Forkurs for kvantitativ metode

Forkurs til SOS4020

Torbjørn Skardhamar

2023-08-10

Table of contents

Forord

Dette materialet er beregnet på et forkurs for masterstudenter i sosiologi ved universitet i Oslo som skal ta emnet SOS4020 Kvantitative metoder. Forkurset dekker de mest sentrale elementene fra BA-nivået som trengs for å ta SOS4020.

Det anbefales å repetere materiale fra kurs i kvantitative metoder på bachelornivå. Forkurset er en oppfrisker av det aller viktigste materialet fra SOSGEO1120.

De som lært annen statistikksoftware enn R på bachelornivå vil ha særlig nytte av dette kurset. Har man tatt bachelorgrad annet sted enn ved UiO kan man ha lært å bruke Stata, SPSS eller noe slikt. Da trenger du en introduksjon til R, men hva vi skal gjøre vil være tilsvarende.

En av de store forskjellene fra SPSS og Stata er at R **ikke** har muligheten for menybaserte analyser. Du kan altså ikke gjøre analyser med "pek-og-klikk"! Det bør du jo egentlig uansett ikke gjøre i annen software heller. R er altså et programmeringsspråk, og det er viktig å lære å skrive kode både for databehandling og analyse. I tillegg til programmeringsspråket skal du lære å lage en hensiktsmessig mappestruktur og lese inn datasett i R.

Datasettet som skal brukes i SOS4020 er tilgangsbegrenset så det er litt formaliteter som må på plass først. Det skal vi gå på plass her slik at dere har lovlig tilgang til dataene. Men gjennomgangen og eksempler i det følgende vil basere seg på det samme datasett som brukes gjennomgående i SOSGEO1120. Til hvert kapittel er det oppgaver. Først og fremst skal dere kunne gjøre de samme operasjonene på egen datamaskin. Dernest skal dere bruke NorLAG til å gjøre noe mer selvstendige analyser med de samme teknikkene.

Dette heftet inneholder også bittelitt mer informasjon enn du trenger (f.eks. appendiks), men som du kan ha bruk for hvis du skal gjøre mer selvstendige analyser senere.

Målsettinger for forkurset

Overordnet sett skal du denne uken bli kjent med hvordan du bruker R til dataanalyse og repetert grunnleggende statistikk. Du jobber i eget tempo. Bruk tid på det som er krevende og hopp gjerne over deler du synes er enkelt. Så ber du om hjelp når du trenger det. Lærere er tilstede to timer på starten av dagen, og stikker antakeligvis innom en tur på slutten av dagen også. Bruk oss!

Husk at det er ingen eksamen og krav på et slikt forkurs, så det er opp til deg å bruke tiden som er best for deg. Det blir vesentlig enklere å ta kurset SOS4020 hvis du har det grunnleggende rimelig på plass først.

Her er en tentativ plan for hva du bør gjennom disse dagene:

Dag 1

- R og Rstudio er installert riktig på egen maskin.
- Mappestruktur og opprettet Rstudio-project.
- Tilgang til NorLAG og avtale for bruk, regler for datalagring
- Innlesning av data i .rds og .dta format
- Startet med praktisk grafikk

Dag 2

- Grafikk og tabeller.
- Grunnleggende om objekter og dplyr-verb, inkludert pipe-operator.

Dag 3

- Grunnleggende regresjon.
- Sannsynlighet: standardfeil, p-verdier og konfidensintervall

Dag 4

- Sannsynlighet (forts.)
- *Anvende* standardfeil, p-verdier og konfidensintervall sammen med teknikker gjennomgått første dag (tabeller og regresjon)

Part I

Del 1: Kom igang

1 Oppgaver

Man lærer aller best ved å jobbe selv og få hjelp underveis. Dette forkurset er derfor lagt opp til at du skal jobbe på egenhånd, og det blir ingen forelesninger. Dette heftet går gjennom hvordan man gjør en rekke praktiske oppgaver med bruk av softwaren R, og dekker dermed grafikk, deskriptiv statistikk, regresjonsmodeller, og statistisk tolkning. Tilhørende er grunnleggende databehandling som jo også trengs.

1.1 Gjør det på egen datamaskin

Heftet bruker gjennomgående et enkelt datasett *abu89* og viser R-koden illustrerer dermed hvordan ting gjøres. Det første du skal gjøre er dermed å gjøre det samme på egen datamaskin og sjekke at du får samme resultat. Dette kan gjøres som en enkel "klipp-og-lim" som du neppe lærer så veldig mye av hvis du ikke tenker litt samtidig, men du får i hvert fall sjekket at koden fungerer. For hver operasjon bør du gjøre noen endringer i koden og se hva som skjer. I kapittelet om grafikk kan du f.eks. eksperimentere med å bytte om på variable, endre farger og annet. Slik får du en bedre forståelse av hva de ulike funksjonene og argumentene betyr. En god måte å finne ut av hvordan ting fungerer er å gjøre endringer og se hva som skjer.

1.2 Gjør tilsvarende med datasettet NorLAG

Jo mer du tenker aktivt selv, jo mer lærer du. Bruk et annet datasett og gjør tilsvarende operasjoner som er vist med *abu89*. Dere får tilgang til et stort datasett fra undersøkelsen NorLAG og kan da undersøke mange muligheter.

Du kan gjerne prøve å replikere noen tidligere studier som du finner i denne publikasjonslista.^(OBS! Å replikere andres studier nøyaktig er langt mer krevende enn man skulle tro. Normalt trenger du at forfatter deler originalt script, men det er ikke alltid lett tilgjengelig, skrevet i et annet programmeringsspråk, inkluderer langt mer avanserte teknikker enn du har lært om, eller bare er generelt uryddig. Så ikke sett det som ambisjon, men få heller inspirasjon til å finne et tema som er litt interessant og søk opp aktuelle variable.) Variabelliste med dokumentasjon finner du filen "Kodebok.html" når du har fått tilgang til delt mappe .

1.3 Bruk helt andre datasett

Det er en god del innebygde datasett i R og i ulike R-pakker. Du kan få en oversikt over tilgjengelig datasett ved funksjonen `data()` som lister opp de dataene som er tilgjengelig i de pakkene du har lastet for øyeblikket.

- causaldata: en pakke med diverse datasett brukt i lærebøker for kausalanalyse. Tilgang: `install.packages(causaldata)`. For en oversikt, se pakkes dokumentasjon.
- gapminder: en pakke med utdrag av Gapminder-data med ulike lands levealder, befolkningsstørrelse og brutto nasjonalprodukt over mange år. Tilgang: `install.packages(gapminder)`

Når en pakke, f.eks. gapminder, er lastet har du automatisk tilgang til dataene ved å bruke navnet på datasettet i en funksjon som følger:

```
library(gapminder)
summary(gapminder)
```

```
       country        continent        year         lifeExp
 Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60
 Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20
 Algeria    :  12   Asia    :396   Median :1980   Median :60.71
 Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47
 Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
 Australia  :  12                  Max.   :2007   Max.   :82.60
 (Other)    :1632
      pop              gdpPercap
 Min.   :6.001e+04   Min.   :   241.2
 1st Qu.:2.794e+06   1st Qu.:  1202.1
 Median :7.024e+06   Median :  3531.8
 Mean   :2.960e+07   Mean   :  7215.3
 3rd Qu.:1.959e+07   3rd Qu.:  9325.5
 Max.   :1.319e+09   Max.   :113523.1
```

2 Installere R og Rstudio

Vi forutsetter grunnleggende kunnskap til bruk av datamaskiner, og hvis du oppdager at det er tekniske ting du ikke får til forutsetter vi at du lærer deg det. Det går også an å spørres seminarleder om hjelp, men gjør det unna tidlig i semesteret. Her er noe av det vi forutsetter:

Laste ned og installere programmer på datamaskinen Lage mapper og mappestruktur på lokal maskin, og holde oversikt over filer på din datamaskin Laste ned en fil direkte til en mappe uten å åpne, herunder lokalisere download-mappen

OBS! Det er mange csv-filer tilknyttet oppgaver i læreboken. Sørg for å laste ned filene uten at de først åpnes i Excel med en gang. Grunnen er at selv om det stort sett går greit, er Excel tilbøyelig til å tenke litt mye selv og kan finne på å forandre datasettet. (Hvis dette skjer på eksamen er du i trøbbel, så unngå det!).

2.1 Installasjon

Installer nyeste versjon av R herfra: https://cran.uib.no/ Du trenger det som heter «base» når man installerer for første gang. Hvis du har R installert på maskinen din fra før, sørg for at du har siste versjon installert. Siste versjon er 4.1.2. Versjon etter 4.0 bør gå bra, men tidligere versjoner vil kunne gi problemer. Installer nyeste versjon av RStudio (gratisversjon) herfra: https://rstudio.com/products/rstudio/download/ Viktig: du må installere R før du installerer Rstudio for Rstudio finner R på din datamaskin og vil gi feilmelding hvis den ikke finner R. Hvis du har en eldre datamaskin og du får feilmelding ved installasjon av RStudio kan du vurdere å installere forrige versjon av Rstudio herfra: https://www.rstudio.com/products/rstudio/older-versions/

R og Rstudio er to programmer er integrert i hverandre og du åpner heretter R ved å åpne RStudio. Merk: R er navnet på programmeringsspråket og programmet som gjør selve utregningene. Det kjører fra en kommandolinje og er ikke veldig brukervennlig alene. RStudio er et "integrated development environment" (IDE) til R. Det integrerer R med en konsoll, grafikk-vindu og en del andre nyttige ting. Det gjør det lettere å bruke R.

Det finnes også andre IDE for R, men vi skal bruke RStudio gjennomgående på dette kurset. (RStudio inneholder også masse annen funksjonalitet vi ikke trenger til dette kurset).

2.1.1 Spesielt om Windows-maskiner: installer Rtools

Hvis du jobber på en Windows-maskin må du også installere Rtools herfra: https://cran.r-project.org/bin/windows/Rtools/

2.1.2 Spesielt om Mac-maskiner

R skal normalt installere på Mac uten problemer. Noen har fått beskjed om at de også trenger å installere XQuartz eller Xcode. I så fall installerer du de også. Se mer informasjon her: https://cran.r-project.org/bin/macosx/tools/

2.1.3 Spesielt om Linux-maskiner

Har du Linux vet du antakelig hva du driver med. Siste versjon av R og Rstudio kan antakeligvis installeres fra distroens repository.

2.1.4 Spesielt om Chromebook

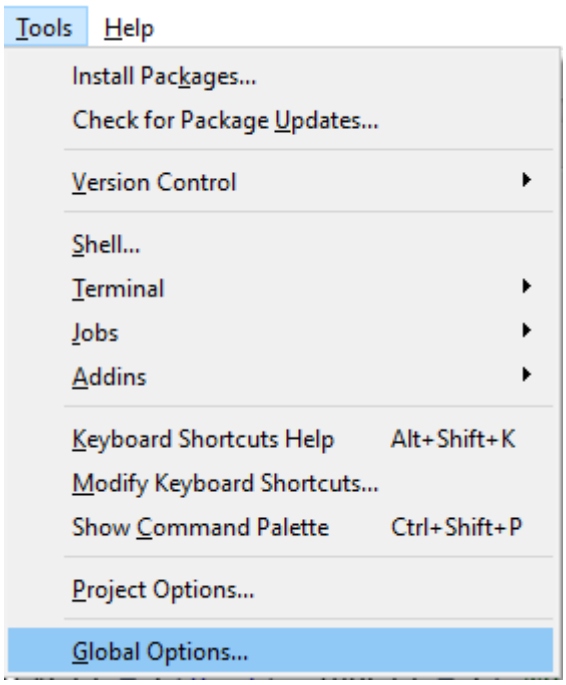
Chromebook kjører et annet operativsystem og R vil ikke uten videre fungere. Derimot kan man på de fleste slike maskiner åpne opp for å kjøre Linux og da kan man installere linux-versjon av R og Rstudio. https://blog.sellorm.com/2018/12/20/installing-r-and-rstudio-on-a-chromebook/ Eller se nedenfor hvordan du kan kjøre R i skyen.

2.2 Oppsett og forberedelser

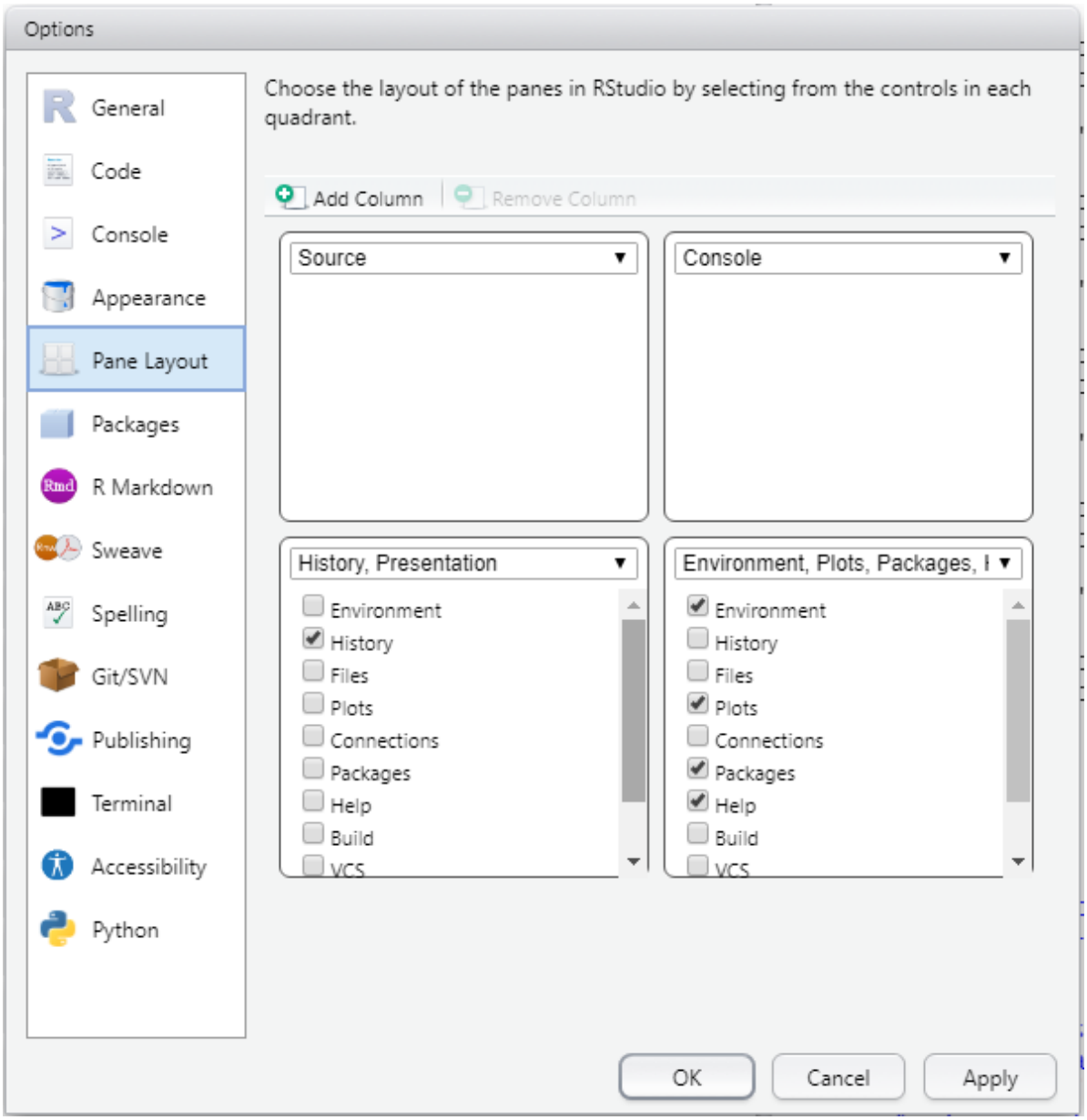
Dette oppsettet gjelder både hvis du har en lokal installasjon og for skyløsninger. Utseendet spiller ingen rolle, og R kan også fungere uten å opprette «projects» som beskrevet her. Men det er lettere å bruke og du har bedre orden hvis du gjør dette.

2.2.1 Utseende i Rstudio

Endre gjerne på oppsettet i RStudio ved å gå til Tools og deretter Global options, så Pane Layout.



Det spiller ingen rolle for funksjonaliteten hvor du har hvilken fane, men her er et forslag.



Dette kan også endres senere og har altså bare med hvordan Rstudio ser ut.

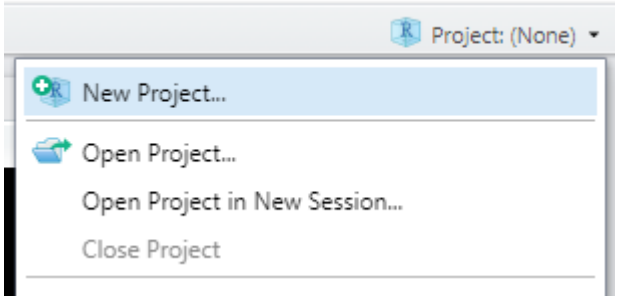
2.3 Rstudio projects

Når du åpner Rstudio skal du alltid åpne som «project» (se video med instruksjon og i R4DS). Arbeidsområdet er da definert og du kan åpne data ved å bruke relative filbaner, dvs. at du

oppgir hvor dataene ligger med utgangspunkt i prosjektmappen. Se kursvideo og instruksjoner i R4DS og gjør følgende:

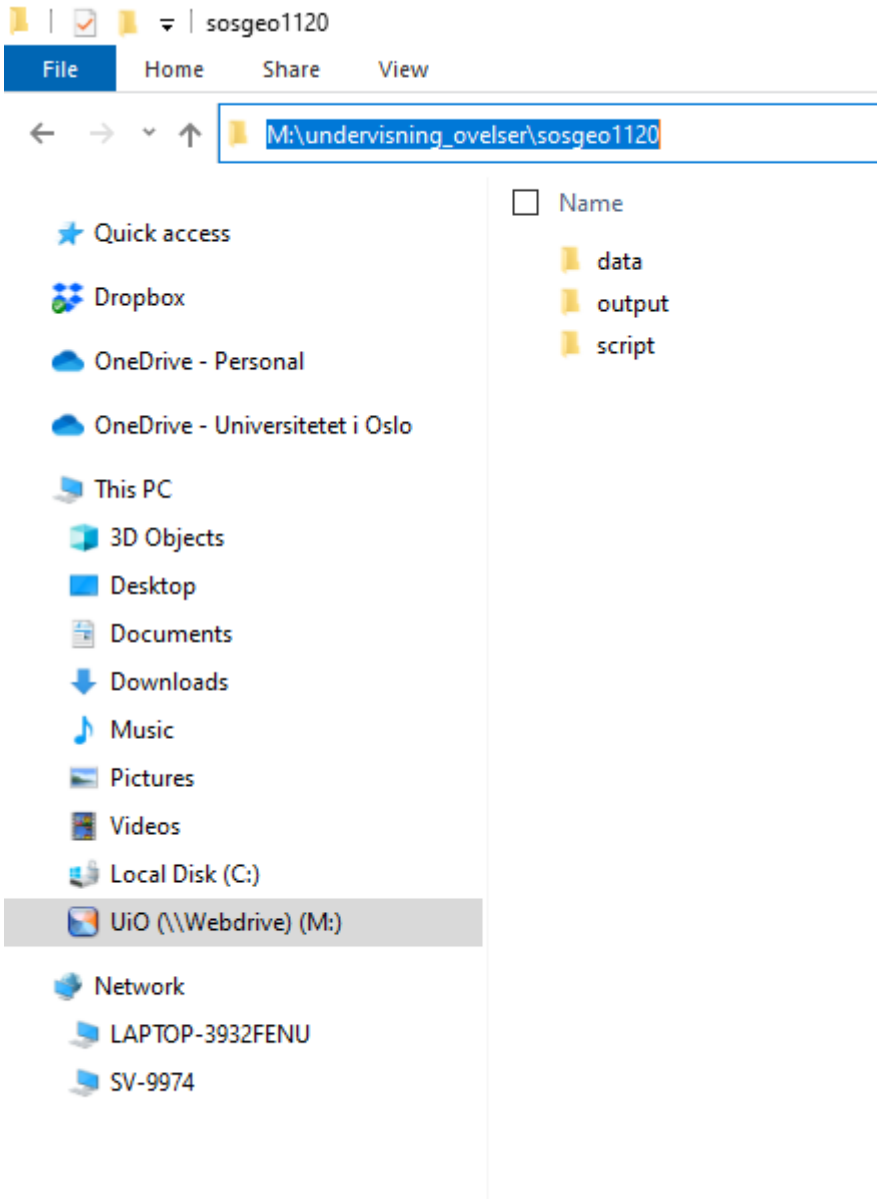
Opprettet mappestruktur med SOSGEO1120 som øverste nivå og egne undermapper for data, script, og output.

2.4 Åpne RStudio og opprett et .Rproject

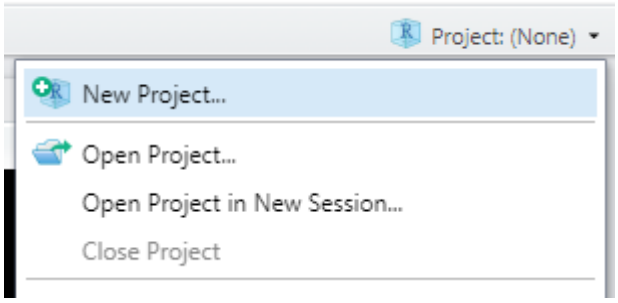


Bruk kommandoen getwd() og se at du har riktig filbane til arbeidsområdet. Hvis du ikke er sikker på hva det betyr, må du spørre noen eller finne det ut på annen måte!

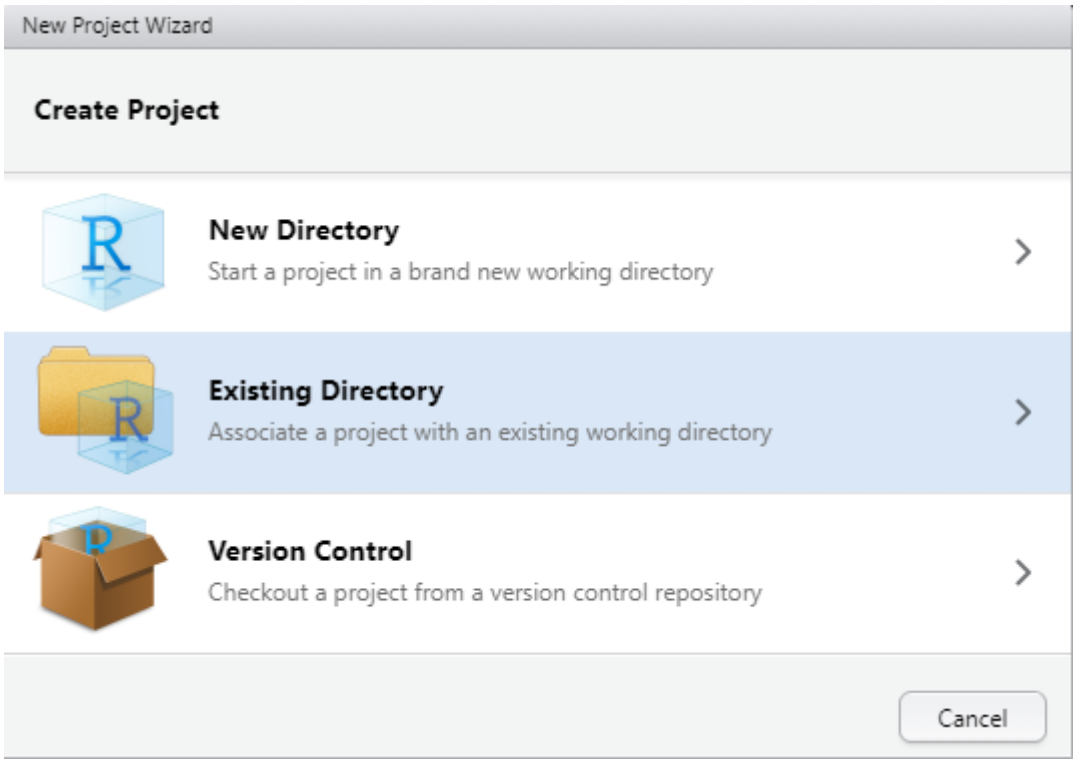
Det første dere må gjøre er å sørge for å ha orden i datasett, script og annet på din egen datamaskin. Å f.eks. lagre alle filer på skrivebordet bør du aldri gjøre, og særlig ikke i dette kurset eller når man jobber med større prosjekter og datasett. For dette kurset skal du ha en mappestruktur med en hovedmappe for SOSGEO1120 og tilhørende undermapper. Det spiller ingen rolle hvor på datamaskinen du legger disse mappene, men du må vite hvor det er. Lag første en mappe som heter SOSGEO1120, og innunder denne mappen lager du tre andre mapper med navnene data, output og script. Du kan ha andre mapper i tillegg ved behov. Det kan se slik ut:



Du skal opprette et Rstudio-prosjekt for hele kurset. Dette er beskrevet nærmere i R4DS i kapittel 6. Når du har åpnet RStudio skal du aller først klikke New Project.

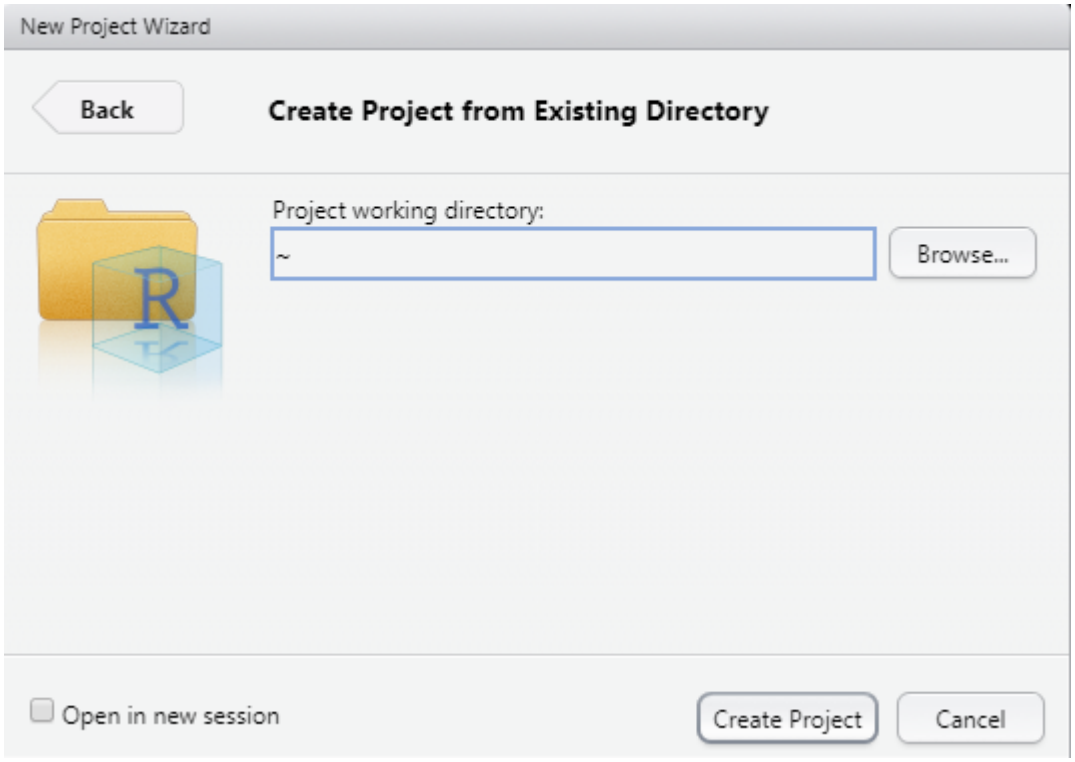


Deretter klikker du du «Existing Directory»

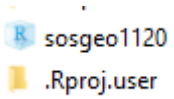


Klikk «Browse» og bla deg så frem til mappen du har laget for SOSGEO1120.

RStudio-prosjektet ligger så i den mappen du har valgt. I filutforsker på datamaskinen vil nå disse to filene dukke opp:



For å starte R videre i dette kurset skal du dobbeltklikke det første ikonet, så vil R åpne seg med riktig arbeidsområde. Mappen .Rproj.user skal du ikke røre. I RStudio vil du se at prosjektet er åpnet ved at det i øvre høyre hjørne er dette ikonet:



En stor fordel med å bruke projects er at du kan flytte hele mappen til et annet sted, eller til en annen datamaskin og alt vil fungere akkurat som før. Hvis du bruker en skytjeneste (OneDrive, Dropbox etc) vil du kunne åpne Rstudio projects på samme måte fra flere maskiner.

3 En veldig kjapp intro til R

Før vi setter igang trengs det en kort introduksjon til noe grunnleggende om hvordan R fungerer. Så lærer man mer underveis, og et senere kapittel går grundigere inn i omkoding av variable.

3.1 Objektorientert

I de innledende kapitlene ble det vist hvordan man leser inn data i R og dataene ble lagt i et "objekt". R er bygd opp rundt å bruke slike objekter i den forstand at alt man jobber med (typisk: datasett) ligger i objekter.

Du kan tenke på objekter som en boks som det står et navn på. Ofte er det bare et datasett oppi boksen, men det kan også være flere ting. Det finnes derfor *flere typer objekter*. Vi skal primært jobbe med datasett, og slike objekter er av typen "data.frame". De kan også være av typen "tibble", men det er for alle praktiske formål på dette nivået akkurat det samme som "data.frame". Men objekter kan også inneholde resultater fra analyser, som f.eks. grafikk, tabeller eller regresjonsresultater. Man kan også legge enkelttall, vektorer og tekststrenger i objekter.

Noen ganger vil et objekt inneholde flere forskjellige ting. Et eksempel er resultat fra regresjonsmodeller som både vil inneholde koeffisienter, standardfeil, residualer, en del statistikker, men også selve datasettet. Men for å se på output er det funksjoner som trekker ut akkurat det vi trenger, så du trenger sjelden forholde deg til hvordan et slikt objekt er bygd opp. Men du kan tenke på det som en velorganisert boks med masse mindre rom oppi.

Men poenget er: Alt du jobber med i R er objekter. Alle objekter har et navn som du velger selv. Du kan legge hva som helst i et objekt. Du kan ikke ha to objekter med samme navn, og hvis du lager et objekt med et navn som eksisterer fra før overskriver du det gamle objektet.

3.2 Funksjoner

Alt man gjør i R gjøres med "funksjoner", og man bruker funksjonene på objekter eller deler av objekter. Funksjonenen har et navn og etterfulgt av en parentes slik som f.eks. `dinfunksjon(...)`. Funksjonen starter og slutter med en parentes. Du kan tenke på funksjoner som en liten maskin der du putter noe inn, og så kommer noe annet ut. Det du

putter inn skal står inni parentes. Det som kommer ut kan du enten legge i et eget objekt eller la det skrives til output-vinduet.

Det du legger inn i funksjonen - altså inni parentesn - kalles "argumenter". Hvert argument har et navn og du skal normalt oppgi i hvert fall hvilket datasett funksjonen skal brukes på. Argumentet for data er nettopp `data =` og så oppgis navnet på det objektet dataene ligger i. En god del slike argumenter har navn som er standardisert på tvers av funksjoner, og `data =` er et eksempel på dette.

I tillegg kan det være en rekke andre argumenter som vi kommer tilbake til i de ulike funksjonene vi bruker. Et poeng er viktig å presisere: argumentene har også en forventet *rekkefølge*. Man kan også oppgi argumentene uten å angi navnet hvis de kommer i riktig rekkefølge. For eksempel vil en funksjon for regresjon ha den forventede rekkefølgen: 1) Spesifisering av utfallsvariabel og forklaringsvariable på en form som heter "formula", deretter og 2) Angitt objektnavnet til dataene. Så kan det være andre argumenter i tillegg. Man kan godt oppgi argumentene i annen rekkefølge, men da er man nødt til å bruke argumentnavnet slik at R forstår hva som er hva.

3.3 R-pakker

Når man installerer R har man svært mye funksjonalitet tilgjengelig uten videre. Dette kalles "base R", altså basic installasjon og funksjonalitet. Men R er i praksis basert på å bruke såkalte "pakker". Dette er funksjoner som utvider R sin funksjonalitet. Så mens "base R" tilbyr infrastrukturen, så er de ulike pakkene laget for spesifikke oppgaver.

R-pakker er et helt økosystem av funksjonalitet som dekker det aller meste du kan finne på å gjøre, fra bittesmå oppgaver, til avansert statistikk og maskinlæring, til hele systemer for dataanalyse. Det finnes mange hundre R-pakker tilgjengelig, og disse ligger på en server som heter CRAN. Hvis du vil se på hva som finnes kan du se på oversikten over tilgjengelige pakker. For nye brukere av R vil dette fremstå som ganske kaotisk. Det finnes også oversikter der viktigste pakker innenfor ulike typer analyse er gruppert slik at man lettere skal kunne finne frem. Dette kalles Task Views. Men du trenger ikke forholde deg til slike oversikter på en god stund ennå. Du får beskjed om hvilke pakker du trenger fortløpende, og det er et begrenset antall.

