

Reprodutseeritav andmeanalüüs kasutades R keelt

Taavi Päll, Ülo Maiväli

2018-02-23

Sisukord

Haara kannel, Vanemuine!	iv
o.1 Sissejuhatus	v
o.2 Tarkvaratööriistad	v
o.2.1 Installeeri vajalikud programmid	v
o.2.2 Loo GitHubi konto	vi
o.2.3 Loo uus R projekt	vi
o.2.4 Git Merge konfliktid	vi
o.2.5 R projekti kataloogi soovitatav minimaalne struktuur	vii
o.2.6 Pakettide installeerimine	ix
o.2.7 R repositooriumid	xi
o.3 R on kalkulaator	xii
o.3.1 Sama koodi saab kirjutada neljal viisil	xiii
o.4 R objektid	xiv
o.4.1 Objekt ja nimi	xv
o.4.2 Nimevorm	xvi
o.4.3 Andmete tüübid	xvii
o.4.4 Vektor	xvii
o.4.5 List	xx
o.4.6 data frame ja tibble	xxi
o.4.7 Tabelit sisse lugedes vaata üle NA-d	xxx
o.4.8 Matrix	xxxiii
o.4.9 Indekseerimine	xxxiv
o.5 Regular expression ja find & replace	xxxvii
o.5.1 Common operations with regular expressions	xxxix
o.5.2 Find and replace	xl
o.6 Funktsioonid on R keele verbid	xlii
o.6.1 Kirjutame R funktsiooni	xliv
o.7 Graafilised lahendused	xlvi
o.7.1 Baasgraafika	xlvi
o.7.2 ggplot2	lv
o.7.3 Regressioonisirgete plottimine	lvii
o.7.4 Facet – pisigraafik	lxxi
o.7.5 Mitu graafikut paneelidena ühel joonisel	lxxiv
o.7.6 Teljed	lxxv
o.7.7 Graafiku pealkiri, alapealkiri ja allkiri	lxxxii
o.7.8 Graafiku legend	lxxxiii
o.7.9 Värviskaalad	lxxxiv
o.7.10 A complex ggplot	xc

0.8	Tosin olulisimat graafikutüüpi	xcii
0.8.1	1. Cleveland plot	xciii
0.8.2	2. Andmepunktid mediaani või aritmeetilise keskmisega	xcvii
0.8.3	3. Histogramm	xcix
0.8.4	3. Tihedusplot	civ
0.8.5	4. Boxplot	cvi
0.8.6	5. Joongraafikud	cviii
0.9	Tidyverse	cxvii
0.9.1	Tidy tabeli struktuur	cxvii
0.9.2	dplyr ja selle viis verbi	cxx
0.9.3	Grouped filters	cxx
0.9.4	separate() one column into several	cxix
0.9.5	Faktorid	cxix
0.10	Statistilised mudelid	cxlii
0.10.1	Suur ja väike maailm	cxlii
0.10.2	Mudeli väike maailm	cxliii
0.10.3	Lineaarsed mudelid	cxlvii

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 töövoog reprodutseeritavaks.

Skriptid ja koodid võimaldavad sul hoida lahus algandmed (mis on püha ja puutumatu) andmete töötlustest 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 nõ tšielik 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 prae-

guseks hetkeks välja töötatud ka suurepäraseks 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 näplereid (viimast küll ainult ‘BayesianFirstAid’ paket). Selle õpiku valmimisel on kasutatud McElreathi (McElreath, 2015), Kruschke (Kruschke, 2015) ja nn. Gelmani (Gelman et al., 2014) õpikuid.

0.2 Tarkvaratööriistad

0.2.1 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 veebilehele¹, laadi alla ja installi sobiv versioon.
2. Googelda “install RStudio” või mine otse RStudio allalaadimise veebilehele², laadi alla ja installi sobiv versioon.

¹<https://cran.r-project.org>

²<https://www.rstudio.com/products/rstudio/download/>

3. Googelda “install git” või mine otse Git allalaadimise veebilehele³, laadi alla ja installi sobiv versioon.

0.2.2 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”.

0.2.3 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”.

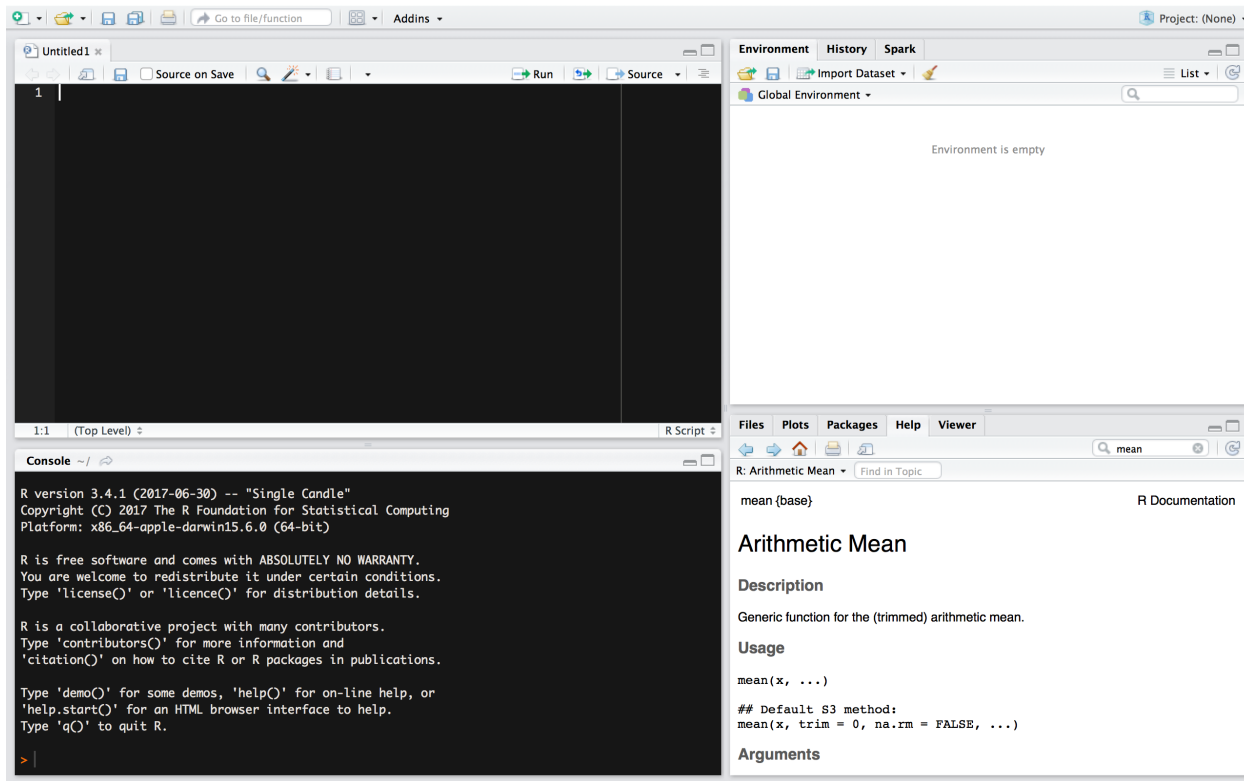
Rohkem infot R projekti loomise kohta leiad RStudio infoleheküljelt: Using Projects⁴.

0.2.4 Git Merge konfliktid

Kollaboreerides üle GitHubi tekivad varem või hiljem konfliktid projekti failide versioonide vahel nn. “merge conflicts”, nende korrektselt lahendamata õppimine on väga oluline.

³<https://git-scm.com/downloads>

⁴<https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects>



Joonis 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 `<i class='fa fa-git' aria-hidden='true'>`. Seal on näha R-i objektid, mis on sulle töökeskkonnas kättesaadavad ja millega sa saad töötada. `<i class='fa fa-git' aria-hidden='true'>` menüüs on võimalik muutusi vaadata ja 'commit'ida ja `<i class='fa fa-github' aria-hidden='true'>`-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).

- 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.

0.2.5 R projekti kataloogi soovitatav minimaalne struktuur

Iga R projekt peab olema täiesti iseseisev (*selfcontained*) ja sisaldama kogu infot, andmeid ja instruktsioone, et projektiga seotud arvutused läbi viia ja raport genereerida. Kõik faili *path*-id peavad olema suhtelised.

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 andmesetide asukoht, vajalik tarkvara jne. **README.md** on markdown formaadis projekti info, sisaldab juhendeid kasutajatele. Igale GitHubi repole on soovitatav 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 programmeerimiselt. Suured failid muudavad versioonikontrolli aeglaseks, samuti on suhteliselt 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.*”

src/ kataloog sisaldab analüüsi skripte, sealhulgas ka andmetötluse skripte.

lib/ kataloogis on kasutaja poolt tehtud funktsioonide definitsioonid sisaldavad R skriptid.

```
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_data.csv   # data files in open formats,
|                   # such as TXT, CSV, TSV etc.
|
|- src/            # any programmatic code
```



```
| +-my_scripts.R      # R code used to analyse and  
|                     # visualise data  
|  
|- lib/               # user generated functions  
| +-my_functions.R   # R code defining functions
```

On ka teisi konventsioone, näiteks R pakside puhul paigutatakse kõik R skriptid taaskasutatavate funktsioonidega kataloogi **R/**. Kui selles kataloogis olevad skriptid on annoteeritud kasutades Roxygen-i (Wickham et al., 2017), siis genereeritakse automaatselt funktsioonide dokumentatsioon kataloogi **man/**. Rohkem projekti pakkimise kohta loe värskest preprindist “Packaging data analytical work reproducibly using R” (Marwick et al., 2017).

0.2.6 Pakettide installeerimine

R *library*-d ehk paketid sisaldavad ühte või enam 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 3.4.2 sisaldab 453 funktsiooni. Enamasti on sarnaseid asju tegevad funktsioonid koondatud kokku raamatukogudesse ehk pakettidesse, mis tuleb eraldi R kesksest repositooriumist CRAN⁵ alla laadida ja installeerida.

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

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

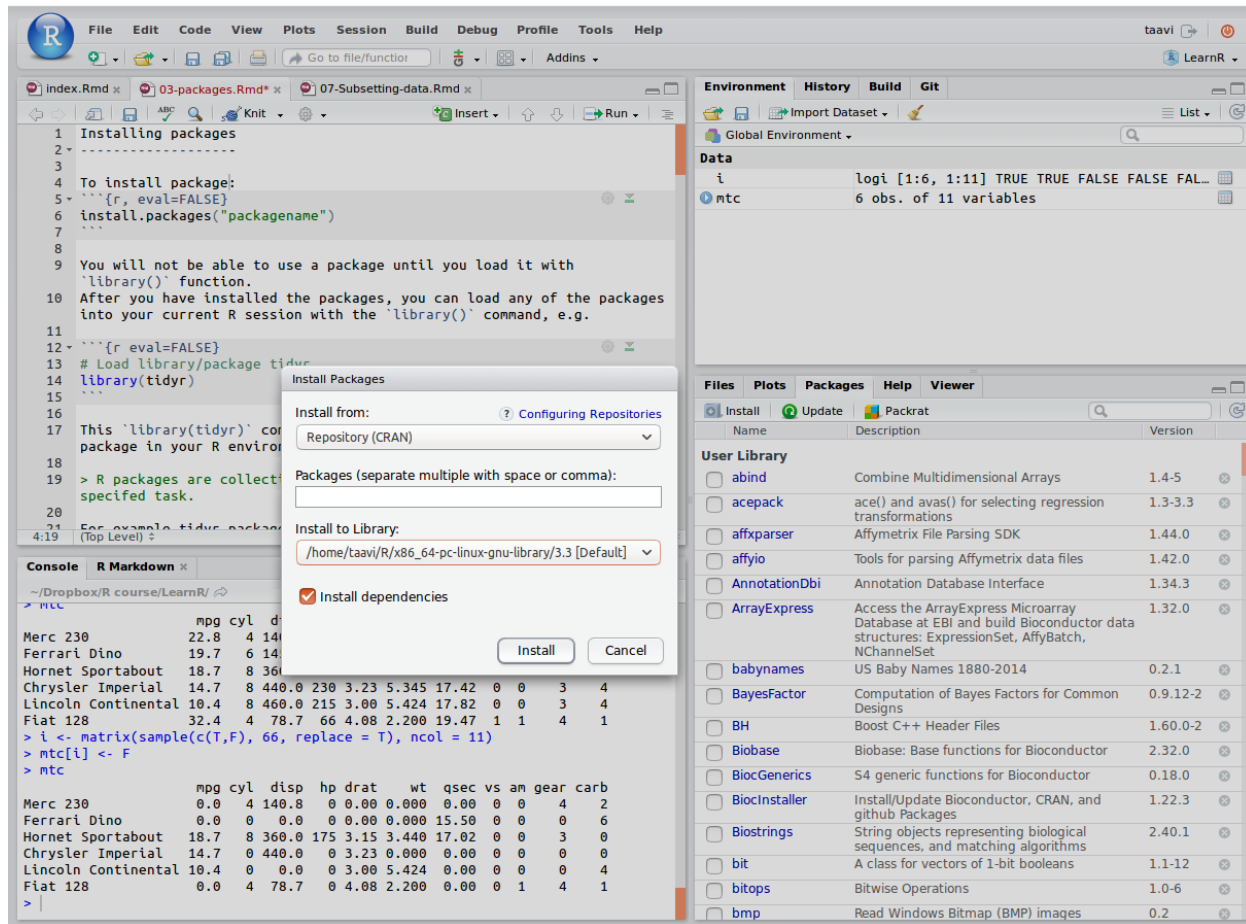
NB! Kui mõni raamatukogu sel viisil alla ei tule, siis guugeldage selle nime + R ja vaadake instruksioone installeerimiseks. Suure tõenäosusega on tegemist mõnes teises repos (näiteks Bioconductor) või ainult GitHubis asuva paketiga.

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:

⁵<https://cran.r-project.org>



Joonis 2: RStudio 'Install Packages' dialoogiaken.

```
## Load library/package dplyr
```

```
library(dplyr)
```

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

Näiteks "dplyr" pakett sisaldab 237 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_tibble"
#> [19] "as.tbl"    "as.tbl_cube"
```

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 klikida raamatukogu nimele ja näha selle help-faile, tutooriale ja kõiki selle raamatukogu funktsioone koos nende help failidega.

0.2.7 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 from bioconductor.org
source("https://bioconductor.org/biocLite.R")
# use 'http' in url if 'https' is unavailable.
biocLite("edgeR")
```

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

```
## Näiteks järgnev käsk installeerib xaringan
## presentation ninja paketi
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.

Paiguta kõigi raamatukogude lugemine koodi algusesse. Enamasti kirjutatakse sisse loetavad raamatukogud kohe R scripti algusesse. Siis on teile endale ja teistele kes teie koodi loevad ilusti näha, mida hiljem vaja läheb.

0.3 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 soovitatav enda poolt loodavatele objektidele mitte panna nimeks “pi”.

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

Ümardamine on oluline tulemuste väljaprintimisel.

0.3.1 Sama koodi saab kirjutada neljal viisil

Hargnevate teede aed: kui me muudame olemasolevat objekti on meil alati kaks valikut. Me kas jätame muudetud objektile vana objekti nime või me anname talle uue nime. Esimesel juhul läheb vana muutmata objekt workspacest kaduma aga nimesid ei tule juurde ja säilib teatud workflow sujuvus. Teisel juhul jäävad analüüsi vaheobjektid meile alles ja nende juurde saab alati tagasi tulla. Samas tekkib meile palju sarnaste nimedega objekte.

Kõigepealt laadime vajalikud raamatukogud.

```
## We need piping operator '%>%' from magrittr.  
## We can import '%>%' via dplyr from tidyverse  
library(dplyr)
```

Esimene võimalus:

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

Teine võimalus:

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

Kolmas võimalus on lühem variant esimesest. Me nimelt ühendame etapid toru `%>%` kaudu. 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.

NB! R hea stiili juhised soovivad siiski ka *pipe*-s kasutada funktsiooni koos sulgudega!

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

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

Neljas võimalus, klassikaline baas R lahendus:

```
a <- c(2, 3)
a1 <- round(sqrt(sum(a)), 2)
a1
#> [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 `1 -> 2 -> 3 -> 4` (vähem lollikindel)

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

0.4 R objektid

R-i töökeskkonnas “workspace” asuvad **objektid**, millega me töötame. Tüüpilised objektid on:

- Vektorid, maatriksid, listid ja tabelid.
- Statistiliste analüüside väljundid (S3, S4 klass).
- Funktsioonid, mille oleme ise sisse lugenud.

Käsk `ls()` annab objektide nimed teie workspace-s:

```
ls()  
#> [1] "old"
```

`rm(a)` removes object a from the workspace

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 soovitatav 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 võtavad kalkulatsioonid või allalaadimised salvesta need eraldi .rds faili ja laadi koodis vastavalt vajadusele.

Nüüd laadime hiljem vaja minevad libraryd:

```
library(tidyverse)  
library(VIM)  
library(readxl)  
## Install gotta read em all as R studio addin  
## install.packages("devtools")  
#devtools::install_github("Stan125/GREA")
```

0.4.1 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 assingmenti sümbol <- ja lõpuks objekt, mis on nimele antud (string “beebi”).

NB! Stringid on jutumärkides, nimed mitte. Nimi üksi evalueeritakse kui “print object”, mis antud juhul on 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 kasutatavad 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.

0.4.2 Nimede vorm

- Nimed algavad tähemärgiga, mitte numbriga ega \$€%&/?~öüä
- 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 meile auto-complete.
- alt + - on otsetee <- jaoks

0.4.3 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.

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

0.4.4 Vektor

Vektor on rida kindlas järjekorras arve, sõnu või TRUE/FALSE loogilisi väärtusi. Iga vektor ja maatriks (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

Loo me numbrilise vektori ja vaatame ta struktuuri:

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

Loo me 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 ekstraheerida numbrid:

```

minu_vektor <- c(1, "A2", "$2", "joe")
## parse_number() is imported from tidyverse 'readr'
minu_vektor <- as.vector(parse_number(minu_vektor))
#> Warning: 1 parsing failure.
#> row # A tibble: 1 x 4 col      row  col expected actual expected  <int> <int> <chr>   <chr> actual 1
minu_vektor
#> [1]  1  2  2 NA
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

```

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

```

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"

```

0.4.4.2 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 $1 + 0 + 1 = 2$. Seda loogiliste väärtuste omadust õpime varsti praktikas kasutama.

0.4.5 List

List on objektitüüp, kuhu saab koondada kõiki teisi objekte, kaasa arvatud listid. See on lihtsalt viis objektid koos hoida ühes suuremas meta-objektis. List on nagu jõuluvana kingikott, kus kommid, sokipaarid ja muud kingid kõik segamini loksuvad.

Näiteks siin list, kus loksuvad 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] 0.1174 0.0413 0.8375 0.4226 0.6060
#>
#> [[2]]
#>      a      b
#> 1 0.1174 -0.286
#> 2 0.0413  1.097
#> 3 0.8375 -0.565
#> 4 0.4226  1.327
#> 5 0.6060 -1.832
#>
#> [[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
```

0.4.6 data frame ja tibble

Andmeraam on eriline list, mis koosneb ühepikkustest vektoritest. Andmeraam on ühtlasi teatud liiki tabel, kus 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 tulba nimele ja ühtlasi vektori nimele). Iga vektori nimi muutub sellises tabelis tulba nimeks. Igas tulbas saab olla ainult ühte tüüpi andmeid.

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

Oluline on, et erinevalt data frame-st saab tibbleesse lisada ka list tulpasid, mis võimaldab sisuliselt suvalisi R objekte tibbleesse paigutada. Põhimõtteliselt piisab ainult ühest andmestruktuurist – tibble, et R-is töötada. Kõik mis juhtub tibbles jääb tibbleesse. Nice and tidy – tidyverse.

“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    1.00     2.00 <chr [26]>  
#> 2 tesco     4.00    32.0  <dbl [10]>  
#> 3 lidl     43.0     NA    <S3: lm>
```

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

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

```
dfs <- try(data.frame(shop, apples, oranges, vabakava))
dfs
#> [1] "Error in as.data.frame.default(x[[i]], optional = TRUE, stringsAsFactors = stringsAsFactors) : \n  cannot coerce type 'list' to vector of type 'character'"
#> attr(,"class")
#> [1] "try-error"
#> attr("condition")
#> <simpleError in as.data.frame.default(x[[i]], optional = TRUE, stringsAsFactors = stringsAsFactors): cannot
```

Mõned asjad, mida tibblega (ja data framega) saab teha:

```
count(fruits, apples)
#> # A tibble: 3 x 2
#>   apples      n
#>   <dbl> <int>
#> 1   1.00     1
#> 2   4.00     1
#> 3  43.0     1

count(fruits, shop)
#> # A tibble: 3 x 2
#>   shop      n
#>   <chr> <int>
#> 1 lidl     1
#> 2 maxima   1
#> 3 tesco     1

summary(fruits)
#>      shop      apples      oranges
#> Length:3      Min.    : 1.0   Min.    : 2.0
#> Class :character 1st Qu.: 2.5   1st Qu.: 9.5
#> Mode  :character Median : 4.0   Median :17.0
#>      Mean    :16.0   Mean    :17.0
#>      3rd Qu.:23.5   3rd Qu.:24.5
#>      Max.    :43.0   Max.    :32.0
#>      NA's    :1
#> vabakava.Length vabakava.Class vabakava.Mode
#> 26      -none-      character
#> 10      -none-      numeric
#> 12      lm         list
#>
#>
#>
#>

names(fruits)
#> [1] "shop"      "apples"    "oranges"   "vabakava"
```

```

colnames(fruits)
#> [1] "shop"      "apples"    "oranges"   "vabakava"
nrow(fruits)
#> [1] 3
ncol(fruits)
#> [1] 4
arrange(fruits, desc(apples)) #sorteerib tabeli veeru "apples" väärtuste järgi langevalt (default on tõusev sort)
#> # A tibble: 3 x 4
#>   shop  apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 lidl    43.0     NA    <S3: lm>
#> 2 tesco    4.00    32.0 <dbl [10]>
#> 3 maxima  1.00     2.00 <chr [26]>
top_n(fruits, 2, apples) #saab 2 rida, milles on kõige rohkem õunu
#> # A tibble: 2 x 4
#>   shop  apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 tesco    4.00    32.0 <dbl [10]>
#> 2 lidl    43.0     NA    <S3: lm>
top_n(fruits, -2, apples) #saab 2 rida, milles on kõige vähem õunu
#> # A tibble: 2 x 4
#>   shop  apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima  1.00     2.00 <chr [26]>
#> 2 tesco    4.00    32.0 <dbl [10]>

```

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 meie tabelile 1 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     1.00     2.00 <chr [26]>
#> 2 tesco      4.00    32.0  <dbl [10]>
#> 3 konsum  132      - 5.00 <NULL>
#> 4 lidl     43.0     NA    <S3: lm>
```

Proovi ise:

```
add_column()
```

Eelnevaid verbe ei kasuta me vist enam kunagi sest tavaliselt loeme me andmed sisse väljaspoolt R-i. Aga väga kasulikud on järgmised käsud:

0.4.6.1 Rekodeerime tibble väärtusi

```
fruits$apples[fruits$apples==43] <- 333
fruits
#> # A tibble: 4 x 4
#>   shop    apples oranges vabakava
#>   <chr>    <dbl>    <dbl> <list>
#> 1 maxima     1.00     2.00 <chr [26]>
#> 2 tesco      4.00    32.0  <dbl [10]>
#> 3 konsum  132      - 5.00 <NULL>
#> 4 lidl     333      NA    <S3: lm>
fruits$shop[fruits$shop=="tesco"] <- "TESCO"
fruits
#> # A tibble: 4 x 4
```

```
#>   shop   apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima   1.00     2.00 <chr [26]>
#> 2 TESCO    4.00    32.0  <dbl [10]>
#> 3 konsum 132      - 5.00 <NULL>
#> 4 lidl    333      NA    <S3: lm>
fruits$apples[fruits$apples>100] <- NA
fruits
#> # A tibble: 4 x 4
#>   shop   apples oranges vabakava
#>   <chr>   <dbl>   <dbl> <list>
#> 1 maxima   1.00     2.00 <chr [26]>
#> 2 TESCO    4.00    32.0  <dbl [10]>
#> 3 konsum  NA      - 5.00 <NULL>
#> 4 lidl     NA      NA    <S3: lm>
```

Remove duplicate rows where specific column (col1) contains duplicated values:

```
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
```

0.4.6.2 Ühendame kaks tibble reaktiivset kaup

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 rida pärit on
bind_rows(dfs, dfs1, .id = "id")
#> # A tibble: 4 x 3
#>   id   colA   colB
#>   <chr> <chr> <dbl>
#> 1 1     a     1.00
#> 2 1     b     2.00
```

```
#> 3 1      c      3.00
#> 4 2      d      4.00
```

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     4.00 <NA>   NA
#> 2 <NA>  NA     d     4.00
```

0.4.6.3 ühendame kaks tibble't 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     1.00  4.00
#> 2 b     2.00  5.00
#> 3 c     3.00  6.00
```

0.4.6.4 tabelite 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 Harrison", "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 Harrison 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 Harrison   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>
```

Filtreerin välja need df1 read, millele vastab rida df2-s.

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

Filtreerin välja need df1 read, millele ei vasta rida df2-s.

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

0.4.6.5 Nii saab tibblest kätte vektori, millega saab 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
```

0.4.6.6 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 “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] "/home/travis/build/rstats-tartu/lectures"
```

Andmete sisselugemine töökataloogist:

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

MS exceli failist saab tabelleid 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 framenä 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 tabelleid oma workspace üles laadida. Selline point-and-click lahendus sobib ehk tabelite esialgseks tutvumiseks, kuid korrektne on andmed importida programmeeriliselt oma skriptis.

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

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

NB! R ei muuda algandmeid, mille te näiteks csv-na sisse loete - need jäävad alati selliseks nagu need instrumendi või andmesisestaja poolt väljastati.

Seega ei ole andmetabelite salvestamine töö vaheproduktidena sageli vajalik sest te jookсутate iga kord, kui te oma projekti juurde naasete, kogu analüüsi uuesti kuni kohani, kuhu te pooleli jäite. See tagab kõige paremini, et teie kood töötab tervikuna. Erandiks on tabelid, mille arvutamine võtab palju aega.

Tibble konverteerimine data frame-ks ja tagasi tibbleks:

```
class(fruits)
#> [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"
```

o.4.7 Tabelit sisse lugedes vaata üle NA-d

```
diabetes <- read.table(file = "data/diabetes.csv", sep = ";", dec = ",", header = TRUE)
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 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 2 4 ...
#> $ 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 ...
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 o NA-d? Mitu % kõikidest ridadest?

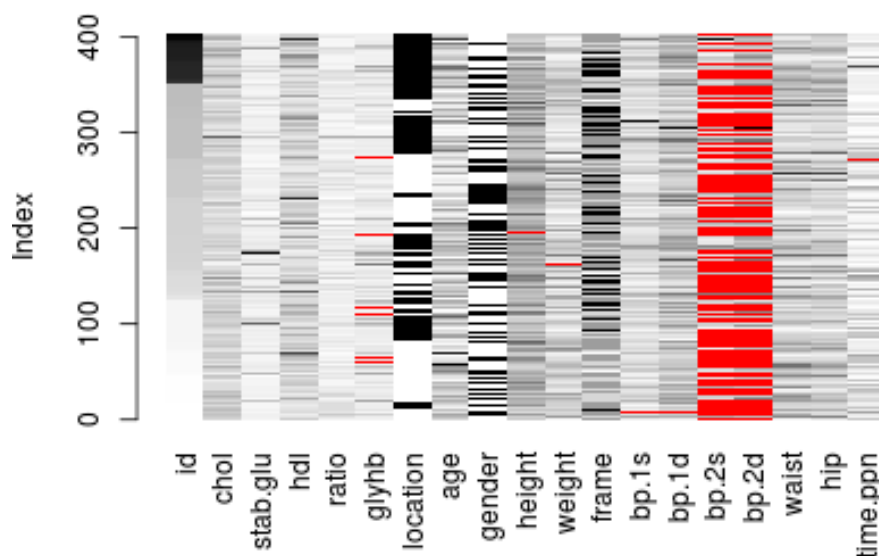
```
nrows <- nrow(diabetes)
ncomplete <- sum(complete.cases(diabetes))
ncomplete #136
#> [1] 136
ncomplete/nrows #34%
#> [1] 0.337
```

Mitu NA-d on igas tulbas?

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

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

```
matrixplot(diabetes)
```

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.

Rekodeerime numbri vm NA-ks

```
na_if(x, y)
```

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

0.4.8 Matrix

Maatriks on 2-dimensionaalne vektor, sisaldab ainult ühte tüüpi andmeid – numbrid, stringid, faktorid. Tip: me saame sageli andmeraami maatriksina kasutada kui me viskame sealt välja mitte-numbrilised tulbad.

Aga saame ka andmeraame konverteerida otse maatriksiks (ja tagasi).

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

0.4.9 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 indeksi mõte väga lühikese käsuga välja võtta R-i objektide üksikuid elemente.

R-s algab indeksi numeratsioon 1-st (mitte 0-st, nagu näiteks Pythonis).

0.4.9.1 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. element number 1  
#> [1] 3 4 5  
my_vector[c(-1, -3)] #kõik elemendid, v.a. element number 1 ja 3  
#> [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
```

0.4.9.2 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 filtreerides. 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" ] #jätab sisse kõik vektori colB väärtused, kus samas tabeli reas olev colA väärtus ei ole "a"
#> [1] 2 3
dat$colA[dat$colB > 1] #jätab sisse kõik vektori colA väärtused, kus samas tabeli reas olev colB väärtus >1.
#> [1] "b" "c"
```

0.4.9.3 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, "s"))
## this list has two elements, a data frame called "a" and a character vector called "b".
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     1.00
#> 2 B     2.00
```

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     2.00
```

```
aa3 <- my_list[[1]][1,]
aa3
#> # A tibble: 1 x 2
#>   colA   colB
#>   <chr> <dbl>
#> 1 A     1.00
```

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.

0.5 Regular expression ja find & replace

Regular expression annab võimaluse lühidalt kirjeldada mitte-üheseid otsinguparameetreid.

regular expression on string, mis kirjeldab mitut stringi

A regular expression⁶ Regular Expressions as used in R⁷

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).

⁶<https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html>

⁷<https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html>

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

Enamus määrke (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: `[[:süia_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
<code>\\n</code>	<code>\n</code> new line (return)
<code>\\t</code>	<code>\t</code> tab
<code>\\s</code>	<code>\s</code> any whitespace (<code>\S</code> - any non-whitespace)
<code>\\d</code>	<code>\d</code> any digit (<code>\D</code> - any non-digit)
<code>\\w</code>	<code>\w</code> any word character (<code>\W</code> - non-word char)
<code>\\b</code>	<code>\b</code> word boundaries

Selleks, et trükkida erimärk tavalise märgina:

trüki	selleks
<code>\\.</code>	<code>.</code>
<code>\\!</code>	<code>!</code>
<code>\\?</code>	<code>?</code>
<code>\\\\</code>	<code>\</code>
<code>\\(</code>	<code>(</code>
<code>\\{</code>	<code>{</code>

- 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 the `\\` **for** the string, so you will 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))`

0.5.1 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. kollapseeb 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
- järjestada stringe
- `str_sort()` annab sorditud character vectori

0.5.2 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] "apple" "enanas" "banana"

# str_replace only replaces at the first occurrence at each string
str_replace(x, "a", "e")
#> [1] "apple" "enanas" "banana"

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

#replaces a and the following character at the end of string with nothing (i.e. deletes 2 chars)
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] "apple" "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, "[a|p|l]", "_") #a or p or l to _
#> [1] "___e" "___e_ie"
str_replace_all(x, "[a|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 mõõtmisob-

jekti identiteeti ja tähe järel asuv number tähistab ajapunkti tundides (h). F ja f tähistavad sama asja. Kõigepealt võtame välja F-i mõõtmisobjekti 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 tühja stringiga
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.integer
#> [1] 1 2 1
```

0.6 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
# 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.

```
?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.

0.6.1 Kirjutame R funktsiooni

Võib ju väita, et funktsiooni ainus mõte on peita teie eest korduvad vajalikud koodiread koodi 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 arvulise väärtuse 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 Environmenti, 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())
## 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 standardvea 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 $N = \text{length}(x)$

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

`x` hoiab lihtsalt kohta andmetele, mida me tahame sinna funktsiooni 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 nõ. 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.

