Seminar 1

Velkommen til første seminar!

Gjennom 6 seminarganger skal vi nå gå igjennom alt fra hva R er og hvordan det fungerer, til å kjøre våre helt egne regresjonsanalyser og holde på med *egen statistikk.* Før hvert seminar kommer jeg til å legge ut 3 dokumenter: et dokument som dette her med grundigere forklaringer, et R-script, og et oppgavesett. Så kommer jeg til å legge ut fasit til oppgavene dagen etter seminaret.

Dersom dere har spørsmål om noe vi går igjennom kan dere sende meg en e-post (elibal@student.sv.uio.no) eller melding på Canvas!

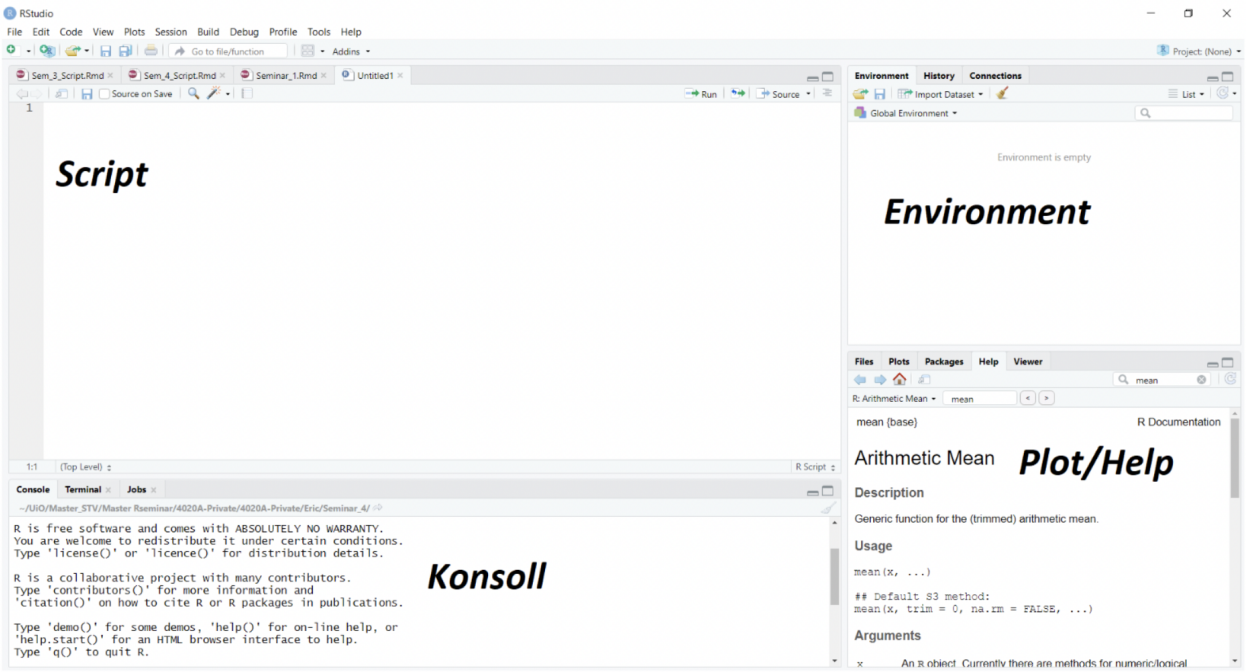
```
#Kode vil dere se har en egen grå bakgrunn. Alt som er skrevet her kan dere
# kopiere inn på deres egen pc, og kjøre for å se hva som skjer. Når jeg har #
# foran betyr det at jeg skriver en kommentar. Dette kan vi gjøre i
# scriptene våre for å forklare hva som skjer, uten at R prøver å kjøre det som
# vanlig kode og derfor gir en feilmelding. Hvis jeg kjører kode her, vil dere
# se resultatene som vanlig tekst under. Vi kan jo gjøre et lite forsøk med noe
# enkel matte! For å kjøre koden setter dere musen ved siden av, og
# trykker ctrl+enter.

100/2+4
```

```
## [1] 54
```

