Reprodutseeritav andmeanalüüs kasutades R keelt

Taavi Päll, Ülo Maiväli

2017-09-19

Sisukord

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).

Peatükk 1

Sissejuhatus

See õpik on kirjutatud inimestele, kes kasutavad, mitte ei uuri, statistikat. Õpiku kasutaja peaks olema võimeline töötama R keskkonnas. Meie lähenemised statistika õpetamisele on arvutuslikud, mis tähendab, et me eelistame meetodi matemaatilise aluse asemel õpetada selle kasutamist ja tulemuste tõlgendamist. See õpik on bayesiaanlik ja ei õpeta sageduslikku statistikat. Me usume, et nii on lihtsam ja tulusam statistikat õppida ja et Bayesi statistikat kasutades saab rahuldada 99% teie tegelikest statistilistest vajadustest paremini, kui see on võimalik klassikaliste sageduslike meetoditega. Me usume ka, et kuigi praegused 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, 2015) 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.

Peatükk 2

Tarkvaratööriistad

2.1 Installeeri vajalikud programmid

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

1. Googelda "install R" või mine otse R allalaadimise 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.

2.2 Loo GitHubi konto

GitHub on veebipõhine versioonikontrolli repositoorium ja veebimajutuse teenus.

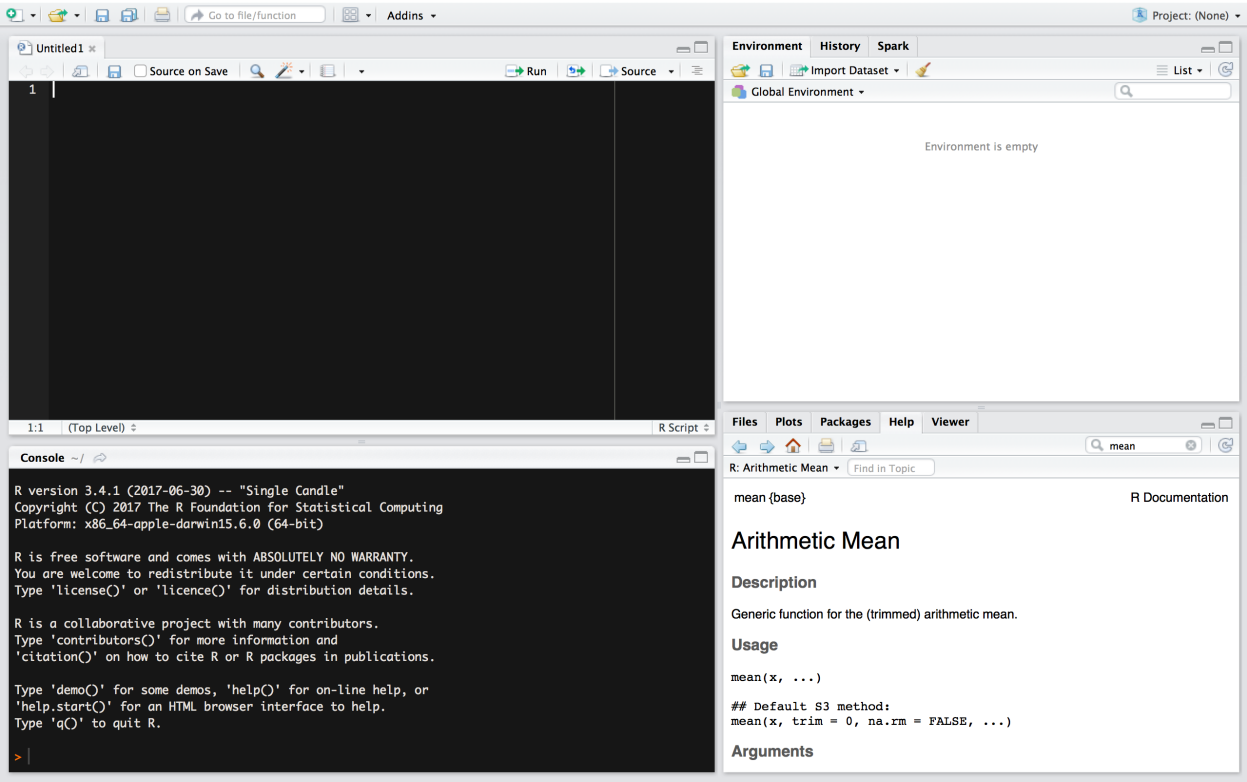
- konto loomiseks mine lehele https://github.com. Loo endale oma nimega seotud avalik konto. Tulevikule mõeldes vali kasutajanimi hoolikalt. Ära muretse detailide pärast, need on võimalik täita hiljem.
- Loo repo nimega `intro_demo`.
- Lisa repole lühike ja informatiivne kirjeldus.
- Vali "Public".
-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

2.3 Loo uus R projekt

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

4. Ava RStudio (R ise töötab taustal ja sa ei pea seda kunagi ise avama)
5. Ava RStudio akna (Joonis 2.1) 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.



Joonis 2.1: RStudio konsoolis on neli akent. Üleval vasakul on sinu poolt nimega varustatud koodi ja teksti editor kuhu kirjutad R skripti. Sinna kirjutad oma koodi ja kommentaarid sellele. All vasakul on konsool. Sinna sisestatakse käivitamisel sinu R kood ja sinna trükitakse väljund. Üleval paremal on Environment aken olulise sakiga . Seal on näha R-i objektid, mis on sulle töökeskkonnas kättesaadavad ja millega sa saad töötada. menüüs on võimalik muutusi vaadata ja 'commit'ida ja -ga suhelda. All paremal on paneel mitme sakiga. Files tab töötab nagu failihaldur. Kui sa lood või avad R projekti, siis näidatakse seal vaikimisi sinu töökataloogi. Kui kasutad R projekti, siis ei ole vaja töökataloogi eraldi seadistada. Plots paneelile ilmuvad joonised, mille sa teed. Packages näitab sulle sinu arvutis olevaid R-i pakette ehk raamatukogusid. Help paneeli avanevad help failid (ka need, mida konsooli kaudu otsitakse).

2.4 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.

2.5 R projekti kataloogi soovitatav minimaalne struktuur

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

R projekti kataloog peaks sisaldama projekti kirjeldavaid faile, mis nimetatakse DESCRIPTION ja README.md. **DESCRIPTION** on tavaline tekstifail ja sisaldab projekti metainfot ja infot projekti sõltuvuste kohta, nagu väliste 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).

2.6 Pakettide installeerimine

R *library*-d ehk paketid sisaldavad ühte või enamat mingit kindlat operatsiooni läbi viivat funktsiooni. **R baaspakett sisaldab juba mitmeid funktsioone.** Kõige esimene sõnum `sum()` help lehel on "sum {base}", mis tähendab, et see funktsioon kuulub nn. baasfunktsioonide hulka. Need funktsioonid on alati kättesaadavad sest neid sisaldavad raamatukogud laetakse vaikimisi teie töökeskkonda. Näiteks "base" raamatukogu versioon 3.4.1 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 installeerida pakett, sisesta järgnev käsurida R konsooli:

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

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

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

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

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

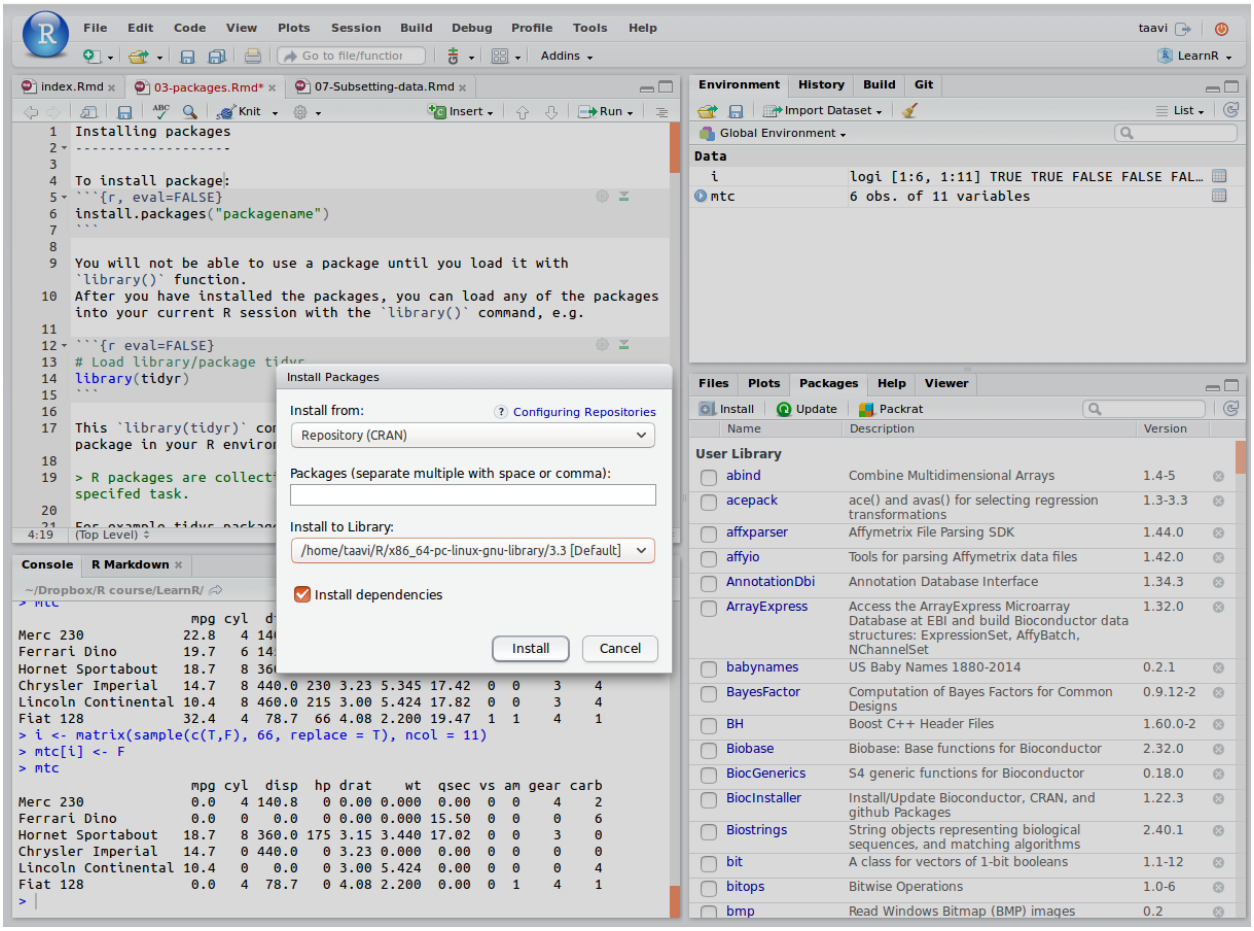
```
## Load library/package tidyr
library(dplyr)
```

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

Näiteks "dplyr" pakett sisaldab 237 funktsiooni:

```
library(dplyr)
## let's look at the head of package list
head(ls("package:dplyr"), 20)
```

```
##  [1] "%>%"           "add_count"     "add_count_"
##  [4] "add_row"       "add_rownames"  "add_tally"
##  [7] "add_tally_"    "all_equal"     "all_vars"
## [10] "anti_join"     "any_vars"      "arrange"
## [13] "arrange_"      "arrange_all"   "arrange_at"
## [16] "arrange_if"    "as_data_frame" "as_tibble"
## [19] "as.tbl"        "as.tbl_cube"
```



Joonis 2.2: RStudio 'Install Packages' dialoogiaken.

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

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

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

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

2.7 R repositooriumid

R pakid on saadaval kolmest põhilisest repositooriumist:

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

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

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

```
# First run biocLite script 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)
```

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

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

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

Peatükk 3

R on kalkulaator

Liidame `2 + 2`.

```
2 + 2
```

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

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

Kontrollime seda:

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

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

Vastus on TRUE, (logical).

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

Veel mõned näidisarvutused:

```
## 3 astmes 2; Please read Note ?'**'
3 ^ 2
## 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] 3.142
```

Ümardamine on oluline tulemuste väljaprintimisel.

3.1 Sama koodi saab kirjutada neljal viisil

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

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 *pipe*-s 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, klassikaline baas R lahendus:

```
a <- c(2, 3)
a1 <- round(sqrt(sum(a)), 2)
a1
```

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

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

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

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

Peatükk 4

R objektid

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

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

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

```
ls()
```

```
## [1] "a"      "a1"     "a2"     "a3"     "answer"
## [6] "result"
```

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

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

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

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

Kui on mingid kaua aega võtvad kalkulatsioonid või allalaadimised salvesta need eraldi .rds faili ja laadi koodis vastavalt vajadusele.

4.1 Objekt ja nimi

Kui teil sünnib laps, annate talle nime.

R-s on vastupidi: nimele antakse objekt

```
babe <- "beebi"
babe
```

```
## [1] "beebi"
```

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

NB! Stringid on jutumärkides, nimed mitte. Nimi üksi evalueeritakse kui "print object", mis antud juhul on string "beebi"

Nüüd muudame objekti nime taga:

```
babe <- c("saatan", "inglike")
babe
```

```
## [1] "saatan"  "inglike"
```

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

```
class(babe)
```

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

Antud juhul character.

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

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

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

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

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

```
##  num 5
```

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

4.2 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

4.3 Andmete tüübid

- numeric / integer
- logical – 2 väärtust TRUE/FALSE
- character

- factor (ordered and unordered) - 2+ diskreetset väärtust, mis võivad olla järjestatud suuremast väiksemani (aga ei asu üksteisest võrdsel kaugusel). Faktoreid käsitleme põhjalikumalt hiljem.

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

4.4 Vektor

Vektor on rida kindlas järjekorras arve, sõnu või TRUE/FALSE loogilisi väärtusi. Iga vektor ja maatriks (2D vektor) sisaldab ainult ühte tüüpi andmeid. Vektor on elementaarüksus, millega me teeme tehteid. Andmetabelis ripuvad kõrvuti ühepikad vektorid (üks vektor = üks tulp) ja R-le meeldib arvutada vektori kaupa vasakult paremale (mis tabelis on ülevalt alla sest vektori algus on üleval tabeli peas). Pikema kui üheelemendise vektori loomiseks kasuta funktsiooni `c()` – combine

Loome numbrilise vektori ja vaatame ta struktuuri:

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

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

Loome vektori puuduva väärtusega, vaatame vektori klassi:

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

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

```
class(minu_vektor)
```

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

Klass jääb *numeric*-uks.

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

```
minu_vektor <- c(1, "2", 2, 4, "joe")
minu_vektor
```

```
## [1] "1"   "2"   "2"   "4"   "joe"
```

```
class(minu_vektor)
```

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

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

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

```
as.numeric(minu_vektor)
```

```
## Warning: NAs introduced by coercion
```

```
## [1]  1  2  2  4 NA
```

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

```
minu_vektor <- factor(c(9, "12", 12, 1.4, "joe"))
minu_vektor
```

```
## [1] 9   12  12  1.4 joe
## Levels: 1.4 12 9 joe
```

```
class(minu_vektor)
```

```
## [1] "factor"
```

```
## Kui muudame faktori otse numbriks, saame faktori taseme numbri
as.numeric(minu_vektor)
```

```
## [1] 3 2 2 1 4
```

Faktorite muutmisel numbriteks tuleb need kõigepealt stringideks muuta:

```
as.numeric(as.character(minu_vektor))
```

```
## Warning: NAs introduced by coercion
```

```
## [1]  9.0 12.0 12.0  1.4   NA
```

Järgneva trikiga saab stringidest ekstraheerida numbrid:

```
library(readr)
minu_vektor <- c(1, "A2", "$2", "joe")
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  2  2 NA
```

```
str(minu_vektor)
```

```
##  num [1:4] 1 2 2 NA
```

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

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

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

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

```
## [1] 1 2 3 4 5
```

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

```
seq(2, 3, by = 0.5)
```

```
## [1] 2.0 2.5 3.0
```

```
seq(2, 3, length.out = 5)
```

```
## [1] 2.00 2.25 2.50 2.75 3.00
```

```
rep(1:2, times = 3)
```

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

```
rep(1:2, each = 3)
```

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

```
rep(c("a", "b"), each = 3, times = 2)
```

```
##  [1] "a" "a" "a" "b" "b" "b" "a" "a" "a" "b" "b" "b"
```

4.4.2 Tehted arvuliste vektoritega

Vektoreid saab liita, lahutada, korrutada ja jagada.

```
a <- c(1, 2, 3)
b <- 4
a + b
```

```
## [1] 5 6 7
```

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

```
a <- c(1, 2, 3)
b <- c(4, 5)
a + b
```

```
## Warning in a + b: longer object length is not a
## multiple of shorter object length
```

```
## [1] 5 7 7
```

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

```
a <- c(1, 2, 3, 4)
b <- c(5, 6)
a + b
```

```
## [1]  6  8  8 10
```

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

```
a <- c(1, 2, 3, 4)
b <- c(5, 6, 7, 8)
a + b
```

```
## [1]  6  8 10 12
```

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

```
a <- c(TRUE, FALSE, TRUE)
sum(a)
```

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

```
mean(a)
```

```
## [1] 0.6667
```

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.

