Veldig praktisk dataanalyse

En innføring i bruk av R

Torbjørn Skardhamar

2023-12-30

Table of contents

Forord

Denne boken er beregnet som en veldig praktisk innføring i dataanalyse med R. Med det som gjennomgås i denne boken skal du være i stand til å skrive en enkel rapport, med deskriptive tabeller og figur, og grunnleggende regresjonsanalyse.

Dette er imidlertid ikke en lærebok i statistikk og metode, så det vil ikke vektlegges dypere forklaringer av statistiske begreper, fordelinger, designvalg osv. Du bør konsultere en annen standard lærebok for slike ting. Men all statistikk gjøres i praksis med datamaskiner og programmering. Denne boken vektlegger denne praktiske siden.

R er et programmeringsspråk spesielt godt egnet for statistisk analyse og databehandling, og det er viktig å lære å skrive kode både for databehandling og analyse. Men boken er *ikke* ment som en innføring i programmering, og målsettingen er ikke at du skal beherske R-programmering som sådan. Det overordnede fokuset er å få gjort analyser raskt og effektivt.

Dette er en bok som lærer deg hvordan få ting gjort med R. Fra import av data til output av publiserbar kvalitet. Det vektlegges arbeidsflyt, mappestruktur og stilistiske elementer. Dette er for å lette arbeidet ditt, gjøre koden lettere å lese og generelt holde orden.

Boken vektlegger også det estetiske. Alle tabeller og figurer skal være *pene* i den forstand at de skal være informative, lette å lese, og bruke en layout som er tiltalende. Dessuten skal alt kunne eksporteres til vanlige tekstbehandlingsprogram. Klipp-og-lim skal være unødvendig.

Et moment ved det estetiske er fargebruk. I tillegg til at R støtter avansert fargebruk er det også enkle fargepaletter tilgjengelig. Men det er viktig at fargebruken er funksjonell så alle kan lese det. Derfor må fargevalg ta hensyn til slike ting som fargeblindhet. Slikt dekkes også.

Hvordan bli god?

Analyse av data er praktiske ferdigheter og for å bli god må du øve. Hvert kapittel inneholder derfor en rekke oppgaver som du må jobbe med. Teksten har mange eksempler med bruk av ett eller flere datasett, og så skal du gjøre tilsvarende med andre datasett. Bruk gjerne helt andre datasett også, og jobb praktisk med oppgaver.

Datasett

De aller fleste datasettene som brukes i denne boken finnes tilgjengelig i diverse R-pakker, og det er referanser til disse gjennomgående slik at du lett kan finne dem. I tillegg er datasettene gjort tilgjengelig på bokens ressursside. Noen datasett som brukes er hentet fra andre kilder, som for eksempel offisiell statistikk fra SSB, eller utdrag fra andre datasett som ikke nødvendigvis er åpent tilgjengelig ellers.

For en oversikt over andre datasett lett tilgjengelig i R, se hjemmesiden til Vincent Arel-Bundock som inneholder en oversikt.

Til studenter

For all kvantitativ analyse er det svært viktig at man jobber med kode. Når du har gjort alt arbeidet med kode kan du enkelt gjøre analysen på nytt med samme data eller nye data. Koden er langt på vei dokumentasjon på den jobben du gjør, og gjør analysene dine reproduserbare. Å lære seg dataanalyse med meny-baserte verktøy er derfor ikke et alternativ for seriøse analyser. Vitenskapen skal være reproduserbar, og da må det skrives kode.

Dette betyr ikke at du behøver være spesielt interessert i datamaskiner eller informatikk. Å kunne kode har visse nerde-assosisasjoner, men ikke la deg affisere av det. Her er det det praktiske som vektlegges og du skal lære så lite programmering som mulig - men nok til å gå gjort det du skal. Du trenger forsåvidt en god del, så det blir mye kode som skal skrives.

Hvis du er av den typen som gjerne skulle vært litt mer nerdete så er du kanskje mer bekymret for at det blir for lite kode? Dette er uansett et godt sted å starte. Denne boken er lagt opp som en innføring på relativt lavt nivå, men utøver konsekvent prinsipper du trenger på et høyere nivå. Alt i denne boken kan påbygges til avansert programmering.

Til undervisere

Denne boken har gjort en del strategiske valg av pakker og funksjoner som brukes gjennomgående. Først og fremst brukes *tidyverse* gjennomgående, og andre funksjoner som brukes vil være kompatible med tidyverse. Dette gir en konsistent og effektiv kode gjennomgående.

Det vektlegges gjennomgående at alle resultater skal bli pene, i publiserbar kvalitet, og i en form som lar seg eksportere til tekstbehandlingsprogram i MS Office, fordi det i praksis er dette de fleste bruker. Men det er også vektlagt å bruke eksportfunksjoner som *også* kan eksportere til andre formater som pdf, html or rft, og er kompatible med Markdown/Quarto. Selv om de fleste nybegynnere i R ikke jobber i disse formatene skal de ikke behøve å bytte funksjoner hvis de skulle ha behov for det senere.

1 Installere R og Rstudio

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.

1.0.1 Ikke alle feilmeldinger er like nøye

Mange av dere vil få en feilmelding av denne typen når dere starter R:

```
Error in file.exists(pythonPath) :
  file name conversion problem -- name too long?
```

Ikke bry dere om akkurat den. Det spiller ingen rolle.

1.0.2 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/

1.0.3 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/

1.0.4 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.

1.0.5 Spesielt om Chromebook

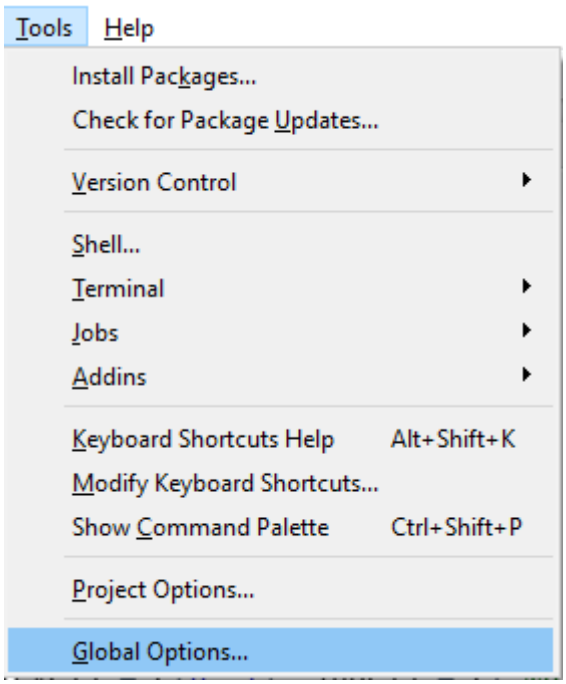
Chromebook kjører et annet operativsystem og R vil ikke uten videre fungere. Derimot kan man på de fleste slike maskiner åpne opp for å kjøre Linux og da kan man installere linuxversjon av R og Rstudio. https://blog.sellorm.com/2018/12/20/installing-r-and-rstudio-on-a-chromebook/

1.1 Oppsett og forberedelser

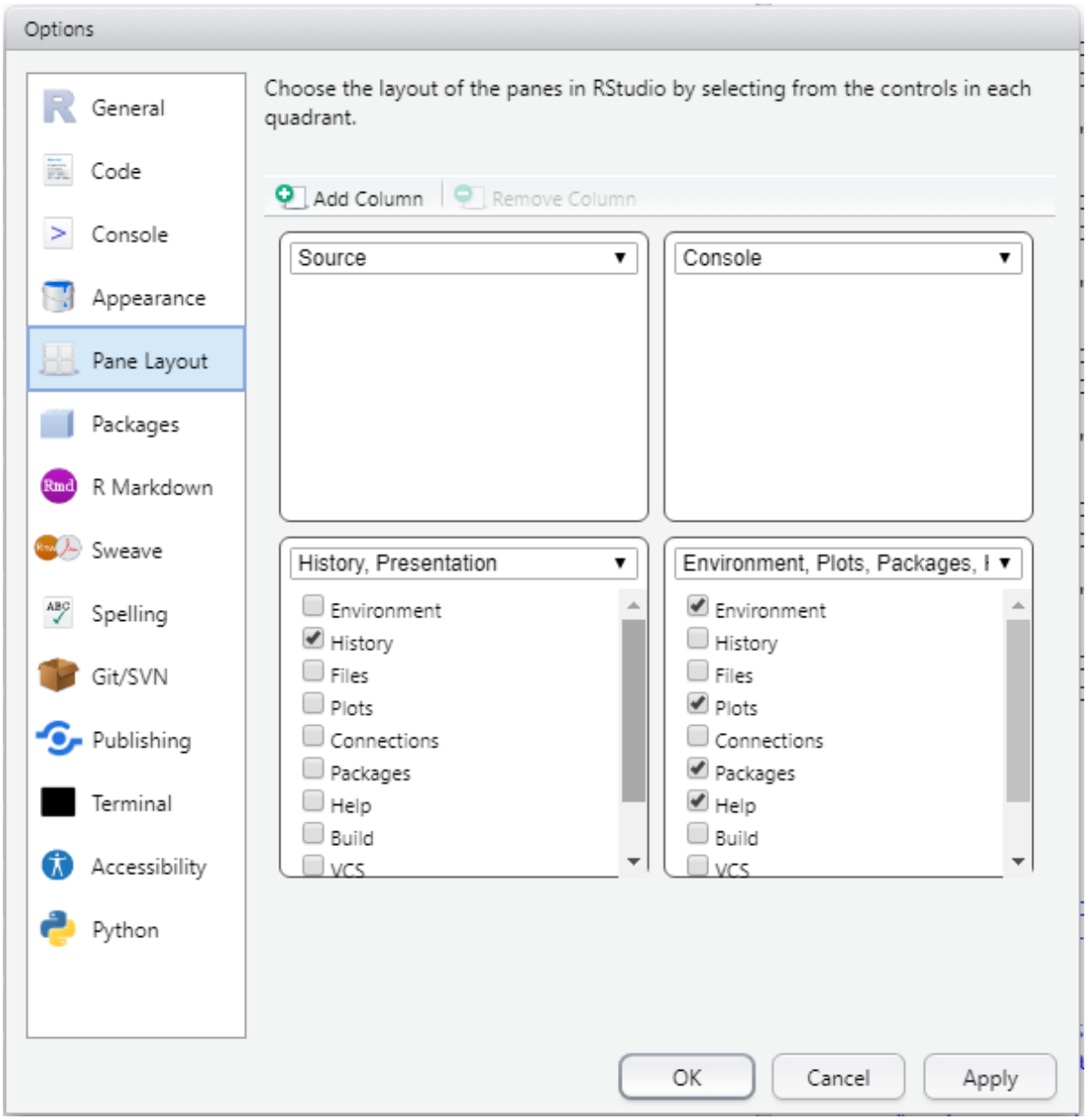
Dette oppsettet gjelder både hvis du har en lokal installasjon og for skyløsninger. Utseendet spiller ingen rolle, og R kan også fungere uten å opprette "projects" som beskrevet her. Men det er lettere å bruke og du har bedre orden hvis du gjør dette.

1.1.1 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.

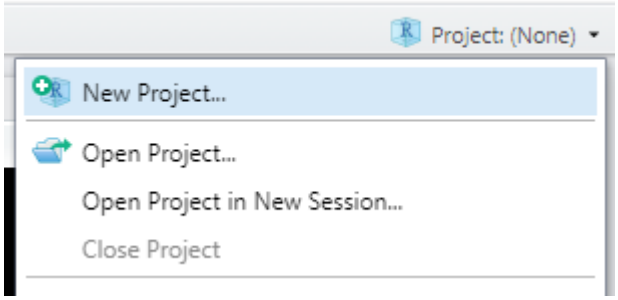
1.2 Rstudio projects

Når du åpner Rstudio skal du alltid åpne som «project» (se video med instruksjon og i R4DS (Wickham and Grolemund (2017))). 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 prosjektmappen som øverste nivå og egne undermapper for data, script, og output.

