Reprodutseeritav andmeanalüüs kasutades R programmi

Taavi Päll, Ülo Maiväli

2017-09-15

Contents

Chapter 1

Haara kannel, Vanemuine!

Bayesi tõlgenduses on tõenäosus teadlase usu määr mingi hüpoteesi kehtimisse. Hüpotees võib näiteks olla, et järgmise juulikuu sademete hulk Vilsandil jääb vahemikku 22 kuni 34 mm. Kui Bayesi arvutus annab selle hüpoteesi tõenäosuseks 0.57, siis oleme me selle teadmise najal nõus maksma mitte rohkem kui 57 senti kihlveo eest, mille alusel makstakse juhul, kui see hüpotees tõeseks osutub, välja 1 euro (ja me saame vähemalt 43 senti kasumit).

Chapter 2

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 kired arengud bayesi statistikas on tänaseks juba viinud selle suurel määral tavakasutajale kättesaadavasse vormi, toovad lähiaastad selles vallas veel suuri muutusi. Nende muutustega koos peab arenema ka bayesi õpetamine.

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

Lugejad, kellele on juba õpetatud sageduslikku statistikat, võivad tahta teada, mille poolest see erineb bayesi statistikast. Ehkki me usume, et bayesi statistika õpetamine võrdlevalt sagedusliku statistikaga ei ole parim lahendus, võrdleme lühidalt järgnevalt sageduslikku ja bayesi paradigmasid. Kes ei ole õppinud sageduslikku statistikat, võiksid selle osa vahele jätta.

Chapter 3

Tarkvaratööriistad

3.1 Installeeri vajalikud programmid

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

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

3.2 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".

3.3 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 3.1) 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 infolehekülgelt: Using Projects.

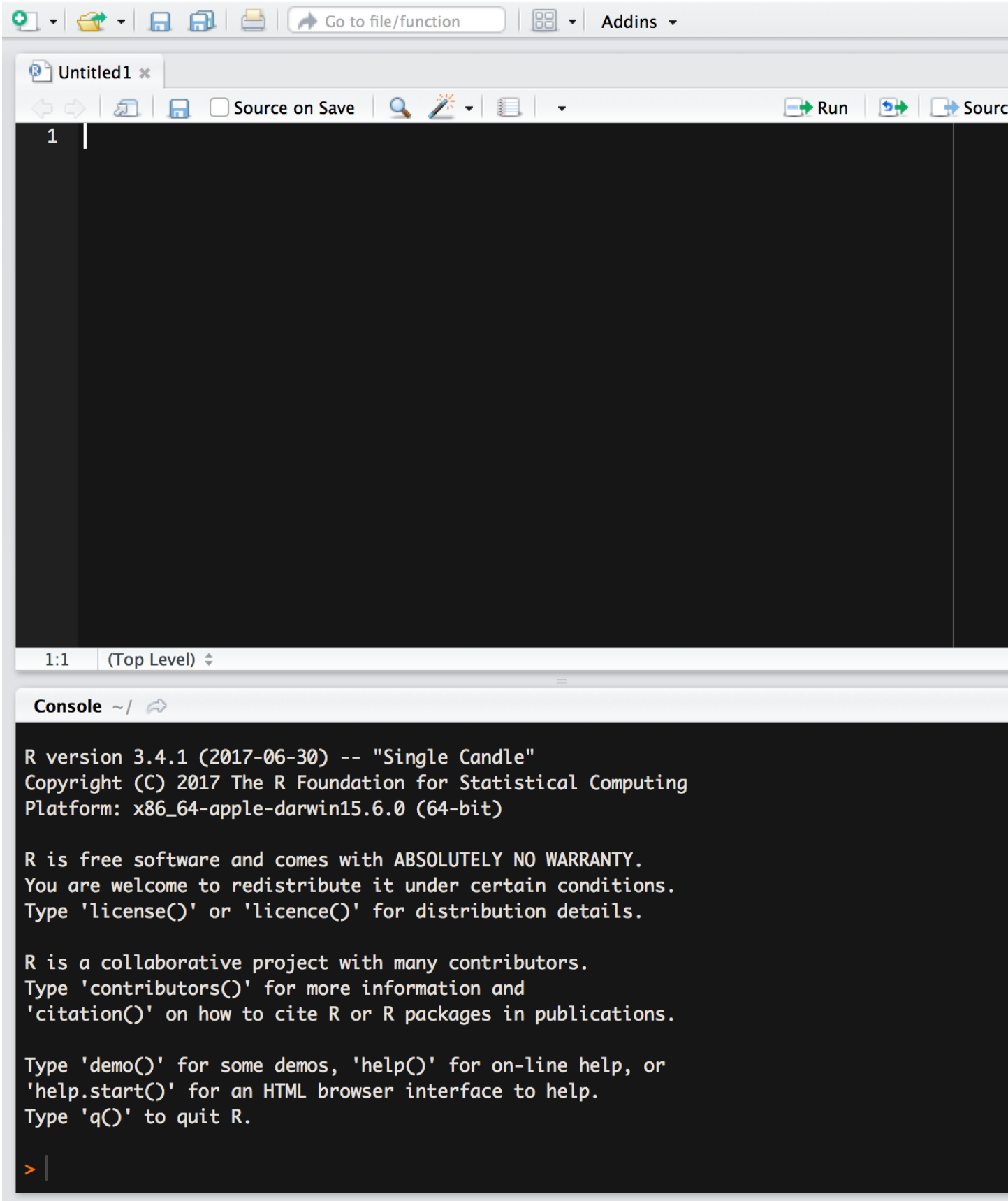


Figure 3.1: RStudio konsoolis on neli akent. Üleval vasakul on sinu poolt nimega varustatud koodi ja teksti editor kuhu kirjutad R skripti. Sinna kirjutad oma koodi ja kommentaarid sellele. All vasakul on konsool. Sinna sisestatakse käivitamisel sinu R kood ja sinna trükitakse väljund. Üleval paremal on Environment aken olulise sakiga . Seal on näha R-i objektid, mis on sulle töökeskkonnas kättesaadavad ja millega sa saad töötada.

3.4 Git *Merge* konfliktid

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

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

Githubi veateadete lugemine ja Google otsing aitavad sind.

3.5 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 andmesettide asukoht, vajalik tarkvara jne. **README.md** on markdown formaadis projekti info, sisaldab juhendeid kasutajatele. Igale GitHubi repole on soovitav koostada README.md, esialgu kasvõi projekti pealkiri ja üks kirjeldav lause. README.md ja DESCRIPTION asuvad projekti juurkataloogis.

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

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

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

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

lib/ kataloogis on kasutaja poolt tehtud funktsioonide definitsioone 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 pakkide 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).

Chapter 4

Pakettide installeerimine

R paketid sisaldavad ühte või enamat mingit kindlat operatsiooni läbi viivat funktsiooni. Selleks, et installeerida pakett, sisesta järgnev käsurida R konsooli:

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

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

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

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

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

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

Näiteks "tidyr" pakett sisaldab 41 funktsiooni:

```
library(tidyr)
ls("package:tidyr")
```

```
##  [1] "%>%"             "complete"        "complete_"
##  [4] "crossing"        "crossing_"       "drop_na"
##  [7] "drop_na_"        "expand"          "expand_"
## [10] "extract"         "extract_"        "extract_numeric"
## [13] "fill"            "fill_"           "full_seq"
## [16] "gather"          "gather_"         "nest"
## [19] "nest_"           "nesting"         "nesting_"
## [22] "population"      "replace_na"      "separate"
## [25] "separate_"       "separate_rows"   "separate_rows_"
## [28] "smiths"          "spread"          "spread_"
## [31] "table1"          "table2"          "table3"
## [34] "table4a"         "table4b"         "table5"
## [37] "unite"           "unite_"          "unnest"
## [40] "unnest_"         "who"
```

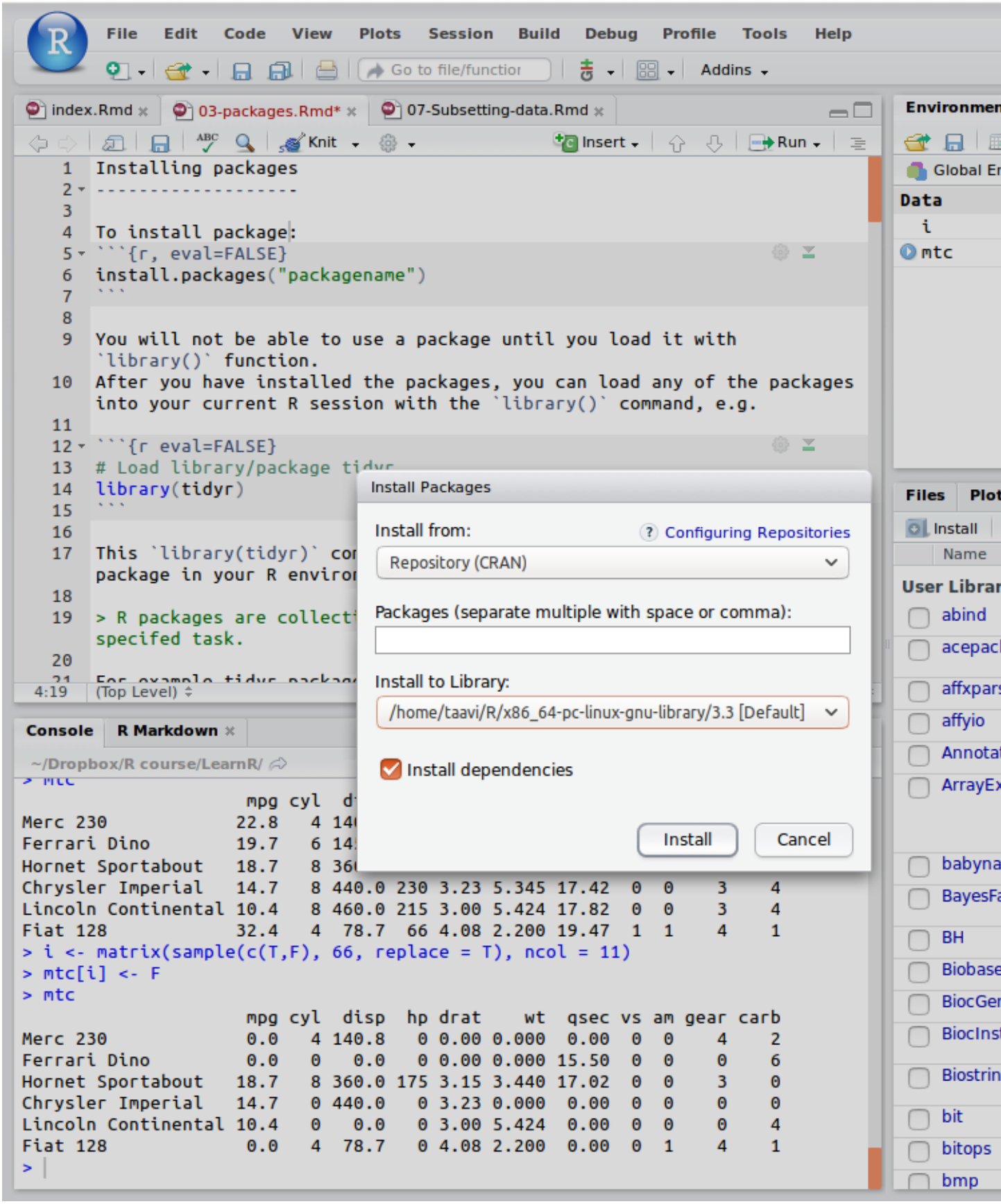


Figure 4.1: RStudio 'Install Packages' dialoogiaken.

4.1 R repositooriumid

R pakid on saadaval kolmest põhilisest repositooriumist:

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

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

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

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

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)
```

Chapter 5

R on kalkulaator

Selleks, et R aru saaks, et tegu on temale mõeldud käsu, aga mitte tavatekstiga, Rmarkdown failis, tuleb see R koodirida spetsiaalsesse **ümbrisesse või koodialasse** (i.k. *chunk*) pakendada. Uue koodiümbrise saad sisestada koodieditori servast "Insert" > "R" abil. Sisetatud R kood evalueeritakse siis, kui vajutad tekinud hallis koodialas rohelisele nupule või alternatiivselt, kui soovid jooksutada ainult osa koodi või üht koodirida, siis pane kursor soovitud reale või võta osa koodi blokki ja vajuta klaviatuuril cmd + enter.

Liidame `2 + 2`.

```
2 + 2
```

```
## [1] 4
```

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

Kontrollime seda:

```
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
3 ** 2
# Ruutjuur 3st
sqrt(3)
# Naturaallogaritm sajast
log(100)
```

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

```
# Ümarda pi neljale komakohale
round(pi, 4)
```

```
## [1] 3.1416
```

Ümardamine on oluline tulemuste väljaprintimisel.

5.1 R objektid

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

- Andmekogud (vektorid, tabelid, maatriksid, listid jm).
- Statistiliste analüüside väljundid.
- Funktsioonid, mille oleme ise sisse lugenud.

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

```
ls()
```

```
## [1] "answer"
```

`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".

5.1.1 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(babe)
```

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

class() annab meile objekti tüübi. Antud juhul *character vector*.

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

```
apples <- 2
bananas <- 3
apples + bananas
```

```
## [1] 5
```

objekt 5 ei ole nimetatud, seega ei ilmu see ka workspace

```
a <- 2
b <- 3
a <- a+b
str(a) #objekti nimega a struktuur
```

```
##  num 5
```

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

5.1.1.1 Nimede vorm

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

5.1.1.2 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.

Esimnene 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

soovitavad siiski ka sellisel juhul kasutada funktsiooni koos sulgudega! See on hea lühike ja inimloetav viis koodi kirjutada, mis on masina jaoks identne esimese koodiga.

```
## we need piping operator '%>%' from magrittr
library(magrittr)
a <- c(2, 3)
result <- a %>% sum() %>% sqrt() %>% round(2)
result
```

```
## [1] 2.24
```

Neljas võimalus: Kõrgharitud programmeerija kirjutaks selle koodi aga nii:

```
a <- c(2, 3)
a <- round(sqrt(sum(a)), 2)
a
```

```
## [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)

Mida vähem on kohti, kus saab koodi töötamist kontrollida, seda halvem teile. Sellepärast ärge kunagi pange üksteise otsa rohkem kui 4 torujuppi. See on teie otsustada, millist koodivormi te millal kasutate, aga te peaksite oskama lugeda neid kõiki.

5.2 Objektide tüübid

5.2.1 Vektor - andmerida

Vektor on rida kindlas järjekorras arve, tähemärke või TRUE/FALSE loogilisi väärtusi. Iga 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 headeri juures). Vektori loomiseks kasuta funktsiooni c() — combine

```
minu_vektor <- c(1, 3, 4)
str(minu_vektor)
```

```
##  num [1:3] 1 3 4
```

```
minu_vektor <- c(1, NA, 4)
minu_vektor
```

```
## [1]  1 NA  4
```

```
class(minu_vektor)
```

```
## [1] "numeric"
```

```
minu_vektor <- c(1, "A1", "4$", "joe")
minu_vektor
```

```
## [1] "1"   "A1"  "4$"  "joe"
```

```
class(minu_vektor)
```

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

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

Järgneva trikiga saab mitte-numbrilisest vektorist numbrilise vektori.

```
library(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    4   N
```

```
minu_vektor
```

```
## [1]  1  1  4 NA
```

```
str(minu_vektor)
```

```
##  num [1:4] 1 1 4 NA
```

```
sort(x, decreasing = FALSE, ...) #sorts vector in ascending order
unique(x) #returns a vector or data frame, but with duplicate elements/rows removed.
```

5.2.2 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"
```

5.2.3 tehted arvuliste vektoritega

Vektoreid saab liita, lahutada, korrutada ja jagada.

```
a <- c(1,2,3)
b <- 4 #ühe elemendiga vektor ei vaja c() enda ümber
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] 0.6666667
```

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.

5.2.4 List – andmekott

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] 0.17657487 0.28478609 0.07219848 0.67306296 0.98082175
##
## [[2]]
##            a          b
## 1 0.17657487  0.2104119
## 2 0.28478609  1.5843747
## 3 0.07219848 -1.8728558
## 4 0.67306296 -0.2245067
## 5 0.98082175 -0.3536892
##
## [[3]]
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Coefficients:
## (Intercept)           hp
##    30.09886     -0.06823
##
##
## [[4]]
## [1] "your grandma on bongos"
```

võtame listist välja elemndi “ab”:

```
happy_list$ab
```

```
## NULL
```

5.3 Tibble ja data frame - andmeraamid

```
library(tidyverse)
```

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: tibble ja data frame, mis on väga sarnased. Tibble on uuem, veidi kaunima väljatrükiga, pisut mugavam kasutada, ja me kasutame põhiliselt seda, välja arvatud hiljem Bayesi arvutustes, kus me tehnilistel põhjustel kasutame data frame. 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

```
shop <- c("maxima", "tesco", "lidl")
apples <- c(1, 4, 43)
oranges <- c(2, 32, NA)
fruits <- tibble(shop, apples, oranges)
fruits
```

```
## # A tibble: 3 x 3
##     shop apples oranges
```

```
##    <chr>  <dbl>   <dbl>
## 1 maxima      1       2
## 2  tesco      4      32
## 3   lidl     43      NA
```

Siin ta on, ilusti meie workspace-s.

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

```
count(fruits, apples)
```

```
## # A tibble: 3 x 2
##   apples     n
##    <dbl> <int>
## 1      1     1
## 2      4     1
## 3     43     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
```

```
names(fruits)
```

```
## [1] "shop"    "apples"  "oranges"
```

```
colnames(fruits)
```

```
## [1] "shop"    "apples"  "oranges"
```

```
nrow(fruits)
```

```
## [1] 3
```

```
ncol(fruits)
```

```
## [1] 3
```

```
arrange(fruits, desc(apples)) #sorteerib tabeli veeru "apples" väärtuste järgi langevalt (default on tõ
```

```
## # A tibble: 3 x 3
##     shop apples oranges
##    <chr>  <dbl>   <dbl>
## 1   lidl     43      NA
## 2  tesco      4      32
```

```
## 3 maxima      1       2
```

```
top_n(fruits, 2, apples) #saab 2 rida, milles on kõige rohkem õunu
```

```
## # A tibble: 2 x 3
##    shop apples oranges
##   <chr>  <dbl>   <dbl>
## 1 tesco      4      32
## 2  lidl     43      NA
```

```
top_n(fruits, -2, apples) #saab 2 rida, milles on kõige vähem õunu
```

```
## # A tibble: 2 x 3
##     shop apples oranges
##    <chr>  <dbl>   <dbl>
## 1 maxima      1       2
## 2  tesco      4      32
```

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]  0.7071068 19.7989899         NA
```

```
# 2 tähendab, et arvuta sd veeru kaupa
apply(fruits_subset, 2, sd)
```

```
##   apples  oranges
## 23.43075       NA
```

Lisame käsitsi meie tabelile 1 rea:

```
fruits <- add_row(fruits,
                  shop = "konsum",
                  apples = 132,
                  oranges = -5,
                  .before = 3)
fruits
```

```
## # A tibble: 4 x 3
##     shop apples oranges
```

```
##    <chr>  <dbl>   <dbl>
## 1 maxima      1       2
## 2  tesco      4      32
## 3 konsum    132      -5
## 4   lidl     43      NA
```

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:

5.3.0.1 rekodeerime tibble väärtusi

```
fruits$apples[fruits$apples==43] <- 333
fruits
```

```
## # A tibble: 4 x 3
##     shop apples oranges
##    <chr>  <dbl>   <dbl>
## 1 maxima      1       2
## 2  tesco      4      32
## 3 konsum    132      -5
## 4   lidl    333      NA
```

```
fruits$shop[fruits$shop=="tesco"] <- "TESCO"
fruits
```

```
## # A tibble: 4 x 3
##     shop apples oranges
##    <chr>  <dbl>   <dbl>
## 1 maxima      1       2
## 2  TESCO      4      32
## 3 konsum    132      -5
## 4   lidl    333      NA
```

```
fruits$apples[fruits$apples>100] <- NA
fruits
```

```
## # A tibble: 4 x 3
##     shop apples oranges
##    <chr>  <dbl>   <dbl>
## 1 maxima      1       2
## 2  TESCO      4      32
## 3 konsum     NA      -5
## 4   lidl     NA      NA
```

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 (väga halb mõte):

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

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

5.3.1 Ühendame kaks tibblet rea kaupa

Tabeli veergude arv ei muutu, ridade arv kasvab.

```
df1 <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
df1.1 <- tibble(colA = "d", colB =  4)
#id teeb veel ühe veeru, mis näitab, kummast algtabelist iga uue tabeli rida pärit on
bind_rows(df1, df1.1, .id = "id")
```

```
## # A tibble: 4 x 3
##      id  colA  colB
##   <chr> <chr> <dbl>
## 1     1     a     1
## 2     1     b     2
## 3     1     c     3
## 4     2     d     4
```

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

Kui `bind_rows()` miskipärast ei tööta, proovi `rbind()` funktsiooni, mis on väga sarnane (?rbind). 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)
bind_rows(df1, df2) #works by guessing your true intention
```

```
## # A tibble: 4 x 4
##    colA  colB  ColC  ColD
##   <chr> <dbl> <chr> <dbl>
## 1     a     1  <NA>    NA
## 2     b     2  <NA>    NA
## 3     c     3  <NA>    NA
## 4  <NA>    NA     d     4
```

5.3.1.1 ühendame kaks tibblet 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))
cbind(df1, dfx)
```

```
##   colA colB colC
## 1    a    1    4
## 2    b    2    5
## 3    c    3    6
```

5.3.1.2 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
```

```
str(ubinad) #see on jälle vektor
```

```
##  num [1:4] 3 6 NA NA
```

5.3.2 Andmeraamide salvestamine (eksport-import)

Andmeraami saame salvestada näiteks csv-na (comma separated file) oma kõvakettale

```
write.csv(fruits, "data/fruits.csv")
```

Kuhu see fail läks? See läks meie projekti juurkataloogi kausta "data", mille leiame käsuga:

```
getwd()
```

```
## [1] "/Users/taavi/Dropbox/2017-R-course/lectures"
```

Andmete sisselugemine töökataloogist:

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

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

R-i saab sisse lugeda palju erinevaid andmeformaate, kaasa arvatud Exceli oma. installi: Gotta read em all R. See läheb ülesse tab-i Addins. Sealt saab selle avada ja selle abil tabeleid oma workspace üles laadida.

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

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

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

> Tüüpiliselt töötate R-s oma algse andmestikuga. Reprodutseeruvaks projektiks on vaja 2 asja: algandmeid ja koodi, millega neid manipuleerida.

NB! R ei muuda algandmeid, mille te näiteks csv-na sisse loete - need jäävad alati neitsilikeks.

Seega ei ole andmetabelite salvestamine töö vaheproduktidena sageli vajalik sest te jooksutate 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 palju aega võtab.

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)
```

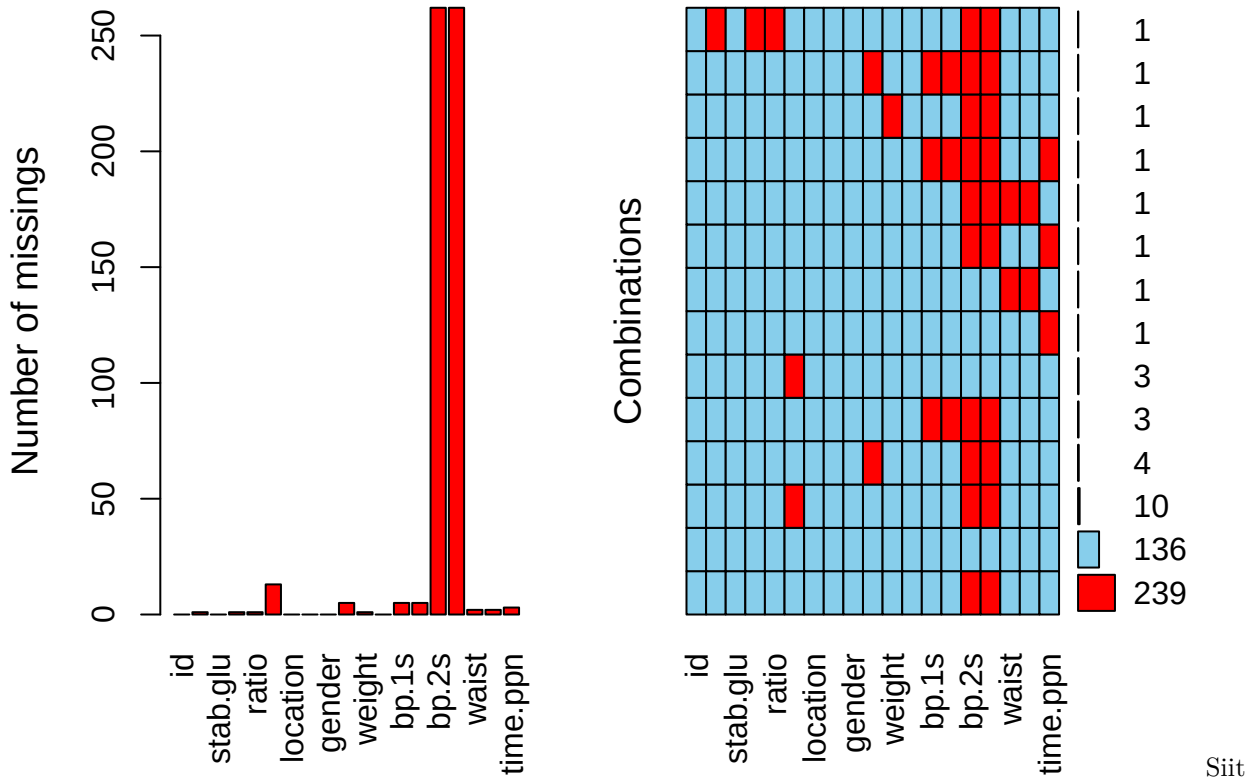
```
## [1] "tbl_df"     "tbl"        "data.frame"
```

5.4 tabelit sisse lugedes vaata üle NA-d

```
library(VIM)
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=T)
```



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

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

```
nrows <- nrow(diabetes)
  ncomplete <- sum(complete.cases(diabetes))
  ncomplete #136
```

```
## [1] 136
```

```
  ncomplete/nrows #34%
```

```
## [1] 0.337469
```

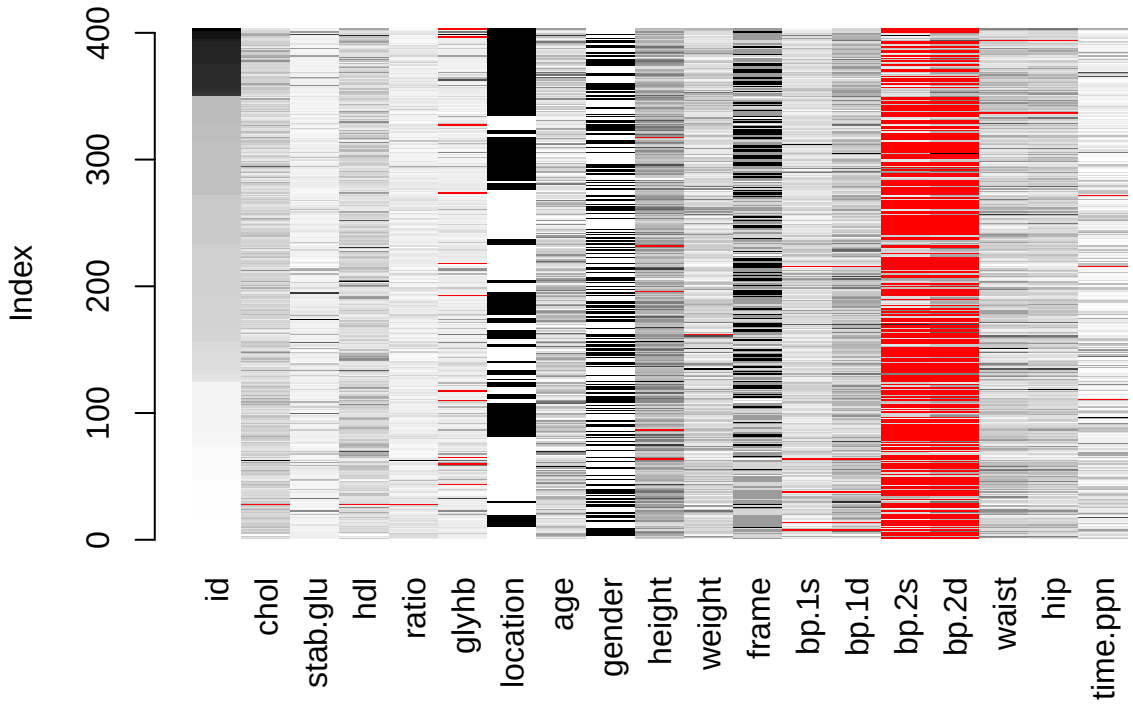
Mitu NA-d igas tulbas?

```
 sapply(diabetes, function(x) sum(is.na(x)))
```

```
##       id     chol stab.glu      hdl    ratio    glyhb location      age
##        0        1        0        1        1       13        0        0
##   gender   height   weight    frame    bp.1s    bp.1d    bp.2s    bp.2d
##        0        5        1        0        5        5      262      262
##    waist      hip time.ppn
##        2        2        3
```

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

```
VIM::matrixplot(diabetes)
```



kuidas rekodeerida NA-d näiteks 0-ks

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

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

5.4.1 4. Matrix - numbriraam

Maatriks koosneb ühepikkustest vektoritest, mis sisaldavad ainult numbreid. Enamasti me ei kasuta maatrikseid, vaid andmeraame. Tip: me saame sageli andmeraami maatriksina kasutada kui me viskame sealt välja mitte-numbrilised tulbad.

Aga saame ka andmeraame konverteerida otse maatriksiks (ja tagasi). Vahest läheb seda vaja, eriti bioconductori funktsioonidega.

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

5.5 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.factor(), as.numeric(), as.character(), as.integer(), as.logical()

5.6 Indekseerimine

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

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

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

5.6.1 Vektorid ja nende indeksid on ühe-dimensionaalsed

```
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
```

5.6.2 andmeraamid ja maatriksid on kahe-dimensionaalsed, nagu ka nende indeksid

2D indeksi kuju on [rea_indeks, veeru_indeks].

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

```
# kõik read peale 1.
dat[-1, ]
# viskab välja 2. veeru
dat[, -2]
# 2 andmepunkti: 2. rida, 1. ja 2. veerg
dat[2, 1:2]
# 2 andmepunkti: 2. rida, 3. ja 4. veerg
dat[2, c(1, 2)]
#viskab välja 1. ja 2. rea
dat[-c(1, 2), ]
#veerg nimega colB, output on erandina vektor!
dat$colB
```

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

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

```
dat <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
dat$colB[dat$colA != "a" ] #jätab sisse kõik vektori colB väärtused, kus samas tabeli reas olev colA vä
```

```
## [1] 2 3
```

```
dat$colA[dat$colB > 1] #jätab sisse kõik vektori colA väärtused, kus samas tabeli reas olev colB väärtu
```

```
## [1] "b" "c"
```

5.6.3 litside indeksid on kolme-dimensionaalsed

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. (Vorm list_name$object_name töötab samamoodi kui kaksiksulud.) 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 df 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]] #class is df  --- we extracted a df from the list
my_tibble
```

```
## # A tibble: 2 x 2
##    colA  colB
##   <chr> <dbl>
## 1     A     1
## 2     B     2
```

```
#my_list$a #sama asi: $ does the same thing as [[ ]]
```

See ei ole enam list

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

```
aa <- my_list[1]
str(aa)
```

```
## List of 1
##  $ a:Classes 'tbl_df', 'tbl' and 'data.frame':   2 obs. of  2 variables:
##   ..$ colA: chr [1:2] "A" "B"
##   ..$ colB: num [1:2] 1 2
```

```
aa1 <- my_list$a[2,] #class is df
aa1
```

```
## # A tibble: 1 x 2
##    colA  colB
##   <chr> <dbl>
## 1     B     2
```

```
aa3 <- my_list[[1]][1,]
aa3
```

```
## # A tibble: 1 x 2
##    colA  colB
##   <chr> <dbl>
## 1     A     1
```

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

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

5.7 Regular expression ja find & replace

Regular expression annab võimaluse lühidalt kirjeldada mitte-üheseid otsinguparameetreid.

> regular expression on string, mis kirjeldab mitut stringi

A regular expression Regular Expressions as used in R

- Most characters, including all letters and digits, are regular expressions that match themselves.
- `.` matches any single character.
- You can refer also to a character class, which is a list of characters enclosed between `[` and `]`, e.g. `[[:alnum:]]` is same as `[A-z0-9]`.
- Most common character classes:
  - `[:alnum:]` includes alphanumerics (`[:alpha:]` and `[:digit:]`);
  - `[:alpha:]`, includes alphabetic characters (`[:upper:]` and `[:lower:]` case);

  - `[:punct:]` includes punctuation characters ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ ] ^ _ ` { | } ~.;
  - `[:blank:]` includes space and tab; etc.
- The metacharacters in regular expressions are `. \ | ( ) [ { ^ $ * + ?`, whether these have a special meaning depends on the context. When matching a metacharacter as a regular character, precede it with a double backslash `\\`.
- Repetition quantifiers put after regex specify how many times regex is matched: `?`, optional, at most once; `*`, 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.

5.7.0.1 Common operations with regular expressions

- Locate a pattern match (positions)
- Extract a matched pattern
- Identify a match to a pattern
- Replace a matched pattern

5.7.0.2 Find and replace

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

#replaces all a-s at the beginning of strings with e-s
str_replace(x, "^a", "e")
```

```
## [1] "epple"  "enanas" "banana"
```

```
# str_replace only replaces at the first occurence at each string
str_replace(x, "a", "e")
```

```
## [1] "epple"  "enanas" "benana"
```

```
#str_replace_all replaces all a-s anywhere in the strings
str_replace_all(x, "a", "e")
```

```
## [1] "epple"  "enenes" "benene"
```

```
#replaces a and the following character at the end of string with nothing (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] "epple"  "enenee" "benene"
```

```
#remove all numbers.
y<-c("as1", "2we3w", "3e")
str_replace_all(y, "\\d", "")
```

```
## [1] "as"  "wew" "e"
```

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

```
## [1] "1"  "23" "3"
```

```
x<- c("apple", "apple pie")
str_replace_all(x, "^apple$","m") #To force to only match a complete string:
```

```
## [1] "m"         "apple pie"
```

```
str_replace_all(x, "\\s","_") #space to _
```

```
## [1] "apple"     "apple_pie"
```

```
str_replace_all(x, "[apl]","_") #a or p or l to _
```

```
## [1] "____e"     "____e _ie"
```

```
str_replace_all(x, "[ap|p.e]","_") # ap or p.e to _
```

```
## [1] "___l_"     "___l_ _i_"
```

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

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

5.8 Tidyverse

Tidyverse on väike 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. Tidyversiga sobib hästi kokku ka ggplot2 graafikasüsteem.

5.8.1 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 weigth
##     <chr>   <chr> <chr> <dbl>  <dbl>  <dbl>
## 1       1     exp     F     3    168     88
## 2       2 placebo     M     3    176     91
```

Nii näeb välja tidy tibble. Kõik analüüsil vajalikud parameetrid tuleks siia tabelisse veeru kaupa sisse tuua. Näiteks, kui mõõtmised on sooritatud erinevates keskustes erinevate inimeste poolt kasutades sama ravimi erinevaid preparaate, oleks hea siia veel 3 veergu lisada (center, experimenter, batch).

5.8.2 Tabeli dimensioonide muutmine (pikk ja lai formaat)

Väga oluline osa tidyverses töötamisest on tabelite pika ja laia formaadi vahel viimine.

See on laias formaadis tabel df, mis ei ole tidy

```
## # A tibble: 3 x 5
##   subject   sex control experiment_1 experiment_2
##     <chr> <chr>   <dbl>        <dbl>        <dbl>
## 1     Tim     M      23           34           40
## 2     Ann     F      31           38           42
## 3    Jill     F      30           36           44
```

Kõigepealt pikka formaati. key ja value argumendid on ainult uute veergude nimetamiseks, oluline on 3:ncol(dat) argument, mis ütleb, et "kogu kokku veerud alates 3. veerust". Alternatiivne viis seda öelda: c(-subject, -sex).

```
dat_lng <- gather(dat, key = experiment, value = value, 3:ncol(dat))
# df_l3<-df %>% gather(experiment, value, 3:ncol(df)) works as well.
#df_l4<-df %>% gather(experiment, value, c(-subject, -sex)) works as well
dat_lng
```

```
## # A tibble: 9 x 4
##   subject   sex   experiment value
##     <chr> <chr>        <chr> <dbl>
## 1     Tim     M      control    23
## 2     Ann     F      control    31
## 3    Jill     F      control    30
## 4     Tim     M experiment_1    34
## 5     Ann     F experiment_1    38
## 6    Jill     F experiment_1    36
## 7     Tim     M experiment_2    40
## 8     Ann     F experiment_2    42
## 9    Jill     F experiment_2    44
```

Paneme selle tagasi algsesse laia formaati: ?spread

```
spread(dat_lng, key = experiment, value = value)
```

```
## # A tibble: 3 x 5
##   subject   sex control experiment_1 experiment_2
## *   <chr> <chr>   <dbl>        <dbl>        <dbl>
## 1     Ann     F      31           38           42
## 2    Jill     F      30           36           44
## 3     Tim     M      23           34           40
```

key viitab pika tabeli veerule, mille väärtustest tulevad laias tabelis uute veergude nimed. value viitab pika tabeli veerule, kust võetakse arvud, mis uues laias tabelis uute veergude vahel laiali jagatakse.

5.8.2.1 Tibble transpose — read veergudeks ja vastupidi

```
dat <- tibble(a = c("tim", "tom", "jill"), b1 = c(1, 2, 3), b2 = c(4, 5, 6))
dat
```

```
## # A tibble: 3 x 3
##       a    b1    b2
##   <chr> <dbl> <dbl>
## 1   tim     1     4
## 2   tom     2     5
## 3  jill     3     6
```

Me kasutame selleks maatriksarvutuse funktsiooni t() — transpose. See võtab sisse ainult numbrilisi veerge, seega anname talle ette df miinus 1. veerg, mille sisu me konverteerime uue tablei veerunimedeks.

```
dat1 <- t(dat[,-1])
colnames(dat1) <- dat$a
dat1
```

```
##    tim tom jill
## b1   1   2    3
## b2   4   5    6
```

5.9 dplyr-i 5 verbi

Need tuleb teil omale pähe ajada sest nende 5 verbiga (pluss gather ja spread) saab lihtsalt teha 90% andmevääänamisest, mida teil elus vaja läheb. NB! Check the data wrangling cheatsheet and dplyr help for further details. dplyr laetakse koos tidyverse-ga automaatselt teie workspace.

5.9.1 `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 )
```

```
##           liik Sepal.Length Sepal.Width
## 1       setosa          5.1         3.5
## 2       setosa          4.9         3.0
## 3       setosa          4.7         3.2
## 4       setosa          4.6         3.1
## 5       setosa          5.0         3.6
## 6       setosa          5.4         3.9
## 7       setosa          4.6         3.4
## 8       setosa          5.0         3.4
## 9       setosa          4.4         2.9
## 10      setosa          4.9         3.1
## 11      setosa          5.4         3.7
## 12      setosa          4.8         3.4
## 13      setosa          4.8         3.0
## 14      setosa          4.3         3.0
## 15      setosa          5.8         4.0
## 16      setosa          5.7         4.4
## 17      setosa          5.4         3.9
## 18      setosa          5.1         3.5
## 19      setosa          5.7         3.8
## 20      setosa          5.1         3.8
## 21      setosa          5.4         3.4
## 22      setosa          5.1         3.7
## 23      setosa          4.6         3.6
## 24      setosa          5.1         3.3
## 25      setosa          4.8         3.4
## 26      setosa          5.0         3.0
## 27      setosa          5.0         3.4
## 28      setosa          5.2         3.5
## 29      setosa          5.2         3.4
## 30      setosa          4.7         3.2
## 31      setosa          4.8         3.1
## 32      setosa          5.4         3.4
## 33      setosa          5.2         4.1
## 34      setosa          5.5         4.2
## 35      setosa          4.9         3.1
## 36      setosa          5.0         3.2
## 37      setosa          5.5         3.5
## 38      setosa          4.9         3.6
## 39      setosa          4.4         3.0
## 40      setosa          5.1         3.4
## 41      setosa          5.0         3.5
## 42      setosa          4.5         2.3
## 43      setosa          4.4         3.2
## 44      setosa          5.0         3.5
## 45      setosa          5.1         3.8
## 46      setosa          4.8         3.0
## 47      setosa          5.1         3.8
## 48      setosa          4.6         3.2
## 49      setosa          5.3         3.7
## 50      setosa          5.0         3.3
## 51  versicolor          7.0         3.2
```

```
## 52  versicolor          6.4         3.2
## 53  versicolor          6.9         3.1
## 54  versicolor          5.5         2.3
## 55  versicolor          6.5         2.8
## 56  versicolor          5.7         2.8
## 57  versicolor          6.3         3.3
## 58  versicolor          4.9         2.4
## 59  versicolor          6.6         2.9
## 60  versicolor          5.2         2.7
## 61  versicolor          5.0         2.0
## 62  versicolor          5.9         3.0
## 63  versicolor          6.0         2.2
## 64  versicolor          6.1         2.9
## 65  versicolor          5.6         2.9
## 66  versicolor          6.7         3.1
## 67  versicolor          5.6         3.0
## 68  versicolor          5.8         2.7
## 69  versicolor          6.2         2.2
## 70  versicolor          5.6         2.5
## 71  versicolor          5.9         3.2
## 72  versicolor          6.1         2.8
## 73  versicolor          6.3         2.5
## 74  versicolor          6.1         2.8
## 75  versicolor          6.4         2.9
## 76  versicolor          6.6         3.0
## 77  versicolor          6.8         2.8
## 78  versicolor          6.7         3.0
## 79  versicolor          6.0         2.9
## 80  versicolor          5.7         2.6
## 81  versicolor          5.5         2.4
## 82  versicolor          5.5         2.4
## 83  versicolor          5.8         2.7
## 84  versicolor          6.0         2.7
## 85  versicolor          5.4         3.0
## 86  versicolor          6.0         3.4
## 87  versicolor          6.7         3.1
## 88  versicolor          6.3         2.3
## 89  versicolor          5.6         3.0
## 90  versicolor          5.5         2.5
## 91  versicolor          5.5         2.6
## 92  versicolor          6.1         3.0
## 93  versicolor          5.8         2.6
## 94  versicolor          5.0         2.3
## 95  versicolor          5.6         2.7
## 96  versicolor          5.7         3.0
## 97  versicolor          5.7         2.9
## 98  versicolor          6.2         2.9
## 99  versicolor          5.1         2.5
## 100 versicolor          5.7         2.8
## 101  virginica          6.3         3.3
## 102  virginica          5.8         2.7
## 103  virginica          7.1         3.0
## 104  virginica          6.3         2.9
## 105  virginica          6.5         3.0
```

```
## 106  virginica          7.6         3.0
## 107  virginica          4.9         2.5
## 108  virginica          7.3         2.9
## 109  virginica          6.7         2.5
## 110  virginica          7.2         3.6
## 111  virginica          6.5         3.2
## 112  virginica          6.4         2.7
## 113  virginica          6.8         3.0
## 114  virginica          5.7         2.5
## 115  virginica          5.8         2.8
## 116  virginica          6.4         3.2
## 117  virginica          6.5         3.0
## 118  virginica          7.7         3.8
## 119  virginica          7.7         2.6
## 120  virginica          6.0         2.2
## 121  virginica          6.9         3.2
## 122  virginica          5.6         2.8
## 123  virginica          7.7         2.8
## 124  virginica          6.3         2.7
## 125  virginica          6.7         3.3
## 126  virginica          7.2         3.2
## 127  virginica          6.2         2.8
## 128  virginica          6.1         3.0
## 129  virginica          6.4         2.8
## 130  virginica          7.2         3.0
## 131  virginica          7.4         2.8
## 132  virginica          7.9         3.8
## 133  virginica          6.4         2.8
## 134  virginica          6.3         2.8
## 135  virginica          6.1         2.6
## 136  virginica          7.7         3.0
## 137  virginica          6.3         3.4
## 138  virginica          6.4         3.1
## 139  virginica          6.0         3.0
## 140  virginica          6.9         3.1
## 141  virginica          6.7         3.1
## 142  virginica          6.9         3.1
## 143  virginica          5.8         2.7
## 144  virginica          6.8         3.2
## 145  virginica          6.7         3.3
## 146  virginica          6.7         3.0
## 147  virginica          6.3         2.5
## 148  virginica          6.5         3.0
## 149  virginica          6.2         3.4
## 150  virginica          5.9         3.0
```

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

```
select(iris, -Sepal.Length, -Sepal.Width, liik = Species)
```

helper functions you can use within select():

starts_with("abc"): matches names that begin with "abc."

ends_with("xyz"): matches names that end with "xyz."

contains("ijk"): matches names that contain "ijk."

matches(“(.)\1”): selects variables that match a regular expression. This one matches any variables that contain repeated characters.

num_range(“x”, 1:3) matches x1, x2 and x3.

```
iris <- as_tibble(iris)
select(iris, starts_with("Petal"))
```

```
## # A tibble: 150 x 2
##    Petal.Length Petal.Width
##           <dbl>       <dbl>
##  1          1.4         0.2
##  2          1.4         0.2
##  3          1.3         0.2
##  4          1.5         0.2
##  5          1.4         0.2
##  6          1.7         0.4
##  7          1.4         0.3
##  8          1.5         0.2
##  9          1.4         0.2
## 10          1.5         0.1
## # ... with 140 more rows
```

```
select(iris, ends_with("Width"))
```

```
## # A tibble: 150 x 2
##    Sepal.Width Petal.Width
##          <dbl>       <dbl>
##  1         3.5         0.2
##  2         3.0         0.2
##  3         3.2         0.2
##  4         3.1         0.2
##  5         3.6         0.2
##  6         3.9         0.4
##  7         3.4         0.3
##  8         3.4         0.2
##  9         2.9         0.2
## 10         3.1         0.1
## # ... with 140 more rows
```

```
# Move Species variable to the front
select(iris, Species, everything())
```

```
## # A tibble: 150 x 5
##    Species Sepal.Length Sepal.Width Petal.Length Petal.Width
##     <fctr>        <dbl>       <dbl>        <dbl>       <dbl>
##  1  setosa          5.1         3.5          1.4         0.2
##  2  setosa          4.9         3.0          1.4         0.2
##  3  setosa          4.7         3.2          1.3         0.2
##  4  setosa          4.6         3.1          1.5         0.2
##  5  setosa          5.0         3.6          1.4         0.2
##  6  setosa          5.4         3.9          1.7         0.4
##  7  setosa          4.6         3.4          1.4         0.3
##  8  setosa          5.0         3.4          1.5         0.2
##  9  setosa          4.4         2.9          1.4         0.2
## 10  setosa          4.9         3.1          1.5         0.1
## # ... with 140 more rows
```

```
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 0.674821453 0.45199261 0.9650919 0.0777362 0.95251413
##  2 0.088036268 0.08661763 0.5294282 0.4636746 0.31293832
##  3 0.987478895 0.19241447 0.2949625 0.6899234 0.38318698
##  4 0.958941816 0.17892002 0.3662149 0.5721688 0.04104426
##  5 0.004289472 0.55541656 0.3095148 0.5544049 0.88296201
##  6 0.001451434 0.89883547 0.1283954 0.4671343 0.40237466
##  7 0.173277477 0.44300419 0.2945075 0.8626319 0.63011042
##  8 0.180363020 0.86323733 0.2672147 0.9557293 0.67398894
##  9 0.437728923 0.70237106 0.8258246 0.4797665 0.04677373
## 10 0.727065481 0.96235164 0.4907869 0.1739951 0.14416891
```

```
select(dat, num_range("V", 9:6))
```

```
## # A tibble: 10 x 4
##             V9         V8         V7         V6
##          <dbl>      <dbl>      <dbl>      <dbl>
##  1 0.674821453 0.45199261 0.22293896 0.95251413
##  2 0.088036268 0.08661763 0.99141345 0.31293832
##  3 0.987478895 0.19241447 0.40687293 0.38318698
##  4 0.958941816 0.17892002 0.57026418 0.04104426
##  5 0.004289472 0.55541656 0.43283991 0.88296201
##  6 0.001451434 0.89883547 0.08970085 0.40237466
##  7 0.173277477 0.44300419 0.65837326 0.63011042
##  8 0.180363020 0.86323733 0.91581596 0.67398894
##  9 0.437728923 0.70237106 0.72657197 0.04677373
## 10 0.727065481 0.96235164 0.06967266 0.14416891
```

```
# Drop variables with -
select(iris, -starts_with("Petal"))
```

```
## # A tibble: 150 x 3
##    Sepal.Length Sepal.Width Species
##           <dbl>       <dbl>  <fctr>
##  1          5.1         3.5  setosa
##  2          4.9         3.0  setosa
##  3          4.7         3.2  setosa
##  4          4.6         3.1  setosa
##  5          5.0         3.6  setosa
##  6          5.4         3.9  setosa
##  7          4.6         3.4  setosa
##  8          5.0         3.4  setosa
##  9          4.4         2.9  setosa
## 10          4.9         3.1  setosa
## # ... with 140 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 Species
##           <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
##  1          5.1         3.5          1.4         0.2  setosa
##  2          4.9         3.0          1.4         0.2  setosa
##  3          4.7         3.2          1.3         0.2  setosa
##  4          4.6         3.1          1.5         0.2  setosa
##  5          5.0         3.6          1.4         0.2  setosa
##  6          5.4         3.9          1.7         0.4  setosa
##  7          4.6         3.4          1.4         0.3  setosa
##  8          5.0         3.4          1.5         0.2  setosa
##  9          4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## # ... with 140 more rows
```

See ?select for more details.

5.9.2 `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 Species
##          <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
## 1          4.4         2.9          1.4         0.2  setosa
## 2          4.3         3.0          1.1         0.1  setosa
## 3          4.4         3.0          1.3         0.2  setosa
## 4          4.4         3.2          1.3         0.2  setosa
```

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 Species
##           <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
##  1          5.1         3.5          1.4         0.2  setosa
##  2          4.9         3.0          1.4         0.2  setosa
##  3          4.7         3.2          1.3         0.2  setosa
##  4          4.6         3.1          1.5         0.2  setosa
##  5          5.0         3.6          1.4         0.2  setosa
##  6          5.4         3.9          1.7         0.4  setosa
##  7          4.6         3.4          1.4         0.3  setosa
##  8          5.0         3.4          1.5         0.2  setosa
##  9          4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## # ... with 40 more rows
```

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    Species
##           <dbl>       <dbl>        <dbl>       <dbl>     <fctr>
```

```
##  1          4.4         2.9          1.4         0.2     setosa
##  2          4.3         3.0          1.1         0.1     setosa
##  3          4.4         3.0          1.3         0.2     setosa
##  4          4.4         3.2          1.3         0.2     setosa
##  5          7.0         3.2          4.7         1.4 versicolor
##  6          6.4         3.2          4.5         1.5 versicolor
##  7          6.9         3.1          4.9         1.5 versicolor
##  8          5.5         2.3          4.0         1.3 versicolor
##  9          6.5         2.8          4.6         1.5 versicolor
## 10          5.7         2.8          4.5         1.3 versicolor
## # ... with 94 more rows
```

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    Species
##           <dbl>       <dbl>        <dbl>       <dbl>     <fctr>
##  1          7.0         3.2          4.7         1.4 versicolor
##  2          6.4         3.2          4.5         1.5 versicolor
##  3          6.9         3.1          4.9         1.5 versicolor
##  4          5.5         2.3          4.0         1.3 versicolor
##  5          6.5         2.8          4.6         1.5 versicolor
##  6          5.7         2.8          4.5         1.3 versicolor
##  7          6.3         3.3          4.7         1.6 versicolor
##  8          4.9         2.4          3.3         1.0 versicolor
##  9          6.6         2.9          4.6         1.3 versicolor
## 10          5.2         2.7          3.9         1.4 versicolor
## # ... with 90 more rows
```

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))
```

5.9.3 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 5.843333 0.8280661   150         3
```

`n()` loeb üles, mitu väärtust läks selle summary statistic-u arvutusse,

`n_distinct()` loeb üles, mitu unikaalset väärtust läks samasse arvutusse.

summarise on kasulikum, kui teda kasutada koos järgmise verbi, group_by-ga.