For å installere en pakke må du vite hva pakken heter og datamaskinen din må være koblet til internett. Funksjonen `install.packages`:

```
install.packages("pakkenavn")
```

Det hender at man får en feilmelding når man prøver installere en pakke. Det er noen veldig vanlige grunner til feilmeldinger som skal være rimelig enkle å finne ut av selv:

1) Du har stavet navnet på pakken feil. Passe på særlig små og store bokstaver.

2) Pakken krever at du har noen andre pakker installert fra før. I så fall vil disse pakkenes navn står i feilmeldingen. Installer disse på samme måte først og prøv igjen.
3) Noen andre pakker trengs å oppdateres for at den nye pakken skal virke. Oppdater alle pakker og prøv på nytt.
4) Din R installasjon må oppdateres. Hvis det er lenge siden du installerte R, så installer på nytt og prøv igjen. Da må alle andre pakker også installeres på nytt.

Når du installerer pakker får du noen ganger spørsmål om du vil installere "from source". Som hovedregel kan du velge nei. "From source" betyr at det finnes en ny versjon som ikke er ferdig kvalitetssjekket på CRAN, men som er tilgjengelig. Du trenger neppe det aller, aller siste av funksjonalitet, så "nei" holder.

Når en pakke er installert på datamaskinen din er disse funksjonalitetene tilgjengelig i R, men ikke helt automatisk. Pakkene ligger i en egen mappe i filstrukturen på datamaskinen og R vet selvsagt hvor dette er. For at pakkene skal være tilgjengelig for deg må du fortelle R at du skal bruke en slik pakke. Vi sier at vi "laster en pakke" (engelsk: "load") og da er disse funksjonene tilgjengelig for deg i hele R-sesjonen. Hvis du restarter R, så må du laste pakkene på nytt før du kan fortsette der du slapp.

Du laster en pakke med funksjonen `library`.

```
library(pakkenavn)
```

Hvis en kode ikke fungerer og du får feilmelding kan dette være grunnen: du har glemt å laste pakken eller pakken er ikke installert på maskinen din.

En annen grunn til at koden ikke fungerer kan være at det er "konflikt" mellom pakker du har lastet. Hvis du bare laster alle pakker du vet du bruker (og noen ekstra som noen på internett har foreslått), så kan det hende at disse pakkene skaper trøbbel for hverandre. Det er typisk at noen funksjoner har samme navn i ulike pakker, og R bruker da en annen funksjon enn du tror. Så da er rådet: ikke last masse pakker du ikke vet hva er. I det etterfølgende introduseres ulike pakker fortløpende og du får da vite hva de brukes til. Utvalget av pakker er dessuten slik at det ikke skal være noen slike konflikter. De pakkene vi skal bruke jobber veldig fint sammen. (Se avsnitt nedenfor om dialekter).

Men det er altså et poeng at du må vite hva slags funksjonalitet de ulike pakkene har, og hvilke du faktisk trenger.

3.4 R-dialekter

De funksjonene som følger med grunnleggende installasjon av R kalles altså "base R" eller bare "base". Dette er grunnstrukturen for programmeringsspråket. Man kan gjøre svært mye analyser med bare bruk av base R, og en god del lærebøker i statistikk og dataanalyse er lagt opp til hovedsakelig bruk av *base*.

Noen R-pakker inneholder ikke bare enkeltfunksjoner, men nesten et helt programmeringsspråk i seg selv. Noen slike pakker er egentlig en hel samling av veldig mange andre pakker som er integrert i hverandre og fungerer sømløst sammen. Det er lurt å holde seg innenfor samme "dialekt" da man ellers kan bli veldig forvirret. I det følgende skal vi holde oss til dialekten "Tidyverse", som er en dominerende variant i R.

Merk at det finnes altså flere dialekter som er spesialiserte for spesifikke formål. Et eksempel er {data.table} som er lynrask for store datasett, {caret} som gir et rammeverk for maskinlæring, og {lattice} som er et eget grafikk-system. Det finnes enda flere. Dette gjør at det kan være vanskelig å søke på nettet etter løsninger fordi du kan få svar (som funker!) i en annen dialekt enn den du kan.

3.5 Tidyverse

Når man laster pakken {tidyverse} laster man egentlig flere pakker som også kan lastes individuelt. Merk at "tidy" betyr jo "ryddig" og hensikten her er et språk som er så ryddig og logisk som mulig. Dette innebærer også at det er innarbeidet en del prinsipper for datastruktur og datahåndtering som hovedarkitekten bak har redegjort for i en egen artikkel. Full oversikt over pakkene som inngår i Tidyverse finner du på deres hjemmeside. Men du trenger ikke sette deg inn i alt det for å bruke softwaren.

3.5.1 Datahåndtering: {dplyr}

Grunnleggende datahåndtering inkluderer først og fremst å *endre variable* ved omkoding, utregninger eller transformasjoner. Pakken {dplyr} inneholder de nødvendige verktøy for dette.

De grunnleggende funksjonene vi bruker kan ordnes sekvensielt og bindes sammen med en "pipe". Norsk oversettelse vil være "rørlegging". Dette er litt rart og uvant, men i første omgang kan du se for deg at det er en flyt av data fra venstre side mot høyre side. Du kan altså gjøre noe med data og "deretter gjøre" noe mer med de dataene du har endret. Vi kommer tilbake til dette nedenfor.

Vi skal bruke et bittelite datasett for å demonstrere. Det er seks observasjoner og to variable. Observasjonene tilhører gruppe a, b, eller c, og variabelen "varA" har en tallverdi. Dataene ser ut som følger:

```
dinedata
```

```
  gruppe varA
1      a    3
2      b    5
```

```
3       b    2
4       a    4
5       c    3
6       c    7
```

3.5.1.1 Grunnleggende verb

For å endre variable brukes funksjonen `mutate`, som har to argumenter: hvilket datasett som skal endres på, og spesifikasjon av gitte variable.

Syntaksen er slik at man *starter* med å angi objektnavnet med dataene, men her skal det *ikke* skrives `data =` av grunner vi kommer tilbake til straks. Deretter skriver man navnet på ny variabel "erlik" utregning av ny verdi. I det følgende lages en ny variabel "varB" som er *2 ganger varA*:

```
mutate(dinedata, varB = 2*varA)
```

```
  gruppe varA varB
1      a    3    6
2      b    5   10
3      b    2    4
4      a    4    8
5      c    3    6
6      c    7   14
```

Man kan også overskrive en eksisterende variabel på samme måte.

Vi kan også velge bort variable med `select`. Merk at det som ble gjort med `mutate` ovenfor ikke er lagt i et nytt objekt, så det er bare printet til konsollen. Objektet "dinedata" er altså *ikke* endret. I følgende kode bruker vi `select` til velge å bare beholde "varA".

```
select(dinedata, varA)
```

```
  varA
1    3
2    5
3    2
4    4
5    3
6    7
```

Vi kan slette variable ved å sette minustegn foran variabelnavnet som følger:

```
select(dinedata, -varA)
```

```
  gruppe
1      a
2      b
3      b
4      a
5      c
6      c
```

3.5.1.2 Pipe %>% med {magrittr}

Vi bruker en "pipe" for å få lettere lesbare koder og slippe å lage mange nye objekter hele tiden. Vi kan binde sammen flere verb i en arbeidsflyt der man kun angir objektnavnet én gang.

```
dinedata %>%
  mutate(varB = 2*varA) %>%
  select(-varA)
```

```
  gruppe varB
1      a    6
2      b   10
3      b    4
4      a    8
5      c    6
6      c   14
```

Operatoren **%>%** betyr "gjør deretter". Kode ovenfor kan dermed skrives i klartekst som følger:

1) start med datasettet *dinedata* og "gjør deretter:"
2) lag en ny variabel med navn *varB* som er 2 ganger verdien av variabelen varA, og "gjør deretter:
3) slett variabel varA

Hvis vi vil legge resultatet i et nytt objekt for å bruke det videre (og det vil vi nesten alltid!) så spesifiseres det med å sette `nyttobjekt <-` helt først som følger:

```
dinedata2 <- dinedata %>%
  mutate(varB = 2*varA) %>%
  select(-varA)
```

3.5.1.3 Logiske operatorer

I mange sammenhenger setter man *hvis*-krav. F.eks. at man skal gi en ny variabel en verdi *hvis* en annen variabel har en bestemt verdig - og en annen verdi hvis ikke. Det kan også gjelde kombinasjoner av variable og verdier. Slike krav er da enten `TRUE` eller `FALSE`.

Her er grunnleggende logiske operatorer.

Uttrykk	Kode
er lik	`==`
er ikke lik	`!=`
og	`&`
eller	`|`
større/mindre enn	`>` eller `<`
større/mindre enn eller er lik	`<=` eller `>=`

For å kode om kategoriske variable trenger vi disse. La oss bruke `mutate` til å gruppere sammen gruppene "a" og "b" ved å gjøre om alle "a" til "b". Da bruker vi funksjonen `ifelse` som har syntaksen: `ifelse(krav, verdi hvis TRUE, verdi hvis FALSE)`. Altså: først kravet, og alle observasjoner som fyller dette kravet får en verdi, mens alle andre får en annen verdi. Her er en kode som sjekker hvem som er i gruppe "a", og gjør alle disse om til "b", og resten beholder verdiene fra variabelen "gruppe".

```
dinedata %>%
  mutate(gruppe2 = ifelse(gruppe == "a", "b", gruppe))
```

```
  gruppe varA gruppe2
1      a    3       b
2      b    5       b
3      b    2       b
4      a    4       b
5      c    3       c
6      c    7       c
```

Logiske krav kan også kombineres med `&` og `|` og også med parenteser for mer kompliserte krav. Her er et eksempel som omkoder basert på verdier på to variable for å lage en tredje variabel:

```
dinedata %>%
  mutate(gruppe2 = ifelse(gruppe == "a" & varA < 5, "a5", "andre"))
```

```
  gruppe varA gruppe2
1      a    3      a5
2      b    5   andre
3      b    2   andre
4      a    4      a5
5      c    3   andre
6      c    7   andre
```

3.5.1.4 Flere verb

Logiske operatorer brukes også til å filtrere dataene, altså å beholde eller slette rader som oppfyller visse krav. Her er en kode som beholder alle observasjoner om *ikke* tilhører gruppe "a":

```
dinedata %>%
  filter(gruppe != "a")
```

```
  gruppe varA
1      b    5
2      b    2
3      c    3
4      c    7
```

`summarise` aggregerer resultater i et datasett. Man må da manuelt oppgi hvordan man ønsker summere opp med funksjoner som `n()`, `sum()` osv. Her er et eksempel som summerer opp med antall observasjoner, og for en variabel regner ut totalsummen for hele datasettet, gjennomsnittet og standardavviket.

```
dinedata %>%
  summarise(antall = n(), totalt = sum(varA), gjennomsnitt = mean(varA), standardavvik = s
```

```
  antall totalt gjennomsnitt standardavvik
1      6     24            4      1.788854
```

Du synes kanskje det virker litt tungvint å lage oppsummeringer på denne måten? Det burde da finnes en egen funksjon som bare spytter ut en standard oppsummering uten å skrive så mye kode! Det gjør det selvsagt, så dette kommer vi tilbake til i *del 2* for deskriptive teknikker.

Man kan også lage oppsummeringer for ulike grupper i datasettet. Funksjonen `group_by` grupperer dataene slik at når man bruker `summarise` etterpå, så blir resultatene per gruppe. Her er samme oppsummering som ovenfor, men gruppert:

```
dinedata %>%
  group_by(gruppe) %>%
  summarise(antall = n(), totalt = sum(varA), gjennomsnitt = mean(varA), standardavvik = s
```

```
# A tibble: 3 x 5
  gruppe antall totalt gjennomsnitt standardavvik
  <chr>   <int>  <dbl>        <dbl>         <dbl>
1 a           2      7          3.5         0.707
2 b           2      7          3.5         2.12
3 c           2     10          5           2.83
```

Merk at når et datasett først er gruppert, så vil alle utregninger fortsette å være gruppert helt til du legger til `... %>% ungroup()`.

Merk at `summarise` gjør at man bare får ut de aggregerte tallene. Noen ganger trenger man å inkludere en aggregert sum i de opprinnelige dataene. Et eksempel er hvis man vil regne ut for hver observasjon om den er over eller under gjennomsnittet i gruppen (eller totalt). Det følgende eksempelet lager nye variable med antall i gruppen og gjennomsnittet, regner avvik fra gjennomsnittet for hver observasjon og så "dummy" for om observasjonen er over gjennomsnittet eller ikke.

```
dinedata %>%
  group_by(gruppe) %>%
  mutate(antall = n(), gjennomsnitt = mean(varA),
         avvik = varA - gjennomsnitt,
         over_snittet = ifelse(avvik > 0, 1, 0))
```

```
# A tibble: 6 x 6
# Groups:   gruppe [3]
  gruppe  varA antall gjennomsnitt avvik over_snittet
  <chr>  <dbl>  <int>        <dbl> <dbl>        <dbl>
1 a          3      2          3.5  -0.5            0
2 b          5      2          3.5   1.5            1
3 b          2      2          3.5  -1.5            0
```

```
4 a          4      2          3.5   0.5            1
5 c          3      2          5    -2              0
6 c          7      2          5     2              1
```

Resultatene kommer ut i samme rekkefølge som de var fra før selv om dataene er gruppert. Noen ganger trenger vi også å sortere dataene med funksjonen `arrange`. Akkurat her kan sortering være greit bare for å få et ryddigere output.

```
dinedata %>%
  group_by(gruppe) %>%
  mutate(antall = n(), gjennomsnitt = mean(varA),
         avvik = varA - gjennomsnitt,
         over_snittet = ifelse(avvik > 0, 1, 0)) %>%
  arrange(gruppe)
```

```
# A tibble: 6 x 6
# Groups:   gruppe [3]
  gruppe  varA antall gjennomsnitt avvik over_snittet
  <chr>  <dbl>  <int>        <dbl> <dbl>        <dbl>
1 a          3      2          3.5  -0.5            0
2 a          4      2          3.5   0.5            1
3 b          5      2          3.5   1.5            1
4 b          2      2          3.5  -1.5            0
5 c          3      2          5    -2              0
6 c          7      2          5     2              1
```

Ovenfor ser du eksempler på at flere funksjoner settes sammen med "pipe", %>%. Man kan sette sammen så mange slike man vil, men det er en fordel å ikke ha så mange at man mister oversikten: da bør du heller dele opp og lage noen nye objekter som mellomtrinn. Merk at i en slik rekke av funksjoner så utføres operasjonene i *rekkefølge*. Hvis du f.eks. lager en ny variabel kan du bruke den til å filtere *etterpå*, men ikke *før* du har laget den.

Her er et eksempel der vi ønsker å få ut den observasjonen i hver gruppe som har høyeste positive avvik fra gjennomsnittet. Da sorteres det først på både gruppe og avvik, men merk at for avviket vil vi ha det sortert fra høyeste verdi til laveste verdi som angis med `desc(avvik)`. (Funksjonen `desc` er forkortelse for "descending", altså synkende). Deretter filtreres det ved å plukke ut den første observasjonen i hver gruppe, og til dette brukes en funksjon som nummererer radene i hver gruppe `row_number()`.

```
dinedata %>%
  group_by(gruppe) %>%
  mutate(antall = n(), gjennomsnitt = mean(varA),
```

```
        avvik = varA - gjennomsnitt) %>%
  arrange(gruppe, desc(avvik)) %>%
  filter(row_number() == 1)
```

```
# A tibble: 3 x 5
# Groups:   gruppe [3]
  gruppe  varA antall gjennomsnitt avvik
  <chr>  <dbl>  <int>        <dbl> <dbl>
1 a          4      2          3.5   0.5
2 b          5      2          3.5   1.5
3 c          7      2          5     2
```

Det er mulig det ovenstående ikke fremstår veldig nyttig. Men poenget er å introdusere noe grunnleggende om hvordan R og *tidyverse* fungerer. Det gjør det lettere å forstå det etterfølgende kapitlene - som er konkret nyttige.

3.5.2 Grafikk: {ggplot2}

"Base R" har en del innebygde funksjoner for å lage grafikk som vi *ikke* dekker her. Grunnen til dette er at vi vektlegger funksjonen fra tidyverse `ggplot`. Det er noen viktige grunner til dette:

1) `ggplot` fungerer det sømløst med arbeidsflyten vi har vist over
2) `ggplot` er en fullstendig *gramatikk* for all slags grafiske fremstillinger av data. Vi ser på det grunnleggende her, men dette kan også brukes til å lage 3D-fremstillinger, kart og animasjoner og mye mer.[1]
3) `ggplot` gir ikke bare funksjonelle plot, men også i professjonelt publiserbar kvalitet. Selv hvis forlag har sære krav til fonter, fargebruk, dimensjoner og formater, så kan det fikses i `ggplot`. Dessuten blir det pent.

Første kapittel om deskriptiv statistikk handler om grafikk og vi går inn i detaljene der etterhvert som det trengs der.

Et viktig moment i `ggplot` er at det er *lagdelt* og hvert lag skilles med `+` på en måte som ligner på "pipe". Man kan så legge på flere lag oppgå det første laget. En vanlig feilmelding i starten er at man bruker `%>%` når det skulle vært `+`.

[1] Dette er i kontrast til annen statistikksoftware som i større grad er basert på enkeltfunksjoner for mer avgrensede typer grafikk.

3.5.3 Import av data: {haven}

R kan importere det aller meste av dataformater, men spesielt for samfunnsvitenskapen er noen formater som primært er brukt i samfunnsvitenskap. Det gjelder Stata, SPSS og SAS. Pakken {haven} er en del av tidyverse og tar seg av dette. Dette er forklart nærmere i kapittelet om import av data.

3.6 Andre nyttige ting

3.6.1 Hjelpfiler / dokumentasjon

Dokumentasjonen i R er ofte ganske vanskelig å lese når man ikke er så god (ennå) i å bruke R. Det tar rett og slett litt tid å bli vant til hvordan ting fungerer. Hjelpfilene er skrevet slik at de er lette å finne frem i for erfarne brukere, men du er kanskje ikke der riktig ennå? Her gis en liten introduksjon for å hjelpe deg til å komme igang. Men ellers vil det meste du trenger å kunne forklares fortløpende etterhvert som funksjonene introduseres. Så foreløpig er rådet å ikke stresse med å finne ut av dette ennå. Men det er greit å vite at de finnes!

Alle R-pakker kommer med egne dokumentasjonsfiler, og det er en slik fil til hver funksjon. Denne åpnes med kommandoen `?` foran navnet på funksjonen. For å se nærmere på funksjonen for å lese inn csv-filer, `read.csv` blir det altså `?read.csv`. Hjelpfilen åpens i en egen fane i Rstudio.

Noen ganger er det flere funksjoner som er varianter av hverandre som står i samme dokumentasjon. F.eks. vil dokumentasjonen for `read.csv` også inneholde `read.table`, `read.delim` og et par andre. De har samme argumenter og struktur, og altså samme dokumentasjon.

Hjelpfiler har en fast struktur. Under overskriften **Usage** står koden med angitte forvalg for funksjonen(e). Hvis man ikke angir annet, så er det disse argumentene som brukes. Det gjør at man ofte ikke behøver å spesifisere så mye kode hver gang. Hvis man ønsker å gjøre noe *annet* må man imidlertid angi de relevante argumentene.

Under overskriften **Arguments** vil det stå spesifisert hva hvert argument gjør, og ofte angitt hvilke verdier som er gyldige å angi.

Under overskriften **Details** vil det gjerne være noe nærmere forklart, gjøre oppmerksom på spesielle utfordringer etc.

Under overskriften **Value** kan det stå noe mer om hva som kommer *ut* av funksjonen. Dette kan være hva slags objekt det blir eller andre ting.

Under overskriften **See Also** vil det være referanser til andre funksjoner som er relevante, enten alternativer eller tilleggsfunksjoner etc.

Overskriften **Examples** er gjerne den mest nyttige. Det er rett og slett noen korte kodesnutter som illustrerer bruken.

3.6.1.1 Vignetter

Alle R-pakker publiseres på en server, CRAN, og hver pakke har sin egen side. Du kan gå inn på denne direkte. Her er lenken til tidyverse på CRAN. Under overskriften **Dokumentation** vil det være en lenke til en referansemanual, som er den samme som når du bruker `?` i R, men her får du alt tilhørende pakken i en samlet pdf-fil.

For mange pakker vil deg også være lenker til "Vignettes". Disse er gjerne mer utførlige tekster som forklarer pakkens struktur og viser bruk. Disse er gjerne de mest nyttige for vanlige brukere. Noen ganger er det egne nettsider for disse pakkene og vignettene. Det er lenket til flere slike i det etterfølgende.

3.6.2 Bruke pakker uten å laste dem

Det hender at man trenger å bruke en funksjon fra en spesifikk pakke én gang og derfor ikke har behov for å laste pakken. Som nevnt ovenfor hender det at funksjoner har samme navn i ulike pakker slik at det finnes en konflikt der R kan komme til å bruke en annen funksjon enn du hadde tenkt. Det burde ikke være et problem i noe av det som dekkes i dette heftet, men kan være greit å vite likevel.

En funksjon fra en spesifikk pakke kan angis med `pakkenavn::funksjon()`. Her er et eksempel der man eksplisitt angir å bruke funksjonen `summarise` fra pakken {dplyr}.

```
dplyr::summarise(dinedata, antall = n(), snitt = mean(varA))
```

```
  antall snitt
1      6     4
```

3.6.3 Addins

Rstudio er det mulig å installere såkalte "addins". Dette gir ikke økt funksjonalitet til R, men til Rstudio. Noen av disse hjelper deg med å skrive kode med bruk av menyer. Dette kan være nyttig for å finne ut av problemer og vanskelig syntaks - men er ikke noe du skal bruke på daglig basis. Det er *hjelp* til å finne ut av ting, så bruk det til å lære!

Addins installeres på samme måte som R-pakker med `install.packages`, men du trenger ikke laste det med `library` for å brukes. I stedet er funksjonene tilgjengelig i Rstudio-menyen "Addins". Det er mulig du må søke opp f

Det er veldig viktig at du bruker slike addins på en måte som gjør at du *lærer deg R* på ordentlig. Du kan ikke belage deg på å bruke addins i det lange løp. De som nevnes nedenfor genererer kode for deg og du bør så lime den koden inn i scriptet ditt!

3.6.3.1 Styler - skriv pent

I R spiller det ingen rolle *hvordan* du skriver kode: linjeskift, innrykk og mellomrom etter parentes osv gir det samme resultatet. (Komma og parenteser er derimot viktig!). Men det er ikke likegyldig for lesbarheten. {styler} kan brukes til å bedre lesbarheten av egen kode. Denne addin'en er laget av de samme som lager tidyverse, og er derfor utmerket verktøy for å skrive bedre kode. "Bedre" er da her i betydningen ryddig og ordentlig, noe som gjør den lettere å lese, de-bugge og at andre forstår koden din.

Du kan se nærmere på vignetten til Styler

3.6.3.2 Esquisse - grafikk

Esquisse kan brukes til å lage grafikk med "drag-and-drop". Noen synes det er lettere i begynnelsen. Men det viktigste med å bruke slike verktøy er at du etterpå kan vise koden slik den lages med `ggplot`.

Du kan se nærmere på vignetten til Esquisse.

3.6.3.3 Questionr - omkode factor

Å omkode factor-variable kan være litt styr. Det er en egen addin for dette formålet.

OBS! Questionr generer kode i base-R. Det er altså *ikke* helt den samme dialekten som ellers er dekket her. Men det er likheter, så det kan være til hjelp likevel.

Du kan se nærmere på vignetten til questionr.

3.6.4 Få hjelp av chatGPT

Det er mye snakk om kunstig intelligens for tiden, og AI er overalt. En av de tingene som AI-verktøy som chatGPT faktisk er god på er å skrive kode i mange språk, deriblant R. Det er imidlertid ingenting ved chatGPT som gjør at du ikke trenger å kunne skrive kode selv.[2] Du må nemlig vite om løsningen gjør det du faktisk ønsker.

Effektiv bruk av chatGPT innebærer at du kan formulere promptet godt. For å få til det bør du derfor vite hva du driver med. Du trenger også kunnskap og erfaring for å se om kodeforslaget

[2]Nei, heller ikke med *code interpreter*.

ser rimelig ut, og vurdere om løsningen bruker riktige pakker. Det er ofte lurt å spesifisere at du vil ha en løsning med tidyverse eller andre spesifikke pakker. Oppfølgingsspørsmål kan også være nødvendig.

Du må *aldri* bruk kode fra chatGPT (eller andre verktøy) uten at du forstår hva koden faktisk gjør. Det kan innebære at du må teste koden på dine data grundig. Det krever også ferdigheter. Du må rett og slett ha et godt grunnlag for å forstå koden. I mellomtiden kan du godt bruke chatGPT til å *lære deg* R.

Hvis du skal bruke chatGPT så bør du starte med å bruke det til å forstå instruksjoner du har problemer med. Prøv ut med følgende prompt:

```
Kan du forklare følgende kode?
  dinedata %>%
  group_by(gruppe) %>%
  mutate(antall = n(), gjennomsnitt = mean(varA),
         avvik = varA - gjennomsnitt) %>%
  arrange(gruppe, desc(avvik)) %>%
  filter(row_number() == 1)
```

Du kan også bruke chatGPT til å finne feil i koden du ikke klarer å finne ut av selv. Du kan legge til en "kontekst" og be chatGPT om å finne feilen som gjør at du ikke får det resultatet du forventer. Prøv ut med følgende prompt:

```
Jeg ønsker å få aggregerte statistikker for et datasett, men det blir ikke riktig. Kan du
  dinedata %>%
  group_by(gruppe) %>%
  mutate(antall = n(), gjennomsnitt = mean(varA),
         avvik = varA - gjennomsnitt) %>%
  arrange(gruppe, desc(avvik)) %>%
  filter(row_number() == 1)
```

Merk at chatGPT da vil foreslå en løsning. Denne er ikke nødvendigvis riktig! Da jeg testet på ovenstående ga chatGPT råd om å bruke funksjonen `reframe` i stedet. Det er ikke riktig, selv om det også er mulig. Funksjonen `summarise` ville være en bedre løsning.

4 Datasettet NorLAG

På SOS4020 skal vi bruke et datasett fra surveyen NorLAG. Vi bruker det også i forkurset slik at du vil spare litt tid og arbeid når SOS4020 starter for alvor. Dette er ikke offentlig tilgjengelige data, så det er en prosedyre for å få tilgang. Her er instruksjonene for dette kurset.

Det viktigste for dette kurset er at dere har et rimelig ryddig datasett slik at dere kan fokusere på andre ting. Det opprinnelige datasettet er levert i Stata-format og det er en del utfordringer med å importere disse dataene bl.a. på grunn av hvordan de har valgt å kode missing og filterverdier og lagt denne typen metadata inn i Stataformatet. I R er det en lite hensiktsmessig måte å jobbe med dataene på, så disse tingene er ryddet opp i her.

4.1 Tilgang og lagring

NorLAG er gule data og har restriksjoner på bruk og lagring. Du er pliktet til å sette deg inn i retningslinjene som du finner på denne siden.

I denne sammenhengen betyr det i praksis at du bør jobbe på UiO-OneDrive. Altså: ikke lagre data på din personlige datamaskin og ikke skytjeneste med en personlig konto. Merk at det er *forskjell* på f.eks. OneDrive gjennom UiO og privat, og privat konto er ikke tillatt for slike data.

For å få tilgang på datasettet må du gjøre følgende:

1) Oppgi din uio-epostadresse i dette nettskjemaet. Emneansvarlig legge deg inn i systemet.
2) Du får tilsendt en lenke fra Sikt med videre instruksjoner om hvordan du signerer en avtale. Les avtalen og signer digitalt.
3) Last ned pdf-versjon av den signerte avtalen og behold den for senere referanse. Du kan også gjøre det senere ved å logge inn på Sikt sine sider for data access.
4) Laster opp den signerte avtalen i dette skjemaet.
5) Emneansvarlig vil dele en mappe med deg i Sharepoint der du kun har lesetilgang. Her ligger tilrettelagte versjoner av datasettet.[1] Kopier alle filene over til en lokal mappe i din UiO-OneDrive (se ovenfor).

[1] Du kan også laste ned et datasett fra plattformen til Sikt, men vi skal kun bruke de tilrettelagte dataene i forkurset. Som du vil lære om i dette kurset er det ikke alltid import av data helt lett.

OBS!! Dere signerer en avtale om bruk av data som er begrenset til å brukes til metodeundervisningen på master i sosiologi ved UiO. Den avtalen har også en begrensning i tid. Les avtalen nøye. Dere har et selvstendig ansvar for å overholde betingelsene, herunder at dataene skal slettes innen angitt dato. Det er fult mulig å bruke disse dataene til senere prosjekter, f.eks. til masteroppgave, men da må det søkes på nytt.[2]

4.1.1 Innlesning av data

Datasettet norlag.rds er altså konvertert til R-formatet rds. Når dette er gjort er du klar for både forkurset og SOS4020.

Nest økt vil omhandle innlesning av data: både rds og andre vanlige formater. For eksemplene her vil det brukes noen forenklede datasett med færre variable.

[2]Hvis dere skriver ryddige script nå, så kan alt dere gjør lett reproduseres senere slik at ny søknad og ny utlevering av data ikke skal medføre merarbeid med data.

5 Innlesning av data

Vi skal bruke følgende pakker i dette kapittelet

```
library(tidyverse)
library(haven)
library(labelled)
```

Data kan være lagret i mange ulike formater, men det er også problemstillinger knyttet til *hvordan* dataene er lagret i et gitt format. Dette handler delvis om hvordan noen har valgt å lagre og distribuere data, ikke bare om dataformatet i seg selv.

Det kan være vanskelig å skille mellom hvorvidt utfordringene du møter skyldes dataformatet, softwaren man bruker eller valg andre har tatt. Det kan være flere av disse, men som hovedregel er problemet at data ofte ikke er distribuert i et universelt format. Permanent lagring og distribusjon av data er krevende, men ikke temaet her.

Uansett: du vil ofte få data i et format som ikke er tilrettelagt verken i eller for R. Å gjøre om data fra et format til et annet kan være en avgjørende oppgave for å få gjort noe som helst.

Dette kan være krøkete og du har virkelig muligheten til å kløne det til skikkelig. For at du skal slippe det gir dette kapittelet en oppskrift for å håndtere slike data slik at du kan jobbe videre med dem i R på en hensiktsmessig måte.

R kan imidlertid håndtere det aller meste av dataformater på en eller annen måte, men vi ser bare på de aller mest vanlige her.

5.1 Generelt om ulike dataformat

5.1.1 rds

Rds-formatet er et format særlig egnet for R.

5.1.2 Laste workspace med `load()`

Filer av typen .Rdat eller .Rdata er egentlig ikke et dataformat, men brukes tidvis for å lagre datafiler. Man kan lagre en eller flere datafiler i samme .Rdat fil på disk.

Du kan også lagre et "speilbilde" av hele ditt workspace på denne måten slik at du kan lukke R og så åpne R senere akkurat på det stedet du var i arbeidet. Det kan være kjekt, men forutsetter at du husker hva du drev med forrige gang. Den klare anbefalingen er derfor å ikke bruke dette rutinemessig.

5.1.3 csv-filer

Såkalte csv-format er ren tekstformat der verdiene i kollonnene har skilletegn. Skilletegnet er nesten alltid komma eller semikolon, men kan i prinsippet være hva som helst. Noen ganger vil slike

5.1.4 Excel

Forbløffende mye data foreligger i Excel-format. Det finnes egne funksjoner for å jobbe direkte med excel-filer. Blant annet pakken `readxl` gir funksjoner til å lese inn denne typen filer. Her er et eksempel.

```
library(readxl)
norlag_xlsx <- read_excel("data/norlag_panel.xlsx")
glimpse(norlag_xlsx)
```

```
Rows: 20,892
Columns: 12
$ ref_nr  <dbl> 5, 5, 10, 10, 10, 12, 12, 15, 15, 18, 18, 22, 23, 23, 25, 27, ~
$ round   <dbl> 1, 2, 3, 2, 1, 3, 1, 1, 2, 3, 2, 1, 3, 1, 1, 3, 2, 1, 2, 1, 3,~
$ ioalder <chr> "68", "72", "59", "49", "44", "61", "47", "58", "63", "67", "5~
$ iolandb <chr> NA, "Norskfødt", NA, "Norskfødt", NA, NA, NA, NA, "Norskfødt",~
$ iokjonn <chr> "Mann", "Mann", "Kvinne", "Kvinne", "Kvinne", "Kvinne", "Kvinn~
$ pa001c  <chr> "Ja", "Ja", "Ja", "Ja", "Nei", "Ja", "Ja", "Nei", "Nei", "Ja",~
$ pa300   <chr> "Partner gjør mest", NA, NA, NA, NA, NA, "IO gjør mest", NA, N~
$ hc230   <chr> "En gang i uken", "En gang i uken", "2-3 ganger i måneden", "E~
$ hc231   <chr> "2-3 ganger i måneden", "2-3 ganger i måneden", NA, "2-3 gange~
$ va207   <chr> "Ganske viktig", "Litt viktig", "Litt viktig", "Ikke viktig", ~
$ hcMCS12 <chr> "59.7766", "60.68044", "58.74768", "60.69717", "55.86777", "53~
$ hcPCS12 <chr> "54.83583", "51.03453", "55.92348", "55.25834", "55.91285", "5~
```

Men Excel-filer kan ha en litt mer komplisert struktur enn dette eksempelet. Data kan ligge i ulike faner i Excel-filen, men det kan da håndteres med å legge til argumentet `sheet = ...`. Hvis excel-arket inneholder mye tekst eller andre ting som gjør at de faktiske dataene kommer litt lengre ned, så kan det spesifiseres hvilket celleområde som det skal leses inn fra ved `range = ...` eller bare hoppe over noen rader med `skip = ...`.