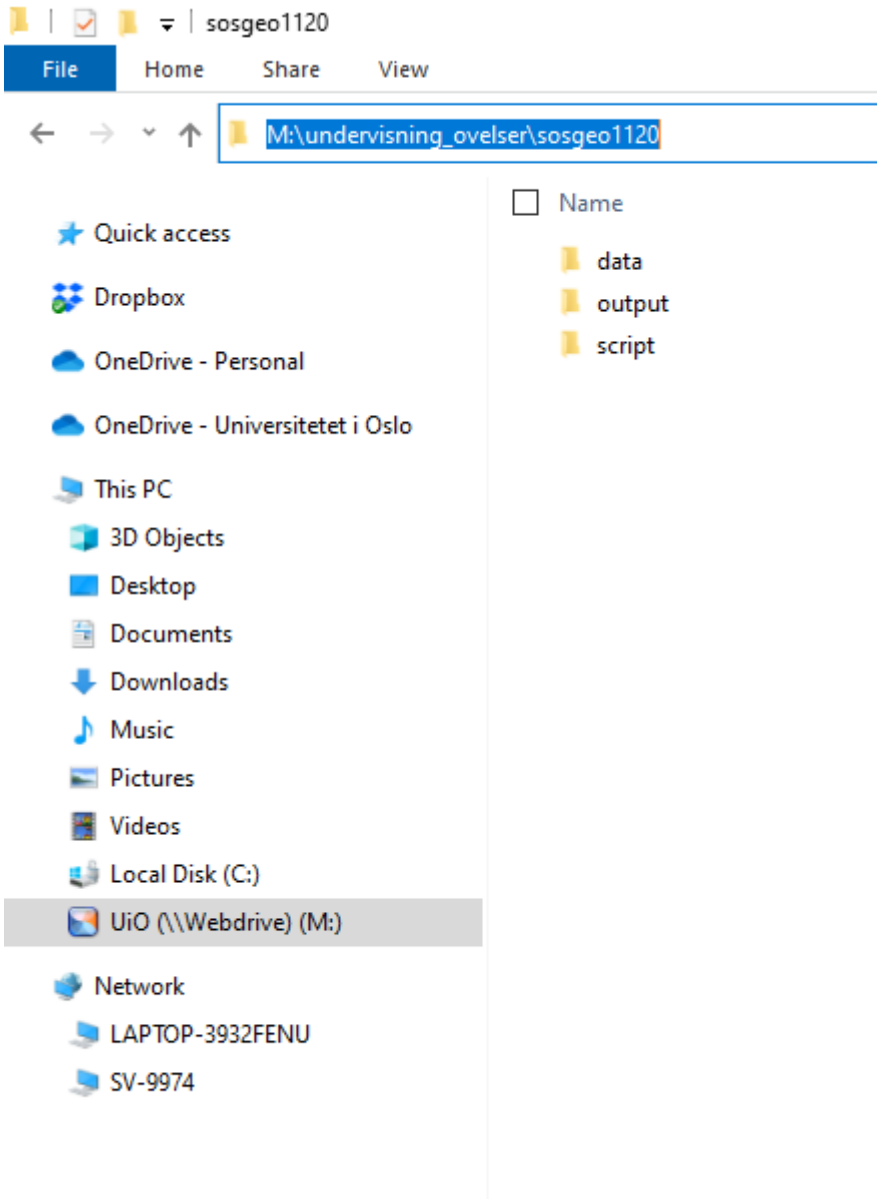
1.3 Åpne RStudio og opprett et .Rproject



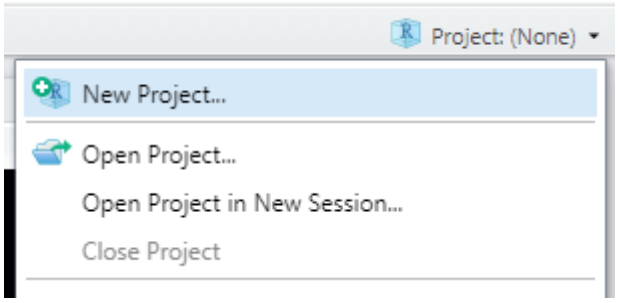
Bruk funksjonen `getwd()` og se at du har riktig filbane til arbeidsområdet. Hvis du ikke er sikker på hva det betyr, må du spørre noen eller finne det ut på annen måte!

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.

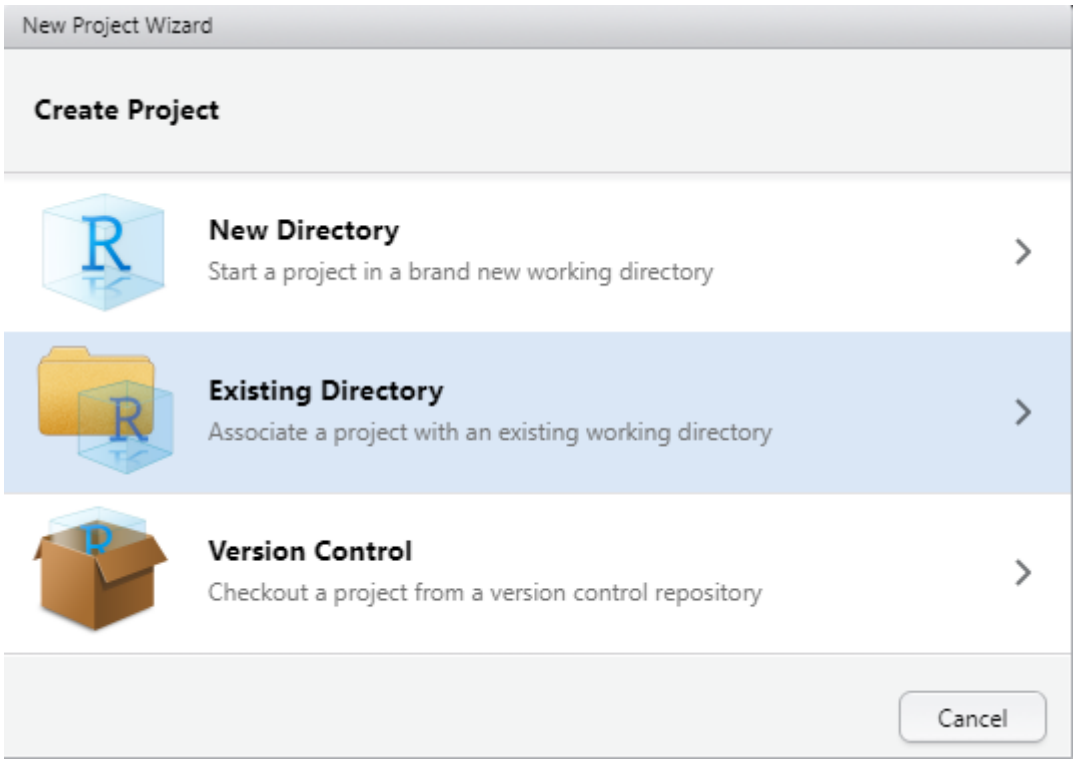
For dette kurset skal du ha en mappestruktur med en hovedmappe for dette kurset og tilhørende undermapper. Det spiller ingen rolle hvor på datamaskinen du legger disse mappene, men du må vite hvor det er. Lag første en mappe med et hensiktsmessig navn for kurset, og innunder denne mappen lager du tre 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.

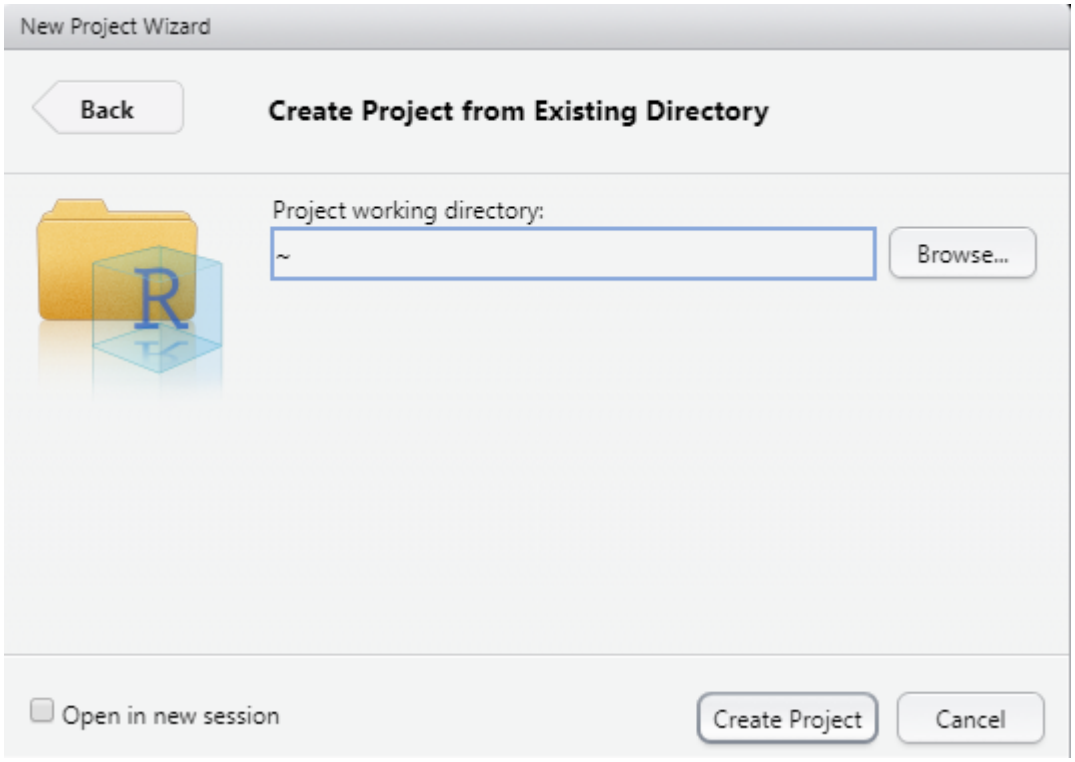


Deretter klikker du du «Existing Directory»

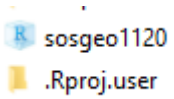


Klikk «Browse» og bla deg så frem til mappen du har laget for dette kurset.

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.

2 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. En grundigere gjennomgang av R finner du i Wickham and Grolemund (2017).

2.0.1 Et par hurtigtaster

Du skriver altså kode i script-vinduet i RStudio. For å kjøre koden kan du klikke på "Run" opp i høyre hjørne av script-vinduet. Det kan være lurt å markere det du vil kjøre før du klikker "Run" slik at du bare kjører akkurat det du har tenkt til. Man blir fort lei av å klikke på den måten. En hurtigtast er **Ctlr + Enter** som altså gjør det samme.

Du vil komme til å skrive '%>%' ganske mange ganger etterhvert. Det er litt styrete å skrive pga hvordan tastene ligger på tastaturet ditt. En hurtigtast for dette tegnet er **Ctrl + Shift + M**.

Det er mange andre hurtigtaster tilgjengelig, men det er disse to du vil ha mest bruk for.

2.0.2 Problemer med æøå?

På noen datamaskiner vil æøå vises feil. Det er jo veldig irriterende. R henter informasjon fra operativsystemet ditt for å vite hva slags tegnsetting som skal brukes. Noen ganger skjer det feil. Her er en kode for å sette tegnsetting til norsk:

```
Sys.setlocale(locale='no_NB.utf8')
```

Det kan hende du må kjøre denne koden hver gang du starter R.

2.1 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.

2.2 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.

2.3 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.

For dette kurset skal vi bruker flere pakker og de kan installeres samlet med koden nedenfor. Hvis du har behov for andre pakker enn dette (f.eks. at vi har glemt å ta med noe i koden), så installerer du det som er angitt på samme måten.

```
install.packages( c("tidyverse", "haven", "gtsummary", "modelsummary", "stargazer",
                    "ggthemes", "ggforce",
                    "labelled", "memisc",
                    "causaldata", "gapminder")
            )
```

Installering av pakker gjør du bare én gang. Hvis du oppdaterer R til en nyere versjon må du imidlertid installere pakkene på nytt. Det er alltid en fordel å ha oppdatert software.

Du må derimot laste en pakke hver gang du starter R på nytt. Det betyr at du gjør pakkens funksjoner tilgjengelig for denne arbeidsøkten. 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.

2.4 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.

2.5 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 er redegjort for i Wickham (2014). 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.

2.5.1 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
```

2.5.1.1 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
```

2.5.1.2 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)
```

2.5.1.3 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
```

2.5.1.4 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.

2.5.1.5 Lagre data

Du kan som et utgangspunkt tenke at du *ikke* skal lagre data på disk unntatt det originale datasettet. Scriptet ditt (evt. flere script) starter med å lese inn dataene i R og gjør så alt du trenger av databearbeiding og analyser fra start til slutt. På den måten har du også sikret at du har reproduserbare script som både dokumenterer jobben du gjør og muliggjør at andre kan ettergå analysene. Dette er veldig viktig for alt vitenskapelig arbeid.

Hvis du likevel trenger å lagre data til disk, så bør det primært være fordi det tar for lang tid å kjøre gjennom fra start. Med store datasett, kompliserte operasjoner og tidkrevende estimeringer kan det være et reelt behov.

I slike tilfeller bør du lagre datasett i .rds-formatet fordi det er veldig greit å lese inn igjen. Det er R sitt eget format.

```
saveRDS(dinedata, "data/dinedata_temp.rds")
```

Hvis du skal lagre et bearbeidet datasett *permanent* (f.eks. hos sikt.no) for å dele med andre, så er det helt andre prosedyrer som gjelder særlig med tanke på dokumentasjon og at filene skal være software-uavhengige. Da vil csv-format ofte være å foretrekke hvis dokumentasjonen og kodelister er på hensiktsmessig format.

```
write_csv(dinedata, "data/dinedata_temp.csv")
```

Det kan være andre grunner til å lagre i helt andre formater. R støtter mange forskjellige, men du må la behovet styre hvilket format du velger. Foreløpig kan du tenke at du primært bruker .rds. Bare hvis du bare skal skal lagre et datasett permanent skal du bruker .rds. For å bruke noe annet trenger du en god grunn.

2.5.2 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 `+`.

2.5.3 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.

