R-seminar 1

STV1020 Vår 2021

Uke 9

Introduksjon til R

Gjennom seks seminarganger skal vi gå gjennom alt fra hva R er og hvordan det fungerer, til å kjøre regresjonsanalyser. Før hvert seminar kommer jeg til å legge ut et dokument som dette. Dette har jeg faktisk laget i R! Her vil finnes det både tekst og kode. Dokumentet inneholder det vi skal gå gjennom på seminaret, og er mer utfyllende enn scriptene vi bruker på selve seminaret.

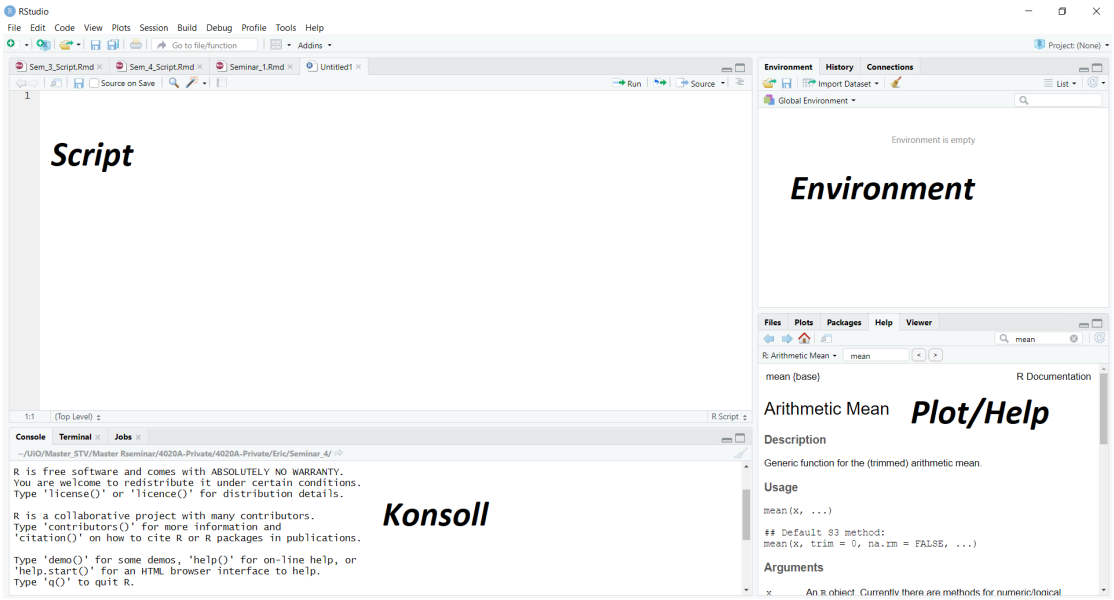
```
# Kodene skrevet i dette dokumentet har grå bakgrunn. Alt som er skrevet her kan
# dere kopiere og lime inn i R på egen pc. Kjør koden og se hva som skjer.
# Når jeg har # foran betyr det at jeg skriver en kommentar.
# Dette er vanlig å ha med i scriptene for å forklare hva som skjer,
# uten at R prøver å kjøre det som en kode, som vil gi feilmelding.
# Hvis jeg kjører kode her, vil dere se resultatene som vanlig tekst under.
# Vi kan teste med enkelt matte. For å kjøre koden, trykker dere ned musepekeren
# foran eller etter kodesnutten, og deretter ctrl+enter på tastatured.

100/2+4

## [1] 54
```

R, RStudio, funksjoner og Syntax-feil

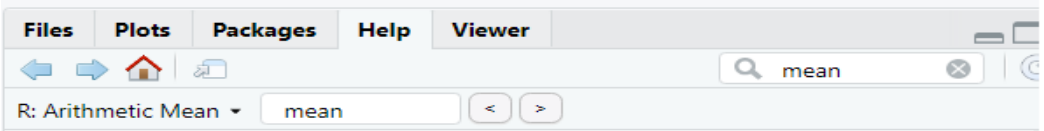
Før seminaret har dere lastet ned R og RStudio. R er programmeringsspråket, mens RStudio er programmet vi bruker for å skrive R. Når vi laster ned R, laster vi samtidig ned et program som gjør at datamaskinen kan forstå det vi skriver. R-scriptet er der man skriver koden. Her har vi for eksempel lett tilgang til hjelpefiler, som er nyttig for å forstå oss på de ulike funksjonene i R.



