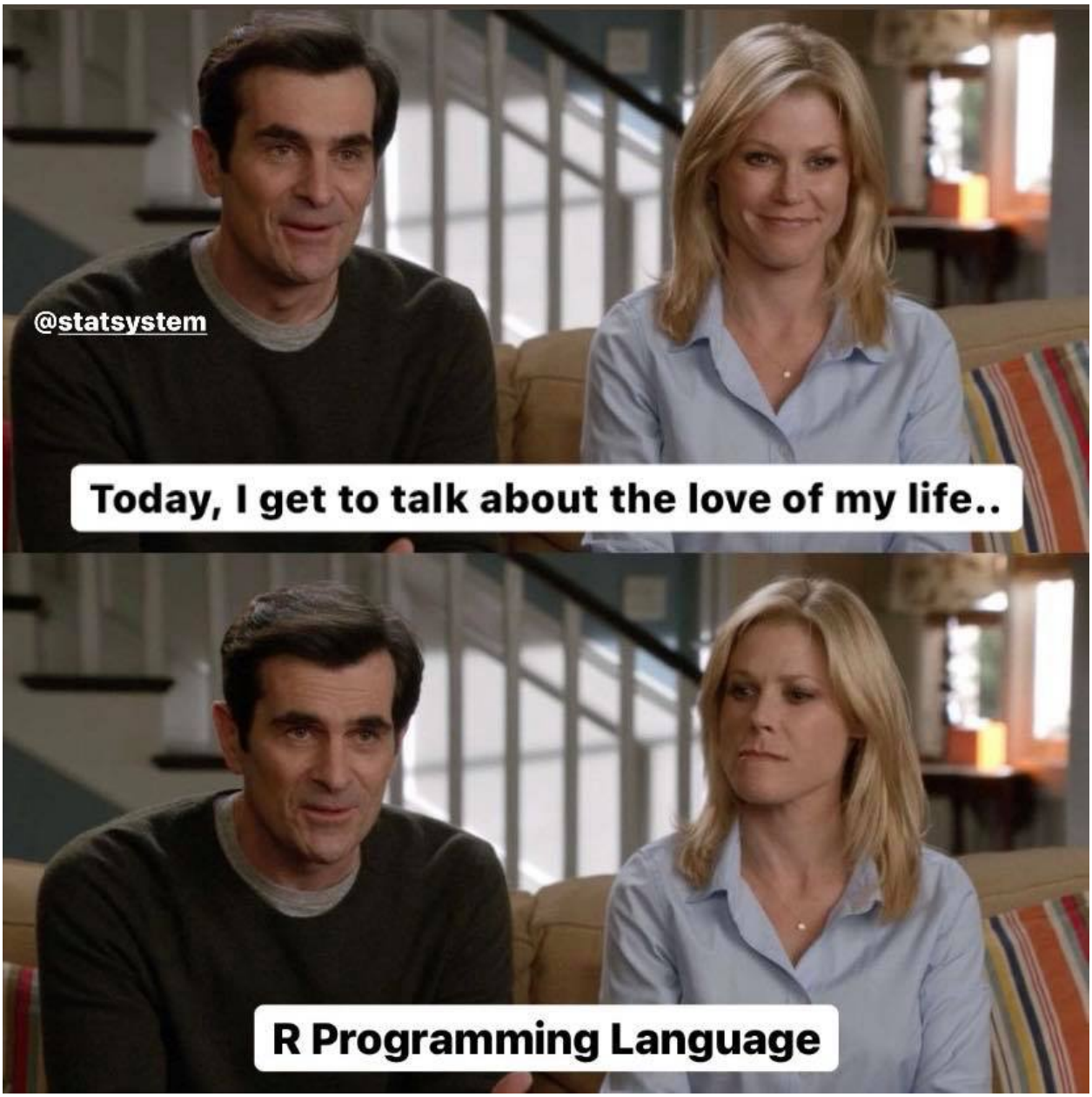
Seminar 1

Eric

18 1 2021

Velkommen til første seminar!