2.6 Andre nyttige ting

2.6.1 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!

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

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.

2.6.1.1 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.

2.6.2 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
```

2.6.3 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 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),
```

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

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

Selvfølgelig! Denne koden ser ut til å være skrevet i R-programmeringsspråket og bruker "dplyr" -pakken for å manipulere og analysere data. La meg forklare hver del av koden trinn for trinn:

1. **`dinedata %>%`**
   Dette er en såkalt "pipe-operator" (**`%>%`**), som er en funksjonalitet fra "dplyr" -pakken som lar deg kjede sammen ulike operasjoner på data uten å måtte referere til dataene igjen og igjen. **`dinedata`** er trolig navnet på datarammen som blir brukt i koden.
2. **`group_by(gruppe) %>%`**
   Her grupperes dataene etter en kolonne kalt "gruppe". Dette betyr at alle påfølgende operasjoner vil bli utført på hver gruppe separat.
3. **`mutate(antall = n(), gjennomsnitt = mean(varA), avvik = varA - gjennomsnitt) %>%`**
   I denne delen blir det laget nye kolonner i dataene.
   - **`antall`** blir satt til antallet rader i hver gruppe ved bruk av funksjonen **`n()`**.
   - **`gjennomsnitt`** blir satt til gjennomsnittet av verdiene i kolonnen "varA" for hver gruppe ved bruk av funksjonen **`mean()`**.
   - **`avvik`** blir beregnet som differansen mellom verdiene i kolonnen "varA" og gjennomsnittet.
4. **`arrange(gruppe, desc(avvik)) %>%`**
   Dette trinnet sorterer hver gruppe i stigende rekkefølge basert på kolonnen "gruppe" og deretter i synkende rekkefølge basert på kolonnen "avvik". Dette betyr at gruppen med det største negative avviket vil være øverst.
5. **`filter(row_number() == 1)`**
   Her filtreres resultatet til kun å inkludere den første raden for hver gruppe. Dette er den raden med det største negative avviket, som ble bestemt i trinn 4.

Så, koden i sin helhet gjør følgende:
Den tar en dataramme kalt **`dinedata`**, grupperer dataene etter en kolonne kalt "gruppe", legger til beregnede kolonner for antall rader i hver gruppe, gjennomsnittet av "varA" i hver gruppe og avviket fra gjennomsnittet, sorterer hver gruppe basert på avviket, og til slutt returnerer den første raden med det største negative avviket for hver gruppe.

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. Jeg øns
  dinedata %>%
  group_by(gruppe) %>%
  mutate(antall = n(), gjennomsnitt = mean(varA),
         avvik = varA - gjennomsnitt) %>%
  arrange(gruppe, desc(avvik)) %>%
  filter(row_number() == 1)
```

Selvfølgelig! Denne koden ser ut til å være skrevet i R-programmeringsspråket og bruker "dplyr" -pakken for å manipulere og analysere data. La meg forklare hver del av koden trinn for trinn:

1. `dinedata %>%`
   Dette er en såkalt "pipe-operator" (`%>%`), som er en funksjonalitet fra "dplyr" -pakken som lar deg kjede sammen ulike operasjoner på data uten å måtte referere til dataene igjen og igjen. `dinedata` er trolig navnet på datarammen som blir brukt i koden.
2. `group_by(gruppe) %>%`
   Her grupperes dataene etter en kolonne kalt "gruppe". Dette betyr at alle påfølgende operasjoner vil bli utført på hver gruppe separat.
3. `mutate(antall = n(), gjennomsnitt = mean(varA), avvik = varA - gjennomsnitt) %>%`
   I denne delen blir det laget nye kolonner i dataene.
   - `antall` blir satt til antallet rader i hver gruppe ved bruk av funksjonen `n()`.
   - `gjennomsnitt` blir satt til gjennomsnittet av verdiene i kolonnen "varA" for hver gruppe ved bruk av funksjonen `mean()`.
   - `avvik` blir beregnet som differansen mellom verdiene i kolonnen "varA" og gjennomsnittet.
4. `arrange(gruppe, desc(avvik)) %>%`
   Dette trinnet sorterer hver gruppe i stigende rekkefølge basert på kolonnen "gruppe" og deretter i synkende rekkefølge basert på kolonnen "avvik". Dette betyr at gruppen med det største negative avviket vil være øverst.
5. `filter(row_number() == 1)`
   Her filtreres resultatet til kun å inkludere den første raden for hver gruppe. Dette er den raden med det største negative avviket, som ble bestemt i trinn 4.

Så, koden i sin helhet gjør følgende:
Den tar en dataramme kalt `dinedata`, grupperer dataene etter en kolonne kalt "gruppe", legger til beregnede kolonner for antall rader i hver gruppe, gjennomsnittet av "varA" i hver gruppe og avviket fra gjennomsnittet, sorterer hver gruppe basert på avviket, og til slutt returnerer den første raden med det største negative avviket for hver gruppe.

Merk at chatGPT da vil foreslå en løsning, men det kan godt hende du får et litt annet svar enn det som er gjengitt ovenfor. Det er ikke gitt at den forstår problemet ditt korrekt og løsningen er heller ikke nødvendigvis riktig!

For å få et godt svar er det viktig at du i promptet gir en god beskrivelse av *konteksten*, som innebærer hva du ønsker å gjøre, hva resultatet blir og hva du tror er feil. For at dette skal fungere godt trenger du altså å forstå problemet ditt og beskrive det så tydelig som mulig. Så må du teste løsningen på din egen datamaskin og sjekke at det faktisk fungerer.

3 Lese inn datasett

I dette kapittelet skal vi bruke følgende pakker:

```
library(tidyverse)
library(memisc)
```

3.1 Generelt om ulike dataformat

Data kan være lagret i mange ulike formater, og du vil kunne 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. R kan imidlertid håndtere det aller meste av dataformater på en eller annen måte. Foreløpig skal vi kun se på et dataformat som er spesielt egnet for R, nemlig rds-formatet. Alle datasett som følger med denne boken vil være i rds-formatet, med unntak av kapittelet der temaet er import av andre formater.

3.2 Lese inn datasett og få oversikt

Vi bruker her et lite utvalg variable fra Ungdata 2010-2020 (NOVA and Bakken (2023)) som er lagret i rds-format. Dette datasettet er tilgjengelig på NSD sine sider. Følgende kode bruker funksjonen `readRDS` for å lese inn datasettet. Filbanen er angitt å ligge i en mappe som heter "data" i prosjektmappen, og filnavnet er "ungdata_alko.rds". Når man leser inn dataene legges de i et "objekt" som vi her kaller *ungdata_alko*.

```
ungdata_alko <- readRDS("data/ungdata_alko.rds")
```

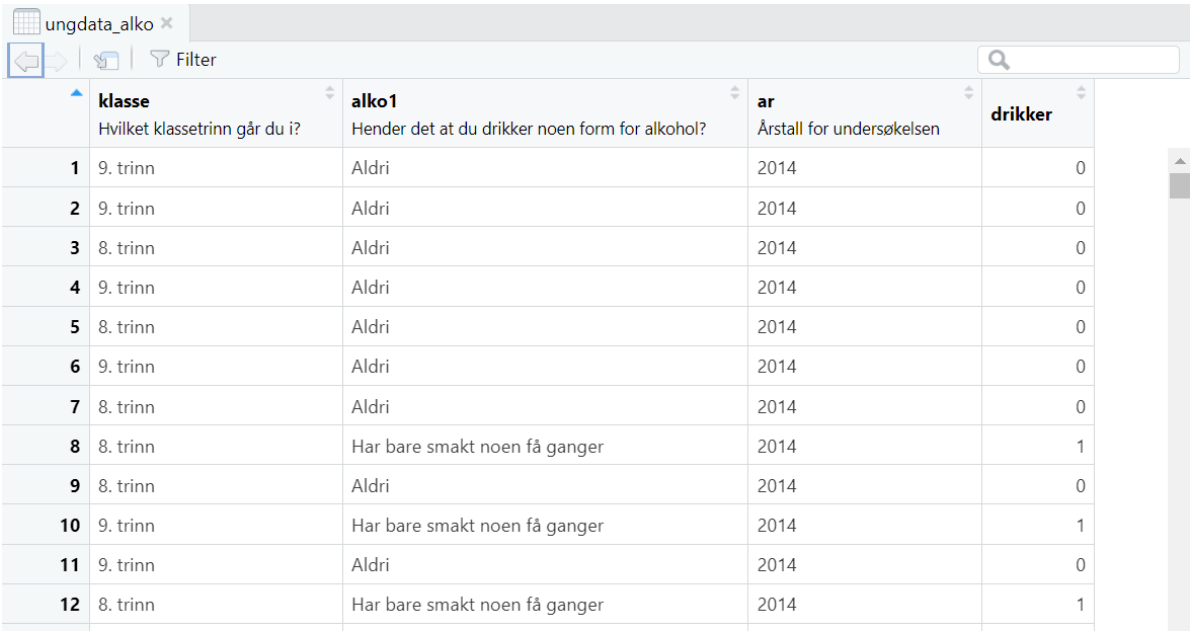
En første ting man bør sjekke er om dataene er lest inn riktig og at det rett og slett ser greit ut. Det er lite som kan gå galt når man leser inn en rds-fil, men det kan være en fordel for deg selv å se på dataene og se hvordan de ser ut. Vi kan se på objektet *ungdata_alko* ved å skrive navnet på objektet i konsollen. Da vil R i utgangspunktet skrive *hele* datasettet i konsollen.

```
ungdata_alko
```

```
# A tibble: 845,100 x 5
   klasse    ar    kjonn  alko1                              drikker
   <fct>     <fct> <fct>  <fct>                                <dbl>
 1 9. trinn 2014  Gutter Aldri                                    0
 2 9. trinn 2014  Gutter Aldri                                    0
 3 8. trinn 2014  Gutter Aldri                                    0
 4 9. trinn 2014  Gutter Aldri                                    0
 5 8. trinn 2014  Gutter Aldri                                    0
 6 9. trinn 2014  Jenter Aldri                                    0
 7 8. trinn 2014  Gutter Aldri                                    0
 8 8. trinn 2014  Gutter Har bare smakt noen få ganger            1
 9 8. trinn 2014  Jenter Aldri                                    0
10 9. trinn 2014  Jenter Har bare smakt noen få ganger            1
# i 845,090 more rows
```

Det er imidlertid sjelden hensiktsmessig å se på hele datasettet på denne måten. Det er for det første ikke plass til å vise hele datasettet i konsollen, og for det andre er det ikke så lett å få oversikt over datasettet på denne måten. Hvis du virkelig vil se på hele datasettet er det bedre å bruke `View`-funksjonen som åpner datasettet i et eget vindu.

```
View(ungdata_alko)
```



	klasse Hvilket klassetrinn går du i?	alko1 Hender det at du drikker noen form for alkohol?	ar Årstall for undersøkelsen	drikker
1	9. trinn	Aldri	2014	0
2	9. trinn	Aldri	2014	0
3	8. trinn	Aldri	2014	0
4	9. trinn	Aldri	2014	0
5	8. trinn	Aldri	2014	0
6	9. trinn	Aldri	2014	0
7	8. trinn	Aldri	2014	0
8	8. trinn	Har bare smakt noen få ganger	2014	1
9	8. trinn	Aldri	2014	0
10	9. trinn	Har bare smakt noen få ganger	2014	1
11	9. trinn	Aldri	2014	0
12	8. trinn	Har bare smakt noen få ganger	2014	1

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

Det er vanligvis ikke så nyttig å se på datasettet på denne måten heller. Det er derfor vanligvis mer hensiktsmessig å se på en del av datasettet med å bare be om å få se de første observasjonene. Da får du et innblikk i datastrukturen, variable og verdier. Dette gjøres med funksjonen `head`.

```
head(ungdata_alko)
```

```
# A tibble: 6 x 5
  klasse   ar    kjonn  alko1 drikker
  <fct>    <fct> <fct>  <fct>   <dbl>
1 9. trinn 2014  Gutter Aldri       0
2 9. trinn 2014  Gutter Aldri       0
3 8. trinn 2014  Gutter Aldri       0
4 9. trinn 2014  Gutter Aldri       0
5 8. trinn 2014  Gutter Aldri       0
6 9. trinn 2014  Jenter Aldri       0
```

Hvis det er mange variable i datasettet vil det ikke bli plass i consoll-vinduet til å vise alle variablene. Da vil R bare vise de første variablene og skrive at det er flere variable som ikke vises. Da kan det være mer hensiktsmessig å bruke funksjonen `glimpse` som viser variabelnavnene i rader, med de tilhørende første verdiene.

