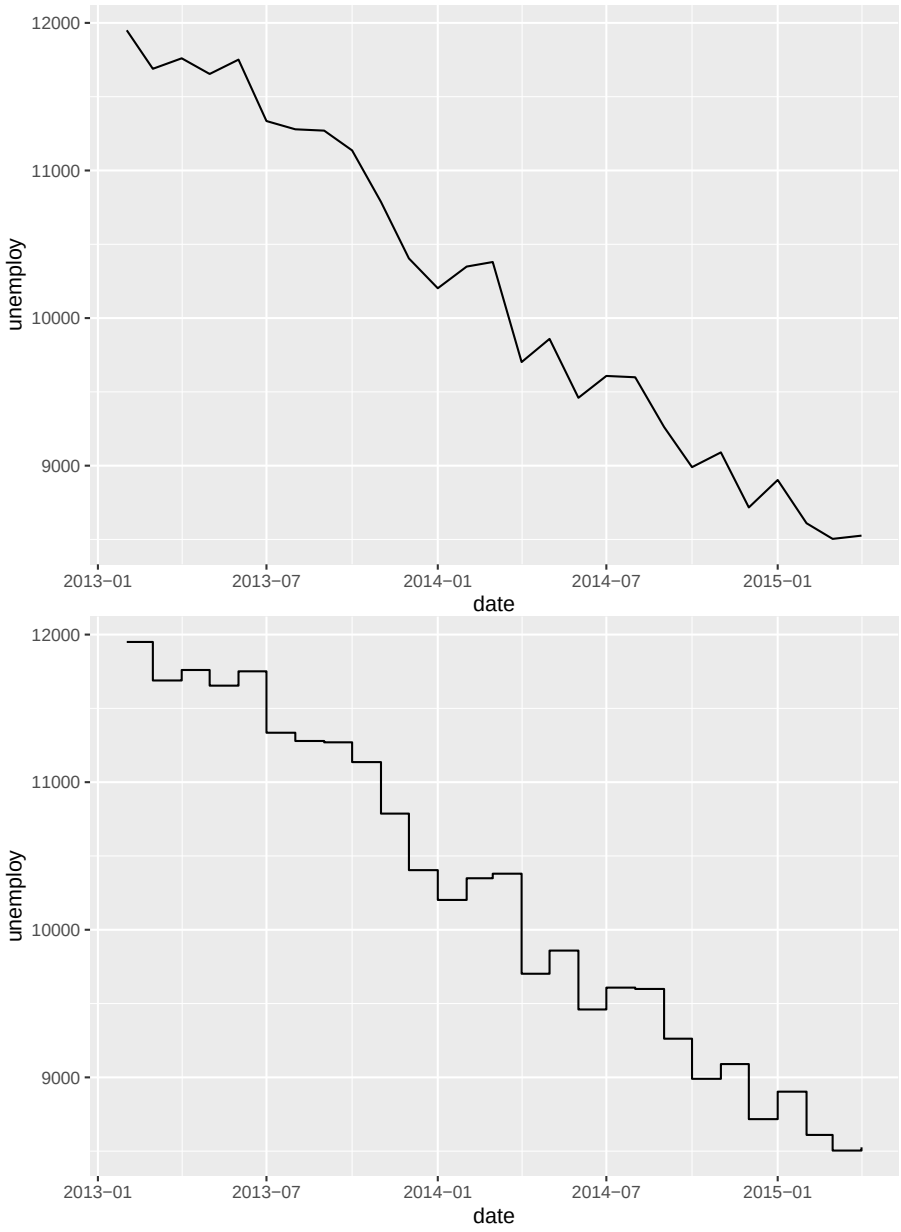
See pilt näitab, et kui jaotus on mitme tipuga, siis võib boxplotist olla rohkem kahju kui kasu.

0.12.6 Joongraafikud

x - pidev muutuja (aeg, konsentratsioon, jms); y - pidev muutuja; x ja y vahel on deterministlik seos (trend)

Joongraafik (geom_line) töötab hästi siis, kui igale x-i väärtusele vastab unikaalne y-i väärtus ja iga kahe mõõdetud x-i väärtuse vahele jääb veel x-i väärtusi, mida pole küll mõõdetud, aga kui oleks, siis vastaks ka neile unikaalsed y-i väärtused. Lisaks me loodame, et y-i suunaline juhuslik varieeruvus ei ole nii suur, et maskeerida meid huvitavad trendid. Kui tahad näidata, kus täpselt muutus toimus, kasuta geom_step funktsiooni.

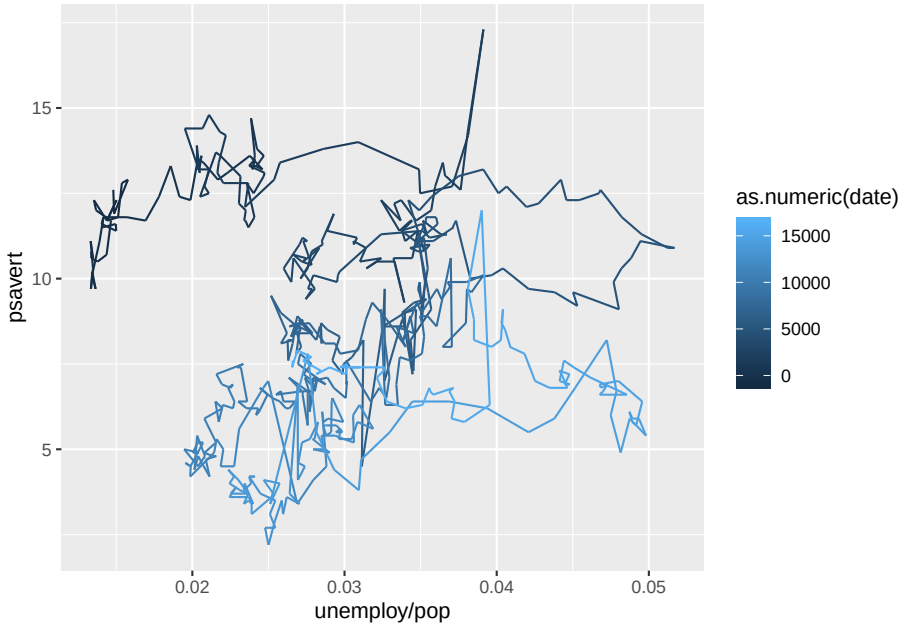
```
recent <- economics[economics$date > as.Date("2013-01-01"), ]
ggplot(recent, aes(date, unemploy)) + geom_line()
ggplot(recent, aes(date, unemploy)) + geom_step()
```



Astmeline graafik on eriti hea olukorras, kus astmete vahel y-dimensioonis muutust ei toimu – näiteks piimapaki hinna dünaamika poes.

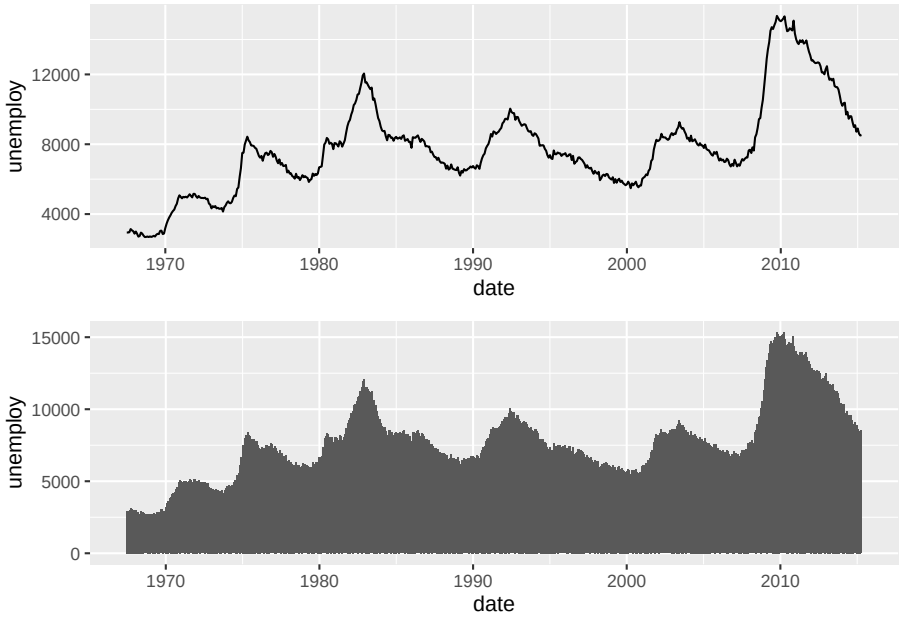
Geom_path võimaldab joonel ka tagasisuunas keerata.

```
# geom_path lets you explore how two variables are related over time,
# e.g. unemployment and personal savings rate
m <- ggplot(economics, aes(unemploy/pop, psavert))
m + geom_path(aes(colour = as.numeric(date)))
```



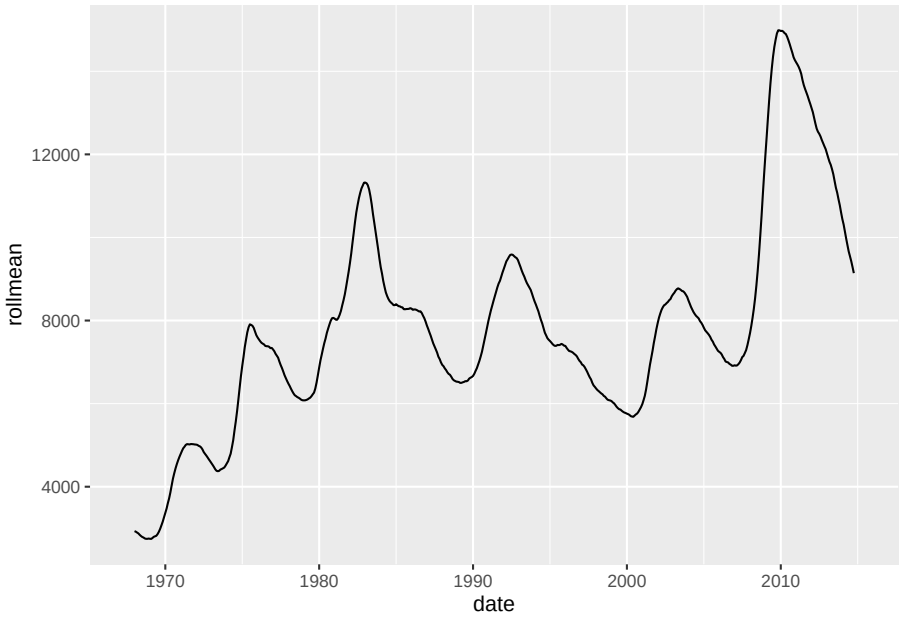
Tulpdiagramm juhib lugeja tähelepanu väikestele teravatele muutustele. Kui see on see, millele sa tahad tähelepanu juhtida, siis kasuta seda.

```
p1 <- ggplot(economics, aes(date, unemploy)) + geom_line()
p2 <- ggplot(economics, aes(date, unemploy)) + geom_bar(stat="identity")
grid.arrange(p1, p2, nrow = 2)
```



Et mürarikkaid andmeid siluda kasutame liikuva keskmise meetodit. Siin asendame iga andmepunkti selle andmepunkti ja tema k lähima naabri keskmisega. k on tavaliselt paaritu arv ja mida suurem k, seda silutum tuleb tulemus.

```
library(zoo)
economics$rollmean <- rollmean(economics$unemploy, k = 13, fill = NA)
ggplot(economics, aes(date, rollmean)) + geom_line()
#> Warning: Removed 12 rows containing missing values
#> (geom_path).
```



Kui on oht, et ebahuvitavad tsüklid ja trendid varjutavad veel mingeid mustreid, mis meile võiks huvi pakkuda, võib proovida lahutada aegrea komponentideks kasutades seasonaalset lahutamist (Seasonal decomposition). R::stl() kasutab selleks loess meetodit lahutades aegrea kolmeks komponendiks. 1) trendikomponent püüab keskmise taseme muutusi ajas. 2) seasonaalne komponent lahutab muutused aastaaegade lõikes (konstantse amplituudiga tsüklilisus aegrea piires) ja 3) irregulaarne komponent on see, mis üle jääb. aegrea osadeks lahutamine võib olla additiivne või mulitlikatiivne. Additiivses mudelis

$$Y_t = Trend_t + Seasonal_t + Irregular_t$$

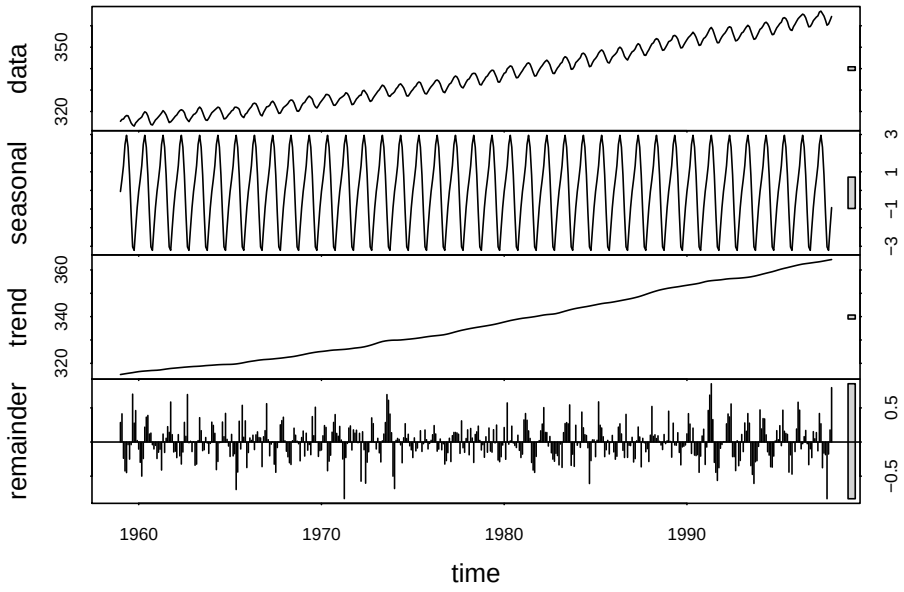
summeeruvad komponendid igas punktis algesele aegreale. Muliplikatiivses mudelis

$$Y_t = Trend_t * Seasonal_t * Irregular_t$$

tuleb selleks teha korrutamistehe.

Näiteks lahutame aegrea, mis käsitleb CO_2 konsentratsiooni muutusi viimase 50 aasta vältel.

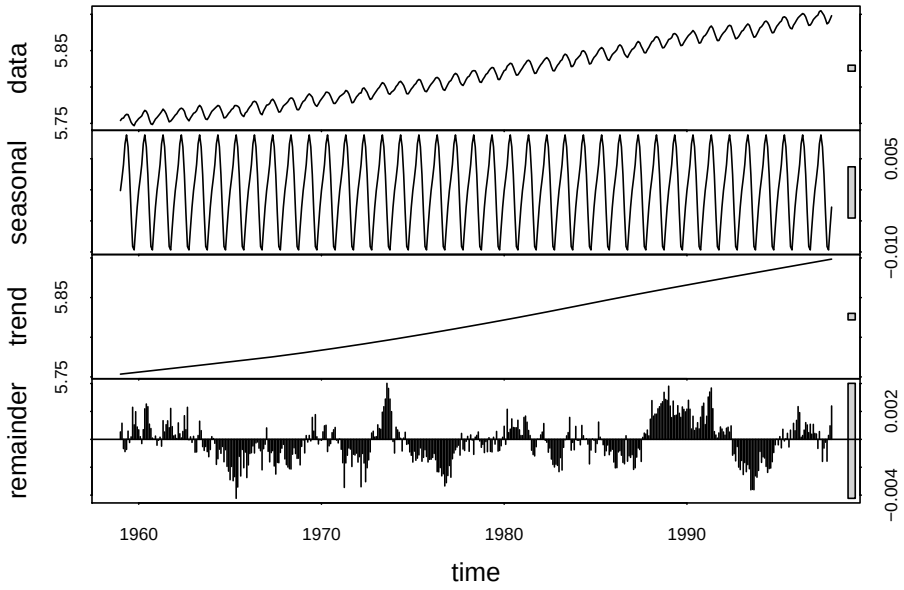
```
require(graphics)
#co2 is a time series object
#stl() takes class "ts" objects only!
plot(stl(co2, "per"))
```



Pane tähele graafiku paremas servas asuvaid halle kaste, mis annavad mõõtkava erinevate paneelide võrdlemiseks. Siit näeme, et "remainder" paneeli andmete kõikumise vahemik on väga palju väiksem kui ülemisel paneelil, kus on plotitud täisandmed.

Nüüd esitame versiooni, kus remainder-i andmeid on tugevasti silutud, et võimalikku signaali mürast eristada.

```
plot(stl(log(co2), s.window = "per", t.window = 199))
# t.window -- the span (in lags) of the loess window for trend extraction,
```



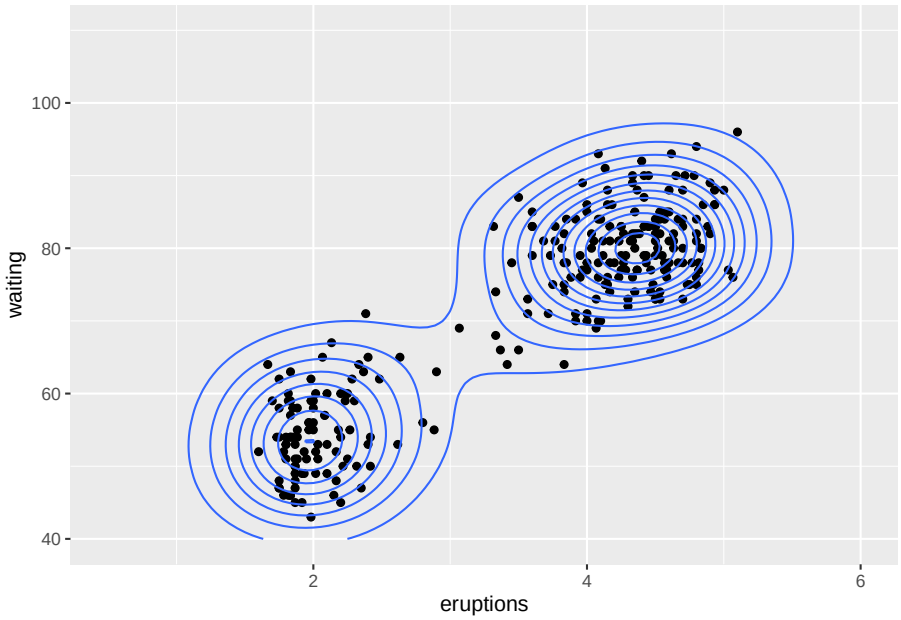
0.12.7 Scatter plot

x - pidev muutuja; y -pidev muutuja; x ja y vahel on tõenäosuslik, mitte deterministlik, seos.

Scatter ploti abil otsime oma andmetest trende ja mustreid.

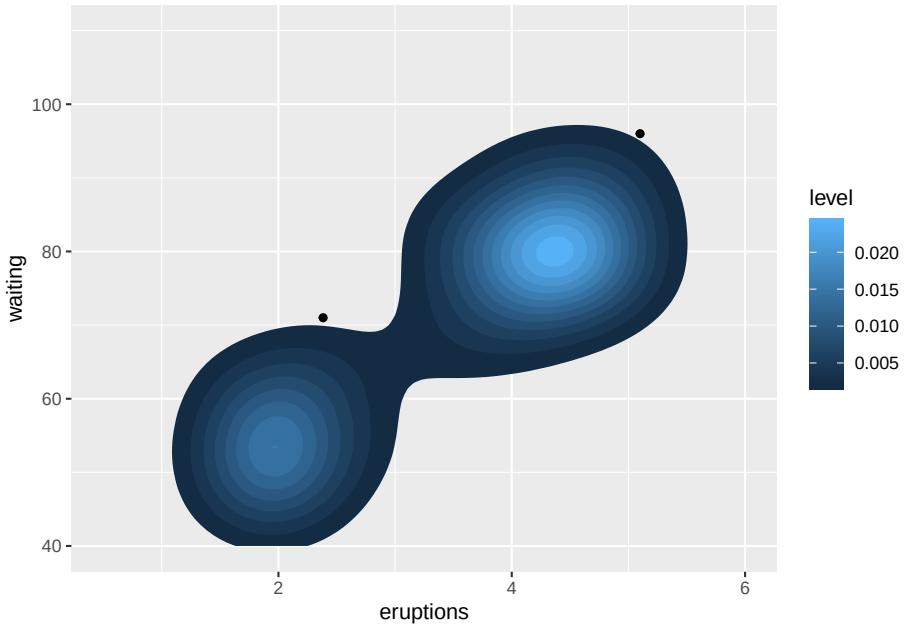
X-teljel on geisri Old Faithful pursete tugevus ja y-teljel pursete vaheline aeg. Kui kahe purske vahel kulub rohkem aega, siis on oodata tugevamat purset. Tundub, et see süsteem töötab kahes diskreetses reziimis.

```
m <- ggplot(faithful, aes(x = eruptions, y = waiting)) +
  geom_point() +
  xlim(0.5, 6) +
  ylim(40, 110)
m + geom_density_2d()
```



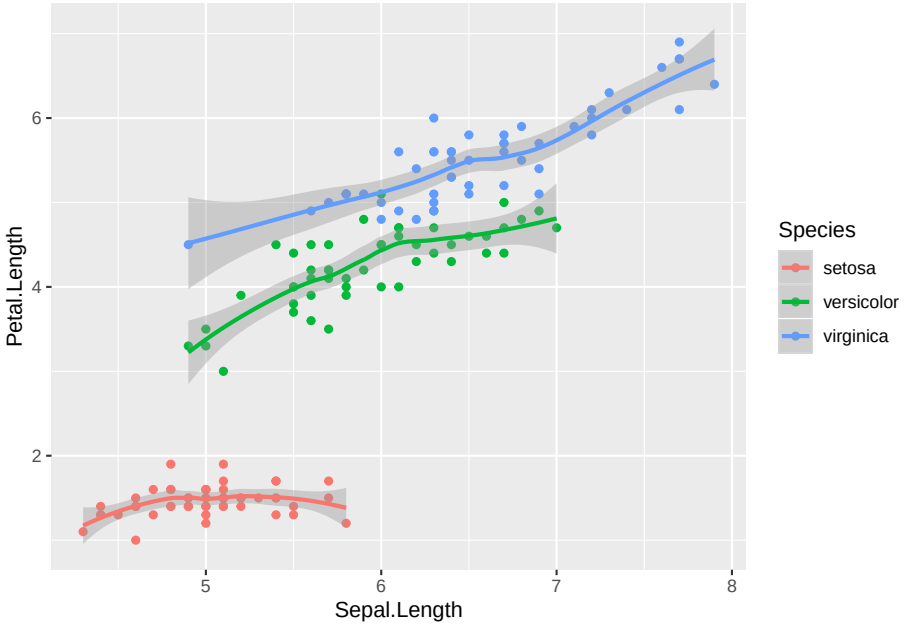
Kui punkte on liiga palju, et välja trükkida, kasuta geom = "polygon" varianti.

```
m + stat_density_2d(aes(fill = ..level..), geom = "polygon")
```



Nüüd plotime 3 iriseliigi õielehe pikkuse seose tolmuka pikkusega, ja lisame igale liigile mittelineaarse mudelennustuse koos 95% usaldusintervalliga. Mudel püüab ennustada keskmist õielehe pikkust igal tolmuka pikkusel, ja 95% CI kehtib ennustusele keskmisest, mitte üksikute isendite õielehtede pikkustele.

```
ggplot(iris, aes(Sepal.Length, Petal.Length, color = Species)) +
  geom_point() +
  geom_smooth()
```

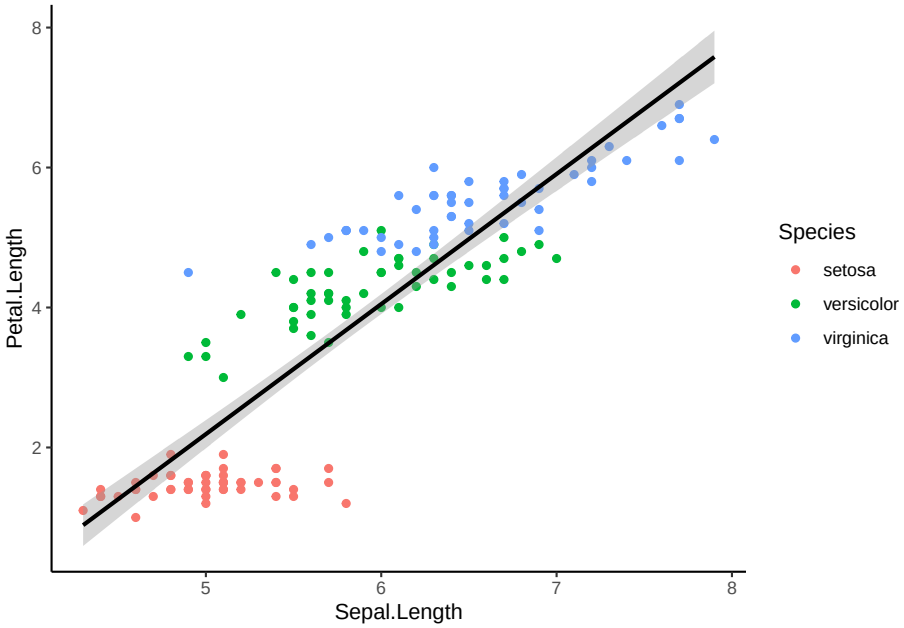


See mudeldamine tehti loess meetodiga, mis kujutab endast lokaalselt kaalutud polünoomset regressiooni. Loessi põhimõte on, et arvuti fitib palju lokaalseid lineaarseid osamudeleid, mis on kaalutud selles mõttes, et andmepunktidel, mis on vastavale osamudelile lähemal, on mudeli fittimisel suurem kaal. Nendest osamudelitest silutakse siis kokku lõplik mudel, mida joonisel näete.

