

Veldig praktisk dataanalyse med R

For studenter i samfunnsvitenskap

Torbjørn Skardhamar

2026-02-25

Table of contents

Forord	3
Hvorfor R?	3
Pakker som brukes	3
Bokens oppbygning	5
I Del I: Kom igang	6
1 Installere R og Rstudio	7
1.1 Installasjon	7
1.1.1 Ikke alle feilmeldinger er like nøyne	8
1.1.2 Spesielt om Windows-maskiner: installer Rtools	8
1.1.3 Spesielt om Mac-maskiner	8
1.1.4 Spesielt om Linux-maskiner	8
1.1.5 Spesielt om Chromebook	8
1.2 Hvis du har problemer med installasjon som vi ikke får løst	9
1.2.1 Rstudio workbench i UiO-skyen	9
1.3 Oppsett og forberedelser	9
1.3.1 Utseende i Rstudio	9
1.4 Rstudio projects	11
1.5 Åpne RStudio og opprett et .Rproject	12
2 En veldig kjapp intro til R	16
2.0.1 Et par hurtigtaster	16
2.0.2 Problemer med æøå?	16
2.1 Objektorientert	17
2.2 Funksjoner	17
2.3 R-pakker	18
2.4 R-dialekter	19
2.4.1 Tidyverse	19
2.4.2 Pipe: %>%	19
2.4.3 Logiske operatorer	19
2.4.4 Lagre data	20
2.5 Hjelpparser og dokumentasjon	20
2.5.1 Bruke pakker uten å laste dem	20

3 Bruk KI som kodehjelp	21
3.1 Hva er en KI-kodehjelp?	21
3.2 Bruk KI til å forstå kode	22
3.3 Bruk KI til feilsøking (debugging)	23
3.4 Bruk KI til å generere kode	24
3.5 Gi KI-verktøy grunnleggende instruksjoner	25
3.6 Vibe coding og agentiske verktøy	25
3.7 Datasikkerhet og personvern	26
3.7.1 Hva er persondata?	26
3.7.2 Trening og datalagring	26
3.7.3 Agentiske verktøy – ekstra utfordringer	27
3.7.4 Hva bør du gjøre?	27
3.7.5 Lokale modeller	27
3.7.6 Regulering og veien videre	27
3.8 Tips for effektiv bruk	28
3.9 En note om læring	28
II Del II: Deskriptiv analyse	29
4 Din første analyse	30
4.1 Laste pakker	30
4.2 Lese inn data	31
4.3 Ta en titt på dataene	31
4.4 Din første figur	32
4.4.1 Steg 1: Bare en linje	32
4.4.2 Steg 2: Legg til punkter	33
4.4.3 Steg 3: Legg til tittel	33
4.4.4 Steg 4: Legg til akselabels og kilde	34
4.4.5 Steg 5: Endre tema og lagre i et objekt	35
4.5 Filtrere data	36
4.6 Større datasett med flere variable	37
4.7 Figur med flere grupper	39
4.7.1 Steg 1: Enkel linje – noe er galt	39
4.7.2 Steg 2: Gruppering og farge	40
4.7.3 Steg 3: Titler og tema	40
4.8 Avgrense ytterligere	41
4.9 Tidstrend for utvalgte aldersgrupper	42
4.10 Oppsummering	43
4.11 Oppgaver	44
5 Grafikk med ggplot	45
5.1 Lagvis grafikk	46

5.2	Kategoriske variabel	47
5.2.1	Stolpediagram	47
5.2.2	Kakediagram	49
5.3	Kontinuerlige variable	51
5.3.1	Histogram	51
5.3.2	Density plot	53
5.3.3	Flerne variable samtidig	58
5.4	Oppgaver	61
6	Deskriptive tabeller	62
6.1	Quick-and-dirty oppsummeringer	62
6.1.1	Enkeltfunksjoner	63
6.2	Professionelle tabeller med gtsummary	65
6.2.1	Eksport av tabeller	70
6.3	Manuelle tabeller	71
6.3.1	For datasettet totalt	71
6.3.2	Grupperte statistikker	71
6.4	Oppgaver	71
7	Kart og romlige data	72
7.1	Geografiske data og sf-pakken	72
7.2	Lese inn kartdata	72
7.3	Grunnleggende kart med ggplot	73
7.4	Koropletkart: fargeleggje regioner etter en variabel	73
7.5	Koble data til kartgeometrier	74
7.6	Legge til punkter pa kartet	75
7.7	Koordinatreferansesystemer (CRS)	75
7.8	Interaktive kart med leaflet	76
7.9	Kilder til norske geodata	76
7.10	Praktisk eksempel: kriminalitetskart over Oslo	77
7.10.1	Lese inn shapefil	77
7.10.2	Grunnkart over Oslo	78
7.10.3	Lese inn og utforske kriminalitetsdata	80
7.11	Punktkart	81
7.12	Rasterkart (heatmap)	82
7.13	Oppdelt etter kriminalitetstype	84
7.14	Oppsummering	85
8	Nettverksanalyse	86
8.1	Hva er et nettverk?	86
8.2	Typer nettverk	86
8.3	Lage nettverksobjekter med igraph	87
8.4	Kantlister og nabomatriser	88

8.5	Nettverksmaal	90
8.5.1	Grad (degree)	90
8.5.2	Mellomleddsentralitet (betweenness)	90
8.5.3	Naerhetssentralitet (closeness)	91
8.6	Visualisering med ggraph	91
8.7	Samfunnsdeteksjon	93
8.8	Praktisk eksempel: samarbeidsnettverk	94
8.9	Nar er nettverksanalyse nyttig?	96
III	Del III: Statistisk modellering	97
9	Regresjon: Sammenheng mellom variable	98
9.1	Scatterplot	98
9.2	Regresjonslinja	100
9.3	Dummy-variable	102
9.3.1	Dummy-variable med mer enn en kategori	104
9.4	Flere variable	106
9.4.1	Interaksjonsledd	108
9.5	Prediksjon	109
9.5.1	Regne ut forventet verdi	109
9.5.2	Predikere for kontinuerlig variabel	110
9.5.3	Predikere kategorisk variabel	110
9.5.4	Predikere for multippel regresjon	111
9.6	Pene tabeller og eksport til fil	112
9.6.1	Alt 1: Bruke <code>modelsummary()</code>	112
9.6.2	Alt 2: Bruke <code>{stargazer}</code>	115
9.6.3	Alt 3: Bruke <code>{gtsummary}</code>	118
10	Utvidelser av regresjonsmodeller	120
10.1	Polynomiske ledd: ikke-lineære sammenhenger	120
10.2	Log-transformasjoner	122
10.2.1	Logaritme av utfallsvariablen	122
10.2.2	Logaritme av forklaringsvariablen	123
10.2.3	Oppsummering av log-transformasjoner	123
10.3	Robuste standardfeil	124
10.3.1	Med <code>modelsummary</code>	124
10.3.2	Med <code>lmtest</code> og <code>sandwich</code>	124
10.4	Klustrede standardfeil	125
10.5	Vektet regresjon	126
10.6	Faste effekter (fixed effects)	126
10.7	Oppsummering	127

11 Modelldiagnostikk	128
11.1 Hva er det vi sjekker?	129
11.2 Residualplot: Residualer mot predikerte verdier	129
11.3 QQ-plot for normalitet	130
11.4 De fire standard diagnostikkplottene	131
11.5 Heteroskedastisitet	133
11.6 Innflytelsesrike observasjoner og Cooks avstand	134
11.7 Multikollinearitet og VIF	135
11.8 Rask diagnostikk med <code>performance</code> -pakken	135
11.9 Hva gjør man hvis antakelsene brytes?	137
12 Lineær sannsynlighetsmodell	138
12.1 Dummy som utfallsvariabel	138
13 Kontrollere for bakenforliggende variable	140
13.1 Simpsons paradoks: Når helheten lyver	140
13.2 Hva betyr det å “kontrollere for” en variabel?	140
13.3 Praktisk eksempel: Kjønnsgapet i lønn	141
13.4 Bivariat vs. multippel regresjon	142
13.5 Sammenligne modeller med <code>modelsummary()</code>	142
13.6 Når skal vi kontrollere – og når skal vi la være?	143
13.7 “Bad controls”: Ikke kontroller for konsekvenser	144
13.8 Steg-for-steg: Bygge modeller i R	144
13.9 Oppsummering	145
14 Logistisk regresjon	146
14.1 Lage en binær utfallsvariabel	146
14.2 Hvorfor ikke bare bruke lineær regresjon?	147
14.3 Logit-transformasjonen og odds	147
14.4 Estimere logistisk regresjon i R	147
14.5 Tolke koeffisientene: odds-ratio	148
14.6 Flere forklaringsvariable	149
14.7 Predikere sannsynligheter	150
14.8 Presentere resultater med <code>modelsummary()</code>	151
14.9 Modelltilpasning	152
14.10 Oppsummering	153
15 Marginaleffekter	154
15.1 Eksempelmodeller	154
15.2 Gjennomsnittlige marginaleffekter (AME)	155
15.3 Marginaleffekter ved bestemte verdier	156
15.4 Predikerte verdier	156
15.5 Sammenligninger mellom grupper	157

15.6 Plotting av marginaleffekter	158
15.7 Sammenhengen med predict()	161
15.8 Oppsummering	162
16 Prediksjon og maskinlæring	163
16.1 Forklaring vs. prediksjon	163
16.2 Overtilpasning	163
16.3 Trening og test: dele opp dataene	164
16.4 En enkel prediksjonsmodell	164
16.5 Mål på prediksjonsevne	165
16.6 Kryssvalidering	166
16.7 Regularisering: LASSO	166
16.8 Klassifisering	167
16.9 Når er prediksjon nyttig i samfunnsvitenskap?	167
16.10 Videre lesning	167
IV Del IV: Statistisk tolkning	168
17 Statistisk tolkning	169
17.1 Stokastiske variabler og sannsynlighetsfordelinger	170
17.1.1 Estimater som stokastiske variabler	170
17.2 Estimater og feilmarginer	171
17.2.1 Estimat	171
17.2.2 Standardfeil	171
17.2.3 Konfidensintervall	173
17.2.4 T-testen	174
17.3 Kan man velge fritt konfidensgrad?	176
17.4 Statistiske tester generelt	176
18 Statistikk i praksis	178
18.1 Deskriptiv statistikk	178
18.2 Regresjon	181
18.2.1 Multippel regresjon	182
18.2.2 Interaksjonsledd	182
V Del V: Tolkning og vitenskapelig praksis	185
19 Forskningsdesign og tolkning	186
19.1 Observasjonsdata vs. eksperimentelle data	186
19.2 Korrelasjon er ikke kausalitet	186
19.3 Seleksjonsskjøvhett	187
19.4 Teoriens rolle	187

19.5 DAGer: Rettet asyklistisk graf	188
19.6 Trusler mot validitet	188
19.6.1 Intern validitet	189
19.6.2 Ekstern validitet	189
19.7 Kvasi-eksperimentelle design	189
19.8 Praktiske råd	189
19.9 Oppsummering	190
20 Reproducerbarhet	191
20.1 Replikasjonskrisen	191
20.2 Bruk script – ikke pek-og-klikk	191
20.3 R-prosjekter og filstier	192
20.3.1 Aldri bruk <code>setwd()</code>	192
20.3.2 <code>here</code> -pakken for filstier	192
20.4 Kommenter koden din	193
20.5 Quarto og R Markdown for rapporter	193
20.6 Versjonskontroll med Git	193
20.7 Pakkeversjoner med <code>renv</code>	194
20.8 Dele data og kode	194
20.9 Dokumenter R-miljøet ditt	195
20.10 Sjekkliste for reproducerebare prosjekter	195
VI Del VI: Importere og utforske data	196
21 Innlesning av data	197
21.1 Generelt om ulike dataformat	197
21.1.1 <code>rds</code>	197
21.1.2 Laste workspace med <code>load()</code>	198
21.1.3 csv-filer	199
21.1.4 Excel	199
21.1.5 Proprietære format: Stata, SPSS og SAS	200
21.1.6 Dataformater for store data	202
22 Import fra Stata og SPSS	203
22.1 Hva er labelled data?	203
22.2 Lese inn data	203
22.3 Inspisere labler	204
22.4 Problemet med labelled data i R	205
22.5 Konvertere til factor	205
22.6 Brukerdefinerte missing-verdier	206
22.7 Lese inn SPSS-filer	207
22.8 Tegnsett og encoding	207

22.9 Anbefalt arbeidsflyt	208
22.10 Oppsummering	208
23 Import av Excel-filer	209
23.1 Lese inn Excel-filer med <code>readxl</code>	209
23.2 Velge ark med <code>sheet</code>	210
23.3 Håndtere overskriftsrader og hoppe over rader	210
23.4 Spesifisere celleområde med <code>range</code>	211
23.5 Vanlige problemer med Excel-filer	211
23.5.1 Datoer	211
23.5.2 Sammenslætte celler	211
23.5.3 Flere tabeller i samme ark	212
23.5.4 Tekst i tallkolonner	212
23.6 Skrive Excel-filer	212
23.7 Tips for rotete Excel-filer	213
24 Data fra SSBs statistikkbank	214
25 Få oversikt over datasettet	222
25.1 Sjekk om innlesning ble riktig	222
25.1.1 Bruke <code>View()</code>	223
25.1.2 Bruke <code>head()</code>	223
25.1.3 Subset med klammeparenteser	224
25.1.4 Bruke ‘ <code>glimpse()</code> ’	225
25.1.5 Undersøke enkeltvariable med <code>codebook()</code> fra pakken {memisc}	225
25.2 Søke i datasettet etter variable	226
VII Del VI: Datahåndtering	228
26 Datahåndtering med tidyverse	229
26.1 Pipe-operatoren: <code>%>%</code>	230
26.2 <code>mutate()</code> – lage nye variable	230
26.2.1 Enkel beregning	230
26.2.2 Lage aldersgrupper med <code>case_when()</code>	231
26.2.3 Lage en binær variabel med <code>ifelse()</code>	232
26.3 <code>select()</code> – velge variable	232
26.3.1 Velge variable	233
26.3.2 Fjerne variable	233
26.3.3 Hjelpefunksjoner i <code>select()</code>	233
26.4 <code>filter()</code> – filtrere rader	234
26.4.1 Filtrere på én betingelse	234
26.4.2 Filtrere på flere betingelser	234

26.4.3 Fjerne manglende verdier	235
26.5 <code>summarise()</code> – oppsummere data	236
26.6 <code>group_by()</code> – grupperte operasjoner	236
26.7 <code>arrange()</code> – sortere data	238
26.8 Kombinere funksjoner med pipe	239
26.9 <code>across()</code> – samme operasjon på flere variable	240
26.10 Oppsummering	241
26.11 Oppgaver	242
27 Omkoding av variable	243
27.1 Factor-variable	243
27.1.1 Lage factor fra tall	244
27.2 Omkoding med <code>ifelse()</code>	244
27.3 Omkoding med <code>case_when()</code>	245
27.4 Lage grupper med <code>cut()</code>	246
27.5 Endre factor-nivåer med <code>forcats</code>	246
27.5.1 Slå sammen kategorier med <code>fct_collapse()</code>	246
27.5.2 Gi nye navn med <code>fct_recode()</code>	247
27.5.3 Endre rekkefølge med <code>fct_relevel()</code>	248
27.5.4 Sortere etter en annen variabel med <code>fct_reorder()</code>	248
27.5.5 Fjerne ubrukte nivåer med <code>fct_drop()</code>	249
27.6 Jobbe med tekst	250
27.7 Praktisk eksempel: komplett omkoding	250
27.8 Oppsummering	251
28 Haandtering av missing-verdier	253
28.1 Hva er NA?	253
28.2 Sjekke for missing-verdier	253
28.3 Oppsummere missing i datasettet	254
28.4 Funksjoner og NA: argumentet <code>na.rm</code>	255
28.5 Filtrere ut missing	255
28.6 Omkode verdier til NA	256
28.7 Erstatte NA med verdier	257
28.8 Missing i regresjonsanalyser	257
28.9 Multippel imputering	258
28.10 Vanlige fallgruver og tips	259
29 Koble sammen og omforme data	260
29.1 Koble datasett med join-funksjoner	260
29.1.1 <code>left_join</code> : Behold alt fra venstre datasett	261
29.1.2 <code>right_join</code> : Behold alt fra høyre datasett	262
29.1.3 <code>inner_join</code> : Bare rader som finnes i begge	262
29.1.4 <code>full_join</code> : Behold alt fra begge datasett	263

29.1.5	Spesifisere koblingsnøkkelen med by-argumentet	263
29.1.6	Hva kan gå galt?	264
29.2	Omforme data: bredt og langt format	265
29.2.1	Fra bredt til langt med pivot_longer()	266
29.2.2	Fra langt til bredt med pivot_wider()	267
29.2.3	Når trenger du hva?	268
30	Håndtering av store datasett	269
30.1	Når blir data “store”?	269
30.2	Sjekke størrelsen på datasett	269
30.3	Effektive filformater: Parquet	270
30.4	data.table: Et raskt alternativ	271
30.5	Lazy evaluation med Arrow	272
30.6	Databasetilkoblinger	272
30.7	Praktiske tips for store datasett	273
30.8	Når R ikke er nok	274
Referanser		275
Appendices		276
A	Rstudio addins	276
A.1	Styler - skriv pent	276
A.2	Esquisse - grafikk	276
A.3	Questionr - omkode factor	277
B	Import av data fra Sikt - håndtering av formater med metadata	278
B.0.1	Hvorfor så vanskelig?	278
B.1	Håndtering av user-NAs	279
B.2	innlesning av data	280
B.3	Hvordan fungerer koden ovenfor?? En intro til mer avansert databehandling	282
B.3.1	across()	282
B.3.2	case_when()	283
B.3.3	fra factor til numerisk	283
C	Lage dictionary-fil	284
C.1	Lese inn html-dokumentasjonen	284
C.1.1	Legge det hele i en funksjon	285
D	Empirisk eksempel	288
D.1	Utvælg av variable	288
E	Leser inn data. Bruker labelled for enklere oversettelse av stata-kode	290

F Kobler på register, beholder bare treff på begge.	291
G Utvalgsstørrelse og personer	292
G.1 Etablering av utvalg for analyse	292
G.2 Deskriptiv statistikk	292
G.3 En kommentar om reproducertbarhet	292
G.3.1 Replikering på uavhengige data	292

Forord

Denne boken er ment som en praktisk innføring i bruk av R til dataanalyse for studenter i samfunnsvitenskap. Fokuset er på *hvordan* man gjør dataanalyse i R – fra innlesning av data, via datahåndtering og deskriptiv analyse, til statistisk modellering og visualisering.

Boken er ment å *komplettere* lærebøker i statistikk og metode, ikke erstatte dem. Statistiske begreper forklares når det er nødvendig for å forstå koden, men den grundige teoretiske behandlingen av statistikk overlates til andre lærebøker. Unntaket er kapittelet om design, tolkning og teori (Del IX), som tar opp noen viktige perspektiver på dataanalyse som sjeldent presenteres samlet i andre lærebøker.

Teksten er ment som en ganske grunnleggende innføring for praktikere. Fokuset er å vise løsninger som fungerer uten for mye mikk-makk. Det er alltid flere måter å gjøre ting på, men hovedteksten dekker anbefalte løsninger basert på hensyn som effektivitet, konsistens og funksjonalitet. Boken kan også brukes som oppslagsverk.

Gjennom boken brukes varierte datasett fra ulike R-pakker, med vekt på data som er relevante for samfunnsvitenskap.

Hvorfor R?

R er et programmeringsspråk laget spesielt for statistikk og dataanalyse. Det er gratis, åpen kildekode, og har et enormt økosystem av pakker for alt fra enkel deskriptiv statistikk til avansert maskinlæring og kartproduksjon.

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 “pekar-og-klikk”. R er et programmeringsspråk, og det er viktig å lære å skrive kode for både databehandling og analyse. Dette kan virke krevende i starten, men gir betydelige fordeler: koden dokumenterer nøyaktig hva som er gjort, analyser kan reproduceres, og komplekse operasjoner kan automatiseres.

Pakker som brukes

Boken bruker en rekke R-pakker. Tabellen nedenfor genereres automatisk ved å skanne alle kapittelfiler, og oppdateres hvis det gjøres endringer i prosjektet.

Table 1: R-pakker brukt i prosjektet (n = 44)

Pakke	Versjon
arrow	23.0.1.1
car	3.1.3
chattr	0.3.1
data.table	1.17.8
DBI	1.2.3
dbplyr	2.5.2
dplyr	1.2.0
equatiomatic	0.4.4
fixest	0.13.2
ggdag	0.2.13
ggforce	0.5.0
ggraph	2.2.2
ggridges	0.5.7
gt	1.3.0
gtsummary	2.5.0
haven	2.5.5
here	1.0.2
igraph	2.2.2
knitr	1.51
labelled	2.16.0
leaflet	2.2.3
lmtest	0.9.40
lobstr	1.1.2
marginaleffects	0.32.0
memisc	0.99.31.8.3
modelsummary	2.6.0
naniar	1.1.0
openxlsx	4.2.8.1
performance	0.16.0
PxWebApiData	1.9.0
readxl	1.4.5
remotes	2.5.0
renv	1.1.7
RSQLite	2.4.6
RStata	1.1.2
rvest	1.0.5
sandwich	3.1.1
scales	1.4.0
sf	1.0.22

Pakke	Versjon
stargazer	5.2.3
tidyverse	1.3.1
tidyverse	2.0.0
wooldridge	1.4.4
writexl	1.5.4

Bokens oppbygning

Boken er organisert i ni deler:

- **Del I** hjelper deg med å komme igang med R og RStudio.
- **Del II** dekker innlesning av data fra ulike formater og kilder, inkludert Stata, SPSS, Excel og SSBs statistikkbank.
- **Del III** handler om datahåndtering: tidyverse-verb, omkoding, manglende verdier, kobling av datasett og håndtering av store data.
- **Del IV** dekker deskriptiv analyse med grafikk og tabeller.
- **Del V** er den mest omfattende delen og dekker statistisk modellering: lineær regresjon med utvidelser (interaksjoner, splines, DiD, RD), diagnostikk, logistisk regresjon, marginaleffekter, og en introduksjon til prediksjon og maskinlæring.
- **Del VI** handler om statistisk tolkning: standardfeil, konfidensintervaller og p-verdier.
- **Del VII** viser hvordan man plasserer data på kart.
- **Del VIII** gir en enkel introduksjon til nettverksanalyse.
- **Del IX** tar opp viktige perspektiver på design, tolkning, teori og reproducirbarhet.

Part I

Del I: Kom igang

1 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!).

1.1 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, sørge 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 integrert i hverandre og du åpner 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).

Du skal også installere noen R-pakker. Det er omtalt i et annet kapittel med oversikt over hva det er og hvilke du trenger.

1.1.1 Ikke alle feilmeldinger er like nøyne

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.

1.1.2 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/>

1.1.3 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/>

1.1.4 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.

1.1.5 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 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.

1.2 Hvis du har problemer med installasjon som vi ikke får løst

1.2.1 Rstudio workbench i UiO-skyen

Hvis du opplever uløselige problemer med å kjøre R og Rstudio på din datamaskin, så finnes det en krise-løsning. Rstudio har også en versjon som kjører i skyen via nettleser. UiO har en slik versjon installert på sine servere som vi kan bruke. Du logger da inn på [Rstudio Workbench](#) med ditt Feide brukernavn og passord. (Det er sendt inn beskjed om at alle på SOS4020 skal ha tilgang, og håper det er i orden nå eller veldig snart).

Rstudio workbench fungerer på samme måte som Rstudio ellers, men du kan ikke installere pakker selv. Det viktigste er tilgjengelig allerede, så det burde gå fint. I fanen “Files” kan du lage en mappestruktur og laste opp/ned filer etter behov.

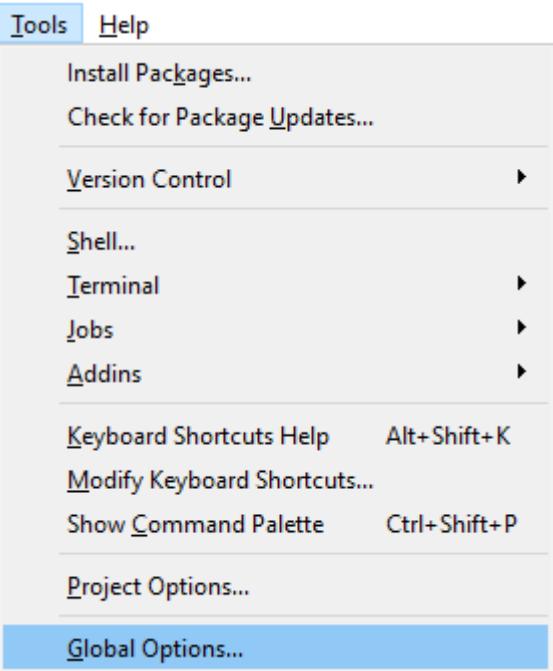
OBS! Workbench-løsningen har et helt trivlet sikkerhetsnivå for data. Den er kun godkjent for å bruke [grønne data](#). Det betyr at du *ikke* kan jobbe med data som ikke er åpne med denne løsningen.

1.3 Oppsett og forberedelser

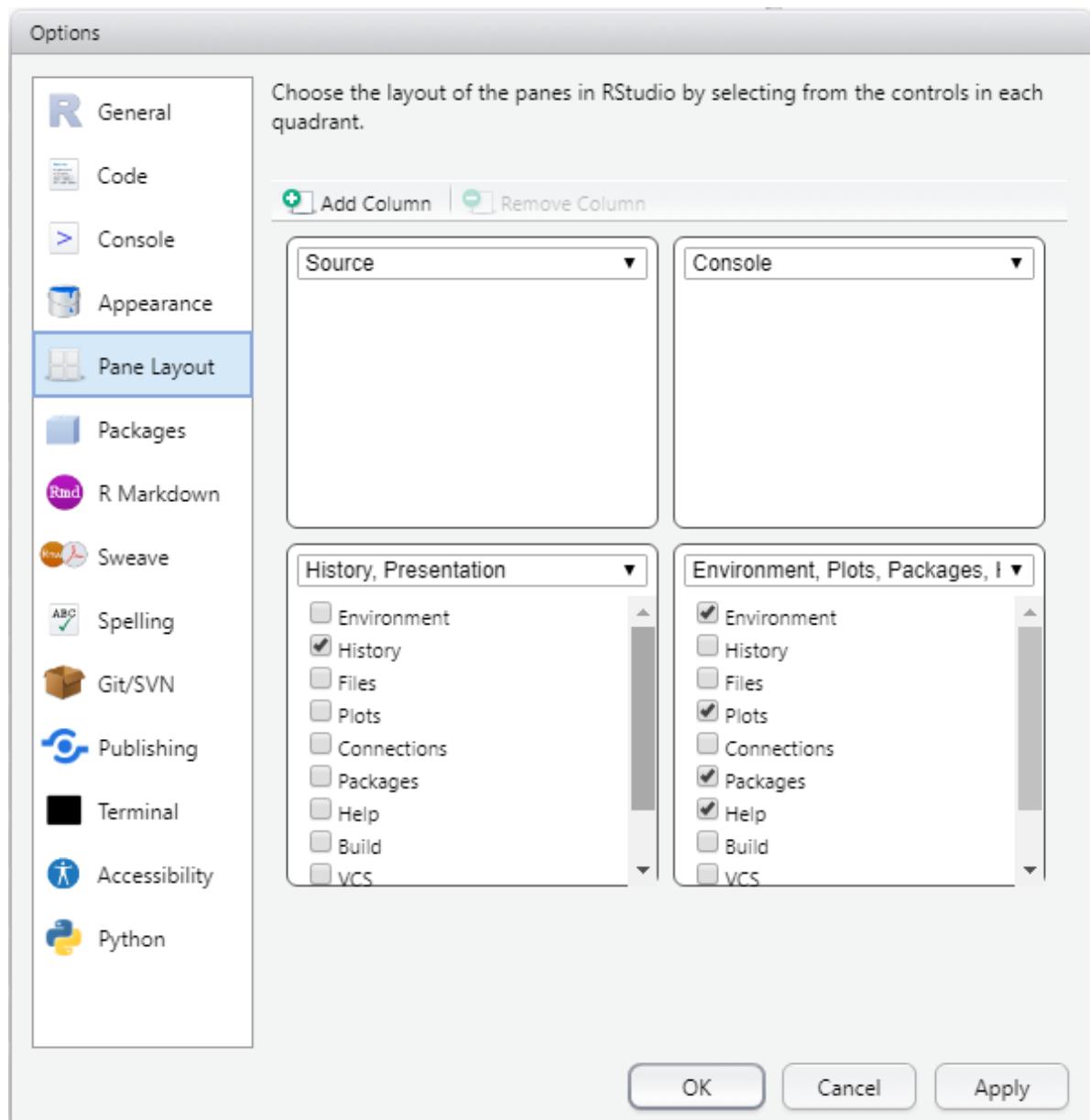
Dette oppsettet gjelder både hvis du har en lokal installasjon og for skyløsninger. Utseendet spiller ingen rolle, og R kan også fungere uten å opprette «projects» som beskrevet her. Men det er lettere å bruke og du har bedre orden hvis du gjør dette.

1.3.1 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.

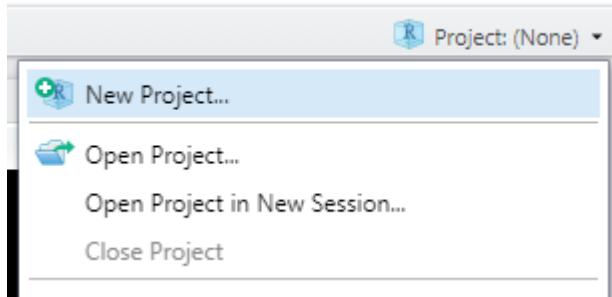
1.4 Rstudio projects

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.