4.5 List

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

Näiteks siin list, kus loksuvad 1 vektor nimega a, 1 tibble nimega b ja 1 list nimega c, mis omakorda sisaldab vektorit nimega d ja tibblet nimega e. Seega on meil tegu rekursiivse listiga.

```
# numeric vector a
a <- runif(5)
# data.frame
ab <- data.frame(a, b = rnorm(5))
# linear model
model <- lm(mpg ~ hp, data = mtcars)
# your grandma on bongos
grandma <- "your grandma on bongos"
# let's creat list
happy_list <- list(a, ab, model, grandma)
happy_list
```

```
## [[1]]
## [1] 0.6134 0.2364 0.5119 0.9812 0.6268
##
## [[2]]
##        a       b
## 1 0.6134 -0.3929
## 2 0.2364  0.7862
## 3 0.5119 -0.4579
## 4 0.9812 -1.1805
## 5 0.6268  0.5536
##
## [[3]]
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Coefficients:
## (Intercept)           hp
##     30.0989      -0.0682
##
##
## [[4]]
## [1] "your grandma on bongos"
```

Võtame listist välja elemndi “ab”:

```
happy_list$ab
```

```
## NULL
```

4.6 data frame ja tibble

```
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: data frame ja tibble, mis on väga sarnased. Tibble on uuem, veidi kaunima väljatrükiga, pisut mugavam kasutada.

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

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

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

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

```
## # A tibble: 3 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1 maxima      1       2 <chr [26]>
## 2  tesco      4      32 <dbl [10]>
## 3   lidl     43      NA   <S3: lm>
```

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

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

```
dfs <- try(data.frame(shop, apples, oranges, vabakava))
dfs
```

```
## [1] "Error in as.data.frame.default(x[[i]], optional = TRUE, stringsAsFactors = stringsAsFactors) : \n  c
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in as.data.frame.default(x[[i]], optional = TRUE, stringsAsFactors = stringsAsFactors): car
```

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
##  vabakava.Length  vabakava.Class  vabakava.Mode
##  26         -none-     character
##  10         -none-     numeric
##  12         lm         list
##
##
##
##
```

```
names(fruits)
```

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

```
colnames(fruits)
```

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

```
nrow(fruits)
```

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

```
ncol(fruits)
```

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

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

```
## # A tibble: 3 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1   lidl     43      NA   <S3: lm>
## 2  tesco      4      32 <dbl [10]>
## 3 maxima      1       2 <chr [26]>
```

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

```
## # A tibble: 2 x 4
##    shop apples oranges   vabakava
```

```
##   <chr>  <dbl>   <dbl>     <list>
## 1 tesco      4      32 <dbl [10]>
## 2  lidl     43      NA   <S3: lm>
```

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

```
## # A tibble: 2 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1 maxima      1       2 <chr [26]>
## 2  tesco      4      32 <dbl [10]>
```

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

```
colSums(fruits[ , 2:3])
```

```
##  apples oranges
##      48      NA
```

```
rowSums(fruits[ , 2:3])
```

```
## [1]  3 36 NA
```

```
rowMeans(fruits[ , 2:3])
```

```
## [1]  1.5 18.0   NA
```

```
colMeans(fruits[ , 2:3])
```

```
##  apples oranges
##      16      NA
```

```
fruits_subset <- fruits[ , 2:3]
# 1 tähendab, et arvuta sd rea kaupa
apply(fruits_subset, 1, sd)
```

```
## [1]  0.7071 19.7990      NA
```

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

```
##  apples oranges
##   23.43      NA
```

Lisame käsitsi meie tabelile 1 rea:

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

```
## # A tibble: 4 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1 maxima      1       2 <chr [26]>
## 2  tesco      4      32 <dbl [10]>
## 3 konsum    132      -5     <NULL>
## 4   lidl     43      NA   <S3: lm>
```

Proovi ise:

```
add_column()
```

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

4.6.1 Rekodeerime tibble väärtusi

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

```
## # A tibble: 4 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1 maxima      1       2 <chr [26]>
## 2  tesco      4      32 <dbl [10]>
## 3 konsum    132      -5     <NULL>
## 4   lidl    333      NA   <S3: lm>
```

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

```
## # A tibble: 4 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1 maxima      1       2 <chr [26]>
## 2  TESCO      4      32 <dbl [10]>
## 3 konsum    132      -5     <NULL>
## 4   lidl    333      NA   <S3: lm>
```

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

```
## # A tibble: 4 x 4
##     shop apples oranges   vabakava
##    <chr>  <dbl>   <dbl>     <list>
## 1 maxima      1       2 <chr [26]>
## 2  TESCO      4      32 <dbl [10]>
## 3 konsum     NA      -5     <NULL>
## 4   lidl     NA      NA   <S3: lm>
```

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

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

Rekodeerime `Inf` ja `NA` väärtused nulliks (väga halb mõte):

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

4.6.2 Ühendame kaks tibblet rea kaupa

Tabeli veergude arv ei muutu, ridade arv kasvab.

```
dfs <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
dfs1 <- tibble(colA = "d", colB =  4)
#id teeb veel ühe veeru, mis näitab, kummast algtabelist iga uue tabeli rida pärit on
bind_rows(dfs, dfs1, .id = "id")
```

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

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

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

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

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

```
df2 <- tibble(ColC = "d", ColD = 4)
## works by guessing your true intention
bind_rows(dfs1, df2)
```

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

4.6.3 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(dfs, dfx)
```

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

4.6.4 Nii saab tibblest kätte vektori, millega saab tehteid teha.

Tibble jääb muidugi endisel kujul alles.

```
ubinad <- fruits$apples
ubinad <- ubinad + 2
ubinad
```

```
## [1]  3  6 NA NA
```

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

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

4.6.5 Andmeraamide salvestamine (eksport-import)

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

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

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

```
getwd()
```

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

Andmete sisselugemine töökataloogist:

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

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

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

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

R-i saab sisse lugeda palju erinevaid andmeformaate. Kui sa soovid oluliselt ohverdada reprodutseeritavust installi RStudio addin: "Gotta read em all R". See läheb ülesse tab-i Addins. Sealt saab selle avada ja selle abil tabeleid oma workspace üles laadida. Selline point-and-click lahendus sobib ehk tabelite esialgseks tutvumiseks, kuid korrektne on andmed importida programmaatiliselt oma skriptis.

```
#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 selliseks nagu need instrumendi või andmesisestaja poolt väljastati.

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)
```

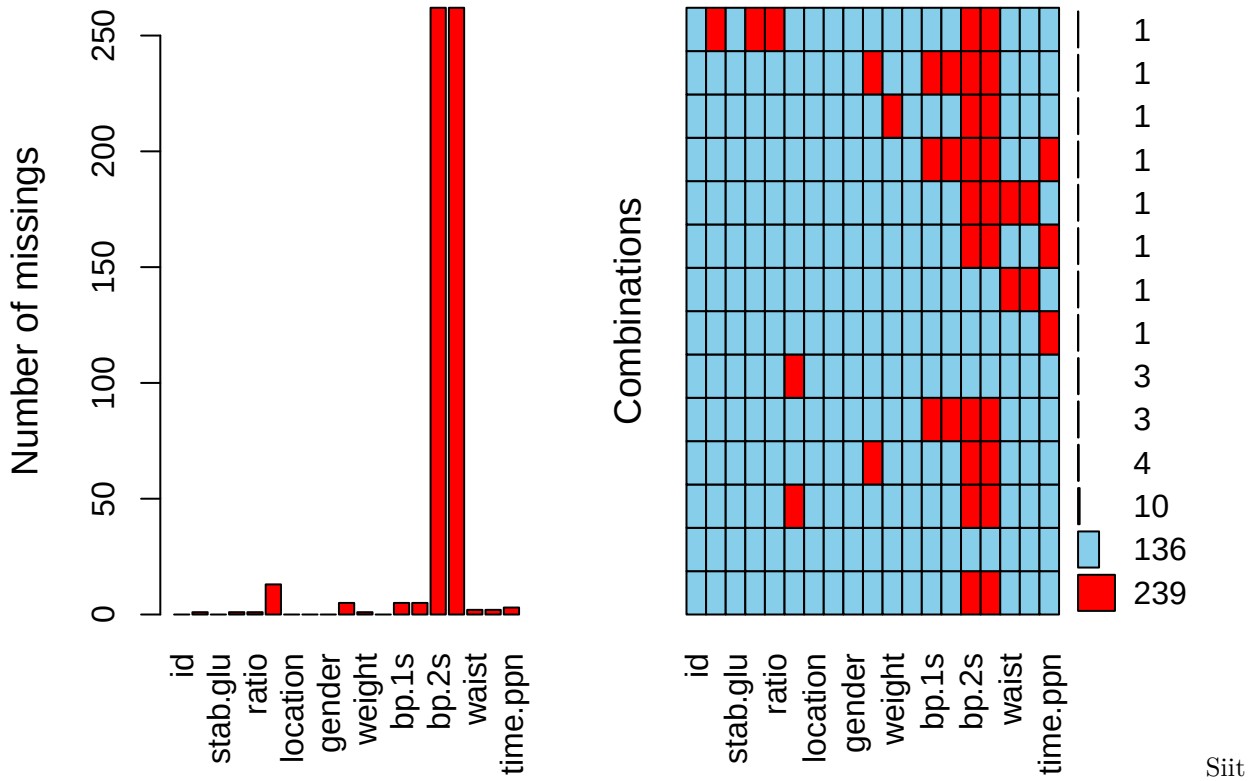
```
## [1] "tbl_df"     "tbl"        "data.frame"
```

4.7 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 = TRUE)
```



