Seminar 1

Eric

17 8 2020

Velkomen til R!

Velkommen til det første seminaret i R! Gjennom 4 seminarganger skal vi nå gå igjennom alt ifra begynnelsen til hva R er, og hvordan det fungerer, til å utføre vår egen logistiske regresjon, dette blir spennende! Før hver seminargang kommer jeg til å legge ut et skript som ser ut som dette. Som dere kan se er det litt annerledes enn de skripene dere skriver selv. Den viktigte forskjelen er at alt dere ser står skrevet som dette med en hvit

bakgrunn er vanlig tekst, og koden vil stå med en grå bakgrunn. All kode kan dere lime direkte inn hos dere selv, og kjøre på egen hånd.

R og RStudio

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 "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 "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 stolpediagramm, og den andre er hjelpefilene hvor en kan slå opp hva forskjellige funksjoner gjør. Denne kan vi faktisk prøve ut med en gang!

```
#Her ser dere at jeg nå skriver kode. Når jeg har en # foran  betyr det
# at jeg skriver en kommentar, som er tekst vi bruker for å forklare hva
#koden vår gjør. Så lenge # er foran vil ikke R kjøre denne delen, og gi
#en feilmelding. Nå ønsket vi å finne en hjelpefil, for å gjøre det kjører vi
# ? + funksjonsnavnet.
?mean #For å kjøre koden setter vi pekeren på samme linje og trykker ctrl+enter
#Her kan vi se at mean() funksjonen returnerer gjennomsnittet av en variabel
```

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!

Objekter, funksjoner, og klasser

Nå som vi har fått åpnet R, og begynt med litt enkel kode kan vi begynne å snakke om objekter. Objekter er rett og slett alle ting i R som vi kan manipulere ved hjelp av kode. Vi kommer i all hovedsak til å forholde

oss til to typer objekter, variabler som er et objekt som inneholder data, f.eks. tall, eller tekst, og funksjoner som er et objekt som gjør noe med en variabel, f.eks. å finne gjennomsnittet. For å lage en variabel bruker "<-". La oss på dette i kode:

```
#Jeg vil ha et objekt som inneholder tallet to, og har navnet "to"

To <- 2 #Når jeg kjører denne koden kan vi se at variabelen dukker opp i
        #envoirement
#Nå kan jeg bruke dette objektet for å gjøre f.eks. matte

2 + To
```

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

```
#Vi kan også lage objekter som inneholder flere ellementer, f.eks. flere tall
Tall <- c(1,2,3,4,5,6,7) #Her bruker jeg c() for å kombinere tallene i ett objekt
                         #og komma for å skille mellom tallene

#Nå kan vi f.eks. be R gi oss gjenomsnittet av disse tallene
mean(Tall)
```

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

Klasser

Alle objekter har en form for klasse, eller "class" som sier hva slags type informasjon et objekt kan inneholde. Visse funksjoner fungerer kun sammen med visse klasser, og det er derfor viktig å holde styr på hva som er hva.

```
#Med funksjonen class() kan vi se hvilken klasse et objekt er:
class(To)
```

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

Her ser dere at R svarer at klassen til objektet "To" er "numeric." Numeric er veldig enkelt en klasse som inneholder et tall. Vi kunne også satt klassen selv, hvis vi f.eks. heller ønsker "integer".

```
To <- as.integer(To)
class(To)
```

```
## [1] "integer"
```

Med funksjonen "as.integer" kan vi gjøre om fra numeric, til integer. Forskjellen her er i praksis veldig sjeldent nyttig, men integer kan kun ha heltall. For de fleste klasser finnes det en "as.klassenavn" funksjon for å gjøre om på objektene, men en bør her være klar over at man kan miste informasjon når en gjør det. Dette kan vi se om vi prøver å gjøre om til "character", som er en klasse for tekst.