RStudio har flere vinduer. Øverst til venstre finner dere scriptet. Her skriver vi kode vi ønsker å lagre og bruke videre. Under denne er "konsollen". Når dere kjører kode vil dere se at 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. Til slutt, nederst til venstre, vises en del informasjon, hvor det i hovedsak er to faner vi kommer til å fokusere mest på. Den ene er "Plots" som viser grafikk vi har laget, f.eks. et histogram, Den andre er hjelpefilene, hvor man kan slå opp hva forskjellige funksjoner gjør. Denne kan vi 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

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

I hjelpefilen får vi en go del informasjon om funksjonen. Men hva er en funksjon? I R jobber vi som oftest med forskjellige typer objekter. Objekter er virtuelle "bokser" hvor vi kan legge informasjon til senere bruk. Et objekt kan for eksempel inneholde informasjon som en tallrekke fra 1 til 10. Når vi har slik data, ønsker vi å hente ut informasjon fra dem. Det kan for eksempel være gjennomsnitt, standardavvvik osv. Vi bruker funksjoner til å gjøre noe med dataene, de er ferdige kodesnutter som er laget for å utføre bestemte oppgaver. Alle funksjoner har til felles at de tar data og skaper et resultat. I tillegg kan funksjoner ha flere argumenter, som gjør at vi kan endre noe på hvordan funksjonen lager resultatet. Argumenter er tilleggsinformasjon vi kan legge til i funksjonene. Dersom vi leser hjelpefilen til "mean", ser vi at den øverst gir en beskrivelse, hvor det står at den returnerer gjennomsnittet. Under der kommer argumentene den godtar, et objekt (x) som inneholder tall, logiske verdier, et argument for å trimme dataene, altså fjerne noe, og na.rm argumentet. Det siste skal vi komme tilbake til. Under Value får vi en beskrivelse av hva som er returnert, før til slutt et eksempel av hvordan den brukes i bunn.

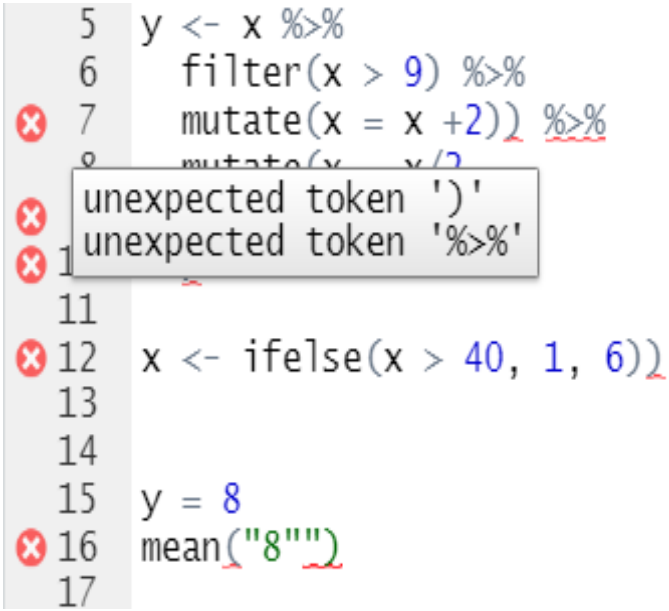
Hjelpefilene er en fin måte å finne ut av hva en funksjon gjør, og hvordan vi kan bruke den. Samtidig kan den ofte være litt kronglete å lese, men da hjelper det ofte å se på eksemplene som alltid er i bunnen av filen. Skulle det fortsatt være vanskelig, er det viktig å huske på at det finnes et stort miljø rundt R, og ofte er det mange som har opplevd samme problem som deg! Med google kommer du ofte langt! For eksempel går det an å google: "How to find mean in R".