På dette kurset skal vi ikke bruke Excel-filer, men det er stor sannsynlighet for at du vil få bruk for dette senere en gang.

5.1.5 Proprietære format: Stata, SPSS og SAS

5.1.5.1 Stata

```
norlag_dta <- read_stata("data/norlag_panel.dta")
glimpse(norlag_dta)
```

```
Rows: 20,892
Columns: 12
$ ref_nr  <dbl> 5, 5, 10, 10, 10, 12, 12, 15, 15, 18, 18, 22, 23, 23, 25, 27, ~
$ round   <dbl> 1, 2, 3, 2, 1, 3, 1, 1, 2, 3, 2, 1, 3, 1, 1, 3, 2, 1, 2, 1, 3,~
$ ioalder <dbl+lbl> 29, 33, 20, 10,  5, 22,  8, 19, 24, 28, 18, 24, 30, 16, 32~
$ iolandb <dbl+lbl> NA,  1, NA,  1, NA, NA, NA, NA,  1, NA,  1, NA, NA, NA, NA~
$ iokjonn <dbl+lbl> 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2~
$ pa001c  <dbl+lbl> 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1~
$ pa300   <dbl+lbl>  1, NA, NA, NA, NA, NA,  2, NA, NA, NA, NA,  3, NA, NA, NA~
$ hc230   <dbl+lbl> 3, 3, 4, 3, 3, 9, 9, 1, 1, 9, 9, 1, 9, 3, 4, 3, 3, 4, 4, 3~
$ hc231   <dbl+lbl>  4,  4, NA,  4,  3, NA,  9,  2,  1, NA,  9,  2, NA,  4,  5~
$ va207   <dbl+lbl> 2, 3, 3, 4, 3, 5, 5, 3, 4, 5, 5, 1, 5, 3, 3, 2, 3, 3, 4, 4~
$ hcMCS12 <dbl+lbl> 6350, 6647, 5994, 6650, 4891, 4121, 6672, 4736, 3529, 4632~
$ hcPCS12 <dbl+lbl> 7085, 6420, 7258, 7149, 7255, 7544, 7355, 6956, 7368, 7342~
```

Legg merke til at den andre kolonnen her viser hva slags variabeltype det er. `<dbl>` betyr at det er numerisk variabel^(Det finnes flere typer numeriske variable som vi for praktiske analyser sjelden behøver å forholde oss til. `<dbl>` står for *Double* som er et lagringsformat som kan ta svært mange desimaler. Det kan også stå `<num>` som håndterer færre desimaler. Det er også vanlig med `<int>` som står for *Integer*, altså heltall uten desimaler.) På noen variable står det også `<dbl+lbl>` der `lbl` står for *labelled* som betyr at det finnes såkalte labler tilhørende variabelen. *Labler* er vanlig å bruke i programmene Stata og SPSS, men er ikke noe som vanligvis brukes i R. Men R leser det inn og kan håndtere dette helt fint. Men som hovedregel er det bedre å rydde opp slik at dataene blir slik vi vanligvis bruker det i R. Dette er grunnen til at dere får en bearbeidet versjon av NorLAG datasettet!

Neste kapittel er spesielt om NorLAG i formatet `.rds`. Hvordan effektivt lese inn fra Stata til R og gjøre om labler er dekket i et appendiks. De av dere som senere skal jobbe med data levert ut fra Sikt kan ha behov for dette, og da kan dere ta en nærmere titt på appedikset. For dette forkurset og SOS4020 vil dere ikke trenge kunne akkurat det.

5.1.5.2 SPSS og SAS

Andre vanlige dataformater er formater fra statistikkpakkene SPSS og SAS, med filhalene henholdsvis *.sav* og *.sas7bdat*. De leses inn på tilsvarende funksjoner tilpasset disse dataformatene. Her er eksempel for innlesning av SPSS-fil:

```
norlag_sav <- read_spss("data/norlag_panel.sav")
```

Her er eksempel for innlesning av SAS-fil:

```
norlag_sas <- read_sas("data/norlag_panel.sas7bdat")
```

5.1.6 Dataformater for store data

Det finnes en hel rekke andre formater for spesielle formål, derav formater for store data. Med store data mener vi her enten at de er så store at det upraktisk lang tid å lese det inn - eller så store at det ikke er plass i minnet på datamaskinen. Formatene `feather` og `parquet` er varianter av det samme og håndteres med pakken *Arrow*. Det finnes også andre pakker for store data, men *Arrow* er nå den anbefalte. En annen grunn til det er at disse datasettene tillater sømløs bytte mellom programmeringsspråkene R og Python. Men det går laaaagt utenfor formålet med dette forkurset.

For mer spesielle behov går det også an å koble mot databaser som MySQL, Spark, Oracle eller noe helt annet, og en oversikt finnes her.

Eneste du trenger være klar over akkurat nå er at R kan håndtere svært mange forskjellige dataformater og koble mot andre løsninger. Kanskje vil du trenge det en gang - kanskje ikke.

5.2 Oppgaver

Exercise 5.1. Les inn datasettet… i rds-format

Exercise 5.2. Les inn datasettet… i xlsx-format

Exercise 5.3. Noen ganger vil datamaskiner være konfigurert slik at filhalene ikke synes. Det betyr jo ikke at de ikke er der, men du ser ikke umiddelbart hva slags fil det er. Finn ut hvordan du endrer dette på din datamaskin. Prøv å skru det av og på. For å finne det ut, søk på internett med søkestrengen "how to display file extension" eller tilsvarende.

6 Få oversikt over datasettet

Hvis man jobber med større spørreskjemaundersøkelser kan det være svært mange variable i datasettet, kanskje hundrevis, og det er vanskelig å ha en oversikt over datasettet. Bare det å finne riktig variabel kan være en utfordring. NorLAG er et eksempel på et slikt datasett. Det vil normalt følge med et dokumentasjonsnotat eller -rapport med oversikt over alle variable. Ofte til det være mest hensiktsmessig å slå opp i denne, men vi kan også ha behov for å se nærmere på dataene i R. En første ting man bør sjekke er om dataene er lest inn riktig og at det rett og slett ser greit ut.

I det følgende er utgangspunktet at man har lest inn hele datasettet og lagret det i et objekt med navn *norlag*.

6.1 Sjekk om innlesning ble riktig

Det første man bør sjekke er jo om innlesning av datasettet ble riktig. Skjer det noe feil her, så blir selvsagt alt annet feil. Men det er lite som kan gå galt når man leser inn fra datasett. Et unntak er csv-filer som ikke har metadata inkludert.

Funksjonen `class()` gir informasjon om hva slags objekt man har. Altså: etter at man har lest inn dataene og lagt det i et objekt. Her sjekkes objektet *norlag*:

```
class(norlag)
```

```
[1] "data.frame"
```

I dette tilfellet får vi tre beskjeder. Det er en kombinert objekttype av *tibble* og *data.frame*. Mens *data.frame* er standard datasett tilsvarende som et regneark, så er *tibble* en utvidelse med noen ekstra funksjoner som er nyttige for avanserte brukere, men er å regne som en utvidelse av *data.frame*. For vårt formål vil det i praksis være det samme. Et datasett som leses inn i R bør altså være av typen tbl eller data.frame. Data kan også ha andre typer strukturer og da vil `class()` rapportere noe annet.

Når man bruker funksjoner i R, så vil noen ganger resultatet avhenge av hva slags type objekt det er.

For å vite hvor mange rader og kolonner det er i datasettet kan man bruke funksjonen `dim()` slik:

```
dim(norlag)
```

```
[1] 20892  2605
```

Her får vi vite at det er 20892 rader (dvs. observasjoner) og 2605 kollonner (dvs. variable).

6.1.1 Bruke `View()`

Særlig når man er uvant med å jobbe i R vil man kunne ha behov for å *se på dataene* slik man er vant til fra regneark eller software som SPSS eller Stata. En mulighet er å bruke funksjonen `View()` så vil hele datafilen åpnes i eget vindu. Dette er kun egnet for å se på dataene og du kan lukke vinduet uten at det påvirker dataene. Dataene ligger fremdeles i det samme objektet på samme måte som før.

```
View(norlag)
```

Hvis variablene ser ut til å ha forventede variabelnavn og verdier, så er det antakeligvis ok.

Et slikt datasett tar imidlertid stor plass og det er vanligvis mer hensiktsmessige måter å se på dataene på som også gir mer informasjon. I R er det ikke meningen at du skal "sitte og se på dataene" på den måten mens man jobber. Men ta gjerne en titt for å få et bedre inntrykk av hvordan dataene ser ut.

Du kan lukke det vinduet med dataene uten at det har noe å si for dataene, som fremdeles er tilgjengelig i minnet på datamaskinen på samme måte som før.

6.1.2 Bruke `head()`

Funksjonen `head()` skriver de første 6 observasjonenen til konsollen i Rstudio. Det gir et første inntrykk av datasettet med variabelnavn og de første verdiene uten å åpne hele datasettet. Hva som faktisk vises vil avhenge av hvor stor skjerm du har, men R vil bare vise de første variablene etter hva som er plass til på skjermen din. For datasett med mer enn noen få variable er ikke dette veldig nyttig, men noen ganger har man små datasett. Med en liten skjerm kan dette da se omtrent slik ut:

```
head(norlag)
```

```
> head(norlag)
# A tibble: 6 x 434
  ref_nr round iointervjumnd    iointervjuyr ioalder      iolandb
   <dbl> <dbl> <dbl+lbl>        <dbl+lbl>    <dbl+lbl>    <dbl+l>
1      5     1       5            2002           68       1 [Nor~
2      5     2      11            2007           72       1 [Nor~
3      5     3  999999 [Deltok ~ 999999       999999 [Del~ 1 [Nor~
4     10     1       5            2002           44       1 [Nor~
5     10     2       5            2007           49       1 [Nor~
6     10     3       5            2017           59       1 [Nor~
# i 428 more variables: iolandb3 <dbl+lbl>, iosvar <dbl+lbl>,
#   iofodselsyr <dbl>, iokjonn <dbl+lbl>, iopafyll <dbl+lbl>,
#   iodeltagelse <dbl+lbl>, ionorlag1kohort <dbl+lbl>,
#   iododyr <dbl+lbl>, ssIO_sivil <dbl+lbl>, hh001c <dbl+lbl>,
#   hh002 <dbl>, pa001c <dbl+lbl>, pa002c <dbl+lbl>,
#   pa003 <dbl+lbl>, pa005 <dbl+lbl>, pa007 <dbl+lbl>,
#   pa014 <dbl+lbl>, pa022 <dbl+lbl>, pa024 <dbl+lbl>, ...
# i Use `colnames()` to see all variable names
```

Legg merke til at under hvert variabelnavn er det en indikasjon på hva slags variabeltype det er. For eksempel betyr `<dbl>` at det er en numerisk variabel mens `<dbl+lbl>` indikerer at variabelen inneholder *labels*.

Det er lite hensiktsmessig å vise alt i konsollen fordi det rett og slett ikke er plass. Nerst står det derfor angitt at det er flere variable som ikke vises og navnet på de første av disse.

6.1.3 Subset med klammeparenteser

En enkel løsning er å bare se på noen få variable om gangen. Med klammeparentes kan vi angi hvilke radnummer og kolonnenummer vi ønsker se på med følgende syntax: `datasett[rader, kolonner]` der altså komma skiller mellom rader og kolonner. Følgende eksempel viser hvordan man kan bruke `head()` for å vise de første observasjonene i datasettet med bare de første 5 variablene (altså: kollonne nr 1-5).

```
head(norlag[, 1:5])
```

```
  ref_nr          iodeltakelse iododaar iofodselsaar iokjonn
1      5     Deltatt T1 og T2     <NA>         1934    Mann
2      5     Deltatt T1 og T2     <NA>         1934    Mann
3     10 Deltatt T1, T2 og T3     <NA>         1957  Kvinne
4     10 Deltatt T1, T2 og T3     <NA>         1957  Kvinne
5     10 Deltatt T1, T2 og T3     <NA>         1957  Kvinne
6     12     Deltatt T1 og T3     <NA>         1955  Kvinne
```

Vi kan altså også angi både rader og kollonner på denne måten. Her er eksempel som viser første 3 rader og variabelnummer 40 til 44.

```
head(norlag[1:3, 40:44])
```

```
  hc201 hc202 hc203 hc204 hc205
1   176    85    Ja   Nei    Ja
2   176    95    Ja   Nei    Ja
3   168    64    Ja   Nei    Ja
```

Legg merke til at under hvert variabelnavn står det en liten tekst, f.eks. eller <S3: haven_labelled>. Det kan også stå andre ting. dbl betyr at det er en kontinuerlig variabel, mens haven_labelled betyr at det er labler til alle eller noe verdier i variabelen.

Vi skal primært jobbe med data som ikke er "labelled", men du vil noen ganger komme borti dette, spesielt hvis du importerer data fra andre statistikksoftware.

6.1.4 Bruke 'glimpse()'

I tidligere kurs skal dere ha lært å bruke funksjonen `glimpse()`, men også her blir det mest rotete fordi det er så mange variable. Det tar rett og slett veldig stor plass på skjermen.

En variant er å bruke `glimpse()` bare på et utvalg variable på tilsvarende måte. Her er et eksempel, men der vi ser på de 20 første variablene ved bruk av klammeparentes tilsvarende som vist over.

```
glimpse(norlag[, 1:20])
```

```
Rows: 20,892
Columns: 20
$ ref_nr          <dbl> 5, 5, 10, 10, 10, 12, 12, 15, 15, 18, 18, 22, 23, 23, ~
$ iodeltakelse    <fct> "Deltatt T1 og T2", "Deltatt T1 og T2", "Deltatt T1, T~
$ iododaar        <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 2006, NA, ~
$ iofodselsaar    <dbl> 1934, 1934, 1957, 1957, 1957, 1955, 1955, 1944, 1944, ~
$ iokjonn         <fct> Mann, Mann, Kvinne, Kvinne, Kvinne, Kvinne, Kvinne, Kv~
$ ionorlag1kohort <fct> Del av NorLAG1 kohort, Del av NorLAG1 kohort, Del av N~
$ iopafyll        <fct> Ingen påfyll, Ingen påfyll, Ingen påfyll, Ingen påfyll~
$ iovektnorlag2   <fct> 143, 143, 51, 51, 51, NA, NA, 195, 195, 215, 215, NA, ~
$ iovektnorlag3   <fct> NA, NA, 73, 73, 73, 135, 135, NA, NA, 227, 227, NA, 39~
$ pr002c          <fct> 1905, 1905, 1933, 1933, 1933, 1918, 1918, 1918, 1918, ~
$ pr003c          <fct> 1991, 1991, NA, NA, NA, 2004, 2004, 1989, 1989, NA, NA~
```

```
$ pr005c          <fct> 1905, 1905, 1933, 1933, 1933, 1915, 1915, 1909, 1909, ~
$ pr006c          <fct> 1996, 1996, NA, NA, NA, 1996, 1996, 1975, 1975, 1976, ~
$ pr007c          <fct> Grunnskole, Grunnskole, Videregående, Videregående, Vi~
$ pr011c          <fct> Videregående, Videregående, Grunnskole, Grunnskole, Gr~
$ round           <dbl> 1, 2, 3, 2, 1, 3, 1, 1, 2, 3, 2, 1, 3, 1, 1, 3, 2, 1, ~
$ ioalder         <fct> 68, 72, 59, 49, 44, 61, 47, 58, 63, 67, 57, 63, 69, 55~
$ iointervjumnd   <fct> 5, 11, 5, 5, 5, 8, 5, 8, 3, 3, 5, 8, 5, 8, 2, 5, 9, 4,~
$ iointervjuaar   <fct> 2002, 2007, 2017, 2007, 2002, 2017, 2002, 2002, 2007, ~
$ iolandb         <fct> NA, Norskfødt, NA, Norskfødt, NA, NA, NA, NA, Norskfød~
```

I denne output'en er den første kollonnen altså variabelnavnene, deretter er det en kollonne som viser hva slags type variabel det er, og deretter de første observasjonene på hver variabel slik at man får et inntrykk av hvordan det ser ut. `glimpse()` gir altså omtrent samme informasjon som `head()`, men er nok mer hensiktsmessig hvis mange variable.

6.1.5 Undersøke enkeltvariable med `codebook()` fra pakken {memisc}

Noen ganger vil man ha litt mer informasjon om enkeltvariablene. Noen datasett vil komme med labler (omtalt annet sted) eller faktorvariable, som gjør at variablene inneholder både tallverdier og tekst.

Å få ut noe deskriptiv statistikk og se på fordelinger er da gjerne neste steg som vil bli behandlet i de etterfølgende kapitlene.

Man vil klare seg greit med det vi har vist ovenfor. Men det finnes flere måter å gjøre det på. Pakken {memisc} inneholder en rekke funksjoner for å håndtere surveydata, som vi ikke skal gå nærmere inn på her. Men akkurat funksjonen `codebook()` gir litt mer informativt output enn `look_for()`.

For å bruke denne må du installere pakken først. I eksempelet nedenfor er pakken ikke lastet med library(), men angitt pakken direkte med `memisc::` først. Dette kan være nyttig hvis man ikke skal bruke noen andre funksjoner fra denne pakken.

```
memisc::codebook(norlag$iokjonn)
```

```
================================================================================

  norlag$iokjonn 'IOs kjønn'

--------------------------------------------------------------------------------

  Storage mode: integer
```

```
Factor with 2 levels

Levels and labels     N Valid

1 'Mann'          10244  49.0
2 'Kvinne'        10648  51.0
```

Poenget her er altså bare å få en penere output og litt deskriptiv statistikk samtidig.

6.2 Søke i datasettet etter variable

For å se nærmere på en variabel går an å bruke funksjonen `look_for()`, som primært er en søke-funksjon, men det gir også informasjon om variabelen.

```
look_for(norlag, "iokjonn")
```

```
 pos variable label     col_type missing values
 5   iokjonn  IOs kjønn fct      0       Mann
                                         Kvinne
```

I output fremgår det at dette er den 10'ende variabelen, inneholder informasjonen "IOs kjønn", er av typen numerisk med tilhørende labler, og verdiene er 1 = Mann og 2 = Kvinne.

Det går også an å bare få ut variabel-label med funksjonen `var_label()` slik:

```
var_label(norlag$iokjonn)
```

```
[1] "IOs kjønn"
```

For å se labels på *verdiene* bruk **val_labels()**.

```
val_labels(norlag$iokjonn)
```

```
NULL
```

Alle datasett skal komme med en dokumentasjon som sier hva hver variabel inneholder og hvilke verdier som finnes i hver variable, og hva de betyr. Dette leveres gjerne som en separat fil, ganske ofte i pdf eller html format. NSD/Sikt leverer dokumentasjonen for Norlag i html-format. (Ideelt burde det vært i et enkelt maskinlesbart format egnet til å bruke til omkoding og labler for de som ønsker det, men de har valgt en annen løsning).

Du kan søke i dokumentasjonen på samme måte som i andre filer, men det kan være litt knotete. Et godt alternativ er å søke direkte i datasettet. Funksjonen `look_for()` søker både i variabelnavn, verdier og labler. Her er et eksempel for hvordan finne variabler som inneholder ordet "yrkesinntekt". Du kan også søke på kortere eller lengre tekststrenger. (Søker du f.eks. bare på "innt" eller "yrke" så får du opp langt flere variable, så du må kanskje prøve deg litt frem).

```
look_for(norlag, "Yrkesinntekt")
```

Det er to variable som inneholder teksten "yrkesinntekt". Den første variabelen har posisjon 353 i datasettet og har variabelnavnet inwyrkinnt. Den andre variabelen har posisjon 371 og har navnet inpartwyrkinnt. Vi fokuserer på den første.

Merk at når labelen avsluttes med ~ (uttales "tilde") indikerer det at teksten er avkortet i outputvinduet. Du får opp hele teksten ved å bruke `val_label()` slik:

```
var_label(norlag$inwyrkinnt)
```

```
NULL
```

Part II

Del 2: Deskriptive teknikker

7 Grafikk med ggplot

Resten av dette heftet belager seg på å bruke datasettet abu89 som er benyttet i en annen lærebok. Dataene kan lastes ned fra den bokens hjemmeside.

Først må dataene leses inn. Siden dette er data i stata-formatet dta, så brukes importfunksjonen `read_stata()`. Som nevnt tidligere bør variabler av typen labelled gjøres om til factor. Vi sletter også labler som ikke er i faktisk bruk. Disse to tingene gjørs med `as_factor()` og `fct_drop()`, men de er lagt inn i en funksjon som går gjennom alle variable i datasettet, `across()` og sjekker om de er av labelled-typen `where(is.labelled)`. Detaljene her er ikke sentrale for dette kurset: bare se å få lest inn dataene i R.

```
library(haven)
library(tidyverse)

abu89 <- read_stata("data/abu89.dta") %>%
  mutate(across(where(is.labelled), ~as_factor(.)),
         across(where(is.factor), ~fct_drop(.)))

glimpse(abu89)
```

```
Rows: 4,127
Columns: 9
$ io_nr    <dbl> 3, 4, 5, 8, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 2~
$ time89   <dbl> 62.00000, NA, 91.32895, 84.23913, 90.42553, 103.28947, 75.000~
$ ed       <dbl> 0, 1, 3, 5, 3, 1, 1, 7, 3, 9, 0, 9, 1, 3, 9, 3, 0, 3, 1, 3, 3~
$ age      <dbl> 58, 24, 44, 46, 40, 36, 31, 31, 26, 29, 54, 58, 25, 25, 56, 5~
$ female   <dbl> 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1~
$ klasse89 <fct> III Rutinefunksjonærer, VIIa Ufaglærte arbeidere, II Nedre se~
$ promot   <fct> NEI, JA, JA, NEI, NEI, NEI, NEI, NEI, JA, NEI, JA, JA, NEI, J~
$ fexp     <dbl> 1.0, 0.3, 1.9, 0.3, 1.0, 1.2, 0.1, 0.4, 0.2, 0.3, 3.5, 3.1, 0~
$ private  <fct> Public, Private, Private, Public, Private, Public, Public, Pr~
```

7.1 Lagvis grafikk

I R er det mange funksjoner for å lage grafikk. Noen er spesialiserte og knyttet til spesielle analysemetoder og gir deg akkurat det du trenger. Vi skal her bruke et *generelt system* for grafikk som heter `ggplot` som kan brukes til all slags grafikk. Funksjonen *ggplot* er bygget opp som en *gramatikk* for grafisk fremstilling. Det ligger en teori til grunn som er utledet i boken ved omtrent samme navn: The grammar of graphics. Det er mye som kan sies om dette, men det viktige er at grafikken er bygget opp rundt noen bestanddeler. Når du behersker disse kan du fremstille nær sagt hva som helst av kvantitativ informasjon grafisk. Dette er altså et system for grafikk, ikke bare kommandoer for spesifikke typer plot. Vi skal likevel bare se på grunnleggende typer plot her.

Systemet er bygd opp *lagvis*. Det gjelder selve koden, men også hvordan det ser ut visuelt. Man kan utvide plottet med flere lag i samme plot og det legges da oppå hverandre i den rekkefølgen som angis i koden.

For enkle plot som vi skal bruke her angir man i denne rekkefølgen og med en `+` mellom hver del (vanligvis per linje, men linjeskift spiller ingen rolle). Hver del av koden spesifiserer enten *hva* som skal plottes eller *hvordan* det plottes, mens andre deler kan kontrollere utseende på akser, fargeskalaer, støttelinjer eller andre ting som har med layout å gjøre.

1) Angi data og *hva* som skal plottes med `ggplot()`
2) Angi *hvordan* det skal plottes med `geom_*()`
3) Angi andre spesifikasjoner (farger, titler, koordinatsystemer osv)

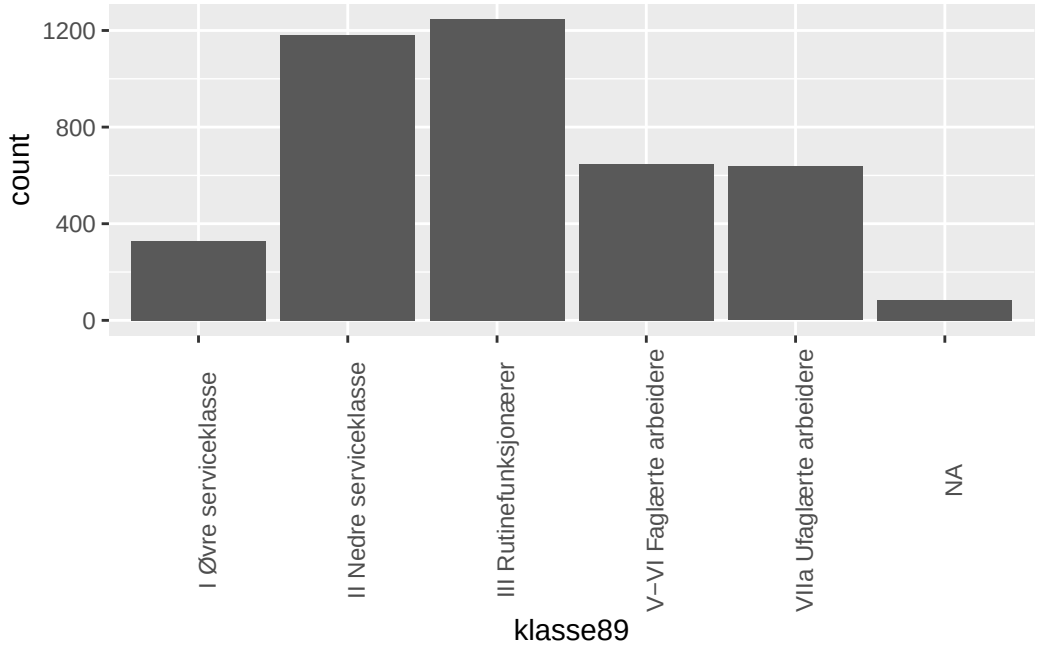
Dette blir tydeligere i eksemplene og forklares underveis.

- Det første argumentet i `ggplot` er data. Altså: hvilket datasett informasjonen hentes fra.
- Inni `ggplot()` må det spesifiseres `aes()`, "*aestethics*", som er *hvilke variable* som skal plottes. Først og fremst hva som skal på x-akse og y-akse (og evt. z-akse), men også spesifikasjon av om linjer (farge, linjetype) og fyllfarger, skal angis etter en annen variabel.
- `geom_*` står for *geometric* og sier noe om *hvordan* data skal se ut. Det kan være punkter, histogram, stolper, linjer osv.
- `coord_*` definerer koordinatsystemet. Stort sett blir dette bestemt av variablene. Men du kan også snu grafen eller definere sirkulært koordinatsystem, eller andre enklere ting.
- `facet_*` definerer hvordan du vil dele opp grafikken i undergrupper

7.2 Kategoriske variabel

7.2.1 Stolpediagram

```
library(ggforce)
ggplot(abu89, aes(x = klasse89)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90))
```

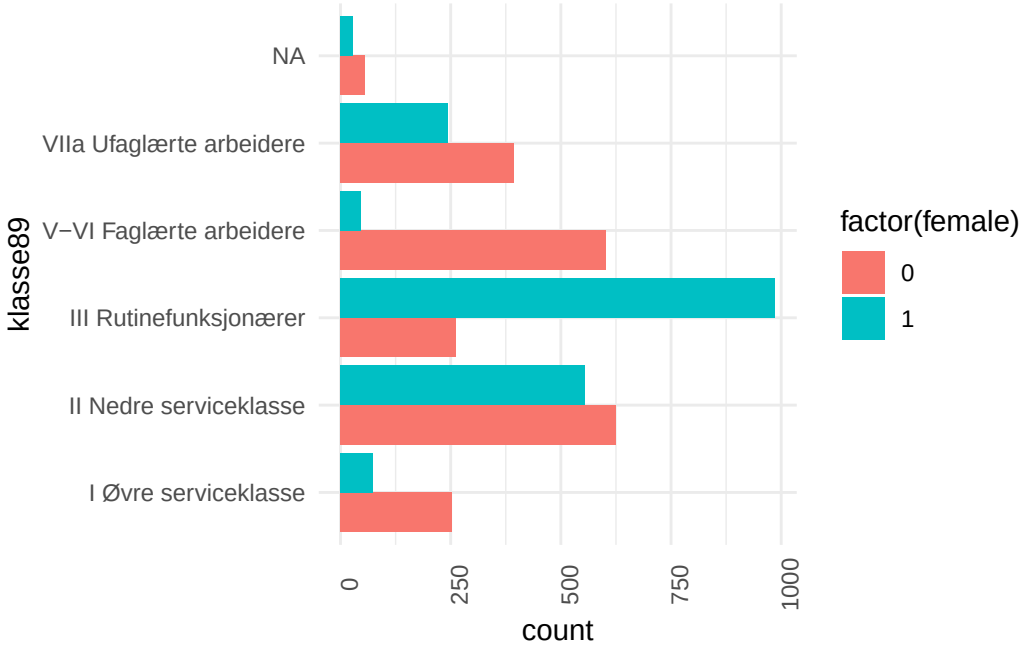


Noen ganger ønsker man å vise fordelingen for to ulike grupper, la oss si for kjønn. En mulighet er da å rett og slett lage to stolpediagram ved siden av hverandre. Til dette kan man spesifisere at dataene er gruppert etter variabelen *female* og at fyllfargen skal settes etter denne variablen med `fill = factor(female)`. Merk bruken av `factor(female)` fordi variabelen er *numerisk* og det vil da ellers brukes en kontinuerlig fargeskale, mens å gjøre om variabelen til kategorisk brukes en annen fargeskala.

I tillegg gjør vi her to ting til: setter et annet grafisk tema med `theme_minimal()` og snur plotvinduet slik at kategoriene er litt lettere å lese. Dette er smak og behag.

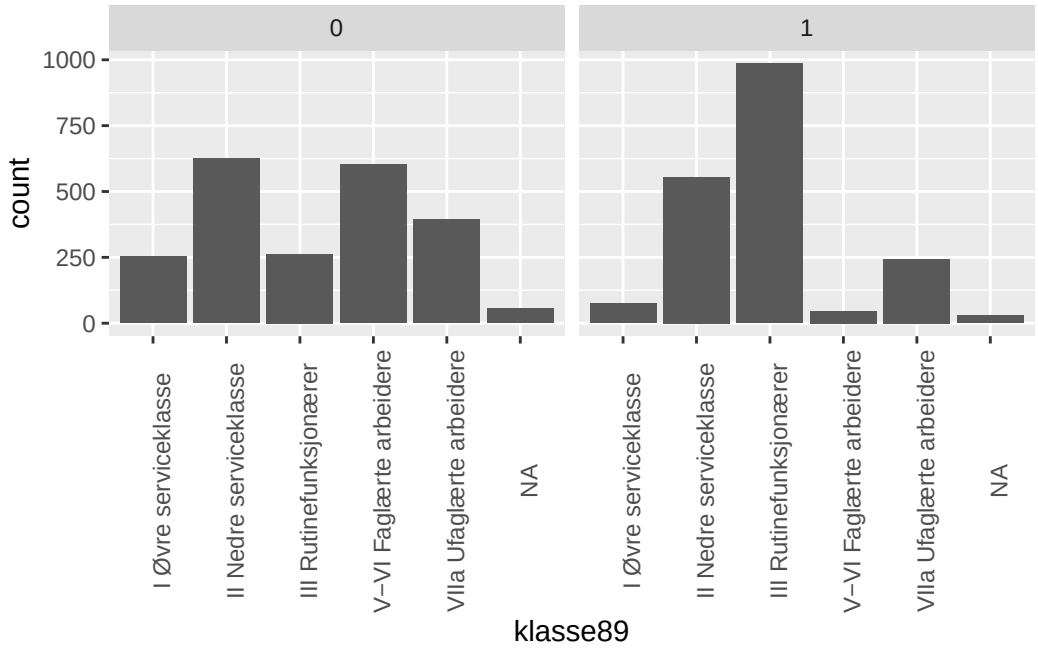
```
ggplot(abu89, aes(x = klasse89, group = female, fill = factor(female))) +
  geom_bar(position="dodge") +
```

```
  theme_minimal()+
  theme(axis.text.x = element_text(angle = 90))+
  coord_flip()
```



Et alternativ er å plassere grafikken for menn og kvinner ved siden av hverandre. Å legge til `facet_wrap()` gjør dette.

```
ggplot(abu89, aes(x = klasse89)) +
  geom_bar() +
  facet_wrap(~factor(female)) +
  theme(axis.text.x = element_text(angle = 90))
```



Et automatisk forvalg for `geom_bar()` er hvordan gruppene plasseres som er `position="stack"`. Det betyr at gruppene stables oppå hverandre. Dette er godt egnet hvis poenget er å vise hvor mange av hvert kjønn som er i hver gruppe. Det er mindre egnet hvis du ønsker å sammenligne menn og kvinner. Da er alternativet å velge `position="dodge"` som følger:

7.2.2 Kakediagram

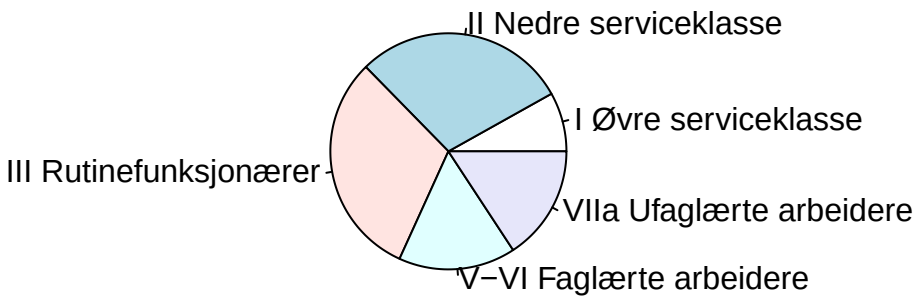
Generelt er ikke kakediagram å anbefale da korrekt tolkning involverer å tolke et areal som inneholder vinkel. Med få kategorier som er rimelig forskjellig kan det gi et ok inntrykk, men ofte ender man opp med å måtte skrive på tallene likevel. Vi tar det med her egentlig bare fordi mange insisterer på å bruke det. Så vet du at det er mulig.