```
glimpse(ungdata_alko)
```

```
Rows: 845,100
Columns: 5
$ klasse  <fct> 9. trinn, 9. trinn, 8. trinn, 9. trinn, 8. trinn, 9. trinn, 8.~
$ ar      <fct> 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 20~
$ kjonn   <fct> Gutter, Gutter, Gutter, Gutter, Gutter, Jenter, Gutter, Gutter~
$ alko1   <fct> "Aldri", "Aldri", "Aldri", "Aldri", "Aldri", "Aldri", "Aldri",~
$ drikker <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0,~
```

`glimpse` gir også noe ytterligere informasjon, som antall observasjoner i datasettet og og hvilken type variablene er. Når det i output står "dbl" betyr at det er en numerisk variabel, og "fct" betyr at det er en faktorvariabel.

Det finnes også andre variabeltyper enn det som er i eksempelet, herunder betyr "chr" at det er en tekstvariabel, "int" betyr at det er en heltallsvariabel, "date" betyr at det er en dato-variabel, og "lgl" betyr at det er en logisk variabel (dvs. en variabel som kan ha verdiene TRUE eller FALSE). Vi kommer tilbake til disse variabeltypene etterhvert.

Funksjonen `class()` gir informasjon om hva slags objekt man har. Her sjekkes objektet *ungdata_alko*:

```
class(ungdata_alko)
```

```
[1] "tbl_df"     "tbl"        "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(ungdata_alko)
```

```
[1] 845100      5
```

Her får vi vite at det er 845100 rader (dvs. observasjoner) og 5 kollonner (dvs. variable).

3.2.1 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.

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.

```
library(memisc)
codebook(ungdata_alko$alko1)
```

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

   ungdata_alko$alko1 'Hender det at du drikker noen form for alkohol?'

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

   Storage mode: integer
   Factor with 5 levels

   Levels and labels                                    N  Valid

   1 'Aldri'                                       374957   44.4
   2 'Har bare smakt noen få ganger'               174322   20.6
   3 'Av og til, men ikke så ofte som månedlig'    145934   17.3
   4 'Nokså jevnt 1-3 ganger i måneden'            112311   13.3
   5 'Hver uke'                                     37576    4.4
```

Grunnen til å bruke `codebook` er å få et raskt innblikk i enkeltvariable, inkludert fordelingen av verdier. Dette er mest informativt for kategoriske variable eller numeriske variable med relativt få verdier.

4 Grafikk med ggplot

I dette kapittelet skal vi bruke følgende pakker:

```
library(tidyverse)
library(ggforce)
library(ggthemes)
library(datasets)
```

I R er det flere systemer for grafikk. Noen er spesialiserte og knyttet til spesielle analysemetoder og gir deg et spesifikt output slik at funksjonen `plot()` gir deg akkurat det du skal ha. Det er forsåvidt veldig praktisk, men er ikke gode nok til å bli med i en rapport.

Vi skal her vektlegge et *generelt system* for grafikk som finnes i pakken {ggplot2} som er en del av tidyverse. Funksjonen `ggplot` kan brukes til all slags grafikk, fra helt grunnleggende til avansert, professjonell grafikk. Funksjonen `ggplot` er bygget opp som en *gramatikk* for grafisk fremstilling. Det ligger altså 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 dekke de mest grunnleggende typer plot her.

4.1 Lagvis grafikk

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.

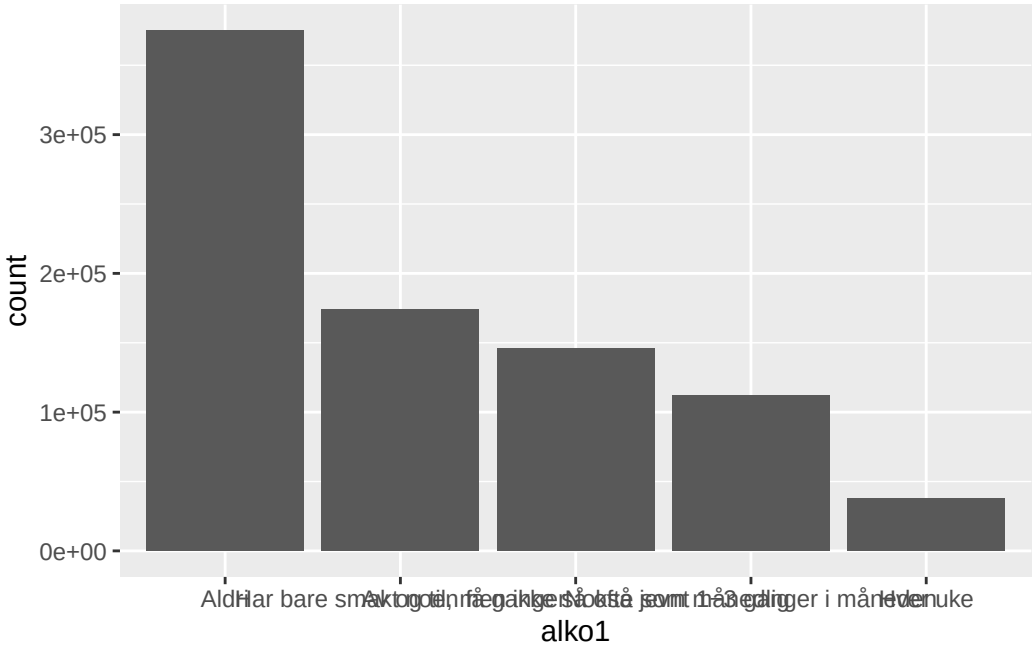
- Det første argumentet i `ggplot` er data. Altså: hvilket datasett informasjonen hentes fra.
- Inni `ggplot()` må det spesifiseres `aes()`, "*aestethics*", som er *hvilke variable* som skal plottes. Først og fremst hva som skal på x-akse og y-akse (og evt. z-akse), men også spesifikasjon av om linjer (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

4.2 Stolpediagram

Eksempeldataene her er fra Ungdata 2010-20202, og vi skal se på selvrapportert bruk av alkohol. Variabelen *alko1* er for spørsmålet om hvor ofte intervjupersonenene drikker alkohol, og det er en kategorisk variabel med 5 kategorier: Aldri, sjelden, av og til, ofte og veldig ofte.

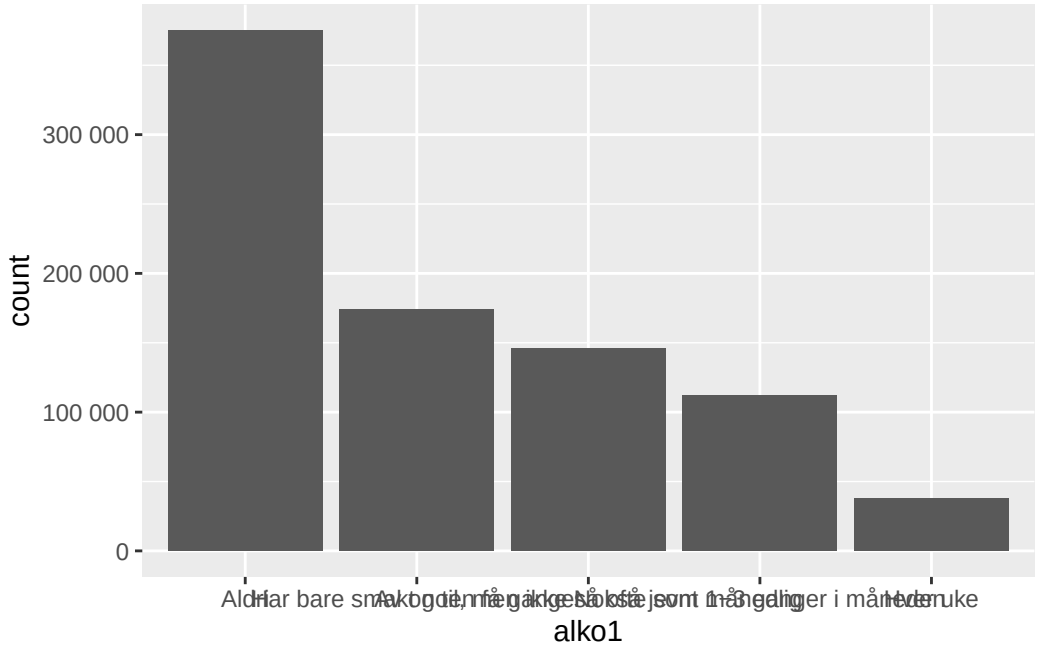
Vi lager et plot med funksjonen `ggplot` med koden nednefor. Første linje spesifiserer hva som skal plottes. Her er det første argumentet datasettet som skal brukes, og det andre er `aes()` som står for *aestethics* og er en funksjon som spesifiserer hvilke variable som skal brukes. Her er det bare en variabel som skal plottes, og det er *alko1* som plasseres langs x-aksen. Det andre argumentet er `geom_bar` som spesifiserer at det skal være stolpediagram. Når man skriver kun `geom_bar` er en forkortelse for `geom_bar(stat = "count")` som betyr at det skal være en stolpe for hver kategori og høyden på stolpen skal være antall observasjoner i hver kategori.

```
ggplot(ungdata_alko, aes(x = alko1)) +
  geom_bar()
```



Merk at y-aksen har fått antallene notert i scientific notation. Det er ikke så pent. Det kan vi fikse på med `scale_y_continuous()` som gir oss mulighet til å spesifisere hvordan akseverdiene skal se ut. Pakken {scales} inneholder en rekke funksjoner for å formatere tallverdier. Her bruker vi `number_format` som spesifiserer nøyaktighet til 1, dvs. heltall.

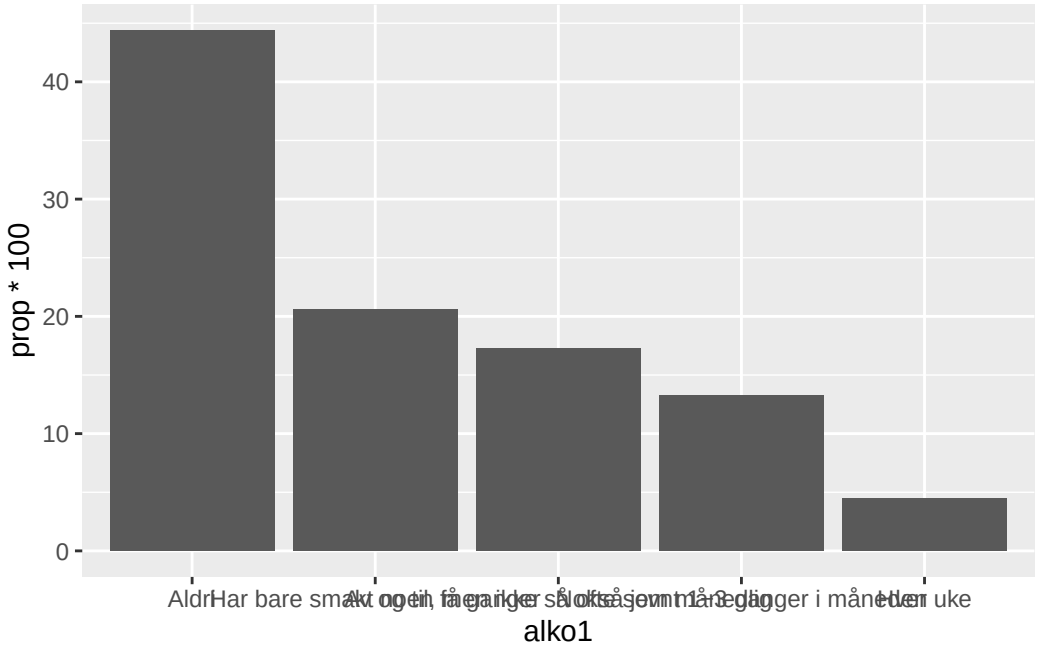
```
ggplot(ungdata_alko, aes(x = alko1)) +
  geom_bar() +
  scale_y_continuous(labels = scales::number_format(accuracy = 1))
```



Det er også mulig å spesifisere at det skal være andeler i stedet for antall. I ggplot finnes det noen spesielle funksjoner som starter med `..` deriblant `..prop..` som gir andeler. Men for at det skal regnes ut riktig må vi også spesifisere at det ikke er andelen i hver kategori, men andelen totalt. Det gjøres med `group = 1` som betyr at alle observasjonene skal grupperes sammen. For å få prosent i stedet for desimaltall ganger vi med 100.

```
ggplot(ungdata_alko, aes(x = alko1, y = ..prop..*100, group = 1)) +
  geom_bar()
```

```
Warning: The dot-dot notation (`..prop..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(prop)` instead.
```




```
# ggplot(ungdata_alko, aes(x = alko1, y = after_stat(prop)*100, group = 1)) +
#   geom_bar()
```

Nå gjenstår det litt ryddejobb og vi gjør disse tingene samtidig. Merk at x-aksen har noen lange tekststrenger som blir uleselige. Det kan vi fikse på med `scale_x_discrete` der vi kan spesifisere label og bruke funksjonen `str_wrap` fra pakken {stringr} som bryter opp tekststrengen etter en gitt bredde. Her har vi satt bredde til 20. Koden er litt komplisert for nybegynnere i R, så det er ikke så farlig om du ikke skjønner alt. Den linjen kan brukes i andre sammenhenger og bare endre `width =` etter behov.