5.9.4 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
##       <fctr> <dbl>     <dbl> <int>     <int>
## 1     setosa 5.006 0.3524897    50         1
## 2 versicolor 5.936 0.5161711    50         1
## 3  virginica 6.588 0.6358796    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
## 2     Ann     F      control    31
## 3    Jill     F      control    30
## 4     Tim     M experiment_1    34
## 5     Ann     F experiment_1    38
## 6    Jill     F experiment_1    36
## 7     Tim     M experiment_2    40
## 8     Ann     F experiment_2    42
## 9    Jill     F experiment_2    44
```

```
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 0.7071068     2     1
## 2     F experiment_1  37.0 1.4142136     2     1
## 3     F experiment_2  43.0 1.4142136     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 0.7071068     2     1
## 2      control     M  23.0        NA     1     1
## 3 experiment_1     F  37.0 1.4142136     2     1
## 4 experiment_1     M  34.0        NA     1     1
## 5 experiment_2     F  43.0 1.4142136     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 Species  MEAN
##           <dbl>       <dbl>        <dbl>       <dbl>  <fctr> <dbl>
##  1          5.1         3.5          1.4         0.2  setosa 5.006
##  2          4.9         3.0          1.4         0.2  setosa 5.006
##  3          4.7         3.2          1.3         0.2  setosa 5.006
##  4          4.6         3.1          1.5         0.2  setosa 5.006
##  5          5.0         3.6          1.4         0.2  setosa 5.006
##  6          5.4         3.9          1.7         0.4  setosa 5.006
##  7          4.6         3.4          1.4         0.3  setosa 5.006
##  8          5.0         3.4          1.5         0.2  setosa 5.006
##  9          4.4         2.9          1.4         0.2  setosa 5.006
## 10          4.9         3.1          1.5         0.1  setosa 5.006
## # ... with 140 more rows, and 1 more variables: 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)
```

5.9.5 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 value column called log_value by `df %>% mutate(log_value = log2(value))`. Or you can create a new column where a constant is substracted from the value column: `df %>% mutate(centered_value = value - mean(value) )`. Here the mean value is substracted 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  3.135494
## 2     Ann     F      control    31  3.433987
## 3    Jill     F      control    30  3.401197
## 4     Tim     M experiment_1    34  3.526361
## 5     Ann     F experiment_1    38  3.637586
## 6    Jill     F experiment_1    36  3.583519
## 7     Tim     M experiment_2    40  3.688879
## 8     Ann     F experiment_2    42  3.737670
## 9    Jill     F experiment_2    44  3.784190
```

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  3.135494
## 2     Ann      F  3.433987
## 3    Jill      F  3.401197
## 4     Tim      M  3.526361
## 5     Ann      F  3.637586
## 6    Jill      F  3.583519
## 7     Tim      M  3.688879
## 8     Ann      F  3.737670
## 9    Jill      F  3.784190
```

```
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  0.7113402 2.667694
## 2     Ann     F      control    31  0.8416290 5.465862
## 3    Jill     F      control    30  0.8144796 5.289544
## 4     Tim     M experiment_1    34  1.0515464 3.943548
## 5     Ann     F experiment_1    38  1.0316742 6.700089
## 6    Jill     F experiment_1    36  0.9773756 6.347453
## 7     Tim     M experiment_2    40  1.2371134 4.639468
## 8     Ann     F experiment_2    42  1.1402715 7.405361
## 9    Jill     F experiment_2    44  1.1945701 7.757998
```

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  0.6509434 3.477273
## 2     Ann     F      control    31  0.8773585 4.686759
## 3    Jill     F      control    30  0.8490566 4.535574
## 4     Tim     M experiment_1    34  0.9622642 5.140317
## 5     Ann     F experiment_1    38  1.0754717 5.745060
## 6    Jill     F experiment_1    36  1.0188679 5.442688
## 7     Tim     M experiment_2    40  1.1320755 6.047432
## 8     Ann     F experiment_2    42  1.1886792 6.349803
## 9    Jill     F experiment_2    44  1.2452830 6.652175
```

5.9.6 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
## 2    t1     4
## 3   t12     9
```

```
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
## 2     1     4
## 3    12     9
```

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
## 2    t1     4
## 3   t12     9
```

```
mutate_at(bad.df, "time", parse_number)
```

```
## # A tibble: 3 x 2
##    time value
##   <dbl> <dbl>
## 1     0     2
## 2     1     4
## 3    12     9
```

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

5.9.7 `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 Procaryota; Bacteria; Alpha-Proteobacteria; Escharichia
## 2     2                                   Eukaryota; Chordata
```

```
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep = '; ', extra = "merge", fill
```

```
## # A tibble: 2 x 5
##   index       riik      hmk               klass    perekond
## * <dbl>      <chr>    <chr>               <chr>       <chr>
## 1     1 Procaryota Bacteria Alpha-Proteobacteria Escharichia
```

```
## 2     2  Eukaryota Chordata                  <NA>        <NA>
```

```
# some special cases:
(dat <- tibble(index = c(1, 2),
               taxon = c("Prokaryota || Bacteria || Alpha-Proteobacteria || Escharichia", "Eukaryota ||
```

```
## # A tibble: 2 x 2
##   index                                                    taxon
##   <dbl>                                                    <chr>
## 1     1 Prokaryota || Bacteria || Alpha-Proteobacteria || Escharichia
## 2     2                                          Eukaryota || Chordata
```

```
(d1 <- dat %>% separate(taxon, c("riik", "hmk", "klass", "perekond"), sep = "\\|\\|", extra = "merge", 
```

```
## # A tibble: 2 x 5
##   index        riik        hmk                   klass     perekond
## * <dbl>       <chr>      <chr>                   <chr>        <chr>
## 1     1 Prokaryota   Bacteria   Alpha-Proteobacteria   Escharichia
## 2     2  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", fil
```

```
## # A tibble: 2 x 5
##   index       riik      hmk                klass    perekond
## * <dbl>      <chr>    <chr>                <chr>       <chr>
## 1     1 Prokaryota Bacteria Alpha-Proteobacteria Escharichia
## 2     2  Eukaryota Chordata                 <NA>        <NA>
```

```
(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 Prokaryota.Bacteria,Alpha-Proteobacteria.Escharichia
## 2     2                                   Eukaryota.Chordata
```

```
(d1 <- dat %>% separate(taxon, c('riik', 'hmk', "klass", "perekond"), sep = '[,\\.]', extra = "merge", 
```

```
## # A tibble: 2 x 5
##   index       riik      hmk                klass    perekond
## * <dbl>      <chr>    <chr>                <chr>       <chr>
## 1     1 Prokaryota Bacteria Alpha-Proteobacteria Escharichia
## 2     2  Eukaryota Chordata                 <NA>        <NA>
```

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

5.10 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
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

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 <- x1 %>% factor() %>% 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
##    <int>        <fctr> <int> <fctr>         <fctr>             <fctr>
##  1  2000 Never married    26  White  $8000 to 9999       Ind,near rep
##  2  2000      Divorced    48  White  $8000 to 9999 Not str republican
##  3  2000       Widowed    67  White Not applicable        Independent
##  4  2000 Never married    39  White Not applicable       Ind,near rep
##  5  2000      Divorced    25  White Not applicable   Not str democrat
##  6  2000       Married    25  White $20000 - 24999    Strong democrat
##  7  2000 Never married    36  White $25000 or more Not str republican
##  8  2000      Divorced    44  White  $7000 to 7999       Ind,near dem
##  9  2000       Married    44  White $25000 or more   Not str democrat
## 10  2000       Married    47  White $25000 or more  Strong republican
## # ... with 21,473 more rows, and 3 more variables: relig <fctr>,
## #   denom <fctr>, tvhours <int>
```

```
gss_cat %>% count(race)
```

```
## # A tibble: 3 x 2
##     race     n
##   <fctr> <int>
## 1  Other  1959
## 2  Black  3129
## 3  White 16395
```

Nii saame ka teada, mitu korda iga faktori tase selles tabelis esineb.

5.10.1 `fct_recode()` rekodeerib faktori tasemed

```
gss_cat %>% count(partyid)
```

```
## # A tibble: 10 x 2
##               partyid     n
##                <fctr> <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
##  7        Independent  4119
##  8       Ind,near dem  2499
##  9   Not str democrat  3690
## 10    Strong democrat  3490
```

```
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
##                  <fctr> <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
## 7        Democrat, weak  3690
## 8      Democrat, strong  3490
```

`fct_recode()` ei puuduta neid tasemeid, mida selle argumendis ei mainita. Lisaks saab mitu vana taset muuta üheks uueks tasemeks.

5.10.2 `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)
```

5.10.3 `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
##       <fctr> <int>
## 1 Protestant 10846
## 2   Catholic  5124
## 3       None  3523
## 4      Other   913
## 5  Christian   689
## 6     Jewish   388
```

5.10.4 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 i
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) #car:recode()
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")
```

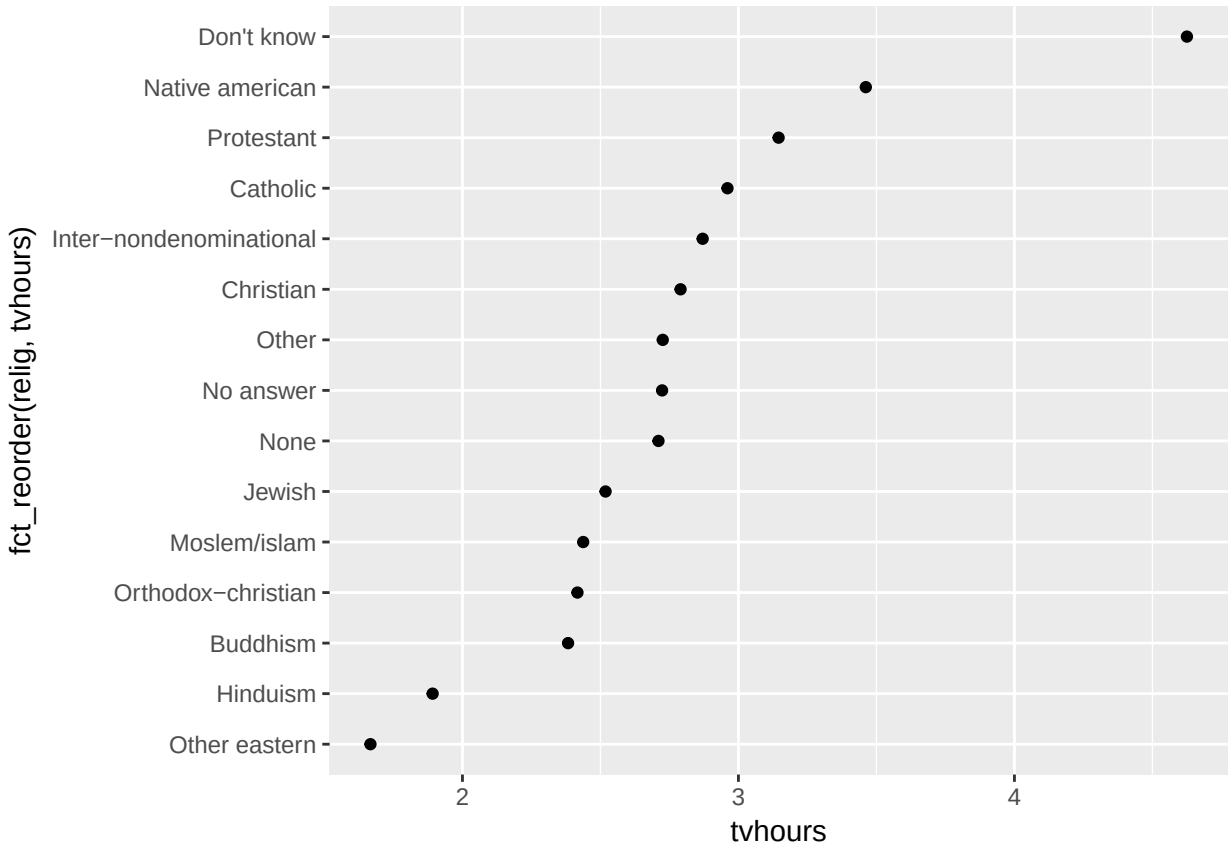
```
## [1] NA NA  3 NA NA  3 NA NA  3
```

```
recode(x, "1:2='A'; 3='B'")
```

```
## [1] "A" "A" "B" "A" "A" "B" "A" "A" "B"
```

5.10.5 Muudame faktori tasemete järjekorda joonisel

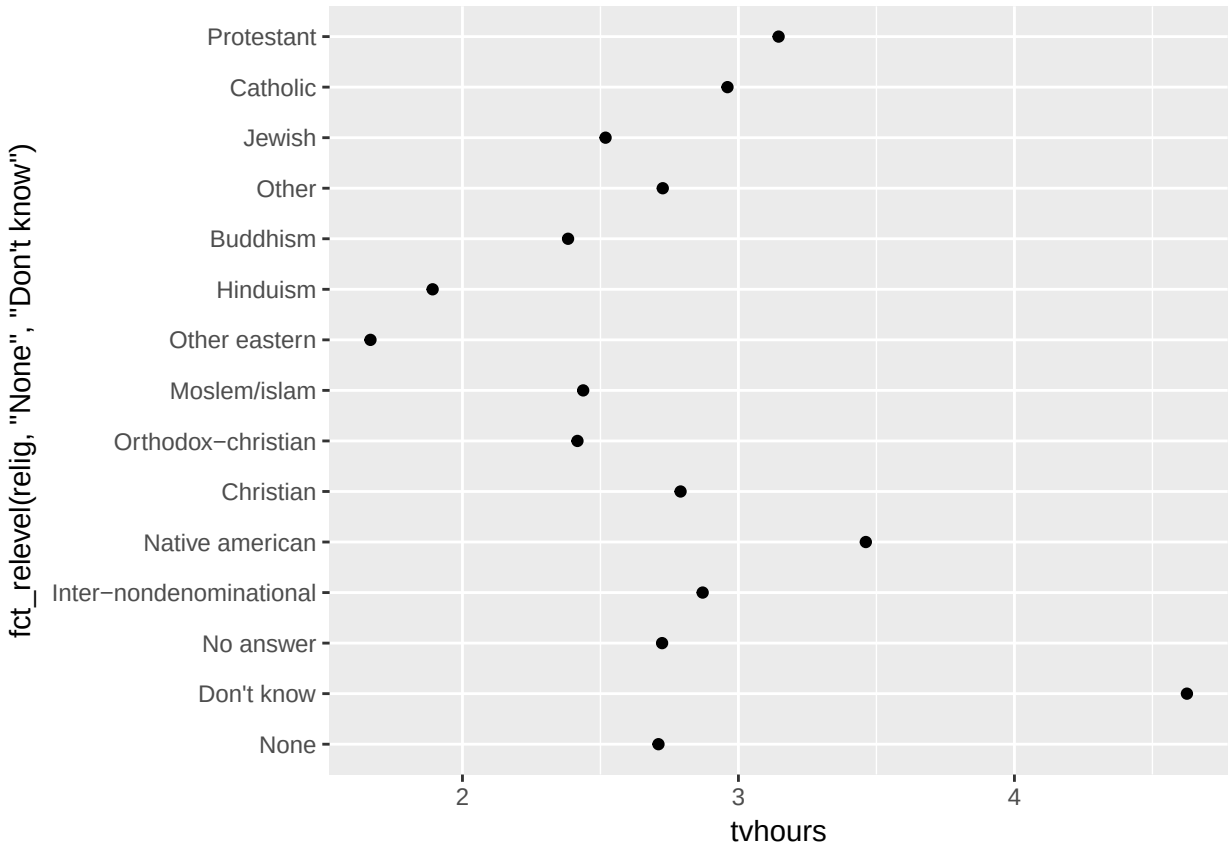
```
## summeerime andmed
gsscat_sum  <- group_by(gss_cat, relig) %>%
  summarise(age = mean(age, na.rm = TRUE),
            tvhours = mean(tvhours, na.rm = TRUE),
            n = n())
## joonistame graafiku
p <- ggplot(gsscat_sum, aes(tvhours, fct_reorder(relig, tvhours))) +
  geom_point()
p
```



5.10.6 `fct_relevel()` tõstab joonisel osad tasemed teistest ettepoole

Argumendid on faktor f ja need tasemed (jutumärkides), mida sa tahad tõsta.

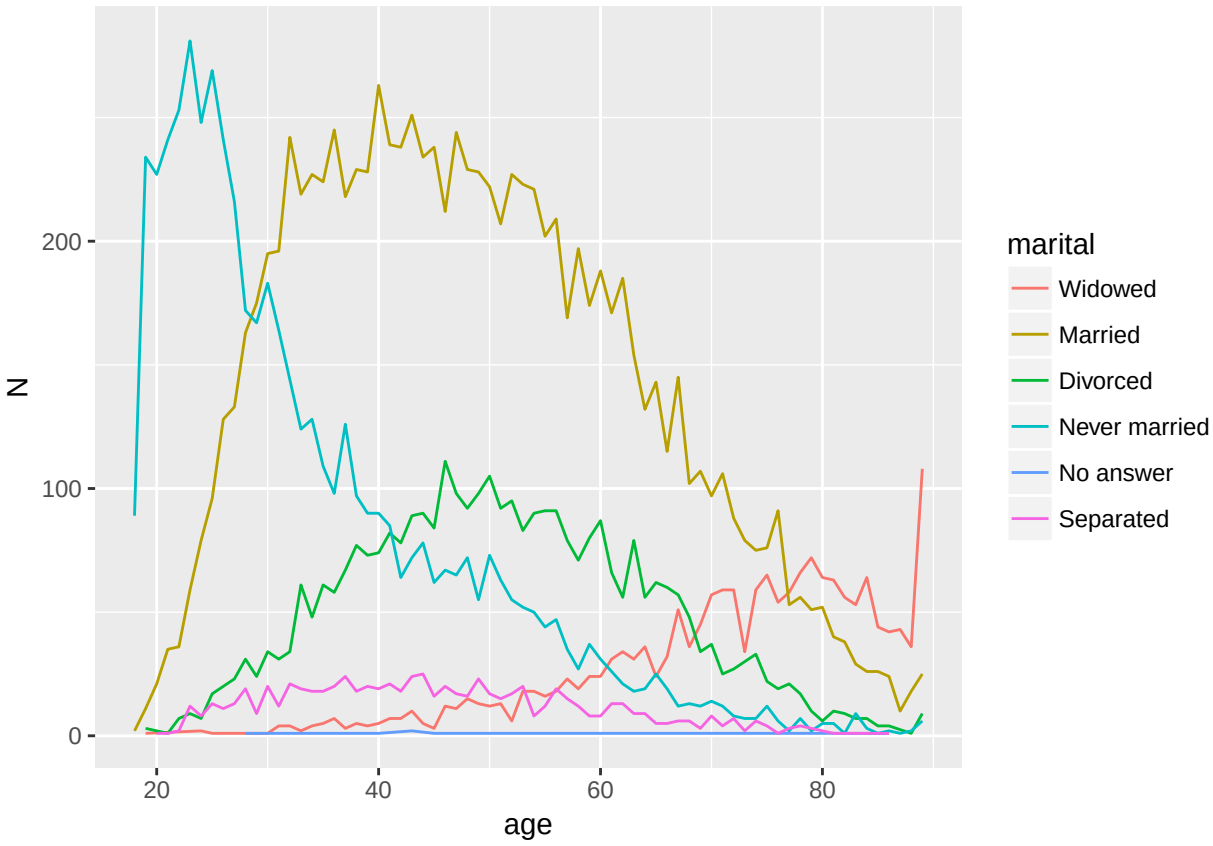
```
## täiendame eelmist graafikut ümberkorraldatud andmetega
p + aes(tvhours, fct_relevel(relig, "None", "Don't know"))
```



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

See muudab ploti paremini jälgitavaks:

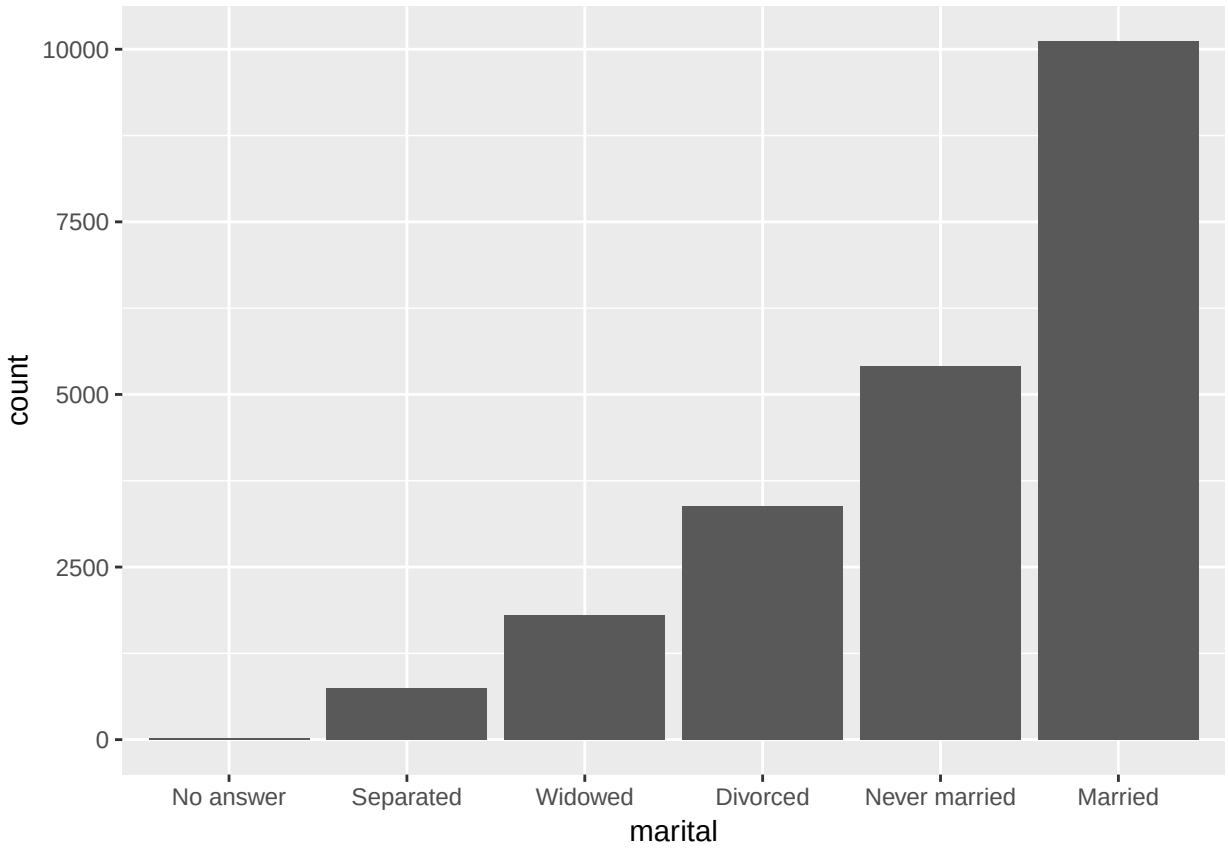
```
## summeerime andmed
gsscat_sum <- filter(gss_cat, !is.na(age)) %>%
  group_by(age, marital) %>%
  mutate(N=n())
## paneme andmed graafikule
ggplot(gsscat_sum, aes(age, N, colour = fct_reorder2(marital, age, N))) +
  geom_line() +
  labs(colour = "marital")
```



5.10.8 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()
```


10000
7500
5000
2500
0
count
No answer
Separated
Widowed
Divorced
Never married
Married
marital

Chapter 6

R on andmeanalüüsi keel, mille verbid on funktsioonid

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] -0.1166019
```

```
# 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] 1.30384
```

```
# 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 on topic 'sum' was found in the following packages:
##
##   Package               Library
##   mosaic                /Library/Frameworks/R.framework/Versions/3.4/Resources/library
##   base                  /Library/Frameworks/R.framework/Resources/library
##   Brobdingnag           /Library/Frameworks/R.framework/Versions/3.4/Resources/library
##
##
## Using the first match ...
```

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 (x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
##     FALSE))
## {
##     if (lazyeval::is_formula(x)) {
##         if (is.null(data))
##             data <- lazyeval::f_env(x)
##         formula <- mosaicCore::mosaic_formula_q(x, groups = groups,
##             max.slots = 3)
##         return(maggregate(formula, data = data, FUN = base::sum,
##             ..., na.rm = na.rm, .multiple = FALSE))
##     }
##     base::sum(x, ..., na.rm = na.rm)
## }
## <environment: namespace:mosaic>
```

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

6.1 Kirjutame oma esimese R funktsiooni

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

Kõige pealt kirjutame natuke koodi.

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

```
## [1] 5
```

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

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

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

Seda koodi jooksutades näeme, et meie funktsioon ilmub R-i 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 standrardvea arvutamiseks (baas R-s sellist funktsiooni ei ole): `sd()` funktsioon arvutab standardhälbe. Sellel on kaks argumenti: x and … and data and groups and na.rm. Me teame, et SEM=SD/sqrt(N) kus N = length(x)

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

x hoiab lihtsalt kohta andmetele, mida me tahame sinna 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] 12.80338
```

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

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

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

calc_sem(numbers)
```

```
## [1] 11.4517
```

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] 12.80338
```

```
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.

6.2 Funktsioonide raamatukogud

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 3.4.1 sisaldab 453 funktsiooni. Enamasti on sarnaseid asju tegevad funktsioonid koondatud kokku raamatukogudesse ehk pakettidesse, mis tuleb eraldi R kesksest repositooriumist CRAN alla laadida ja installeerida.

Selleks, et kasutada raamatukogudes leiduvaid funktsioone, tuleb kõigepealt antud raamatukogu oma arvutisse alla laadida (seda peab tegema ainult üks kord) ja seejärel see raamatukogu töökeskkond sisse lugeda (seda tuleb teha igal R sessioonil, kus me vastava raamatukogu funktsioone kasutame).

Näiteks, hea laps ei lahku kunagi kodust ilma raamatukogu "tidyverse"-ta.

Kõigepealt installeerime selle oma arvutisse.

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

NB! kui mõni raamatukogu sel viisil alla ei tule, siis guugeldage selle nime + R ja vaadake instruktsioone allalaadimiseks.

Raamatukogud elavad *repositorides*, millest suurim on CRAN. `install.packages()` otsib, laadib alla ja installeerib teie soovitud raamatukogu just sealt. Teised suuremad repod on GitHub https://github.com, kus elab sageli R pakettide source kood ja dev versioonid, ja Bioconductor (seal on tuhandeid bioloogiliseks analüüsiks vajalikke pakette). Neist allalaadimine on pisut erinev, aga mitte kuigi keeruline. Kõigi pakettide kohta on võrgus ka allalaadimise õpetus. Googeldage paketi nime järgi.

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.

```
library(tidyverse)
```

Antud kursuse raames tuleb seda teha iga kord, kui te R sessiooni alustate. Pakette/library-sid on soovitav laadida töökeskkonda muidu ikka ainult vastavalt vajadusele. "tidyverse"-gi puhul on soovitatav eelistatult laadida ainult see/need raamatukogud, mida tegelikult kasutama ka hakatakse. Vaikimisi kogu aeg kogu "tiduverse" rammatukogu laadimine ei pruugi olla reprodutseeritavuse mõttes hea, sest paratamatult võivad tekkida konfliktid teiste mitte-tidyverse raamatukogude funktsioonidega. Näiteks on `select()` funktsioon lisaks "dplyr" librarile olemas ka mitmes teises paketis.

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

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

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

6.3 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.

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

Nüüd võite näiteks kasutada funktsiooni `drop_na()`, mis eemaldab teie andmetabelist read, mis sisaldavad NA-sid. See funktsioon asub "tidyr" raamatukogus, mis koos ülejäänud "tidyverse"-ga sisse loeti.

```
?drop_na
```

Verb: `drop_na()` loe: "viska NA-d välja".

```
# creates a two-column table
df <- tibble(apples = c(2, 3, NA, 5), oranges = c(4, NA, 6, 4))
df
```

```
## # A tibble: 4 x 2
##   apples oranges
##    <dbl>   <dbl>
## 1      2       4
## 2      3      NA
## 3     NA       6
## 4      5       4
```

Nüüd filtreerime tabelist välja selle rea mis sisaldavad NA väärtusi tulbas "apples". Tabeli mõlemas tulbas on üks NA, seega jääb tabelisse tulbas "oranges" NA-d sisaldav rida alles.

```
df1 <- drop_na(df, apples)
df1
```

```
## # A tibble: 3 x 2
##   apples oranges
##    <dbl>   <dbl>
## 1      2       4
## 2      3      NA
## 3      5       4
```

6.4 Hurraa, uus arvuti! aga mis libraryd mul olid...

Kui te uuendate oma R versiooni või olete ostnud uue arvuti, siis lähevad kaduma kõik teie paketid. Nende automaatseks allalaadimiseks võite teha nii:

```
## in the old version/machine run:
installed <- as.data.frame(installed.packages())
write.csv(installed, "data/installed_previously.csv")

## when new machine, move "data/installed_previously.csv" into new computer

## and in the new one:
installedPreviously <- read.csv("data/installed_previously.csv")
baseR <- as.data.frame(installed.packages())
toInstall <- setdiff(installedPreviously$Package, baseR$Package)
install.packages(toInstall)
```

Tundub juskui tark ja mõistlik, kuid:

- Massiline pakettide installatsioon uude arvutisse võib kergesti takerduda erinevate R-i väliste sõltuvuste taha ja tekitab frustratsiooni.
- Lisaks, ebavajalikest pakettidest vabanemiseks on mõistlik vajaminevad paketid jooksvalt tagasi installeerida.
- Kiire internet on laialt saadaval ja ka kompileeritavad paketid ei võta installatsiooniks rohkem kui mõni minut aega.

Chapter 7

Bayesiaanlik statistika

7.1 Ajaloost ja tõenäosusest

Bayesiaanlik ja sageduslik statistika leiutati üksteise järel Pierre-Simon Laplace poolt, kes arendas välja kõigepealt bayesi statistika alused ning seejärel sagedusliku statistika omad (ca. 1800 - 1812). Sagedusliku statistika tekkimise ja hilisema õitsengu põhjus 20. sajandil oli arvutuslik lihtsus. Bayesi meetoditega ei olnud võimalik korralikult teadust teha enne 1990-ndaid aastaid, mil personaalarvutite levik algatas buumi nende meetodite arendamises. Praegu on maailmas bayesi ja sageduslikku statistikat umbes pooleks (vähemalt uute meetodite arendustöö poole pealt). Eestis bayesi statistika 2017 aasta seisuga peaaegu, et puudub.

Kahe statistika põhiline erinevus ei tule matemaatikast — mõlemad harud lähtuvad samadest tõenäosusteooria aksioomidest ja nende vahel puuduvad matemaatilised lahkarvamused — vaid tõenäosuse tõlgendusest.

Bayesi tõlgenduses on tõenäosus teadlase usu määr mingi hüpoteesi kehtimisse. Hüpotees võib näiteks olla, et järgmise juulikuu sademete hulk Vilsandil jääb vahemikku 22 kuni 34 mm. Kui Bayesi arvutus annab selle hüpoteesi tõenäosuseks 0.57, siis oleme me selle teadmise najal nõus maksma mitte rohkem kui 57 senti kihlveo eest, mille alusel makstakse juhul, kui see hüpotees tõeseks osutub, välja 1 EUR (ja me saame vähemalt 43 senti kasumit).

Sageduslikud statistikud usuvad, et selline tõenäosuse tõlgendus on ebateaduslik, kuna see on "subjektiivne". On võimalik, et n teadlast arvutavad korrektselt samade andmete põhjal n erinevat tõenäosust ja usuvad seega samade tõendite põhjal erinevaid asju. See võib juhtuda siis, kui nad lähtuvad väga erinevatest taustauskumustest oma hüpoteeside kehtimise kohta. Kui te usute, et teie taustateadmised ei tohi mingil juhul mõjutada järeldusi, mis te oma andmete põhjal teete, siis te ei ole bayesiaan. Sel juhul pakub alternatiivi sageduslik tõenäosuse tõlgendus. Sageduslik tõenäosus on defineeritud kui teatud tüüpi andmete esinemise pikaajaline suhteline sagedus. Näiteks, kui me viskame münti palju kordi, siis peaks kullide (või kirjade) suhteline sagedus meile andma selle mündi tõenäosuse langeda kiri üleval. Selline tõenäosus on omistatav ainult sellistele sündmustele, mille esinemisel on sagedus. Teaduslik teooria ei ole selline sündmus. Seega ei ole sageduslikus statistikas võimalik rääkida ka hüpoteesi kehtimise tõenäosusest. Sageduslik lahendus on selle asemel, et rääkida meie hüpoteesi tõenäosusest meie andmete korral, rääkida andmete, mis sarnanevad meie andmetega, esinemise tõenäosusest null-hüpoteesi (mis ei ole meie hüpotees) kehtimise korral. Seega omistatakse sagedus (= tõenäosus) andmetele, mitte hüpoteesile. Järgnevalt toome näite, kuidas bayesiaan ja sageduslik statistik lahendavad sama ülesande.

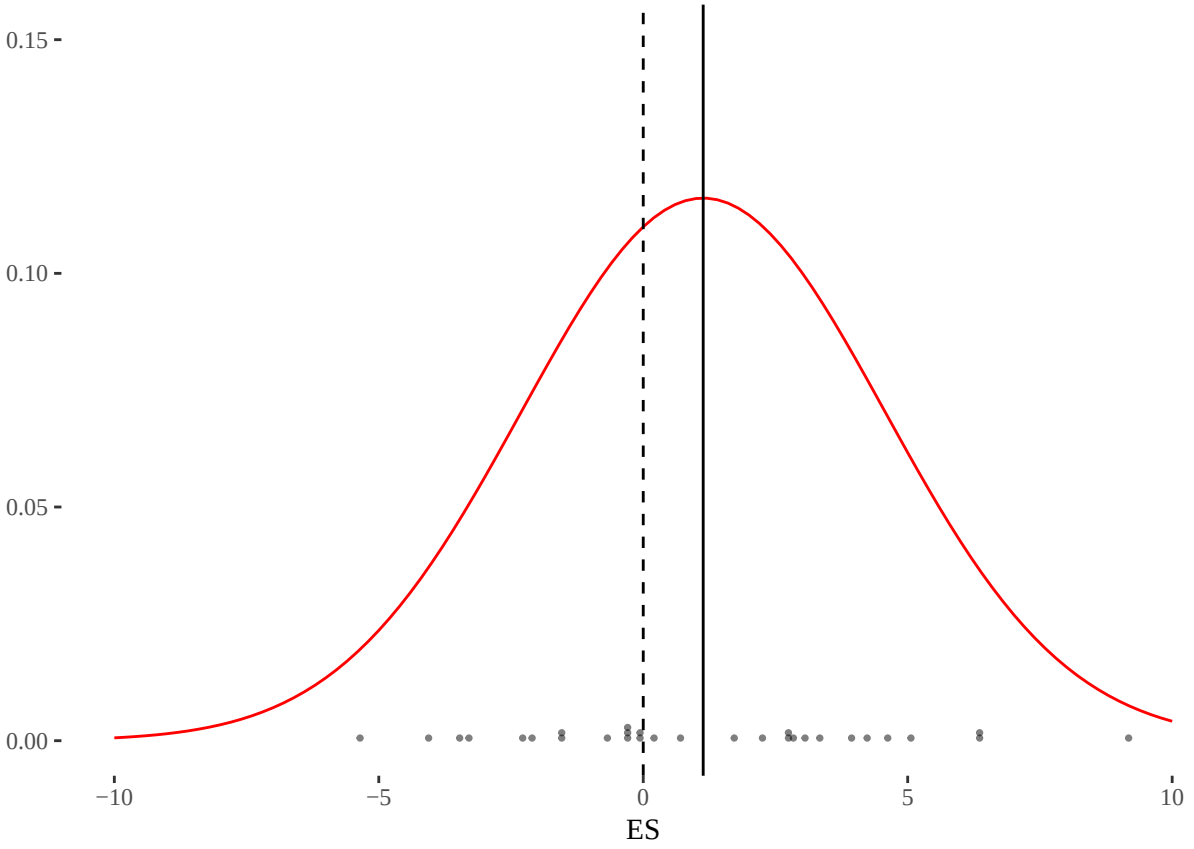
7.1.1 Näide: kahe grupi võrdlus

meil on 2 gruppi, katse ja kontroll, millest kummagis 30 mõõtmist ja me soovime teada, kui palju katsetingimus mõjutab mõõtmistulemust. Meie andmed on normaaljaotusega ja andmepunktid, mida me analüüsime, on efektisuurused (katse1 - kontroll1 = es1 jne).

7.1.1.1 Bayesiaan

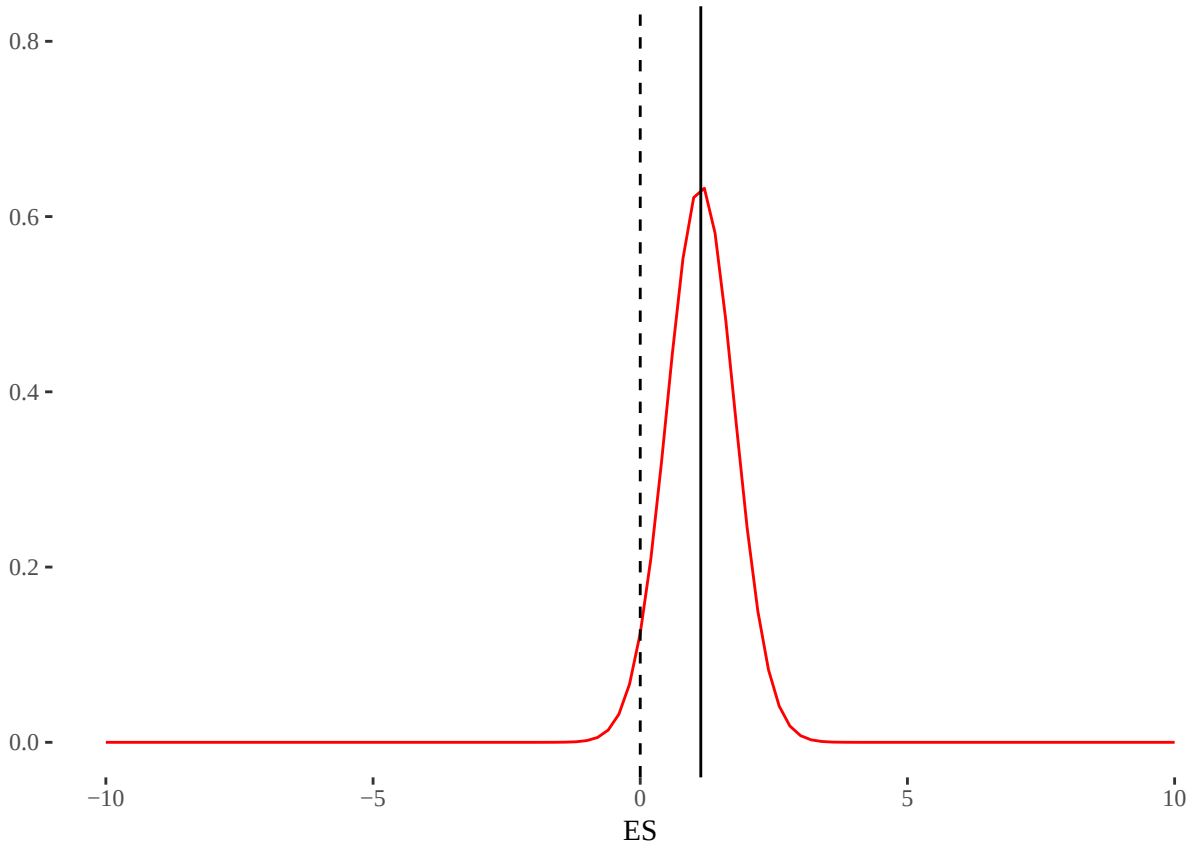
Statistiline küsimus on Bayesiaanil ja sageduslikul statistikul sama: kas ja kui palju erinevad kahe grupi keskväärtused? Bayesiaan alustab sellest, et ehitab kaks mudelit: andmete tõepäramudel ja taustateadmiste mudel ehk prior.

Kui andmed on normaaljaotusega, siis on ka tõepäramudel normaaljaotus. Alustame sellest, et fitime oma valimiandmed (üksikud efekti suurused) normaaljaotuse mudelisse.



Joonis 1. paariviisiline katse - kontroll disain. Katset on korratud 30 korda. X teljel on effekti suurused (ES). 30 ES-i on näidatud punktidena. Must joon näitab keskmist ES-i. Andmed on mudeldatud normaaljaotusena.

See ei ole veel tõepäramudel, sest me tahame hinnangut ES **keskväärtuse** kõige tõenäolisemale väärtusele, ja lisaks veel hinnangut ebakindlusele selle punkt-hinnangu ümber (usalduslpiire). Seega tuleb eelmine jaotus kitsamaks tõmmata, et ta kajastaks meie teadmisi ES-ide keskväärtuste, mitte individuaalsete ES-de, kohta. Uue jaotusmudeli sd = eelmise jaotuse sd/sqrt(30).

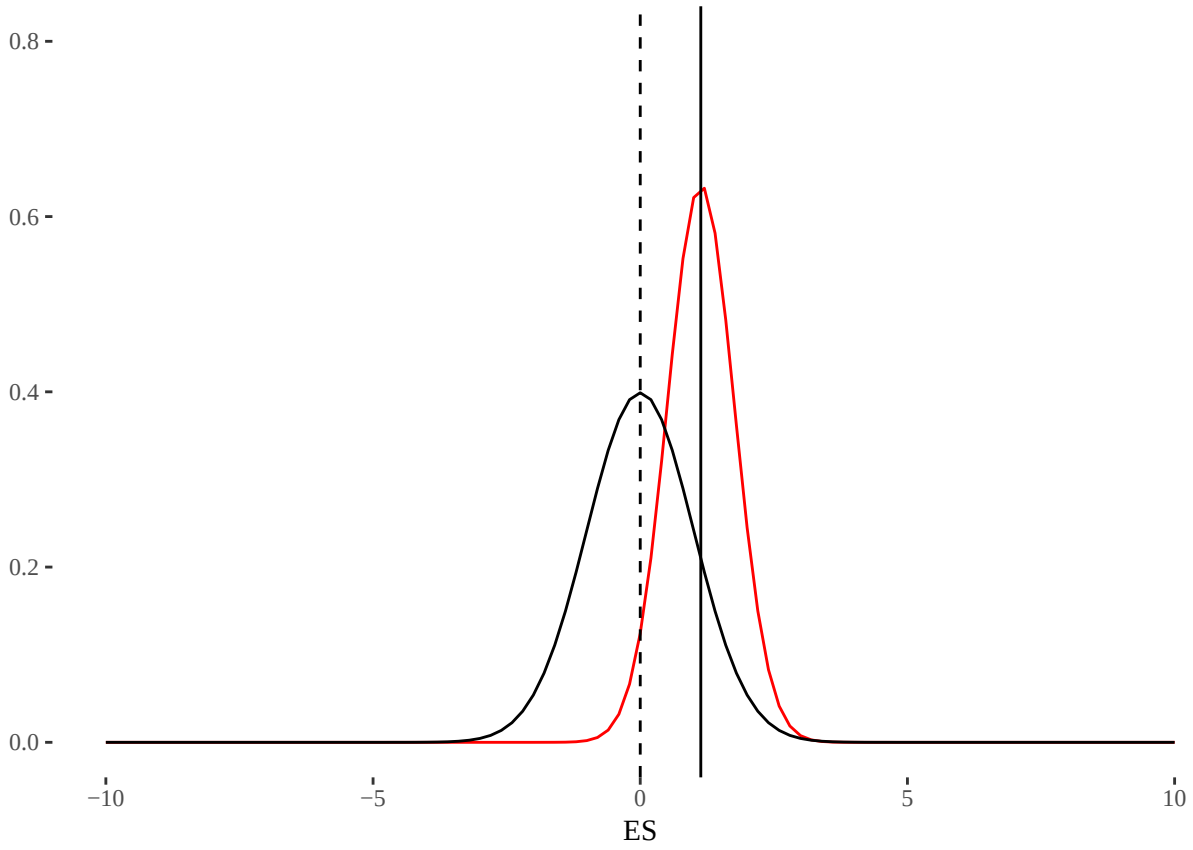


Joonis 2. See jaotus iseloomustab keskmise ES paiknemist puhtalt meie andmete põhjal.

Täpsemalt, selle joonise põhjal võib arvutada, milline on meie valimi keskväärtuse kohtamise tõenäosus igal võimalikul tõelisel ES-i väärtusel. Kõige tõenäolisemad on andmed siis, kui tegelik ES = andmete keskväärtusega (seda kohta näitab must joon). Kui me jagame musta joone pikkuse punase kurvi all läbi katkendjoone pikkusega sama kurvi all, saame teada, mitu korda on meie andmed tõenäolisemad siis, kui tegelik ES = mean(valimi ES), võrreldes olukorraga, kus tegelik ES = 0. Loomulikult võime sama näitaja arvutada ükskõik millise hüpoteeside paari kohta (näiteks, andmed on miljon korda tõenäolisemad hüpoteesi ES = 0.02 all kui hüpoteesi ES = -1 all; mis aga ei tähenda, et andmed oleksid väga tõenäolised kummagi võrreldud hüpoteesi all).

Aga see ei ole veel Bayes. Lisame andmemudelile taustateadmiste mudeli. Sellega tühistame me väga olulise eelduse, mis ripub veskikivina sagedusliku statistika kaelas. Nimelt, et valimi andmed peavad olema esinduslikud populatsiooni suhtes. Me võime olla üsna kindlad, et väikeste valimite korral see eeldus ei kehti ja sellega seoses ei tööta ka sageduslik statistika viisil, milleks R.A. Fisher selle kunagi lõi. Taustateadmiste mudeli roll (kuigi mitte ainus) on õrnalt suunata meie hinnangut õiges suunas vähendades halbade andmete võimet meile kahju teha. Kui sul on väike valim, siis sinu andmed vajavad sellist kantseldamist.

Olgu meie taustateadmise mudel normaaljaotus keskväärtusega 0 ja standardhälbega 1



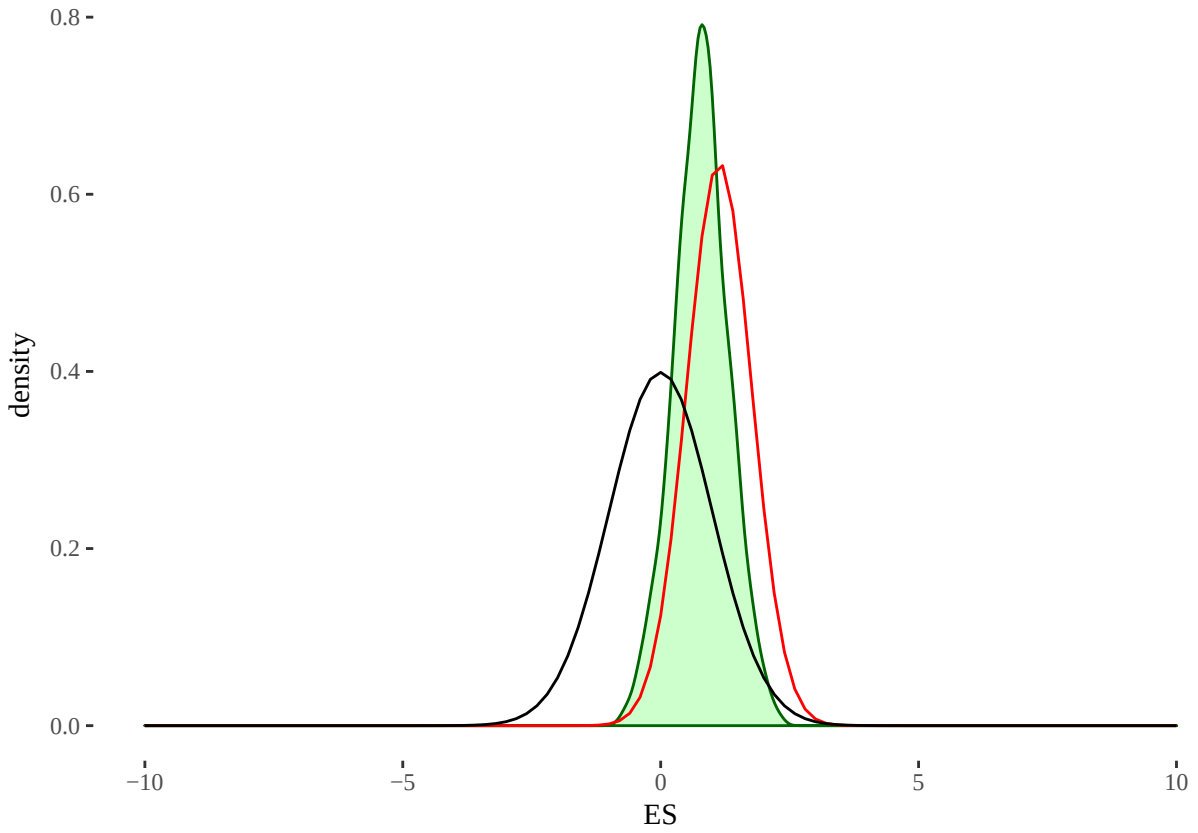
Joonis 3. Taustateadmiste mudel e prior on must normaaljaotus, mille ülesanne on veidi vähendada ekstreemsete valimite kahjulikku mõju.

Taustateadmiste mudel on sageli normaaljaotus. Kui meil on palju taustateadmisi, siis on see jaotus kõrge ja kitsas, kui meil on vähe taustateadmisi, siis on see madal ja lai.

```
Mida teha, kui sa ei taha, et taustateadmiste mudel sinu
posteeriori kuju mõjutab? Sellisel juhul kasutatakse nõrgalt
informatiivseid prioreid, mis tähendab, et priori
jaotus on palju laiem kui tõepäramudeli laius. Miks mitte
kasutada mitte-informatiivseid tasaseid prioreid?
Põhjused on arvutuslikud, seega tehnilist laadi.
```

Igal juhul järgmise sammuna korrutab bayesiaan selle jaotuse andmejaotusega, saades tulemuseks kolmanda normaaljaotuse, mille ta seejärel normaliseerib nii, et jaotuse alune pindala = 1. See kolmas jaotus on posterioorne tõenäosusjaotus, mis sisaldab kogu infot, millest saab arvutada kõige tõenäolisema katseefekti suuruse koos ebakindluse määraga selle ümber (mida rohkem andmeid, seda väiksem ebakindlus) ja tõenäosused, et tegelik katseefekt jääb ükskõik milllisesse meid huvitavasse vahemikku.

Nüüd ei ole siis muud kui bayesi mudel läbi arvutada.