Järgmiseks värvime eelnevalt tehtud plotil punktid iirise liigi kaupa aga joonistame ikkagi regressioonisirge läbi kõikide punktide. Seekord on tegu tavapärase lineaarse mudeliga, mis fititud vähimruutude meetodiga (vt ptk ....).

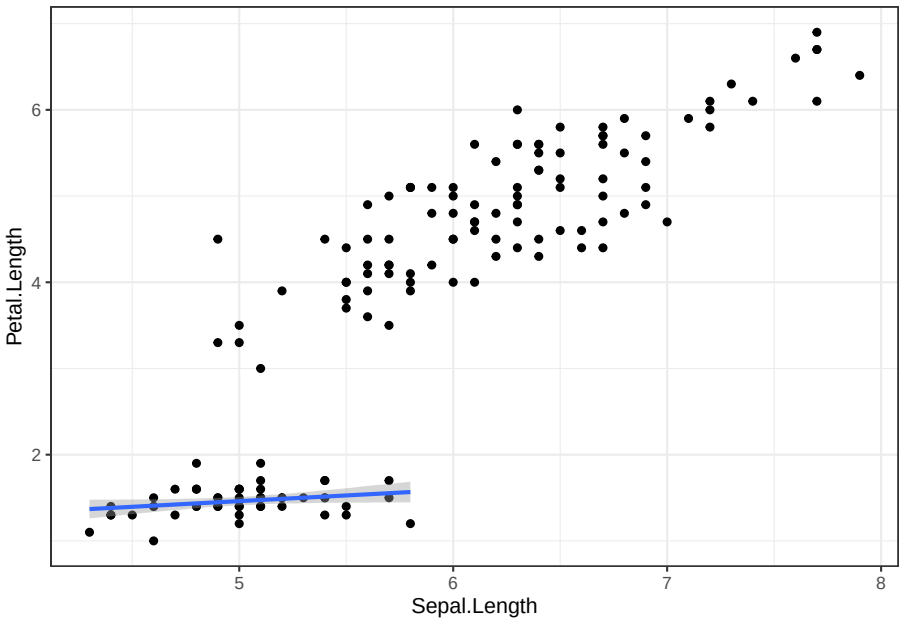
Vaata mis juhtub, kui värvide lahutamine toimub `ggplot()`-i enda `aes()`-s. `theme_classic()` muudab graafiku üldist väljanägemist.

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(aes(color = Species)) +
  geom_smooth(method = "lm", color = "black") +
  theme_classic()
```



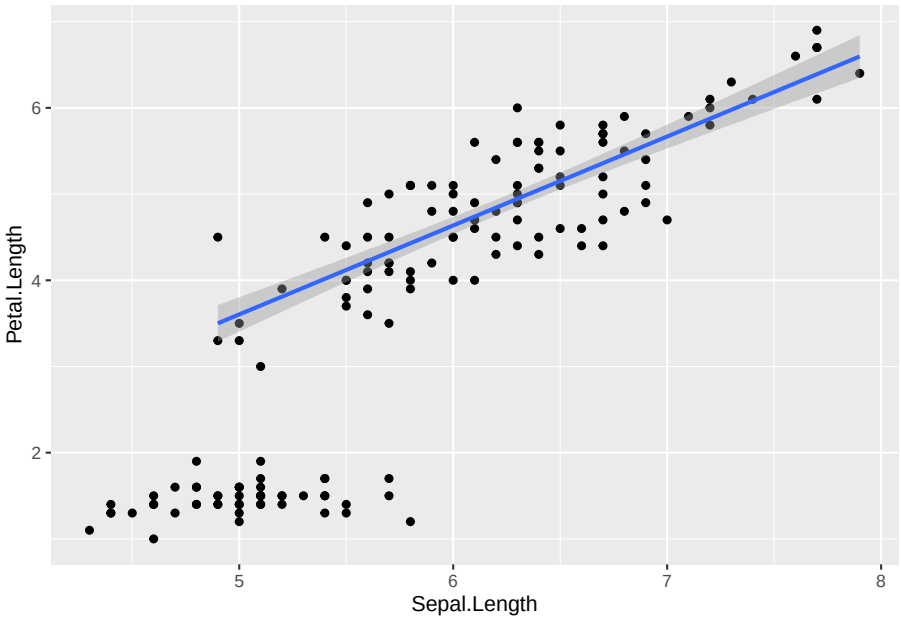
Me võime `geom_smooth()`-i anda erineva andmeseti kui `ggplot()` põhifunktsiooni. Nii joonistame me regressioonisirge ainult nendele andmetele. Proovi ka `theme_bw()`.

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  geom_smooth(data = filter(iris, Species == "setosa"), method = lm) +
  theme_bw()
```



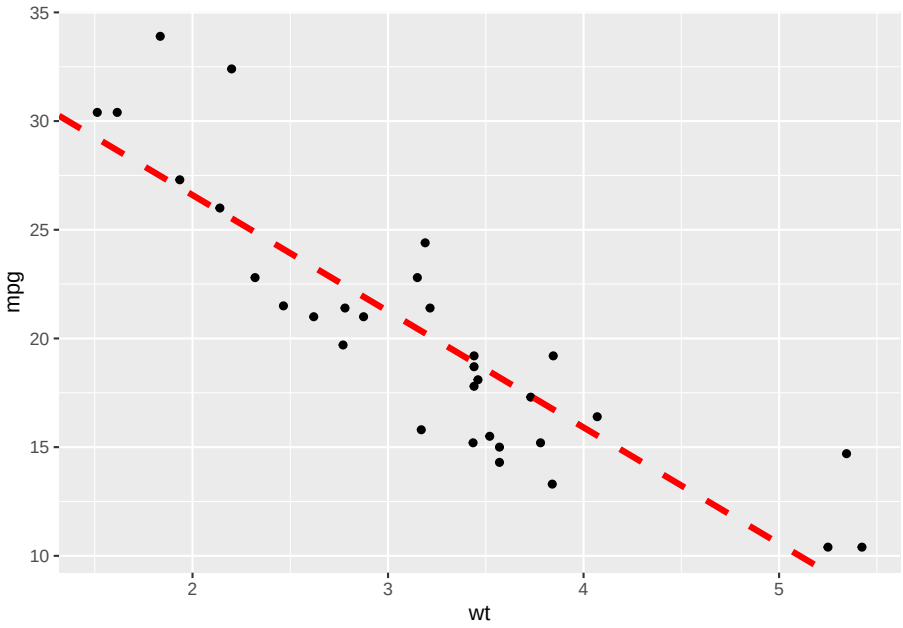
Alljärgnevalt näiteks moodus kuidas öelda, et me soovime regressioonijoont näidata ainult iiriseliikide virginica või versicolor andmetele.

```
## First we filter only data that we want to use for regressionline
smooth_data <- filter(iris, Species %in% c("virginica", "versicolor"))

## Then we use this filtered dataset in geom_smooth
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  geom_smooth(data = smooth_data, method = "lm")
```



Järgnev kood võimaldab eksplitsiitselt kasutada fititud regressioonikoefitsiente, kasutades regeressioonijoone määramiseks koordinaatteljestikus x-telje lõikumispunkti ja sirge tõusu. Lineaarse mudeli fittimist õpime peatükis .... Kasuta `geom_abline()`.

```
## Create plot
p <- ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point()

## Fit model and extract coefficients
model <- lm(mpg ~ wt, data = mtcars)
coefs  <- coef(model)

## Add regressionline to the plot
p + geom_abline(intercept = coefs[1],
                slope = coefs[2],
                color = "red",
                linetype = "dashed",
                size = 1.5)
```



0.12.7.1 Kaalutud lineaarne mudel

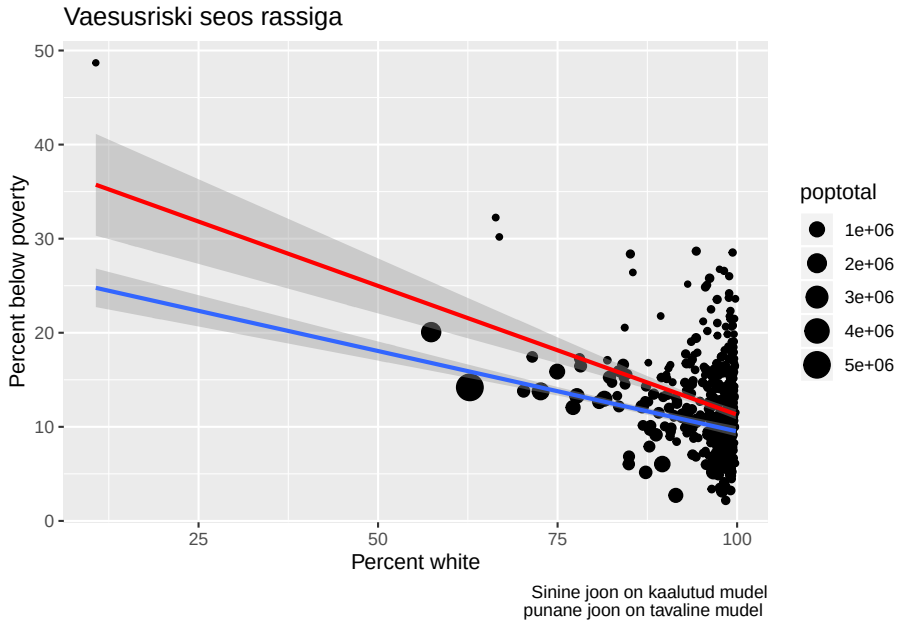
Kaalutud lineaarne mudel on viis anda andmepunktidele, mida me tähtsamaks peame (või mis on täpsemalt mõõdetud) suurem kaal. Kõigepealt, siin on USA demograafilised andmed `midwest` "ggplot2" library-st erinevate kesk-lääne omavalitsuste kohta (437 omavalitsust).

Me valime `midwest` andmetest välja kolm muutujat: "percwhite", "percbelowpoverty", "poptotal".

```
midwest_subset <- midwest %>% select(percwhite, percbelowpoverty, poptotal)
```

Me tahame teada, kuidas valge rassi osakaal ennustab vaesust, aga me arvame, et suurematel omavalitsustel peaks selles ennustuses olema suurem kaal kui väiksematel. Sest me arvame, et väikestel omavalitsustel võib olla suurem valimiviga ja need võivad olla mõjutatud meie mudelis kontrollimata teguritest, nagu mõne suure tööandja käekäik. Selleks lisame `geom_smooth()`-i lisaargumendi "weight".

```
ggplot(midwest_subset, aes(percwhite, percbelowpoverty)) +
  geom_point(aes(size = poptotal)) +
  geom_smooth(aes(weight = poptotal), method = lm, size = 1) +
  geom_smooth(method = lm, color = "red") +
  labs(x = "Percent white",
       y = "Percent below poverty",
       caption = "Sinine joon on kaalutud mudel\npunane joon on tavaline mude
       title = "Vaesusriski seos rassiga")
```



Kaalumine mitte ainult ei muutnud sirge asukohta, vaid vähendas ka ebakindlust sirge tõusu osas.

0.12.8 Tulpdiagramm

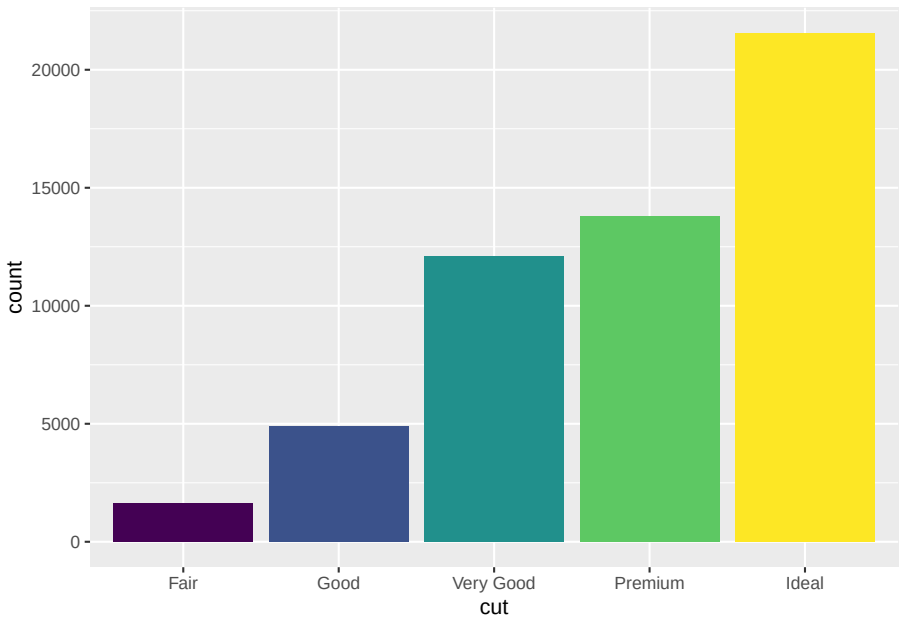
x - faktormuutuja; y - protsent; x - faktormuutuja; y - sündmuse esinemiste arv

Tulpdiagramme on hea kasutada kahel viisil: 1. lugemaks üles, mitu korda midagi juhtus ja 2. näitamaks osa tervikust (proportsiooni).

```
str(diamonds)
#> Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables
#>  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
#>  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3
#>  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2
#>  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3
#>  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
#>  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
#>  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
#>  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
#>  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
#>  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

loeb üles, mitu korda esineb iga cut

```
ggplot(diamonds) +
  geom_bar(aes(x = cut, fill = cut)) +
  theme(legend.position="none")
```



Pane tähele, et y-teljel on arv, mitu korda esineb tabelis iga cut. See

arv ei ole tabelis muutuja. geom_bar, geom_hist, geom_dens arvutavad plotile uued y väärtused — nad jagavad andmed binidesse ja loevad üles, mitu andmepunkti sattus igasse bini.

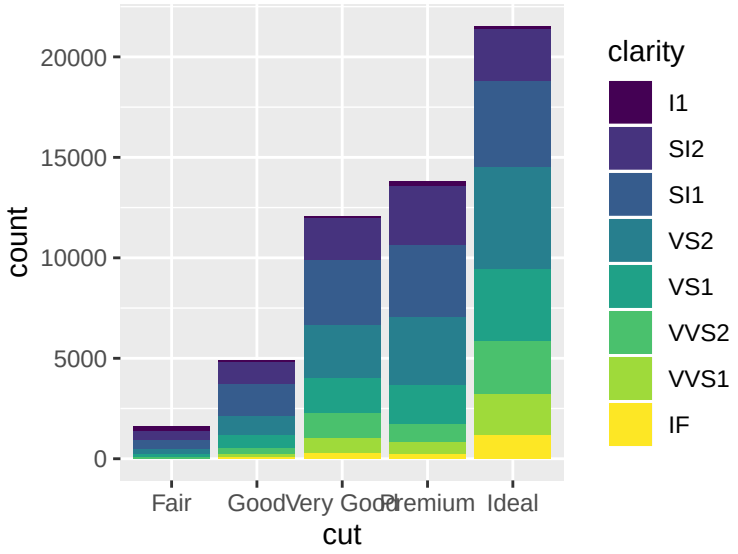
Kui tahad tulpdiagrammi proportsioonidest, mitu korda eineb tabelis igat cut-i, siis tee nii:

```
ggplot(diamonds) +
  geom_bar(aes(x = cut, y = ..prop.., group = 1))
```

Pane tähele et tulpade omavahelised suhted jäid samaks. Muutus ainult y-telje tähistus.

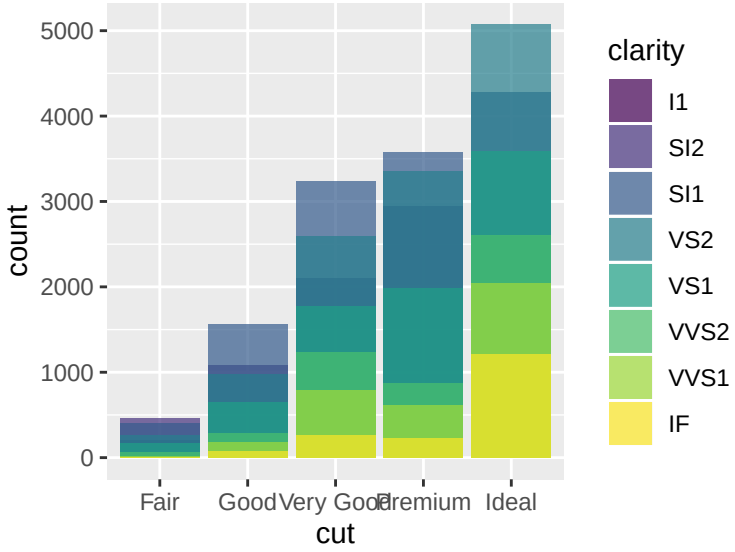
Edasi lisame eelnevale veel ühe muutuja: clarity. Nii saame üles lugeda kõigi cut-i ja clarity kombinatsioonide esinemise arvu või sageduse. Erinvate clarity tasemete esinemiste arv samal cut-i tasemel on siin üksteise otsa kuhjatud, mis tähendab, et tulpade kõrgus ei muutu võrreldes eelnevaga.

```
ggplot(diamonds) +
  geom_bar(aes(x = cut, fill = clarity))
```



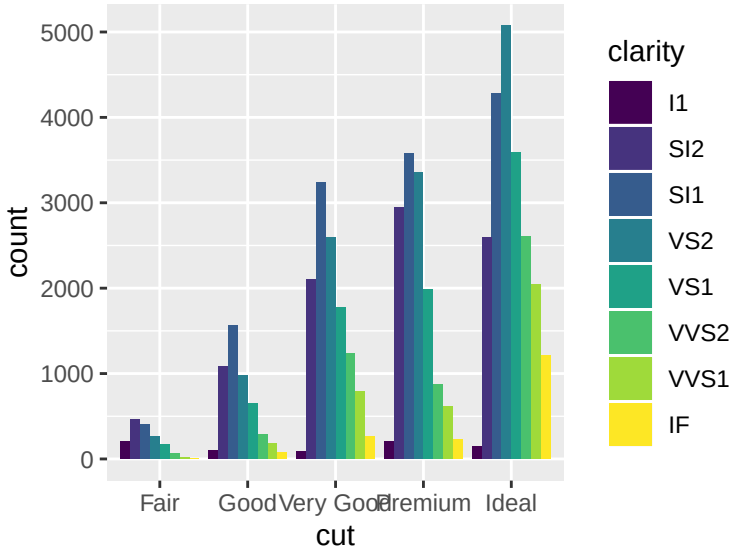
Kui me tahame, et cut-i ja clarity kombinatsioonid oleks kastidena ükteise sees, pigem kui üksteise otsa kuhjatud, siis kasutame position = “identity” argumenti.

```
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(alpha = 0.7, position = "identity")
```



ka see graafik pole väga lihtne lugeda. Parem viime clarity klassid üksteise kõrvale

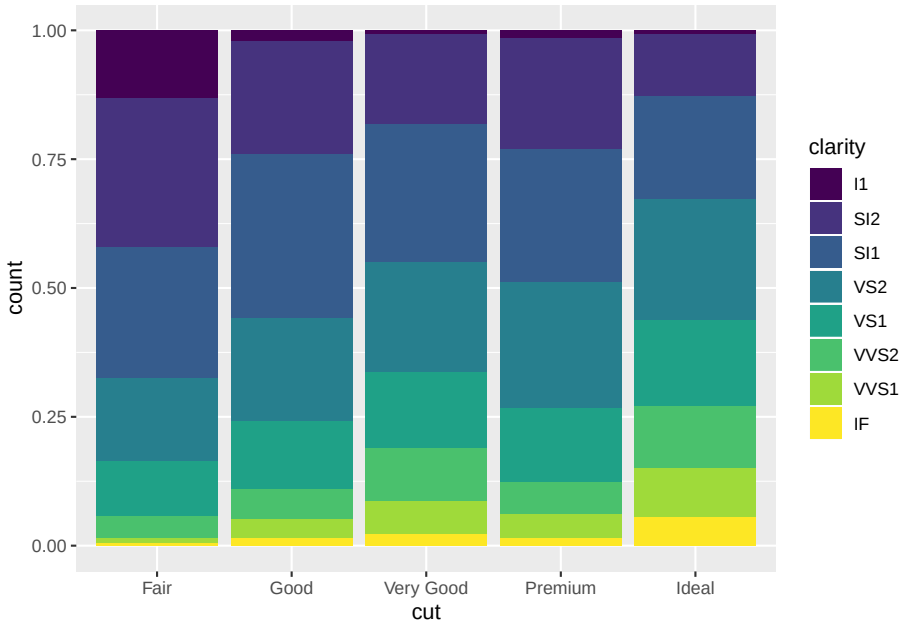
```
ggplot(data = diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position = "dodge")
```



Eelnev on hea viis kuidas võrrelda clarity tasemete esinemissagedusi ühe cut-i taseme piires.

Ja lõpuks, position="fill" normaliseerib tulbad, mis muudab selle, mis toimub iga cut-i sees, hästi võrreldavaks. See on hea viis, kuidas võrrelda clarity tasemete proportsioone erinevate cut-i tasemete vahel.