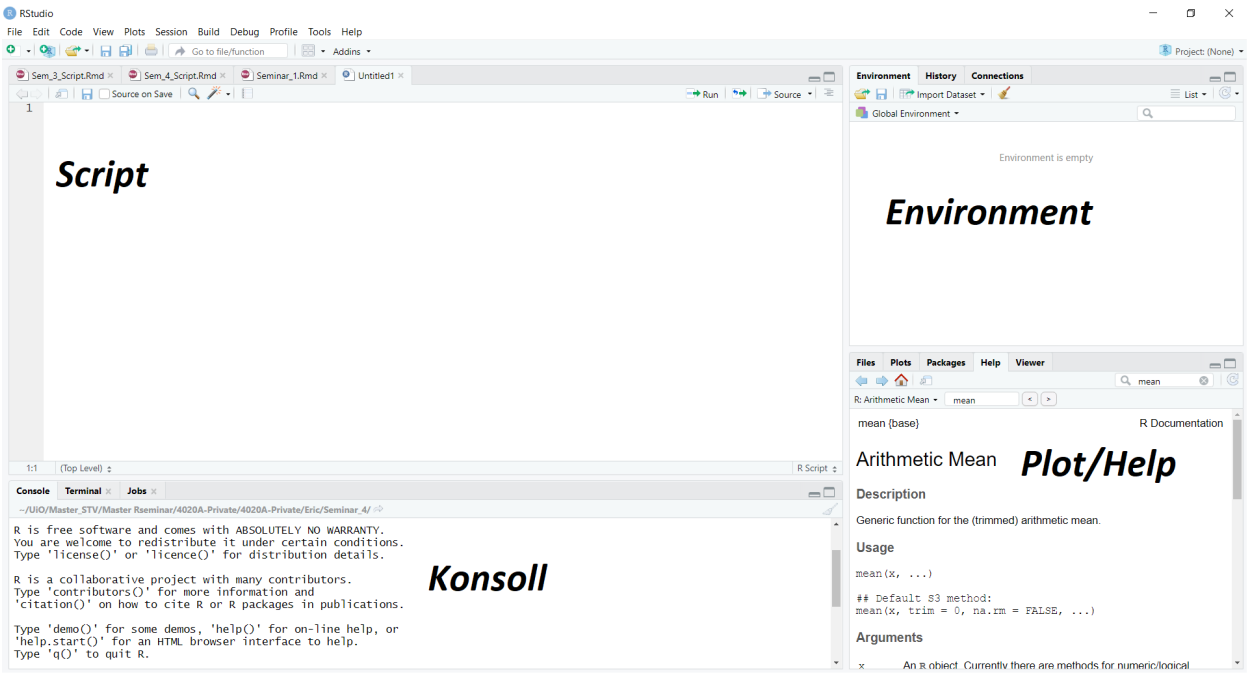
Gjennom 6 seminarganger skal vi nå gå igjennom alt fra hva R er og hvordan det fungerer, til å kjøre våre helt egne regresjonsanalyser og holde på med *egen statistikk.* Jeg gleder meg! Før hvert seminar kommer jeg til å legge ut et dokument som dette. Dette har jeg faktisk laget i R! Her vil dere se at jeg har noe tekst (og memes, dere må like memes), og noe kode. Dokumentet er det jeg har planlagt at vi skal gå igjennom på seminaret, men det kan godt hende at vi gjør noe mindre der. Etter det siste kommer det til å være en kort prøve. Om dere har noen spørsmål må dere bare sende en melding! Det kan dere både gjøre på e-post (egnilsen@student.sv.uio.no), og på Canvas!

```
#Kode vil dere se har en egen grå bakgrunn. Alt som er skrevet her kan dere kopiere inn på deres egen p
#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 fors
# med noe enkel matte! For å kjøre koden setter dere musen ved siden av, og trykker ctrl+enter.
100/2+4
```