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

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

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

Som vi kan se fikk vi nå "NA", isteden for et faktisk gjennomsnitt. NA betyr missing, og vi fikk den fordi vi ikke kan gjøre en mattematisk funksjon på et tekstobjekt.

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 variabeln
#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"
```

Vi kan også se at funksjoner har en klasse, som er "function." Dette kan være nyttig å vite om dere vil gjøre litt mer avansert programering i R, hvor en jo kan bruke en funksjon som et objekt i en annen funksjon.

```r
class(mean)
```

```
## [1] "function"
```

Data og "data.frames"

Når vi jobber med data i R vil vi som oftest ikke lage dataene selv, sånn som her, men få dem fra en kilde, f.eks. European Social Surey, eller SSB. Derfor må vi vitte hvordan vi kan laste inn data fra andre steder. Dere vil fort legge merke til at data kommer i mange forskjellige typer, og stiler. Derfor finnes det også mange måter å laste dem inn på! I seminarene kommer vi i all hovedsak til å benytte det som kalles "comma seperated values", forkortet ".csv", eller .rds filer som er rene R-objekter. Disse kan vi laste inn med funksjonen read.csv(), og readRDS(). Her kan det ofte være lurt å lese hjelpefilen før du bruker den.

```r
ESS_Data <- read.csv("https://raw.githubusercontent.com/egen97/4020A_RSeminar/master/ESS_Selected.csv")
#ESS_Data <- readRDS("file.path") #Her ville vi hatt navnet på en fil
```

Her ser vi at vi har data fra et prosjekt som heter European Social Survey, dette er en spørreundersøkelse som går i flere europeiske land gjennom så langt 9 runder, med data som er innteresant for sammfunsvitere. Selve prosjektet har flere tusen spørsmål, men jeg har valgt ut 24 som jeg tenker kan være spennende for oss. En oppsumering av disse kan dere finne nedenfor:

Variabelnavn og type:

- Time_News: Antall minutter brukt på nyheter hver dag
- Trust_People: "Stoler du på andre mennesker?" 1:10 (10 = Stoler på alle)
- People_Fair: "Er de fleste mennesker rettferdige?" 1:10 (10 = "Most people can be trusted")
- Pol_Interest: "Er du interesert i politikk?" 1:4 (1 = "Høy interesse")
- Trust_Police: "Stoler du på politiet?" 1:10 (10 = "Høy tillit")
- Trust_Politicans: "Stoler du på politikere?" 1:10 (10 = "Høy Tillit")
- Vote: "Stemte du ved sist valg?": 1 = ja/2 = Nei
- Party_Voted_NO: "Hva stemte du på?" (Kun for norske respondenter)
- Left_Right: "Høyre-Venstre skala" 0 = Venstre/10 = Høyre
- Satisfied_Gov: "Hvor fornøyd er du med regjeringen?" 1:10 (10 = "Veldig fornøyd")
- Gov_Reduce_IncomDif: "Burde regjeringen jobbe for å redusere inntektsforskjeller?" 1:5 (1 = "Agree")
- LGBT_Free: "LGBT mennesker bør få leve det live de selv ønsker": 1:5 (1 = "Agree")

- Religous: “Hvor religiøs er du?”: 1:10 (10 = “Veldig religiøs”)
- Climate_Human: “Skjer klimaforandringer pga. mennesker?”: 1:5 (5 = “Kun mennesker”)
- Responsibility_Climate: “Har du et personlig ansvar for klimaforandringene?”: 0:10 (10 = “Stort ansvar”)
- Goverment_Climate: “Gjør regjeringen nok for å stanse klimaforandringene?”: 0:10 (10 = “Ja”)
- Basic_Income: “Støtter du borgerlønn?(Universal Basic Income/UBI)”: 1:4 (4 = “Sterk støtte”)
- Important_Rules: “Det er viktig å følge regler, og gjøre som du får beskjed om”: 1:6 (6 = “Veldig enig”)
- Important_Equal_Oppurtunities: “Det er viktig at alle har like muligheter”: 1:6 (6 = “Veldig enig”)
- Income: Inntekt i desiler (ti-deling)
- Gender: 1 = Mann/2=Dame
- Age: Alder i antall år
- Country: Land
- essround: Hvilken runde de deltok i

Data Wrangling og pakker

Når vi får data er de ofte ikke i den formen, og typen vi ønsker. Dette kan vi se her, hvor vi har alle individene som har svar på ESS noensinne. Dette er åpenbart en vanskelig måte å arbeide med dataene på. For dagens seminar kan vi prøve å arbeide med den siste runden i Norge. For å finne denne må vi gjøre det som kalles “data wrangling”, som er en måte å få data i et format over til et annet som er lettere å arbeide med. Det vi må få gjort her er først å finne ut hvilken runde som er den siste Norge var med i, og så hente ut denne. Det er flere måter vi kan finne den siste runden til Norge i, jeg kommer til å vise dere to alternativer. Den ene er det vi kaller “base R” som kun bruker funksjonene som er i R orginalt. Vi kommer også til og se på hvordan vi enklere kan gjøre det med funksjoner fra Tidyverse. Tidyverse er en pakke, som rett og slett er ekstra funksjoner vi kan laste ned og bruke i R. Det finnes tusenvis av pakker tilgjengelig for R, og mange av dem gjør forskjellige typer arbeid veldig mye enklere.