Det enkleste er å bruke funksjonen `pie()` som gir følgende resultat.

```
tab <- table(abu89$klasse89)
tab
```

```
   I Øvre serviceklasse   II Nedre serviceklasse   III Rutinefunksjonærer
                    328                     1181                     1248
V-VI Faglærte arbeidere VIIa Ufaglærte arbeidere
                    648                      637
```

```r
pie(tab)
```

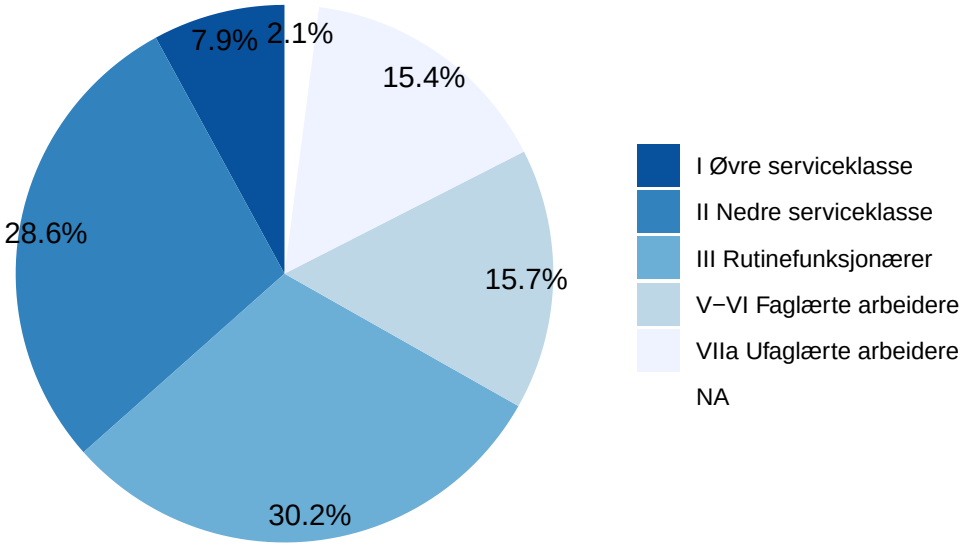


Men hvis man skal bruke ggplot er det litt mer jobb. Fordelen med ggplot er at du har bedre kontroll for å lage publiserbar kvalitet. (Akkurat for kakediagram er det kanskje ikke så farlige, for du bør ikke bruke det i publikasjoner hvis du kan la være).

```r
pc <- abu89 %>%
  group_by(klasse89) %>%
  summarise(n = n()) %>%
  mutate(pct = n/sum(n)*100) %>%
  ungroup()

ggplot(pc, aes(x = "", y = pct, fill = (klasse89))) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0) +
  theme_void()+
  geom_text( aes(label = paste0( round(pct,1), "%"), x = 1.4),
             position = position_stack(vjust=.5), check_overlap = F) +
  labs(x = NULL, y = NULL, fill = NULL)+
  theme(axis.line = element_blank(),
          axis.text = element_blank(),
```

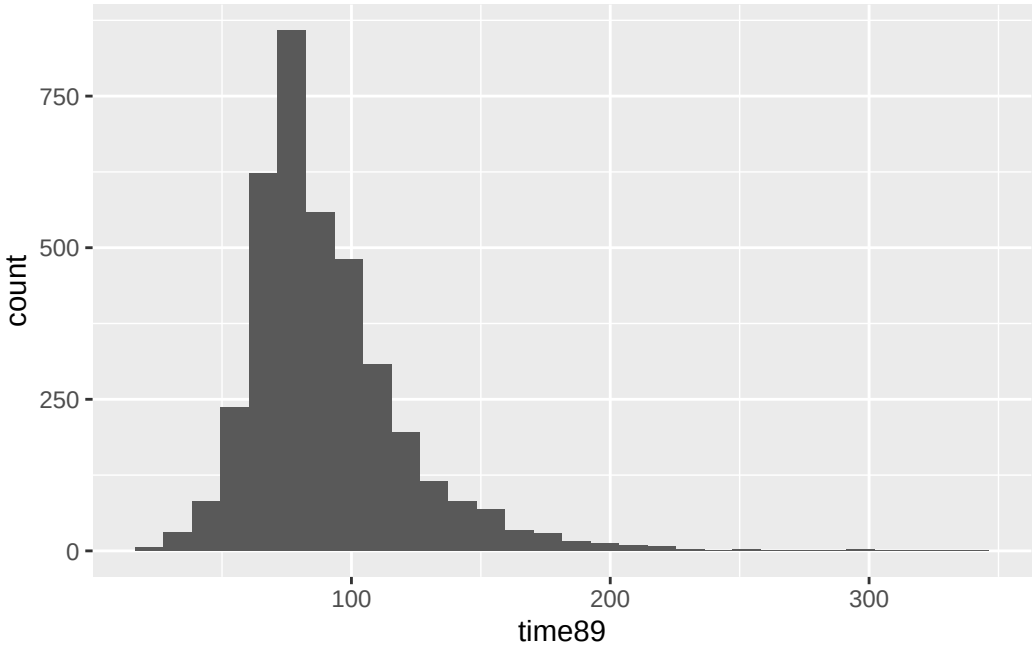
```
        axis.ticks = element_blank()) +
  scale_fill_brewer(palette="Blues", direction = -1)
```



7.3 Kontinuerlige variable

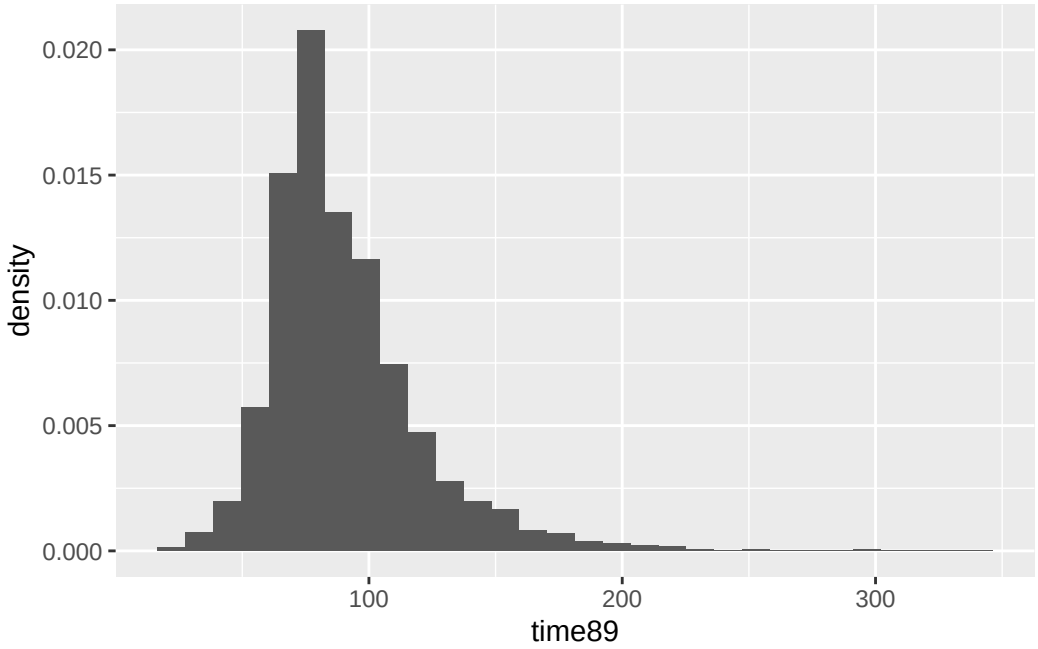
7.3.1 Histogram

```
ggplot(abu89, aes(x = time89)) +
  geom_histogram()
```



Det er også vanlig å fremstille det samme på en "tetthetsskala", der arealet summeres til 1. Det betyr at arealet for hvert intervall tilsvarer en andel. Visuelt sett er det vel så mye arealet vi oppfatter som høyden på stolpene. Men det er bare skalaen på y-aksen som har endret seg. Visuelt sett, ser histogrammene helt like ut.

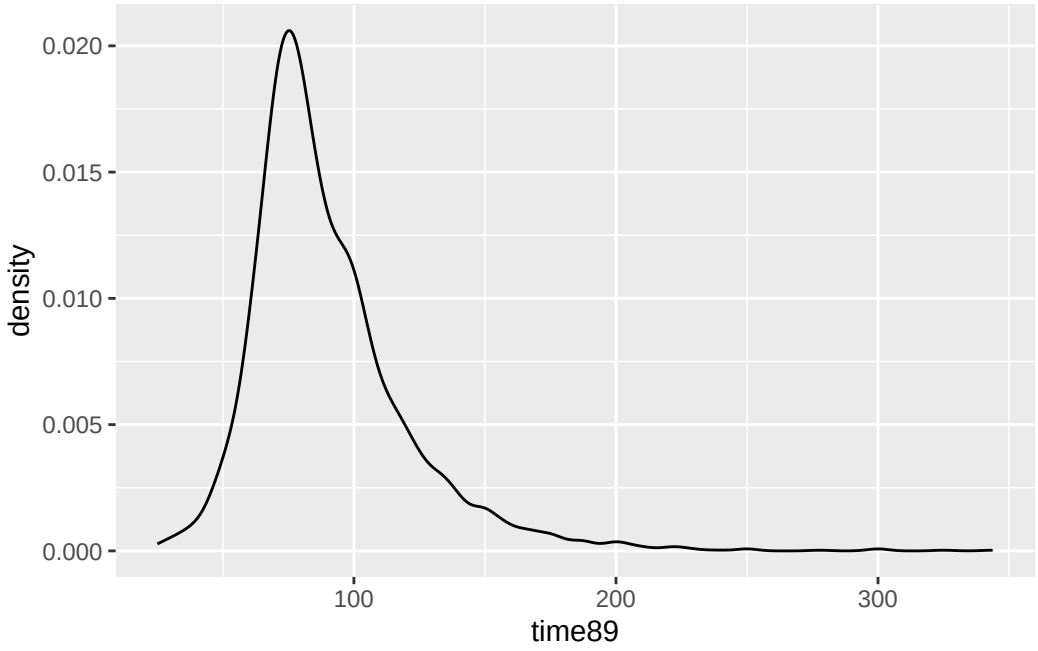
```
ggplot(abu89, aes(x = time89, y = ..density..)) +
  geom_histogram()
```



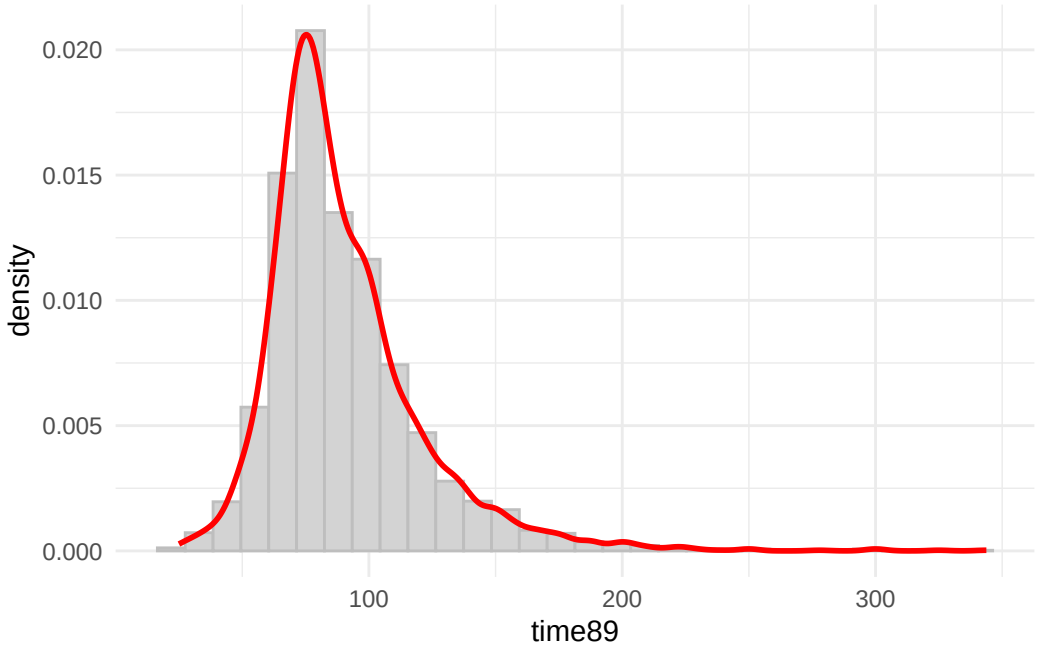
7.3.2 Density plot

Density plot er en måte å fremstille det samme på, men i stedet for å dele inn i intervaller som i histogram lager vi en glattet kurve. Det blir på skalaen "tetthet" som i histogrammet ovenfor.

```
ggplot(abu89, aes(x = time89)) +
  geom_density()
```

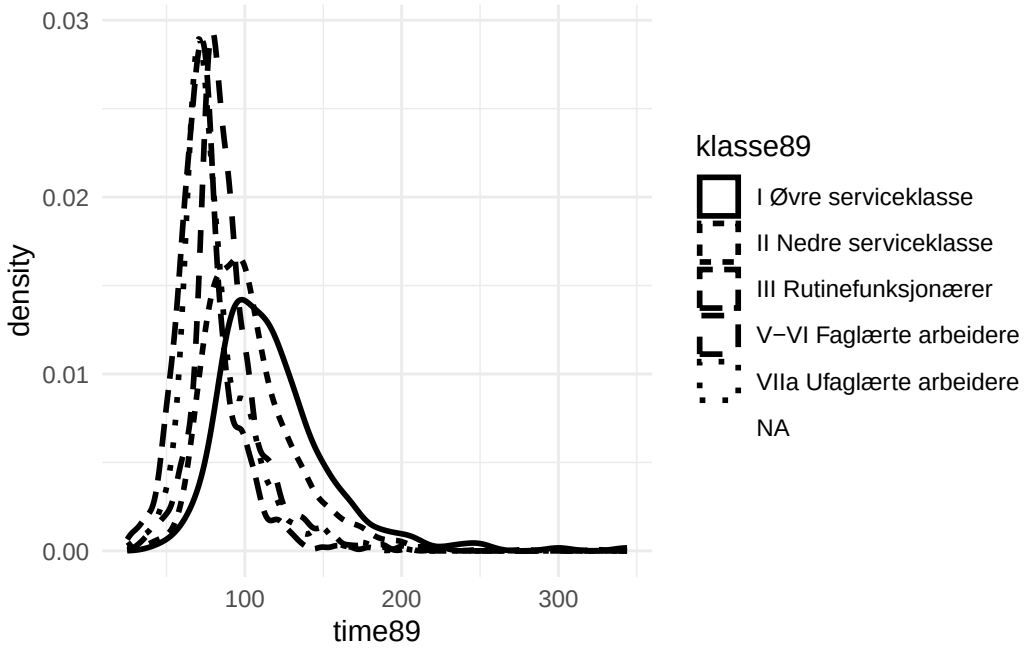



```
ggplot(abu89, aes(x = time89)) +
  geom_histogram(aes(y = ..density..), fill = "lightgrey", col = "grey") +
  geom_density(col = "red", linewidth = 1) +
  theme_minimal()
```

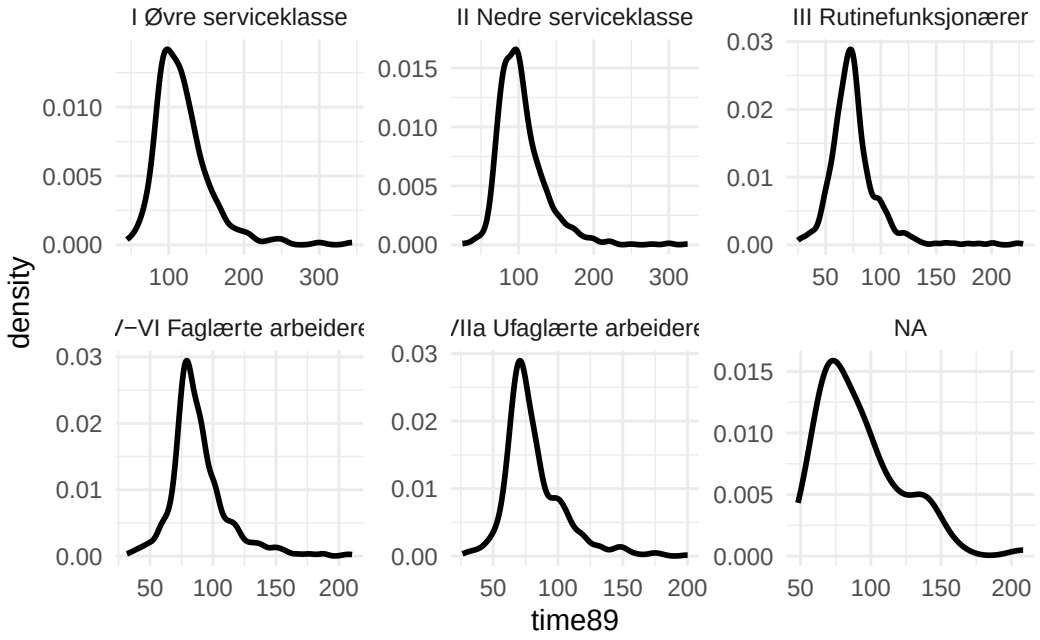


En fordel med denne fremstillingen er at det er lettere å sammenligne grupper. Her er et eksempel med density plot etter hvor mye man drikker.

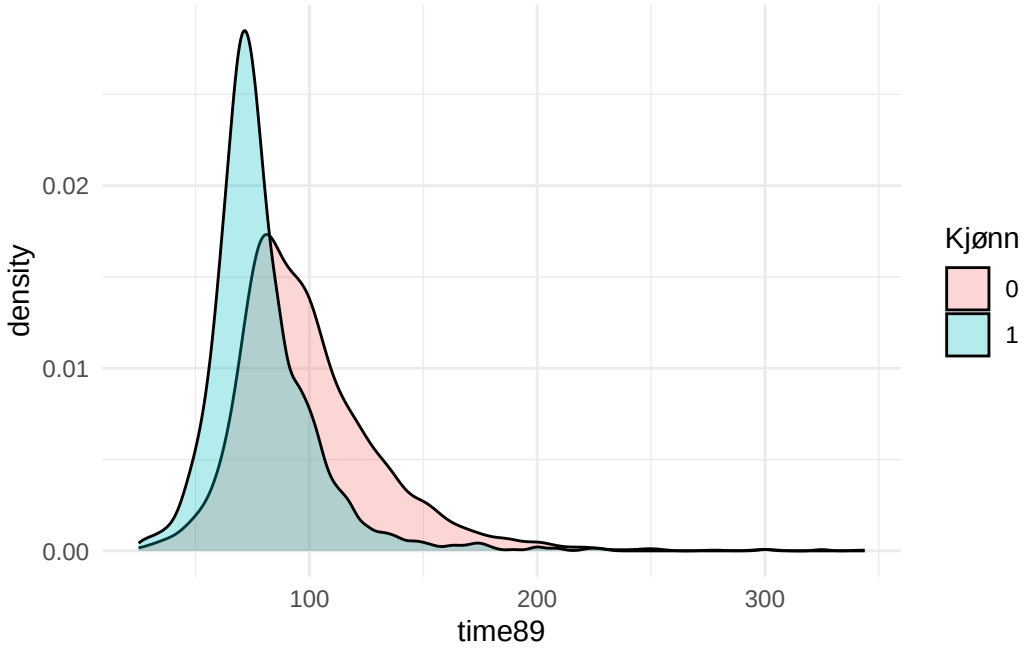
```
ggplot(abu89, aes(x = time89, group = klasse89, linetype = klasse89)) +
  geom_density(linewidth = 1)+
  guides(fill = guide_legend(override.aes = list(shape = 1 ) ) ) +
  theme_minimal()
```




```
ggplot(abu89, aes(x = time89)) +
  geom_density(linewidth = 1)+
  theme_minimal()+
  facet_wrap(~klasse89, scales="free")
```



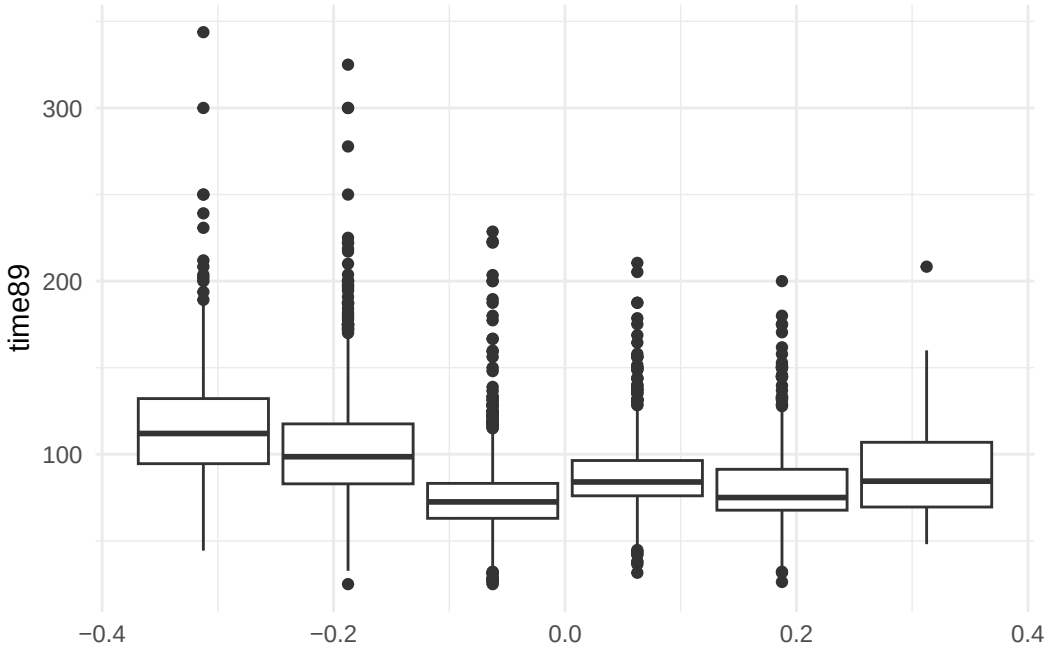

```
ggplot(abu89, aes(x = time89, group = female,  fill = factor(female))) +
  geom_density(alpha = .3)+
  guides(fill=guide_legend(title="Kjønn"))+
  theme_minimal()
```



7.3.3 Flere variable samtidig

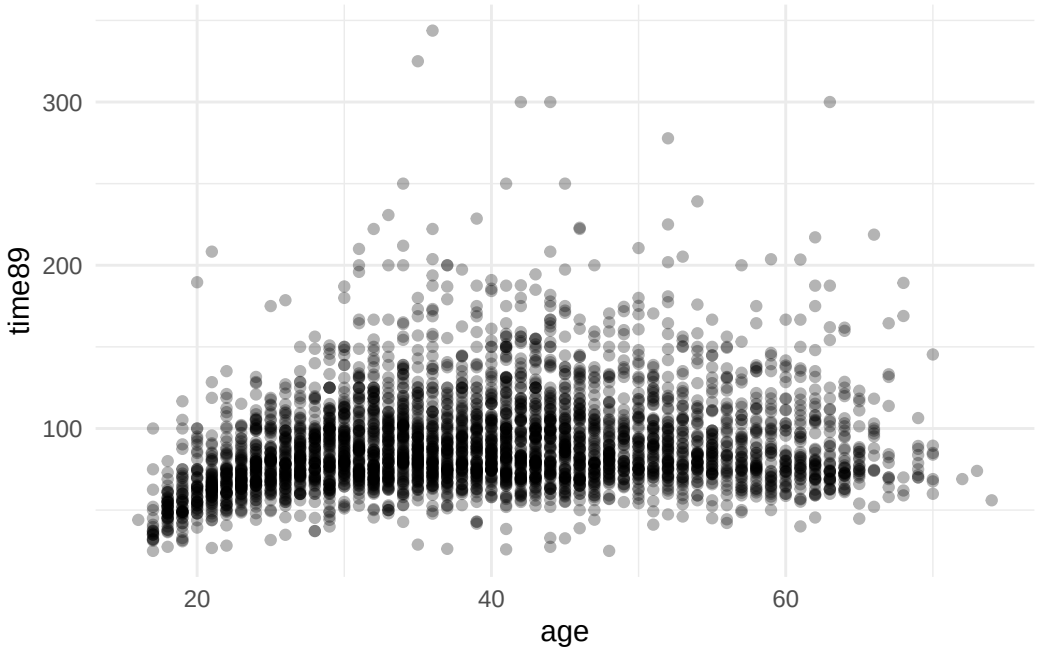
7.3.3.1 Boksplot

```
ggplot(abu89, aes(y = time89, group = klasse89)) +
  geom_boxplot()+
  theme_minimal()
```

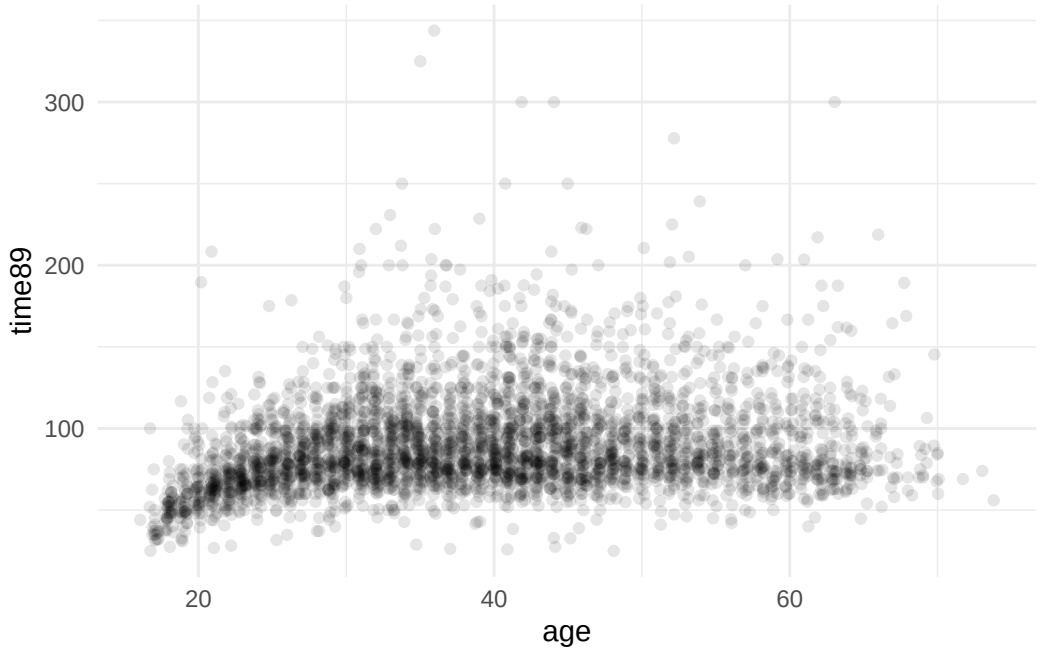


7.3.3.2 Scatterplot

```
ggplot(abu89, aes(x = age, y = time89)) +
  geom_point(alpha=.3)+
  theme_minimal()
```



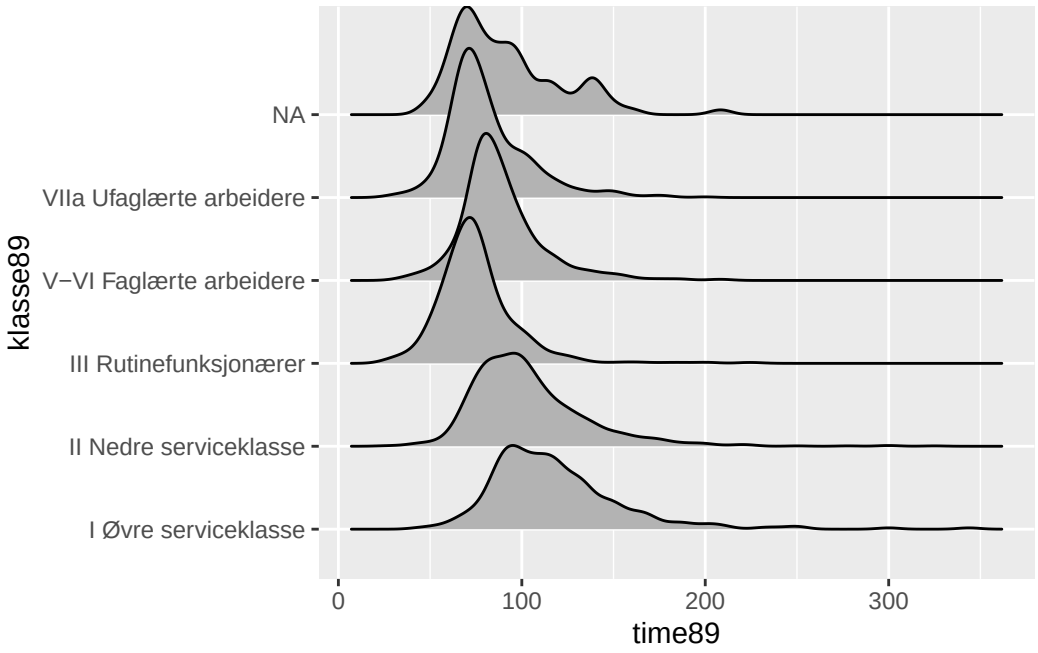

```
ggplot(abu89, aes(x = age, y = time89)) +
  geom_jitter(alpha=.1, width = .3)+
  theme_minimal()
```



7.3.3.3 Ridgeplot

Ridgeplot er en annen måte å sammenligne en kontinuerlig fordeling betinget på en gruppering.

```
library(ggridges)
ggplot( abu89,  aes(y = klasse89, x = time89)) +
  geom_density_ridges()
```



7.4 Oppgaver

Slå opp i boken R for data science hvis du står fast eller ikke skjønner hva koden betyr.

Exercise 7.1. Last ned datasettet abu89 fra angitt hjemmeside og les inn dataene til R som vist ovenfor. Lag den samme grafikken som vist her, gjør noen endringer på kodene for å endre utseendet på plottene. Det er et mål at du skal forstå hva hver enkelt kommando gjør.

Exercise 7.2. Last inn datasettet NorLAG i R. Velg noen variable som du selv tenker kan være informative å se nærmere på. Bruk de samme teknikkene på disse variablene.

8 Deskriptive tabeller

```
library(tidyverse)
library(gtsummary)
```

Det kan være ulike grunner til å lage deskriptiv statistikk, og hva du skal bruke tabellene til kan ha betydning for hvordan du lager dem. Noen ganger skal du bare sjekke noen tall, og da er det ingen grunn til å bruke tid på å gjøre tabellen spesiell pen. Andre ganger skal tabellen publiseres i en rapport, på en nettside eller i en vitenskapelig artikkel - eller mest aktuelt på kort sikt: i en masteroppgave. Da må tabellene se ordentlige ut. Nedenfor skal vi se på begge mulighetene.

8.1 Quick-and-dirty oppsummeringer

Først og fremst har vi funksjonen `summary()`. Når denne brukes på et objekt vil hva slags output du får avhenge av objekttypen. Derfor vil `summary()` gi forskjellig output om det er en vektor, et datasett eller et regresjonsobjekt etc. Vi avgrenser oss til datasett her.

Her er output for hele datasettet.

```
summary(abu89)
```

```
    io_nr          time89             ed             age       
 Min.   :   3   Min.   : 25.00   Min.   : 0.00   Min.   :16.00  
 1st Qu.:1542   1st Qu.: 71.00   1st Qu.: 1.00   1st Qu.:30.00  
 Median :3093   Median : 83.33   Median : 3.00   Median :39.00  
 Mean   :3105   Mean   : 90.15   Mean   : 2.69   Mean   :39.65  
 3rd Qu.:4644   3rd Qu.:102.56   3rd Qu.: 3.00   3rd Qu.:48.00  
 Max.   :6258   Max.   :343.75   Max.   :11.00   Max.   :74.00  
                NA's   :368                                     
    female                          klasse89    promot          fexp       
 Min.   :0.0000   I Øvre serviceklasse    : 328   NEI:2568   Min.   :0.0000  
 1st Qu.:0.0000   II Nedre serviceklasse  :1181   JA :1559   1st Qu.:0.2000  
 Median :0.0000   III Rutinefunksjonærer  :1248              Median :0.7000  
```

```
Mean   :0.4686   V-VI Faglærte arbeidere : 648             Mean   :0.9451
3rd Qu.:1.0000   VIIa Ufaglærte arbeidere: 637             3rd Qu.:1.4000
Max.   :1.0000   NA's                    :  85             Max.   :4.9000

   private
Public :1602
Private:2525
```

Merk at `summary()` rapporterer forskjellig basert på om variabelen er kontinuerlig eller kategorisk. For kontinuerlige variable gis min/max, kvartiler, median og gjennomsnitt. For kategoriske variable gis det antall i hver kategori. Hvis det er manglende verdier på en variabel står det oppført nederst som antall `NA's`.