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

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

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

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

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

```
## [1] 0.3375
```

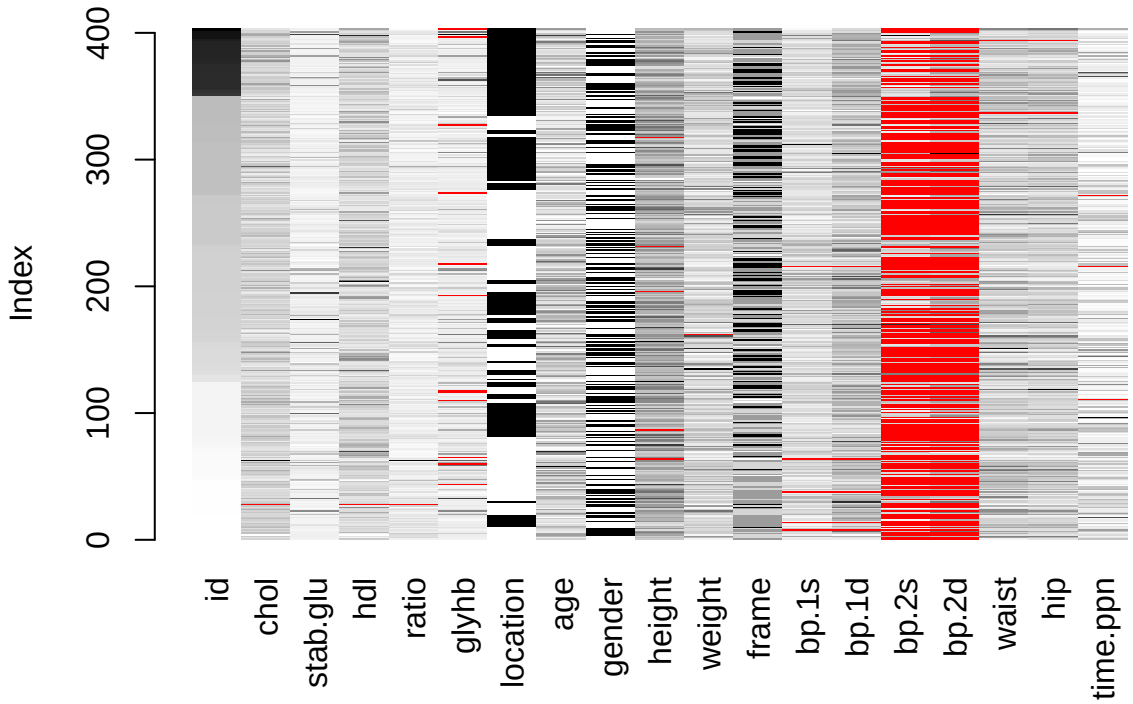
Mitu NA-d on igas tulbas?

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

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

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

```
matrixplot(diabetes)
```



Kuidas rekodeerida NA-d näiteks 0-ks:

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

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

4.8 Matrix

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

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

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

4.9 Indekseerimine

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

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

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

4.9.1 Vektorid ja nende indeksid on ühedimensionaalsed

```
my_vector <- 2:5
my_vector
```

```
## [1] 2 3 4 5
```

```
my_vector[1] #1. element ehk number 2
```

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

```
my_vector[c(1,3)] #1. ja 3. element
```

```
## [1] 2 4
```

```
my_vector[-1] #kõik elemendid, v.a. element number 1
```

```
## [1] 3 4 5
```

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

```
## [1] 3 5
```

```
my_vector[3:5] #elemendid 3, 4 ja 5 (element 5 on määramata, seega NA)
```

```
## [1]  4  5 NA
```

```
my_vector[-(3:length(my_vector))] #1. ja 2. element
```

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

4.9.2 Andmeraamid ja maatriksid on kahedimensionaalsed, nagu ka nende indeksid

2D indeksi kuju on [rea_indeks, veeru_indeks].

```
dat <- tibble(colA = c("a", "b", "c"), colB = c(1, 2, 3))
dat
# üks andmepunkt: 1 rida, 2. veerg
dat[1, 2]
# 1. rida, kõik veerud
dat[1, ]
# 2. veerg, kõik read
dat[, 2]
# kõik read peale 1.
dat[-1, ]
# viskab välja 2. veeru
dat[, -2]
# 2 andmepunkti: 2. rida, 1. ja 2. veerg
dat[2, 1:2]
# 2 andmepunkti: 2. rida, 3. ja 4. veerg
dat[2, c(1, 2)]
#viskab välja 1. ja 2. rea
dat[-c(1, 2), ]
#veerg nimega colB, output on erandina vektor!
dat$colB
```

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

Nüüd veidi keerulisemad konstruktsioonid, mis võimaldavad tabeli ühe kindla veeru väärtusi välja tõmmata teise veeru väärtuste järgi 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"
```

4.9.3 Listide indekseerimine

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

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

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

```
my_list <- list(a = tibble(colA = c("A", "B"), colB = c(1, 2)), b = c(1, NA, "s"))
## this list has two elements, a data frame called "a" and a character vector called "b".
str(my_list)
```

```
## List of 2
##  $ a:Classes 'tbl_df', 'tbl' and 'data.frame':   2 obs. of  2 variables:
##   ..$ colA: chr [1:2] "A" "B"
##   ..$ colB: num [1:2] 1 2
##  $ b: chr [1:3] "1" NA "s"
```

Tõmbame listist välja tibble:

```
my_tibble <- my_list[[1]]
my_tibble
```

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

See ei ole enam list.

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

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

```
## List of 1
##  $ a:Classes 'tbl_df', 'tbl' and 'data.frame':   2 obs. of  2 variables:
```

```
##   ..$ colA: chr [1:2] "A" "B"
##   ..$ colB: num [1:2] 1 2
```

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

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

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

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

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

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

Peatükk 5

Regular expression ja find & replace

Regular expression annab võimaluse lühidalt kirjeldada mitte-üheseid otsinguparameetreid.

> regular expression on string, mis kirjeldab mitut stringi

A regular 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.

5.1 Common operations with regular expressions

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

5.2 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 yo

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

Peatükk 6

Funktsioonid on R keele verbid

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

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

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

```
# calculate correlation
cor(x, y)
```

```
## [1] -0.1166
```

```
# 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] 1.304
```

```
# return quantiles
quantile(x)
```

```
##   0%  25%  50%  75% 100%
##    3    3    4    5    6
```

```
# return maximum value
max(x)
```

```
## [1] 6
```

```
# return minimum value
min(x)
```

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

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

```
?sum
```

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

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

Description:

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

Usage

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

Arguments

... - numeric or complex or logical vectors.

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

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

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

```
## [1] 78
```

Niimoodi saab arvutada summat vektorile nimega "apples".

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

```
sum
```

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

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

6.1 Kirjutame R funktsiooni

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

Kõige pealt kirjutame natuke koodi.

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

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

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

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

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

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

Niiüd midagi kasulikumat!

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

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

x hoiab lihtsalt kohta andmetele, mida me tahame sinna funktsiooni suunata. `sd()`, `sqrt()` ja `length()` on olemasolevad baas R funktsioonid, mille me oma funktsiooni hõlmame.

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

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

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

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

```
numbers_filtered <- na.omit(numbers)
calc_sem(numbers_filtered)
```

```
## [1] 12.8
```

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

NA-de vaikimisi eemaldamiseks on hetkel mitu võimalust, kasutame kõigepealt 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] 11.45
```