0.7 Graafilised lahendused

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

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

Kuigi “ggplot2” ja tema sateliit-raamatukogud on meie põhilised huviobjektid, 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(ggjoy)
library(wesanderson)
```

0.7.1 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         5.10         3.50         1.40         0.200
#> 2         4.90         3.00         1.40         0.200
#> 3         4.70         3.20         1.30         0.200
#> 4         4.60         3.10         1.50         0.200
#> 5         5.00         3.60         1.40         0.200
#> 6         5.40         3.90         1.70         0.400
#> # ... 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 vaikselt `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:

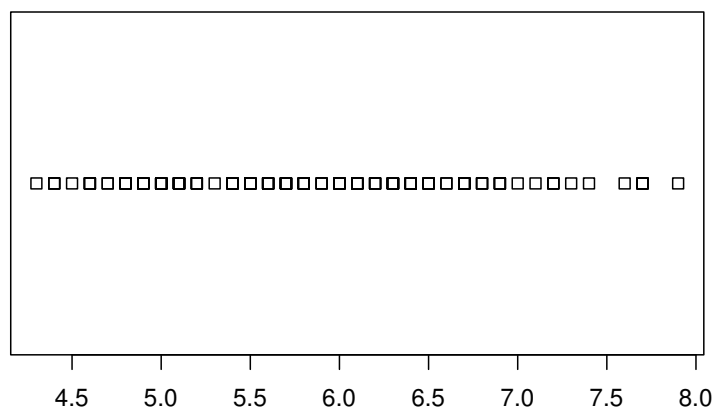
```
plot(iris$Sepal.Length)
```



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()`

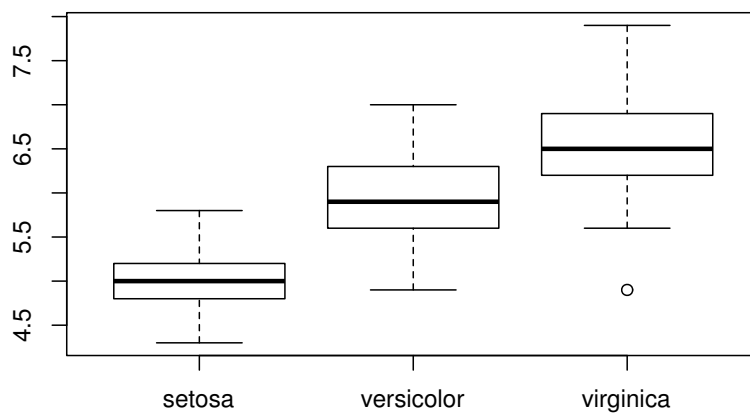
```
stripchart(iris$Sepal.Length)
```



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.

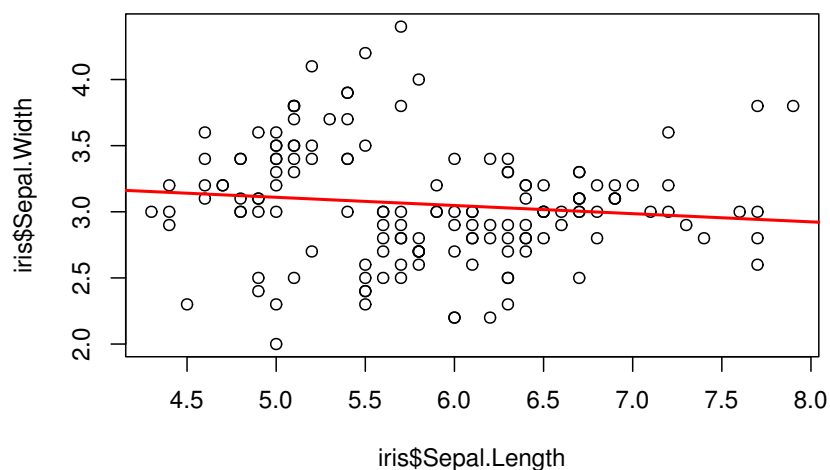
Järgmiseks kahe pideva muutuja scatterplot, kus me veel lisaks värvime punktid liikide järgi.

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



Ja lõpuks tõmbame läbi punktide punase regressioonijoone:

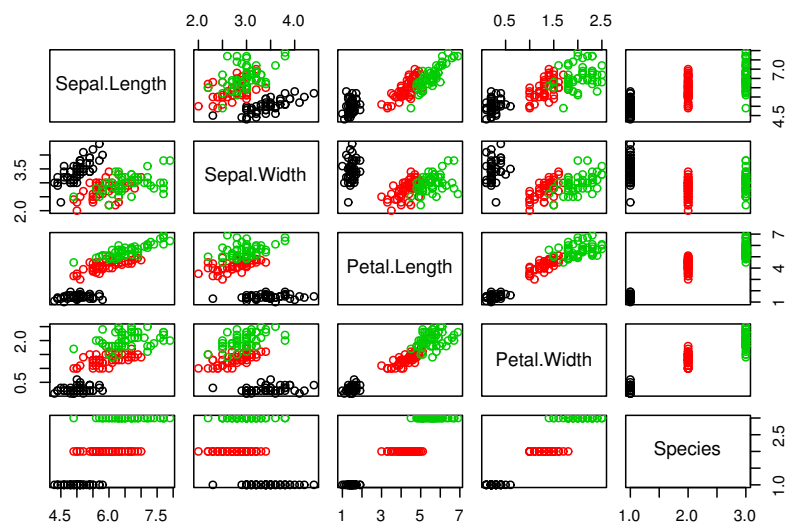
```
plot(iris$Sepal.Length, iris$Sepal.Width)
model <- lm(iris$Sepal.Width ~ iris$Sepal.Length)
abline(model, col = "red", lwd = 2)
```



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

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

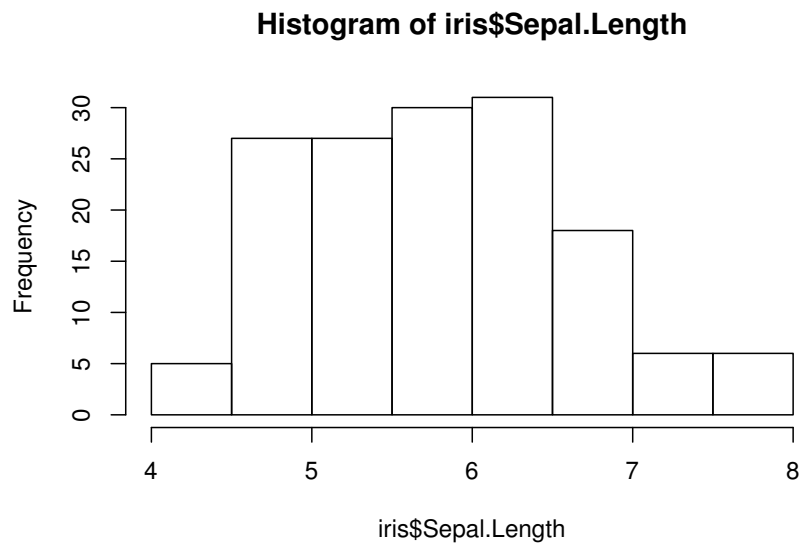
```
plot(iris, col = iris$Species)
```



Juhhei, tulemus on paariviisiline graafik kõigist muutujate kombinatsioonidest.

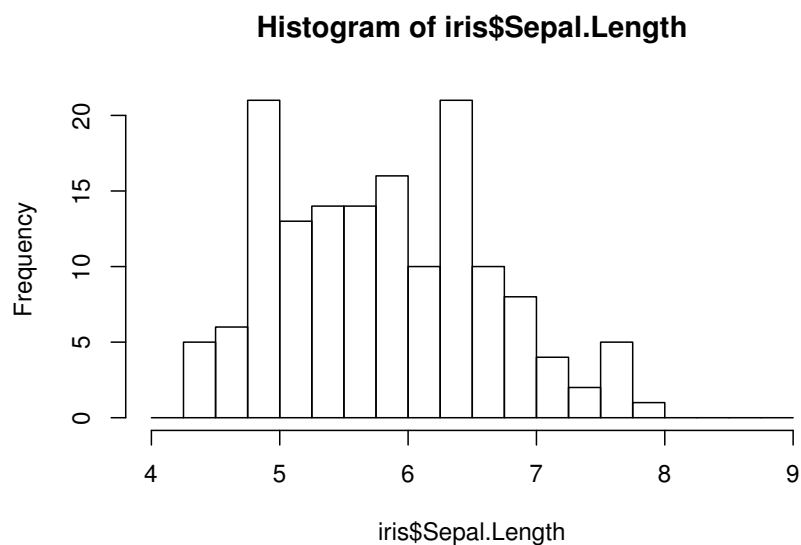
Ainus mitte-plot verb, mida baasgraafikas vajame, on `hist()`, mis joonistab histogrammi.

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

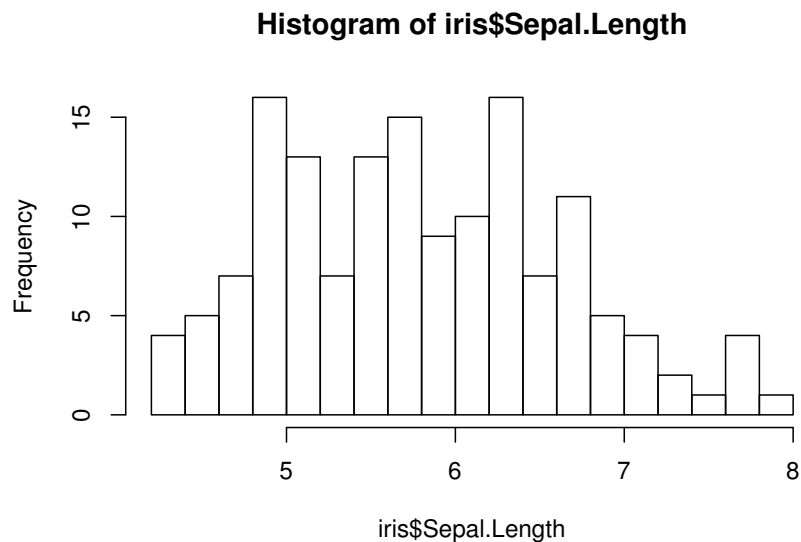


Histogrammi tegemiseks jagatakse andmepunktid nende väärtuste järgi bin-idesse ja plotitakse igasse bin-i sattunud andmepunktide arv. Näiteks esimeses bin-is on “Sepal.Length” muutuja väärtused, mis jäävad 4 ja 4.5 cm vahele ja selliseid väärtusi on kokku viis. Histogrammi puhul on oluline teada, et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

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



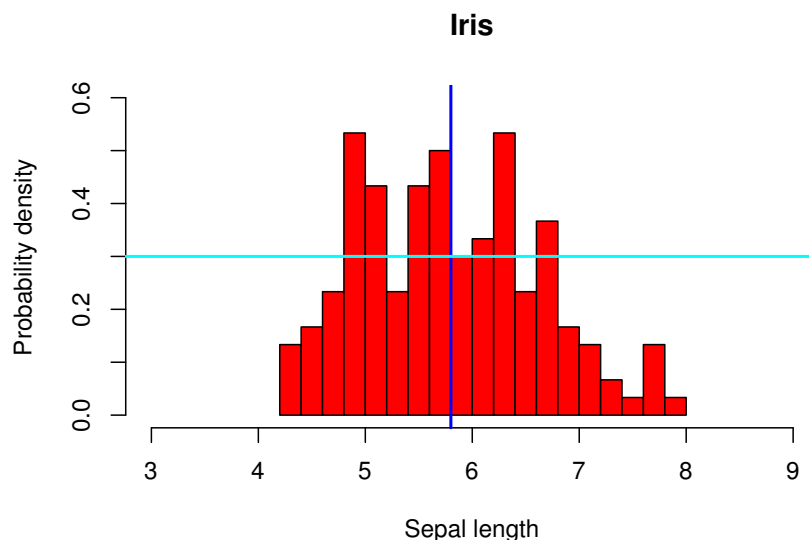
```
hist(iris$Sepal.Length, breaks = 15)
```



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)
```



0.7.2 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 arvates tasulised programmid!).

ggploti töövoog on järgmine, minimaalselt pead ette 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.

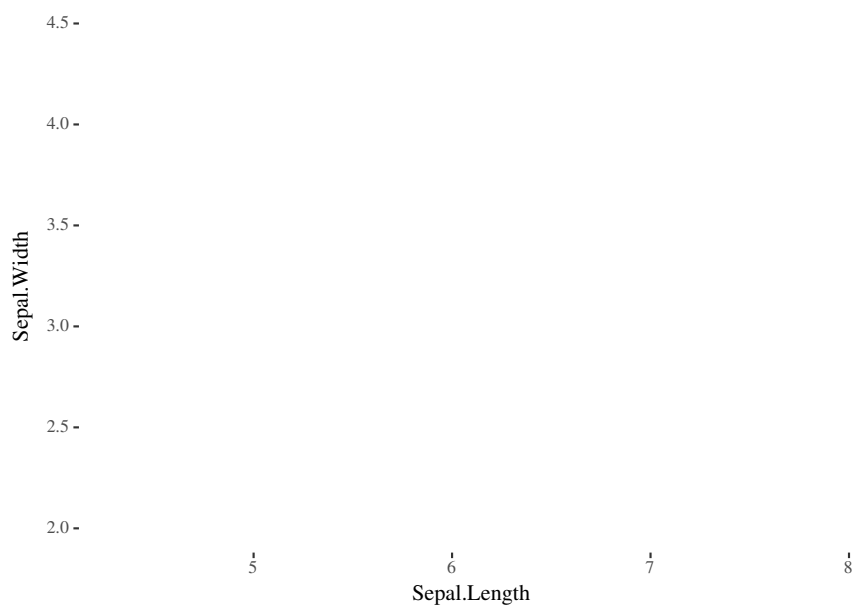
Lisaks määrad sa `aes()`-is, kas ja kuidas sa tahad grupeerida pidevaid muutujaid faktori tasemete järgi. Kõigepealt suuname oma andmed `ggplot()` funktsiooni:

```
ggplot(iris)
```

Saime tühja ploti. 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.

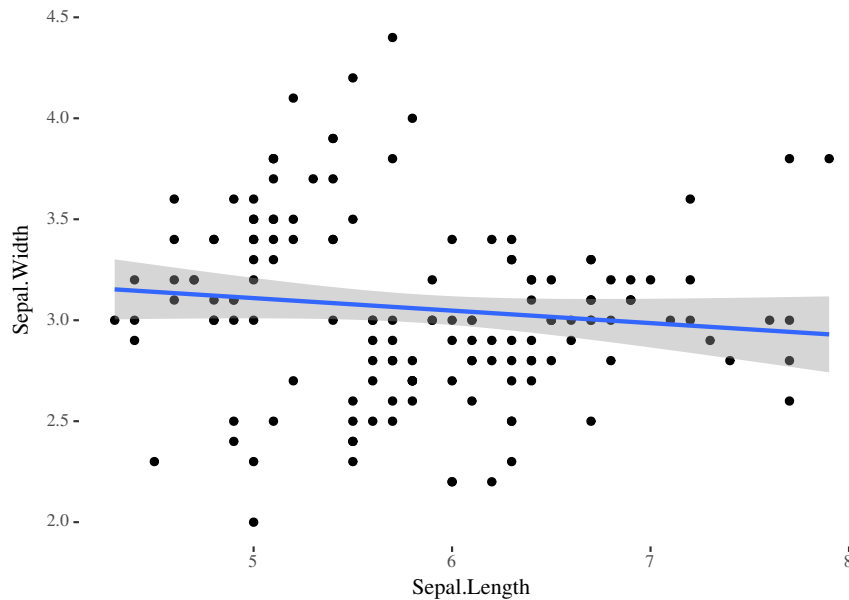
Nüüd ütleme, et x-teljele pannakse “Sepal.Length” ja y-teljele “Sepal.Width” andmed.

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



Aga graafik on ikka tühi sest me pole ggplotile öelnud, millist visualiseeringut me tahame. Teeme seda nüüd.


```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point() +  
  geom_smooth(method = "lm")
```



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

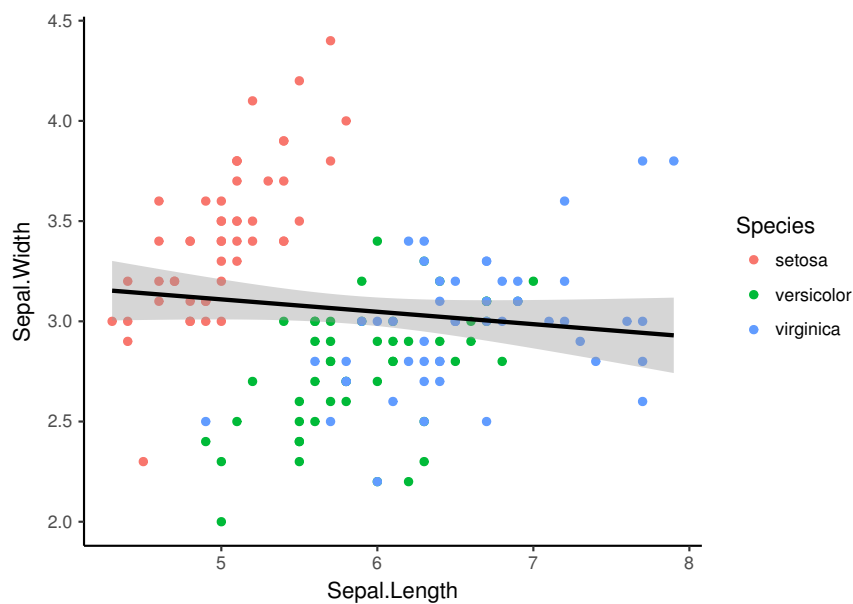
Plussmärk peab `ggplot`-i koodireas olema vana rea lõpus, mitte uue rea (kihi) alguses

0.7.3 Regressioonisirgete plottimine

Järgmiseks värvime eelnevalt tehtud plotil punktid iirise liigi kaupa aga joonistame ikkagi regressioonisirge läbi kõikide punktide.

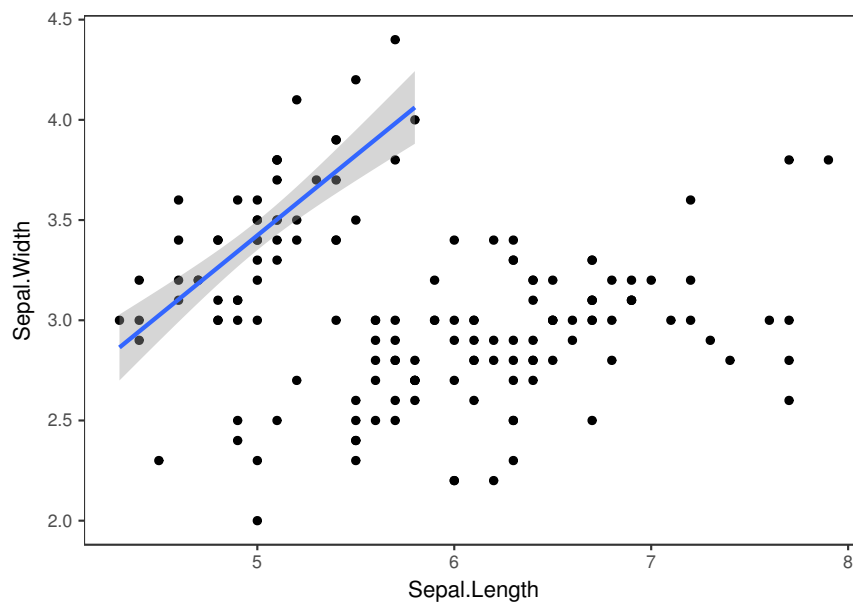
Vaata mis juhtub, kui värvide lahutamine toimub `ggplot()`-i enda `aes()`-s. `theme_classic()` muudab graafiku üldist väljanägemist.

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



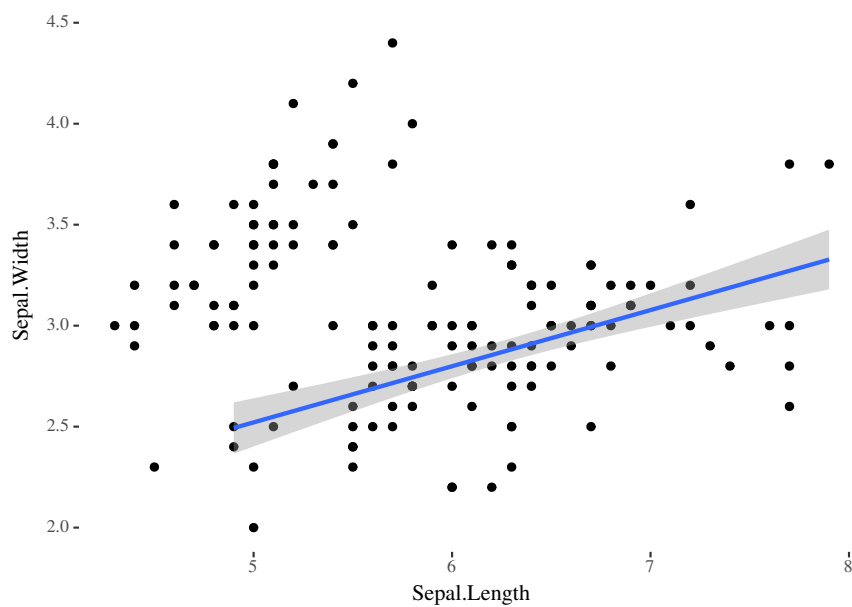
Me võime `geom_smooth()`-i anda erineva andmeseti kui `ggplot()` põhifunktsiooni. Nii joonistame me regressioonisirge ainult nendele andmetele. Proovi ka `theme_bw()`.

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



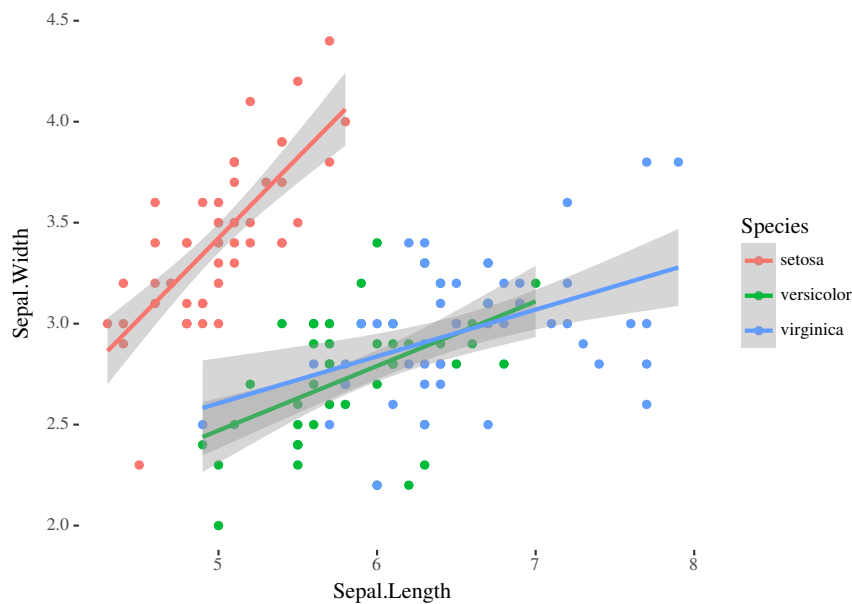
Alljärgnevalt näiteks moodus kuidas öelda, et me soovime regressioonijoont näidata ainult iiriseliikide *virginica* või *versicolor* andmetele.

```
## 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 = Sepal.Width)) +  
  geom_point() +  
  geom_smooth(data = smooth_data, method = lm)
```



Ja lõpuks joonistame kolm regressioonisirget – üks igale liigile.

```
iris %>% ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +  
  geom_point() +  
  geom_smooth(method = "lm")
```



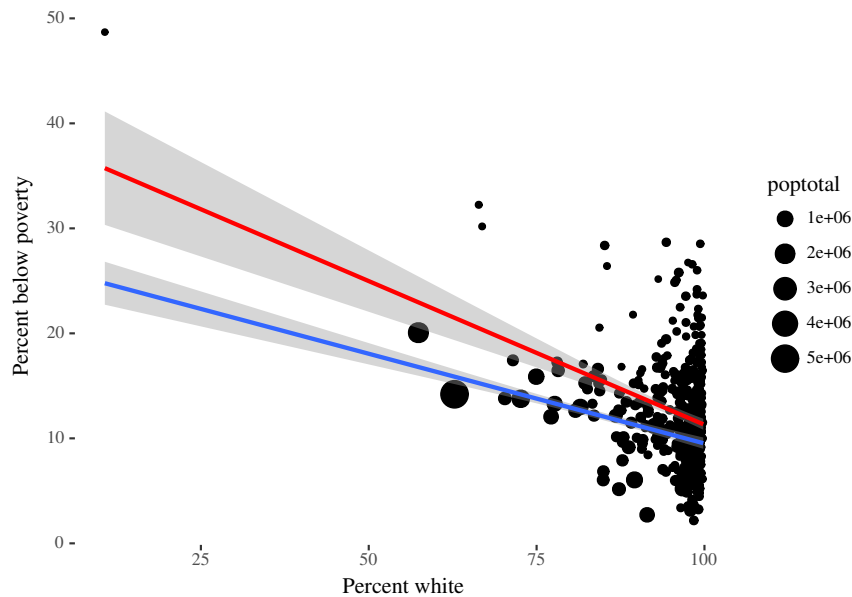
Nüüd üks näide teiste andmetega. Kaalutud lineaarne mudel on viis anda andmepunktidele, mida me tähtsamaks peame (või mis on täpsemalt mõõdetud) suurem kaal. Kõigepealt, siin on USA demograafilised andmed `midwest` “`ggplot2`” library-st erinevate kesk-lääne omavalitsuste kohta (437 omavalitsust).

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

```
midwest_subset <- midwest %>% select(percwhite, percbelowpoverty, poptotal)
midwest_subset
#> # A tibble: 437 x 3
#>   percwhite percbelowpoverty poptotal
#>   <dbl>         <dbl>     <int>
#> 1    96.7           13.2    66090
#> 2    66.4           32.2    10626
#> 3    96.6           12.1    14991
#> 4    95.3            7.21   30806
#> 5    90.2           13.5     5836
#> 6    98.5           10.4   35688
#> # ... with 431 more rows
```

Me tahame teada, kuidas valge rassi osakaal ennustab vaesust, aga me arvame, et suurematel omavalitsustel peaks selles ennustuses olema suurem kaal kui väiksematel. 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")
```



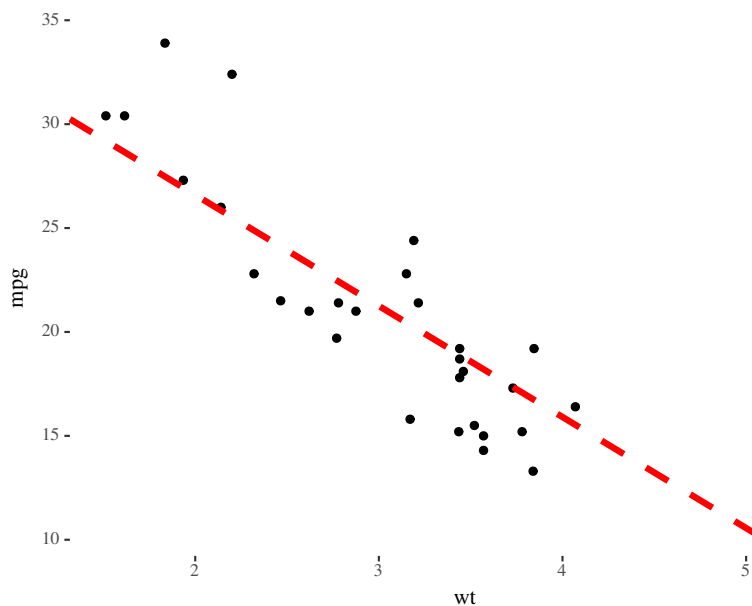
Punane on kaalumata regressioonisirge ja sinine on populatsioonisuuruse suhtes kaalutud regressioonisirge. Kaalumine mitte ainult ei muutnud sirge asukohta vaid vähendas ka ebakindlust sirge asukoha kohta.

Regressioonijoone saab ggplotil määrata ka x-telje lõikumispunkti ja tõusu abil. See on kasulik mudelite visualiseerimisel mudeli koefitsientide põhjal. 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)
```

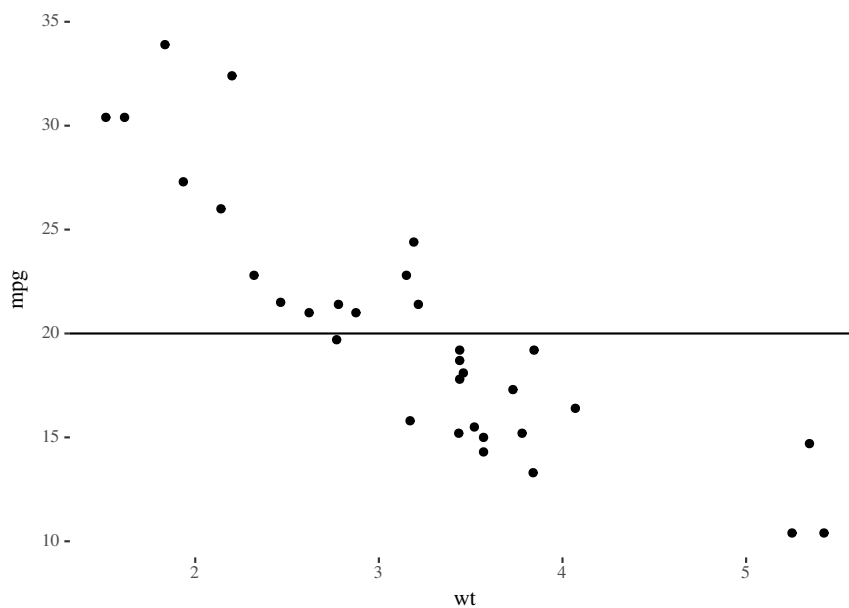


0.7.3.1 Lisame plotile sirgjooni

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.

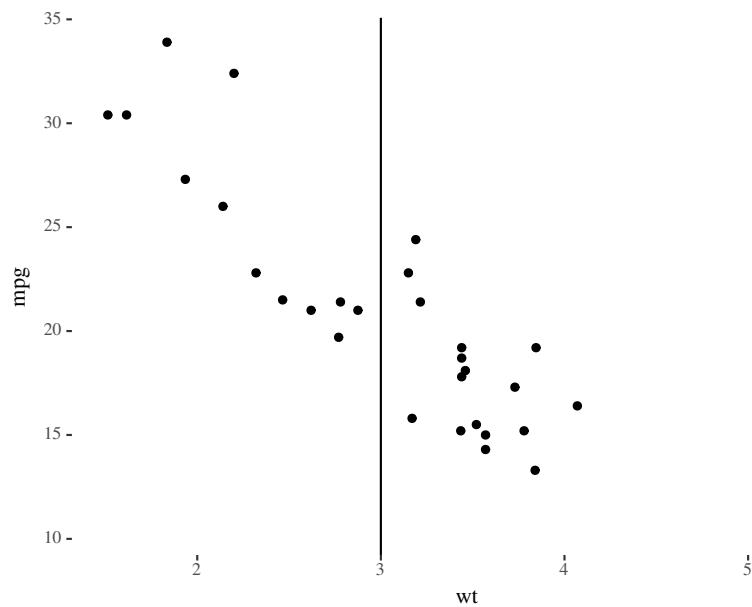
Lisame graafikule horisontaaljoone $y = 20$:

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



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

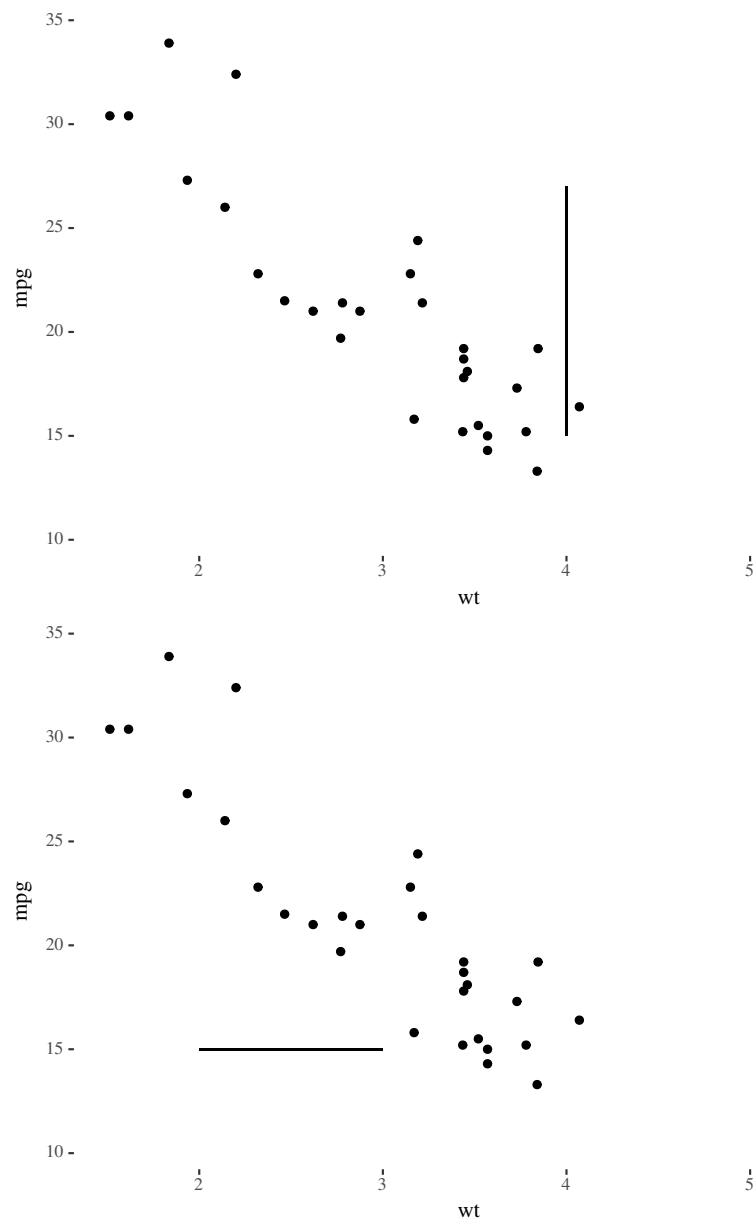
```
# Add a vertical line at x = 3  
p + geom_vline(xintercept = 3)
```



0.7.3.2 Segmendid ja nooled

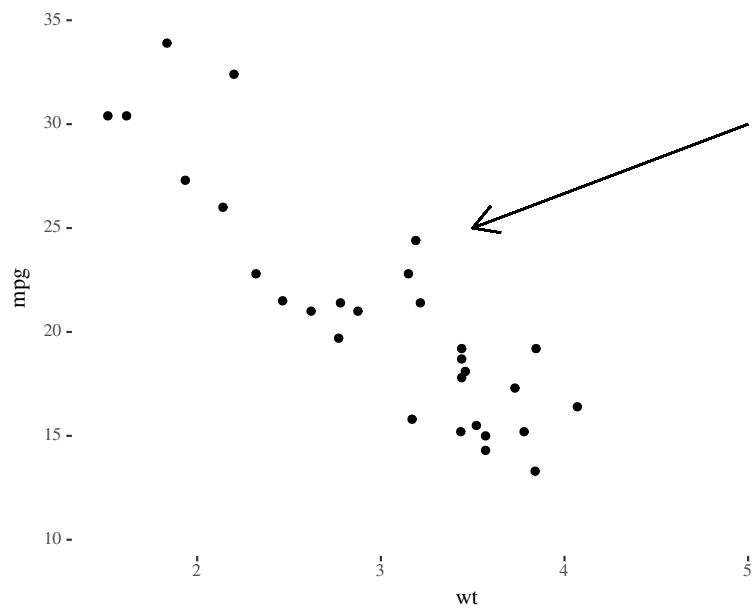
“ggplot2” funktsioon `geom_segment()` lisab joonejupi, mille algus ja lõpp on ette antud.

```
# 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")))
```