Merk her at variabelen female er definert som kontinuerlig selv om det bare er to verdier. Det ville være mer hensiktsmessig å gjøre om denne variabelen til kategorisk.

Man kan også bruke `summary()` på enkeltvariable med bruk av `$` som følger:

```
summary(abu89$time89)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  25.00   71.00   83.33   90.15  102.56  343.75     368
```

Da får man altså bare tallene for den variabelen man har angitt etter dollartegnet.

8.1.1 Enkeltfunksjoner

Man kan hente ut hvert av disse tallene spesifikt fremfor å bruke `summary()`. Det er egne funksjoner for dette, og de kan også brukes når man gjør databearbeiding for litt andre formål. Vi ser her på de viktigste.

Hva om man vil ha en kvartil som ikke er oppgitt i forvalget? Da kan man bruke funksjonen `quantile()`. Argumentene i denne funksjonen er hvilken variabel og hvilket prosentil. Som vi ovenfor inneholder time89 noen NA. Vi må i tillegg bestemme hva vi ønsker å gjøre med NA i beregningen, og vi vil her se bort fra disse ved å angi `na.rm = TRUE`. Ellers får man feilmelding.

Her er eksempel med første kvartil som skal gi samme svar som ovenfor:

```r
quantile(abu89$time89, .25, na.rm = TRUE)
```

```
25%
 71
```

Her er en variant der man ber om 95-prosentilen:

```r
quantile(abu89$time89, .95, na.rm = TRUE)
```

```
     95%
148.0362
```

Man kan også be om flere prosentiler. Da listes disse opp innenfor en `c()` som følger. Her gis prosentilene for 5, 10, 90 og 95 prosent.

```r
quantile(abu89$time89, c(.05, .10, .90, .95), na.rm = TRUE)
```

```
       5%       10%       90%       95%
 54.91651  61.00000 127.77778 148.03618
```

Gjennomsnittet av en variabel gis ved funksjonen `mean()`:

```r
mean(abu89$time89, na.rm = TRUE)
```

```
[1] 90.14948
```

Standardavviket gis ved `sd()`:

```r
sd(abu89$time89, na.rm = TRUE)
```

```
[1] 30.31473
```

Medianen kan angis med `quantile()`, men enklere med `median()`:

```r
median(abu89$time89, na.rm = TRUE)
```

```
[1] 83.33333
```

Vi trenger også ofte antall. `nrow()` gir antall rader, dvs. antall observasjoner i datasettet

```
nrow(abu89)
```

```
[1] 4127
```

Tilsvarende gir `ncol()` antall kolonner, mens `dim()` gir begge deler:

```
ncol(abu89)
```

```
[1] 9
```

```
dim(abu89)
```

```
[1] 4127    9
```

8.2 Professjonelle tabeller med gtsummary

For å lage ordentlig professjonelle tabeller kreves det mer. For det første skal de se ordentlige ut, men de skal også kunne eksporteres til andre formater på en hensiktsmessig måte.

I R finnes det en hel rekke slike funksjoner. Her har vi vektlagt pakken `gtsummary` fordi den gir gode tabeller fra helt enkle til ganske avanserte relativt lett. Det er også mange muligheter for å justere tabellene slik du vil. Dessuten kan resultatene eksporteres lett til de fleste aktuelle formater (Word, html, pdf, Excel, latex).

Avanserte brukere vil muligens se begrensningene i denne pakken og foretrekke noe annet. De fleste vil kunne lage det aller meste med denne pakken.

Vi starter med en enkel oversiktstabell med alle variablene i datasettet. Men vi fjerner løpenummeret for person, nemlig variabelen *io_nr* fordi den ikke inneholder noe analyserbar informasjon.

```
abu89 %>%
  select(-io_nr) %>%
  tbl_summary()
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**N = 4,127**
Gjennomsnittlig timelønn 1989	83 (71, 103)
Unknown	368
År utdanning	
0	839 (20%)
1	1,156 (28%)
3	1,121 (27%)
5	483 (12%)
7	308 (7.5%)
9	205 (5.0%)
11	15 (0.4%)
Alder	39 (30, 48)
Respondentens kjønn	1,934 (47%)
Goldthorpe klasse 1989	
I Øvre serviceklasse	328 (8.1%)
II Nedre serviceklasse	1,181 (29%)
III Rutinefunksjonærer	1,248 (31%)
V-VI Faglærte arbeidere	648 (16%)
VIIa Ufaglærte arbeidere	637 (16%)
Unknown	85
Noen gang forfremmet	
NEI	2,568 (62%)
JA	1,559 (38%)
Bedriftserfaring	0.70 (0.20, 1.40)
Privat sektor	
Public	1,602 (39%)
Private	2,525 (61%)

Legg merke til at `tbl_summary` gjør en del ting automatisk. Først og fremst er bruker den *variabel label* og *factor levels* i sidespalten. Ofte vil ikke variable ha slike labler, og da vil det vises variabelnavnene. Variabelen *kjønn* har ikke angitt factor levels, og variabelen har bare verdiene 0 og 1, og da rapporteres kun den ene kategorien (dvs. verdien 1). Vi kan legge til annen tekst hvis vi ønsker.

Dernest er det en forhåndsinnstilling som angir at det for kontinuerlige variable skal rapporteres median og interquartile range (IQR), dvs. nedre og øvre kvartil i parentes. Det gir en god beskrivelse av variablene, men vi skal endre dette nedenfor. For kategoriske variable rapporteres det antall observasjoner og andelen i prosent i parentes.

Men merk at for antall år utdanning og kjønn, så er det rapportert som kategoriske variable selv om variabeltypen er kontinuerlig. `tbl_summary` gjør dette fordi det er relativt få kategorier slik at median og IQR ikke er så interessant uansett.

La oss først endre slik at det rapporteres gjennomsnitt og standardavvik i stedet. Det er mer vanlig å gjøre selv om det ikke er noen regel for dette. Funksjonen `theme_gtsummary_mean_sd()` endrer standardvalget for `tbl_summary` i *alle etterfølgende tabeller*. Dermed slipper du endre neste gang. Flere themes finner du på pakkens hjemmeside. For å gå tilbake til opprinnelig theme brukes funksjonen `reset_gtsummary_theme()`.

Vi kan endre andre ting ved tabellen med noen enkle grep. Alle variable kan endre navn i forspalten med å legge til argumentet `label =`. Nedenfor er to variable endret for å vise hvordan man endrer flere variable. Når det er flere variable må de spesifiseres innenfor argumentet `list()` som nedenfor. Her endrer vi også label for variabelen female og klasse89.

Noen ganger kan man også ønske å endre hvordan en variabel presenteres. Et vanlig behov er å presisere hvilken type en variabel er. I dette tilfellet er utdanning antall år etter obligatorisk skolenivå, så det er egentlig en kontinuerlig variabel selv om antall verdier er få. Vi kan velge å presisere at denne er av typen *continuous*. Nedenfor presiserer vi også at female er kategorisk, *dichotomous*, selv om denne ble presentert riktig uansett. Vi bruker argumentet `type =` og flere variable må oppgis innenfor `list()`.

En siste ting vi kan endre er å ikke rapportere NA. Det er ikke oppgitt timelønn for alle, så antall NA er rapportert for seg. Det kan være fint, men kan også hende vi ikke ønsker det. Nedenfor er det derfor også lagt til `missing = "no"`.

```
theme_gtsummary_mean_sd()
abu89 %>%
  select(-io_nr) %>%
  tbl_summary(label = list(female ~ "Kjønn", klasse89 = "Klasse"),
              type = list(ed ~ "continuous", female ~ "dichotomous"),
              missing = "no")
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**N = 4,127**
Gjennomsnittlig timelønn 1989	90 (30)
År utdanning	2.69 (2.56)
Alder	40 (12)
Kjønn	1,934 (47%)
Klasse	
I Øvre serviceklasse	328 (8.1%)
II Nedre serviceklasse	1,181 (29%)
III Rutinefunksjonærer	1,248 (31%)

Characteristic	N = 4,127
V-VI Faglærte arbeidere	648 (16%)
VIIa Ufaglærte arbeidere	637 (16%)
Noen gang forfremmet	
NEI	2,568 (62%)
JA	1,559 (38%)
Bedriftserfaring	0.95 (0.91)
Privat sektor	
Public	1,602 (39%)
Private	2,525 (61%)

Ofte vil vi ha en tabell som ikke bare viser univariat fordeling, men bi-variate, altså fordelt på to eller flere grupper. Det er f.eks. ganske vanlig å vise tabeller fordelt på kjønn. Det kan vi også gjøre her ved å legge til argumentet `by = female`. Nedenfor er det også forenklet argumentene for `label =` og `type =`. I slike tilfeller vil vi ofte ha totalen i tillegg til per gruppe, og det gjør vi ved å legge til funksjonen `add_overall()`.

For de kontinuerlige variablene får vi ikke antallet som inngår i beregningene. Vi vil gjerne vise antall ikke-missing verdier - særlig fordi vi tok vekk NA som egen rad ovenfor. Dette gjør vi ved å legge til funksjonen `add_n()`.

```
abu89 %>%
  select(-io_nr) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
    tbl_summary(by = female,
                label = list(klasse89 = "Klasse"),
              type = list(ed ~ "continuous"),
              missing = "no") %>%
  add_overall() %>%
  add_n()
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	N	**Overall**, N = 4,127	**Kvinner**, N = 1,934	**Menn**, N = 2,193
Gjennomsnittlig timelønn 1989	3,759	90 (30)	79 (24)	100 (32)

Characteristic	N	**Overall**, N = 4,127	**Kvinner**, N = 1,934	**Menn**, N = 2,193
År utdanning	4,127	2.69 (2.56)	2.38 (2.40)	2.96 (2.66)
Alder	4,127	40 (12)	40 (13)	40 (12)
Klasse	4,042			
I Øvre serviceklasse		328 (8.1%)	74 (3.9%)	254 (12%)
II Nedre serviceklasse		1,181 (29%)	555 (29%)	626 (29%)
III Rutinefunksjonærer		1,248 (31%)	986 (52%)	262 (12%)
V-VI Faglærte arbeidere		648 (16%)	46 (2.4%)	602 (28%)
VIIa Ufaglærte arbeidere		637 (16%)	244 (13%)	393 (18%)
Noen gang forfremmet	4,127			
NEI		2,568 (62%)	1,308 (68%)	1,260 (57%)
JA		1,559 (38%)	626 (32%)	933 (43%)
Bedriftserfaring	4,127	0.95 (0.91)	0.83 (0.81)	1.05 (0.97)
Privat sektor	4,127			
Public		1,602 (39%)	1,016 (53%)	586 (27%)
Private		2,525 (61%)	918 (47%)	1,607 (73%)

Men vi kan lage mer kompliserte tabeller også. La oss si at vi ønsker å lage den samme tabellen som over, men fordelt på to grupper. Det kan være relevant å sammenligne offentlig og privat sektor. En mulighet er å lage en ny grupperingsvariabel ved å slå sammen kjønn og sektor slik at vi får fire kategorier. Men vi får et bedre resultat ved å lage en stratifisert tabell med funksjonen `tbl_strata()`. Det er litt kryptisk syntaks, men det viktige er å angi hvilken variabel det skal stratifiseres etter med argumentet `strata =` etterfulgt av `.tbl_fun = ~ .x %>%`, så kommer `tble_summary` etter dette. Her er det også lagt til en ekstra header med antall observasjoner.

```
abu89 %>%
  select(-io_nr) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
  tbl_strata(strata = private,
             .tbl_fun =
               ~ .x %>%
               tbl_summary(by = female,
              label = list(klasse89 = "Klasse"),
              type = list(ed ~ "continuous"),
              missing = "no") %>%
               add_n(),
    .header = "**{strata}**, N = {n}"
```

```
    )
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**N**	**Kvinner**, N = 1,016	**Menn**, N = 586	**N**	**Kvinner**, N = 918	**Menn**, N = 1,607
Gjennomsnittlig timelønn 1989	1,403	82 (23)	100 (28)	2,356	76 (24)	100 (33)
År utdanning	1,602	2.88 (2.67)	4.22 (3.12)	2,525	1.82 (1.92)	2.50 (2.31)
Alder	1,602	42 (12)	43 (11)	2,525	37 (13)	39 (12)
Klasse	1,592			2,450		
I Øvre serviceklasse		55 (5.4%)	150 (26%)		19 (2.1%)	104 (6.7%)
II Nedre serviceklasse		340 (34%)	196 (34%)		215 (24%)	430 (28%)
III Rutine-funksjonærer		507 (50%)	64 (11%)		479 (54%)	198 (13%)
V-VI Faglærte arbeidere		5 (0.5%)	114 (20%)		41 (4.6%)	488 (31%)
VIIa Ufaglærte arbeidere		104 (10%)	57 (9.8%)		140 (16%)	336 (22%)
Noen gang forfremmet	1,602			2,525		
NEI		696 (69%)	318 (54%)		612 (67%)	942 (59%)
JA		320 (31%)	268 (46%)		306 (33%)	665 (41%)
Bedriftserfaring	1,602	0.93 (0.85)	1.15 (0.94)	2,525	0.72 (0.74)	1.01 (0.98)

Det går også an å lage langt mer avanserte tabeller enn dette, og alle deler kan modifiseres. Men vi går ikke inn på dette her. Ved behov finner du instruksjoner på pakkens hjemmeside.

8.2.1 Eksport av tabeller

Du skal aldri bruke "klipp og lim" for å få en tabell over i et tekstbehandlingsprogram. Trikset er å konvertere tabellen til gt-format som har en eksportfunksjon til MS Word.

Først lagres tabellen i et eget objekt.

```
fintabell <- abu89 %>%
  select(-io_nr) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
  tbl_strata(strata = private,
             .tbl_fun =
               ~ .x %>%
               tbl_summary(by = female,
              label = list(klasse89 = "Klasse"),
              type = list(ed ~ "continuous"),
              missing = "no") %>%
               add_n(),
    .header = "**{strata}**, N = {n}"
    )
```

Så kan tabellen eksporteres til Word, og evt. redigeres videre der hvis det trengs. På dette nivået kan det være mer tidsbesparende å gjøre siste justeringer i Word fremfor å lære alle triks for å lage tabellen fiks ferdig i R. (Skal du lage mange tabeller kan det likevel lønne seg å gjøre mest mulig i R).

```
fintabell %>%
  as_gt() %>%
  gt::gtsave(filename = "output/fintabell.docx")
```

Merk at eksport til docx-formatet krever at du har en relativt ny installasjon av pakkene {gt} og {gtsummary}. Filhalen ".docx" innebærer at filen lagres i dette formatet. Tilsvarende kan du lagre i .html, .pdf, .rtf, .png, .tex eller .ltex bare ved å endre filhalen.

En tilsvarende variant som noen av dere har lært på sosgeo1120 er å bruke `as_flextable` og en tilsvarende eksportfunksjon. Det er selvsagt også helt ok. En tidligere versjon av {gt} kunne som sagt ikke eksportere til Word, så da var {flextable} beste løsning. Men pakken {flextable} har vist seg å være litt trøblete å installere på noen pc'er, så da er det bedre å bruke {gt}.

8.3 Manuelle tabeller

Noen ganger trenger man å lage ganske spesifikke ting.

8.3.1 For datasettet totalt

8.3.2 Grupperte statistikker

8.4 Oppgaver

Slå opp i boken R for data science hvis du står fast eller ikke skjønner hva koden betyr.

Exercise 8.1. Bruk datasettet abu89 og lag de samme tabellene som vist her, gjør noen endringer på kodene for å endre utseendet på tabellene. Det er et mål at du skal forstå hva hver enkelt kommando gjør.

Exercise 8.2. Last inn datasettet NorLAG i R. Velg noen variable som du selv tenker kan være informative å se nærmere på. Bruk de samme teknikkene på disse variablene.

Part III

Del 3: Flere variable på en gang

9 Regresjon: Sammenheng mellom variable

```
library(tidyverse)
library(gtsummary)
library(modelsummary)
```

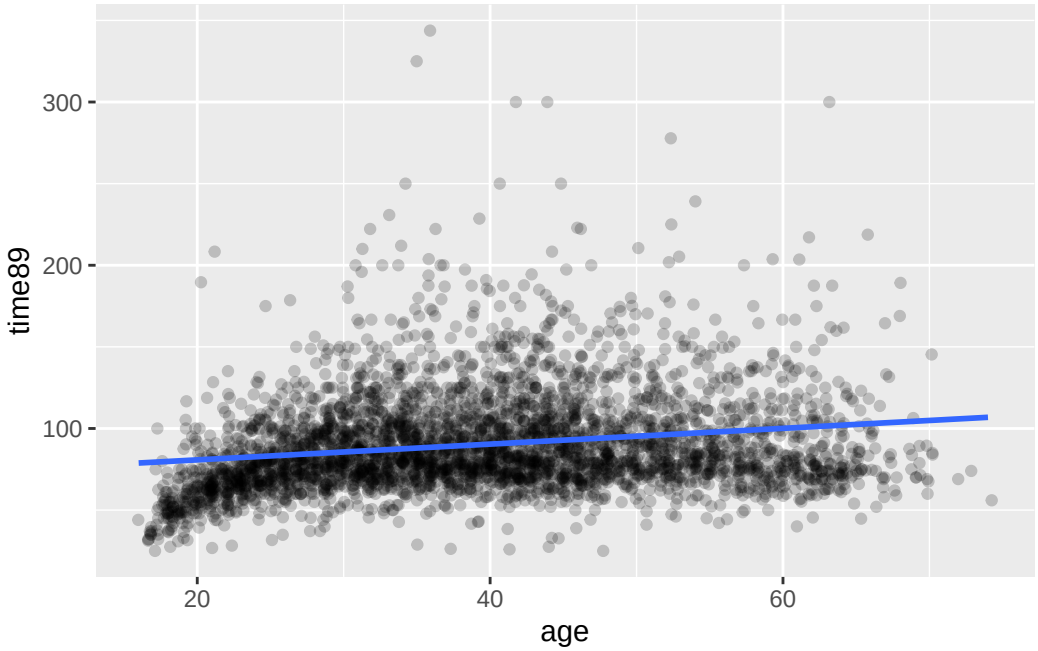
Vi skal her se på helt grunnleggende lineær regresjon med en og to forklaringsvariable.

9.1 Scatterplot

Bivariat regresjon beskriver sammenhengen mellom to variable. En naturlig start er å se på et scatterplot. Her er en figur som viser hvordan timelønn varierer med alder. I det nedenforstående er det brukt jitter og gjennomsiktig farge for å håndtere overplotting.

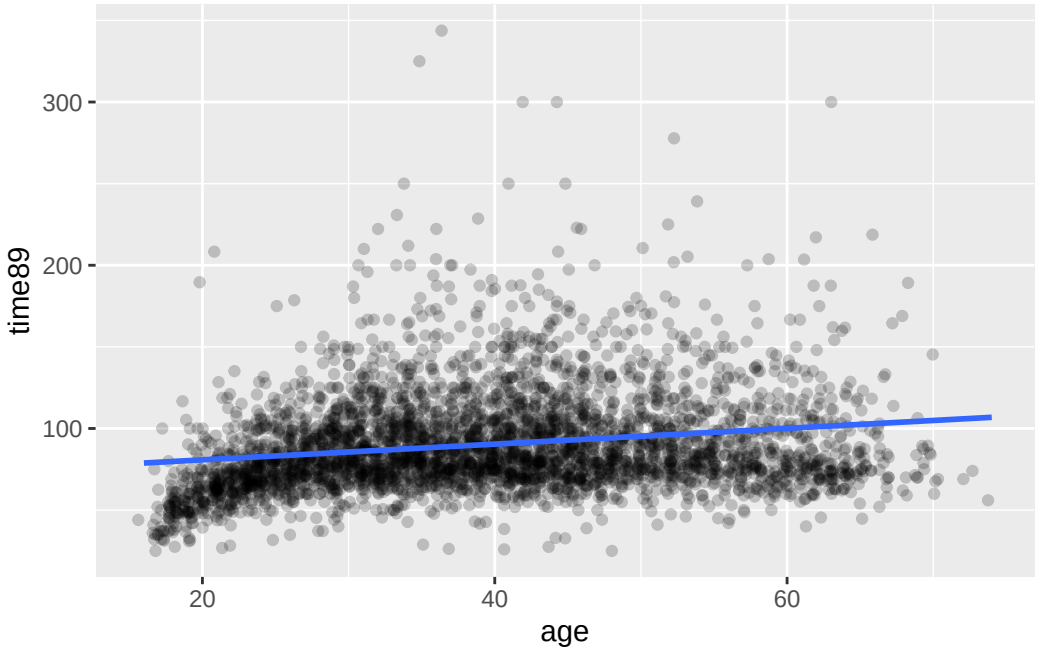
I tillegg er det tegnet inn en linje som illustrerer *trenden i gjennomsnittlig lønn* med alder. Denne linjen skrår svakt oppover, som altså betyr at gjennomsnittlig lønn øker noe med alder. Vi ser med det blotte øyet at en rett linje ikke beskriver denne sammenhengen perfekt. Først og fremst er det en stor variasjon rundt denne linjen, så det er mye annet som påvirker lønna enn alder. Det er også verd å legge merke til at i de yngste aldersgruppene er lønna en god del lavere - og kanskje litt lavere i eldste aldersgrupper også. Så en rett linje er kanskje ikke optimalt i utgangspunktet. Fordelen med en rett linje er at vi kan si noe slikt som at "gjennosmsnittslønna øker med x antall kroner for hvert år eldre man blir". Hvis linja er kurvlineær blir det litt mer komplisert. Så et første poeng er at en slik linje er en *forenkling*, og det er en tilsiktet forenkling.

```
ggplot(abu89, aes(x =age, y = time89))+
  geom_jitter(alpha = .2)+
  geom_smooth(method = "lm", se = FALSE)
```



Det er en viss tendens til at lønnen øker med alder, men det er ikke helt lett å si hvor mye. Poenget med lineær regresjon er å beskrive en gjennomsnittlig trend.

```
ggplot(abu89, aes(x =age, y = time89))+
  geom_jitter(alpha = .2)+
  geom_smooth(method = "lm", se = FALSE)
```



Denne trendlinja er hva vi vanligvis kaller regresjonslinje.

9.2 Regresjonslinja

Regresjonslinja kan beskrives med et *stigningstall*, som sier hvor bratt linjen er. Substansielt sett betyr det hvor mye utfallsvariabelen (y-aksen) endres med økning i forklaringsvariabelen (x-aksen). I tillegg trenger vi også vite hvor høyt/lavt linjen ligger.[1] Til det bruker vi startpunktet for linjen, der hvor x har verdien 0. Dette må regnes ut, og det er akkurat dette estimering av lineær regresjon gir oss.

Utregningen av regresjonslinja går vi ikke inn på her, men intuitivt sett ønsker vi jo *den beste linja* og ikke en hvilken som helst omtrentlig linje. Datapunktene (de svarte punktene i grafen) er spredt rundt linja, og avstanden mellom linje og punkt kalles *residualer*. Summen av disse residualene er grunnlaget for mål på hvor godt regresjonslinja beskriver de faktiske dataene. Den beste linja er definert som den som minimerer residualene. Det er dette som kalles "minste kvadraters metode".

I R estimeres regresjonsmodeller med funksjonen `lm`. Første argument er en formel på formen `utfallsvariabel ~ forklaringsvariabel`. Rekkefølgen variablene oppgis i er altså viktig.

[1] Du kan jo tenke deg flere parallelle linjer i plottet ovenfor med samme stigningstall

Dernest må det spesifiseres hvilket datasett som skal brukes med `data = .`[2]

Legg alltid resultatene i et eget objekt med et navnt som er rimelig enkelt å forstå hva er. I følgende kode legges resultatet i en nytt objekt `lm_est1`. Deretter bruker kan man hente ut de delene av resultatet vi er interessert i. I aller første omgang er bare interessert i regresjonslinjas konstantledd (startpunktet) og stigningstall. Disse kaller vi vanligvis *regresjonskoeffisienter*. Det kan vi få ut ved å bruke funksjonen `coef`. (Vi kommer tilbake til å se på de fulle resultatene senere, som vi oftest er interessert i).

```
lm_est1 <- lm(time89 ~ age, data = abu89)
coef(lm_est1)
```

```
(Intercept)         age
 71.1101883   0.4828415
```

Regresjonslingningen kan skrives på formel der α er konstantleddet og β er stigningstallet slik:

$$\text{time89} = \alpha + \beta_1(\text{age}) + \epsilon$$

Når vi setter inn de estimerte koeffisientene inn i ligningen får vi følgende:

$$\hat{\text{time89}} = 71.11 + 0.48(\text{age})$$

Tolkningen her er at gjennomsnittlig forskjell i timelønn mellom grupper der aldersforskjellen er ett år er 0.48 kroner i favør av den eldre gruppen.[3] Merk enheten her: stigningstallet tolkes på den skalaen utfallsvariabelen er på, i dette tilfellet kroner. Det er også uttrykt endring ved at forklaringsvariabelen endres med nøyaktig 1.

Vi sier gjerne at regresjonslinjen er *estimert*, og det innebærer at det er usikkerhet i estimatene. Vi kommer tilbake til dette, men en vanligere output fra regresjonsmodeller er å bruke funksjonen `summary`. Da får man med mye mer detaljer og output vil se ut som følger:

```
summary(lm_est1)
```

[2] Grunnen til det siste er at R kan ha flere datasett oppe samtidig, så R vet ikke nødvendigvis hvilket datasett du tenker på

[3] Noen ganger sier man at gjennomsnittslønna øker med 0.48 kroner for hvert år eldre man blir. Men det er ikke helt riktig, for dataene beskriver jo ikke individuell endringer over tid! Men hvis du synes det er lettere å tenke på det på den måten er det ok - men prøv å husk at det også er litt feil.

```
Call:
lm(formula = time89 ~ age, data = abu89)

Residuals:
    Min      1Q  Median      3Q     Max
-69.287 -19.131  -6.304  12.864 255.258

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 71.11019    1.62232   43.83   <2e-16 ***
age          0.48284    0.03926   12.30   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.73 on 3757 degrees of freedom
  (368 observations deleted due to missingness)
Multiple R-squared:  0.0387,    Adjusted R-squared:  0.03844
F-statistic: 151.2 on 1 and 3757 DF,  p-value: < 2.2e-16
```

9.3 Dummy-variable

Hvis en forklaringsvariabel har kun to verdier vil vi typisk gi den ene kategorien verdien 0 og den andre kategorien 1. Dette kalles en «dummy variabel» eller en «indikator variabel». For eksempel vil et datasett ofte ha en variabel for kjønn med verdiene «Mann» og «Kvinne». Da kan vi la mann få verdien 0 og kvinne verdien 1. Ofte vil man da gi variabelen et navn som indikerer hvilken verdi som er 1. Så i dette eksempelet er det hensiktsmessig å gi den nye variabelen navnet «Kvinne». I dette eksempelet vil man også kunne si at variabelen er en «dummy for kvinne» (altså: den kategorien som får verdien 1).

Det spiller ingen rolle hvilken kategori som får verdien 0 og 1. I dette eksempelet kunne man like gjerne gjort det motsatt, og latt det være en «dummy for mann». Da ville det være naturlig å kalle variabelen «mann» i stedet for «kvinne». Som vi skal se nedenfor vil det bare påvirke fortegnet når vi bruker variabelen i en regresjonsanalyse.

I datasettet abu89 er variabelen “female” en slik variabel som har verdiene 0 eller 1, og der 0 betyr “mann” og 1 betyr “kvinne”.

Kjønn	Female
Mann	0
Kvinne	1
Kvinne	1

Kjønn	Female
Mann	0
Kvinne	1
Kvinne	1
Kvinne	1
Mann	0

I R vil vi ofte ha slike variable som factor-variable. Da er variabelen definert som kategorisk og selv om det er tekst-verdier i variabelen, så vil R automatisk behandle den som om verdiene var 0 og 1 i estimeringen av regresjonsmodellen.

```
theme_gtsummary_mean_sd()
abu89 %>%
  select(female, time89) %>%
  tbl_summary(by = female)
```

Characteristic	**0**, N = 2,193	**1**, N = 1,934
Gjennomsnittlig timelønn 1989	100 (32)	79 (24)
Unknown	190	178

Vi ser altså at menn hadde i gjennomsnitt høyere timelønn enn kvinner, nærmere bestemt 21 kroner mer. Dette kan vi også undersøke med lineær regresjon som følger:

```
lm_est_i <- lm(time89 ~ female , data = abu89)

coef(lm_est_i)
```

```
(Intercept)      female
   99.84382   -20.75229
```

Regresjonskoeffisenten for variabelen female uttrykker nettopp differansen mellom menn og kvinner, som er 21 kroner. Husk at β er hvor mye y-variabelen endres når man sammenligner x-variabelen med akkurat 1 enhets forskjell. Her er mann 0 og kvinner 1, så da er dette faktisk 1 enhets forskjell. Altså: når x går fra 0 til 1, så reduseres y med 21. Derfor negativt fortegn.

I R vil vi ofte gjøre om kategoriske variable til såkalte factor-variable. En factor-variabel vil håndtere kategoriske variable som tekst, men med en underliggende numerisk verdi. Da kan man bruke factor-variable i alle standard analysemetoder. I regresjon vil R automatisk bruke den første kategorien som referansekategori.

Vi kan gjøre den samme analysen med kjønn som en factor variabel og få de samme resultatene som ovenefor.

```r
abu89 <- abu89 %>%
  mutate(sex = factor(ifelse(female == 1, "Female", "Male"), levels = c("Male", "Female"))
  filter(!is.na(time89))

lm_est2 <- lm(time89 ~ female , data = abu89)

coef(lm_est2)
```

```
(Intercept)      female
   99.84382   -20.75229
```

9.3.1 Dummy-variable med mer enn en kategori

Noen ganger har vi forklaringsvariable med flere enn to kategorier. Det kan vi løse på en tilsvarende måte ved å lage flere dummy-variable. Et eksempel kan være sosial klasse. I datasettet abu89 er det fem kategorier.

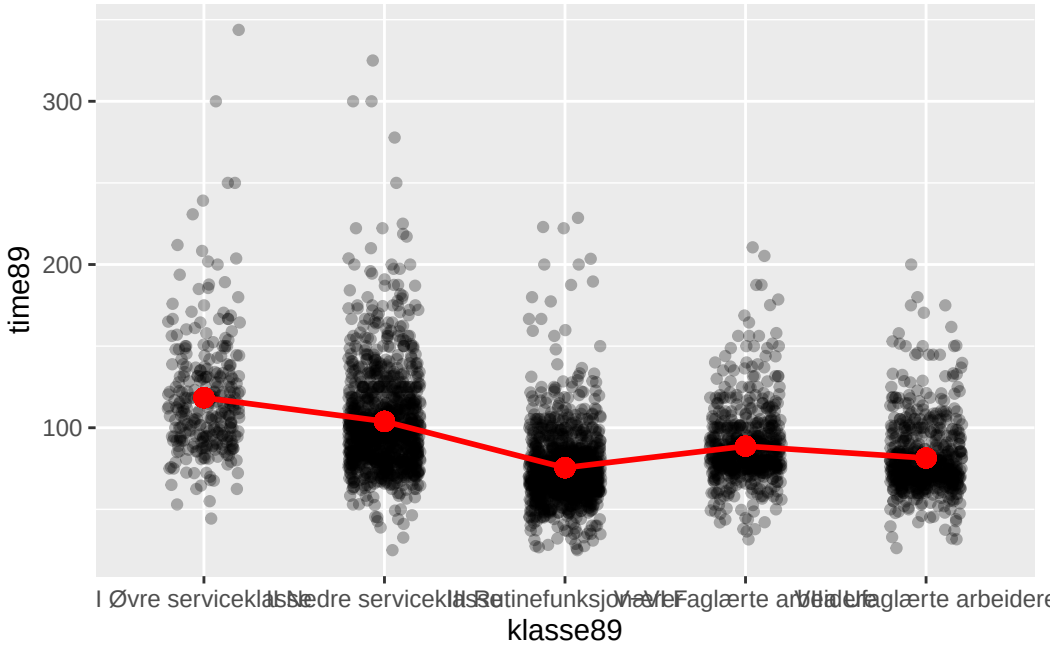
En dummy-variabel har bare to kategorier: 0 og 1, men vi kan lage flere dummy-variable. Vi kan lager en ny variabel «klasse II» som har verdien 1 hvis personen tilhører denne klassen og 0 ellers. Altså en dummy. Så kan vi lage en ny variabel «Klasse III» som har verdien 1 hvis personen tilhører denne klassen og 0 ellers. Slik kan man lage en dummy-variabel for hver av kategoriene. Da har vi altså flere dummy-variable som til sammen fanger opp informasjonen i den opprinnelige variabelen.