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

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

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

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

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

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

calc_sem(numbers)
```

```
## Warning in calc_sem(numbers): Removed NAs from vector.
```

```
## [1] 12.8
```

```
length(numbers)
```

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

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

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

Peatükk 7

Tidyverse

Tidyverse on osa R-i ökosüsteemist, kus kehtivad omad reeglid. Tidyverse raamatukogud lähtuvad ühtsest filosoofiast ja töötavad hästi koos. Tidyverse algab andmetabeli struktuurist ja selle funktsioonid võtavad reeglina sisse õige struktuuriga tibble ja väljastavad samuti tibble, mis sobib hästi järgmise tidyverse funktsiooni sisendiks. Seega on tidyverse hästi sobiv läbi torude %>% laskmiseks. Tidyverse-ga sobib hästi kokku ka ggplot2 graafikasüsteem.

7.1 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).

7.1.1 Tabeli dimensioonide muutmine (pikk ja lai formaat)

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

See on laias formaadis tabel df, mis ei ole tidy

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

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

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

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

7.1.2 Tibble transpose — read veergudeks ja vastupidi

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

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

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

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

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

7.2 dplyr ja selle viis 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.

7.2.1 `select()` columns

`select()` selects, renames, and re-orders columns.

Select columns from sex to value:

```
iris
select(iris, Petal.Length:Species)
select(iris, -(Petal.Length:Species)) #selects everything, except those cols
```

To select 3 columns and rename *subject* to *SUBJ* and put liik as the 1st col:

```
select(iris, liik = Species, Sepal.Length, Sepal.Width) %>% dplyr::as_data_frame()
```

```
## # A tibble: 150 x 3
##      liik Sepal.Length Sepal.Width
##    <fctr>        <dbl>       <dbl>
##  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
## # ... with 140 more rows
```

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

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

helper functions you can use within select():

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

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

`contains("ijk")`: matches names that contain “ijk.”

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

`num_range("x", 1:3)` matches x1, x2 and x3.

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

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

```
## #   Petal.Width <dbl>
```

```
dat <- as.data.frame(matrix(runif(100), nrow = 10))
dat <- tbl_df(dat[c(3, 4, 7, 1, 9, 8, 5, 2, 6, 10)])
select(dat, V9:V6)
```

```
## # A tibble: 10 x 5
##          V9      V8      V5     V2      V6
##       <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
##  1 0.369165 0.01380 0.12653 0.6138 0.04549
##  2 0.464169 0.48879 0.94613 0.2591 0.92570
##  3 0.798048 0.22343 0.64678 0.4235 0.17134
##  4 0.326344 0.08323 0.48601 0.7477 0.12729
##  5 0.604960 0.91620 0.78617 0.4165 0.94463
##  6 0.033697 0.31995 0.44796 0.3548 0.08401
##  7 0.809581 0.26890 0.07461 0.4264 0.93989
##  8 0.003745 0.33940 0.96476 0.8863 0.36196
##  9 0.699465 0.02408 0.38421 0.9057 0.82617
## 10 0.598144 0.17839 0.47127 0.7378 0.37653
```

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

```
## # A tibble: 10 x 4
##          V9      V8      V7      V6
##       <dbl>   <dbl>   <dbl>   <dbl>
##  1 0.369165 0.01380 0.99092 0.04549
##  2 0.464169 0.48879 0.88863 0.92570
##  3 0.798048 0.22343 0.11302 0.17134
##  4 0.326344 0.08323 0.83519 0.12729
##  5 0.604960 0.91620 0.54884 0.94463
##  6 0.033697 0.31995 0.86620 0.08401
##  7 0.809581 0.26890 0.53966 0.93989
##  8 0.003745 0.33940 0.07382 0.36196
##  9 0.699465 0.02408 0.18527 0.82617
## 10 0.598144 0.17839 0.05661 0.37653
```

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

7.2.2 `filter()` rows

Keep rows in Iris that have Species level "setosa" **and** Sepal.Length value <4.5.

```
filter(iris, Species=="setosa" & Sepal.Length < 4.5)
```

```
## # A tibble: 4 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1          4.4         2.9          1.4         0.2
## 2          4.3         3.0          1.1         0.1
## 3          4.4         3.0          1.3         0.2
## 4          4.4         3.2          1.3         0.2
## # ... with 1 more variables: Species <fctr>
```

Keep rows in Iris that have Species level "setosa" **or** Sepal.Length value <4.5.

```
filter(iris, Species=="setosa" | Sepal.Length < 4.5)
```

```
## # A tibble: 50 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
##           <dbl>       <dbl>        <dbl>       <dbl>
##  1          5.1         3.5          1.4         0.2
##  2          4.9         3.0          1.4         0.2
##  3          4.7         3.2          1.3         0.2
##  4          4.6         3.1          1.5         0.2
##  5          5.0         3.6          1.4         0.2
##  6          5.4         3.9          1.7         0.4
##  7          4.6         3.4          1.4         0.3
##  8          5.0         3.4          1.5         0.2
##  9          4.4         2.9          1.4         0.2
## 10          4.9         3.1          1.5         0.1
## # ... with 40 more rows, and 1 more variables:
## #   Species <fctr>
```

Keep rows in Iris that have Species level "not setosa" **or** Sepal.Length value <4.5.

```
filter(iris, Species !="setosa" | Sepal.Length < 4.5)
```

```
## # A tibble: 104 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
##           <dbl>       <dbl>        <dbl>       <dbl>
##  1          4.4         2.9          1.4         0.2
##  2          4.3         3.0          1.1         0.1
##  3          4.4         3.0          1.3         0.2
##  4          4.4         3.2          1.3         0.2
##  5          7.0         3.2          4.7         1.4
##  6          6.4         3.2          4.5         1.5
##  7          6.9         3.1          4.9         1.5
##  8          5.5         2.3          4.0         1.3
##  9          6.5         2.8          4.6         1.5
## 10          5.7         2.8          4.5         1.3
## # ... with 94 more rows, and 1 more variables:
## #   Species <fctr>
```

Kui tahame samast veerust filtreerida "või" ehk "|" abil mitu väärtust, on meil valida kahe samaväärse variandi vahel (tegelikult töötab 2. variant ka ühe väärtuse korral)

```
filter(iris, Species =="setosa" | Species =="versicolor")
filter(iris, Species %in% c("setosa", "versicolor") )
```

Nagu näha, 2. variant on oluliselt lühem.

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

```
library(stringr)
filter(iris, str_detect(Species, "^v"))
```

```
## # A tibble: 100 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
##           <dbl>       <dbl>        <dbl>       <dbl>
##  1          7.0         3.2          4.7         1.4
##  2          6.4         3.2          4.5         1.5
##  3          6.9         3.1          4.9         1.5
##  4          5.5         2.3          4.0         1.3
##  5          6.5         2.8          4.6         1.5
##  6          5.7         2.8          4.5         1.3
##  7          6.3         3.3          4.7         1.6
##  8          4.9         2.4          3.3         1.0
##  9          6.6         2.9          4.6         1.3
## 10          5.2         2.7          3.9         1.4
## # ... with 90 more rows, and 1 more variables:
## #   Species <fctr>
```

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))
```

7.2.3 `summarise()`

Many rows summarised to a single value

```
summarise(iris,
          MEAN = mean(Sepal.Length),
          SD = sd(Sepal.Length),
          N = n(),
          n_species = n_distinct(Species))
```

```
## # A tibble: 1 x 4
##    MEAN     SD     N n_species
##   <dbl>  <dbl> <int>     <int>
## 1 5.843 0.8281   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.

7.2.4 group_by()

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

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

```
iris_grouped <- group_by(iris, Species)
summarise(iris_grouped,
          MEAN = mean(Sepal.Length),
          SD = sd(Sepal.Length),
          N = n(),
          n_species = n_distinct(Species))
```

```
## # A tibble: 3 x 5
##      Species  MEAN     SD     N n_species
##       <fctr> <dbl>  <dbl> <int>     <int>
## 1     setosa 5.006 0.3525    50         1
## 2 versicolor 5.936 0.5162    50         1
## 3  virginica 6.588 0.6359    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 0.7071     2     1
## 2     F experiment_1  37.0 1.4142     2     1
## 3     F experiment_2  43.0 1.4142     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 0.7071     2     1
## 2      control     M  23.0     NA     1     1
## 3 experiment_1     F  37.0 1.4142     2     1
## 4 experiment_1     M  34.0     NA     1     1
## 5 experiment_2     F  43.0 1.4142     2     1
## 6 experiment_2     M  40.0     NA     1     1
```

pro tip if you want to summarise and then display the summary values as new column(s), which are added to the original non-shrunk df, use `mutate()` instead of `summarise()`.