Joonis 4. Triplot. Bayesi väljund on posterioorne tõenäosusjaotus (roheline). Nagu näha, ei ole selle jaotuse tipp täpselt samas kohas kui andmejaotuse tipp ehk keskväärtus. Prior tõmbab seda veidi nulli suunas. Lisaks on posteerior veidi kitsam kui andmemudel, mis tähendab, et hinnang ES-le tuleb väiksema ebakindluse määraga.

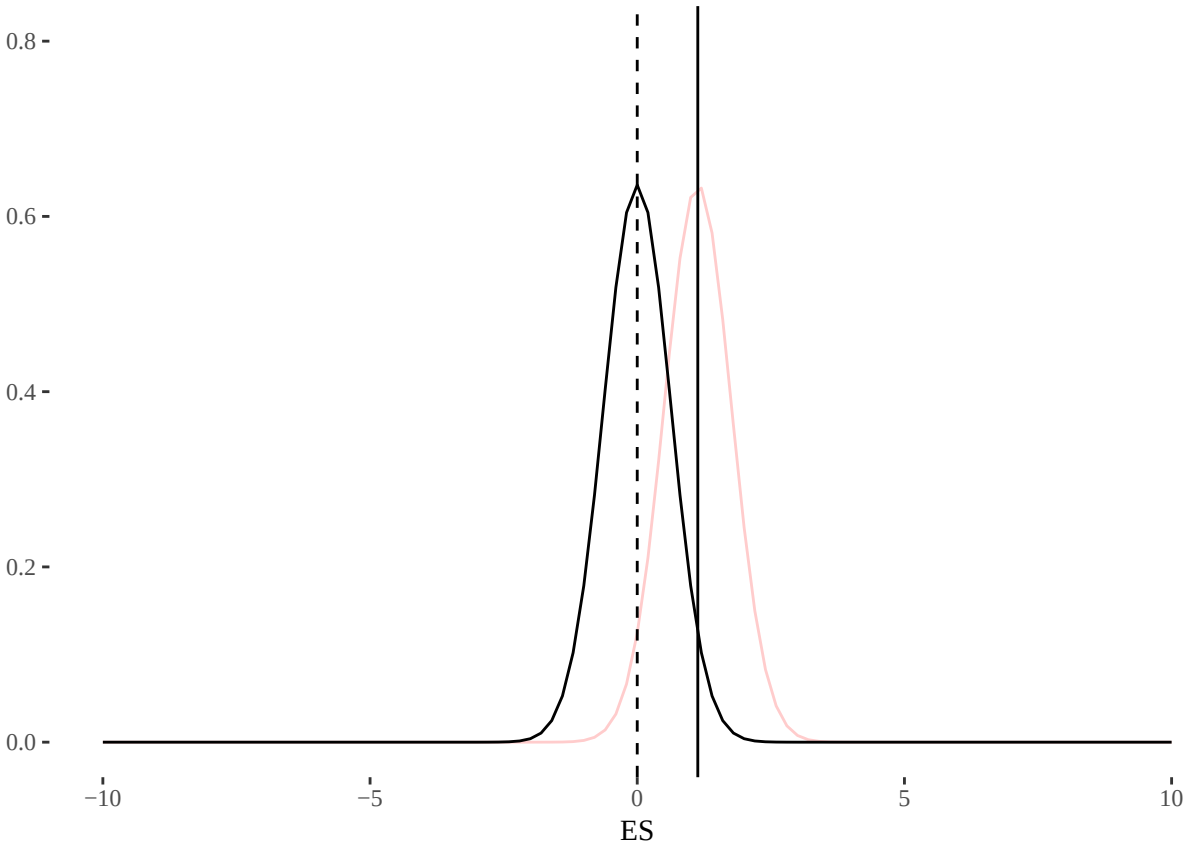
Posteerior sisaldab endas kogu infot, mis meil ES-i tõelise väärtuse kohta on. Siit saame arvutada

1. parima hinnangu ES-i punktväärtusele,
2. usaldusintervalli, ehk millisest ES-ide vahemikust loodame leida tõelise ES-i näit 90% tõenäosusega,
3. iga mõeldava ES-i väärtuste vahemiku kohta tõenäosuse, millega tõeline ES jääb sellesse vahemikku.
4. saame ES-i põhjal arvutada mõne muu statistiku, näiteks ES1 = log(ES), kasutades selleks ES-i posterioorset jaotust. Sel viisil kanname oma ES-i hinnangus peituva ebakindluse üle ES1-le, millele saame samuti rakendada punkte 1-3 (sest ES1 on posterioorne jaotus).
5. uute andmete lisandumisel saame kasutada ES-i posteeriorit uue priorina ja arvutada uue täiendatud posteeriori. Põhimõtteliselt võime seda teha pärast iga üksiku andmepunkti lisandumist. See avab ka head võimalused metaanalüüsiks.
6. lisaks saame oma algsest mudelist ka posteeriori andmepunkti tasemel varieeruvusele (pole näidatud). Seda kasutame uute andmete simuleerimiseks (meie näites üksikud ES-d).

7.1.1.2 Sageduslik statistik

Sageduslik lähenemine sisaldab ainult ühte mudelit, mida võrreldakse valimi andmetega. Sageduslik statistik alustab selles lihtsas näites täpselt samamoodi nagu bayesiaan, tekitades eelmisega identse andmemudeli, mis on keskendatud valimi keskväärtusele (Joonis 2). Seejärel nihutab ta oma andmemudelit niipalju, et normaaljaotuse tipp ei ole enam valimi keskväärtuse kohal vaid hoopis 0-efekti kohal. Jaotuse laius nihutamisel

ei muutu.



Seda nullile tsentreeritud mudelit kutsutakse null-hüpoteesiks (H0). Nüüd võrdleb ta oma valimi keskväärtust (must joon) H0 jaotusega. Kui valimi keskväärtuse kohal on H0 jaotus kõrge, siis on andmete tõenäosus H0 kehtimise korral suur. Ja vastupidi, kui valimi keskväärtuse kohal on H0 madal, siis on andmete esinemise tõenäosus H0 all madal. Seda tõenäosust kutsutakse p väärtuseks. Mida väiksem on p, seda vähem tõenäolised on teie andmed juhul, kui H0 on tõene ja katseefekt võrdub nulliga. P on defineeritud kui "teie andmete või 0-st veel kaugemal asuvate andmete esinemise pikaajaline suhteline sagedus tingimusel, et H0 kehtib".

7.1.1.3 tulemuste tõlgendamine

Kui sageduslik statistik kirjutab, et tema "efekti suurus on statistiliselt oluline 0.05 olulisusnivool", siis ta ütleb sellega, et tema poolt arvutatud $p < 0.05$. Selle väite korrektne tõlgendus on, et juhul kui statistik pika aja jooksul võtab omaks "statistiliselt olulistena" kõik tulemused, millega kaasnev $p < 0.05$ ja lükkab tagasi kõik tulemused, mille $p > 0.05$, siis sooritab ta 5% sagedusega tüüp 1 vigu. See tähendab, et igast sajast tõesest H0-st, mida ta testib, võtab ta keskeltläbi 5 vastu, kui statistiliselt olulised. Sageduslik statistika on parim viis tüüp 1 vigade sageduse pikaajaliseks fikseerimiseks. Paraku ei tea me ühegi üksiku testi kohta ette, kas see testib kehtivat või mittekehtivat H0-i, mis teeb raskeks katseseeriate ühekaupa tõlgendamise. Tuletame meelde, et sageduslikus statistikas ei saa rääkida H0 kehtimise tõenäosusest vaid peab rääkima andmete tõenäosusest (ehk andmete esinemise sagedusest) tingimusel, et H0 kehtib.

Kas ühte p väärtust saab tõlgendada kui hinnangut tõendusmaterjali hulgale, mida teie valim pakub H0 vastu? Selle üle on vaieldud juba üle 80 aasta, kuid tundub, et ainus viis seda kas või umbkaudu teha on bayesiaanlik. Igal juhul, p väärtust, mis on defineeritud pikaajalise sagedusena, on raske rakendada üksiksündmusele. Bayesiaanliku p väärtuste tõlgendamiskalkulaatori leiate aadressilt http://www.graphpad.com/quickcalcs/interpretPValue1/

```
Kujutle mass spektroskoopia katset, kus mõõdame 2000 valgu
tasemeid katse-kontroll skeemis ja katset korratakse n korda.
Sageduslik statistik kasutab adjusteeritud p väärtusi või q
väärtusi, et tõmmata piir, millest ühele poole jäävad statistiliselt
olulised ES-d ja teisele poole mitteolulised null-efektid. Edasi
tõlgendab ta mitteolulisi efekte kui ebaolulisi ja diskuteerib vaid
"olulisi" efekte. Paraku, p väärtuste arvutamine ja adjusteerimine
saab toimuda mitmel erineval moel ja usalduspiiri panekule just 95-le
protsendile, mitte näiteks 89% või 99.2%-le, pole ühtegi ratsionaalset
põhjendust. Seega tõmbab ta sisuliselt juhuslikus kohas joone läbi
efektide, misjärel ignoreerib kõiki sellest joonest valele poole jäänud
efekte. Meetod, mis väga hästi töötab pikaajalises kvaliteedikontrollis,
ei ole kahjuks kuigi mõistlik katse tulemuste ükshaaval tõlgendamises.
Mis juhtub, kui oleme kavalad ja proovime mitmeid erinevaid p väärtustega
töötamise meetodeid, et valida välja see usalduspiir, millest õigele poole
jäävaid andmeid on teaduslikult kõige parem tõlgendada? Ehkki ükshaaval
võisid kõik meie poolt läbi arvutatud meetodid olla lubatud (ja isegi
võrdselt head), ei fikseeri p enam tüüp 1 vigade sagedust. See tähendab,
et p on kaotanud definitsioonijärgse tähenduse ja te oleksite võinud oma
olulisuspiiri sama hästi tõmmata tunde järgi.
```

tüüpiline tulemuse kirjeldus artiklis:

1. sageduslik: the effect is statistically significant ($p < 0.01$).
2. bayesiaanlik: the most likely effect size is q1 (90% CI = q2, q3) and the probability that the true effect is < 0 is q4 percent.

90% CI — credible interval — tähendab, et me oleme 90% kindlad, et tegelik efekti suurus asub vahemikus q2 … q3.

7.1.1.4 kahe paradigma erinevused

1. sageduslikus statistikas võrdub punkt-hinnang tegelikule efekti suurusele valimi keskmise ES-ga. Bayesi statistikas see sageli nii ei ole, sest taustateadmiste mudel mõjutab seda hinnangut. Paljud mudelid püüavad ekstreemseid valimeid taustateadmiste abil veidi mõistlikus suunas nihutada, niiviisi vähendades ülepaisutatud efektide avaldamise ohtu.
2. sageduslik statistika töötab tänu sellele, et uurija võtab vastu pluss-miinus otsuseid: iga H0 kas lükatakse ümber või jäetakse kehtima. Seevastu bayesiaan mõtleb halli varjundites: sissetulevad andmed kas suurendavad või vähendavad hüpoteeside tõenäosusi (mis jäävad aga alati > 0 ja < 1).
3. p väärtused kontrollivad tüüp 1 vigade sagedust ainult siis, kui katse disaini ja hilisema tulemuste analüüsi detailid on enne katse sooritamist jäigalt fikseeritud (või eelnevalt on täpselt paika pandud lubatud variatsioonid katse- ja analüüsi protokollis). Eelkõige tähendab see, et valimi suurus ja kasutatavad statistilinsed testid peavad olema eelnevalt fikseeritud. Tüüpiliselt saame p väärtuse arvutada vaid üks kord ja kui $p = 0.051$, siis oleme sunnitud H0 paika jätma ning efekti deklareerimisest loobuma. Me ei saa lihtsalt katset juurde teha, et vaadata, mis juhtub. Bayesiaan seevastu võib oma posterioorse tõenäosuse arvutada kasvõi pärast iga katsepunkti kogumist ning katse peatada kohe (või alles siis), kui ta leiab, et tema posterioorne jaotus on piisavalt kitsas, et teaduslikku huvi pakkuda.
4. sagedusliku statistika pluss-miinus iseloom tingib selle, et kui tegelik efekti suurus on liiga väike, et sattuda õigele poole olulisusnivood, siis annavad statistiliselt olulisi tulemusi ülepaisutatud efektid, mida tekib tänu valimiveale. Nii saab süstemaatiliselt kallutatud teaduse. Bayesi statistikas seda probleemi ei esine, kuna otsused ei ole pluss-miinus tüüpi.

5. bayesi statistika ei fikseeri tüüp 1 vigade sagedust. See-eest võitleb see nn valehäirete vastu, milleks kaasajal kasutatakse enim hierarhilisi shrinkage mudeleid. See on bayesi vaste sageduslikus statistikas kasutatavatele multiple testingu korrektsioonidele. Kui sageduslik statistik võitleb valehäiretega p väärtusi adjusteerides ja selle läbi olulisusnivood nihutades, siis bayesiaan kasutab shrinkage mudelit, et parandada hinnanguid üksikute efektide keskväärtustele ja nende sd-le, kasutades paindlikult kogu andmesetis leiduvat infot.

See on kõik, mida me sagedusliku statistika kohta ütleme. Mitte miski, mis järgneb, ei eelda sagedusliku paradigma tundmist.

Chapter 8

1. osa: Mudel ja maailm

Andmeanalüüs ja statistika (siin sünonüümid) on lahutamatu osa igast loodusteadusest. Järgnevalt seletan, miks.

8.0.1 Suur ja väike maailm

Kuna maailm on liiga suur ja keeruline, et seda otse uurida, lõikavad teadlased selle väiksemateks tükkideks, kasutades tordilabidana teaduslike hüpoteese. Tüüpiline hüpotees pakub välja mittematemaatilise seletuse mõnele kitsalt piiritletud loodusnähtusele. Näiteks darvinistlik evolutsiooniteooria püüab seletada evolutsiooni toimemehhanisme. Seda teooriat võib võrrelda empiiriliste andmetega.

Mis juhtub, kui teie lemmikhüpotees on andmetega kooskõlas? Kas see tähendab, et see hüpotees on tõene? 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 (näit. lamarksistlik evolutsiooniteooria) ning rohkem kui üks üksteist välistav hüpotees võib olla olemasolevate andmetega võrdses kooskõlas. Asja teeb veel hullemaks, et teoreetiliselt on võimalik sõnastada lõpmatult palju erinevaid teooriaid, mis kõik pakuvad alternatiivseid ja üksteist välistavaid seletusi samale nähtusele.

Olgu peale, kui me vaatame maailma kõiketeadja jumala perspektiivist, siis tema võib vaadelda kõikehõlmava tõendusmaterjali sobivust kõigi võimalike teooriatega ning valida välja selle ainsa teooria, mis kõige paremini tõendusmaterjaliga sobib. Kuigi, see eeldaks, et tal on lõpmata palju andmeid, sest muidu ei oleks tal loogiliselt võimalik lõpmata paljude teooriate vahel valida - aga jumala jaoks on kõik võimalik. Igal juhul meie, surelike, jaoks tähendab see, et teaduslikus "faktis" saab alati kahelda, sest kunagi ei või kindel olla, et parimad teooriad lõpmata suurest teooriapilvest ei ole meil täiesti tähelepanuta jäänud ning, et meie jaoks eksisteerivad andmed kajastaksid hästi kõiki võimalikke andmeid. On selge nagu seebivesi, et mida vähem aega me kulutame teoorialoomeks ja andmete kogumiseks, seda vähem usutavad on ka meie teaduslikud järeldused. Sageli on nii, et mida kehvem on olukord andmerindel, seda rohkem vajame statistikat. Kui meil õnnestuks oma andmetest ilma statistikata saia teha, ei kõhkleks me hetkegi! Eriti, kuna statistikaga käivad käsikäes statistilised mudelid.

8.0.2 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 me tavakeelsest ja laiahaardelisest teaduslikust teooriast täpselt formuleeritud matemaatilise mudeli ning seejärel asume uurima oma mudelit.

Mudeli maailm erineb päris maailmast selle poolest, et mudeli maailmas on kõikvõimalikud sündmused, mis põhimõtteliselt võivad juhtuda, juba ette teada ja üles loetud (seda sündmuste kogu kutsutakse parameetriruumiks). Seega, tehniliselt on mudeli maailmas üllatused võimatud.

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 ehk oleks tõene. 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 mitteparameetriline) 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.

8.0.2.1 Näide: Aristoteles, Ptolemaios ja Kopernikus

Aristoteles lõi teooria maailma toimimise kohta, mis domineeris haritud Eurooplase maailmapilti enam kui 1200 aasta vältel. Selle kohaselt asub universumi keskpunktis 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, mis on omavahel seotud (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 maapeal kohtame. Kaasa arvatud meie sündimine, elukäik ja surm (mis on kõik liikumised). Kõik, mis maapeal 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 kaasaegses mõistes keemiast ja kosmoloogiast kuni bioloogia, maateaduse ja isegi geograafiani. Sellised ühendteooriad on nagu 500 aastased sekvoiad; nad on rasked langetada, aga kui mõni siiski kukub, kostab ragin kaugele. Samas, ühte Aristotelese kosmoloogia olulist puudust nähti kohe. Nimelt ei suuda Aristoteles seletada, miks osad planeedid teavavõlvil vahest suunda muudavad ja mõnda aega lausa vastupidises suunas liiguvad (retrogressioon). Kuna astronoomia põhiline kasutusala oli astroloogia, siis põõrati planeetide liikumisele suurt tähelepanu. Lahenduseks ei olnud aga mitte suure teooria ümbertegemine või ümberlükkamine, vaid nõudlus uue teaduse järele, mis "päästaks fenomenid". Siin tuli appi Ptolemaios, kes lõi matemaatilise mudeli, kus planeedid mitte lihtsalt ei liigu ringtrajektoori mõõda vaid samal ajal teevad ka väiksemaid ringe ümber esimese suure ringjoone. Neid väiksemaid ringe kutsutakse epitsükliteks. See mudel suutis planeetide liikumist taevavõlvil piisavalt hästi ennustada, et astroloogide nõudlik seltskond sellega rahule jäi.

Ptolemaiosel ja tema järgijatel oli tegelikult mitu erinevat mudelit. Osad neist ei sialdanud 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. On oluline, et leidus epitsüklitega mudel ja ilma epitsükliteta mudel, mis olid matemaatiliselt ekvivalentsed ja seega andsid võrdseid ennustusi. Oli selge, et Aristotelese teooria ja fenomenide päästmise mudelid olid fundamentaalselt erinevad asjad. Samal ajal kui Aritoteles **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 1000 aasta vältel 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 Koperniku juurde, kes teadusajaloolaste arvates vallandas 17. sajandi teadusliku revolutsiooni avaldades raamatu, kus ta asetab päikese universumi keskele ja paneb maa selle ümber ringtrajektooril tiirlema. Kas Kopernikus tõrjus sellega kõrvale Aristotelese, Ptolemaiose või mõlemad? Kaasaegne seisukoht on,

et kuigi Kopernikus soovis teha kolmandat, arvasid tema rängalt matemaatilise teose avaldamisele järgnenud 40 aasta vältel pea kõik asjatundlikud astronoomid, et ta soovis välja pakkuda vaid lihtsama alternatiivi epitsüklitega mudelile, mis selleks ajaks oli muutunud väga keerukaks (aga ka samavõrra ennustustäpseks). Kuna Kopernikuse raamat läks trükki ajal, mil selle autor oli juba surivoodil, kirjutas sellele eessõna üks tema vaimulikust sõber, kes püüdis oodatavat kiriklikku pahameelt leevendada sugereerides, et päikese keskele viimine ei ole muud kui 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 ringikujuliste trajektooride juurde, mis tähendas, et selleks, et tema mudel fenomenide päästmisel hätta ei jääks ja astroloogidele kasutu ei oleks, oli ta sunnitud maad ja planeete liigutama ümber päikese mõõda epitsükleid. Kokkuvõttes oli Koperniku mudel sama keeruline kui Ptolemaiose standardmudel (neis oli võrdne arv epitsükleid) ja selle abil tehtud ennustused planeetide liikumise kohta olid väiksema täpsusega.

Koperniku mudel suutis samas ennustada mõningaid nähtusi (planeetide näiv heledus muutub ning 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 tuunimimst.

Kui vaadata Koperniku produkti teooriana, mitte mudelina, siis oli sel selgeid eeliseid Aristotelese ees. Juba ammu oli nähtud komeete üle taevavõlvi lendamas (mis Aristotelese järgi asusid kinnistähtede muutumatus sfääris) ja supernoova tekkimist ja kadu, ning enam ei olnud kaugel ka 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). Seega, teooria ja mudeli eristus on tähtis!

Koperniku teooriast tuleneb loogilise paratamatusena, et tähtedel esineb maapealt vaadates parallaks. See tähendab, et kui maakera koos astronoomiga teeb poolringi ümber päikese, siis kinnistähe näiv asukoht taevavõlvil muutub sest astronoom vaatleb 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. Tähtede parallaksi püüti mõõta juba Aleksandrias 1000 aastat enne Kopernikust, et leida kinnitust teooriale, mille kohaselt maakera tiirleb ümber päikese. Mõõtmised ei näidanud aga parallaksi olemasolu (sest maa trajektoori diameeter on palju lühem kui maa kaugus tähtedest). Parallaks mõõdeti edukalt alles 1838, siis kui juba ammu iga koolijüts uskus, et maakera tiirleb ümber päikese!

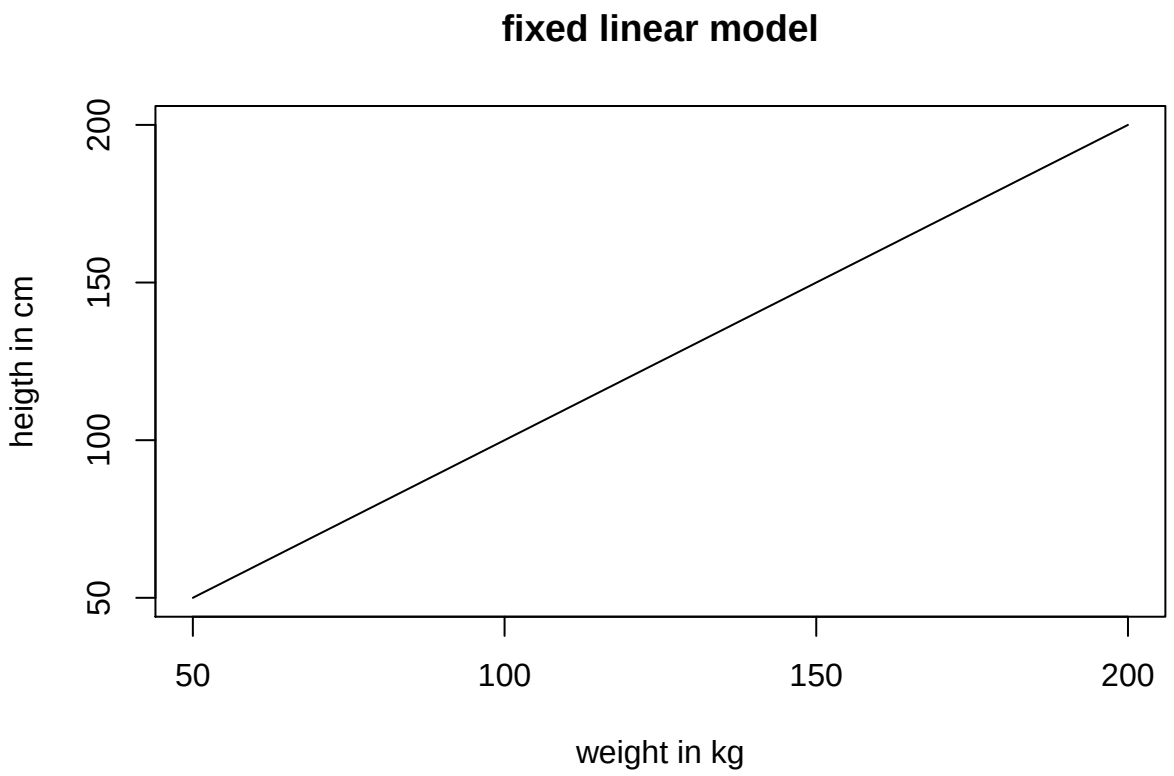
8.0.3 Millest koosneb mudel?

> Mudel on matemaatilise formalism, mis püüab kirjeldada füüsikalist protsessi. mudeli struktuuris on komponent, mis kirjeldab ideaalseid ennustusi ja eraldi veakomponent, mis kirjeldab kuidas andmepunktid varieeruvad selle ideaalse ennustuse ümber.

Näiteks, sageli kirjeldame me produkti kuhjumist ensüümreaktsioonis eksponentsiaalse funktsioni (mudeli) abil. Kui meie andmed seda tüüpi funktiooniga sobivad, ütleb see meile midagi konkreetse ensüümi töömehhanismi kohta. Teisest küljest, need mudelid, mis on “generatiivsed”, suudavad lisaks simuleerida ka uusi andmeid. Sealhulgas ka selliseid, mida päris maailmas ei saa kunagi esineda, sest seal puuduvad vastavad tingimused. Mudelisse saab aga sisse kirjutada igasuguseid tingimusi ehk parameetri väärtusi (näit substraadi konsentratsioone, mida me ei suuda “päriselt” saavutada).

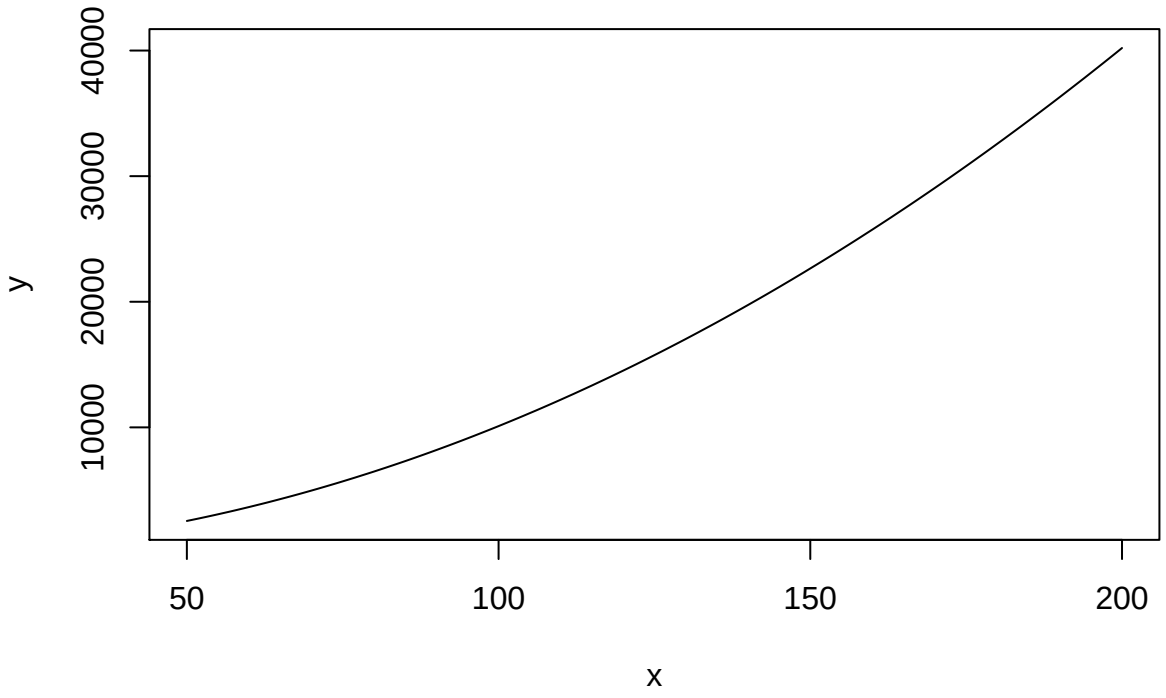
Oletame, et me mõõtsime N inimese pikkuse cm-s ja kaalu kg-s ning meid huvitab, kuidas inimeste pikkus sõltub nende kaalust. Lihtsaim mudel pikkuse sõltuvusest kaalust on pikkus = kaal (formaliseeritult: y = x) ja see mudel ennustab, et kui Johni kaal = 80 kg, siis John on 80 cm pikkune. Selle mudeli saame graafiliselt kujutada nii

```
x <- 50:200 #y = kaal
y <- x # x = pikkus
plot(y~x, type="l", xlab="weight in kg", ylab="heigth in cm", main="fixed linear model")
```



kõigepealt painutame sirget. See joon on ikka veel täielikult fikseeritud, aga ta pole enam sirge (ehki tehniliselt on meil ikka lineaarne seos x ja y vahel)

```
x <- 50:200
y <- x + x**2
plot(y~x, type="l")
```



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: ta on sirge kujuline ja selle sirge asukoht parameetriruumis on rangelt fikseeritud. Sirge lõikab y telge alati 0-s (ehk mudeli keeles: selle 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 selles mudelis ei ole parameetreid, mida me saaksime vabalt muuta ehk tuunida.

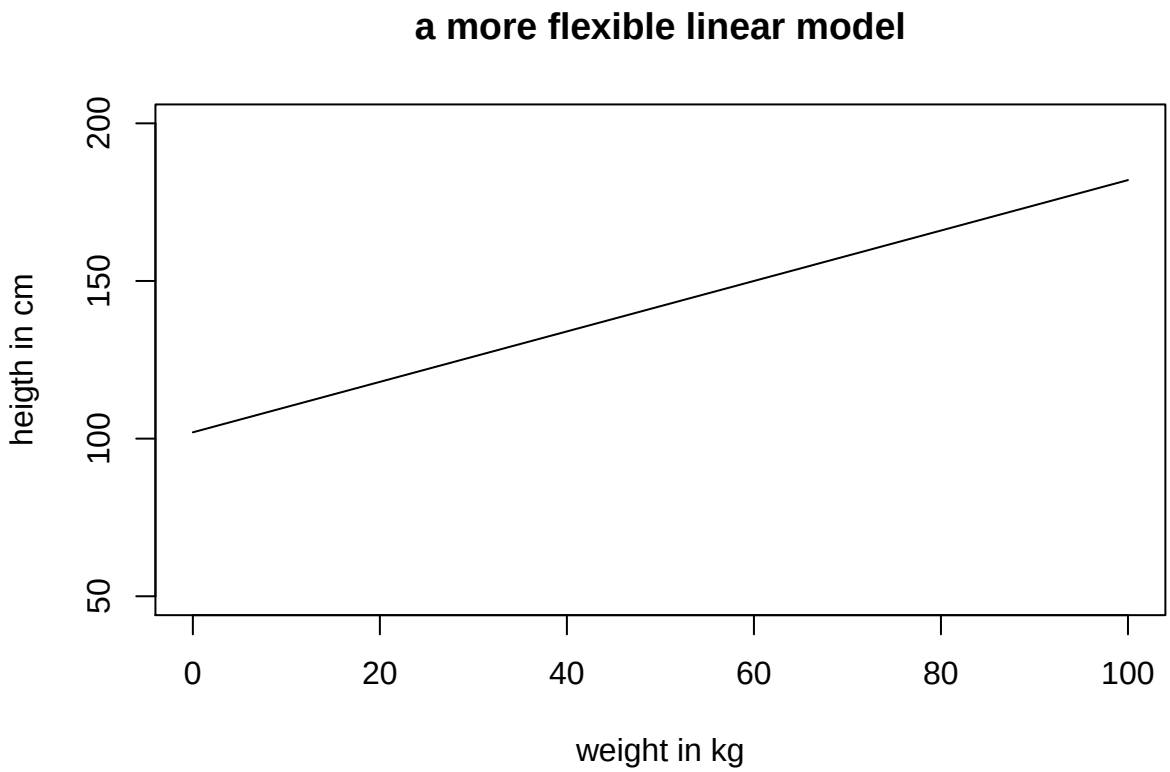
Kuidas aga kirjeldada sirget, mis võib paikneda 2-mõõtmelises ruumis ükskõik millises asendis? Selleks lisame mudelisse kaks parameetrit, intercept (a) ja tõus (b). Kui a=0 ja b=0, saame me eelpool kirjeldatud mudeli y = x. Kui a = 102, siis sirge lõikab y telge väärtusel 102. Kui b = 0.8, siis x-i tõustes 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 (mis juhtub, kui a = Inf?). Seega, Teades a ja b väärtusi ning omistades x-le suvalise meid huvitava väärtuse, saab ennustada y-i keskmist väärtust. Näiteks, olgu andmete vastu fititud mudel:

pikkus(cm) = 102 + 0.8 * kaal(kg) 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="heigth in cm", main="a more flexible linear model", ylim=
```



See mudel ennustab, et 0 kaalu juures on pikku 102 cm, mis on rumal, aga mudelite puhul tavaline, olukord. Me tuunime mudelit andmete peal, mis ei sisalda 0-kaalu (sest 0-kaaluga inimesi pole olemas). 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.

8.0.4 4 mõistet

X ja Y on muutujad, a ja b on parameetrid. Muutujate väärtused fikseeritakse andmete poolt, parameetrid fititakse muutujate väärtuste põhjal. Fititud mudel ennustab igale X-i väärtusele vastava kõige tõenäolisema Y väärtuse (Y keskväärtuse sellel X-i väärtusel).

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

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

muutuja (variable) - iga asi, mida me valimis mõõdame (X ja Y on kaks muutujat). Muutuja väärtused on fikseeritud andmete poolt. 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 me mudeli võimalikult hästi sobituma andmetega.

8.0.5 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 + xˆ2 tagab, et mudeli y = a + b1x + b2xˆ2 tüüp on parabool, ning parameetrite a, b1 ja b2 väärtused määravad selle parabooli täpse kuju. Ja nii edasi.

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;

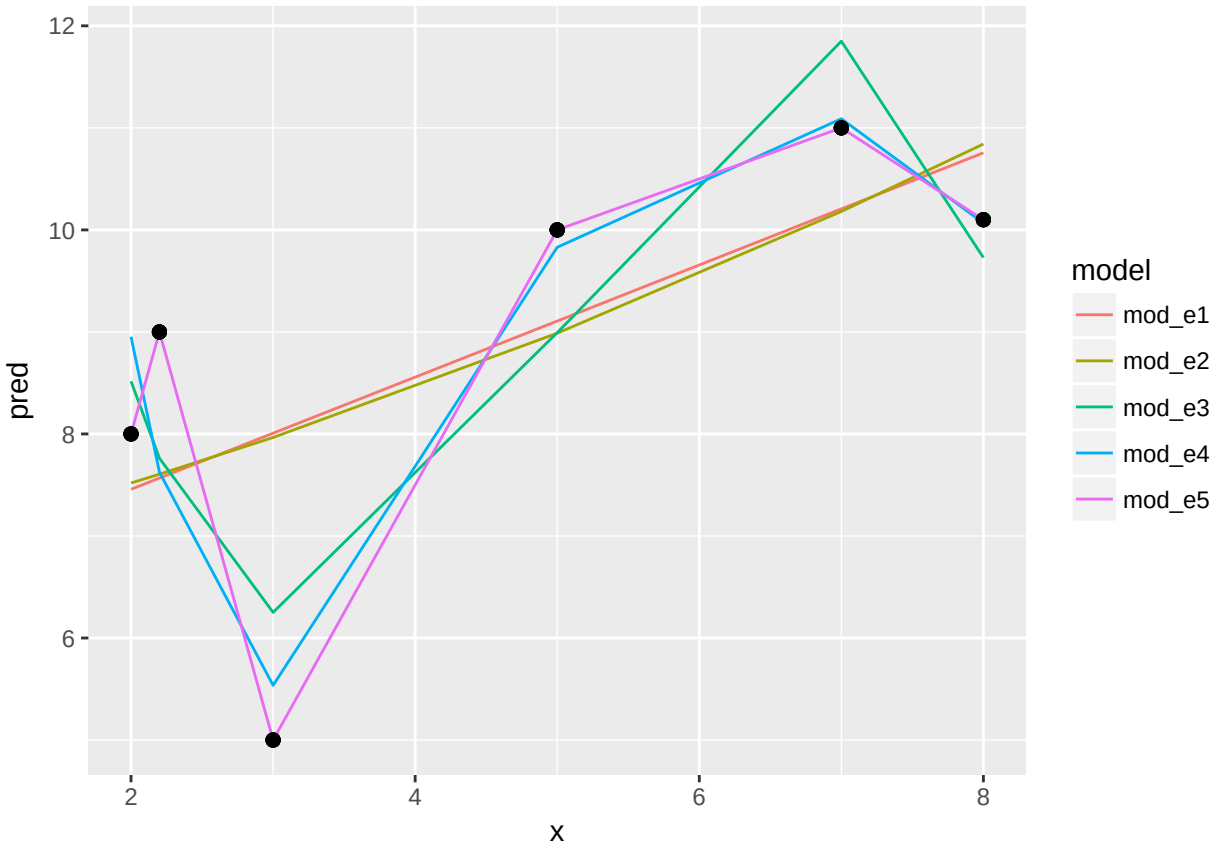
(3) genereerib usutavaid simuleeritud andmeid (see näitab mudeli kvaliteeti).

Sageli fititkse 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 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.

8.0.5.1 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 on suur mudeli ülefittimise oht. Ülefititud mudel järgib nii täpselt valimiandmeid, et sobitub hästi valimis leiduva juhusliku müraga ning seetõttu sobitub halvasti järgmise valimiga samast populatsioonist (sest igal valimil on oma juhuslik müra). Üldiselt, mida rohkem on mudelis tuunitavaid parameetreid, seda paindlikum mudel, seda kergem on seda valimiandmetega sobitada ja seda raskem on seda mudelit tõlgendada. Veelgi enam, alati on võimalik konstrueerida mudel, mis sobitub täiuslikult lõpliku arvu andmepunktidega (selle mudeli parameetrite arv = N). Selline mudel on täpselt sama informatiivne kui andmed, mille põhjal see fititi — ja täiesti kasutu.



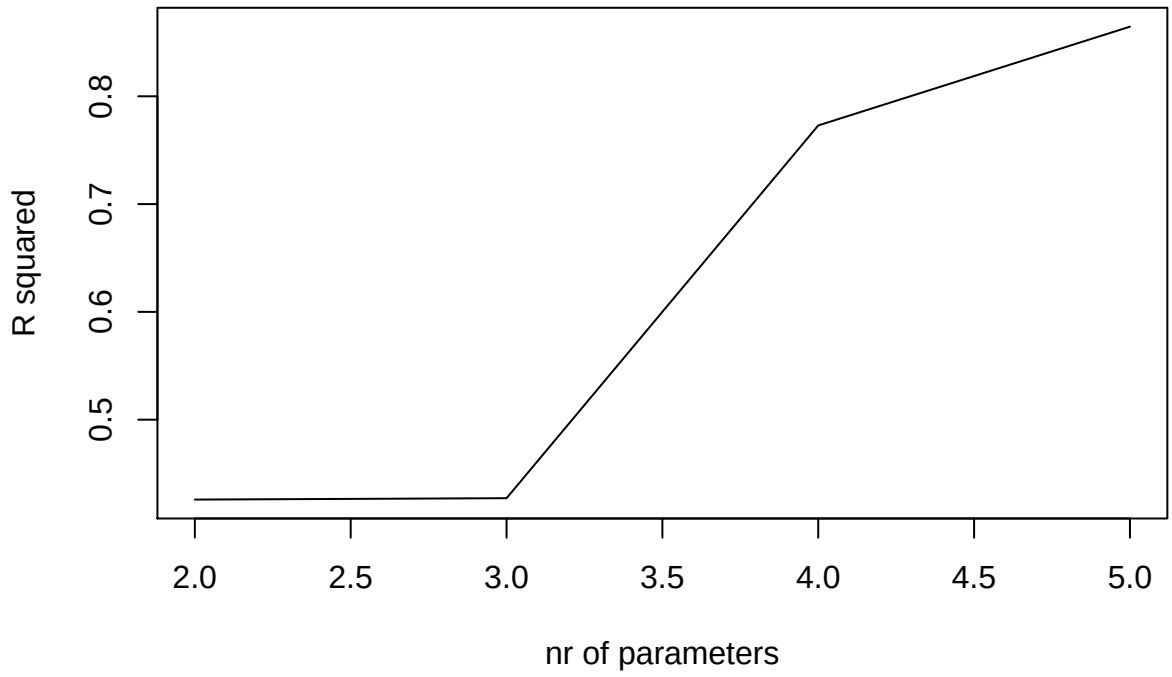
Joonis: Kasvava paindlikusega polünoomsed mudelid. mod_e1 on sirge võrrand y = a + b1x (2 parameetrit: a ja b1), mod_e2 on lihtsaim võimalik polünoom: y= a + b1x + b2x^2 (3 parameetrit), ..., mod_e5: y= a + b1x + b2x^2 + b3x^3 + b4x^4 + b5x^5 (6 parameetrit). mod_e5 vastab täpselt andmepunktidele (N = 6).

```
AIC(mod_e1, mod_e2, mod_e3, mod_e4, mod_e5)
```

```
##        df      AIC
## mod_e1  3 27.77993
## mod_e2  4 29.76669
## mod_e3  5 26.21330
## mod_e4  6 25.11245
## 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?

```
AIC - Akaike Informatsiooni Kriteerium - vaatab mudeli sobivust andmetega
ja mudeli parameetrite arvu.
Väikseim AIC tähitab parimat fitti 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: sedamööda kuidas parameetrite arv mudelis kasvab, kasvab ka R ruut. R ruut 0.8 tähendab, et x-i varieeruvus suudab seletada kuni 80% y-i varieeruvusest. Lisaparameetri lisamine ei saa põhimõtteliselt R ruutu vähendada. Aga selle kasvu kiirus on aeglustuv. Ühel hetkel ei õigusta mudeli fiti paranemine enam mudeli paindlikuse kasvu (mis mõlemad saavutatakse parameetreid lisades).

```
Ülefittimise vältimiseks kasutavad Bayesi mudelid informatiivseid prioreid,
mis välistavad ekstreemsed parameetriväärtused.
Vt http://elevanth.org/blog/2017/08/22/there-is-always-prior-information/
```

8.0.6 Veamudel

Eelpool kirjeldatud mudelid on deterministlikud — nad ei sisalda hinnangut andmete varieeruvusele ennustuse ümber. Neid kutsutakse ka **protsessi mudeliteks** sest nad modelleerivad protsessi täpselt. Ehk kui mudel ennustab, et 80 kg inimene on 166 cm pikkune, siis protsessi mudel ei ütle, kui suurt kaalust sõltumatut pikkuste varieeruvust võime oodata 80 kg-ste inimeste hulgas? Selle hinnangu andmiseks tuleb mudelile lisada veel üks komponent, **veamudel** ehk veakomponent, mis sageli tuuakse sisse normaaljaotuse kujul. Veakomponent modelleerib üksikute inimeste pikkuste varieeruvust (mitte keskmise pikkuse varieeruvust) igal mõeldaval ja mittemõeldaval kaalul. Tänu sellele ei ole mudeli ennustused enam deterministlikud, vaid tõenäosuslikud.

Kuidas veakomponent lineaarsesse mudelisse sisse tuua?

ilma veakomponendita mudel: $y = a + bx$

Veakomponent tähendab, et y-i väärtus varieerub ümber mudeli poolt ennustatud keskväärtuse ja seda varieeruvust normaaljaotusega modelleerides saame

$y \sim dnorm(mu, sigma)$

kus *mu* on mudeli poolt ennustatud keskväärtus ja *sigma* on mudeli poolt ennustatud standardhälve ehk varieeruvus andmepunktide tasemel. Tilde ~ tähistab seose tõenäosuslikkust.

Sirge mudelisse varieeruvuse sisse toomiseks defineerime mu ümber nõnda:

$mu = a + bx$, mis tähendab, et

y ~ dnorm(a + bx, sigma)

See ongi sirge mudel koos veakomponendiga. Peatükis 3 õpime me selliste mudelitega töötama.

Kõik statistilised mudelid on tõenäosusmudelid ning sisaldavad veakomponenti.

8.0.7 Statistiline mudel koosneb 3 komponendist:

```
> (1) matemaatiline struktuur, mis sisaldab muutujaid ja annab mudeli tüübi,

> (2) tuunitavad parameetrid ja

> (3) veamudel.
```

Muide, kõik veamudelid, millega me edaspidi töötame, modelleerivad igale x-i väärtusele (kaalule) sama suure y-i suunalise varieeruvuse (pikkuste sd). Suurem osa statistikast kasutab eeldusi, mida keegi päriselt tõe pähe ei võta, aga millega on arvutuslikus mõttes lihtsam elada.

8.0.7.1 Enimkasutatud veamudel on normaaljaotus.

Oletame, et meil on kolm andmepunkti ning me usume, et need andmed on juhuslikult tõmmatud normaaljaotusest või sellele lähedasest jaotusest. Normaaljaotuse mudelit kasutades me sisuliselt deklareerime, et me usume, et kui me oleksime olnud vähem laisad ja 3 mõõtmise asemel sooritanuks 3000, siis need mõõtmised sobituksid piisavalt hästi meie 3 väärtuse peal fititud normaaljaotusega. Seega, me usume, et omades 3 andmepunkti me teame juba umbkaudu, millised tulemused me oleksime saanud korjates näiteks 3 miljonit andmepunkti. Oma mudelist võime simuleerida ükskõik kui palju andmepunkte.

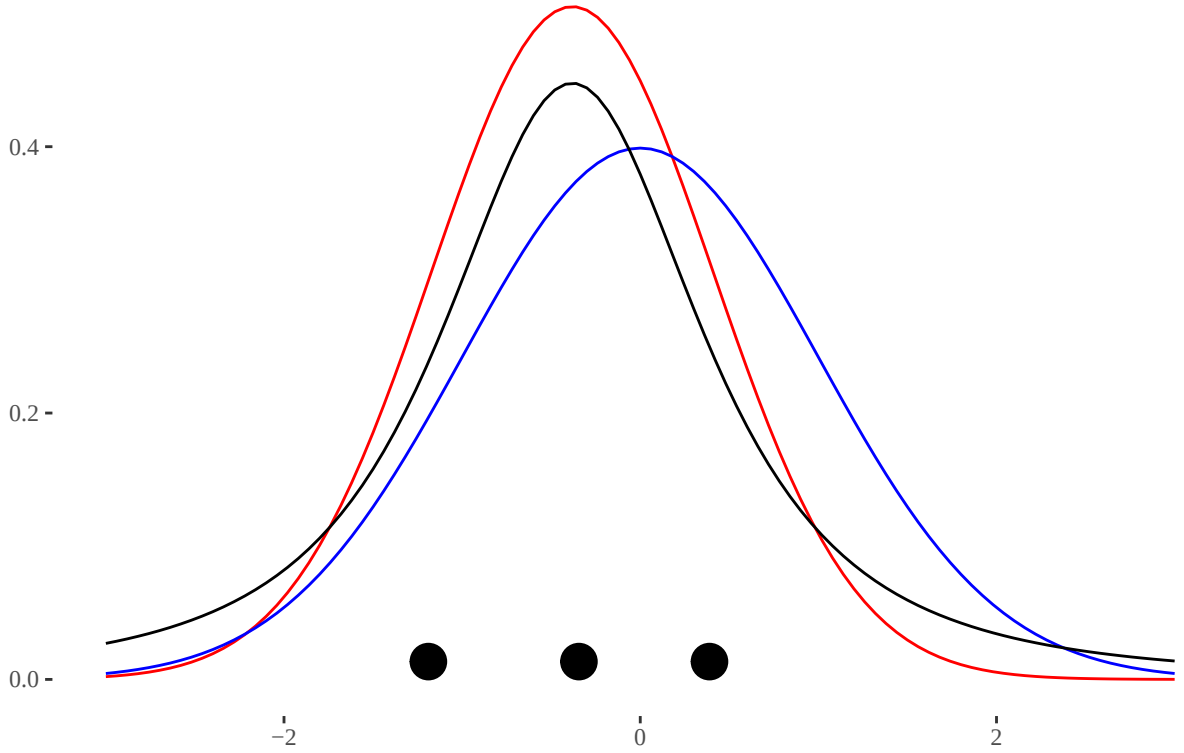
Aga pidage meeles, et selle mudeli fittimiseks kasutame me ainult neid andmeid, mis meil päriselt on — ja kui meil on ainult 3 andmepunkti, on tõenäoline, et fititud mudel ei kajasta hästi tegelikkust.

Halvad andmed ei anna kunagi head tulemust.

Eelnev ei kehti Bayesi mudelite kohta, mis toovad priorite kaudu sisse lisainfot, mis ei kajastu valimiandmetes ja võib analüüsi päästa.

Kuidas panna skeptik uskuma, et statistilised meetodid töötavad halvasti väikestel valimitel? Siin aitab simulatsioon, kus me tõmbame 3-se valimi etteantud populatsioonist ning üritame selle valimi põhjal ennustada selleasama populatsiooni struktuuri. Kuna tegemist on simulatsiooniga, teame täpselt, et populatsioon, kust me tõmbame oma kolmese valimi, on normaaljaotusega, et tema keskväärtus = 0 ja et tema sd = 1. Me fitime oma valimi andmetega 2 erinevat mudelit: normaaljaotuse ja Studenti t jaotuse.

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



Joonis: juhuvalim normaaljaotusest, mille keskmine=0 ja sd=1 (n=3; andmepunktid on näidatud mustade munadena). Sinine joon - popualtsioon, millest tõmmati valim; punane joon - normaaljaotuse mudel, mis on fititud valimi andmetel; must joon - Studenti t jaotuse mudel, mis on fititud samade andmetega.

Mõlemad mudelid on süstemaatiliselt nihutatud väiksemate väärtuste poole ja alahindavad varieeruvust. t jaotuse mudel on oodatult paksemate sabadega ja ennustab 0-st kaugele palju rohkem väärtusi kui normaaljaotuse mudel. Kuna me teame, et populatsioon on normaaljaotusega, pole väga üllatav, et t jaotus modeleerib seda halvemini kui normaaljaotus.

Igal juhul, mõni teine juhuvalim annaks meile hoopis teistsugused mudelid, mis rohkem või vähem erinevad algsest populatsioonist.

Mis juhtub kui me kasutame oma normaaljaotuse mudelit uute andmete simuleerimiseks? Kui lähedased on need simuleeritud andmed populatsiooni andmetega ja kui lähedased valimi andmetega, millega me normaaljaotuse mudeli fittisime?