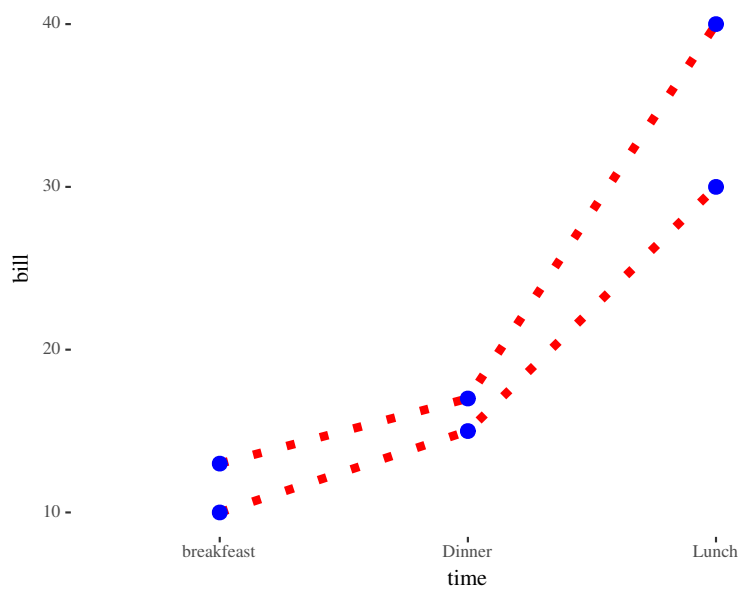



0.7.3.3 Joongraafikud

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

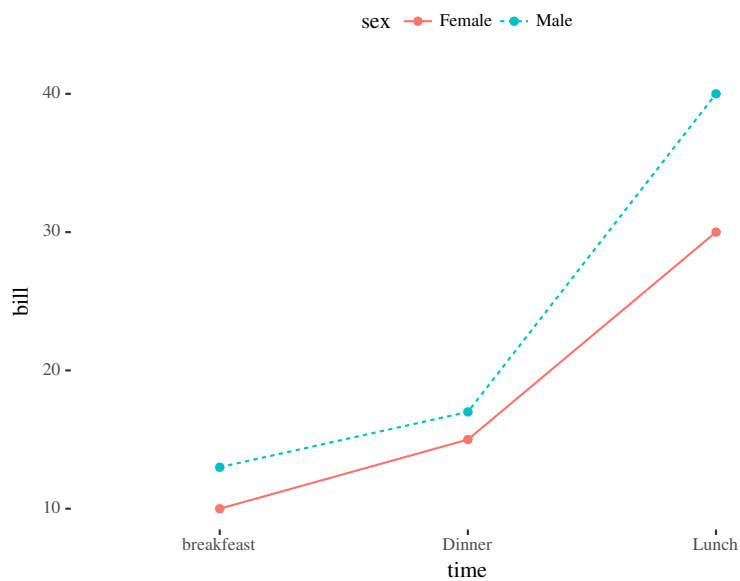
```
meals <- data.frame(sex = rep(c("Female", "Male"), each = 3),
  time = c("breakfast", "Lunch", "Dinner"),
  bill = c(10, 30, 15, 13, 40, 17) )

# 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")
```

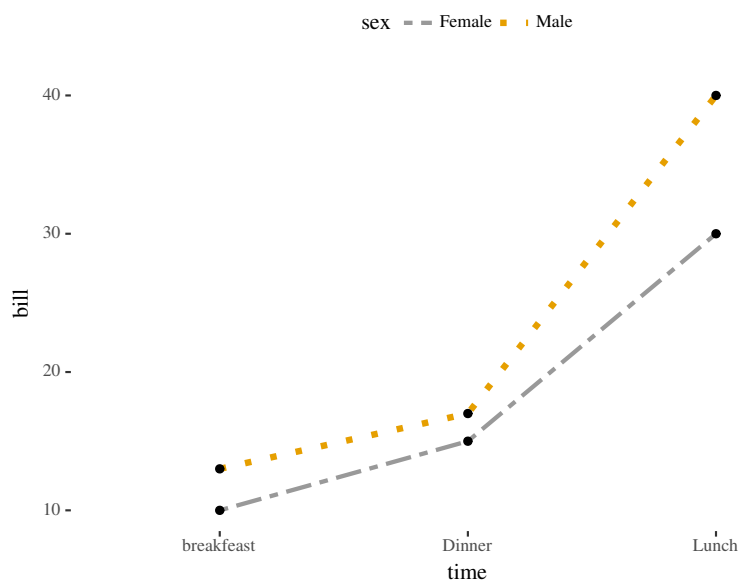


Muuda jooni käsitsi:

- `scale_linetype_manual()`: joone tüüp

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

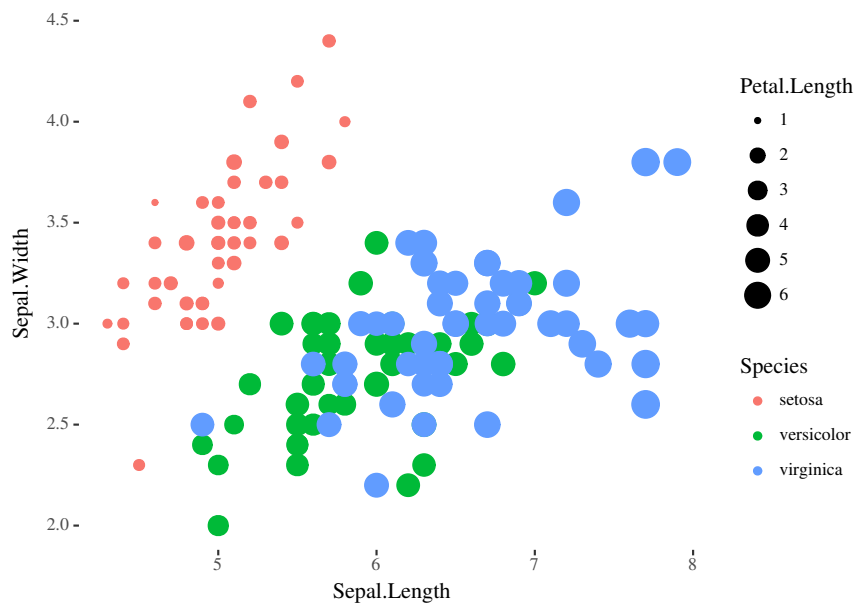
```
ggplot(meals, aes(x = time, y = bill, group = sex)) +
  geom_line(aes(linetype = sex, color = sex, size = sex)) +
  geom_point() +
  scale_linetype_manual(values = c("twodash", "dotted")) +
  scale_color_manual(values = c('#999999', '#E69F00')) +
  scale_size_manual(values = c(1, 1.5)) +
  theme(legend.position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



0.7.3.4 Punktide tähistamise trikid

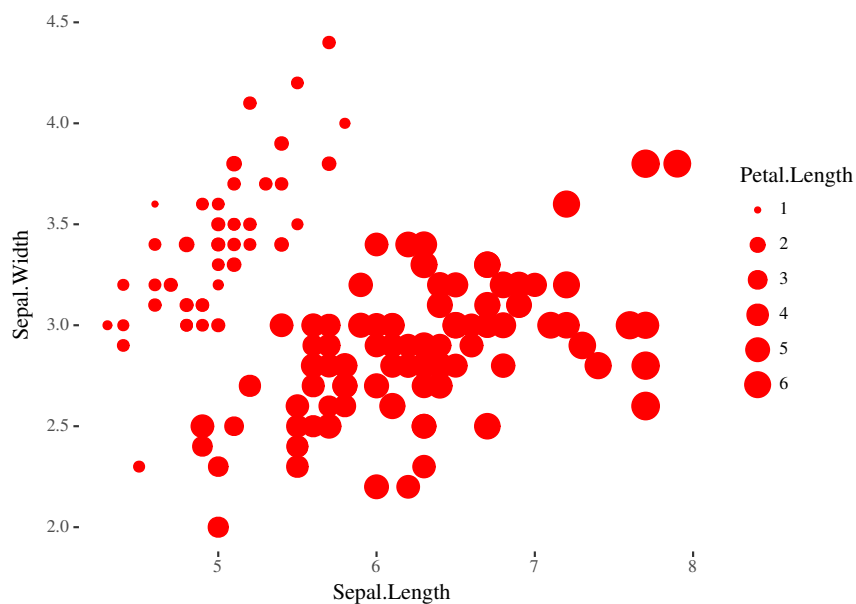
`aes()` töötab nii `ggplot()` kui `geom_` funktsioonides.

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



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), color = "red")
```



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
```

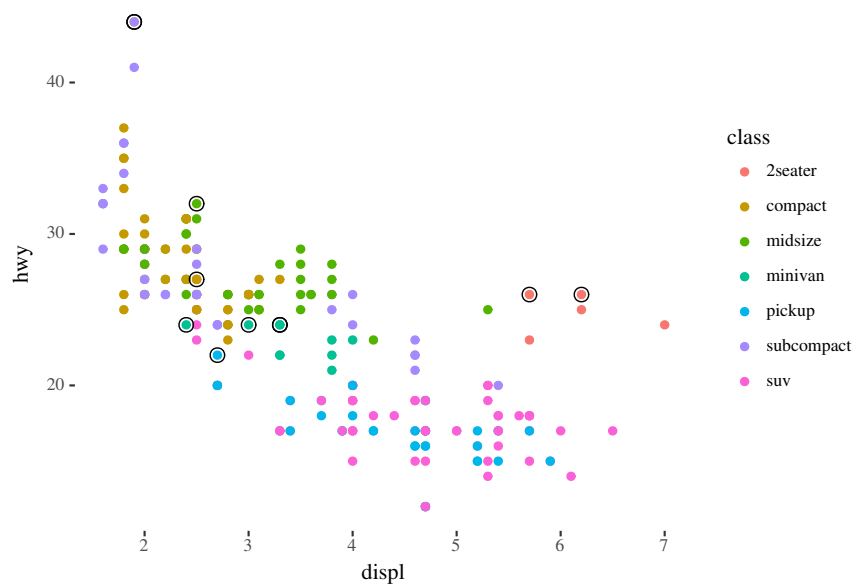
```
#>   <chr>         <chr> <dbl> <int> <int> <chr>   <chr>
#> 1 audi         a4     1.80  1999     4 auto(l5) f
#> 2 audi         a4     1.80  1999     4 manual(m~ f
#> # ... with 4 more variables: cty <int>, hwy <int>,
#> #   fl <chr>, class <chr>

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    corvette  5.70  1999     8 manua~ r
#> 2 chevrolet    corvette  6.20  2008     8 manua~ r
#> 3 dodge        caravan~  2.40  1999     4 auto(~ f
#> 4 dodge        caravan~  3.00  1999     6 auto(~ f
#> 5 dodge        caravan~  3.30  2008     6 auto(~ f
#> 6 dodge        caravan~  3.30  2008     6 auto(~ f
#> # ... with 4 more variables: cty <int>, hwy <int>,
#> #   fl <chr>, class <chr>
```

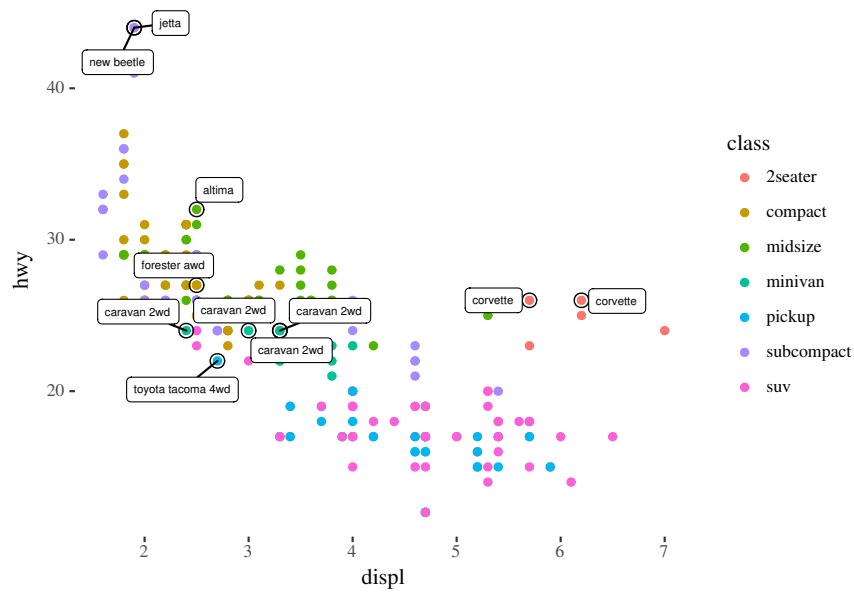
Siin läheb kitsam andmeset uude `geom_point()` kihti ja teeb osad punktid teistsuguseks. Need on oma klassi parimad autod.

```
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)
```

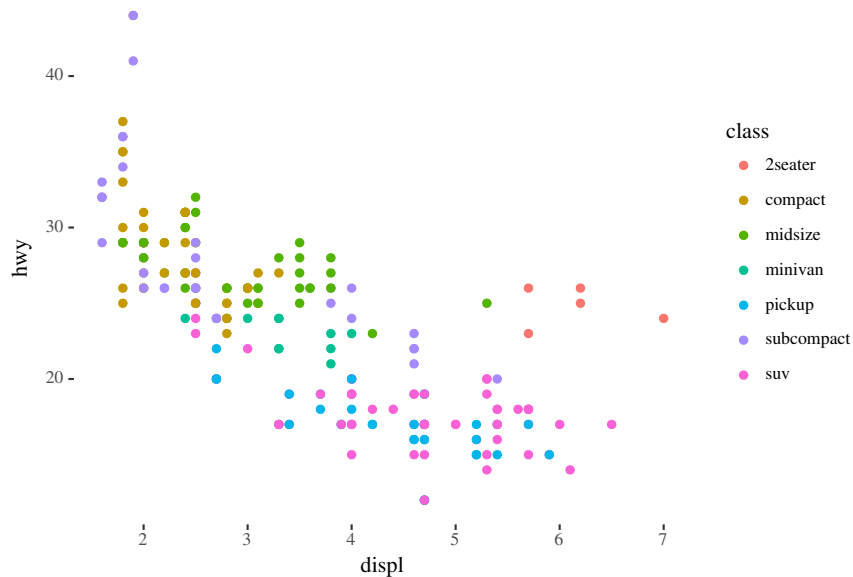


0.7.4 Facet – pisigraafik

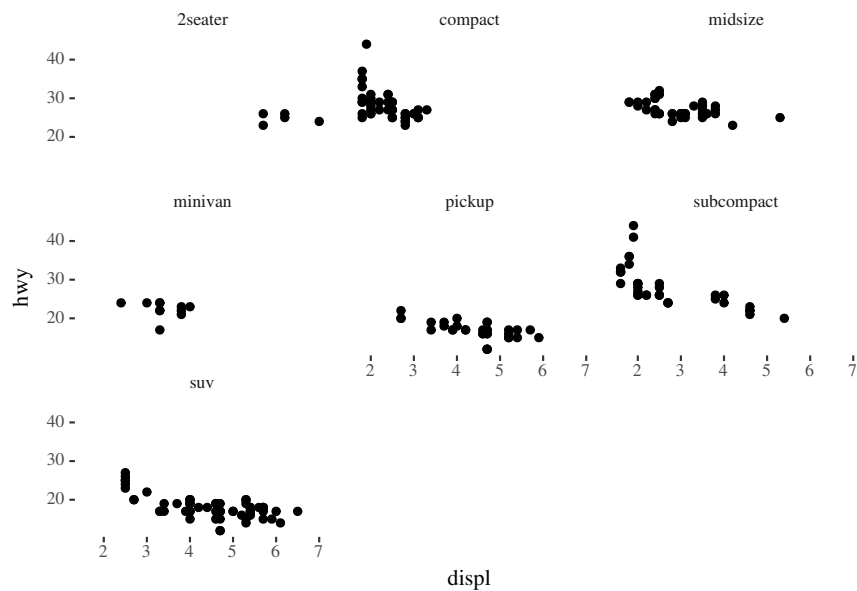
Kui teil on mitmeid muutujaid või nende alamhulki, on teil kaks võimalust.

1. grupeeri pidevad muutujad faktormuutujate tasemetega 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.

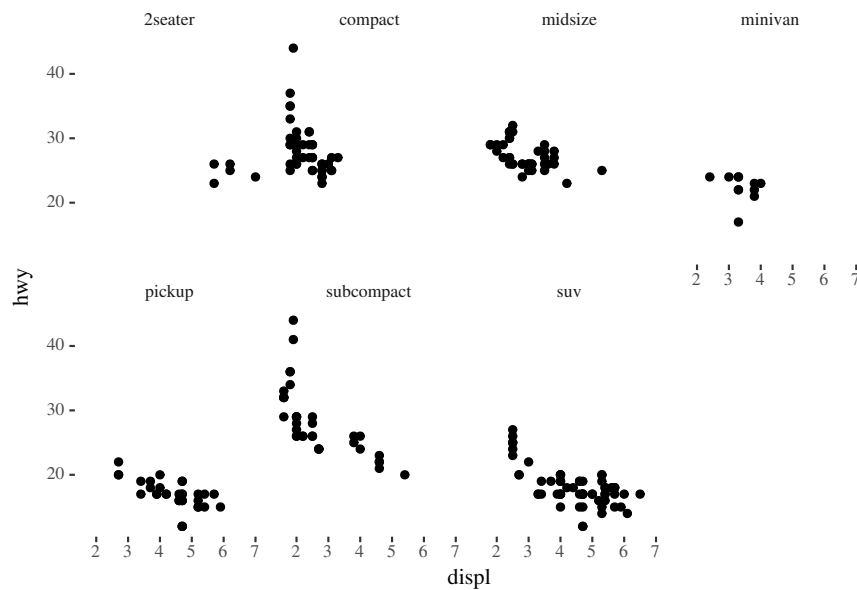
```
#here we separate different classes of cars into different colors  
p <- ggplot(mpg, aes(displ, hwy))  
p + geom_point(aes(colour = class))
```



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

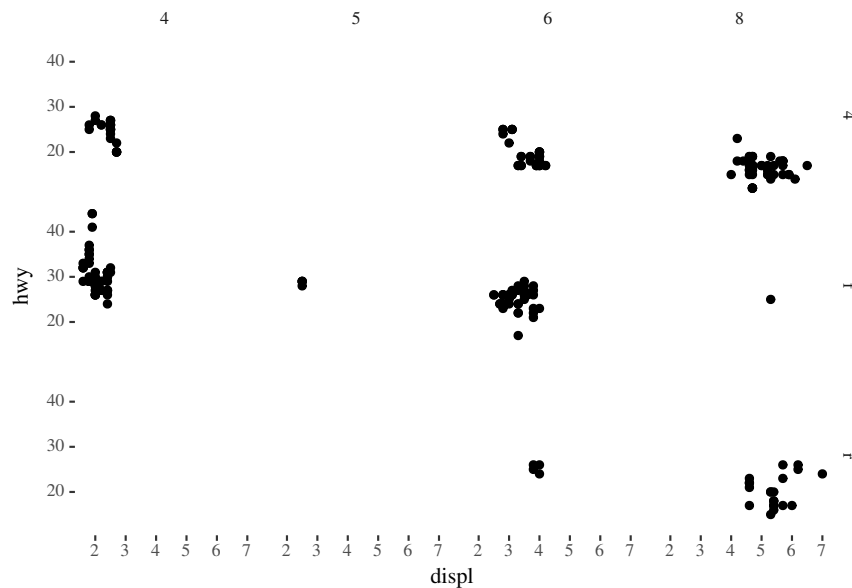


```
p + geom_point() +  
  facet_wrap(~ class, nrow = 2)
```



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

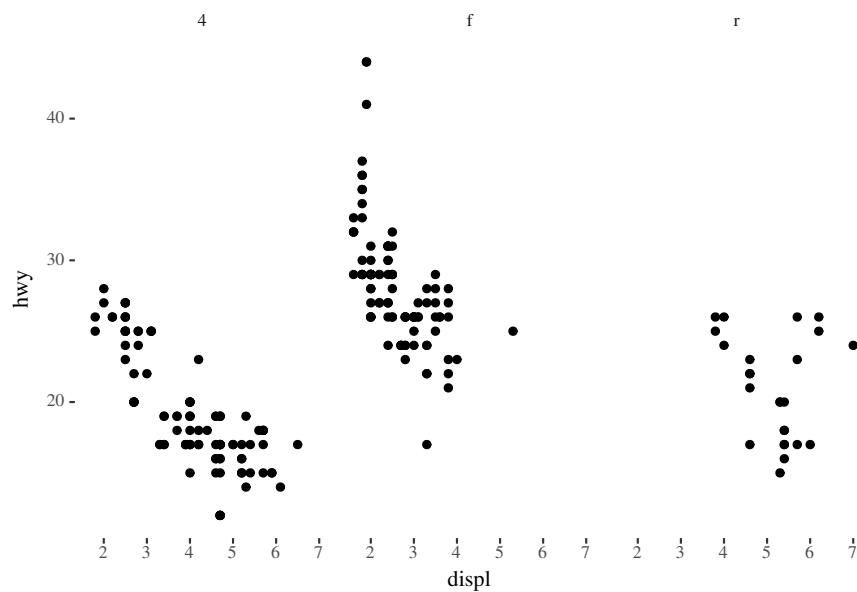

```
p + geom_point() +  
  facet_grid(drv ~ cyl)
```



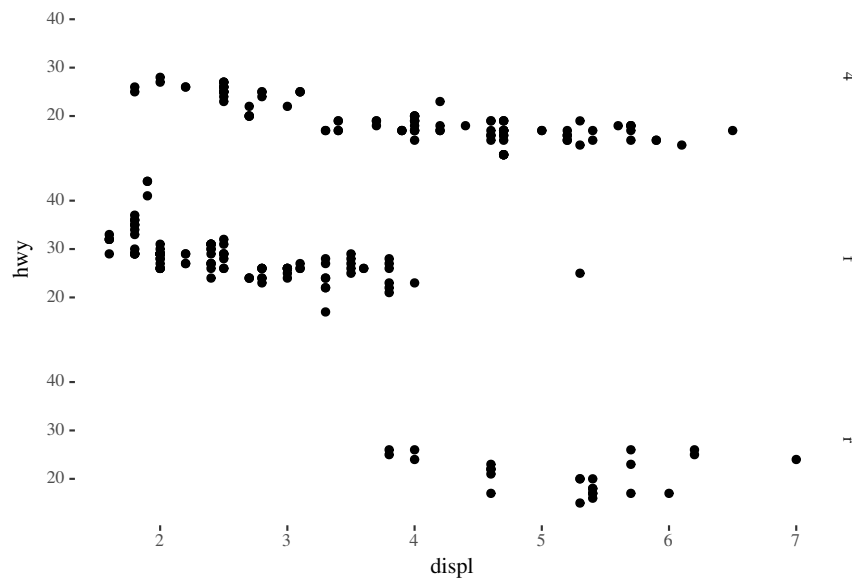
- “drv” – drive - 4(-wheel), f(oward), r(ear).
- “cyl” – cylinders - 4, 5, 6, or 8.

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 ~ .)
```



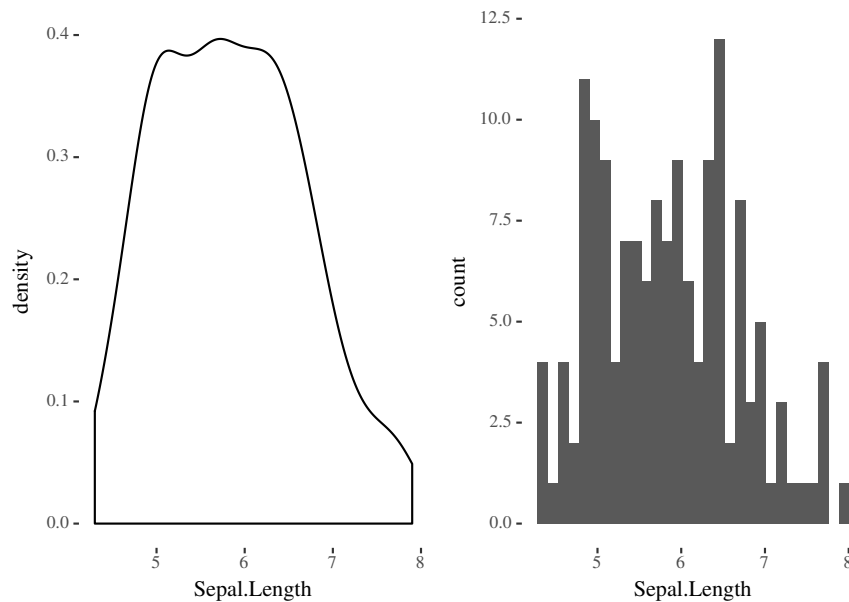
0.7.5 Mitu graafikut paneelidena ühel joonisel

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
```



o.7.6 Teljed

o.7.6.1 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` on parameetrid selle sees: `coord_cartesian(xlim = c(5, 7), ylim = c(10, 30))`

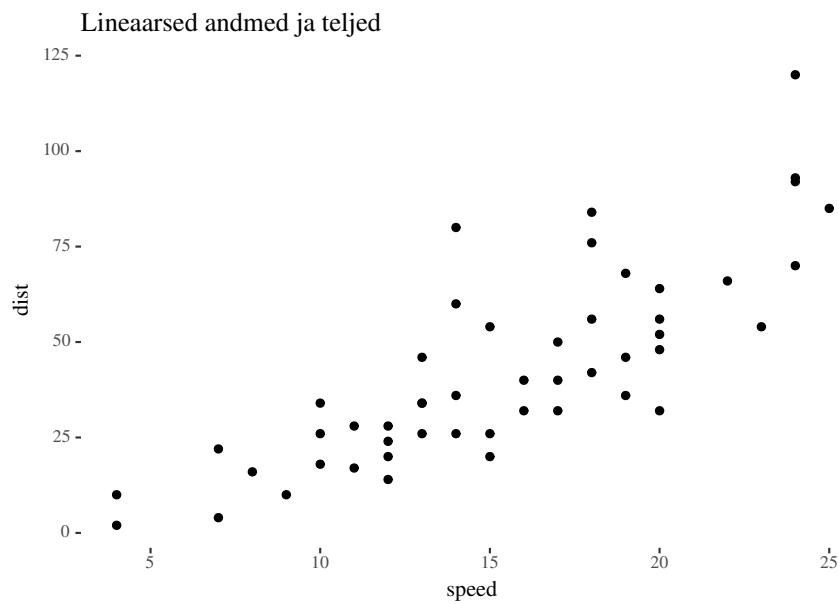
Telgede ulatust saab muuta ka x- ja y-teljele eraldi:

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

o.7.6.2 Log skaalas teljed

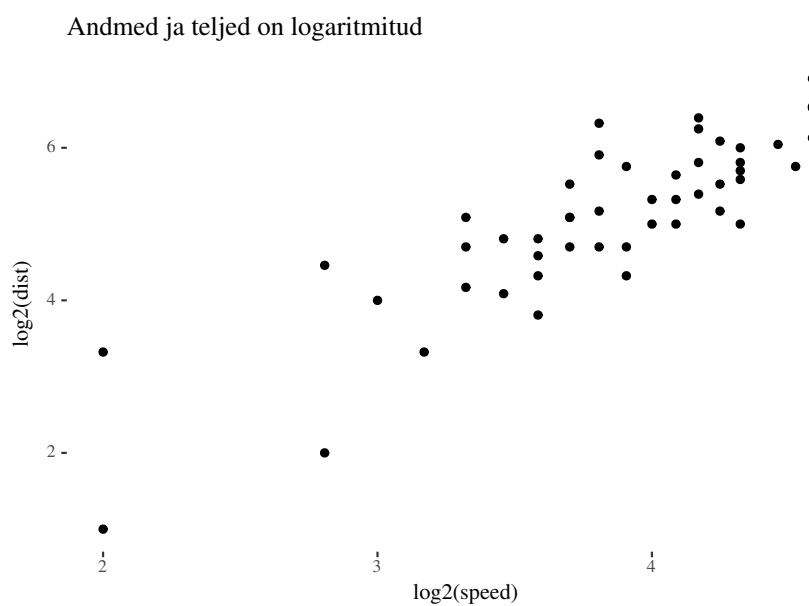
1. Lineaarsed andmed lineaarsetel telgedel.

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



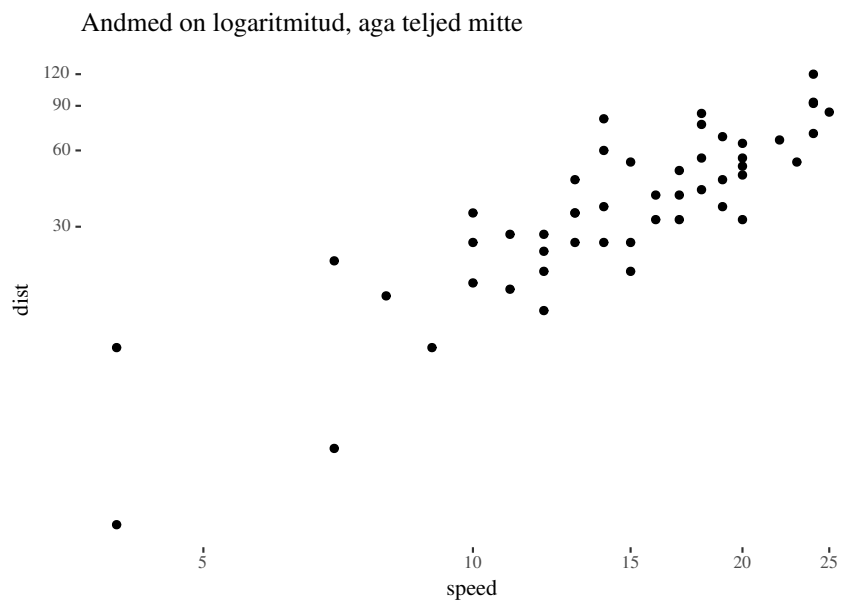
2. Logaritmi andmed `aes()`-s.

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



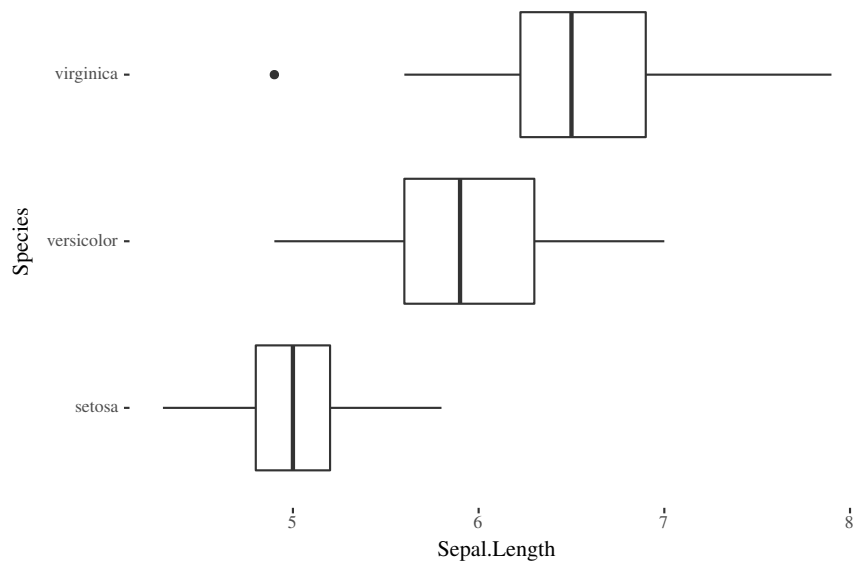
3. Andmed on logaritmitud, aga teljed mitte.

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



0.7.6.3 Pöörame graafikut 90 kraadi

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



0.7.6.4 Muudame telgede markeeringuid

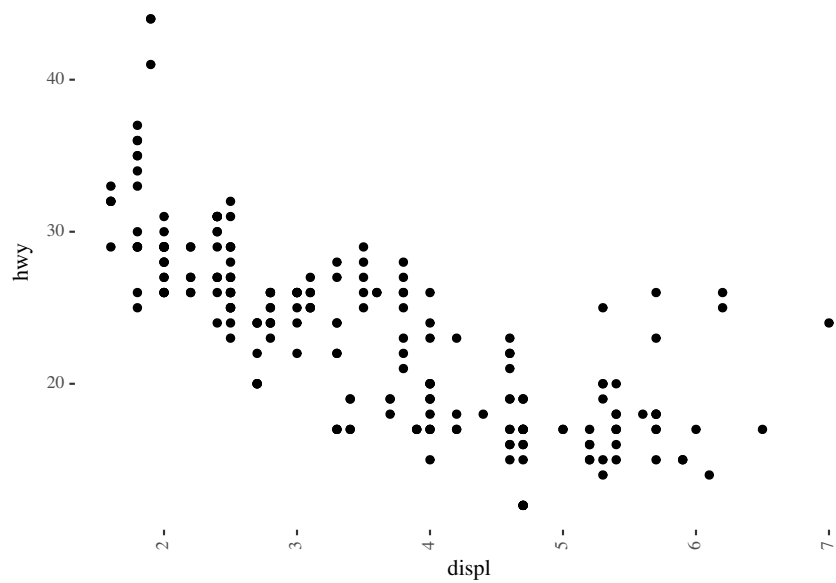
Muudame y-telje markeeringut:

```
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")
```



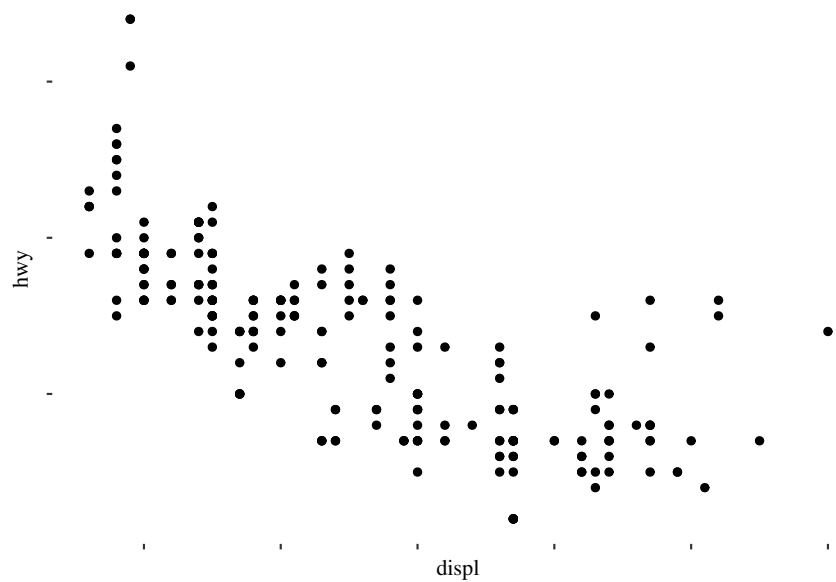
Muudame x-telje markeeringute nurka muutes `theme()` funktsiooni argumenti “axis.text.x”:

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



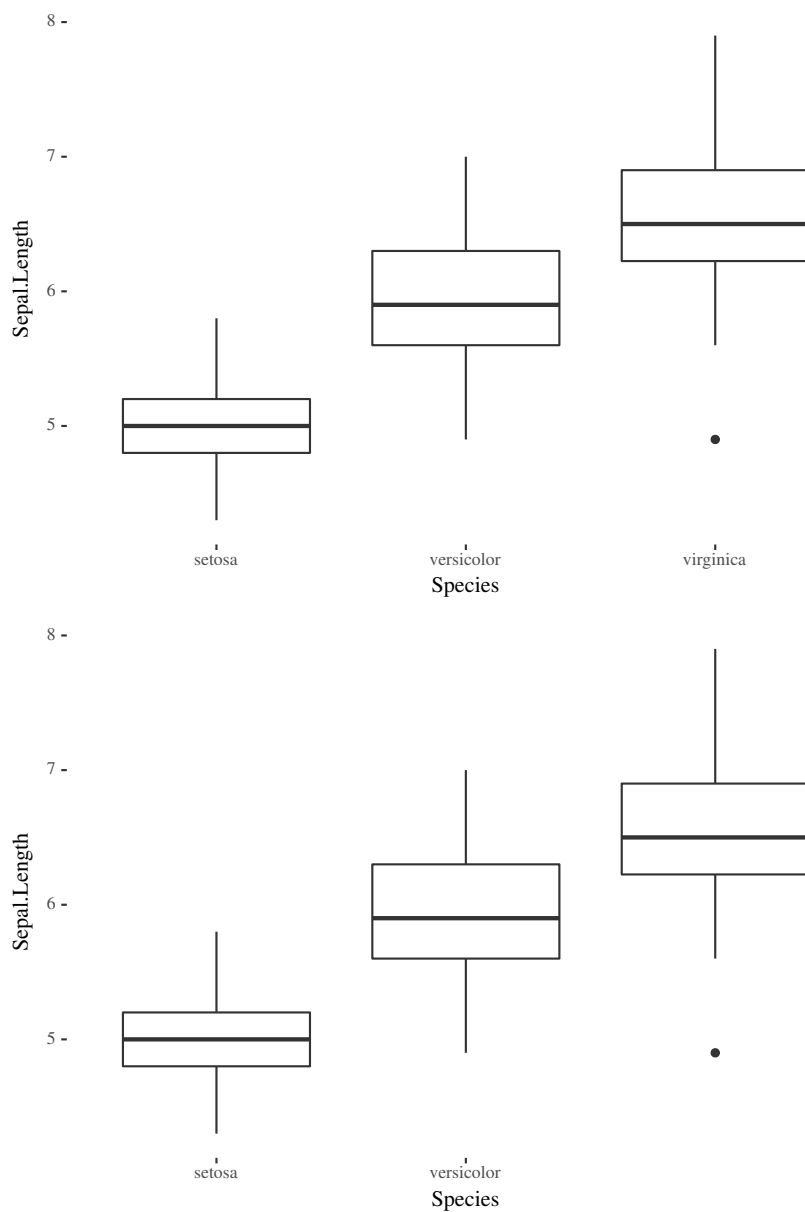
Eemaldame telgede markeeringud, ka läbi `theme()` funktsiooni:

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



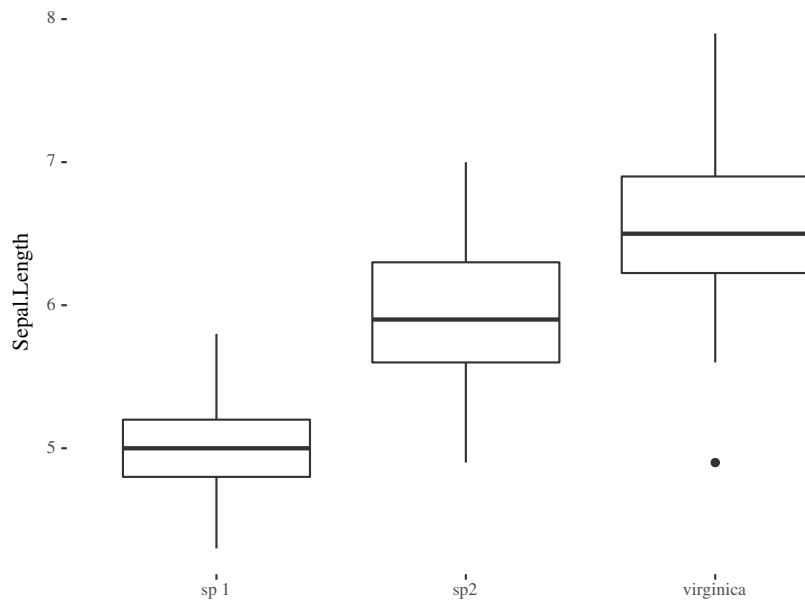
Muudame teljemarkeeringute järjekorda

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



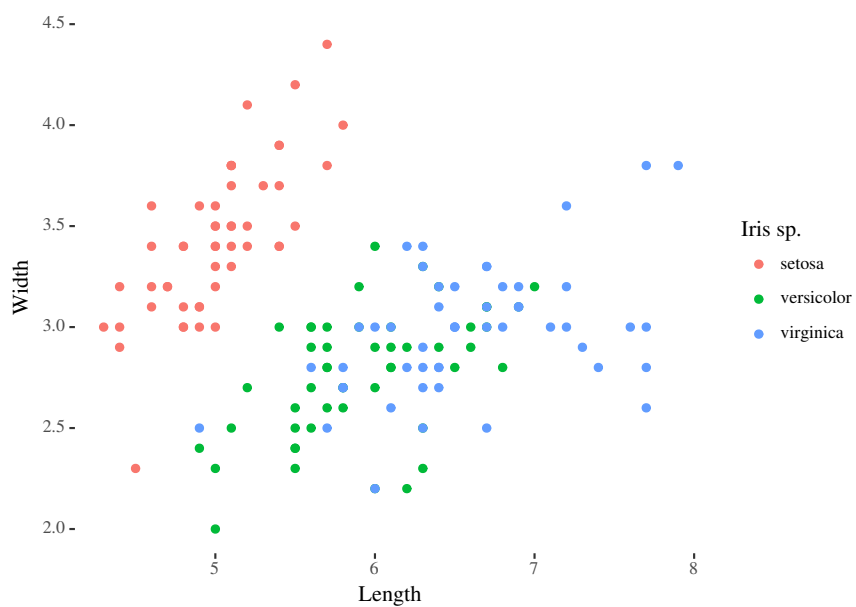
Muuda teljemarkeringuid ja kustuta telje nimi.

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

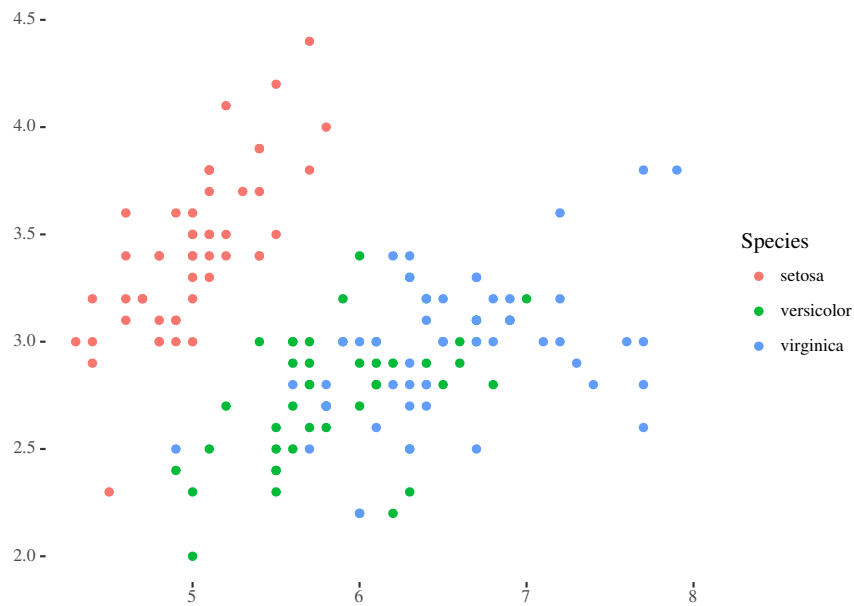
0.7.6.5 telgede ja legendi nimed

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



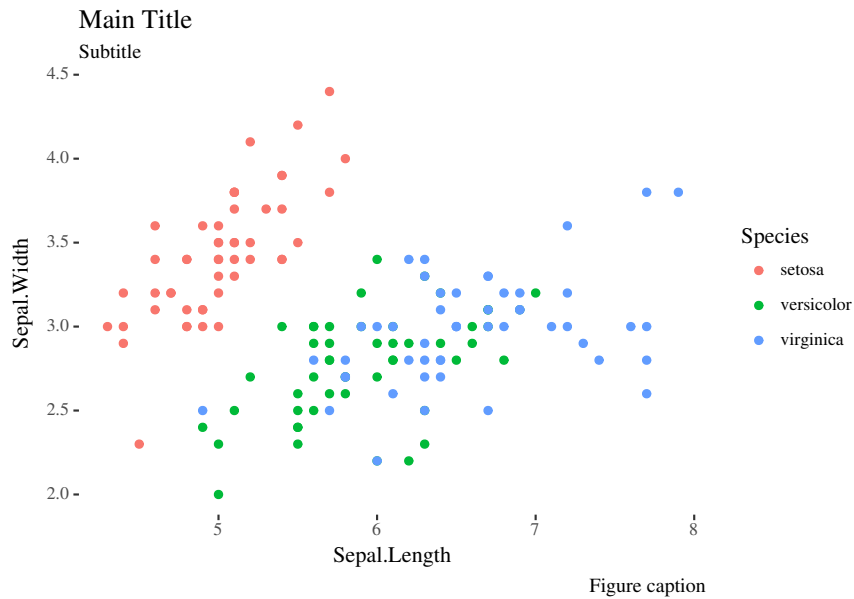
Eemaldame telgede nimed:

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



o.7.7 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

0.7.8 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 column

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, 0.5))
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
```

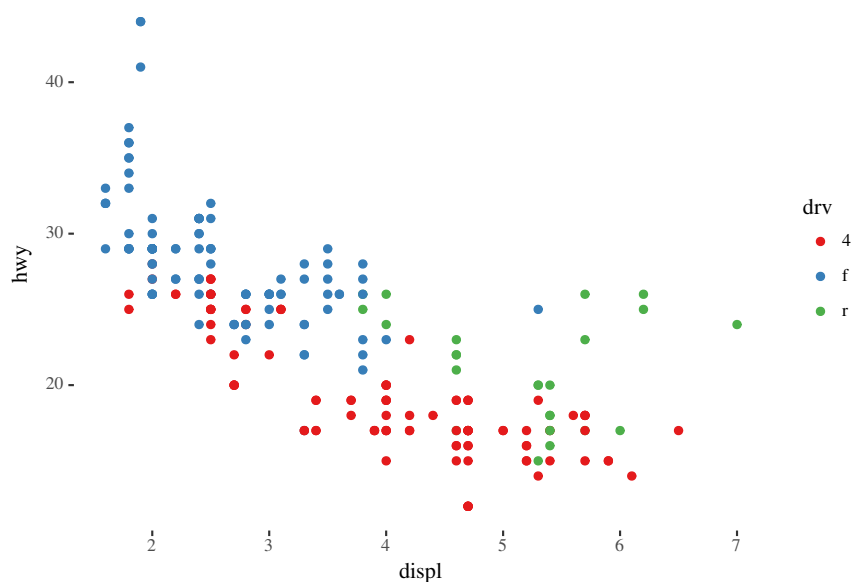
eemalda legend

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

0.7.9 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")
```



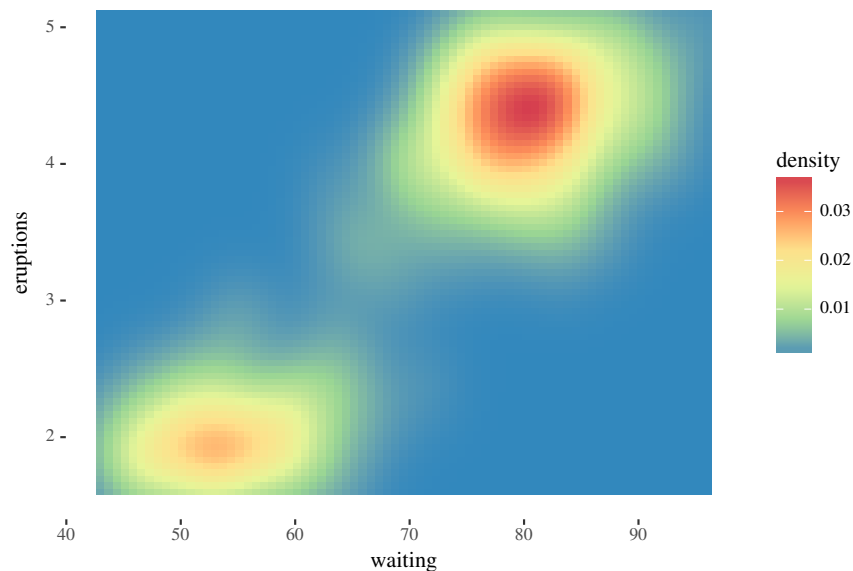
0.7.9.1 Värviskaalad pidevatele muutujatele

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



0.7.9.2 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"))
```

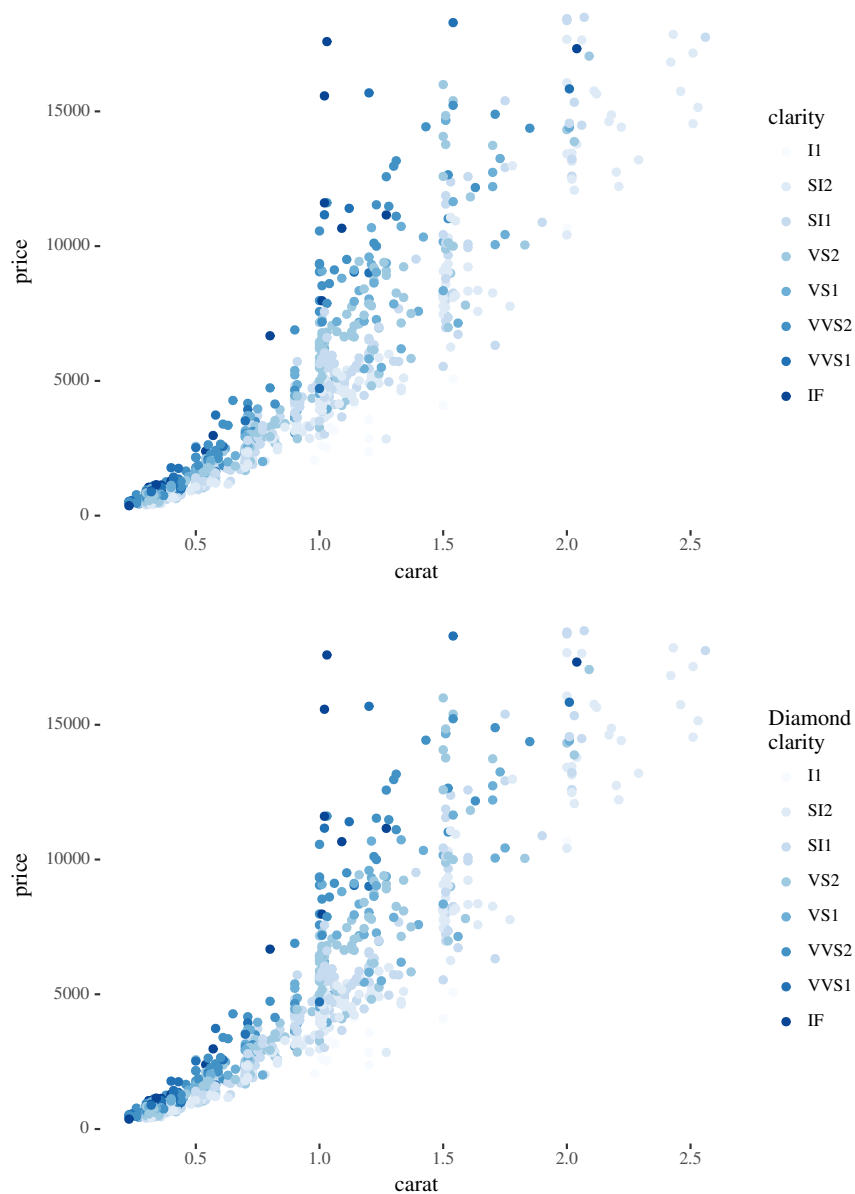
Colour_brewer-i skaalad on loodud faktormuutujaid silmas pidades.

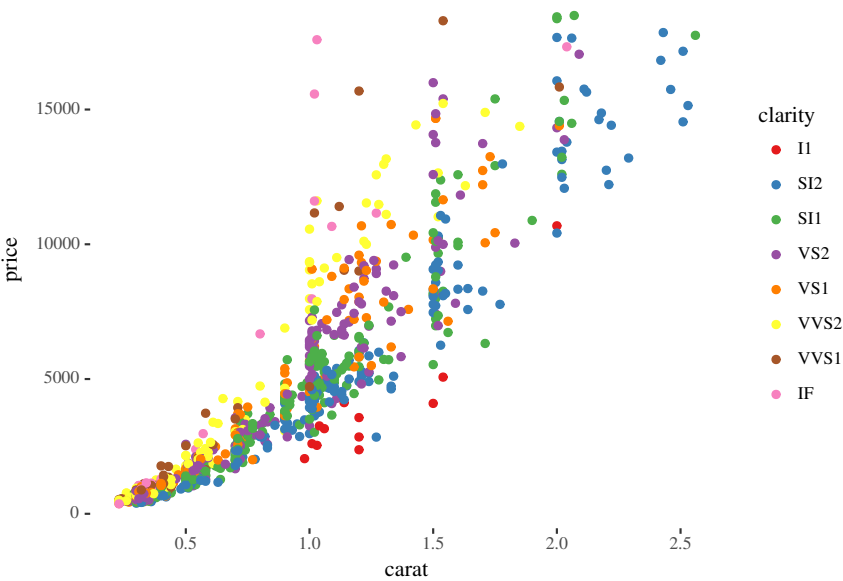
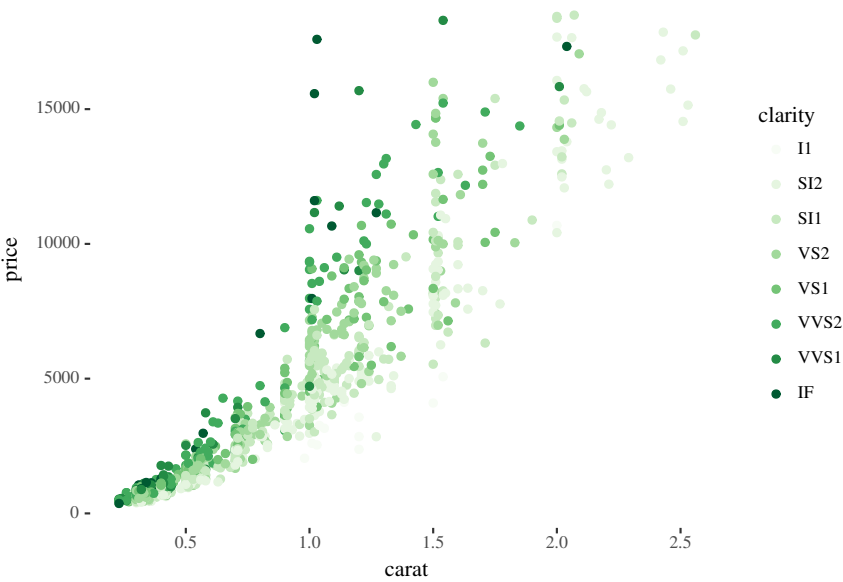
```
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
d <- ggplot(dsamp, aes(carat, price)) +
  geom_point(aes(colour = clarity))
d + scale_colour_brewer()

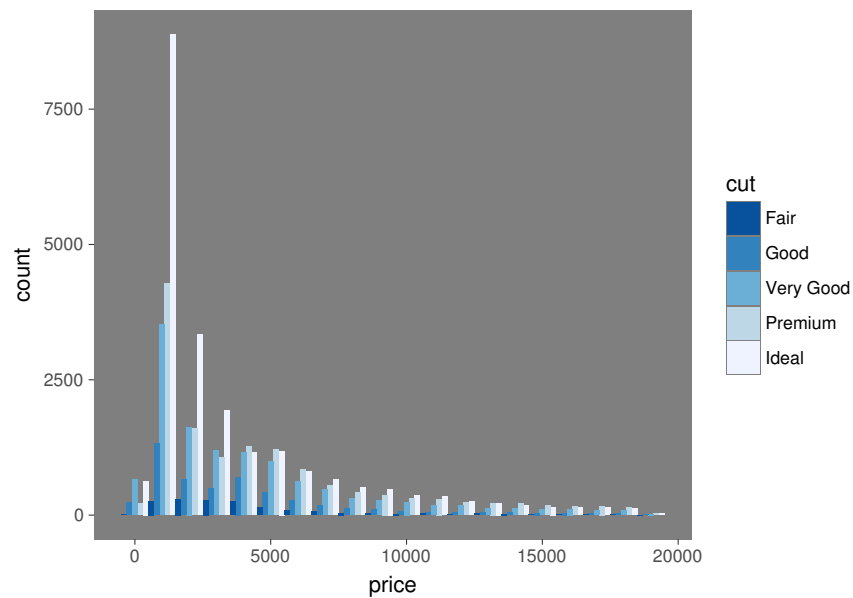
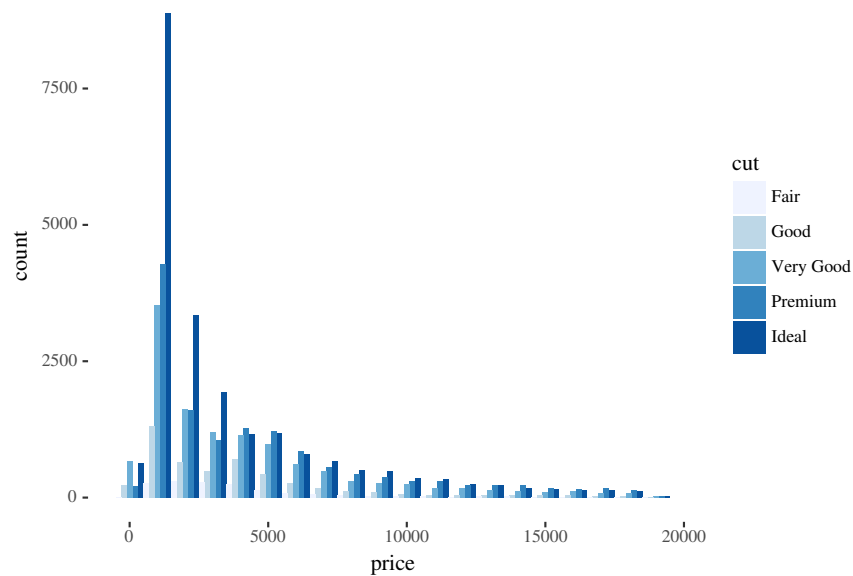
# Change scale label
d + scale_colour_brewer("Diamond\nclearity")

# 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()
```







Väga lähedad 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 continuous scale?
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.

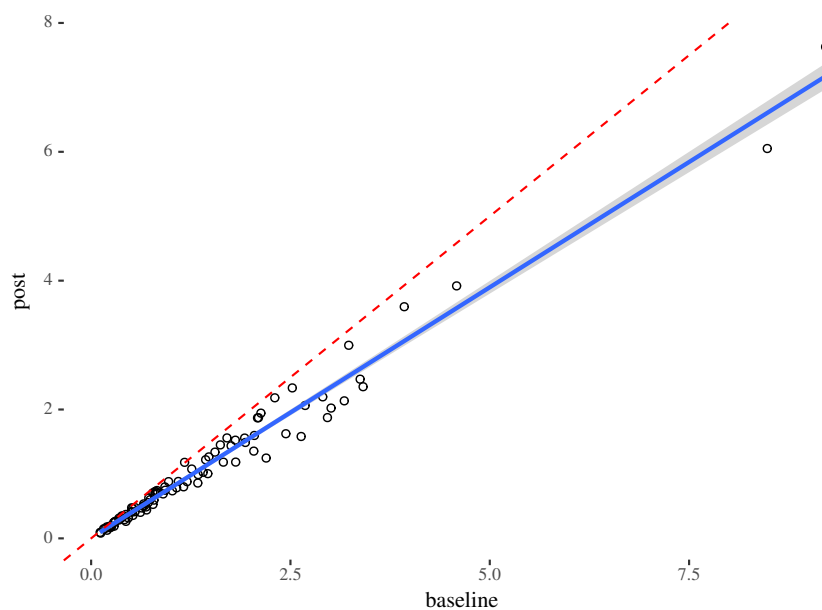
0.7.10 A complex ggplot

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 properly.
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)

```

We can also obviously do the percent difference.

```

diffplotp <- ggplot(my.data, aes(baseline, diffp)) +
  geom_point(size=2, colour = rgb(0,0,0, alpha = 0.5)) +
  theme_bw() +
  geom_hline(yintercept = 0, linetype = 3) +
  geom_hline(yintercept = mean(my.data$diffp)) +
  geom_hline(yintercept = mean(my.data$diffp) + 2*sd.diffp, linetype = 2) +
  geom_hline(yintercept = mean(my.data$diffp) - 2*sd.diffp, linetype = 2) +
  ylab("Difference pre and post Storage (%)") +
  xlab("Baseline Concentration (mg/L)")

ggMarginal(diffplotp, type="histogram", bins = 25)

```

o.8 Tosin olulisimat graafikutüüpi

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.

0.8.1 1. Cleveland plot

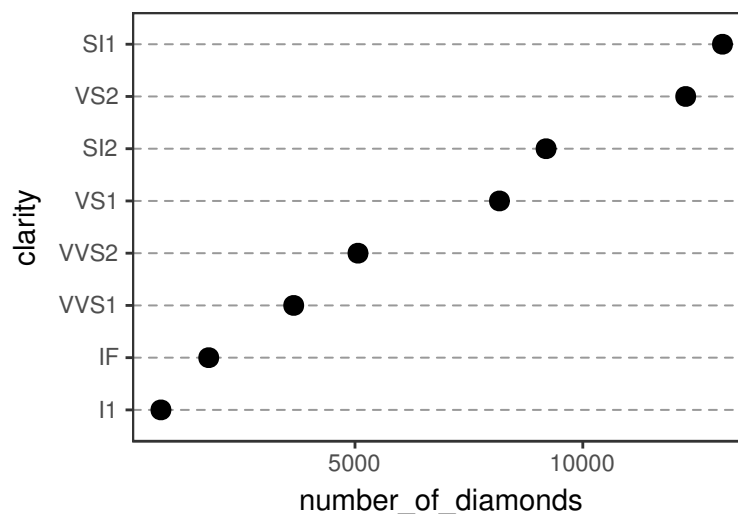
x- pidev muutuja; y - faktormuutuja

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 suurus, mis on $\frac{1}{3}$), 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 plottime, mitu korda esinevad diamond 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_bw() +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_line(colour="grey60", linetype="dashed")) +
  labs(y="clarity")
```

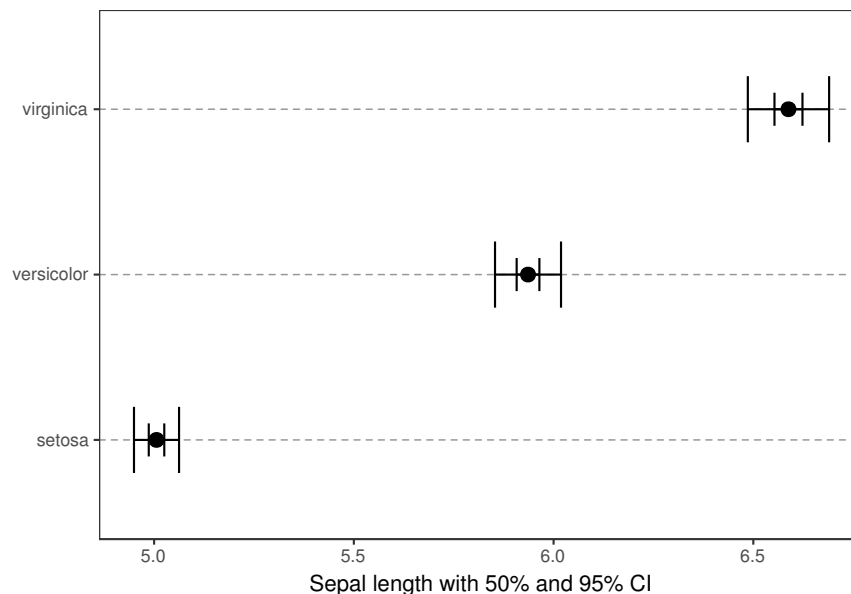


Graafiku loetavuse huvides on mõistlik on Cleveland plotil Y- telg sorteerida väärtuste järgi.

Järgmisel joonisel on näha irise tabeli Sepal length veeru keskmised koos 50% ja 95% usaldusintervallidega.

```
iris1 <- iris %>% group_by(Species) %>%
  summarise(Mean= mean(Sepal.Length), SEM= sd(Sepal.Length)/sqrt(nrow(iris)))

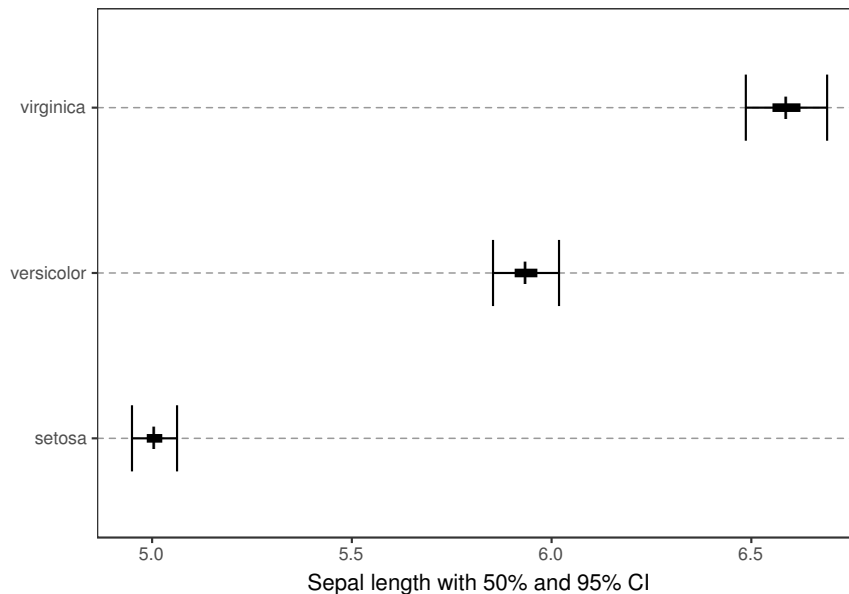
ggplot(data= iris1, aes(x=Mean, y=Species)) +
  geom_point(size=3) +
  geom_errorbarh(aes(xmin= Mean - 0.675*SEM, xmax= Mean + 0.675*SEM), height=0.2)+
  geom_errorbarh(aes(xmin= Mean - 1.96*SEM, xmax= Mean + 1.96*SEM), height=0.4)+
  theme_bw() +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_line(colour="grey60", linetype="dashed"))+
  labs(x="Sepal length with 50% and 95% CI", y=NULL)
```



Alternatiivne graafiku kuju:

```
ggplot(data= iris1, aes(x=Mean, y=Species)) +
  geom_point(size=5, shape=108) +
  geom_errorbarh(aes(xmin= Mean - 0.675*SEM, xmax= Mean + 0.675*SEM), height=0, size=2)+
  geom_errorbarh(aes(xmin= Mean - 1.96*SEM, xmax= Mean + 1.96*SEM), 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="dashed")+
labs(x="Sepal length with 50% and 95% CI", y=NULL)
```



NB! Siin on usaldusintervallide arvutamiseks kasutatud kiiret ad hoc meetodit, mis eeldab muuhulgas, et valimi suurus ei ole väike. Kui $n < 30$, või kui valimi andmejaotus on väga kaugel normaaljaotusest (jaotus on näiteks väga pika õlaga), tuleks usaldusintervalli arvutamiseks kasutada bayesiaanlikke meetodeid, mida tutvustame hilisemates peatükkides. Igal juhul, kui valimi suurus on piisav ja normaaljaotus pole meie andmetest liiga kaugel, siis saame kasutada järgmisi heuristikuid:

```
#> # A tibble: 7 x 2
#>   CI_percentage nr_of_SEMs
#>   <dbl>         <dbl>
#> 1    50.0         0.675
#> 2    75.0         1.15
#> 3    90.0         1.64
#> 4    95.0         1.96
#> 5    97.0         2.17
#> 6    99.0         2.58
#> # ... with 1 more row
```

SEM on standardviga ja see arvutatakse jagades valimi standardhälbe ruutjuurega valimi suurusest.

Järgneb alternatiivne viis sama arvutust teha:

Järgmiseks plotime irise liikide Sepal.length andmete keskmise koos 50% ja 95% protsentiil intervallidega. Protsentiil intervallid näitavad piirkonda mis sisaldab, antud näite põhjal, vastavalt 50% ja 95% andmeid.