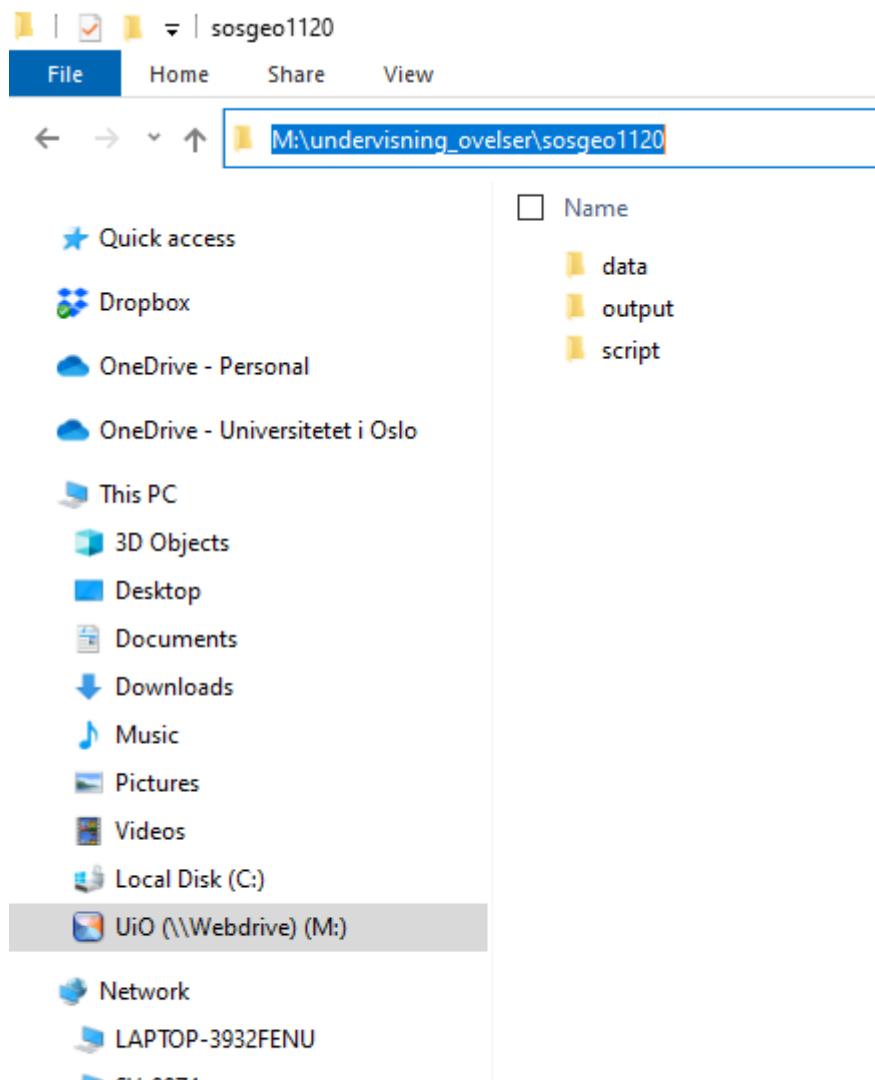
1.5 Åpne RStudio og opprett et .Rproject



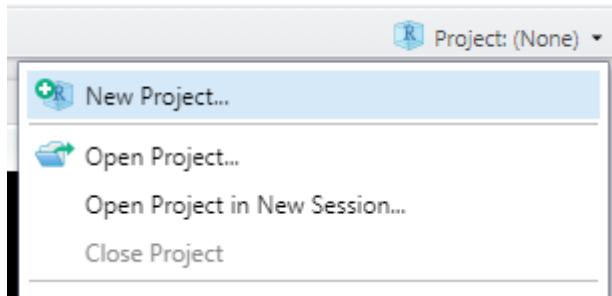
Bruk funksjonen `getwd()` og se at du har riktig filbane til arbeidsområdet. Hvis du ikke er sikker på hva det betyr, må du spørre noen eller finne det ut på annen måte!

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.

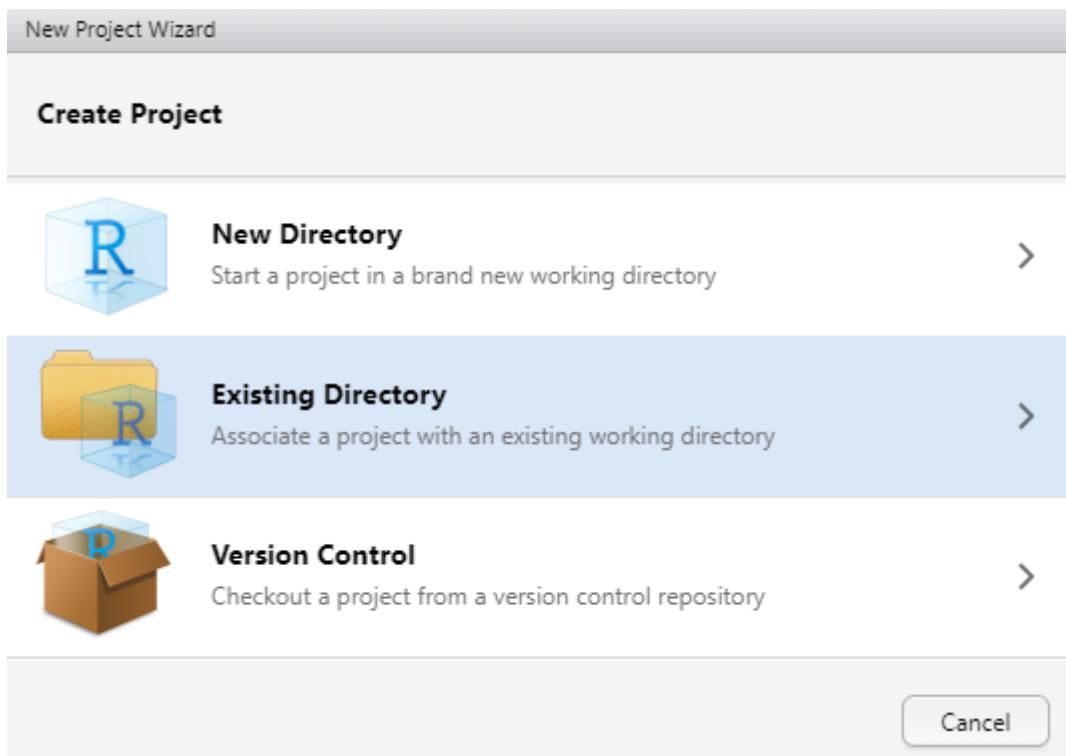
For dette kurset skal du ha en mappestruktur med en hovedmappe for dette kurset og tilhørende undermapper. Det spiller ingen rolle hvor på datamaskinen du legger disse mappe, men du må vite hvor det er. Lag første en mappe med et hensiktsmessig navn for kurset, og innunder denne mappen lager du tre andre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

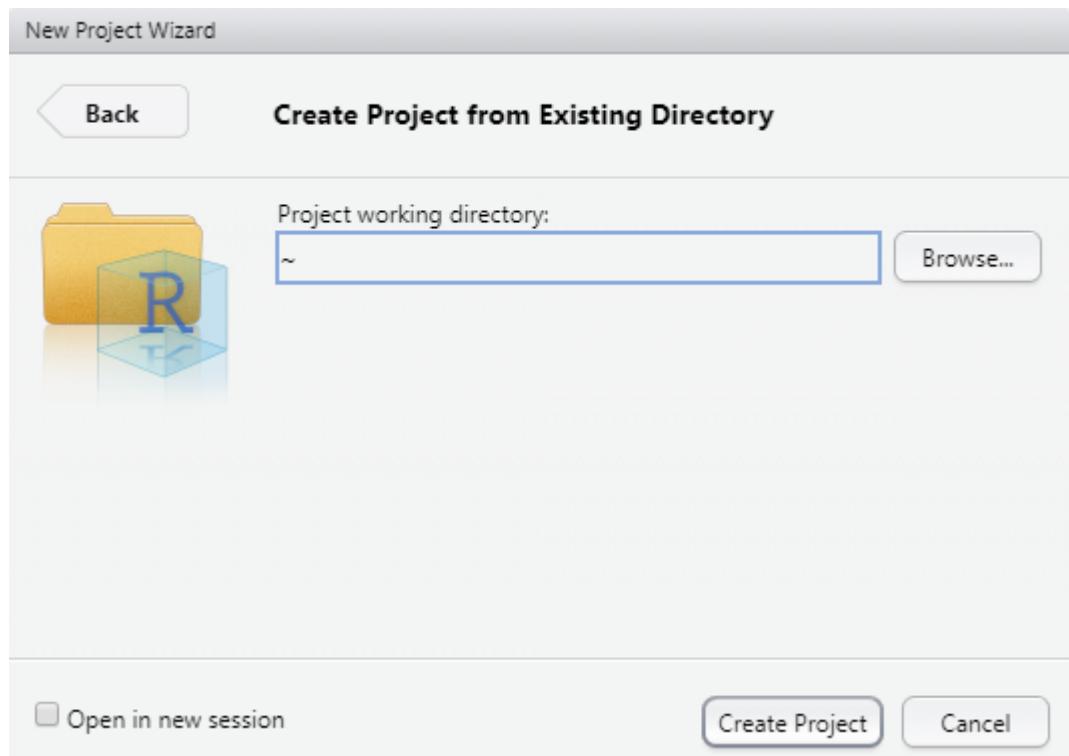


Deretter klikker du du «Existing Directory»

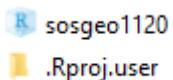


Klikk «Browse» og bla deg så frem til mappen du har laget for dette kurset.

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 datahåndtering og 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 **Ctrl + 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

R er bygd opp rundt å bruke “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 sjeldent forholde deg til hvordan et slikt objekt er bygd opp.

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. Funksjonen har et navn etterfulgt av en parentes slik som f.eks. `difunksjon(...)`. 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 stå inni parentesen. 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 parentesen – 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.

I tillegg kan det være en rekke andre argumenter. 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. 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”. Men R er i praksis basert på å bruke såkalte “pakker”. Dette er funksjoner som utvider R sin funksjonalitet.

R-pakker er et helt økosystem av funksjonalitet, og det finnes mange tusen R-pakker tilgjengelig på en server som heter CRAN. For nye brukere av R vil dette fremstå som ganske kaotisk, men du får beskjed om hvilke pakker du trenger fortløpende.

For å installere en pakke må du vite hva pakken heter og datamaskinen din må være koblet til internett:

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

Vanlige grunner til feilmeldinger ved installering:

- 1) Du har stavet navnet på pakken feil. Pass på små og store bokstaver.
- 2) Pakken krever at du har noen andre pakker installert fra før. Installer disse først og prøv igjen.
- 3) Noen andre pakker trenger oppdatering. Oppdater alle pakker og prøv på nytt.
- 4) Din R-installasjon må oppdateres.

Når en pakke er installert må du “laste” den for at funksjonene skal være tilgjengelig i din R-sesjon. Hvis du restarter R, så må du laste pakkene på nytt.

```
library(pakkenavn)
```

For denne boken trenger du følgende pakker:

```
install.packages( c("tidyverse", "haven", "gtsummary", "gt", "modelsummary",
                     "labelled", "marginaleffects",
                     "causaldata", "gapminder", "palmerpenguins",
                     "sf", "tmap", "spData",
                     "tidygraph", "ggraph")
)
```

Installering av pakker gjør du bare én gang. Du må derimot laste pakker hver gang du starter R på nytt.

2.4 R-dialekter

De funksjonene som følger med grunnleggende installasjon av R kalles “base R”. Man kan gjøre svært mye med bare base R. Men noen R-pakker inneholder ikke bare enkeltfunksjoner, men nesten et helt programmeringsspråk i seg selv. Det er lurt å holde seg innenfor samme “dialekt” da man ellers kan bli veldig forvirret. I denne boken bruker vi dialekten **tidyverse** konsekvent.

Merk at det finnes andre dialekter som er spesialiserte for spesifikke formål. Et eksempel er `{data.table}` som er lynrask for store datasett (se eget kapittel). Dette gjør at det kan være vanskelig å søke på nettet etter løsninger fordi du kan få svar i en annen dialekt enn den du kan.

2.4.1 Tidyverse

Når man laster pakken `{tidyverse}` laster man egentlig flere pakker som også kan lastes individuelt. “Tidy” betyr “ryddig” og hensikten er et språk som er så ryddig og logisk som mulig. Full oversikt over pakkene som inngår i [Tidyverse finner du på deres hjemmeside](#).

Grunnleggende datahåndtering med tidyverse dekkes i et eget kapittel (Del III). Grafikk med ggplot dekkes i Del IV. Her introduseres bare det aller mest grunnleggende.

2.4.2 Pipe: %>%

Et viktig konsept i tidyverse er “pipen” `%>%` (hurtigtast: Ctrl + Shift + M). Den betyr “gjør deretter”. Du kan binde sammen flere operasjoner i en arbeidsflyt:

```
dinedata %>%
  mutate(ny_variabel = gammel * 2) %>%
  filter(gruppe == "a")
```

Dette leses som: “start med dinedata, lag deretter en ny variabel, filtrer deretter på gruppe a.”

2.4.3 Logiske operatorer

I mange sammenhenger setter man *hvis*-krav. Her er grunnleggende logiske operatorer:

Uttrykk	Kode
er lik	<code>==</code>

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

2.4.4 Lagre data

Du kan som et utgangspunkt tenke at du *ikke* skal lagre bearbeidede data på disk. Scriptet ditt starter med å lese inn de originale dataene og gjør alt du trenger fra start til slutt. På den måten har du reproducable script.

Hvis du likevel trenger å lagre data til disk, bruk .rds-formatet:

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

For permanent lagring og deling av data, bruk csv-format:

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

2.5 Hjelppakker og dokumentasjon

Dokumentasjonen i R åpnes med `? foran funksjonsnavnet`, f.eks. `?read.csv`. Hjelppakker har en fast struktur: **Usage** viser syntaks med forvalgsverdier, **Arguments** forklarer hvert argument, **Examples** viser kodeeksempler.

Mange pakker har også “vignetter” som gir mer utfyllende forklaringer. Disse finnes på pakkens CRAN-side eller på egne nettsider.

2.5.1 Bruke pakker uten å laste dem

En funksjon fra en spesifikk pakke kan angis med `pakkenavn::funksjon()`. Dette er nyttig hvis det er navnekonflikt mellom pakker:

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

3 Bruk KI som kodehjelp

KI-verktøy (kunstig intelligens) har blitt svært gode til å skrive kode i mange språk, deriblant R. Du kan bruke dem til å forstå kode, finne feil, og generere nye løsninger. Det finnes mange slike verktøy, blant annet ChatGPT, Claude, GitHub Copilot og Gemini. Prinsippene i dette kapittelet gjelder uansett hvilket verktøy du bruker, og vi omtaler dem samlet som *KI-verktøy*.

Men la det være helt klart: KI-verktøy gjør *ikke* at du kan hoppe over å lære å kode selv. Du må forstå koden for å vite om den gjør det du faktisk ønsker, og du må kunne vurdere om resultatene er riktige. KI er et hjelpemiddel – ikke en erstatning for forståelse.

Mange universiteter og høyskoler har egne retningslinjer for bruk av KI-verktøy. Ved UiO finnes det for eksempel en egen tjeneste kalt [GPT UiO](#) som er tilpasset personvernkravene ved universitetet. Sjekk hvilke verktøy som er godkjent ved din institusjon.

Disse verktøyene utvikler seg raskt. Konkrete funksjoner og grensesnitt kan endres, men prinsippene for hvordan du bruker dem effektivt er de samme.

3.1 Hva er en KI-kodehjelp?

Med KI-kodehjelp mener vi verktøy basert på store språkmodeller som du kan kommunisere med via tekst. Du skriver en instruksjon – et *prompt* – og verktøyet gir deg et svar i form av kode, forklaringer, eller begge deler.

Det finnes to hovedtyper:

- **Chat-baserte verktøy** (ChatGPT, Claude, Gemini): Du åpner en nettside, limer inn kode eller skriver et spørsmål, og får svar i en samtale. Dette er den enkleste måten å komme i gang på.
- **Integrasjonsverktøy** (GitHub Copilot, Positron-assistenten): Disse er bygd inn i kodeeditoren din og kan foreslå kode mens du skriver. Det krever litt mer oppsett, men kan være effektivt når du har kommet i gang.

For nybegynnere anbefaler vi å starte med et chat-basert verktøy. Det gjør at du ser hele samtalen og kan følge med på hva som skjer. De fleste har gratis versjoner som er mer enn tilstrekkelige for studentbruk.

3.2 Bruk KI til å forstå kode

Det beste stedet å begynne er å bruke KI til å *forstå* kode du har problemer med. Det er mye mer lærerikt enn å la verktøyet skrive koden for deg.

For å få et godt svar bør du spesifisere hva det er du ikke forstår. Si gjerne at du er nybegynner, og at du foretrekker forklaringer som bruker tidyverse. Her er et eksempel på et prompt du kan prøve:

Kan du forklare følgende R-kode linje for linje?
Jeg er nybegynner og bruker tidyverse.

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

Eksempel på svar fra KI-verktøy

Denne koden bruker tidyverse og gjør følgende steg for steg:

1. `dinedata %>%` — tar datasettet `dinedata` og sender det videre med pipe-operatoren (`%>%`).
2. `group_by(gruppe)` — grupperer dataene etter variabelen `gruppe`. Alle påfølgende operasjoner gjøres innenfor hver gruppe.
3. `mutate(...)` — legger til tre nye kolonner *uten* å fjerne eksisterende rader:
 - `antall = n()` — antall observasjoner i hver gruppe.
 - `gjennomsnitt = mean(varA)` — gjennomsnittet av `varA` innenfor gruppen.
 - `avvik = varA - gjennomsnitt` — hvor mye hver observasjon avviker fra gruppegjennomsnittet.
4. `arrange(gruppe, desc(avvik))` — sorterer radene etter gruppe, og innenfor hver gruppe etter avvik i synkende rekkefølge (størst avvik først).
5. `filter(row_number() == 1)` — beholder bare den første raden i hver gruppe, altså observasjonen med størst avvik fra gjennomsnittet.

Resultatet er et datasett med én rad per gruppe: den observasjonen som avviker mest fra gruppens gjennomsnitt.

Du vil typisk få en forklaring av hva hver linje gjør. Men husk: du må selv vurdere om forklaringen gir mening i din kontekst. Prøv å kjøre koden selv og se om resultatet stemmer med det verktøyet forklarte.

3.3 Bruk KI til feilsøking (debugging)

KI-verktøy er også nyttige for å finne feil i kode som ikke fungerer som forventet. For å få et godt svar er det viktig å gi verktøyet god *kontekst*. Beskriv:

- Hva du forsøkte å gjøre
- Hva du forventet skulle skje
- Hva som faktisk skjedde (inkluder gjerne feilmeldingen)

Her er et eksempel:

```
Jeg ønsker å få aggregerte statistikker for et datasett  
med en linje per gruppe, men resultatet gir en linje  
per observasjon i stedet. Kan du finne feilen?
```

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

Eksempel på svar fra KI-verktøy

Problemet er at du bruker `mutate()`, som legger til nye kolonner men beholder alle radene. For å få én rad per gruppe med aggregerte verdier, bør du bruke `summarise()` i stedet.

Her er den korrigerte koden:

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

`summarise()` kollapser hver gruppe til én rad og beregner de ønskede statistikkene. `mutate()` derimot legger til kolonner mens alle opprinnelige rader beholdes, og det er derfor du fikk én linje per observasjon.

Et nyttig tips er å lime inn feilmeldinger direkte. KI-verktøy er gode til å tolke R-feilmeldinger og kan ofte forklare hva som gikk galt.

Merk at verktøyet vil foreslå en løsning, men det er ikke gitt at det forstår problemet ditt korrekt. Løsningen er heller ikke nødvendigvis riktig. Du må alltid teste den foreslalte løsningen selv og sjekke at den faktisk gir de resultatene du forventer.

3.4 Bruk KI til å generere kode

Du kan også be KI-verktøy om å skrive helt ny kode for deg. For at dette skal fungere godt, må du ha en klar idé om hva du ønsker. Vær spesifikk om:

- Hvilke pakker du vil bruke (f.eks. tidyverse, ggplot2)
- Hva slags data du har (variabler, typer)
- Hva resultatet skal se ut som

Her er et eksempel på et prompt:

```
Lag en ggplot-figur som viser sammenhengen mellom  
variablene alder og lønn med et scatterplot.  
Bruk ulike farger for variabelen kjønn.  
Bruk tidyverse og ggplot2. Datasettet heter abu89.
```

Eksempel på svar fra KI-verktøy

Her er koden for et scatterplot med ulike farger for kjønn:

```
library(tidyverse)  
  
ggplot(abu89, aes(x = alder, y = lønn, color = kjønn)) +  
  geom_point(alpha = 0.5) +  
  labs(x = "Alder", y = "Lønn", color = "Kjønn") +  
  theme_minimal()
```

Koden bruker `aes()` til å knytte variabler til visuelle egenskaper: `x` og `y` for aksene, og `color` for å gi ulike farger per kjønn. `alpha = 0.5` gjør punktene halvt gjennomsiktige, slik at overlappende punkter blir synlige.

Første forsøk er sjeldent perfekt. Bruk oppfølgingsspørsmål for å justere: “Kan du legge til en regresjonslinje?”, “Kan du endre til `theme_minimal()`?”, osv. Det er slik man jobber effektivt med KI-verktøy – som en samtale der du gradvis presiserer hva du vil ha.

Det viktigste prinsippet: **les og forstå hver linje i den genererte koden før du bruker den.** Hvis det er noe du ikke forstår, be verktøyet forklare akkurat den delen. Kopier aldri kode blindt inn i analysene dine.

3.5 Gi KI-verktøy grunnleggende instruksjoner

De fleste KI-verktøy lar deg sette opp faste instruksjoner som gjelder for alle samtaler. I ChatGPT heter dette “Custom Instructions”, i Claude kan du opprette “Projects” med egne instruksjoner, og andre verktøy har tilsvarende løsninger. Konseptet er det samme: du forteller verktøyet hvem du er og hva du foretrekker, slik at du slipper å gjenta deg.

Her er et eksempel på slike instruksjoner som er tilpasset dette kurset:

Jeg er nybegynner i dataanalyse i R innen samfunnsvitenskap. Jeg har bare grunnleggende forståelse av R. Jeg foretrekker kode basert på tidyverse fremfor base-R og data.table. Kode som er lettlest er viktigere enn effektivitet. For deskriptive tabeller, bruk pakken gtsummary. For regresjonstabeller, bruk pakken modelsummary. Forklar koden trinn for trinn.

De viktigste tingene å spesifisere er:

- Ditt nivå (nybegynner)
- Foretrukne pakker (tidyverse, gtsummary, modelsummary)
- Kodestil (lettlest fremfor effektivt)

3.6 Vibe coding og agentiske verktøy

Det har i det siste dukket opp to begreper som er verdt å kjenne til: *vibe coding* og *agentiske verktøy*.

Vibe coding er et begrep som ble populært i 2025 og beskriver en arbeidsform der du beskriver hva du vil i naturlig språk og lar KI generere all koden – uten at du leser eller forstår koden selv. Du styrer etter “vibben”: ser resultatet riktig ut? Fungerer det? Hvis ja, gå videre. Noen erfarte programmerere bruker dette for å lage prototyper raskt, men da kan de også vurdere og feilsøke resultatet når noe går galt.

For studenter er vibe coding **ikke en god strategi**. Poenget med et metodekurs er å lære å forstå dataanalyse, og det innebærer å forstå koden. Hvis du lar KI skrive alt uten å lese det, lærer du ingenting – og du kan ikke oppdage feil i analysen. Det er som å bruke maskinoversettelse til å skrive en stil i et språk du holder på å lære: resultatet kan se greit ut, men du har ikke lært noe, og du kan ikke fange opp feil.

Agentiske verktøy er en nyere kategori KI-verktøy som går et steg videre. Verktøy som Claude Code, OpenAI Codex og Cursor kan ikke bare skrive kode, men også kjøre den, lese resultatene, og justere koden automatisk – alt uten at du gjør noe. Dette er kraftige verktøy for erfarte brukere, men for nybegynnere innebærer det enda mindre kontakt med selve koden.

Vær oppmerksom på at disse verktøyene finnes. Når du har bygget opp grunnleggende ferdigheter – for eksempel i forbindelse med en masteroppgave eller i arbeidslivet – kan de bli svært nyttige. Men først må du ha kompetansen til å vurdere det de produserer. Det finnes ingen snarvei forbi det å lære seg å kode.

3.7 Datasikkerhet og personvern

Når du bruker et KI-verktøy, sendes alt du skriver i promptet til en ekstern server for prosessering. Det samme gjelder konteksten som verktøyet bruker: hvis du limer inn kode, data eller feilmeldinger, havner dette hos KI-leverandøren. For agentiske verktøy som Claude Code eller Cursor gjelder dette i enda større grad – de kan lese hele filer og prosjektmapper automatisk for å bygge kontekst, uten at du eksplisitt velger hva som sendes.

Dette har viktige implikasjoner for personvern og datasikkerhet.

3.7.1 Hva er persondata?

Etter personvernforordningen (GDPR) er persondata all informasjon som kan knyttes til en identifiserbar person. Det inkluderer opplagte ting som navn, fødselsnummer og e-postadresser, men også kombinasjoner av variabler som sammen kan identifisere enkeltpersoner – for eksempel alder, bosted og yrke. Helseopplysninger, etnisitet og politisk tilhørighet er i tillegg regnet som *særlege kategorier* med ekstra strengt vern.

I samfunnsvitenskapelig forskning er dette svært relevant. Datasettene du jobber med kan inneholde slike opplysninger, og å lime inn rader fra et datasett i en KI-chat kan i praksis utgjøre en overføring av persondata til en tredjepart.

3.7.2 Trening og datalagring

De fleste KI-tjenester bruker samtaledata til å trenne og forbedre modellene sine – med mindre du aktivt reserverer deg. ChatGPT og Claude (i gratisversjonene) gjør dette som standard, men tilbyr mulighet for å skru det av i innstillingene. Data kan også lagres i en periode, typisk fra noen dager til flere uker, avhengig av tjeneste og abonnementstype.

API-tilgang og bedriftsavtaler har vanligvis strengere regler: data brukes normalt ikke til trening, og lagringstiden er kortere. Men detaljene varierer mellom leverandører og endres jevnlig. Les alltid gjeldende vilkår, og anta som hovedregel at det du skriver kan bli lagret.

3.7.3 Agentiske verktøy – ekstra utfordringer

Mens du i et chatvindu selv velger hva du limer inn, kan agentiske verktøy lese og sende fil- innhold automatisk. Et verktøy som Claude Code eller Cursor kan indeksere hele prosjektmap- pen din for å gi bedre svar. Dersom prosjektet inneholder datasett med personopplysninger, konfigurasjonsfiler med passord, eller andre sensitive data, kan disse bli sendt til leverandørens servere uten at du er klar over det.

Hvis du jobber med sensitive data, er det avgjørende å forstå hvilke filer verktøyet har tilgang til, og hva som faktisk sendes. Noen verktøy tilbyr en «Privacy Mode» som begrenser hva som deles, men dette må aktiveres eksplisitt.

3.7.4 Hva bør du gjøre?

- **Aldri lim inn reelle persondata** i et KI-verktøy. Bruk anonymiserte eller syntetiske eksempeldata når du trenger hjelp med kode.
- **Sjekk institusjonens retningslinjer.** De fleste universiteter har egne regler for bruk av KI-verktøy – følg disse.
- **Vurder abonnementstype.** For prosjekter med sensitive data kan API-tilgang eller bedriftsavtaler med strengere personvernveilkår være nødvendig.
- **Vær ekstra varsom med agentiske verktøy** som har tilgang til hele prosjektmappen. Sørg for at sensitive datafiler ikke ligger i samme mappe som kodeprosjektet, eller bruk verktøyets innstillinger for å ekskludere dem.
- **Vurder om du trenger KI-hjelp i det hele tatt** for den konkrete oppgaven. Noen ganger er det tryggere å løse problemet selv.

3.7.5 Lokale modeller

For situasjoner der data ikke kan forlate maskinen, finnes det modeller som kan kjøres helt lokalt. Verktøy som [Ollama](#) gjør det mulig å kjøre åpne modeller som Llama og DeepSeek på egen maskin, og [OpenClaw](#) lar deg koble lokale modeller til agentiske arbeidsflyter. Da sendes ingen data til eksterne servere. Per i dag krever dette en viss teknisk kompetanse og en maskin med tilstrekkelig kapasitet, men terskelen er i ferd med å synke raskt.

3.7.6 Regulering og veien videre

Norge innfører en ny lov om kunstig intelligens, basert på EUs AI Act, med planlagt ikraft- tredelse sommeren 2026. Loven stiller krav til transparens, dokumentasjon og risikovurdering for KI-systemer. Datatilsynet følger utviklingen tett og har blant annet en egen sandkasse for KI-prosjekter som ønsker veiledning om personvern.

Forvent at det kommer tydeligere regler for bruk av KI-verktøy i forskning og utdanning i årene fremover. Grunnprinsippet vil uansett bestå: du er selv ansvarlig for at persondata behandles i tråd med regelverket, uansett hvilket verktøy du bruker.

3.8 Tips for effektiv bruk

1. **Start med å forstå**, deretter feilsøk, og til slutt generer ny kode. Denne rekkefølgen gir best læring.
2. **Les alltid koden** du får fra KI-verktøy før du bruker den. Hvis du ikke forstår en linje, be verktøyet forklare den.
3. **Test på egne data.** Kode som ser riktig ut kan likevel gi feil resultater med dine data.
4. **Vær spesifikk i promptene.** Nevn pakker, variabelnavn, og hva slags output du forventer.
5. **Bruk oppfølgingsspørsmål.** Første svar er sjeldent perfekt – juster med nye instruksjoner.
6. **Sett opp faste instruksjoner** med dine preferanser, slik at du får konsistente svar.
7. **Husk at KI kan ta feil.** Verktøyene høres alltid selvsikre ut, også når svaret er galt. Dobbelsjekk alltid.

3.9 En note om læring

Dette er et godt sted for en liten bemerkning om læring og studier, selv om denne boken ellers er svært praktisk orientert. Det er åpenbart slik at KI kan lage ferdige produkter for deg, uten at du selv behøver å skjønne så mye av hva som skjer. Hvorfor skal du derfor lære disse tingene selv? Noen svar har blitt antydet ovenfor knyttet til de-bugging etc.

En viktigere grunn er at utdanning handler om å trenere opp analytiske evner og ferdigheter. For å kunne analysere data må du skjønne en del om data og databehandling, analysemuligheter og -teknikker. Å jobbe med kode kan være krevende og tidvis svært frustrerende når man ikke skjønner hvorfor noe ikke fungerer. Men det er her læringen ligger: du lærer bare når hjernen anstrenger seg. Du lærer mer når du finner ut av ting selv enn når du får løsningen servert. Du må selv vurdere hvor mye du skal jobbe selv før du bruker f.eks. KI til å hjelpe deg.

For de som er glad i fysisk trening (sport etc), så er en parallel til at du blir ikke sterkere av å henge rundt på treningssenteret. Du blir sterkere av å løfte tungt (eller hva du nå driver med), så det faktum at det finnes kraner og vinsjer som kan løfte for deg er ikke relevant. For læring gjelder det samme: det er din egen anstrengelse som er hele poenget. Hvis du ikke gjør det, vil du neppe lære noe særlig og du kaster egentlig bort tiden.

Part II

Del II: Deskriptiv analyse

4 Din første analyse

I de forrige kapitlene har du installert R og RStudio, og fått en kort introduksjon til grunnleggende konsepter. Nå skal vi gjøre noe annerledes: vi skal kjøre gjennom en hel analyse fra start til mål – lese inn ekte data og lage ekte figurer.

Målet med dette kapittelet er at du skal oppleve at R faktisk *fungerer*, og at du kan produsere noe meningsfullt med bare noen få linjer kode. Ikke bekymre deg om du ikke forstår alle detaljene. Hvert tema forklares grundig i senere kapitler. Her handler det om å følge med, kjøre koden, og se resultatene.

Dataene vi bruker er kriminalitetsstatistikk fra SSB (Statistisk sentralbyrå), hentet fra [tabell 09415](#). De viser siktede for lovbrudd per 1000 innbyggere, fordelt på alder.

Åpne RStudio, sorg for at prosjektet ditt er aktivt (se kapittelet om installering), og opprett et nytt R-script (File → New File → R Script). Skriv koden inn i scriptet og kjør den linje for linje med **Ctrl + Enter**.

4.1 Laste pakker

Før vi kan bruke spesialiserte funksjoner, må vi laste pakkene som inneholder dem. Hvis du ikke allerede har installert disse pakkene, gjør det først:

```
# Installer pakker (trenger bare gjøres én gang)
install.packages(c("tidyverse", "readxl"))
```

Deretter laster vi pakkene. Dette må gjøres hver gang du starter R på nytt:

```
# Laste pakker (gjøres hver gang du starter R)
library(tidyverse)    # Pakke for datahåndtering og grafikk
library(readxl)       # Pakke for å lese Excel-filer
```

Funksjonen `install.packages()` laster ned pakken til maskinen din, mens `library()` aktiverer den i den gjeldende R-sesjonen. Se kapittelet om introduksjon til R for mer om pakker.