```
ggplot(data = diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position = "fill")
```



Ja lõpetuseks, kui teile miskipärast ei meeldi Cleveland plot ja te tahate plottida tulpdiagrammi nii, et tulba kõrgus vastaks tabeli ühes lahtris olevale numbrile, mitte faktortunnuse esinemiste arvule tabelis, siis kasutage `geom_col()`

```
df <- tibble(a=c(2.3, 4, 5.2), b=c("A", "B", "C"))
ggplot(df, aes(b, a)) + geom_col()
```

0.12.9 Residuaalide plot

Alustame lineaarse mudeli fittimisest ja mudeli ennustuse lisamisest algsele andmetabelile. Me fitime polünoomsse mudeli:

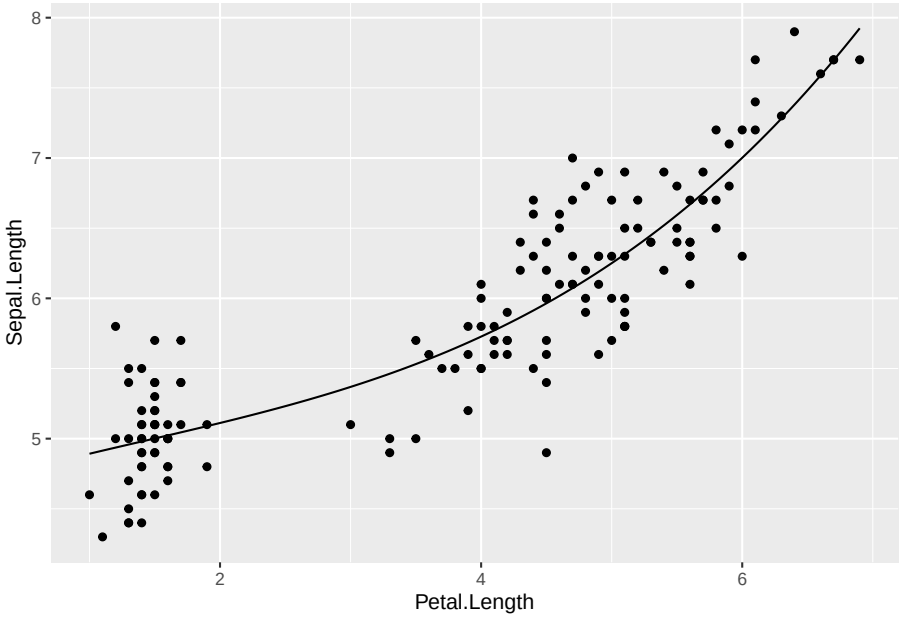
$$Sepal.Length = intercept + b_1 * Petal.Length + b_2 * Petal.Length^2 + b_3 * Petal.Length^3$$

Mudeli ennustused keskmisele õielehe pikkusele (Sepal.Length) saame arvutada fikseerides mudeli koefitsiendid nende fititud väärtustega ja andes mudeli valemisse ühtlase rea võimalikke tolmuka pikkusi. Nii saame igale selle rea liikmele vastava ennustuse õielehe keskmisele pikkusele. Selleks teeme ühetulbalise andmeraami pred_matrix, millele lisame abifunktsiooni add_predictions() abil arvutatud mudeli ennustused. Need ilmuvad tabelisse uue tulbana "pred".

```
library(modelr)
#fit the model
m1 <- lm(Sepal.Length~poly(Petal.Length, 3) , data= iris)

#make prediction matrix (equally spaced non-empirical Petal Length values)
pred_matrix <- tibble(Petal.Length=seq(min(iris$Petal.Length),
                                       max(iris$Petal.Length),
                                       length.out= 100))

#add prediction to each value in the prediction matrix
pred_matrix <- add_predictions(pred_matrix, m1)
```

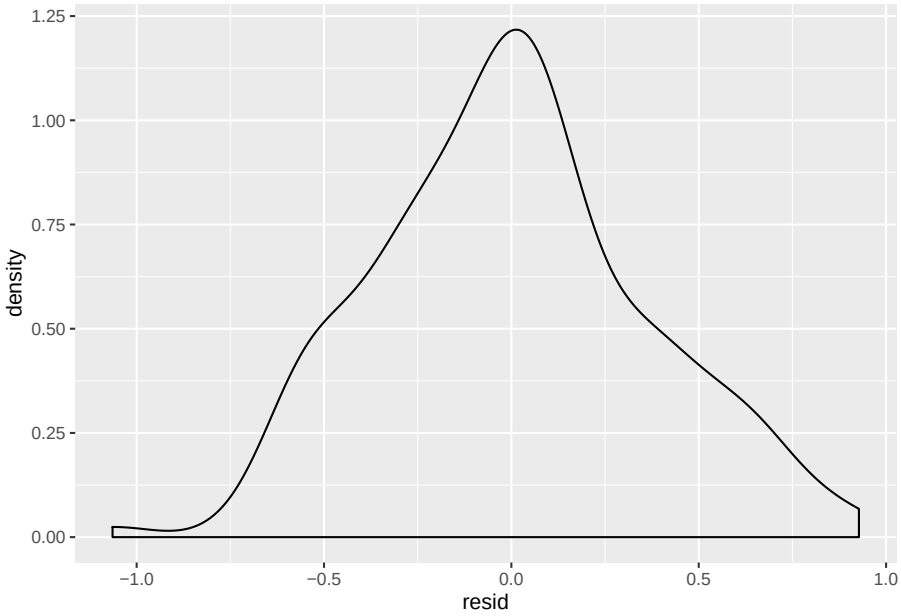
Nii saab mugavalt visualiseerida ka keeruliste mudelite ennustusi.

```
ggplot(pred_matrix, aes(x = Petal.Length)) +
  geom_point(data= iris, aes(y = Sepal.Length)) +
  geom_line(aes(y = pred))
```



Nüüd lisame irise tabelisse residuaalid mugavusfunktsiooni add_residual() abil (tekib tulp "resid"). Residuaal on lihtsalt andmepunkti Sepal.Length väärtus miinus mudeli ennustus.

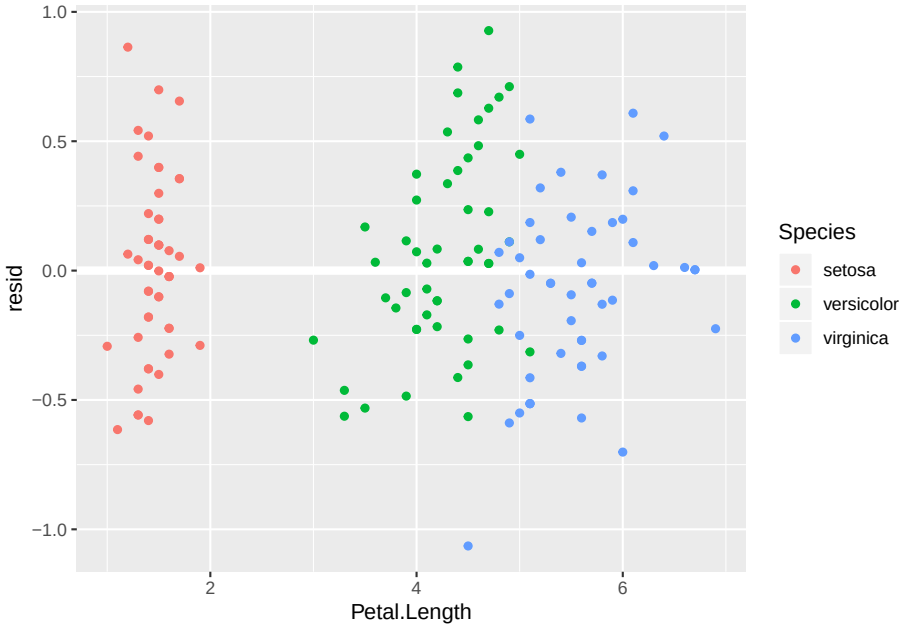
```
iris1 <- iris
iris1 <- add_residuals(iris1, m1)
ggplot(iris1, aes(resid)) + geom_density()
```



See plot näitab, et residuaalid on enam vähem 0-i ümber koondunud, aga negatiivseid residuaale paistab veidi enam olevat.

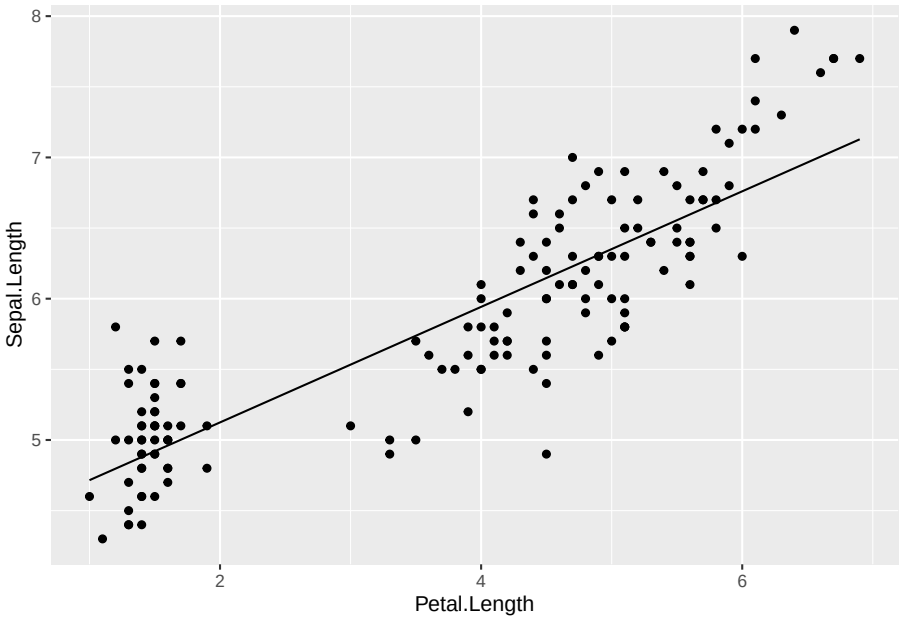
Tegelik residuaaliplot näeb välja selline:

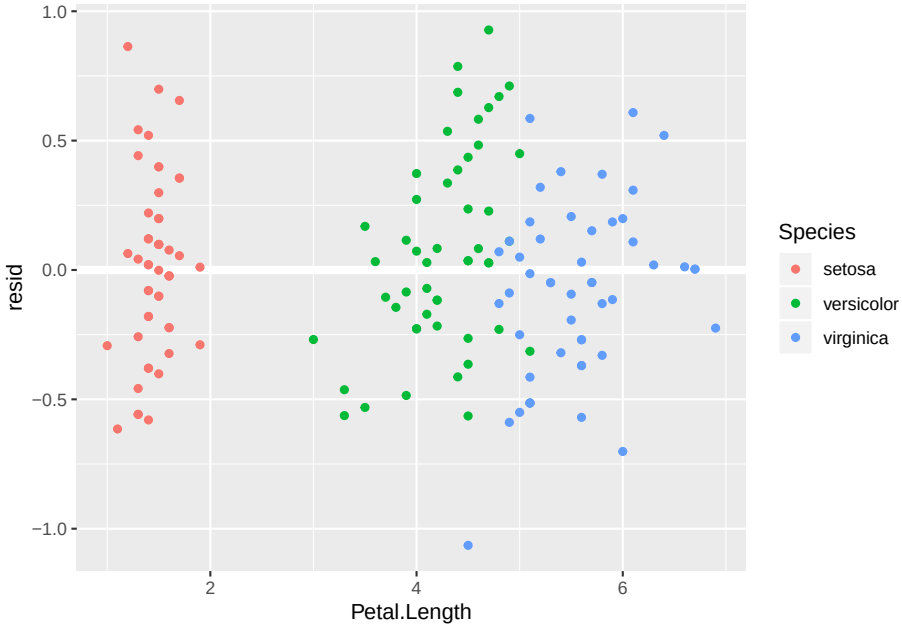
```
ggplot(iris1, aes(Petal.Length, resid, color=Species)) +
  modelr::geom_ref_line(h = 0) +
  geom_point()
```



See võimaldab otsustada, kas mudel ennustab võrdselt hästi erinevatel predikrori (Petal.Length) väärtustel. Antud mudelis ei näe me süstemaatilisi erinevusi residuaalides üle õielehtede pikkuste vahemiku.

Proovime sama lihtsa lineaarse mudeliga $Sepal.Length = intercept + b * Petal.Length$.



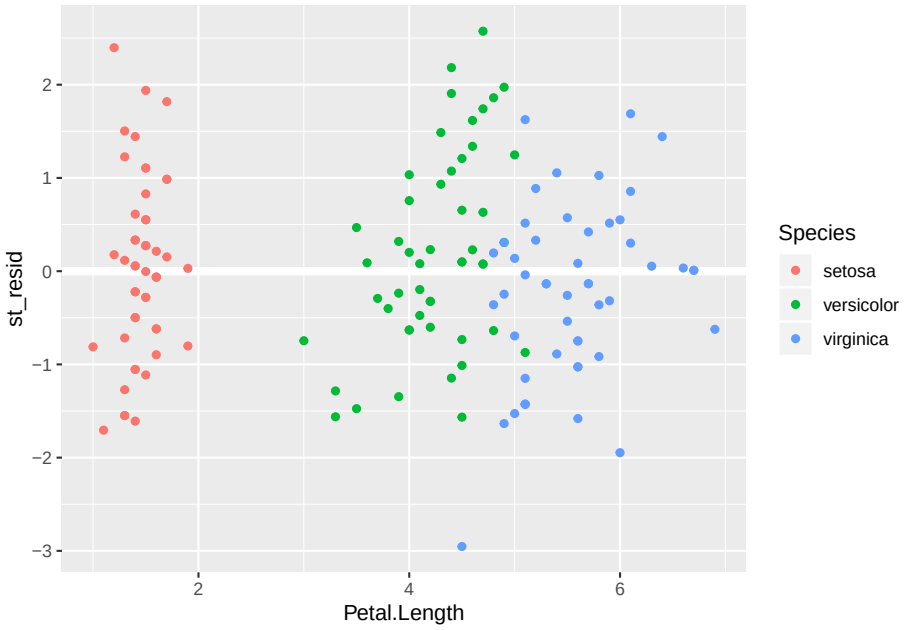


Siit näeme, et I. setosa puhul on residuaalid pigem >0 ja et see mudel töötab paremini I. versicolor ja I. virginica puhul.

Eelnevatelpiltidel on residuaalid algsetes Sepal Length-i mõõtühikutes (cm).

Et otsustada, kas üks või teine residuaal on 0-st piisavalt kaugel, avaldame residuaalid standardhälvete ühikutes (nn Studentized residuals). Residuaalide muster joonisel sellest ei muutu, muutub vaid y-telje tähistus.

```
iris1 <- mutate(iris1, st_resid=resid/sd(resid))
ggplot(iris1, aes(Petal.Length, st_resid, color=Species)) +
  geom_ref_line(h = 0) +
  geom_point()
```



Nüüd näeme I. virginica isendit, mille koha pealt mudel ülehindab 3 standardhälbega.

Kumb mudel on parem?

```
anova(m1, m2)
#> Analysis of Variance Table
#>
#> Model 1: Sepal.Length ~ poly(Petal.Length, 3)
```

```
#> Model 2: Sepal.Length ~ Petal.Length
#>   Res.Df  RSS Df Sum of Sq    F  Pr(>F)
#> 1    146 19.4
#> 2    148 24.5 -2     -5.18 19.5 3.1e-08 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

m1 on selgelt parem (RSS 19 vs 25, p = e-08)

0.12.10 Tukey summa-erinevuse graafik

Seda gaafikutüüpi tuntakse meditsiinis ka Bland-Altmani graafikuna. Te sooritate korraga palju paralleelseid mõõtmisi – näiteks mõõdate mass-spektroskoopiaga 1000 valgu taset. Kui teete seda katset kaks korda (või katse vs. kontroll n korda) ja tahate näha süstemaatilisi erinevusi, siis tasub joonistada summa-erinevuse graafik. See on hea olukordades, kus ei ole vahet, mis läheb x ja mis läheb y teljele (erinevalt regressioonimudelitest ja residuaaliplottidest, kus see on väga tähtis). Meie graafik on x ja y suhtes sümmeetriline.

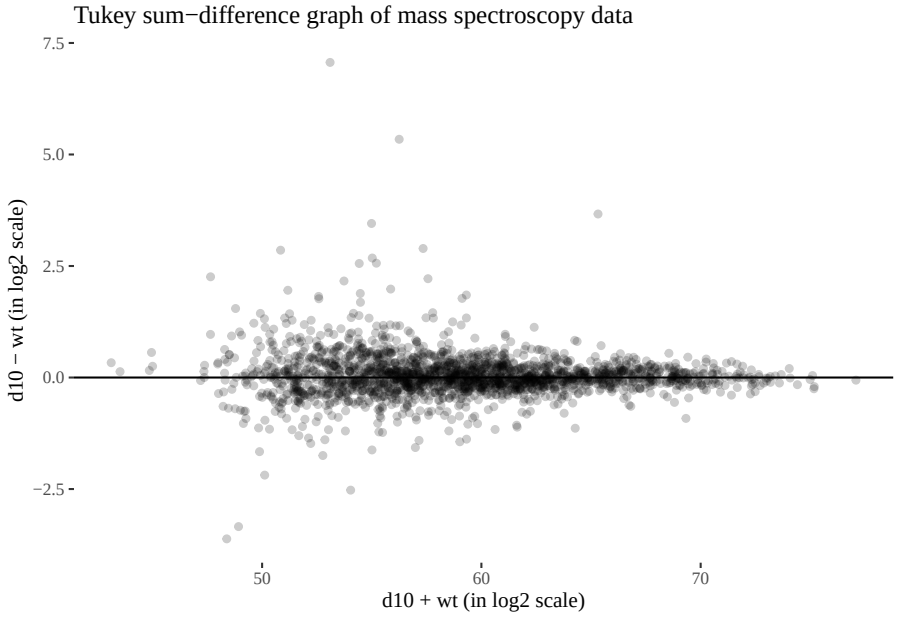
Graafik ise on lihtsalt scatterplot, kus horisontaalsele teljele plotitud x + y väärtused ja vertikaalsele teljele plotitud y - x väärtused. Me lisame ka horisontaalsele teljele 0 - joone, et meil oleks lihtsam oma vaimusilmas efekti suuruste punktipilve tsentreerida.

Näituseks plottime mass spektroskoopia andmed, kus kahel tingimusel (d10 ja wt) on kummagil tehtud kolm iseseisvat katset. Järgneb tabel df_summary2, kus on 2023 valgu tasemete keskväärtused kahel tingimusel, ning Tukey summa-erinevuse graafik

```
head(df_summary2, 3)
#> # A tibble: 3 x 3
#> # Groups:   gene [2,023]
#>   gene    d10    wt
#>   <fct> <dbl> <dbl>
#> 1 aaeR   25.6  25.6
```

```
#> 2 aas    28.8  28.8
#> 3 accA   33.7  33.6
```

```
ggplot(df_summary2, aes(x = d10 + wt, y = d10 - wt)) +
  geom_point(alpha=0.2) +
  geom_hline(yintercept = 0) +
  labs(title="Tukey sum-difference graph of mass spectroscopy data", y="d10 -
  theme_tufte()
```



Meil näha on ilusti tsentreeritud keskmised 3st mõõtmisest kahele tingimusele, kus iga punkt vastab ühele valule. x telg annab suhtelised valgukogused log2 skaalas (selles skaalas on originaallandmed) ja y telg annab efekti suuruse (tingimus 1 miinus tingimus 2).

Me näeme sellelt pildilt, et

1. mida väiksem on valgu kogus, seda suurema tõenäosusega saame tugeva efekti (mis viitab valimivea rollile, eriti suuremate efektide puhul),

2. efektipilv on kenasti nullile tsentreeritud (see näitab, et andmete esialgne töötlus on olnud korralik),
3. enamus valgud ei anna suuri efekte (bioloog ohkab siinkohal kergendatult) ja
4. positiivse märgiga efektid kipuvad olema suuremad, kui negatiivsed efektid (2.5 ühikuline effektisuurus log2 skaalas tähendab 2**2.5 = 5.7 kordset erinevust katse ja kontrolli vahel).

Proteoomikas on praegu populaarsed MA-fraafikud, kus y-teljel (M) on katse vs kontroll erinevus log2 skaalas ja x-teljel (A) on keskmine tase.

`Log2 skaala koos lahutamistehtega on mugav sest üks log2 ühik y-teljel vastab kahekordsele efektile (kaks ühikut neljakordsele, kolm ühikut kaheksakordsele, jne) ja 0 vastab ühekordsele ehk null-efektile.`

0.12.10.1 Vulkaaniplot

Tukey summa-erinevuse graafiku vaene sugulane on vulkaaniplot, kus horisontaalsel teljel on y - x (soovitavalt log2 skaalas) ja vertikaalsel teljel on p väärtused, mis arvutatud kahe grupi võrdluses, kusjuures p väärtused on -log10 skaalas. Vulkaaniplotti tutvustame mitte selle pärast, et seda soovitada, vaid ainult selle tõttu, et seda kasutatakse massiliselt näiteks proteoomika vallas. Vulkaaniplot on tõlgendamise mõttes kolmemõõtmeline ja pigem keeruline, näitlikustades korraga efekti suurust (ES), varieeruvust (sd) ja valimiviga (see sõltub valimi suurusest, aga ka mõõtmisobjekti/valgu tasemest).

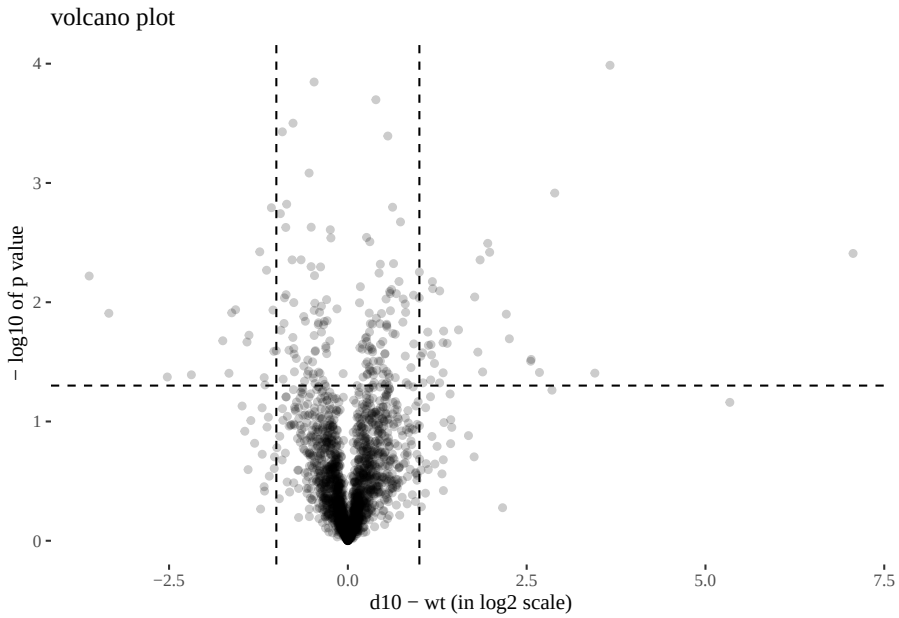
Joonistame vulkaani samade andmete põhjal, mida kasutasime Tukey summa-erinevusgraafiku valmistamieks. Me alustame tabeli "df" ettevalmistamisest: d10_1, d10_2 ja d10_3 on kolm iseseisvat katset ja wt_1, wt_2 ja wt_3 on kolm iseseisvat kontrolli.

```
head(df, 3)
#>   gene d10_1 d10_2 d10_3 wt_1 wt_2 wt_3
```

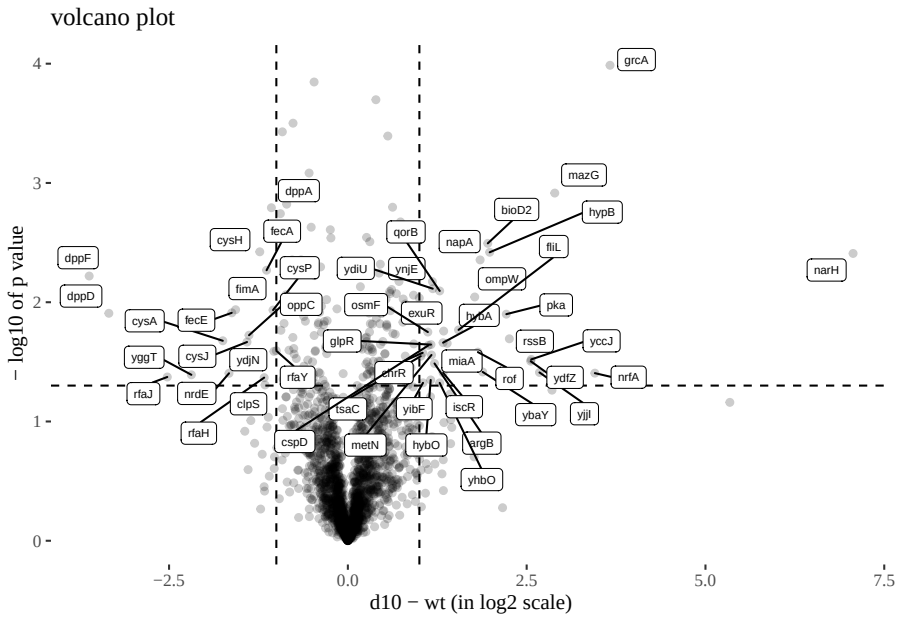
```
#> 1 rpoC  36.3  36.3  36.4 36.2 36.3 36.4
#> 2 rpoB  36.2  36.3  36.2 36.1 36.3 36.3
#> 3 mukB  32.9  33.0  33.2 32.9 33.1 33.3
```

Me lisame tabelile veeru p väärtustega ja veeru effekti suurustega (ES), kasutades apply() funktsiooni sees tavapärast indekseerimist (vt ptk . . .).

```
df_x <- df[2:7] #numeric variables only
#p values
df$p <- apply(df_x, 1, function(x) t.test(x[1:3], x[4:6])$p.value)
#effect sizes (mean experiment - mean control)
df$ES <- apply(df_x, 1, function(x) mean(x[1:3]) - mean(x[4:6]))

plot <- ggplot(df, aes(ES, -log10(p))) +
  geom_point(alpha=0.2) +
  geom_hline(yintercept = -log10(0.05), linetype=2) + #add horizontal line
  geom_vline(xintercept = c(-1, 1), linetype=2)+ #add 2 vertical lines
  labs(x="d10 - wt (in log2 scale)",
       y= "- log10 of p value",
       title="volcano plot")+
  theme_tufte()
plot
```




```
library(ggrepel) #for geom_text_repel()
d <- df %>% filter(ES > 1 | ES < -1) %>% filter(p < 0.05) #data for text lab
plot + geom_label_repel(data=d, aes(label=gene), cex=2)
#alternative: geom_text_repel(data=d, aes(label=gene), cex=2)
```



Sellel pildil markeerib horisontaalne punktiirjoon p = 0.05 ja vertikaalsed punktiirid 2-kordse efektisuuruse (üks ühik log2 skaalal; ühekordne ES võrdub sellel skaalal nulliga). Inimesed, kes paremini ei tea, kipuvad vulkaaniplotti tõlgendama nii: kui punkt (loe: valk) asub horisontaalsest joonest kõrgemal ja ei asu kahe vertikaalse joone vahel, siis on tegu "päris" efektiga. Seevastu inimesed, kes teavad, teavad ka seda, et p väärtuste ühekaupa tõlgendamine ei ole sageli mõistlik. Iga p väärtus koondab endasse informatsiooni kolmest muutujast: valimi suurus (N), varieeruvus (sd) ja efekti suurus (ES = katse - kontroll). Kuigi me saame vulkaaniplotil asuvaid punkte võrreldes ignoreerida valimi suuruse mõju (kuna me teame, et meil on iga punkti taga 3 + 3 mõõtmist), koondab iga p väärtus endasse infot nii ES kui sd kohta viisil, mida me ei oska hästi üksteisest lahutada (siiski, pane tähele, et horisontaalsel teljel on ES). Me teame, et igas punktis on nii ES kui sd mõjutatud valimiveast, mis on kummagi näitaja suhtes teisest sõltumatu. Seega, igal neljandal valgul on valimiveaga seose topeltprobleem: ülehinnatud ES ja samal ajal alahinnatud sd, mis viib oodatust ohtlikult väiksemale p väärtusele.