Kõigepealt arvutame igale irise liigile Sepal.Length usaldusintervallid ja keskväärtuse kasutades

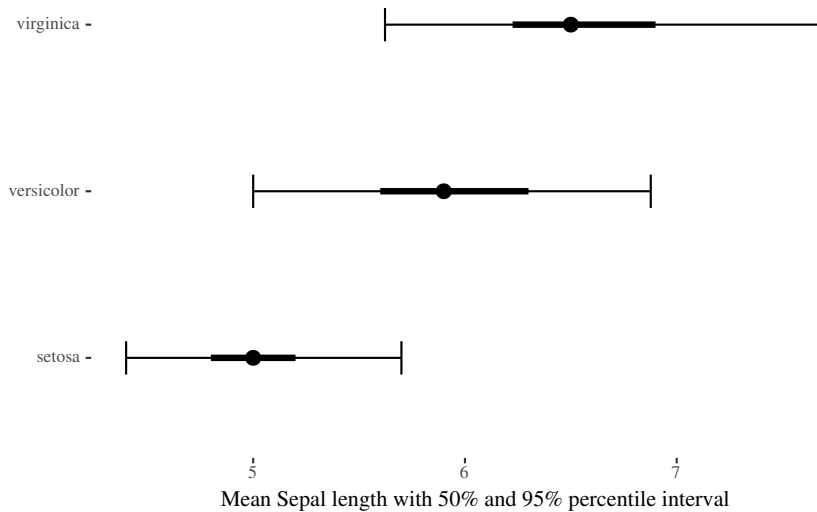
`percentile_intervals` funktsiooni, soovitud usaldusintervallid spetsifitseeritakse argumendiga `prob = c(0, 0.5, 0.95)`. Kusjuures, 0% andmeid sisaldav piirkond asub keskväärtusel.

```
iris_sepal_length_pi <- iris %>%
  group_by(Species) %>%
  do(percentile_intervals(.$Sepal.Length, prob = c(0, 0.5, 0.95)))

iris_sepal_length_pi
#> # A tibble: 3 x 6
#> # Groups:   Species [3]
#>   Species      `3%` `25%` `50%` `75%` `98%`
#>   <fct>      <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 setosa      4.40  4.80  5.00  5.20  5.70
#> 2 versicolor  5.00  5.60  5.90  6.30  6.88
#> 3 virginica   5.62  6.22  6.50  6.90  7.70
```

Saadud summaarseid andmeid kasutame graafiku loomiseks. Y-telje väärtused on kõigile geoomidele sama `ggplot(aes(y = Species))`. Lisame geoomid, kõigepealt paneme paika keskmise kasutades punkti geoomi `geom_point()`. Seejärel lisame ükshaaval usalduspiirid kasutades horisontaalsete veapiiride geoomi `geom_errorbarh()`. Arumendid `height = 0`, `size = 1.5` määravad antud juhul, vastavalt, veapiiri otstes olevate “vuntside” kõrgust ja joone paksust.

```
iris_sepal_length_pi %>%
  ggplot(aes(y = Species, x = `50%`)) +
  geom_point( size = 3) +
  geom_errorbarh(aes(xmin = `3%`, xmax = `98%`), height = 0.2) +
  geom_errorbarh(aes(xmin = `25%`, xmax = `75%`), height = 0, size = 1.5) +
  theme(axis.title.y = element_blank()) +
  labs(x = "Mean Sepal length with 50% and 95% percentile interval")
```

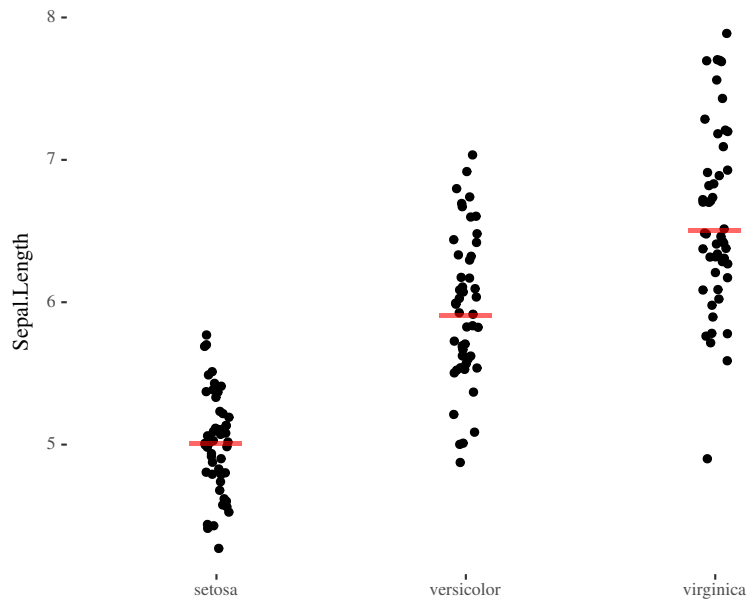
Sellel joonisel on näha irise tabeli Sepal length veeru keskmised (punkt) koos 50% (rasvane joon) ja 95% (vuntsid) usaldusintervallidega.

0.8.2 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.

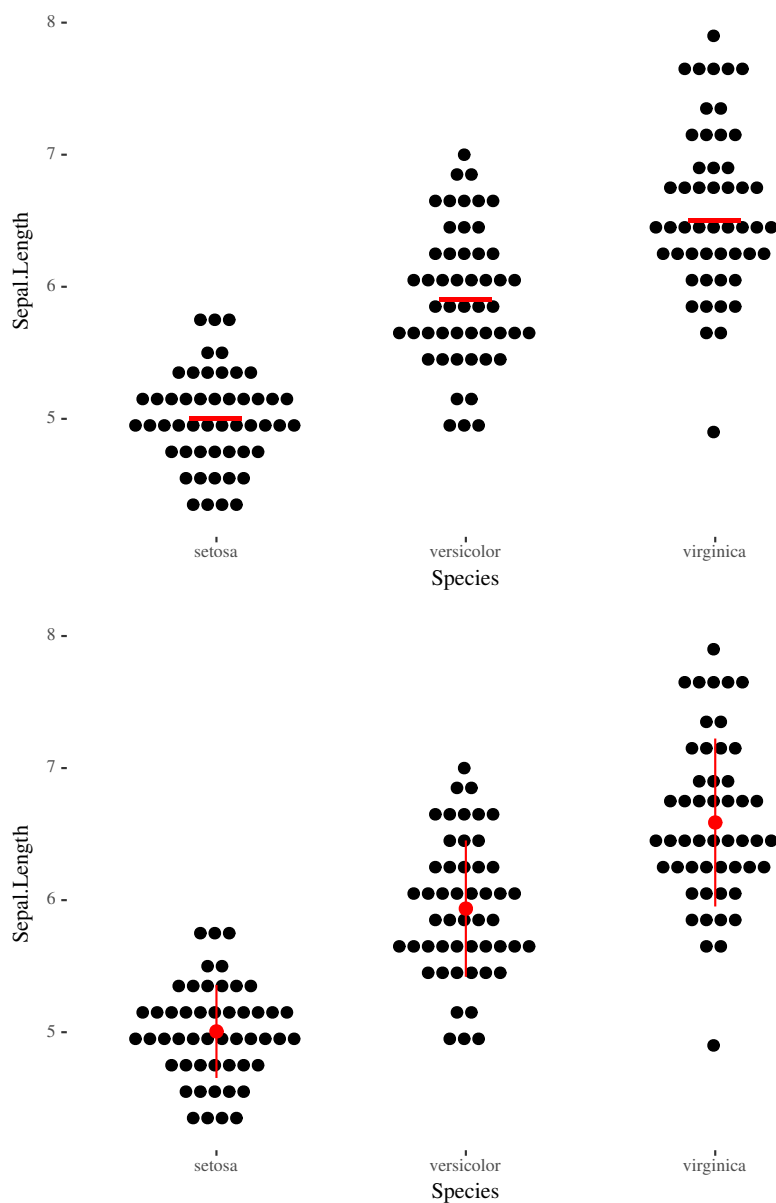
Meil on võimalik teha sellest graafikust versioon, mis ei pane andmepunkti y skaalal täpselt õigesse kohta, vaid tekitab histogrammilaadsed andmebinnid, kus siiski iga punkt on eraldi näidatud. See lihtsustab veidi “kirjude” kompleksete andemsettide esitust. Kuid kaotab informatsiooni andmepunktide täpse asukoha kohta. Siin on eesmärk muuta graafik kiiremini loetavaks, ehk 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="pointrange", color="red")

#use errorbar instead of pointrange:
#p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1), geom="errorbar", color="red", width=0.2, size=1)
```



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

0.8.3 3. Histogramm

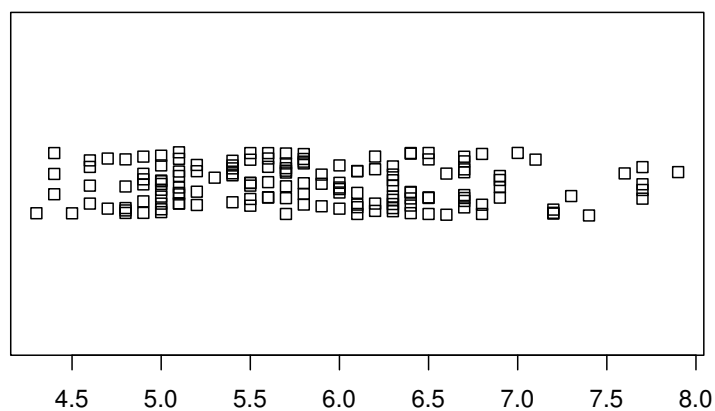
x - pidev muutuja

Kui teil on palju andmepunkte (>50) ning soovite uurida nende jaotust (ja/või võrrelda mitme and-

mestiku jaotust) siis tasub kindlasti alustada histogrammist. Histogrammi koostamine näeb välja järgmine:

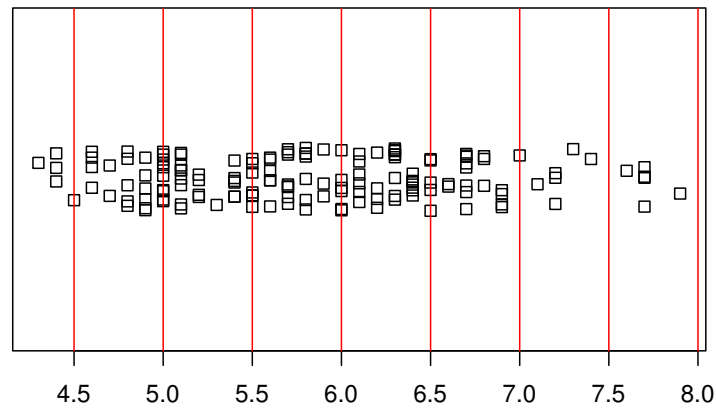
1. ploti andmepunktid x - teljele (järgnev on põhimõtteliselt ühedimensionaalne plot, kuigi andmepunktid on üksteise suhtes veidi nihutatud, et nad üksteist ei varjutaks).

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



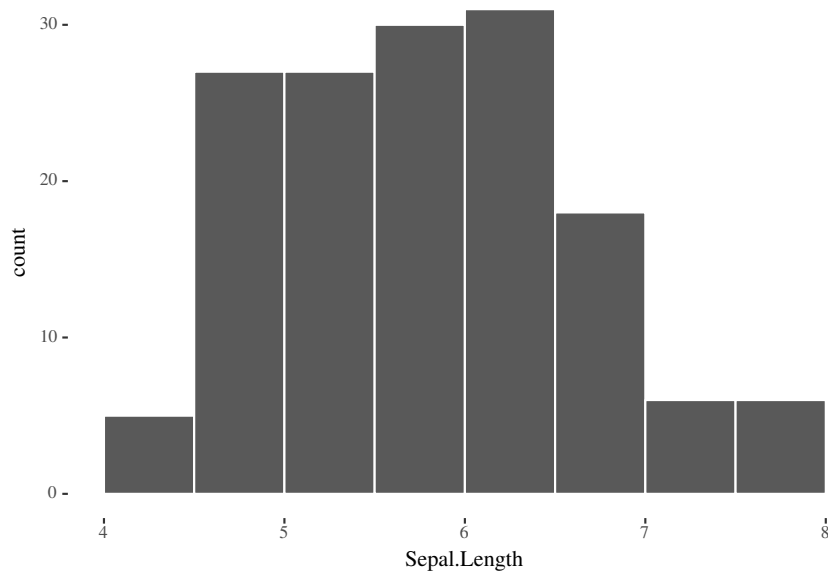
2. jaga andmestik x-teljel võrdse laiusega vahemikesse (binnidesse)

```
stripchart(iris$Sepal.Length, method = "jitter")  
abline(v=seq(4, 8, by=0.5), col="red")
```



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

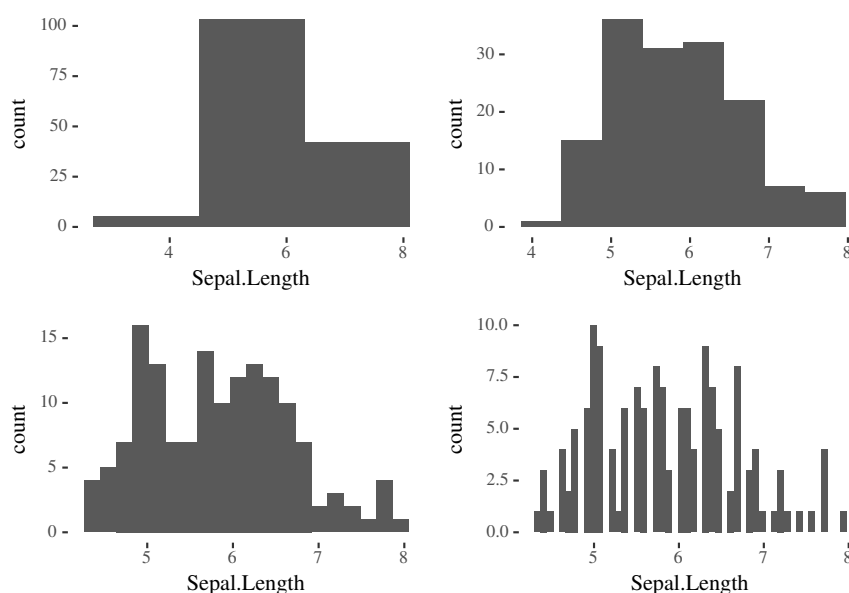
```
ggplot(iris, aes(x=Sepal.Length)) + geom_histogram(breaks= seq(4, 8, by=0.5), color="white")
```



Tavaliselt on siiski mõistlik määrata histogrammi binnide laius ja asukoht mitte *breaks* argumeniga vaid kas argumendiga *bins*, mis annab binnide arvu, või argumendiga *binwidth*, mis annab binni laiuse. Vt ka `geom_boxplot()` funktsiooni helpi.

NB! Väga tähtis on mõista, et binnide laius on meie suva järgi määratud. Samade andmete põhjal joonestatud 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)
```



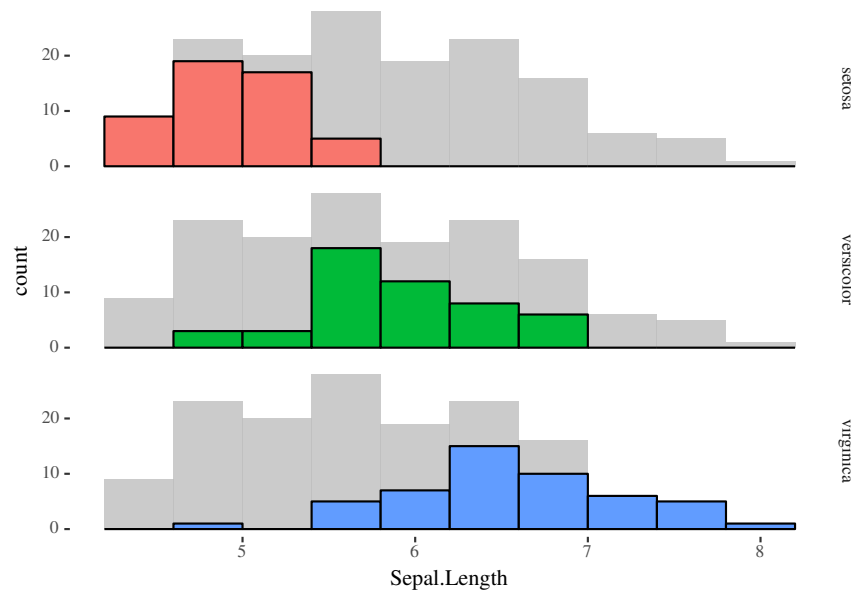
Seega on tasub joonistada samadest andmetest mitu erineva binnilaiusega histogrammi, et oma andmeid vaadata mitme nurga alt.

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 võrdluskalaat üle kolme paneeli.

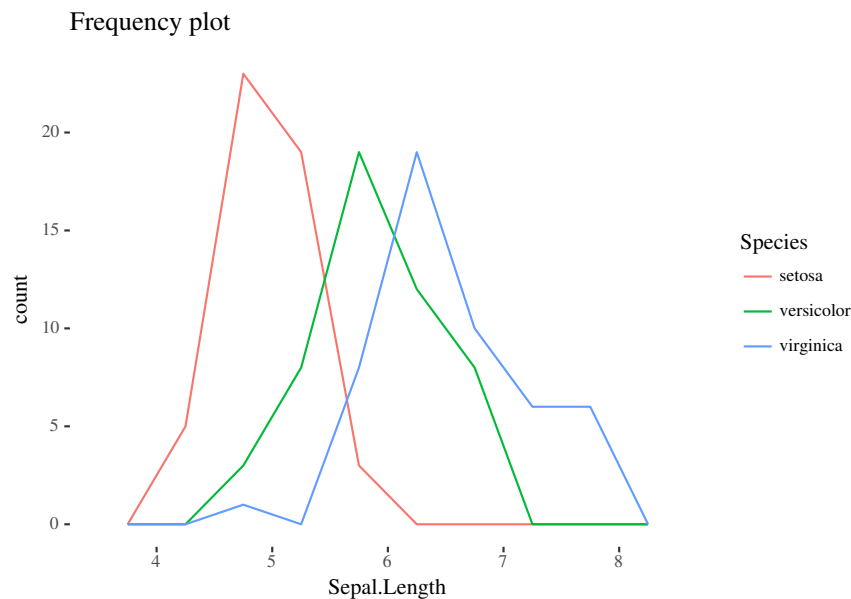
```
d_bg <- iris[, -5] # Background Data - full without the 5th column (Species)

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
```



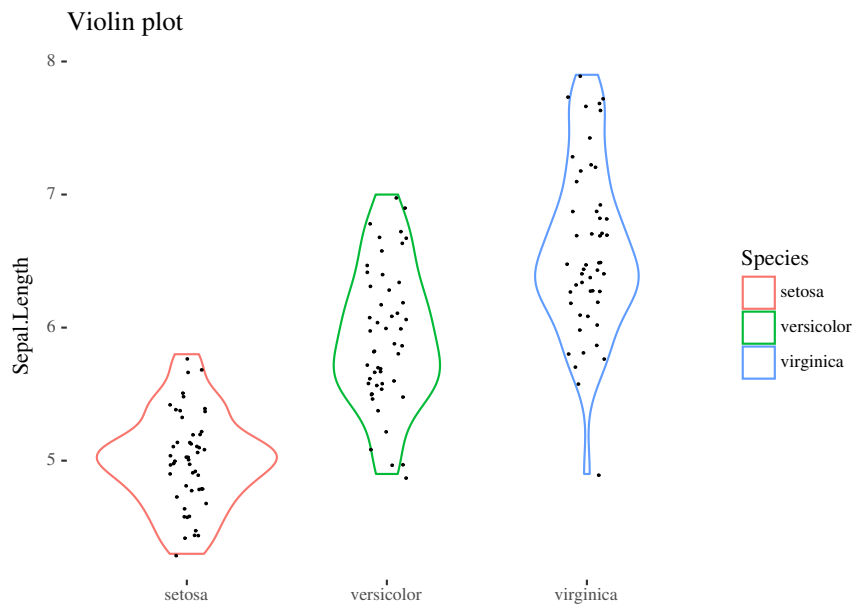
Teine võimalus on näidata kõiki koos ühel paneelil kasutades histogrammi asemel sageduspolügoni. See töötab täpselt nagu histogramm, ainult et tulpade asemel joonistatakse binnitippude vahele jooned. Neid on lihtsam samale paneelile üksteise otsa laduda.

```
ggplot(iris, aes(Sepal.Length, color=Species)) + geom_freqpoly(breaks= seq(4, 8, by=0.5)) + theme_tufte() + labs
```



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

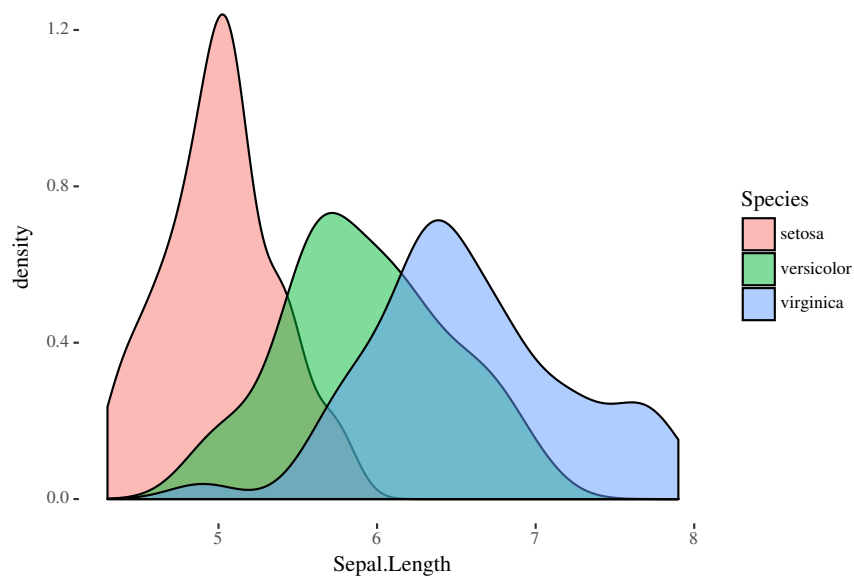
```
ggplot(iris, aes(Species, Sepal.Length)) + geom_violin(aes(color=Species)) +  
  geom_jitter(size=0.2, width=0.1) + labs(title="Violin plot", x=NULL)
```



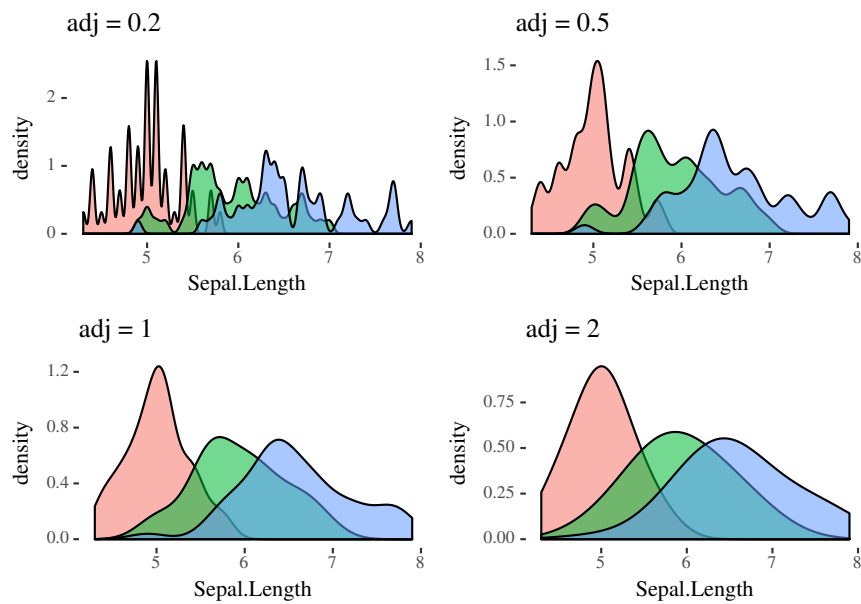
0.8.4 3. Tihedusplot

Hea alternatiiv histogrammile on joonistada silutud andmejaotus, mis käitub silutud histogrammina.

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

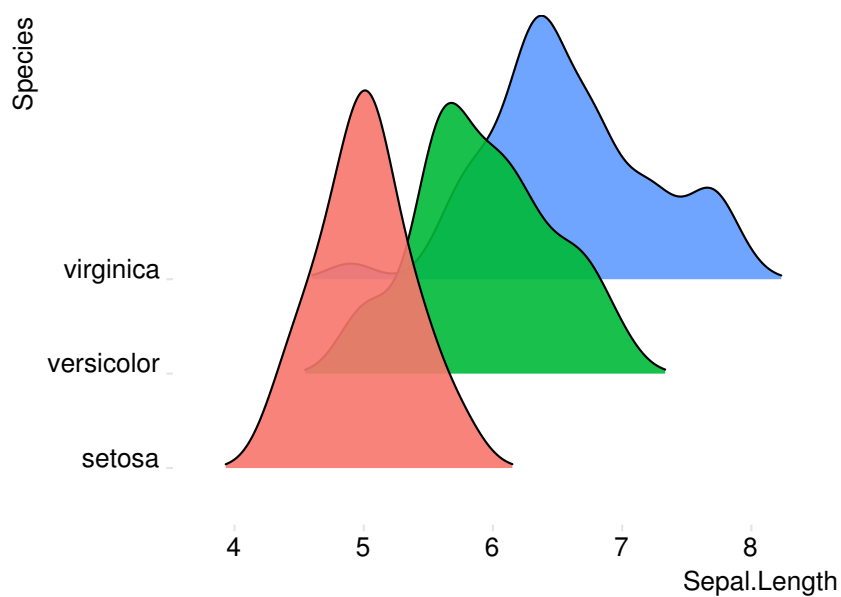


Adjust parameeter reguleerib silumise määra.

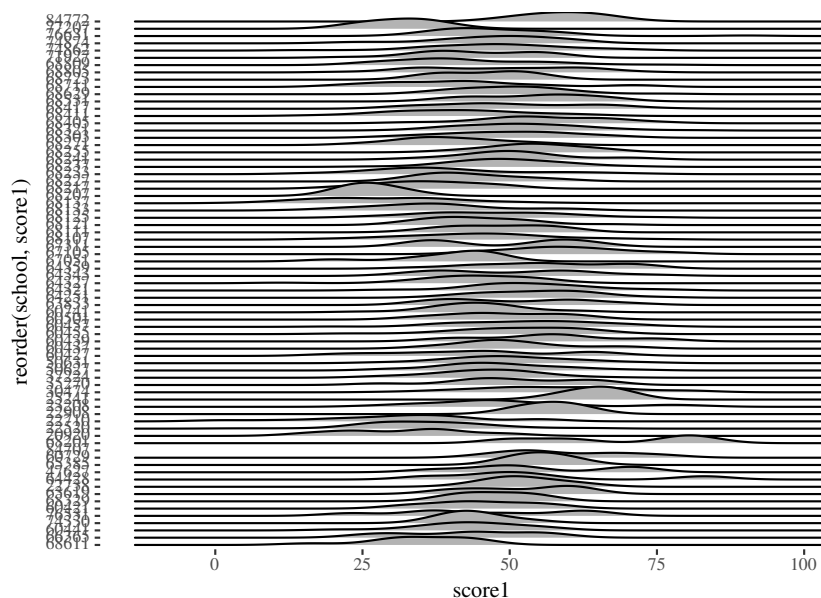


Veel üks võimalus jaotusi kõrvuti vaadata on joyplot, mis paneb samale paneelile kasvõi sada tihedusjaotust.

```
library(ggjoy)
ggplot(iris, aes(x=Sepal.Length, y=Species, fill=Species)) +
  geom_joy(scale=4, rel_min_height=0.01, alpha=0.9) +
  theme_joy(font_size = 13, grid=TRUE) +
  theme(legend.position = "none")
```



```
sch <- read.csv("data/schools.csv")
sch$school <- as.factor(sch$school)
ggplot(sch, aes(score1, y=reorder(school, score1))) +
  geom_joy() + theme_tufte()
#> Warning: Removed 202 rows containing non-finite values
#> (stat_density_ridges).
```

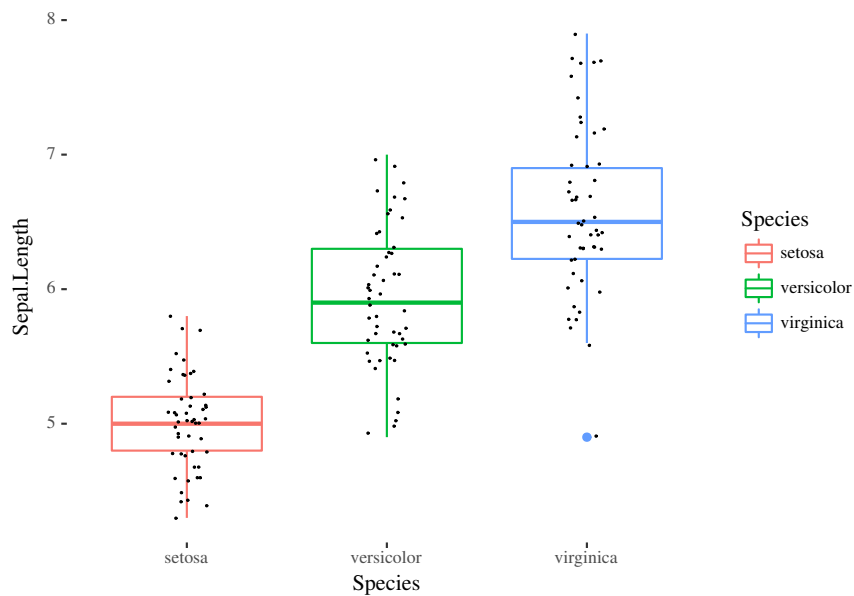


o.8.5 4. Boxplot

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 järkevalt.

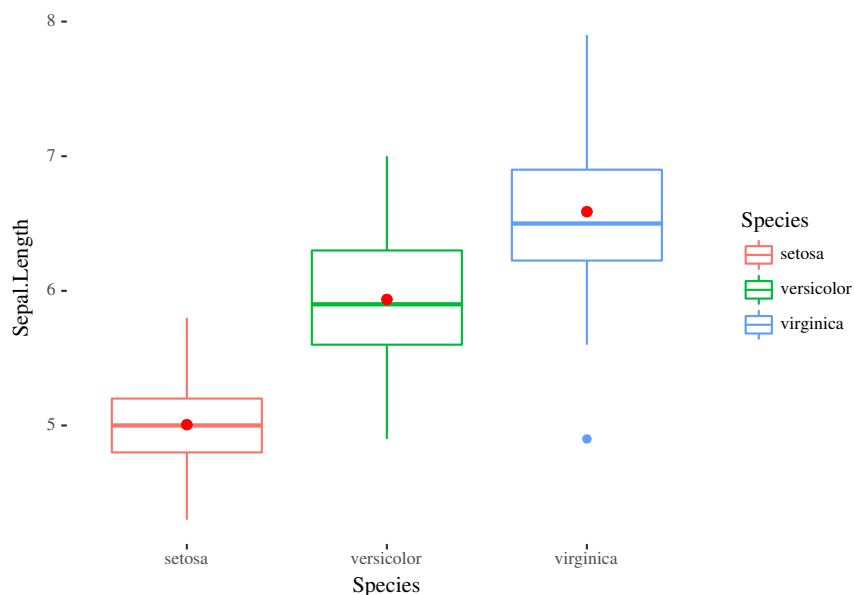
1. joonista andmepunktid 1D-s välja (nagu me tegime histogrammi puhul)
2. keskmine andmepunkt on mediaan. Selle kohale tuleb boxplotil keskmine kriips
3. ümbritse kastiga pooled andmepunktid (mõlemal pool mediaani). Nii määrad nn. interkvartiilse vahemiku (IQR).
4. pooleteistkordne IQR (y-telje suunas) 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 $1.5 \times \text{IQR}$ -i joonistatakse eraldi välja kui outlierid.