```
mutate(iris_grouped,
       MEAN = mean(Sepal.Length),
       SD = sd(Sepal.Length))
```

```
## # A tibble: 150 x 7
## # Groups:   Species [3]
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
##           <dbl>       <dbl>        <dbl>       <dbl>
##  1          5.1         3.5          1.4         0.2
##  2          4.9         3.0          1.4         0.2
##  3          4.7         3.2          1.3         0.2
##  4          4.6         3.1          1.5         0.2
##  5          5.0         3.6          1.4         0.2
```

```
##  6          5.4         3.9          1.7         0.4
##  7          4.6         3.4          1.4         0.3
##  8          5.0         3.4          1.5         0.2
##  9          4.4         2.9          1.4         0.2
## 10          4.9         3.1          1.5         0.1
## # ... with 140 more rows, and 3 more variables:
## #   Species <fctr>, MEAN <dbl>, SD <dbl>
```

Anna igast grupist 3 kõrgeimat väärtust ja 2 madalaimat väärtust. Samad numbrid erinevates ridades antakse kõik - selle pärast on meil tabelis rohkem ridu.

```
top_n(iris_grouped, 3, Sepal.Length)
top_n(iris_grouped, -2, Sepal.Length)
```

7.2.5 mutate()

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

If in your tibble called 'df' you have a column called 'value', you can create a new log2 transformed value 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     3.135
## 2     Ann     F      control    31     3.434
## 3    Jill     F      control    30     3.401
## 4     Tim     M experiment_1    34     3.526
## 5     Ann     F experiment_1    38     3.638
## 6    Jill     F experiment_1    36     3.584
## 7     Tim     M experiment_2    40     3.689
## 8     Ann     F experiment_2    42     3.738
## 9    Jill     F experiment_2    44     3.784
```

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     3.135
## 2     Ann      F     3.434
## 3    Jill      F     3.401
## 4     Tim      M     3.526
```

```
## 5     Ann      F     3.638
## 6    Jill      F     3.584
## 7     Tim      M     3.689
## 8     Ann      F     3.738
## 9    Jill      F     3.784
```

```
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     0.7113  2.668
## 2     Ann     F      control    31     0.8416  5.466
## 3    Jill     F      control    30     0.8145  5.290
## 4     Tim     M experiment_1    34     1.0515  3.944
## 5     Ann     F experiment_1    38     1.0317  6.700
## 6    Jill     F experiment_1    36     0.9774  6.347
## 7     Tim     M experiment_2    40     1.2371  4.639
## 8     Ann     F experiment_2    42     1.1403  7.405
## 9    Jill     F experiment_2    44     1.1946  7.758
```

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     0.6509  3.477
## 2     Ann     F      control    31     0.8774  4.687
## 3    Jill     F      control    30     0.8491  4.536
## 4     Tim     M experiment_1    34     0.9623  5.140
## 5     Ann     F experiment_1    38     1.0755  5.745
## 6    Jill     F experiment_1    36     1.0189  5.443
## 7     Tim     M experiment_2    40     1.1321  6.047
## 8     Ann     F experiment_2    42     1.1887  6.350
```

```
## 9    Jill     F experiment_2    44     1.2453  6.652
```

7.3 Grouped filters

Keep all groups bigger than a threshold:

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

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

```
ungroup(dat)
```

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

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

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

```
library(stringr)
(bad.df <- tibble(time = c("t0", "t1", "t12"), value = c(2, 4, 9)))
```

```
## # A tibble: 3 x 2
##    time value
##   <chr> <dbl>
## 1    t0     2
## 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.

7.4 separate() one column into several

Siin on veel üks verb, mida aeg-ajalt kõigil vaja läheb. `separate()` võtab ühe veeru sisu (mis peab olema character string) ning jagab selle laiali mitme uue veeru vahel. Kui teda kasutada vormis `separate(df, old_Column, into=c("new_col1", "new_col2", "ja_nii_edasi"))` siis püüab programm ise ära arvata, kustkohalt veeru sisu hakkida (tühikud, komad, semikoolonid, koolonid jne). Aga te võite eksplitsiitselt ette anda separaatori sep = "". sep = 2 tähendab "peale 2. tähemärki". sep = -6 tähendab "enne tagantpoolt 6. tähemärki"

```
(dat <- tibble(country = c("Albania"), disease.cases = c("80/1000")))
```

```
## # A tibble: 1 x 2
##   country disease.cases
##     <chr>         <chr>
## 1 Albania       80/1000
```

```
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand")))
```

```
## # A tibble: 1 x 3
##   country cases thousand
## *   <chr> <chr>    <chr>
## 1 Albania    80     1000
```

```
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand"), sep = "/"))
```

```
## # A tibble: 1 x 3
##   country cases thousand
## *   <chr> <chr>    <chr>
## 1 Albania    80     1000
```

```
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand"), sep = 2))
```

```
## # A tibble: 1 x 3
##   country cases thousand
## *   <chr> <chr>    <chr>
## 1 Albania    80    /1000
```

```
(df.sep <- dat %>% separate(disease.cases, into=c("cases", "thousand"), sep = -6))
```

```
## # A tibble: 1 x 3
##   country cases thousand
## *   <chr> <chr>    <chr>
## 1 Albania    80    /1000
```

```
(dat <- tibble(index = c(1, 2),
               taxon = c("Procaryota; Bacteria; Alpha-Proteobacteria; Escharichia", "Eukaryota; Chordata
```

```
## # A tibble: 2 x 2
##   index
##   <dbl>
```

```
## 1     1
## 2     2
## # ... with 1 more variables: taxon <chr>
```

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

```
## # A tibble: 2 x 5
##   index       riik      hmk                klass
## * <dbl>      <chr>    <chr>                <chr>
## 1     1 Procaryota Bacteria Alpha-Proteobacteria
## 2     2  Eukaryota Chordata                 <NA>
## # ... with 1 more variables: perekond <chr>
```

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

```
## # A tibble: 2 x 2
##   index
##   <dbl>
## 1     1
## 2     2
## # ... with 1 more variables: taxon <chr>
```

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

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

```
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
## * <dbl>      <chr>    <chr>                <chr>
## 1     1 Prokaryota Bacteria Alpha-Proteobacteria
## 2     2  Eukaryota Chordata                 <NA>
## # ... with 1 more variables: perekond <chr>
```

```
(dat <- tibble(index = c(1,2),
               taxon = c("Prokaryota.Bacteria,Alpha-Proteobacteria.Escharichia", "Eukaryota.Chordata")))
```

```
## # A tibble: 2 x 2
##   index
##   <dbl>
## 1     1
## 2     2
## # ... with 1 more variables: taxon <chr>
```

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

```
## # A tibble: 2 x 5
##   index       riik      hmk                klass
```

```
## * <dbl>      <chr>    <chr>                <chr>
## 1     1 Prokaryota Bacteria Alpha-Proteobacteria
## 2     2  Eukaryota Chordata                 <NA>
## # ... with 1 more variables: perekond <chr>
```

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

7.5 Faktorid

Faktor on andmetüüp, mis oli ajalooliselt tähtsam kui ta praegu on. Sageli saame oma asja ära ajada character vectori andmetüübiga ja ei vaja faktorit. Aga siiski läheb faktoreid aeg-ajalt kõigil vaja.

> Faktorite abil töötame kategooriliste muutujatega, millel on fikseeritud hulk võimalikke väärtusi, mida me kõiki teame.

Faktori väärtusi kutsutakse "tasemeteks" (levels). Näiteks: muutuja sex on 2 tasemega faktor (M, F)

NB! Faktoriks muutes saame character vectori liikmete järjekorra muuta mitte-tähestikuliseks

Me kasutame faktoritega töötamisel forcats paketti. Kõigepealt loome character vectori x1 nelja kuu nime ingliskeelse lühendiga.

```
library(forcats)
x1 <- c("Dec", "Apr", "Jan", "Mar")
```

Nüüd kujutlege, et vektor x1 sisaldab 10 000 elementi. Seda vektorit on raske sorteerida, ja trükivead on ka raskesti leitavad. Mõlema probleemi vastu aitab, kui me konverteerime x1-e faktoriks. Selleks, et luua uus faktor, peaks kõigepealt üles lugema selle faktori kõik võimalikud tasemed:

Nüüd loome uue faktori ehk muudame x1 character vektori y1 factor vektoriks. Erinevalt x1-st seostub iga y1 väärtusega faktori tase. Kui algses vektoris on mõni element, millele ei vasta näiteks trükivea tõttu ühtegi faktori taset, siis see element muudetakse NA-ks. Proovige see ise järele, viies trükivea sisse x1-e.