Utdanning	Klasse II	Klasse III	Klasse V-VI	Klasse VII
Klasse I	0	0	0	0
Klasse II	1	0	0	0
Klasse III	0	1	0	0
Klasse V-VI	0	0	1	0
Klasse VII	0	0	0	1

Merk at her er det ingen dummy for "Klasse I". Denne gruppen brukes som referansekategori slik at estimatene for *de andre* dummyene blir tolkbare som forskjellen til denne referansekategorien. Mer om det siden, men man kan velge å bruke en annen referansekategori hvis man vil.

La oss først se på et plot. Her er det brukt en jitter-plot. Den røde linjen viser endring i gjennomsnitt mellom de kategoriene. En regresjonsanalyse vil gi slike estimater på differanser, men det er enklest hvis alle endringene er i forhold til samme referansekategori.

```
ggplot( abu89x, aes(x =klasse89, y = time89))+
  geom_jitter(alpha = .3, width = .2)+
  geom_point(aes(y=gr_snitt), col = "red", size = 3)+
  #geom_hline(yintercept = mean(abu89x$time89, na.rm = TRUE))+
  geom_line(aes(x = as.numeric(klasse89), y = smooth), col = "red", linewidth = 1)
```



Den generelle regresjonsligningen skrives som $y = a + bx$, der x er forklaringsvariabelen. Regresjonskoeffisienten, b, tolkes som hvor forskjellen i gjennomsnittet på utfallsvariabelen, y, mellom de som er en enhets forskjell på x-variabelen.

Så kan vi kjøre en regresjon og få ut regresjonskoeffisientene på samme måte som før. Akkurat her er `coef` lagt inn i en parentes for `data.frame`, men det er bare for at det skal bli en pen kolonne. Vi kommer altså tilbake til teknikker for penere output nedenfor.

```
est2 <- lm(time89 ~ klasse89, data = abu89)
data.frame(coef(est2))
```

```
                                coef.est2.
(Intercept)                      118.39851
klasse89II Nedre serviceklasse   -14.49078
klasse89III Rutinefunksjonærer   -42.89842
```

```
klasse89V-VI Faglærte arbeidere   -29.68788
klasse89VIIa Ufaglærte arbeidere  -36.95234
```

Hvert estimat for kategori for klasse sammenlignes med *den første kategorien* (altså den som mangler): klasse I. Det betyr at klasse VII (ufaglærte arbeidere) har en timelønn på -37 kroner mindre enn klasse I (øvre serviceklasse). Mens klasse II (nedre serviceklasse) tjener -14.5 kroner mindre enn klasse I.

Forskjellen mellom andre grupper er således differansen mellom disse estimatene. Altså: klasse VII tjener mindre enn klasse V-VI: $36.9 - 29.7 = 7.2$ kroner. Se på plottet over, så ser du at disse tallene ser riktige ut.

9.4 Flere variable

Det er ikke så ofte vi bruker regresjon med bare en forklaringsvariabel, såklat "enkel lineær regresjon".[4] Langt mer vanlig er å bruke flere variable samtidig i det vi kaller "multippel regresjon".[5] I multippel regresjon kan man altså beskrive mer kompliserte mønstre i dataene.

Vi fortsetter med eksempelet om lønn og alder, men utvider med en dimensjon til, nemlig kjønn. La oss først se på kjønnsforskjellene i gjennomsnittlig timelønn.

```
abu89 <- abu89 %>%
  mutate(sex = factor(ifelse(female == 1, "Female", "Male"), levels = c("Male", "Female"))
  filter(!is.na(time89))

abu89 %>%
  select(sex, time89) %>%
  tbl_summary(by = sex)
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**Male**, N = 2,003	**Female**, N = 1,756
Gjennomsnittlig timelønn 1989	100 (32)	79 (24)

Vi ser altså at menn hadde i gjennomsnitt høyere timelønn enn kvinner, nærmere bestemt 21 kroner mer. Dette kan vi også undersøke med lineær regresjon som følger:

[4] Det er selvsagt ingenting *enkelt* med slike modeller utover at det finnes mer kompliserte varianter.

[5] Noen kaller dette også for *multivariat* regresjon, men det er tvetydig da det også kan bety modeller med flere utfallsvariable, som er noe ganske annet.

```
lm_est2 <- lm(time89 ~ sex , data = abu89)

coef(lm_est2)
```

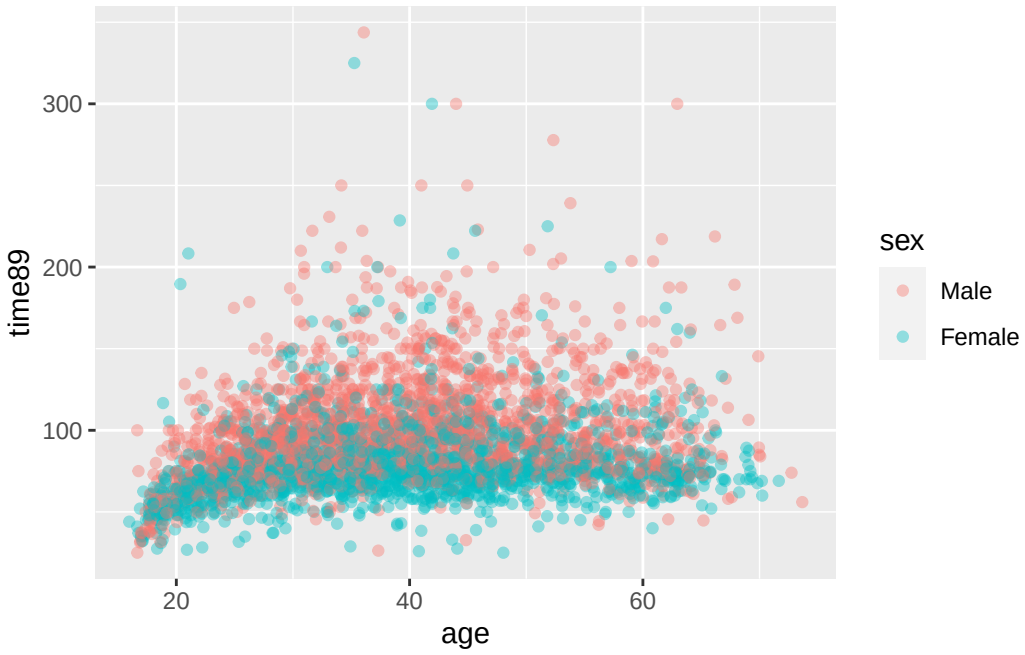
```
(Intercept)   sexFemale
   99.84382   -20.75229
```

Det er altså slik at koeffisienten, β, gir den samme differansen som en enkel sammenligning av to gjennomsnitt.

Vi har allerede sett på alder og lønn, så vi kan utvide dette til å inkludere kjønn samtidig i et scatterplot.

Grafisk er det da greit å bruke farger og slik vise for menn og kvinner for seg. I `ggplot` spesifiseres da `group = sex` og at fargene skal settes etter sammen grupperingen `col = sex` slik:

```
ggplot(abu89, aes(x = age, y = time89, group = sex, col = sex)) +
  geom_jitter(alpha = .4)
```



```r
lm_est3 <- lm(time89 ~ sex + age, data = abu89)

summary(lm_est3)
```

```
Call:
lm(formula = time89 ~ sex + age, data = abu89)

Residuals:
   Min     1Q Median     3Q    Max
-72.37 -17.12  -4.90  10.99 247.94

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  81.10147    1.58497   51.17   <2e-16 ***
sexFemale   -20.62511    0.91186  -22.62   <2e-16 ***
age           0.47380    0.03684   12.86   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 27.89 on 3756 degrees of freedom
Multiple R-squared:  0.1539,    Adjusted R-squared:  0.1535
F-statistic: 341.7 on 2 and 3756 DF,  p-value: < 2.2e-16
```

9.5 Prediksjon

Lineær regresjon kan også brukes til *prediksjon* selv om dette i liten grad er hva samfunnsvitere bruker regresjonsmodeller til. Vanligvis vil vi primært være interessert i å tolke regresjonskoeffisientene som sammenligninger mellom grupper. Man kan si at man da er opptatt av *forklaringsvariablene*. Når man predikerer er man derimot opptatt av *utfallsvariabelen*. Hvis man skulle være interessert i temaer som maskinlæring, vil dette være en god inngang til det.[6]

9.5.1 Regne ut forventet verdi

Merk at konstantleddet er tolkbart som forventet verdi på utfallsvariabelen når alle andre prediktorer er null. I eksempelet ovenfor med timelønn og klassetilhøringhet, er det altså ikke noen koeffisient for klasse I. Gjennomsnittlig timelønn i klasse I er når de andre dummyene er

[6] f.eks. Et introduksjonskurs i maskinlæring som SOS2901 starter gjerne med nettopp prediksjon med lineær regresjon.

0, altså 118.4 kroner. Gjennomsnittlig timelønn for klasse II er tilsvarende 118.4 + (-14.49) = 103.9.

Hvis du skulle gjette timelønna til en person uten å vite noe annet enn klasseposisjon ville det være fornuftig å gjette på gjennomsnittet for denne klassen. Så vi kan bruke regresjonsmodeller til å regne ut gjennomsnittslønn for gitte verdier av forklaringsvariable. I dette eksempelet er det kanskje litt i overkant komplisert da vi jo også bare kunne regnet ut gjennomsnittet per gruppe i en enkel tabell. Det ville faktisk gitt akkurat samme resultat. Det er mer nyttig med kontinuerlige variable og mer kompliserte modeller.

La oss se på regresjonsmodellen for hvordan timelønn varierer med alder i stedet. Alder er kontinuerlig, så det er få personer på hvert alderstrinn.

```
coef(lm_est1)
```

```
(Intercept)         age
 71.1101883   0.4828415
```

Gjennomsnittlig timelønn ved 30 år vil da være $71.1 + 30 \times 0.5 = 86.1$ kroner. Ved 35 år blir det tilsvarende $71.1 + 35 \times 0.5 = 88.6$ kroner.

Dette kan vi altså regne ut for hånd, men man kan også bruke en r-funksjon, nemlig `predict`. Denne funksjonen tar som argument et regresjonsobjekt og en data.frame (altså et datasett eller tilsvarende struktur) med samme variabelnavn som ble brukt i opprinnelig regresjonsmodell.[7]

9.5.2 Predikere for kontinuerlig variabel

Koden nedenfor lager først et datasett med én variabel: alder med noen verdier man er interessert i utregning for. Så bruker man `mutate` til å lage en kolonne til med predikerte verdier.

```
nyedata <- data.frame(age = c(17, 20, 30, 40, 50, 60))

nyedata %>%
  mutate(pred = predict(lm_est1, newdata = nyedata))
```

```
  age      pred
1  17  79.31849
2  20  80.76702
```

[7] Hvis man ikke spesifiserer data.frame brukes opprinnelige data og predikerer for hver person. Det kan man også gjøre for å f.eks. regne ut residualer, men vi gjør ikke det her nå.

```
3  30  85.59543
4  40  90.42385
5  50  95.25226
6  60 100.08068
```

9.5.3 Predikere kategorisk variabel

Tilsvarende kan vi predikere for kategoriske variable slik som ble benyttet i regresjonsmodellen for klasse. I det nye datasettet spesifiseres f.eks. alle nivåene av en factor-variabel med funksjonen `levels` og predikerer for disse.

```
nyedata <- data.frame(klasse89 = levels(abu89$klasse89))

nyedata %>%
  mutate(pred = predict(est2, newdata = nyedata))
```

```
                klasse89      pred
1     I Øvre serviceklasse 118.39851
2   II Nedre serviceklasse 103.90773
3   III Rutinefunksjonærer  75.50009
4  V-VI Faglærte arbeidere  88.71063
5 VIIa Ufaglærte arbeidere  81.44618
```

9.5.4 Predikere for multippel regresjon

Nytten av predict-funksjonen kommer mer til sin rett ved mer kompliserte modeller. Her er et eksempel med flere variable:

Så kan vi regne ut estimert gjennomsnittlig verdi for de ulike kombinasjonene av utvalgte verdier som følger:

```
nyedata <- expand.grid(age = 30:35,
                       sex = c("Female", "Male"))
nyedata %>%
  mutate(pred = predict(lm_est3, newdata = nyedata))
```

```
   age    sex     pred
1   30 Female 74.69049
2   31 Female 75.16429
3   32 Female 75.63810
```

```
4   33 Female 76.11190
5   34 Female 76.58571
6   35 Female 77.05951
7   30   Male 95.31560
8   31   Male 95.78940
9   32   Male 96.26320
10  33   Male 96.73701
11  34   Male 97.21081
12  35   Male 97.68462
```

Funksjonen `predict` fungerer på tilsvarende måte uansett hvor komplisert modellen måtte være. Men det kreves at det nye datasettet har samme variabelnavn og variabeltype som i opprinnelige data.

9.6 Pene tabeller og eksport til fil

Slik resultatene ser ut med bruk av `summary` er forsåvidt fint, og du får den informasjonen du trenger. Men det er ikke særlig presentabelt som ferdig produkt i en analyse. Du trenger typisk to ting: 1) Samle flere regresjonsmodeller i samme tabell, og 2) gjøre tabellene penere og lettere å lese, og 3) eksportere til det tekstbehandlingsprogrammet du bruker, typisk Microsoft Word.

R har en hel rekke funksjoner for dette. Det spiller egentlig ingen rolle hvilke funksjoner du bruker da det er noe smak og behag her, så det viktigste er at det fungerer rimelig greit for deg. Nedenfor presenteres tre pakker for dette formålet. Velg én av dem. Hvis du ikke har egne preferanser, så velg det første alternativet: {modelsummary}. Alle disse funksjonene håndterer svært mange typer modeller, gir gode muligheter for å ferdigstille tabellene fullstendig før eksport, og eksporterer til de formatene som er mest aktuelle. De har også mer avansert funksjonalitet som f.eks. å rapportere robuste standardfeil (av forskjellig type) i stedet for vanlige standardfeil (dette er pensum på SOS4020).

Vi vil som regel ha behov for å flytte resultatene over til et tekstbehandlingsprogram. En strategi som går ut på "klipp og lim" eller skjermbilde etc er uaktuelt og må unngås for nærmest enhver pris.[8] Resultatene skal skrives til en fil på en effektiv måte. Det er en fordel om tabellene da ser ganske ok ut i utgangspunktet og du kan bruke samme prosedyre for å eksportere til flere typer format hvis behovet skulle melde seg. Det er jo MS Word som er viktigst for de fleste, mens de øvrige formatene nedenfor er for spesielt interessert - men noen av dere vil kanskje bli det på et senere tidspunkt. De viktigste formatene som er:

- MS Word - det vanligste tekstbehandlingsprogrammet som de aller fleste av dere bruker.

[8] Hvis du blir tatt i å gjøre slikt vil faglærer sette fyr på datamaskinen din som straff.

	(1)	(2)
(Intercept)	99.844	81.101
	(0.637)	(1.585)
sexFemale	−20.752	−20.625
	(0.932)	(0.912)
age		0.474
		(0.037)
Num.Obs.	3759	3759
R2	0.117	0.154
R2 Adj.	0.116	0.153
AIC	35 854.9	35 694.9
BIC	35 873.6	35 719.8
Log.Lik.	−17 924.434	−17 843.437
F	496.278	341.699
RMSE	28.49	27.88

- rtf - rikt tekstformat. Er et enklere format som fungerer på tvers av de fleste programmer. Kan brukes i Word også.
- html - for websider
- latex - for mer tekniske dokumenter, særlig hvis du har mye formler og stæsj
- Markdown/Quarto - for dynamiske dokumenter med integrert R-kode og tekst, og kan eksportere ferdig dokument til alle ovennevnte formater[9] Det som fungerer med Markdown fungerer også med Quarto for samme formål.

9.6.1 Alt 1: Bruke `modelsummary()`

Eksporterer til bl.a. følgende formater: Word, rtf, html, latex, markdown

Fordel: Gir pene og oversiktlige tabeller med enkel kode, og relativt enkelt å modifisere videre. Eksporterer direkte til alle viktigste formater. Kan også lett integreres med andre eksterne verktøy, først og fremst "grammar of tables" i pakket {gt} Ulempe:

Her er kode for en enkel tabell med to regresjonsmodeller som vist ovenfor. Merk at objektene med regresjonsresultatene må legges inni funksjonen `list()`.

```
library(modelsummary)
modelsummary(list(lm_est2, lm_est3))
```

Denne tabellen inneholder mer enn du er interessert i. Nedre del av tabellen inneholder "goodness of fit" statistikker, altså mål på hvordan modellen passer til dataene. Det finnes

[9] F.eks. dette dokumentet er skrevet i Quarto

	(1)	(2)
(Intercept)	99.8 (0.6)	81.1 (1.6)
	[98.2, 101.5]	[77.0, 85.2]
sexFemale	−20.8 (0.9)	−20.6 (0.9)
	[−23.2, −18.4]	[−23.0, −18.3]
age		0.5 (0.0)
		[0.4, 0.6]
Num.Obs.	3759	3759
R2	0.117	0.154
F	496.278	341.699

mange slike, men ingen grunn til å gå seg vill i disse her. De kan fjernes med argumentet `gof_omit =` og så angis statistikkene med de navnene du ser i tabellen. Det skrives på en spesiell måte: som en tekststreng angitt med anførselstegn rundt, og `|` mellom hver. I koden nedenfor beholdes kun antall observasjoner, r^2 og F.[10]

Vi gjør et par andre justeringer samtidig for å demonstrere noe funksjonalitet. I stedet for å oppgi estimatet og standardfeil på forskjellig linje kan vi spesifisere å ha det på samme linje med argumentet `estimate =`. Merk at den statistikken du vil rapportere settes i parentes {…} og mellomrom og parentes er ellers som det står. Man har også andre valg, derav det vanligste i bruk er å angi p-verdier eller *stjerner* for å vise disse på en forenklet måte. Det angis ved `{p.value}` eller `{stars}` på tilsvarende måte.

I stedet for standardfeil på egen linje er det her angitt konfidensintervall på neste linje. For konfidensintervall vil det som forvalg være 95%, men vi kan angi f.eks. 99% konfidensintervall i stedet ved `conf_level =`. Hvis man ikke vil ha noe på neste linje kan man angi `statistic = NULL` i stedet. Man kan også velge å sette inn `p.value` eller `stars` på denne linjen.

Merk at utfallsvariabelen i modellene er timelønn i kroner. I forrige tabell ble estimatene gitt med tre desimaler. Det er i overkant mange desimaler. En desimal er mer passende og nedenfor endres dette med `fmt =`.

```
modelsummary(list(lm_est2, lm_est3),
             fmt = 1,
             estimate = "{estimate} ({std.error})",
             statistic = 'conf.int',
             conf_level = .99,
             gof_omit = 'DF|Deviance|R2 Adj.|AIC|BIC|Log.Lik.|RMSE')
```

To siste ting å ta med her er å endre navn på variablene til noe mer presentabelt og eksportere til Word. Med argumentet `coef_rename =` angis variabelen slik den ser ut i output

[10] Øvrige statistikker har selvsagt sitt bruksområde. De nevnte holder for de fleste formål.

	(1)	(2)
Konstant	99.8 (0.6)	81.1 (1.6)
	[98.2, 101.5]	[77.0, 85.2]
Kvinne	−20.8 (0.9)	−20.6 (0.9)
	[−23.2, −18.4]	[−23.0, −18.3]
Alder		0.5 (0.0)
		[0.4, 0.6]
Num.Obs.	3759	3759
R2	0.117	0.154
F	496.278	341.699

og spesifiserer hva du vil skal stå. Koden nedenfor viser eksempel.

For å eksportere til Word settes `output =` med filbane og filnavn, og der filhalen *.docx* angir Word format. Du kan eksportere til annet format ved å angi annen filhale f.eks. .rtf eller .html.

```
modelsummary(list(lm_est2, lm_est3),
             fmt = 1,
             estimate = "{estimate} ({std.error})",
             statistic = 'conf.int',
             conf_level = .99,
             gof_omit = 'DF|Deviance|R2 Adj.|AIC|BIC|Log.Lik.|RMSE',
             coef_rename = c("sexFemale" = "Kvinne",
                             "age" = "Alder",
                            "(Intercept)" = "Konstant"),
             output = "output/reg_table.docx")
```

Merk at Word vil vise tabellen med de fonter etc som er forvalgt for Word. Dette kan du endre i Word etterpå. Det er en rekke funksjoner i Word for å formattere tabeller som du kan bruke.

Pakken {modelsummary} har også en rekke andre funksjoner for å redigere tabeller som du kan utforske ved behov. For avanserte brukere kan man også gjøre om tabellen til et gt-objekt og redigere videre med pakken {gt} eller tilsvarende med pakken {flextable}. Det er altså tilnærmet uendelige muligheter for avanserte tabeller. Dette går imidlertid langt utenfor hva de fleste av dere vil trenge. {modelsummary} har egen hjemmeside med mer detaljer og instruksjoner.

9.6.2 Alt 2: Bruke {stargazer}

Mange R-brukere foretrekker pakken {stargazer}. Dette er en noe eldre funksjon og er derfor godt etablert.

Eksporterer til bl.a. følgende formater: rtf, html, latex, markdown

Fordel: Er en stand-alone pakke men gir enkelt veldig fine tabeller som antakeligvis er det du trenger Ulempe: Eksport til Word er ikke den beste, men god nok.

Stargazer lager tabeller i kun tre formater: latex, html, og ren tekst. Vi velger derfor `type = "text"` for at det skal se ok ut her.

```
library(stargazer)
stargazer(lm_est2, lm_est3, type = "text")
```

```
===============================================================
                                Dependent variable:
                    ---------------------------------------------
                                        time89
                              (1)                    (2)
---------------------------------------------------------------
sexFemale                  -20.752***             -20.625***
                            (0.932)                (0.912)

age                                                0.474***
                                                   (0.037)

Constant                    99.844***              81.101***
                            (0.637)                (1.585)

---------------------------------------------------------------
Observations                 3,759                  3,759
R2                           0.117                  0.154
Adjusted R2                  0.116                  0.153
Residual Std. Error    28.495 (df = 3757)     27.891 (df = 3756)
F Statistic         496.278*** (df = 1; 3757) 341.699*** (df = 2; 3756)
===============================================================
Note:                                *p<0.1; **p<0.05; ***p<0.01
```

Vi kan modifisere tabellen tilsvarende som vi gjorde med {modelsummary}. Forklaringer av de enkelte argumenter finnes i manualen for stargazer.

`covariate.labels =` Angir teksten for variabelnavn. Merk at det oppgis i den *rekkefølgen* det skal stå, så være veldig nøye hvis du har mange variable! `report =` angir hva som skal inngå i tabellen, der hver bokstav viser til spesifikke deler: v = variabelnavn, c = koeffisient/estimat, s = standardfeil. `single.row =` setter statistikkene på samme linje fremfor under hverandre. `keep.stat =` angir hvilke "model fit statistics" som skal rapporteres. Hvis du skriver "all" her får du en lang remse tilsvarende vi fikk med {modelsummary}. `digits =` angir antall desimaler

```
stargazer(lm_est2, lm_est3,
          type = "text",
          covariate.labels = c("Kvinne", "Alder", "Konstant"),
          report = "vcs",
          single.row = TRUE,
          keep.stat = c("n","rsq", "ser"),
          digits = 1)
```

```
===================================================
                            Dependent variable:
                        -------------------------------
                                    time89
                              (1)              (2)
---------------------------------------------------
Kvinne                    -20.8 (0.9)      -20.6 (0.9)
Alder                                       0.5 (0.04)
Konstant                   99.8 (0.6)       81.1 (1.6)
---------------------------------------------------
Observations                3,759            3,759
R2                           0.1              0.2
Residual Std. Error 28.5 (df = 3757) 27.9 (df = 3756)
===================================================
Note:                       *p<0.1; **p<0.05; ***p<0.01
```

For å eksportere til Word kan man bruke rikt tekstformat (.rtf) eller html. rtf-formatet er som navnet tilsier ren tekst og selv om det ser greit ut, så er videre redigering i et tekstbehandlingsprogram krøkete. (Prøv og se selv). Bruk heller html fordi da beholdes tabell-strukturen. Du kan åpne html-tabeller fra Word og redigere videre der ved behov.

```
stargazer(lm_est2, lm_est3,
          type = "text",
          covariate.labels = c("Kvinne", "Alder", "Konstant"),
```

```
          report = "vcs",
          single.row = TRUE,
          keep.stat = c("n","rsq", "ser"),
          digits = 1,
          out = "output/reg_starg.html")
```

Mer detaljer finner du i {stargazer} sin vignette.

9.6.3 Alt 3: Bruke {gtsummary}

Vi har tidligere brukt {gtsummary} for å lage deskriptive tabeller, som er det pakken er best til. Men den kan også lage gode regresjonstabeller. Det er imidlertid en stor ulempe for nybegynnere i R: det er ganske krøkete å sette sammen flere regresjonsmodeller i en samlet tabell. Derfor er rådet å *ikke bruke* denne pakken med mindre du har veldig lyst til å prøve.

Fordel med å bruke denne pakken er at man slipper å lære enda en ny pakke og slik sett ha ett sett med konsistent syntaks. En mulighet er selvsagt å ikke lage tabellene så ferdig i R, men eksportere til Word og redigere ferdig der mer manuelt.

{gtsummary} kan eksportere til følgende formater: Word, rtf, html, latex og markdown. Resultatene kan også lett integreres med andre funksjoner, først og fremst "grammar of tables" i pakket {gt} og {flextable} - altså for mer avanserte ting som vi ikke dekker her.

Prinsippet er å lage hver tabell for seg og så slå dem sammen med `tbl_merge` etterpå. Det innebærer en del mer kode, rett og slett. I utgangspunktet virker det ganske greit, men det er alltid en del småting som krever litt mer.

Følgende kode lager først en ryddig tabell for hver regresionsmodell og så kobler sammen disse to tabellene.

```
lm_tab1 <- tbl_regression(lm_est2, intercept = T,
                          estimate_fun = function(x) style_number(x, digits = 1),
                          show_single_row = "sex",
                          label = list(sex ~ "Kvinne")) %>%
    add_glance_table(include = c(nobs, r.squared, sigma))

lm_tab2 <- tbl_regression(lm_est3, intercept = T,
                          estimate_fun = function(x) style_number(x, digits = 1),
                          show_single_row = "sex",
                          label = list(age ~ "Alder", sex ~ "Kvinne")) %>%
    add_glance_table(include = c(nobs, r.squared, sigma)
  )
```

```r
tbl_merge(tbls = list(lm_tab1, lm_tab2)) %>%
  modify_table_body(
    ~.x %>%
      dplyr::arrange(
        row_type == "glance_statistic", # sort glance table to bottom
        var_label                       # sort by the variable label (a hidden column)
      )
  )
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**Beta**	**95% CI**	**p-value**	**Beta**	**95% CI**	**p-value**
(Intercept)	99.8	98.6, 101.1	<0.001	81.1	78.0, 84.2	<0.001
Alder				0.5	0.4, 0.5	<0.001
Kvinne	-20.8	-22.6, -18.9	<0.001	-20.6	-22.4, -18.8	<0.001
No. Obs.	3,759			3,759		
R^2	0.117			0.154		
Sigma	28.5			27.9		

10 Lineær sannsynlighetsmodell

```
library(tidyverse)
library(gtsummary)
```

I samfunnsvitenskapen har vi ganske ofte kategoriske variable både som forklaringsvariable og utfallsvariable, eller en kombinasjon. I en regresjon vil vi behandle kategoriske variable dem som om de er tall på kontinuerlig akse, men der det typisk bare finnes to verdier: 0 og 1. Dette kalles en dummyvariabel eller en indikatorvariabel.

Når man bruker kategoriske variable i en lineær regresjon er det derfor ikke egentlig noe nytt. Det som står i læreboken om kontinuerlige variable gjelder også for kategoriske variable (i hvert fall for alle praktiske formål som dekkes for dette kurset).

10.1 Dummy som utfallsvariabel

I samfunnsvitenskapen er utfallsvariabelen ganske ofte kategorisk. I en regresjon vil vi behandle kategoriske variable dem som om de er tall på kontinuerlig akse, også der det typisk bare finnes to verdier: 0 og 1. Dette kalles en dummyvariabel eller en indikatorvariabel.

Når man bruker kategoriske variable i en lineær regresjon er det derfor ikke egentlig noe nytt. Det som står i boken om kontinuerlige variable gjelder også for kategoriske variable (i hvert fall for alle praktiske formål som dekkes for dette kurset).

Husk at tolkningen av regresjonskoeffisienten, b, tolkes på den skalaen y-variabelen er på. Altså: hvis utfallsvariabelen er i kroner, så er tolkningen av b i måleenheten kroner. Hvis y-variabelen er i antall timer, så er tolkningen av b i antall timer, osv. Husk også at vi estimerer endring i gjennomsnitt.

Når utfallsvariabelen er en dummy, så har den verdiene 0 eller 1. Da er gjennomsnittet det samme som en andel. For eksempel: hvis utfallet er om man er i jobb eller ikke, og koder å være i jobb som 1 og 0 ellers. Hvis man har 5 personer, derav 3 er i jobb får man så: $\bar{y} = \frac{(0+0+1+1+1)}{5} = \frac{3}{5} = 0.6$ som er det samme som 60%.

I regresjon med slike variable er dermed utfallet en andel og dette kalles derfor ofte en «lineær sannsynlighetsmodell». Men det er egentlig en helt vanlig regresjonsmodell. Vi tolker fremdeles på den skalaen y-variabelen er på, som altså er en andel. (Vi skal forresten senere omtale

andeler som estimater på sannsynligheter). En økning i x-variabelen tilsvarer altså en endring, b, i andelen med den egenskapen som er kodet 1 på y-variabelen.

La oss si at vi er interessert i å beskrive kjønnsforskjell i hvorvidt menn og kvinner jobber i privat vs offentlig sektor. Variabelen "private" er en factor-variabel der første kategori er "public", som da blir referansekategorien. R regner da privat sektor som 1 mens offentlig sektor er 0. Koeffisientene vil da uttrykke forskjell i sannsynlighet for å være i privat sektor.

```
lm(private ~ female + ed + age, data = abu89)
```

```
Call:
lm(formula = private ~ female + ed + age, data = abu89)

Coefficients:
(Intercept)       female           ed          age
   2.167030    -0.287998    -0.049017    -0.007274
```

Vi ser her at sannsynligheten for at kvinner jobber i privat sektor er 0.28 lavere enn for menn, dvs. 28 prosentpoeng lavere. Dette er da kontrollert for utdanning og alder, slik at vi kan se bort fra at utdanningsnivå og forskjell i aldersfordeling i dataene kan være grunnen til forskjellene.

Part IV

Del 4: statistisk tolkning

11 Design og tolkning

11.1 Tre nivåer av regresjonanalyse

Richard Berk beskriver i sin lærebok tre nivåer av regresjonsanalyse basert på hvordan dataene ble til. Dette er et godt utgangspunkt som burde klargjøre betydelig, i hvert fall som et først skritt.

11.1.1 Nivå I: Ikke tilfeldig utvalg fra en veldefinert populasjon

Grunnlaget for statistisk tolkning (sannsynligheter, p-verdier og sånn) er at dataene er en tilfeldig realisering av en underliggende sann verdi. Typisk betyr dette bare at man har trukket et tilfeldig utvalg fra en populasjon. Da vil man få et godt mål på f.eks. gjennomsnittsverdi i populasjonen, men på grunn av tilfeldighet vil det være en feilmargin på denne målingen.

Det avgjørende er altså at det finnes en veldefinert populasjon som det kan generaliseres til. Grunnen til å bruke begrepet *veldefinert* er at det må være rimelig spesifisert.

Hvis dataene *ikke* er fra en veldefinert populasjon kalles dette noen ganger for *convenience sample*. Altså, at man gjorde et uttrekk av beleilighetsgrunner, men uten at det var en veldefinert populasjon.

Et eksempel kan være en arbeidsmiljøundersøkelse i en bestemt bedrift. Det skal litt til at disse resultatene skal gjelde utover denne bedriften. Man kan selvsagt argumentere for at erfaringene gjelder med generelt, men en slik slutning vil da hvile først og fremst på disse argumentene - ikke på statistiske utregninger.

Det kan være veldig nyttig å analysere slike data, og det kan bringe innsikt og kunnskaper. Men med slike data gir det ikke mye mening å regne på statistisk usikkerhet. Hvis man ikke skal si noe utover de dataene man har (ikke generalisere), så er det heller ikke denne typen usikkerhet i målingene.

Slike ikke-tilfeldige utvalg kan betraktes nærmest som case-studier. En dataanalyse vil gi oss kunnskaper om de erfaringene som gjøre akkurat der. Størrelsen på datasettet kan gi oss mer pålitelig informasjon om dette caset, men hjelper ikke for å generalisere utover caset.

11.1.2 Nivå II: Tilfeldig utvalg fra en veldefinert populasjon

Tilfeldig utvalg er akkurat det det høres ut som, og er den foretrukne metoden for alle surveyundersøkelser. Teorien bak er at utvalget vil gjenspeile populasjonen, og avvik fra "de sanne verdiene" skyldes tilfeldigheter. Disse tilfeldighetene er grunnlaget for statistisk tolkning ved at vi kan si noe om *samplingfordelingen* (se annet kapittel) og dermed har grunnlag for å regne på standardfeil og p-verdier osv. Med andre ord: generalisering til populasjonen.