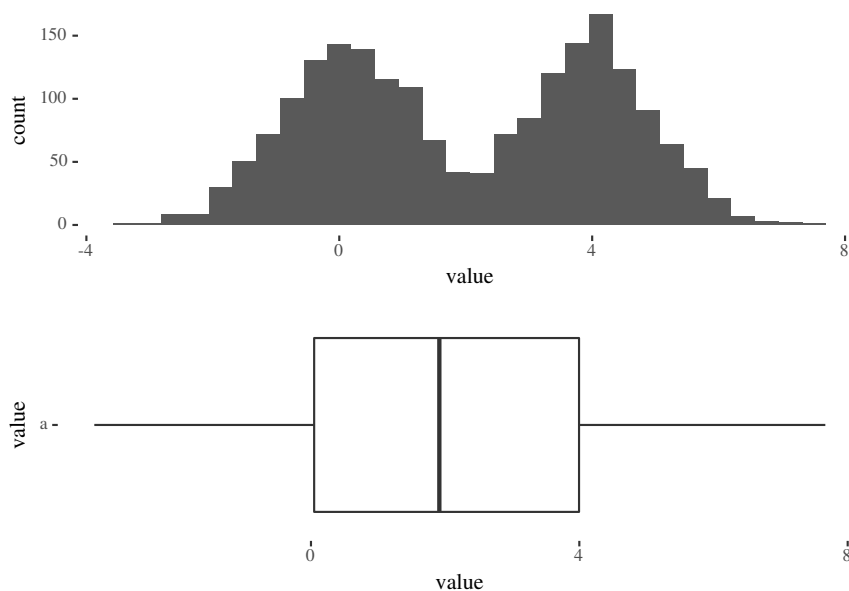
```
ggplot(iris, aes(Species, Sepal.Length, color = Species)) +
  geom_boxplot() +
  geom_jitter(width = 0.1, size=0.1, color="black")
```



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)
```





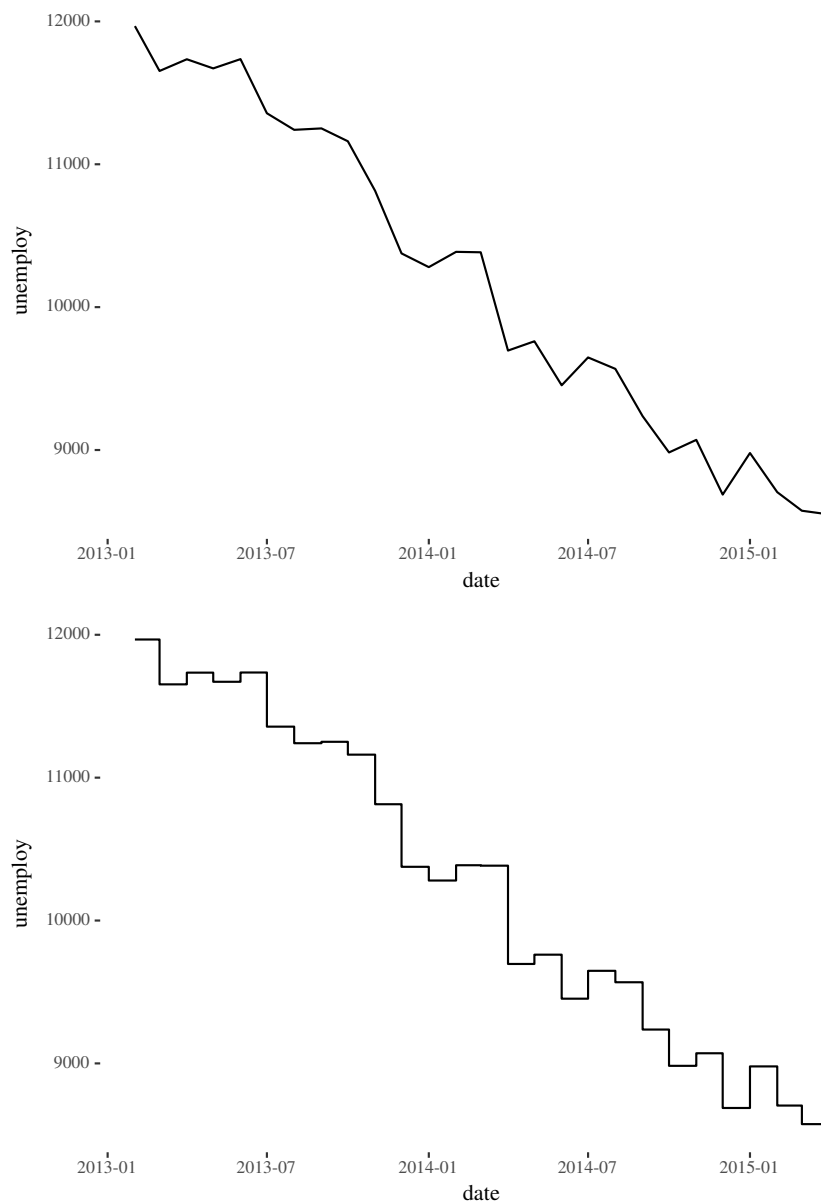
See pilt näitab, et kui jaotus on mitme tipuga, siis võib boxplotist olla rohkem kahju kui kasu.

0.8.6 5. Joongraafikud

x - pidev muutuja (aeg, konsentratsioon, jms); y - pidev muutuja

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.

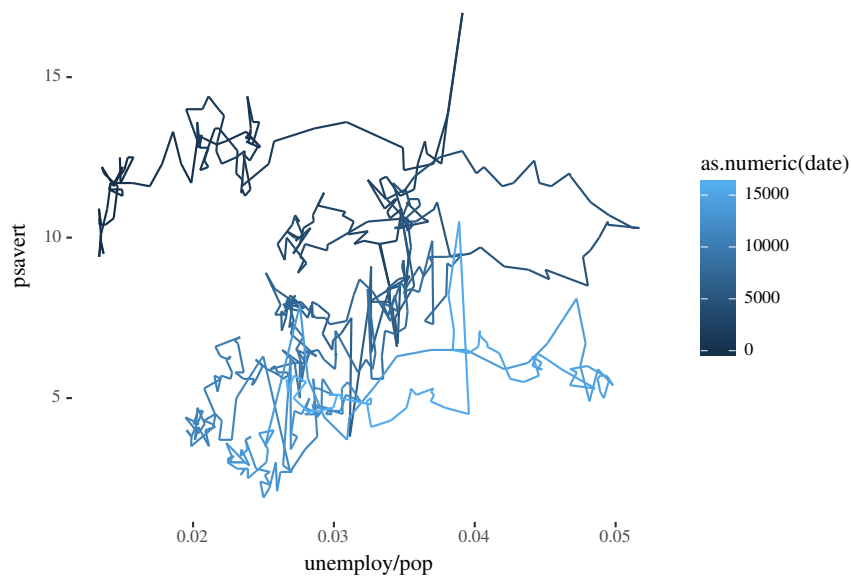
```
recent <- economics[economics$date > as.Date("2013-01-01"), ]
ggplot(recent, aes(date, unemploy)) + geom_line()
ggplot(recent, aes(date, unemploy)) + geom_step()
```



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

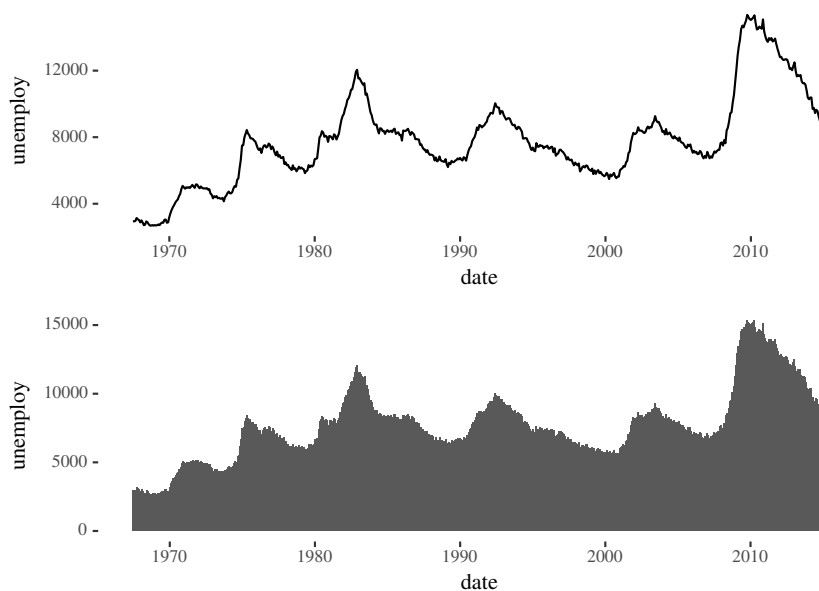
Geom_path võimaldab joonel ka tagasisuunas keerata.

```
# 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)))
```



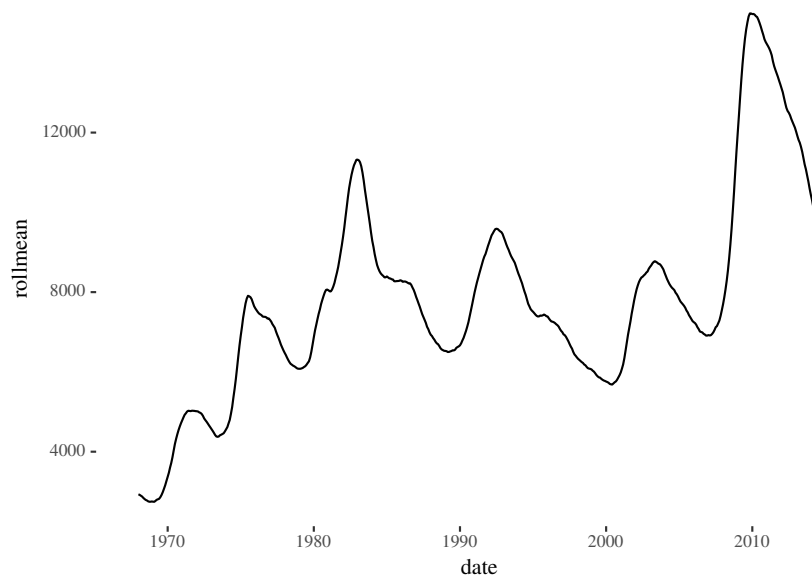
Tulpdiagramm juhib lugeja tähelepanu väikestele teravatele muutustele. Kui see on see, millele sa tahad tähelepanu juhtida, siis kasuta seda.

```
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 mürarikaid 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 aastaegade 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. Multiplikatiivses mudelis

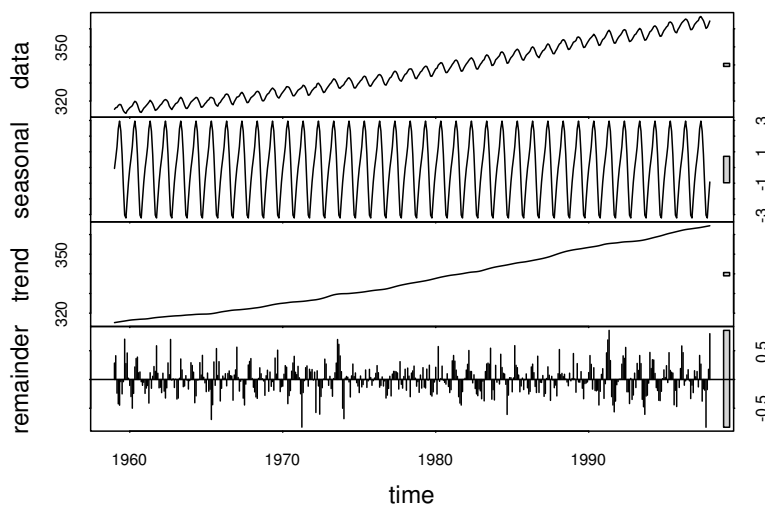
$$Y_t = Trend_t * Seasonal_t * Irregular_t$$

tuleb selleks teha korrutamistehe.

Näiteks lahutame aegrea, mis käsitleb CO₂ konsentratsiooni muutusi viimase 50 aasta vältel.

```
require(graphics)
#co2 is a time series object
```

```
#stl() takes class "ts" objects only!
plot(stl(co2, "per"))
```

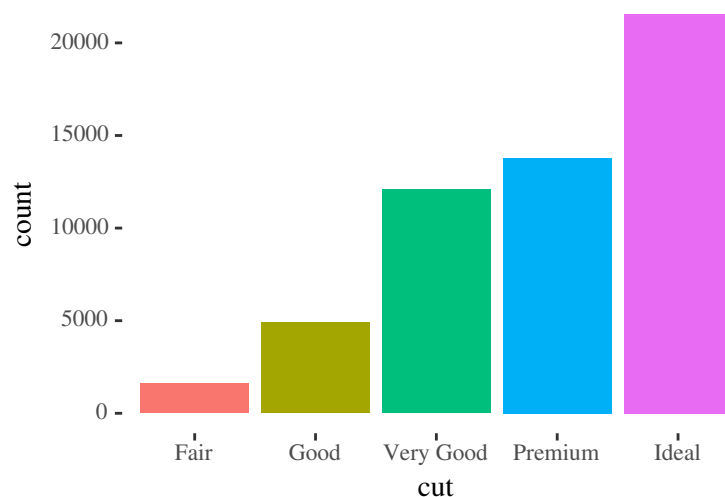


0.8.6.1 Tulpdiagrammid mõõdavad counte ja proportsioone

```
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 1 3 ...
#> $ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<...: 2 2 2 6 7 7 6 5 2 5 ...
#> $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<...: 2 3 5 4 2 6 7 3 4 5 ...
#> $ 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.

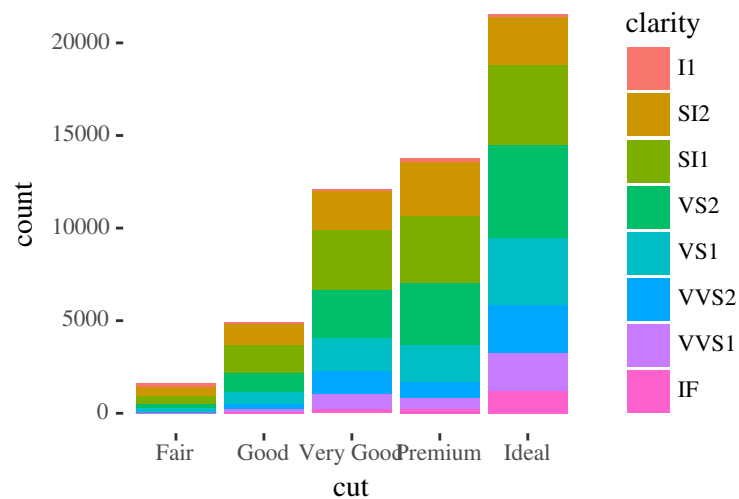
Kui tahad tulpdiagrammi proportsioonidest, mitu korda eineb tabelis iga cut-i, siis tee nii:

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

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

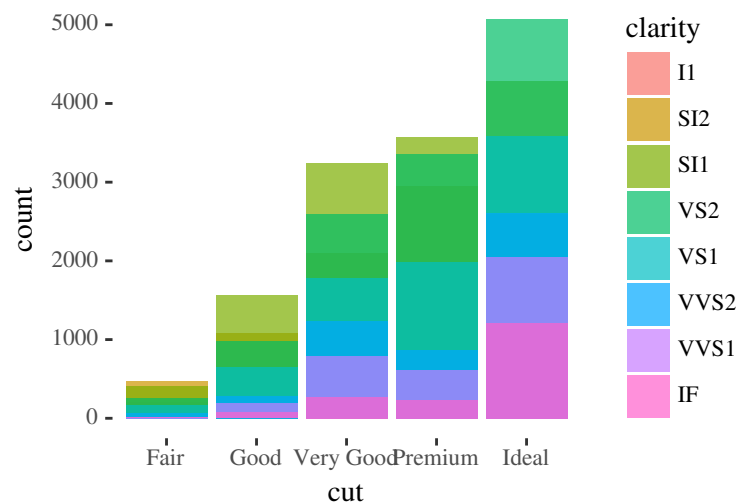
Edasi lisame eelnevale veel ühe muutuja: clarity. Nii saame üles lugeda kõigi cut-i ja clarity kombinatsioonide esinemise arvu või sageduse. Erinivate clarity tasemete esinemiste arv samal cut-i tasemel on siin üksteise otsa kuhjatud, mis tähendab, et tulpade kõrgus ei muutu võrreldes eelnevaga.

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



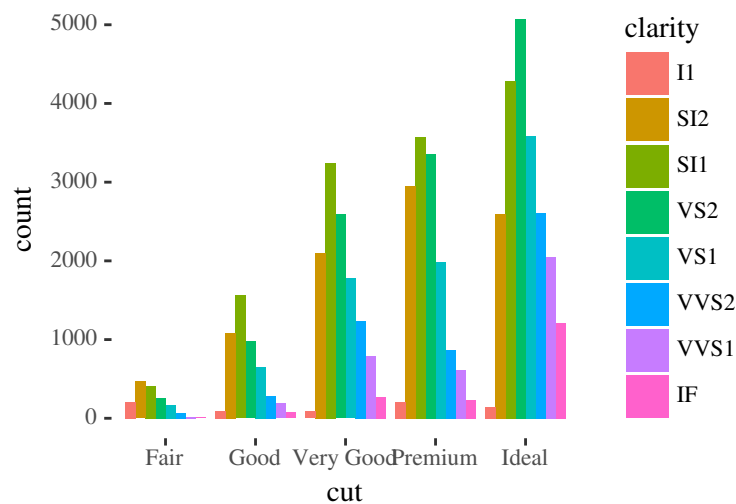
Kui me tahame, et cut-i ja clarity kombinatsioonid oleks kastidena üksteise sees, pigem kui üksteise otsa kuhjatud, siis kasutame `position = "identity"` argumenti.

```
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

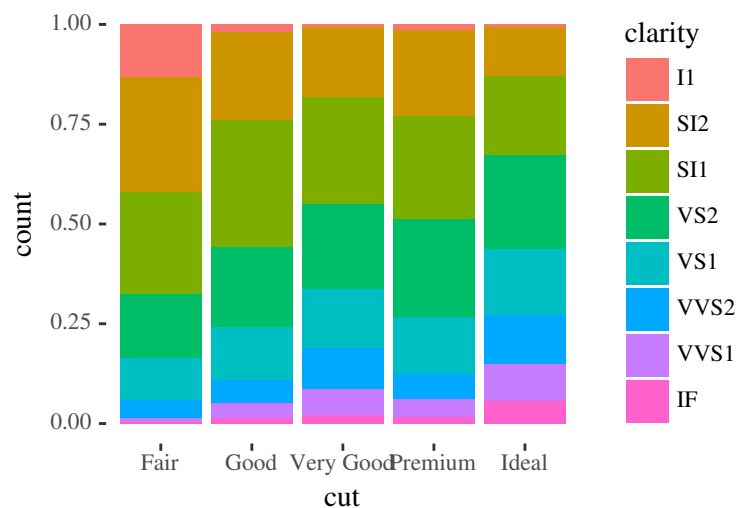
```
ggplot(data = diamonds, aes(x = cut, fill = clarity)) +  
  geom_bar(position = "dodge")
```



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

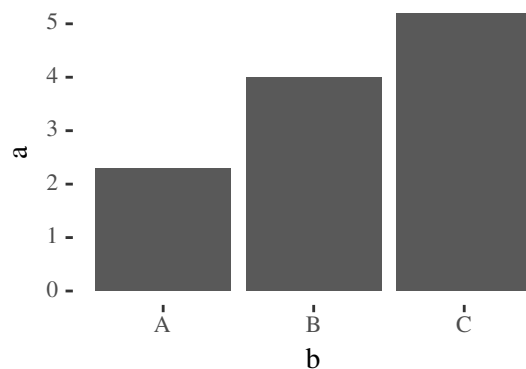
Ja lõpuks, `position="fill"` normaliseerib tulbad, mis muudab selle, mis toimub iga cut-i sees, hästi võrreldavaks. See on hea viis, kuidas võrrelda clarity tasemete proportsioone erinevate cut-i tasemete vahel.

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



Ja lõpuks, kui te tahate teha midagi, mis on enamasti keskmiselt rumal valik, ehk plottida tulpdiagrammi viisil, et tulba kõrgus vastaks tabeli ühes lahtris olevale numbrile, mitte faktortunnuse esinemiste arvule tabelis, siis kasutage: `geom_bar(stat = "identity")`

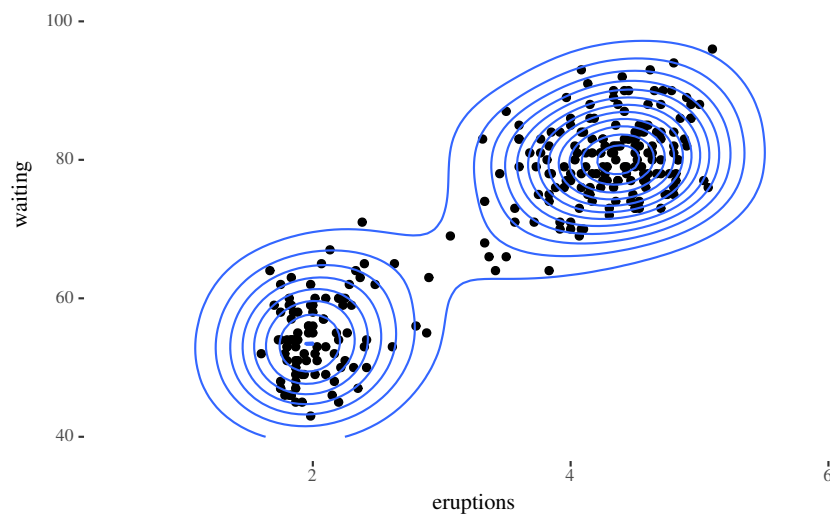
```
df <- tibble(a=c(2.3, 4, 5.2), b=c("A", "B", "C"))  
ggplot(df, aes(b, a)) + geom_bar(stat = "identity")
```



0.8.6.2 Kahe muutuja koos-varieeruvus

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 režiimis.

```
m <- ggplot(faithful, aes(x = eruptions, y = waiting)) +
  geom_point() +
  xlim(0.5, 6) +
  ylim(40, 110)
#m + stat_density_2d(aes(fill = ..level..), geom = "polygon")
m + geom_density_2d()
```



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

0.9 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. Tidyversega 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 vaikumisi prindifunktsioon, kus vaikumisi näidatakse 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 tulpade nimesid ei muudeta vaikumisi/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.

0.9.1 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 1)

- **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 weight
#>   <chr>   <chr>  <chr> <dbl>  <dbl>  <dbl>
#> 1 1      exp    F      3.00   168   88.0
#> 2 2      placebo M      3.00   176   91.0
```

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).

0.9.1.1 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       23.0        34.0        40.0
#> 2 Ann     F       31.0        38.0        42.0
#> 3 Jill    F       30.0        36.0        44.0
```

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       23.0
#> 2 Ann     F     control       31.0
#> 3 Jill    F     control       30.0
#> 4 Tim     M   experiment_1  34.0
#> 5 Ann     F   experiment_1  38.0
#> 6 Jill    F   experiment_1  36.0
#> # ... 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       31.0        38.0        42.0
#> 2 Jill    F       30.0        36.0        44.0
#> 3 Tim     M       23.0        34.0        40.0
```

`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.

0.9.1.2 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>
```

```
#>   <chr> <dbl> <dbl>
#> 1 tim    1.00  4.00
#> 2 tom    2.00  5.00
#> 3 jill   3.00  6.00
```

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
```

0.9.2 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.

0.9.2.1 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 cols
```

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_frame()
#> # A tibble: 150 x 3
#>   liik    Sepal.Length Sepal.Width
#>   <fct>         <dbl>         <dbl>
#> 1 setosa         5.10           3.50
#> 2 setosa         4.90           3.00
#> 3 setosa         4.70           3.20
#> 4 setosa         4.60           3.10
```



```
#> 5 setosa      5.00      3.60
#> 6 setosa      5.40      3.90
#> # ... with 144 more rows
```

To select all cols, except sex and value, and rename the *subject* col:

```
select(iris, -Sepal.Length, -Sepal.Width, link = 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         1.40         0.200
#> 2         1.40         0.200
#> 3         1.30         0.200
#> 4         1.50         0.200
#> 5         1.40         0.200
#> 6         1.70         0.400
#> # ... with 144 more rows
select(iris, ends_with("Width"))
#> # A tibble: 150 x 2
#>   Sepal.Width Petal.Width
#>       <dbl>       <dbl>
#> 1         3.50         0.200
#> 2         3.00         0.200
#> 3         3.20         0.200
#> 4         3.10         0.200
#> 5         3.60         0.200
#> 6         3.90         0.400
#> # ... 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         5.10           3.50           1.40
#> 2 setosa         4.90           3.00           1.40
#> 3 setosa         4.70           3.20           1.30
#> 4 setosa         4.60           3.10           1.50
#> 5 setosa         5.00           3.60           1.40
#> 6 setosa         5.40           3.90           1.70
#> # ... 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 0.811 0.148 0.551 0.585 0.236
#> 2 0.235 0.344 0.106 0.552 0.914
#> 3 0.0683 0.426 0.694 0.654 0.156
#> 4 0.752 0.165 0.194 0.838 0.252
#> 5 0.0374 0.482 0.900 0.171 0.487
#> 6 0.838 0.440 0.539 0.697 0.729
#> # ... 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 0.811 0.148 0.451 0.236
#> 2 0.235 0.344 0.152 0.914
#> 3 0.0683 0.426 0.215 0.156
#> 4 0.752 0.165 0.991 0.252
#> 5 0.0374 0.482 0.224 0.487
#> 6 0.838 0.440 0.902 0.729
#> # ... 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           5.10         3.50 setosa
#> 2           4.90         3.00 setosa
#> 3           4.70         3.20 setosa
#> 4           4.60         3.10 setosa
#> 5           5.00         3.60 setosa
#> 6           5.40         3.90 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         5.10         3.50         1.40         0.200
#> 2         4.90         3.00         1.40         0.200
#> 3         4.70         3.20         1.30         0.200
#> 4         4.60         3.10         1.50         0.200
#> 5         5.00         3.60         1.40         0.200
#> 6         5.40         3.90         1.70         0.400
#> # ... with 144 more rows, and 1 more variable:
#> #   Species <fct>
```

0.9.2.2 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         4.40         2.90         1.40         0.200
#> 2         4.30         3.00         1.10         0.100
#> 3         4.40         3.00         1.30         0.200
#> 4         4.40         3.20         1.30         0.200
#> # ... 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         5.10         3.50         1.40         0.200
#> 2         4.90         3.00         1.40         0.200
#> 3         4.70         3.20         1.30         0.200
#> 4         4.60         3.10         1.50         0.200
#> 5         5.00         3.60         1.40         0.200
#> 6         5.40         3.90         1.70         0.400
#> # ... 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         4.40         2.90         1.40         0.200
#> 2         4.30         3.00         1.10         0.100
#> 3         4.40         3.00         1.30         0.200
#> 4         4.40         3.20         1.30         0.200
#> 5         7.00         3.20         4.70         1.40
#> 6         6.40         3.20         4.50         1.50
#> # ... 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.

Filtering with regular expression: we keep the rows where *subject* starts with the letter “T”

```
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      7.00      3.20      4.70      1.40
#> 2      6.40      3.20      4.50      1.50
#> 3      6.90      3.10      4.90      1.50
#> 4      5.50      2.30      4.00      1.30
#> 5      6.50      2.80      4.60      1.50
#> 6      5.70      2.80      4.50      1.30
#> # ... with 94 more rows, and 1 more variable:
#> #   Species <fct>
```

As you can see there are endless vistas here, open for a regular expression fanatic. I wish I was one!

remove NAs with `filter()`

```
filter(flights, !is.na(dep_delay), !is.na(arr_delay))
```

0.9.2.3 summarise()

Many rows summarised to a single value

```
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 kasulik, kui teda kasutada koos järgmise verbi, `group_by`-ga.

0.9.2.4 group_by()

`group_by()` groups values for summarising or mutating-

When we summarise by *sex* we will get two values for each summary statistic: for males and females. Aint that sexy?!

```
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.

Now we group previously generated `dat_lng` data frame first by *sex* and then inside each group again by *experiment*. This is getting complicated ...

```

dat_lng
#> # A tibble: 9 x 4
#>   subject sex  experiment  value
#>   <chr>   <chr> <chr>         <dbl>
#> 1 Tim    M    control      23.0
#> 2 Ann    F    control      31.0
#> 3 Jill   F    control      30.0
#> 4 Tim    M    experiment_1  34.0
#> 5 Ann    F    experiment_1  38.0
#> 6 Jill   F    experiment_1  36.0
#> # ... with 3 more rows
group_by(dat_lng, sex, experiment) %>%
  summarise(MEAN = mean(value),
            SD = sd(value),
            N = n(),
            n_sex = n_distinct(sex))
#> # A tibble: 6 x 6
#> # Groups:   sex [?]
#>   sex  experiment    MEAN    SD    N n_sex
#>   <chr> <chr>         <dbl> <dbl> <int> <int>
#> 1 F    control      30.5  0.707    2    1
#> 2 F    experiment_1  37.0  1.41     2    1
#> 3 F    experiment_2  43.0  1.41     2    1

```

```
#> 4 M      control      23.0 NA      1      1
#> 5 M      experiment_1  34.0 NA      1      1
#> 6 M      experiment_2  40.0 NA      1      1
```

Now we group first by sex and then by variable. Spot the difference!

```
group_by(dat_lng, experiment, sex) %>%
  summarise(MEAN = mean(value),
            SD = sd(value),
            N = n(),
            n_sex = n_distinct(sex))

#> # A tibble: 6 x 6
#> # Groups:   experiment [?]
#>   experiment sex    MEAN    SD    N n_sex
#>   <chr>      <chr> <dbl> <dbl> <int> <int>
#> 1 control    F      30.5  0.707    2     1
#> 2 control    M      23.0  NA      1     1
#> 3 experiment_1 F      37.0  1.41    2     1
#> 4 experiment_1 M      34.0  NA      1     1
#> 5 experiment_2 F      43.0  1.41    2     1
#> 6 experiment_2 M      40.0  NA      1     1
```

pro tip if you want to summarise and then display the summary values as new column(s), which are added to the original non-shrunk df, use `mutate()` instead of `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      5.10      3.50      1.40      0.200
#> 2      4.90      3.00      1.40      0.200
#> 3      4.70      3.20      1.30      0.200
#> 4      4.60      3.10      1.50      0.200
#> 5      5.00      3.60      1.40      0.200
#> 6      5.40      3.90      1.70      0.400
#> # ... with 144 more rows, and 3 more variables:
#> #   Species <fct>, MEAN <dbl>, SD <dbl>
```

Anna igast grupist 3 kõrgeimat väärtust ja 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)
```

0.9.2.5 mutate()

Mutate põhikasutus on siiski uute veergude tekitamine, mis võtavad endale inputi rea kaupa. Seega tabeli ridade arv ei muutu.

If in your tibble called 'df' you have a column called 'value', you can create a new log2 transformed value column called log_value by `df %>% mutate(log_value = log2(value))`. Or you can create a new column where a constant is subtracted from the value column: `df %>% mutate(centered_value = value - mean(value))`. Here the mean value is subtracted from each individual value.

Mutate adds new columns (and transmute() creates new columns while losing the previous columns)

Here we firstly create a new column, which contains log-transformed values from the *value* column, and name it *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      23.0     3.14
#> 2 Ann     F     control      31.0     3.43
#> 3 Jill    F     control      30.0     3.40
#> 4 Tim     M     experiment_1  34.0     3.53
#> 5 Ann     F     experiment_1  38.0     3.64
#> 6 Jill    F     experiment_1  36.0     3.58
#> # ... with 3 more rows
```

The same with transmute: note the dropping of some of the original cols, keeping the original *subject* col and renaming the *sex* col.

```
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
```

```
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.

Here we first group and then mutate. Note that now, instead of a single constant, we divide by as many different constant as there are discrete factor levels in the sex variable (two, in our case):

```
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    0.711    2.67
#> 2 Ann     F     control      31.0    0.842    5.47
#> 3 Jill    F     control      30.0    0.814    5.29
#> 4 Tim     M     experiment_1  34.0    1.05     3.94
#> 5 Ann     F     experiment_1  38.0    1.03     6.70
#> 6 Jill    F     experiment_1  36.0    0.977    6.35
#> # ... with 3 more rows
```

Compare with a “straight” mutate to see the difference in values.

```
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    0.651    3.48
#> 2 Ann     F     control      31.0    0.877    4.69
```

```
#> 3 Jill      F      control      30.0      0.849      4.54
#> 4 Tim       M      experiment_1  34.0      0.962      5.14
#> 5 Ann       F      experiment_1  38.0      1.08       5.75
#> 6 Jill      F      experiment_1  36.0      1.02       5.44
#> # ... with 3 more rows
```

0.9.2.5.1 *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.

0.9.3 Grouped filters

Keep all groups bigger than a threshold:

```
popular_dests <- flights %>%
  group_by(dest) %>%
  filter(n() > 365)
```

If you need to remove grouping, and return to operations on ungrouped data, use `ungroup()`.

```
ungroup(dat)
```

`str_replace_all()` helps to deal with unruly labelling inside columns containing strings

The idea is to find a pattern in a collection of strings and replace it with something else. String == character vector.

To find and replace we use `str_replace_all()`, whose base R analogue is `gsub()`.

```
library(stringr)
(bad.df <- tibble(time = c("t0", "t1", "t12"), value = c(2, 4, 9)))
#> # A tibble: 3 x 2
#>   time value
#>   <chr> <dbl>
#> 1 t0     2.00
#> 2 t1     4.00
#> 3 t12    9.00
get_numeric <- function(x, ...) as.numeric(str_replace_all(x, ...))
(bad.df <- mutate_at(bad.df, "time", get_numeric, pattern = "t", replacement = ""))
#> # A tibble: 3 x 2
#>   time value
#>   <dbl> <dbl>
#> 1 0      2.00
#> 2 1.00   4.00
#> 3 12.0   9.00
```

now we have a numeric time column, which can be used in plotting.

or

```
library(readr)
(bad.df <- tibble(time = c("t0", "t1", "t12"), value = c(2, 4, 9)))
#> # A tibble: 3 x 2
#>   time value
#>   <chr> <dbl>
#> 1 t0     2.00
#> 2 t1     4.00
#> 3 t12    9.00
```

```
mutate_at(bad.df, "time", parse_number)
#> # A tibble: 3 x 2
#>   time value
#>   <dbl> <dbl>
#> 1  0     2.00
#> 2 1.00   4.00
#> 3 12.0   9.00
```

Here we did the same thing more elegantly by directly parsing numbers from a character string.

0.9.4 `separate()` one column into several

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 arvata, 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 = 2))
#> # 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 = -6))
#> # 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; Escharichia", "Eukaryota; Chordata")))
#> # A tibble: 2 x 2
#>   index taxon
#>   <dbl> <chr>
#> 1   1.00 Procaryota; Bacteria; Alpha-Proteobacteria; E~
#> 2   2.00 Eukaryota; Chordata
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep = '; ', extra = "merge", fill = "right"))
#> # A tibble: 2 x 5
#>   index riik      hmk      klass      perekond
#> * <dbl> <chr>    <chr>    <chr>    <chr>
#> 1   1.00 Procaryota Bacteria Alpha-Proteoba~ Escharich~
#> 2   2.00 Eukaryota Chordata <NA>      <NA>
```

```
# some special cases:
(dat <- tibble(index = c(1, 2),
               taxon = c("Prokaryota || Bacteria || Alpha-Proteobacteria || Escharichia", "Eukaryota || Chordata")))
#> # A tibble: 2 x 2
#>   index taxon
#>   <dbl> <chr>
#> 1   1.00 Prokaryota || Bacteria || Alpha-Proteobacteri~
#> 2   2.00 Eukaryota || Chordata
(d1 <- dat %>% separate(taxon, c("riik", "hmk", "klass", "perekond"), sep = "\\|\\|", extra = "merge", fill = "right"))
#> # A tibble: 2 x 5
#>   index riik      hmk      klass      perekond
#> * <dbl> <chr>    <chr>    <chr>    <chr>
#> 1   1.00 "Prokaryota " " Bacteria " " Alpha~ " Eschar~
#> 2   2.00 "Eukaryota " " Chordata" <NA>      <NA>
```

```
dat <- tibble(index = c(1, 2),
              taxon = c("Prokaryota.Bacteria.Alpha-Proteobacteria.Escharichia", "Eukaryota.Chordata"))
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep = '[.]', extra = "merge", fill = "right"))
#> # A tibble: 2 x 5
#>   index riik      hmk      klass      perekond
#> * <dbl> <chr>    <chr>    <chr>    <chr>
#> 1   1.00 Prokaryota Bacteria Alpha-Proteoba~ Escharich~
#> 2   2.00 Eukaryota Chordata <NA>      <NA>
```

```
(dat <- tibble(index = c(1,2),
               taxon = c("Prokaryota.Bacteria,Alpha-Proteobacteria.Escherichia", "Eukaryota.Chordata")))
#> # A tibble: 2 x 2
#>   index taxon
#>   <dbl> <chr>
#> 1   1.00 Prokaryota.Bacteria,Alpha-Proteobacteria.Esch~
#> 2   2.00 Eukaryota.Chordata
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep = '[,\\.]', extra = "merge", fill = 'NA'))
#> # A tibble: 2 x 5
#>   index riik      hmk      klass      perekond
#> * <dbl> <chr>    <chr>    <chr>    <chr>
#> 1   1.00 Prokaryota Bacteria Alpha-Proteoba~ Escharich~
#> 2   2.00 Eukaryota Chordata <NA>      <NA>
```

The companion FUN to separate is unite() - see help.

0.9.5 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
#>   <int> <fct>   <int> <fct> <fct>   <fct>   <fct>
#> 1  2000 Never m~   26 White $8000 t~ Ind,near~ Prote~
#> 2  2000 Divorced  48 White $8000 t~ Not str ~ Prote~
#> 3  2000 Widowed   67 White Not app~ Independ~ Prote~
#> 4  2000 Never m~   39 White Not app~ Ind,near~ Ortho~
#> 5  2000 Divorced   25 White Not app~ Not str ~ None
#> 6  2000 Married   25 White $20000 ~ Strong d~ Prote~
#> # ... with 2.148e+04 more rows, and 2 more variables:
#> #   denom <fct>, 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.