4.2 Lese inn data

Nå leser vi inn en Excel-fil med kriminalitetsdata. Funksjonen `read_excel()` leser filen, og <- lagrer resultatet i et objekt vi kaller `krim`:

```
# Lese inn en Excel-fil med kriminalitetsdata fra SSB
# read_excel() leser filen, <- lagrer resultatet i objektet "krim"
krim <- read_excel("data/SSB_09415_enkel2024.xlsx")
```

Nå ligger dataene i objektet `krim`. Filstien "data/SSB_09415_enkel2024.xlsx" betyr at filen ligger i en undermappe som heter `data` i prosjektmappen din. Innlesning av data dekkes grundig i Del II.

4.3 Ta en titt på dataene

Før vi lager figurer bør vi sjekke at dataene ser fornuftige ut. To nyttige funksjoner for dette er `head()` og `glimpse()`:

```
# Se på de første radene i datasettet
# head() viser de 6 første observasjonene
head(krim)
```

```
# A tibble: 6 x 2
  alder   vold
  <dbl> <dbl>
1     5   0.02
2     6   0.05
3     7   0.03
4     8   0.19
5     9   0.27
6    10   0.63
```

```
# En annen måte å se på dataene
# glimpse() viser variabelnavn, datatype og de første verdiene
glimpse(krim)
```

```
Rows: 45
Columns: 2
$ alder <dbl> 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2~
$ vold   <dbl> 0.02, 0.05, 0.03, 0.19, 0.27, 0.63, 1.18, 2.80, 6.36, 9.41, 5.36~
```

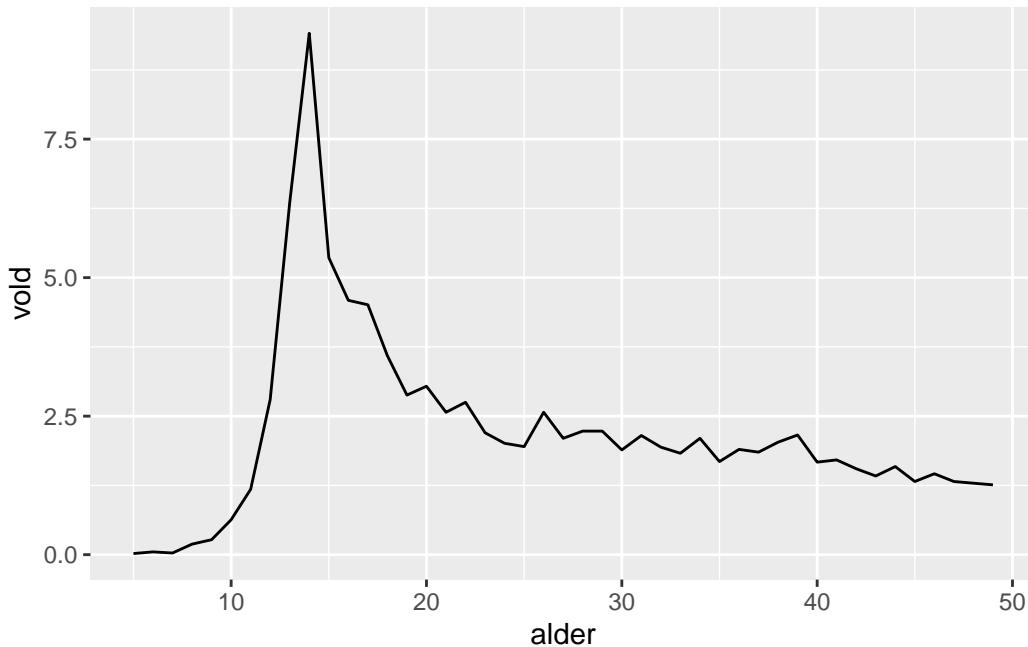
Vi ser at datasettet har to kolonner: `alder` (gjerningspersonens alder) og `vold` (antall siktede for vold per 1000 innbyggere i den aldersgruppen, for 2024). La oss se hva disse tallene viser grafisk.

4.4 Din første figur

Nå lager vi vår første figur. Vi bruker `ggplot`, som er det standard systemet for grafikk i R. Prinsippet er lagvis oppbygging: du starter med et grunnlag og legger til nye elementer ett om gangen. La oss se det steg for steg.

4.4.1 Steg 1: Bare en linje

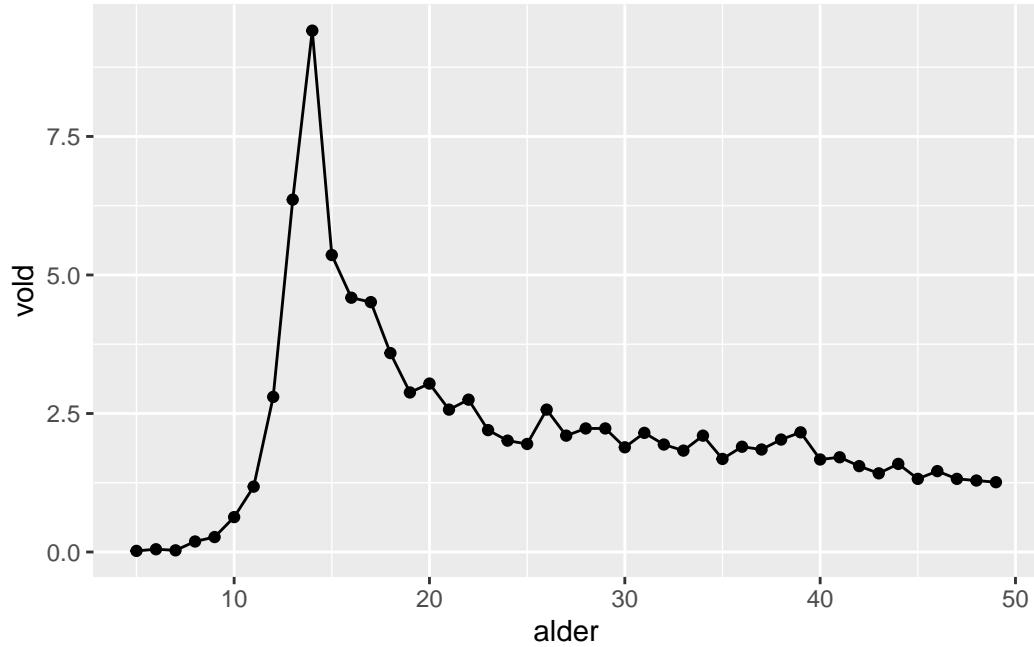
```
# Start et plot: alder på x-aksen, vold på y-aksen
# geom_line() tegner en linje mellom datapunktene
ggplot(krim, aes(x = alder, y = vold)) +
  geom_line()
```



Her skjer tre ting: `ggplot(krim, ...)` sier “bruk datasettet `krim`”. Inni `aes()` (kort for *aesthetics*) angir vi at `alder` skal på x-aksen og `vold` på y-aksen. Til slutt sier `geom_line()` at dataene skal vises som en linje. Resultatet er et minimalt, men fungerende plot.

4.4.2 Steg 2: Legg til punkter

```
# Legg til geom_point() for å vise hvert datapunkt
ggplot(krim, aes(x = alder, y = vold)) +
  geom_line() +
  geom_point()
```

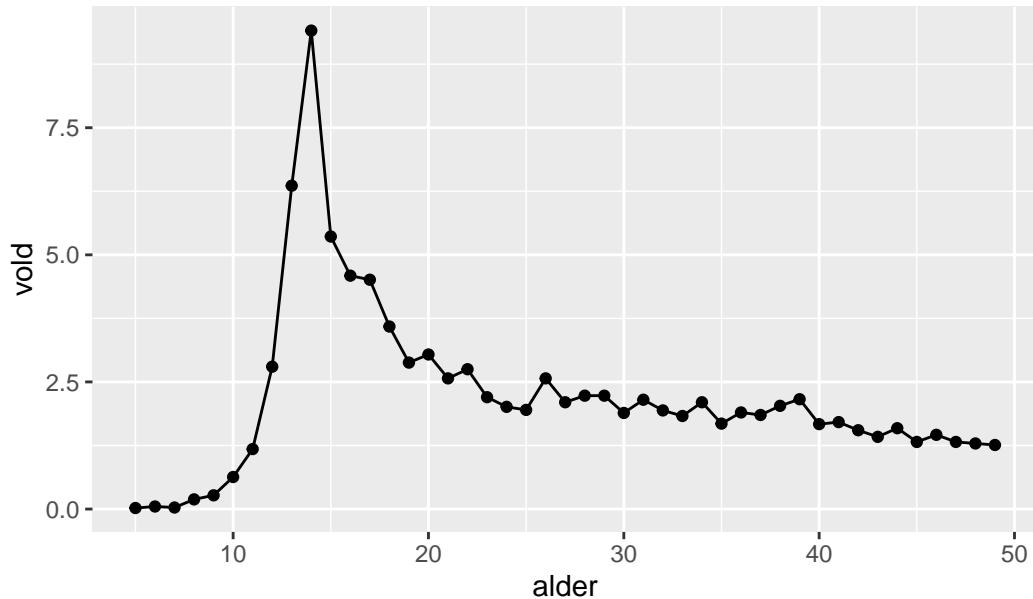


Vi legger til `geom_point()` med `+`. Nå ser vi hvert enkelt datapunkt i tillegg til linjen. `+` i `ggplot` betyr "legg til et nytt lag".

4.4.3 Steg 3: Legg til tittel

```
# Legg til en overskrift med ggtitle()
ggplot(krim, aes(x = alder, y = vold)) +
  geom_line() +
  geom_point() +
  ggtitle("Voldsløvbrudd etter alder")
```

Voldslovbrudd etter alder

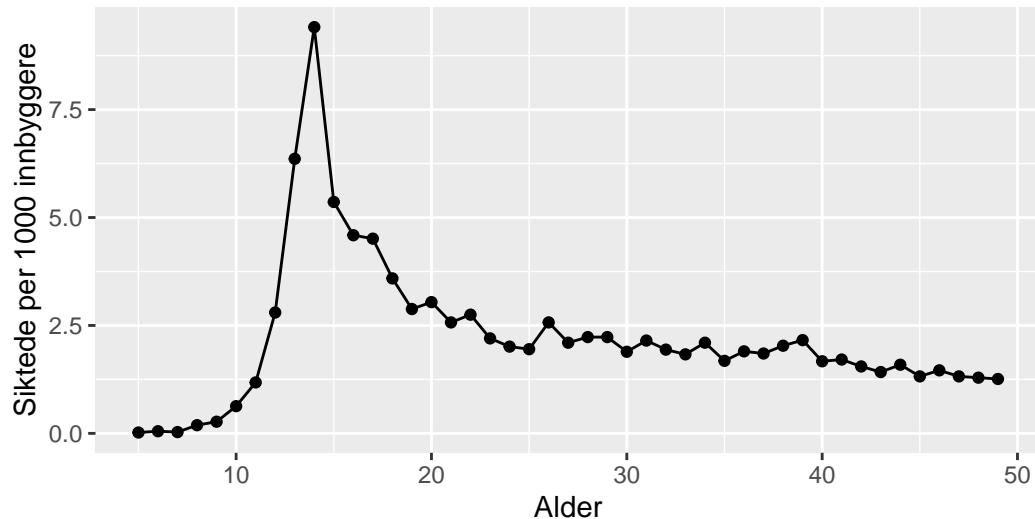


ggtitle() legger til en overskrift på figuren. Enda et lag oppå de forrige.

4.4.4 Steg 4: Legg til akselabels og kilde

```
# labs() styrer akseetiketter, undertittel og kildeangivelse
ggplot(krim, aes(x = alder, y = vold)) +
  geom_line() +
  geom_point() +
  ggtitle("Voldslovbrudd etter alder",
         subtitle = "Siktede per 1000 innbyggere, 2024") +
  labs(x = "Alder",
       y = "Siktede per 1000 innbyggere",
       caption = "Kilde: SSB, tabell 09415")
```

Voldslovbrudd etter alder
Siktede per 1000 innbyggere, 2024



Kilde: SSB, tabell 09415

`labs()` lar deg sette egne etiketter på aksene. `subtitle` legger til en undertekst under tittelen, og `caption` legger til en kildeangivelse nederst.

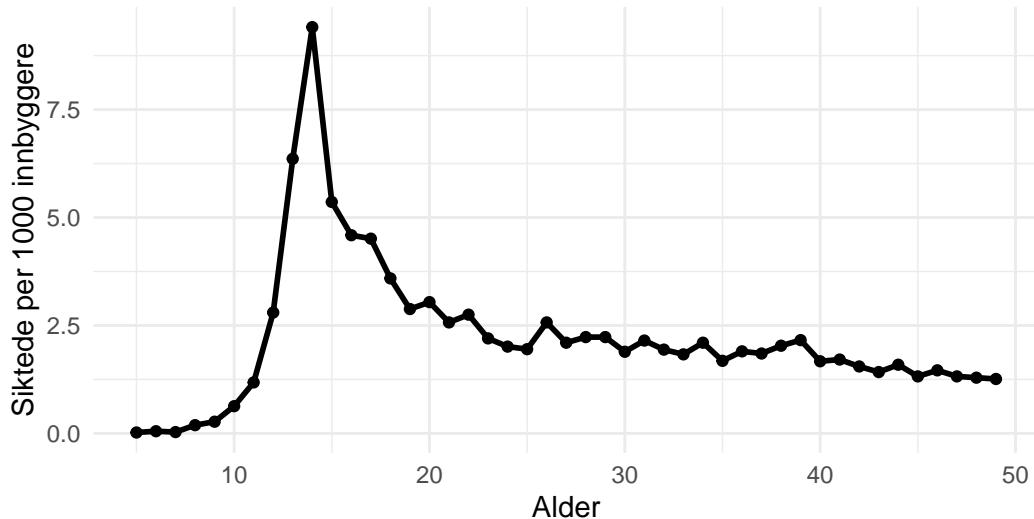
4.4.5 Steg 5: Endre tema og lagre i et objekt

```
# theme_minimal() gir et renere utseende
# Vi lagrer figuren i objektet p1 slik at vi kan bruke den igjen
p1 <- ggplot(krim, aes(x = alder, y = vold)) +
  geom_line(linewidth = 1) +
  geom_point() +
  ggtitle("Voldslovbrudd etter alder",
         subtitle = "Siktede per 1000 innbyggere, 2024") +
  labs(x = "Alder",
       y = "Siktede per 1000 innbyggere",
       caption = "Kilde: SSB, tabell 09415") +
  theme_minimal()

# Vis figuren
p1
```

Voldslovbrudd etter alder

Siktede per 1000 innbyggere, 2024



Kilde: SSB, tabell 09415

`theme_minimal()` endrer det visuelle temaet til et renere design uten grå bakgrunn. `linewidth = 1` gjør linjen litt tykkere. Ved å skrive `p1 <- ggplot(...)` lagrer vi hele figuren i et objekt, slik at vi kan vise den igjen ved å skrive `p1`.

Legg merke til hva figuren forteller: voldskriminalitet stiger bratt i tenårene, topper rundt 18–20 år, og faller deretter jevnt. Du har nå laget en informativ figur med ekte data, fra start til slutt.

4.5 Filtrere data

Hva om vi vil se nærmere på bare ungdomsgruppen? Da kan vi *filtrere* dataene med funksjonen `filter()`, som velger ut rader som oppfyller et krav:

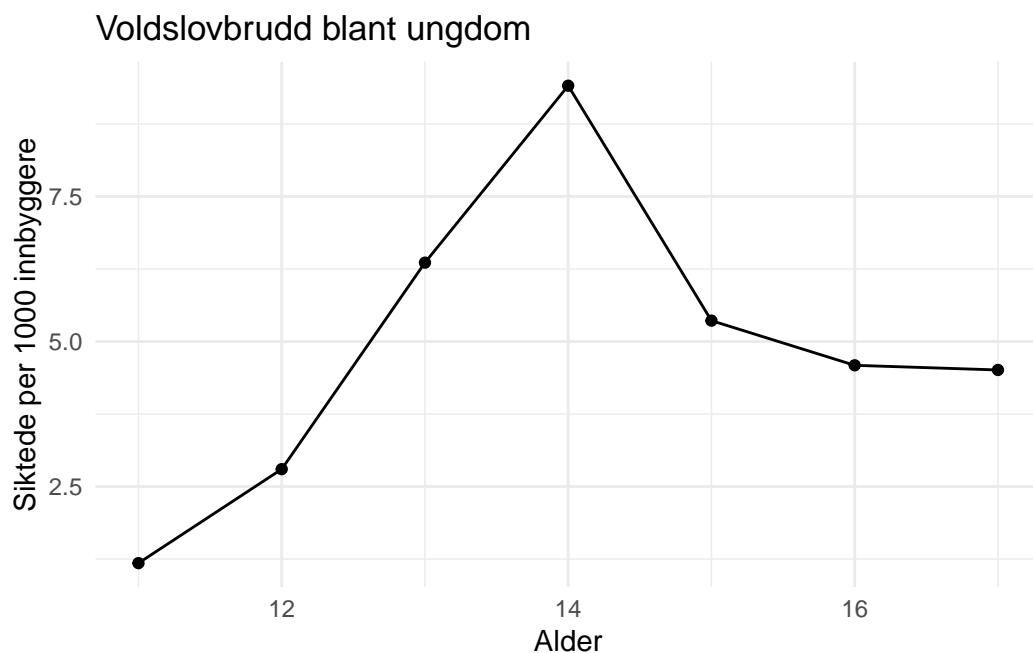
```
# Velg ut bare aldersgruppen 10-18 år
# %>% (pipe) sender dataene videre til neste operasjon
# filter() beholder rader der kravet er oppfylt
# & betyr "og" -- begge krav må være oppfylt
krim_ung <- krim %>%
  filter(alder > 10 & alder < 18)

# Lag en rask figur av ungdomsgruppen
ggplot(krim_ung, aes(x = alder, y = vold)) +
```

```

geom_line() +
geom_point() +
ggtitle("Voldslovbrudd blant ungdom") +
labs(x = "Alder", y = "Siktede per 1000 innbyggere") +
theme_minimal()

```



Her bruker vi `%>%` (pipe-operatoren) som betyr “ta dette, og gjør deretter”. Vi tar `krim`, sender det til `filter()`, og beholder bare rader der alder er mellom 10 og 18. Resultatet lagres i et nytt objekt `krim_ung`.

Figuren viser den bratte økningen i ungdomsårene enda tydeligere. Pipen (`%>%`) og `filter()` er grunnleggende verktøy i tidyverse som dekkes grundig i Del III.

4.6 Større datasett med flere variable

La oss nå jobbe med et større datasett fra samme SSB-tabell, men med mer detalj: fordelt på type lovbrudd, kjønn og år. Denne Excel-filen er litt mer komplisert – de tre første radene inneholder overskriftstekst fra SSB som ikke er data, og kolonnene har ikke gode variabelnavn:

```

# Lese inn et større datasett
# skip = 3 hopper over de 3 første radene (overskriftstekst fra SSB)
# col_names angir variabelnavn vi velger selv

```

```
# Vi bruker variabelnavn uten ør for å unngå tegnkodingsproblemer
krim_alle <- read_excel("data/SSB_09415_alderSiktet.xlsx",
                        skip = 3,
                        col_names = c("lovbruddsgruppe", "kjonn",
                                      "alder", "aar", "pr1000"))
```

```
# Se på de første radene
head(krim_alle)
```

```
# A tibble: 6 x 5
  lovbruddsgruppe     kjonn      alder   aar   pr1000
  <chr>           <chr>       <dbl> <chr>   <dbl>
1 Alle lovbruddsgrupper Begge kjønn      5 2002    0.26
2 Alle lovbruddsgrupper Begge kjønn      5 2003    0.27
3 Alle lovbruddsgrupper Begge kjønn      5 2004    0.13
4 Alle lovbruddsgrupper Begge kjønn      5 2005    0.12
5 Alle lovbruddsgrupper Begge kjønn      5 2006    0.1
6 Alle lovbruddsgrupper Begge kjønn      5 2007    0.07
```

```
# Se hvilke lovbruddsgrupper som finnes
# table() teller opp hvor mange ganger hver verdi forekommer
table(krim_alle$lovbruddsgruppe)
```

Alle lovbruddsgrupper		Annet lovbrudd
3174		3174
Annet vinningslovbrudd		Eiendomsskade
3174		3174
Eiendomstyveri	Ordens- og integritetskrenkelse	
3174		3174
Rusmiddellovbrudd		Seksuallovbrudd
3174		3174
Trafikkovertredelse		Vold og mishandling
3174		3174

Vi ser at datasettet inneholder flere lovbruddsgrupper. \$-tegnet brukes for å hente ut én enkelt variabel fra datasettet. La oss filtrere til bare voldskriminalitet:

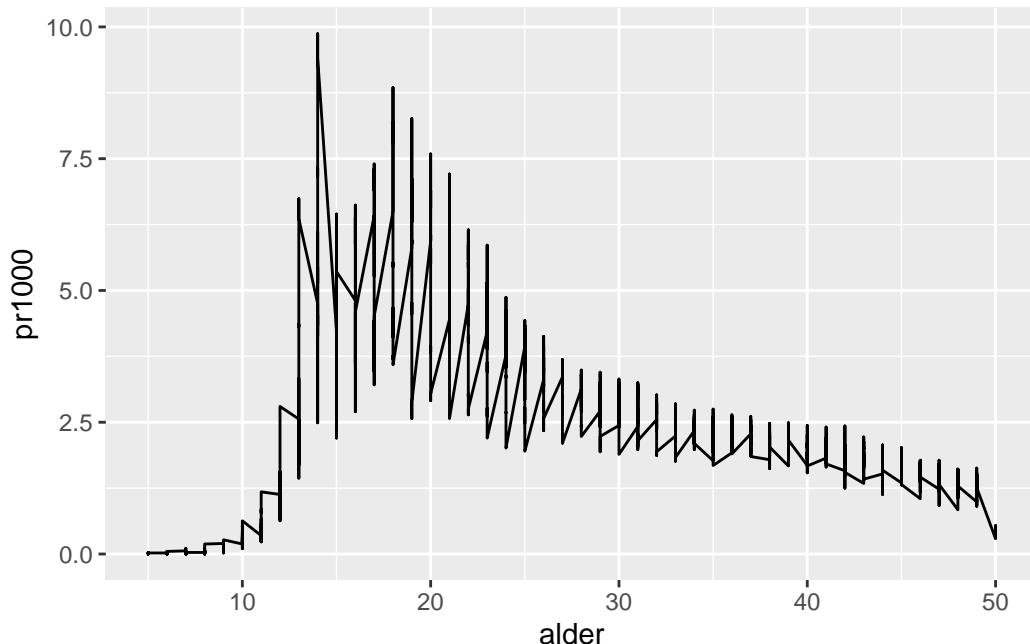
```
# Velg ut bare voldskriminalitet for begge kjønn
krim_vold <- krim_alle %>%
  filter(lovbruddsgruppe == "Vold og mishandling",
    kjonn == "Begge kjønn")
```

4.7 Figur med flere grupper

Nå har vi data for voldskriminalitet over flere år. La oss bygge opp en figur steg for steg igjen.

4.7.1 Steg 1: Enkel linje – noe er galt

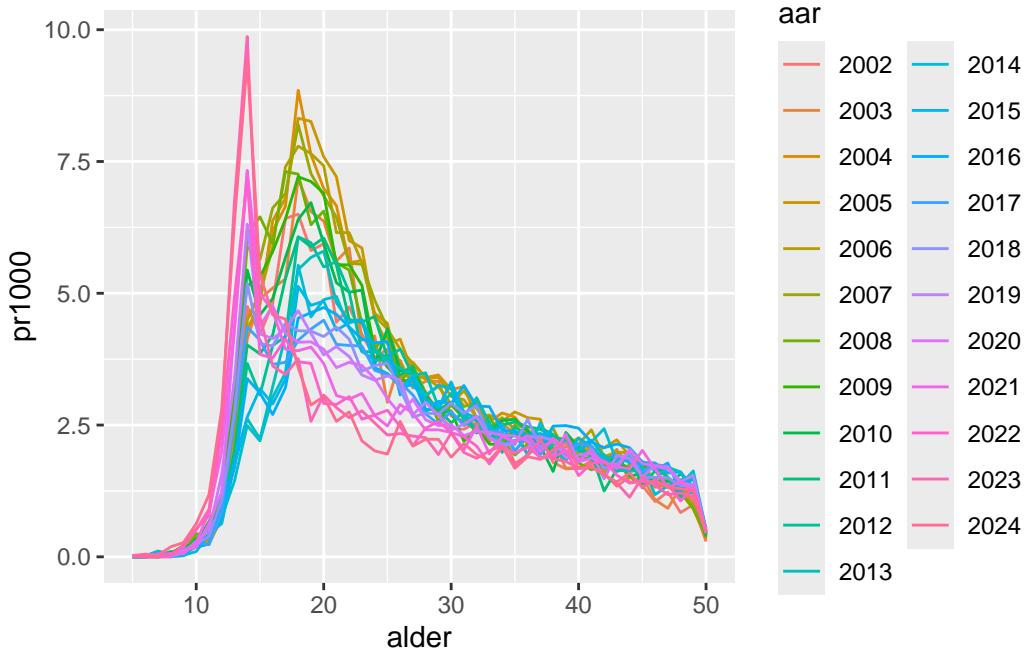
```
# Prøv å tegne en enkel linje -- dette blir rotete!
ggplot(krim_vold, aes(x = alder, y = pr1000)) +
  geom_line()
```



Denne figuren ser kaotisk ut. Det er fordi ggplot prøver å tegne alle datapunktene – fra alle år – som én sammenhengende linje. Vi må fortelle ggplot at hvert år er en egen gruppe.

4.7.2 Steg 2: Gruppering og farge

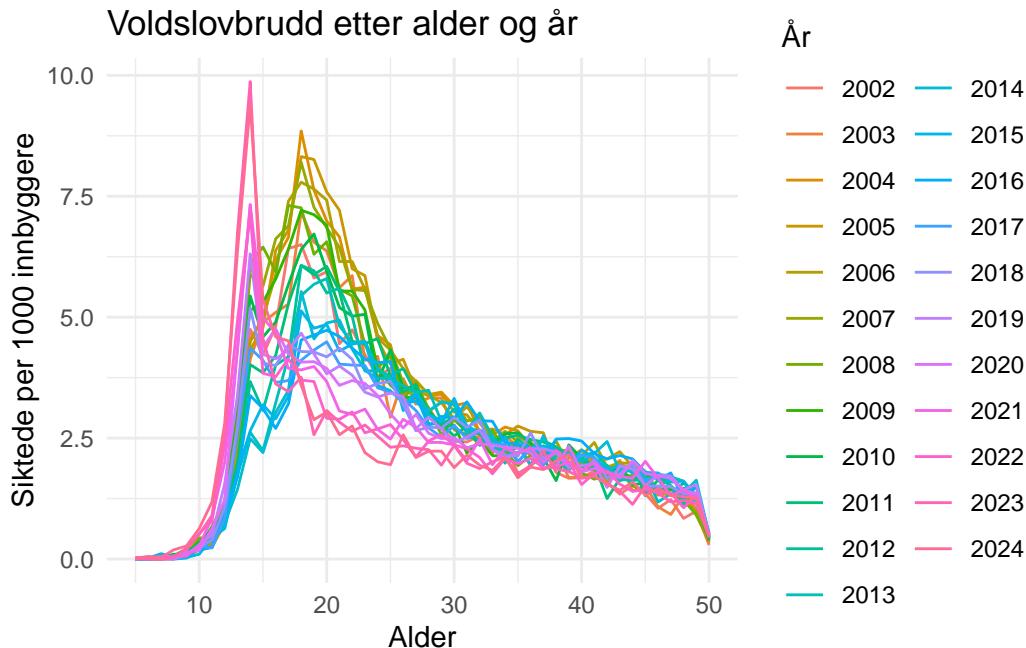
```
# group = aar gjør at hvert år får sin egen linje
# color = aar gir ulike farger for hvert år
ggplot(krim_vold, aes(x = alder, y = pr1000, group = aar, color = aar)) +
  geom_line()
```



Nå er det mye bedre. `group = aar` forteller ggplot at datapunktene skal deles inn etter år, og `color = aar` gir hver gruppe en egen farge. Vi ser alderskurven for voldskriminalitet for hvert år.

4.7.3 Steg 3: Titler og tema

```
# Legg til titler og renere tema
ggplot(krim_vold, aes(x = alder, y = pr1000, group = aar, color = aar)) +
  geom_line() +
  ggtitle("Voldslovbrudd etter alder og år") +
  labs(x = "Alder",
       y = "Siktede per 1000 innbyggere",
       color = "År") +
  theme_minimal()
```



Samme prinsipp som før: vi legger til lag for tittel, aksellabels og tema. `color = "År"` i `labs()` endrer etiketten på fargelegenden.

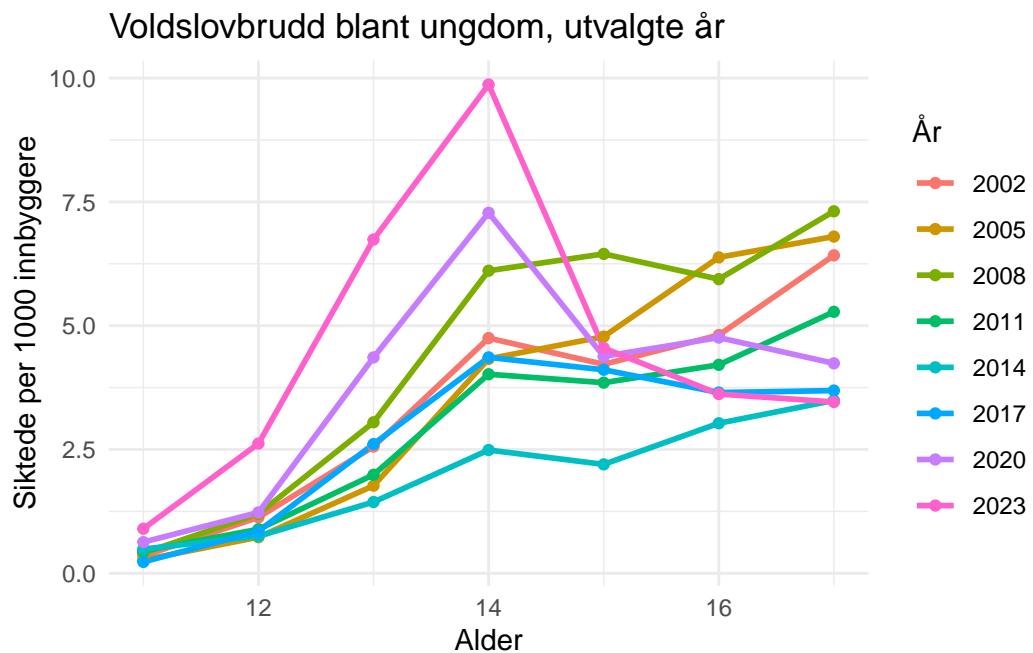
4.8 Avgrense ytterligere

Figuren ovenfor har mange linjer og kan være vanskelig å lese. La oss avgrense til ungdom og velge ut bare noen utvalgte år:

```
# Avgrense til alder 10-18 og hvert tredje år fra 2002 til 2024
# %in% sjekker om en verdi finnes i en liste
# seq() lager en tallrekke: 2002, 2005, 2008, ..., 2024
krim_utvalg <- krim_vold %>%
  filter(alder > 10, alder < 18) %>%
  filter(aar %in% seq(2002, 2024, by = 3))

# Figur med tykkere linjer og punkter
ggplot(krim_utvalg, aes(x = alder, y = pr1000, group = aar, color = aar)) +
  geom_line(linewidth = 1) +
  geom_point() +
  ggtitle("Voldslovbrudd blant ungdom, utvalgte år") +
  labs(x = "Alder",
       y = "Siktede per 1000 innbyggere",
```

```
color = "År") +
theme_minimal()
```



Her kombinerer vi flere filtreringer: først avgrenser vi til ungdom, deretter velger vi ut hvert tredje år. `seq(2002, 2024, by = 3)` lager tallrekken 2002, 2005, 2008, ..., 2024, og `%in%` sjekker om verdien av `aar` finnes i denne listen.