Lisaks, p väärtuse definitsioonist (p on sinu andmete või neist ekstreemsemate andmete tõenäosus tingimusel, et nullhüpotees kehtib) tuleneb, et kui null hüpotees on tõene (tegelik ES = 0), siis on meil täpselt võrdne tõenäosus saada oma p väärtus ükskõik kuhu nulli ja ühe vahele. Seega, nullhüpoteesi kehtimise korral ei sisalda individuaalne p väärtus mitte mingisugust kasulikku informatsiooni.

Oluline on mõista, et p väärtuse arvutamine toimub nullhüpoteesi all, mis kujutab endast põhimõtteliselt lõpmatu hulga hüpoteetiliste valimite põhjal – mille N = 3 ja sd = valimi sd – arvutatud lõpmatu hulga hüpoteetiliste valimikeskmiste jaotust (iga geeni jaoks eraldi arvutatuna). Seega demonstreerib p väärtus statistikat oma kõige abstraktsemas vormis.

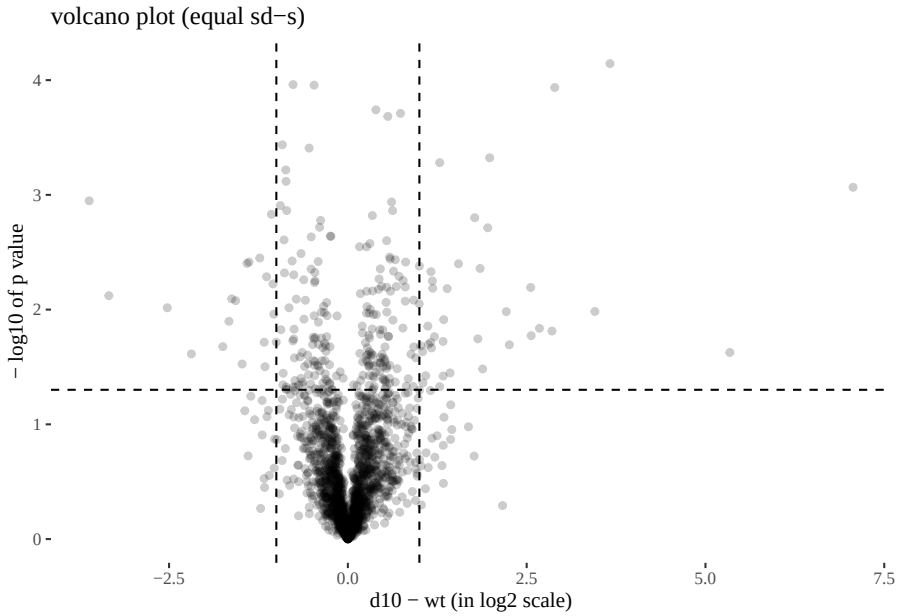
Igal juhul peaks olema siililegi selge, et kui valimi suurus on nõnda väike kui 3, siis valimi põhine sd ega valimi põhine efekti suurus ei ole kuigi usaldusväärsed ennustama tegelikku populatsiooni sd-d ega ES-i!

Kuidas ikkagi vulkaani tõlgendada?

1. Enamus efektisuuruseid < 2 (see on hea)
2. Enamus p väärtusi > 0.05 (ka hea)
3. vulkaan on pisut ebasümmeetriline – meil on rohkem positiivseid effekte, kus d10 > wt (see on teaduslikult oluline uudis)
4. Enamus valke, mille $p < 0.05$, annavad $ES < 2$. (See viitab, et meil on palju katseid, kus iseseisvate katsete vaheline varieeruvus on väga madal.)
5. Oluline osa valke (võib-olla ca 40%), mille $ES > 2$, annavad $p > 0.05$. (Viitab valimivea olulisele osale meie tulemustes.)
6. Enamus kõige suuremate ES-dega valke on üllatavalt kõrge p väärtusega. (Viitab valimivea olulisele osale meie tulemustes.)

Seega ei ole meil ES-i ja p väärtuse vahel selget suhet, kus suurtel efektidel oleks selgelt madalam p väärtus kui väikestel efektidel. Kuna meil pole põhust arvata, et valkudel, millel on suurem ES, on süstemaatiliselt suurem varieeruvus, siis paistab, et meie vulkaan dokumenteerib eelkõige valimivigu, ja seega pigem katse üldist madalat kvaliteeti, kui üksikute efektide "tõelisust". Seega tundub, et tegu on mudavulkaaniga.

Hea küll, joonistame oma vulkaani uuesti p väärtuste põhjal, mis seekord on arvutatud eeldusel, et mõlema grupi (d10 ja wt) varieeruvused on geeni kaupa võrdsed. See tähendab, et kui ES-i arvutamisel on valimi suurus 3 (kolme katse ja kolme kontrolli keskmine), siis sd arvutamisel, mis omakorda läheb p väärtuse arvutamise valemisse, on valimi suurus mõlemale grupile 6.

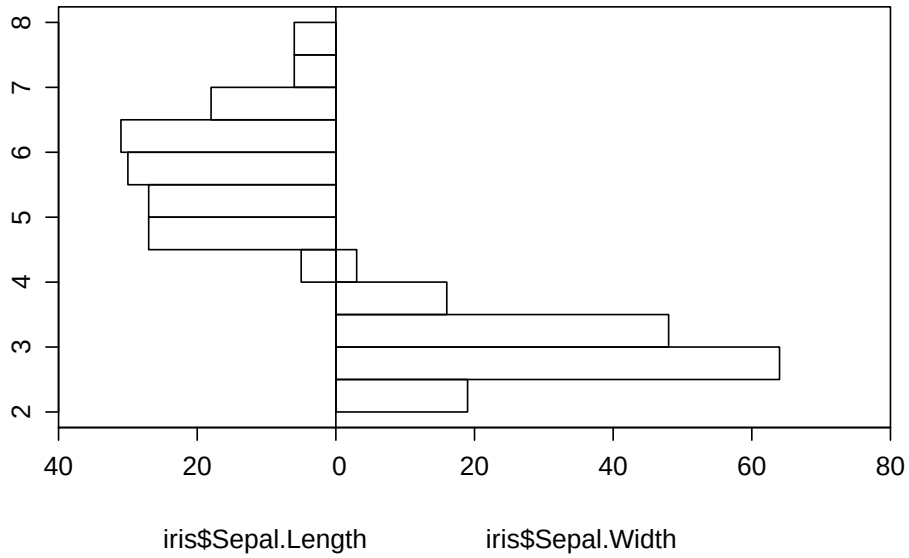


Pilt on küll detailides erinev, aga suures plaanis üsna sarnane eelmisega.

0.12.11 QQ-plot

Kuidas võrrelda kahte jaotust? Kõige lihtsam on joonistada bihistogramm, mis töötab ühtlasi t testi ekslploratoorse analoogina (ei anna ühte numbrit, aga selle eest annab palju parema ülevaate kui t test, kuidas kahe grupi valimid – kuigi mitte tingimata nende taga olevad populatsioonid – tegelikult erinevad).

```
library(Hmisc)
histbackback(iris$Sepal.Length, iris$Sepal.Width)
```



See bihistogramm, mis küll veidi jaburalt võrdleb 3 Irise liigi tolmukate pikkusi ja laiusi, näitab, et kahe grupi keskmised on selgelt erinevad (ülekate peaaegu puudub), aga et ka jaotused ise erinevad omajagu (tolmukate laiuste jaotus on kitsam ja teravam).

Kuidas aga võrrelda oma andmete jaotust teoreetilise jaotusega, näiteks normaaljaotusega? Selleks on parim viis kvantiil-kvantiil plot ehk qq-plot. Kvantiil tähendab lihtsalt andmepunktide osakaalu, mis on väiksemad kui mingi etteantud väärtus. Näiteks kvantiil 0.3 (mis on sama, mis 30s protsentiil) tähistab väärtust, millest 30% kogutud andmeid on väiksemad ja 70% on suuremad.

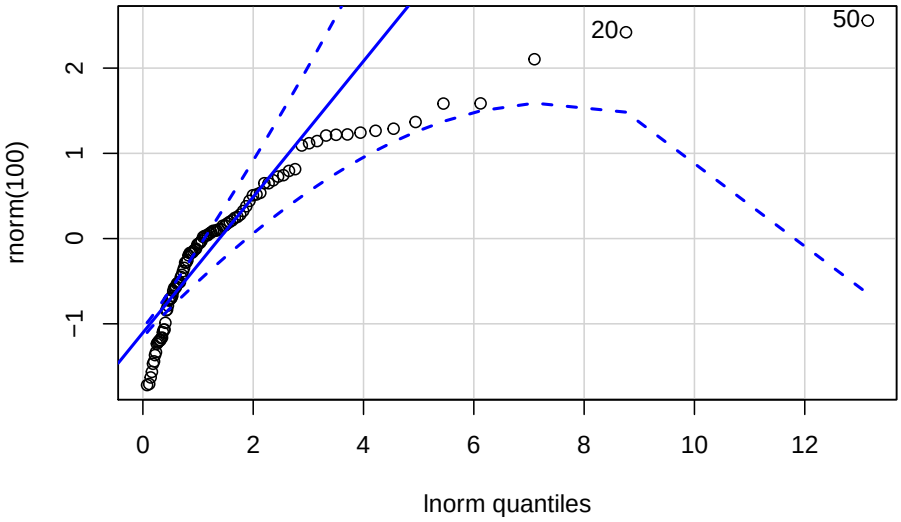
Näiteks standartse normaaljaotuse (mean = 0, sd = 1) 0.5-s kvantiil on 0 ja 0.95-s kvantiil on 1.96.

QQ-plot annab empiiriliste andmete kvantiilid (y teljel) teoreetilise jaotuse kvantiilide vastu (x teljel). Punktide arv graafikul vastab teie andmepunktide arvule. Referentsjoon oma 95% veapiiridega (punased katkendjooned) vastab ideaalsele olukorrale, kus teie andmete jaotus vastab teoreetilisele jaotusele (milleks on enamasti normaaljaotus).

QQ-plot põõrab eelkõige tähelepanu jaotuste sabadele/õlgadele, mis on OK, sest sabad on sageli probleemiks vähimruutude meetodiga regressioonil. Kui me võrdleme normaaljaotusega paremale kiivas jaotust (*positive skew*), siis tulevad punktid mõlemas servas kõrgemale kui referentsjoon. See juhtub näit Chi-ruut jaotuse korral. Kui meil on paksude õlgadega sümmeetriline jaotus, nagu studenti t, siis tulevad ülemise otsa punktid kõrgemale ja alumise otsa punktid madalamale kui referentsjoon.

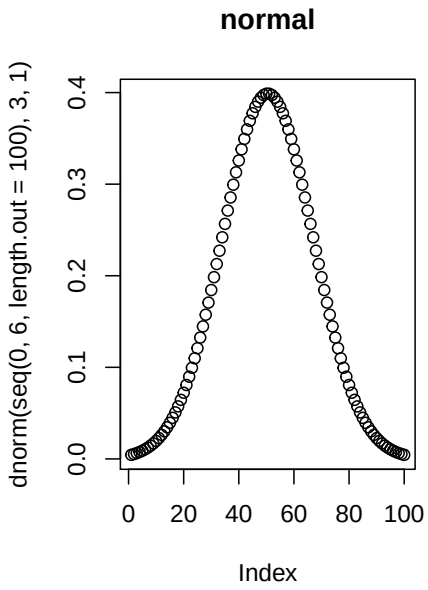
Kõigepealt demonstreerime siiski olukorda, kus meie andmed on normaaljaotusega ja me võrdleme neid teoreetilise lognormaaljaotuse vastu. Võrdluseks saame kasutada kõiki R-s defineeritud jaotusi (distribution = "jaotus").

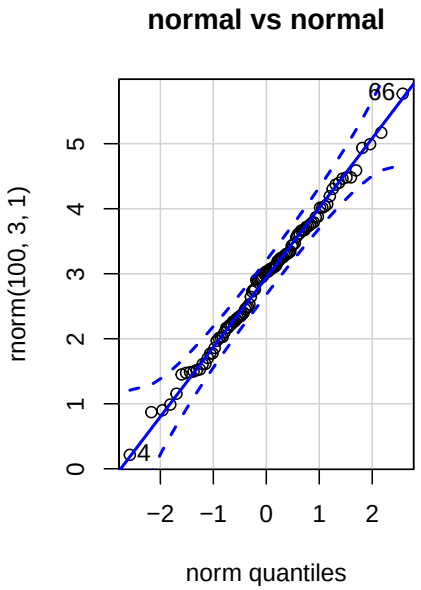
```
library(car)
qqPlot(rnorm(100), distribution = "lnorm")
#> [1] 50 20
```



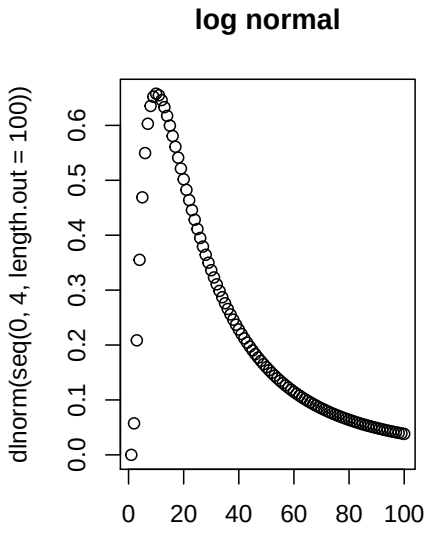
Proovime erinevaid jaotusi normaaljaotuse vastu. Kõigepealt jaotused:

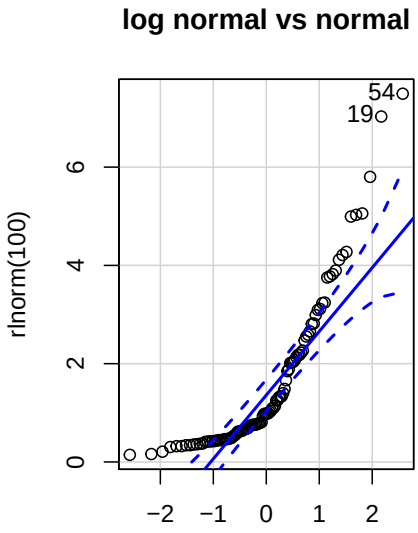
```
par(mfrow = c(1, 2))
plot(dnorm(seq(0,6, length.out = 100), 3, 1), main = "normal")
qqPlot(rnorm(100, 3, 1), main = "normal vs normal") #default on vrdls normal
#> [1] 66  4
par(mfrow=c(1,1))
```



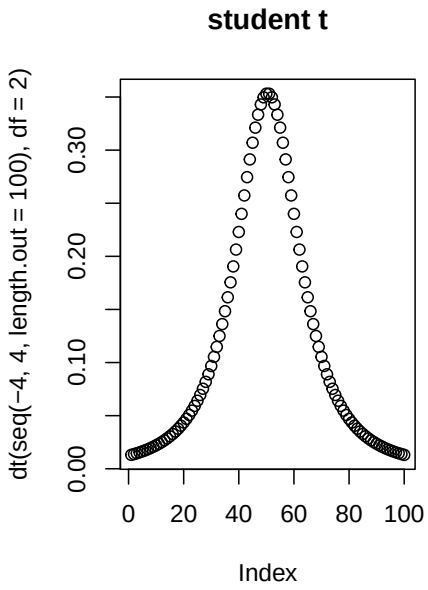


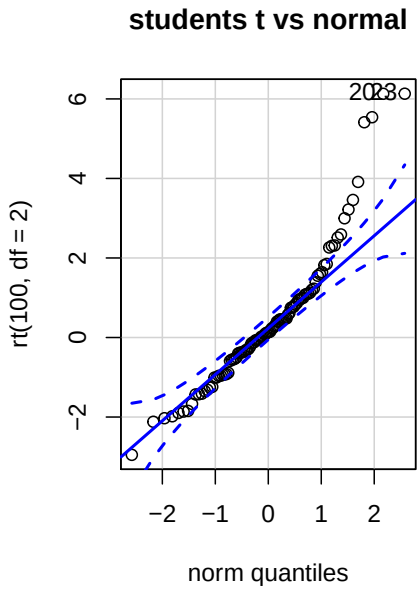

```
par(mfrow = c(1, 2))
plot(dlnorm(seq(0,4, length.out = 100)), main = "log normal")
qqPlot(rlnorm(100), main = "log normal vs normal")
#> [1] 54 19
par(mfrow=c(1,1))
```



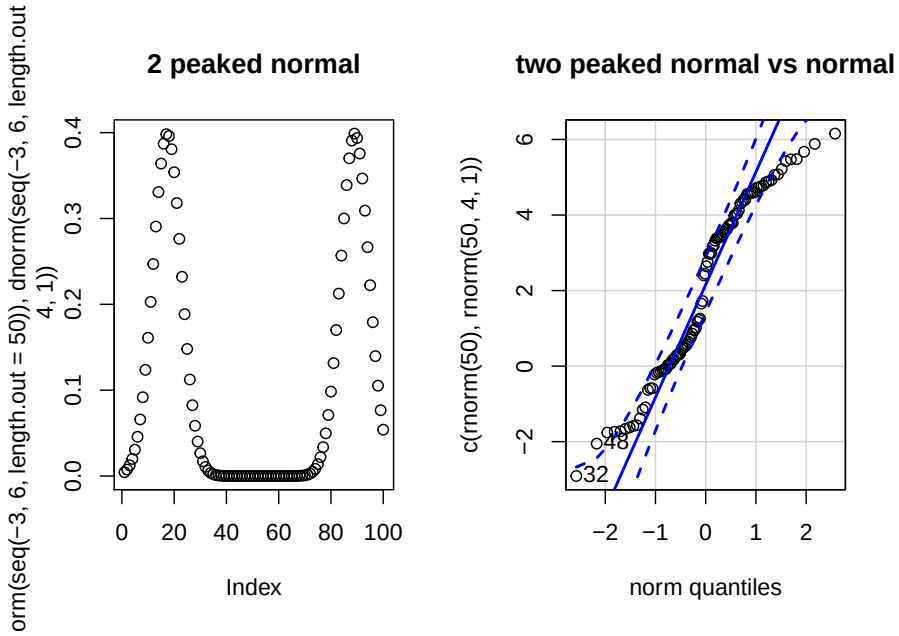



```
par(mfrow = c(1, 2))
plot(dt(seq(-4,4, length.out = 100), df=2), main = "student t")
qqPlot(rt(100, df=2), main = "students t vs normal")
#> [1] 23 20
par(mfrow=c(1,1))
```

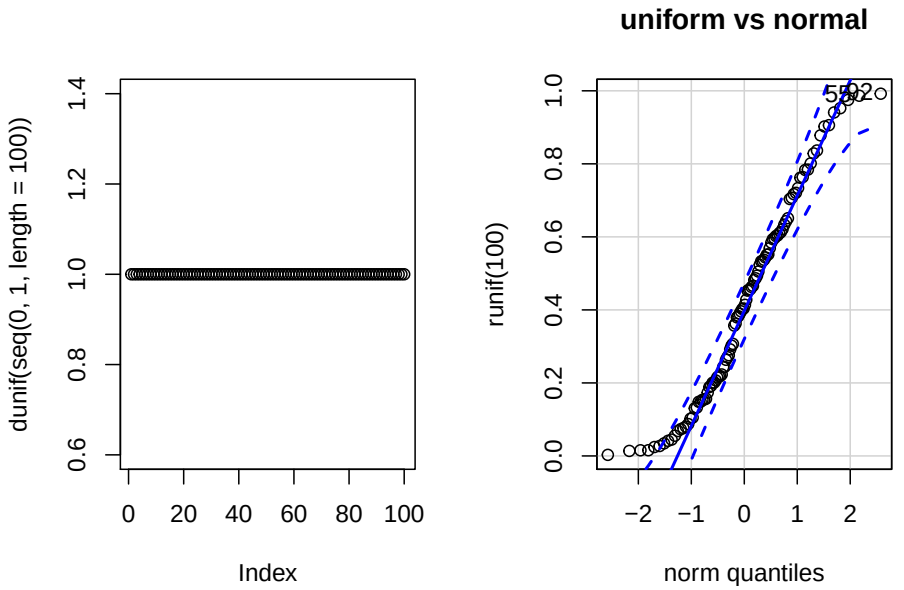




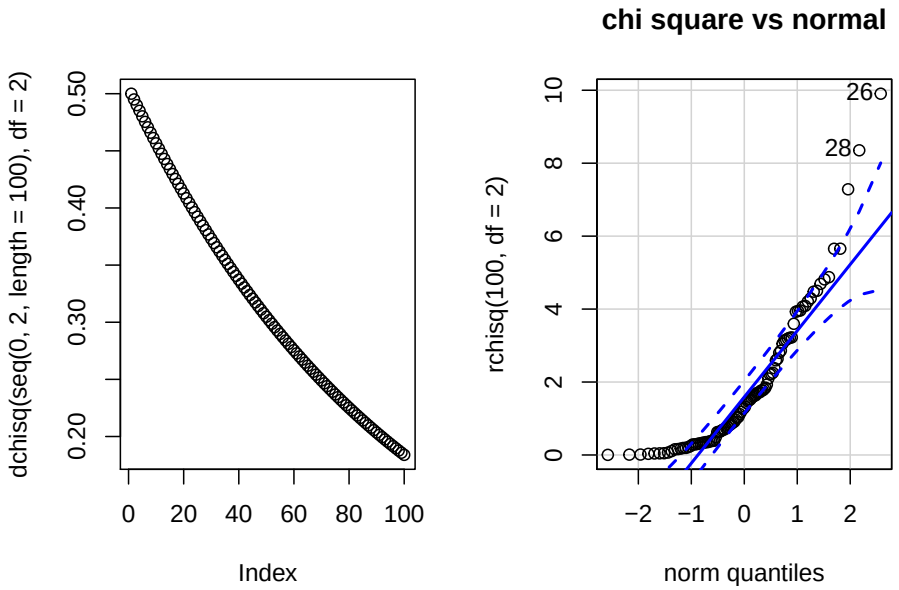

```
par(mfrow = c(1, 2))
plot(c(dnorm(seq(-3,6, length.out = 50)), dnorm(seq(-3,6, length.out = 50),
qqPlot(c(rnorm(50), rnorm(50, 4,1)), main = "two peaked normal vs normal")
#> [1] 32 48
par(mfrow=c(1,1))
```