R, RStudio, og Syntax-feil

Før seminaret har dere lastet ned R, og RStudio. R er selve programeringsspråket vi skriver i, og som gjør at vi kan skrive kode. Når vi laster ned R laster vi egentlig ned et program som gjør at datamaskingen vår kan forstå det vi skriver, og gjør det vi ønsker at den gjør. Selve R-scriptet, eller koden om en vil, kunne vi egentlig skrevet i word, eller notisblokk. RStudio, programmet vi kommer til å bruke, er det som kalles et "Integrated development environment" (IDE), og brukes for å gjøre det lettere å skrive scriptet. Her har vi f.eks. enkel tilgang til hjelpefiler, den markerer hva forskjellig kode er ved hjelp av farger, og presenterer resultatene på en (vanligvis) lettleselig måte.



Dere vil fort legge merke til at RStudio har flere vinduer. Øverst til venstre (gitt standard konfigurasjonen, dere kan lett endre dette selv om dere ønsker) finner dere selve scriptet. Det er her vi vil skrive kode vi ønsker å lagre, og bruke videre. Under denne er det vi kaller en "Console" eller "intepreter", når dere kjører kode vil dere se at selve kodelinjen blir "sendt" ned dit, og det er der resultatet vises. Vi kan også skrive kode direkte inn i konsollen, men da blir det ikke lagret for senere bruk. Øverst til høyre har vi "environment", her vises alle objekter som vi har laget i scriptet, hva dette er for noe kommer vi tilbake til senere. Til slutt nederst til venstre vises en del informasjon, hvor det er i hovedsak to faner vi kommer til å bruke. Den ene er "Plots" som viser grafikk vi har laget, f.eks. et stolpediagram, og den andre er hjelpefilene hvor en kan slå opp hva forskjellige funksjoner gjør. Denne kan vi faktisk prøve ut med en gang!

```
# Ofte når vi bruker R er vi usikre på hvordan forskjellige funksjoner fungerer.
# Da kan det være nyttig å lese hjelpefilene som forteller hva en funksjon gjør,
# og hvordan en skal bruke den. For å gjøre dette skriver du et spørsmålstegn
# før navnet på funksjonen. La oss prøve dette med "mean()" funksjonen, som
# finner gjennomsnitt:

?mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An `R` object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

I hjelpefilen kan vi se at vi får en del informasjon om funksjonen, men la oss først tenkte litt på hva en funksjon er. I R jobber vi som oftest med forskjellige typer objekter. Vi skal straks komme tilbake til hva objekter er, men la oss i første omgang tenke på en tallrekke. Si vi har tallene fra 1-10. Hovedpoenget med R er at vi kan hente ut ulike typer informasjon fra dataene vi jobber med. Dette kan f.eks. være gjennomsnitt eller standardavvik.

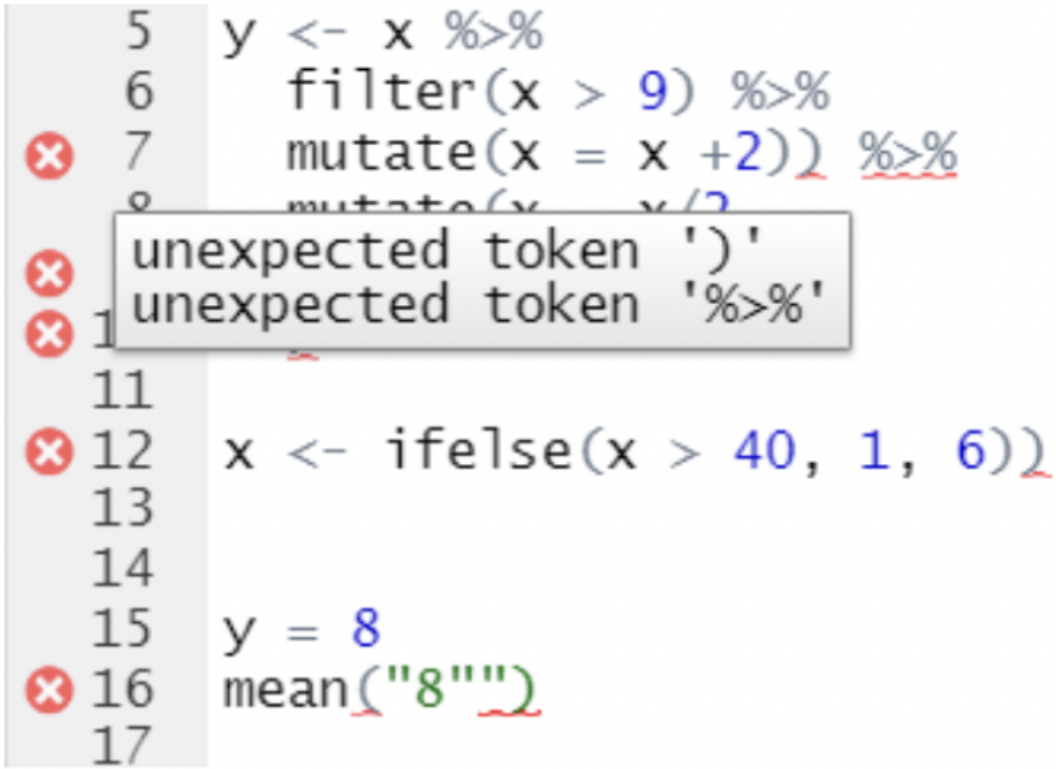