```
#Først kan vi se hvordan vi kan finne den siste norske runden uten tidyverse:
unique(ESS_Data$essround[which(ESS_Data$Country == "NO")])
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
#Det første jeg gjør her å bruke funksjonen "unique", den sier at R kun skal hente frem unike verdier
#og ikke reptere den samme verdien flere ganger. Videre bruker jeg ESS_Data$essround for å vise hvilken
#kollone i datasettet jeg vil bruke. Klammeparantesen viser at jeg kun vil ha visse rader fra kolonnen.
#For å vise hvilke bruker jeg which() funksjonen som finner de radene som får TRUE på en logisk test,
#testen er hvorvidt Country-kollonen har verdien "NO", som står for Norge. Siden det er dobbelt
#likhetstegn vet R at jeg vil sjekke om de er like.
```

Som dere sikkert ser kan de å gjøre disse tingene i base R være litt kronglete. Derfor vil vi heller bruke en pakke som heter tidyverse. Tidyverse er strengt tatt en samling av flere pakker, laget for å fungere godt sammen. Disse gjør det veldig mye lettere å jobbe med data når en skal forandre på strukturen, o.l. Før vi kan bruke en pakke må vi innstalere den, det gjør vi med install.packages(). Videre må vi kjøre library() hver gang vi skal bruke pakken, sånn at R henter inn funksjonene.

```
#install.packages("tidyverse") Kjør denne for å innstalere, om du ikke har gjort det før
library(tidyverse) #Denne må du kjøre hver gang du åpner R på nytt for å bruke funksjonen
```

```
## -- Attaching packages ------------------------------------------------------------------- tidyvers
```

```
## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ---------------------------------------------------------------------- tidyverse_con
```

```
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
#Tidyverse gir oss flere nye funksjoner, en av de viktigste er den som kalles "pipe" %>%
#Den gjør at vi kan kjøre kode over flere linjer, hvor funkjonene etter  får verdiene vi
#skapte over. Her kan vi prøve å kjøre funksjonen filter() for å finne siste runden til Norge

Norge_9 <- ESS_Data %>% #Her sier jeg at Norge_9 er et nytt objekt, som kommer fra ESS_Data.
  filter(Country == "NO") %>% #Her henter jeg ut kun Norge
  filter(essround == max(essround)) #Her bruker jeg max() for å finne maxverdien på runde,og filtrer ut
```

Univariat og bivariat statistikk

Nå som vi har et datasett å arbeide med kan vi se på litt forskjellige former for statistikk. R er jo nettopp laget for dette, så her finnes det et hav av muligheter for hva dere kan finne ut av! La oss se om vi f.eks. kan finne ut hva gjennomsnittelig tillit til politikere er i Norge, og hva standardaviket er.