Det neste som trengs er å endre aksetekstene. Det gjøres med `labs()` der vi kan spesifisere x- og y-akse og også tittel og kilde.

```
ggplot(ungdata_alko, aes(x = alko1, y = ..prop..*100, group = 1)) +
  geom_bar() +
  scale_x_discrete(labels = function(x) str_wrap(x, width = 20)) +
  labs(x = "",
       y = "Prosent",
       title = "Antall som drikker alkohol",
       caption = "Kilde: Ungdata 2010-2020")
```

Kilde: Ungdata 2010–2020

4.3 Stolpediagram med flere variable

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 *kjonn* og at fyllfargen skal settes etter denne variablen med `fill = kjonn`.

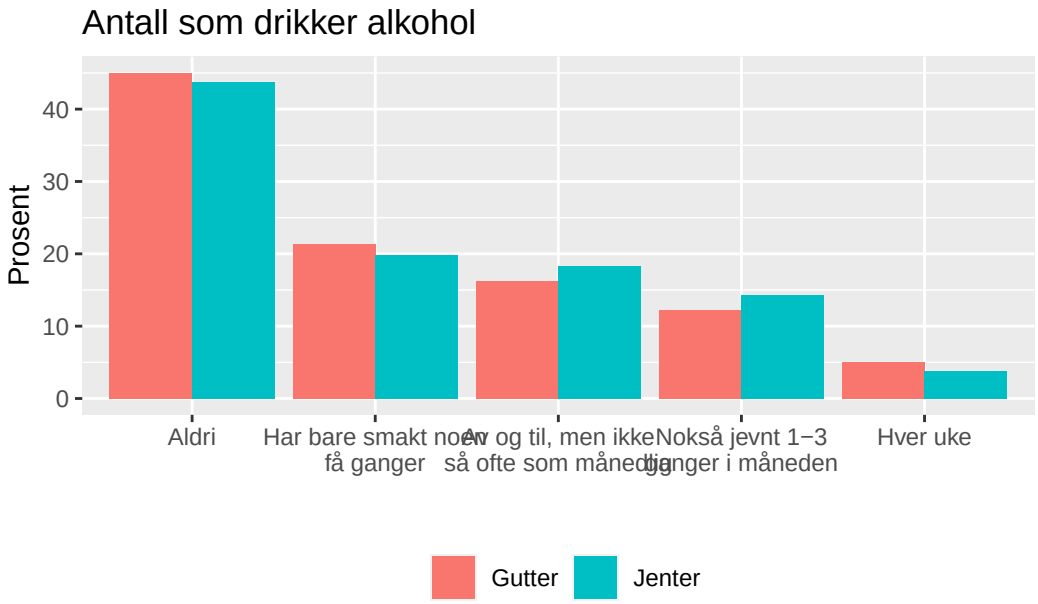
Det er også mulig å spesifisere at stolpene skal plasseres ved siden av hverandre i stedet for oppå hverandre. Det gjøres med `position = "dodge"`. [1]

Tegnforklaringen plasseres automatisk til høyre for plottet. Vi kan imidlertid flytte den til bunnen med `theme(legend.position = "bottom")` og fjerne tittelen på tegnforklaringen med `legend.title = element_blank()`.

```
ggplot(ungdata_alko, aes(x = alko1, y = ..prop..*100, group = kjonn, fill = kjonn)) +
  geom_bar(position = "dodge") +
  scale_x_discrete(labels = function(x) str_wrap(x, width = 20)) +
  labs(x = "",
       y = "Prosent",
       title = "Antall som drikker alkohol",
```

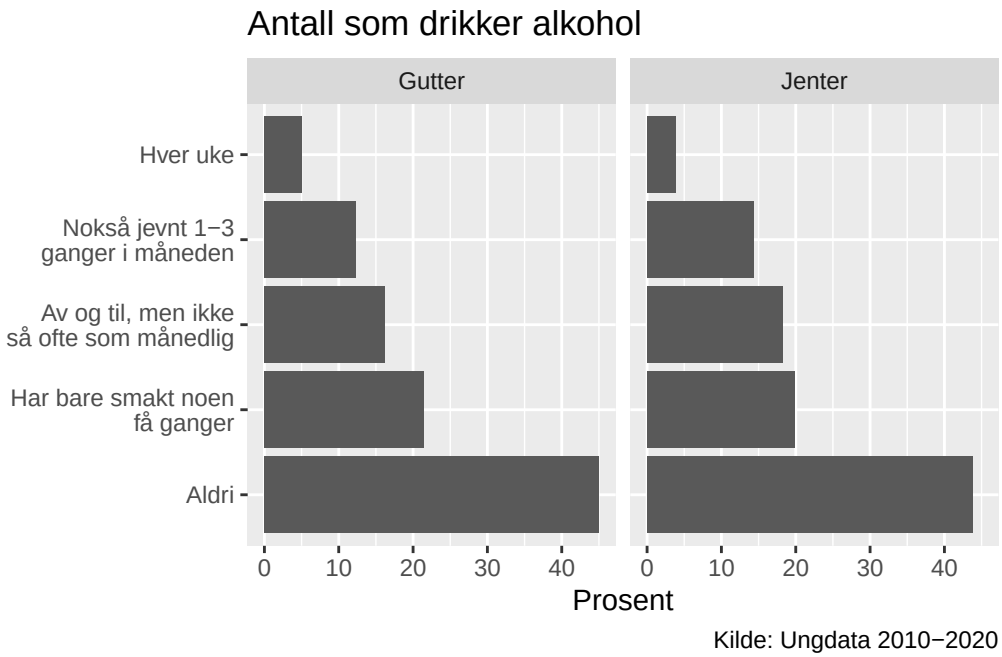
[1] Alternativet, som er det automatisk forvalget, er `position = "stack"` hvor stolpene ville blitt plassert oppå hverandre.

```
      caption = "Kilde: Ungdata 2010-2020") +
  theme(legend.position = "bottom", legend.title = element_blank())
```



Et alternativ er å plassere grafikken for menn og kvinner ved siden av hverandre. Å legge til `facet_wrap` gjør dette. Da blir det trangere på x-aksen til teksten, så en mulighet er da å snu plottet med `coord_flip`.

```
ggplot(ungdata_alko, aes(x = alko1, y = ..prop..*100, group = 1)) +
  geom_bar() +
  scale_x_discrete(labels = function(x) str_wrap(x, width = 20)) +
  labs(x = "",
       y = "Prosent",
       title = "Antall som drikker alkohol",
       caption = "Kilde: Ungdata 2010-2020") +
  facet_wrap(~kjonn) +
  coord_flip()
```



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:

4.3.1 Kakediagram

Generelt er ikke kakediagram å anbefale da korrekt tolkning involverer å tolke et areal som inneholder vinkel. Det er intuitivt vanskelig å se hvor store hvert kakestykke er med det blotte øyet - med mindre man skriver tallene på,da. Men da kunne man jo også bare laget en tabell?

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.

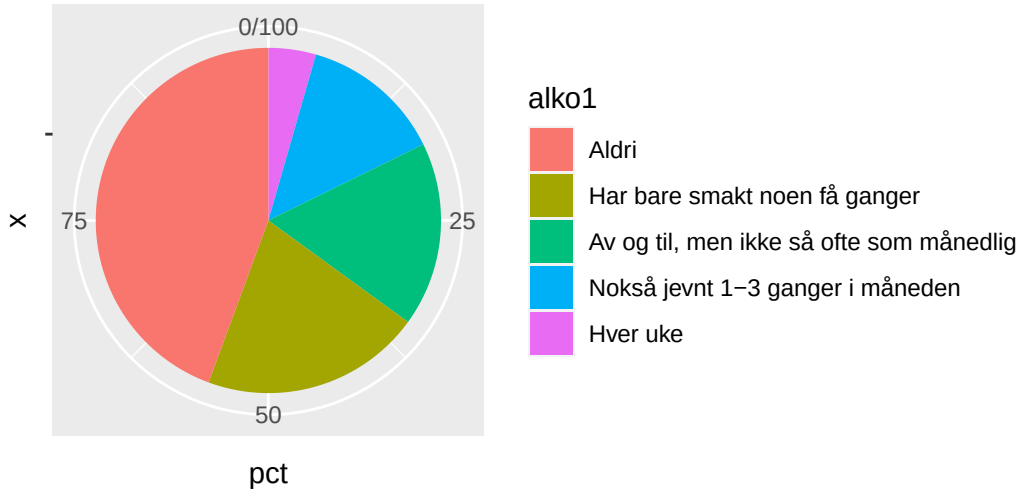
Første steg er å lage et aggregert datasett med antall observasjoner og prosent i hver kategori, som vist nedenfor. Dette dekker vi i et senere kapittel og fokuserer på selve plottet her.

```
# A tibble: 5 x 4
  alko1                                    n   pct lab.pos
  <fct>                                <int> <dbl>   <dbl>
```

```
1 Hver uke                                   37576  4.45    2.22
2 Nokså jevnt 1-3 ganger i måneden          112311 13.3    11.1
3 Av og til, men ikke så ofte som månedlig  145934 17.3    26.4
4 Har bare smakt noen få ganger             174322 20.6    45.3
5 Aldri                                     374957 44.4    77.8
```

Utgangspunktet er et stolpediagram som vi har laget ovenfor. Det er bare å bytte ut `geom_bar` med `geom_col` som er en forkortelse for `geom_col(stat = "identity")`. Det betyr at høyden på stolpene er gitt i datasettet. For å lage et kakediagram må vi også legge til `coord_polar` som gjør at det blir sirkulært.

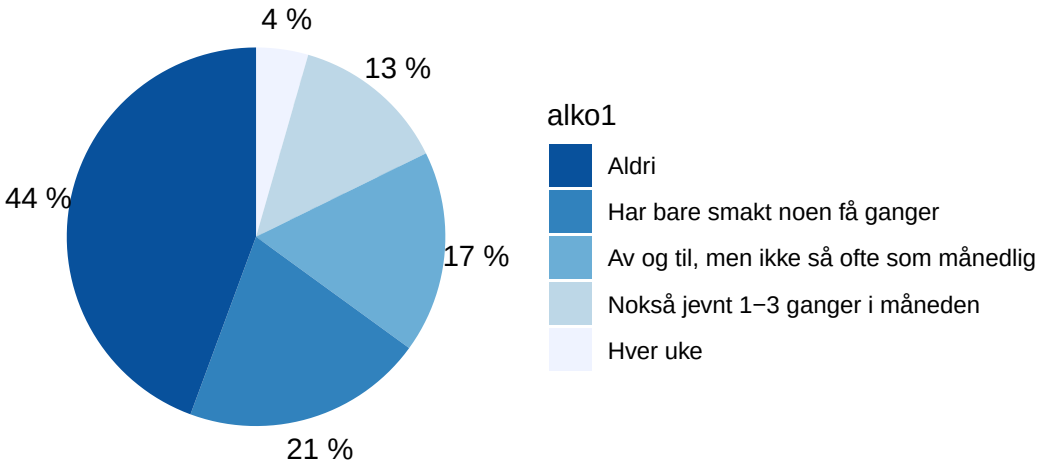
```
ggplot(alko1_sum, aes(x = "", y = pct, fill = alko1)) +
  geom_col() +
  coord_polar("y", start=0)
```



For å gjøre plottet penere fjerner vi hjelpelinjene, legger til prosenttallene innenfor kakestykkene og endrer fargepaletten.

```
ggplot(alko1_sum, aes(x = "", y = pct, fill = alko1)) +
  geom_col() +
  coord_polar("y", start=0) +
```

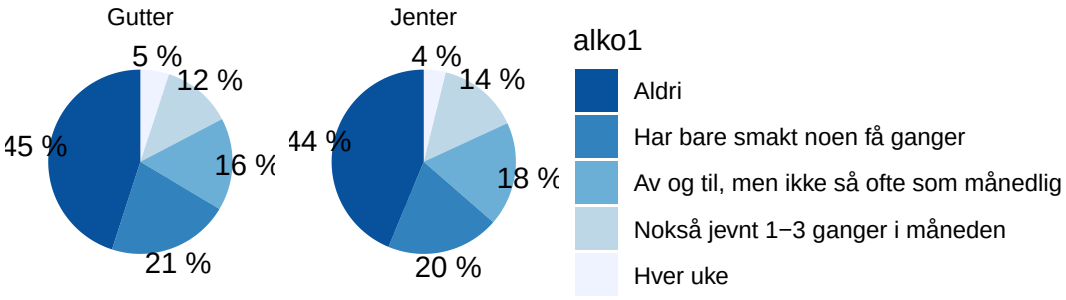
```
  theme_void()+
  geom_text(aes(y = lab.pos, x=1.6, label = paste(round(pct), "%" )))+
  scale_fill_brewer(palette="Blues", direction = -1)
```