```
set.seed(19) #muudab simulatsiooni korratavaks
#tõmbame 3 juhuslikku arvu normaalhaotusest, mille keskväärtus = 0 ja sd = 1.
df <- tibble(sample_data=rnorm(3))
#fitime normaaljaotuse mudeli valimi keskmise ja sd-ga
mean(df$sample_data); sd(df$sample_data)
```

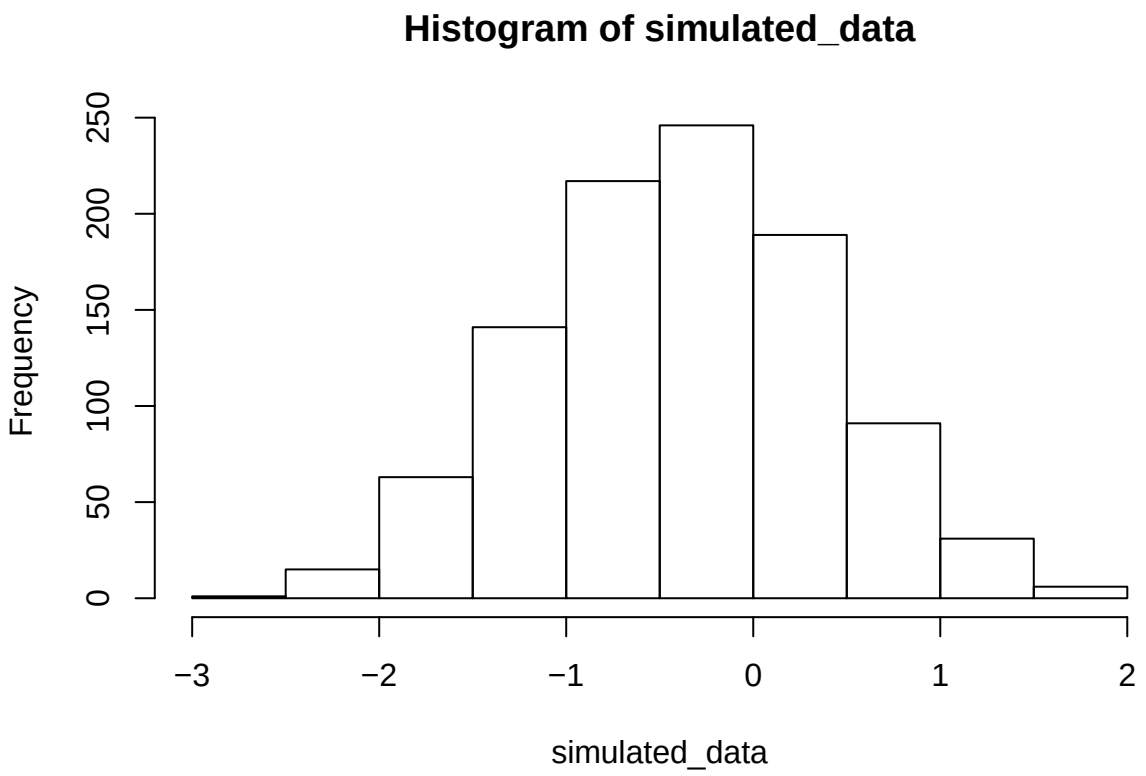
```
## [1] -0.3817353
```

```
## [1] 0.7896821
```

```
#simuleerime 1000 uut andmepunkti fititud mudelist
simulated_data <- rnorm(1000, mean(df$sample_data), sd(df$sample_data))
#arvutame simuleeritud andmete keskmise ja sd ning joonistame neist histogrammi
mean(simulated_data); sd(simulated_data); hist(simulated_data)
```

```
## [1] -0.3848133
```

```
## [1] 0.7749198
```



Nagu näha, on uute (simuleeritud) andmete keskväärtus ja SD väga sarnased algsete andmete omale, mida kasutasime mudeli fittimisel. Kahjuks ei ole need aga kaugeltki nii sarnased algsele jaotusele, mille kuju me püüame oma andmete ja mudeli pealt ennustada. Seega on meie mudel üle-fittitud, mis tähendab, et ta kajastab liigselt neid valimi aspekte, mis ei peegelda algse populatsiooni omadusi. Loomulikult ei vasta ükski mudel päriselt tegelikkusele. Küsimus on pigem selles, kas mõni meie mudelitest on piisavalt hea, et olla kasulik. Vastus sellele sõltub, milleks plaanime oma mudelit kasutada.

```
mean(simulated_data>0); mean(simulated_data>1)
```

```
## [1] 0.317
```

```
## [1] 0.037
```

Kui populatsioonivääртustest on 50% suuremad kui 0, siis mudeli järgi vaevalt 32%. Kui populatsioonivääртustest on 16% suuremad kui 1, siis mudeli järgi vaevalt 4%. See illustreerib hästi mudeli kvaliteeti.

```
library(brms)
sim_t <- rstudent_t(1000, 2, mean(df$sample_data), sd(df$sample_data))
mean(sim_t>0); mean(sim_t>1)
```

```
## [1] 0.338
```

```
## [1] 0.11
```

Samad ennustused t jaotusest on isegi paremad! Aga kumb on ikkagi parem mudel populatsioonile?

8.0.8 normaaljaotuse ja lognormaaljaotuse erilisus

Normaaljaotus ja lognormaaljaotus on erilised sest

(1) keskne piirteoreem ütleb, et olgu teie valim ükskõik millise jaotusega, paljudest valimitest arvutatud **aritmeetilised keskmised** on alati enam-vähem normaaljaotusega (kui n>30). Selle matemaatilise formalismi tuletus füüsikalisse maailma on nn “elementaarsete vigade hüpotees”, mille kohaselt paljude

väikeste üksteisest sõltumatute juhuslike efektide (vigade) summa annab tulemuseks normaaljaotuse. Paraku annavad enamus bioloogilisi mõõtmisi eranditult mitte-negatiivseid tulemusi. Sageli on selliste mõõtmiste tulemuste jaotused ebasümmeetrilised (v.a. siis, kui cv = sd/mean on väike) ja siis on meil sageli tegu lognormaaljaotusega, mis tekkib log-normaalsete muutujate korrutamisest (mitte liitmisest, nagu normaaljaotuse puhul). Keskne piirteoreem 2: suvalise jaotusega muutujate **geomeetrilised keskmised** on lognormaaljaotusega. Elementaarsete vigade hüpotees 2: Kui juhuslik varieeruvus tekib paljude juhuslike efektide korrutamisel, on tulemuseks lognormaaljaotus. Lognormaaljaotuse elementide (arvude) logaritmimisel saame normaaljaotuse.

(2) Mõlemad jaotused (normaal ja lognormaal) on maksimaalse entroopiaga jaotused. Entroopiat vaadeldakse siin informatsiooni/müra kaudu — maksimaalse entroopiaga süsteem sisaldab maksimaalselt müra ja minimaalselt informatsiooni (Shannoni informatsiooniteooria). See tähendab, et väljaspool oma parameetrite tuunitud väärtusi on need normaal- ja lognormaaljaotused minimaalselt informatiivsed. Näiteks normaaljaotusel on kaks parameetrit, mu ja sigma (ehk keskmine ja standardhälve). Seega, andes normaaljaotusele ette keskväärtuse ja standardhälbe fikseerime üheselt jaotuse ehk mudeli kuju ja samas lisame sinna minimaalselt muud (sooviamtut) informatsiooni. Teised maksimaalse entroopiaga jaotused on eksponentsiaalne jaotus, binoomjaotus ja poissoni jaotus. Maksimaalse entroopiaga jaotused sobivad hästi Bayesi prioriteks sest me suudame paremini kontrollida, millist informatsiooni me neisse surume.

8.1 Küsimused, mida statistika küsib

Statistika abil saab vastuseid järgmisetele küsimustele:

1) kuidas näevad välja teie andmed ehk milline on just teie andmete jaotus, keskväärtus, varieeruvus ja koos-varieeruvus? Näiteks, mõõdetud pikkuste ja kaalude koos-varieeruvust saab mõõta korrelatsioonikordaja abil.

2) mida me peaksime teie valimi andmete põhjal uskuma populatsiooni parameetri tegeliku väärtuse kohta? Näiteks, kui meie andmete põhjal arvutatud keskmine pikkus on 178 cm, siis kui palju on meil põhjust arvata, et tegelik populatsiooni keskmine pikkus > 185 cm?

3) mida ütleb statistilise mudeli struktuur teadusliku hüpoteesi kohta? Näiteks, kui meie poolt mõõdetud pikkuste ja kaalude koos-varieeruvust saab hästi kirjeldada kindlat tüüpi lineaarse regressioonimudeliga, siis on meil ehk tõendusmaterjali, et pikkus ja kaal on omavahel sellisel viisil seotud ja eelistatud peaks olema teaduslik teooria, mis just sellise seose tekkimisele bioloogilise mehhanismi annab.

4) mida ennustab mudel tuleviku kohta? Näiteks, meie lineaarne pikkuse-kaalu mudel suudab ennustada tulevikus kogutavaid pikkuse andmeid. Aga kui hästi?

statistika ülesanne on lähtuvalt piiratud hulgast andmetest ja mudelitest kvantifitseerida parimal võimalikul viisil kõhedust, mida peaksime tundma vastates eeltoodud küsimustele.

Statistika ei vasta otse teaduslikele küsimustele ega küsimustele päris maailma kohta. Statistilised vastused jäävad alati kasutatud andmete ja mudelite piiridesse. Sellega seoses peaksime eelistama hästi kogutud rikkalikke andmeid ja paindlikke mudeleid. Siis on lootust, et hüpe mudeli koefitsientidest päris maailma kirjeldamisse tuleb üle kitsama kuristiku. Bayesil on siin eelis, sest osav statistik suudab koostöös teadlastega priori mudelisse küllalt palju kasulikku infot koguda. Samas, amatöör suudab bayesi abil samavõrra käkki keerata. Mida paindlikum on meetod, seda vähem automaatne on selle mõistlik kasutamine.

Chapter 9

2 osa. Kuidas näevad välja teie andmed?

9.1 summaarsed statistikud

Summaarne statistik = üks number.
Milliseid statistikuid arvutada ja milliseid vältida, sõltub statistilisest mudelist

> summaarse statistika abil iseloomustame a) tüüpilist valimi liiget (keskmist), b) muutuja sisest varieeruvust, c) erinevate muutujate (pikkus, kaal vms) koos-varieeruvust

9.1.1 keskväärtused

Keskväärtust saab mõõta paaril tosinal erineval viisil, millest järgnevalt kasutame kolme või nelja. Enne kui te arvutama kukute, mõelge järele, miks te soovite keskväärtust teada. Kas teid huvitab valimi tüüpiline liige? Kuidas te sooviksite seda tüüpilisust defineerida? Kas valimi keskmise liikmena või valimi kõige arvukama liikmena? või veel kuidagi? See, millist keskväärtust kasutada sõltub sageli andmejaotuse kujust. Sümmeetrilisi jaotusi on lihtsam iseloomustada ja mitmetipulised jaotused on selles osas kõige kehvemad.

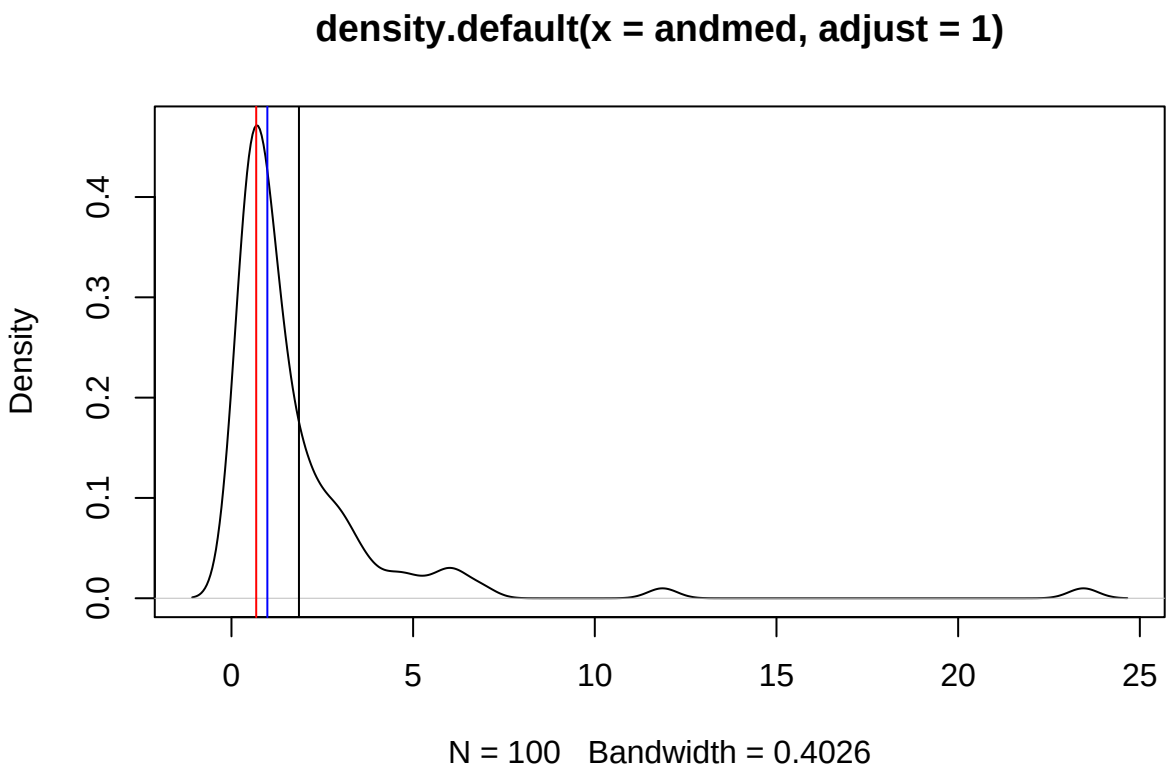
Mina eelistan selliseid nõuandeid (mis on rangelt soovituslikud):

(1) Kui valim on normaaljaotusega (histogramm on sümmeetriline), hinda tüüpilist liiget läbi aritmeetilise keskmise (mean).

(2) Muidu kasuta mediaani (median). Kui valim on liiga väike, et jaotust hinnata (aga > 4), eelista mediaani. Mediaani saamiseks järjestatakse mõõdetud väärtused suuruse järgi ja võetakse selle rea keskmine liige. Mediaan on vähem tundlik ekstreemsete väärtuste (outlierite) suhtes kui mean.

(3) Valimi kõige levinumat esindajat iseloomustab mood ehk jaotuse tipp. Seda on aga raskem täpselt määrata ja mitmetipulisel jaotusel on mitu moodi. Töötamisel posterioorsete jaotustega on mood sageli parim lahendus.

```
## [1] 0.6817168
```



Joonis: Simuleeritud lognormaaljaotusega andmed. Punane joon - mood; sinine joon - mediaan; must joon - aritmeetiline keskmine (mean). Milline neist vastab parimini teie intuitsiooniga nende andmete "keskväärtusest"? Miks?

9.1.2 muutuja sisene varieeruvus

Mean-iga käib kokku standardhälve (SD).

SD on sama ühikuga, mis andmed (ja andmete keskväärtus). Statistikute hulgas eelistatud formaat on mean (SD), mitte mean (+/- SD). 1 SD katab 68% normaaljaotusest, 2 SD – 96% ja 3 SD – 99%. Normaaljaotus langeb servades kiiresti, mis tähendab, et tal on peenikesed sabad ja näiteks 5 SD kaugusel keskmisest paikneb vaid üks punkt miljonist.

Näiteks: inimeste IQ on normaaljaotusega, mean=100, sd=15. See tähendab, et kui sinu IQ=115 (ülikooli astujate keskmine IQ), siis on tõenäosus, et juhuslikult kohatud inimene on sinust nutikam, 18% ((100% - 68%)/2 = 18%).

Kui aga "tegelikul" andmejaotusel on "paks saba" või esinevad outlierid, siis normaaljaotust eeldav mudel tagab ülehinnatud SD ja seega ülehinnatud varieeruvuse. Kui andmed saavad olla ainult positiivsed, siis SD > mean/2 viitab, et andmed ei sobi normaaljaotuse mudeliga (sest mudel ennustab negatiivsete andmete esinemist küllalt suure sagedusega).

```
Normaaljaotus on defineeritud ka mõnede teiste jaotuste jaoks peale normaaljaotuse
(Poissioni jaotus, binoomjaotus).
Funktsioon sd() ja selle taga olev valem, mis on loodud normaaljaotuse tarbeks,
ja neid alternatiivseid standardhälbeid ei arvuta.
Seega tasub meeles pidada, et tavapärane sd kehtib normaaljaotuse mudeli piirides ja ei kusagil mujal!
```

Kui andmed ei sobi normaaljaotusesse siis võib pakkuda kahte alternatiivset lahendust:

9.1.2.1 (1) logaritmi andmed.

Kui logaritmimine muudab andmed normaalseks, siis saab logaritmitud andmetest arvutada mean-i ja SD ja seejärel mõlemad anti-logaritmida. Sellisel juhul avaldad sa lõpuks geomeetrilise keskmise ja multiplikatiivse SD (multiplikatiivne SD = geom mean x SD; geom mean/SD). Geomeetriline keskmine on alati väiksem kui aritmeetiline keskmine. Lisaks on SD interval nüüd asümmeetriline ja SD on alati > 0. Nagu ennegi, 68% lognormaalsetest andmetest jääb 1SD vahemikku ning 95.5% andmetest jääb 2SD vahemikku.

lognormaalsete andmete tavapärane iseloomustus keskmise ja standardhälbega: mean(sd) on 1.8(1.9). See sd interval on sümmeetriline, ehkki andmete jaotus on vägagi ebasümmeetriline. Lisaks ennustab standardhälve, mis on suurem kui keskväärtus, suure sagedusega negatiivseid väärtusi. Sageli on aga negatiivsed muutuja väärtused võimatud (näiteks nädalas suitsetatud sigarettide arv). See on näide halvast mudelist!

Juhul kui tegu lognormaalsete andmetega on meil võimalus kasutada palju paremat mudelit varieeruvusele - multiplikatiivset standardhälvet:

```
## # A tibble: 4 x 4
##                   SD     MEAN      lower    upper
##                <chr>    <dbl>      <dbl>    <dbl>
## 1  multiplicative_SD 1.084891  0.4010893 2.934481
## 2 multiplicative_2SD 1.084891  0.1482845 7.937367
## 3        additive_SD 1.857924 -0.9636351 4.679482
## 4       additive_2SD 1.857924 -3.7851938 7.501041
```

Tavalise aritmeetitilise keskmise asemel on meil nüüd geomeetriline keskmine. Võrdluseks on antud ka tavaline (aritmeetiline) keskmine ja (aditiivne) SD. Additiivne SD on selle jaotuse kirjeldamiseks selgelt ebaadekvaatne (vt jaotuse pilti ülalpool ja võrdle mulitplikatiivse SD-ga).

Kuidas aga töötab multiplikatiivne standardhälve normaaljaotusest pärit andmetega? Kui normaalsete andmete peal multiplikatiivse sd rakendamine viib katastroofini, siis pole sel statistikul suurt praktilist kasutusruumi.

```
set.seed(5363)
norm_andmed <- rnorm(3, 100, 20)
multiplicative_sd(norm_andmed)
```

```
## # A tibble: 4 x 4
##                   SD     MEAN    lower    upper
##                <chr>    <dbl>    <dbl>    <dbl>
## 1  multiplicative_SD 108.1088 92.80205 125.9403
## 2 multiplicative_2SD 108.1088 79.66252 146.7128
## 3        additive_SD 108.9603 92.08395 125.8367
## 4       additive_2SD 108.9603 75.20756 142.7131
```

Nagu näha, on multiplikatiivse sd kasutamine normaalsete andmetega üsna ohutu (kuigi, ainult niikaua, kuni meil puuduvad negatiivsed andmed). Seega, kui sa ei ole kindel, kas tegu on normaaljaotusega või lognormaaljaotusega, arvesta, et lognormaaljaotus on bioloogias üsna tavaline (eriti ensüümreaktsioonide ja kasvuprotsesside juures). Seega on mõistlik alati kasutada multiplicative_sd() funktsiooni ja kui mõlema SD väärtused on sarnased, siis võib loota, et andmed on normaalsed ning saab avaldada tavapärase additiivse SD refereede rõõmuks.

> kui n < 10, siis mõlemad SD-d alahindavad tehnilistel põhjustel tegelikku sd-d. Ettevaatust väikeste valimitega!

9.1.2.2 (2) iseloomusta andmeid algses skaalas: median (MAD).

MAD —- median absolute deviation — on vähem tundlik outlierite suhtes ja ei eelda normaaljaotust. Puuduseks on, et MAD ei oma tõlgendust, mille kohaselt ta hõlmaks kindlat protsenti populatsiooni või valimi andmejaotusest. Seevastu sd puhul võime olla kindlad, et isegi kõige hullema jaotuse korral jäävad vähemalt 75% andmetest 2 SD piiridesse.

```
mad(andmed, constant = 1)
```

```
## [1] 0.5950562
```

```
Ära kunagi avalda andmeid vormis: mean (MAD) või median (SD).
Korrektne vorm on mean(SD) või median(MAD).
```

9.1.3 muutujate koos-varieeruvus

Andmete koos-varieeruvust mõõdetakse korrelatsiooni abil. Tulemuseks on üks number - korrelatsioonikordaja r, mis varieerub -1 ja 1 vahel.

```
r = 0 - kahte tüüpi mõõtmised (x=pikkus, y=kaal) samadest mõõteobjektidest varieeruvad üksteisest sõltumatul
r = 1: kui ühe muutuja väärtus kasvab, kasvab ka teise muutuja väärtus alati täpselt samas proportsioonis.
r = -1: kui ühe muutuja väärtus kasvab, kahaneb teise muutuja väärtus alati täpselt samas proportsioonis.
```

Kui r on -1 või 1, saame me x väärtust teades täpselt ennustada y väärtuse (ja vastupidi, teades y väärrust saame täpselt ennustada x väärtuse).
Kuidas tõlgendame aga tulemust r = 0.9? Mitte kuidagi. Selle asemel tõlgendame r2 = 0.9**2 = 0.81 – mis tähendab, et x-i varieeruvus suudab seletada 81% y varieeruvusest ja vastupidi, et Y-i varieeruvus suudab seletada 81% X-i varieeruvusest.

```
#correlation<-cor.test(iris$Sepal.Length, iris$Sepal.Width, na.rm=T, method = "pearson") # a list of 9

#names(correlation)
#str(correlation)
#correlation$conf.int
cor(iris$Sepal.Length, iris$Sepal.Width, use="complete.obs")
```

```
## [1] -0.1175698
```

Korrelatsioonikordaja väärtus sõltub mitte ainult andmete koos-varieeruvusest vaid ka andmete ulatusest. Suurema ulatusega andmed X ja/või Y teljel annavad keskeltläbi 0-st kaugemal oleva korrelatsioonikordaja. Selle pärast sobib korrelatsioon halvasti näiteks korduskatsete kooskõla mõõtmiseks.

Lisaks, korrelatsioonikordaja mõõdab vaid andmete *lineaarset* koos-varieeruvust: kui andmed koos-varieeruvad mitte-lineaarselt, siis võivad ka väga tugevad koos-varieeruvused jääda märkamatuks.

> Kõik summaarsed statistikud kaotavad enamuse teie andmetes leiduvast infost – see kaotus on õigustatud ainult siis, kui teie poolt valitud statistik iseloomustab hästi andmete sügavamat olemust (näiteks tüüpilist mõõtmistulemust või andmete varieeruvust).

```
#Kuidas arvutada correlatsioonimaatriksit koos adjusteeritud p väärtustega
#numeric columns only!
print(psych::corr.test(iris[-5], use="complete"), short = FALSE)
```

9.2 2.2 EDA — eksploratoorne andmeanalüüs

Kui ühenumbriline andmete summeerimine täidab eelkõige kokkuvõtliku kommunikatsiooni eesmärki, siis EDA on suunatud teadlasele endale. EDA eesmärk on andmeid eelkõige graafiliselt vaadata, et saada aimu 1) andmete kvaliteedist ja 2) lasta andmetel "sellisena nagu nad on" kõneleda ja sugereerida uudseid teaduslikke hüpoteese. Neid hüpoteese peaks siis testima formaalse statistilise analüüsi abil.

> EDA: mida rohkem graafikuid, seda rohkem võimalusi uute mõtete tekkeks!

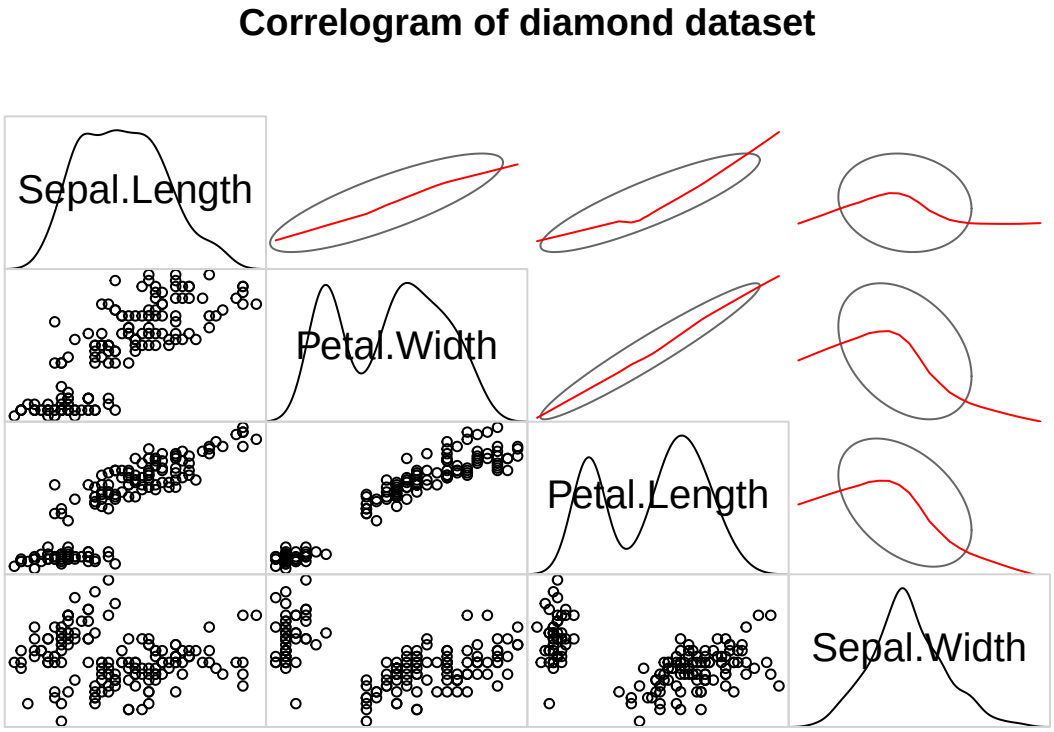
Kõigepealt vaatame andmeid numbrilise kokkuvõttena:

```
psych::describe(iris)
```

```
##              vars   n mean   sd median trimmed  mad min max range  skew
## Sepal.Length    1 150 5.84 0.83   5.80    5.81 1.04 4.3 7.9   3.6  0.31
## Sepal.Width     2 150 3.06 0.44   3.00    3.04 0.44 2.0 4.4   2.4  0.31
## Petal.Length    3 150 3.76 1.77   4.35    3.76 1.85 1.0 6.9   5.9 -0.27
## Petal.Width     4 150 1.20 0.76   1.30    1.18 1.04 0.1 2.5   2.4 -0.10
## Species*        5 150 2.00 0.82   2.00    2.00 1.48 1.0 3.0   2.0  0.00
##              kurtosis   se
## Sepal.Length    -0.61 0.07
## Sepal.Width      0.14 0.04
## Petal.Length    -1.42 0.14
## Petal.Width     -1.36 0.06
## Species*        -1.52 0.07
```

Millised korrelatsioonid võiksid andmetes esineda?

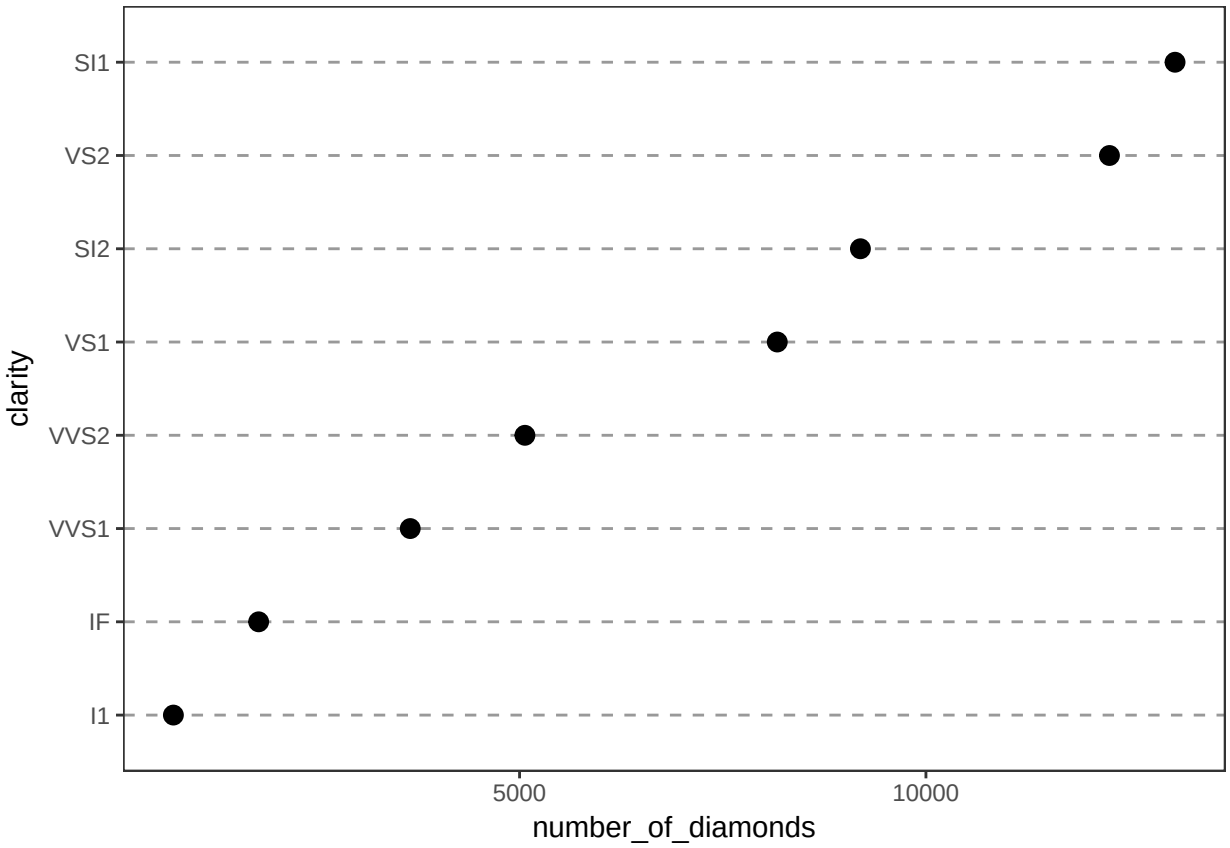
```
library(corrgram) #PCA for ordering

corrgram(iris, order=TRUE,
         lower.panel = panel.pts,
         upper.panel = panel.ellipse,
         diag.panel = panel.density,
         main="Correlogram of diamond dataset")
```



9.2.1 Kuidas uurida muutuja sisest varieeruvust

Muutuja - midagi, mida mõõdeti (näiteks mõõteobjektide kaal). Kui iga muutuja kohta on vaid üks number, mida plottida, kasuta Cleveland plotti:

```
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")
```



9.2.1.1 Ploti valik sõltub valimi suurusest.

1) N < 20 - ploti iga andmepunkt eraldi (stripchart(), plot()) ja keskmine või mediaan.

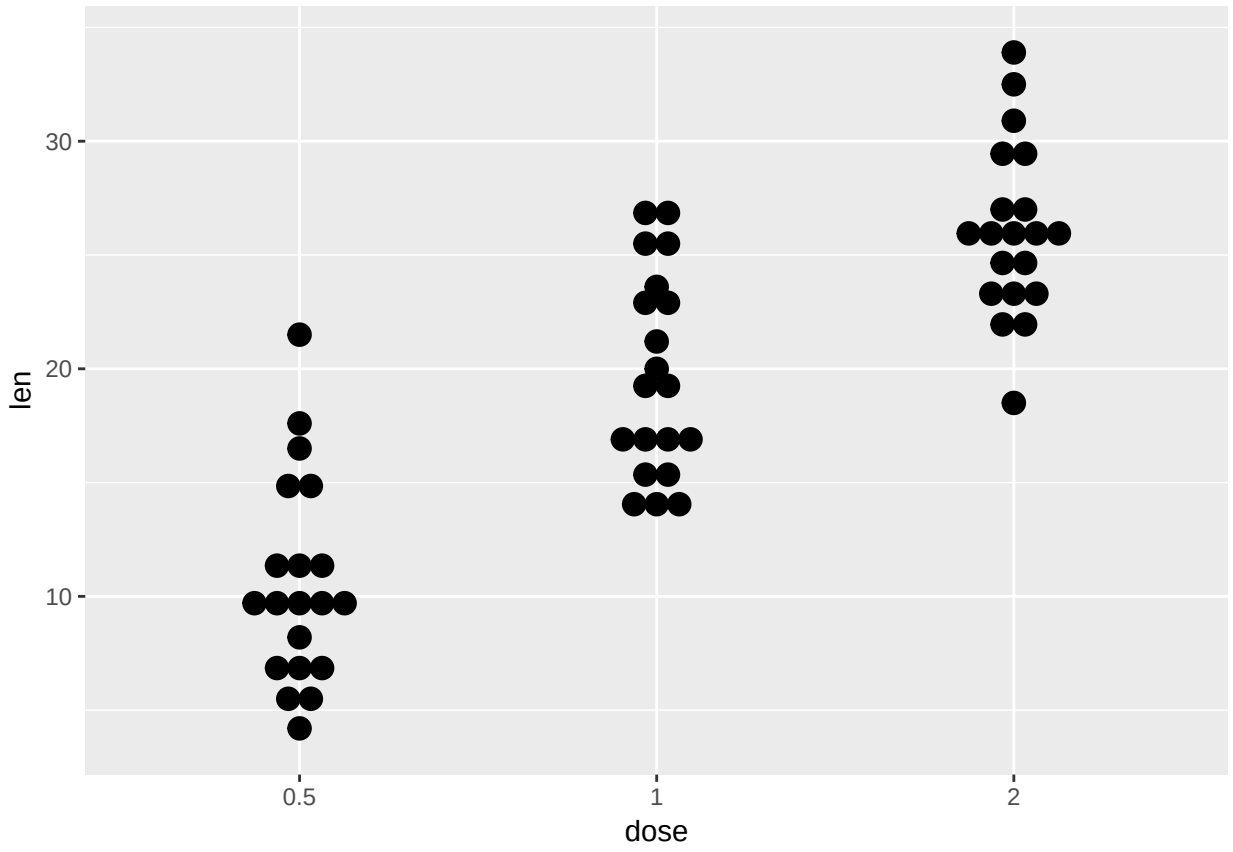
2) 20 > N > 100: geom_dotplot() histogrammi vaates

3) N > 100: geom_histogram(), geom_density() — nende abil saab ka 2 kuni 6 jaotust võrrelda

4) Mitme jaotuse kõrvuti vaatamiseks kui N > 15: geom_boxplot() or, when N > 50, geom_violin(), geom_joy()

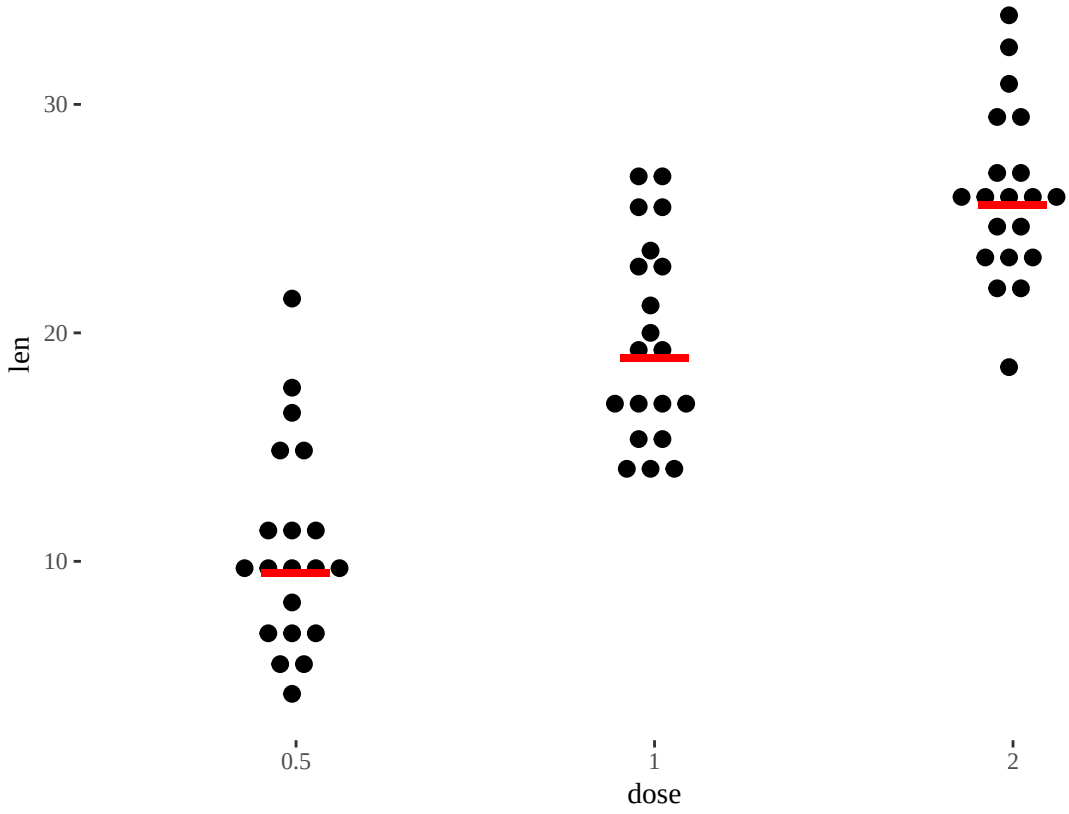
```
ToothGrowth <- ToothGrowth
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
p<-ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_dotplot(binaxis='y', stackdir='center')
p
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



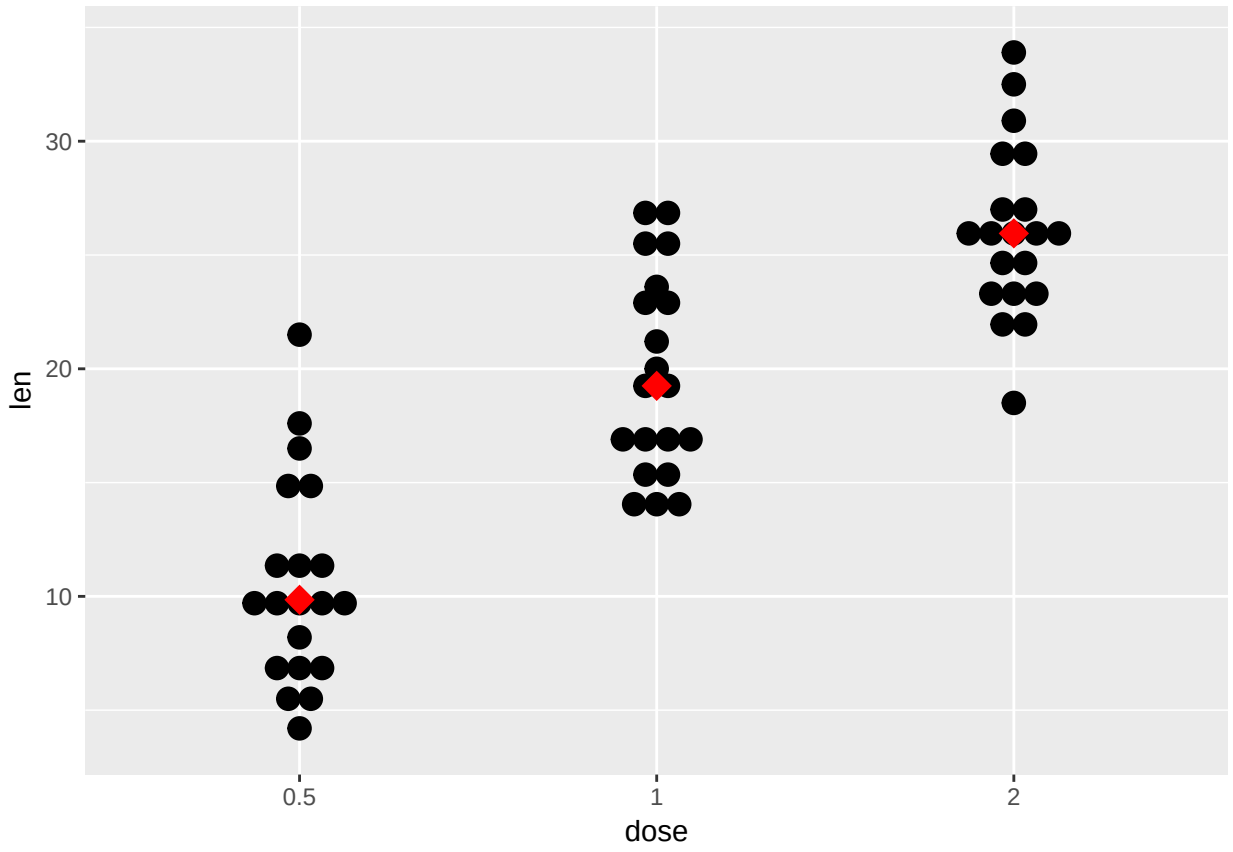

```
# Change dotsize and stack ratio, add line or dot for median
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_dotplot(binaxis='y', stackdir='center',
               stackratio=1.5, dotsize=0.7)+
  stat_summary(fun.y = median, geom = "point", shape = 95,
               color = "red", size = 15) +
  theme_tufte()
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



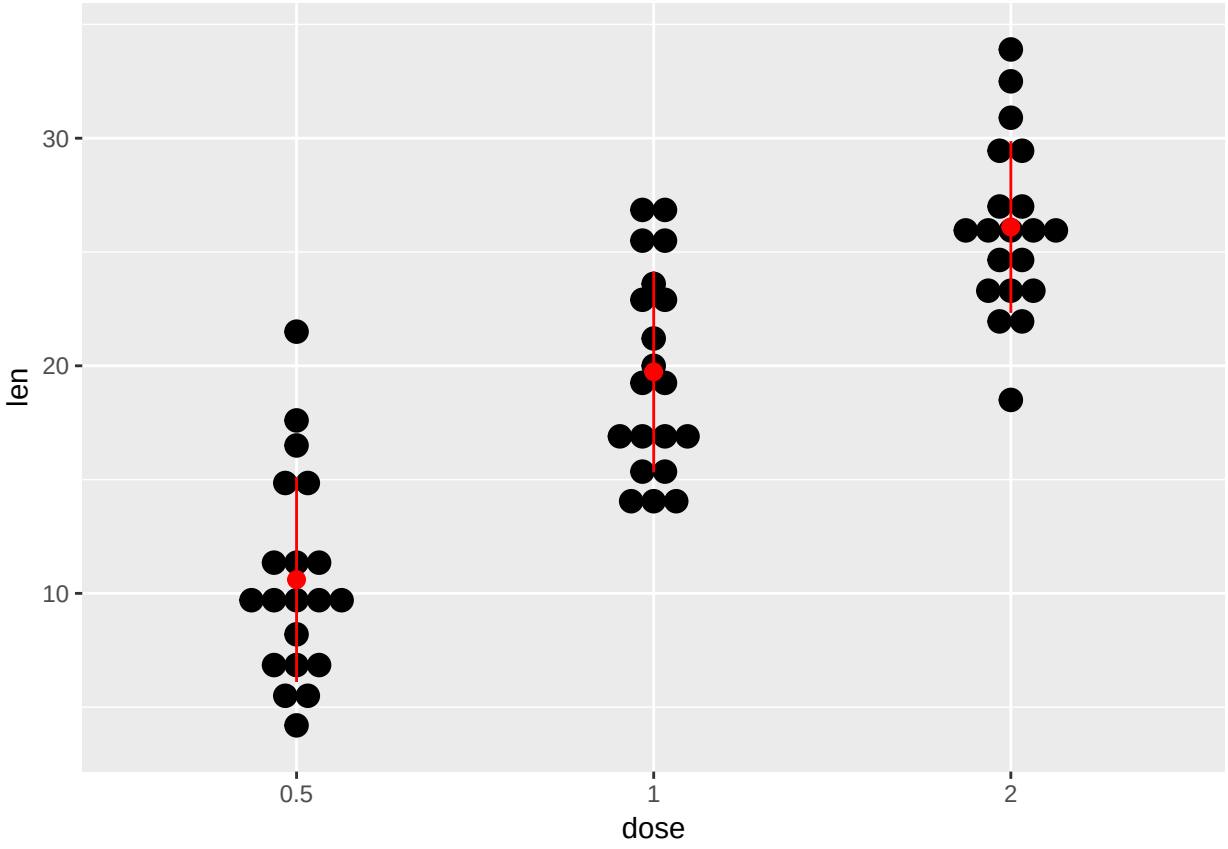

```
p + stat_summary(fun.y=median, geom="point", shape=18,
                 size=5, color="red")
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



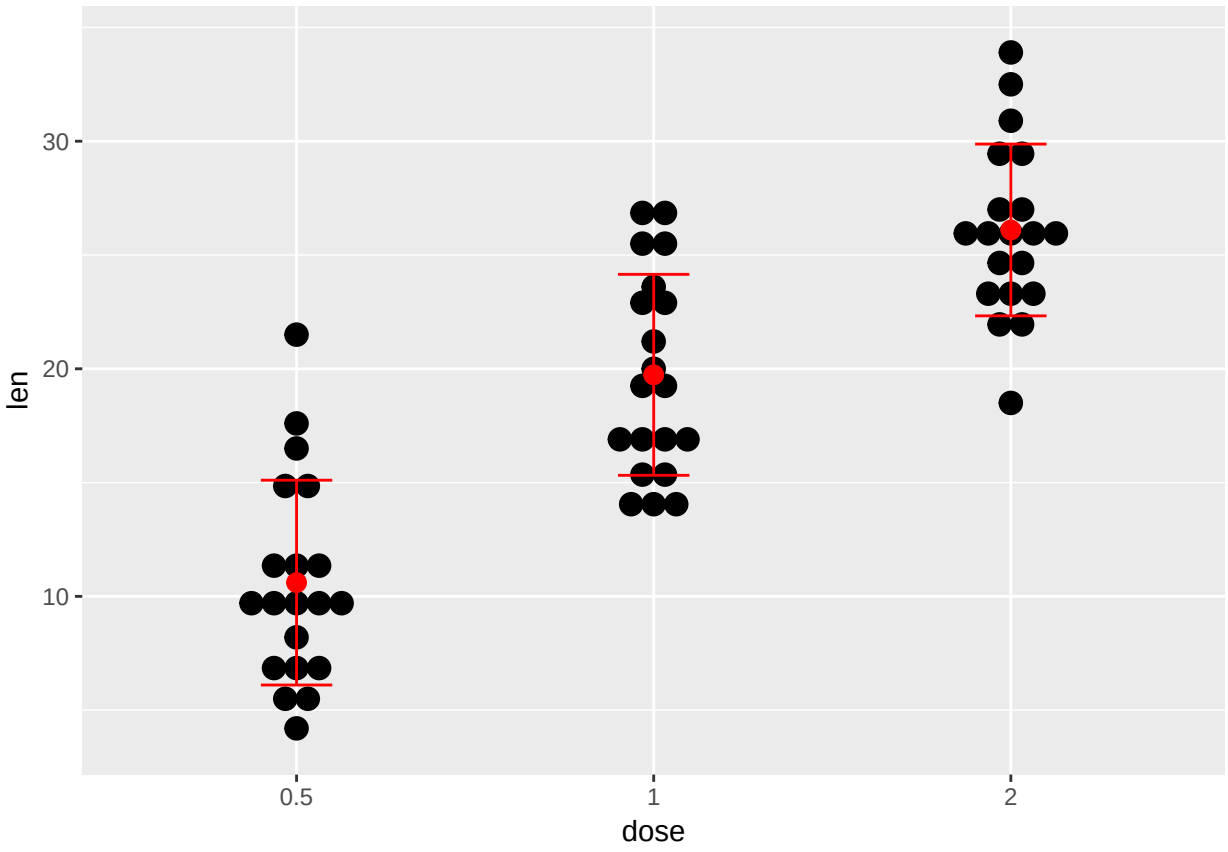

```
#add mean and SD, use pointrange
p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1),
                 geom="pointrange", color="red")
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#use errorbars
p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1),
        geom="errorbar", color="red", width=0.2) +
  stat_summary(fun.y=mean, geom="point", size=3, color="red")
```

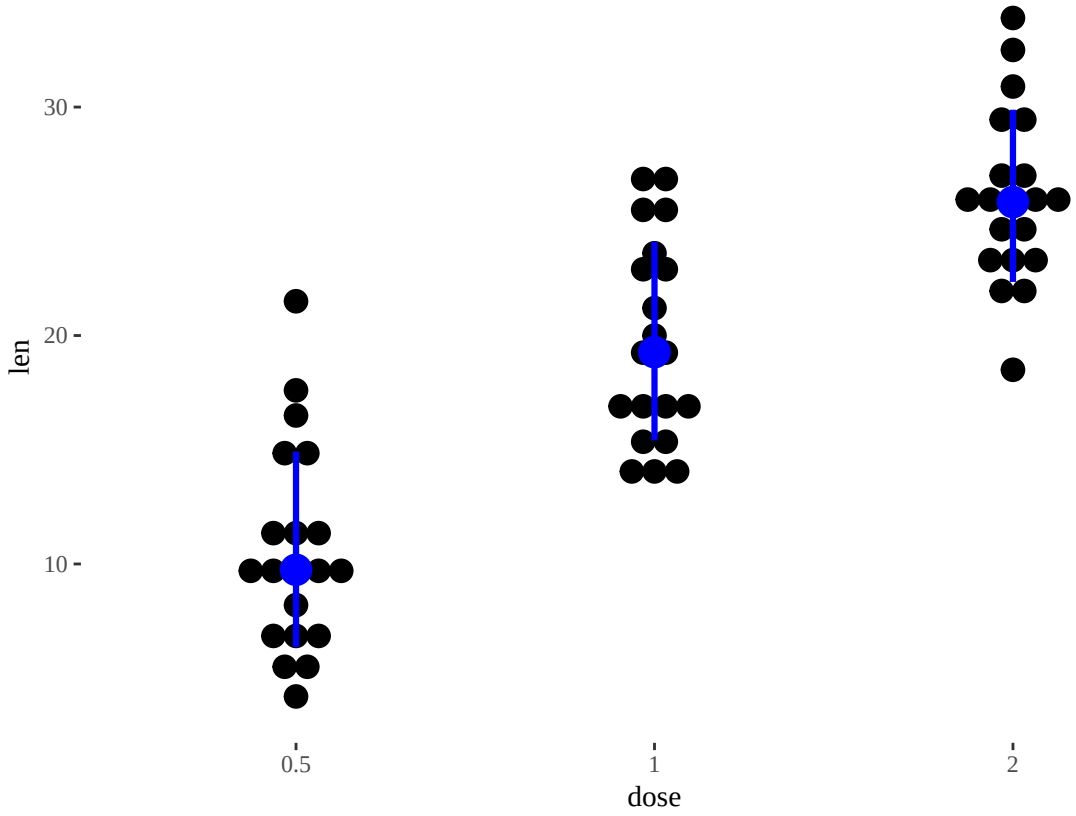
```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



Use a custom summary function :

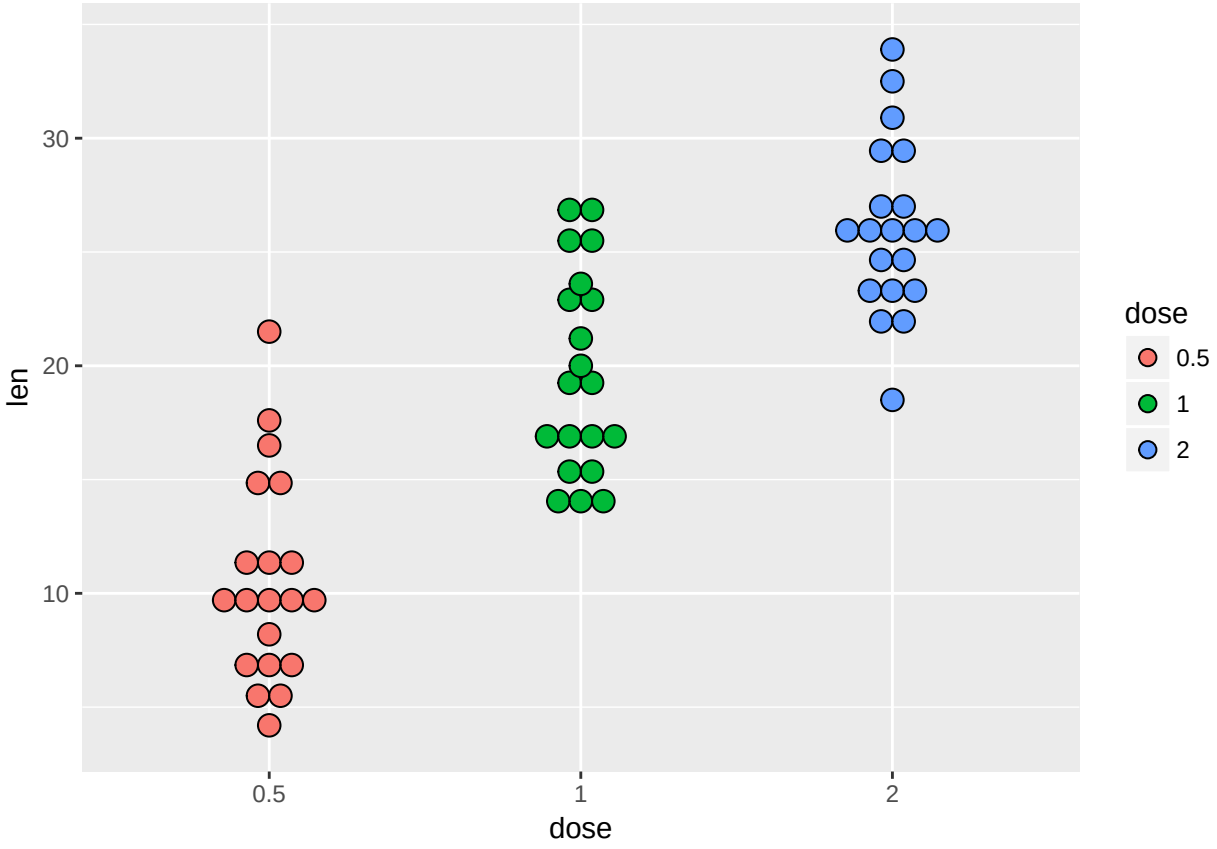
```
# Function to produce summary statistics (geometric mean and multipülicative sd)
multi_sd <- function(x) {
  x <- na.omit(x)
  a <- log10(x)
  b <- mean(a)
  c <- sd(a)
  g_mean <- 10**b
  msd <- 10**c
  ymin <- g_mean/msd
  ymax <- g_mean * msd
 return(c(y = g_mean, ymin = ymin, ymax = ymax))
}
p + stat_summary(fun.data=multi_sd, color="blue", size=1.1) + theme_tufte()
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Change dot plot colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_dotplot(binaxis='y', stackdir='center')
p
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