```
#For å finne gjennomsnitt kan vi gå tilbake til mean() funksjonen
mean(Norge_9$Trust_Politicians)
```

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

Her fikk vi svaret NA, og det er jo ikke veldig informativt. NA betyr som sagt bare missing, og når minst en observasjon er NA vil funksjonene som oftest gi dette som svar. For å faktisk få et resultat må vi si til R at den skal ignorere NA observasjoner. Dette gjør vi med "na.rm = TRUE" som betyr "NA remove".

```
mean(Norge_9$Trust_Politicians, na.rm = TRUE)
```

```
## [1] 5.280401
```

Nå ser vi at vi får et mye mer logisk svar, nordmenn har generelt *5,2* støtte til politikere, så er jo spørsmålet hvordan en tolker det? Uansett, vi er også nysjerrig i hva standardavviket, altså *gjennomsnittelig avstand til gjennomsnittet*, er. For å gjøre dette kan vi bruke funksjonen "sd()"

```
sd(Norge_9$Trust_Politicians, na.rm = TRUE)
```

```
## [1] 1.975527
```

Standardavviket på denne variabelen er altså ca. lik 2 skalapunktet. For andre univariate mål dere kan prøve ut finnes f.eks. var() som måler varianse, og median() som viser dere medianen, i tilegg til mange fler!

Bivariat

Ofte er vi nysjerrig på om det er en sammenheng mellom flere variabler. På neste seminar skal vi se på OLS, men i dag kan vi se på noe enkelt som korrelasjon mellom variabler med pearsons r. For å finne denne bruker vi funksjonen cor(), med minst to variable:

```
cor(Norge_9$Time_News, Norge_9$Trust_Politicians, use = "pairwise.complete.obs")
```

```
## [1] 0.02084702
```

Her kan dere se at jeg brukte et annet argument for å få frem hva den skal gjøre med NA, nemlig "use". Dette er fordi det finnes flere forskjellige måter å håndtere NA på når det kommer til denne typen modeller. Forskjellene kommer vi tilbake til på neste seminar. Resultatet her viser en veldig svak, men positiv korrelasjon mellom tiden en bruker på nyheter og tilliten til politikere. En kan jo lure på om denne korrelasjonen er signifikant, eller bare pga. tilfeldigheter. For å finne ut av denne trenger vi en signifikanstest:

```
cor.test(Norge_9$Time_News, Norge_9$Trust_Politicians, use = "pairwise.complete.obs")
```