Det er også mulig å lage et kakediagram for hver gruppe. Da må vi legge til `facet_wrap` som vi har brukt tidligere. Det forutsetter at dataene er tilsvarende laget for hver gruppe. Objektet *alko1_sum2* har aggregert data for kjønn.

```
# A tibble: 10 x 5
   alko1                                  kjonn       n   pct lab.pos
   <fct>                                  <fct>   <int> <dbl>   <dbl>
 1 Hver uke                               Jenter  16528  3.86    1.93
 2 Hver uke                               Gutter  21048  5.05    2.53
 3 Nokså jevnt 1-3 ganger i måneden        Gutter  51195 12.3    11.2
 4 Nokså jevnt 1-3 ganger i måneden        Jenter  61116 14.3    11.0
 5 Av og til, men ikke så ofte som månedlig Gutter  67666 16.2    25.5
 6 Av og til, men ikke så ofte som månedlig Jenter  78268 18.3    27.3
 7 Har bare smakt noen få ganger           Jenter  85078 19.9    46.3
 8 Har bare smakt noen få ganger           Gutter  89244 21.4    44.3
 9 Aldri                                   Jenter 187380 43.7    78.1
10 Aldri                                   Gutter 187577 45.0    77.5
```

```
ggplot(alko1_sum2, aes(x = "", y = pct, fill = alko1)) +
  geom_col() +
  coord_polar("y", start=0) +
  theme_void()+
  geom_text(aes(y = lab.pos, x=1.6, label = paste(round(pct), "%" )))+
  scale_fill_brewer(palette="Blues", direction = -1) +
  facet_wrap(~kjonn)
```



4.4 Grafikk for kontinuerlige data

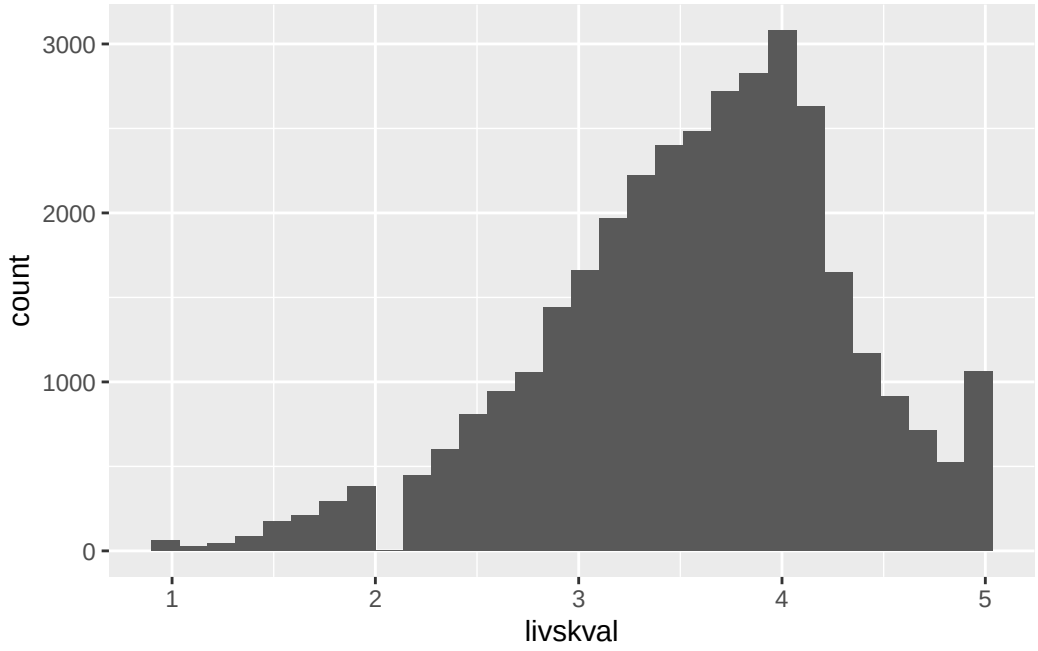
Vi skal nå se på hvordan vi kan fremstille kontinuerlige data. Vi bruker et annet uttrekk fra Ungdata der en rekke spørsmål er aggregert til en indeks for livskvalitet og en indeks for atferdsproblemer. Indeksene er laget ved å ta gjennomsnittet av de enkelte spørsmålene, og konstruert slik at høyere verdier betyr bedre livskvalitet og færre atferdsproblemer.

4.4.1 Histogram

Histogram er en vanlig måte å fremstille kontinuerlige data på. Det er en måte å gruppere dataene i intervaller og høyden på stolpene representerer antall observasjoner i hvert intervall. Alternativt kan stolpene representere *andelen* i hvert intervall eller *tettheten*.

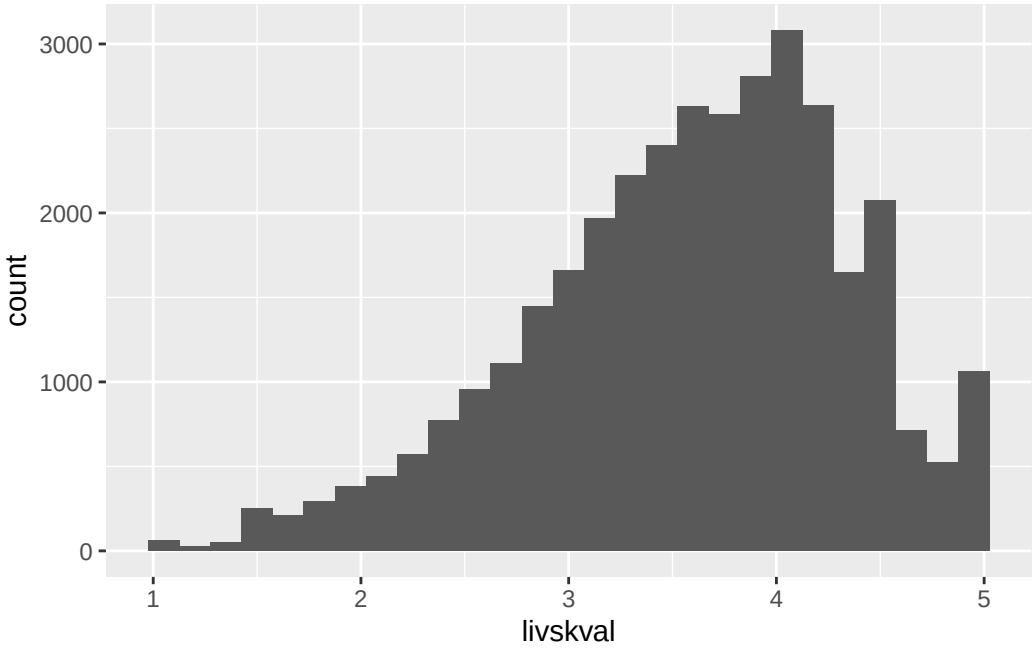
Vi starter med å lage et histogram for livskvalitet. Det gjøres med `geom_histogram`.

```
ggplot(ungdata_kont, aes(x = livskval)) +
  geom_histogram()
```



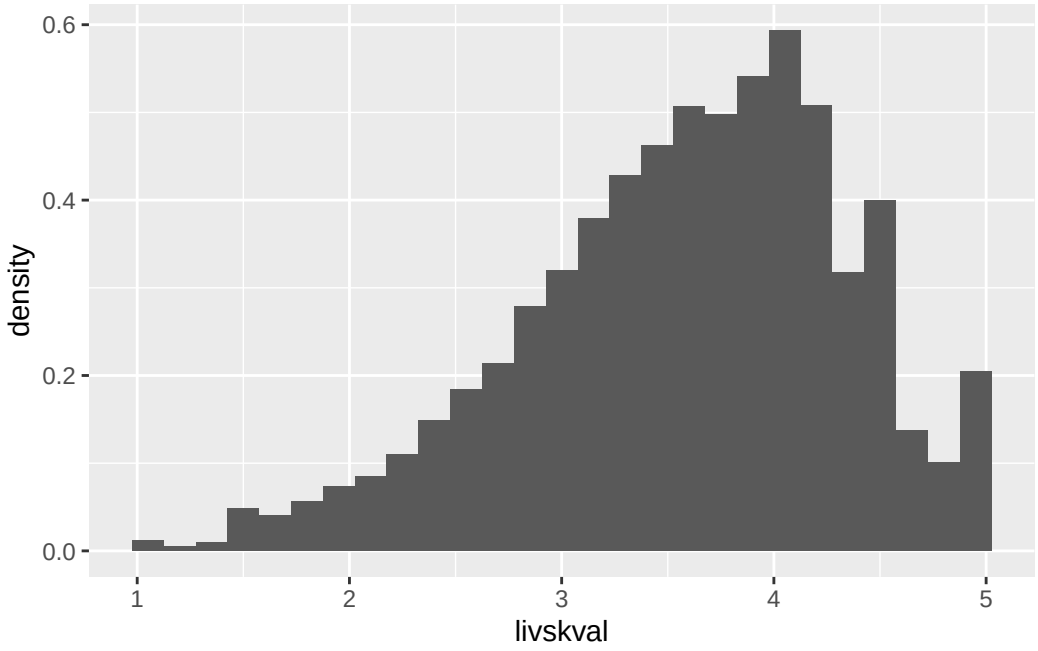
Antall og bredden på intervallene bestemmes automatisk av R. Det er ikke alltid optimale valg, og det kan være greit å justere dette selv. Det gjøres med `binwidth` som angir bredden på intervallene. I eksempelet over ser fordelingen litt pussig ut med et tomt intervall, men det kan skyldes hvordan intervallene tilfeldigvis ble plassert. Hvis plottet ser pussig ut bør man sjekke om det er fordi intervallene er for brede eller smale. Følgende kode gjør en liten endring, og du kan selv sjekke med ulike verdier for `binwidth` og se hvordand et påvirker resultatet.

```
ggplot(ungdata_kont, aes(x = livskval)) +
  geom_histogram(binwidth = .15)
```



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.

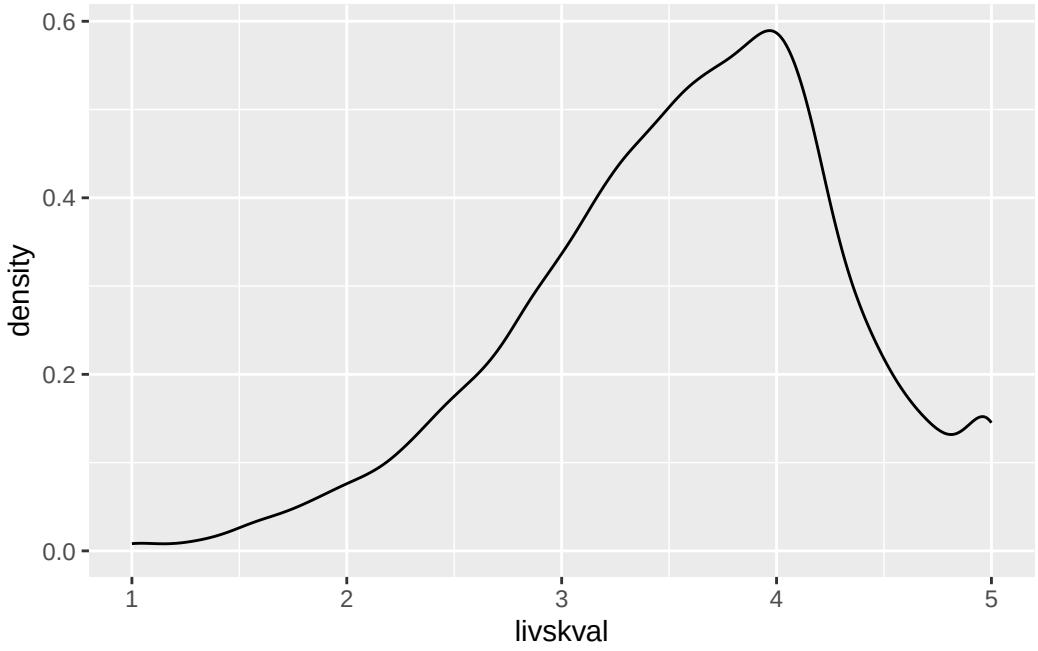
```
ggplot(ungdata_kont, aes(x = livskval, y = ..density..)) +
  geom_histogram(binwidth = .15)
```



4.4.2 Density plot

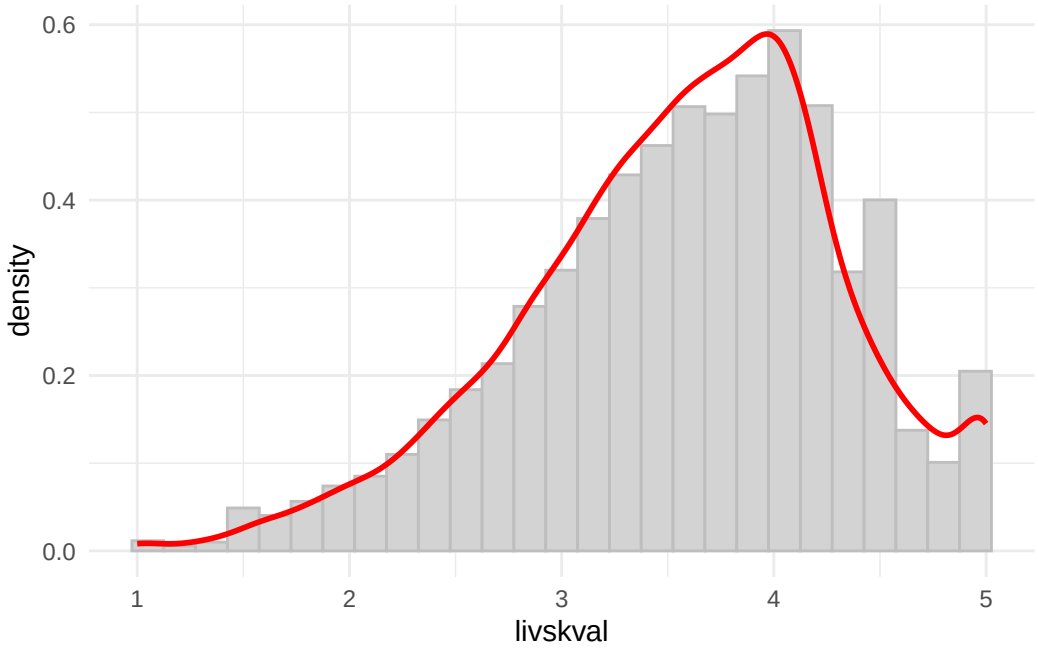
Density plot er en måte å fremstille det samme på, men i stedet for å dele inn i intervaller som i histogram lager vi en glattet kurve. Det blir på skalaen "tetthet" som i histogrammet ovenfor. På tilsvarende måte som `binwidth` kan vi justere hvor glatt kurven skal være med `bw` som står for *bandwidth*. Høyere tall gir mer glattet kurve, mens lavere tall gir mer detaljert kurve. Vanligvis vil det automatiske forvalget som R gjør for deg være bra nok, men det kan også være aktuelt å justere dette selv som vist nedenfor.

```
ggplot(ungdata_kont, aes(x = livskval)) +
  geom_density(bw = .1)
```