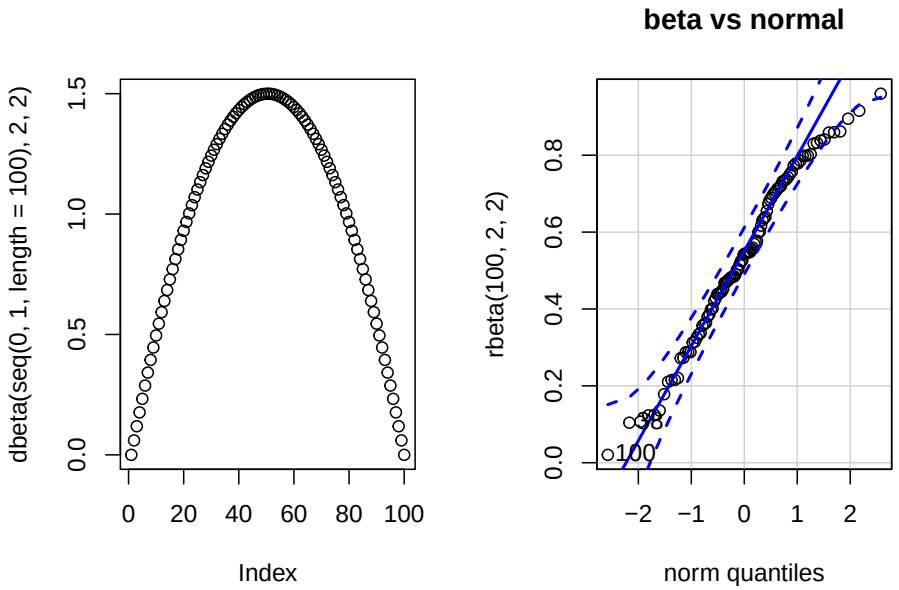

```
par(mfrow = c(1, 2))
plot(dunif(seq(0, 1, length=100)))
qqPlot(runif(100), main = "uniform vs normal") #default on vrdls normaaljao
#> [1] 92 55
par(mfrow=c(1,1))
```




```
par(mfrow = c(1, 2))
plot(dchisq(seq(0, 2, length=100), df=2))
qqPlot(rchisq(100, df=2), main = "chi square vs normal")
#> [1] 26 28
par(mfrow=c(1,1))
```

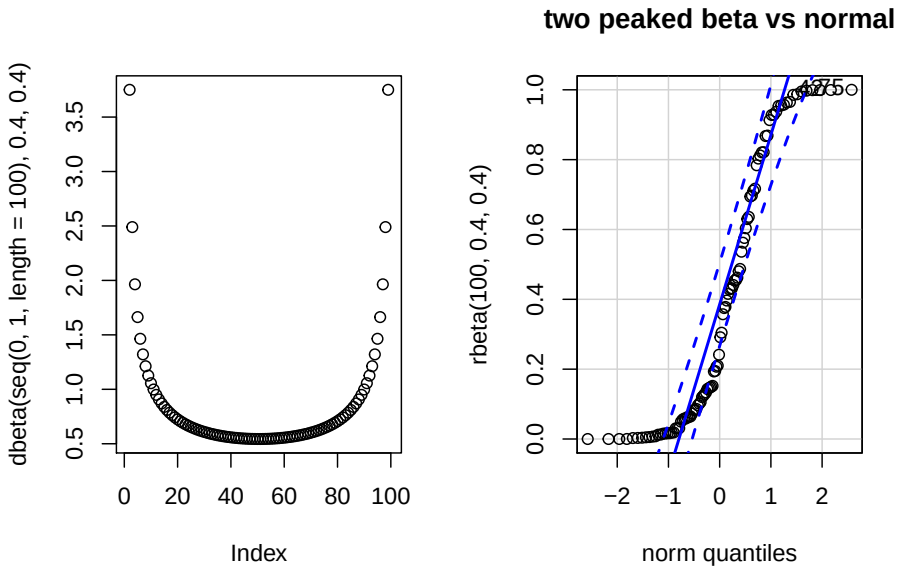



```
par(mfrow = c(1, 2))
plot(dbeta(seq(0, 1, length=100), 2, 2))
qqPlot(rbeta(100, 2, 2), main = "beta vs normal")
#> [1] 100  38
par(mfrow=c(1,1))
```



Proovime veel erinevaid jaotusi normaaljaotuse vastu. Kõigepealt jaotused:

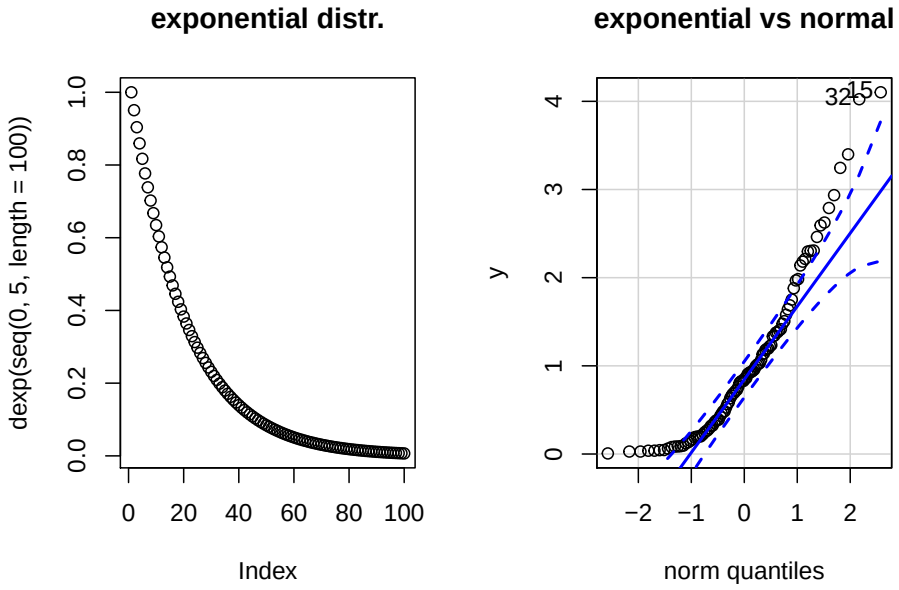
```
#> [1] 75 42
```



Nagu näha, beta jaotus, mis on normaaljaotusest palju laiem, on qq-plotil sellest halvasti eristatav. Erinevus on väga madalatel ja väga kõrgetel kvantiilidel (jaotuste otstes).

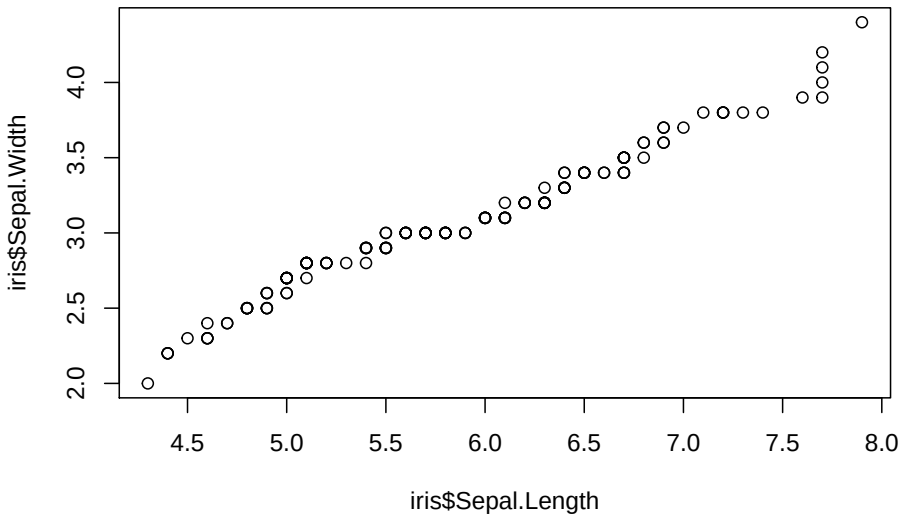
Ja exponentsiaalse jaotuse korral:

```
y <- rexp(100)
par(mfrow=c(1,2))
plot(dexp(seq(0, 5, length=100)), main="exponential distr.")
qqPlot(y, main = "exponential vs normal")
#> [1] 15 32
par(mfrow=c(1,1))
```



QQ-plotiga saab võrrelda ka kahte empiirilist jaotust, näiteks Irise liikide tolmukate pikkuste ja tolmukate laiuste jaotusi (vt ka peatüki algusest bihistogrammi). Selle meetodi oluline eelis on, et võrreldavad jaotused võivad olla erineva suurusega (N-ga). Siin kasutame base::R qqplot() funktsiooni.

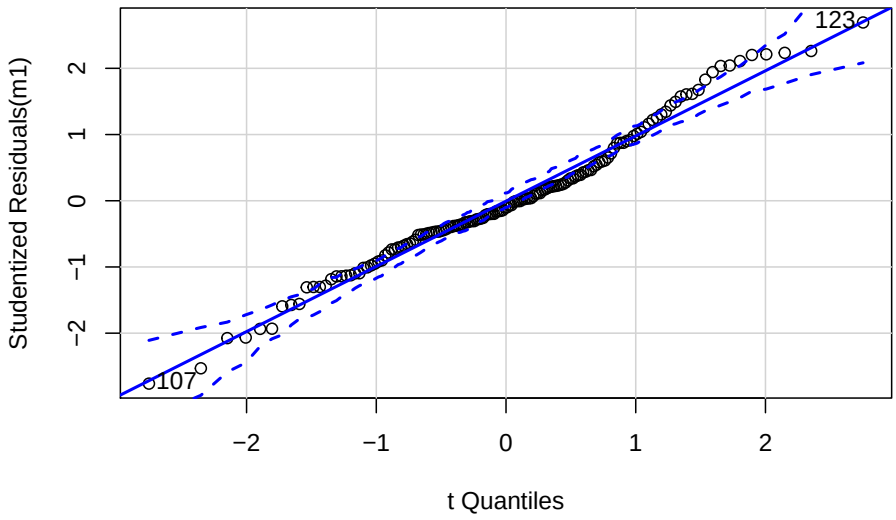
```
qqplot(iris$Sepal.Length, iris$Sepal.Width)
```



Nagu näha, erinevad jaotused põhiliselt kõrgemates kvantiilides, kus tolmuka pikkus > 7.5 ja tolmuka laius > 3.6.

car::qqPlot saab kasutada ka lineaarse regressiooni normaalsuseelduse kontrollimiseks. Kui ennustatav y-muutuja on normaaljaotusega, siis peaksid residuaalid olema normaaljaotusega (keskväärtus $= 0$). Selle normaalsuse määramiseks plotitakse standardiseeritud residuaalid teoreetiliste normalsete kvantiilide vastu. Selleks anname qqPlot() funktsiooni lm mudeliobjekti

```
m1 <- lm(Sepal.Length~ Sepal.Width + Petal.Width, data = iris)
qqPlot(m1)
#> [1] 107 123
```

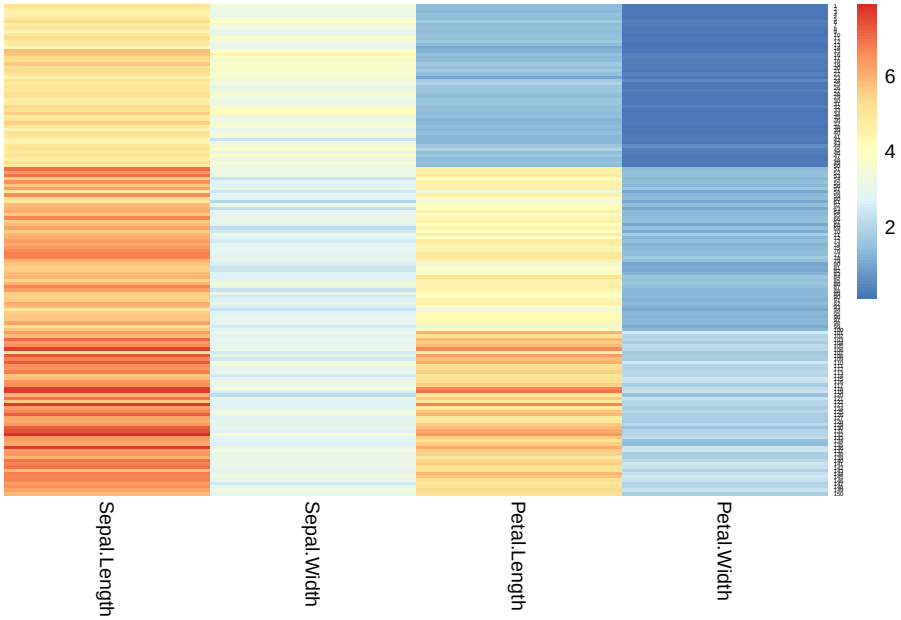


0.12.12 Heat map

Heat map asendab tabelis numbrid värvidega, muutes nii keerulised tabelid kiiremini haaratavateks. Samas, inimese aju ei ole kuigi edukas värvitoone pidevate muutujate numbrilisteks väärtusteks tagasi konverteerima, mis tähendab, et heat map võimaldab lugejal kiiresti haarata mustreid andmetes, aga ei võimalda teha täpseid võrdlusi tabeli üksikute lahtrite vahel.

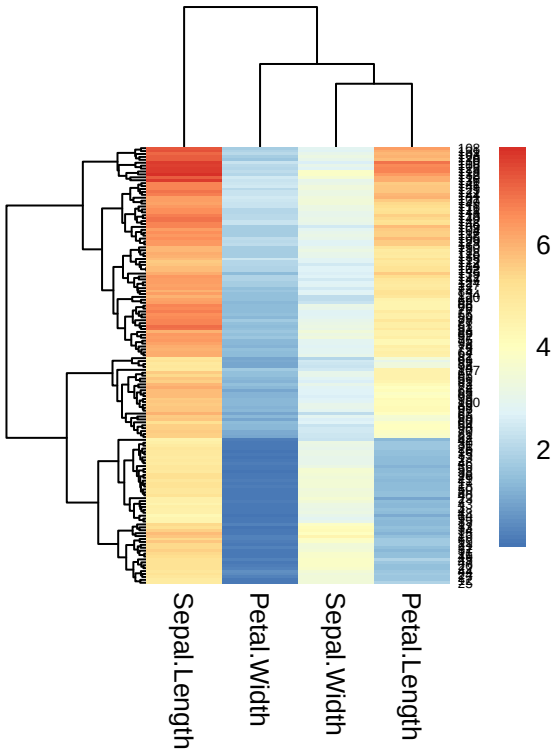
Kõigepealt lihtne heat map, kus irise tabeli numbrilistes veergudes on asendatud arvud värvitoonidega, aga tabeli üldine kuju ei muutu:

```
library(pheatmap)
pheatmap(iris[1:4], fontsize_row = 3, cluster_cols = FALSE, cluster_rows = F
```



Et andmetes leiduvad mustrid paremini välja paistaksid, tasub heat mapil andmed ümber paigutada kasutades näiteks hierarhilist klassifitseerimist. Seega lisanduvad heat mapile ka dendrogrammid.

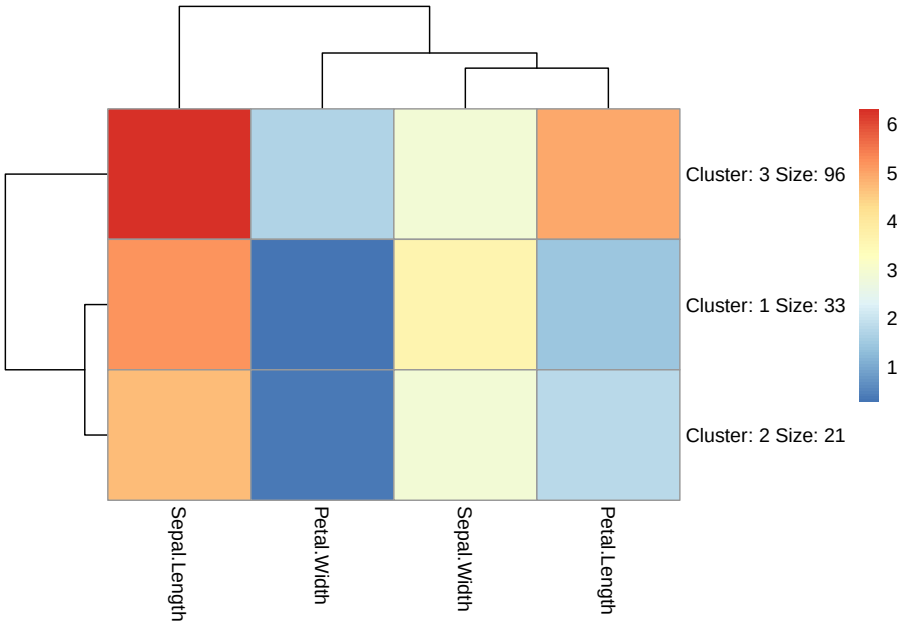
```
pheatmap(iris[1:4], fontsize_row = 5)
```



Irise tabel on nüüd mõlemas dimensioonis sorteeritud hierarhilise klasterdamise läbi, mida omakorda kajastab 2 dendrogrammi (üks kummagis tabeli dimensioonis). Dendrogramm mõõdab erinevust/sarnasust. Dendrogrammi lugemist tuleb alustada selle harunenud otstest. Kõigepealt jagab dendrogramm vaatlused paaridesse, misjärel hakkab järk-järgult lähimaid paare klastritesse ühendama kuni lõpuks kõik vaatlused on ühendatud ainsasse klastrisse. Dendrogrammi harude pikkused markeerivad selle kriteeriumstatistiku väärtust, mille järgi dendrogramm koostati (siin on palju võimalusi, aga kõige levinum on eukleidiline kaugus). Igal juhul, mida pikem haru, seda suuremat erinevust see kajastab. Me võime igal tasemel tõmmata läbi dendrogrammi joone ja saada just nii palju klastreid, kui palju harunemisi jääb sellest joonest ülespoole. Dendrogrammi harud võivad vabalt pöörelda oma vartel, ilma et see dendrogrammi topograafiat muudaks – seega on joonisel olev dendrogrammi kuju lihtsalt üks juhuslikult fikseeritud olek paljudest.

Nüüd me ütleme, et me tahame oma irise liigid ajada täpselt kolme k-means klastrisse. NB! k-means klustrid on arvutatud hoopis teisel viisil kui eelmisel joonisel olevad hierarhilised klastrid. Siin alustame k = 3 tsentroidist, assigneerime iga andmepunkti oma lähimale tsentroidile, arvutame tsentroidid ümber kui klastri kõikide andmepunktide keskmised, assigneerime uuesti kõik andmepunktid oma tsentroidile ja kordame seda tsüklit näiteks 10 korda.

```
a <- pheatmap(iris[1:4], kmeans_k = 3)
```

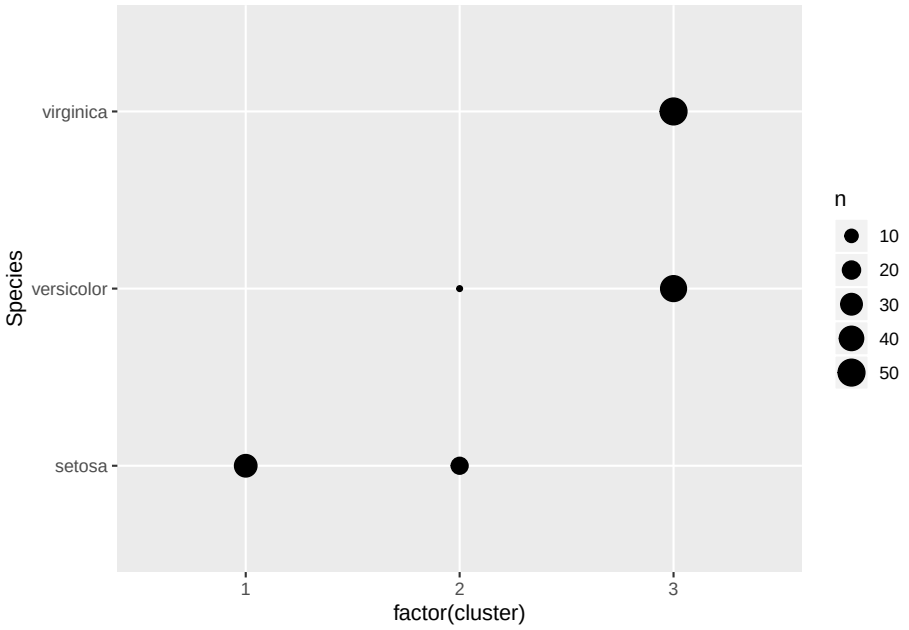


Lisame klastrid irise tabelisse ja vaatame, kui hästi klastrid tabavad kolme irise liiki:

```
iris$cluster <- a$kmeans$cluster
table(iris$Species, iris$cluster)
#>
#>                1  2  3
#>   setosa      33 17  0
#>   versicolor   0  4 46
#>   virginica    0  0 50
```

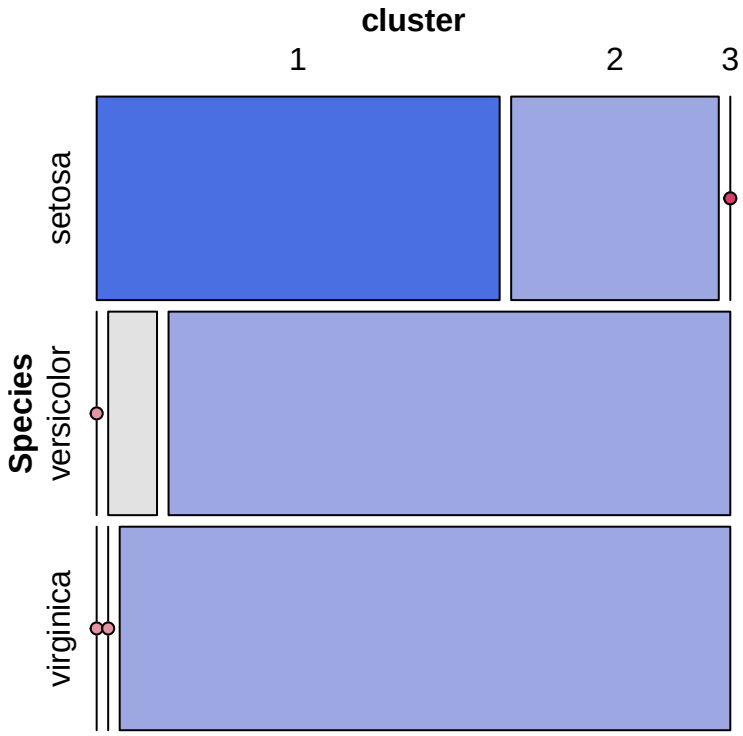
Ja sama graafiliselt:

```
ggplot(iris, aes(factor(cluster), Species)) + geom_count()
```



Või alternatiivina esitatuna tulpade pikkustena mosaiikgraafikul (tulpade pikkusi on lihtsam võrrelda kui pindalasid eelmisel graafikul):

```
library(vcd)
iris_x <- iris %>% select(Species, cluster)
iris_x$cluster <- as.factor(iris_x$cluster)
mosaic(~Species + cluster, data= iris_x, shade=T, legend=FALSE)
```

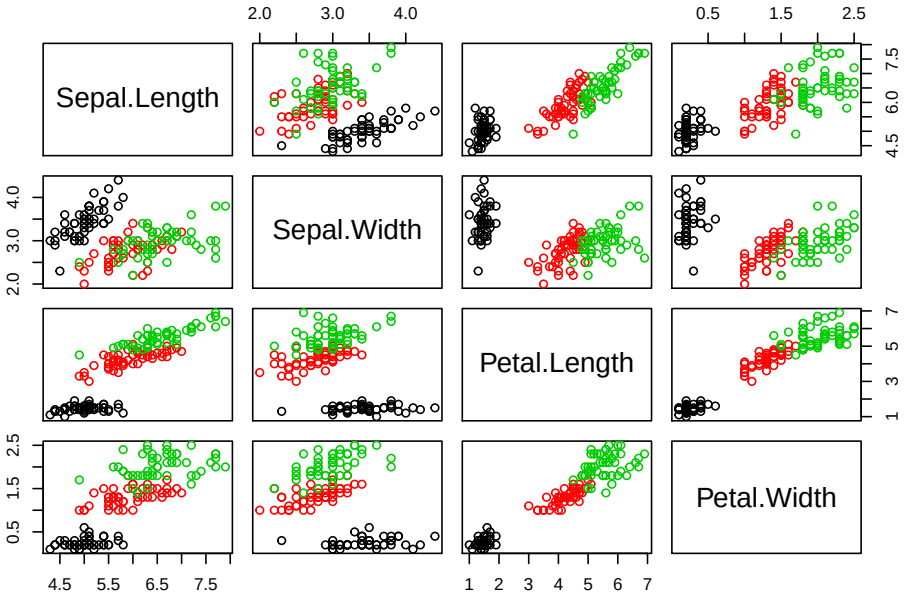


0.12.12.1 Korrelatsioonimaatriksid heat mapina

Heat map on ka hea viis visualiseerida korrelatsioonimaatrikseid.