It is also possible to change manually dot plot colors using the functions :

scale_fill_manual() : to use custom colors

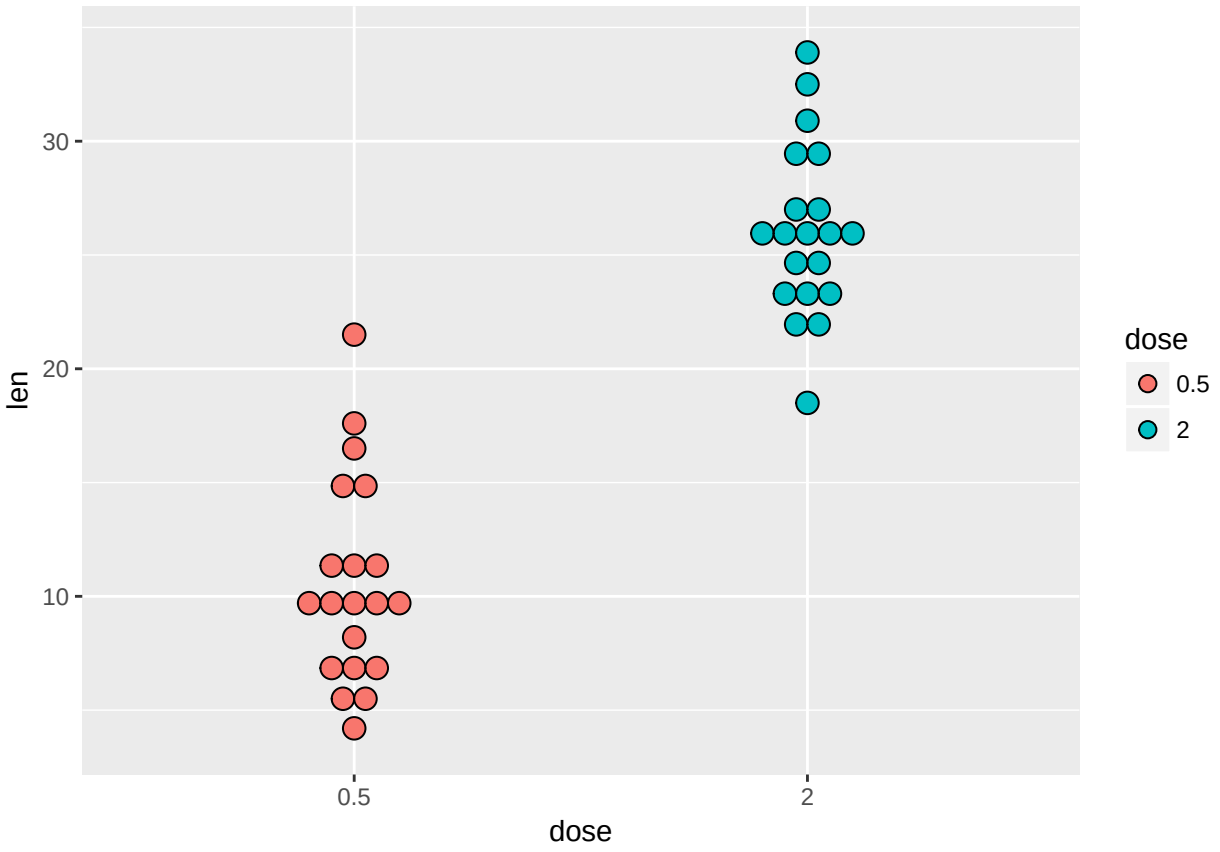
scale_fill_brewer() : to use color palettes from RColorBrewer package

scale_fill_grey() : to use grey color palettes

```
#Choose which items to display :
p + scale_x_discrete(limits=c("0.5", "2"))
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```

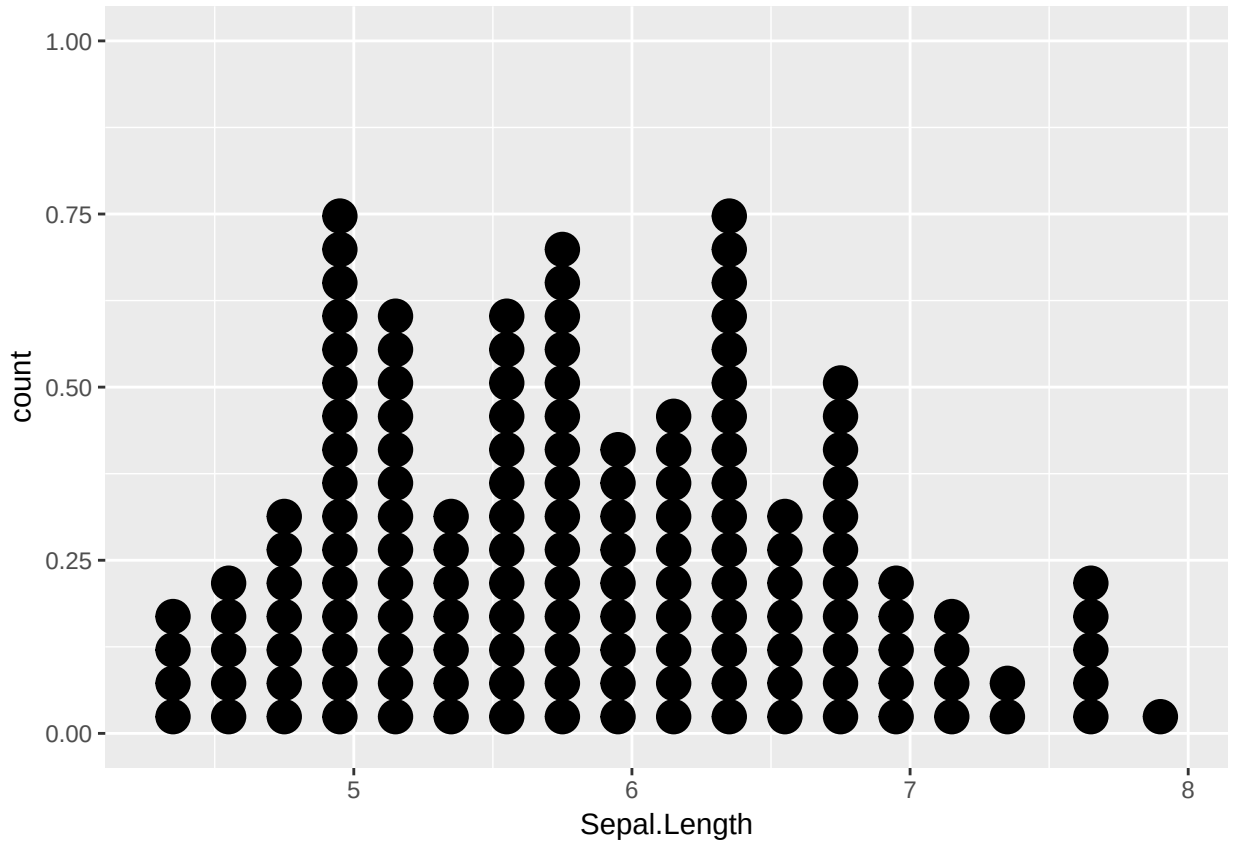
```
## Warning: Removed 20 rows containing non-finite values (stat_bindot).
```



Dotplot kui histogram:

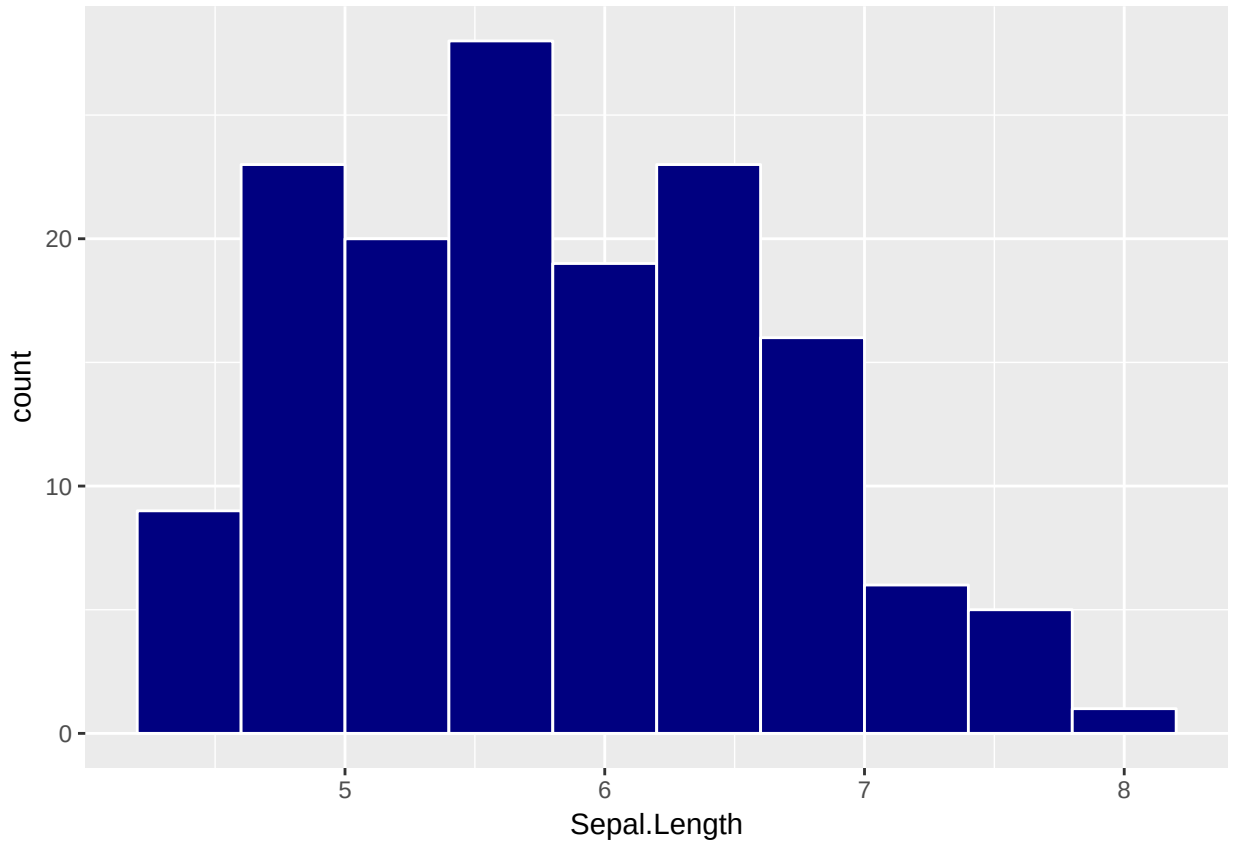
```
ggplot(iris, aes(Sepal.Length)) + geom_dotplot()
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```

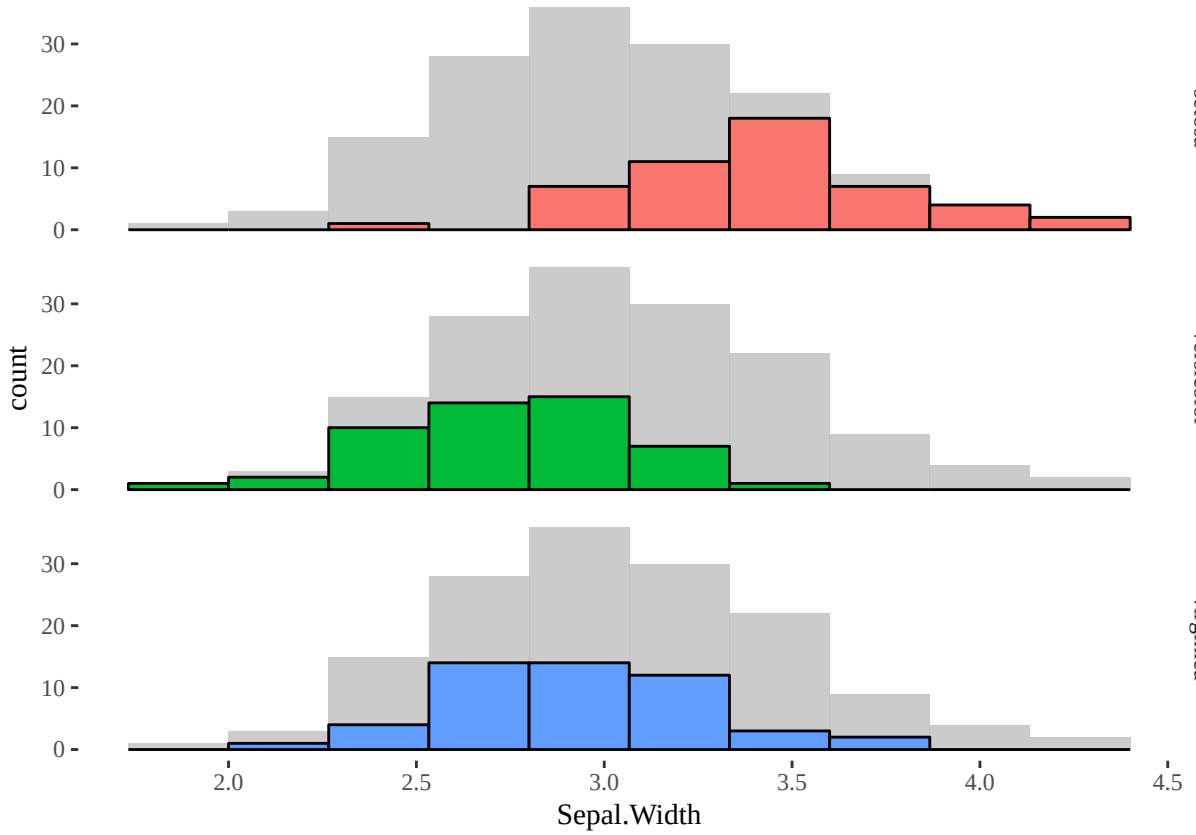


Histogram:

```
ggplot(iris, aes(Sepal.Length)) +
  geom_histogram(bins = 10, color="white", fill = "navyblue")
```

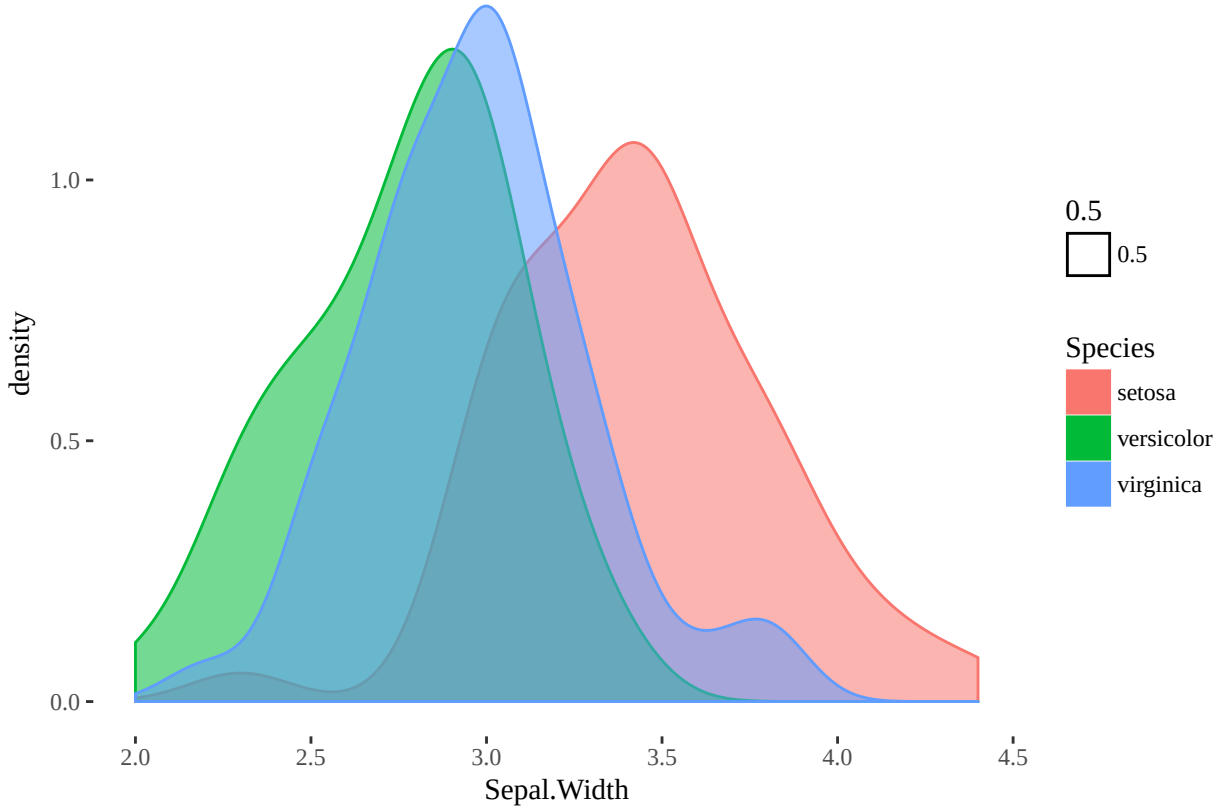



```
library(ggthemes)
d <- iris        # Full data set
d_bg <- d[, -5]  # Background Data - full without the 5th column (Species)

ggplot(data = d, aes(x = Sepal.Width, 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
```



density plot:

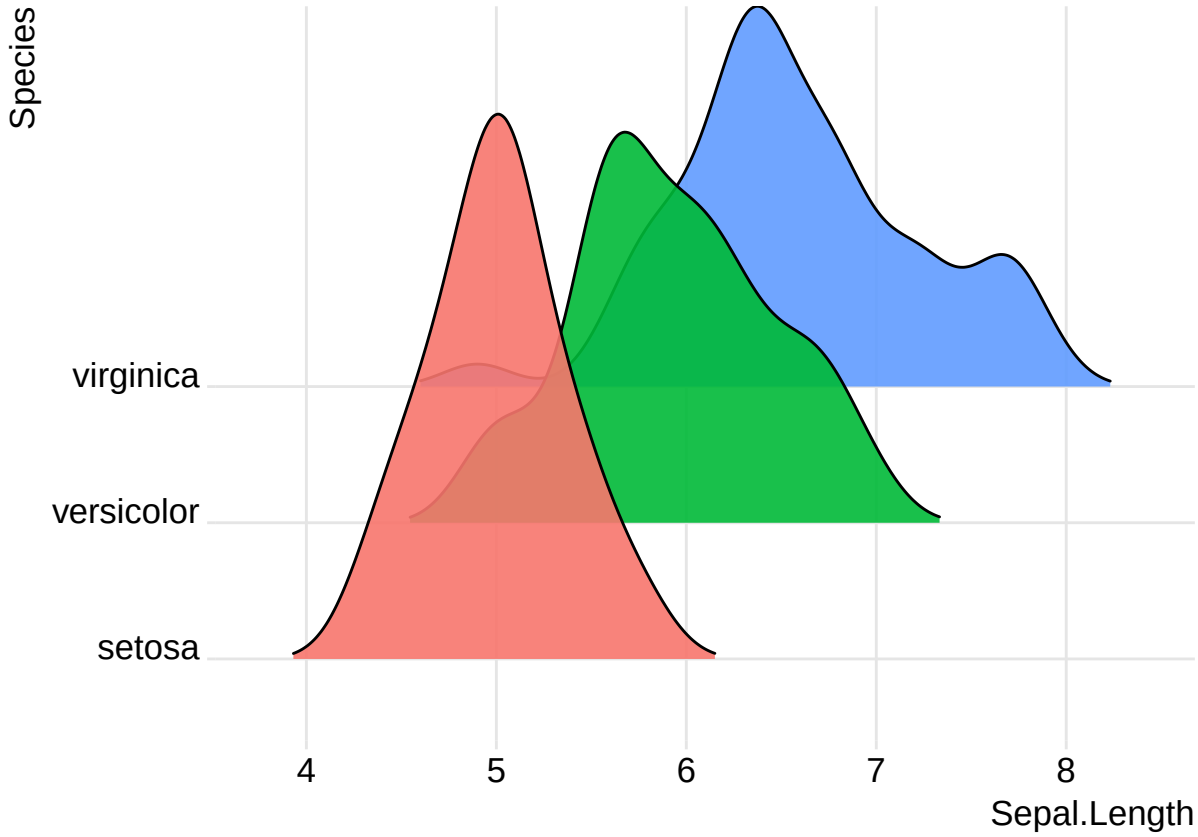
```
iris%>%ggplot()+
  geom_density(aes(Sepal.Width, fill=Species, color=Species, alpha=0.5))+
  theme_tufte()
```



joyplot võimaldab kõrvuti panna isegi sadu density plotte

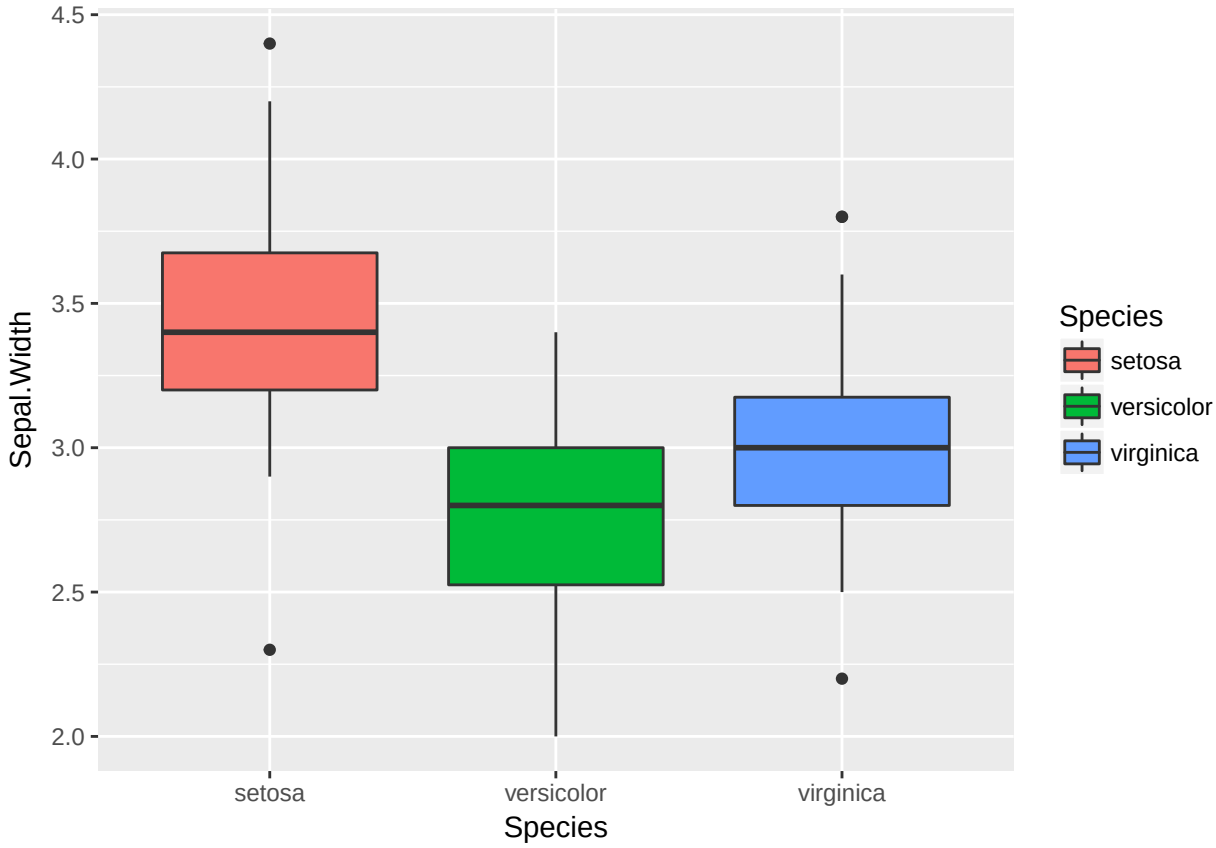
```
library(ggjoy)
ggplot(iris, aes(x=Sepal.Length, y=Species, fill=Species)) +
  geom_joy(scale=4, rel_min_height=0.01, alpha=0.9) +
  theme_joy(font_size = 13, grid=TRUE) +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 0.181
```



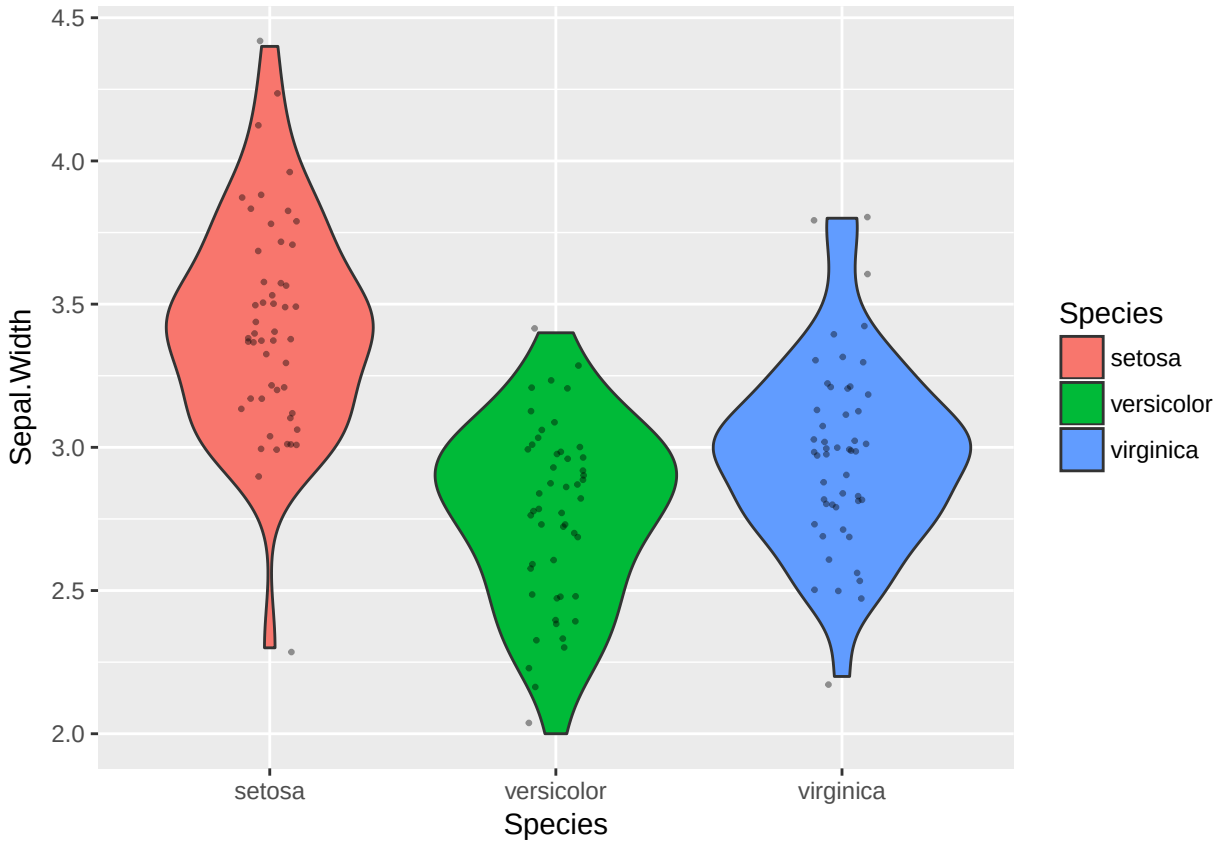
Boxplot:

```
ggplot(iris, aes(Species, Sepal.Width, fill=Species)) + geom_boxplot()
```



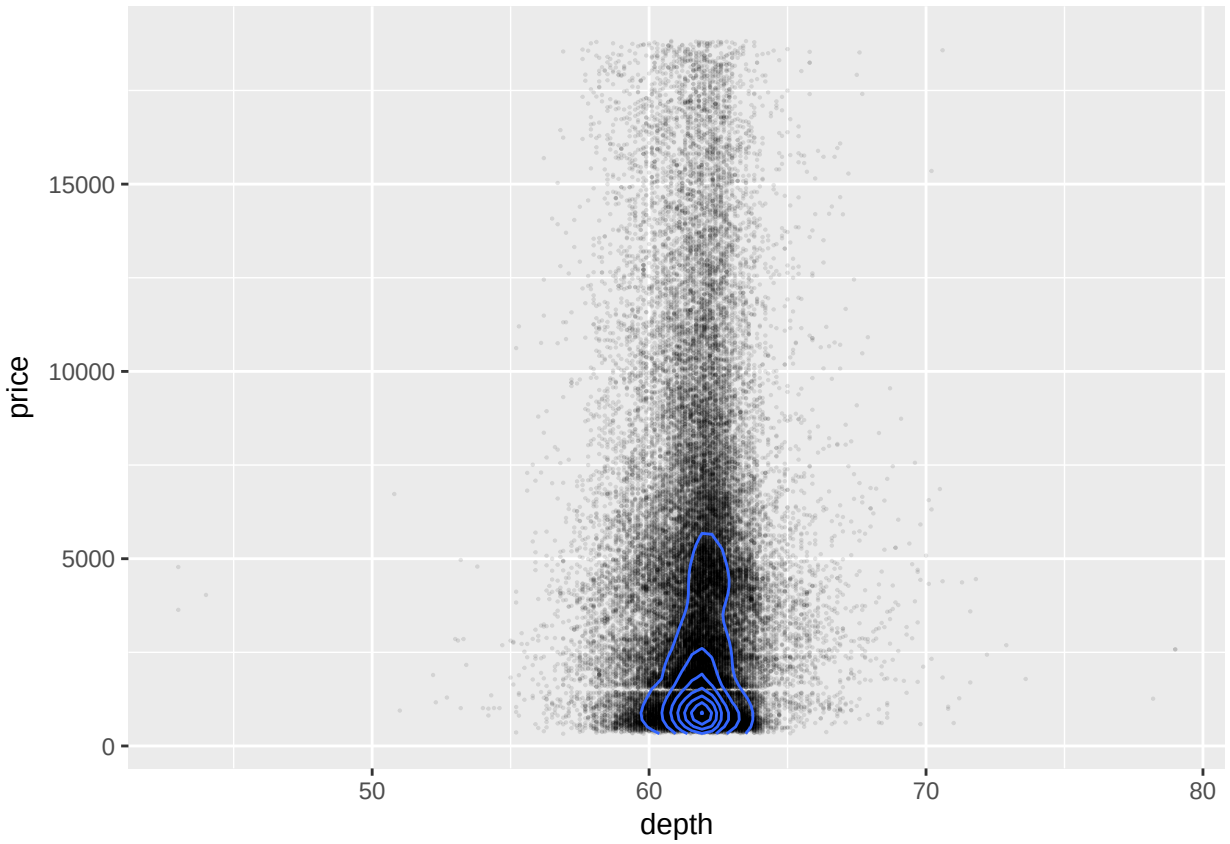
violin plot plus jitterplot:

```
ggplot(iris, aes(Species, Sepal.Width)) +
  geom_violin(aes(fill=Species)) +
  geom_jitter(width = 0.1, alpha=0.4, size=0.5)
```



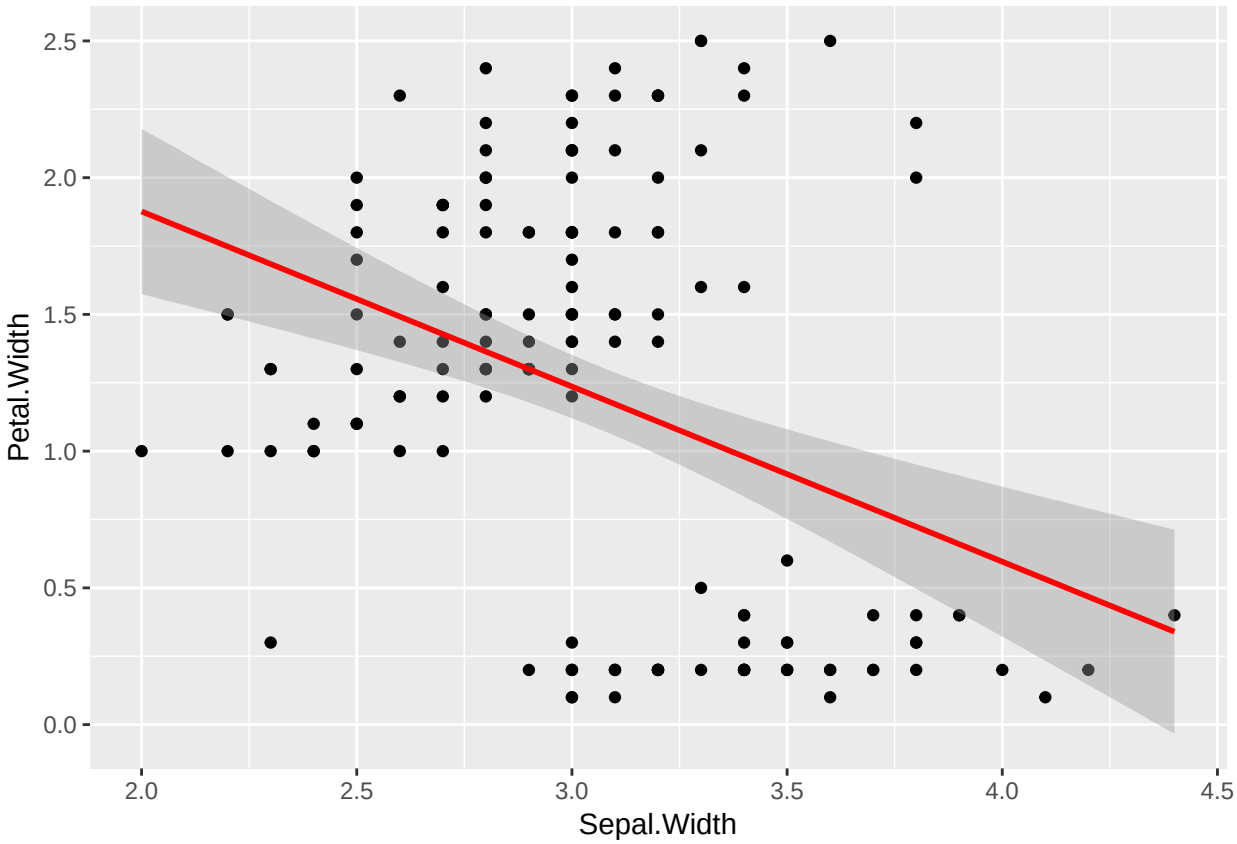
9.2.2 Kahe muutuja koos-varieerumine

```
ggplot(data = diamonds, aes(x = depth, y = price)) +
  geom_point(size=0.1, alpha=0.1) +
  geom_density2d()
```



Fit a linear model and plot the dots and model:

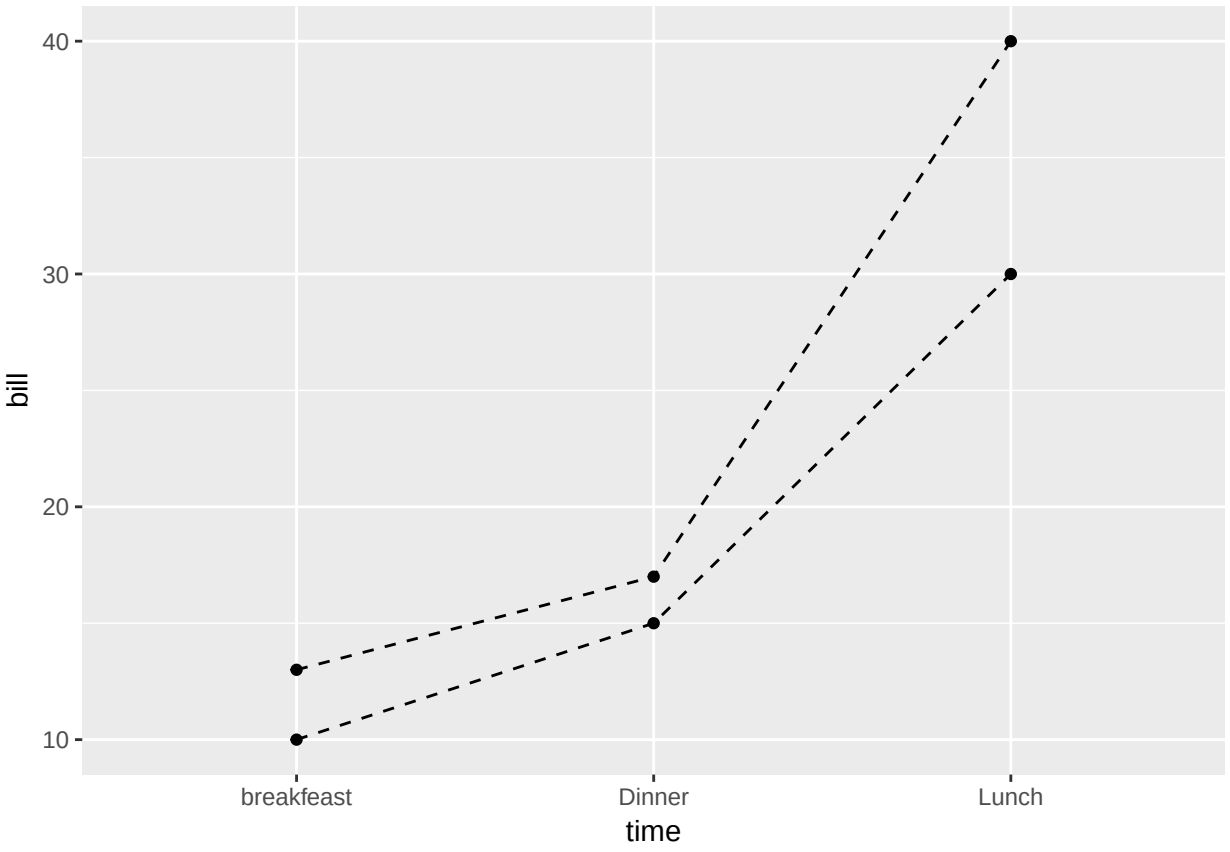
```
ggplot(data=iris, aes(Sepal.Width, Petal.Width))+
  geom_point()+
  geom_smooth(method="lm", color="red")
```



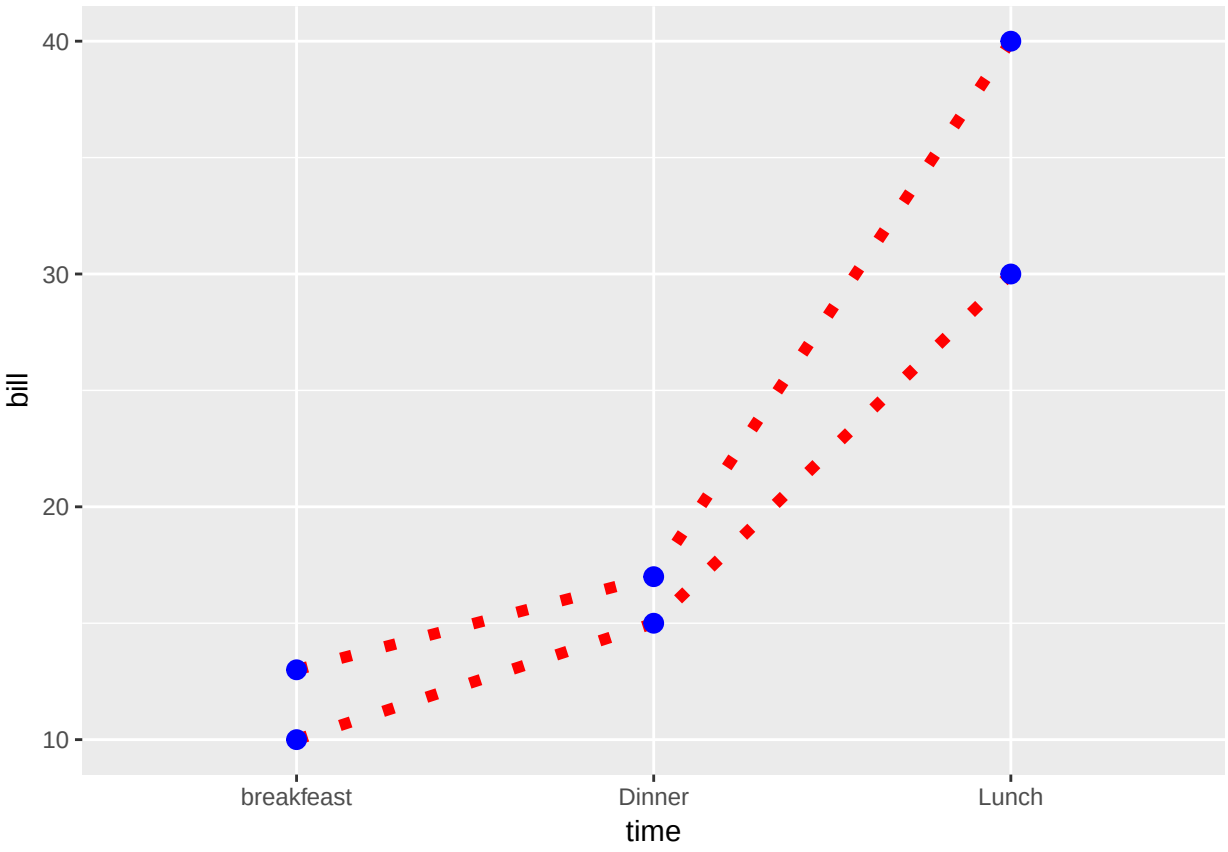
9.3 Joongraafikud

Joonetüübid : “blank”, “solid”, “dashed”, “dotted”, “dotdash”, “longdash”, “twodash”.

```
df2 <- data.frame(sex = rep(c("Female", "Male"), each=3),
                  time=c("breakfeast", "Lunch", "Dinner"),
                  bill=c(10, 30, 15, 13, 40, 17) )
# Change line types
ggplot(data=df2, aes(x=time, y=bill, group=sex)) +
  geom_line(linetype="dashed")+
  geom_point()
```

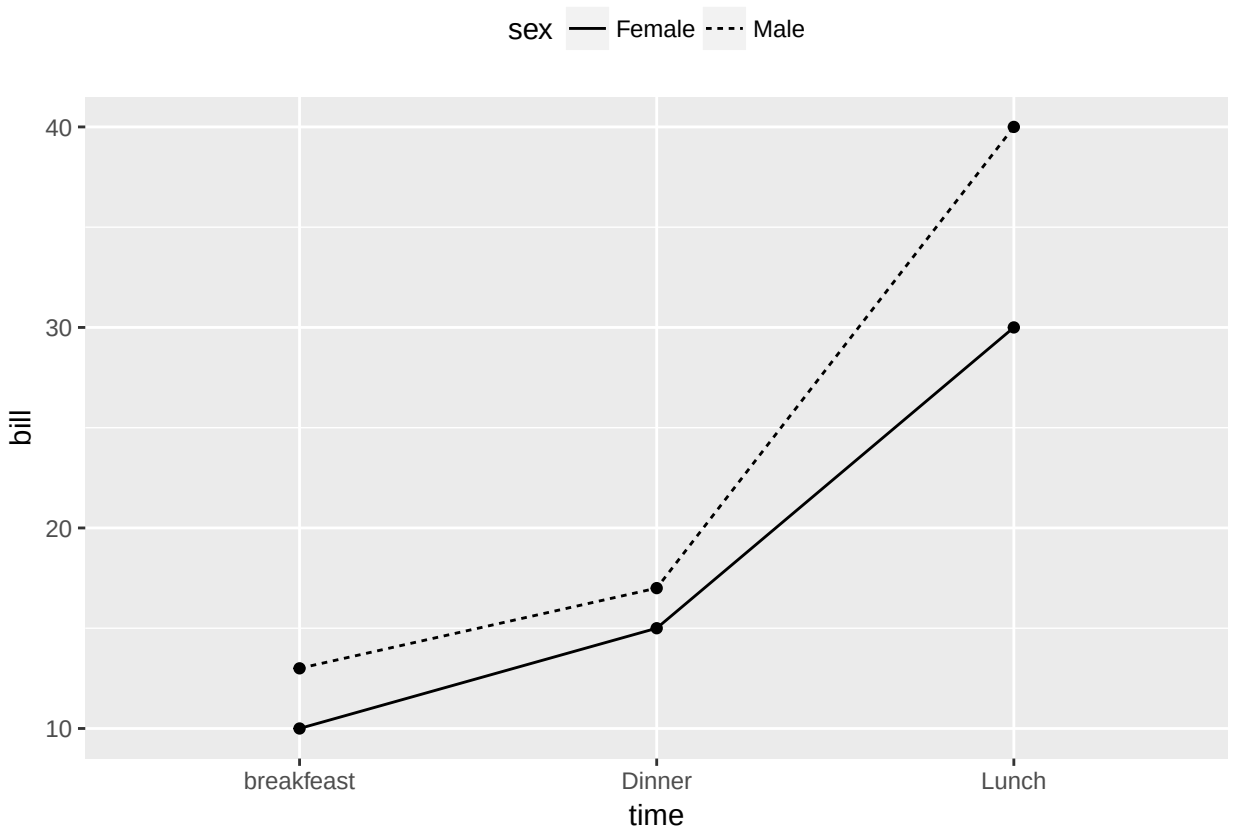



```
# Change line colors and sizes
ggplot(data=df2, aes(x=time, y=bill, group=sex)) +
  geom_line(linetype="dotted", color="red", size=2)+
  geom_point(color="blue", size=3)
```

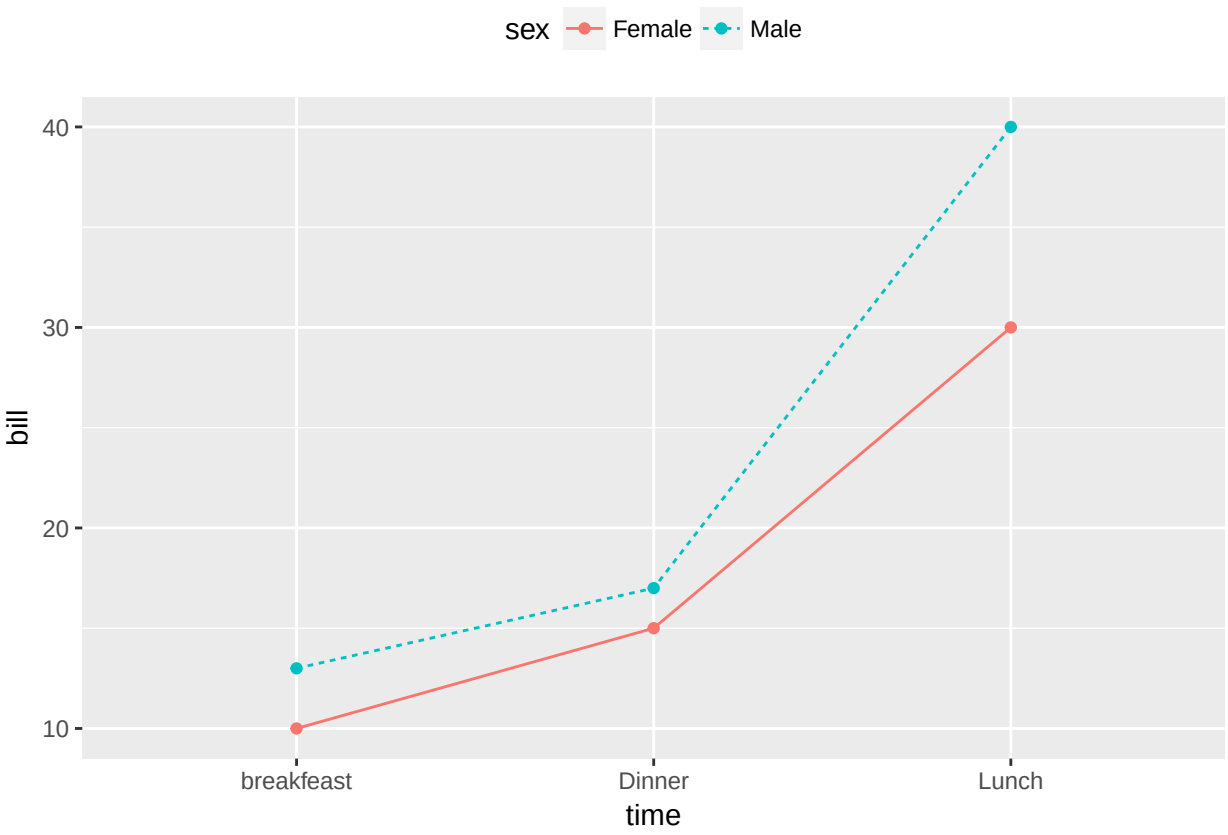


Muudab tüüpi automaatselt muutuja sex taseme järgi

```
# Change line types by groups (sex)
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex))+
  geom_point()+
  theme(legend.position="top")
```




```
# Change line types + colors
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex, color=sex))+
  geom_point(aes(color=sex))+
  theme(legend.position="top")
```