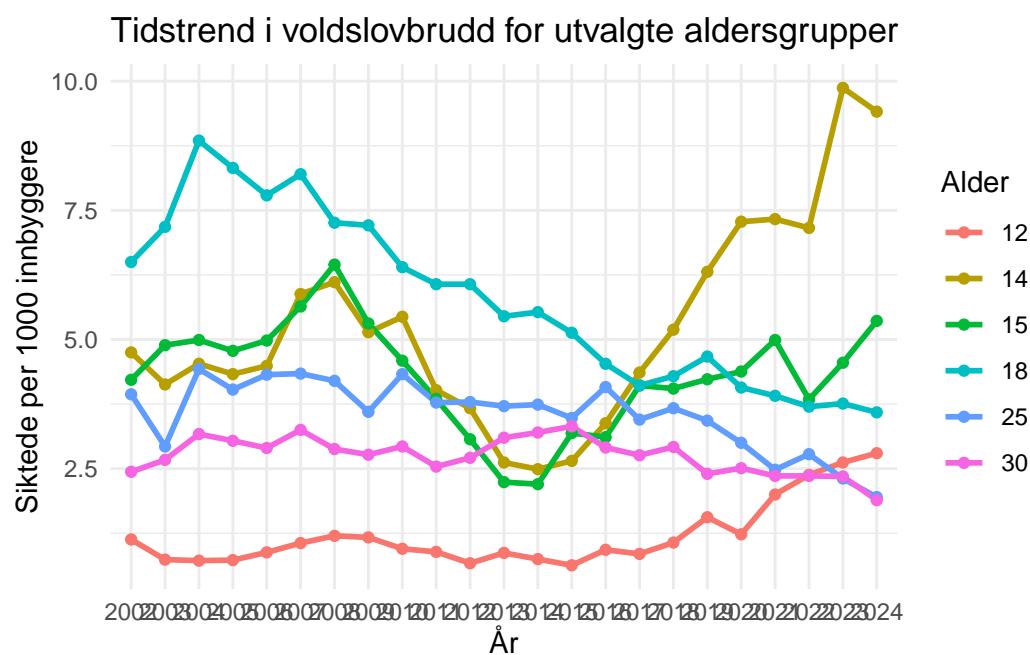
Figuren blir mye lettere å lese, og mønsteret er tydelig: vi kan se hvordan ungdomskriminaliteten har utviklet seg over tid.

4.9 Tidstrend for utvalgte aldersgrupper

Til nå har vi hatt alder på x-aksen og brukt farger for å skille mellom år. Vi kan også snu perspektivet: sette *år* på x-aksen og bruke farger for å skille mellom *aldersgrupper*. Da ser vi tidstrenden – hvordan kriminaliteten har utviklet seg over tid for ulike aldersgrupper.

```
# Velg ut noen utvalgte aldersgrupper
# factor() gjør alder om til en kategori slik at fargene blir diskrete
krim_alder <- krim_vold %>%
  filter(alder %in% c(12, 14, 15, 18, 25, 30)) %>%
  mutate(alder = factor(alder))
```

```
# Tidstrend: år på x-aksen, en linje per aldersgruppe
ggplot(krim_alder, aes(x = aar, y = pr1000, group = alder, color = alder)) +
  geom_line(linewidth = 1) +
  geom_point() +
  ggtitle("Tidstrend i voldslovbrudd for utvalgte aldersgrupper") +
  labs(x = "År",
       y = "Siktede per 1000 innbyggere",
       color = "Alder") +
  theme_minimal()
```



Legg merke til at koden er nesten identisk med den forrige figuren – vi har bare byttet hva som er på x-aksen (`aar` i stedet for `alder`) og hva som styrer fargene (`alder` i stedet for `aar`). `factor(alder)` gjør at alder behandles som en kategori med diskrete farger i stedet for en sammenhengende skala.

Figuren viser at utviklingen er svært ulik for ulike aldersgrupper. 18-åringene har det høyeste nivået, men alle aldersgrupper kan studeres over tid. Med noen få endringer i koden har vi et helt annet perspektiv på de samme dataene.

4.10 Oppsummering

I dette kapittelet har du gjennomført en hel analyse fra start til mål:

1. Lastet pakker med `library()`
2. Lest inn data fra en Excel-fil med `read_excel()`
3. Utforsket dataene med `head()`, `glimpse()` og `table()`
4. Laget figurer med `ggplot()`, `geom_line()` og `geom_point()`
5. Lagt til titler og etiketter med `ggtitle()` og `labs()`
6. Endret utseende med `theme_minimal()`
7. Filtrert data med `filter()` og pipen `%>%`

Alt dette er verktøy du vil bruke gjennom hele boken. Hvert tema dekkes grundig i sine egne kapitler: innlesning av data i Del II, datahåndtering med tidyverse i Del III, og grafikk med ggplot i Del IV.

Hvis det er noe du ikke helt forsto – det er helt normalt. Poenget nå var å se at R fungerer, at du kan produsere noe meningsfullt med noen få linjer kode, og at den lagvise oppbyggingen av figurer gir deg full kontroll over resultatet.

4.11 Oppgaver

Exercise 4.1. Kjør all koden i dette kapittelet selv i RStudio. Gjør noen endringer og kjør på nytt – for eksempel, prøv å endre tittelen på en figur, legg til `geom_point(color = "red")` i stedet for `geom_point()`, eller filtrer på en annen aldersgruppe. Målet er at du skal se at endringene dine faktisk fungerer.

Exercise 4.2. Filtrer `krim_alle` på en annen lovbruddsgruppe enn “Vold og mishandling”. Bruk `table(krim_alle$lovbruddsgruppe)` for å se hvilke grupper som finnes. Lag en figur tilsvarende den vi lagde for voldskriminalitet.

5 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, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1~
$ klasse89 <fct> III Rutinefunksjonærer, VIIa Ufaglærte arbeidere, II Nedre se~
$ promot    <fct> NEI, JA, JA, NEI, NEI, NEI, NEI, JA, NEI, 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~
```

5.1 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()`, “*aesthetics*”, 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

5.2 Kategoriske variabel

5.2.1 Stolpediagram

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

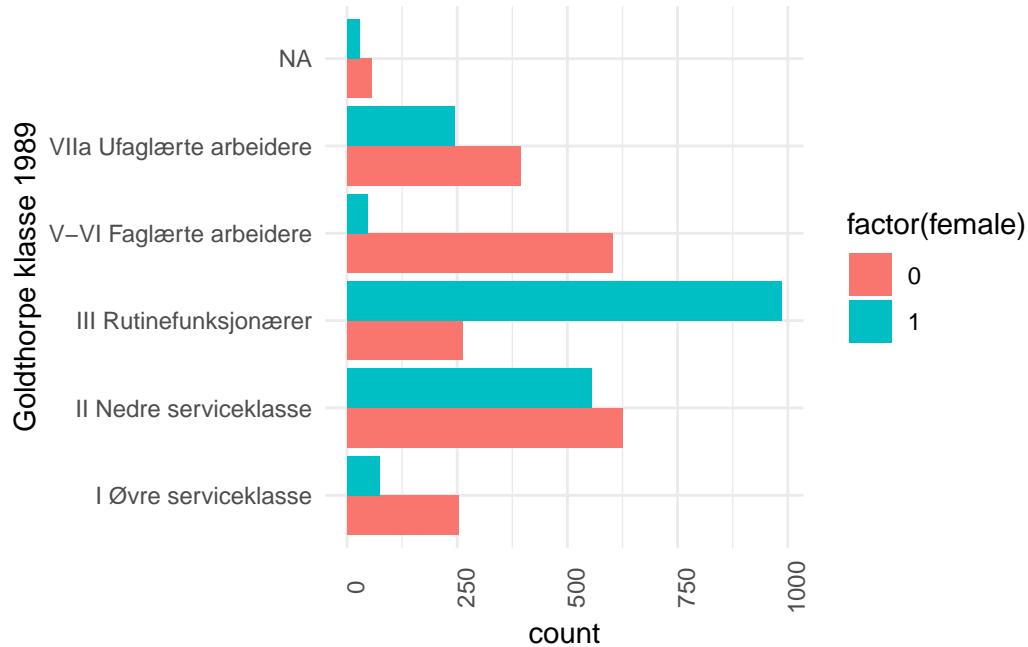


Noen ganger ønsker man å vise fordelingen for to ulike grupper, la oss si for kjønn. En mulighet er da å rett og slett lage to stolpediagram ved siden av hverandre. Til dette kan man spesifisere at dataene er gruppert etter variablen *female* og at fyllfargen skal settes etter denne variablen med `fill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

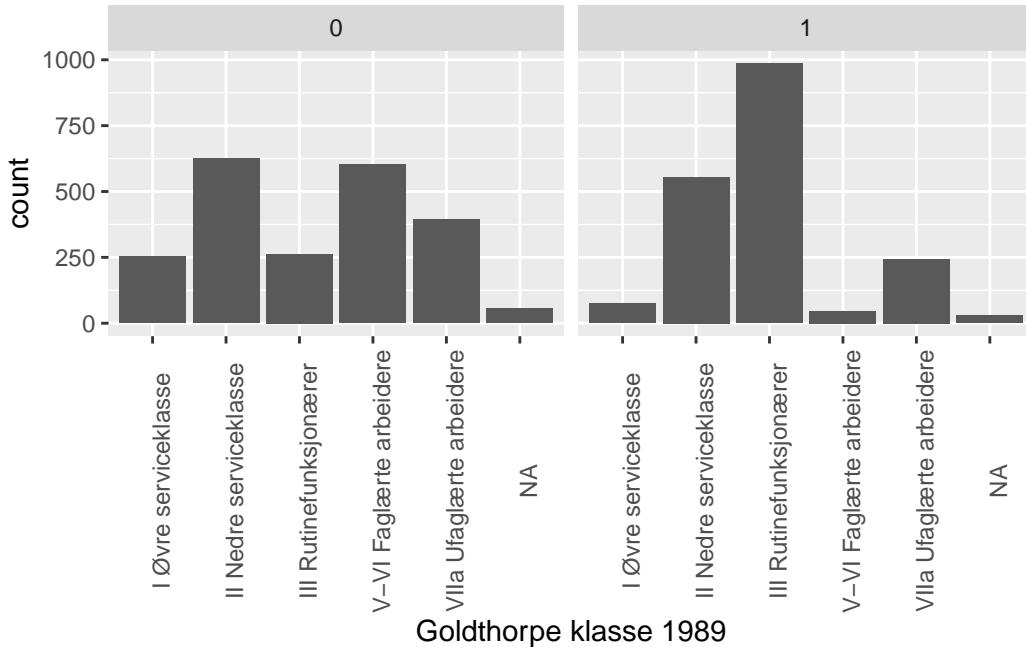
```
ggplot(abu89, aes(x = klasse89, group = female, fill = factor(female))) +
  geom_bar(position="dodge") +
  theme_minimal()
```

```
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:

5.2.2 Kakediagram

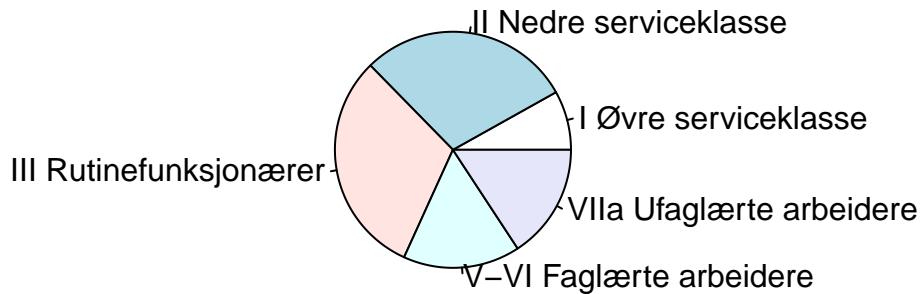
Generelt er ikke kakediagram å anbefale da korrekt tolkning involverer å tolke et areal som inneholder vinkel. Med få kategorier som er rimelig forskjellig kan det gi et ok inntrykk, men ofte ender man opp med å måtte skrive på tallene likevel. Vi tar det med her egentlig bare fordi mange insisterer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

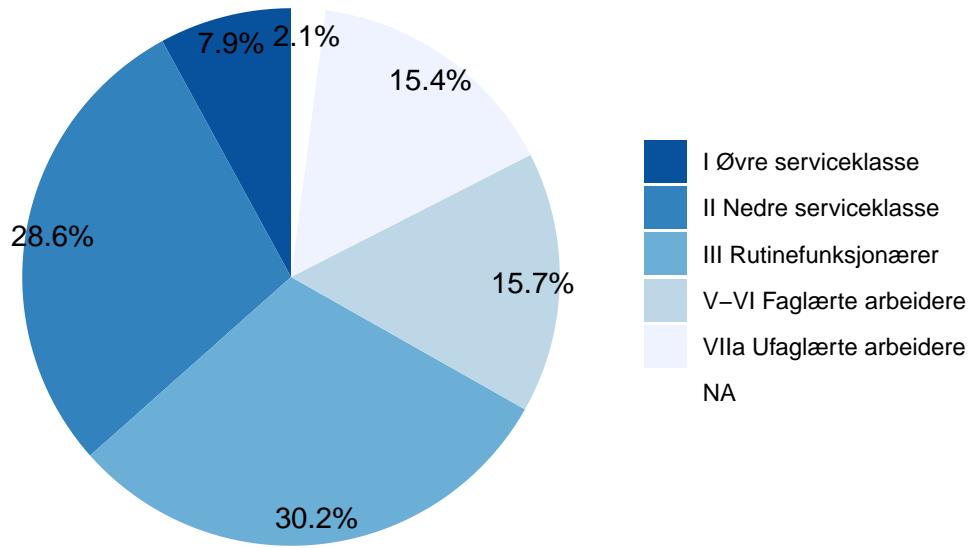
```
pie(tab)
```



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).

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

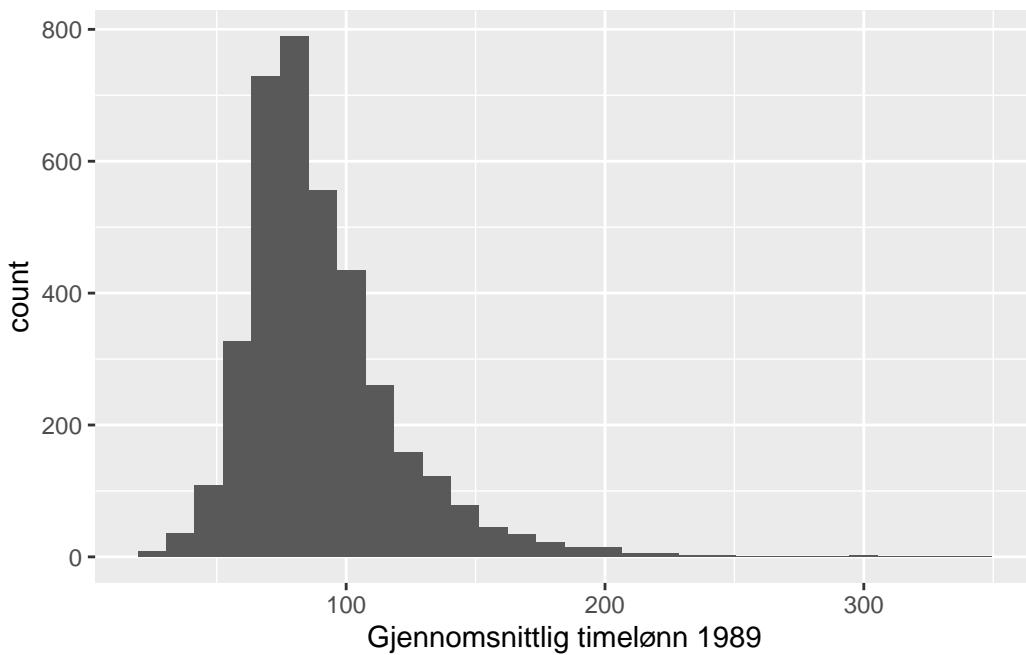
ggplot(pc, aes(x = "", y = pct, fill = (klasse89))) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0) +
  theme_void()+
  geom_text( aes(label = paste0( round(pct,1), "%"), x = 1.4),
             position = position_stack(vjust=.5), check_overlap = F) +
  labs(x = NULL, y = NULL, fill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



5.3 Kontinuerlige variable

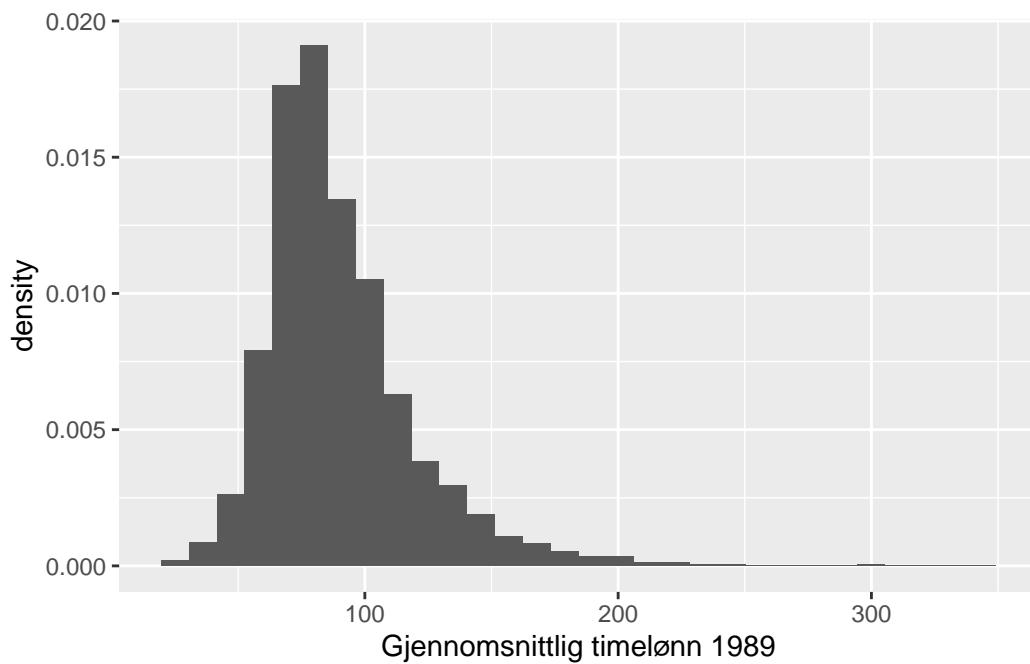
5.3.1 Histogram

```
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.

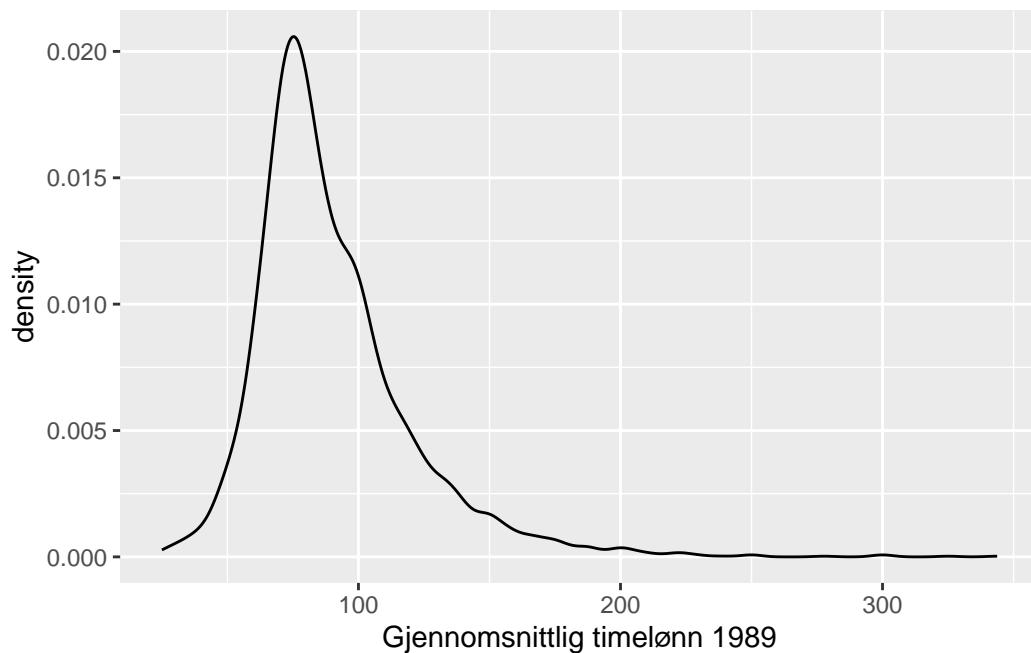
```
ggplot(abu89, aes(x = time89, y = ..density..)) +  
  geom_histogram()
```



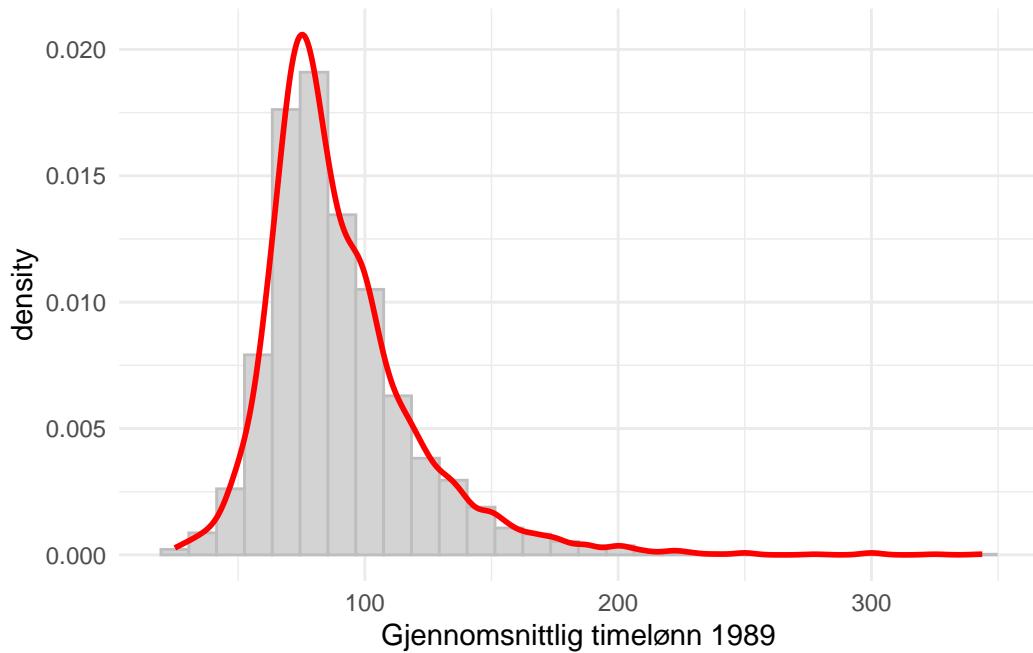
5.3.2 Density plot

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()
```

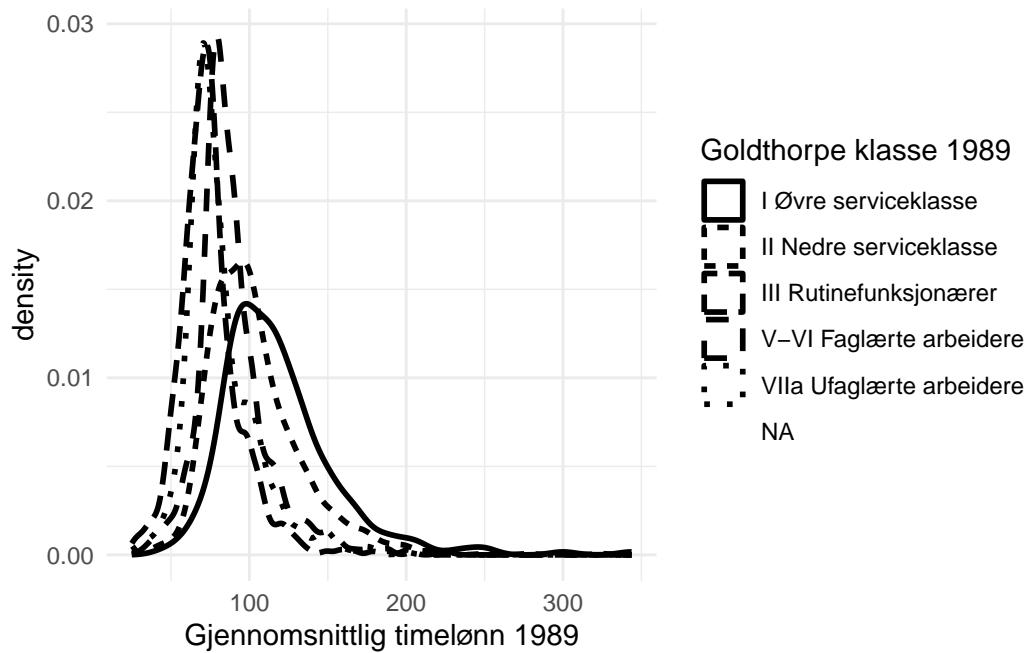


```
ggplot(abu89, aes(x = time89)) +  
  geom_histogram(aes(y = ..density..), fill = "lightgrey", col = "grey") +  
  geom_density(col = "red", linewidth = 1) +  
  theme_minimal()
```

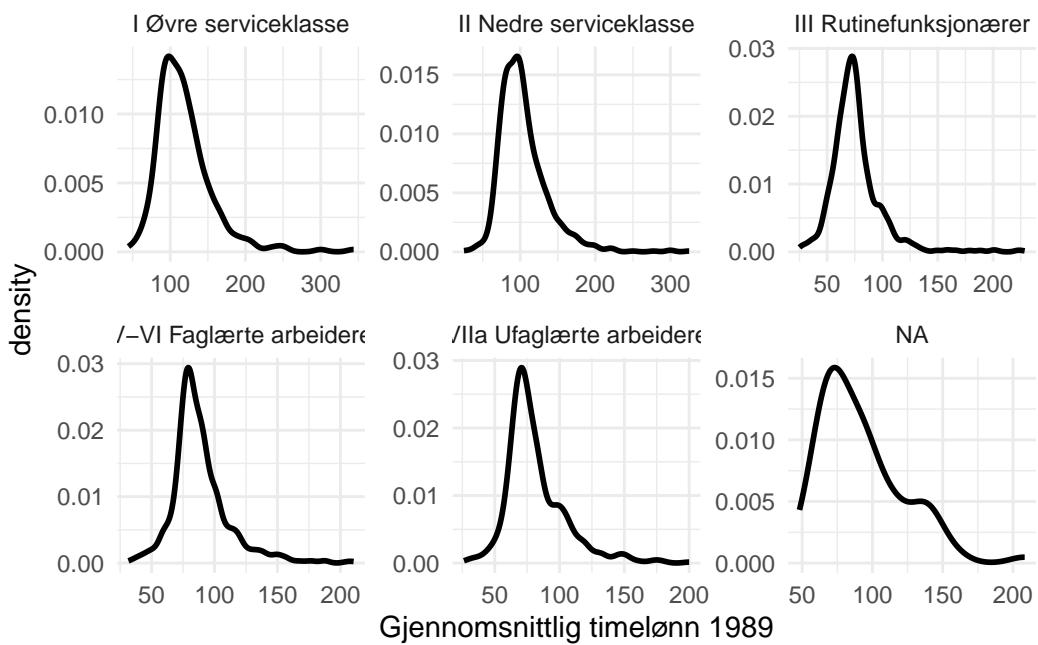


En fordel med denne fremstillingen er at det er lettere å sammenligne grupper. Her er et eksempel med density plot etter hvor mye man drikker.