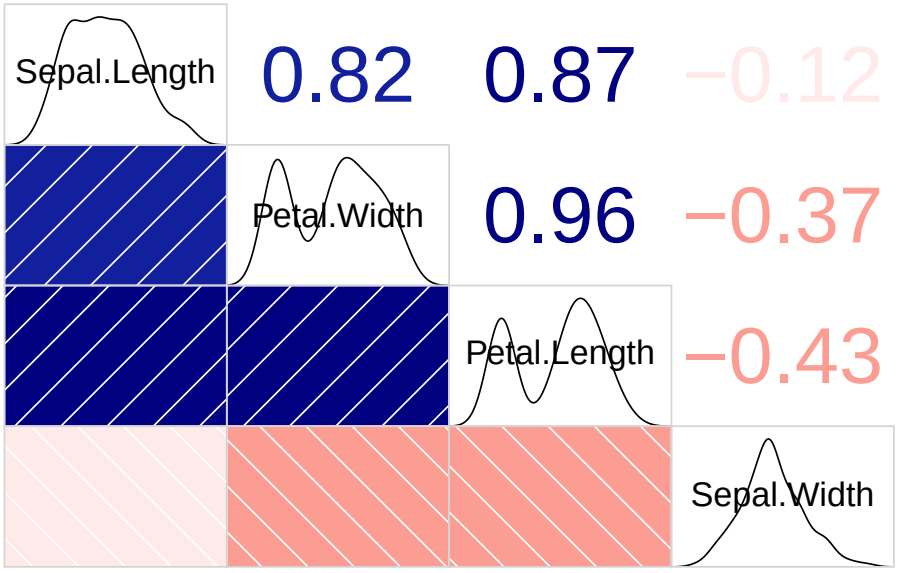
Kõigepealt tavaline scatterplot maatriks.

```
plot(iris[1:4], col=iris$Species)
```



Seejärel korrogramm, kus diagonaalist allpool tähistavad värvid korrelatsioone ja diagonaalist ülalpool on samad korrelatsioonid numbritega. Me sorteerime mustrite parema nägemise huvides ka andmetulbad ümber (order=TRUE), seekord kasutades selleks peakomponent analüüsi (PCA).

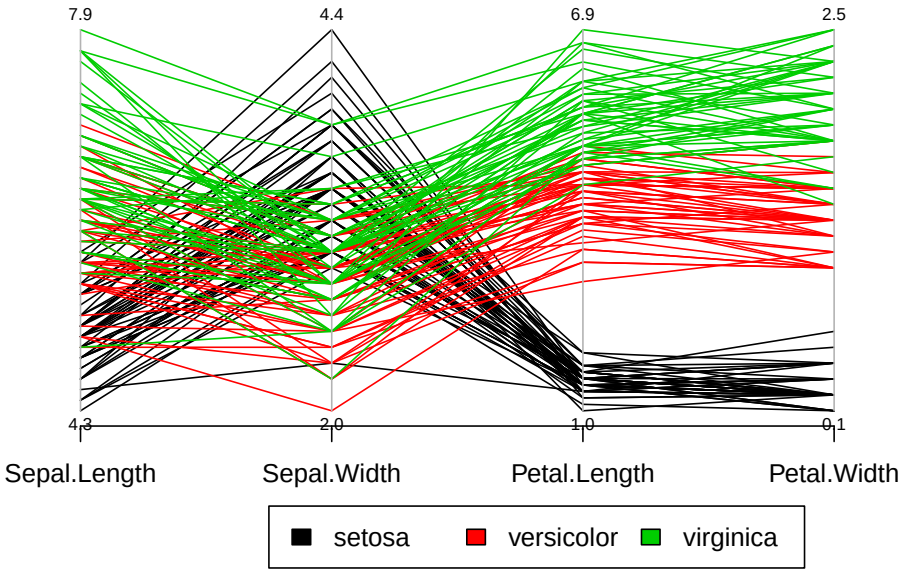
```
library(corrgram)
corrgram(iris[1:4], order = TRUE, lower.panel=corrgram::panel.shade,
         upper.panel=panel.cor, diag.panel=panel.density)
```



0.12.12.2 Paraleelkoordinaatgraafik

Alternatiivne võimalus on scatterplot maatriksile on joonistada grrafik läbi paraleelsete koordinaatide.

```
library(MASS)
parcoord(iris[1:4], col = iris$Species, var.label = TRUE, lwd = 1)
par(xpd = TRUE)
legend(x = 1.75, y = -.25, cex = 1,
   legend = as.character(levels(iris$Species)),
    fill = unique(iris$Species), horiz = TRUE)
```



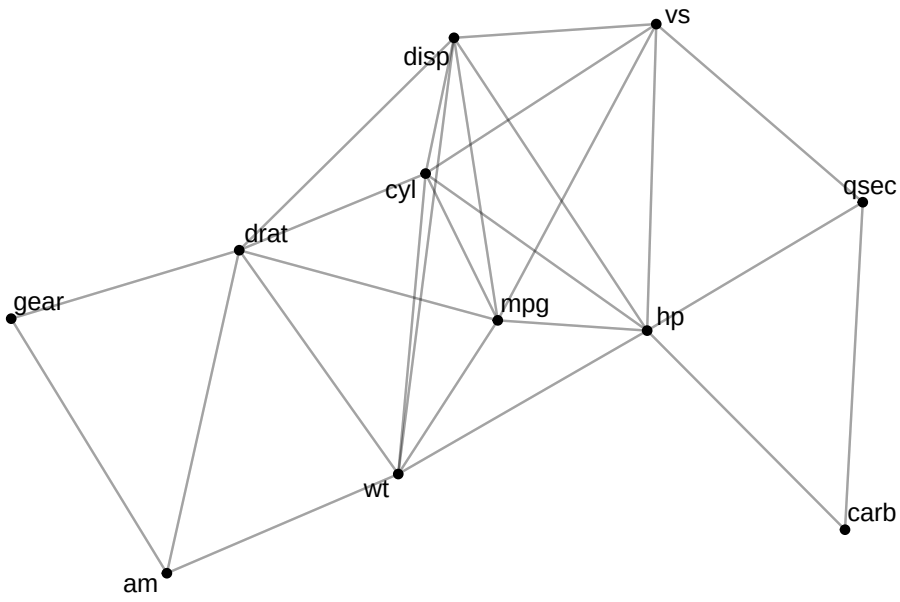
Siit näeme, kuidas Petal length ja Petal width on parim viis, et setosat teistest eristada.

0.12.12.3 Korrelatsioonid võrgustikuna

Võrgustik koosneb sõlmedest ja nende vahel olevatest servadest (nodes and edges). Meie eesmärk on joonisel näidata sõlmedena ainult need muutujaid, millel esineb mingist meie poolt etteantud numbrist suurem korrelatsioon mõne teise muutujaga. Korrelatsioone endid tähistavad võrgu servad. Järgnevasse voogu, kuhu sisestame kogu mtcars tabeli, lähevad ainult numbrilised muutujad (faktormuutujad tuleb tabelist välja visata).

```
library(corrr)
library(igraph)
library(ggraph)

tidy_cors <- mtcars %>%
  correlate() %>%
  stretch()
```

```
# Convert correlations stronger than some value
# to an undirected graph object
graph_cors <- tidy_cors %>%
  filter(abs(r) > 0.6) %>%
  graph_from_data_frame(directed = FALSE)

# Plot
ggraph(graph_cors) +
  geom_edge_link(alpha=0.2) +
  geom_node_point() +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_graph()
```

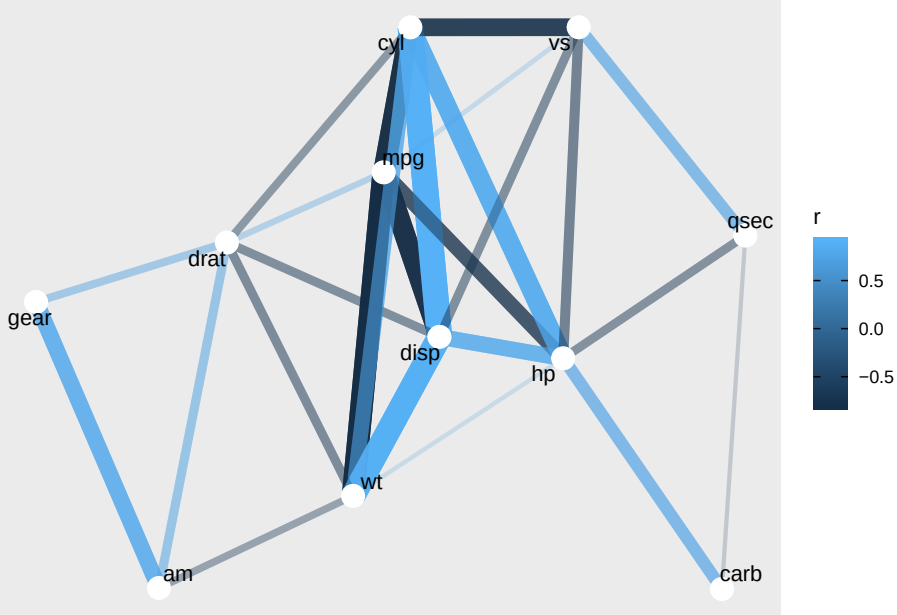


Siin on tabeli mtcars kõik korrelatsioonid, mis on suuremad kui absoluutväärtus 0.6-st.

Kenam (ja informatiivsem) versioon eelmisest on

```
ggraph(graph_cors) +
  geom_edge_link(aes(edge_alpha = abs(r), edge_width = abs(r), color = r)) +
  guides(edge_alpha = "none", edge_width = "none") +
```

```
  geom_node_point(color = "white", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE)
```



Nipp! Kui teile ei meeldi võrgustiku üldine kuju, jooksutage koodi uuesti – vähegi keerulisemad võrgud tulevad iga kord ise kujuga (säilitades siiski sõlmede ja servade kontaktid).

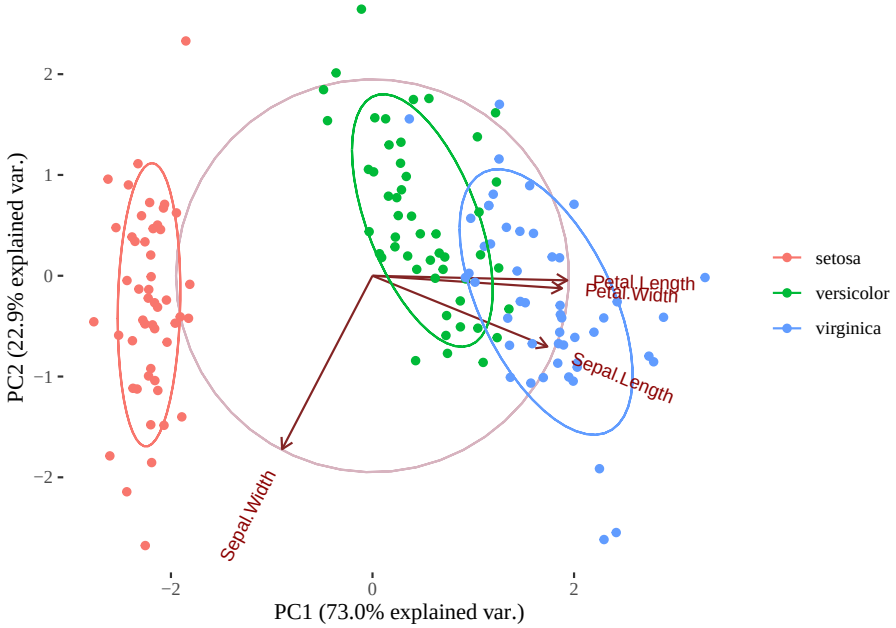
0.12.13 Biplot ja peakomponentanalüüs

Kui teil on andmetes rohkem dimensioone, kui te jõuate plottida, siis peakomponentanalüüs (PCA) on üks võimalus multidimensionaalseid andmeid kahedimensionaalsena joonistada. PCA on lineaarne meetod, mis püüab omavahel korrelleeritud muutujad aesendada uute muutujatega, mis oleks võimalikult vähe korrelleeritud. PCA joonise teljed (peakomponent 1 ja peakomponent 2) on valitud nii, et need oleks üksteisega võimalikult vähe korreleeritud ja samas säiliks võimalikult suur osa andmete algsest multidimensionaalsest varieeruvusest. Eesmärk on saavutada 2D (või 3D) muster, mis oleks võimalikult lähedane algse multi-D mustriga. Seega, PCA

projitseerib multidimensionaalse andmemustri 2D pinnale viisil, mis püüab säilitada maksimaalse osa algsest andmete varieeruvusest. PCA teeb seda, kasutades lineaarset additiivset mudelit.

See analüüs on mõistlik ainult siis, kui andmed varieeruvad kõige rohkem suunas, mis on ka teaduslikult oluline (ei ole juhuslik müra) ja muutujate vahel ei ole mittelineaarseid interaktsioone (muutujad on sõltumatud). Te ei tea kunagi ette, kas ja millal PCAst võib kasu olla reaalsete mustrite leidmisel – seega tuleks PCA tõlgendamisega olla pigem ettevaatlik, sest inimaju on suuteline mustreid nägema ka seal, kus neid ei ole. Lisaks, isegi kui PCAs ilmuv muster on ehtne, on PCA dimensioone sageli palju raskem teaduslikult tõlgendada kui originaalseid muutujaid.

```
library(ggbiplot)
ir.species <- iris[, 5]
ir.pca <- prcomp(iris[,1:4], center = TRUE, scale = TRUE)
g <- ggbiplot(ir.pca, obs.scale = 1, var.scale = 1,
              groups = iris$Species, ellipse = TRUE,
              circle = TRUE)
g <- g + scale_color_discrete(name = '') + theme_tufte()
print(g)
```



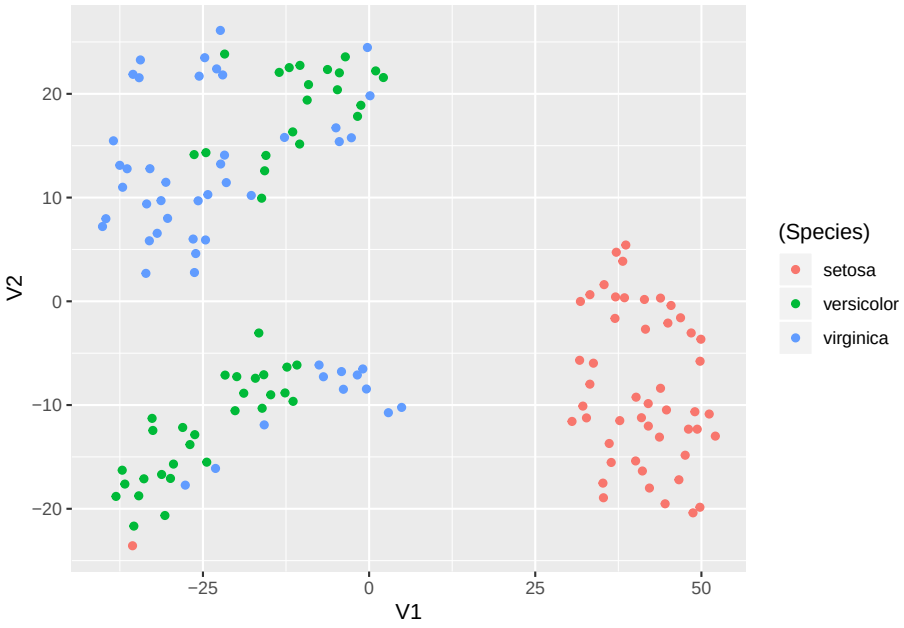
Seega taandasime 4D andmestiku 2D-sse, säilitades seejuurse suure osa algsest andmete varieeruvusest (esimene pekomponent sisaldab 73% algsest varieeruvusest ja 2. peakomponent 23%). Punkidena on näidatud irise isendid, mis on värvitud liigi järgi, ja lisaks on antud vektorid, mis näitavad, millised algsetest muutujatest korreleeruvad millise peakomponendiga. Siit näeme, et Petal.Length, Petal.Width ja Sepal.Width-i varieeruvus kajastub valdavas enamuses PC1 teljel (vektorid on PC1 teljega enam-vähem paralleelsed) ja et Sepal-Width muutuja varieeruvus kajastub suures osas PC2 teljel.

0.12.13.1 t-sne

Populaarne mittelineaarne viis multidimensionaalsete andmete 2D-sse redutseerimiseks on t-sne (t-Distributed Stochastic Neighbor Embedding), mis vaatab andmeid lokaalselt (mitte kogu andmeruumi tervikuna). Parameeter perplexity tuunitakse kasutaja poolt ja see määrab tasakaalu, millega algoritm vaatab andmeid lokaalselt ja globaalselt. Väiksem perplexity tõstab lokaalse vaatluse osakaalu. Perplexity annab hinnangu, mitu lähimat naabrit igal andmepunk-

til võiks olla. Üldiselt on soovitus jooksutada t-sne algoritmi mitu korda varieerides perplexity-d 5 ja 50 vahel. Enne selle meetodi kasutamist loe kindlasti https://distill.pub/2016/misread-tsne/

```
library(tsne)
ts <- tsne(iris[1:4], perplexity = 10)
ts <- as_tibble(ts)#output is a table of 2D t-sne coordinates
ism1 <- bind_cols(iris, ts)
ggplot(ism1, aes(x = V1, y = V2, color = (Species))) +
  geom_point()
```



0.13 Üldised jooniste printsiibid

Kõigepealt, joonise tüüp peaks vastama joonise sõnumile - sõnasta järeldus, mille sa oma joonise pealt teed. Kas mõni teine joonise tüüp illustreerib seda järeldust paremini/on ühemõttelisem/kiiremini loetav?

1. optimaalne data ink/non-data ink suhe.
2. Joonisel on rõhutatud/silmapaistvad need data ink-i elemendid, mis on ka teaduslike tulemuste seisukohalt kõige olulisemad.
3. data ink on silmatorkavam kui näiteks teljed ja abijooned
4. Kasuta nooli, et juhtida tähelepanu olulistele tulemustele (suuna joonise lugeja tähelepanu).
5. kui võimalik, pane andmete tähised (kirjad) otse joonisele - niikaua kui see muudab joonise kiiremini loetavaks.
6. Ära kuhja joonist üle - keskendu oluliste tulemuste visualiseerimisele.

0.14 Tidyverse

Tidyverse on osa R-i ökosüsteemist, kus kehtivad omad reeglid. Tidyverse raamatukogud lähtuvad ühtsest filosoofiast ja töötavad hästi koos. Tidyverse algab andmetabeli struktuurist ja selle funktsioonid võtavad reeglina sisse õige struktuuriga tibble ja väljastavad samuti tibble, mis sobib hästi järgmise tidyverse funktsiooni sisendiks. Seega on tidyverse hästi sobiv läbi torude `%>%` laskmiseks. Tidyverse-ga sobib hästi kokku ka ggplot2 graafikasüsteem.

Laadime tidyverse metapaketi raamatukogud:

```
library(tidyverse)
```

Nagu näha laaditakse tidyverse raamatukoguga 8 paketti:

- ggplot2
- purrr
- tibble
- dplyr

- tidyr
- stringr
- readr
- forcats
- tibble pakett sisaldab tidyverse-spetsiifilise andmeraami (data_frame) loomiseks ja manipuleerimiseks vajalike funktsioone. Erinevalt baas R-i andmeraamist (data.frame) iseloomustab tibble-t vaikimisi prindifunktsioon, kus vaikimisi näidataksegi ainult tabeli peast 10 esimest rida. Oluliseks erinevuseks on ka **list tulpade toetus** (data.frame tulbad saavad olla ainult vektorid). List tulbad võimaldavad andmeraami paigutada kõige erinevamaid objekte: näiteks vektoreid, andmeraame, lineaarseid mudeleid ja valgeid puudleid. Lisaks ei ole tibble tabelitel veerunimesid ja veidraid tulbanimesid ei muudeta vaikimisi/automaatselt.
- tidyr pakett sisaldab eelkõige funktsioone tibble-de kuju muutmiseks laiast formaadist pikka ja tagasi.
- readr paketi funktsioonid vastutavad andmete impordi eest tekstipõhistest failidest lähtuvalt tidyverse reeglitest ja asendavad vastavad baas R-i funktsioonid.
- purrr pakett sisaldab funktsioone töötamaks listidega ja asendavad baas R-i apply perekonna funktsioone.
- dplyr pakett sisaldab põhilisi andmetöötlusverbe.
- stringr ja forcats paketid sisaldavad vastavalt tekstipõhiste ja kategooriliste andmetega töötamise funktsioone.

0.14.1 Tidy tabeli struktuur

- **väärtus** (*value*) — ühe mõõtmise tulemus (183 cm)
- **muutuja** (*variable*) — see, mida sa mõõdad (pikkus) või faktor (sex)

- **andmepunkt** (*observation*) — väärtused, mis mõõdeti samal katsetingimusel (1. subjekti pikkus ja kaal 3h ajapunktis)
- **vaatlusühik** (*unit of measurement*) — keda mõõdeti (subjekt nr)
- **vaatlusühiku tüüp** — inimene, hiir, jt

  vaatlusühiku tüüp = tabel

  muutuja = veerg

  andmepunkt = rida

  vaatlusühikute koodid on kõik koos ühes veerus

Veergude järjekord tabelis on 1. vaatlusühik, 2. faktor, mis annab katse-kontrolli erisuse, 3. kõik see, mida otse ei mõõdetud (sex, batch nr, etc.), 4. numbritega veerud (iga muutuja kohta üks veerg)

```
#> # A tibble: 2 x 6
#>   subject drug    sex    time length weigth
#>   <chr>   <chr>   <chr> <dbl>  <dbl>  <dbl>
#> 1 1       exp     F         3    168     88
#> 2 2       placebo M         3    176     91
```

Nii näeb välja tidy tibble. Kõik analüüsil vajalikud parameetrid tuleks siia tabelisse veeru kaupa sisse tuua. Näiteks, kui mõõtmised on sooritatud erinevates keskustes erinevate inimeste poolt kasutades sama ravimi erinevaid preparaate, oleks hea siia veel 3 veergu lisada (center, experimenter, batch).

0.14.1.1 Tabeli dimensioonide muutmine (pikk ja lai formaat)

Väga oluline osa tidyverses töötamisest on tabelite pika ja laia formaadi vahel viimine.