En funksjon er det vi bruker i R for å gjøre noe med dataene, f.eks. å finne gjennomsnittet. Alle funksjoner har til felles at de tar data, f.eks. tall, og gir oss et resultat. I tillegg har noen funksjoner argumenter, som gjør at vi kan endre noe på hvordan funksjonen lager resultatet. Leser vi hjelpefilen til "mean" ser vi at den øverst gir en beskrivelse, hvor det står at den returnerer gjennomsnittet. Under det kommer argumentene den godtar, et objekt (x) som inneholder tall, logiske verdier (kommer tilbake til hva det er), ett argument for å trimme dataene, altså fjerne noe, og na.rm argumentet. Det siste skal vi også komme tilbake til. Under "Value" får vi en beskrivelse av hva som er returnert, og så kommer det et eksempel på hvordan den brukes helt nederst.

Hjelpefilene er en flott måte å finne ut hva en funksjon gjør, og hvordan vi kan bruke den. Samtidig kan den ofte være litt kronglete å lese, men da hjelper det ofte å se på eksemplene som alltid er helt nederst.

Skulle det fortsatt være vanskelig er det viktig å huske at det finnes et stort miljø rundt R, og ofte er det mange som har opplevd samme problem som deg! Litt kjapp googling, og et søk på https://stackoverflow.com/ vil fort gi gode svar!

Syntax-feil

Når vi går igjennom kode vil vi fort få en del feil. Det er helt vanlig, og noe som er helt uungåelig! Det kan være lurt å merke seg "syntax-feil", som er skrivefeil vi gjør når vi skriver kode. F.eks. kan det være å skrive men() istedenfor mean(), glemme å lukke en parantes sånn at vi skriver mean(. Noe av det fine med RStudio er at den markere sånne feil for oss!



Her er et eksempel på kode som gir syntaksfeil. Som dere kan se på siden er det flere røde kryss ved siden av linjenumrene. Dette er steder hvor RStudio mener jeg har gjort feil, og holder man musen over dem får man

opp hva som er feilen. "Unexpected token ')' " betyr at RStudio mener det er en parantes på linjen som ikke skulle vært der. I tillegg er det røde streker under de delene av koden som RStudio mener er feil. Svært ofte er feilene vi gjør i R er enkle skrivefeil/syntax-feil som dette. Derfor er det veldig nyttig at RStudio viser oss disse feilene på denne måten!

Objekter, funksjoner, og klasser

Vi har allerede sett på litt enkel kode, men framover skal vi gått litt dypere inn i hvordan kode faktisk fungerer. Koding er en måte fortelle pc-en hva vi vil den skal gjøre gjennom tekst. Når vi skriver kode må vi vite hva vi skal skrive for at pc-en skal forstå det. Man lærer seg mange av de vanligste kodene og funksjonene etterhvert, men ellers kan man google seg frem til koder og funksjoner man ikke kan eller de man ikke bruker så ofte.

Det første vi skal se på er objekter. Objekter er navngitte ting i R som kan inneholde noe annet. Vi skal i hovedsak forholde oss til to typer objekter: vektorer, og funksjoner. Hva disse kan man vise med et eksempel:

```
# I første omgang kan vi prøve å lage en vektor. Dette er et objekt som
# inneholder flere elementer, f.eks. tall av samme klasse.

# La oss først prøve å lage en med et tall. For å gjøre dette må vi først velge
# et navn, så bruke det som heter en "assigner", og så skrive hva den skal
# inneholde. Her lager jeg en vektor som heter "to" og som inneholder tallet 2.

to <- 2

# <- er assigneren. Den sier bare at det som kommer på venstresiden skal lagres
# med navnet som er på høyresiden. Kjører dere koden vil dere se i environment
# at det kommer en linje hvor det står "to 2", dette betyr at vi har laget en
# variabel med navnet "to" som inneholder verdien 2.

# Vi kan f.eks bruke objektet vi har laget til litt enkel matte:

2 + to
```

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

```
# Her får vi resultatet fire, ettersom R nå leser to som 2.
```

```
# Det fine med objekter er at de kan inneholde veldig mye informasjon!
# I første omgang kan vi prøve å lagre flere tall. Det er flere måter vi kan
# gjøre dette på, f.eks. kan vi skrive 1:10 for å få alle heltallene mellom 1 og
# 10, eller skrive c(1,22,5,2,1) for å lage en rekke tall. I det siste skiller
# jeg tallene med komma.

tall <- 1:10
flere_tall <- c(1,4,56,8,4,2,4)
# Du kan gi objektene akkurat de navnene du vil, men man kan ikke ha mellomrom
# i navnene eller tall som første tegn, og det er god kutyme å unnåæ/ø/å
# generelt i script.

# La oss se om vi kan finne gjennomsnittet av disse vektorene.

mean(tall)
```

```
## [1] 5.5
```

```r
mean(flere_tall)
```

```
## [1] 11.28571
```

Med mean-funksjonen her ser vi at vi får gjennomsnittet for hele vektoren. Som oftest er det det vi ønsker, men hva hvis vi kun ønsket gjennomsnittet av noen spesifikke tall? Om dere ser tilbake til environment vil dere merke at etter navnet på vektoren står det først "num" og så [1:7]. Den første teksen sier at dette er et numerisk objekt. Klasser skal vi straks gå inn på. Det neste viser lengden på vektoren vår. *flere_tall* har sitt første tall i plassen 1, og siste i 7. Altså er det 7 elementer. Om vi ser på *tall* ser vi at det står 1:10, og denne har 10 elementer. For å få tak i et spesifikt element kan vi bruke klammeparanteser.

```r
#La oss si at vi vil ha element nr. 5 i vektoren flere_tall

tall[5]
```

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

```r
# Når vi kjører denne ser vi at vi får ut tallet 5,og dette kan vi jo også
# sjekke i environment for å se at stemmer at dette er det femte tallet i
# vektoren.