0.9.5.1 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 “other”.

```
library(tidyverse)
i <- iris %>% mutate(
  fact = case_when(
    Sepal.Length > 5.8 | Sepal.Width > 4 ~ "large",
    Species == "setosa" ~ "I. setosa",
    TRUE ~ "other"
  ))
```

0.9.5.2 `dropLevels()` viskab välja kasutamata faktori tasemed

```
df1$sex <- dropLevels(df1$sex)
```

0.9.5.3 `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.

0.9.5.4 `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 party"),
                                rep = c("Strong republican", "Not str republican"),
                                ind = c("Ind,near rep", "Independent", "Ind,near dem"),
                                dem = c("Not str democrat", "Strong democrat")

  )) %>%
  count(partyid)

```

0.9.5.5 `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
```

0.9.5.6 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?

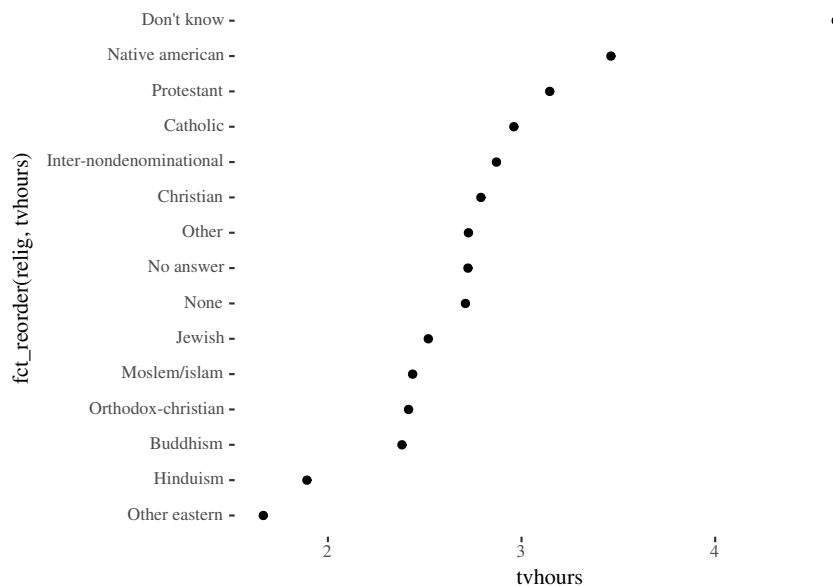
```
z <- 1:10
z1 <- cut(z, breaks = c(0, 3, 6, 10), labels = c("A", "B", "C"))
z1
#> [1] A A A B B B C C C C
#> Levels: A B C
#Note that to include 1 in level "A" you need to start the first cut <1, while at the right side 3 is included
z2 <- cut(z, breaks = 3, labels = c("A", "B", "C"))
z2
#> [1] A 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"
```

0.9.5.7 Muudame faktori tasemete järjekorda joonisel

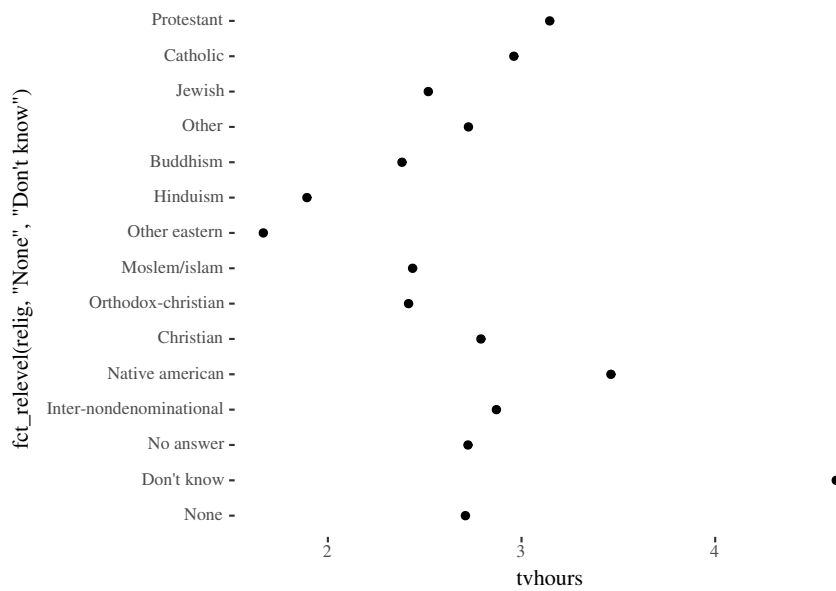
```
## summeerime andmed
gsscat_sum <- group_by(gss_cat, relig) %>%
  summarise(age = mean(age, na.rm = TRUE),
            tvhours = mean(tvhours, na.rm = TRUE),
            n = n())
## joonistame graafiku
p <- ggplot(gsscat_sum, aes(tvhours, fct_reorder(relig, tvhours))) +
  geom_point()
p
```



0.9.5.8 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"))
```

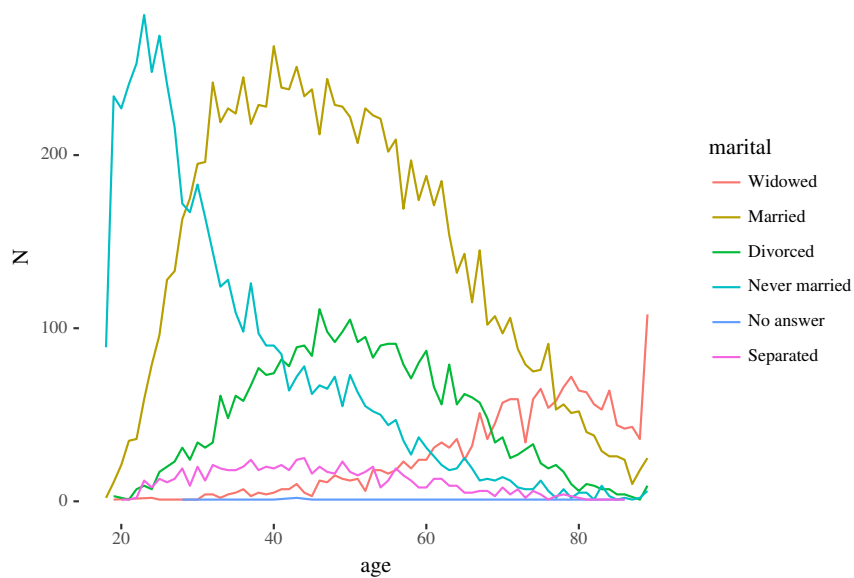


0.9.5.9 Joontega plotil saab `fct_reorder2()` abil assotseerida y väärtused suurimate x väärtustega

See muudab ploti paremini jälgitavaks:

```
## summeerime andmed
gsscat_sum <- filter(gss_cat, !is.na(age)) %>%
  group_by(age, marital) %>%
  mutate(N=n())

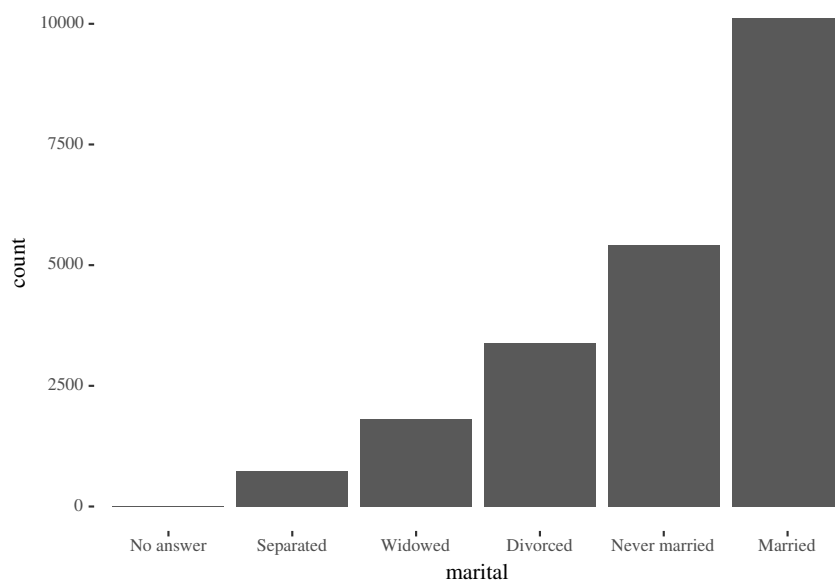
## paneme andmed graafikule
ggplot(gsscat_sum, aes(age, N, colour = fct_reorder2(marital, age, N))) +
  geom_line() +
  labs(colour = "marital")
```



0.9.5.10 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()
```



0.10 Statistilised mudelid

```
library(tidyverse)
library(ggthemes)
library(scatterplot3d)
library(modelr)
library(broom)
```

0.10.1 Suur ja väike maailm

Kuna maailmas on kõik kõigega seotud, on seda raske otse uurida. Teadus töötab tänu sellele, et teadlased lõikavad reaalsuse väikesteks tükkideks, kasutades tordilabidana teaduslike hüpoteese, ning uurivad seda tükikaupa lootuses, et kui kõik tükid on korralikult läbi nätsutatud, saab sellest taas tordi kokku panna. Tüüpiline bioloogiline hüpotees pakub välja tavakeelse (mitte matemaatilise) seletuse mõnele piiritletud loodusnähtusele.

Näiteks antibiootikume uuritakse keemilise sideme tasemel kasutades orgaanilise keemia meetodeid. Antibiootikumide molekulaarseid märklaudu uuritakse molekulaarbioloogiliste meetoditega, nende toimet uuritakse rakubioloogia ja füsioloogia meetoditega, aga kaasajal on väga olulised ka ökoloogilised, evolutsioonilised, meditsiinilised, põllumajanduslikud, majanduslikud ja psühholoogilised aspektid. Kõigil neil tasanditel on loodud palju hüpoteese, millest kokku moodustub meie teadmine antibiootikumide kohta. Neid väga erinevaid asju, mida me kutsume hüpoteesideks, ühendab see, et neist igauht võib võrrelda empiiriliste andmetega. Samuti, enamust neist saab kirjeldada matemaatiliste formalismide ehk mudelite abil, ja neid mudeleid saab omakorda võrrelda andmetega. Kuigi erinevate tasemete hüpoteesid on tavakeeles üksteisest väga erinevad, on neid kirjeldavad mudelid sageli matemaatiliselt sarnased.

Kui mudel on teooria lihtsustus, siis teooria on maailma lihtsustus.

Mis juhtub, kui teie hüpotees on andmetega kooskõlas? Kas see tähendab, et see hüpotees vastab tõele? Või, et see on tõenäoliselt tõene? Kahjuks on vastus mõlemale küsimusele eitav. Põhjuseks on asjaolu, et enamasti leiab iga nähtuse seletamiseks rohkem kui ühe alternatiivse teadusliku hüpoteesi ning rohkem kui üks üksteist välistav hüpotees võib olla olemasolevate andmetega võrdses kooskõlas. Asja teeb veelgi hullemaks, et teoreetiliselt on võimalik sõnastada lõpmata palju erinevaid teooriaid, mis kõik pakuvad alternatiivseid ja üksteist välistavaid seletusi samale nähtusele. Kuna hüpoteese on lõpmatu hulk, aga andmeid on alalt lõplik hulk, siis saab igas teaduslikus “faktis” kahelda.

Kunagi ei või kindel olla, et parimad teooriad ei ole täiesti tähelepanuta jäänud ning, et meie poolt kogutud vähesed andmed kajastavad hästi kõiki võimalikke andmeid.

Ca. 1910 mõtlesid filosoofid Russell ja Moore välja tõe vastavusteooria, mille kohaselt tõest propositsiooni eristab väärast “vastavus” füüsikalisele maailmale. Selle kohaselt on tõesed need laused, mis vastavad asjadele. Ehkki keegi ei oska siiani öelda, mida “vastavus” selles kontekstis ikkagi tähendab, või kuidas seda saavutada, on vastavusteooria senini kõige populaarsem tõeteooria filosoofide hulgas (mis on kõnekas alternatiivide kohta). Samamoodi, kui lausete vastavusest maailmaga, võime rääkida ka võrrandite (ehk mudelite) vastavusest lausetega. Vastavusest lausetaga sellepärast, et mudelid on koostatud teaduslike teooriate, mitte otse maailma, kirjeldusena. Seega ei pea me muretsema mudelite tõeväärtuse pärast. Võib lausa väita, et mudeli tõeväärtusest rääkimine on kohatu.

Teeoria ja mudeli seose kohta selline näide. Meil on hüpotees, mille kohaselt valijad eelistavad demokraatlikus süsteemis kandidaate, kes on ennast juba tõestanud sellega, et saavad hakkama riigi majanduse edendamiseks. Seega, kompetentsed poliitikud valitakse tagasi. Sellest hüpoteesist saab tuletada kaks järeldust 1. - majandusel läheb keskmiselt paremini juba tagasi valitud poliitikute all kui esimest korda valitud poliitikute all, keda ei ole veel elektoraadi poolt harvendatud ja 2. - majandusnäitajate varieeruvus on esimesel juhul väiksem, sest kehvemad poliitikud on juba valija poolt valimist eemaldatud (Achen, C. H., & Bartels, L. M. (2016). *Democracy for Realists*). Esimese järelduse testimiseks kasutati statistilise mudelina aritmeetilist keskmist koos standardveaga ja teise järelduse jaoks standardhälvet. Tulemused olid vastupidised hüpoteesi poolt ennustatutega. Seega: andmed (kas neid on piisavalt? on nad representatiivsed?) → mudel (kindlasti on siin alternatiivseid võimalusi sama küsimuse modelleerimiseks) → teooria järeldus (sama teooria annab ka teisi järeldusi. Mis juhtub, kui osad neist on andmetega kooskõlas ja teised ei ole?) → laiem teooria → järeldus demokraatia toimimise kohta laias maailmas.

0.10.2 Mudeli väike maailm

Ülalmainitud teadusliku meetodi puudused tingivad, et meie huvides on oma teaduslikke probleeme veel ühe taseme võrra lihtsustada, taandades need statistilisteks probleemideks. Selleks tuletame tavakeelsest teaduslikust teooriast täpselt formuleeritud matemaatilise mudeli ning seejärel asume uurima oma mudelit lootuses, et mudeli kooskõla andmetega ütleb meile midagi teadusliku hüpoteesi kohta. Enamasti töötab selline lähenemine siis, kui mudeli ehitamisel arvestati võimaliku andmeid genereeriva mehhanismiga – ehk, kui mudeli matemaatiline struktuur koostati teaduslikku hüpoteesi silmas pidades. Mudelid, mis ehitatakse silmas pidades puhtalt matemaatilist sobivust andmetega, ei kipu omama teaduslikku seletusjõudu, kuigi neil võib olla väga hea ennustusjõud.

Meil on kaks hüpoteesi, A ja B. Juhul kui A on tõene ja B on väär, kas on võimalik, et B on tõele lähemal kui A? Kui A ja B on teineteist välistavad punkthüpoteesid parameetri väärtuse kohta, siis on vastus eitav. Aga mis juhtub, kui A ja B on statistilised mudelid? Näiteks, kui tõde on, et eesti meeste keskmine pikkus on 178.3 cm

ja A ütleb, et keskmine pikkus jääb kuhugi 150 cm ja 220 cm vahele ning B ütleb, et see jääb kuhugi 179 cm ja 182 cm vahele, siis on B tõele lähemal selles mõttes, et meil on temast teaduslikus mõttes rohkem kasu. Siit on näha oluline erinevus teadusliku hüpoteesi ja statistilise mudeli vahel: hüpotees on orienteeritud tõele, samal ajal kui mudel on orienteeritud kasule.

Mudeli maailm erineb päris maailmast selle poolest, et mudeli maailmas on kõik sündmused, mis põhimõtteliselt võivad juhtuda, juba ette teada ja üles loendatud (seda sündmuste kogu kutsutakse parameetruuruumiks). Tehniliselt on mudeli maailmas üllatused võimatud.

Lisaks, tõenäosusteooriat, ja eriti Bayesi teoreemi, kasutades on meil garantii, et me suudame mudelis leiduva informatsiooniga ümber käia parimal võimalikul viisil. Kõik see rõõm jääb siiski mudeli piiridesse. Mudeli eeliseks teooria ees on, et hästi konstrueeritud mudel on lihtsamini mõistetav — erinevalt vähegi keerulisemast teaduslikust hüpoteesist on mudeli eeldused ja ennustused läbinähtavad ja täpselt formuleeritavad. Mudeli puuduseks on aga, et erinevalt teooriast ei ole mingit võimalust, et mudel vastaks tegelikkusele. Seda sellepärast, et mudel on taotluslikult lihtsustav (erandiks on puhtalt ennustuslikud mudelid, mis on aga enamasti läbinähtamatu struktuuriga). Mudel on kas kasulik või kasutu; teooria on kas tõene või väär. Mudeli ja maailma vahel võib olla kaudne peegeldus, aga mitte kunagi otsene side. Seega, ükski number, mis arvutatakse mudeli raames, ei kandu sama numbrina üle teaduslikku ega päris maailma. Ja kogu statistika (ka mitteparameetiline) toimub mudeli väikses maailmas. Arvud, mida statistika teile pakub, elavad mudeli maailmas; samas kui teie teaduslik huvi on suunatud päris maailmale. Näiteks 95% usaldusintervall ei tähenda, et te peaksite olema 95% kindel, et tõde asub selles intervallis – sageli ei tohiks te seda nii julgelt tõlgendada isegi kitsas mudeli maailmas.

0.10.2.1 Näide: Aristoteles, Ptolemaios ja Kopernikus

Aristoteles (384–322 BC) lõi teooria maailma toimimise kohta, mis domineeris haritud eurooplase maailmapilti enam kui 1200 aasta vältel. Tema ühendteooria põhines maailmapildil, mis oli üldtunnustatud juba sajandeid enne Aristotelest ja järgneva 1500 aasta jooksul kahtlesid selles vähesed mõistlikud inimesed. Selle kohaselt asub universumi keskpunktis statsionaarne maakera ning kõik, mida siin leida võib, on tehtud neljast elemendist: maa, vesi, õhk ja tuli. Samas, kogu maailmaruum alates kuu sfäärist on tehtud viiendast elemendist (eeter), mida aga ei leidu maal (nagu nelja elementi ei leidu kuu peal ja sealt edasi). Taevakehad (kuu, päike, planeedid ja kinnistähed) tiirlevad ümber maa kontsentrilistes sfäärides, mille vahel pole vaba ruumi. Seega on kogu liikumine eetri sfäärides ühtlane ja ringikujuline ja see liikumine põhjustab pika põhjus-tagajärg ahela kaudu kõiki liikumisi, mida maa peal kohtame. Kaasa arvatud sündimine, elukäik ja surm. Kõik, mis maa peal huvitavat, ehk kogu liikumine, on algselt põhjustatud esimese liikumise poolt, mille käivitab kõige välimises sfääris paiknev meie jaoks mõistetamatu intellektiga “olend”.

Aristotelese suur teooria ühendab kogu maailmapildi alates meie mõistes keemiast ja kosmoloogiast kuni bioloogia, maateaduse ja isegi geograafiani. Sellist ühendteooriat on erakordselt raske ümber lükata, sest seal on kõik kõigega seotud.

Schema huius præmissæ diuisionis Sphærarum.

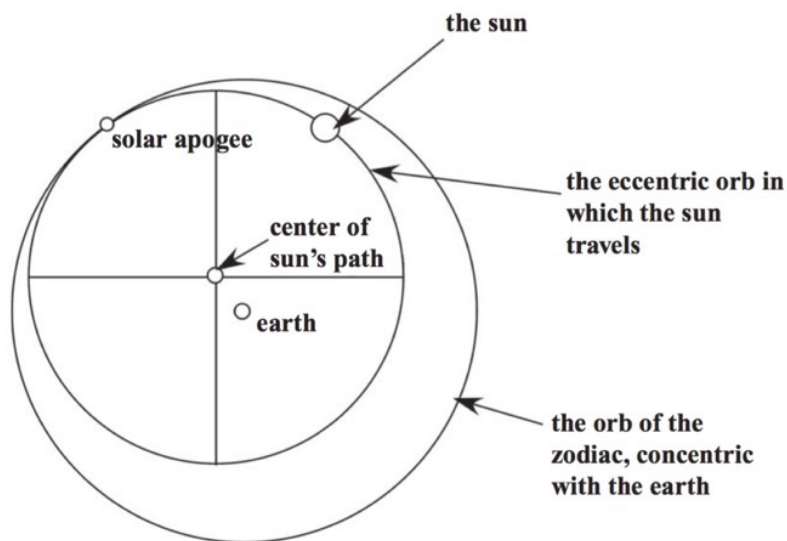


Joonis 3: Keskaegne aristotellik maailm.

Aristarchus (c. 310 – c. 230 BC) proovis seda siiski, väites, et tegelikult tiirleb maakera ümber statsionaarse päikese. Ta uskus ka, et kinnistähed on teised päikesed, et universum on palju suurem kui arvati (ehkki kaasaegne seisukoht oli, et universumi mastaabis ei ole maakera suurem kui liivatera) ning, et maakera pöörleb ümber oma telje. Paraku ei suutnud Aristarchuse geotsentriline teooria toetajaid leida, kuna see ei pidanud vastu vaatluslikule testile. Geotsentrilisest teooriast tuleneb nimelt loogilise paratametusena, et tähtedel esineb maalt vaadates parallaks. See tähendab, et kui maakera koos astronoomiga teeb poolringi ümber päikese, siis kinnistähe näiv asukoht taevavõlvil muutub, sest astronoom vaatab teda teise nurga alt. Pange oma nimetissõrm näost u 10 cm kaugusele, sulgege parem silm, seejärel avage see ning sulgege vasak silm ja te näete oma sõrme parallaksi selle näiva asukoha muutusena. Mõõtmised ei näidanud aga parallaksi olemasolu (sest maa trajektoori diameeter on palju lühem maa kaugusest tähtedest). Parallaksi suudeti esimest korda mõõta alles 1838, siis kui juba iga koolijüts uskus, et maakera tiirleb ümber päikese!

Ühte Aristotelese kosmoloogia olulist puudust nähti siiski kohe. Nimelt ei suuda Aristoteles seletada, miks osad planeedid taevavõlvil vahest suunda muudavad ja mõnda aega lausa vastupidises suunas liiguvad (retrogressioon). Kuna astronoomiat kasutasid põhiliselt astroloogid, siis pöörati planeetide liikumisele suurt tähelepanu. Lahenduseks ei olnud aga mitte suure teooria ümbertegemine või ümberlükkamine, vaid uue teaduse nõudmine, mis “päästaks fenomenid”. Siin tuli appi Ptolemaios (c. AD 100 – c. 170), kes lõi matemaatilise mudeli, kus planeedid mitte lihtsalt ei liigu ringtrajektoori mööda, vaid samal ajal teevad ka väiksemaid ringe ümber esimese suure ringjoone. Neid väiksemaid ringe kutsutakse epitsükliiteks. See mudel suutis planeetide liikumist taevavõlvil piisavalt hästi ennustada, et astroloogide seltskond sellega rahule jäi.

Ptolemaiosel ja tema järgijatel oli tegelikult mitu erinevat mudelit. Osad neist ei sisaldanud epitsükleid ja maakera ei asunud tema mudelites universumi keskel, vaid oli sellest punktist eemale nihutatud — nii et päike ei teinud ringe ümber maakera vaid ümber tühja punkti. Kuna leidus epitsükliitega

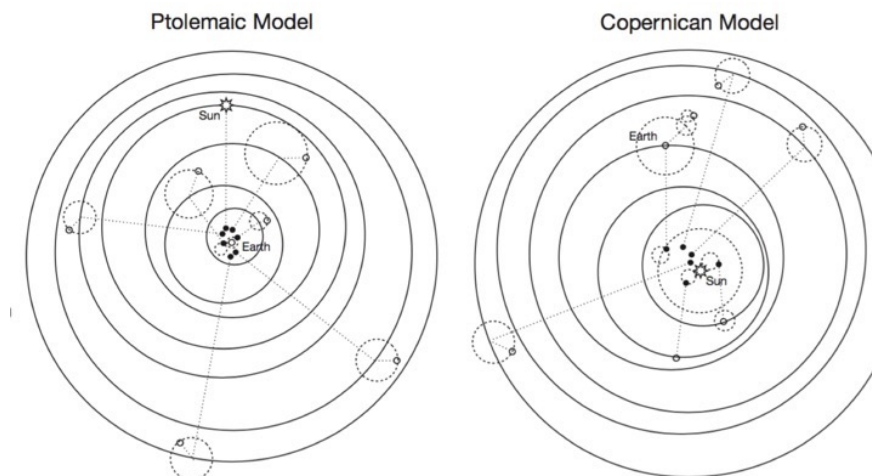


Joonis 4: Ilma epitsükliiteta ptolemaline mudel.

mudel ja ilma epitsükliiteta mudel, mis andsid identseid ennustusi, on selge, et Aristotelese teooria ja fenomenide päästmise mudelid on põhimõtteliselt erinevad asjad. Samal ajal, kui Aristoteles **seletas** maailma põhiolemust põhjuslike seoste jadana (mitte matemaatiliselt), **kirjeldas/ennustas** Ptolemaios sellesama maailma käitumist matemaatiliste (mitte põhjuslike) struktuuride abil.

Nii tekkis olukord, kus maailma mõistmiseks kasutati Aristotelese ühendteooriat, aga selle kirjeldamiseks ja tuleviku ennustamiseks hoopis ptolemailisi mudeleid, mida keegi päriselt tõeks ei pidanud ja mida hinnati selle järgi, kui hästi need “päästsid fenomene”.

See toob meid Kopernikuse (1473 – 1543) juurde, kes teadusajaloolaste arvates vallandas 17. sajandi teadusliku revolutsiooni, avaldades raamatu, kus ta asetab päikese universumi keskele ja paneb maa selle ümber ringtrajektooriga tiirlema. Kas Kopernikus tõrjus sellega kõrvale Aristotelese, Ptolemaiiose või mõlemad? Tubdub, et Kopernikus soovis kolmandat, suutis esimest, ning et tolleaegsete lugejate arvates üritas ta teha teist — ehk välja pakkuda alternatiivi ptolemailistele mudelitele, mis selleks ajaks olid muutunud väga keerukaks (aga ka samavõrra ennustustäpseks). Kuna Kopernikuse raamat läks trükki ajal, mil selle autor oli juba oma surivoodil, kirjutas sellele eessõna üks tema vaimulikust sõber, kes püüdis oodatavat kiriklikku pahameelt leevendada vihjates, et päikese keskele viimine on vaid mudeldamise trikk, millest ei tasu järeldada, et maakera ka tegelikult ümber päikese tiirleb (piibel räägib, kuidas jumal peatas taevavõlvil päikese, mitte maa). Ja kuna eessõna oli anonüümne, eeldasid lugejad muidugi, et selle kirjutas autor. Lisaks, kuigi Kopernikus tõstis päikese keskele, jäi ta planeetide ringikujuliste trajektooride juurde, mis tähendas, et selleks, et tema teooria fenomenide päästmisel hätta ei jääks, oli ta sunnitud maad ja planeete liigutama ümber päikese mõõda epitsükleid. Kokkuvõttes oli Kopernikuse mudel pea-aegu sama keeruline kui Ptolemaiiose standardmudel ja selle abil tehtud ennustused planeetide liikumise kohta olid väiksema täpsusega. Seega, ennustava mudelina ei olnud tal suuri eeliseid ptolemailike mudelite ees.



Joonis 5: Ptolemaiose ja Kopernikuse mudelid on üllatavalt sarnased.

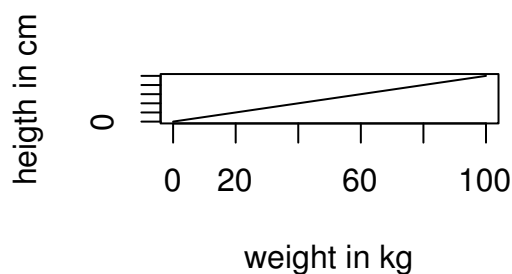
Kopernikuse mudel suutis siiski ennustada mõningaid nähtusi (planeetide näiv heledus jõuab maksimumi nende lähimas asukohas maale), mida Ptolemaiose mudel ei ennustanud. See ei tähenda, et need fenomenid oleksid olnud vastuolus Ptolemaiose mudeliga. Lihtsalt, nende Ptolemaiose mudelisse sobitamiseks oli vaja osad mudeli parameetrid fikseerida nii-öelda suvalistele väärtustele. Seega Koperniku mudel töötas nii, nagu see oli, samas kui Ptolemaiose mudel vajas ad hoc tuunimist.

Kui vaadata Koperniku produkti teooriana, mitte mudelina, siis oli sellel küll selgeid eeliseid Aristotelese maailmateooria ees. Juba ammu oli nähtud komeete üle taevavölvi lendamas (mis Aristotelese järgi asusid kinnistähtede muutumatus sfääris), nagu ka supernoova tekkimist ja kadu, ning enam ei olnud kaugel aeg, mil Galileo joonistas oma teleskoobist kraatreid kuu pinnal, näidates, et kuu ei saanud koosneda täiuslikust viiendast elemendist ja et sellel toimusid ilmselt sarnased füüsikalised protsessid kui maal. On usutav, et kui Kopernikus oleks jõudnud oma raamatule ise essõna kirjutada, oleks tema teooria vastuvõtt olnud palju kiirem (ja valulisem).

0.10.3 Lineaarsed mudelid

Oletame, et me mõõtsime N inimese pikkuse cm-s ja kaalu kg-s ning meid huvitab, kuidas inimeste pikkus sõltub nende kaalust. Lihtsaim mudel pikkuse sõltuvusest kaalust on $\text{pikkus} = \text{kaal}$ (formaliseeritult: $y = x$) ja see mudel ennustab, et kui Johni kaal = 80 kg, siis John on 80 cm pikkune. Siin on pikkus muutuja, mille väärtust ennustatakse ja kaal muutuja, mille väärtuste põhjal ennustatakse pikkusi. Selle mudeli saame graafiliselt kujutada nii:

```
x <- 0:100 #y = kaal
y <- x # x = pikkus
plot(y ~ x,
     type = "l",
     xlab = "weight in kg",
     ylab = "height in cm")
```



Mudeli keeles tähistame me seda, mida me ennustame (antud juhul pikkus) Y-ga ja seda, mille väärtuse põhjal me ennustame (antud juhul kaal) X-ga. Seega sirge mudeli matemaatiline formalism on $Y = X$.

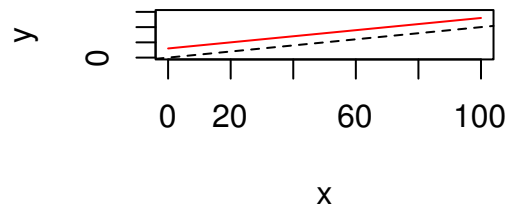
See on äärmiselt jäik mudel: sirge, mille asukoht on rangelt fikseeritud. Sirge lõikab y telge alati 0-s (mudeli keeles: sirge intercept ehk lõikepunkt Y teljel = 0) ja tema tõusunurk saab olla ainult 45 kraadi (mudeli keeles: mudeli slope ehk tõus = 1). Selle mudeli jäikus tuleneb sellest, et temas ei ole parameetreid, mille väärtusi me saaksime vabalt muuta ehk tuunida.

Mis juhtub, kui me lisame mudelisse konstandi, mille liidame x-i väärtustele?

$$y = a + x$$

See konstant on mudeli parameeter, mille väärtuse võime vabalt valida. Järgnevalt anname talle väärtuse 30 (ilma konkreetse põhjusega).

```
x <- 0:100
a <- 30
y <- a + x
plot(y ~ x,
     xlim = c(0, 100),
     ylim = c(0, 150),
     col = "red",
     type = "l")
abline(c(0, 1), lty = 2)
```



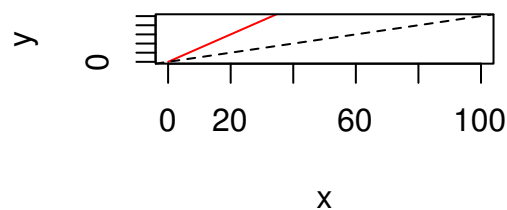
Meie konstant a määrab y väärtuse, kui $x=0$, ehk sirge lõikepunkti y teljel. Teisisõnu, a = mudeli intercept

Mis juhtub, kui me mitte ei liida, vaid korrutame x -i konstandiga?

$$y = bx$$

Jällegi, me anname mudeli parameetrile b suvalise väärtuse, 3.

```
x <- 0:200 #y = kaal
b <- 3
y <- b*x # x = pikkus
plot(y~x,
     xlim=c(0, 100),
     ylim=c(0, 100),
     col="red",
     type="l")
abline(c(0,1), lty=2)
```



Nüüd muutub sirge tõusunurk, ehk kui palju me ootame y -t muutumas, kui x muutub näiteks ühe ühiku võrra. Kui $b = 3$, siis x -i tõustes ühe ühiku võrra suureneb y kolme ühiku võrra. Proovi järgi, mis juhtub, kui $b = -3$.

Selleks, et sirget kahes dimensioonis vabalt liigutada, piisab kui me kombineerime eelnevad näited ühte:

$$y = a + bx$$

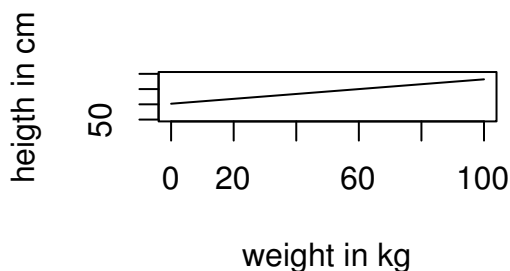
Selleks lisame mudelisse kaks parameetrit, intercept (a) ja tõus (b). Kui $a = 0$ ja $b = 1$, saame me eelpool kirjeldatud mudeli $y = x$. Kui $a = 102$, siis sirge lõikab y telge väärtusel 102. Kui $b = 0.8$, siis x-i tõustes 1 ühiku võrra tõuseb y-i väärtus 0.8 ühiku võrra. Kui $a = 100$ ja $b = 0$, siis saame sirge, mis on paraleelne x-teljega ja lõikab y-telge väärtusel 100. Seega, Teades a ja b väärtusi ning omistades x-le suvalise meid huvitava väärtuse, saab ennustada y-i keskmist väärtust sellel x-i väärtusel. Näiteks, olgu andmete vastu fititud model:

$$\text{pikkus(cm)} = 102 + 0.8 * \text{kaal(kg)} \text{ ehk}$$

$$y = 102 + 0.8x.$$

Omistades nüüd kaalule väärtuse 80 kg, tuleb mudeli poolt ennustatud keskmine pikkus $102 + 0.8 * 80 = 166$ cm. Iga kg lisakaalu ennustab mudeli kohaselt 0.8 cm võrra suuremat pikkust.