Prosjekter og mapper

- Prosjekter hva/hvordan/hvorfor
- Mappestruktur, hold det enkelt

Syntax-feil

Når vi går gjennom kode hender det ofte at vi får feilmeldinger. Det er helt vanlig, og nesten uungåelig. En type feil kan likevel være verdt å merke seg med en gang. "Syntax-feil" er skrivefeil vi gjør når vi skriver kode. For eksempel 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.



Som dere ser på bildet, dukker det opp røde kryss ved siden av linjenumrene. Dette er steder hvor RStudio mener jeg har gjort feil, og holder jeg musen over dem får jeg opp hva som er feilen. "Unexpected token ')'" betyr at RStudio mener det er en parantes der som ikke skulle vært der. I tillegg er det røde streker under de delene av koden som RStudio mener er feil. Mange feil er enkle feil som dette, at man skriver komma der det egentlig skulle vært punktum, osv.

Objekter, vektorer og klasser

Objekter og vektorer

Vi har allerede sett på litt enkel kode, men framover skal vi gått litt dypere inn i hvordan kode faktisk fungerer. Koder forteller datamaskinen hva vi vil den skal gjøre gjennom tekst. Istedenfor å trykke på nedtrekksmenyer, som på Word, skriver vi her kun tekst. Den vanligste måten å lære nye koder på er å google etter spesifikke ting du ønsker å gjøre. Men denne seminargangen blir dere også kjent med grunnleggende koder, funksjoner m.m.

Det første vi skal se på nå er objekter og vektorer. Objekter er som nevnt ting hvor vi kan legge informasjon til senere bruk. En vektor er en serie med tall eller bokstaver/ord som er lagret i et objekt. Den lages ved hjelp av kommandoen c().

Det er lettest å vise ved eksempel:

```
# Her ønsker jeg å lage et objekt. La oss først prøve å lage en med ett
# tall. For å gjøre dette må vi først velge et navn, så bruke det som heter en
# "assigner", før vi skriver hva den skal inneholde.
# Her lager jeg et objekt som heter "To" og som inneholder tallet 2.
To <- 2

# <- er assigneren. Den sier at det som kommer på venstresiden skal lagres med
# navnet som er på høyresiden. Om dere kjører 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 navn To som inneholder verdien 2. Nå som vi har et objekt kan vi
```

```r
# begynne å bruke det til noe. Først kan vi prøve å gjøre matte igjen:
2 + To

## [1] 4

# Som dere ser kan jeg nå skrive 2 + To og få ut resultatet fire. Når vi nå
# skriver "To" vet R at vi egentlig mener tallet 2. Men for så enkle ting som
# dette er nok enklere å bare skrive 2 eller bruke en kalkulator. Uansett,
# det fine med objekter er at de kan inneholde mye informasjon!

# Vi kan også spørre R direkte om et uttrykk er lik, større/mindre enn, ulikt.
# Disse spørsmålene er svært nyttige når man skal omkode variabler
2 == To #er lik/sant

## [1] TRUE

2 != To #er ulik

## [1] FALSE

2 < 3 #objekt større enn et tall

## [1] TRUE

2 > 3 #objekt mindre enn et tall

## [1] FALSE

# Vi kan også forsøke å lage vektorer, som altså inneholder serier med tall
# eller bokstaveer. La oss forsøke med tall. Det er flere måter vi kan gjøre
# dette på. Vi kan 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. c() binder
# samen disse tallene. I det siste skiller jeg tallene med komma.
# Objektene kan hete hva du vil forøvrig.
Hva_Du_Vil <- 1:100
Forovrig <- c(1,4,56,8,4,2,4)

# Merk, man kan ikke ha mellomrom i navnene eller tall som første tegn.
# Man burde også unngå æ/ø/å i scriptet.

# Nå som vi har et script med flere elementer kan vi prøve å kjøre noen
# funksjoner på dem. La oss se om vi kan finne gjennomsnittet av disse vektorene.
mean(Forovrig)

## [1] 11.28571

mean(Hva_Du_Vil)

## [1] 50.5
```