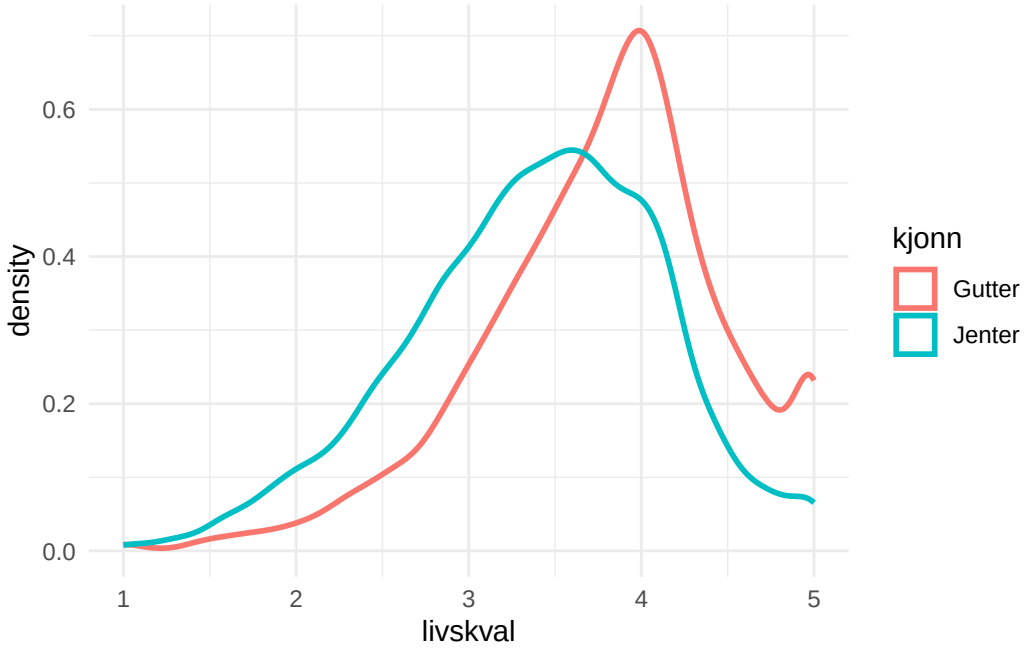
Vi kan legge et histogram og density plot oppå hverandre da y-skalen er lik. Da ser man lettere at det er samme informasjon som fremstilles på ulike måter.

```
ggplot(ungdata_kont, aes(x = livskval)) +
  geom_histogram(aes(y = ..density..), binwidth = .15,  fill = "lightgrey", col = "grey") +
  geom_density(col = "red", linewidth = 1, bw = .1) +
  theme_minimal()
```



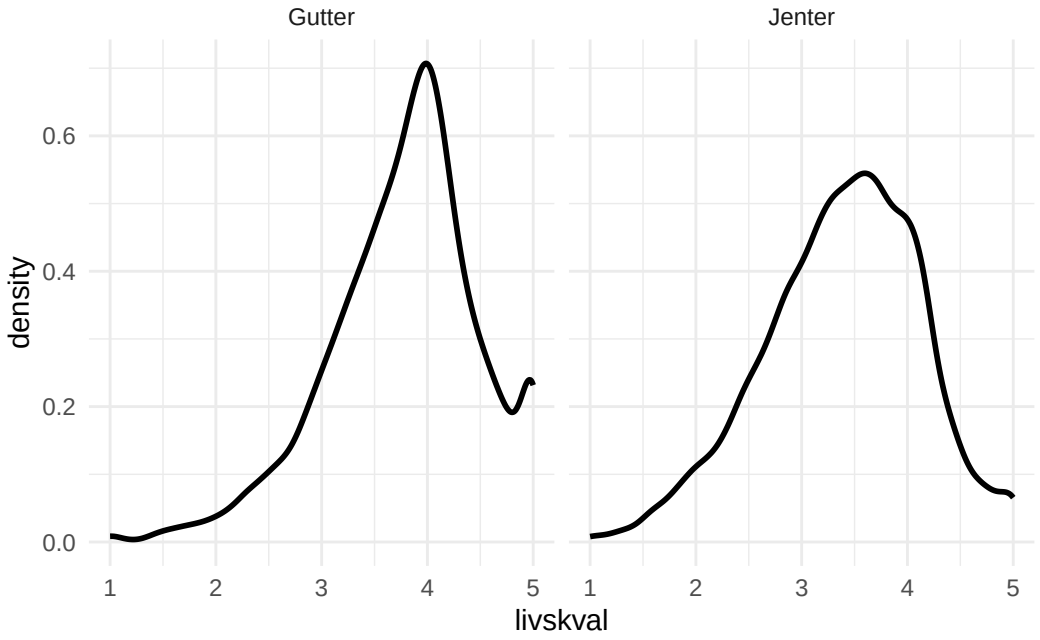
En fordel med tetthetsplot er at det er lettere å sammenligne grupper. Her er et eksempel som viser fordelingen av livskvalitet etter kjønn. Vi ser at gutter gjennomgående rapporterer høyere livskvalitet enn jenter. Begge kurvene er skjeve med en lang hale til venstre, men kurven for gutter er skjevere enn for jenter.

```
ggplot(ungdata_kont, aes(x = livskval, group = kjonn, color = kjonn)) +
  geom_density(linewidth = 1, bw = .1)+
  guides(fill = guide_legend(override.aes = list(shape = 1 ) ) ) +
  theme_minimal()
```



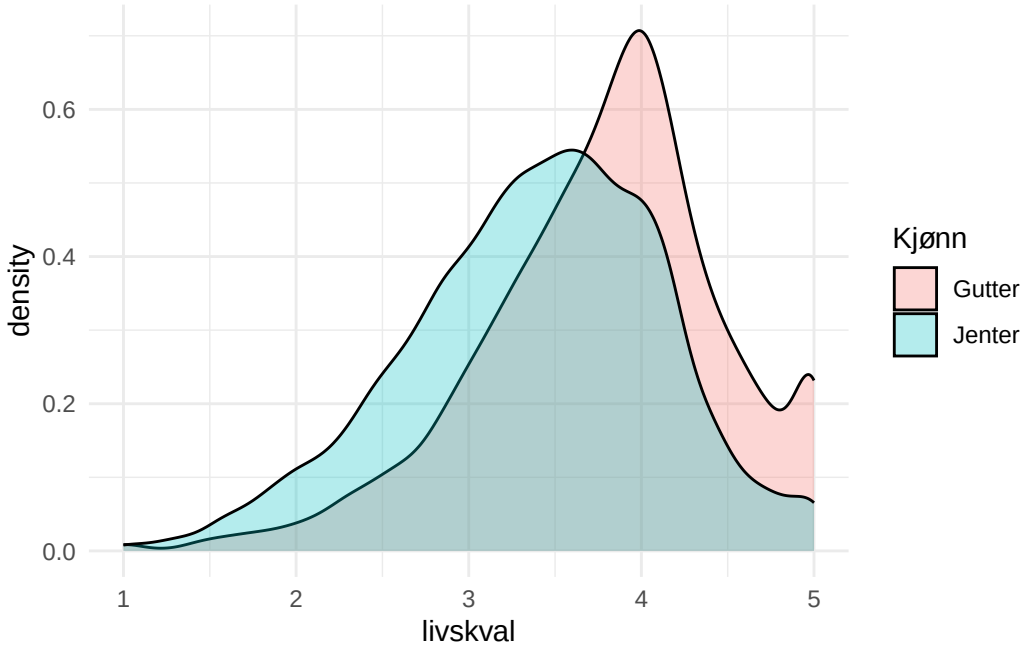
Vi har tidligere vist bruken av `facet_wrap` for å lage flere plott ved siden av hverandre. Det kan også brukes her. Da må vi legge til `facet_wrap(~kjonn)` som betyr at vi vil ha to plott, ett for hver verdi av kjønn. Hvilken fremstilling som er best i et gitt tilfelle kommer an på hva man ønsker fremheve og hva som er mest informativt. Generelt sett er det å legge plottene oppå hverandre best når man ønsker gjøre en direkte sammenligning, mens ved siden av hverandre hvis man ønsker å se på hver gruppe for seg.

```
ggplot(ungdata_kont, aes(x = livskval)) +
  geom_density(linewidth = 1, bw = .1)+
  theme_minimal()+
  facet_wrap(~kjonn)
```



En variant av samme fremstilling er å bruke fyllfarge i stedet for linjefarge. Da må vi legge til `fill = kjonn` i `aes()`. Det er også mulig å endre tittel på tegnforklaring fra variabelnavnet til noe mer gramatisk korrekt med `guides(fill=guide_legend(title="Kjønn"))`.

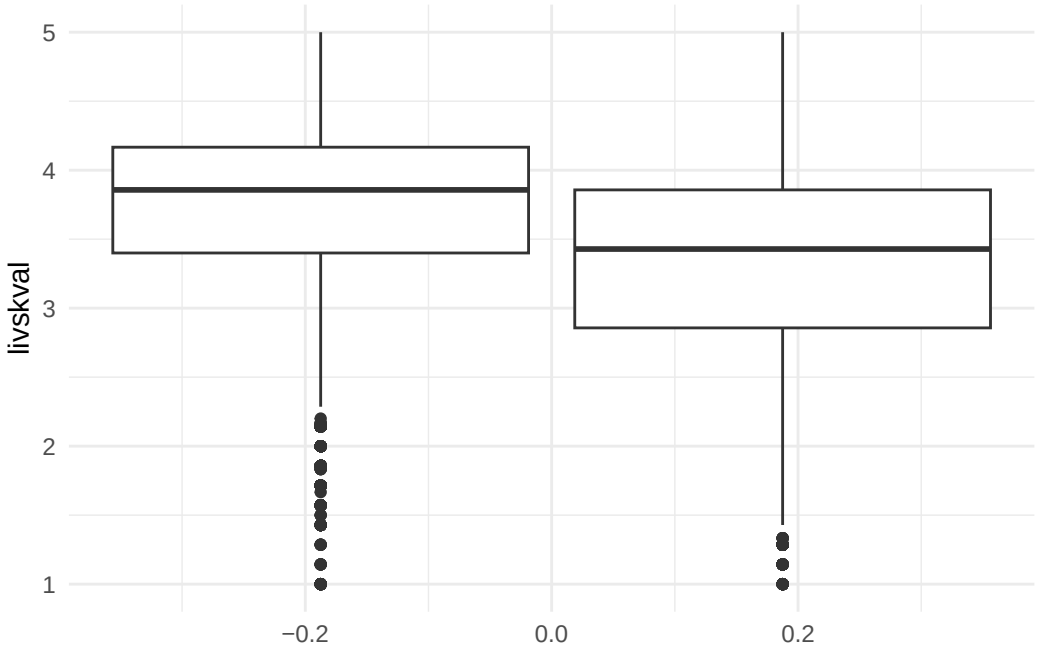
```
ggplot(ungdata_kont, aes(x = livskval, group = kjonn,  fill = kjonn)) +
  geom_density(alpha = .3, bw = .1)+
  guides(fill=guide_legend(title="Kjønn"))+
  theme_minimal()
```