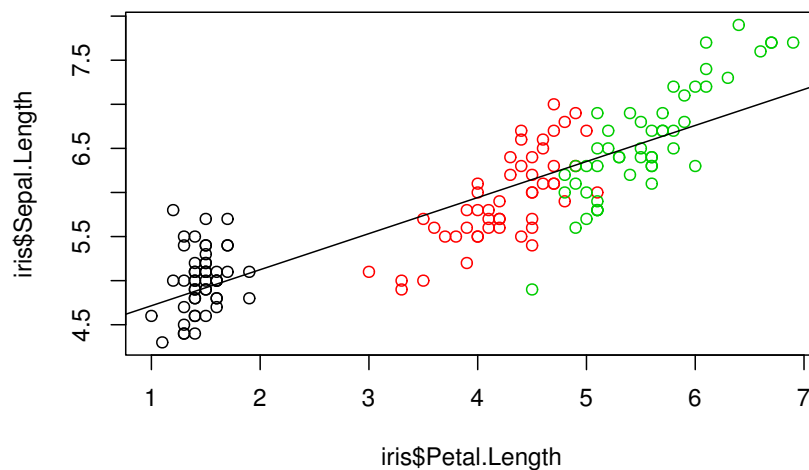
```
a <- 102
b <- 0.8
x <- 0:100
y <- a + b * x
plot(y ~ x,
     type = "l",
     xlab = "weight in kg",
     ylab = "height in cm",
     ylim = c(50, 200))
```



See mudel ennustab, et 0 kaalu juures on pikkus 102 cm, mis on rumal, aga mudelite puhul tavaline, olukord. Me tuunime mudelit andmete peal, mis ei sisalda 0-kaalu. Meie valimiandmed ei peegelda täpselt inimpopulatsiooni. Sirge mudel ei peegelda täpselt pikkuse-kaalu suhteid vahemikus, kus meil on reaalseid kaaluandmeid; ja ta teeb seda veelgi vähem seal, kus meil mõõdetud kaalusid ei ole. Seega pole mõtet imestada, miks mudeli intercept meie üle irvitab.

Kahe parameetriga sirge mudel ongi see, mida me fitime kahedimensiooniliste andmetega. Näiteks nii:

```
# fit a linear model and name the model object as m1
m1 <- lm(Sepal.Length ~ Petal.Length, iris)
# make a scatter plot, colored by the var called "Species"
plot(iris$Sepal.Length ~ iris$Petal.Length, col = iris$Species)
# draw the fitted regression line from m1
abline(m1)
```



Mudeli fitimine tähendab siin lihtsalt, et sirge on 2D ruumi asetatud nii, et see oleks võimalikult lähedal kõikidele punktidele.

oletame, et meil on n andmepunkti ja et me fitime neile sirge. Nüüd plotime fititud sirge koos punktidega ja tõmbame igast punktist mudelsirgeni joone, mis on paraleelne y -teljega. Seejärel mõõdame nende n joone pikkused. Olgu need pikkused a, b, \dots i. $\text{lm}()$ funktsioon fitib sirge niimoodi, et summa $a^2 + b^2 + \dots + i^2$ oleks minimaalne. Seda kutsutakse vähimruutude meetodiks.

Fititud koefitsientide väärtused saame nii

```
coef(m1)
#> (Intercept) Petal.Length
#>      4.307      0.409
```

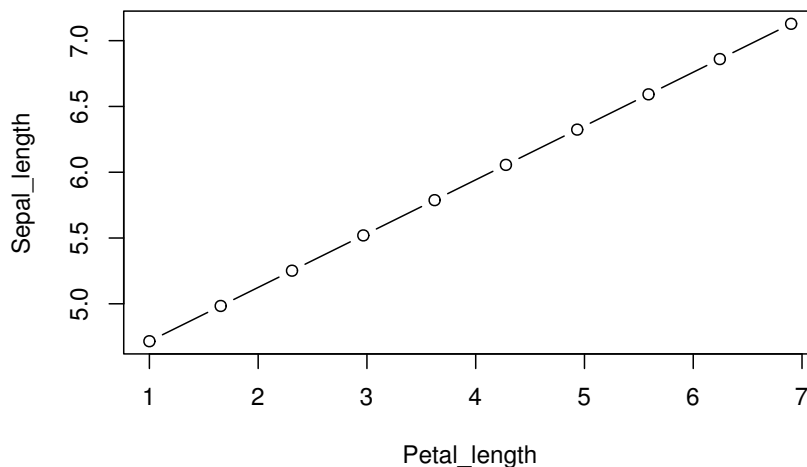
Siin $a = (\text{Intercept})$ ehk 4.31 ja $b = \text{Petal.Length}$ ehk 0.41.

Ennustus lineaarsest mudelist

Anname x-le rea väärtusi, et ennustada y keskmisi väärtusi nendel x-i väärtustel. Siin me ennustame y (Sepal_length) keskväärtusi erinevatel x-i (Petal_length) väärtustel, mitte individuaalseid Sepal_length väärtusi. Me kasutame selleks deterministlikku mudelit kujul $\text{Sepal_length} = a + b * \text{Petal_length}$. Hiljem õpime ka bayesiaanlike meetoditega individuaalseid Sepal_length-e ennustama.

Järgnev kood on sisuliselt sama, millega me üle-eelmisel plotil joonistasime mudeli $y = a + bx$. Me fikseerime mudeli koefitsiendid fititud irise mudeli omadega ja anname Petal_length muutujale 10 erinevat väärtust originaalse muutuja mõõtmisvahemikus. Aga sama hästi võiksime ekstrapoleerida ja küsida, mis on oodatav Sepal_length, kui Petal_length on 100 cm? Loll küsimus, aga mudel ei tea seda. Proovi seda kodus.

```
Petal_length <- seq(min(iris$Petal.Length),  
                    max(iris$Petal.Length),  
                    length.out = 10)  
  
a <- coef(m1)[1]  
b <- coef(m1)[2]  
  
Sepal_length <- a + b*Petal_length  
  
plot(Sepal_length~Petal_length, type="b")
```



Siin ennustasime 10 y väärtust 10-l x-i väärtusel.

0.10.3.1 Neli mõistet

Mudel $y = a + bx$ on x ja y muutujad, ning a ja b on parameetrid. Muutujate väärtused fikseeritakse andmete poolt, parameetrid fititakse andmete põhjal. Fititud mudel ennustab igale x -i väärtusele vastava kõige tõenäolisema y väärtuse (y keskvaartuse sellel x -i väärtusel).

Y - mida me ennustame (dependent variable, predicted variable)

X - mille põhjal me ennustame (independent variable, predictor)

muutuja (variable) - iga asi, mida me valimis mõõdame (X ja Y on kaks muutujat). Muutujal on sama palju fikseeritud väärtusi kui meil on selle muutuja kohta mõõtmisandmeid.

parameeter (parameter) - mudeli koefitsient, millele võib omistada suvalisi väärtusi. Parameetreid tuunides fitime mudeli võimalikult hästi sobituma andmetega.

Mudel on matemaatilise formalism, mis püüab kirjeldada füüsikalist protsessi. Statistilise mudeli struktuuris on komponent, mis kirjeldab ideaalseid ennustusi (nn protsessi mudel) ja eraldi veakomponent (ehk veamudel), mis kirjeldab looduse varieeruvust nende ideaalsete ennustuste ümber. Mudeli koostisosad on (i) muutuja, mille väärtusi ennustatakse, (ii), muutuja(d), mille väärtuste põhjal ennustatakse, (iii) parameetrid, mille väärtused fititakse ii põhjal ja (iv) konstandid.

0.10.3.2 Mudeli fittimine

Mudelid sisaldavad (1) matemaatilisi struktuure, mis määravad mudeli tüübi ning (2) parameetreid, mida saab andmete põhjal tuunida, niiviisi täpsustades mudeli kuju.

Seda tuunimist nimetatakse mudeli fittimiseks. Mudelit fittides on eesmärk saavutada antud tüüpi mudeli maksimaalne sobivus andmetega. Näiteks võrrand $y = a + bx$ määrab mudeli, kus $y = x$ on on see struktuur, mis tagab, et mudeli tüüp on sirge, ning a ja b on parameetrid, mis määravad sirge asendi.

Seevastu struktuur $y = x + x^2$ tagab, et mudeli $y = a + b_1x + b_2x^2$ tüüp on parabool, ning parameetrite a , b_1 ja b_2 väärtused määravad selle parabooli täpse kuju. Ja nii edasi.

lineraarse mudeli parima sobivuse andmetega saab tagada kahel erineval viisil: (i) vähimruutude meetod mõõdab y telje suunaliselt iga andmepunkti kauguse mudeli ennustusest, võtab selle kauguse ruutu, summeerib kauguste ruudud ning leiab sirge asendi, mille korral see summa on minimaalne; (ii) Bayesi teoreem annab väheinformatiivse prioriori korral praktiliselt sama fiti.

Hea mudel on

- (1) võimalikult lihtsa struktuuriga, mille põhjal on veel võimalik teha järeldusi protsessi kohta, mis genereeris mudeli fittimiseks kasutatud andmeid;
- (2) sobitub piisavalt hästi andmetega (eriti uute andmetega, mida ei kasutatud selle mudeli fittimiseks), et olla relevantne andmeid genereeriva protsessi kirjeldus;
- (3) genereerib usutavaid simuleeritud andmeid.

Sageli fititakse samade andmetega mitu erinevat tüüpi mudelit ja püütakse otsustada, milline neist vastab kõige paremini eeltoodud tingimustele. Näiteks, kui sirge suudab kaalu järgi pikkust ennustada paremini kui parabool, siis on sirge mudel paremas kooskõlas teadusliku hüpoteesiga, mis annaks mehhanismi protsessile, mille käigus kilode lisandumine viiks laias kaaluvahemikus inimeste pikkuse kasvule ilma, et pikkuse kasvu tempo kaalu tõustes langeks.

See, et teie andmed sobivad hästi mingi mudeliga, ei tähenda automaatselt, et see fakt oleks teaduslikult huvitav. Mudeli parameetrid on mõtekad mudeli matemaatilise kirjelduse kontekstis, aga mitte tingimata suure maailma põhjusliku seletamise kontekstis. Siiski, kui mudeli matemaatiline struktuur loodi andmeid genereeriva loodusliku protsessi olemust silmas pidades, võib mudeli koefitsientide uurimisest selguda olulisi tõsiasi suure maailma kohta.

Mudeli fittimine: X ja Y saavad oma väärtused otse andmetest; parameetrid võivad omandada ükskõik millise väärtuse.

Fititud mudelist ennustamine: X-le saab omistada ükskõik millise väärtuse; parameetrite väärtused on fikseeritud; Y väärtus arvutatakse mudelist.

0.10.3.2.1 Üle- ja alafittimine

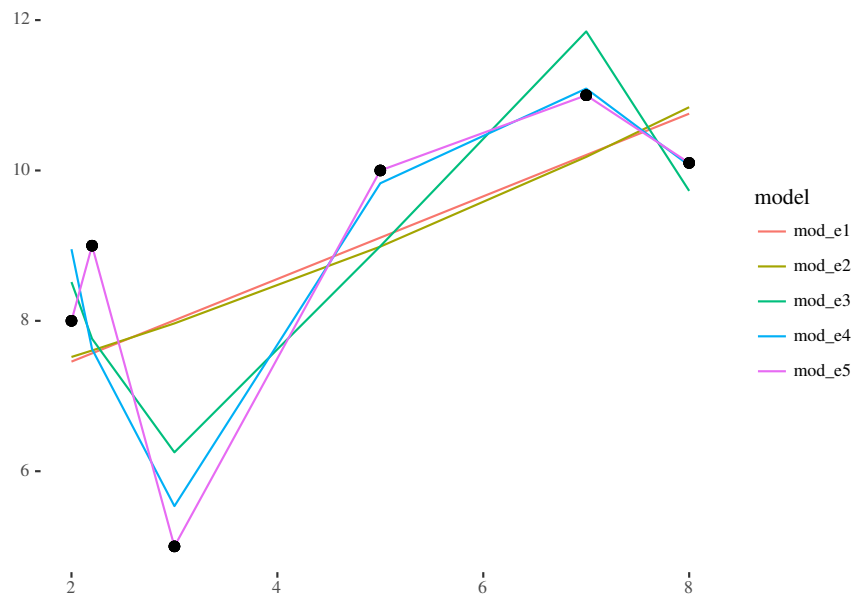
Osad mudelite tüübid on vähem paindlikud kui teised (parameetreid tuunides on neil vähem liikumisruumi). Kuigi sellised mudelid sobituvad halvemini andmetega, võivad need ikkagi paremini kui mõni paindlikum mudel välja tuua andmete peidetud olemuse. Mudeldamine eeldab, et me usume, et meie andmetes leidub nii müra (mida mudel võiks ignoreerida), kui signaal (mida mudel püüab tabada). Kuna mudeli jaoks näeb müra samamoodi välja, kui signaal, on iga mudel kompromiss üle- ja alafittimise vahel. Me lihtsalt loodame, et meie mudel on piisavalt jäik, et mitte liiga palju müra modelleerida ja samas piisavalt paindlik, et piisaval määral signaali tabada.

Üks kõige jäigemaid mudeleid on sirge, mis tähendab, et sirge mudel on suure tõenäosusega alafittitud. Keera sirget kuipalju tahad, ikka ei sobitu ta enamiku andmekogudega. Ja need vähesed andmekogud, mis sirge mudeliga sobivad, on genereeritud teatud tüüpi lineaarsete protsesside poolt. Sirge on seega üks kõige paremini tõlgendatavaid mudeleid. Teises äärmuses on polünoomsed mudelid, mis on väga paindlikud, mida on väga raske tõlgendada ja mille puhul esineb suur mudeli ülefittimise oht. Ülefittitud mudel järgib nii täpselt valimiandmeid, et sobitub hästi valimis leiduva juhusliku müraga ning seetõttu sobitub halvasti järgmise valimiga samast populatsioonist (igal valimil on oma juhuslik müra). Üldiselt, mida rohkem on mudelis tuunitavaid parameetreid, seda paindlikum on mudel, seda kergem on seda valimiandmetega sobitada ja seda raskem on seda tõlgendada. Veelgi enam, alati on võimalik konstrueerida mudel, mis sobitub täiuslikult kõikide andmepunktidega (selle mudeli parameetrite arv = N). Selline mudel on täpselt sama informatiivne kui andmed, mille põhjal see fititi — ja täiesti kasutu.

```
dfr <- tibble(x = c(2, 3, 2.2, 5, 7, 8),
              y = c(8, 5, 9, 10, 11, 10.1))
mod_e1 <- lm(y ~ x, data = dfr)
mod_e2 <- lm(y ~ poly(x, 2), data = dfr)
mod_e3 <- lm(y ~ poly(x, 3), data = dfr)
mod_e4 <- lm(y ~ poly(x, 4), data = dfr)
mod_e5 <- lm(y ~ poly(x, 5), data = dfr)

dfr %>%
  gather_predictions(mod_e1, mod_e2, mod_e3, mod_e4, mod_e5) %>%
  ggplot(aes(x, pred, colour = model)) +
  geom_line() +
  geom_point(aes(x, y), color = "black", size = 2) +
  theme(axis.title = element_blank())
```

Vähimruutude meetodil fititud mudeleid saame võrrelda AIC-i näitaja järgi. AIC - Akaike Informatsiooni Kriteerium - vaatab mudeli sobivust andmetega ja mudeli parameetrite arvu. Väikseim AIC tähtib parimat fititi väikseima parameetrite arvu juures (kompromissi) ja väikseima AIC-ga mudel on eelistatuim mudel. Aga seda ainult võrreldud mudelite hulgas. AIC-i absoluutväärtus ei loe - see on suhteline näitaja.



Joonis 6: Kasvava paindlikusega polünoomsed mudelid. mod_e1 on sirge võrrand $y = a + b_1x$ (2 parameetrit: a ja b_1), mod_e2 on lihtsaim võimalik polünoom: $y = a + b_1x + b_2x^2$ (3 parameetrit), ..., mod_e5: $y = a + b_1x + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5$ (6 parameetrit). mod_e5 vastab täpselt andmepunktidele ($N = 6$).

```
AIC(mod_e1, mod_e2, mod_e3, mod_e4, mod_e5)
#>      df  AIC
#> mod_e1  3 27.8
#> mod_e2  4 29.8
#> mod_e3  5 26.2
#> mod_e4  6 25.1
#> mod_e5  7 -Inf
```

AIC näitab, et parim mudel on mod_e4. Aga kas see on ka kõige kasulikum mudel? Mis siis, kui 3-s andmepunkt on andmesisestaja näpuviga?

Ülefittimise vältimiseks kasutavad Bayesi mudelid informatiivseid prioreid, mis välistavad ekstreemsed parameetriväärtused.

Vt <http://eleventh.org/blog/2017/08/22/there-is-always-prior-information/>

0.10.3.3 kaks lineaarse mudeli laiendust.

0.10.3.3.1 mitme sõltumatu prediktoriga mudel

Esiteks vaatame mudelit, kus on mitu prediktorit x_1, x_2, \dots, x_n , mis on additiivse mõjuga. See tähendab,

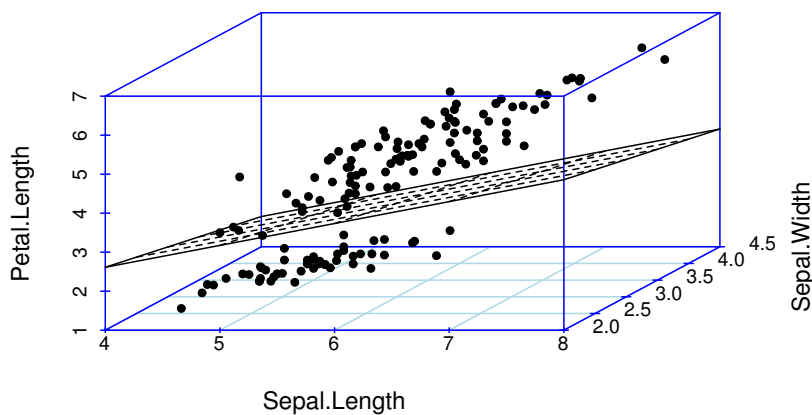
et me liidame nende mõjud, mis omakorda tähendab, et me usume, et $x_1 \dots x_n$ mõjud y -i väärtusele on üksteisest sõltumatud. Mudel on siis kujul

$$y = a + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

mitme prediktoriga mudeli iga prediktori tõus (beta koefitsient) ütleb, mitme ühiku võrra ennustab mudel y muutumist juhul kui see prediktor muutub ühe ühiku võrra ja kõik teised prediktorid ei muutu üldse. Seega pole teiste (kollapseeritud) prediktorite absoluutväärtus ennustusel oluline.

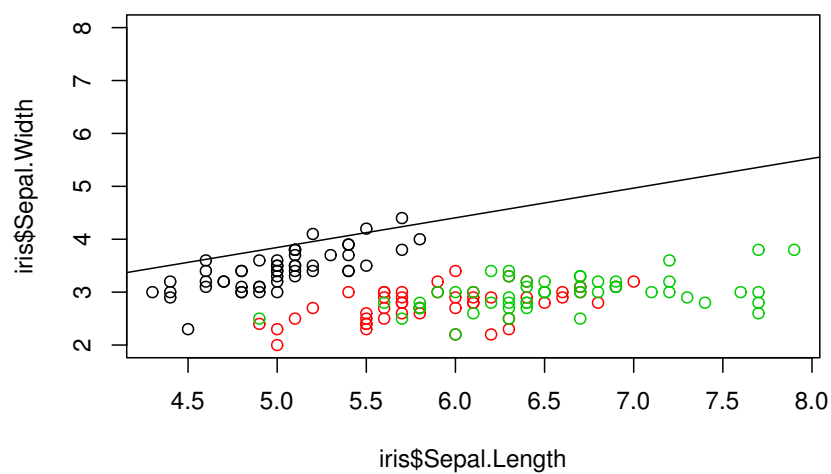
Kui meie andmed on mõõdetud 3D-s ja me tahame ennustada ühe muutuja väärtust kahe teise muutuja väärtuste põhjal (meil on 2 prediktorit), siis tuleb meie 3 parameetriga lineaarne regressioonimudel tasapinna kujul. Kui meil on 4 prediktoriga mudel, siis me liigume 4-mõõtmelisse ruumi, jne. 3D ruumi on veel võimalik mõistlikult plottida.

```
library(scatterplot3d)
# prepare a df of 3 cols:
# Sepal.Length -- Sepal.Width -- Petal.Length (x -- y -- z)
iris1 <- iris[, 1:3]
s3d <- scatterplot3d(iris1, angle = 55, scale.y = 0.7, pch = 20,
                     col.axis = "blue", col.grid = "lightblue" )
my.lm <- lm(Sepal.Width ~ Sepal.Length + Petal.Length, data = iris1)
s3d$plane3d(my.lm, lty.box = "solid")
```



Seda mudelit saab kaeda 2D ruumis, kui kollapseerida kolmas mõõde konstandile.

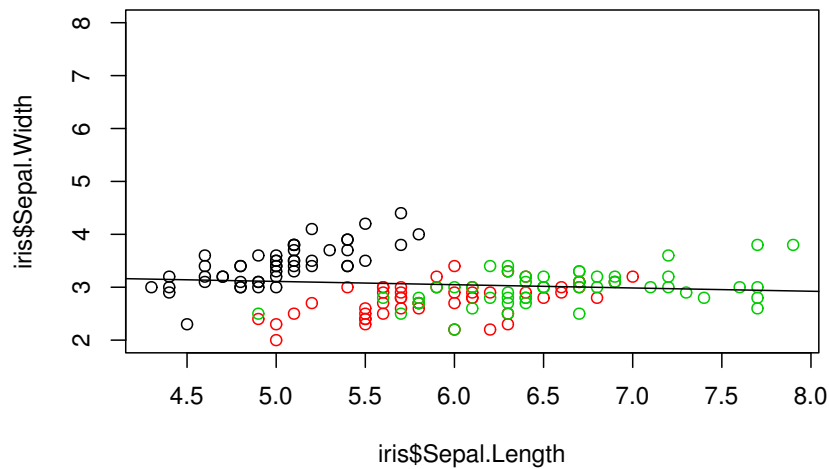
```
m1 <- lm(Sepal.Width ~ Sepal.Length + Petal.Length, data = iris)
plot(iris$Sepal.Width ~ iris$Sepal.Length, ylim = c(2, 8), col = iris$Species)
abline(m1)
#> Warning in abline(m1): only using the first two of 3
#> regression coefficients
```



Enam ei läbi sirge punkte, nagu ta seda 3D ruumis tegi.

Võrlduseks ühe prediktoriga mudel

```
m <- lm(Sepal.Width ~ Sepal.Length, data = iris)
plot(iris$Sepal.Width ~ iris$Sepal.Length, ylim = c(2, 8), col = iris$Species)
abline(m)
```



Nõnda võrdleme kahe mudeli koefitsiente

```
coef(m)
#> (Intercept) Sepal.Length
#>      3.4189      -0.0619
coef(m1)
#> (Intercept) Sepal.Length Petal.Length
#>      1.038      0.561      -0.335
```

Nagu näha, mudeli m b_1 koefitsient erineb oluliselt mudeli m_1 vastavast koefitsiendist.

Kumb mudel on siis parem? AIC-i järgi on m_1 kõvasti parem, kui m .

```
AIC(m, m1)
#>    df    AIC
#> m    3 179.5
#> m1   4  92.1
```

Ennustused sõltumatute prediktoritega mudelist

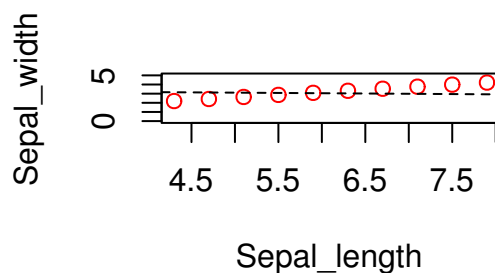
Siin on idee kasutada fititud mudeli struktuuri enustamiseks y keskmisi väärtusi erinevatel x_1 ja x_2 väärtustel. Kuna mudel on fititud, on parameetrite väärtused fikseeritud.

```
Sepal_length <- seq(min(iris$Sepal.Length),
                    max(iris$Sepal.Length),
                    length.out = 10)
Petal_length <- mean(iris$Petal.Length)
a <- coef(m1)[1]
```

```

b1 <- coef(m1)[2]
b2 <- coef(m1)[3]
Sepal_width <- a + b1 * Sepal_length + b2 * Petal_length
plot(Sepal_width ~ Sepal_length,
     type = "b",
     ylim = c(0, 5),
     col = "red")
abline(c(coef(m)[1], coef(m)[2]), lty = 2) # prediction from the single predictor model

```



Nüüd joonistame 3D pildi olukorrast, kus nii x_1 kui x_2 omandavad rea väärtusi. Mudeli ennustus on ikkagi sirge kujul – mis sest, et 3D ruumis.

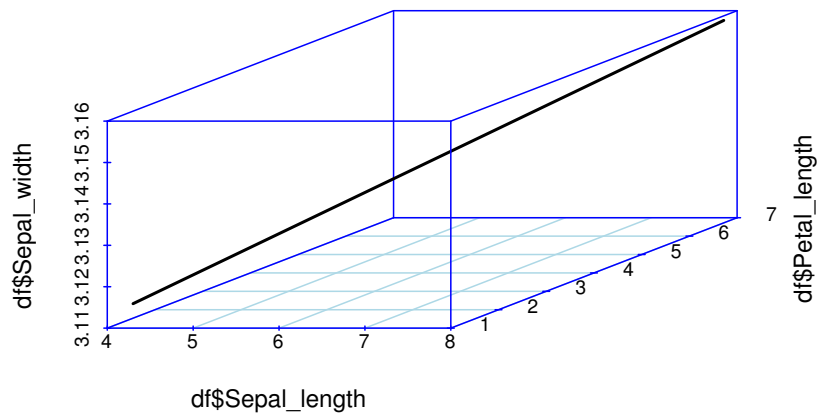
```

Sepal_length <- seq(min(iris$Sepal.Length),
                    max(iris$Sepal.Length),
                    length.out = 10)
Petal_length <- seq(min(iris$Petal.Length),
                    max(iris$Petal.Length),
                    length.out = 10)

a <- coef(m1)[1]
b1 <- coef(m1)[2]
b2 <- coef(m1)[3]
Sepal_width <- a + b1 * Sepal_length + b2 * Petal_length
df <- data.frame(Sepal_width = Sepal_width,
                  Sepal_length = Sepal_length,
                  Petal_length = Petal_length)

scatterplot3d(df$Sepal_length,
              df$Petal_length,
              df$Sepal_width,
              col.axis = "blue",
              col.grid = "lightblue",
              type = "l", lwd = 2)

```

0.10.3.4 Interaktsioonimudel - ühe prediktori mõju sõltub teise prediktori väärtusest

$$y = a + b_1x_1 + b_2x_2 + b_3x_1x_2$$

Interaktsioonimudeli koefitsientide tõlgendamine on keerulisem. b_1 on otse tõlgendatav ainult siis, kui $x_2=0$ (ja b_2 ainult siis, kui $x_1=0$). Hiljem õpime selliseid mudeleid graafiliselt tõlgendama. Mudeli koefitsientide otse tõlgendamine ei ole siin sageli perspektiivikas.

Interaktsioonimudelis sõltub x_1 mõju tugevus y -le x_2 väärtusest. Selle sõltuvuse määra kirjeldab b_3 (x_1 & x_2 interaktsiooni tugevus). Samamoodi ja sümmeetriliselt erineb ka x_2 mõju erinevatel x_1 väärtustel. Ainult siis, kui $x_2 = 0$, ennustab x_1 tõus 1 ühiku võrra y muutust b_1 ühiku võrra.

Interaktsioonimudeli 2D avaldus on kurvatuuriga tasapind, kusjuures kurvatuuri määrab b_3 .

Interaktsiooniga mudel on AIC-i järgi pisut vähem eelistatud võrreldes m_1 -ga. Seega, eriti lihtsuse huvides, eelistame m_1 -e.

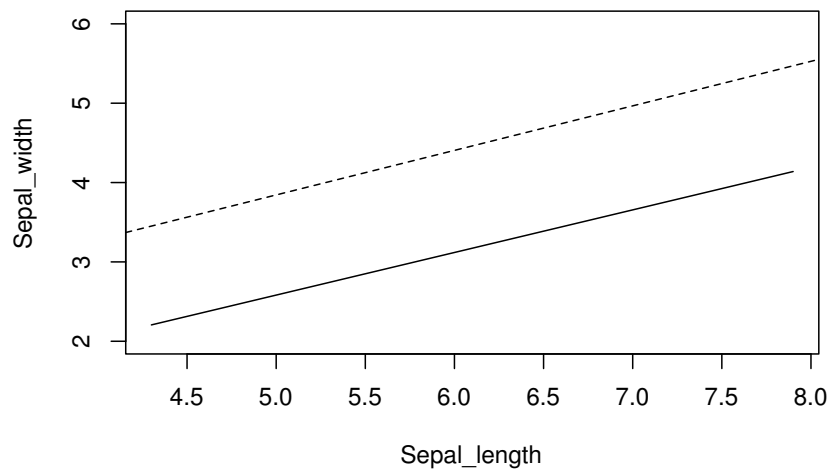
```
m2 <- lm(Sepal.Width~Sepal.Length +
          Petal.Length +
          Sepal.Length*Petal.Length, data = iris)
AIC(m, m1, m2)
#> df AIC
#> m 3 179.5
```

```
#> m1  4  92.1  
#> m2  5  93.4
```

ennustused interaktsioonimudelitest

Kõigepealt anname rea väärtusi x_1 -le ja hoiame x_2 konstantsena.

```
Sepal_length <- seq(min(iris$Sepal.Length),  
                    max(iris$Sepal.Length),  
                    length.out = 100)  
Petal_length <- mean(iris$Petal.Length)  
  
a <- coef(m2)[1]  
b1 <- coef(m2)[2]  
b2 <- coef(m2)[3]  
b3 <- coef(m2)[4]  
  
Sepal_width <- a + b1 * Sepal_length +  
              b2 * Petal_length +  
              b3 * Sepal_length * Petal_length  
  
plot(Sepal_width ~ Sepal_length, type = "l", ylim = c(2, 6))  
abline(m1, lty = 2)  
#> Warning in abline(m1, lty = 2): only using the first  
#> two of 3 regression coefficients
```



Tulemuseks on sirge, mis on paralleelne ilma interaktsioonita mudeli ennustusele (katkendjoon)

Nagu näha, korrutamistehe viib selleni, et interaktsioonimudeli tõus erineb ilma interaktsioonita mudeli tõusust.

Kui aga interaktsioonimudel plottida välja 3D-s üle paljude x_1 ja x_2 väärtuste, saame me regressioonikurvi (mitte sirge), kus b_3 annab kurvatuuri.

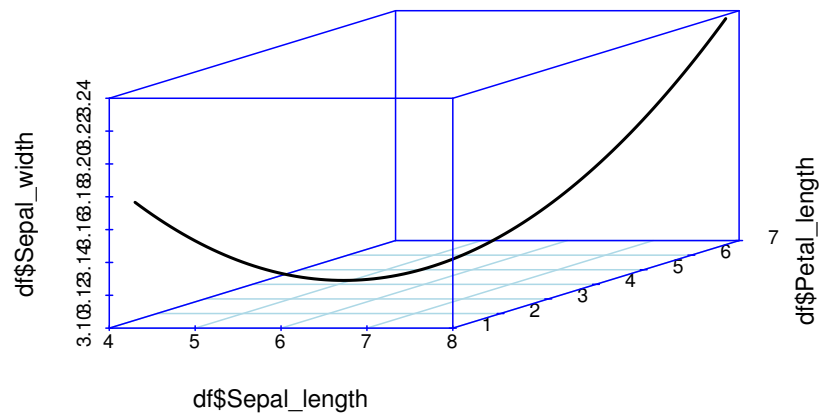
```
Sepal_length <- seq(min(iris$Sepal.Length),
                    max(iris$Sepal.Length),
                    length.out = 100)
Petal_length <- seq(min(iris$Petal.Length),
                    max(iris$Petal.Length),
                    length.out = 100)

a <- coef(m2)[1]
b1 <- coef(m2)[2]
b2 <- coef(m2)[3]
b3 <- coef(m2)[4]

Sepal_width <- a + b1 * Sepal_length +
  b2 * Petal_length +
  b3 * Sepal_length * Petal_length

df <- data.frame(Sepal_width = Sepal_width,
                 Sepal_length = Sepal_length,
                 Petal_length = Petal_length)

scatterplot3d(df$Sepal_length,
              df$Petal_length,
              df$Sepal_width,
              col.axis = "blue",
              col.grid = "lightblue",
              type = "l", lwd = 2)
```



Vau! See on alles ennustus!

Bibliography

- Bååth, R. (2013). Bayesian first aid. *tba*.
- Bååth, R. (2016). *bayesboot: An Implementation of Rubin's (1981) Bayesian Bootstrap*. R package version 0.2.1.
- Bürkner, P.-C. (2017). brms: An R package for bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28.
- Gabry, J. and Mahr, T. (2017). *bayesplot: Plotting for Bayesian Models*. R package version 1.4.0.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2014). *Bayesian data analysis*, volume 2. CRC press Boca Raton, FL.
- Kruschke, J. (2015). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press / Elsevier, 2nd edition.
- Marwick, B., Boettiger, C., and Mullen, L. (2017). Packaging data analytical work reproducibly using r (and friends). *PeerJ Preprints*, 5:e3192v1.
- McElreath, R. (2015). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. CRC Press.
- McElreath, R. (2016). *rethinking: Statistical Rethinking book package*. R package version 1.59.
- Stan Development Team (2016). *rstanarm: Bayesian applied regression modeling via Stan*. R package version 2.13.1.
- Wickham, H., Danenberg, P., and Eugster, M. (2017). *roxygen2: In-Line Documentation for R*. R package version 6.0.1.