Med mean()-funksjonen ser vi at vi får gjennomsnittet for hele vektoren. Som oftest er det det vi ønsker, men hva hvis vi kun vil ha gjennomsnittet av noen av tallene? Om dere ser i environmentet, ser dere at etter navnet på vektoren står

det først "num" og så [1:7]. Den første teksen sier at dette er et numerisk objekt. Dette dreier seg om klasser, og er noe vi straks skal se på. Det neste viser lengden på vektoren vår. Forovrig har sitt første tall i plassen 1, og siste i 7. Altså er det 7 elementer. Om vi ser på hva du vil ser vi at det står 1:100, og denne har altså 100 elementer. For å få tak i et spesifikt element kan vi bruke disse klammeparantesene [].

```
# La oss si at vi vil ha element nr. 5 i vektoren Forovrig.
Forovrig[5] #Når vi kjører denne ser vi at vi får ut tallet 5,

## [1] 4

# Dette kan vi også sjekke i envorement for å se om stemmer.

# På samme måte som vi definerte en rekke tall i stad, kan vi også bruke dette
# for å få ut en rekke elementer.
Forovrig[3:6]

## [1] 56  8  4  2

Forovrig[c(3,5,3,6)]

## [1] 56  4 56  2

# Her kan vi også finne gjennomsnittet av kun disse tallene
mean(Forovrig[c(3,5,3,6)])

## [1] 29.5

# Eller bruke disse som en ny vektor ved å lagre i et nytt objekt
Ny_Vektor <- Forovrig[c(3,5,3,6)]
```

Klasser

Så langt har vi kun jobbet med tallverdier. Ofte har vi variabler som ikke er tall, men 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 og kan inneholde desimaler. De fleste mattefunksjoner krever at dataene er numeric.

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

## [1] TRUE

# Her ser vi at vi får opp "TRUE" som betyr at det stemmer at Hva_Du_Vil
# er et numerisk objekt/vektor
```

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.

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

```
Tekst <- "skriv tekst her"
# Denne klassen kan inneholde tekst, men vil ikke kunne brukes til matematiske
# operasjoner
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, for eksempel
# hvis vi gjør Forovrig om til character.
Forovrig <- as.character(Forovrig)
mean(Forovrig)

## Warning in mean.default(Forovrig): 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 environmentet står det også nå at Forovrig er chr (charater) 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 factorer på er å tenke på dem som ordinale variabler, hvor vi kan vite rekkefølgen på nivåene men ikke avstanden, f.eks. Barneskole, Ungdomskole, videregående.

```
# Faktorer er vektorer der hvert element er en kategori.
Skolenivaer <- factor(c("Barneskole", "Ungdomskole", "Videregaende",
                        "Videregaende", "Ungdomskole"),
levels = c("Barneskole", "Ungdomskole", "Videregaende"))

# Først definerer vi de forskjellige verdiene som er i variabelen.
# Så skriver vi hvilke nivåer den kan ha, i den rekkefølgen vi ønsker.
# Om vi ikke hadde definert nivåene ville R gjort det automatisk i alfabetisk
# rekkefølge, som oftest går det greit men noen ganger ønsker vi det annerledes
# Nå kan vi først se på hva som er i variabelen
Skolenivaer #Kjører vi bare denne ser vi alle verdiene

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