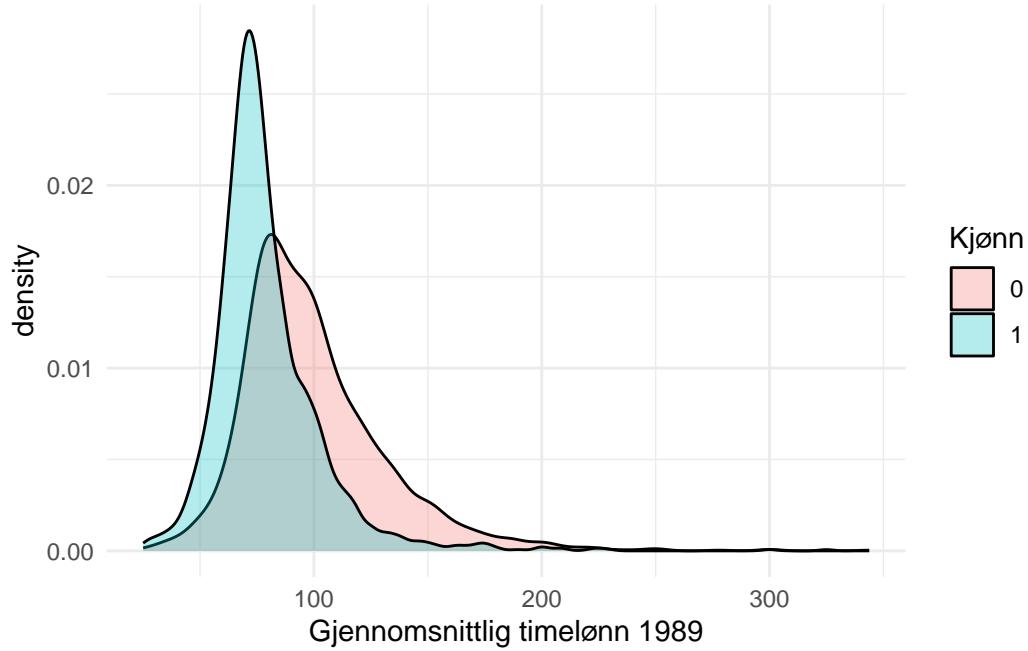
```
ggplot(abu89, aes(x = time89, group = klasse89, linetype = klasse89)) +  
  geom_density(linewidth = 1) +  
  guides(fill = guide_legend(override.aes = list(shape = 1))) +  
  theme_minimal()
```



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



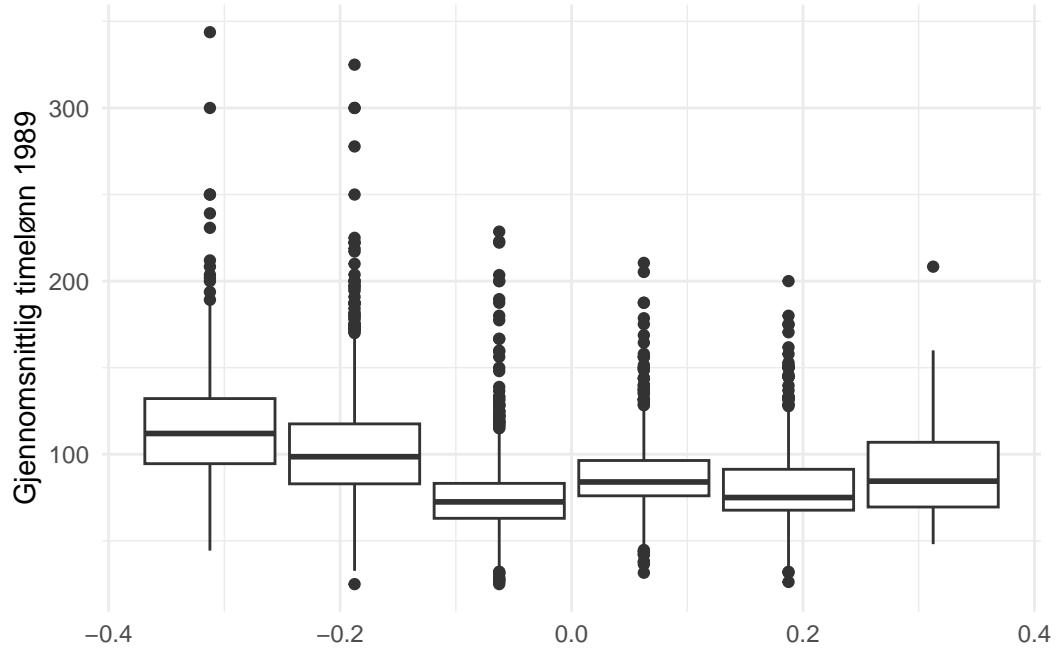
```
ggplot(abu89, aes(x = time89, group = female, fill = factor(female))) +
  geom_density(alpha = .3) +
  guides(fill=guide_legend(title="Kjønn"))+
  theme_minimal()
```



5.3.3 Flere variable samtidig

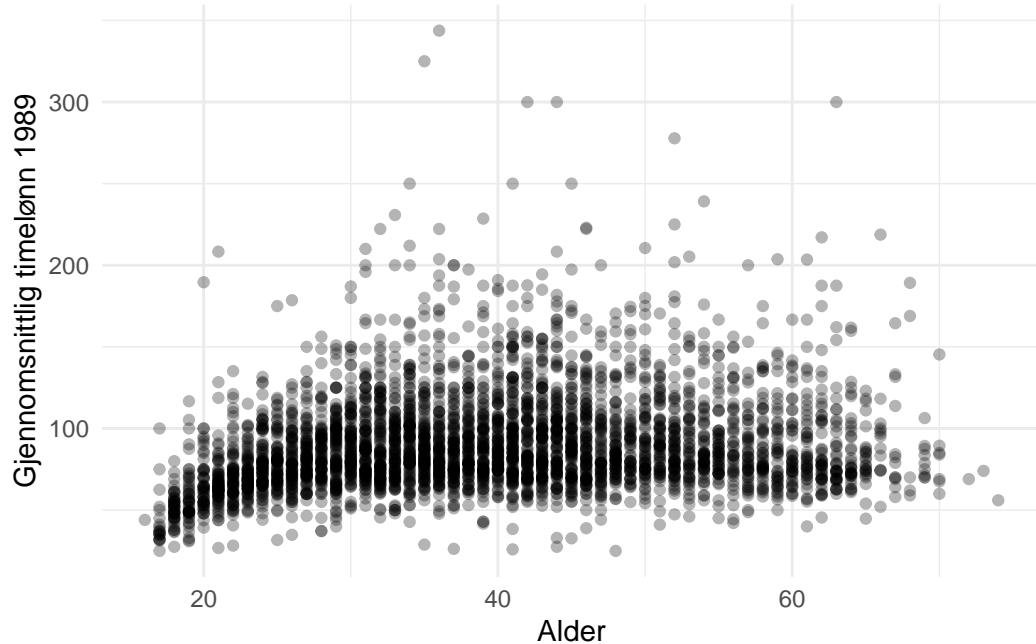
5.3.3.1 Bokspot

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

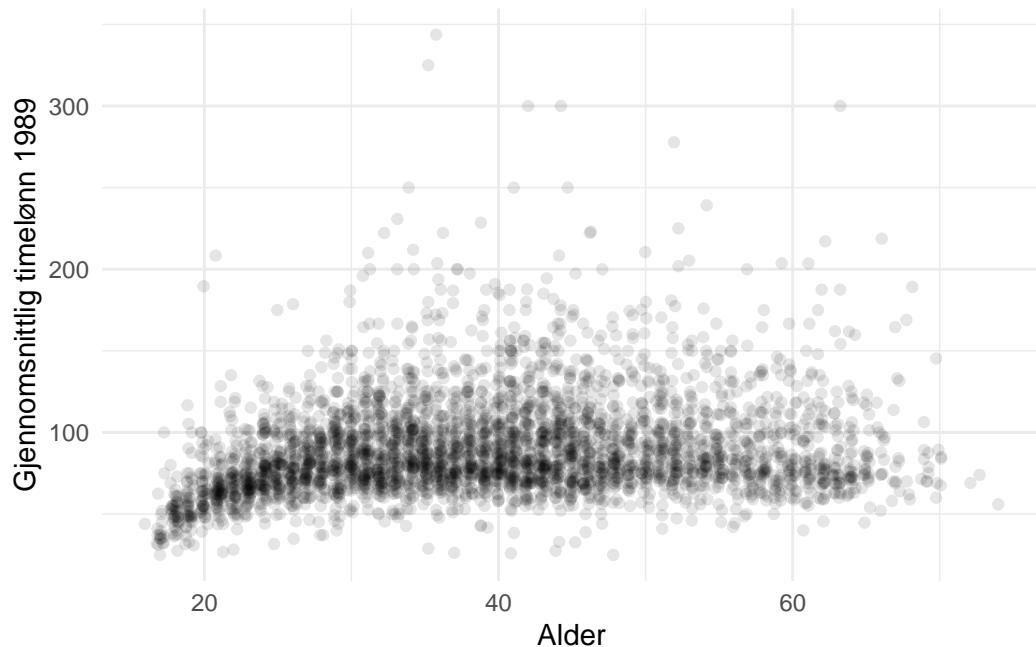


5.3.3.2 Scatterplot

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



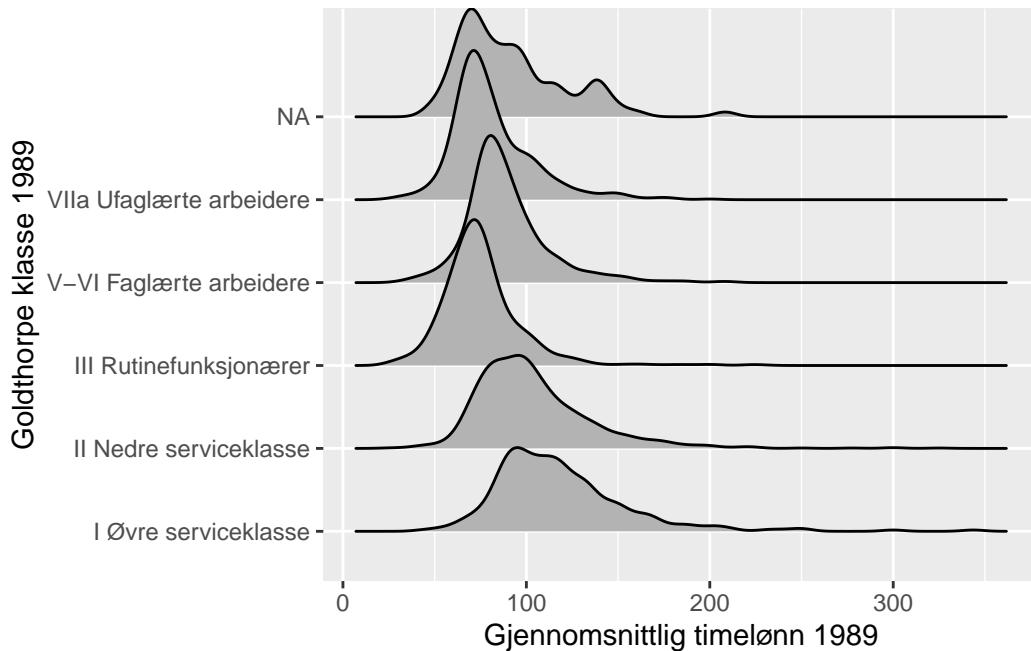
```
ggplot(abu89, aes(x = age, y = time89)) +  
  geom_jitter(alpha=.1, width = .3)+  
  theme_minimal()
```



5.3.3.3 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()
```



5.4 Oppgaver

Slå opp i boken [R for data science](#) hvis du står fast eller ikke skjønner hva koden betyr.

Exercise 5.1. 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 5.2. Last inn datasettet wagepan fra pakken `wooldridge` i R. Velg noen variable som du selv tenker kan være informative å se nærmere på. Bruk de samme teknikkene på disse variablene.

6 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.

6.1 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 regressjonsobjekt 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 variablene 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 variablene man har angitt etter dollar tegnet.

6.1.1 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:

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

```
25%
71
```

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

```
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.

```
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()`:

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

```
[1] 90.14948
```

Standardavviket gis ved `sd()`:

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

```
[1] 30.31473
```

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

```
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
```

6.2 Professionelle tabeller med `gtsummary`

For å lage ordentlig profesjonelle 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 analyserbart informasjon.

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

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.

Characteristic	N = 4,127¹
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%)

¹Median (Q1, Q3); n (%)

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 variablene 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")
```

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()`.

Characteristic	N = 4,127 ¹
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%)
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%)

¹Mean (SD); n (%)

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()
```

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`

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

¹Mean (SD); n (%)

= ~ .x %>% , så kommer `tbl_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}"
)
```

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](#).

Characteristic	Public, N = 1602			Private	
	N	Kvinner N = 1,016 ¹	Menn N = 586 ¹	N	Kvinner N
Gjennomsnittlig timelønn 1989	1,403	82 (23)	100 (28)	2,356	76 (24)
År utdanning	1,602	2.88 (2.67)	4.22 (3.12)	2,525	1.82 (1.9)
Alder	1,602	42 (12)	43 (11)	2,525	37 (13)
Klasse	1,592			2,450	
I Øvre serviceklasse		55 (5.4%)	150 (26%)		19 (2.1%)
II Nedre serviceklasse		340 (34%)	196 (34%)		215 (24%)
III Rutinefunksjonærer		507 (50%)	64 (11%)		479 (54%)
V-VI Faglærte arbeidere		5 (0.5%)	114 (20%)		41 (4.6%)
VIIa Ufaglærte arbeidere		104 (10%)	57 (9.8%)		140 (16%)
Noen gang forfremmet	1,602			2,525	
NEI		696 (69%)	318 (54%)		612 (67%)
JA		320 (31%)	268 (46%)		306 (33%)
Bedriftserfaring	1,602	0.93 (0.85)	1.15 (0.94)	2,525	0.72 (0.7)

¹Mean (SD); n (%)

6.2.1 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}`.

6.3 Manuelle tabeller

Noen ganger trenger man å lage ganske spesifikke ting.

6.3.1 For datasettet totalt

6.3.2 Grupperte statistikker

6.4 Oppgaver

Slå opp i boken [R for data science](#) hvis du står fast eller ikke skjønner hva koden betyr.

Exercise 6.1. 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 6.2. Last inn datasettet wagepan fra pakken `wooldridge` i R. Velg noen variable som du selv tenker kan være informative å se nærmere på. Bruk de samme teknikkene på disse variablene.

7 Kart og romlige data

```
library(tidyverse)
library(sf)
```

I samfunnsvitenskapen er vi ofte interessert i geografisk variasjon. Hvordan varierer kriminaliteten mellom kommuner? Hvor er det flest barnefamilier? Er det regionale forskjeller i valgdeltakelse? Slike spørsmål er mye lettere å forstå med et kart enn med en tabell. Et kart gir deg umiddelbar oversikt over romlige mønstre som kan være vanskelig å fange opp ellers.

I R kan vi lage kart med de samme verktøyene vi allerede kjenner fra ggplot. Det er pakken **sf** (som står for *simple features*) som gjør det mulig å jobbe med geografiske data, og den fungerer smutt sammen med tidyverse.

7.1 Geografiske data og sf-pakken

Geografiske data er i bunn og grunn vanlige datasett med en ekstra kolonne som inneholder geometri – altså de geografiske formene. Det kan være polygoner (som kommunegrenser), linjer (som veier) eller punkter (som adresser). Pakken **sf** håndterer alt dette og lar deg bruke vanlige tidyverse-funksjoner som `filter()`, `mutate()` og `left_join()` på geografiske data.

Hvis du ikke har installert **sf** før, gjør du det slik:

```
install.packages("sf")
```

7.2 Lese inn kartdata

Kartdata kommer typisk i formater som *shapefile* (.shp) eller *GeoJSON* (.geojson). Funksjonen `st_read()` leser begge deler. Her henter vi kommunegrenser fra Geonorge, som er den nasjonale portalen for geografiske data i Norge.

```
komuner <- st_read("https://nedlasting.geonorge.no/geonorge/Basisdata/Kommuner/GeoJSON/Basisdata-Kommuner.geojson")
```

Denne filen er ganske stor, sa det kan ta litt tid a laste ned. Hvis du har lastet ned filen til din egen maskin, leser du den inn slik:

```
kommuner <- st_read("sti/til/kommuner.geojson")
```

La oss se hva vi har fatt:

```
glimpse(kommuner)
```

Du vil se at dette ser ut som en vanlig data.frame, men med en ekstra kolonne som heter `geometry`. Det er der de geografiske formene ligger. Ellers har du kolonner med kommunenummer, kommunenavn og annen informasjon.

7.3 Grunnleggende kart med ggplot

Nar du har et sf-objekt, kan du plotte det direkte med `geom_sf()`. Det er like enkelt som annen ggplot-grafikk:

```
ggplot(kommuner) +  
  geom_sf() +  
  theme_minimal()
```

Dette gir deg et enkelt kart over alle kommuner i Norge. Merk at du ikke trenger a spesifisere x- og y-akser – det ordner `geom_sf()` selv basert pa geometrien.

7.4 Koropletkart: fargelegge regioner etter en variabel

Et koropletkart er et kart der omradene er fargelagt etter en variabel. For a lage dette trenger vi forst noen data a koble pa kartet. La oss lage et lite eksempldatasett med fiktive verdier:

```
# Anta at kommuner-datasettet har en kolonne "kommunenummer"  
# Vi lager noen eksempledata  
eksempel <- kommuner %>%  
  mutate(tilfeldig_verdi = rnorm(n()))
```

Na kan vi fargelegge kartet etter denne variabelen:

```
ggplot(eksempel) +
  geom_sf(aes(fill = tilfeldig_verdi)) +
  scale_fill_viridis_c() +
  theme_minimal() +
  labs(fill = "Verdi")
```

Funksjonen `scale_fill_viridis_c()` gir en fargeskala som er lesbar også for fargeblinde. Du kan også bruke andre fargeskalaer, f.eks. `scale_fill_gradient(low = "white", high = "red")`.

7.5 Koble data til kartgeometrier

I praksis har du som regel dataene dine i et eget datasett og kartgeometriene i et annet. Da må du koble dem sammen med `left_join()`. Nokkelfeltet er typisk kommunenummer.

La oss si at du har et datasett med befolkningstall per kommune:

```
# Eksempel: les inn dine data
mine_data <- data.frame(
  kommunenummer = c("0301", "1103", "4601", "5001"),
  innbyggere = c(709037, 144704, 291189, 212660),
  kommune = c("Oslo", "Stavanger", "Bergen", "Trondheim")
)

# Koble til kartdata
kart_med_data <- kommuner %>%
  left_join(mine_data, by = "kommunenummer")

# Plott
ggplot(kart_med_data) +
  geom_sf(aes(fill = innbyggere)) +
  scale_fill_viridis_c(labels = scales::comma) +
  theme_minimal() +
  labs(fill = "Innbyggere")
```

Merk at kolonnenavnet for kommunenummer må være likt i begge datasettene. Sjekk også at formatet er det samme (f.eks. at begge er tekststrenger med ledende nuller).

7.6 Legge til punkter på kartet

Noen ganger ønsker du å plotte enkeltlokasjoner på kartet, for eksempel hendelser eller institusjoner. Da trenger du koordinater (lengde- og breddegrad) som du gjør om til et sf-objekt med `st_as_sf()`.

```
steder <- data.frame(
  navn = c("Oslo", "Bergen", "Trondheim", "Tromsø"),
  lon = c(10.75, 5.33, 10.40, 18.96),
  lat = c(59.91, 60.39, 63.43, 69.65)
)

# Gjør om til sf-objekt med koordinatreferansesystem WGS84 (EPSG:4326)
steder_sf <- st_as_sf(steder, coords = c("lon", "lat"), crs = 4326)

# Transformer til samme CRS som kartdataene
steder_sf <- st_transform(steder_sf, st_crs(kommuner))
```

Nå kan du legge punktene oppå kartet som et eget lag:

```
ggplot() +
  geom_sf(data = kommuner, fill = "grey90", color = "grey70") +
  geom_sf(data = steder_sf, color = "red", size = 3) +
  theme_minimal()
```

Legg merke til at vi her bruker `ggplot()` uten å spesifisere data først, og i stedet angir `data =` i hvert `geom_sf()`-lag. Det gir oss full kontroll over hvilke datasett som brukes i hvert lag.

7.7 Koordinatreferansesystemer (CRS)

Alle geografiske data har et koordinatreferansesystem (CRS) som sier noe om hvordan posisjoner på jordkloden er representert. De to viktigste å vite om er:

- **EPSG:4326** (WGS84): Lengde- og breddegrad i grader. Dette er det GPS bruker og det du finner på Google Maps.
- **EPSG:25833** (UTM sone 33N): Et projisert system i meter som er vanlig for norske kartdata. Geonorge bruker typisk dette.

Hvis du skal kombinere data med ulike CRS, må du transformere til samme system:

```
# Sjekk CRS
st_crs(kommuner)

# Transformer til en annen CRS
kommuner_wgs84 <- st_transform(kommuner, 4326)
```

I praksis trenger du sjeldent å tenke så mye på dette så lenge du transformer til samme system for du kombinerer datasett.

7.8 Interaktive kart med leaflet

Noen ganger er det nyttig med et interaktivt kart der man kan zoome og klikke. Pakken `leaflet` gjør dette enkelt:

```
install.packages("leaflet")

library(leaflet)

# Leaflet bruker WGS84, så vi transformer først
kommuner_ll <- st_transform(kommuner, 4326)

leaflet(kommuner_ll) %>%
  addTiles() %>%
  addPolygons(
    fillColor = "steelblue",
    weight = 1,
    color = "white",
    fillOpacity = 0.5
  )
```

Leaflet-kart fungerer best i HTML-dokumenter og er særlig nyttige når du utforsker dataene selv. De er ikke egnet for trykte rapporter.

7.9 Kilder til norske geodata

For norske kartdata finnes det flere gode kilder:

- **Geonorge** (geonorge.no): Norges nasjonale portal for geografiske data. Her finner du kommunegrenser, fylkesgrenser, kystlinjer og mye mer. Data er gratis og tilgjengelig i flere formater.
- **SSB** (ssb.no): Statistisk sentralbyra tilbyr kartdata tilpasset deres statistikk, inkludert grunnkretser og delområder.
- **Kartverket** (kartverket.no): Forvalter de offisielle kartdataene i Norge og tilbyr bl.a. topografiske data, stedsnavn og eiendomsdata.

Til vanlig bruk i samfunnsvitenskapelige analyser er kommunegrensene fra Geonorge det du oftest trenger. Husk at kommuneinndelingen endrer seg over tid (kommunesammenslainger), så pass på at kartdataene matcher tidsperioden i dine analysedata.

7.10 Praktisk eksempel: kriminalitetskort over Oslo

La oss nå bruke det vi har lært til å lage et kriminalitetskort. Vi bruker en shapefil med Oslos 16 bydeler og et datasett med syntetiske kriminalitetshendelser som har x/y-koordinater. Dataene er generert for undervisningsformål, men illustrerer en realistisk arbeidsflyt for romlig analyse.

7.10.1 Lese inn shapefil

Shapefiler er et vanlig format for geografiske data. De består egentlig av flere filer (.shp, .dbf, .prj, .shx), men `st_read()` håndterer dette automatisk – du peker bare på .shp-filen:

```
oslo <- st_read("data/shapefiles/oslo.shp", quiet = TRUE)
```

```
glimpse(oslo)
```

```
Rows: 16
Columns: 5
$ AREA      <dbl> 13752297, 7727095, 8257591, 7506340, 4752848, 7041704, 1224-
$ PERIMETER <dbl> 24549.228, 15484.756, 28349.785, 37783.397, 12090.874, 2304-
$ KODE       <chr> "030112", "030109", "030105", "030101", "030102", "030110", ~
$ BYDELSNAVN <chr> "ALNA", "BJERKE", "FROGNER", "GAMLE OSLO", "GRÜNERLØKKA", "~"
$ geometry   <MULTIPOLYGON [m]> MULTIPOLYGON (((268660.9 66..., MULTIPOLYGON (~
```

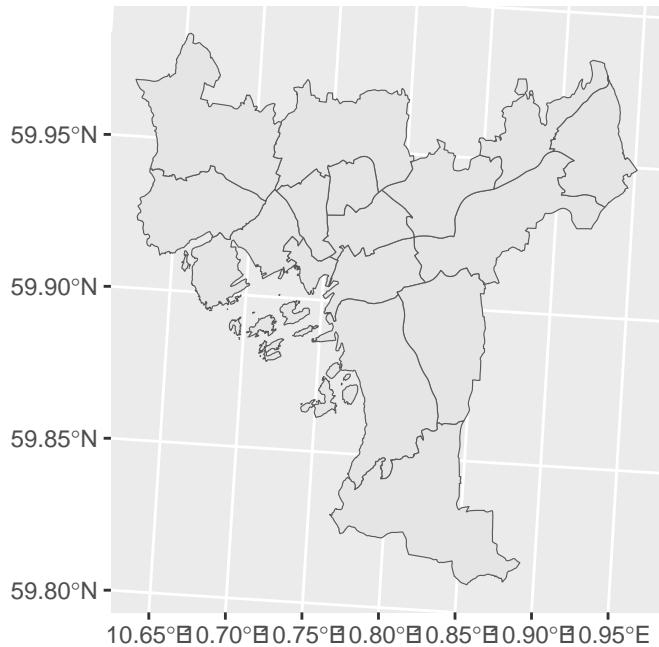
Vi ser at datasettet har 16 rader (en per bydel) med bydelskode, bydelsnavn og geometri. Geometrikolonnen inneholder polygonene som tegner bydelsgrensene.

7.10.2 Grunnkart over Oslo

La oss bygge opp et kart steg for steg, akkurat som med vanlig ggplot.

Det enkleste mulige kartet:

```
ggplot(oslo) +  
  geom_sf()
```



`geom_sf()` forstår geometrien automatisk – du trenger ikke spesifisere x- og y-akser. La oss forbedre utseendet:

```
ggplot(oslo) +  
  geom_sf(fill = "gray95") +  
  theme_void()
```



`theme_void()` fjerner akser og bakgrunn, som passer bedre for kart enn `theme_minimal()`.
Na lagrer vi grunnkartet i et objekt slik at vi kan legge nye lag oppa:

```
kart <- ggplot(oslo) +  
  geom_sf(fill = "gray95") +  
  coord_sf(datum = st_crs(oslo)) +  
  theme_void()  
kart
```



`coord_sf(datum = st_crs(oslo))` sørger for at projeksjonen matcher kartdataene.

7.10.3 Lese inn og utforske kriminalitetsdata

Nå leser vi inn kriminalitetsdataene. Disse er lagret som en RDS-fil, som er et R-format vi har sett tidligere:

```
krim <- readRDS("data/shapefiles/syntetisk_krim.rds")
```

```
head(krim)
```

	saksnr	krimtypekodenavn	aar	maaned	dag	time	x	y
1	14039877	05 NARKOTIKA	2017	Mar	4	21	270800	6653100
2	13917288	03 VOLD	2016	Nov	2	3	261900	6649700
3	14377224	05 NARKOTIKA	2018	Feb	6	20	262900	6649300
4	13796941	02 VINNING	2016	Jul	3	23	262500	6649700
5	14406573	02 VINNING	2017	Apr	7	0	264300	6649700
6	14449912	02 VINNING	2018	May	4	14	261100	6649100

```
glimpse(krim)
```

```

Rows: 198,365
Columns: 8
$ saksnr          <dbl> 14039877, 13917288, 14377224, 13796941, 14406573, 144-
$ krimtypekodenavn <chr> "05 NARKOTIKA", "03 VOLD", "05 NARKOTIKA", "02 VINNIN-
$ aar              <dbl> 2017, 2016, 2018, 2016, 2017, 2018, 2018, 2017, 2016, ~
$ maaned           <chr> "Mar", "Nov", "Feb", "Jul", "Apr", "May", "Mar", "Sep~
$ dag               <dbl> 4, 2, 6, 3, 7, 4, 6, 5, 6, 5, 3, 6, 2, 3, 3, 5, 3, 1, ~
$ time              <dbl> 21, 3, 20, 23, 0, 14, 0, 0, 14, 12, 21, 0, 18, 0, ~
$ x                 <dbl> 270800, 261900, 262900, 262500, 264300, 261100, 26400-
$ y                 <dbl> 6653100, 6649700, 6649300, 6649700, 6649700, 6649100, ~

```

```
table(krim$krimtypekodenavn)
```

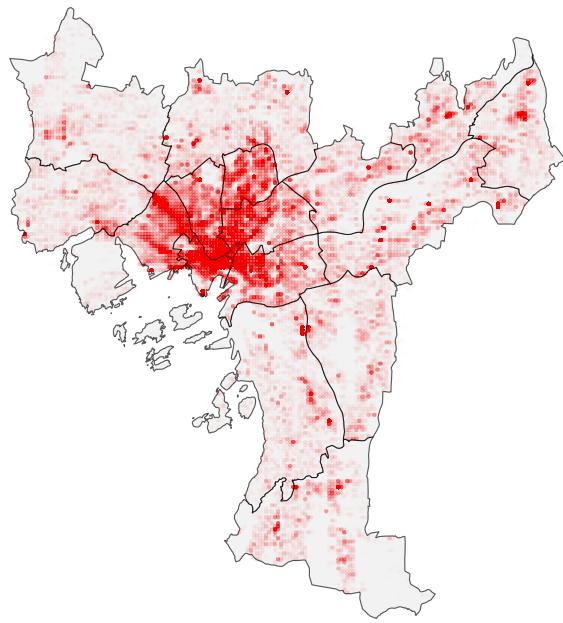
02 VINNING	03 VOLD	05 NARKOTIKA	06 SKADEVERK
119387	29921	27455	21602

Datasetssettet har omrent 198 000 hendelser. Hver rad er en kriminell hendelse med koordinater (x og y i UTM-meter), kriminalitetstype, ar, maned, ukedag og klokkeslett. Koordinatene er avrundet til 100-meters ruter.

7.11 Punktkart

Den enkleste maten a vise hendelsene pa kartet er a plotte hvert punkt:

```
kart +
  geom_point(data = krim, aes(x = x, y = y),
              color = "red", alpha = 0.01, size = 0.01)
```



Med `alpha = 0.01` (nesten gjennomsiktig) og `size = 0.01` (sma punkter) kan vi vise alle hendelsene uten at kartet blir helt rødt. Tettere områder får sterkere farge fordi mange halvgjennomsiktige punkter legges oppa hverandre.

7.12 Rasterkart (heatmap)

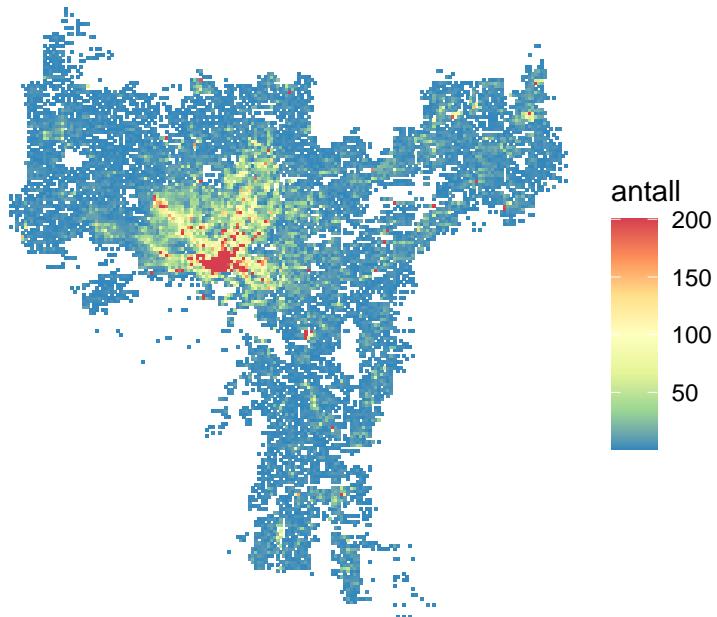
Punktkartet over er nyttig for å se det overordnede mønsteret, men for å sammenligne områder mer presist trenger vi å *aggregere* dataene. Siden koordinatene allerede er avrundet til 100-meters ruter, kan vi telle antall hendelser per rute:

```
krimplot <- krim %>%
  group_by(x, y) %>%
  summarise(antall = n()) %>%
  mutate(antall = ifelse(antall >= 200, 200, antall))
```

Vi begrenser verdiene til maksimalt 200 for å unngå at noen ekstremt tette ruter (typisk sentrum) dominerer hele fargeskalaen.

Nå kan vi lage et rasterkart der hver rute er fargelagt etter antall hendelser:

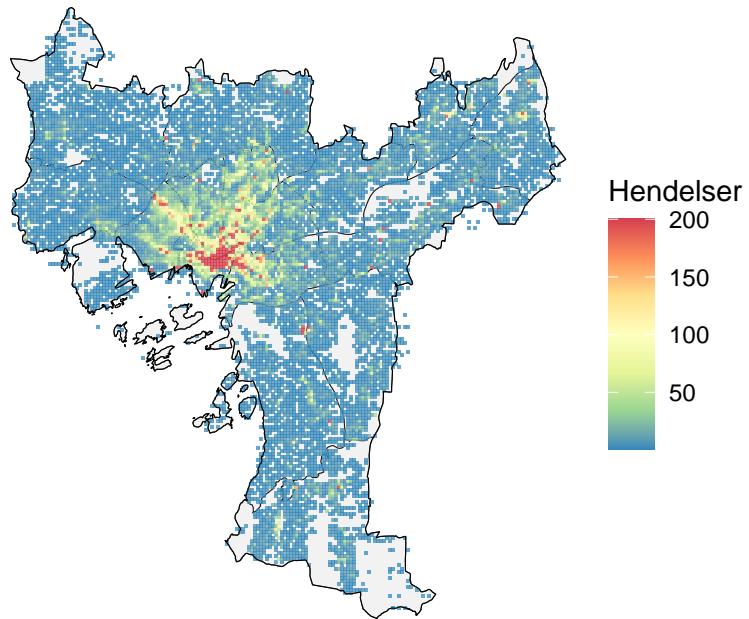
```
ggplot(krimplot, aes(x = x, y = y, fill = antall)) +  
  geom_tile() +  
  coord_sf(datum = st_crs(oslo)) +  
  theme_void() +  
  scale_fill_distiller(palette = "Spectral")
```



`geom_tile()` tegner en firkant per rute, fargelagt etter antall hendelser. `scale_fill_distiller(palette = "Spectral")` gir en fargeskala som går fra blatt (fa hendelser) til rødt (mange hendelser).