# På samme måte som vi definerte en rekke med tall tidligere, kan vi også bruke
# dette for å få ut en rekke med elementer.

flere_tall[3:6]
```

```
## [1] 56  8  4  2
```

```r
flere_tall[c(3,5,1,6)]
```

```
## [1] 56  4  1  2
```

```r
# Vi kan også finne gjennomsnittet av kun disse tallene:

mean(flere_tall[c(3,5,1,6)])
```

```
## [1] 15.75
```

```r
# Eller vi kan bruke disse som en ny vektor:

ny_vektor <- flere_tall[c(3,5,1,6)]
```

Klasser

Så langt har vi kun jobbet med tallverdier. Ofte har vi variabler som ikke er tall, men f.eks. tekst eller ordinalverdier. I R vil vi også se at visse funksjoner krever at dataene er i visse klasser. Hovedklassene vi kommer til å bruke er; numeric, character, logical, og factor. Numeric er tall (logisk nok). De fleste mattefunksjoner krever at dataene er numeric.

```r
# For å sjekke om noe er numeric kan vi bruke funksjonen is.numeric():

is.numeric(tall)
```

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

```r
# Her ser vi at vi får opp "TRUE" som betyr at "tall" er et numerisk objekt.
```

Dere vil noen ganger se at det skilles mellom "numeric" og "integer". Forskjellen er at integer kun kan inneholde heltall, mens numeric kan ha desimaler. Dette er noe som henger igjen fra gammelt av, og er svært sjeldent interessant for vår del.

Når vi vil skrive tekst bruker vi klassen "character". En tekststring må alltid ha "" rundt seg, men ellers definerer vi den som vanlig.

```
# Når vi vil skrive tekst, bruker vi klassen character og tekst må alltid ha
# " " rundt seg.

tekst <- "Jeg elsker R! <3"
is.character(tekst)
```

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

```
# Klassen character kan inneholde tekst, men vil f.eks. ikke kunne brukes til
# matte.

mean(tekst)
```

```
## Warning in mean.default(tekst): argument is not numeric or logical: returning NA
```

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

```
# Her ser dere at vi får en feilmelding, som sier at argumentet ikke er
# numerisk eller logisk. Funksjonen gir oss derfor resultatet NA, som
# betyr missing, altså at det ikke eksisterer et resultat.

# Vi kan også kreve at et objekt skal ha en viss klasse. Det gjør vi med
# as."klassenavn". Det kan føre til noen uforventede resultater. Si hvis gjør
# flere_tall om til character.

flere_tall <- as.character(flere_tall)
mean(flere_tall)
```

```
## Warning in mean.default(flere_tall): argument is not numeric or logical:
## returning NA
```

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

Grunnen til at vi får en feilmelding her er fordi vi ikke kan ta gjenomsnittet av tekst. Om dere ser i environment står det også nå at flere_tall er chr (character) og det "" rundt alle tegnene.

Den siste klassen vi kommer til å bruke ofte (men det finnes flere) er "factor." Faktor er en variabel som kan ha flere forhåndsdefinerte nivåer, og brukes ofte når vi skal kjøre statistiske modeller. En lett måte å forstå faktorer på er å tenke på dem som ordinale variabler, hvor vi vet rekkefølgen på nivåene men ikke avstanden mellom dem, f.eks. Barneskole, Ungdomskole, VGS.

```
skolenivaer <- factor(c("Barneskole", "Ungdomskole", "Videregaende",
                        "Videregaende", "Ungdomskole"),
                      levels = c("Barneskole", "Ungdomskole", "Videregaende"))

# Her kan vi se at vi først definerer de forskjellige verdiene som er i variabelen
# Så skriver vi hvilke nivåer den kan ha, i den rekkefølgen vi ønsker dem.