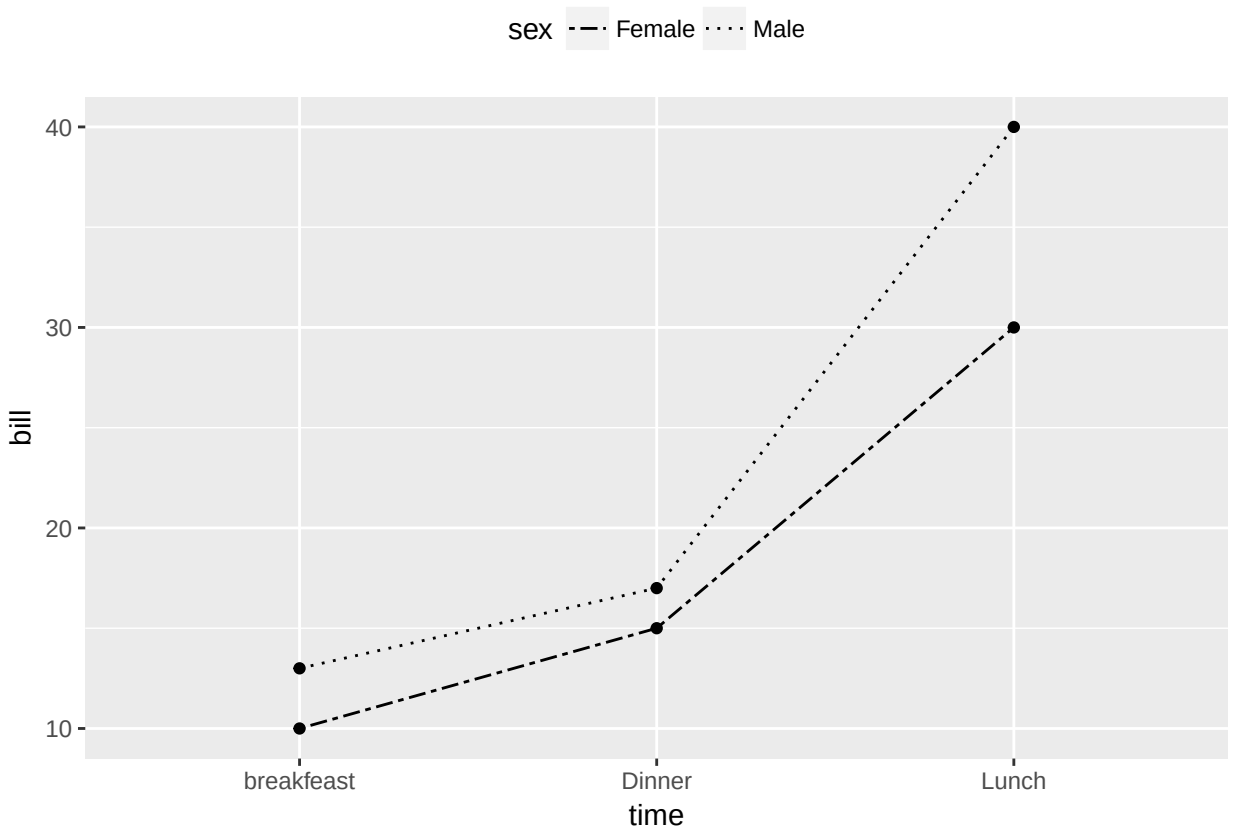
Muuda jooni käsitsi:

scale_linetype_manual() : joone tüüp

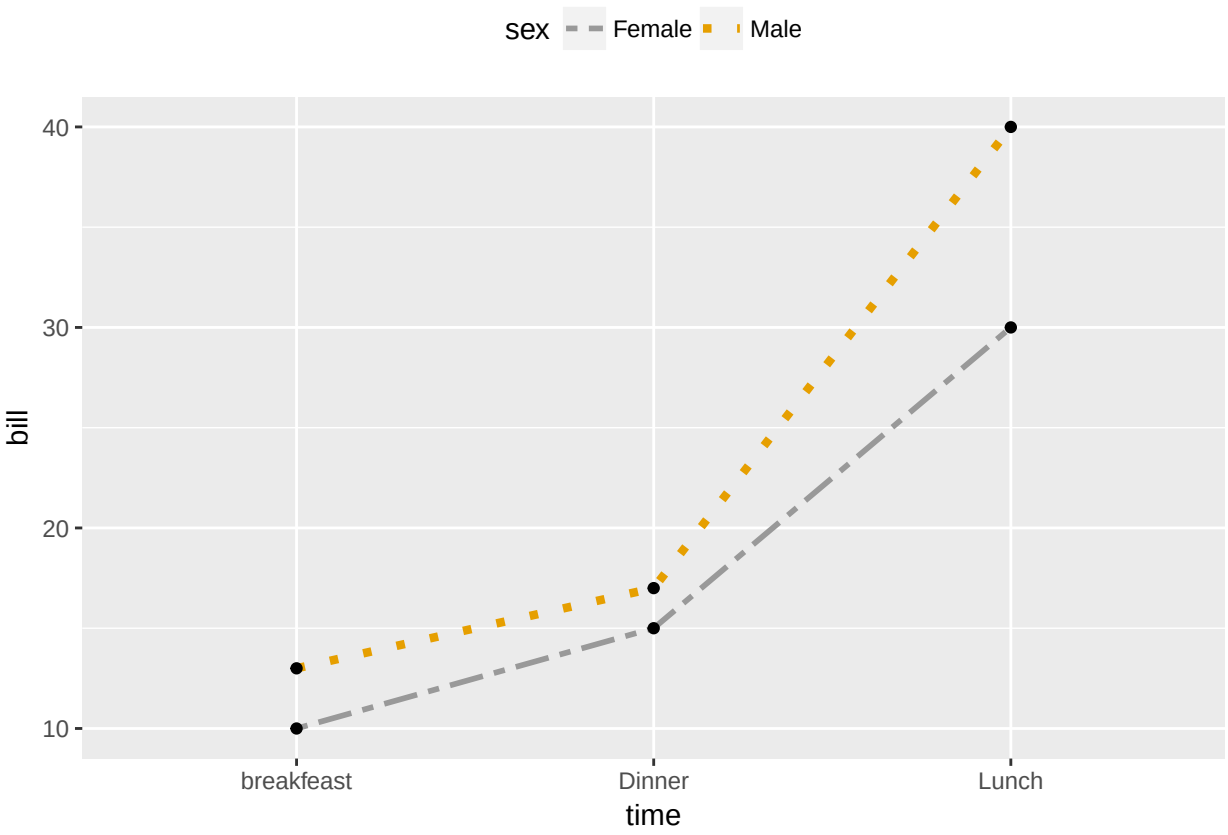
scale_color_manual() : joone värv

scale_size_manual() : joone laius

```
# Set line types manually
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex))+
  geom_point()+
  scale_linetype_manual(values=c("twodash", "dotted"))+
  theme(legend.position="top")
```




```
# Change line colors and sizes
ggplot(df2, 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")
```



9.3.0.1 Kokkuvõte:

a. Andmepunktide plottimine säilitab maksimaalselt andmetes olevat infot (nii kasulikku infot kui müra). Aitab leida outliereid (valesti sisestatud andmeid, valesti mõõdetud proove jms). Kui valim on väiksem kui 20, piisab täiesti üksikute andmepunktide plotist koos mediaaniga. Dot-plot ruulib.

b. Histogramm – kõigepealt mõõtskaala ja seejärel andmed jagatakse võrdse laiusega binnidesse ja plotitakse binnide kõrgused. Bin, kuhu läks 20 andmepunkti on 2X kõrgem kui bin, kuhu läks 10 andmepunkti. Samas, bini laius/ulatus mõõteskaalal pole teile ette antud – ja sellest võib sõltuda histogrammi kuju. Seega on soovitav proovida erinevaid bini laiusi ja võrrelda saadud histogramme. Histogramm sisaldab vähem infot kui dot plot, aga võimaldab paremini tabada seaduspärasid & andmejaotust & outliereid suurte andmekoguste korral.

c. Density plot. Silutud versioon histogrammist, mis kaotab infot aga toob vahest välja signaali müra arvel. Density plotte on hea kõrvuti vaadelda joy ploti abil.

d. Box-plot – sisaldab vähem infot kui histogramm, kuid neid on lihtsam kõrvuti võrrelda. Levinuim variant (kuid kahjuks mitte ainus) on Tukey box-plot – mediaan (joon), 50% IQR (box) ja 1,5x IQR (vuntsid), pluss outlierid eraldi punktidena.

e. Violin plot – informatiivsuselt box-ploti ja histogrammi vahepeal – sobib paljude jaotuste kõrvuti võrdlemiseks

f. Line plot – kasuta ainult siis kui nii X kui Y teljele on kantud pidev väärtus (pikkus, kaal, kontsentratsioon, aeg jms). Ära kasuta, kui teljele kantud punktide vahel ei ole looduses mõtet omavaid pidevaid väärtusi (näiteks X teljel on katse ja kontroll või erinevad valgumutatsioonid, mille aktiivsust on mõõdetud)

g. Suhete võrdlemine (pie vs bar)

h. Cleveland plot countide jaoks. Kasuta Barplotti ainult siis, kui Cleveland plot vm plot mingil põhjusel ei sobi. Barplot võiks olla viimane valik.

   Informatsiooni hulk kahanevalt: iga andmepunkt plotitud (dot plot) -> histogram -> density plot/violin plot -> box plot -> bar plot standardhälvetega -> Cleveland plot (barplot ilma veapiirideta)

Chapter 10

Jäta meelde:

1. statistika uurib formaalseid mudeleid, mitte teooriaid ega päris maailma.
2. Statistika jagatakse kahte ossa: kirjeldav ja järeldav (inferential).
3. Kirjeldav statistika kirjeldab teie andmeid summaarsete statistikute ning graafiliste meetodite abil.
4. Järeldav statistika püüab teie andmete põhjal teha järeldusi statistilise populatsiooni kohta, millest need andmed pärinevad
5. Statistika põhiline ülesanne on kvantifitseerida ebakindlust, mis ümbritseb neid järeldusi.

10.1 Sõnastik

- Statistiline populatsioon – objektide kogum, millele soovime teha statistilist üldistust. Näiteks hinnata keskmist ravimi mõju patsiendipopulatsioonis. Või Escherichia coli ensüümi X keskmist Kcat-i.
- Valim – need objektid (patsiendid, ensüümiprepid), mida me reaalselt mõõdame.
- Juhuvalim – valim, mille liikmed on populatsioonist valitud juhuslikult ja iseseisvalt. See tähendab, et kõigil populatsiooni liikmetel (kõikidel patsientidel või kõikidel võimalikel ensüümipreparatsioonidel) on võrdne võimalus sattuda valimisse JA, et valimisse juba sattunud liikme(te) põhjal ei ole võimalik ennustada järgmisena valimisse sattuvat liiget.
- Esinduslik valim – Valim on esinduslik, kui ta peegeldab hästi statistilist populatsiooni. Ka juhuvalim ei pruugi olla esinduslik (juhuslikult).
- Statistik – midagi, mis on täpselt arvutatud valimi põhjal (näiteks pikkuste keskmine)
- Parameetri väärtus – teadmata suurus, mille täpset väärtust me saame ainult umbkaudu ennustada aga mitte kunagi täpselt teada. (näiteks mudeli intercept, populatsiooni keskmine pikkus; efekti suurus = katsegrupi keskmine – kontrollgrupi keskmine)
- Statistiline mudel – matemaatiline formaliseering, mis sageli koosneb 2st osast: determinismlik protsessi-mudel pluss juhuslik vea/varieeruvuse-mudel. Protsessi-mudeli näiteks kujutle, et mõõdad mitme inimese pikkust (X muutuja) ja kaalu (Y muutuja). Sirge võrrandiga Y = a + b * X (kaal = a + b * pikkus) saab anda determinismliku lineaarse ennustuse kaalu kohta: kui X (pikkus) muutub ühe ühiku (cm) võrra, siis muutub Y (kaal) väärtus keskmiselt b ühiku (kg) võrra. Seevastu varieeruvuse-mudel on tõenäosusjaotus (näit normaaljaotus). Selle abil modelleeritakse Y-suunalist andmete varieeruvust igal X väärtusel (näiteks, milline on 182 cm pikkuste inimeste oodatav kaalujaotus). Mudel on seega tõenäosuslik: me saame näiteks küsida: millise tõenäosusega kaalub 182 cm pikkune inimene üle 100 kilo. Mida laiem on varieeruvuse mudeli Y-i suunaline jaotus igal X-i

väärtusel, seda kehvemini ennustab mudel, millist Y väärtust võime konkreetselt oodata mingi X-i väärtuse korral. Lineaarsete mudelite eesmärk ei ole siiski mitte niivõrd uute andmete ennustamine (seda teevad paremini keerulised mudelid), vaid mudeli struktuurist lähtuvalt põhjuslike hüpoteeside püstitamine/kontrollimine (kas inimese pikkus võiks otseselt reguleerida/kontrollida tema kaalu?). Kuna selline viis teadust teha töötab üksnes lihtsate mudelite korral, on enamkasutatud statistilised mudelid taotluslikult lihtsustavad ja ei pretendeeri tõelähedusele.

- Tehniline replikatsioon – sama proovi (patsienti, ensüümipreparatsiooni, hiire pesakonna liiget) mõõdetakse mitu korda. Mõõdab tehnilist varieeruvust ehk mõõtmisviga. Seda püüame kontrollida parandades mõõtmisaparatuuri/protokolle või siis juba andmete tasemel, statistilise analüüsiga. Näiteks saame andmeid agregeerida ja arvutada keskväärtuse. Kui andmepunkte on piisavalt ja varieeruvus on sümmeetriline ümber tõelise populatsiooniväärtuse, siis annab keksväärtus meile hea hinnangu parameetri tõelisele väärtusele.
- Bioloogiline replikatsioon – erinevaid patsiente, ensüümipreppe, erinevate hiirepesakondade liikmeid mõõdetakse, igaüht üks kord. Eesmärk on mõõta Bioloogilist varieeruvust, mis tuleneb mõõteobjektide reaalsetest erinevustest: iga patsient ja iga ensüümimolekul on erinev kõigist teistest omasugustest. Bioloogiline varieeruvus on teaduslikult huvitav ja seda saab visualiseerida algandmete tasemel (mitte keskväärtuse tasemel) näiteks histogrammina. Teaduslikke järeldusi tehakse bioloogiliste replikaatide põhjal. Tehnilised replikaadid seevastu kalibreerivad mõõtesüsteemi täpsust. Kui te uurite soolekepikest E. coli, ei saa te teha formaalset järeldust kõigi bakterite kohta. Samamoodi, kui te uurite vaid ühe hiirepesakonna/puuri liikmeid, ei saa te teha järeldusi kõikide hiirte kohta. Kui teie katseskeem sisaldab nii tehnilisi kui bioloogilisi replikaate on lihtsaim viis neid andmeid analüüsida kõigepealt keskmistada üle tehniliste replikaatide ning seejärel kasutada saadud keskmisi edasistes arvutustes üle bioloogiliste replikaatide (näiteks arvutada nende pealt uue keskmise, standardhälve ja/või usaldusintervalli). Selline kahe-etapiline arvutuskäik ei ole siiski optimaalne. Optimaalne, kuid keerukam, on panna mõlemat tüüpi andmed ühte hierarhilisse mudelisse.

10.1.0.1 Tõenäosuse (P) reeglid on ühised kogu statistikale:

- P jääb 0 ja 1 vahele; P(A) = 1 tähendab, et sündmus A toimub kindlasti.
- kui sündmused A ja B on üksteist välistavad, siis tõenäosus, et toimub sündmus A või sündmus B on nende kahe sündmuse tõenäosuste summa — P(A v B) = P(A) + P(B).
- Kui A ja B ei ole üksteist välistavad, siis P(A v B) = P(A) + P(B) – P(A & B).
- kui A ja B on üksteisest sõltumatud (A toimumise järgi ei saa ennustada B toimumist ja vastupidi) siis tõenäosus, et toimuvad mõlemad sündmused on nende sündmuste tõenäosuste korrutis —- P(A & B) = P(A) x P(B).
- Kui B on loogiliselt A alamosa, siis P(B) < P(A)
- P(A | B) —- tinglik tõenäosus. Sündmuse A tõenäosus, juhul kui peaks toimuma sündmus B. P(vihm | pilves ilm) ei ole sama, mis P(pilves ilm | vihm).
- Juhul kui P(B)>0, siis P(A | B) = P(A & B)/P(B) ehk
- P(A | B) = P(A) x P(B | A)/P(B) —- Bayesi teoreem.

Kuigi kõik statistikud lähtuvad tõenäosustega töötamisel täpselt samadest matemaatilistest reeglitest, tõlgendavad erinevad koolkonnad saadud numbreid erinevalt. Kaks põhilist koolkonda on sageduslikud statistikud ja Bayesiaanid.

- Tõenäosus, sageduslik tõlgendus – pikaajaline sündmuste suhteline sagedus. Näiteks 6-te sagedus paljudel täringuvisetel. Sageduslik tõenäosus on teatud tüüpi andmete sagedus, tingimusel et nullhüpotees (H0) kehtib; ehk P(andmed | H0). Nullhüpotees ütleb enamasti, et uuritava parameetri (näiteks ravimiefekti suurus) väärtus on null. Seega, kui P on väike, ei ole seda tüüpi andmed kooskõlas arvamusega, et parameetri väärtus on null (mis aga ei tähenda automaatselt, et sa peaksid uskuma, et parameetri väärtus ei ole null).
- Tõenäosus, Bayesi tõlgendus – usu määr mingisse hüpoteesi. Näiteks 62% tõenäosus (et populatsiooni keskmine pikkus < 180 cm) tähendab, et sa oled ratsionaalse olendina nõus kulutama mitte rohkem

kui 62 senti kihlveo peale, mis võidu korral toob sulle sisse 1 EUR (ja 38 senti kasumit). Bayesi tõenäosus omistatakse statistilisele hüpoteesile (näiteks, et ravimiefekti suurus jääb vahemikku a kuni b), tingimusel, et sul on täpselt need andmed, mis sul on; ehk P(hüpotees | andmed).

Chapter 11

Järeldav statistika

```
library(tidyverse)
library(bayesboot)
library(boot)
library(rethinking) #HPDI() and PI(). Requires Stan.
#source("R/")
```

11.0.1 3.1. Simulatsiooni jõud

Järeldav statistika püüab valimi põhjal teha järeldusi statistilise populatsiooni kohta, millest see valim pärineb. Sellisel tegevusel on mõtet ainult siis, kui ühest küljest valim peegeldab populatsiooni ja teisest küljest valim ei ole sama asi, mis populatsioon. Seega on statistika abil tehtud järeldused alati rohkem või vähem ebamäärased ning meil on vaja meetodit selle ebamäärasuse mõõtmiseks. Aga kõigepealt illustreerime valimi ja populatsiooni erinevust.

11.0.2 3.1.1. Valim ei ole sama, mis populatsioon

Simuleerimine on lahe sest erinevalt päris maailmast elavad simulatsioonid mudeli väikses maailmas ning seega teame me täpselt, mida me teeme ja mida on meil selle tagajärjel oodata. Simulatsioonidega saame me hõlpsalt kontrollida, kas ja kuidas meie mudelid töötavad ning genereerida olukordi (parameetrite väärtuste kombinatsioone), mida päris maailmas kunagi ette ei tule. Selles mõttes on mudelid korraga nii väiksemad kui suuremad kui päris maailm.

Alustuseks simuleerime juhuvalimi n=3 lõpmata suurest normaaljaotusega populatsioonist, mille keskmine on 100 ja sd on 20. Päris elus on korraliku juhuvalimi tõmbamine tehniliselt tõsine ettevõtmine ja, mis veelgi olulisem, me ei tea kunagi, milline on populatsiooni tõeline jaotus, keskmine ja sd. Elagu simulatsioon!

```
set.seed(1) #makes random number generation reproducible
(Sample <- rnorm(n = 3, mean = 100, sd = 20)) #extra parentheses work as print()
```

```
## [1]  87.47092 103.67287  83.28743
```

```
mean(Sample)
```

```
## [1] 91.47707
```

```
sd(Sample)
```

```
## [1] 10.76701
```

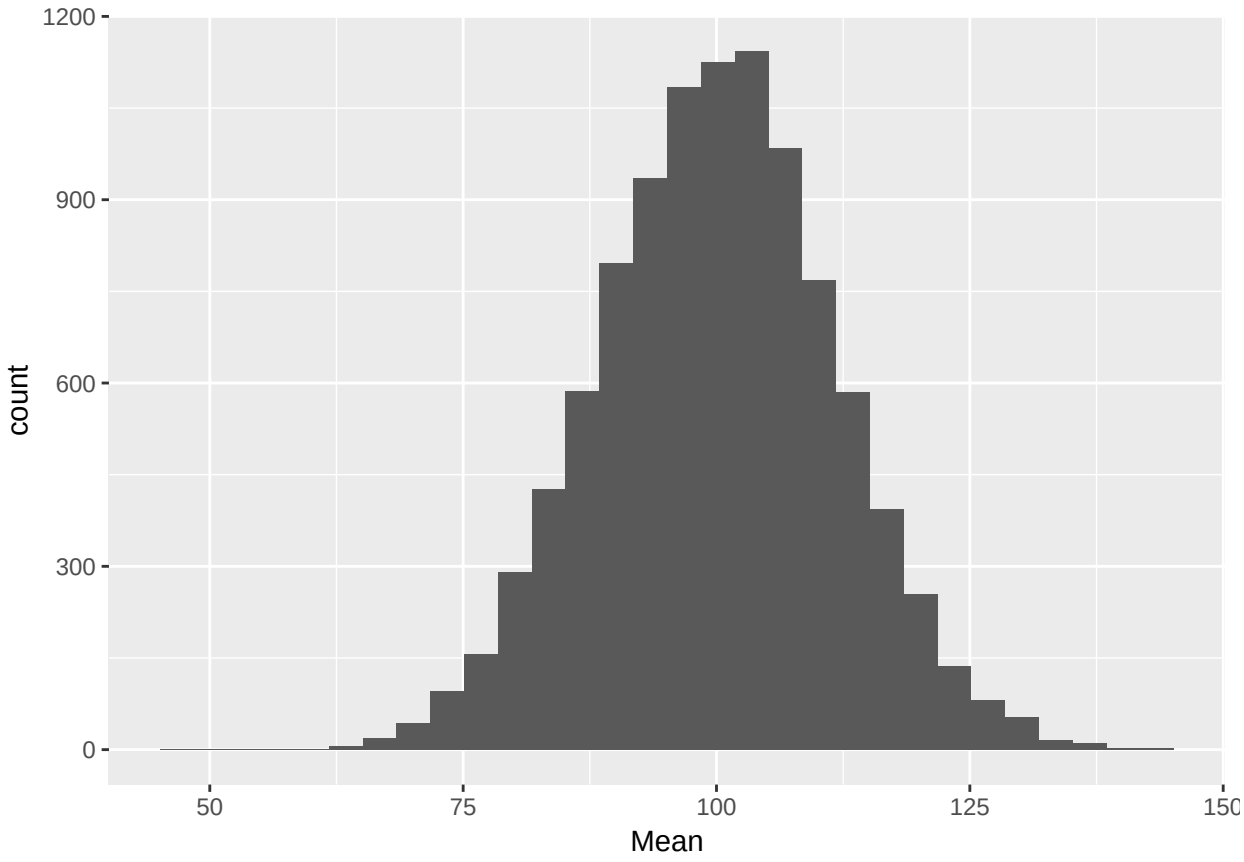
Nagu näha on meie valimi keskmine 10% väiksem kui peaks ja valimi sd lausa kaks korda väiksem kui peaks. Seega peegeldab meie valim halvasti populatsiooni — aga me teame seda ainult tänu sellele, et tegu on simulatsiooniga.

Kui juba simuleerida, siis robinal: tõmbame ühe valimi asemel 10 000, arvutame seejärel 10 000 keskmist ja sd-d ning vaatame omakorda nende statistikute jaotusi ja keskväärtusi. Simulatsioon on nagu tselluliit — see on nii odav, et igaüks võib seda endale lubada.

Meie lootus on, et kui meil on palju valimeid, millel kõigil on juhuslik viga, mis neid populatsiooni suhtes ühele või teisele poole kallutab, siis rohkem on valimeid, mis asuvad tõelisele populatsioonile pigem lähemal kui kaugemal.

```
N <- 3
N_simulations <- 10000
df <- tibble(a = rnorm(N * N_simulations, 100, 20), b = rep(1:N_simulations, each = N) )
Summary <-  df %>% group_by(b) %>% summarise(Mean=mean(a), SD= sd(a) )

Summary %>% ggplot(aes(Mean) ) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
mean(Summary$Mean)
```

```
## [1] 99.98043
```

```
mean(Summary$SD)
```

```
## [1] 17.76452
```

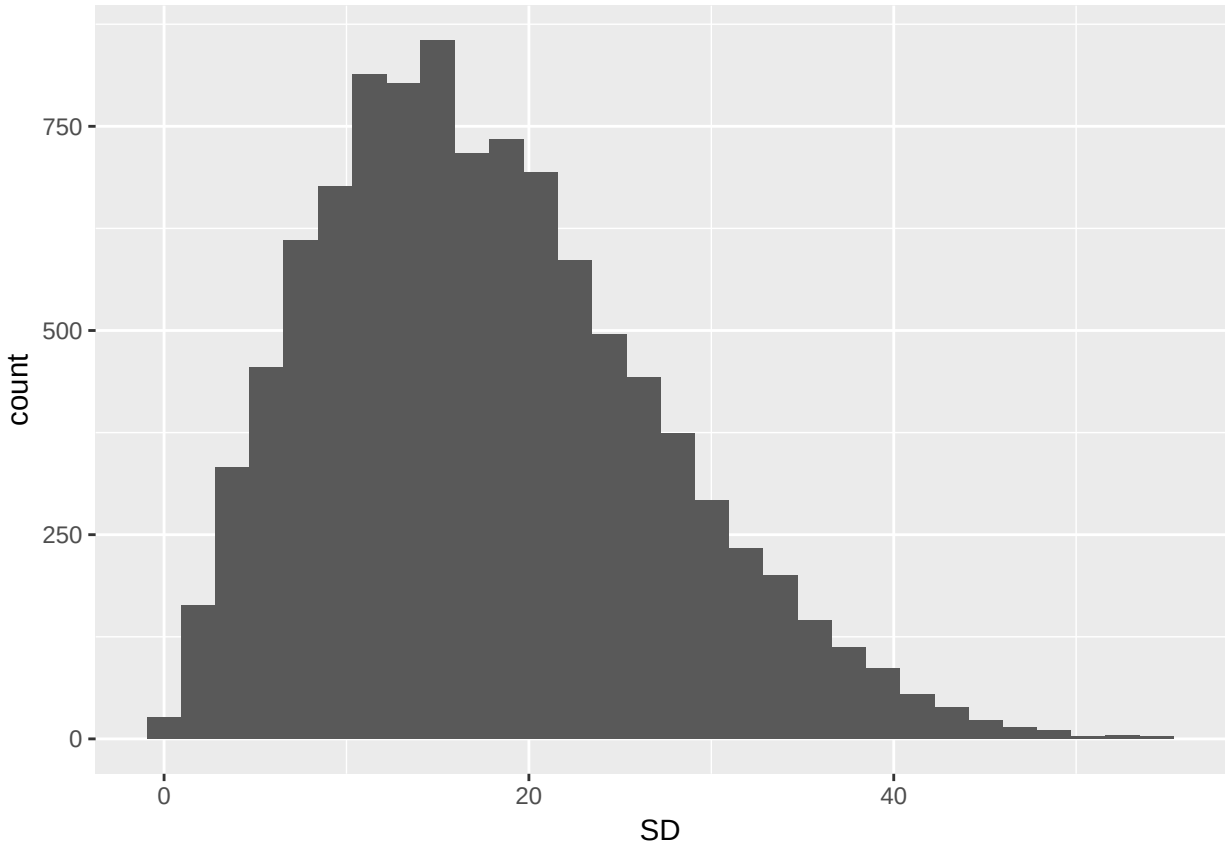
Oh-hooo. Paljude valimite keskmiste keskmine ennustab väga täpselt populatsiooni keskmist aga sd-de

keskmise keskmine alahindab populatsiooni sd-d. Valem, millega sd-d arvutatakse tõõtab lihtsalt kallutatult, kui n on väike (<10)

Ja nüüd 10 000 SD keskväärtused:

```
Summary %>% ggplot(aes(SD)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
mode <-  function(x, adjust=1) {
  x <- na.omit(x)
  dx <- density(x, adjust=adjust)
y_max <- dx$x[which.max(dx$y)]
y_max
}
mode(Summary$SD)
```

```
## [1] 14.07554
```

SD-de jaotus on ebasümmeetriline ja mood ehk kõige tõenäolisem valimi sd väärtus, mida võiksime oodata, on u 14, samal ajal kui populatsiooni sd = 20. Lisaks on sd-de jaotusel paks saba, mis tagab, et tesest küljest pole ka vähetõenäoline, et meie valimi sd populatsiooni sd-d kõvasti üle hindab.

Arvutame, mitu % valimite sd-e keskmistest on > 25

```
sum(Summary$SD>25)/10000
```

```
## [1] 0.2114
```

Me saame >20% tõenäosusega pahasti ülehinnatud SD.

```
sum(Summary$SD<15)/10000
```

```
## [1] 0.4344
```

Ja me saame >40% tõenäosusega pahasti alahinnatud sd. Selline on väikeste valimite traagika. (Jooksuta sama simulatsiooni n = 100 korral.)

Aga vähemalt populatsiooni keskmise saame me palju valimeid tõmmates ilusasti kätte — ka väga väikeste valimitega.

Kahjuks pole meil ei vahendeid ega kannatust 10 000 valimi loodusest kogumiseks. Enamasti on meil üksainus valim. Õnneks pole sellest väga hullu sest meil on olemas analoogne meetod, mis töötab üsna hästi ka ühe valimiga. Seda kutsutakse *bootstrappimiseks* ja selle võttis esimesena kasutusele parun von Münchausen. Too jutukas parun nimelt suutis end soomülkast iseenda patsi pidi välja tõmmata, mis ongi bootstrappimise põhimõte.

11.0.3 3.1.2. Bootstrappimine

> Populatsioon on valimile, mis valim on bootstrappitud valimile.

Nüüd alustame ühestainsast empiirilisest valimist ja genereerime sellest 10 000 virtuaalset valimit. Selleks tõmbame me oma valimist virtuaalselt 10 000 uut juhuvalimit (bootstrap valimit), igaüks neist sama suur kui algne valim. Trikk seisneb selles, et bootstrap valimite tõmbamine käib asendusega, st iga empiirilise valimi element, mis bootstrap valimisse tõmmatakse, pannakse kohe ka algsesse valimisse tagasi. Seega saab seda elementi kohe uuesti samasse bootstrap valimisse tõmmata (kui juhus nii tahab). Seega sisaldab tüüpiline bootstrap valim osasid algse valimi numbreid mitmes korduses ja teisi üldse mitte. Bootstrap valimid plotitakse sama moodi nagu me ennist tegime oma valimitega lõpmatust populatsioonist. Ainsad erinevused on, et bootstrapis on piiratud algse andmekogu suurus, millest valimeid tõmmatakse ning, et iga bootstrapi valim on sama suur kui algne andmekogu.

Bootstrap empiirilisele valimile suurusega n töötab nii:

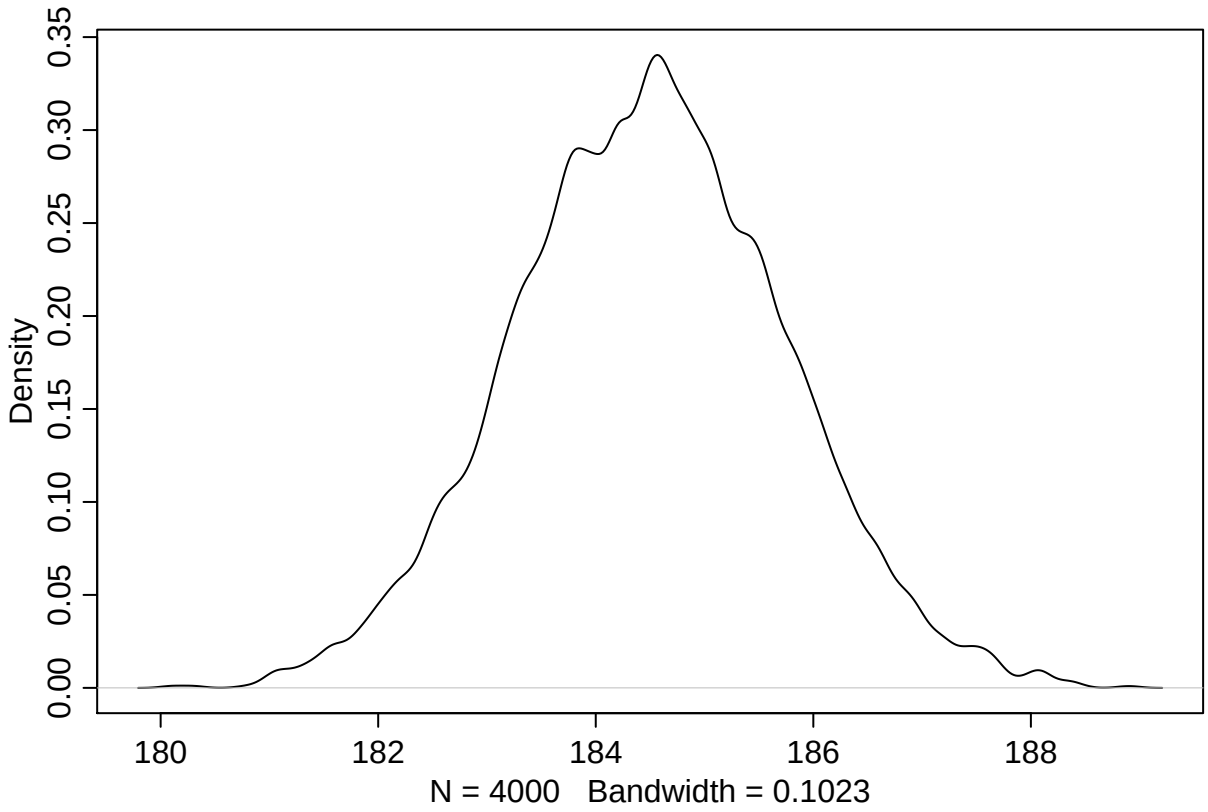
1) tõmba empiirilisest valimist k uut virtuaalset valimit, igaüks suurusega n
2) arvuta keskmine, sd või mistahes muu statistik igale bootstrapi valimile. Tee seda k korda.
3) joonista oma statistiku väärtustest histogramm või density plot
4) nende andmete põhjal saab küsida palju toreidaid küsimusi — vt allpool.

Mis on USA presidentide keskmine pikkus? Meil on valim 10 presidendi pikkusega.

```
heights <- c(183, 192, 182, 183, 177, 185, 188, 188, 182, 185) %>% as_tibble(heights)
```

```
## Warning in as.data.frame.numeric(value, stringsAsFactors = FALSE, ...):
## 'row.names' is not a character vector of length 10 -- omitting it. Will be
## an error!
```

```
n <- 10 #sample size
nr_boot_samples <- 4000 #the nr of bootstrap samples
a <- sample_n(heights, n * nr_boot_samples, replace=TRUE)  #create random sample with replacement
a$key <- rep(1:nr_boot_samples, each = n) #create a column "key" that cuts the sample into slices of si
a1 <- a %>% group_by(key) %>% summarise(Value= mean(value)) #calculate the mean for each slice of n val
dens(a1$Value)
```



Alternatiivne ja aeglasem kood, mis kasutab mosaic::do().

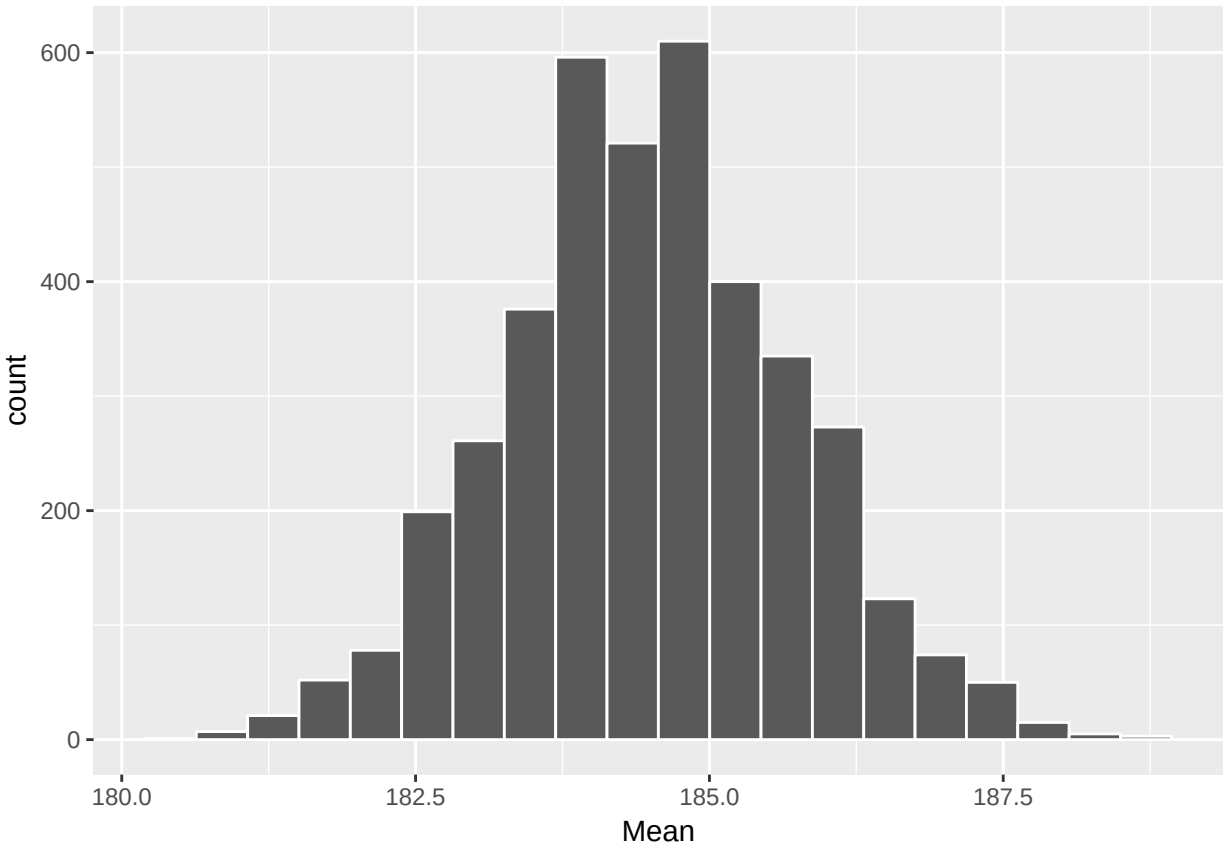
```
library(mosaic)
heights <- c(183, 192, 182, 183, 177, 185, 188, 188, 182, 185) %>% as_tibble(heights)
```

```
## Warning in as.data.frame.numeric(value, stringsAsFactors = FALSE, ...):
## 'row.names' is not a character vector of length 10 -- omitting it. Will be
## an error!
```

```
sample_means <- mosaic::do(4000) *
    (heights %>% sample_n(size = 10, replace = TRUE))
```

```
## Using parallel package.
##   * Set seed with set.rseed().
##   * Disable this message with options(`mosaic:parallelMessage` = FALSE)
```

```
#võta heigths andmeraamist 10-ne valim (tagasi panekuga) ja tee seda 4000 korda järjest. tulemuseks on

sample_means1 <- sample_means %>% group_by(.index) %>% summarise(Mean=mean(value))

ggplot(data = sample_means1, aes(x = Mean)) +
  geom_histogram(color = "white", bins = 20)
```



Nii saab do() abil 2 veeruga tibble, mille igas veerus on 3 juhuslikku arvu. NB! kuna ka tidyverse/dplyr sisaldab do() nimelist funktsiooni, on kasulik seda siin eksplitsiitselt mosaic::do() kaudu esile manada.

```
mosaic::do(3)* rnorm(2)
```

```
## Using parallel package.
##   * Set seed with set.rseed().
##   * Disable this message with options(`mosaic:parallelMessage` = FALSE)
```

```
##           V1         V2
## 1 -0.1073985 -0.4952203
## 2  0.6102722  1.0238552
## 3 -1.2747053  0.3952969
```

Mida selline keskväärtuste jaotus tähendab? Me võime seda vaadelda posterioorse tõenäosusjaotusena. Selle tõlgenduse kohaselt iseloomustab see jaotus täpselt meie usku presidentide keskmise pikkuse kohta, niipalju kui see usk põhineb bootstrappimises kasutatud andmetel. Senikaua, kui meil pole muid relevantseid andmeid, on kõik, mida me usume teadvat USA presidentide keskmise pikkuse kohta, peidus selles jaotuses. Need pikkused, mille kohal jaotus on kõrgem, sisaldavad meie jaoks tõenäolisemalt tegelikku USA presidentide keskmist pikkust kui need pikkused, mille kohal posterioorne jaotus on madalam.

Kuidas selle jaotusega edasi töötada? See on lihtne: meil on 4000 arvu ja me teeme nendega kõike seda, mida parasjagu tahame.

Näiteks me võime arvutada, millisesse pikkuste vahemikku jääb 92% meie usust USA presidentide tõelise keskmise pikkuse kohta. See tähendab, et teades seda vahemikku peaksime olema valmis maksma mitte rohkem kui 92 senti pileti eest, mis juhul kui USA presidentide keskmine pikkus tõesti jääb sinna vahemikku, toob meile võidu suuruses 1 EUR (ja 8 senti kasumit). Selline kihlveokontor on muide täiesti respektaabel ja akadeemiline tõenäosuse tõlgendus; see on paljude arvates lausa parim tõlgendus, mis meil on.

Miks just 92% usaldusinterval? Vastus on, et miks mitte? Meil pole ühtegi universaalset põhjust eelistada üht usaldusvahemiku suurust teisele. Olgu meil usaldusinteval 90%, 92% või 95% — tõlgendus on ikka sama. Nimelt, et me usume, et suure tõenäosusega jääb tegelik keskväärtus meie poolt arvutatud vahemikku. Mudeli ja maailma erinevused tingivad niikuinii selle, et konkreetne number ei kandu mudelist üle pärismaailma. NB! pane tähele, et eelnevalt mainitud kihlveokontor töötab mudeli maailmas, mitte teie kodu lähedasel hipodroomil.

92% usaldusintervalli arvutamiseks on kaks meetodit, mis enamasti annavad vaid veidi erinevaid tulemusi. 1) HPDI — Highest Density Probability Interval — alustab jaotuse tipust (tippudest) ja isoleerib 92% jaotuse kõrgema(te) osa(de) pindalast 2) PI — Probability Interval — alustab jaotuse servadest ja isoleerib kummagist servast 4% jaotuse pindalast. See on sama, mis arvutada 4% ja 96% kvantiilid

```
HPDI(a1$Value, prob=0.92) #highest probability density interval - non-symmetric
```

```
## |0.92 0.92|
## 182.3 186.6
```

```
PI(a1$Value, prob=0.92) #symmetric method.
```

```
##    4%   96%
## 182.3 186.7
```

HPDI on üldiselt parem mõõdik kui PI, aga teatud juhtudel on seda raskem arvutada. Kui HPDI ja PI tugevalt erinevad, on hea mõte piirduda jaotuse enda avaldamisega — jaotus ise sisaldab kogu informatsiooni, mis meil on oma statistiku väärtuse kohta. Intervallid on lihtsalt summaarsed statistikud andmete kokkuvõtlikuks esitamiseks.

Kui suure tõenäosusega on USA presidentide keskmine pikkus suurem kui USA populatsiooni meeste keskmine pikkus (178.3 cm mediaan)?

```
sum(a1$Value>178.3)/4000
```

```
## [1] 1
```

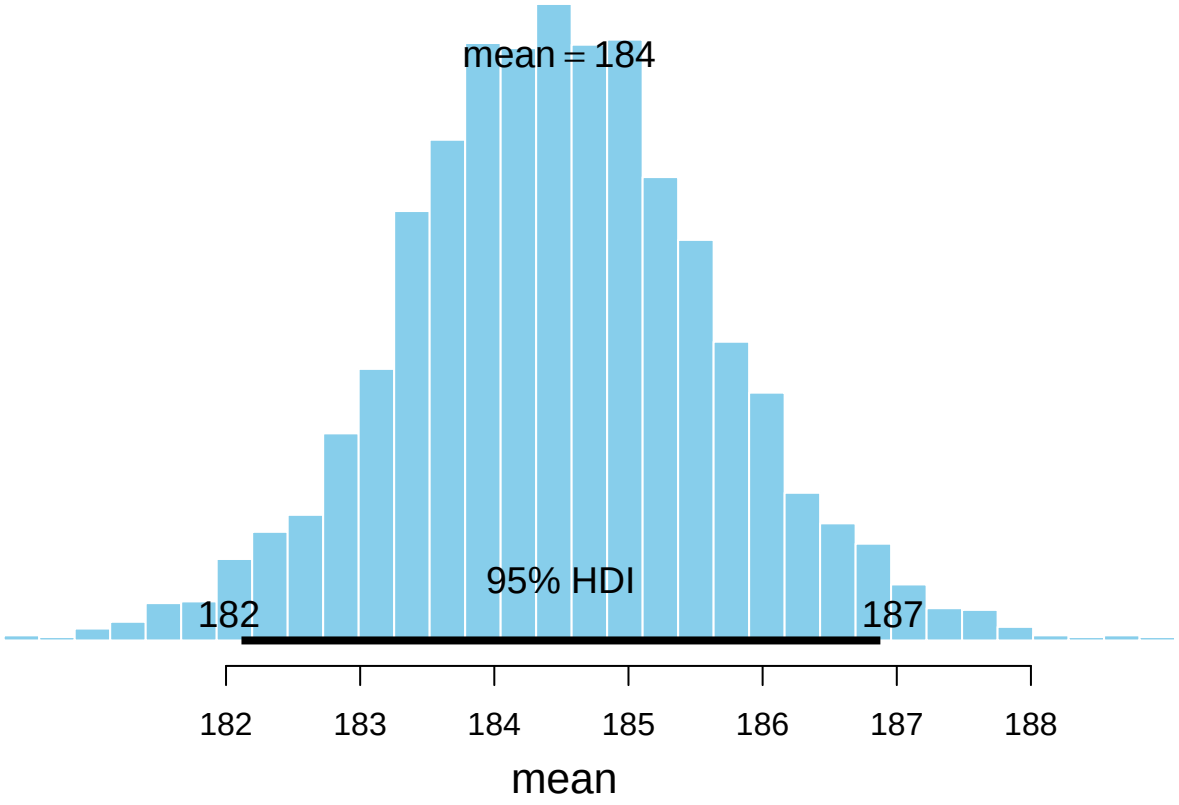
```
mode(a1$Value)
```

```
## [1] 184.5883
```

Ligikaudu 100% tõenäosusega (valimis on 1 mees alla 182 cm, ja tema on 177 cm). Lühikesed jupatsid ei saa Ameerikamaal presidendiks!

Edaspidi kasutame bootstrappimiseks veidi moodsamat meetodit — Bayesian bootstrap — mis töötab veidi paremini väikeste valimite korral. Aga üldiselt on tulemused sarnased.

```
library(bayesboot)
# Heights of the last ten American presidents in cm (Kennedy to Obama).
heights <- c(183, 192, 182, 183, 177, 185, 188, 188, 182, 185);
b1 <- bayesboot(heights, mean)
plot(b1)
```




```
#summary(b1)
#hist(b1)
HPDI(b1$V1, prob=0.95)
```

```
##    |0.95    0.95|
## 182.1153 186.8777
```

```
# it's more efficient to use the a weighted statistic (but you can use a normal statistic like mean() o
b2 <- bayesboot(heights, weighted.mean, use.weights = TRUE)
```

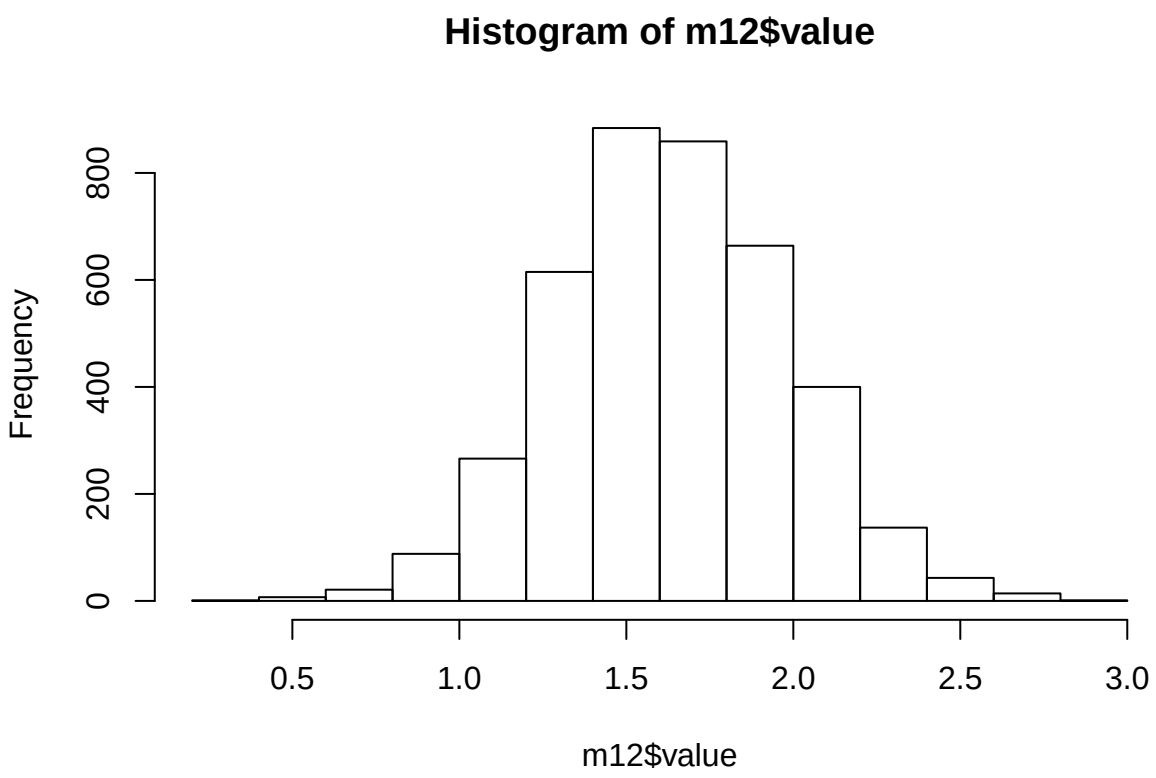
It can also be easily post processed.

```
# the probability that the mean is > 182 cm.
mean( b1[,1] > 182)
```

```
## [1] 0.98075
```

2 keskväärtuse erinevus (keskmine1 - keskmine2):

```
df <- tibble(a=rnorm(10), b=rnorm(10,1,1))
m1 <- bayesboot(df$a, mean)
m2 <- bayesboot(df$b, mean)
m12 <- bind_cols(m1, m2)
m12 <- m12 %>% mutate(value=m12[,2] - m12[,1])
hist(m12$value)
```




```
median(m12$value)
```

```
## [1] 1.624235
```

```
mode(m12$value)
```

```
## [1] 1.543269
```

```
HPDI(m12$value)
```

```
##    |0.89    0.89|
## 1.116852 2.183880
```

```
#library(BayesianFirstAid); bayes.t.test(m1, m2)
#will give a similar, but fully Bayesian, result
#requires JAGS.
```

Bayesian bootstrap SD arvutamiseks.

When use.weights = FALSE it is important that the summary statistics does not change as a function of sample size. This is the case with the sample standard deviation, so here we have to implement a function calculating the population standard deviation.