4.4.3 Flere variable samtidig

4.4.3.1 Boksplot

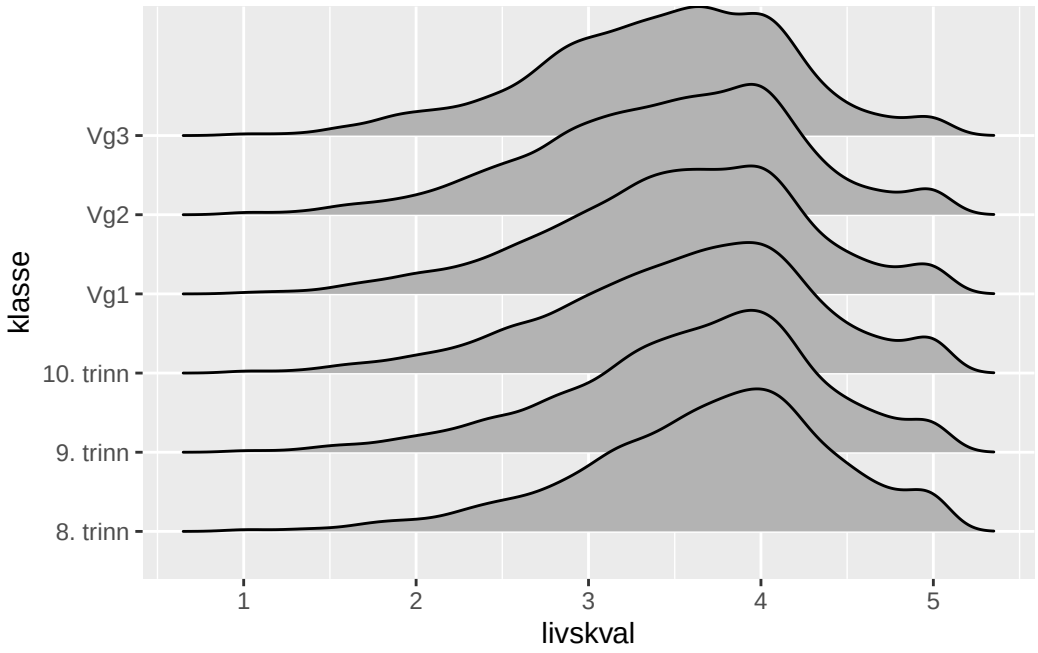
```
ggplot(ungdata_kont, aes(y = livskval, group = kjonn)) +
  geom_boxplot()+
  theme_minimal()
```



4.4.3.2 Ridgeplot

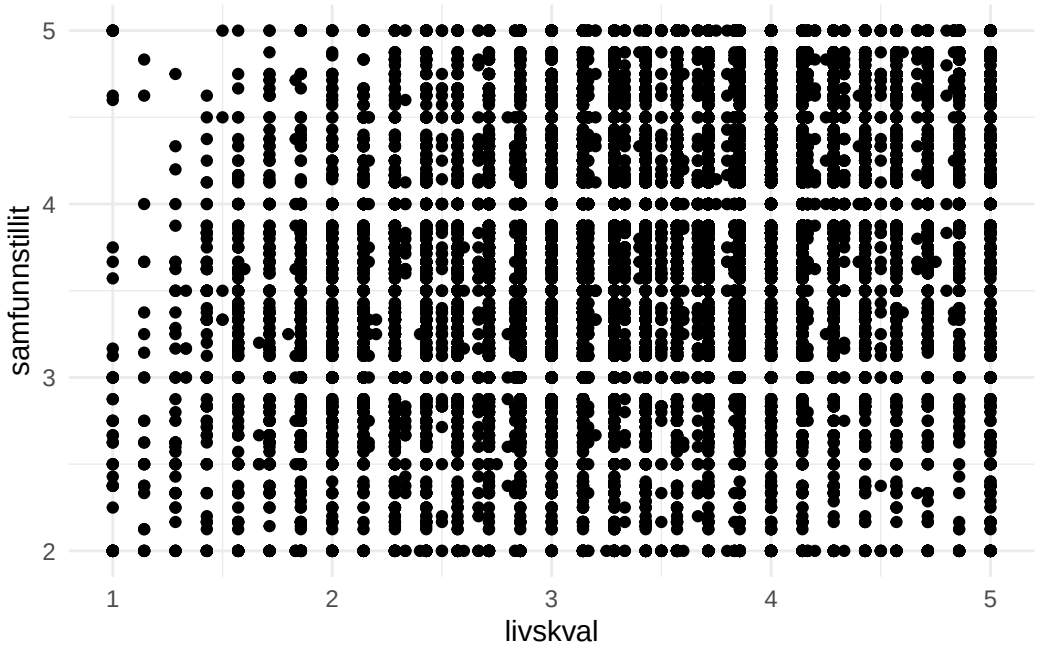
Ridgeplot er en annen måte å sammenligne en kontinuerlig fordeling betinget på en gruppering.

```
library(ggridges)
ggplot( ungdata_kont, aes(x = livskval, y = klasse)) +
  geom_density_ridges()
```



4.4.3.3 Scatterplot

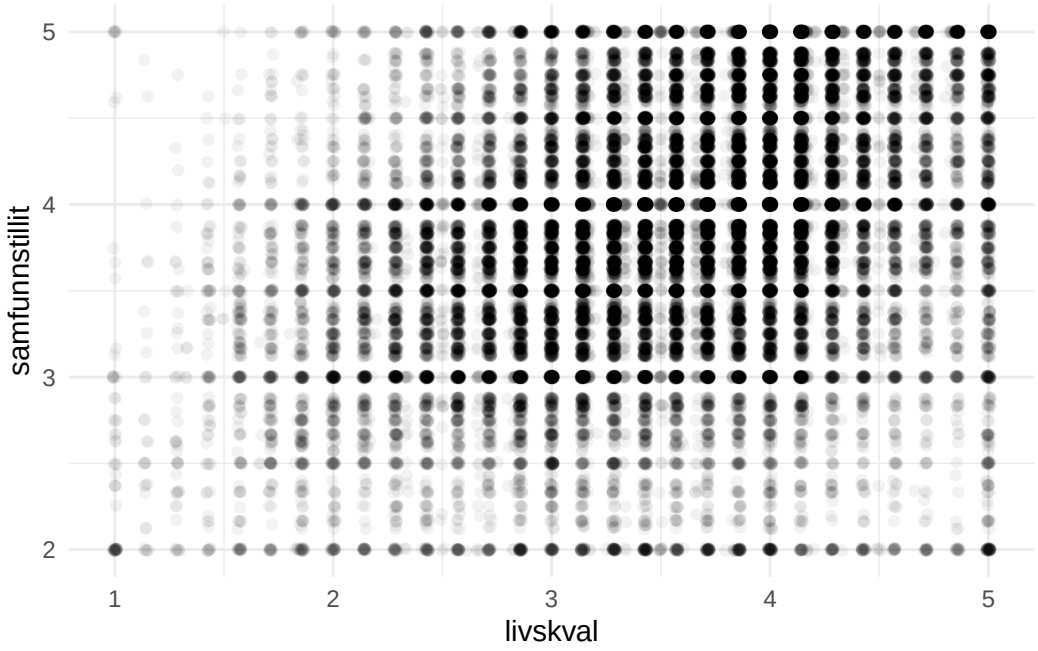
```
ggplot(ungdata_kont, aes(x = livskval, y = samfunnstillit)) +
  geom_point()+
  theme_minimal()
```



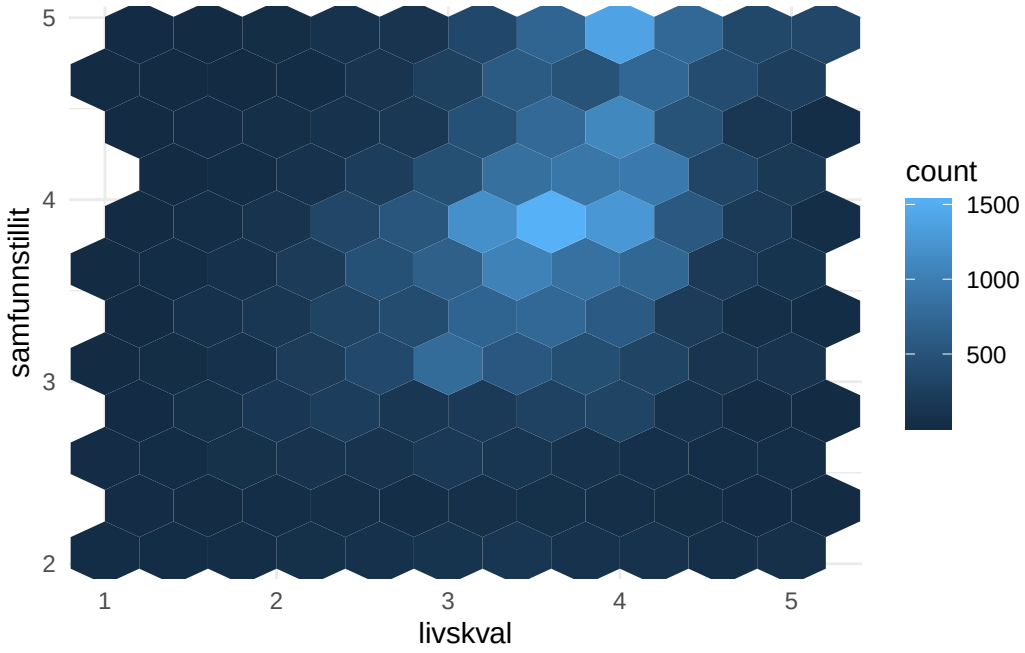
Når det er mange observasjoner kan det bli litt rotete med punkter som overlapper hverandre. Det er vanskelig å se noen mønstre i et slikt plot. Det kan bedres med noen teknikker for å fremheve nettopp hvor de fleste datapunktene er. I det følgende gjør vi to ting samtidig: legger til gjennomsiktig farge på punktene med `alpha =` som angir grad av gjennomsiktighet. Når alpha er 1 er det ingen gjennomsiktighet, og ved 0 er det helt gjennomsiktig. Her må man prøve seg frem. I tillegg bruker vi `geom_jitter` som legger til litt tilfeldig støy på hvert datapunkt slik at de ikke så ofte ligger akkurat oppå hverandre.

Her har vi valgt en veldig høy grad av gjennomsiktighet slik at relativt vanlige verdier synes tydelig, mens mer sjeldne verdier blir nesten usynlige.

```
ggplot(ungdata_kont, aes(x = livskval, y = samfunnstillit)) +
  geom_jitter(alpha=.05)+
  theme_minimal()
```

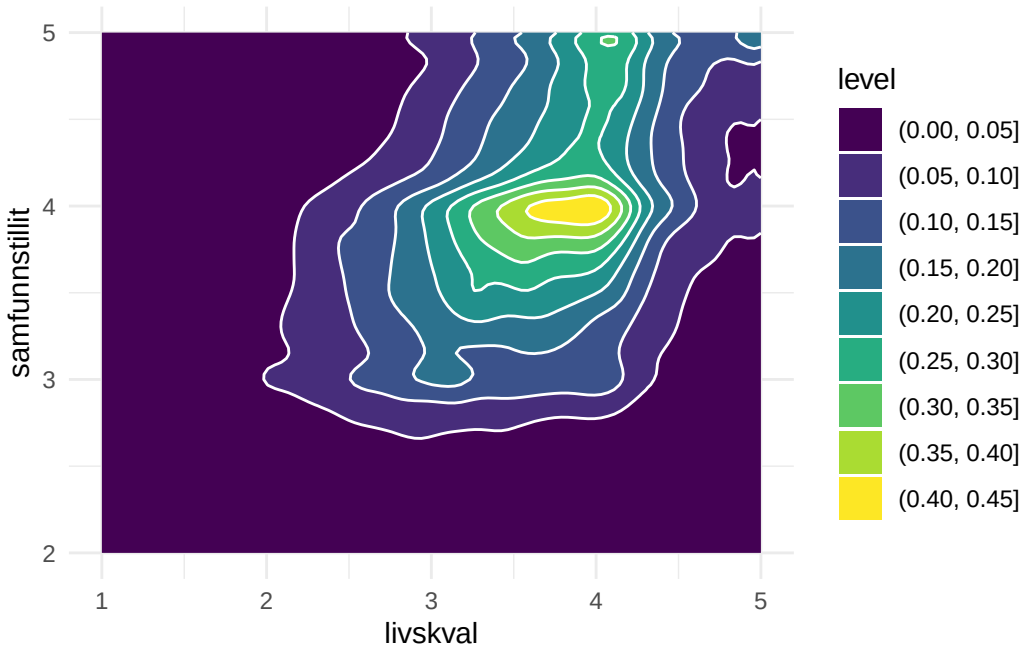



```
ggplot(ungdata_kont, aes(x = livskval, y = samfunnstillit)) +
  geom_hex(bins = 10)+
  theme_minimal()
```



Det er også mulig å legge til en glattet kurve som viser tettheten av punktene. Det gjøres med `geom_density_2d_filled` som lager et konturplot der arealet under kurven er fargekodet. Det er også mulig å legge til konturlinjer med `geom_density_2d`. Nedenfor er begge brukt samtidig, men man kan også velge bare en av dem.

```
ggplot(ungdata_kont, aes(x = livskval, y = samfunnstillit)) +
  geom_density_2d_filled()+
  geom_density_2d(col = "white")+
  theme_minimal()
```



NOVA, and Anders Bakken. 2023. "Ungdata 2010-2023." https://doi.org/10.18712/NSD-NSD3157-V1; Sikt.

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59. https://doi.org/10.18637/jss.v059.i10.

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science*. Beijing, China: https://r4ds.had.co.nz/index.html; O'Reilley.