For bedre kontekst legger vi til Oslos omriss oppa heatmapen:

```
oslo_omkr <- st_union(oslo)  
  
kart +  
  geom_tile(data = krimplot, aes(x = x, y = y, fill = antall), alpha = 0.75) +  
  scale_fill_distiller(palette = "Spectral") +  
  geom_sf(data = oslo_omkr, color = "black", fill = NA) +  
  labs(fill = "Hendelser")
```



`st_union()` slar sammen alle bydelspolygonene til en ytre grense. `fill = NA` gjør at bare grenselinjen tegnes, slik at heatmapen synes gjennom. Merk at vi bruker grunnkartet (`kart`) som utgangspunkt og legger nye lag oppa – akkurat samme prinsipp som med vanlige ggplot-figurer.

7.13 Oppdelt etter kriminalitetstype

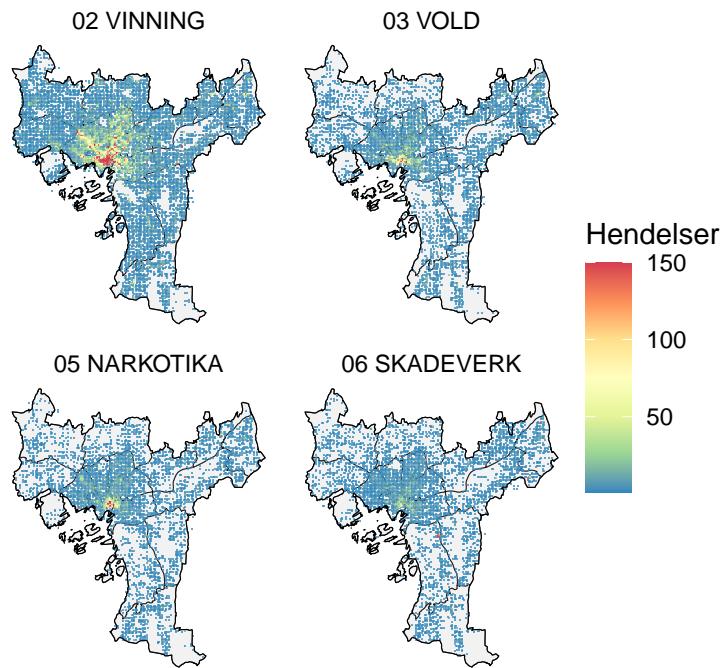
Med `facet_wrap()` kan vi lage ett delkart per kriminalitetstype. Vi aggregerer på nytt, men nå også gruppert etter type:

```

krimplot_type <- krim %>%
  group_by(x, y, krimtypekodenavn) %>%
  summarise(antall = n()) %>%
  mutate(antall = ifelse(antall >= 150, 150, antall))

kart +
  geom_tile(data = krimplot_type, aes(x = x, y = y, fill = antall), alpha = 0.75) +
  scale_fill_distiller(palette = "Spectral") +
  geom_sf(data = oslo_omkr, color = "black", fill = NA) +
  facet_wrap(~krimtypekodenavn) +
  labs(fill = "Hendelser")

```



Na ser vi at de ulike kriminalitetstypene har svart forskjellig geografisk fordeling. Vinning-skriminalitet er spredd over hele byen, mens narkotika er mer konsentert. `facet_wrap()` fungerer pa kart akkurat som pa vanlige ggplot-figurer.

7.14 Oppsummering

Kart i R følger samme logikk som annen ggplot-grafikk: du har data, du velger en geometrisk form (`geom_sf()`), og du tilpasser utseendet. Med `sf`-pakken kan du lese inn kartfiler, koble pa dine egne data, og lage både statiske og interaktive kart. De viktigste funksjonene a huske er:

Funksjon	Hva den gjor
<code>st_read()</code>	Leser inn geografiske data
<code>geom_sf()</code>	Plotter sf-objekter i ggplot
<code>st_as_sf()</code>	Gjor vanlig data.frame om til sf-objekt
<code>st_transform()</code>	Endrer koordinatreferansesystem
<code>st_crs()</code>	Sjekker koordinatreferansesystemet
<code>st_union()</code>	Slar sammen geometrier til en
<code>geom_tile()</code>	Tegner rasterkart (heatmap)
<code>facet_wrap()</code>	Deler kartet opp etter en variabel

8 Nettverksanalyse

```
library(tidyverse)
library(igraph)
```

Nettverksanalyse handler om relasjoner mellom enheter. Mens vi i regresjonsanalyse fokuserer på egenskaper ved individer (f.eks. alder, inntekt, utdanning), fokuserer nettverksanalyse på *forbindelsene* mellom dem. Hvem kjenner hvem? Hvem samarbeider med hvem? Hvilke organisasjoner har overlappende styremedlemmer? Dette er typiske spørsmål som nettverksanalyse kan belyse.

8.1 Hva er et nettverk?

Et nettverk består av to grunnleggende byggeklosser: *noder* og *kanter*. Nodene er enhetene i nettverket (f.eks. personer, organisasjoner, land), mens kantene er forbindelsene mellom dem (f.eks. vennskap, handelsrelasjoner, kommunikasjon).

I samfunnsvitenskap er vi ofte interessert i *sosiale nettverk*, der nodene er personer og kantene representerer en eller annen form for relasjon. Men nettverksanalyse brukes også på mange andre typer data.

8.2 Typer nettverk

Det finnes ulike typer nettverk, og det er viktig å vite hvilken type man jobber med:

- **Urettet nettverk:** Kantene gar begge veier. Vennskap er et typisk eksempel: hvis A er venn med B, så er B venn med A.
- **Rettet nettverk:** Kantene har en retning. Twitter-folging er et eksempel: A kan følge B uten at B følger A tilbake.
- **Vektet nettverk:** Kantene har en styrke. F.eks. kan man vekte etter hvor mange ganger to personer har kommunisert.

8.3 Lage nettverksobjekter med igraph

Pakken `igraph` er det mest brukte verktøyet for nettverksanalyse i R. La oss starte med å lage et lite eksemplennettverk. Tenk deg en liten vennegjeng:

```
venner <- graph_from_literal(Anna -- Bjorn,
                               Anna -- Cecilie,
                               Bjorn -- Cecilie,
                               Bjorn -- David,
                               Cecilie -- Elin,
                               David -- Elin,
                               David -- Fredrik)

venner
```

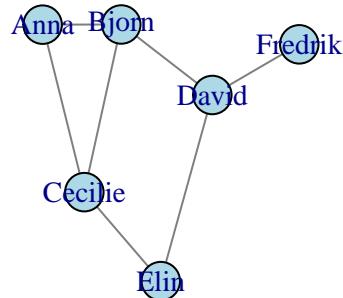
```
IGRAPH 23765de UN-- 6 7 --
+ attr: name (v/c)
+ edges from 23765de (vertex names):
[1] Anna --Bjorn   Anna --Cecilie Bjorn --Cecilie Bjorn --David
[5] Cecilie--Elin   David --Elin    David --Fredrik
```

Her har vi laget et urettet nettverk med seks personer og syv relasjoner. Dobbel bindestrek -- betyr urettet kant. Hadde vi brukt +-+ ville det blitt rettet.

Vi kan plotte dette med en gang:

```
plot(venner,
      vertex.color = "lightblue",
      vertex.size = 30,
      vertex.label.cex = 0.9,
      edge.color = "gray50",
      main = "Vennenettverket")
```

Vennenettverket



8.4 Kantlister og nabomatriser

I praksis lager man sjeldent nettverk for hånd slik som ovenfor. Data kommer ofte som en *kantliste* (edge list), som er en tabell med to kolonner: fra-node og til-node. Hver rad representerer en forbindelse mellom to noder.

La oss lese inn det samme vennenettverket fra en CSV-fil:

```
kantliste <- read_csv("data/venner_kantliste.csv")
```

La oss se på datastrukturen:

```
kantliste
```

```
# A tibble: 7 x 2
  fra     til
  <chr>   <chr>
1 Anna    Bjorn
2 Anna    Cecilie
3 Bjorn   Cecilie
4 Bjorn   David
```

```

5 Cecilie Elin
6 David   Elin
7 David   Fredrik

```

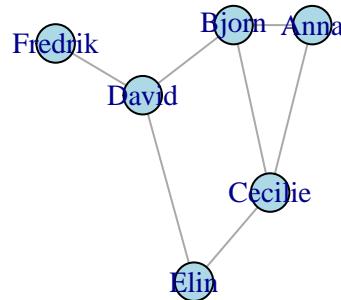
Kantlisten er en helt vanlig tabell med to kolonner: `fra` og `til`. Hver rad er en kant i nettverket. Dette er den vanligste maten a lagre nettverksdata pa, og formatet er lett a lage i Excel eller eksportere fra andre systemer.

Fra en slik kantliste kan vi lage et nettverksobjekt:

```

g <- graph_from_data_frame(kantliste, directed = FALSE)
plot(g,
      vertex.color = "lightblue",
      vertex.size = 30,
      vertex.label.cex = 0.9)

```



En annen vanlig representasjon er en *nabomatrise* (adjacency matrix), der radene og kolonnene er noder, og cellene angir om det er en kant mellom dem (1) eller ikke (0):

```
as_adjacency_matrix(g, sparse = FALSE)
```

	Anna	Bjorn	Cecilie	David	Elin	Fredrik
Anna	0	1	1	0	0	0

Bjorn	1	0	1	1	0	0
Cecilie	1	1	0	0	1	0
David	0	1	0	0	1	1
Elin	0	0	1	1	0	0
Fredrik	0	0	0	1	0	0

Nabomatrisen er symmetrisk fordi nettverket er urettet. For rettede nettverk ville matrisen typisk ikke vært symmetrisk.

8.5 Nettverksmaal

Nettverksanalyse tilbyr en rekke mål for å beskrive både hele nettverket og enkeltnoders posisjon. Her ser vi på tre av de mest sentrale:

8.5.1 Grad (degree)

Grad er det enkleste målet: hvor mange kanter har en node? I et vennskapsnettverk betyr det rett og slett hvor mange venner en person har.

`degree(g)`

Anna	Bjorn	Cecilie	David	Elin	Fredrik
2	3	3	3	2	1

Bjorn og David har flest forbindelser, mens Fredrik bare har en.

8.5.2 Mellomleddsentralitet (betweenness)

Mellomleddsentralitet mäter hvor ofte en node ligger på den korteste veien mellom andre noder. En person med høy mellomleddsentralitet fungerer som en *bro* i nettverket.

`betweenness(g)`

Anna	Bjorn	Cecilie	David	Elin	Fredrik
0.0	3.0	1.5	4.5	1.0	0.0

8.5.3 Naerhetssentralitet (closeness)

Naerhetssentralitet maler hvor naer en node er alle andre noder i nettverket. En person med høy nærhetssentralitet kan ha alle andre raskt.

```
closeness(g)
```

```
Anna      Bjorn     Cecilie    David     Elin      Fredrik
0.11111111 0.14285714 0.12500000 0.14285714 0.12500000 0.09090909
```

La oss samle disse malene i en tabell:

```
sentralitet <- tibble(
  navn = V(g)$name,
  grad = degree(g),
  mellomled = round(betweenness(g), 1),
  nærhet = round(closeness(g), 3)
)
sentralitet
```

```
# A tibble: 6 x 4
  navn     grad mellomled nærhet
  <chr>   <dbl>     <dbl>    <dbl>
1 Anna      2        0       0.111
2 Bjorn     3        3       0.143
3 Cecilie   3        1.5     0.125
4 David     3        4.5     0.143
5 Elin      2        1       0.125
6 Fredrik   1        0       0.091
```

Vi ser at Bjorn og David skiller seg ut som de mest sentrale personene i dette nettverket, uansett hvilket mål vi bruker.

8.6 Visualisering med ggraph

For finere grafisk kontroll kan vi bruke pakken `ggraph`, som bygger på ggplot-logikken:

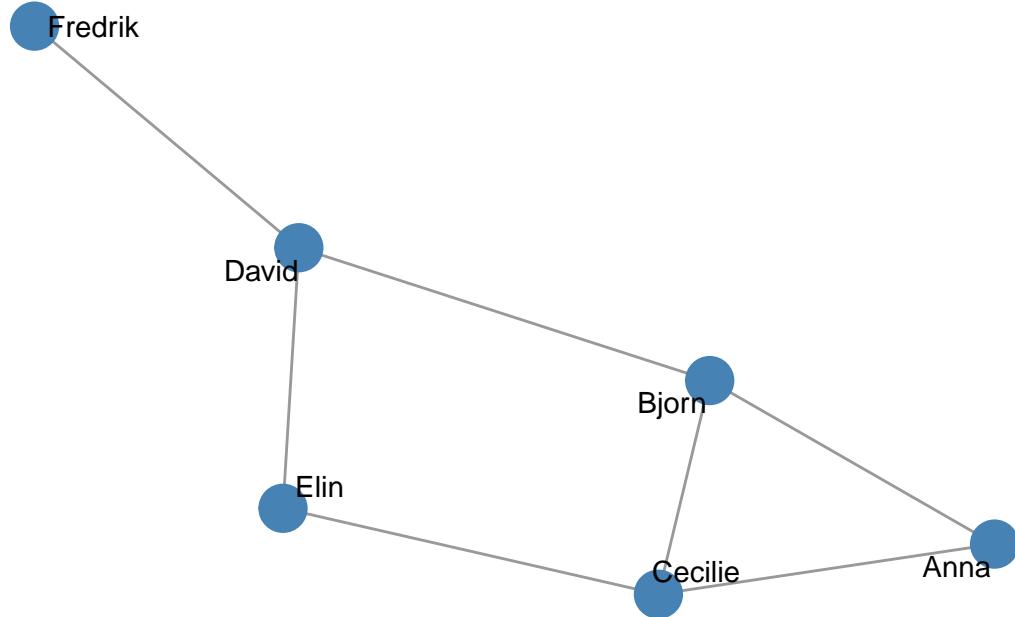
```

library(ggraph)

ggraph(g, layout = "fr") +
  geom_edge_link(color = "gray60") +
  geom_node_point(size = 8, color = "steelblue") +
  geom_node_text(aes(label = name), repel = TRUE, size = 4) +
  theme_void() +
  ggtitle("Vennenettverket med ggraph")

```

Vennenettverket med ggraph



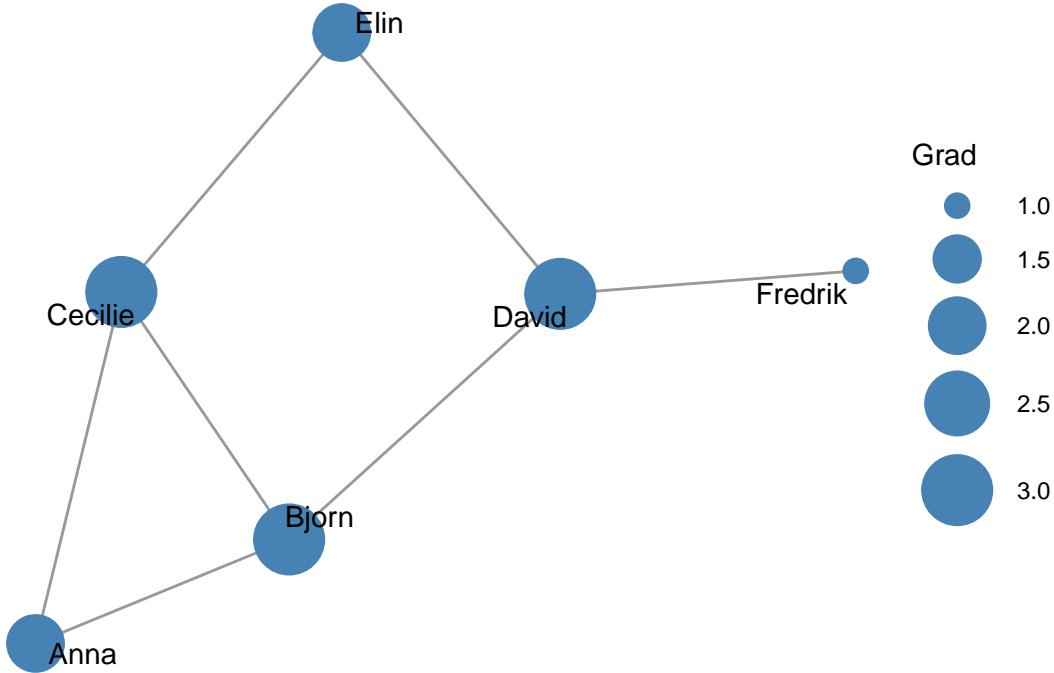
Her bruker vi Fruchterman-Reingold-algoritmen (`layout = "fr"`) for å plassere nodeene. `geom_edge_link()` tegner kantene, `geom_node_point()` tegner nodeene, og `geom_node_text()` legger på navnene. Merk at dette følger ggplot-logikken med lag oppå lag.

Vi kan også la nodestørrelsen reflektere grad:

```

ggraph(g, layout = "fr") +
  geom_edge_link(color = "gray60") +
  geom_node_point(aes(size = degree(g)), color = "steelblue") +
  geom_node_text(aes(label = name), repel = TRUE, size = 4) +
  scale_size_continuous(range = c(4, 12), name = "Grad") +
  theme_void()

```



8.7 Samfunnsdeteksjon

En vanlig oppgave i nettverksanalyse er å identifisere *grupper* (communities/klynger) av noder som er tettere knyttet til hverandre enn til resten av nettverket. Det finnes flere algoritmer for dette. Her bruker vi en enkel metode:

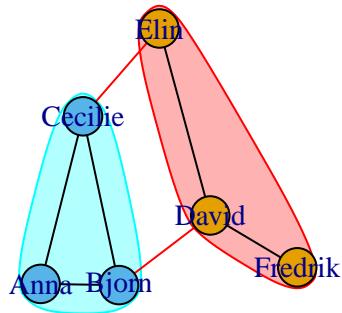
```
grupper <- cluster_walktrap(g)
membership(grupper)
```

Anna	Bjorn	Cecilie	David	Elin	Fredrik
2	2	2	1	1	1

Vi kan fargelegge nettverket etter gruppemedlemskap:

```
plot(grupper, g,
      vertex.size = 30,
      vertex.label.cex = 0.9,
      main = "Grupper i vennenettverket")
```

Grupper i vennenettverket



8.8 Praktisk eksempel: samarbeidsnettverk

Tenk deg et forskningssamarbeid mellom ansatte på et institutt. Vi leser inn en kantliste der hver rad er et samarbeid mellom to forskere, og kolonnen `antall_artikler` angir hvor mange felles artikler de har:

```
samarbeid <- read_csv("data/samarbeid_kantliste.csv")
```

La oss se på datastrukturen:

```
samarbeid
```

```
# A tibble: 11 x 3
  forsker1 forsker2 antall_artikler
  <chr>    <chr>          <dbl>
1 Ola      Kari            5
2 Ola      Per              2
3 Kari     Liv              3
4 Kari     Mona             1
5 Per      Jan              4
6 Liv      Jan              6
7 Ola      Fredrik          0
8 Kari     Fredrik          0
9 Per      Fredrik          0
10 Liv     Fredrik          0
11 Fredrik Ola              0
12 Fredrik Kari             0
13 Fredrik Per              0
14 Fredrik Liv              0
15 Fredrik Mona             0
```

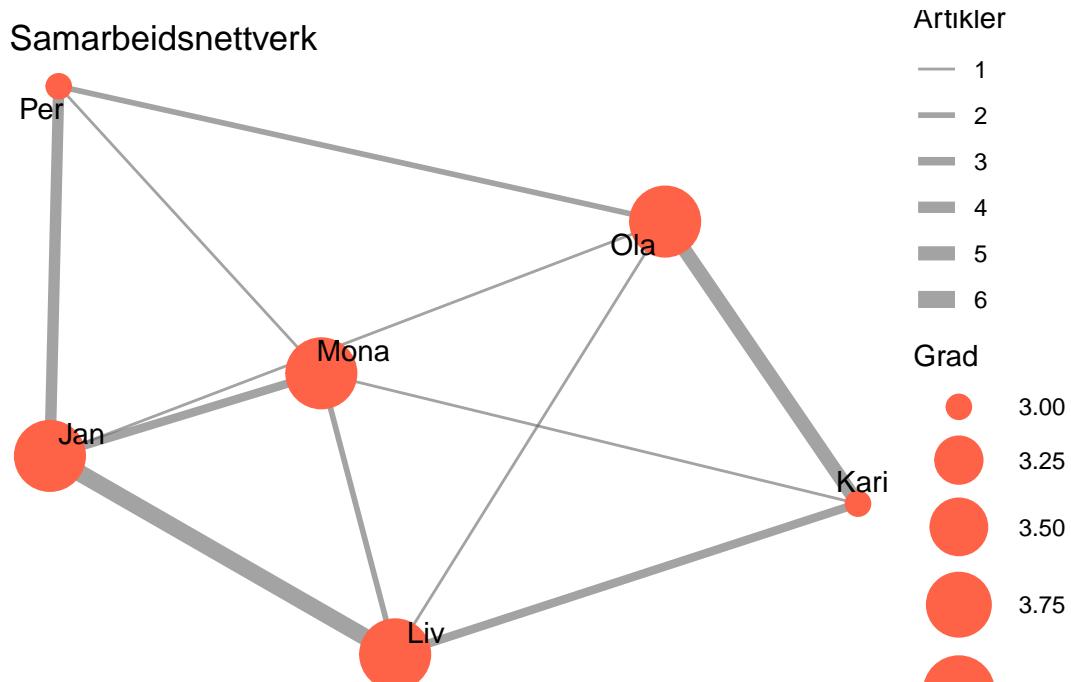
7	Liv	Mona	2
8	Liv	Ola	1
9	Jan	Mona	3
10	Jan	Ola	1
11	Mona	Per	1

Denne kantlisten har tre kolonner: `forsker1` og `forsker2` angir hvem som samarbeider, mens `antall_artikler` er en vekt som sier noe om styrken på relasjonen. Slike vektede kantlister er svart vanlige i praksis.

Nå kan vi lage et nettverksobjekt og bruke vektene:

```
g2 <- graph_from_data_frame(samarbeid, directed = FALSE)
E(g2)$weight <- samarbeid$antall_artikler

ggraph(g2, layout = "fr") +
  geom_edge_link(aes(width = weight), alpha = 0.6, color = "gray40") +
  geom_node_point(aes(size = degree(g2)), color = "tomato") +
  geom_node_text(aes(label = name), repel = TRUE, size = 4) +
  scale_edge_width_continuous(range = c(0.5, 3), name = "Artikler") +
  scale_size_continuous(range = c(4, 12), name = "Grad") +
  theme_void() +
  ggtitle("Samarbeidsnettverk")
```



Her ser vi at kanttykkelsen viser antall felles artikler, mens nodestorrelsen viser antall samarbeidspartnere. Liv og Jan ser ut til å være sentrale i dette nettverket.

8.9 Når er nettverksanalyse nyttig?

Nettverksanalyse er et godt valg når:

- Du er interessert i *relasjoner* mellom enheter, ikke bare egenskaper ved enhetene selv.
- Du vil identifisere sentrale aktorer i et nettverk (f.eks. nøkkelpersoner i en organisasjon).
- Du vil forstå hvordan informasjon eller innflytelse sprer seg.
- Du vil finne grupper eller klynger i data.
- Du vil studere strukturen i et sosialt system.

Nettverksanalyse er mye brukt i sosiologi, statsvitenskap, kriminologi, folkehelse og mange andre fagfelt. Det er et kraftig verktoy for å forstå sosiale strukturer som ikke lar seg fange med tradisjonelle regresjonsmodeller.

Part III

Del III: Statistisk modellering

9 Regresjon: Sammenheng mellom variable

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

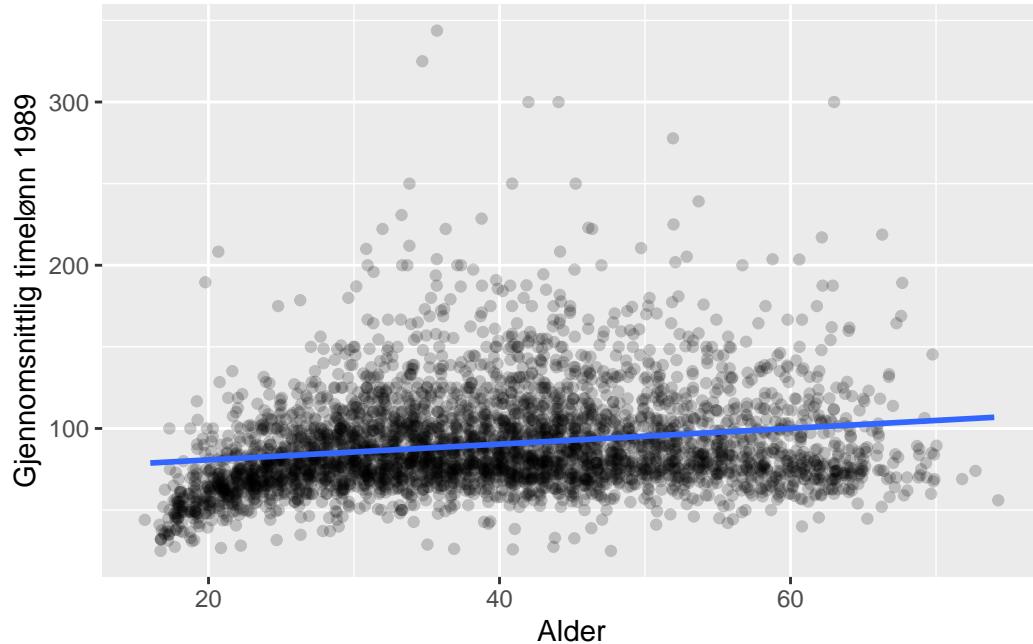
Vi skal her se på helt grunnleggende lineær regresjon med en og to forklaringsvariable.

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.

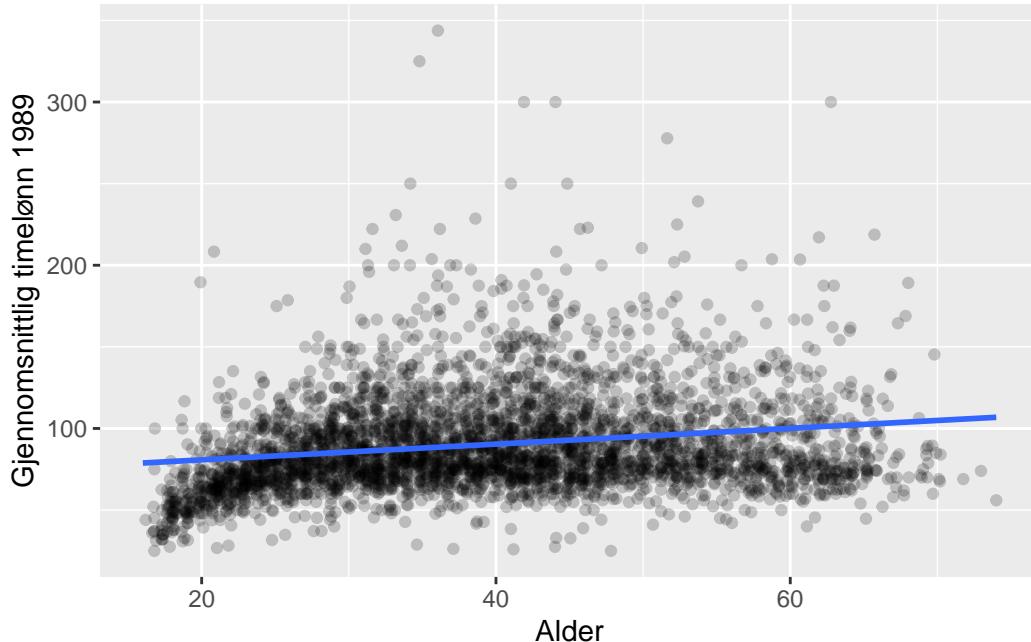
I tillegg er det tegnet inn en linje som illustrerer *trenden i gjennomsnittlig lønn* med alder. Denne linjen skrår svakt oppover, som altså betyr at gjennomsnittlig lønn øker noe med alder. Vi ser med det blotte øyet at en rett linje ikke beskriver denne sammenhengen perfekt. Først og fremst er det 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 "gjennomsnittslønna 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.