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

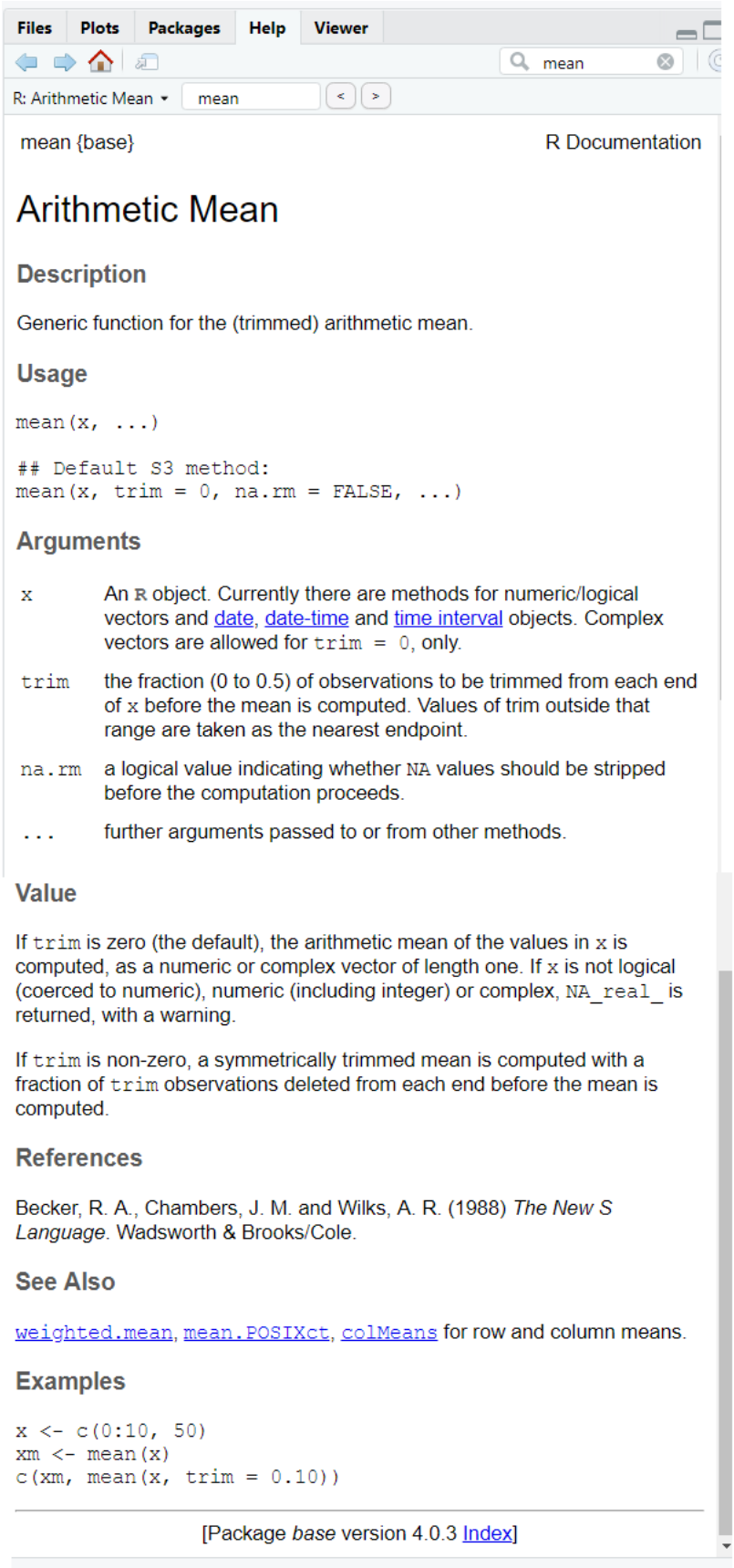
R, RStudio, og Syntax-feil

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-skriptet, 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 skriptet. 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 skriptet. Det er her vi vil skrive kode vi ønsker å lagre, og bruke videre. Under denne er det vi kaller enn "Konsoll" 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 "envoirement", her vises alle objekter som vi har laget i skriptet, 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 gjøre bruk av. Den ene er "Plots" som veldig enkelt 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 logisk nok finner gjennomsnitt:
?mean
```

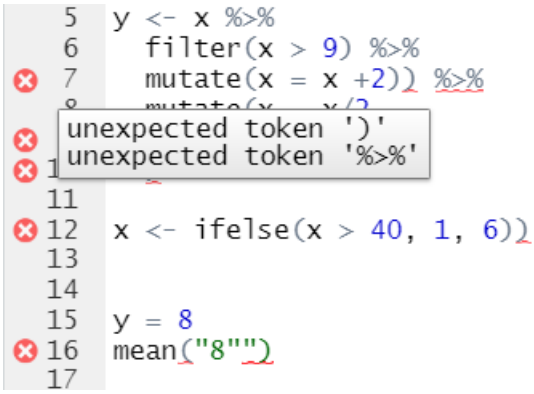

Files
Plots
Packages
Help
Viewer
mean
R: Arithmetic Mean
mean
mean {base}
R Documentation
Arithmetic Mean
Description
Generic function for the (trimmed) arithmetic mean.
Usage
mean(x, ...)
Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
Arguments
x
An R object. Currently there are methods for 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))
[Package base version 4.0.3 Index]

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 rekke tall. Si vi har tallene fra 1-10. Hovedpoenget med R er at når vi har slike data, ønsker vi å hente ut noe *informasjon* fra dem. Dette kan f.eks. være gjennomsnitt, standardavvvik ++. En funksjon er det vi bruker i R for å gjøre noe med dataene, f.eks. å hente ut denne informasjonen. Alle funksjoner har til felles at de tar noe data, f.eks. tall, og skaper et resultat. I tilegg har noen 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 der 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, før til slutt et eksempel av hvordan den brukes i bunn.

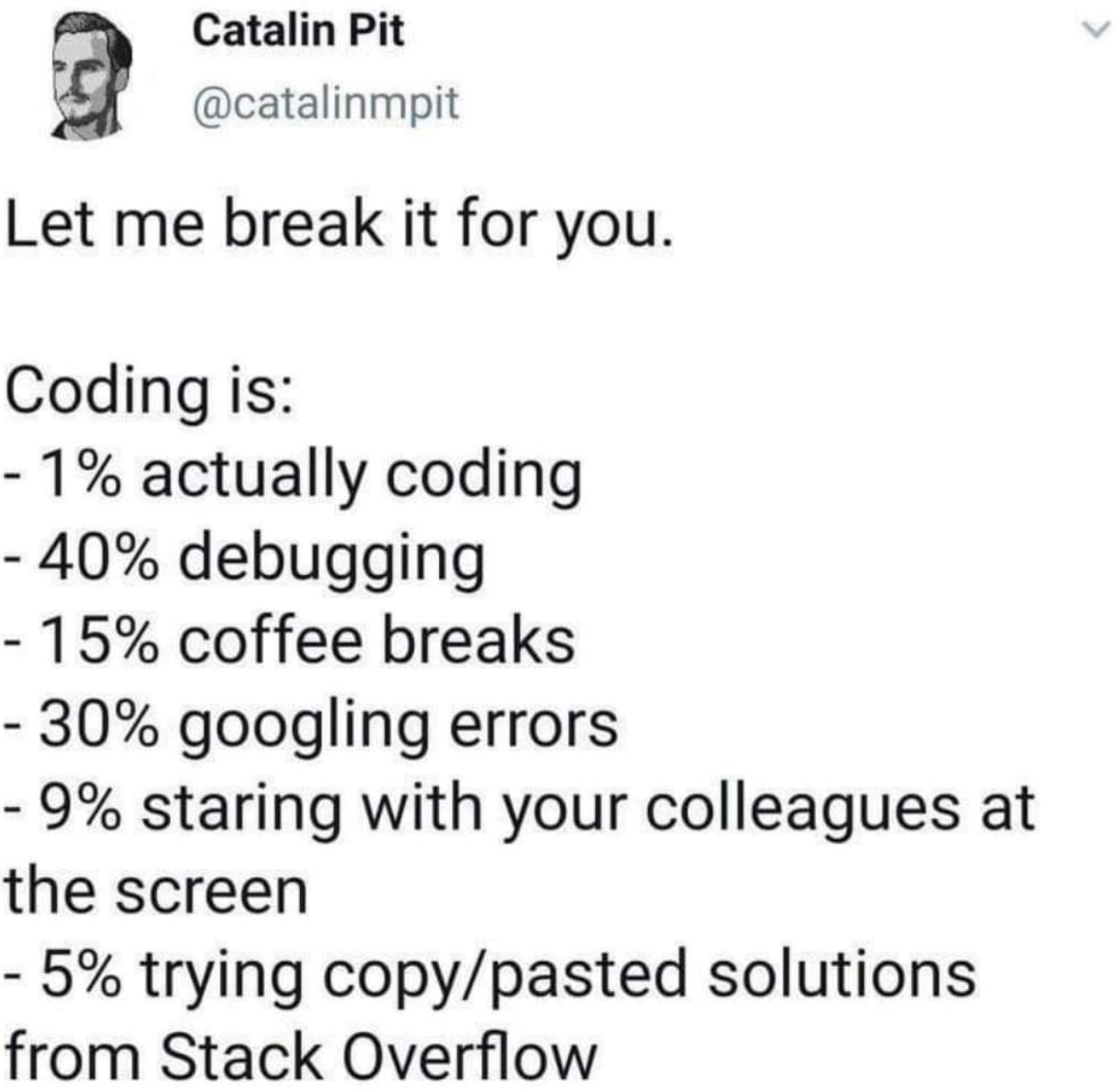