See on laias formaadis tabel df, mis ei ole tidy

```
#> # A tibble: 3 x 5
#>   subject sex   control experiment_1 experiment_2
#>   <chr>   <chr>   <dbl>        <dbl>        <dbl>
#> 1 Tim     M          23           34           40
#> 2 Ann     F          31           38           42
```

```
#> 3 Jill    F          30          36          44
```

Kõigepealt pikka formaati. key ja value argumendid on ainult uute veergude nimetamiseks, oluline on 3:ncol(dat) argument, mis ütleb, et "kogu kokku veerud alates 3. veerust". Alternatiivne viis seda öelda: c(-subject, -sex).

```
dat_lng <- gather(dat, key = experiment, value = value, 3:ncol(dat))
# df_l3<-df %>% gather(experiment, value, 3:ncol(df)) works as well.
#df_l4<-df %>% gather(experiment, value, c(-subject, -sex)) works as well
dat_lng
#> # A tibble: 9 x 4
#>   subject sex   experiment   value
#>   <chr>   <chr> <chr>        <dbl>
#> 1 Tim     M     control         23
#> 2 Ann     F     control         31
#> 3 Jill    F     control         30
#> 4 Tim     M     experiment_1    34
#> 5 Ann     F     experiment_1    38
#> 6 Jill    F     experiment_1    36
#> # ... with 3 more rows
```

Paneme selle tagasi algsesse laia formaati: ?spread

```
spread(dat_lng, key = experiment, value = value)
#> # A tibble: 3 x 5
#>   subject sex   control experiment_1 experiment_2
#>   <chr>   <chr>   <dbl>        <dbl>        <dbl>
#> 1 Ann     F          31           38           42
#> 2 Jill    F          30           36           44
#> 3 Tim     M          23           34           40
```

key viitab pika tabeli veerule, mille väärtustest tulevad laias tabelis uute veergude nimed. value viitab pika tabeli veerule, kust võetakse arvud, mis uues laias tabelis uute veergude vahel laiali jagatakse.

0.14.1.2 Tibble transpose — read veergudeks ja vastupidi

```
dat <- tibble(a = c("tim", "tom", "jill"), b1 = c(1, 2, 3), b2 = c(4, 5, 6))
dat
#> # A tibble: 3 x 3
#>   a         b1    b2
#>   <chr> <dbl> <dbl>
#> 1 tim       1     4
#> 2 tom       2     5
#> 3 jill      3     6
```

Me kasutame selleks maatriksarvutuse funktsiooni t() — transpose. See võtab sisse ainult numbrilisi veerge, seega anname talle ette df miinus 1. veerg, mille sisu me konverteerime uue tablei veerunimedeks.

```
dat1 <- t(dat[,-1])
colnames(dat1) <- dat$a
dat1
#>    tim tom jill
#> b1   1   2    3
#> b2   4   5    6
```

0.14.2 dplyr ja selle viis verbi

Need tuleb teil omale pähe ajada sest nende 5 verbiga (pluss gather ja spread) saab lihtsalt teha 90% andmeväänamisest, mida teil elus ette tuleb. NB! Check the data wrangling cheatsheet and dplyr help for further details. dplyr laetakse koos tidyverse-ga automaatselt teie workspace-i.

0.14.2.1 `select()` columns

`select()` selects, renames, and re-orders columns.

Select columns from sex to value:

```
iris
select(iris, Petal.Length:Species)
select(iris, -(Petal.Length:Species)) #selects everything, except those col
```

To select 3 columns and rename *subject* to *SUBJ* and put liik as the 1st col:

```
select(iris, liik = Species, Sepal.Length, Sepal.Width) %>% dplyr::as_data_f
#> Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind th
#> This warning is displayed once per session.
#> # A tibble: 150 x 3
#>   liik   Sepal.Length Sepal.Width
#>   <fct>         <dbl>       <dbl>
#> 1 setosa          5.1         3.5
#> 2 setosa          4.9         3
#> 3 setosa          4.7         3.2
#> 4 setosa          4.6         3.1
#> 5 setosa          5           3.6
#> 6 setosa          5.4         3.9
#> # ... with 144 more rows
```

To select all cols, except sex and value, and rename the *subject* col:

```
select(iris, -Sepal.Length, -Sepal.Width, liik = Species)
```

helper functions you can use within select():

`starts_with("abc")`: matches names that begin with "abc."

`ends_with("xyz")`: matches names that end with "xyz."

`contains("ijk")`: matches names that contain "ijk."

`matches("(.)\\1")`: selects variables that match a regular expression. This one matches any variables that contain repeated characters.

`num_range("x", 1:3)` matches x1, x2 and x3.

```
iris <- as_tibble(iris)
select(iris, starts_with("Petal"))
```

```
#> # A tibble: 150 x 2
#>   Petal.Length Petal.Width
#>          <dbl>       <dbl>
#> 1          1.4         0.2
#> 2          1.4         0.2
#> 3          1.3         0.2
#> 4          1.5         0.2
#> 5          1.4         0.2
#> 6          1.7         0.4
#> # ... with 144 more rows
select(iris, ends_with("Width"))
#> # A tibble: 150 x 2
#>   Sepal.Width Petal.Width
#>         <dbl>       <dbl>
#> 1         3.5         0.2
#> 2         3           0.2
#> 3         3.2         0.2
#> 4         3.1         0.2
#> 5         3.6         0.2
#> 6         3.9         0.4
#> # ... with 144 more rows

# Move Species variable to the front
select(iris, Species, everything())
#> # A tibble: 150 x 5
#>   Species Sepal.Length Sepal.Width Petal.Length
#>   <fct>          <dbl>       <dbl>        <dbl>
#> 1 setosa           5.1         3.5          1.4
#> 2 setosa           4.9         3            1.4
#> 3 setosa           4.7         3.2          1.3
#> 4 setosa           4.6         3.1          1.5
#> 5 setosa           5           3.6          1.4
#> 6 setosa           5.4         3.9          1.7
#> # ... with 144 more rows, and 1 more variable:
#> #   Petal.Width <dbl>

dat <- as.data.frame(matrix(runif(100), nrow = 10))
```

```
dat <- tbl_df(dat[c(3, 4, 7, 1, 9, 8, 5, 2, 6, 10)])
select(dat, V9:V6)
#> # A tibble: 10 x 5
#>       V9    V8    V5     V2     V6
#>    <dbl> <dbl> <dbl>  <dbl>  <dbl>
#> 1 0.492 0.495 0.167 0.0974 0.430
#> 2 0.658 0.691 0.776 0.896  0.150
#> 3 0.665 0.320 0.126 0.163  0.119
#> 4 0.613 0.865 0.933 0.253  0.0728
#> 5 0.193 0.542 0.255 0.554  0.0989
#> 6 0.593 0.330 0.169 0.0989 0.391
#> # ... with 4 more rows
select(dat, num_range("V", 9:6))
#> # A tibble: 10 x 4
#>       V9    V8     V7     V6
#>    <dbl> <dbl>  <dbl>  <dbl>
#> 1 0.492 0.495 0.936  0.430
#> 2 0.658 0.691 0.212  0.150
#> 3 0.665 0.320 0.0853 0.119
#> 4 0.613 0.865 0.677  0.0728
#> 5 0.193 0.542 0.473  0.0989
#> 6 0.593 0.330 0.875  0.391
#> # ... with 4 more rows

# Drop variables with -
select(iris, -starts_with("Petal"))
#> # A tibble: 150 x 3
#>    Sepal.Length Sepal.Width Species
#>           <dbl>       <dbl> <fct>
#> 1           5.1         3.5 setosa
#> 2           4.9         3   setosa
#> 3           4.7         3.2 setosa
#> 4           4.6         3.1 setosa
#> 5           5           3.6 setosa
#> 6           5.4         3.9 setosa
#> # ... with 144 more rows
```

```
# Renaming ----------------------------------------
# select() keeps only the variables you specify
# rename() keeps all variables
rename(iris, petal_length = Petal.Length)
#> # A tibble: 150 x 5
#>   Sepal.Length Sepal.Width petal_length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
#> 1          5.1         3.5          1.4         0.2
#> 2          4.9         3            1.4         0.2
#> 3          4.7         3.2          1.3         0.2
#> 4          4.6         3.1          1.5         0.2
#> 5          5           3.6          1.4         0.2
#> 6          5.4         3.9          1.7         0.4
#> # ... with 144 more rows, and 1 more variable:
#> #   Species <fct>
```

0.14.2.2 `filter()` rows

Keep rows in Iris that have Species level "setosa" **and** Sepal.Length value <4.5.

```
filter(iris, Species=="setosa" & Sepal.Length < 4.5)
#> # A tibble: 4 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
#> 1          4.4         2.9          1.4         0.2
#> 2          4.3         3            1.1         0.1
#> 3          4.4         3            1.3         0.2
#> 4          4.4         3.2          1.3         0.2
#> # ... with 1 more variable: Species <fct>
```

Keep rows in Iris that have Species level "setosa" **or** Sepal.Length value <4.5.

```
filter(iris, Species=="setosa" | Sepal.Length < 4.5)
#> # A tibble: 50 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
```

```
#> 1          5.1         3.5          1.4         0.2
#> 2          4.9         3            1.4         0.2
#> 3          4.7         3.2          1.3         0.2
#> 4          4.6         3.1          1.5         0.2
#> 5          5           3.6          1.4         0.2
#> 6          5.4         3.9          1.7         0.4
#> # ... with 44 more rows, and 1 more variable:
#> #   Species <fct>
```

Keep rows in Iris that have Species level "not setosa" **or** Sepal.Length value <4.5.

```
filter(iris, Species !="setosa" | Sepal.Length < 4.5)
#> # A tibble: 104 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
#> 1          4.4         2.9          1.4         0.2
#> 2          4.3         3            1.1         0.1
#> 3          4.4         3            1.3         0.2
#> 4          4.4         3.2          1.3         0.2
#> 5          7           3.2          4.7         1.4
#> 6          6.4         3.2          4.5         1.5
#> # ... with 98 more rows, and 1 more variable:
#> #   Species <fct>
```

Kui tahame samast veerust filtreerida "või" ehk "|" abil mitu väärtust, on meil valida kahe samaväärse variandi vahel (tegelikult töötab 2. variant ka ühe väärtuse korral)

```
filter(iris, Species =="setosa" | Species =="versicolor")
filter(iris, Species %in% c("setosa", "versicolor") )
```

Nagu näha, 2. variant on oluliselt lühem.

Filtreerime regulaarekspressiooniga: read, kus Species algab v tähega

```
library(stringr)
filter(iris, str_detect(Species, "^v"))
```

```
#> # A tibble: 100 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
#> 1          7           3.2          4.7         1.4
#> 2          6.4         3.2          4.5         1.5
#> 3          6.9         3.1          4.9         1.5
#> 4          5.5         2.3          4           1.3
#> 5          6.5         2.8          4.6         1.5
#> 6          5.7         2.8          4.5         1.3
#> # ... with 94 more rows, and 1 more variable:
#> #   Species <fct>
```

eemalda NAdega read kahe veeru põhjal

```
filter(flights, !is.na(dep_delay), !is.na(arr_delay))
```

0.14.2.3 `summarise()`

Mitu rida summeeritakse üheks väärtuseks veeru kaupa. Kõigepealt summeerime kogu tabeli nii, et saame (1) keskmise Sepal-length-i, (2) standardhälbe samast, (3) tabeli ridade arvu ja (4) mitu erinevat Species-t on tabelis

```
summarise(iris,
          MEAN = mean(Sepal.Length),
          SD = sd(Sepal.Length),
          N = n(),
          n_species = n_distinct(Species))
#> # A tibble: 1 x 4
#>    MEAN    SD     N n_species
#>   <dbl> <dbl> <int>     <int>
#> 1  5.84 0.828   150         3
```

`n()` loeb üles, mitu väärtust läks selle summary statistic-u arvutusse,

`n_distinct()` loeb üles, mitu unikaalset väärtust läks samasse arvutusse.

Summarise on kasulikum, kui teda kasutada koos järgmise verbi, group_by-ga.

0.14.2.4 `group_by()`

`group_by()` grupeerib väärtused, nii et neid saab grupi kaupa summeerida või muteerida. Näiteks grupeerides Species kaupa, saame arvutada summaarsed statistikud igale liigile

```
iris_grouped <- group_by(iris, Species)
summarise(iris_grouped,
          MEAN = mean(Sepal.Length),
          SD = sd(Sepal.Length),
          N = n(),
          n_species = n_distinct(Species))
#> # A tibble: 3 x 5
#>   Species     MEAN    SD     N n_species
#>   <fct>      <dbl> <dbl> <int>     <int>
#> 1 setosa      5.01 0.352    50         1
#> 2 versicolor  5.94 0.516    50         1
#> 3 virginica   6.59 0.636    50         1
```

`summarise()` argumendid on indentsed eelmise näitega aga tulemus ei ole. Siin me rakendame summarise verbi mitte kogu tabelile, vaid 3-le virtuaalsele tabelile, mis on saadud algsest tabelist.

`group_by()`-le saab anda järjest mitu grupeerivat muutujat. Siis ta grupeerib kõigepealt neist esimese järgi, seejärel lõõb saadud grupid omakorda lahku teise argumendi järgi ja nii edasi kuni teie poolt antud argumendid otsa saavad.

pro tip Kui tahad summaarseid statistikuid algse pika tabeli sisse uute veergudena (igale grupeeringu tasemele vastavad siis summeeriva statistiku identsed väärtused rea kaupa), kasuta peale `group_by()` verbi `mutate()`, mitte `summarise()`.

```
mutate(iris_grouped,
       MEAN = mean(Sepal.Length),
       SD = sd(Sepal.Length))
```

```
#> # A tibble: 150 x 7
#> # Groups:   Species [3]
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>          <dbl>       <dbl>        <dbl>       <dbl>
#> 1          5.1         3.5          1.4         0.2
#> 2          4.9         3            1.4         0.2
#> 3          4.7         3.2          1.3         0.2
#> 4          4.6         3.1          1.5         0.2
#> 5          5           3.6          1.4         0.2
#> 6          5.4         3.9          1.7         0.4
#> # ... with 144 more rows, and 3 more variables:
#> #   Species <fct>, MEAN <dbl>, SD <dbl>
```

Anna igast grupist 3 kõrgeimat väärtust või 2 madalaimat väärtust. Samad numbrid erinevates ridades antakse kõik - selle pärast on meil tabelis rohkem ridu.

```
top_n(iris_grouped, 3, Sepal.Length)
top_n(iris_grouped, -2, Sepal.Length)
```

0.14.2.5 `mutate()`

Mutate põhikasutus on siiski uute veergude tekitamine, mis võtavad endale inputi rea kaupa. Seega tabeli ridade arv ei muutu.

tranformeeri tabeli "df" veerg "value" uueks veeruks "log_value", kus on log2-transformeeritud numbrid: `df %>% mutate(log_value = log2(value))`.

Uues veerus on vana veeru numbritest lahutatud konstant (näiteks vana veeru keskväärtus): `df %>% mutate(centered_value = value - mean(value) )`.

Mutate() lisab veerge ja `transmute()` kaotab ühtlasi ära vanad veerud

Uus veerg log-väärtustega, mis põhineb "value" veerul, millele anname nime "log_value".

```
mutate(dat_lng, log_value = log(value))
#> # A tibble: 9 x 5
#>   subject sex   experiment   value log_value
#>   <chr>   <chr> <chr>        <dbl>     <dbl>
#> 1 Tim     M     control         23      3.14
#> 2 Ann     F     control         31      3.43
#> 3 Jill    F     control         30      3.40
#> 4 Tim     M     experiment_1    34      3.53
#> 5 Ann     F     experiment_1    38      3.64
#> 6 Jill    F     experiment_1    36      3.58
#> # ... with 3 more rows
```

Sama transmute() kasutades. Me säilitame lisaks "subject" veeru ja säilitame ning nimetame ümber "sex" veeru.

```
transmute(dat_lng, subject, gender = sex, log_value = log(value))
#> # A tibble: 9 x 3
#>   subject gender log_value
#>   <chr>   <chr>      <dbl>
#> 1 Tim     M           3.14
#> 2 Ann     F           3.43
#> 3 Jill    F           3.40
#> 4 Tim     M           3.53
#> 5 Ann     F           3.64
#> 6 Jill    F           3.58
#> # ... with 3 more rows
```

Selekteerime veerud "year" kuni "day", veerud, mille nimed lõppevad stringiga "delay", veerud "distance" ja "air_time". Seejärel loome uue veerud "gain" (kasutades selleks arr_delay ja dep_delay andmeid rea kaupa), "hours" (air_time jagatud konstandiga) ja "gain_per_hour":

```
flights_sml <- select(flights,
                      year:day,
                      ends_with("delay"),
                      distance,
                      air_time) %>%
```

```
  mutate(gain = arr_delay - dep_delay,
    hours = air_time / 60,
    gain_per_hour = gain / hours)
```

mutate_all(), mutate_if() and mutate_at() and the three variants of transmute() (transmute_all(), transmute_if(), transmute_at()) make it easy to apply a transformation to a selection of variables. See help.

Kõigepealt grupeeri, siis muteeri. Konstandid Mean(value) ja sd(value) arvutatakse igale grupile eraldi selle grupi väärtuste pealt (siin grupeerime faktormuutuja "sex" kahe taseme järgi).

```
group_by(dat_lng, sex) %>%
  mutate(norm_value = value / mean(value),
         n2_val = value / sd(value))
#> # A tibble: 9 x 6
#> # Groups:   sex [2]
#>   subject sex   experiment   value norm_value n2_val
#>   <chr>   <chr> <chr>        <dbl>      <dbl>  <dbl>
#> 1 Tim     M     control         23      0.711   2.67
#> 2 Ann     F     control         31      0.842   5.47
#> 3 Jill    F     control         30      0.814   5.29
#> 4 Tim     M     experiment_1    34      1.05    3.94
#> 5 Ann     F     experiment_1    38      1.03    6.70
#> 6 Jill    F     experiment_1    36      0.977   6.35
#> # ... with 3 more rows
```

Võrdluseks ilma grupeerimata olukord, kus konstandil alati sama väärtus:

```
mutate(dat_lng,
       norm_value = value / mean(value),
       n2_val = value / sd(value))
#> # A tibble: 9 x 6
#>   subject sex   experiment   value norm_value n2_val
#>   <chr>   <chr> <chr>        <dbl>      <dbl>  <dbl>
#> 1 Tim     M     control         23      0.651   3.48
#> 2 Ann     F     control         31      0.877   4.69
```

```
#> 3 Jill    F     control          30      0.849   4.54
#> 4 Tim     M     experiment_1     34      0.962   5.14
#> 5 Ann     F     experiment_1     38      1.08    5.75
#> 6 Jill    F     experiment_1     36      1.02    5.44
#> # ... with 3 more rows
```

0.14.2.5.1 kahest veerust kolmanda tegemine nii, et NA-d esimeses veerus asendatakse numbritega teisest

```
y <- c(1, 2, 5, NA, 5)
z <- c(NA, NA, 7, 4, 5)
coalesce(z, y)
#> [1] 1 2 7 4 5
```

0.14.2.5.2 Summarise(), mutate(), transmute() ja filter() töötavad ka mitme veeru kaupa.

Need variandid sisaldavad suffikseid _if, _at ja _all.

_if võimaldab valida veerge teise funktsiooni, nagu näiteks is.numeric() või is.character() alusel.

_at võimaldab valida veerge sama süntaksiga, mis select().

_all valib kõik veerud.

`summarise_all(df, mean)` teeb sama asja, mis colMeans().

`summarise_all(df, funs(min, max))` võtab iga veeru min ja max väärtuse.

`summarise_all(df, ~ sd(.) / mean(.))` arvutab iga veeru CV (pane tähele ~ kasutust)

`summarise_all(df, funs(cv = sd(.) / mean(.), mean))` arvutab iga veeru CV ja keskmise (~ puudub, kui meil on >1 funktsiooni)

`summarise_at(df, vars(-z), mean)` keskmine kõigist veergudest, v.a. z.

`summarise_at(df, vars(x, y), funs(min, max))` kahe veeru min ja max.

`summarise_if(is.numeric, mean, na.rm = TRUE)` ainult numbritega veerud

`mutate_all(df, log10)` võta log10 kõikidest veergudest

`mutate_all(df, ~ round(. * 25))` teeb kõik veerud täisarvulisteks ja korrutab 25-ga

`mutate_all(df, funs(half = . / 2, double = . * 2))` rakendab 2 funktsiooni

`transmute_all(df, funs(half = . / 2, double = . * 2))` jätab alles ainult uued veerud

`filter_all(weather, any_vars(is.na(.)))` näitab ridu, mis sisaldavad NA-sid

`filter_at(weather, vars(starts_with("wind")), all_vars(is.na(.)))` read, kus veerg, mis sisaldab wind, on NA.

0.14.2.5.3 Kasutame group_by %>% summarise toru, et arvutada rea kaupa statistik (p väärtused)

Näiteks t test tidy tabelist. Meil on 5 geeni, N=3, võrreldakse kahte tingimust (indeks veerg, "E" ja "C").

```
library(tidyverse)
a <- tibble(gene= rep(1:5, each=6),
            value= rnorm(30),
            indeks= rep(c("E", "C"), each= 3, times=5))
head(a)
#> # A tibble: 6 x 3
#>    gene  value indeks
```

```
#>   <int>  <dbl> <chr>
#> 1     1 -0.961 E
#> 2     1  1.70  E
#> 3     1  2.09  E
#> 4     1  0.773 C
#> 5     1  1.58  C
#> 6     1 -0.799 C
```

```
a %>% group_by(gene) %>% summarise(p = t.test(value~indeks)$p.value)
#> # A tibble: 5 x 2
#>    gene     p
#>   <int> <dbl>
#> 1     1 0.740
#> 2     2 0.698
#> 3     3 0.730
#> 4     4 0.610
#> 5     5 0.260
```

0.14.2.5.4 Grupiviisiline filtreerimine

Säilita lennureiside sihtkohad, kuhu viib >365 lennu:

```
popular_dests <- flights %>%
  group_by(dest) %>%
  filter(n() > 365)
```

grupeeringu mahavõtmiseks, et töötada grupeerimata andmetega, kasuta `ungroup()`.

0.14.3 `separate()` üks veerg mitmeks

Siin on veel üks verb, mida aeg-ajalt kõigil vaja läheb. `separate()` võtab ühe veeru sisu (mis peab olema character string) ning jagab selle laiali mitme uue veeru vahel. Kui teda kasutada vormis `separate(df, old_Column, into=c("new_col1", "new_col2", "ja_nii_edasi"))` siis püüab programm ise ära ar-

vata, kustkohalt veeru sisu hakkida (tühikud, komad, semikoolonid, koolonid jne). Aga te võite eksplitsiitselt ette anda separaatori sep = “”. sep = 2 tähendab “peale 2. tähemärki”. sep = -6 tähendab “enne tagantpoolt 6. tähemärki”

```
(dat <- tibble(country = c("Albania"), disease.cases = c("80/1000")))
#> # A tibble: 1 x 2
#>   country disease.cases
#>   <chr>   <chr>
#> 1 Albania 80/1000
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand")))
#> # A tibble: 1 x 3
#>   country cases thousand
#>   <chr>   <chr> <chr>
#> 1 Albania 80    1000
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand"), sep
#> # A tibble: 1 x 3
#>   country cases thousand
#>   <chr>   <chr> <chr>
#> 1 Albania 80    1000
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand"), sep
#> # A tibble: 1 x 3
#>   country cases thousand
#>   <chr>   <chr> <chr>
#> 1 Albania 80    /1000
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand"), sep
#> # A tibble: 1 x 3
#>   country cases thousand
#>   <chr>   <chr> <chr>
#> 1 Albania 80    /1000
```

```
(dat <- tibble(index = c(1, 2),
               taxon = c("Procaryota; Bacteria; Alpha-Proteobacteria; Eschar
#> # A tibble: 2 x 2
#>   index taxon
#>   <dbl> <chr>
#> 1     1 Procaryota; Bacteria; Alpha-Proteobacteria; Es~
#> 2     2 Eukaryota; Chordata
```

```
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep =
#> # A tibble: 2 x 5
#>   index riik       hmk     klass              perekond
#>   <dbl> <chr>      <chr>   <chr>              <chr>
#> 1     1 Procaryota Bacter~ Alpha-Proteobact~ Escharich~
#> 2     2 Eukaryota  Chorda~ <NA>               <NA>
```

```
# some special cases:
(dat <- tibble(index = c(1, 2),
               taxon = c("Prokaryota || Bacteria || Alpha-Proteobacteria || I
(d1 <- dat %>% separate(taxon, c("riik", "hmk", "klass", "perekond"), sep =
```

```
dat <- tibble(index = c(1, 2),
              taxon = c("Prokaryota.Bacteria.Alpha-Proteobacteria.Escharichia
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep =
```

```
(dat <- tibble(index = c(1,2),
               taxon = c("Prokaryota.Bacteria,Alpha-Proteobacteria.Escharichi
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep =
```

Anti-separate funktsioon on `unite()` - vt. help.