# Om vi ikke hadde definert nivåene ville R gjort det automatisk i alfabetisk
# rekkefølge.

# Nå kan vi se hva som er i variabelen, ved å bare kjøre navnet til variabelen:

skolenivaer
```

```
## [1] Barneskole   Ungdomskole  Videregaende Videregaende Ungdomskole
```

```
## Levels: Barneskole Ungdomskole Videregaende
```

```
#Vi kan også se hvilke nivåer som er i variabelen:

levels(skolenivaer) #Og får ut de tre nivåene
```

```
## [1] "Barneskole"   "Ungdomskole"  "Videregaende"
```

Tidligere sa jeg at en vektor var et objekt som inneholdt elementer av *samme* klasse. Så langt har vi også holdt oss til det gjennom å kun lage objekter med tekst eller tall. Hva skjer om vi prøver å blande tekst og tall?

```
# Vi kan også lage et objekt som inneholder både tekst og tall:

tekst_tall <- c(1,4,0,4, "matte", "R", "seminarer")

# Så kan vi bruke class-funksjonen for å se hvilken klasse dette nye
# objektet har:

class(tekst_tall)
```

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

Som vi kan se er her klassen blitt character, også for tallene! Det er fordi at hvis vi definerer en vektor som har flere klasser, blir det slått sammen til den klassen som har minst informasjon. Dette kalles "implicit coercion", og rekkefølgen går: logical -> integer -> numeric -> complex -> character.

Dataframes

Noen ganger har vi lyst til å slå sammen data som er av forskjellige typer. F.eks. kan det være at vi har data om alder, navn, fylke etc. og vil ha dette som et objekt. For å gjøre dette bruker vi data.frames. En dataframe består av flere kolonner, hvor hver kolonne er en vektor. Disse kan ha forskjellige typer, med f.eks. en character-vektor, og ett tall. Videre vil hver rad være en enhet. Dette kan f.eks. være en person. Dataframes er ofte noe en laster ned når en skal ha data, f.eks. fra en survey, men vi kan også lage dem selv. En viktig regel for dataframes er at alle vektorene må ha lik lengde. Dersom vi har tomme celler, må vi finne en måte å "fylle" disse tomme cellene på. Det gjør vi med NA.

```
navn <- c("Emma", "Leo", "Thea", "Kim", "Mari", "Bhavna", "Emil")
alder <- c(60, 45, 19, 19, NA, 87, 92)
fylke <- c("Vestland", "Troms og Finnmark", "Agder", NA, "Trøndelag", "Viken",
           "Viken" )
by <- c("Bergen", "Karasjok", "Arendal", NA, "Trondheim", "Halden", "Fredrikstad")

# Her lager jeg først et sett med vektorer, med litt forskjellig informasjon.
# I environment kan dere se at alle har en lengde på 7.
# Dette kan vi også sjekke med length-funksjonen:

length(navn)
```

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

```
#For å lage en data.frame kan vi bruke funksjones as.data.frame()

personer <- data.frame(navn, alder, fylke, by)
```

I environment vil dere nå se at det dukker opp en ny type verdi, under "Data" med navnet "personer". Når det står 7 obs (observasjoner) av 4 variabler betyr dette at vi har en dataframe med 7 rader og 4 kolonner.

Klikker dere på den vil dere se dette:

	navn	alder	fylke	by
1	Emma	60	Vestland	Bergen
2	Leo	45	Troms og Finnmark	Karasjok
3	Thea	19	Agder	Arendal
4	Kim	19	*NA*	*NA*
5	Mari	*NA*	Trøndelag	Trondheim
6	Bhavna	87	Viken	Halden
7	Emil	92	Viken	Fredrikstad

Første observasjonen her er rad 1, som er Emma på 60 år fra Bergen i Vestland. Det viktigste med dataframes er at vi kan sette sammen flere typer informasjon om samme enhet på en gang. Det er flere måter vi kan bruke dette på. La oss først se på hvordan vi kan gjøre enkle analyser av en kolonne.