Hjelpefilene er en flott måte å finne ut hva en funksjon gjør, og hvordan vi kan bruke den. Samtidig kan den ofte være litt kronglete å lese, men da hjelper det ofte å se på eksemplene som alltid er i bunn av teksten. Skulle det fortsatt være vanskelig er det viktig å huske at det finnes et stort miljø 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! Dere vil nok fort merke at jeg gjør en hel del feil når jeg skal vise dere i seminaret. En type feil kan likevell være godt å merke seg med en gang, mest fordi jeg gjør den hver gang jeg skal skrive noe kode. "Syntax-feil" 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 har jeg skrevet noe tullekode for å vise ossen dette funker. Som dere kan se på siden får jeg flere røde kryss ved siden av linjenummerene. 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 tilegg er det røde streker under de delene av koden som RStudio mener er feil. Jeg vil hevde, hvertfall etter min egen erfaring, at 90 % av feilene vi gjør i R er enkle skrivefeil/syntax-feil som dette. Derfor er det veldig nyttig at RStudio viser det 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. Logisk nok er koding en måte fortelle pc'n hva vi vil den skal gjøre gjennom tekst. Sånn sett er det nærmest det samme som når du klikker på noe, bare at dette er noe mer effektivt. Når vi skriver kode må vi vite hva vi skal skrive for at pc'n skal forstå det. All den tid det ikke finnes noen enorm ordbok som forteller alt vi kan skrive i R er dette noe vi må lære oss, og kanskje pugge, eller bare søkte etter når vi trenger det. Den vanligste måten å lære nye koder på er å google etter spesifikke ting du ønsker å gjøre.

Det første vi skal se på nå er objekter. Objekter er i alle "ting" i R som kan inneholde noe annet. Vi skal i hovedsak forholde oss til to typer objekter vektorer, og funksjoner. Hva disse er nok lettere å vise ved eksempel:

```
#Her ønsker jeg å lage et objekt. 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 ett
```

```
#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 det som er assigneren. Den sier bare at det som kommer på venstresiden skal lagres med
#navnet som er på høyresiden. Om dere kjører koden (ctrl+enter med musen ved siden av, eller
#teksten markert) vil dere se i envoirnment 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 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. For så enkle ting som dette er sikkert enklere å bare skrive 2, eller
#bare bruke en kalkulator for den saks skyld. 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. Objektene kan hete hva du vil forøvri
Hva_Du_Vil <- 1:100
Forovrig <- c(1,4,56,8,4,2,4)
#Eneste er at du ikke kan ha mellomrom i navnene eller tall som første tegn, og det er god kutyme å unn
#æ/ø/å generelt i script.
#Nå som vi har et script med flere elementer kan vi prøve å kjøre noen funksjoner på dem
#Det kan nevnes at funksjoner faktisk er objekter de og, men det blir først innteresant når du holder
#på med litt mer avansert kode. 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 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 tall? Om dere ser tilbake til envoirenment 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. *Forovrig* har sitt første tall i plassen 1, og siste i 7. Altså er det 7 ellementer. 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
```

```
            #og dette kan vi jo også sjekke i envorement for å se at stemmer.

#På samme måte som vi definerte en rekke tall istad, 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
```

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

```
## [1] 29.5
```

```r
#Eller bruke disse som en ny vekter
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 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(Hva_Du_Vil)
```

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

```r
#Her ser vi at vi får opp "TRUE" som betyr at Hva_Du_Vil 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.

```r
Tekst <- "Hei, jeg elsker R! <3"
#Denne klassen 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
```

```r
#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
#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 test. Om dere ser i envoirement 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, vgs.