```r
levels(Skolenivaer) #Og får ut de tre nivåene

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

Tidligere i dokumentet stod det at en vektor er et objekt som inneholder elementer av samme klasse. Så langt har vi også holdt oss til det gjennom å kune lage objekter med tekst eller tall. Hva skjer da om vi prøver å blande?

```r
# Nå kan vi lage et objekt som inneholder både tekst og tall:
TekstTall <- c(1, 4, 0, 4, "Ja", "Nei", "R-Seminar", 42, "Tekst")

# Nå kan vi bruke funksjonen "class()" for å se hvilken klasse dette nye
# objektet har
class(TekstTall)

## [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 ett 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 lastes som regel ned fra nettet når en skal ha data, for eksempel fra en surveyundersøkelse. Men, vi kan også lage dem selv. En viktig regel for dataframes er at alle vektorene må ha lik lengde. Om vi dermed mangler noen observasjoner må vi finne en måte å "fylle" disse tomme cellene. Det gjør vi med NA.

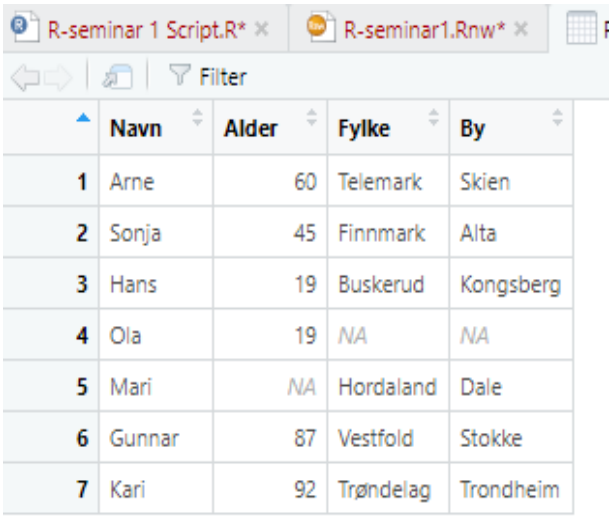
```r
# Lagrer informasjon i objekter, men vektorer
Navn <- c("Arne", "Sonja", "Hans", "Ola", "Mari", "Gunnar", "Kari")
Alder <- c(60, 45, 19, 19, NA, 87, 92)
Fylke <- c("Telemark", "Finnmark", "Buskerud", NA, "Hordaland", "Vestfold",
           "Trøndelag")
By <- c("Skien", "Alta", "Kongsberg", NA, "Dale", "Stokke", "Trondheim")
# Her lager jeg først et sett med vektorer, med litt forskjellig informasjon.
# Dere kan se i environment 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 funksjonen data.frame()
Personer <- data.frame(Navn, Alder, Fylke, By)
```

I environment dukker det opp en ny type verdi, under "Data" med navnet Personer. Når det står 7 obs (observasjoner) of 4 variables, betyr dette at vi har en dataframe med 7 rader og 4 kolonner. Klikker dere på den vil dere se datasettet.

```
View(Personer)
```



	Navn	Alder	Fylke	By
1	Arne	60	Telemark	Skien
2	Sonja	45	Finnmark	Alta
3	Hans	19	Buskerud	Kongsberg
4	Ola	19	*NA*	*NA*
5	Mari	*NA*	Hordaland	Dale
6	Gunnar	87	Vestfold	Stokke
7	Kari	92	Trøndelag	Trondheim

Første observasjonen her er rad 1, som er Arne på 60 år fra Skien i Telemark. Det viktigste med en dataframe er at vi nå 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:
Personer[2, 1] #Med klammeparanteser kan vi velge rad og kolonne.

## [1] "Sonja"

               #Rad først, så kolonne.
Personer[ ,2]  #Skriver vi en tom rad, får vi alle verdiene i kolonne 2

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

Personer[2, ]  #Skriver vi en tom kolonne, får vi alle verdiene i rad 2

##    Navn Alder    Fylke   By
## 2 Sonja    45 Finnmark Alta

# Noen ganger ønsker vi å velge ut noen grupper i datasettet.
# Samtidig blir det fort vanskelig å huske tallet til plasseringen,
# En vanligere måte å hente ut kolonner på er med dollartegn, '£'.
Personer$Alder #Her skriver jeg først navnet på dataframen, og så variabelen

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

# Da får vi alle verdiene på variabelen alder

# Her kan vi bruke mattematiske formler på samme måte som i stad.
# La oss prøve å få ut gjennomsnitt og alder på personene.
mean(Personer$Alder)

## [1] NA
```