Så er det viktig å påpeke at forutsetningen her er at det må være et utvalg fra en *veldefinert* populasjon. Hvis vi ikke vet hvem resultatene generaliserer til, så blir det jo tullete, og vi er egentlig på nivå 1.

11.1.3 Nivå III: Estimering av kausale effekter

Fra et teknisk perspektiv er det ingenting som skiller studier av eksperimenter fra observasjonsstudier. De samme regresjonsmodellene kan estimeres og de samme utregningene av usikkerhet. Hva som bestemmer tolknigen (og hvorvidt modellspesifikasjonen er rimelig etc) avhenger av forskningsdesignet. Kort sagt kreves det et eksperiment. Hvis man har en *treatment*-gruppe og en kontrollgruppe, så vil β beskrive forskjellen mellom disse gruppene som i andre typer data. Det som gir β en *kausal* tolkning er om dataene tilfredsstiller kravene til et eksperiment.

Vi kan også regne inn kvasi-eksperimentelle studier eller naturlige eksperimenter her. Enten er disse studiene gode nok til å kvalifisere til å tolke som kausaleffekter - eller så er de det ikke, men da hører de hjemme på nivå II.

11.1.4 Mot et nivå IV?

I Berk sin fremstilling av de tre nivåene får man en følelse av at den vitenskapelige verdien øker ved hvert nivå. Mange vil da også mene akkurat det. Men logisk sett er det litt mer tvetydig enn som så.

Vi har snakket om to dimensjoner: kausalitet (ja/nei) og generalisering (ja/nei). Dette gir fire logisk mulige kombinasjoner.

Table 11.1: Nivåer av regresjonsanalyse og hvor vanlige de er

	Ikke tilfeldig utvalg fra veldefinert populasjon	Tilfeldig utvalg fra veldefinert populasjon
Deskriptiv	Overraskende mange	Det aller meste
Kausal	Mye, men burde nok vært mer	Ganske sjelden

Jeg har ingen empiri for å si hvor vanlig hver enkelt type analyse er. Men jeg tror det nokså omtrentlige angivelsen i tabellen er ganske riktig, basert på egen erfaring fra studier jeg har lest og presentasjoner jeg har sett.

I Berk sin fremstilling er Nivå III hele nederste rad, men da er det altså ikke gjort skille mellom om resultatene kan generaliseres videre eller ikke. Et slik skille bør man nok gjøre.

Eksperimenter omtales noen ganger - og i noen fagmiljøer - som *gullstandarden*. Men altså: ethvert eksperiment kan ikke være en gullstandard, ikke engang når formålet er å estimere kausaleffekter. Nivå IV er i så fall det vi ser etter, da nivå III har begrenset gyldighet. I tilsvarende ånd omtaler Berk eksperimenter som *bronsestandarden*, med den begrunnelse at det i praksis ikke er noe på palleplassene sølv og gull.

11.2 Hva er poenget her, egentlig?

Poenget er at tolkning av resultatene handler vel så mye om hvordan dataene har *blitt til* som hvordan de er *analysert*. Hvis du har data på nivå I, så finnes det ingen statistiske krumspring du kan gjøre som løfter det til et annet nivå. Det samme gjelder nivå II og nivå III. Du kan fremdeles gjøre svært så nyttige og informative analyser på det nivået du har data på. De statistiske analyseteknikkene er det ellers ikke så stor forskjell på.

11.3 Hva med sosiologisk teori?

Vi bruker kvantitative metoder til å beskrive statistiske sammenhenger. Men vi er jo ikke interessert i *variablene* som sådan. Poenget er å beskrive sosiale fenomener. Å si hva det betyr krever imidlertid en teoretisk tolkning. Å teste om et estimat er statistisk signifikant er ikke det samme som å teste en teori, selv om begge deler kan omtales med ordet "test".

- Gitt et fenomen (beskrevet med statistikk), hvordan kan vi forklare det?
- Hva skjer med fenomenet vi er interessert i hvis vi gjør en intervensjon?
- Gitt en teori, er observasjonsdata (dvs. statistikk) konsistent med teorien?
- Gitt en teori, kan vi sjekke om observasjonsdata (dvs. statistikk) *ikke* er konsistent med teorien?

Den første varianten handler om å først beskrive - gjerne eksplorerende - og så bruke teori til å forklare hvorfor det er slik. Det følger gjerne flere analyser som gjør beskrivelsen mer nyanserte og utforsker ulike muligheter.

Den andre varianten handler om å måle effekter, gjerne et naturlig eksperiment eller et felteksperiment. Hvis man ønsker å vite hva *effekten* av et tiltak er, så må man endre noe slik at man kan observere resultatet. Randomisering handler om å håndtere seleksjon.

Den tredje varianten er en konfirmerende strategi: En teori bør jo være konsistent med hvordan verden ser ut. Det er viktig å sjekke at dette er tilfellet. Hvis det er konsistent vet man jo det, men det er ikke en *test* av noe som helst fordi det kan jo være alternative teorier som forklarer minst like godt.

Den fjerde varianten krever at teorien gir en *empirisk forventning* - helst som er motstridende med en annen teori. Hvis det kan vises empiriske mønster som er inkonsistente med teori, så må teorien enten forkastes eller i det minste justeres. Hvor mye vil være helt avhengig av den teoretiske påstanden.

Statistiske tester, med p-verdier og konfidensintervaller, brukes til å skille mellom tilfeldig variasjon og systematisk variasjon på en tilsvarende måte for alle strategier.

12 Statistisk tolkning

Det vi omtaler som *statistisk tolkning* eller *statistisk inferens* handler om å skille systematikk fra støy. Altså: håndtering av usikkerhet ved estimeringen. Dette er relevant for analyser på nivå II og III (se forrige kapittel). På nivå II handler det om å *generalisere* til en veldefinert populasjon, mens det på nivå III handler om å si om en kausal effekten kan skilles fra tilfeldig støy. De statistiske teknikken er imidlertid de samme. I praksis handler dette om (på dette nivået) følgende:

1) standardfeil
2) konfidensintervall
3) p-verdi

Disse tre henger nøye sammen og er forskjellige uttrykk for *feilmarginen* ved et estimat. Her skal vi ikke gjennomgå begrunnelsene og det teoretiske grunnlaget for hvordan dette fungerer, men hoppe rett til det praktiske. En skikkelig forklaring følger fra *sentralgrenseteoremet*[1].

Litt enkelt kan vi si at det hvis man gjør en studie på en ordenlig måte, så er ganske sannsynlig at man får en estimat som er lik den sanne verdien. Men det er også ganske *lite sannsynlig* at man får et estimat som er *nøyaktig* lik den sanne verdien. Vi må regne med at estimatet avviker noe på grunn av *tilfeldigheter*! Det er veldig nyttig å vite noe om hvor mye feil man kan forvente å få av tilfeldige grunner. Altså: hva er feilmarginen til den metoden vi bruker til å estimere?

Denne feilmarginen avhenger først og fremst av hvordan undersøkelsen er gjennomført. Utvalgsprosedyren er det viktigste: tilfeldig trukket utvalg er det mest grunnleggende momentet. Hvis det er systematiske skjevheter i dataene, så vil estimatet bli systematisk skjevt på måter vi ikke så lett kan håndtere med statistiske teknikker.

Dernest avhenger feilmarginen av utvalgsstørrelsen. Det er rett og slett slik at større data gir sikrere estimatet. Hvis utvalget består av 10 personer, så vil estimatet være langt mer usikkert enn hvis det hadde bestått av 5000 personer. Selv om det finnes en statistisk forklaring på dette, så er det relativt intuitivt å forstå. I utregninger vil standardfeilen bli mindre hvis antall observasjoner er større.

[1] Hvis detter ukjent stoff for det kan du se en som er dekket i f.eks. Moore Notz og Fligner (2021) *The basic practice of statistics*, kapittel 15, etterfulgt av forlengelsen til konfidensintervall og statistiske tester i kapittel 16 og 17.

Til sist avhenger feilmarginen av en grense vi selv setter, som gjerne kalles *konfidensgrad*. Denne setter vi selv, men det er vanlig å sette denne til 95%. Det gjør at man noen ganger sier "95% sikker", hvilket er en nokså sleivete måte å si det på, men ikke helt galt under visse forutsetninger som vi kommer tilbake til.

12.1 Estimater og feilmarginer

12.1.1 Estimat

La oss si at du ønsker å si noe om gjennomsnittet i *populasjonen*, men har bare data om et tilfeldig *utvalg* fra denne populasjonen. Når du da regner ut gjennomsnittet i utvalget er det din beste gjetning på hva gjennomsnittet er i populasjonen. En slik gjetning kaller vi et *estimat*.

12.1.2 Standardfeil

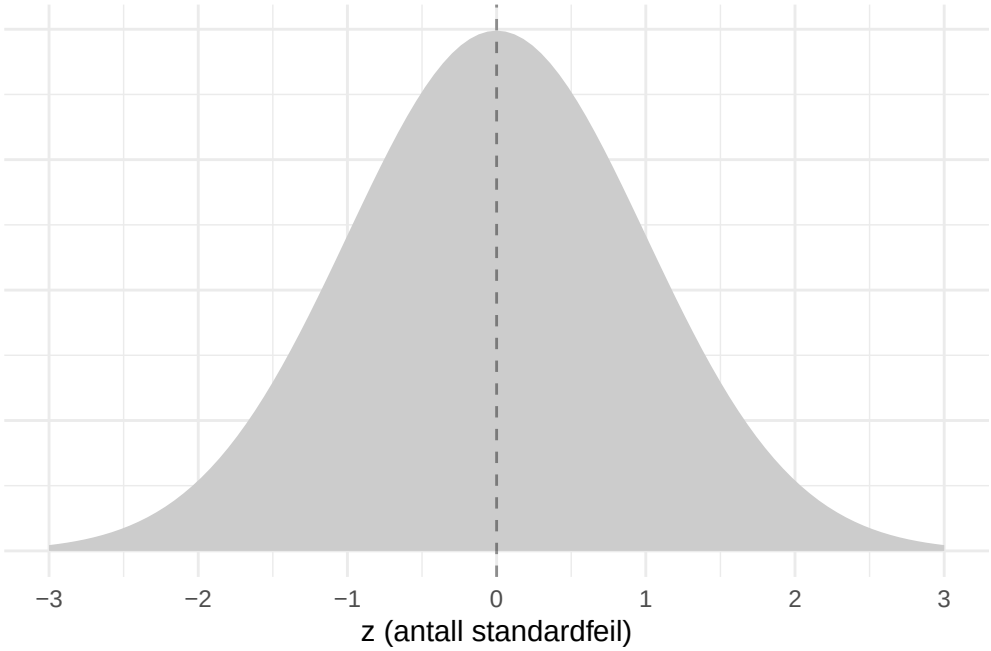
Standardfeilen uttrykker usikkerheten ved *estimatet*. Standardfeilen til estimatet er et mål på usikkerheten ved målemetoden. Usikker målemetode gjør at feilen kan være større.

Ordet standard*feil* er lett å blande sammen med standard*avvik*, så la oss ta det med det samme. Standardavviket beskriver variasjon i data, f.eks. hvis man vil beskrive hvordan personers inntekt varierer rundt gjennomsnittet. Standardavviket beskriver altå variasjon i *data*.

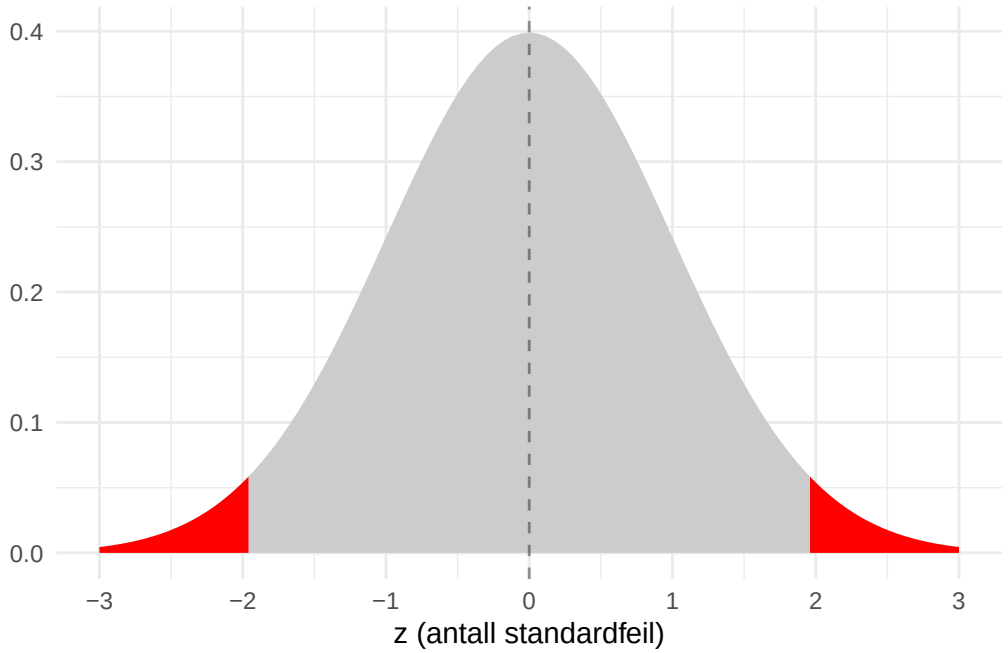
Standard*feilen* beskriver derimot ikke data, men sannsynlighetsfordelingen for hvordan vi forventer at **estimatet** vil kunne avvike fra den sanne verdien på grunn av tilfeldigheter.

Sentralgrenseteoremet sier at estimatet på et gjennomsnitt vil ha tilfeldige feil som er *normalfordelt*, og dermed kan vi bruke normalfordelingen til å si noe om usikkerheten ved estimatet. Dette er forsøkt illustrert nedenfor der x-aksen viser hvor mye estimatet kan avvike fra den sanne verdien, mens kurven viser hvor sannsynligsfordelingen til avviket. Den stiplede linjen viser den sanne verdien. Når x-aksen er avvik fra sanne verdien, så vil altså $x = 0$ bety null avvik fra sanne verdien: helt riktig estimat.

Altså: det er aller mest sannsynlig å få et estimat som ligger nærme den sanne verdien, men litt avvik (større eller lavere estimat) er nesten like sannsynlig. Jo lengre til hver av sidene man går (større feil), jo mindre sannsynlig er det å få et slikt estimat.



Skalaen på x-aksen er z, som i denne sammenheng kan tolkes som antall standardfeil. Den følger en standard normalfordeling som har kjente og faste egenskaper. Vi vet f.eks. at andelen *nedenfor* -1.96 er 0.025, og det samme gjelder *ovenfor* 1.96. Dette er illustrert i figuren nedenfor. Det er altså 0.05 (dvs 5%) sannsynlighet for å få et estimat som ligger 1.96 standardfeil unna den sanne verdien. Motsatt er sannsynligheten for å få et estimatet innenfor intervallet mellom -1.96 og 1.96 tilsvarende 95%. Dette er grunnlaget for det vi kaller *konfidensintervall*.



12.1.3 Konfidensintervall

Hvis man ønsker å være "95% sikker", så bruker man altså et 95% konfidensintervall. Det er ikke noe magisk ved akkurat 95% og er primært blitt en norm. Grunnen er bare at det skal være en ganske lav sannsynlighet for at estimatet skyldes tilfeldig variasjon.

Til ethvert estimat er knyttet en feilmargin som uttrykkes ved $z \times se$, der se er forkortelse for standardfeil (engelsk: "standard error"). z er et tall som er knyttet til grad av usikkerhet ved feilmarginen. En feilmargin basert på 95% konfidensgrad er dermed $1.96 \times se$. Hvis man så tar denne feilmarginen til hver side, så gir det samme som illustrert i figuren ovenfor.

Et konfidensintervall er altså bare å ta hensyn til feilmarginen til hver side av estimatet. Når vi bruker dette i praksis baserer vi oss på denne normalfordelingen og trenger bare å få regnet ut standardfeilen i tillegg.

```
[1] 91.0712
```

Hvis man f.eks. har estimert gjennomsnittlig timelønn til å være 90.2 og standardfeilen er 0.5. Da blir 95% konfidensintervallet som følger:

$$90.1 \pm 1.96 \times 0.47 = [89.2, 91.1]$$

Dette betyr at vi har brukt en målemetode som har en feilmargin som gjør at vi kan være 95% sikker på at den sanne verdien ligger innenfor dette intervallet.

12.1.3.1 Er man egentlig "95% sikker"?

Det sies ofte at feilmarginen uttrykker hvor sikker man er. Det er jo ikke helt riktig - eller det er riktig under noen spesielle forutsetninger om hva man mener med "sikker". La oss derfor ta dette med en gang.

Et 95% konfidensintervall er vårt anslag på hvor god vår *målemetode* er. Vi har jo regnet ut f.eks. et gjennomsnitt og det er jo greit nok. Usikkerheten kommer fra utvalgsprosedyren og variasjonen i data.

Vi vet ikke hvorvidt vårt estimat ligger nærme eller langt unna den sanne verdien. Det vi derimot vet noe om er påliteligheten i den metoden vi har brukt. Det viktigste her er altså tilfeldig utvalg, og hvis utvalget ikke er tilnærmet tilfeldig trukket, så bryter det hele sammen.

Når man sier at konfidensintervallet uttrykker at man er "95% sikker" på at den sanne verdien ligger i det intervallet mener man da følgende: Man har brukt en *metode* (dvs utvalg og utregninger og det hele) som har en *feilmargin*. Denne feilmarginen er slik at hvis man gjorde estimeringen (altså nytt utvalg hver gang) på samme måte svært mange ganger (f.eks. uendelig mange ganger), så ville 95% av resultatene ligget innenfor et slikt intervall.

Man gjør selvsagt ikke samme undersøkelse tusenvis av ganger, så dette er en hypotetisk tanke. Man man kan også tenke seg at mange ulike forskere gjør en tilsvarende studie og får litt forskjellige resultat. Disse resultatene vil (i teorien) fordele seg som en normalfordeling rundt den sanne verdien. Noen ganske få vil ligge langt unna sannheten.

12.1.4 T-testen

Hvis man skal sammenligne to grupper, så vet vi i utgangspunktet at dataene fra et utvalg antakeligvis vil vise at de ikke er helt like på grunn av tilfeldig variasjon. Det kan altså være at gruppene er like i virkeligheten, bare at dataene våre tilfeldigvis ble litt forskjellige. Det må vi jo regne med, men det er begrenset hvor forskjellig vi kan forvente at gruppene er på grunn av rene tilfeldigheter.

Se igjen på figuren over av normalfordelingen i omtalen av konfidensintervall. Gitt at det ikke er noen sann forskjell mellom gruppene, så vil vi forvente at estimatet ligger *innenfor* en viss feilmargin. Det betyr at en observert forskjell i dataene som ligger *innenfor* denne feilmarginen vil være konsistent med at forskjellene bare skyldes tilfeldig variasjon. Motsatt: hvis estimatet ligger *utenfor* denne feilmarginen, ja da kan vi si at det ikke er konsistent med dette utgangspunktet om at forskjellene bare skyldes tilfeldig variasjon.

En av de meste brukte statistiske testene i praksis er "t-testen". Du kan tenke på det som en **beslutningsregel**: hva skal til for at du skal bestemme deg for å tro at forskjellen *ikke skyldes tilfeldigheter*? Standardfeil og feilmarginer er det samme som før, så du må bare bestemme deg for hvor stor feilmargin du er villig til å operere med. Hvis estimatet på en differanse er *større* enn feilmarginen, da forkastes hypotesen om at forskjeller skyles tilfeldigher. Altså: forskjellene i data må da skyldes noe mer systematisk.

Så i utgangspunktet så må du altså ta stilling til om du mener det er en forskjell - eller ikke. Du kan ikke konkludere med at det "kanskje er en forskjell", men må ta et valg. Derfor kaller vi gjerne dette for hypotesetesting i en litt snever forstand. Det er bare to mulige hypoteser:

H_0: Det er egentlig ingen forskjell mellom gruppene, og forskjell i *dataene* skyldes bare tilfeldig variasjon. (Nullhypotesen). H_A: Det er faktisk en forskjell mellom gruppene, og forskjellen i *dataene* er for stor til av det er sannsynlig at det skyldes tilfeldig variasjon. (Alternativ hypotese).

12.1.4.1 Formler og slikt

T-testen i prinsippet en sammenligning mellom estimatets størrelse og standardfeilen til estimatet. Vi kan skrive det som følger:

$$\frac{\mu}{SE(\mu)} = t$$

Eller sagt på en annen måte:

$$\frac{estimat}{standardfeil} = t$$

Det betyr at t-verdien egentlig bare er forholdstallet mellom estimatet og standardfeilen. Intuitivt kan man vel forstå at hvis usikkerheten bør være mindre enn estimatet. Altså: hvis du har estimert en forskjell i timelønn mellom to grupper, og feilmarginen til dette estimatet er større enn forskjellen, ja, da er det vanskelig å lære noe særlig fra det estimatet.

Verdien t tilsvarer verdien z som vi nevnte i forbindelse med konfidensintervall. Så hvis t er større enn z, så ligger estimatet *utenfor* konfidensintervallet.

Tolkningen av t-verdien brukes gjerne som en en beslutningsregel: Ja/Nei. Litt firkantet, med andre ord. Men t-verdien er også knyttet til normalfordelingen på samme måte som nevnt ovenfor i forbindelse med konfidensintervaller. Ethvert mulig resultat er knyttet til en viss sannsynlighet for at det skal skje ved en tilfeldighet.

12.1.4.2 P-verdi

Tolkningen av p-verdien er i hvilken grad det er sannsynlig å få det observerte resultatet *ved en tilfeldighet* hvis NULL-hypotesen er riktig. Dette høres ganske pussig ut. Tanken er at man nesten alltid vil observere noe forskjell fra null, og det kan skje ved en tilfeldighet. Hvis null-hypotesen er riktig er det mindre sannsynlig at vi observerer en veldig stor forskjell. Men hvor stor forskjell er det, egentlig? Løsningen er å se avstanden fra null i lys av standardfeilen. Hvis man bruker en usikker målemetode, så er det mer sannsynlig å observere en stor forskjell ved tilfeldigheter enn om man bruker en veldig nøyaktig målemetode.

I praksis: Tenk at du observerer en stor forskjell mellom to grupper. Med "stor" mener vi f.eks. at forskjellen er over dobbelt så stor som standardfeilen. Da får vi en p-verdi som er $p < 0.05$. Da kan vi si at hvis nullhypotesen er sann, så er det lite sannsynlig at vi ville fått et slikt resultat på grunn av tilfeldigheter.^(Hvis vi ønsker være pinlig korrekte kan vi også si noe slikt som at hvis man gjorde målingen tusenvis av ganger, så ville 5% av resultatene ligge så langt unna null (eller lengre).)

Så er logikken videre at vi som hovedregel ikke tror på resultater som er usannsynlige. Så i stedet for å holde fast på nullhypotesen velger vi i stedet å tro på den alternative hypotesen.

12.2 Kan man velge fritt konfidensgrad?

Det er ingenting magisk med tallet 1.96 eller $p < 0.05$. Det er en konvensjon. Konfidensgrad er nemlig noe *du* velger. All tolkning av "statistisk signifikans" er basert på en gitt konfidensgrad.

Problemet oppstår hvis du først ser på resultatene og så velger en konfidensgrad som passer til det du har mest lyst til å konkludere med. Det er rett og slett juks. For at du skal velge en annen konfidensgrad må du si det høyt og tydelig *før* du gjennomfører undersøkelsen - og da må du faktisk etterleve det når resultatene foreligger. Du bør også kunne argumentere selvstendig for en annen konfidensgrad, altså før resultatene foreligger. Dette innebærer at det ikke er godt nok å si det høyt ut i luften der du sitter alene for deg selv. Det må *pre-registeres* på et offentlig sted, f.eks. osf.io eller tilsvarende websider.

Dette er egentlig en mye større diskusjon, men verd å være obs på. Du *kan* velge konfidensgrad, men i så fall må du gjøre det på en ordenlig måte hvis du vil bli tatt seriøst av andre. Vi skal ikke drive med cherry-picking av resultater og konklusjoner!

12.3 Statistiske tester generelt

Det finnes en hel haug av statistiske tester. Prinsippet er gjerne variasjoner av t-testen og har disse komponentene:

1. en nullhypotese og et alternativ
2. en *statistikk*, altså et måltall som er et avstandsmål mellom observert resultat og hva man forventer under nullhypotesen
3. en statistisk modell for samplingfordelingen som sier noe om fordelingen av tilfeldige feil
4. en uttalt beslutningsregel for konklusjonen. Et vanlig mål er at hvis $p < 0.05$, så forkastes nullhypotesen.

Du har sikker lært om χ^2 testen for krysstabeller. Den er forskjellig på mange måter fra t-testen, men logikken er tilsvarende: χ^2 er et avstandsmål for hva vi forventer gitt hypotesen om ingen forskjell. Hvis resultatet fra dataanalysen er for langt unna dette, så beslutter vi å tro at forskjellen skyldes systematikk.

Part V

Del 5: Setter det hele sammen

13 Statistikk i praksis

```
library(tidyverse)
library(gtsummary)
```

Statistiske analyser innebærer å analysere data og vurdere usikkerhet. En kjerneoppgave er å *sammenligne*. Enten mellom grupper eller på ulike steder langs en kontinuerlig skala. Når vi sammenligner Usikkerheten i sammenligningen uttrykkes ved p-verdier og konfidensintervaller.

13.1 Deskriptiv statistikk

Når man har en tabell med deskriptiv statistikk fordelt på grupper, så gjør man jo en sammenligning av disse gruppene på de aktuelle variablene. Da kan man bare legge til en statistisk test for denne sammenligningen. I følgende eksempel brukes `tbl_summary` med tilhørende `add_difference`. I første omgang tar vi bare med kontinuerlige variable. Resultatet blir tilsvarende som i det tidligere kapittelet for deskriptiv statistikk, men her legges det til tre kolonner: forskjellen i gjennomsnitt, konfidensintervallet og p-verdi fra en t-test.[^](Legg merke til fotnoten som spesifiserer "Welch two sample t-test". Dette er den vanlig t-testen. Den opprinnelige "Student's t-test" forutsetter lik varians i begge grupper, noe som Welch t-test ikke gjør. Vi kaller det bare for t-test. Dette bare til oppklaring.)

```
theme_gtsummary_mean_sd()
abu89 %>%
  #select(-io_nr) %>%
  select(female, time89, ed,  fexp, age) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
    tbl_summary(by = female,
                label = list(klasse89 = "Klasse"),
                type = list(ed ~ "continuous"),
                missing = "no") %>%
  add_difference()
```

Characteristic	**Kvinner**, N = 1,934	**Menn**, N = 2,193	**Difference**	**95% CI**	**p-value**
Gjennomsnittlig timelønn 1989	79 (24)	100 (32)	-21	-23, -19	<0.001
År utdanning	2.38 (2.40)	2.96 (2.66)	-0.58	-0.74, -0.43	<0.001
Bedriftserfaring	0.83 (0.81)	1.05 (0.97)	-0.22	-0.27, -0.16	<0.001
Alder	40 (13)	40 (12)	-0.17	-0.93, 0.58	0.7

Legg merke til at kollonnen "Difference" er forskjellen i gjennomsnitt i de to gruppene, og konfidensintervallet gjelder for denne differansen. Den gjennomsnittlige forskjellen i timelønn for menn er altså 21 kroner høyere enn for kvinner, men når vi tar feilmarginen med i beregningen er det rimelig å si at den ligger mellom 19 og 23 kroner høyere for menn enn for kvinner, siden et 95% konfidensintervall tilsier det.

```
abu89 %>%
  select(female, time89, ed,  fexp, age) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
    tbl_summary(by = female,
                label = list(klasse89 = "Klasse"),
                type = list(ed ~ "continuous"),
                missing = "no") %>%
  add_p()
```

Characteristic	**Kvinner**, N = 1,934	**Menn**, N = 2,193	**p-value**
Gjennomsnittlig timelønn 1989	79 (24)	100 (32)	<0.001
År utdanning	2.38 (2.40)	2.96 (2.66)	<0.001
Bedriftserfaring	0.83 (0.81)	1.05 (0.97)	<0.001
Alder	40 (13)	40 (12)	0.7

For kategoriske variable bruker man ikke en t-test, men en test som omtales som χ^2 test (uttales som "kji-kvadrat test").[1]

```
abu89 %>%
  select(female, klasse89, promot, private) %>%
```

[1]Denne gir identisk resultat som z-test for andeler.

```
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
    tbl_summary(by = female,
                label = list(klasse89 = "Klasse"),
                missing = "no") %>%
  add_p()
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**Kvinner**, N = 1,934	**Menn**, N = 2,193	**p-value**
Klasse			<0.001
I Øvre serviceklasse	74 (3.9%)	254 (12%)	
II Nedre serviceklasse	555 (29%)	626 (29%)	
III Rutinefunksjonærer	986 (52%)	262 (12%)	
V-VI Faglærte arbeidere	46 (2.4%)	602 (28%)	
VIIa Ufaglærte arbeidere	244 (13%)	393 (18%)	
Noen gang forfremmet			<0.001
NEI	1,308 (68%)	1,260 (57%)	
JA	626 (32%)	933 (43%)	
Privat sektor			<0.001
Public	1,016 (53%)	586 (27%)	
Private	918 (47%)	1,607 (73%)	

Det kan også settes sammen i en felles tabell.

```
abu89 %>%
  select(-io_nr) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
    tbl_summary(by = female,
                label = list(klasse89 = "Klasse"),
              type = list(ed ~ "continuous"),
              missing = "no") %>%
  add_overall() %>%
  add_p()
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**Overall**, N = 4,127	**Kvinner**, N = 1,934	**Menn**, N = 2,193	**p-value**
Gjennomsnittlig timelønn 1989	90 (30)	79 (24)	100 (32)	<0.001
År utdanning	2.69 (2.56)	2.38 (2.40)	2.96 (2.66)	<0.001
Alder	40 (12)	40 (13)	40 (12)	0.7
Klasse				<0.001
I Øvre serviceklasse	328 (8.1%)	74 (3.9%)	254 (12%)	
II Nedre serviceklasse	1,181 (29%)	555 (29%)	626 (29%)	
III Rutinefunksjonærer	1,248 (31%)	986 (52%)	262 (12%)	
V-VI Faglærte arbeidere	648 (16%)	46 (2.4%)	602 (28%)	
VIIa Ufaglærte arbeidere	637 (16%)	244 (13%)	393 (18%)	
Noen gang forfremmet				<0.001
NEI	2,568 (62%)	1,308 (68%)	1,260 (57%)	
JA	1,559 (38%)	626 (32%)	933 (43%)	
Bedriftserfaring	0.95 (0.91)	0.83 (0.81)	1.05 (0.97)	<0.001
Privat sektor				<0.001
Public	1,602 (39%)	1,016 (53%)	586 (27%)	
Private	2,525 (61%)	918 (47%)	1,607 (73%)	

Part VI

Del 6: Omkoding og datahåndtering

14 Datahåndtering med *Tidyverse*

Et helt vanlig problem er at data i den virkelige verden ofte er ganske grisete. Det er mye rot, manglende verdier, andre kategorier enn du er interessert i osv. Kort sagt: dataene er ikke helt slik de bør være for at du skal kunne gjøre den analysen du har tenkt til. Altså må du fikse dataene før du får estimert det du har tenkt. Enten dette gjelder grafikk, tabeller eller regresjonsmodeller.

Tidyverse er en rekke funksjoner som til sammen utgjør et programmeringsspråk for datahåndtering. Dette er en variant av R, som vi kan si er en dialekt av R. Det finnes andre dialekter, men vi anbefaler sterkt å lære tidyverse godt først.

De grunnleggende *verbene* gjør noe med dataene, så da er altså dataene *substantivene*.

14.1 Lage ny variabel: `mutate`

Alle verbene i tidyverse starter med å angi hvilket objekt man skal gjøre noe med, altså datasettet.

Her er et eksempel der man lager en ny variable som summen av eksisterende variablene x og z.

```
nyttobjekt <- mutate(dinedata, nyvariabel = x + z)
```

Her er et eksempel der man lager to variable samtidig der den andre er x delt på z.

```
nyttobjekt <- mutate(dinedata, nyvariabel = x / z,
                     nyvariabel2 = x + z)
```

14.2 Rørlegging: Hva i alle dager betyr %>% ??

Symbolet **%>%** kalles in "pipe" eller på norsk: rørlegging. Det betyr at det som står til venstre flyttes over til høyre. Eller sagt på en annen måte betyr det: "Gjør deretter følgende". Vi vil bruke denne syntaxen konsekvent fra nå når vi introduserer de ulike "verbene".

```
nyttobjekt <- dinedata %>%
  mutate(nyvariabel = x / z,
         nyvariabel2 = x + z)
```

Denne koden sier følgende, linje for linje:

- lag en kopi av `dinedata` og lagre det i `nyttobjekt` ^*deretter gjør du følgende:*^
- lag de nye variablene `nyvariabel` som får verdier fra variablene x delt på y
- og `nyvariabel2`som summen av x og z

14.3 Beholde og slette variable: `select`

14.4 Aggregere: `summarise`

14.5 Grupperte utregninger: `group_by`

14.6 Sette det hele sammen

15 Omkoding av variable

```
library(haven)       # Importere data fra SAS, SPSS og Stata
library(tidyverse)   # Pakker for generell datahåndtering og grafikk
library(labelled)    # Håndtering av variable med labler, importert fra annen software
library(forcats)     # Lettere omkoding av faktorvariable
library(gtsummary)
```

I denne delen skal vi bruke et uttrekk NorLAG som ikke er ryddet skikkelig i forkant. Kategoriske variable er riktignok gjort om til factor-variable, men det er beholdt ulike typer missing-verdier som vi ellers ville luket bort.

```
Rows: 20,892
Columns: 2
$ iokjonn <fct> Mann, Mann, Kvinne, Kvinne, Kvinne, Kvinne, Kvinne, Kvinne, Kv~
$ wr117zz <fct> NA, filter: ikke i arbeid, filter: jobber deltid, filter: jobb~
```

15.1 Endre variabelnavn med `rename` og `mutate`

15.2 Kontinuerlige variable

Å omkode kontinuerlige variable er i utgangspunktet det enkleste. Dette er tall og man kan gjøre normale regneoperasjoner på dem.