Kuidas käituda siis, kui teil on veerus üks või mitu sissekannet, näiteks komadega eraldatud, ja te tahaksite need suruda mitte eraldi veergudesse (seda teeb separate()), vaid kõik ühte pikka veergu? Nüüd kasutame str_plit() ja unnest() funktsioone.

```
df1 <- tibble(a = c("w", "w, e, f", "g"), b=1:3)
df2 <- df1 %>% mutate(pikk_a = str_split(a, ",")) %>% select(pikk_a) %>% unn
#> Warning: `cols` is now required.
#> Please use `cols = c(pikk_a)`
df3 <- df1 %>% mutate(pikk_a = str_split(a, ",")) %>% unnest()
#> Warning: `cols` is now required.
#> Please use `cols = c(pikk_a)`
```

Pane tähele, et df3 on tabel, kus b veeru elemendid on duplitseeritud nii, et nad kataksid uue pikema pikk_a veeru vastavad elemndid. df2 on andmeraam, mis sisaldab ainult üht veergu.

0.14.4 Faktorid

Faktor on andmetüüp, mis oli ajalooliselt tähtsam kui ta praegu on. Sageli saame oma asja ära ajada character vectori andmetüübiga ja ei vaja faktorit. Aga siiski läheb faktoreid aeg-ajalt kõigil vaja.

> Faktorite abil töötame kategooriliste muutujatega, millel on fikseeritud hulk võimalikke väärtusi, mida me kõiki teame.

Faktori väärtusi kutsutakse "tasemeteks" (levels). Näiteks: muutuja sex on 2 tasemega faktor (M, F)

NB! Faktoriks muutes saame character vectori liikmete järjekorra muuta mitte-tähestikuliseks

Me kasutame faktoritega töötamisel forcats paketti. Kõigepealt loome character vectori x1 nelja kuu nime ingliskeelse lühendiga.

```
library(forcats)
x1 <- c("Dec", "Apr", "Jan", "Mar")
```

Nüüd kujutlege, et vektor x1 sisaldab 10 000 elementi. Seda vektorit on raske sorteerida, ja trükivead on ka raskesti leitavad. Mõlema probleemi vastu aitab, kui me konverteerime x1-e faktoriks. Selleks, et luua uus faktor, peaks kõigepealt üles lugema selle faktori kõik võimalikud tasemed:

Nüüd loome uue faktori ehk muudame x1 character vektori y1 factor vektoriks. Erinevalt x1-st seostub iga y1 väärtusega faktori tase. Kui algses vektoris on mõni element, millele ei vasta näiteks trükivea tõttu ühtegi faktori taset, siis see element muudetakse NA-ks. Proovige see ise järele, viies trükivea sisse x1-e.

```
y1 <- factor(x1, levels = month.abb)
y1
#> [1] Dec Apr Jan Mar
#> 12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep ... Dec
```

NB! `month.abb` on R objekt mis sisaldab kuude ingliskeelseid lühendeid.

Kui sa faktorile tasemeid ette ei anna, siis need tekivad andmetest automaatselt ja tähestikulises järjekorras.

Kui sa tahad, et faktori tasemed oleks samas järjekorras kui selle taseme esmakordne ilmumine teie andmetes siis:

```
f2 <- factor(x1) %>% fct_inorder()
f2
#> [1] Dec Apr Jan Mar
#> Levels: Dec Apr Jan Mar
```

`levels()` annab faktori tasemed ja nende järjekorra

```
levels(f2)
#> [1] "Dec" "Apr" "Jan" "Mar"
```

Kui faktorid on tibbles oma veeruna, siis saab nende tasemed `count()` kasutades:

```
gss_cat #tibble, mille veerg "race" on faktor.
#> # A tibble: 21,483 x 9
#>    year marital   age race  rincome partyid relig denom
#>   <int> <fct>   <int> <fct> <fct>   <fct>   <fct> <fct>
#> 1  2000 Never ~    26 White $8000 ~ Ind,ne~ Prot~ Sout~
#> 2  2000 Divorc~    48 White $8000 ~ Not st~ Prot~ Bapt~
#> 3  2000 Widowed    67 White Not ap~ Indepe~ Prot~ No d~
#> 4  2000 Never ~    39 White Not ap~ Ind,ne~ Orth~ Not ~
#> 5  2000 Divorc~    25 White Not ap~ Not st~ None  Not ~
#> 6  2000 Married    25 White $20000~ Strong~ Prot~ Sout~
#> # ... with 2.148e+04 more rows, and 1 more variable:
#> #   tvhours <int>
gss_cat %>% count(race)
#> # A tibble: 3 x 2
#>   race      n
#>   <fct> <int>
#> 1 Other  1959
#> 2 Black  3129
#> 3 White 16395
```

Nii saame ka teada, mitu korda iga faktori tase selles tabelis esineb.

0.14.4.1 tekitame faktortulba keerulisemal teel

dplyr::case_when(). Kui Sepal.Length on > 5.8 või Sepal.Width >4, siis uues veerus nimega fact ilmub tase "large", kui Species = setosa, siis ilmub tase "I. setosa", igal muul juhul ilmub .

```
library(tidyverse)
i <- iris %>% mutate(
    fact = case_when(
      Sepal.Length > 5.8 | Sepal.Width > 4 ~ "large",
      Species == "setosa"         ~ "I. setosa",
      TRUE                       ~  NA_character_
    ))
```

case_when() teeb loogilisi tehteid samas järjekorras, mis sa ette andsid. Seega kui mõni väärtus võiks minna mitmesse teie poolt spetsifitseeritud tingimusse, siis ta läheb tegelikult esimesena ette tulevasse tõesesse tingimusse.

0.14.4.2 `droplevels()` viskab välja kasutamata faktori tasemed

```
df1$sex <- droplevels(df1$sex)
```

0.14.4.3 `fct_recode()` rekodeerib faktori tasemed

```
gss_cat %>% count(partyid)
#> # A tibble: 10 x 2
#>   partyid               n
#>   <fct>             <int>
#> 1 No answer           154
#> 2 Don't know            1
#> 3 Other party         393
#> 4 Strong republican  2314
#> 5 Not str republican 3032
#> 6 Ind,near rep       1791
#> # ... with 4 more rows
gss_cat %>%
```

```
  mutate(partyid = fct_recode(partyid,
                              "Republican, strong"    = "Strong republican",
                              "Republican, weak"      = "Not str republican",
                              "Independent, near rep" = "Ind,near rep",
                              "Independent, near dem" = "Ind,near dem",
                              "Democrat, weak"        = "Not str democrat",
                              "Democrat, strong"      = "Strong democrat",
                              "Other"                 = "No answer",
                              "Other"                 = "Don't know",
                              "Other"                 = "Other party"
  )) %>%
  count(partyid)
#> # A tibble: 8 x 2
#>   partyid                   n
#>   <fct>                 <int>
#> 1 Other                   548
#> 2 Republican, strong     2314
#> 3 Republican, weak       3032
#> 4 Independent, near rep  1791
#> 5 Independent            4119
#> 6 Independent, near dem  2499
#> # ... with 2 more rows
```

`fct_recode()` ei puuduta neid tasemeid, mida selle argumendis ei mainita. Lisaks saab mitu vana taset muuta üheks uueks tasemeks.

0.14.4.4 `fct_collapse()` annab argumenti sisse vanade tasemete vektori, et teha vähem uusi tasemeid.

```
gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
                                other = c("No answer", "Don't know", "Other p
                                rep = c("Strong republican", "Not str republi
                                ind = c("Ind,near rep", "Independent", "Ind,n
                                dem = c("Not str democrat", "Strong democrat"
  )) %>%
  count(partyid)
```

0.14.4.5 `fct_lump()` lööb kokku kõik vähem arv kordi esinevad tasemed.

n parameeter ütleb, mitu algset taset tuleb alles jätta:

```
gss_cat %>%
  mutate(relig = fct_lump(relig, n = 5)) %>%
  count(relig, sort = TRUE) %>%
  print()
#> # A tibble: 6 x 2
#>   relig          n
#>   <fct>      <int>
#> 1 Protestant 10846
#> 2 Catholic    5124
#> 3 None        3523
#> 4 Other        913
#> 5 Christian    689
#> 6 Jewish       388
```

0.14.4.6 Rekodeerime pideva muutuja faktoriks

`cut()` jagab meie muutuja väärtused intervallidesse ja annab igale intervallile faktori taseme.

```
cut(x, breaks, labels = NULL, ordered_result = FALSE,
...)
```

breaks - either a numeric vector of two or more unique cut points or a single number >1, giving the number of intervals into which x is to be cut. labels - labels for the levels of the resulting category. ordered_result - logical: should the result be an ordered factor?

```
z <- 1:10
z1 <- cut(z, breaks = c(0, 3, 6, 10), labels = c("A", "B", "C"))
z1
#>  [1] A A A B B B C C C C
#> Levels: A B C
```

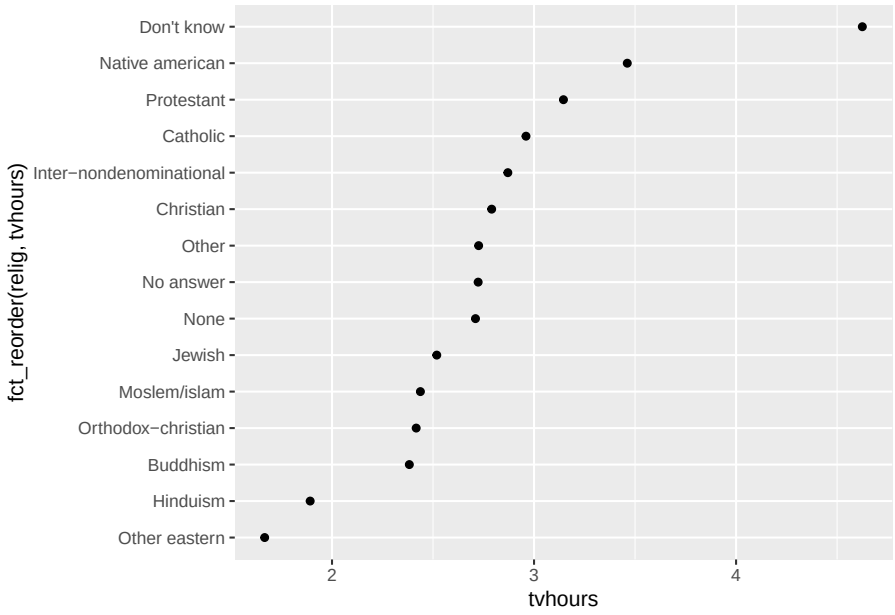
```
#Note that to include 1 in level "A" you need to start the first cut <1, wh
z2 <- cut(z, breaks = 3, labels = c("A", "B", "C"))
z2
#>  [1] A A A A B B B C C C
#> Levels: A B C
```

`car::recode` aitab rekodeerida

```
library(car)
x <- rep(1:3, 3)
x
#> [1] 1 2 3 1 2 3 1 2 3
recode(x, "c(1,2) = 'A'; else = 'B'")
#> [1] "A" "A" "B" "A" "A" "B" "A" "A" "B"
recode(x, "c(1,2) = NA")
#> [1] NA NA  3 NA NA  3 NA NA  3
recode(x, "1:2 = 'A'; 3 = 'B'")
#> [1] "A" "A" "B" "A" "A" "B" "A" "A" "B"
```

0.14.4.7 Muudame faktori tasemete järjekorda joonisel

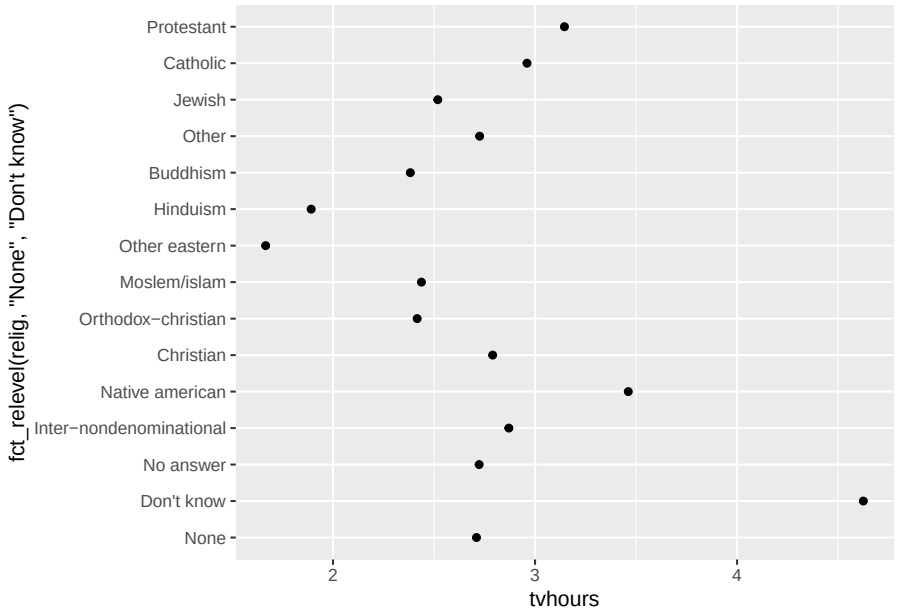
```
## summeerime andmed
gsscat_sum  <- group_by(gss_cat, relig) %>%
  summarise(age = mean(age, na.rm = TRUE),
            tvhours = mean(tvhours, na.rm = TRUE),
            n = n())
## joonistame graafiku
p <- ggplot(gsscat_sum, aes(tvhours, fct_reorder(relig, tvhours))) +
  geom_point()
p
```



0.14.4.8 `fct_relevel()` tõstab joonisel osad tasemed teistest ettepoole

Argumendid on faktor f ja need tasemed (jutumärkides), mida sa tahad tõsta.

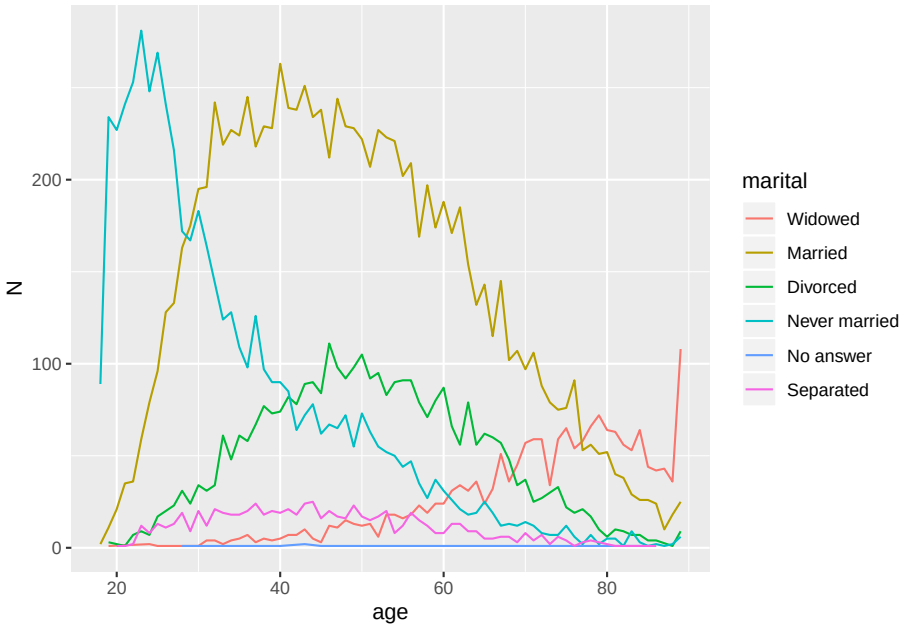
```
## täiendame eelmist graafikut ümberkorraldatud andmetega
p + aes(tvhours, fct_relevel(relig, "None", "Don't know"))
```



0.14.4.9 Joontega plotil saab `fct_reorder2()` abil assotseerida y väärtused suurimate x väärtustega

See muudab ploti paremini jälgitavaks:

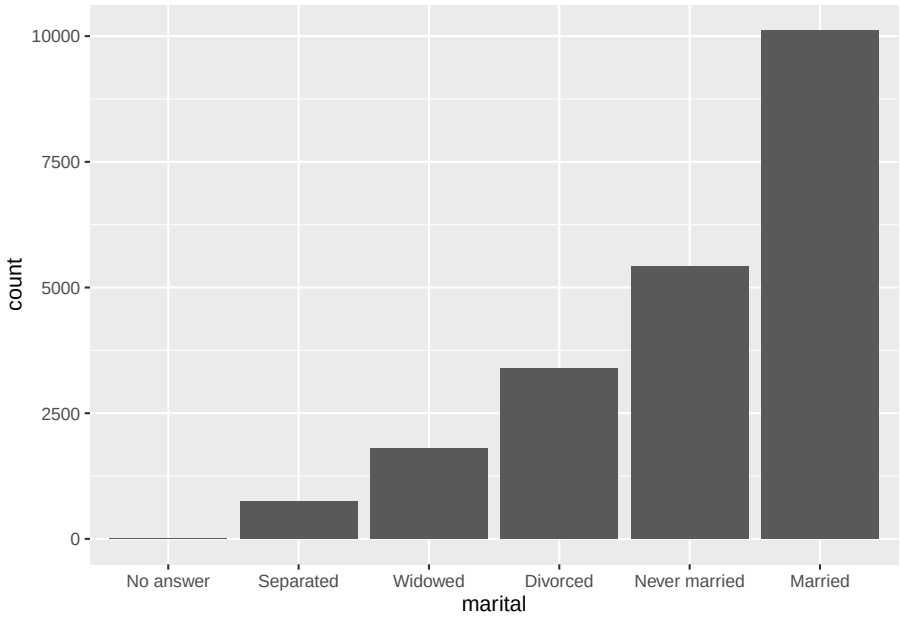
```
## summeerime andmed
gsscat_sum <- filter(gss_cat, !is.na(age)) %>%
  group_by(age, marital) %>%
  mutate(N=n())
## paneme andmed graafikule
ggplot(gsscat_sum, aes(age, N, colour = fct_reorder2(marital, age, N))) +
  geom_line() +
  labs(colour = "marital")
```



0.14.4.10 Tulpdiagrammide korral kasuta `fct_infreq()`

Loeme kokku erineva perekondliku staatusega isikud ja paneme need andmed tulpdiagrammi grupi suurusele vastupidises järjekorras st. väiksemad grupid tulevad enne.

```
mutate(gss_cat, marital = fct_infreq(marital) %>% fct_rev()) %>%
  ggplot(aes(marital)) + geom_bar()
```



0.15 lisa: funktsioonid

0.15.1 raamatukogud

`install.packages("ggplot2")` laadib raamatukogu CRAN-ist alla `library(ggplot2)` muudab raamatukogu funktsioonid kättesaadavaks

0.15.1.1 bioconductor

First run biocLite script fron bioconductor.org
`source("https://bioconductor.org/biocLite.R")`
use 'http' in url if 'https' is unavailable. `biocLite("edgeR")`

0.15.1.2 Github

https://github.com following command installs xaringan (presentation ninja) package from GitHub user yihui `devtools::install_github("yihui/xaringan")`

0.15.2 failide sisselugemine

- `readit::readit()`kasutab tidyverse, loeb sisse mida iganes
- `readr::read_delim(file, delim="")` saad määrata delimiteri
- `readr::read_csv2()` loeb sisse Exceli csv-d
- `readr::read_csv()` loeb sisse csv-d
- Addins/Gotta Read Em All - interaktiivne
- `readr::write_csv()  & base::write.csv()`

0.15.3 matemaatika

- `sum(x, na.rm = TRUE)`
- `sqrt()` võtab ruutjuure
- `prod()` korrutab
- `log()` naturaallogaritm alusel e
- `exp()` anti-logaritm alusele e
- `log2()`
- `log10()`
- `round(x, 2)` ümardab 2le komakohale
- ^ - astendamine (** töötab ka)
- * - korrutamine
- / - jagamine
- == - võrdusmärk

0.15.4 andmeraamid

- `tibble()` andmeraami sisestamine rea vektoritena
- `tribble()` andmeraami sisetamine tabelina
- `str(df)` näitab tabeli struktuuri
- `nrow()` annab tabeli ridade arvu

- `ncol()`
- `class()` annab objekti klassi
- `as_tibble() & as.data.frame()`
- `add_row()`
- `add_column()`
- `distinct()` eemalda duplikaatread
- `count()` loeb üles tabeli rea väärtuste aesinemise arvu
- `add_count()` lisab faktori esinemiste arvu tabelile uue veeruna "n"
- `table()` annab väärtuste aesinemise arvu igal faktorite taseme kombinatsioonil
- `summary()` summeerib tabeli
- `psych::describe()` summeerib tabeli
- `colnames()`
- `rownames()`
- `rownames_to_column()`
- `remove_rownames()`
- `rowid_to_column()` adds a column of ascending nrs starting at 1.
- `arrange(df, desc(column_name))` sordib read
- `top_n()` annab n suureima/väikseima väärtusega rida
- `colSums()`
- `rowSums()`
- `rowMeans()`
- `apply()`
- `bind_rows(df1, df2, .id = "id")` (base::rbind)
- `bind_cols()` (base::cbind)
- `full_join(df1, df2)`
- `left_join(df1, df2)` ühendab df2 df1-e ridadega nii, et uusi ridu ei teki.
- `semi_join(df1, df2)` filter: 2 tabeli ühisosa
- `anti_join(df1, df2)` filter: 1 tabeli read, mis puuduvad 2. tabelis

0.15.5 NA-d

- `VIM::aggr()` näitab puuduvad väärtused

- `sapply(diabetes, function(x) sum(is.na(x)))` Mitu NA-d on igas tulbas.
- `VIM::matrixplot(x)` NA-de plot
- `filter_all(x, any_vars(is.na(.)))` annab read, mis sisaldavad NA-sid
- `na_if(x, y)` rekodeerib vektoris x väärtused y NA-deks
- `coalesce(x, 0L)` rekodeerime NA 0-ks
- “
- “
- “
- “
- “
- “
- “
- “

Bååth, Rasmus. 2013. “Bayesian First Aid.” *Tba*. tba.

———. 2016. *Bayesboot: An Implementation of Rubin’s (1981) Bayesian Bootstrap*. https://CRAN.R-project.org/package=bayesboot.

Bürkner, Paul-Christian. 2017. “brms: An R Package for Bayesian Multilevel Models Using Stan.” *Journal of Statistical Software* 80 (1): 1–28. doi:10.18637/jss.v080.i01.

Gabry, Jonah, and Tristan Mahr. 2017. *Bayesplot: Plotting for Bayesian Models*. http://mc-stan.org/bayesplot.

Gelman, Andrew, John B Carlin, Hal S Stern, David B Dunson,

Aki Vehtari, and Donald B Rubin. 2014. *Bayesian Data Analysis*. Vol. 2. CRC press Boca Raton, FL.

Kruschke, John. 2015. *Doing Bayesian Data Analysis: A Tutorial with R, Jags, and Stan*. 2nd ed. Academic Press / Elsevier.

Marwick, Ben, Carl Boettiger, and Lincoln Mullen. 2017. "Packaging Data Analytical Work Reproducibly Using R (and Friends)." *PeerJ Preprints* 5: e3192v1. doi:10.7287/peerj.preprints.3192v1.

McElreath, Richard. 2015. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. CRC Press.

———. 2016. *Rethinking: Statistical Rethinking Book Package*.

Stan Development Team. 2016. *Rstanarm: Bayesian Applied Regression Modeling via Stan*. http://mc-stan.org/.

Wickham, Hadley, Peter Danenberg, and Manuel Eugster. 2017. *Roxygen2: In-Line Documentation for R*. https://CRAN.R-project.org/package=roxygen2.