```
##
##  Pearson's product-moment correlation
##
## data:  Norge_9$Time_News and Norge_9$Trust_Politicians
## t = 0.76953, df = 1362, p-value = 0.4417
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.03226621  0.07384284
## sample estimates:
##        cor
## 0.02084702
```

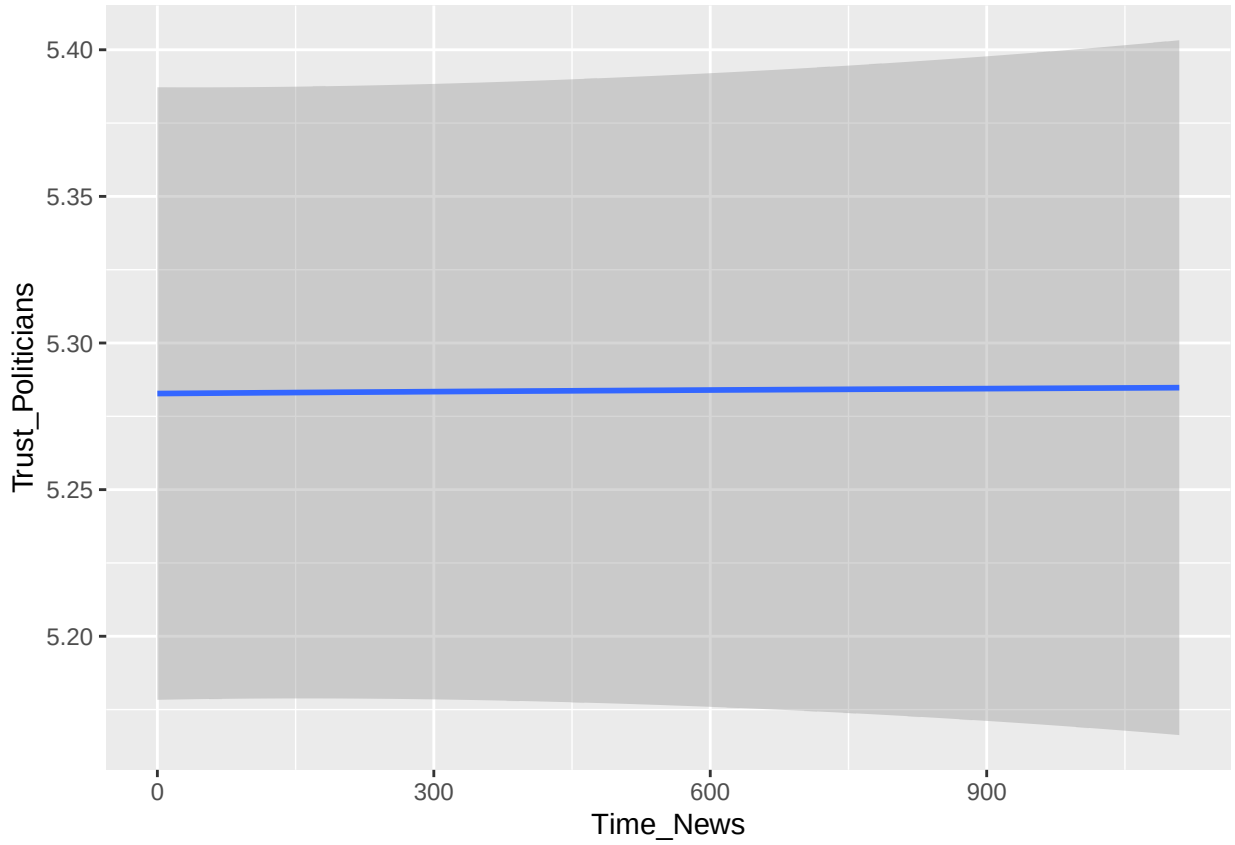
Her kan vi se at korrelasjonen ikke er signifikant, med en p-verdi på ca. *0,5* og et konfidensinterval som krysser null, altså kan vi ikke si at denne korrelasjonen ikke kommer fra tilfeldigheter.

Grafikk og plots!

Det siste vi skal prøve oss på idag er grafikk. Dette er etter min mening noe av det morsomste en kan gjøre i R, og er en veldig god måte å vise frem funnene sine på. For å gjøre dette skal vi bruke ggplot, en pakke vi allerede har lastet inn som en del av tidyverse. Ggplot fungerer på den måten at vi først definerer hvilke variabler vi ønsker å se på, og så hvordan vi ønsker å plotte dem. Det finnes utrolig mange plottyper du kan velge mellom, så her er det bare å google i veg, men akkurat nå skal vi forsøke oss på et linjeplott.

```
ggplot(Norge_9, aes(Time_News, Trust_Politicians)) + #Her skriver jeg først datasettet, og så variablen
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Den blå linjen her viser sammenhengen, som vi jo vet er veldig svak. Det grå område er konfidensintervallet, som vi også ser er veldig stort.

Det var det for denne gangen! Lykke til med oppgavene!