Utrekningen av regresjonslinja går vi ikke inn på her, men intuitivt sett ønsker vi jo *den beste linja* og ikke en hvilken som helst omrentlig 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. Dernest må det spesifiseres hvilket datasett som skal brukes med `data =`.²

¹Du kan jo tenke deg flere parallele linjer i plottet ovenfor med samme stigningstall

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

Legg alltid resultatene i et eget objekt med et navn 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)
```

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

³Noen 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.

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
Mann	0
Kvinne	1
Kvinne	1
Kvinne	1

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

¹Mean (SD)

Kjønn	Female
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)
```

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

Regresjonskoeffisienten 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.

```

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.

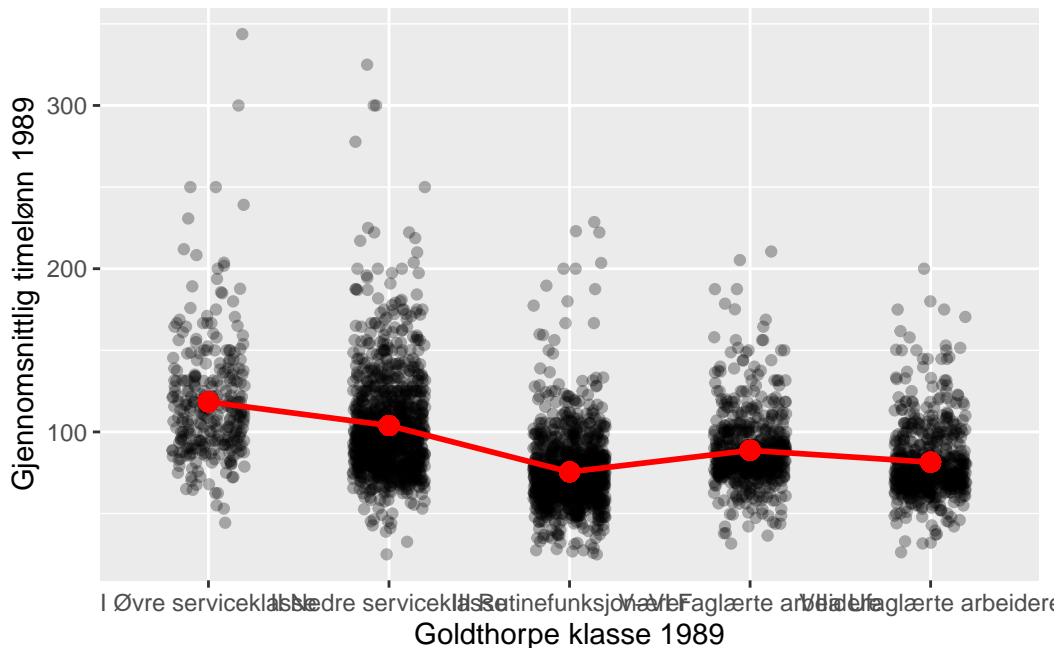
En dummy-variabel har bare to kategorier: 0 og 1, men vi kan lage flere dummy-variable. Vi kan lager en ny variabel «klasse II» som har verdien 1 hvis personen tilhører denne klassen og 0 ellers. Altså en dummy. Så kan vi lage en ny variabel «Klasse III» som har verdien 1 hvis personen tilhører denne 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 referansekatégorien. 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 estimer 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

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

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)
```

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_est2 <- lm(time89 ~ sex , data = abu89)

coef(lm_est2)
```

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

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

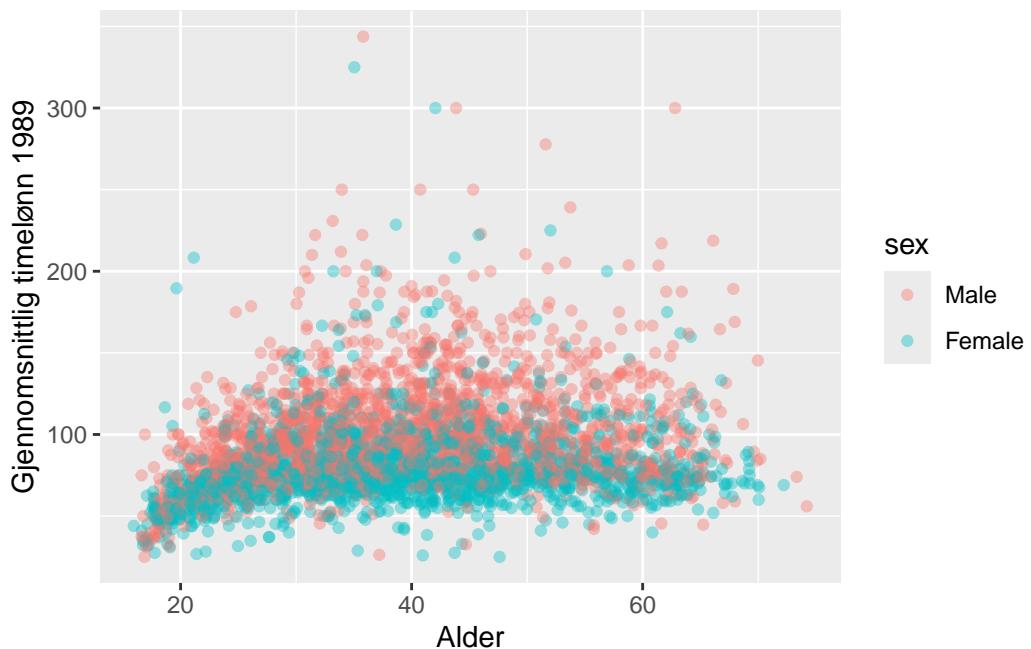
```
(Intercept)    sexFemale
99.84382     -20.75229
```

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)
```



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

coef(lm_est3)
```

```
(Intercept)    sexFemale        age
81.1014711   -20.6251051   0.4738042
```

9.4.1 Interaksjonsledd

Det kan tenkes at hvordan inntekten varierer med alder er *forskjellig* for menn og kvinner. I modellen ovenfor er det derimot antatt at denne er *lik*. Vi kan beskrive forskjellen i aldersgradierten i inntekt mellom menn og kvinner ved å inkludere interaksjonsledd (samspill) mellom kjønn og alder.

Dette gjør vi ved å spesifisere en regresjonslikning som følger:

$$time89 = \alpha + \beta_1(age) + \beta_2(sex) + \beta_3(age \times sex)$$

Eller med andre ord: timelønn varierer med *alder* og *kjønn*, og *kombinasjonen* av disse. Merk hvordan dette skrives i R:

```
lm_est4 <- lm(time89 ~ sex + age + sex * age, data = abu89)  
coef(lm_est4)
```

(Intercept)	sexFemale	age	sexFemale:age
75.8535901	-9.9048054	0.6064699	-0.2719530

Merk nå at referansekategoriene for kjønn er menn, så den estimerte koeffisienten er for kvinner. Det betyr at i utregningen får menn verdien 0 og kvinner får verdien 1. Sette man det inn i regresjonslikningen for menn blir det da som følger:

$$time89 = \alpha + \beta_1(age) + \beta_2 \times 0 + \beta_3(age \times 0)$$

Siden alt som multipliseres med 0 blir 0 kan vi også skrive dette som:

$$time89 = \alpha + \beta_1(age)$$

For kvinner skal det derimot multipliseres med 1 og vi får:

$$time89 = \alpha + \beta_1(age) + \beta_2 \times 1 + \beta_3(age \times 1)$$

Det som multipliseres med 1 endres jo ikke, så da kan det skrives som:

$$time89 = \alpha + \beta_1(age) + \beta_2 + \beta_3(age)$$

Men siden vi har to termer for alder kan det forkortes til:

$$time89 = \alpha + (\beta_1 + \beta_3)(age) + \beta_2$$

Kortere sagt: forskjellen i stigningstallet mellom menn og kvinner er β_3 . I det empiriske eksempelet over betyr det at timelønn øker med alder for begge kjønn, men for kvinner øker det med 0.27 kroner mindre per år sammenlignet med menn. Eller sagt motsatt: menns lønn øker med 0.27 mer enn kvinner per år.

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 regresjonsko-effisientene 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 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

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

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
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))
```

⁷Hvis 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.

	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.
- rtf - rikt tekstformat. Et 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 stasj
- Markdown/Quarto - for dynamiske dokumenter med integrert R-kode og tekst, og kan eksportere ferdig dokument til alle ovennevnte formater⁹ 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:

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

⁹F.eks. dette dokumentet er skrevet i Quarto

	(1)	(2)
(Intercept)	99.844 (0.637)	81.101 (1.585)
sexFemale	-20.752 (0.932)	-20.625 (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

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 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.

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

	(1)	(2)
(Intercept)	99.8 (0.6) [98.2, 101.5]	81.1 (1.6) [77.0, 85.2]
sexFemale	-20.8 (0.9) [-23.2, -18.4]	-20.6 (0.9) [-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

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 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',
```

	(1)	(2)
Konstant	99.8 (0.6) [98.2, 101.5]	81.1 (1.6) [77.0, 85.2]
Kvinne	-20.8 (0.9) [-23.2, -18.4]	-20.6 (0.9) [-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

```

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*** (0.932)	-20.625*** (0.912)
age		0.474*** (0.037)
Constant	99.844*** (0.637)	81.101*** (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 nøyne 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 regresjonsmodell 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))
)

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)
      )
)
```

Characteristic	Table 1			Table 2		
	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		

Abbreviation: CI = Confidence Interval

The number rows in the tables to be merged do not match, which may result in rows appearing out of order.

i See `tbl_merge()` (`?gtsummary::tbl_merge()`) help file for details. Use `quiet=TRUE` to silence message.

10 Utvidelser av regresjonsmodeller

```
library(tidyverse)
library(haven)
library(modelsummary)
library(lmtest)
library(sandwich)
```

I forrige kapittel så vi på helt grunnleggende lineær regresjon. Det er et godt utgangspunkt, men i praksis trenger man ofte å utvide modellene litt. I dette kapittelet ser vi på noen av de vanligste utvidelsene: ikke-lineære sammenhenger, log-transformasjoner, og robuste standardfeil. Alt dette er ting du vil møte i pensumlitteraturen og i empiriske artikler, så det er nyttig å vite hvordan det gjøres i R.

10.1 Polynomiske ledd: ikke-lineære sammenhenger

I forrige kapittel så vi at sammenhengen mellom alder og timelønn ikke nødvendigvis er helt lineær. Kanskje lønna øker raskt i starten av karrieren, flater ut, og så synker litt mot slutten? En rett linje fanger ikke opp dette. En enkel løsning er å legge til et *andregradsledd*, altså alder i andre potens (*alder²*). Da får vi en kurve i stedet for en rett linje.

I R bruker vi funksjonen `poly()` eller legger til `I(age^2)` direkte i formelen. La oss prøve begge:

```
est_lin <- lm(time89 ~ age, data = abu89)
est_poly <- lm(time89 ~ age + I(age^2), data = abu89)
```

La oss sammenligne de to modellene:

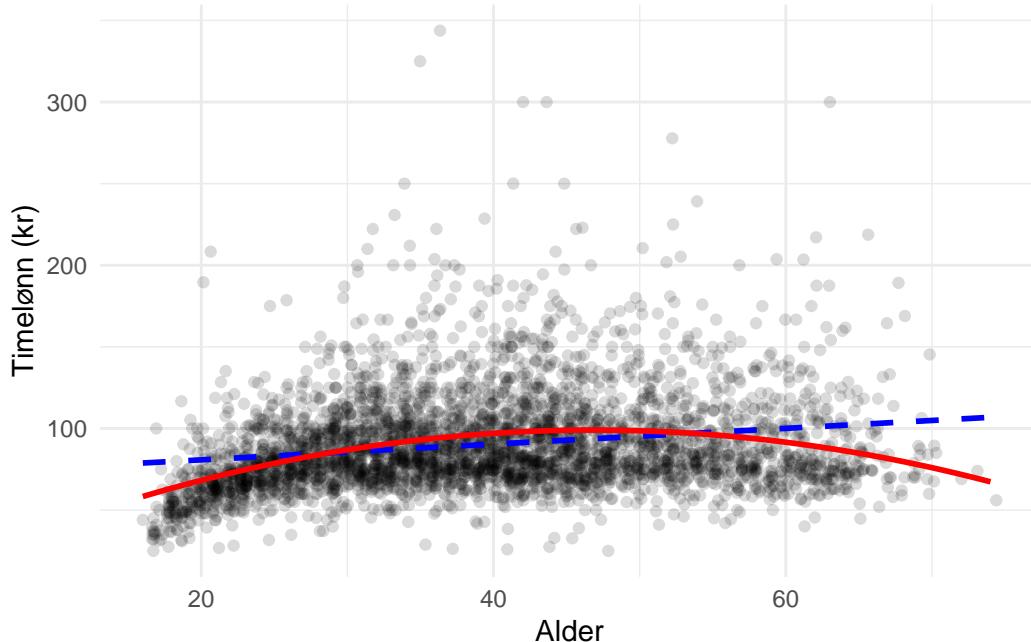
```
modelsummary(list("Lineær" = est_lin, "Kvadratisk" = est_poly),
            fmt = 2,
            gofomit = 'DF|Deviance|AIC|BIC|Log.Lik.|RMSE')
```

	Lineær	Kvadratisk
(Intercept)	71.11 (1.62)	5.11 (4.60)
age	0.48 (0.04)	4.01 (0.23)
I(age ²)		-0.04 (0.00)
Num.Obs.	3759	3759
R2	0.039	0.095
R2 Adj.	0.038	0.094
F	151.239	197.015

Koeffisienten for $I(\text{age}^2)$ er negativ, noe som betyr at kurven bøyer nedover. Altså: lønna øker med alder, men økningen avtar. Dette gir en omvendt U-form som er ganske typisk for inntekt over livsløpet.

Merk at når du har med et andregradsledd er det ikke lenger meningsfylt å tolke koeffisienten for `age` alene. De to koeffisientene må tolkes *sammen* fordi de beskriver ulike deler av den samme kurven. Den beste måten å forstå effekten på er å se på et plot:

```
ggplot(abu89, aes(x = age, y = time89)) +
  geom_jitter(alpha = .15) +
  geom_smooth(method = "lm", formula = y ~ x,
              se = FALSE, col = "blue", linetype = "dashed") +
  geom_smooth(method = "lm", formula = y ~ x + I(x^2),
              se = FALSE, col = "red") +
  labs(x = "Alder", y = "Timelønn (kr)") +
  theme_minimal()
```



Den stiplete blå linjen er den lineære modellen, mens den røde kurven er modellen med andregradsledd. Vi ser at den røde kurven fanger opp mønsteret i dataene bedre, særlig for de yngste og de eldste.

Man kan i prinsippet legge til høyere ordens ledd ($I(\text{age}^3)$ osv.), men i praksis er det sjeldent nødvendig med mer enn andregradsledd. Høyere ordens polynomer gir fort rare kurver som er vanskelige å tolke og som gjerne bare tilpasser seg støy i dataene.

10.2 Log-transformasjoner

En annen vanlig måte å håndtere ikke-lineære sammenhenger på er å bruke *logaritmen* av variablene. Dette er særlig nyttig for variable som inntekt, lønn, og andre økonomiske størrelser som ofte har en skjev fordeling med en lang hale til høyre.

10.2.1 Logaritme av utfallsvariabelen

Hvis vi tar logaritmen av utfallsvariabelen (`time89`), endrer vi tolkningen av koeffisientene. Nå uttrykkes endringer i *prosent* i stedet for i kroner:

	Kroner	Log(kroner)
(Intercept)	71.110	4.226
	(1.622)	(0.016)
age	0.483	0.006
	(0.039)	(0.000)
Num.Obs.	3759	3759
R2	0.039	0.053
R2 Adj.	0.038	0.053
F	151.239	210.357

```
est_log <- lm(log(time89) ~ age, data = abu89)

modelsummary(list("Kroner" = est_lin, "Log(kroner)" = est_log),
            fmt = 3,
            gofomit = 'DF|Deviance|AIC|BIC|Log.Lik.|RMSE')
```

I den første modellen (i kroner) betyr koeffisienten for alder at timelønna øker med ca. 0.48 kroner per år. I den andre modellen (log) betyr koeffisienten at timelønna øker med ca. 0.5% per år.¹

10.2.2 Logaritme av forklaringsvariablen

Man kan også ta logaritmen av forklaringsvariabelen. Da endres tolkningen til å handle om prosentvis endring i forklaringsvariabelen. Vi bruker et annet eksempel her fordi log av alder ikke er så meningsfylt, men log av f.eks. arbeidstid eller lignende variable kan gi mening i andre datasett.

10.2.3 Oppsummering av log-transformasjoner

Modell	Tolkning av β
$y = \alpha + \beta x$	x opp 1 enhet $\rightarrow y$ endres med β enheter
$\log(y) = \alpha + \beta x$	x opp 1 enhet $\rightarrow y$ endres med ca. $\beta \times 100$ prosent
$y = \alpha + \beta \log(x)$	x opp 1% $\rightarrow y$ endres med ca. $\beta/100$ enheter

¹Teknisk sett er tolkningen at en enhets økning i alder er assosiert med en endring i log-timelønn på 0.005. For å få prosent-tolkning multipliserer man med 100. Denne tilnærmingen fungerer godt for små koeffisienter (under ca. 0.1), men for større koeffisienter bør man bruke den eksakte formelen: $(e^\beta - 1) \times 100$.

Modell	Tolkning av β
$\log(y) = \alpha + \beta \log(x)$	x opp 1% $\rightarrow y$ endres med ca. β prosent (elastisitet)

Denne tabellen er verd å ha i bakhodet. Det siste tilfellet, der begge er log-transformert, er det man kaller en *elastisitet* i økonomi.

10.3 Robuste standardfeil

Standardfeilene fra vanlig OLS-regresjon bygger på en antakelse om at variansen i feilene er *konstant* (homoskedastisitet). Hvis denne antakelsen er brutt – noe den ofte er i praksis – kan standardfeilene bli feil. Konsekvensen er at konfidensintervaller og p-verdier ikke er til å stole på.

Løsningen er å bruke *robuste standardfeil*, gjerne kalt “Huber-White” eller “sandwich” standardfeil. Disse gir korrekte standardfeil selv om variansen ikke er konstant.²

10.3.1 Med `modelsummary`

Den enkleste måten å rapportere robuste standardfeil i R er å bruke `vcov`-argumentet i `modelsummary`. Da trenger du ikke engang å laste ekstra pakker utover det du allerede har:

```
modelsummary(list("Vanlige SE" = est_lin, "Robuste SE" = est_lin),
            vcov = list("classical", "HC1"),
            fmt = 2,
            gofomit = 'DF|Deviance|AIC|BIC|Log.Lik.|RMSE')
```

Her er det nøyaktig *samme* modell i begge kolonnene, men med ulike standardfeil. "HC1" er den vanligste varianten av robuste standardfeil og tilsvarer det Stata bruker som forvalg.³

10.3.2 Med `lmtest` og `sandwich`

Du kan også bruke pakkene `lmtest` og `sandwich` direkte for å se robuste standardfeil:

²Det som endres er altså bare standardfeilene og dermed p-verdier og konfidensintervaller. Selve koeffisientene (estimatene) er de samme.

³Det finnes flere varianter: HC0, HC1, HC2, HC3. HC1 er den mest brukte i praksis og inkluderer en korreksjon for antall observasjoner.

	Vanlige SE	Robuste SE
(Intercept)	71.11 (1.62)	71.11 (1.52)
age	0.48 (0.04)	0.48 (0.04)
Num.Obs.	3759	3759
R2	0.039	0.039
R2 Adj.	0.038	0.038
F	151.239	155.345
Std.Errors	Classical	HC1

```
coeftest(est_lin, vcov = vcovHC(est_lin, type = "HC1"))
```

t test of coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	71.11019	1.51953	46.797	< 2.2e-16 ***
age	0.48284	0.03874	12.464	< 2.2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1				

Dette gir deg koeffisientene, robuste standardfeil, t-verdier og p-verdier. Funksjonen `vcovHC` beregner den robuste varians-kovariansmatrisen, og `coeftest` bruker denne til å beregne nye teststatistikker.

10.4 Klustrede standardfeil

I noen datasett er observasjonene gruppert, f.eks. elever innenfor skoler, eller ansatte innenfor bedrifter. Da kan observasjoner innenfor samme gruppe være mer like hverandre enn observasjoner på tvers av grupper. Vanlige standardfeil tar ikke hensyn til dette og kan bli for *lave*, noe som gjør at man feilaktig finner “statistisk signifikante” resultater.

Løsningen er *klustrede standardfeil*, som tar hensyn til gruppestrukturen. I `modelsummary` kan dette gjøres med `vcov`-argumentet ved å angi en formel med klustervariabelen:

```
modelsummary(list(est_lin),
            vcov = ~klasse89,
            fmt = 2)
```

Klustrede standardfeil er særlig viktig i paneldata og flernivådata. Vi går ikke dypere inn på dette her, men det er viktig å vite at det finnes og at det er enkelt å implementere.

10.5 Vektet regresjon

Noen ganger har vi data der observasjonene har ulik *vekt*. Det kan for eksempel være at noen observasjoner er mer pålitelige enn andre, eller at datasettet er et stratifisert utvalg der noen grupper er overrepresentert. I slike tilfeller bruker vi *vektet regresjon*.

I R legger man til `weights`-argumentet i `lm`:

```
est_vektet <- lm(time89 ~ age + female, data = abu89, weights = en_vekt_variabel)
```

Vektet regresjon er vanlig i analyser av surveydata der ulike grupper har ulik trekkesannsynlighet. Mange norske datasett fra SSB kommer med slike vekter. Prinsippet er at observasjoner med høyere vekt teller mer i estimeringen.

10.6 Faste effekter (fixed effects)

Faste effekter er en teknikk som brukes mye i paneldata, altså data der man observerer de samme enhetene over flere tidspunkter. Ideen er å kontrollere for *alle* tidskonstante egenskaper ved enhetene, også de man ikke har observert. For eksempel: hvis man ser på lønnsutvikling over tid, vil faste effekter for person fjerne all variasjon som skyldes uboserte, stabile personegenskaper (motivasjon, evner, osv.).

I praksis fungerer det ved at man sammenligner *endringer* innenfor hver enhet over tid, i stedet for å sammenligne *nivåer* mellom enheter. Det gjør at tidskonstante forstyrrende variable elimineres fra analysen.

En enkel (men lite effektiv) måte å estimere faste effekter på i R er å inkludere grupperingsvariabelen som en faktorvariabel:

```
lm(time89 ~ age + factor(gruppe_id), data = paneldata)
```

Dette fungerer greit med få grupper, men er upraktisk med mange grupper (f.eks. tusenvis av personer). Da bruker man heller spesialiserte pakker som `plm` eller `fixest`. Spesielt `fixest` er blitt veldig populær fordi den er rask og håndterer mange typer faste effekter elegant:

```
library(fixest)
feols(time89 ~ age | person_id, data = paneldata)
```

Syntaksen `| person_id` angir at det skal inkluderes faste effekter for person. Man kan inkludere faste effekter for flere grupperinger samtidig ved å skille dem med `+`, f.eks. `| person_id + year`.

Vi går ikke dypere inn på faste effekter her, men dette er et helt sentralt verktøy i kvantitativ samfunnsforskning og noe du vil møte i mer avanserte kurs.⁴

10.7 Oppsummering

I dette kapittelet har vi sett på noen vanlige utvidelser av den lineære regresjonsmodellen:

- **Polynomiske ledd** lar oss fange opp kurvlineære sammenhenger ved å legge til f.eks. x^2 som forklaringsvariabel.
- **Log-transformasjoner** er nyttige for skjeve fordelinger og gir tolkninger i prosent.
- **Robuste standardfeil** korrigerer for heteroskedastisitet og bør brukes som standard i de fleste analyser.
- **Klustrede standardfeil** tar hensyn til gruppestruktur i dataene.
- **Vektet regresjon** brukes når observasjonene har ulik betydning.
- **Faste effekter** kontrollerer for uobserverte, tidskonstante egenskaper og er sentralt i paneldataanalyse.

Felles for alle disse utvidelsene er at de bygger videre på den grunnleggende lineære regresjonsmodellen. Selve R-koden er ofte bare små justeringer, men tolkningen kan endre seg en del. Det er derfor viktig å tenke gjennom hva man gjør og hvorfor, fremfor å bare kjøre kode mekanisk.

⁴For en grundig innføring i faste effekter anbefales f.eks. ([huntington2022?](#)) eller tilsvarende pensumlitteratur for kausalanalyse.

11 Modelldiagnostikk

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

Når vi estimerer en regresjonsmodell gjør vi en del antakelser om dataene. Disse antakelsene er ikke bare formaliteter – de har betydning for om vi kan stole på resultatene. Modelldiagnostikk handler om å sjekke om disse antakelsene holder rimelig godt. Heldigvis finnes det enkle grafiske verktøy og tester som gjør dette ganske oversiktlig.

Vi bruker en enkel regresjonsmodell som utgangspunkt for diagnostikken. Her predikerer vi timelønn med alder og kjønn:

```
mod1 <- lm(time89 ~ age + female, data = abu89)
summary(mod1)
```

Call:

```
lm(formula = time89 ~ age + female, 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 ***
age	0.47380	0.03684	12.86	<2e-16 ***
female	-20.62511	0.91186	-22.62	<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

11.1 Hva er det vi sjekker?

De viktigste antakelsene i lineær regresjon er:

1. **Linearitet:** Sammenhengen mellom forklaringsvariable og utfallsvariabel er tilnærmet lineær.
2. **Normalfordelte residualer:** Residualene (avvikene fra regresjonslinjen) er tilnærmet normalfordelt.
3. **Homoskedastisitet:** Variansen i residualene er omrent lik for alle verdier av forklaringsvariablene.
4. **Ingen ekstreme observasjoner** som alene driver resultatene.
5. **Ingen alvorlig multikollinearitet:** Forklaringsvariablene er ikke for sterkt korrelert med hverandre.

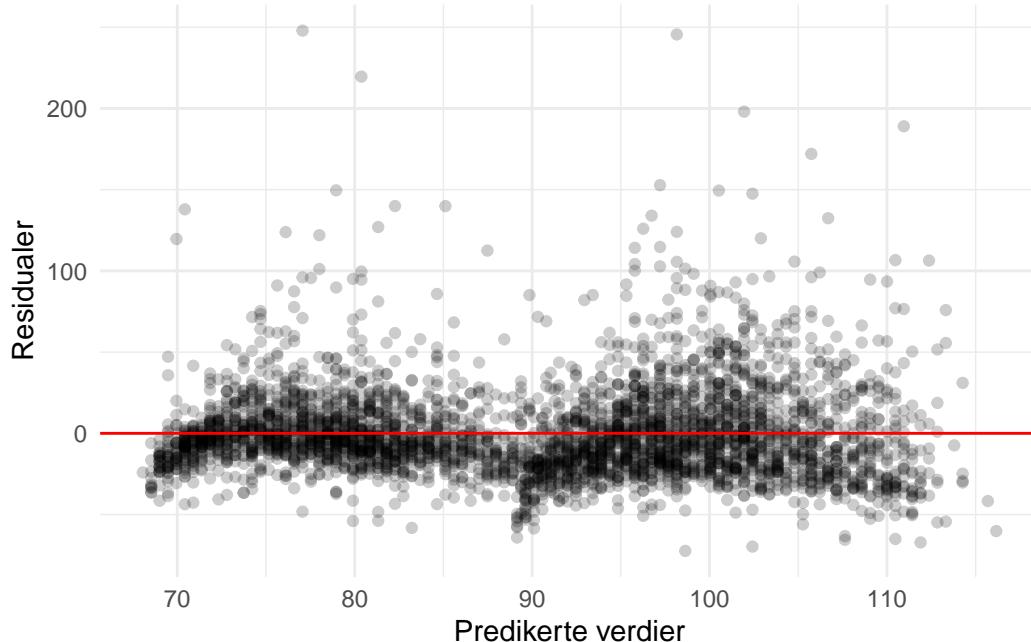
Ingen av disse trenger å holde *perfekt*. Regresjon er ganske robust, og poenget er å avdekke grove brudd som kan påvirke konklusjonene.

11.2 Residualplot: Residualer mot predikerte verdier

Det mest grunnleggende diagnostiske plottet er å sette residualene opp mot de predikerte (fitted) verdiene. Hvis alt er som det skal, vil punktene ligge tilfeldig spredt rundt null uten noe tydelig mønster.

```
abu89 <- abu89 %>%
  mutate(fitted_val = fitted(mod1),
        resid_val = residuals(mod1))

ggplot(abu89, aes(x = fitted_val, y = resid_val)) +
  geom_point(alpha = .2) +
  geom_hline(yintercept = 0, col = "red") +
  labs(x = "Predikerte verdier", y = "Residualer") +
  theme_minimal()
```

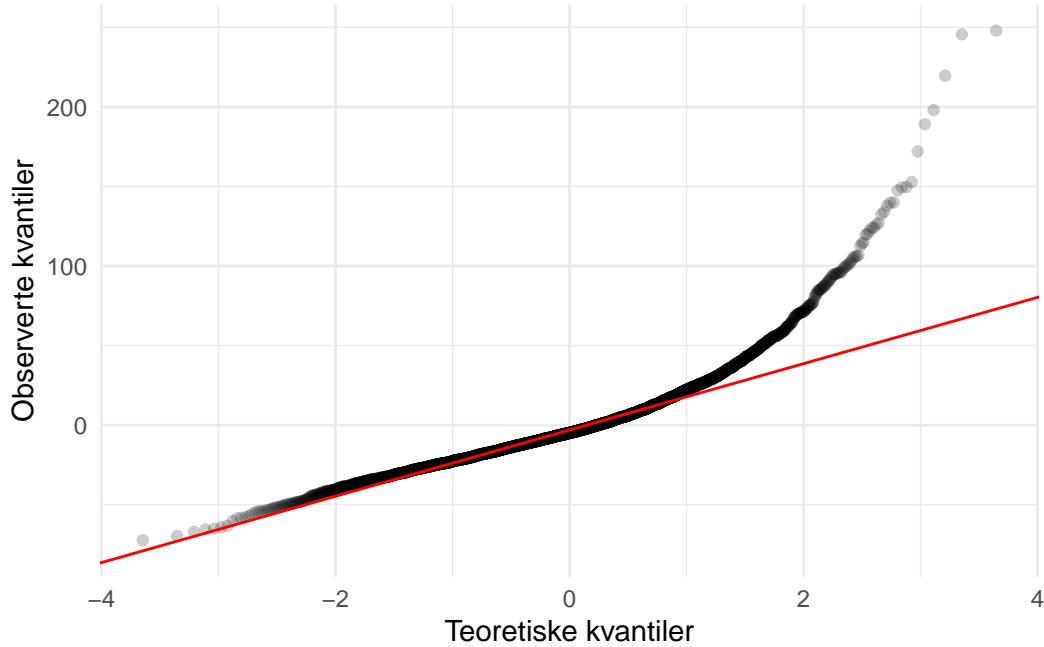


Her ser vi etter systematiske mønstre. Hvis punktene danner en trakt (vifteform), tyder det på heteroskedastisitet. Hvis det er en kurve, kan det tyde på at sammenhengen ikke er lineær. I dette tilfellet ser vi at variansen ser ut til å øke noe med predikerte verdier, og det er noen observasjoner med veldig store residualer.

11.3 QQ-plot for normalitet

Et QQ-plot (quantile-quantile plot) sammenligner fordelingen av residualene med en teoretisk normalfordeling. Hvis residualene er normalfordelt, vil punktene ligge omtrent langs en rett linje.

```
ggplot(abu89, aes(sample = resid_val)) +
  stat_qq(alpha = .2) +
  stat_qq_line(col = "red") +
  labs(x = "Teoretiske kvantiler", y = "Observerte kvantiler") +
  theme_minimal()
```

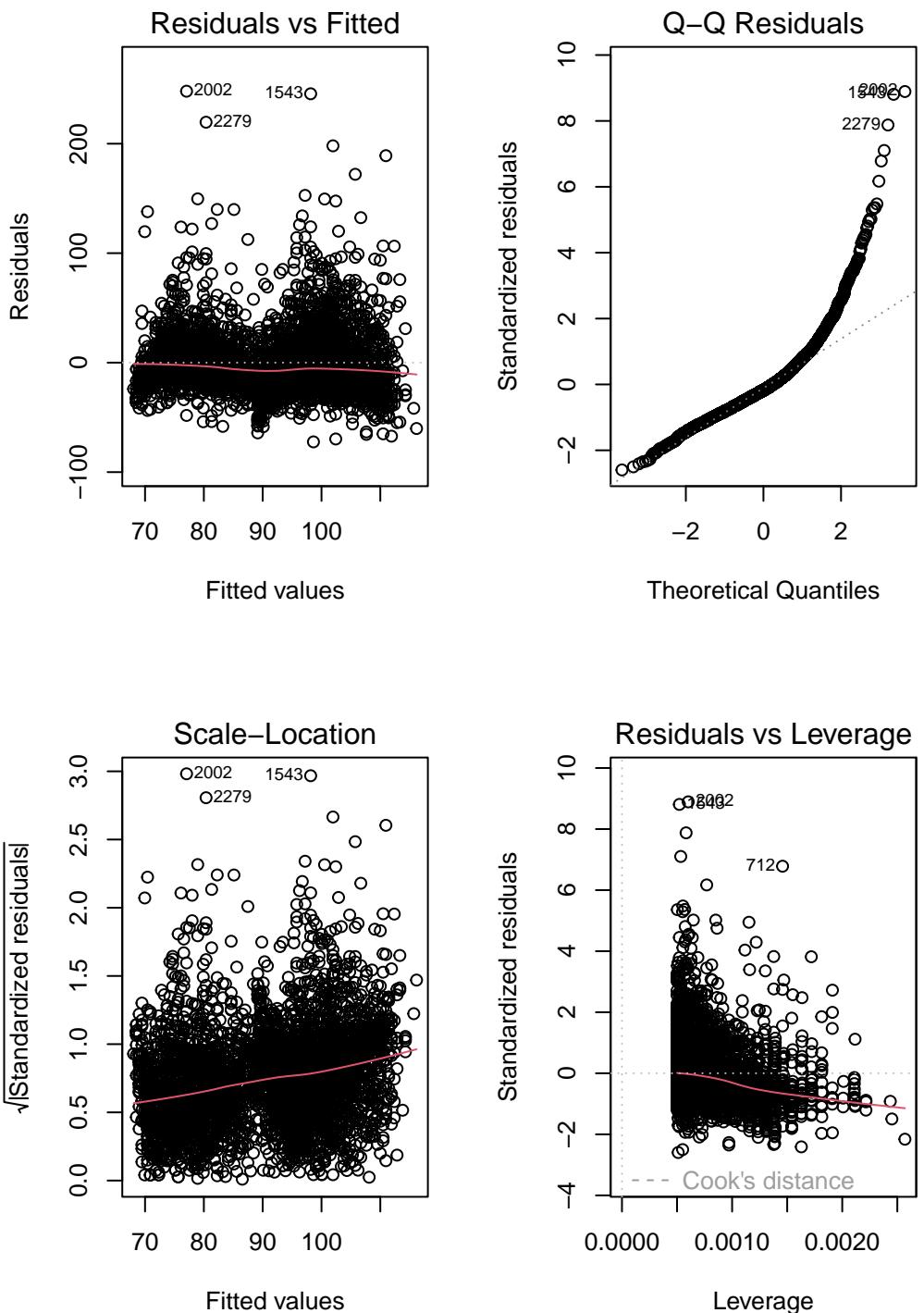


Her ser vi at punktene følger linjen rimelig bra i midten, men avviker i halene. Det betyr at fordelingen har tyngre haler enn normalfordelingen. Med store utvalg er dette sjeldent et stort problem for estimatene, men det er greit å vite om.