```r
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, 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
#Vi kan også se hvilke nivåer som er i variabelen
levels(Skolenivaer) #Og får ut de tre nivåene
```

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

I toppen her 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 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, "Bamse", "R", "R Seminarer er de BESTE seminarer", 42, "the answer")
#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 et objekt. For å gjøre dette bruker vi data.frames. En dataframe består av flere kolloner, hvor hver kollone 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. Data.frames 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. Om vi dermed mangler noen observasjoner må vi finne en måte å "fylle" disse tomme cellene. Det gjør vi med NA.

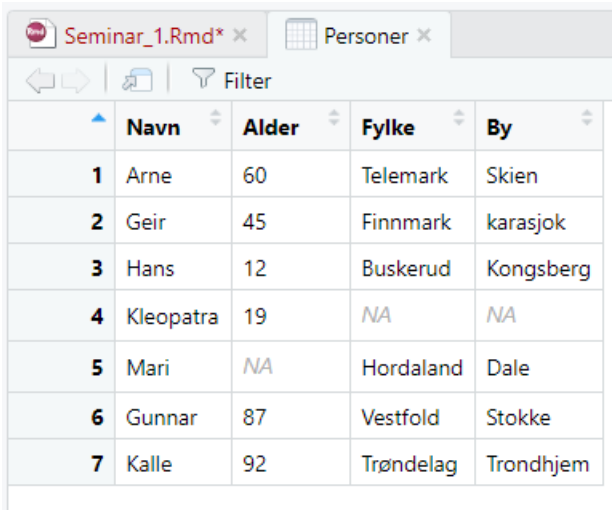
```r
Navn <- c("Arne", "Geir", "Hans", "Kleopatra", "Mari", "Gunnar", "Kalle")
Alder <- c(60, 45, 19, 19, NA, 87, 92)
Fylke <- c("Telemark", "Finnmark", "Buskerud", NA, "Hordaland", "Vestfold", "Trøndelag" )
By <- c("Skien", "karasjok", "Kongsberg", NA, "Dale", "Stokke", "Trondhjem")
#Her lager jeg først et sett med vektorer, med litt forskjellig informasjon.
#Dere kan se i envoirnment at alle har en lengde på 7. Dette kan vi også sjekke med
#length() funksjonen.
length(Navn)
```

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

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

Personer <- data.frame(Navn, Alder, Fylke, By)
```

I envoirnment 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 kolloner. Klikker dere på den vil dere se dette.



	Navn	Alder	Fylke	By
1	Arne	60	Telemark	Skien
2	Geir	45	Finnmark	karasjok
3	Hans	12	Buskerud	Kongsberg
4	Kleopatra	19	*NA*	*NA*
5	Mari	*NA*	Hordaland	Dale
6	Gunnar	87	Vestfold	Stokke
7	Kalle	92	Trøndelag	Trondhjem

Første observasjonen her er rad 1, som er Arne på 60 år fra Skien i Telemark (Norges beste by og fylke forøvrig.) 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 kollone.

```
#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. Rad kommer først, og så kolonnen.
```

```
## [1] "Geir"
```

```
Personer[,2] #Skriver vi en tom får vi alle kollonene/radene
```

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

```
Personer[2,]
```

```
##   Navn Alder    Fylke       By
## 2 Geir    45 Finnmark karasjok
```

```
#Noen ganger er det ønskelig å velge ut noen grupper i datasettet.
#Samtidig blir det fort vanskelig å huske tallet til plasseringen,
#neste gang skal vi derfor se på noen lettere måter å gjøre dette på.

#En mer vanlig måte å hente ut kolonner på er med '$'.
Personer$Alder #Her skriver jeg først navnet på dataframen, og så variabelen
```

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

```
               #Som dere ser får jeg ut verdien på alle aldrene

#Her kan vi bruke mattematiske formler på samme måte som istad.
#La oss prøve å få ut gjennomsnitt og alder på personene.

mean(Personer$Alder)
```

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

Hm. 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 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 jo strengt tatt 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 52.5 istedet.
```

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

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

#For å finne standardavvik kan vi bruke sd() funksjonen
sd(Personer$Alder, na.rm = TRUE)
```

```
## [1] 31.93535
```

```
#Eller median for å finne median
median(Personer$Alder, na.rm = TRUE)
```