15.3 Tekstvariable (strings)

15.4 Factorvariable

R har en egen variabeltype for kategoriske variable som kalles "factor". I utgangspunktet er kategoriske variable mer å regne som tekstvariable enn som tall, men i en del beregninger vil softwaren bruke numeriske verdier uansett. Hvis man gjør om en tekst-variabel til en factor-variabel beholdes teksten, men kategoriene numeriske verdier 1, 2, 3, … osv. Disse tallene

kan du tenke på som rekkefølgen på kategoriene. For kategoriske variable er det jo ikke noen egentlig rekkefølge, men det kan være grunner til å foretrekke rekkfølgen av andre grunner som vi kommer tilbake til.

Hvis variabelen er ordnet f.eks. på en skala fra 1 til 5 eller annen naturlig rekkefølge[1], så kan man også angi dette.

15.4.1 Få oversikt over factor-levels med `levels()`

```
levels(norlag_ex$wr117zz)
```

```
[1] "Nei"
[2] "Ja"
[3] "filter: jobber deltid"
[4] "filter: selvstendig næringsdrivende (NorLAG3 inkl frilanser/annet)"
[5] "filter: ikke i arbeid"
[6] "vil ikke svare"
[7] "vet ikke"
[8] "mangler data"
[9] "Deltok ikke i runden"
```

15.4.2 Enkel omkoding med `fct_recode()` og `fct_collapse()`

15.4.3 Endre rekkefølgen på faktorene med `fct_reorder()`

15.5 Betinget omkoding med `ifelse()` og `case_when()`

Du kan lære mer om effektiv håndtering av kategoriske variable med forcats-pakken, som er en del av "tidyverse".

15.6 Factorvariable med skikkelig lang tekst

Når man omkoder en variabel må man skrive hele tekstverdien man ønsker endre, og det må være *nøyaktig* likt stavet. Særlig i survey-data vil disse tekststrengene kunne være lange og det gir jo større muligheter for å skrive feil og du kan få andre resultater enn forventet. Det kan også være vanskelig å finne feilen i lange tekststrenger! Så det er altså noe hærk. Kan man gjøre dette på en lurere måte? Minst mulig tårer? Ja, selvsagt.

[1] Noen ganger kalles slike variable å være på "ordinalnivå"

Vi jobber normalt med factorvariable for kategoriske variable. I NorLAG er variabelen wr117zz svar på et spørsmål om "Mulighet for å redusert arbeidstid (deltid)". Når denne variabelen er gjort om til factor kan man se hvilke verdier variabelen har med bruke av funksjonen *levels()* slik:

```
levels(norlag_ex$wr117zz)
```

```
[1] "Nei"
[2] "Ja"
[3] "filter: jobber deltid"
[4] "filter: selvstendig næringsdrivende (NorLAG3 inkl frilanser/annet)"
[5] "filter: ikke i arbeid"
[6] "vil ikke svare"
[7] "vet ikke"
[8] "mangler data"
[9] "Deltok ikke i runden"
```

```
table(norlag_ex$wr117zz)
```

```
                                                                    Nei
                                                                   1360
                                                                     Ja
                                                                   4146
                                                  filter: jobber deltid
                                                                   1964
filter: selvstendig næringsdrivende (NorLAG3 inkl frilanser/annet)
                                                                   1171
                                                  filter: ikke i arbeid
                                                                   6238
                                                         vil ikke svare
                                                                      9
                                                               vet ikke
                                                                    382
                                                          mangler data
                                                                     67
                                                   Deltok ikke i runden
                                                                      0
```

La oss si at vi vil kode om slik at vi får en variabel som bare er om vedkommende har mulighet til å jobbe deltid eller ikke. De som allerede jobber deltid har jo åpenbart mulighet til det, så

de skal kodes om til "Ja". De andre kategoriene er egentlig grunner til at det mangler data, så de skal settes til NA. En mulighet er da å omkode som følger:

```
norlag_omkodet <- norlag_ex %>%
  mutate(redarbtid = replace(wr117zz, wr117zz == "filter: jobber deltid", "Ja"),
         redarbtid = replace(redarbtid, redarbtid == "filter: selvstendig næringsdrivende
         redarbtid = replace(redarbtid, redarbtid == "filter: ikke i arbeid", NA),
         redarbtid = replace(redarbtid, redarbtid == "vil ikke svare", NA),
         redarbtid = replace(redarbtid, redarbtid == "vet ikke", NA),
         redarbtid = replace(redarbtid, redarbtid == "mangler data", NA),
         redarbtid = replace(redarbtid, redarbtid == "Deltok ikke i runden", NA)) %>%
  droplevels()

norlag_omkodet %>%
  select(redarbtid) %>%
  gtsummary::tbl_summary()
```

Characteristic	**N = 20,892**
redarbtid	
Nei	1,360 (18%)
Ja	6,110 (82%)
Unknown	13,422

Dette funker, men blir ganske mye tekst å skrive, og da kan man altså lett skrive feil. Husk at faktornivåene må angis helt nøyaktig slik de er skrevet! Merk at den siste funksjone, `droplevels`, bare fjerner faktor-levels som ikke er i bruk.

I output for faktor-levels angir klammeparentesen gir rekkefølgen på disse verdiene. Vi kan bruke denne informasjonen direkte i omkodingen for å unngå å skrive så veldig mye. Når man bruker `levels()` får man en vektor med verdier, og disse kan man altså henvise til med rekkefølgen. Her er et eksempel for bare å bytte ut de som jobber deltid til "Ja":

```
norlag_omkodet <- norlag_ex %>%
  mutate(redarbtid = replace(wr117zz, wr117zz == levels(wr117zz)[3], "Ja")) %>%
  droplevels()
```

Trikset her er altså å bruke `levels` og vise til hvilket nummer i rekkefølgen. Da unngår vi også faren for skrivefeil.

Vi vil også kode om alle de andre verdiene, nummer 4-9 til NA. Det kan vi gjøre på samme måte, men vi behøver ikke skrive en ny linje for hver verdi. Den logiske operatoren `==` kan man bruke når man skal sjekke om to verdier er like. Hvis vi skal se om en verdi er lik en av

flere mulige kan vi bruke `%in%` og så en liste med verdier. `levels` gir en liste med verdier, så da kan vi angi den direkte og alle verdiene 4 til 9 ved å skrive `4:9`. Samlet blir det da slik:

```
norlag_omkodet <- norlag_ex %>%
  mutate(redarbtid = replace(wr117zz, wr117zz == levels(wr117zz)[3], "Ja"),
         redarbtid = replace(redarbtid, redarbtid %in% levels(wr117zz)[4:9], NA)) %>%
  droplevels()

memisc::codebook(norlag_omkodet$redarbtid)
```

```
================================================================================

   norlag_omkodet$redarbtid

--------------------------------------------------------------------------------

   Storage mode: integer
   Factor with 2 levels

   Levels and labels     N Valid Total

    1 'Nei'           1360  18.2   6.5
    2 'Ja'            6110  81.8  29.2
   NA                13422        64.2
```

15.7 Spesielle problemstillinger ved veldig mange kategorier

For disse eksemplene skal vi bruke et litt annet datasett, nemlig et lite uttrekk fra European Social Survey. Her er det 3 variable: yrkeskode, kjønn og politisk interesse.

```
polit <- read.csv2("data/politics.csv", colClasses = "character")

glimpse(polit)
```

```
Rows: 49,519
Columns: 3
$ isco08  <chr> "3333", "7122", "4221", "4311", "6130", "7212", "5131", "5223"~
$ gndr    <chr> "1", "1", "2", "1", "2", "1", "1", "2", "1", "2", "1", "2", "1~
$ polintr <chr> "3", "2", "4", "3", "2", "2", "4", "3", "3", "4", "2", "2", "1~
```

Vi kan sjekke hvor mange kategorier det er ved å lage en tabell over kodene og se hvor mange det er. Det er imidlertid upraktisk da den tabellen tar veldig mye plass. Koden nedenfor gjør en enklere opptelling ved å trekke ut unike verdier og telle hvor mange det er:

```
antall_koder <- polit %>%
  pull(isco08) %>%   # trekker ut en vektor med kun en variabel
  unique() %>%       # beholder kun unike verdier
  length()           # lengden på gjenværende vektor

antall_koder
```

```
[1] 561
```

Det er altså 561 unike yrkeskoder i datasettet.

15.7.1 Hierarkisk strukturerte tall som tekststrenger

Noen ganger er det hundrevis av verdier. Et slik eksempel er yrkesklassifisering der hver type yrke har en spesifikk kode. Det finnes mange typer yrker, så det er omlag 800 koder. For de fleste typer analyser er dette altfor detaljert og du trenger å gruppere til færre kategorier. SSB har en kodeliste offentlig tilgjengelig. Kort fortalt er det en kode med 4 siffer, der det første sifferet er en grov gruppering, og de etterfølgende sifrene innebærer en økt detaljeringsgrad innenfor grupperingen angitt ved første siffer.

Hvis du skulle omkodet yrker slik som forklart i et tidligere avsnitt om omkoding ville det tatt veldig lang tid, men det ville også være veldig lett å gjøre feil. Det vil rett og slett være et mareritt å de-bugge koden for å finne feil eller kvalitetssjekke. Altså: en slik tilnærming er helt uaktuelt. En langt bedre tilnærming er å bare trekke ut det første sifferet fra koden. Funksjonen `str_sub()` gjør akkurat slike ting ved å angi hvilken del av tekststrengen du vil trekke ut, angitt ved posisjonen du starter ved og slutter ved. Her er det altså første posisjon.

```
polit <- polit %>%
  mutate(occupation = str_sub(isco08, start = 1, end = 1))

polit %>%
  select(gndr, occupation) %>%
  tbl_summary(by = gndr)
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	1, N = 23,020	2, N = 26,499
occupation		
	1,847 (8.0%)	2,947 (11%)
0	5 (<0.1%)	1 (<0.1%)
1	2,185 (9.5%)	1,350 (5.1%)
2	3,665 (16%)	5,018 (19%)
3	2,736 (12%)	3,250 (12%)
4	1,071 (4.7%)	2,862 (11%)
5	2,418 (11%)	5,569 (21%)
6	704 (3.1%)	445 (1.7%)
7	4,162 (18%)	1,158 (4.4%)
8	2,495 (11%)	966 (3.6%)
9	1,732 (7.5%)	2,933 (11%)

15.7.2 Bruke kataloger for kodeverk

Noen ganger har slike lange lister med unike koder en standard gruppering som ikke er hierarkisk, eller man ønsker å lage en annen type gruppering av andre grunner. Et slikt eksempel er gruppering av yrker etter klasseskjemaer, slik som f.eks. **ORDC class schema** utviklet av HISTCLASS-prosjektet. Dette er et kodeskjemae for isco-koder til klasser og en katalog er tilgjengelig fra prosjektets hjemmeside. Det er omtrent 800 unike verdier og de er lagret i et Excel-format med gruppering for hver kode. De første linjene i arket ser ut som følgende:

	A	B	G	H	I	J	K	L	M	N	O	P	Q	R
1	http://www.ilo.org/public/english/bureau/stat/isco/isco88/major.htm													
2														
3														
4							STATA				STATA			
5	ISCO-88 Title EN	ISCO-88 code		ORDC for relativ inntekt	ORDC_YRK	ORDC for relativ inntekt (bortsett fra 3)	Bare basert på yrker				Hele øk.fraksjon er satt til 6			
6	Research and development department managers	1237		1	1	1	replace ordc_yrk=1 if isco88==1237				replace ordc=1 if isco88==1237			
7	Architects, town and traffic planners	2141		1	1	1	replace ordc_yrk=1 if isco88==2141				replace ordc=1 if isco88==2141			
8	College, university and higher education teaching professionals	2310		1	1	1	replace ordc_yrk=1 if isco88==2310				replace ordc=1 if isco88==2310			
9	Education methods specialists	2351		1	1	1	replace ordc_yrk=1 if isco88==2351				replace ordc=1 if isco88==2351			
10	Sociologists, anthropologists and related professionals	2442		1	1	1	replace ordc_yrk=1 if isco88==2442				replace ordc=1 if isco88==2442			
11	Philosophers, historians and political scientists	2443		1	1	1	replace ordc_yrk=1 if isco88==2443				replace ordc=1 if isco88==2443			
12	Philologists, translators and interpreters	2444		1	1	1	replace ordc_yrk=1 if isco88==2444				replace ordc=1 if isco88==2444			

Dette Excel-arket er også lagt til rette for omkoding med bruk av Stata, men du kan se bort fra de siste kolonnene.

Filen kan leses inn med `read_excel()`, der første linje typisk leses inn som variabelnavn. Men i dette tilfellet skal ikke de første linjene brukes og flere kolonner skal heller ikke brukes. Derfor ber vi R droppe de første linjenene, endrer variabelnavn og beholder kun isco-kodene og tilhørende ORDC-grupperingen. Variabelnavn bør ikke inkludere mellomrom, så vi legger til et argument som endrer til gyldige variabelnavn ved å bytte ut mellomrom med punktum.

Vi også spesifiserer `col_types = "text"` for å unngå at tallverdier tolkes som numeriske verdier.

```
list.files("data/")
```

```
 [1] "3_codes_isco88_ordc.xlsx" "abu89.dta"
 [3] "CODES_isco88_ordc.xlsx"   "dat_dict.rds"
 [5] "ESS2016.dta"              "ESS2016.rds"
 [7] "ISCO_ORDC.do"             "norlag.rds"
 [9] "norlag_labelled.rds"      "norlag_panel.csv"
[11] "norlag_panel.dta"         "norlag_panel.Rdata"
[13] "norlag_panel.rds"         "norlag_panel.sas7bdat"
[15] "norlag_panel.sav"         "norlag_panel.xlsx"
[17] "norlag_panel2022.dta"     "ordc.ado"
[19] "ordc.sthlp"               "politics.csv"
```

```
isco <- readxl::read_excel(path = "data/CODES_isco88_ordc.xlsx", skip = 4, .name_repair =
  select(2,9)
head(isco)
```

```
# A tibble: 6 x 2
  ISCO.88.code ORDC_YRK
  <chr>        <chr>
1 1237         1
2 2141         1
3 2310         1
4 2351         1
5 2442         1
6 2443         1
```

Now, we can merge the data with this catalogue. So that every record in the catalogue is merged to each record with the same code. To do this, we use `left_join()`, and store in a new object.

```
isco <- isco %>%
  rename(isco08 = ISCO.88.code)

polit2 <- left_join(polit, isco, by = "isco08")

head(polit2)
```

```
  isco08 gndr polintr occupation ORDC_YRK
1   3333    1       3          3     <NA>
2   7122    1       2          7       10
3   7122    1       2          7       10
4   4221    2       4          4        8
5   4311    1       3          4     <NA>
6   6130    2       2          6       12
```

What happened here is that the recoding happened almost automatically by adding a new column with the new variable.

Now, you can make e.g. a cross-tabulation of social class by gender.

```
polit2 %>%
  select(ORDC_YRK, gndr) %>%
  tbl_summary(by = gndr)
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**1**, N = 29,285	**2**, N = 33,950
ORDC_YRK		
1	448 (2.4%)	364 (2.0%)
10	5,887 (32%)	3,222 (17%)
11	3,757 (20%)	5,563 (30%)
12	1,222 (6.6%)	1,187 (6.4%)
2	1,367 (7.4%)	1,333 (7.1%)
3	30 (0.2%)	13 (<0.1%)
4	561 (3.0%)	1,251 (6.7%)
5	2,000 (11%)	2,669 (14%)
6	1,681 (9.1%)	1,850 (9.9%)
7	180 (1.0%)	267 (1.4%)
8	1,129 (6.1%)	726 (3.9%)
9	134 (0.7%)	105 (0.6%)
996	84 (0.5%)	109 (0.6%)
997	5 (<0.1%)	1 (<0.1%)
Unknown	10,800	15,290

15.7.3 Noen ganger finnes det en pakke

Siden R støtter pakker laget av brukere rundt omkring i verden, så er det alltids en sjanse for at noen har laget noe lurt som fikser akkurat ditt problem. Du kan altså ha flaks.

Når det gjelder omkoding av ISCO-koder til klasseskjemaer, så har noen faktisk gjort dette. Pakken {DIGCLASS} koder om til flere forskjellige klasseskjemaer ganske greit. Se gjerne nærmere på vignetten til pakken.

Denne pakken finnes imidlertid ikke på CRAN i skrivende stund. Derimot finnes den tilgjengelig på nettet. Følgende kode installerer pakken. Hvis du får feilmelding om at du trenger {devtools}, så installer denne pakken først på vanlig måte.

```
devtools::install_git("https://code.europa.eu/digclass/digclass.git")
```

Denne pakken inneholder også funksjonen for å rydde opp i isco-kodene. Det er noen vanlige problemer som omkodes enkelt med `repair_isco`. I følgende kode sjekkes isco-kodene *før* funksjonen `isco88_to_ordc` brukes til selve omkodingen. Den har en versjon som gir tallkode og en som gir tekstverdier, som er kjekt, så da kjøres begge to.

Denne funksjonen spytter også ut mange beskjeder i output som vi strengt tatt ikke trenger. Den melder fra om alle verdier som ikke inngår i klasseskjemaet og som derfor settes til NA. Det er fint, men tar i dette tilfellet ganske mye plass. Vet å bruke funksjonen `suppressMessages({...})` og parenteser rundt hele koden slipper vi dette. Normalt skal du ikke bruke denne funksjonen da du vanligvis vil ha slike beskjeder. Her tar det bare litt mye plass.

```
library(DIGCLASS)

suppressMessages({
polit3 <- polit %>%
  mutate(isco08 = repair_isco(.$isco08),
         orcd = isco88_to_ordc(isco08, label = FALSE),
         orcd_lab = isco88_to_ordc(isco08, label = TRUE))
})

head(polit3)
```

```
  isco08 gndr polintr occupation orcd                     orcd_lab
1   3333    1       3          3 <NA>                         <NA>
2   7122    1       2          7   10        Skilled working class
3   4221    2       4          4    8 Lower-middle class: balanced
4   4311    1       3          4 <NA>                         <NA>
```

```
5   6130    2       2          6   12     Primary-sector employees
6   7212    1       2          7   10        Skilled working class
```

Da kan vi lage tabellen omigjen bare for å sjekke at vi får samme svar.

```
polit3 %>%
  select(orcd, gndr) %>%
  tbl_summary(by = gndr)
```

Characteristic	**1**, N = 23,020	**2**, N = 26,499
orcd		
1	190 (1.6%)	182 (1.6%)
10	4,188 (36%)	1,890 (17%)
11	2,075 (18%)	2,564 (23%)
12	723 (6.2%)	669 (6.0%)
13	80 (0.7%)	101 (0.9%)
2	1,014 (8.7%)	838 (7.5%)
3	10 (<0.1%)	2 (<0.1%)
4	316 (2.7%)	604 (5.4%)
5	947 (8.1%)	1,062 (9.5%)
6	572 (4.9%)	639 (5.7%)
7	226 (1.9%)	422 (3.8%)
8	1,137 (9.8%)	1,882 (17%)
9	167 (1.4%)	333 (3.0%)
Unknown	11,375	15,311

Gjøre samme ting med mange variable med `across()`

::: {.quarto-book-part}

`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJhcHBlbmRpeCIsImJvb
```

Appendices

:::

16 Import av data fra Sikt - håndtering av formater med metadata

For de som vil ha en litt mer utfordring kan man lese inn filen i Stata-format. Dette er slik det blir levert fra Sikt[1]

Stata-formatet dta har to utfordringer når vi importerer til R. Det er tilsvarende problemstilling hvis man importerer fra SPSS eller SAS. For det første er det noen ganger gitt en egen kode for manglende verdi, såkalt "missing". For det andre lagres informasjonen på en annen måte enn i R. I Stata er ofte kategoriske variable lagret som en numerisk variabel med en tilhørende "label". Når man leser inn dta-fil til R vil disse variablene være av typen "labelled". I R er det langt bedre å gjøre de om til factor-variable, men det er litt styr å kode om hvis det er veldig mange variable i datasettet - slik det ofte er i surveydata.

Vi presenterer en samlet løsning først, så tar vi hver del for seg etterpå for å forklare. Nedenforstående kode gjør omtrent følgende:

- Leser inn en dta-fil
- Omkoder alle spesifiserte missing-verdier til NA. Dette gjøres for *alle* variable i hele datasettet (altså hundrevis av variable)
- Fjerner alle labler som ikke er i bruk (altså: labler for ulike missing-verdier)
- Gjør om alle variable av typen "labelled" til factor-variable

16.1 Håndtering av user-NAs

For disse dataene vil det være ulike sett av missing-verdier for de ulike variablene. Dette kan helt fint håndteres manuelt variabel for variabel. Men for å ha ordentlig kontroll på at det blir riktig bør det automatiseres. Logikken i denne delen går et stykke utover hva vi forventer at den jevne sosiologistudent skal lære.

En første sted er å lese inn dokumentasjonsrapporten fra en html-fil slik den leveres fra Sikt og gjør det om til et håndterbart oppslags-datasett. Dette er beskrevet i eget appendix. Det følgende tar utgangspunkt i at en slik oppslagsfil finnes.

[1] Det kan sies mye om å levere ut data på denne måten, men det vil ikke ta seg ut å gjøre det i undervisningsmateriale.

Det er noen verdier som i dokumentasjonen er spesifisert som spesielle typer missing. Disse skal vi kode om til NA. Disse verdiene har labler som starter med "filter:" eller "vil ikke svare" etc. Disse danner basis for omkoding til `NA`. Dette er ikke en komplett liste over koder som innebærer at det egentlig mangler informasjon. (Dvs. fordi koden indikerer grunner til at det mangler informasjon). Etter denne oppryddingen kan det altså fremdeles hende at det dukker opp noe slikt, så vær påpasselig med å sjekke variabelens fordeling før du analyserer med regresjonsmodeller.

Funksjonen nedenfor skal brukes innenfor et steg der man går gjennom alle variablene en om gangen. For hver variabel slås det opp de aktuelle missing-verdiene som gjelder for denne og bruker `replace` til å omkode til `NA` for disse verdiene. Når denne funksjonen kalles for hver variabel senere, så brukes det altså ulike definisjoner av missing-verdier for hver variabel.^(Basert på kode fra https://tim-tiefenbach.de/post/2023-recode-columns/)

```
# leser inn kodeliste/dokumentasjon
dat_dict <- readRDS("data/dat_dict.rds")

# velger kun missing-lablene
dat_dict_na <- dat_dict %>%
  filter(  str_sub(tolower(label), 1, 7) == "filter:" |
           str_sub(tolower(label), 1, 10) == "filterfeil" |
           str_sub(tolower(label), 1, 12) == "ikke besvart" |
           str_sub(tolower(label), 1, 9) %in% c("filter t2", "filter t1", "filter t3") |
           tolower(label) %in% c("vil ikke svare",
                                 "deltok ikke i runden",
                                 "mangler data", "manglende data",
                                 "mangler verdi", "ubesvart spørsmål",
                                 "ugyldig verdi", "oppgitt verdi ikke et årstall",
                                 "filter -",
                                 "ikke svart post/web skjema"))

# Funksjon for å recode som har gitte user-NA.
# (Må brukes innenfor `across()` nedenfor)
recode_col_na <- function(x, dict) {
  recode_vec <- dict %>%
    filter(col_nm == cur_column()) %>%
    mutate(value = as.numeric(value)) %>%
    pull(value)
  replace(x, x %in% recode_vec, NA)
}
```

16.2 innlesning av data

```
library(tidyverse)
library(haven)
library(labelled)


# data
faste <- read_stata( paste0(infilbane, "NorLAG-lengde-faste.dta"), encoding = "utf-8")

lang <- read_stata( paste0(infilbane, "NorLAG-lengde-intervju.dta"), encoding = "utf-8")



norlag_lbl <- merge(faste, lang, by = c("ref_nr"), all.y = TRUE) %>%
  filter(iodeltakelse == 1 |
           iodeltakelse == 2 & round %in% c(1, 3) |
           iodeltakelse == 3 & round %in% c(2, 3) |
           iodeltakelse == 4 & round %in% c(1) |
           iodeltakelse == 5 & round %in% c(1, 2) |
           iodeltakelse == 6 & round %in% c(2)
  )



# vektorer av variable som skal omkodes
cols_vec_all <- unique(dat_dict$col_nm)
vars <- unique(names(norlag_lbl))
cols_vec <- cols_vec_all[cols_vec_all %in% vars]
cols_vec_na <- cols_vec[(cols_vec %in% unique(dat_dict_na$col_nm))]

norlag <- norlag_lbl %>%
  mutate(across(all_of(cols_vec_na),
                \(x,dic) recode_col_na(x, .env$dat_dict_na))) %>%
  mutate(across(where(is.labelled), ~as_factor(.))) %>%
  mutate(across(where(is.factor), ~fct_drop(.)))
```

I tilegg skal vi lage en variabel for det vi kan kalle hovedaktivitet som sysselsettingsstatus. Det er om man er yrkesaktiv, arbeidsledig, student eller annet. I hver runde av NorLAG ble svarkategoriene utformet litt forskjellig, så derfor er svarene fordelt over tre variable. Nedenfor samles disse sammen og kodes om basert på tekststrenger.

```r
## Omkoder hovedaktivitet
fs <- lvls_union( list(norlag$wr001, norlag$wr002, norlag$wr003c)) %>% tolower() %>% uniqu

norlag <- norlag %>%
  mutate(across(wr001:wr003c, ~factor(tolower(.), levels=fs))) %>%
  mutate(hovedaktivitet = case_when(round == 1 ~ wr001,
                                    round == 2 ~ wr002,
                                    round == 3 ~ wr003c) ) %>%
  mutate(hovedaktivitet2 = case_when( str_sub(hovedaktivitet, 1, 5) == "yrkes" ~ "Yrkesakt
                                      str_detect(hovedaktivitet, "arbeidsledig") ~ "Trygde
                                      str_detect(hovedaktivitet, "student") ~ "Trygdet/arb
                                      str_detect(hovedaktivitet, "trygd") ~ "Trygdet/arbei
                                      str_detect(hovedaktivitet, "annet") ~ "Trygdet/arbei
                                      str_sub(hovedaktivitet,1,6)   == "hjemme" ~ "hjemmev
                                      str_detect(hovedaktivitet, "pensjonist") ~ "pensjoni
                                      is.na(hovedaktivitet) ~ "Trygdet/arbeidsledig/stud/a
```

Og dermed har vi et datasett i et svært så ryddig R-format.

16.3 Hvordan fungerer koden ovenfor?? En intro til mer avansert databehandling

16.3.1 Sjekk datastruktur og bruk av `filter`

16.3.2 Omkode bruker-spesifiserte missing-verdier til `NA`

16.3.3 Kode om på tvers av mange variable med `across`

16.3.4 Fjerne nivåer som ikke brukes: `drop_unused_value_labels`

16.3.5 Gjør om til factor med `unlabelled`

16.4 For spesielt interesserte: jobbe med labelled-data

17 Lage dictionary-fil

Dokumentasjonen som følger med NorLAG og andre datasett fra Sikt er en html-fil (dvs. webside) med en oversikt over alle variable og hvordan de er kodet. Dette er fint for manuelt oppslag, men er ikke ideelt til bruk for maskinell behandling.

Det vi ideelt skulle hatt er et samlet datasett der kodeskjemaet er knyttet til variabelnavnene. Dette kalles noen ganger en "dictionary" fil. All informasjonen vi trenger for å lage en slik ligger i html-filen man får sammen med datasettet fra Sikt, bare på en knotete form. Det skal vi fikse.

Nedenfor skal vi vise hvordan vi gjør følgende:

- Leser inn en fullstendig html-fil, inkludert alle html-kodene
- Identifiserer den delen av html-filen som inneholder kodeskjema og begrenser filen til disse
- Plukker ut alle tabeller og gjør dem om til data.frame-struktur i R
- Rensker opp i filen til å bare beholde de relevante radene

Dette er egentlig en svært enkel introduksjon til webscraping for et spesfikt formål. Vi skal bruke pakken `rvest` som er laget nettopp for webscraping.

```
library(rvest)       # scrape html-sider
library(tidyverse)   # generell databehandling
```

17.1 Lese inn html-dokumentasjonen

Første sted er å lese inn html-filen. Funksjonen `read_html()` gjør dette. For å skjønne litt mer av hvordan dette ser ut kan du åpne den opprinnelige html-filen i ren tekst, f.eks. med bruk av Notepad. Det er dette som leses inn. Jeg legger det i et nytt objekt som jeg har kalt `cb` (forkortelse for codebook).

```
#library(XML)

# read html file
u <- "C:/Users/torbskar/OneDrive - Universitetet i Oslo/Dokumenter/Undervisning/SOS4020_fo
```

```
cb <- read_html(u)
```

For NorLAG er dokumentasjonsdokumentet inndelt i flere deler, og det er bare den siste delen som inneholder kodeskjemaene. Det er denne siste delen vi trenger, så første utfordring er å plukke ut denne delen.

En html-fil er strukturert innenfor "noder" som har en start og en slutt. Et avsnitt starter med en kode `<a>` og avsluttes med `</a>`. Tilsvarende koder finnes for tabeller og andre elementer. Disse delene har er oftest gitt et navn som man kan identififiseres og brukes til lage lenker til spesifikke deler av siden (jf. innholdsfortegnelsen). Vi bruker denne til å filtrere filen.

I akkurat denne filen trenger vi informasjonen som ligger etter overskriften "Variables Description". For å finne navnet på dette avsnittet kan man undersøke lenken i innholdsfortegnelsen der det fremkommer som `#variables`. Eller man kan åpne html-filen i et tekstdokument og søke opp tittelen, så finner man koden `name='variables` innenfor det avsnittet.

Vi bruker `html_nodes` til å trekke ut bare dette avsnittet som følger.

```
# Find the specific heading
variables <- cb %>%
  html_nodes("a[name='variables']")
```

I denne dokumentasjonen er hver variabel lagret i en egen tabell. I html-kode angis begynnelsen av en tabell med `<table>` og denne brukes til å trekke ut bare tabellene.

```
tables <- variables %>%
    #html_node(xpath = "//a[@name='variables']") %>%
    html_nodes(xpath = "./following::table")
```

Så kan vi bruke funksjonen `html_table` til å trekke ut hver enkelt tabell i en struktur som er lettere å jobbe med, nemlig en "data.frame", altså en rektangulær struktur med rader og kolonner slik datasett vanligvis ser ut. Hver tabell blir et eget data.frame-objekt, og når man legger dette i et nytt objekt blir det av typen "list". En "list" er en samling objekter som har hver sin plass i det samme objektet. (Du kan tenke på det som en eske med flere mindre ekster oppi). Vi kommer tilbake til hvordan de slås sammen.

```
table_data <- html_table(tables)
```

17.1.1 Legge det hele i en funksjon

```
# Check if the heading exists
if (length(variables) > 0) {
  # Find the tables after the heading
  tables <- variables %>%
    html_node(xpath = "//a[@name='variables']") %>%
    html_nodes(xpath = "./following::table")

  # Extract the table data
  table_data <- html_table(tables)
  } else {
  cat("The specified heading was not found.")
}


tbslist <- list()
for(i in 1:length(table_data)){
  if("X2" %in% names(table_data[[i]]) &
     "X3" %in% names(table_data[[i]]) &
     !("X4" %in% names(table_data[[i]])) ){

    tbslist[[i]] <- table_data[[i]] %>%
      mutate(col_nm = strsplit(as.character(.[1,1]), split = ":")[[1]][1],
             spm = strsplit(as.character(.[1,1]), split = ":")[[1]][2]) %>%
      rename( value = X2,
              label = X3) %>%
      filter( str_detect(X1, "Values and categories")) %>%
      filter( !is.na(value)) %>%
      select(-X1)
  }
  else{
    if(i==1){
      teller <- 0
      }
    teller <- teller + 1
  }
  if(i == length(table_data)){
    print(paste("Antall variable som ikke har omkodinger: ", teller))
  }
}
```

```
[1] "Antall variable som ikke har omkodinger:  1"
```

```
dat_dict <- bind_rows(tbslist) %>%
  select(col_nm, value, label, spm)
```

```
saveRDS(dat_dict, "data/dat_dict.rds")
```