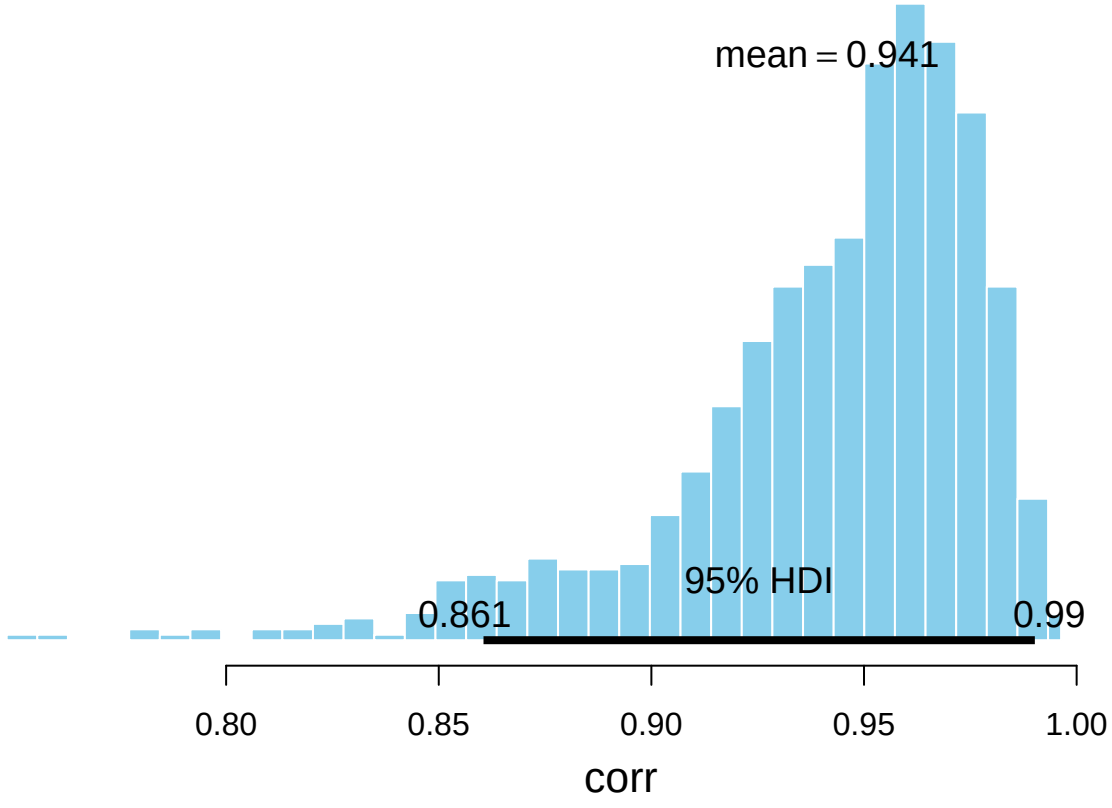
```
pop.sd <- function(x) {
  n <- length(x)
  sd(x) * sqrt( (n - 1) / n)
}

b3 <- bayesboot(heights, pop.sd)
summary(b3)
```

```
## Bayesian bootstrap
##
```

```
## Number of posterior draws: 4000
##
## Summary of the posterior (with 95% Highest Density Intervals):
##  statistic     mean        sd  hdi.low hdi.high
##         V1 3.663116 0.7391744 2.230017 5.031605
##
## Quantiles:
##  statistic    q2.5%     q25%   median     q75%   q97.5%
##         V1 2.323455 3.121658 3.654356 4.159498 5.158932
##
## Call:
##  bayesboot(data = heights, statistic = pop.sd)
```

A Bayesian bootstrap analysis of a correlation coefficient

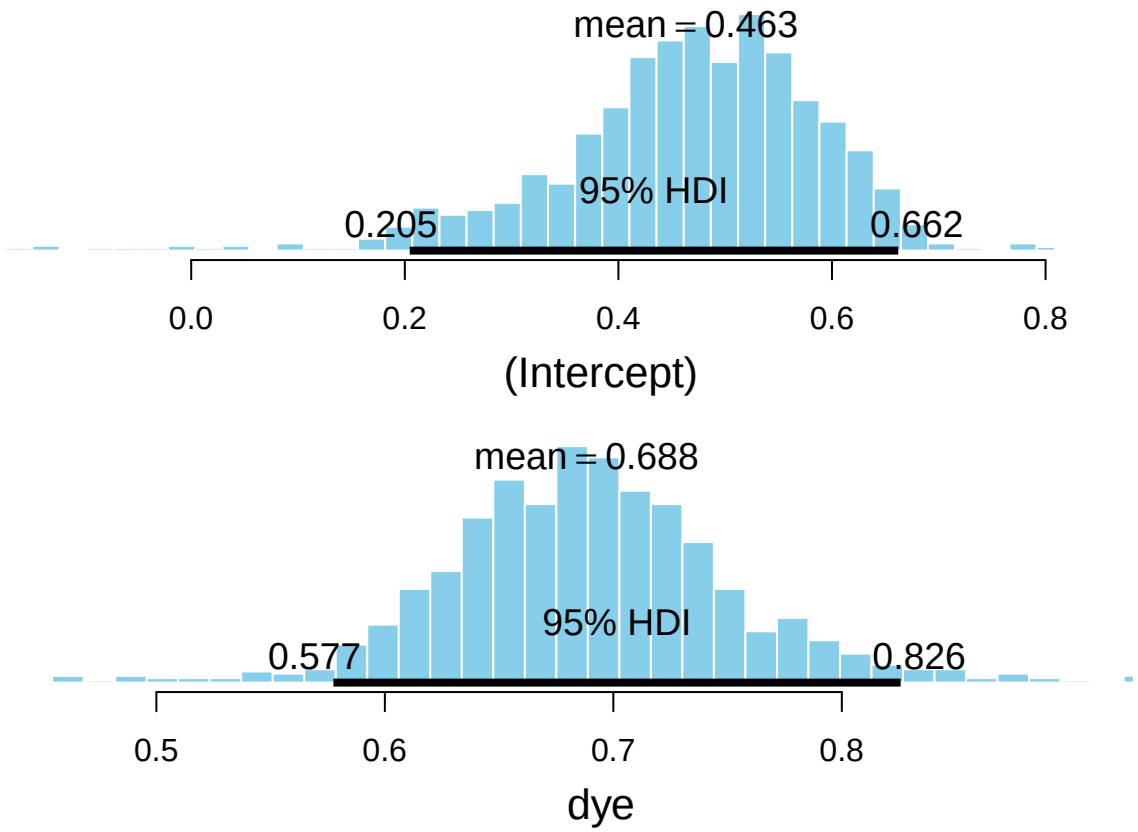
```
#Data comparing two methods of measuring blood flow.
blood.flow <- data.frame(
  dye = c(1.15, 1.7, 1.42, 1.38, 2.80, 4.7, 4.8, 1.41, 3.9),
  efp = c(1.38, 1.72, 1.59, 1.47, 1.66, 3.45, 3.87, 1.31, 3.75))

# Using the weighted correlation (corr) from the boot package.
library(boot)
b4 <- bayesboot(blood.flow, corr, R = 1000, use.weights = TRUE)
plot(b4)
```



A Bayesian bootstrap analysis of lm coefficients

```
# A custom function that returns the coefficients of
# a weighted linear regression on the blood.flow data
lm.coefs <- function(d, w) {
```

```
  coef( lm(efp ~ dye, data = d, weights = w) )
}

b5 <- bayesboot(blood.flow, lm.coefs, R = 1000, use.weights = TRUE)
plot(b5)
```



Bootstrappimine on väga hea meetod, mis sõltub väiksemast arvust eeldustest kui statistikas tavaks. Bootstrap ei eelda, et andmed on normaaljaotusega või mõne muu matemaatiliselt lihtsa jaotusega. Tema põhiline eeldus on, et valim peegeldab populatsiooni — mis ei pruugi kehtida väikeste valimite korral ja kallutatud (mitte-juhuslike) valimite korral.

Bootstrappimisel on üks konseptuaalne puudus, mis on eriti tõsine väikeste valimite korral. Võtame oma näite USA presidentidest. Meie valimi liikmed on kõik presideerinud viimase 50 aasta jooksul, aga selle aja jooksul on inimeste keskmine pikkus jõudsalt kasvanud. Kui me tahame ennustada järgmise 50 aasta keskmist presidentide pikkust, peaksime selle faktiga arvestama. Aga bootstrap ei jäta meile sellist võimalust. Vähemalt mitte lihtsat.

Siin tuleb appi Bayesi statistika oma täies hiilguses. Selles paradigmas ei arvesta me mitte ainult andmetega vaid ka taustateadmistega, sünteesides need kokku üheks posterioorseks jaotuseks ehk järeljaotuseks. Selle jaotuse arvutamine erineb bootstrapist, kuid tema tõlgendus ja praktiline töö sellega on samasugune. Erinevalt bootstrapist on Bayes parameetriline meetod, mis sõltub andmete modelleerimisest modeleerija poolt ette antud jaotustesse (normaaljaotus, t jaotus jne). Tegelikult peame me Bayesi arvutuseks modelleerima vähemalt kaks erinevat jaotust: andmete jaotus, mida me kutsume likelihoodiks ehk tõepäraks, ning eelneva teadmise mudel ehk prior, mida samuti modeleeritakse tõenäosusjaotusena.

11.1 3.2. Bayesi põhimõte

Bayesi arvutuseks on meil vaja teada

1) milline on "*parameter space*" ehk parameetriruum? Parameetriruum koosneb kõikidest loogiliselt võimalikest parameetriväärtustest. Näiteks kui me viskame ühe korra münti, koosneb parameetriruum kahest elemendist: 0 ja 1, ehk null kulli ja üks kull. See ammendab võimalike sündmuste nimekirja. Kui me aga hindame mõnd pidevat suurust (keskmine pikkus, tõenäosus 0 ja 1 vahel jms), koosneb parameetriruum lõpmata paljudest elementidest (arvudest).

2) milline on "*likelihood function*" ehk tõepärafunktsioon? Me omistame igale parameetriruumi elemendile (igale võimalikule parameetri väärtusele) tõepära. Tõepära parameetri väärtusel x on defineeritud kui tõenäosus, millega me võiksime kohata oma andmeid, juhul kui x oleks päriselt õige parameetri väärtus. Teisisõnu, tõepära on kooskõla määr andmete ja parameetri väärtuse x vahel. Tõepära = Pr(andmed I parameetri väärtus). Näiteks, kui tõenäolised on meie andmed, kui USA keskmine president on juhtumisi 183.83629 cm pikkune? Sageli on tõepära modelleeritud pideva tõenäosusfunktsioonina (näiteks normaaljaotusena), mis täielikult katab pideva väärtusruumi. Tõepärafunktsioon ei summeeru 100-le protsendile — see on normaliseerimata.

3) milline on "*prior function*" ehk prior? Kui tõepära on modelleeritud pideva funktsioonina siis on ka prior modelleeritud pideva tõenäosusjaotusena (aga ta ei pea olema samas jaotusmudelis, kus tõepära). Erinevus seisneb selles, et kui tõepärafunktsioon annab meie andmete tõenäosuse igal parameetriväärtusel, siis prior annab iga parameetriväärtuse tõenäosuse olukorras, kus me üldse ei arvesta oma andmete olemasoluga. Me omistame igale parameetriruumi väärtusele eelneva tõenäosuse, et see väärtus on üks ja ainus tõene väärtus. Prior jaotus summeerub 1-le. Prior kajastab meie konkreetsetest andmetest sõltumatut arvamust, kui suure tõenäosusega on just see parameetri väärtus tõene; seega seda, mida me usume enne oma andmete nägemist. Nendel parameetri väärtustel, kus prior (või tõepära) = 0%, on ka posteerior garanteeritult 0%. See tähendab, et kui te olete 100% kindel, et mingi sündmus on võimatu, siis ei suuda ka mäekõrgune hunnik uusi andmeid teie uskumust muuta (eelduselt, et te olete ratsionaalne inimene).

http://optics.eee.nottingham.ac.uk/match/uncertainty.php aitab praktikas priorit modelleerida (proovige *Roulette* meetodit).

Kui te eelnevast päriselt aru ei saanud, ärge muretsege. Varsti tulevad puust ja punaseks näited likelihoodi ja priori kohta.

Edasi on lihtne. Arvuti võtab tõepärafunktsiooni ja priori, korrutab need üksteisega läbi ning seejärel normaliseerib saadud jaotuse nii, et jaotusealune pindala võrdub ühega. Saadud tõenäosusjaotus ongi posteeriorne jaotus ehk posteerior ehk järeljaotus. Kogu lugu.

Me teame juba palju aastakümneid, et Bayesi teoreem on sellisele ülesandele parim võimalik lahendus. Lihtsamad ülesanded lahendame me selle abil täiuslikult. Kuna parameetrite arvu kasvuga mudelis muutub Bayesi teoreemi läbiarvutamine eksponentsiaalselt arvutusmahukamaks (sest läbi tuleb arvutada kõikide parameetrite väärtuste kõikvõimalikud kombinatsioonid), oleme sunnitud vähegi keerulisemad ülesanded lahendama umbkaudu, asendades Bayesi teoreemi *ad hoc* MCMC algoritmidega, mis teie arvutis peituva propelleri Karlsoni kombel lendu saadavad selleks, et "otse" sämplida arve posterioorsest jaotusest. Meie kasutatava MCMC *Hamiltonian Monte Carlo* mootori nimi on Stan. See on R-st eraldiseisev programm, millel on aga R-i interface R-i pakettide rstan(), rethinking(), rstanarm() jt kaudu. Meie töötame ka edaspidi puhtalt R-s, mis automaatselt suunab meie mudelid ja muud andmed Stani, kus need läbi arvutatakse ja seejärel tulemused R-i tagasi saadetakse. Tulemuste töötlus ja graafiline esitus toimub meie poolt jällegi R-is. Seega ei pea me ise kordagi Stani avama. (Stanil on sarnased interfaced ka Pythoni, Matlabi jt keeltega.)

Alustame siiski lihtsa näitega, mida saab käsitsi läbi arvutada.

11.1.1 1. näide

Me teame, et suremus haigusesse on 50% ja meil on palatis 3 patsienti, kes seda haigust põevad. Seega on meil kaks andmetükki (50% ja n=3). Küsimus: mitu meie patsienti oodatavalt hinge heidavad? Eeldusel, et meie patsiendid on iseseisvad (näiteks ei ole sugulased), on meil tüüpiline mündiviske olukord.

Parameetriruum: 0 elus, 1 elus, 2 elus ja 3 elus. Seega on meil neljaliikmeline parameetriruumi.

Edasi loeme üles kõik võimalikud sündmusteahelad, mis saavad loogiliselt juhtuda (et saada tõepärafunktsioon).

Me heidame münti 3 korda: H - kiri, T - kull

Võimalikud sündmused on:

HHH, HTH, THH, HHT, HTT, TTH, THT, TTT,

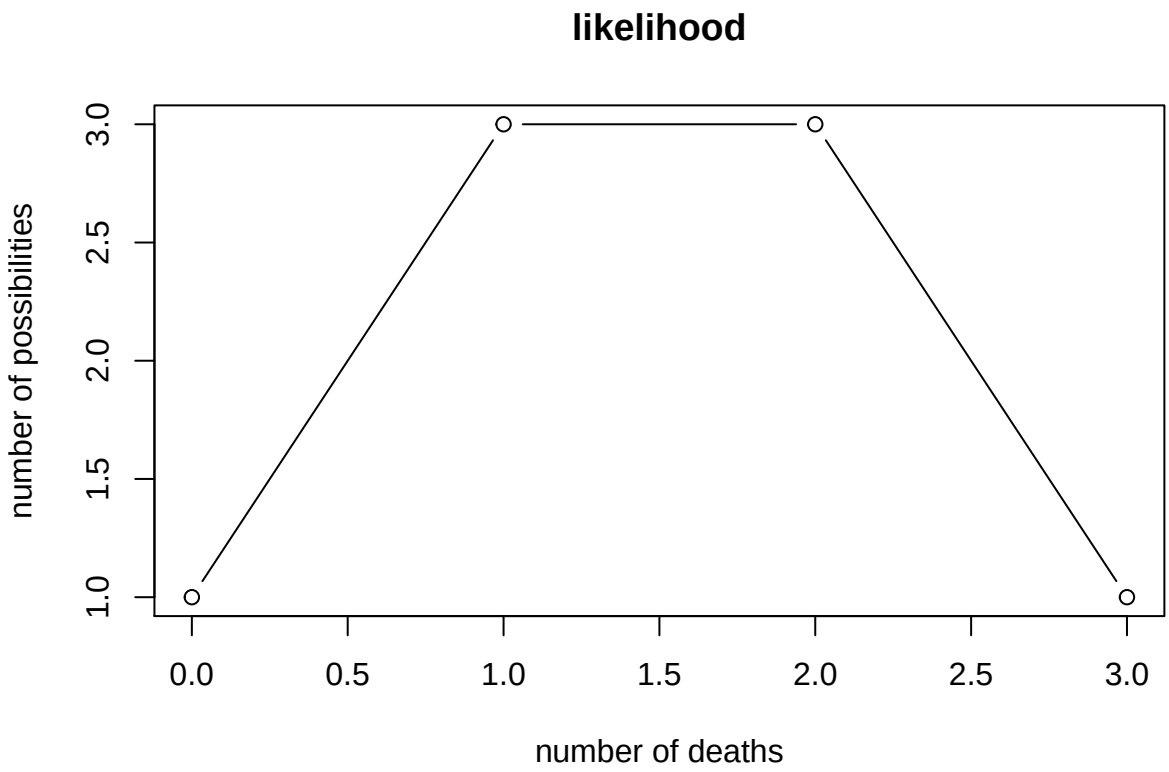
Kui Pr(H) = 0.5 ning H = elus ja T = surnud, siis lugedes kokku kõik võimalikud sündmused:

- 0 elus - 1,
- 1 elus - 3,
- 2 elus - 3,
- 3 elus - 1

Nüüd teame parameetriruumi igale liikme kohta, kui suure tõenäosusega me ootame selle realiseerumist. Näiteks, Pr(0 elus) = 1/8, Pr(1 elus) = 3/8, Pr(1 või 2 elus) = 6/8 jne

Selle teadmise saab konverteerida tõepärafunktsiooniks

```
x<-seq(from= 0, to=3) #parameter space as a grid
y<-c(1,3,3,1) #likelihood -
plot(x,y, ylab="number of possibilities", xlab="number of deaths", type="b", main="likelihood")
```

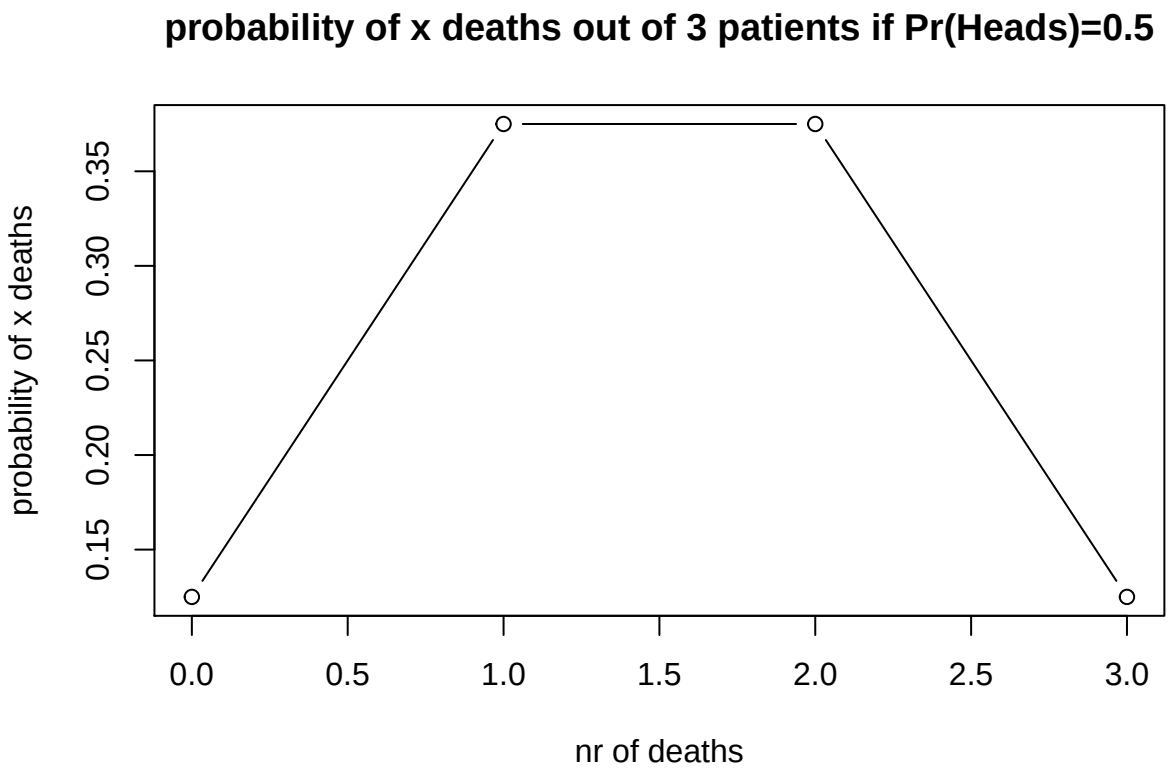


Siit näeme, et üks surm ja kaks surma on sama tõenäolised ja üks surm on kolm korda tõenäolisem kui null surma (või kolm surma).

Tõepära annab meile tõenäosuse Pr(mortality=0.5 & N=3) igale loogiliselt võimalikule surmade arvule (0 kuni 3).

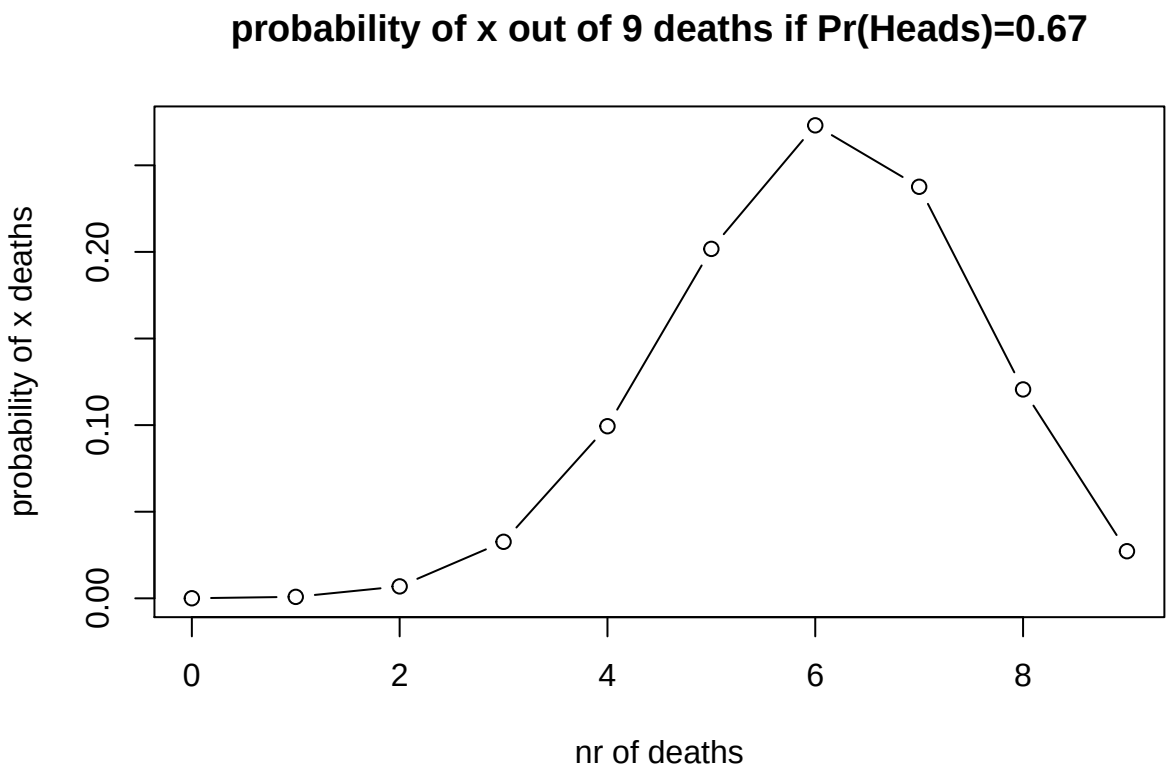
Me saame sama tulemuse kasutades binoomjaotuse mudelit (sest mitteformaalselt kasutasime ka ennist sama mudelit).

```
x<-seq(0, 3)
y <- dbinom(x,3, 0.5)
plot(x, y, type="b", xlab="nr of deaths", ylab="probability of x deaths", main="probability of x deaths
```



Nüüd proovime seda koodi olukorras, kus meil on 9 patsienti ja suremus on 0.67

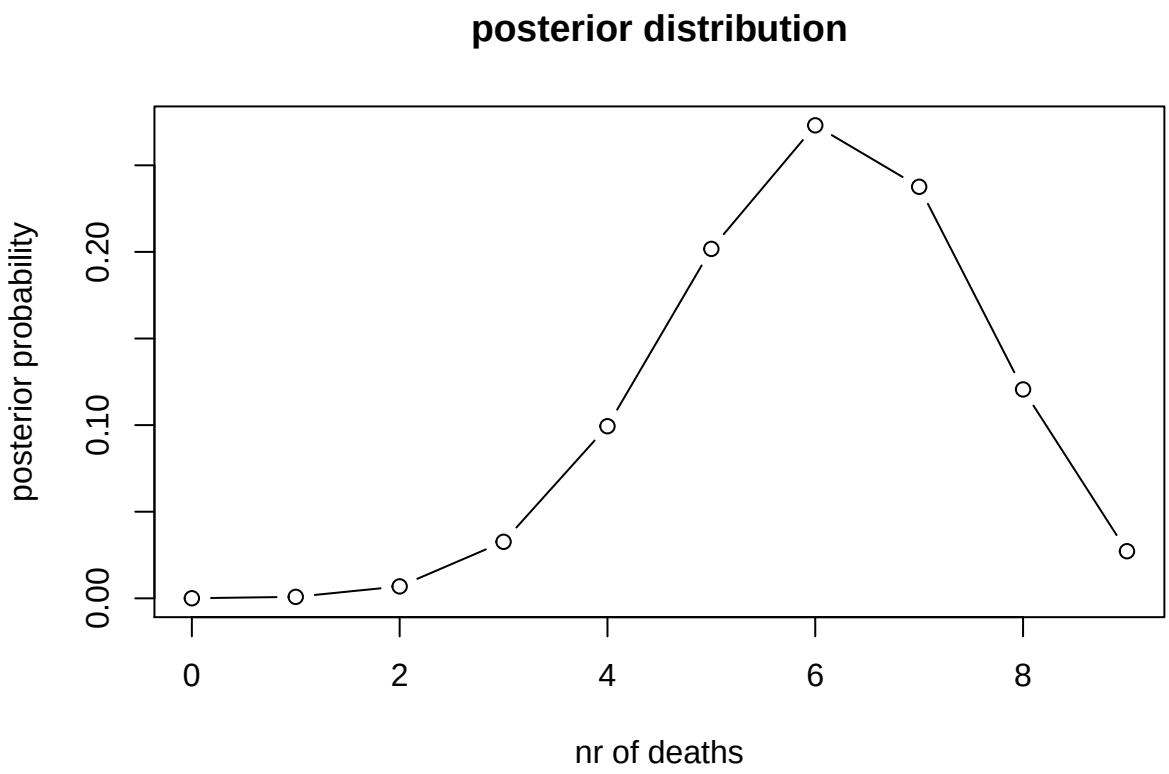
```
x<-seq(0, 9)
y <- dbinom(x,9, 0.67)
plot(x, y, type="b", xlab="nr of deaths", ylab="probability of x deaths", main="probability of x out of
```



Lisame siia tasase priori (lihtsuse huvides) ja arvutame posterioorse jaotuse kasutades Bayesi teoreemi. Igale parameetri väärtusele, tõepära * prior on proportsionaalne posterioorse tõenäosusega, et just see parameetri väärtus on see ainus tõene väärtus. Posterioorsed tõenäosused normaliseeritakse nii, et nad summeeruksid 1-le.

Me defineerime X telje kui rea 10-st arvust (0 kuni 9 surma) ja arvutame tõepära igale neist 10-st arvust. Sellega ammendame me kõik loogiliselt võimalikud parameetri väärtused.

```
# define grid
p_grid <- seq( from=0 , to=9 )
# define flat prior
prior <- rep( 1 , 10 )
# compute likelihood at each value in grid
likelihood <- dbinom( p_grid , size=9 , prob=0.67 )

# compute product of likelihood and prior
unstd.posterior <- likelihood * prior

# normalize the posterior, so that it sums to 1
posterior <- unstd.posterior/sum(unstd.posterior)
# sum(posterior) == 1

plot( x = p_grid , y = posterior , type="b" ,
    xlab="nr of deaths" , ylab="posterior probability", main="posterior distribution" )
```



See on parim võimalik teadmine, mitu kirstu tasuks tellida, arvestades meie priori ja likelihoodi mudelitega. Näiteks, sedapalju, kui surmad ei ole üksteisest sõltumatud, on meie tõepäramudel (binoomjaotus) vale.

11.1.2 2. näide: sõnastame oma probleemi ümber

Mis siis, kui me ei tea suremust ja tahaksime seda välja arvutada? Kõik, mida me teame on, et 6 patsienti 9st surid. Nüüd koosnevad andmed 9 patsiendi morbiidsusinfost (parameeter, mille väärtust me eelmises näites arvutasime) ja parameeter, mille väärtust me ei tea, on surmade üldine sagedus (see parameeter oli eelmises näites fikseeritud, ja seega kuulus andmete hulka).

Seega on meil 1) parameetriruum 0% kuni 100% suremus (0st 1-ni), mis sisaldab lõpmata palju numbreid.

2) kaks võimalikku sündmust (surnud, elus), seega binoomjaotusega modelleeritud tõepärafunktsioon. Nagu me juba teame, on r funkrsioonis dbinom() kolm argumenti: surmade arv, patsientide koguarv ja surmade tõenäosus. Seekord oleme me fikseerinud esimesed kaks ja soovime arvutada kolmanda.

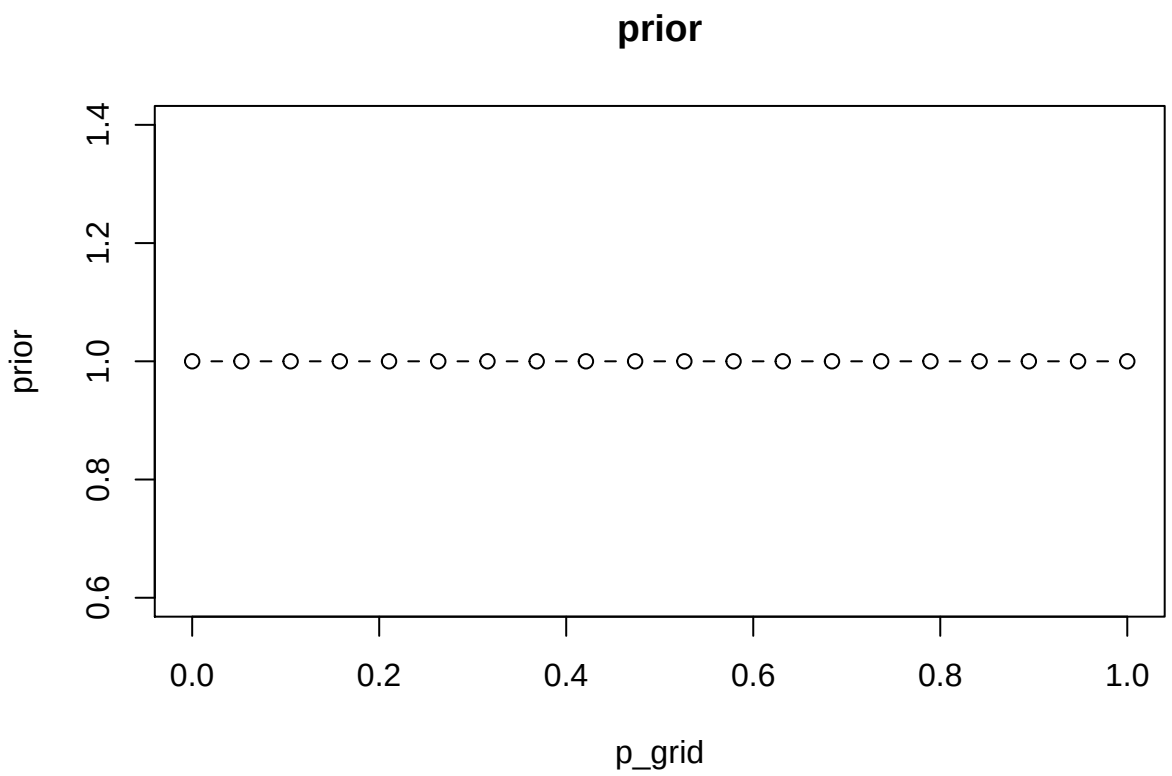
3) tasane prior, mis ulatub 0 ja 1 vahel. Me valisime selle priori selleks, et mitte muuta tõepärafunktsiooni kuju. See ei tähenda, et me arvaksime, et tasane prior on mitteinformatiivne. Tasane prior tähendab, et me usume, et suremuse kõik väärtused 0 ja 1 vahel on võrdselt tõenäolised. See on vägagi informatsioonirohke (ebatavaline) viis maailma näha ükskõik mis haiguse puhul!

Tõepära parameetri väärtusel x on tõenäosus kohata meie andmeid, kui x on juhtumisi parameetri tegelik väärtus. Meie näites koosneb tõepärafunktsioon tõenäosustest, et kuus üheksast patsiendist surid igal võimalikul suremuse väärtusel (0…1). Kuna see on lõpmatu rida, teeme natuke sohki ja arvutame tõepära 20-l valitud suremuse väärtusel

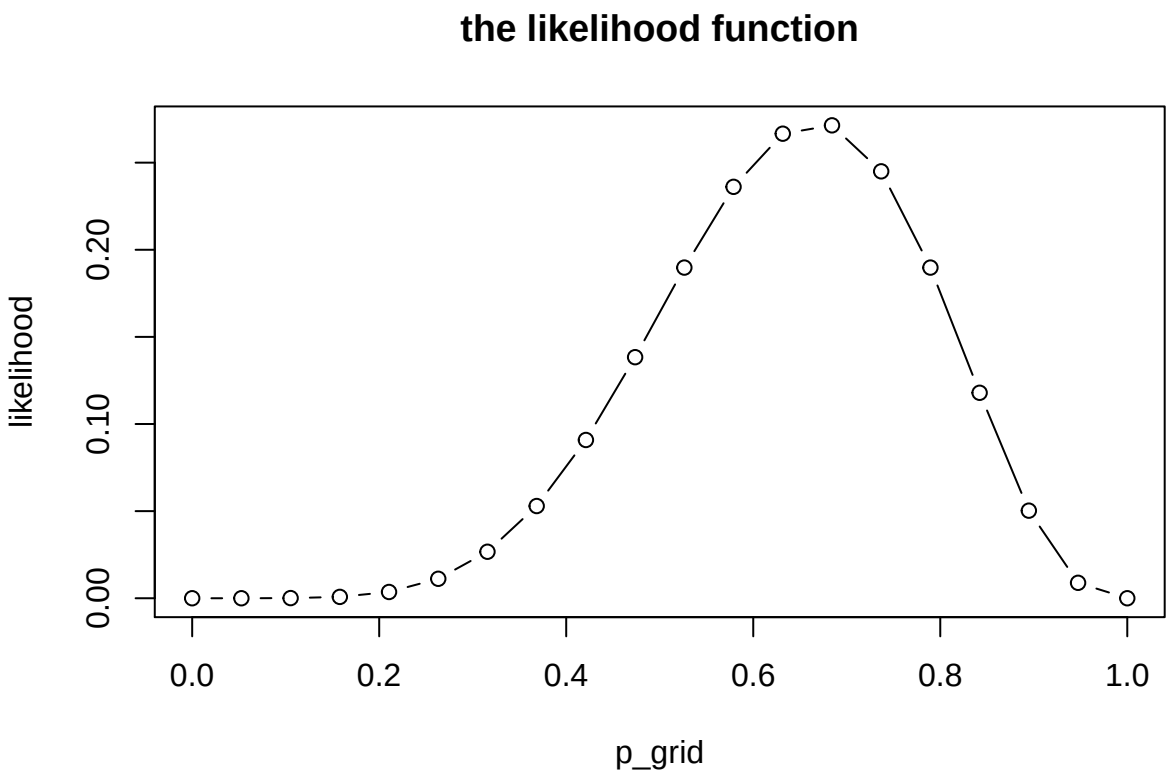
`Tehniliselt on sinu andmete tõepärafunktsioon agregeeritud iga üksiku andmepunkti tõepärafunktsioonist. Seega vaatab Bayes igat andmepunkti eraldi (andmete sisestamise järjekord ei loe).`

```
# define grid (mortality at 20 evenly spaced probabilities from 0 to 1)
p_grid <- seq( from=0 , to=1 , length.out=20 )
```

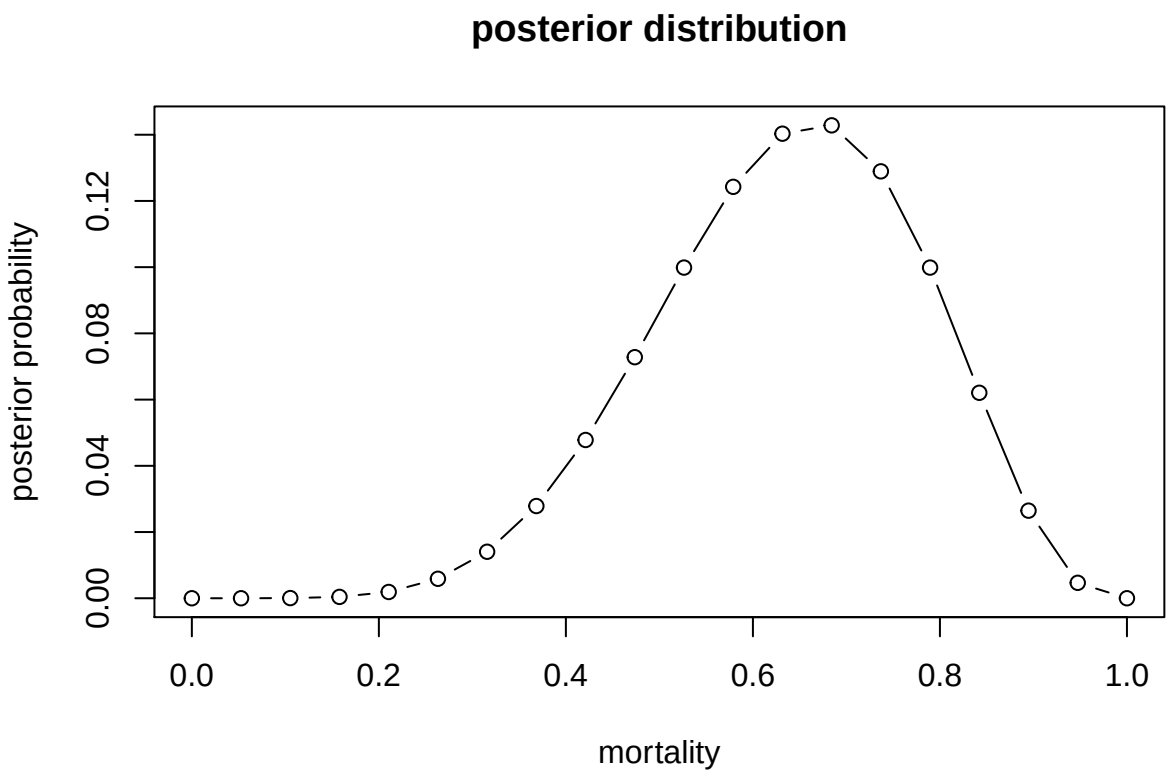
```
# define prior
prior <- rep( 1 , 20 )
# compute likelihood at each value in grid
likelihood <- dbinom( 6 , size=9 , prob=p_grid )
# compute product of likelihood and prior

plot(p_grid, prior, type="b", main="prior")
```




```
p_grid <- seq( from=0 , to=1 , length.out=20 )
likelihood <- dbinom( 6 , size=9 , prob=p_grid )
plot(p_grid, likelihood, type="b", main="the likelihood function")
```




```
unstd.posterior <- likelihood * prior
# standardize the posterior, so that it sums to 1
posterior <- unstd.posterior / sum(unstd.posterior)

plot( x = p_grid , y = posterior , type="b" ,
    xlab="mortality" , ylab="posterior probability", main="posterior distribution" )
```



Nüüd on meil posterioorne tõenäosusfunktsioon, mis summeerub 1-le ja mis sisaldab kogu meie teadmist suremuse kohta.

11.1.2.1 Kui n=1

Bayes on lahe sest tema hinnangud väiksele N-le on loogiliselt sama pädevad kui suurele N-le. See ei ole nii klassikalises sageduslikus statistikas, kus paljud testid on välja töötatud N = Inf eeldusel ja töötavad halvasti väikeste valimitega.

Hea küll, me arvutame jälle suremust.

Bayes töötab andmepunkti kaupa (see et me talle ennist kõik andmed korraga ette andsime, on puhtalt meie mugavus)

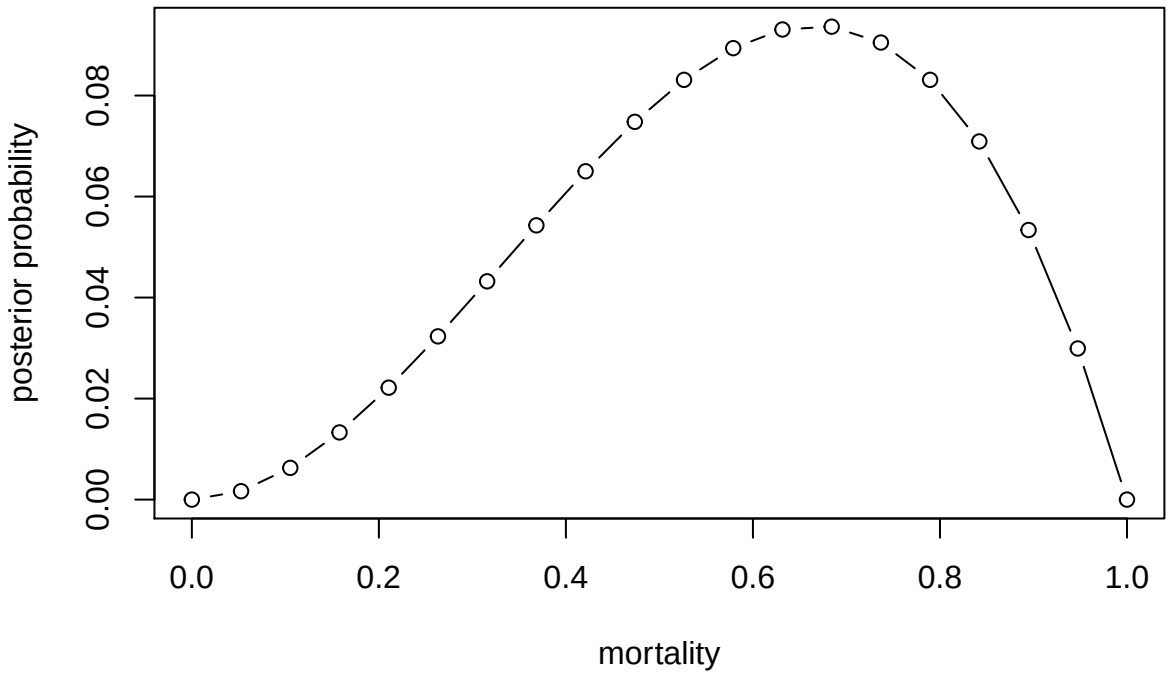
```
# define grid
p_grid <- seq( from=0 , to=1 , length.out=20 )
# define prior
prior <- rep( 1 , 20 )
# compute likelihood at each value in grid
likelihood <- dbinom( 1 , size=1 , prob=p_grid )
# compute product of likelihood and prior
unstd.posterior <- likelihood * prior
# standardize the posterior, so that it sums to 1
posterior <- unstd.posterior / sum(unstd.posterior)

plot( x = p_grid , y = posterior , type="b" ,
    xlab="mortality" , ylab="posterior probability" )
```

esimene patsient suri - 0 mortaalsus ei ole enam loogiliselt võimalik (välja arvatud siis kui prior selle koha peal = 0) ja mortaalsus 100% on andmetega (tegelikult andmega) parimini kooskõlas. Posteerior on nulli ja 100% vahel sirge sest vähene sissepandud informatsioon lihtsalt ei võimalda enamat.

```
# define grid
p_grid <- seq( from=0 , to=1 , length.out=20 )
# define prior
prior <- posterior
# compute likelihood at each value in grid
likelihood <- dbinom( 1 , size=1 , prob=p_grid )
# compute product of likelihood and prior
unstd.posterior <- likelihood * prior
# standardize the posterior, so that it sums to 1
posterior1 <- unstd.posterior / sum(unstd.posterior)

plot( x = p_grid , y = posterior1 , type="b" ,
    xlab="mortality" , ylab="posterior probability" )
```

Teine patsient suri. Nüüd ei ole 0 ja 1 vahel enam sirge posteerior. Posteerior on kaldu 100 protsendi poole, mis on ikka kõige tõenäolisem väärtus.

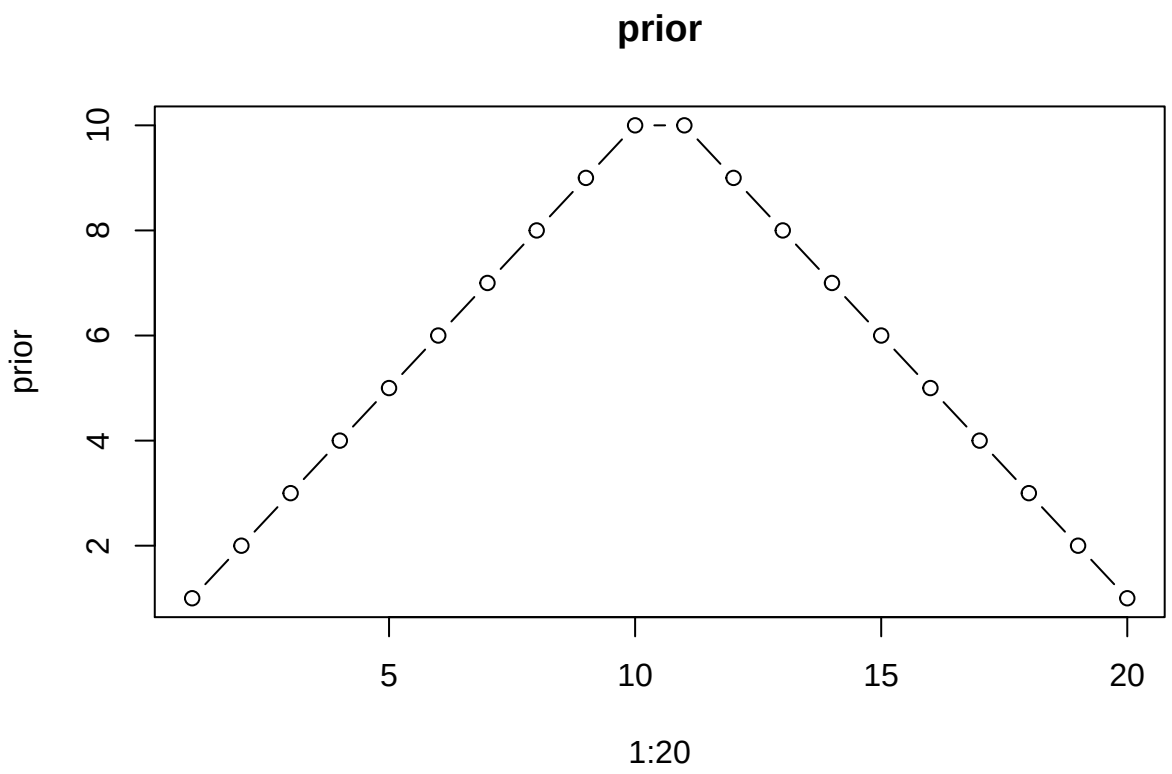
```
# define grid
p_grid <- seq( from=0 , to=1 , length.out=20 )
# define prior
prior <- posterior1
# compute likelihood at each value in grid
likelihood <- dbinom( 0 , size=1 , prob=p_grid )
# compute product of likelihood and prior
unstd.posterior <- likelihood * prior
# standardize the posterior, so that it sums to 1
posterior2 <- unstd.posterior / sum(unstd.posterior)

plot( x = p_grid , y = posterior2 , type="b" ,
    xlab="mortality" , ylab="posterior probability" )
```



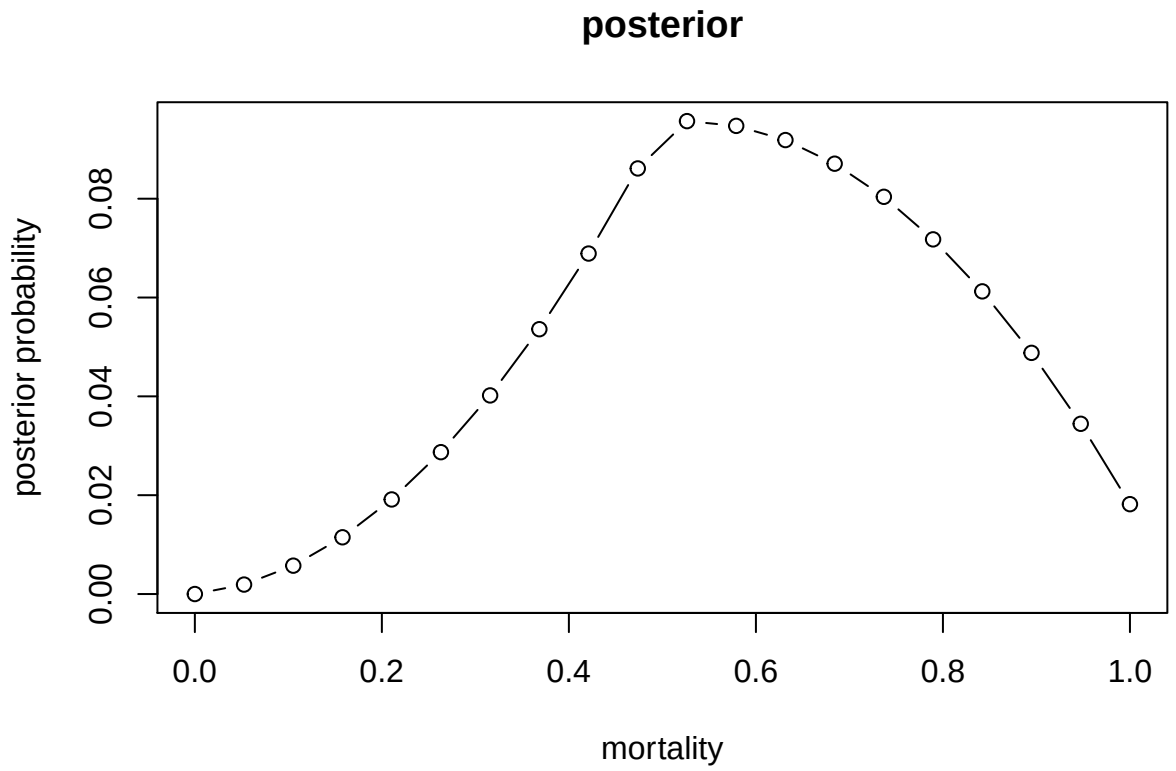
Kolmas patsient jäi ellu - 0 ja 100% mortaalsus on seega võimaluste nimekirjast maas ning suremus on ikka kaldu valimi keskmise poole (75%).