```
y1 <- factor(x1, levels = month.abb)
y1
```

```
## [1] Dec Apr Jan Mar
## 12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep ... Dec
```

NB! `month.abb` on R objekt mis sisaldab kuude ingliskeelseid lühendeid.

Kui sa faktorile tasemeid ette ei anna, siis need tekivad andmetest automaatselt ja tähestikulises järjekorras.

Kui sa tahad, et faktori tasemed oleks samas järjekorras kui selle taseme esmakordne ilmumine teie andmetes siis:

```
f2 <- factor(x1) %>% fct_inorder()
f2
```

```
## [1] Dec Apr Jan Mar
## Levels: Dec Apr Jan Mar
```

`levels()` annab faktori tasemed ja nende järjekorra

```
levels(f2)
```

```
## [1] "Dec" "Apr" "Jan" "Mar"
```

Kui faktorid on tibbles oma veeruna, siis saab nende tasemed `count()` kasutades:

```
gss_cat #tibble, mille veerg "race" on faktor.
```

```
## # A tibble: 21,483 x 9
##     year       marital   age   race        rincome
##    <int>        <fctr> <int> <fctr>         <fctr>
##  1  2000 Never married    26  White  $8000 to 9999
##  2  2000      Divorced    48  White  $8000 to 9999
##  3  2000       Widowed    67  White Not applicable
##  4  2000 Never married    39  White Not applicable
##  5  2000      Divorced    25  White Not applicable
##  6  2000       Married    25  White $20000 - 24999
##  7  2000 Never married    36  White $25000 or more
##  8  2000      Divorced    44  White  $7000 to 7999
##  9  2000       Married    44  White $25000 or more
## 10  2000       Married    47  White $25000 or more
## # ... with 21,473 more rows, and 4 more variables:
## #   partyid <fctr>, 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.

7.5.1 `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.

7.5.2 `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)
```

7.5.3 `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
```