Her fikk vi NA til svar istedenfor for det gjennomsnittet vi ønsket. NA betyr missing, altså at vi ikke har informasjon om noe. For Mari i datasettet har vi ikke informasjon om hvor gammel hun er. Når minst en av verdiene er NA vil flere funksjoner også returnere NA. Dette fordi vi ikke kan vite gjennomsnittet om vi ikke vet alle verdiene. For å få ut et resultat må vi derfor fortelle R at vi ønsker å fjerne NA verdiene, og heller få gjenomsnittet av de verdiene som er tilstede.

```
mean(Personer$Alder, na.rm = TRUE) #Her ser dere at vi får svaret 53.6667.

## [1] 53.66667

# na.rm betyr NA remove, og når vi setter den til TRUE ber R
# om å fjerne disse NA.

# Vi kan bruke funksjonen median() for å finne median
median(Personer$Alder, na.rm = TRUE)

## [1] 52.5

# En enklere måte å få ut alle disse på er ved å bruke summary()-funksjonen.
# Da trenger vi heller ikke bruke na.rm, fordi den heller sier hvor mange NA
# det er
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 idag er en kort introduksjon til hvordan vi kan visualisere data. For å gjøre dette må vi først laste ned en pakke som heter Tidyverse. Pakker er tilleggsmoduler til R som gjør at du kan laste ned flere funksjoner, og ofte gjør visse ting enklere. "Raw" R, når det lastes ned uten noen pakker, kalles "base R." Om noe er vanskelig i base R, finnes det høyst sannsynlig en pakke som gjør det lettere. Tidyverse, som vi vil bruke mye, er et sett med pakker som gjør databehandling mye enklere. For å bruke denne må vi først innstalere pakken. Om dere har gjort dette på forhånd trenger dere ikke gjøre dette på nytt. Å installere gjør vi kun en gang, og så evt. på nytt om det kommer en oppdatering.

```
install.packages("tidyverse")

## Installing package into 'C:/Users/sk_gr/OneDrive/Documents/R/win-library/4.0'
## (as 'lib' is unspecified)
## Error in contrib.url(repos, "source"):  trying to use CRAN without setting a mirror