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

```
#En lettere 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 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 idag 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 tilegg til R som gjør at du kan laste ned flere funksjoner, og ofte gjør visse ting enklere. R som det kommer når det lastes ned kalles "base R." Om noe er vanskelig i base R, finnes det høyst sansynelig 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 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")
#For å innstalere bruker vi funksjonen install.packages, og skriver navnet på
#pakken i parantesene med hermetegn
```

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 gjøre dette bruker vi funksjonen library()
library("tidyverse")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

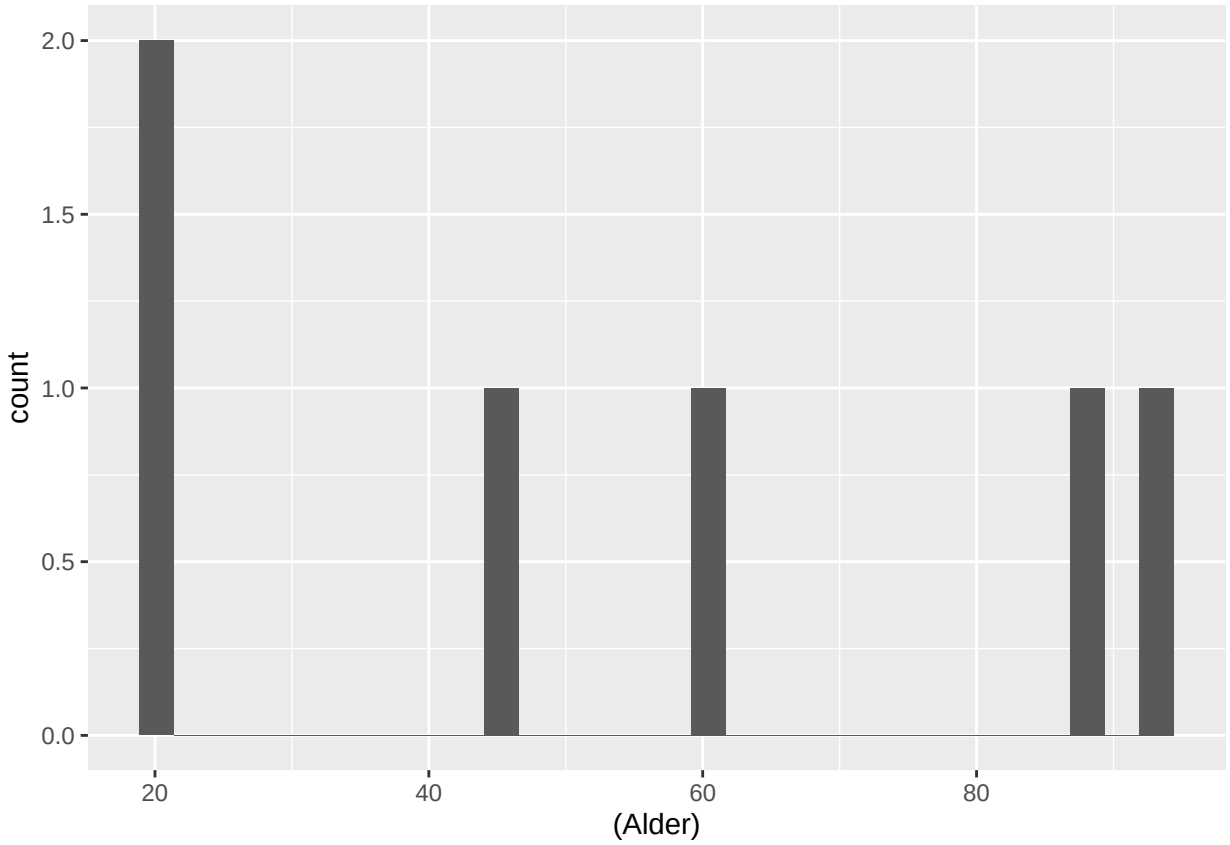
```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tidyverse skal vi bruke masse tid på nesten gang, men akkurat nå skal vi se 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 kan velge
#å definere 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 linje
  geom_histogram() #Her velger jeg hva slags type plott jeg vil ha, denne gangen et 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!