```
# Før har vi kun skrevet navnet på vektoren. Nå som vi har det i en dataframe, må vi først
# velge denne, og så kolonnen. Det er to måter vi kan gjøre dette på:

# Med klammeparanteser kan vi velge rad og kolonne. Raden kommer først, og så
# kolonnen:
personer[2,1]
```

```
## [1] "Leo"
```

```
# Lar vi en plass være tom får vi enten alle kolonnene eller alle radene til en
# bestemt rad eller kolonne.

personer[,2]
```

```
## [1] 60 45 19 19 NA 87 92
```

```
personer[2,]
```

```
##   navn alder             fylke       by
## 2  Leo    45 Troms og Finnmark Karasjok
```

```
# En mer vanlig måte å hente ut kolonner på er med '$'. Da skriver man først
# navnet på datarammen, og så variabelen:

personer$alder # Her får man verdiene til alle aldrene i datarammen.
```

```
## [1] 60 45 19 19 NA 87 92
```

```
# Vi kan bruke matematiske formler her på samme måte som tidligere, f.eks.
# for å finne alders gjennomsnittet i datarammen:

mean(personer$alder) # Her får vi NA?
```

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

Her kan dere se at vi fikk NA til svar istedet for det gjennomsnittet vi ønsket. NA betyr som sagt bare missing, altså at vi mangler informasjon i denne kolonnen for en eller flere av radene/enhetene. I datasettet

har vi ikke informasjon om hvor gammel Mari er. Når minst en av verdiene er NA vil mange funksjoner returnere NA når vi kjører koden. Dette fordi vi jo strengt tatt ikke kan vite gjennomsnittet når vi ikke vet alle verdiene. For å få ut et resultat må vi derfor fortelle R at vi ønsker å fjerne NA-verdiene, og da vil vi få gjennomsnittet til de enhetene vi faktisk har alderen til.

```
mean(personer$alder, na.rm = TRUE) # Her får man svaret 52.5 istedet.
```

```
## [1] 53.66667
```

```
# na.rm betyr NA remove, og når vi setter den til TRUE ber R om å fjerne NA i
# variabelen når den regner ut gjennomsnittet.


# For å finne medianen:

median(personer$alder, na.rm = TRUE)
```

```
## [1] 52.5
```

```
# En lettere måte å få ut alle disse deskriptive verdiene er  å bruke
# summary()-funksjonen.
# Da trenger vi heller ikke bruke na.rm, fordi den heller sier hvor mange NA
# det er i vektoren

summary(personer$alder)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   19.00   25.50   52.50   53.67   80.25   92.00       1
```

Visualisering

Det siste vi skal på i dag er en kort intro til hvordan vi kan visualisere data. For å gjøre dette må vi først laste ned en pakke som heter Tidyverse. Pakker er tillegg til R som gjør at du kan laste ned flere funksjoner, og disse kan gjøre ting enklere. R som det kommer når det lastes ned kalles "base R." Om noe er vanskelig i base R, finnes det mest sannsynligvis en pakke som gjør det lettere! Tidyverse, som vi vil bruke mye, er et sett med pakker som gjør databehandling mye, mye enklere. For å bruke denne, må vi først installere pakken. Om dere har gjort dette på forhånd trenger dere ikke å gjøre dette på nytt. Å installere gjør vi kun en gang, og så evt. på nytt om det kommer en oppdatering.

```
# For å installere en pakke bruker vi funksjonen install.packages, og skriver
# navnet på pakken i parantesene med hermetegn:

install.packages("tidyverse")
```

Hver gang vi skal bruke pakken må vi fortelle R at vi skal bruke den. Det må vi gjøre hver gang vi åpner R på nytt.