# For å installere bruker vi funksjonen install.packages(), og skriver navnet
# på pakken i parentes med hermetegn rundt
```

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. Da henter vi den opp fra biblioteket.

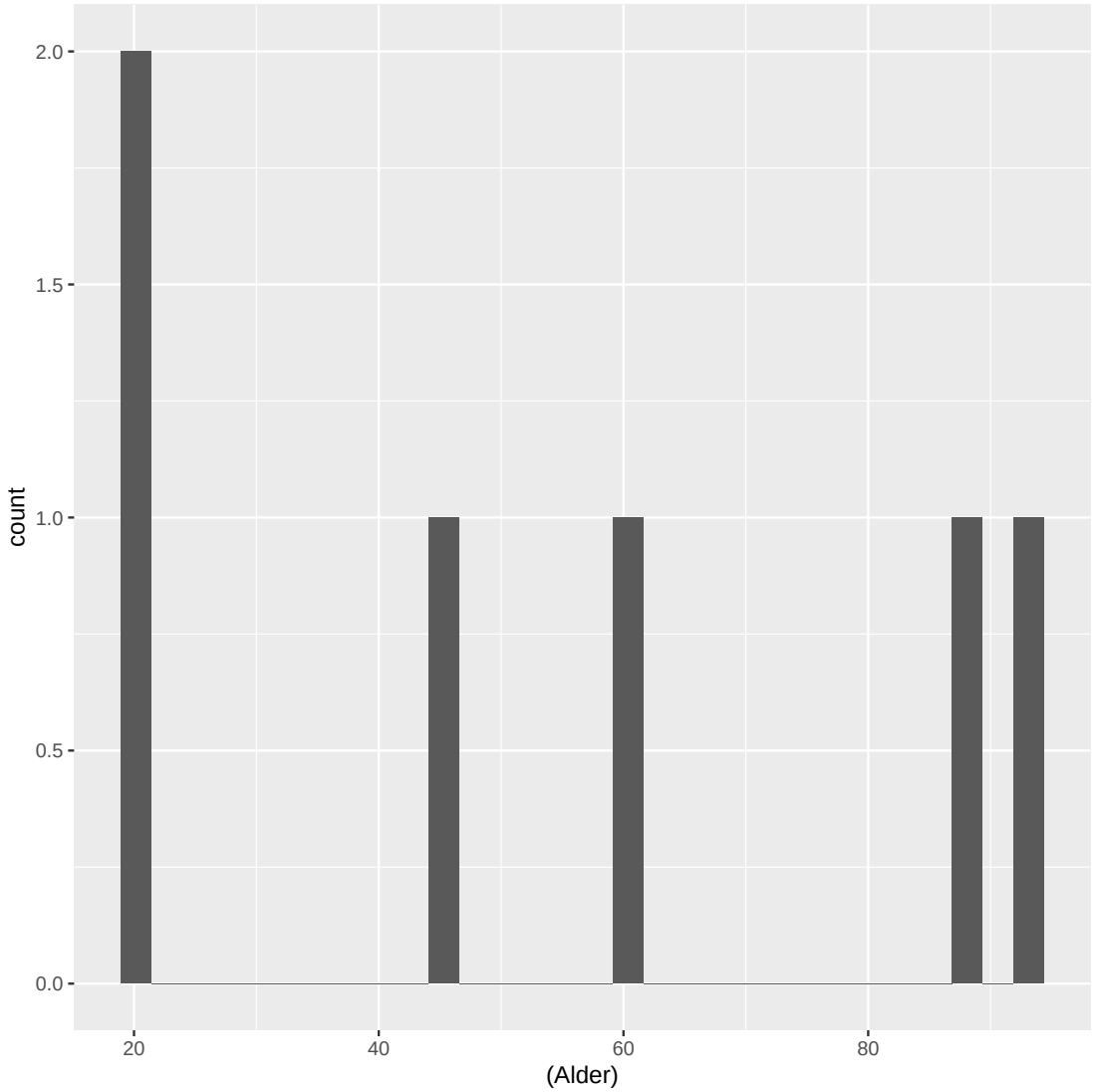
```
# For å gjøre dette bruker vi funksjonen library(), men uten hermetegn
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.0
--
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.6     v dplyr   1.0.3
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts()
--
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tidyverse skal vi bruke masse tid på utover i seminarrekken, men her kommer en liten introduksjon på en del av det 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 variabler
ggplot(Personer, aes((Alder))) + #Første argument er navnet på datasettet.
                                 #Så skriver jeg aes() som står for aesthetic.
                                 #Der kan vi skrive navnet på variabelen.
                                 #Jeg skriver også en + fordi jeg skal legge
                                 #til mer på neste linjem et nytt argument
  geom_histogram() #Her velger jeg hva slags type plott jeg vil ha,

## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
## Warning:  Removed 1 rows containing non-finite values (stat_bin).
```




```
                    #denne gangen et histogram
```

På Canvas og GitHub ligger det noen oppgaver dere kan jobbe med. Benytt dere av hverandre dersom dere lurer på noe! Om du sliter med noe, er det stor sannsynlighet for at en annen gjør det samme, eller at dere kommer frem til løsning i fellesskap. Lykke til!