Teeme sedasama prioriga, mis ei ole tasane. See illustreerib tõsiasja, et kui N on väike siis domineerib prior posteerior jaotust. (Suure N korral on vastupidi, priori kuju on sageli vähetähtis.)

```
# define grid
p_grid <- seq( from=0 , to=1 , length.out=20 )
# define prior
prior <- c( seq(1:10), seq(from= 10, to= 1) )
# compute likelihood at each value in grid
likelihood <- dbinom( 1 , size=1 , prob=p_grid )
# compute product of likelihood and prior
unstd.posterior <- likelihood * prior
# standardize the posterior, so that it sums to 1
posterior <- unstd.posterior / sum(unstd.posterior)

plot(x=1:20, y=prior, type="b", main="prior")
```



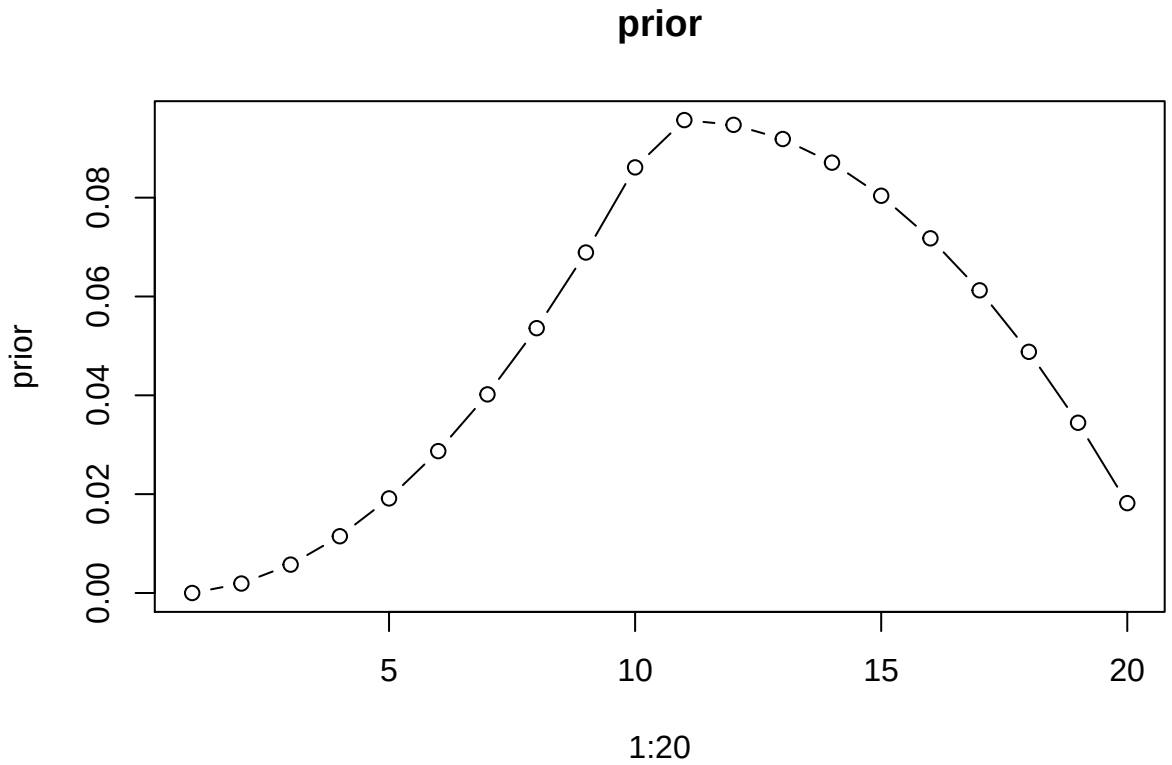

```
plot( x = p_grid , y = posterior , type="b" ,
    xlab="mortality" , ylab="posterior probability", main="posterior")
```



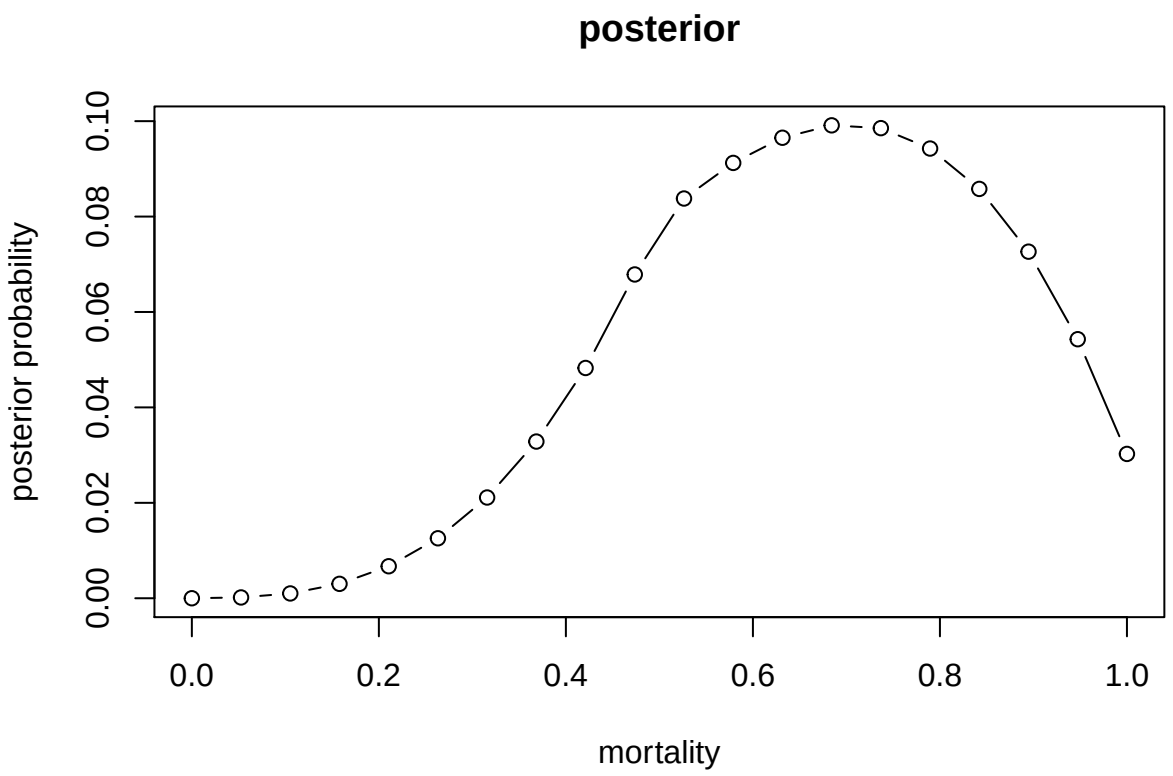
1.

patsient suri

```
# define grid
p_grid <- seq( from=0 , to=1 , length.out=20 )
# define prior
prior <- posterior
# compute likelihood at each value in grid
likelihood <- dbinom( 1 , size=1 , prob=p_grid )
# compute product of likelihood and prior
unstd.posterior <- likelihood * prior
# standardize the posterior, so it sums to 1
posterior1 <- unstd.posterior / sum(unstd.posterior)

plot(x=1:20, y=prior, type="b", main="prior")
```

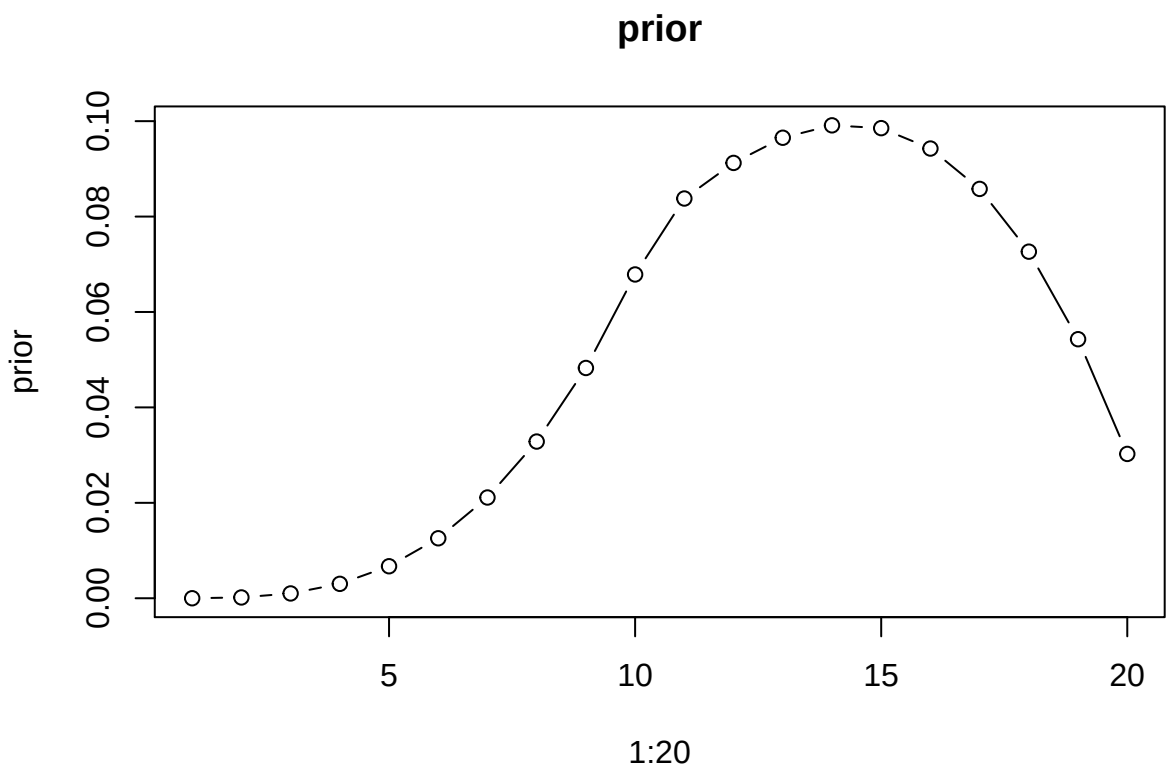



```
plot( p_grid , posterior1 , type="b" ,
    xlab="mortality" , ylab="posterior probability", main="posterior" )
```

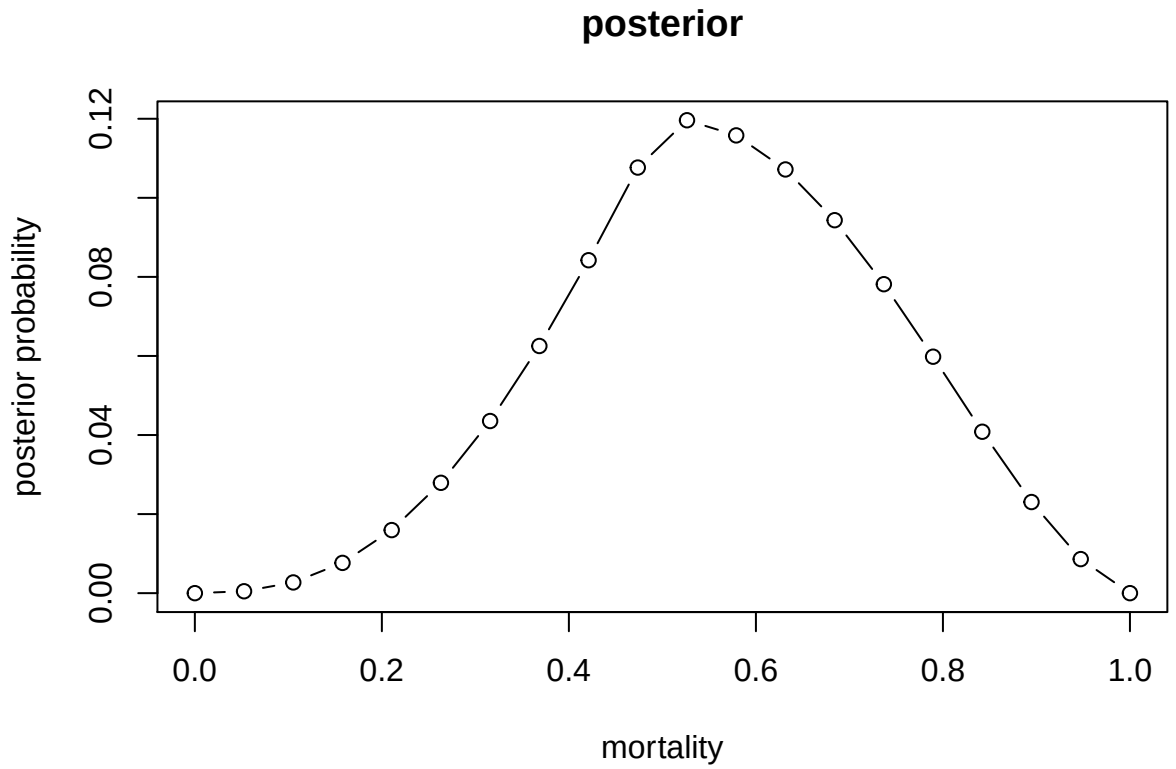


Teine patsient suri

```
# define grid
p_grid <- seq( from=0 , to=1 , length.out=20 )
# define prior
prior <- posterior1
# compute likelihood at each value in grid
likelihood <- dbinom( 0 , size=1 , prob=p_grid )
# compute product of likelihood and prior
unstd.posterior <- likelihood * prior
# standardize the posterior, so that it sums to 1
posterior2 <- unstd.posterior / sum(unstd.posterior)
plot(x=1:20, y=prior, type="b", main="prior")
```




```
plot( x = p_grid , y = posterior2 , type="b" ,
    xlab="mortality" , ylab="posterior probability", main="posterior"  )
```



3. patsient jäi ellu. Nüüd on posteeriori tipp mitte 75% juures nagu ennist, vaid kuskil 50% juures — tänu priorile.

11.2 2.3. Mudelite keel

Siin vaatame kuidas kirjeldada mudelit nii, et masin selle ära tunneb. Meie mudelid töötavad läbi rethinking() paketi. See raamatukogu pakub kaks võimalust, kuidas mudelit arvutada, mis mõlemad kasutavad sama notatsiooni. Mõlemad võimalused arvutavad posteeriori mitte Bayesi teoreemi kasutades (nagu me ennist tegime), vaid kasutades stohhastilisi meetodeid, mis iseloomustavad posteeriori umbkaudu (aga piisavalt täpselt). Põhjuseks on, et keerulisemate mudelite korral on Bayesi teoreemi kasutamine liialt arvutusmahukas.

Esiteks rethinking::map() leiab posteeriori tipu ja selle lähedal funktsiooni tõusunurga. Siin on eelduseks, et posteerior on normaaljaotus. See eeldus kehtib alati, kui nii prior kui tõepära on modelleeritud normaaljaotusena (ja ka paljudel muudel juhtudel).

Teine võimalus on rethinking::map2stan(), mis suunab teie kirjutatud mudeli Stan-i. Stan teeb *Hamiloniian Monte Carlo* simulatsiooni, kasutades valget maagiat selleks, et tõmmata valim otse posteerioorsest jaotusest. See on väga moodne lähenemine statistikale, töötab oluliselt aeglasemalt kui map, aga ei sõltu normaaljaotustest ning suudab arvutada hierarhilisi mudeleid, mis map-le üle jõu käivad.

Me võime sama mudeli kirjelduse sõõta mõlemasse funktsiooni.

Lihtne mudel näeb välja niimodi:

dead ~ dbinom(9, p) , # binomial likelihood

p ~ dunif(0, 1) # uniform prior

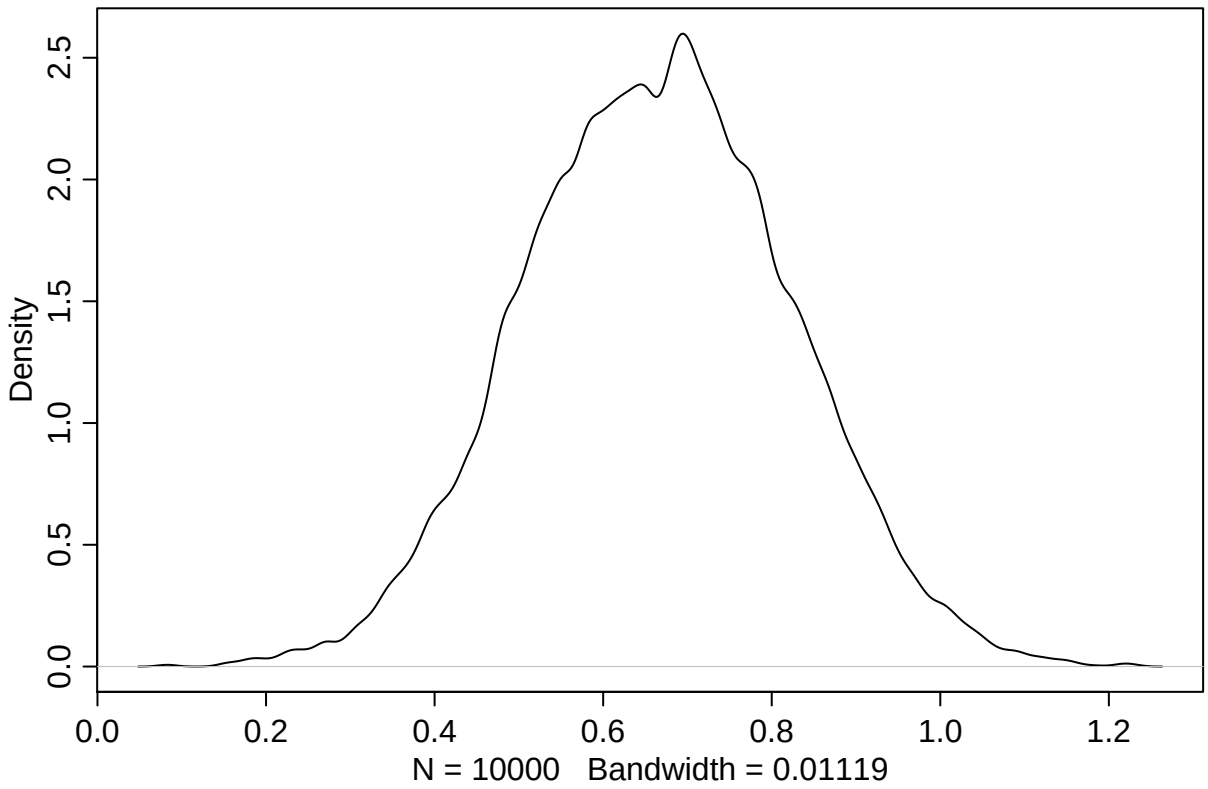
Tõepärafunktsioon on modeleeritud binoomjaotusena. Parameeter, mille väärtust määratakse on p, ehk suremus. See on ainus parameeter, mille väärtust me siin krutime. NB! igale parameetrile peab vastama oma prior. Meil on selles mudelis täpselt 1 parameeter ja 1 prior. Vastuseks saame selle ainsa parameetri posterioorse jaotuse. Hiljem näeme, et kui meil on näiteks 452 parameetrit, mille väärtusi me koos arvutame, siis on meil ka 452 priorit ja 452 posterioorset jaotust.

```
library(rethinking)
m1 <- map(
    alist(
        dead ~ dbinom(9, p) ,  # binomial likelihood
        p ~ dunif(0, 1)     # uniform prior
), data=list(dead=6) )

precis( m1 ) # summary of quadratic approximation
```

```
##   Mean StdDev 5.5% 94.5%
## p 0.67   0.16 0.42  0.92
```

Nüüd tõmbame posteerioroorsest jaotusest valimi n=10 000. Selleks on funktsioon extract.samples()

```
samples<-extract.samples(m1)
#hist(samples$p)
dens(samples$p)
```




```
#PI(samples$p, prob = 0.95) #leaves out equal 2.5% at both sides
HPDI(samples$p, prob = 0.95) #highest density 95% at the center
```

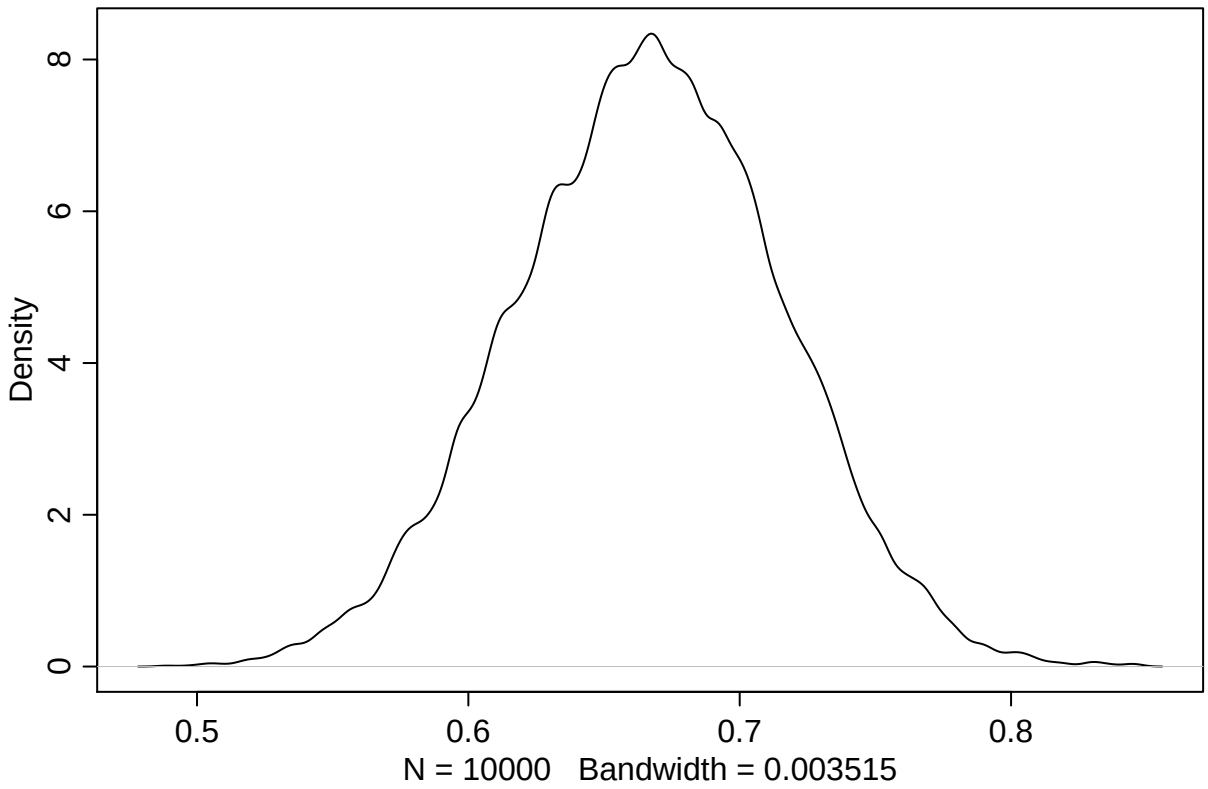
```
##     |0.95     0.95|
## 0.3592174 0.9678783
```

Kuus patsienti üheksast surid ja nüüd me usume, et tegelik suremus võib olla nii madal kui 37% ja nii kõrge kui 97%. Kui me tahame paremat hinnangut on meil vaja kas rohkem patsiente või informatiivsemat priorit (paremat taustainfot).

```
library(rethinking)
m2 <- map(
    alist(
        dead ~ dbinom(90,p) ,  # binomial likelihood
        p ~ dunif(0,1)     # uniform prior
), data=list(dead=60) )
# display summary of quadratic approximation
precis( m2 )
```

```
##   Mean StdDev 5.5% 94.5%
## p 0.67   0.05 0.59  0.75
```

```
samples<-extract.samples(m2)
dens(samples$p)
```




```
#PI(samples$p, prob = 0.95) #leaves out equal 2.5% at both sides
HPDI(samples$p, prob = 0.95) #highest density 95% at the center
```

```
##     |0.95     0.95|
## 0.5687602 0.7611587
```

10 korda rohkem andmeid: nüüd on suremus määratud kuskile 57% ja 77% vahele (suure tõenäosusega)

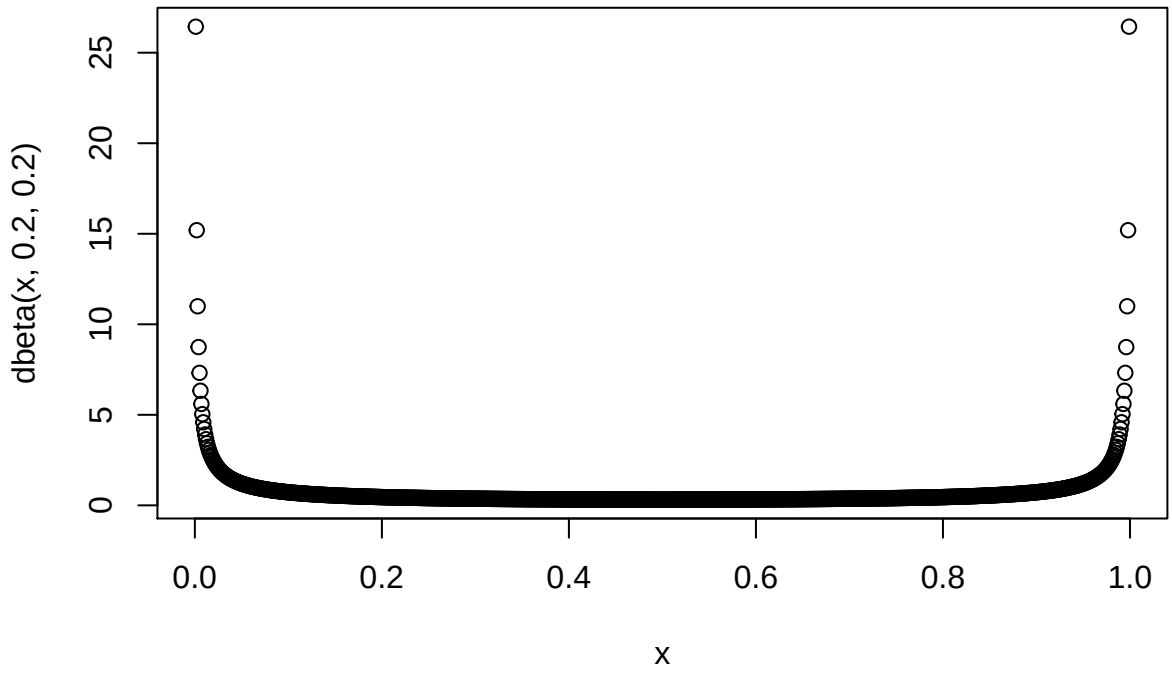
11.2.1 beta prior

Nüüd anname sisse mõistlikuma struktuuriga priori: beta-jaotuse

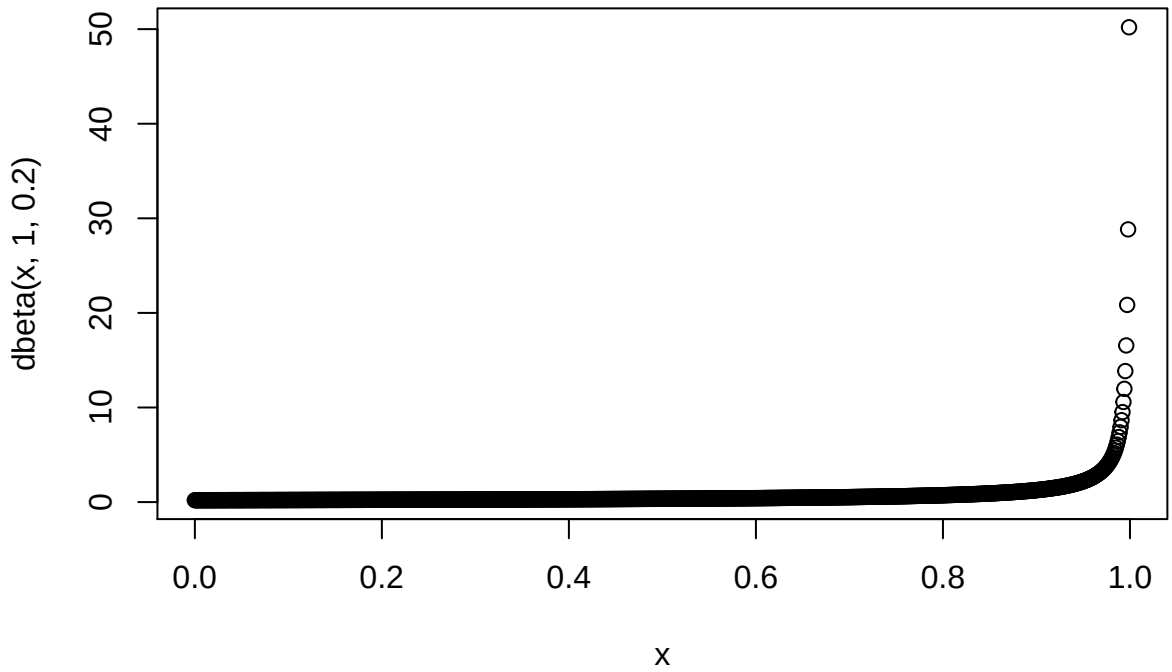
Beta-prior katab vahemiku 0st 1ni ja sellel on 2 parameetrit, a ja b.

Siin mõned näited erinevatest beta parametriseeringutest

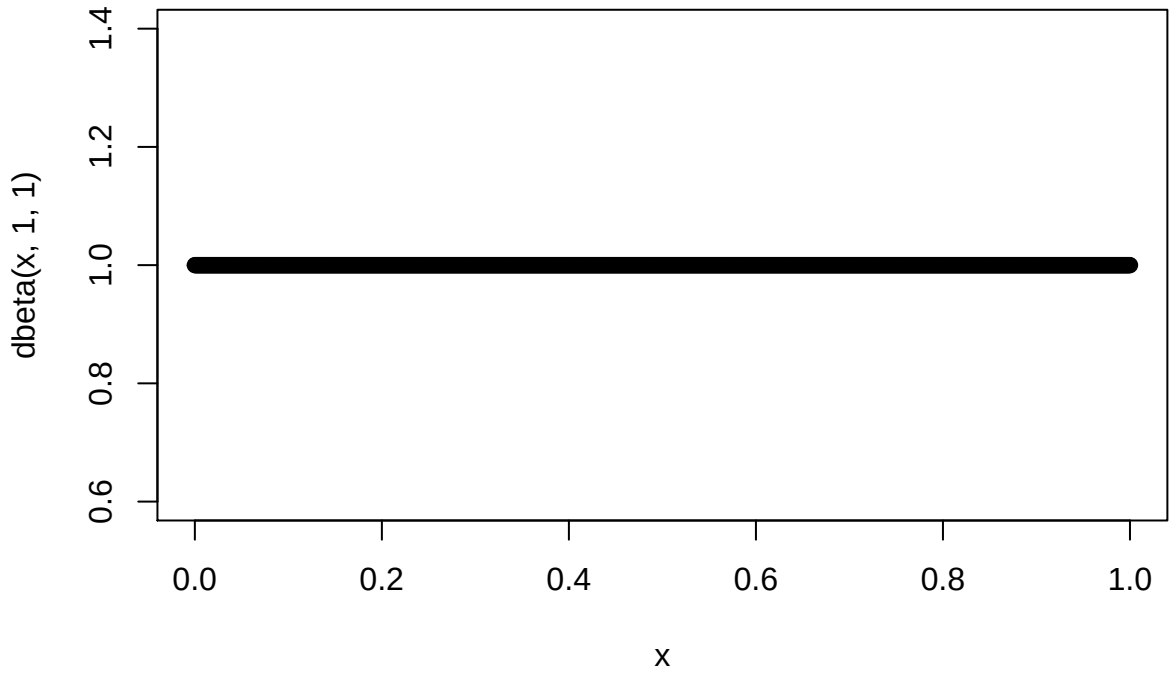
```
x <- seq(0, 1, length = 1000)
plot(x, dbeta(x, 0.2, 0.2))
```



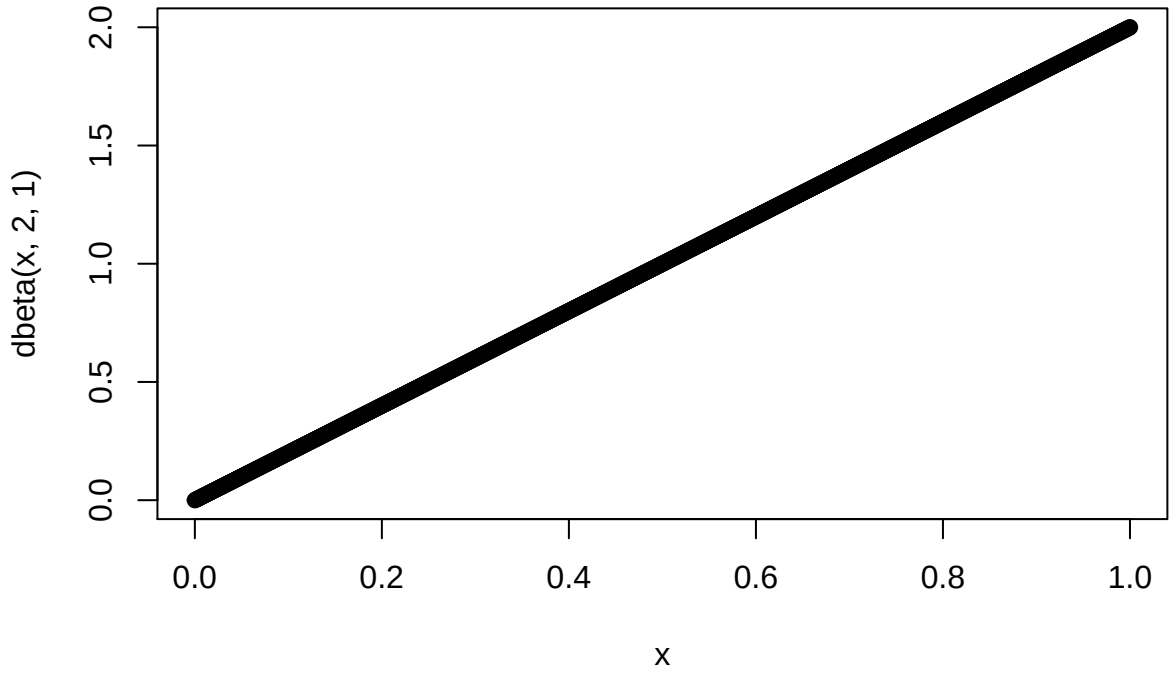

```
plot(x, dbeta(x, 1, 0.2))
```



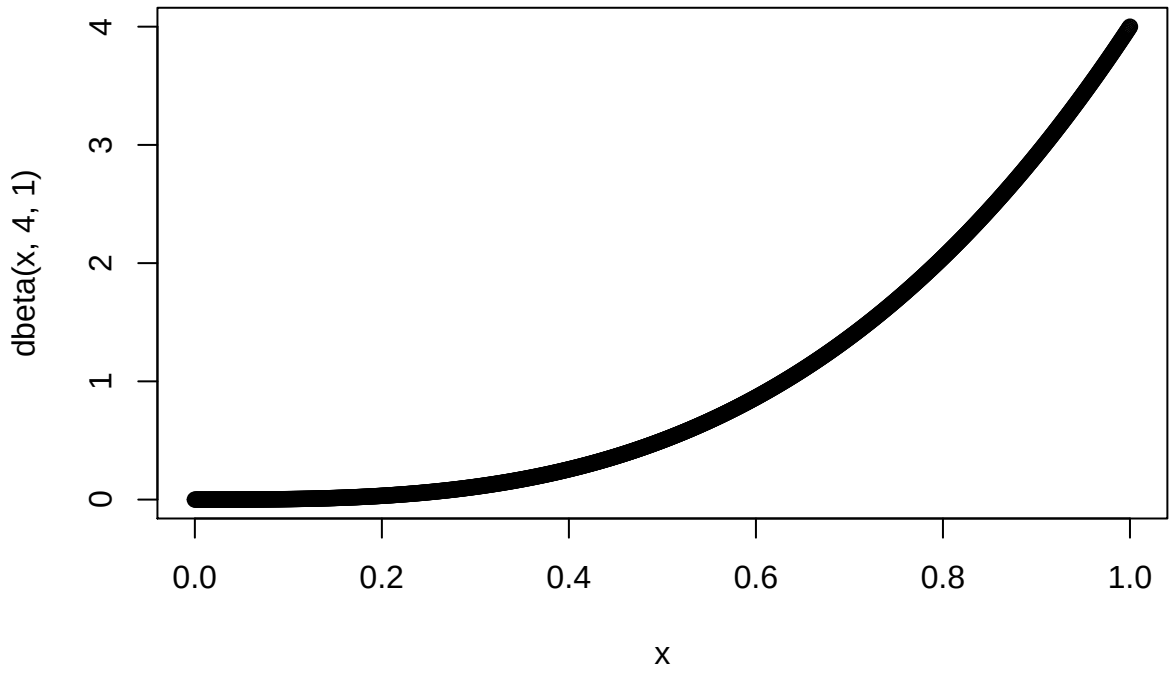

```
plot(x, dbeta(x, 1, 1))
```



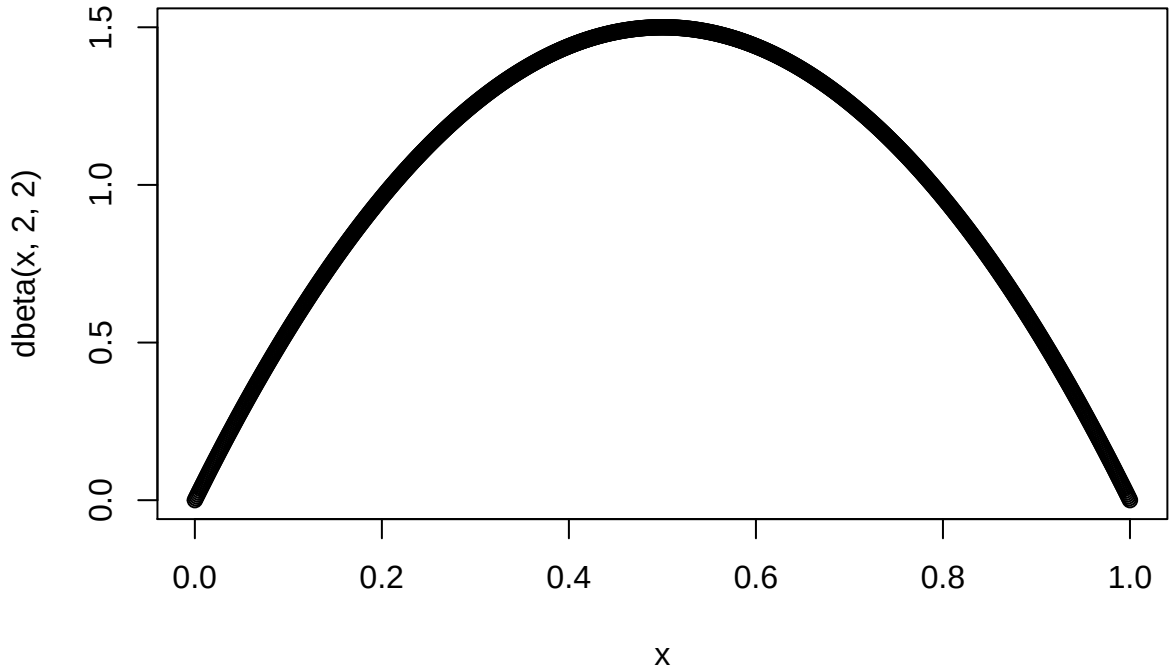

```
plot(x, dbeta(x, 2, 1))
```



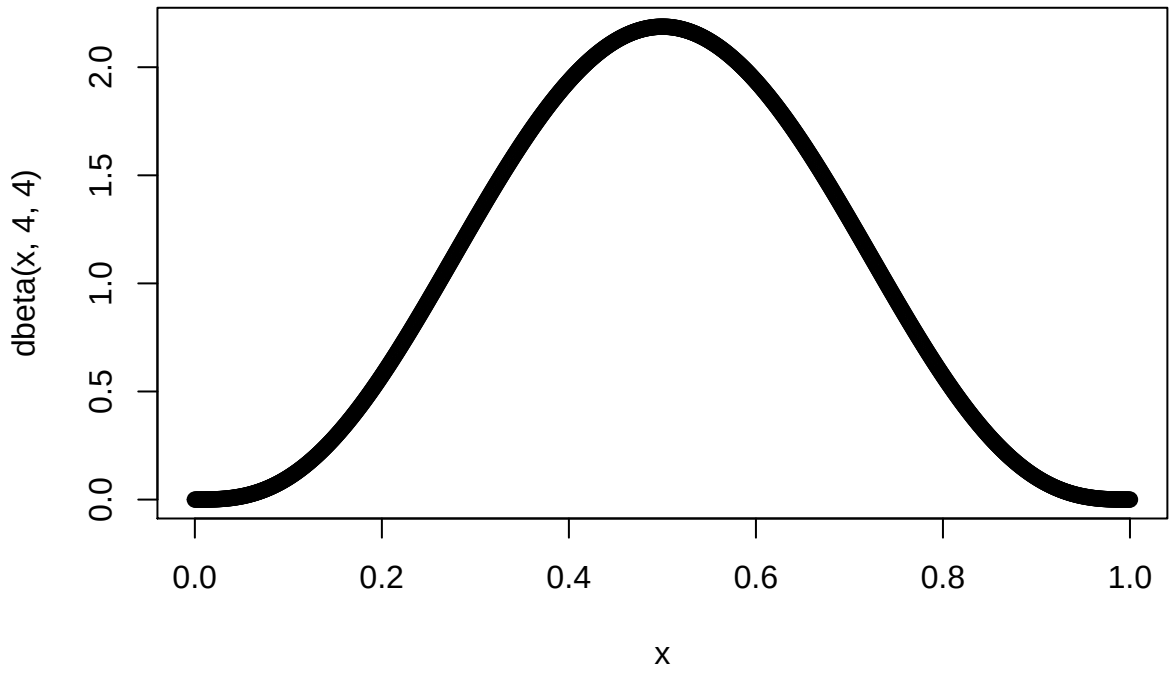

```
plot(x, dbeta(x, 4, 1))
```



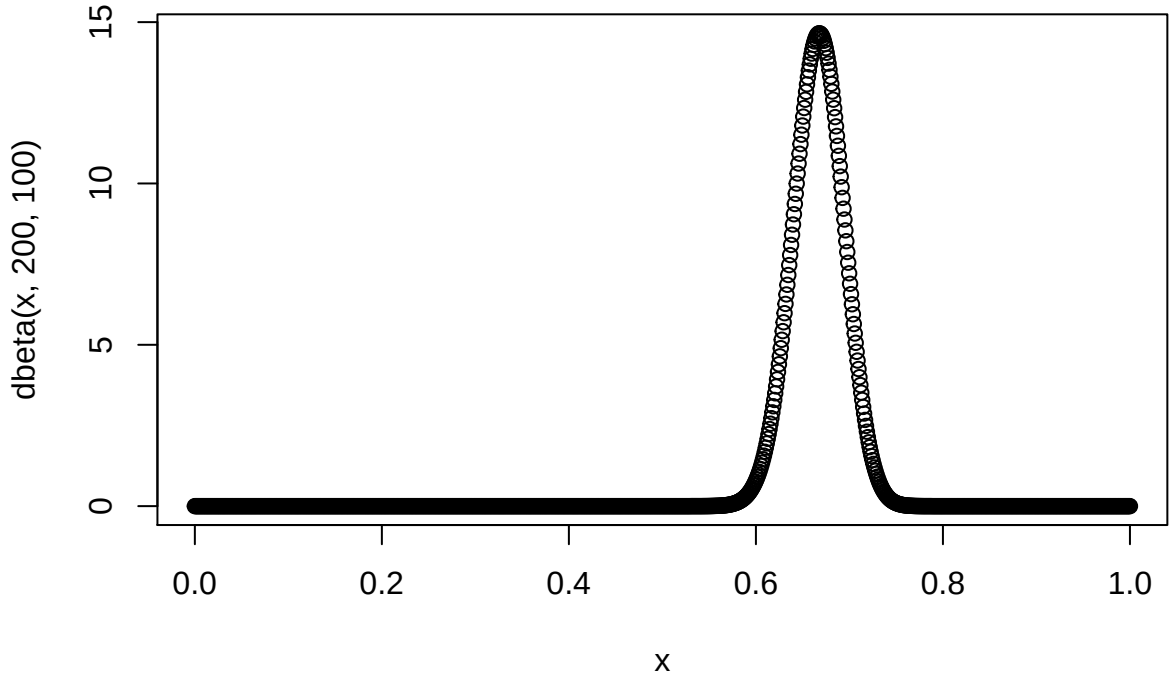

```
plot(x, dbeta(x, 2, 2))
```




```
plot(x, dbeta(x, 4, 4))
```




```
plot(x, dbeta(x, 200, 100))
```



beta(| a, b) jaotuse keskväärtus on

$= a/(a + b)$

ja mood on

$= (a - 1)/(a + b - 2)$ (juhul kui $a > 1$ ja $b > 1$).

Seega, kui a=b, siis on keskmine ja mood 0.5. Kui a > b, on keskmine ja mood > 0.5 ja kuid a < b, on mõlemad < 0.5.

Beta jaotuse "laiuse" annab "konsentratsioon" $= a + b$. Mida suurem , seda kitsam jaotus.

$a =$

$b = (1 - \ )$

$a = \ (\ - 2) + 1$

$b = (1 - \ )(\ - 2) + 1$ for $\ > 2$

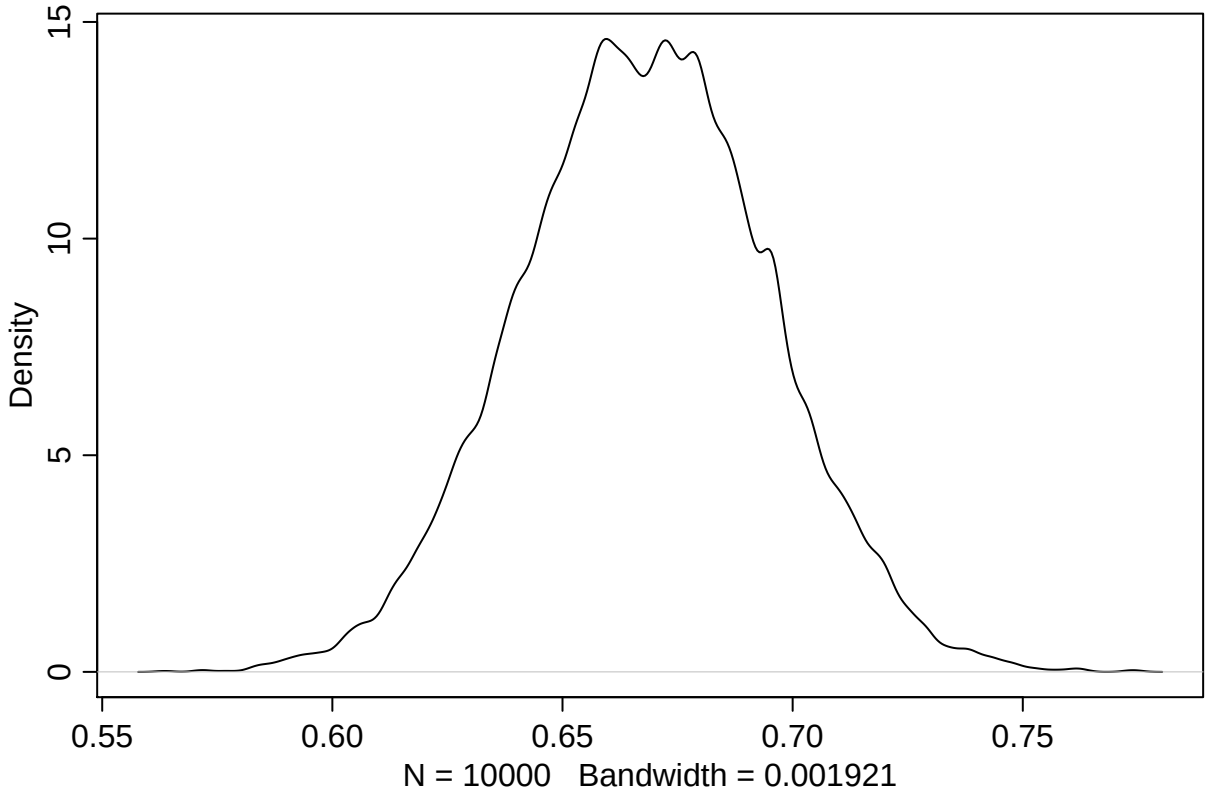
Me võime -le omistada väärtuse nagu see oleks mündivisete arv, mis iseloomustab meie priori tugevust (juhul kui tõepära funktsioon tuleb andmetest, mis koosnevad selle sama mündi visetest). Kui meie jaoks piisaks ainult mõnest mündiviset, et priorist (eelnevast teadmisest) lahti ütelda, peaks meie prior sisaldama väikest kappat.

Näiteks, mu prior on, et münt on aus (= 0.5; a = b), aga ma ei ole selles väga kindel.
Niisiis ma arvan, et selle eelteadmise kaal võrdub sellega, kui ma oleksin näinud 8 mündiviske tulemust.
Seega = 8, mis tähendab, et a = = 4 and b = (1 -) = 4.
Aga mis siis kui me tahame beta priorit, mille mood = 0.8 ja = 12?
Siis saame valemist, et a = 9 ja b = 3.

```
library(rethinking)
m3 <- map(
    alist(
        dead ~ dbinom(9,p) ,  # binomial likelihood
        p ~ dbeta(200,100)     # beta prior
), data=list(dead=6) )
# display summary of quadratic approximation
precis( m3 )
```

```
##   Mean StdDev 5.5% 94.5%
## p 0.67   0.03 0.62  0.71
```

```
samples<-extract.samples(m3)
dens(samples$p)
```




```
HPDI(samples$p, prob = 0.95) #highest density 95% at the center
```

```
##     |0.95     0.95|
## 0.6161019 0.7209454
```

Nagu näha on ka kitsa priori mõju üsna väika, isegi kui kui n=9.

11.2.1.1 Lõpetuseks veel prioritest üldiselt.

Neid võib jagada kolmeks: mitteinformatiivsed, väheinformatiivsed ehk "regularizing" ja informatiivsed. Mitteinformatiivseid prioreid ei ole sisuliselt olemas ja neid on soovitav vältida. Väheinformatiivsed priorid kujutavad endast kompromissi: nad muudavad võimalikult vähe tõepärafunktsiooni kuju, aga samas piiravad seda osa parameetriruumist, kust MCMC ahelad posteeriori otsivad (mis on arvutuslikult soodne). Nende taga on filosoofiline eeldus, et teadlast huvitavad eelkõige tema enda andmed ja see, mida need ühe või teise hüpoteesi (parameetri väärtuse) kohta ütlevad. See eeldus on vaieldav aga kui selle järgi käia, siis kulub vähem mõttejõudu eelteadmiste mudelisse formaliseerimiseks. Vähemalt suured farmaatsiafirmad seda hoiakut ei jaga ja kulutavad usinalt oma miljoneid korralike informatiivsete priorite tootmiseks. Selles protsessis saavad kokku statistikud, teaduseksperdid ja psühholoogid, et inimkonna teadmisi võimalikult adekvaatselt vormida tõenäosusjaotustesse. Meie töötame siin siiski enamasti väheinformatiivsete prioritega, mis on hetkel moes. Aga teile oma teaduses soovitan siiralt arendada informatiivseid prioreid. Vähemalt nõnda toimides te mõtlete oma teaduse üle põhjalikult järele.

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. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

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.