```
#For å hente inn pakken bruker vi funksjonen library():

library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------------- tidyverse 1
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------- tidyverse_conflict
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

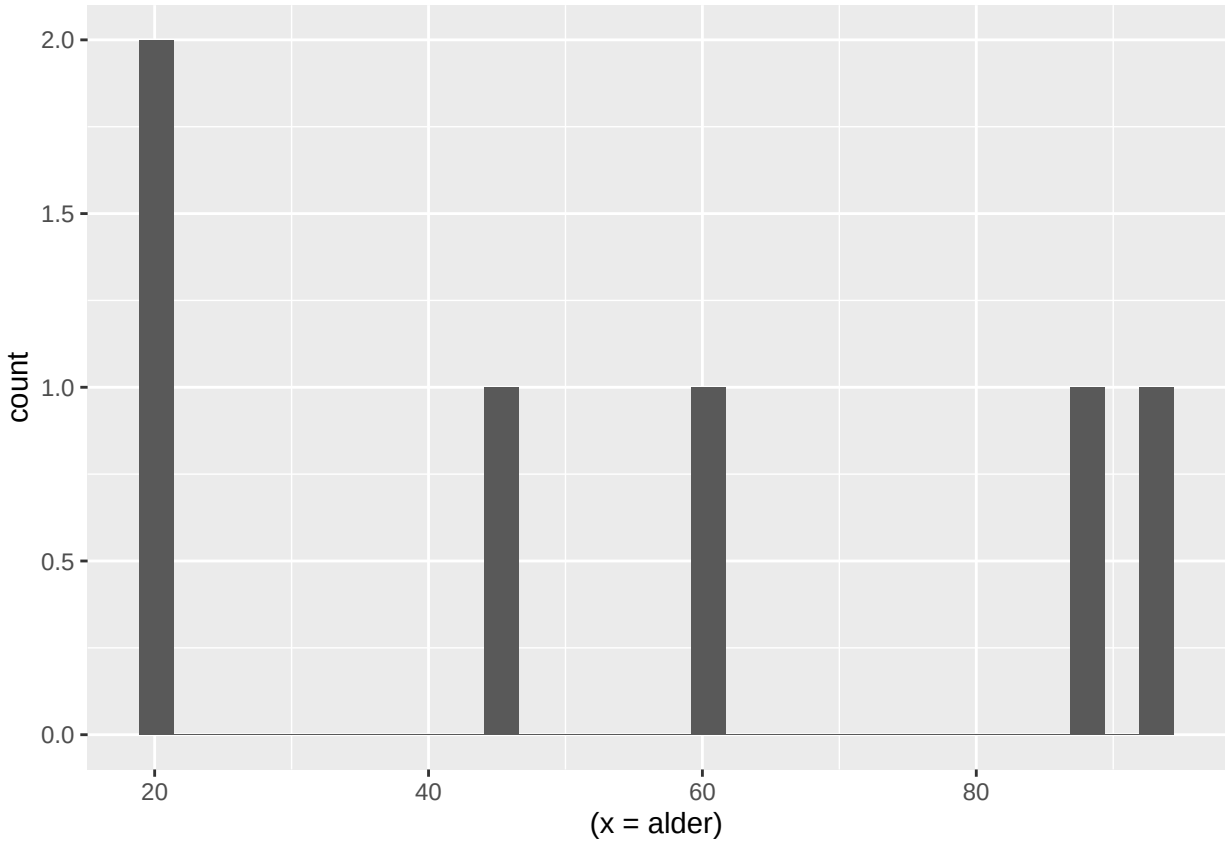
Her bruker vi en del av tidyverse, som heter ggplot. Ggplot er en måte å lage grafikk i R på.

```
#For å lage en figur starter vi alltid med å definere datasettet, og kan velge
# å definere variabler

# Første argument er navnet på datasettet, så skriver man aes() som står for
# aesthetic. Der kan man skrive navnet på variabelen man vil se på. Jeg skriver
# også en + fordi jeg skal legge til mer på neste linje. Så velger jeg hva slags
# type plott jeg vil ha, denne gangen et histogram

ggplot(data = personer, aes((x = alder))) +
  geom_histogram()
```

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



Det var det for denne gang! På Canvas kommer det til å ligge noen oppgaver dere kan jobbe med, og bare send spørsmål om dere har noen! Gleder meg til neste seminar!