7.5.4 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)
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")
```

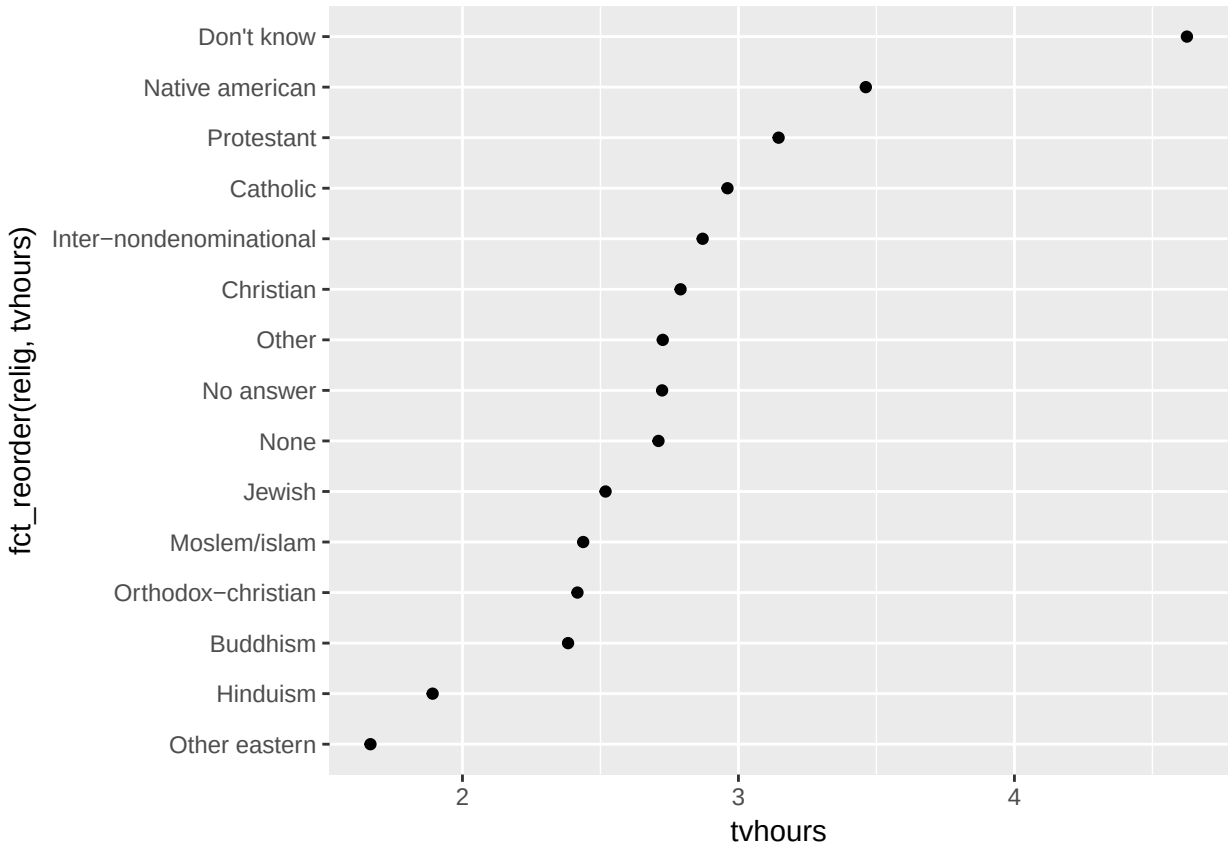
```
## [1] NA NA  3 NA NA  3 NA NA  3
```

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

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

7.5.5 Muudame faktori tasemete järjekorda joonisel

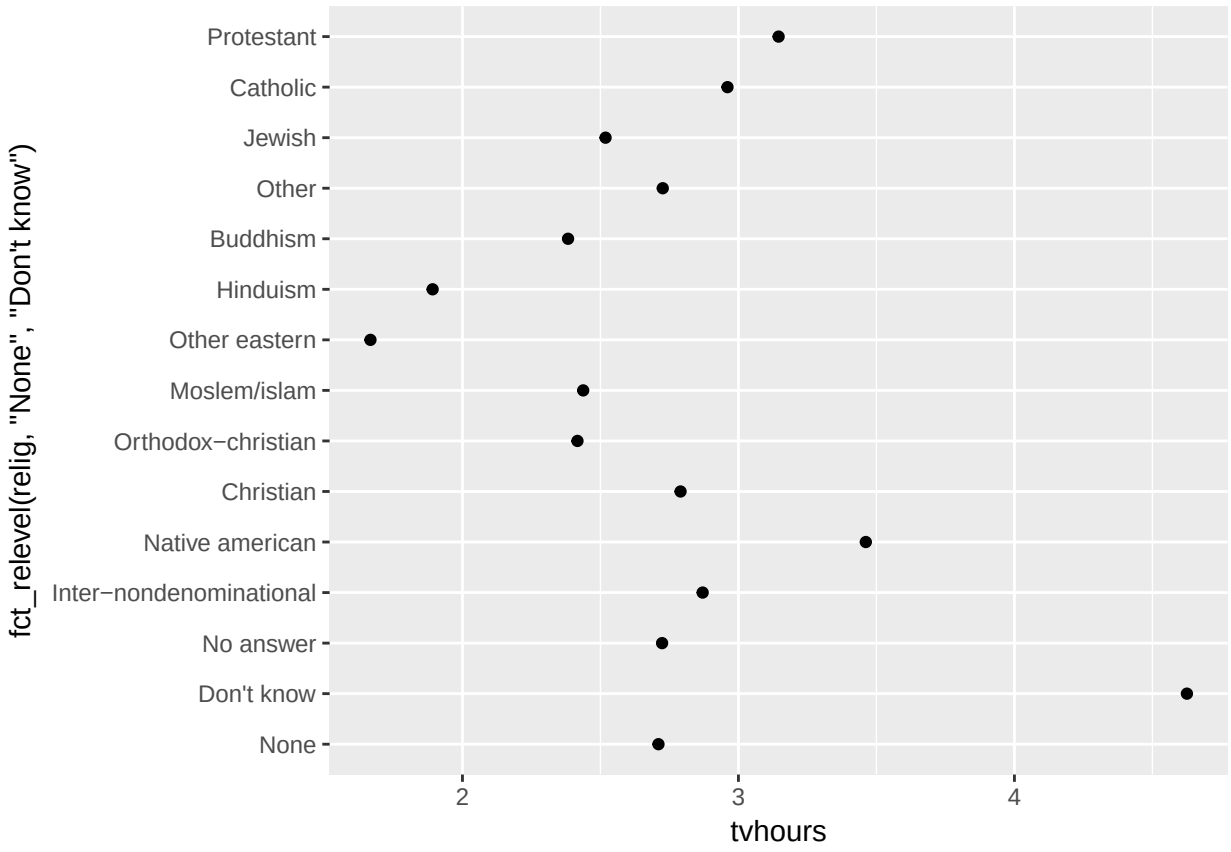
```
## summeerime andmed
gsscat_sum  <- group_by(gss_cat, relig) %>%
  summarise(age = mean(age, na.rm = TRUE),
            tvhours = mean(tvhours, na.rm = TRUE),
            n = n())
## joonistame graafiku
p <- ggplot(gsscat_sum, aes(tvhours, fct_reorder(relig, tvhours))) +
  geom_point()
p
```



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

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

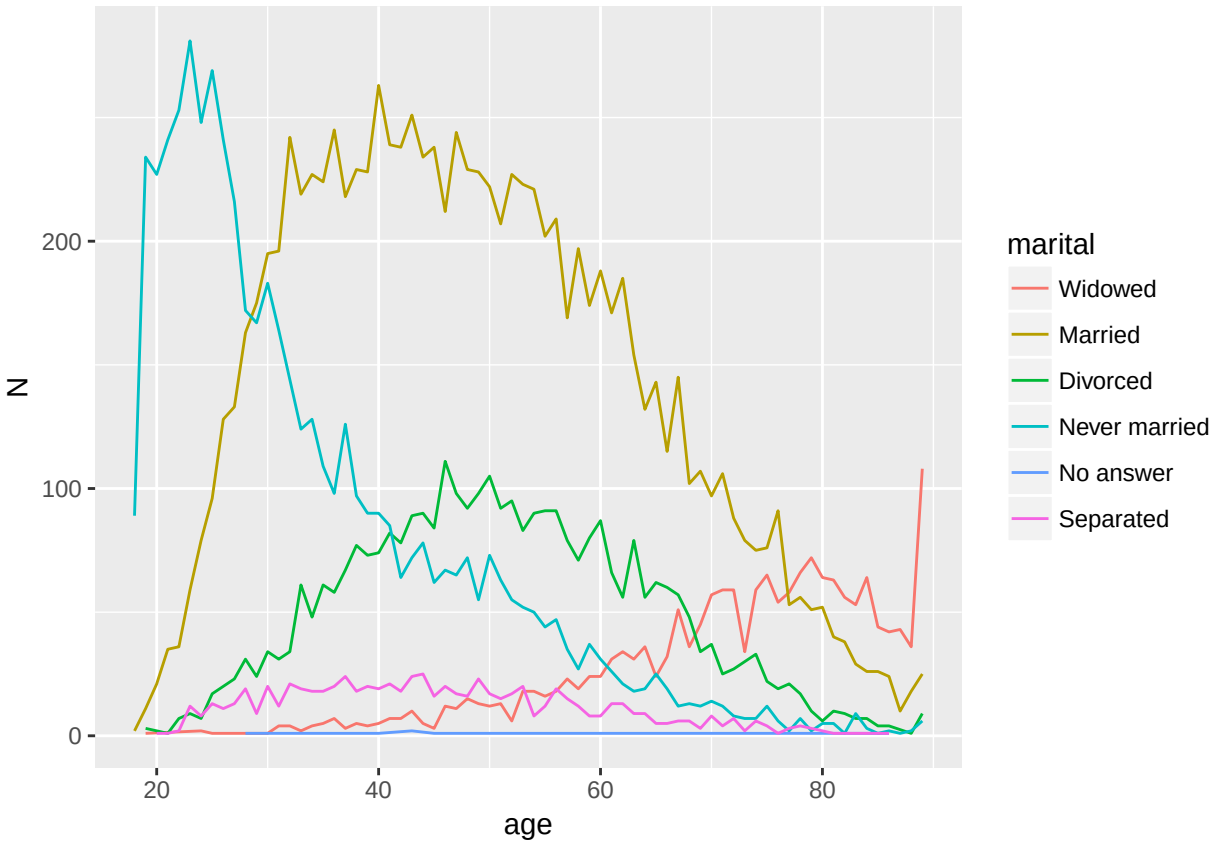
```
## täiendame eelmist graafikut ümberkorraldatud andmetega
p + aes(tvhours, fct_relevel(relig, "None", "Don't know"))
```



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

See muudab ploti paremini jälgitavaks:

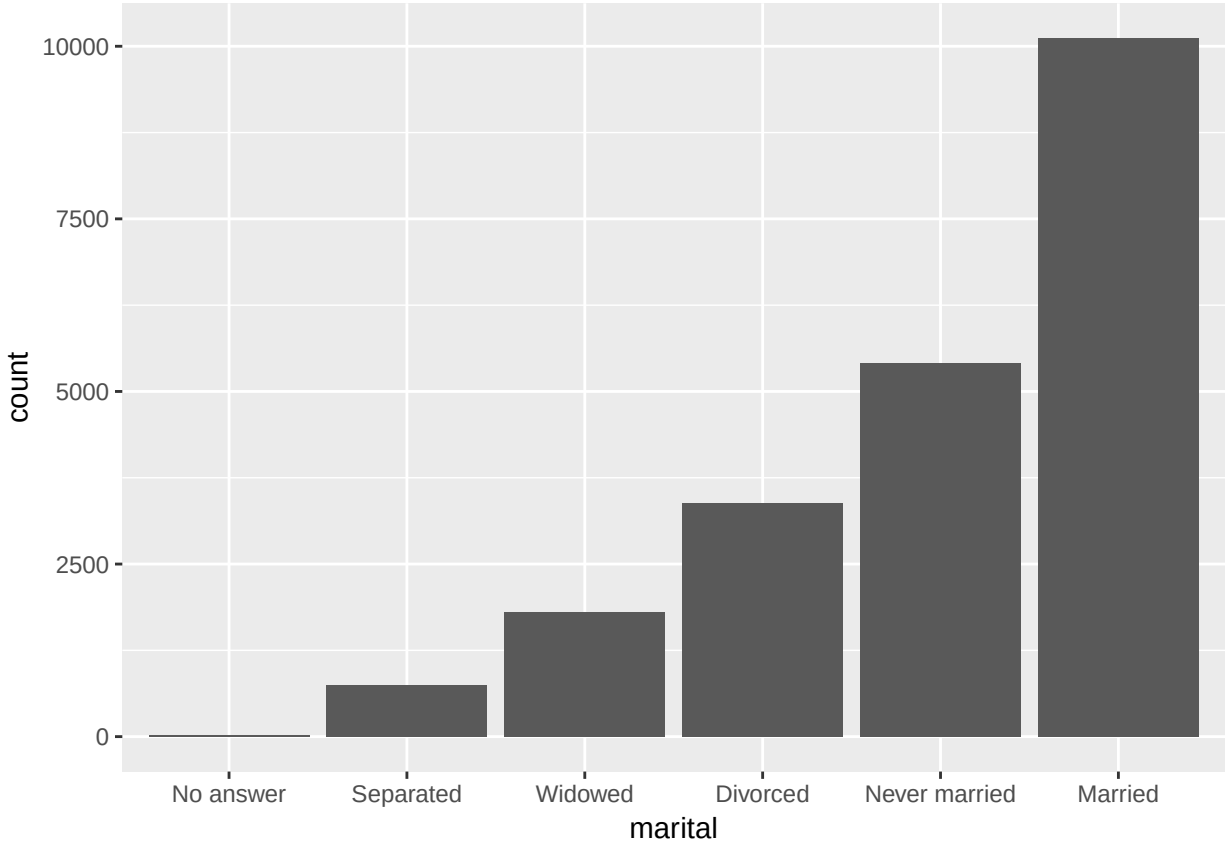
```
## summeerime andmed
gsscat_sum <- filter(gss_cat, !is.na(age)) %>%
  group_by(age, marital) %>%
  mutate(N=n())
## paneme andmed graafikule
ggplot(gsscat_sum, aes(age, N, colour = fct_reorder2(marital, age, N))) +
  geom_line() +
  labs(colour = "marital")
```



7.5.8 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

Bibliography

Bååth, R. (2013). Bayesian first aid. *tba*.

Bååth, R. (2016). *bayesboot: An Implementation of Rubin's (1981) Bayesian Bootstrap*. R package version 0.2.1.

Bürkner, P.-C. (2017). brms: An R package for bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28.

Gabry, J. and Mahr, T. (2017). *bayesplot: Plotting for Bayesian Models*. R package version 1.4.0.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2014). *Bayesian data analysis*, volume 2. CRC press Boca Raton, FL.

Kruschke, J. (2015). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press / Elsevier, 2nd edition.

Marwick, B., Boettiger, C., and Mullen, L. (2017). Packaging data analytical work reproducibly using r (and friends). *PeerJ Preprints*, 5:e3192v1.

McElreath, R. (2015). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. CRC Press.

McElreath, R. (2016). *rethinking: Statistical Rethinking book package*. R package version 1.59.

Stan Development Team (2016). *rstanarm: Bayesian applied regression modeling via Stan.* R package version 2.13.1.

Wickham, H., Danenberg, P., and Eugster, M. (2017). *roxygen2: In-Line Documentation for R*. R package version 6.0.1.