11.4 De fire standard diagnostikkplottene

R har en innebygd funksjon som gir fire diagnostiske plot samtidig. Man bruker bare `plot()` på et `lm`-objekt:

```
par(mfrow = c(2, 2))
plot(mod1)
```



```
par(mfrow = c(1, 1))
```

Disse fire plottene viser:

1. **Residuals vs Fitted:** Sjekker linearitet og homoskedastisitet (det vi allerede har sett på).
2. **Normal Q-Q:** Sjekker normalitet i residualene.
3. **Scale-Location:** Sjekker homoskedastisitet. Linjen bør være tilnærmet flat.
4. **Residuals vs Leverage:** Identifiserer innflytelsesrike observasjoner.

Legg merke til at R merker noen observasjoner med radnummer. Det er de mest avvikende observasjonene som du kanskje bør se nærmere på.

11.5 Heteroskedastisitet

Heteroskedastisitet betyr at variansen i residualene varierer systematisk. Det påvirker ikke selve estimatene, men det gjør at standardfeilene kan bli feil – og dermed også p-verdier og konfidensintervaller.

Vi har allerede sett visuelt etter dette i residualplottet. Man kan også bruke en formell test. Breusch-Pagan-testen sjekker om variansen i residualene korrelerer med de predikerte verdiene:

```
library(lmtest)
bptest(mod1)
```

```
studentized Breusch-Pagan test

data: mod1
BP = 30.972, df = 2, p-value = 1.882e-07
```

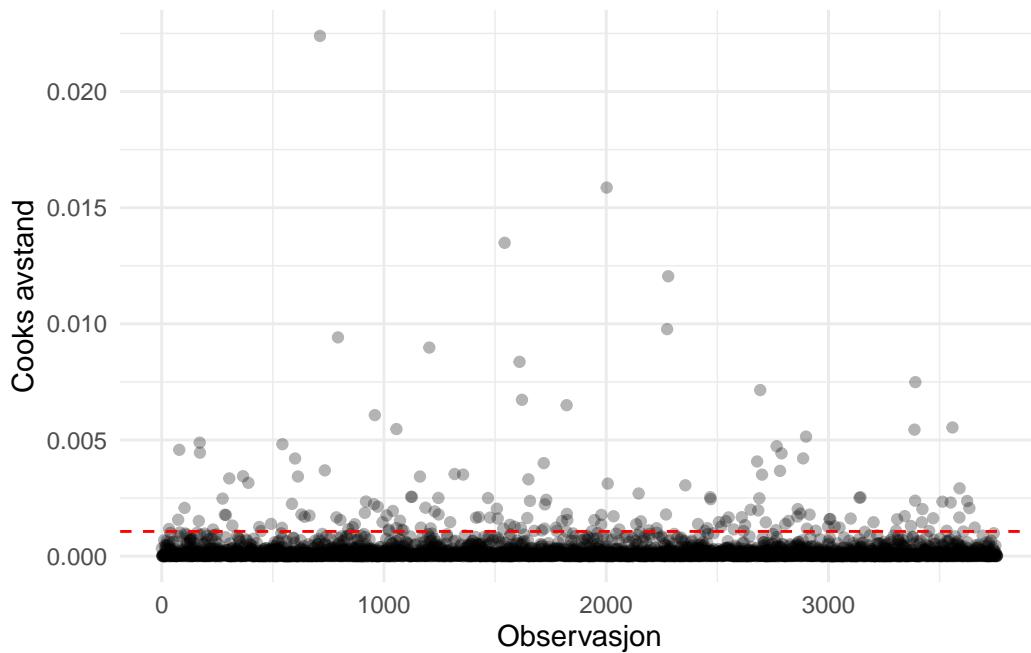
En lav p-verdi (under 0.05) tyder på at heteroskedastisitet er til stede. Men husk at med store utvalg vil selv små avvik bli statistisk signifikante, så det visuelle inntrykket fra residualplottet er vel så viktig.

11.6 Innflytelsesrike observasjoner og Cooks avstand

Noen enkeltobservasjoner kan ha uforholdsmessig stor innflytelse på regresjonsresultatene. Cooks avstand (Cook's distance) er et samlemaål for hvor mye hvert datapunkt påvirker de estimerte koeffisientene. En tommelfingerregel er at verdier over $4/n$ (der n er antall observasjoner) bør undersøkes nærmere.

```
abu89 <- abu89 %>%
  mutate(cooks_d = cooks.distance(mod1))

ggplot(abu89, aes(x = seq_along(cooks_d), y = cooks_d)) +
  geom_point(alpha = .3) +
  geom_hline(yintercept = 4 / nrow(abu89), col = "red", linetype = "dashed") +
  labs(x = "Observasjon", y = "Cooks avstand") +
  theme_minimal()
```



Punktene over den røde linjen er observasjoner som har relativt stor innflytelse. Det betyr ikke nødvendigvis at de er feil eller at de bør fjernes, men det er lurt å sjekke hva slags observasjoner det er. I lønnsdata er det gjerne folk med uvanlig høy inntekt som trekker resultatene.

11.7 Multikollinearitet og VIF

Multikollinearitet oppstår når forklaringsvariablene er sterkt korrelert med hverandre. Det gjør at det blir vanskelig å skille effektene fra hverandre, og standardfeilene blåses opp.

VIF (Variance Inflation Factor) mäter dette. En VIF på 1 betyr ingen korrelasjon med andre forklaringsvariable. Tommelfingerregler varierer, men VIF over 5 gir grunn til bekymring, og over 10 er et klart problem.

La oss utvide modellen med noen flere variable og sjekke VIF:

```
mod2 <- lm(time89 ~ age + female + ed, data = abu89)
```

```
library(car)  
vif(mod2)
```

```
age    female      ed  
1.004787 1.015293 1.019662
```

Her ser vi at VIF-verdiene er lave, noe som betyr at multikollinearitet ikke er et problem i denne modellen. Hvis du hadde inkludert variable som i praksis mäter det samme, ville VIF blitt høyere.

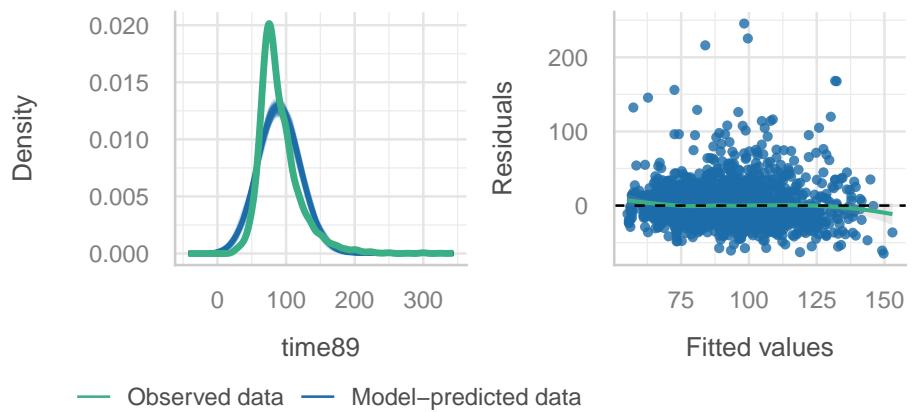
11.8 Rask diagnostikk med performance-pakken

Pakken {performance} har en veldig nyttig funksjon som gjør mye av diagnostikken i ett steg. Funksjonen `check_model()` lager et sett med diagnostiske plot som dekker de viktigste antakelsene.

```
library(performance)  
check_model(mod2)
```

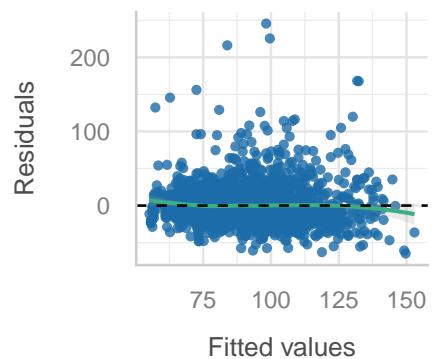
Posterior Predictive Check

Model-predicted lines should resemble observed data lines.



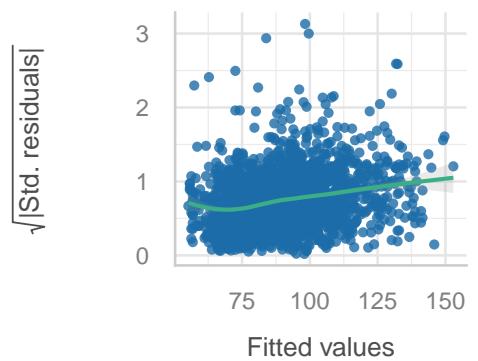
Linearity

Residuals should be flat and horizontal.



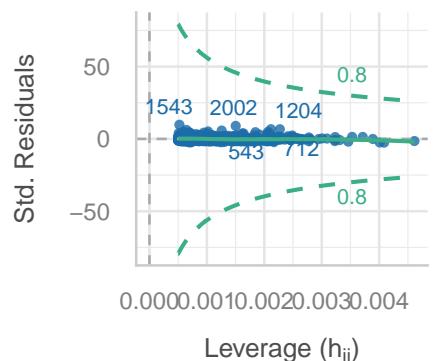
Homogeneity of Variance

Reference line should be flat and horizontal.



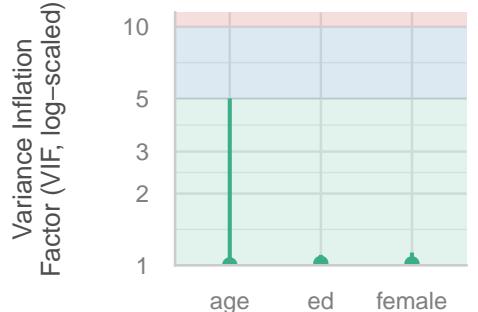
Influential Observations

Points should be inside the contour lines.



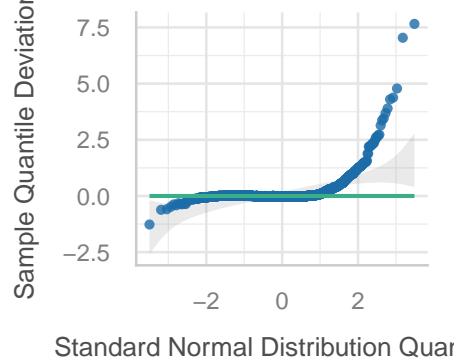
Collinearity

High collinearity (VIF) may inflate parameter estimates.



Normality of Residuals

Residuals should fall along the line.



Dette gir deg et samlet bilde av modellens egenskaper, inkludert linearitet, homoskedastisitet, normalitet, innflytelsesrike observasjoner og multikollinearitet. Det er en fin snarvei for en rask sjekk.

Du kan også få en tekstbasert oppsummering:

```
check_model(mod2, check = "all") |> summary()
```

	Length	Class	Mode
VIF	3	check_collinearity	list
QQ	2	data.frame	list
NORM	3	data.frame	list
NCV	2	data.frame	list
HOMOGENEITY	2	data.frame	list
OUTLIERS	3759	check_outliers	logical
INFLUENTIAL	7	data.frame	list
PP_CHECK	51	performance_pp_check	list

11.9 Hva gjør man hvis antakelsene brytes?

I praksis er det sjeldent at alle antakelser holder perfekt. Her er noen praktiske råd:

Heteroskedastisitet: Bruk robuste standardfeil. Det endrer ikke estimatene, men gir korrekte standardfeil og p-verdier. I R kan du bruke pakken `{sandwich}` sammen med `{lmtest}`, eller rapportere robuste standardfeil direkte via `{modelsummary}`.

Ikke-normalfordelte residualer: Med store utvalg (over noen hundre observasjoner) er dette sjeldent et problem for estimatene eller standardfeilene takket være sentralgrenseteoremet. Men det kan tyde på at modellen mangler noe viktig.

Innflytelsesrike observasjoner: Sjekk hva slags observasjoner det er. Kjør modellen med og uten disse observasjonene. Hvis resultatene endrer seg mye, bør du diskutere dette.

Ikke-linearitet: Vurder å transformere variablene (f.eks. logaritme) eller inkludere polynomiske ledd.

Multikollinearitet: Vurder om du trenger alle variablene i modellen. Kanskje noen av dem mäter det samme og du kan droppe en av dem.

Det viktigste er at du *faktisk sjekker* antakelsene. Det er mye bedre å gjøre en enkel sjekk og diskutere eventuelle problemer enn å ignorere diagnostikk fullstendig. Og husk: ingen modell er perfekt. Poenget er at den er nyttig nok til å fortelle oss noe meningsfullt om dataene.

12 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).

12.1 Dummy som utfallsvariabel

I samfunnsvitenskapen er utfallsvariablene 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 utfallsvariablene 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 utfallsvariablene 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:

$$\bar{y} = \frac{(0+0+1+1+1)}{5} = \frac{3}{5} = 0.6 \text{ som er det samme som } 60\%.$$

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 estimerer 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. Variablen "private" er en factor-variabel der første kategori er "public", som da blir referansekategori. 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.

13 Kontrollere for bakenforliggende variable

```
library(tidyverse)
library(haven)
library(modelsummary)
```

I forrige kapittel så vi hvordan vi kan estimere regresjonsmodeller med en eller flere forklaringsvariable. Men *hvorfor* vil vi ha flere variable i modellen? Det handler om noe av det mest sentrale i kvantitativ samfunnsvitenskap: å kontrollere for bakenforliggende variable. I dette kapittelet skal vi se på hva det betyr, hvorfor det er viktig, og hvordan vi gjør det i praksis.

13.1 Simpsons paradoks: Når helheten lyver

La oss starte med et tankeksperiment. Tenk deg at du sammenligner to sykehus for å finne ut hvilket som er best. Sykehus A har høyere dødelighet enn sykehus B totalt sett. Men når du ser på lette og alvorlige tilfeller *hver for seg*, er sykehus A faktisk bedre i *begge* gruppene. Hvordan er det mulig?

Forklaringen er at sykehus A er et spesialistsykehus som tar imot de mest alvorlige pasientene. Alvorlige tilfeller har naturligvis høyere dødelighet uansett sykehus. Når vi *kontrollerer for* alvorligetsgrad – altså sammenligner likt med likt – snur bildet seg.

Dette fenomenet kalles *Simpsons paradoks*: en sammenheng som går i en retning totalt sett kan snu når man deler opp etter en tredje variabel. Det er ikke bare et teoretisk kuriosum. Det skjer hele tiden i samfunnsvitenskapelige data, og det er en av hovedgrunnene til at vi bruker multippel regresjon.

13.2 Hva betyr det å “kontrollere for” en variabel?

Uttrykket “kontrollere for” er noe man hører hele tiden i metodeundervisning. Men hva betyr det egentlig?

Kort sagt betyr det at vi sammenligner grupper som er *like* på den variablen vi kontrollerer for, men *forskjellige* på den variablen vi er interessert i. Hvis vi kontrollerer for sosial klasse i

en analyse av kjønnsforskjeller i lønn, betyr det at vi sammenligner menn og kvinner *innenfor samme klasse*. Vi sammenligner altså likt med likt, så godt det lar seg gjøre.

I praksis gjør multippell regresjon dette for oss matematisk: når vi legger til en variabel i modellen, “renser” vi sammenhengen mellom de andre variablene og utfallet for den variabelen vi la til. Det er dette vi mener med å kontrollere for.

13.3 Praktisk eksempel: Kjønnsgapet i lønn

La oss se på et konkret eksempel med abu89-datasettet. Vi starter med den enkle sammenhengen mellom kjønn og timelønn.

```
mod1 <- lm(time89 ~ female, data = abu89)
coef(mod1)
```

	female
(Intercept)	99.84382
female	-20.75229

Koeffisienten for `female` viser forskjellen i gjennomsnittlig timelønn mellom kvinner og menn. Kvinner tjente altså betydelig mindre enn menn. Men er hele denne forskjellen fordi arbeidsgivere betaler kvinner mindre? Eller kan noe av forskjellen skyldes at menn og kvinner i gjennomsnitt har forskjellige yrker, utdanninger og klasseposisjoner?

La oss legge til sosial klasse i modellen:

```
mod2 <- lm(time89 ~ female + klasse89, data = abu89)
coef(mod2)
```

	female
(Intercept)	122.63676
klasse89II Nedre serviceklasse	klasse89III Rutinefunksjonærer
-10.39630	-32.70807
klasse89V-VI Faglærte arbeidere	klasse89VIIa Ufaglærte arbeidere
-32.60689	-34.03860

Legg merke til hva som skjedde med koeffisienten for `female`. Den ble mindre (i absoluttverdi). Det betyr at *noe* av lønnsforskjellen mellom menn og kvinner kan tilskrives at de befinner seg i ulike klasseposisjoner. Når vi sammenligner menn og kvinner *i samme klasse*, er forskjellen mindre.

Vi kan utvide enda mer og legge til utdanning:

```
mod3 <- lm(time89 ~ female + klasse89 + ed, data = abu89)
coef(mod3)
```

(Intercept)	female
104.807681	-17.197749
klasse89II Nedre serviceklasse	klasse89III Rutinefunksjonærer
-4.945598	-20.364392
klasse89V-VI Faglærte arbeidere	klasse89VIIa Ufaglærte arbeidere
-19.696567	-19.227326
ed	
2.812244	

Også her kan vi se at koeffisienten for `female` endrer seg noe når vi legger til flere kontrollvariable.

13.4 Bivariat vs. multippel regresjon

I en *bivariat* regresjon (kun en forklaringsvariabel) beskriver koeffisienten den *totale* sammenhengen mellom variablene og utfallet. Denne sammenhengen kan inneholde både en direkte effekt og indirekte effekter via andre variable.

I en *multippel* regresjon (flere forklaringsvariable) beskriver koeffisienten sammenhengen mellom variablene og utfallet *gitt at de andre variablene holdes konstant*. Det er dette “kontrollere for” betyr i praksis.

Forskjellen kan oppsummeres slik:

- **Bivariat:** Hva er den gjennomsnittlige lønnsforskjellen mellom menn og kvinner?
- **Multippel:** Hva er den gjennomsnittlige lønnsforskjellen mellom menn og kvinner *som er i samme sosiale klasse og har samme utdanningsnivå*?

Det er viktig å forstå at begge spørsmålene kan være relevante. Det avhenger av hva du er interessert i.

13.5 Sammenligne modeller med `modelsummary()`

For å virkelig se hva som skjer når vi legger til kontrollvariable er det nyttig å sette modellene ved siden av hverandre i en tabell. Da kan vi direkte se hvordan koeffisienten for `female` endres.

	Bivariat	Med klasse	Med klasse og utd.
Konstant	99.8 (0.6)	122.6 (1.5)	104.8 (2.0)
Kvinne	-20.8 (0.9)	-18.2 (1.0)	-17.2 (1.0)
klasse89II Nedre serviceklasse		-10.4 (1.7)	-4.9 (1.7)
klasse89III Rutinefunksjonærer		-32.7 (1.8)	-20.4 (2.0)
klasse89V-VI Faglærte arbeidere		-32.6 (1.8)	-19.7 (2.0)
klasse89VIIa Ufaglærte arbeidere		-34.0 (1.8)	-19.2 (2.1)
ed			2.8 (0.2)
Num.Obs.	3759	3680	3680
R2	0.117	0.280	0.312
F	496.278		

```
modelsummary(list("Bivariat" = mod1,
                  "Med klasse" = mod2,
                  "Med klasse og utd." = mod3),
            fmt = 1,
            estimate = "{estimate} ({std.error})",
            statistic = NULL,
            gofomit = 'DF|Deviance|R2 Adj.|AIC|BIC|Log.Lik.|RMSE',
            coef_rename = c("female" = "Kvinne",
                           "educ" = "Utdanning (år)",
                           "(Intercept)" = "Konstant"))
```

I den bivariate modellen (modell 1) ser vi den totale lønnsforskjellen mellom menn og kvinner. I modell 2 har vi kontrollert for klasse, og forskjellen er noe redusert. I modell 3 har vi i tillegg kontrollert for utdanning.

Legg merke til to ting: For det første, koeffisienten for `female` endres mellom modellene. For det andre, R^2 (forklart varians) øker når vi legger til flere variable. Det betyr at modellen passer bedre til dataene. Men det betyr *ikke* nødvendigvis at modellen er bedre for det vi er interessert i.

13.6 Når skal vi kontrollere – og når skal vi la være?

Det er fristende å tenke at man bør kontrollere for *alt* man har tilgang til. Men det er ikke riktig. Hvorvidt du bør kontrollere for en variabel avhenger av hva rollen til variablene er i den sammenhengen du studerer.

En nyttig tommelfingerregel er at du bør kontrollere for variable som:

1. Påvirker *både* forklaringsvariabelen og utfallsvariabelen (konfunderende variable)
2. Ikke selv er et *resultat* av forklaringsvariabelen

Punkt 2 er viktig og leder oss til det som kalles *bad controls*.

13.7 “Bad controls”: Ikke kontroller for konsekvenser

En vanlig feil er å kontrollere for variable som er *konsekvenser* av den forklaringsvariabelen du er interessert i. Tenk deg at du studerer effekten av utdanning på inntekt. Hvis du kontrollerer for *yrke*, kan du faktisk fjerne en viktig del av effekten – fordi utdanning påvirker inntekt delvis gjennom hvilket yrke man får. Ved å kontrollere for yrke fjerner du denne mekanismen fra estimatet.

Tilsvarende: Hvis du er interessert i effekten av kjønn på lønn, bør du tenke nøye gjennom om du skal kontrollere for variabler som arbeidstid eller stillingsnivå. Hvis kvinner systematisk styres mot lavere stillinger *på grunn av* kjønn, er stillingsnivå en *konsekvens* av kjønn – og da fjerner du en del av den faktiske kjønnseffekten ved å kontrollere for det.

Det finnes ingen enkel mekanisk regel for dette. Du må tenke gjennom *rekkefølgen* av variablene: Hva kommer først? Hva påvirker hva? Et nyttig verktøy for å tenke gjennom dette er såkalte DAGs (Directed Acyclic Graphs), men det går vi ikke inn på her.¹

13.8 Steg-for-steg: Bygge modeller i R

I praksis vil du ofte bygge opp modeller steg for steg. Her er en oppsummering av fremgangsmåten:

Steg 1: Start med den bivariate sammenhengen du er interessert i.

```
mod1 <- lm(time89 ~ female, data = abu89)
```

Steg 2: Legg til variable du mener er konfunderende (bakenforliggende).

```
mod2 <- lm(time89 ~ female + klasse89, data = abu89)
```

Steg 3: Legg eventuelt til ytterligere kontrollvariable.

¹DAGs er et tema som dekkes grundigere i videregående metodekurs.

```
mod3 <- lm(time89 ~ female + klasse89 + ed, data = abu89)
```

Steg 4: Sammenlign modellene i en tabell.

```
modelsummary(list("Modell 1" = mod1,
                  "Modell 2" = mod2,
                  "Modell 3" = mod3),
            fmt = 1,
            estimate = "{estimate} ({std.error})",
            statistic = NULL,
            gofomit = 'DF|Deviance|R2 Adj.|AIC|BIC|Log.Lik.|RMSE')
```

Det viktige er ikke bare *tallene*, men *hvordan* koeffisienten du er interessert i endrer seg fra modell til modell. Hvis den endres mye, tyder det på at de variablene du la til er viktige konfunderende variable. Hvis den knapt endrer seg, betyr det at disse variablene ikke forklarer mye av den opprinnelige sammenhengen.

13.9 Oppsummering

- Simpsons paradoks viser at sammenhenger kan snu når man kontrollerer for en tredje variabel.
- Å “kontrollere for” betyr å sammenligne grupper som er like på kontrollvariabelen.
- Multippel regresjon gjør dette for oss: koeffisienten uttrykker sammenhengen *gitt at de andre variablene holdes konstant*.
- Du bør kontrollere for variable som påvirker både forklaringsvariabel og utfall (konfunderende variable).
- Du bør *ikke* kontrollere for variable som er konsekvenser av forklaringsvariabelen (“bad controls”).
- Bygg modeller steg for steg og sammenlign med `modelsummary()` for å se hva kontrollvariablene gjør med estimatene.

14 Logistisk regresjon

```
library(tidyverse)
library(haven)
library(modelsummary)
```

I forrige kapittel om den lineære sannsynlighetsmodellen så vi at man kan bruke vanlig lineær regresjon selv om utfallsvariabelen er binær (0/1). Det fungerer greit i mange sammenhenger, men har noen begrensninger. Den viktigste er at modellen kan gi predikerte sannsynligheter som er under 0 eller over 1, noe som jo ikke gir mening. Logistisk regresjon løser dette problemet.

Logistisk regresjon er den vanligste metoden når utfallsvariabelen er binær, altså har to verdier. I samfunnsvitenskapen brukes det veldig mye: er personen i jobb eller ikke? Stemte personen ved valget eller ikke? Har personen høy eller lav inntekt? Osv.

14.1 Lage en binær utfallsvariabel

Vi skal bruke et eksempel der vi prøver å predikere hvem som har høy timelønn. Vi lager en ny variabel `hoylønn` som er 1 hvis timelønna er over medianen og 0 ellers.

```
abu89 <- abu89 %>%
  mutate(hoylønn = ifelse(time89 > median(time89, na.rm = TRUE), 1, 0))
```

La oss sjekke fordelingen:

```
table(abu89$hoylønn)
```

0	1
1901	1858

Omtrent halvparten har høy lønn, noe som gir mening siden vi delte ved medianen.

14.2 Hvorfor ikke bare bruke lineær regresjon?

Som vi så i kapittelet om [lineær sannsynlighetsmodell](#), kan vi bruke vanlig `lm()` også med binær utfallsvariabel. Koeffisientene tolkes da som endring i sannsynlighet (andel). Problemet oppstår spesielt når vi predikerer: vi kan få verdier under 0 eller over 1. Logistisk regresjon sørger for at predikerte sannsynligheter alltid ligger mellom 0 og 1.

14.3 Logit-transformasjonen og odds

Ideen bak logistisk regresjon er å modellere *log-oddsen* i stedet for sannsynligheten direkte. Men hva er egentlig odds?

Tenk deg at 80 av 100 personer i en gruppe er i jobb. Da er sannsynligheten 0.80 (80%). Odds er forholdet mellom sannsynligheten for å være i jobb og sannsynligheten for å *ikke* være i jobb: $0.80/0.20 = 4$. Altså: det er fire ganger så sannsynlig å være i jobb som å ikke være det.

Logistisk regresjon modellerer logaritmen av odds (log-odds). Fordelen er at log-odds kan variere fritt fra minus uendelig til pluss uendelig, mens sannsynligheter er begrenset mellom 0 og 1. Matematisk sett gjør dette at modellen alltid gir gyldige sannsynligheter.

Du trenger ikke bekymre deg så mye om den matematiske detaljen her. Det viktigste å huske er:

- Koeffisientene fra logistisk regresjon er på log-odds-skalaen
- Man kan konvertere til odds-ratio ved å eksponentiere: `exp(koeffisient)`
- Man kan regne om til sannsynligheter med `predict(..., type = "response")`

14.4 Estimere logistisk regresjon i R

I R bruker vi `glm()` med argumentet `family = binomial` for å estimere logistisk regresjon. Syntaksen er ellers helt lik `lm()`.

La oss starte enkelt med kjønn som forklaringsvariabel:

```
logit1 <- glm(hoylenn ~ female, data = abu89, family = binomial)
summary(logit1)
```

```

Call:
glm(formula = hoylenn ~ female, family = binomial, data = abu89)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.66334   0.04717 14.06   <2e-16 ***
female      -1.48581   0.07007 -21.20   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 5210.6 on 3758 degrees of freedom
Residual deviance: 4728.7 on 3757 degrees of freedom
(368 observations deleted due to missingness)
AIC: 4732.7

Number of Fisher Scoring iterations: 4

```

Koeffisienten for `female` er negativ, som betyr at kvinner har lavere log-odds for å ha høy lønn sammenlignet med menn. Men log-odds er ikke veldig intuitivt, så vi konverterer til odds-ratio.

14.5 Tolke koeffisientene: odds-ratio

For å gjøre koeffisientene mer tolkbare konverterer vi fra log-odds til odds-ratio med `exp()`:

```
exp(coef(logit1))
```

```
(Intercept)      female
1.9412628    0.2263188
```

En odds-ratio på 1 betyr ingen forskjell. Over 1 betyr høyere odds, under 1 betyr lavere odds. Koeffisienten for `female` gir en odds-ratio under 1, som altså betyr at kvinner har lavere odds for høy lønn enn menn.

Man kan også få ut konfidensintervaller for odds-ratioene:

```
exp(confint(logit1))
```

	2.5 %	97.5 %
(Intercept)	1.7706203	2.130296
female	0.1971601	0.259492

14.6 Flere forklaringsvariable

La oss utvide modellen med klasse og alder i tillegg til kjønn:

```
logit2 <- glm(hoylenn ~ female + klasse89 + age, data = abu89, family = binomial)
summary(logit2)
```

Call:

```
glm(formula = hoylenn ~ female + klasse89 + age, family = binomial,
     data = abu89)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	1.806579	0.239652	7.538	4.76e-14 ***
female	-1.618252	0.093501	-17.307	< 2e-16 ***
klasse89II Nedre serviceklasse	-0.926952	0.217423	-4.263	2.01e-05 ***
klasse89III Rutinefunksjonærer	-2.751740	0.217219	-12.668	< 2e-16 ***
klasse89V-VI Faglærte arbeidere	-2.624620	0.224599	-11.686	< 2e-16 ***
klasse89VIIa Ufaglærte arbeidere	-2.990663	0.225225	-13.279	< 2e-16 ***
age	0.025410	0.003199	7.944	1.96e-15 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 5100.9 on 3679 degrees of freedom

Residual deviance: 3886.2 on 3673 degrees of freedom

(447 observations deleted due to missingness)

AIC: 3900.2

Number of Fisher Scoring iterations: 5

Tolkningen av koeffisientene er litt annerledes enn i lineær regresjon. Hver koeffisient viser endring i log-odds for utfallet (høy lønn) per enhets endring i forklaringsvariabelen, kontrollert for de andre variablene. La oss se på odds-ratioene:

```
exp(coef(logit2))
```

(Intercept)	female
6.08958125	0.19824496
klasse89II Nedre serviceklasse	klasse89III Rutinefunksjonærer
0.39575797	0.06381675
klasse89V-VI Faglærte arbeidere	klasse89VIIa Ufaglærte arbeidere
0.07246732	0.05025410
age	
1.02573524	

Odds-ratioene tolkes slik: en odds-ratio for `age` på f.eks. 1.02 ville bety at for hvert års økning i alder øker oddsen for høy lønn med 2%, kontrollert for kjønn og klasse.

14.7 Predikere sannsynligheter

En veldig nyttig ting med logistisk regresjon er at vi kan predikere sannsynligheter. Vi bruker `predict()` med `type = "response"` for å få sannsynligheter i stedet for log-odds:

```
nyedata <- expand.grid(
  female = c(0, 1),
  klasse89 = levels(abu89$klasse89)[1:2],
  age = c(30, 40, 50)
)

nyedata$pred_sannsynlighet <- predict(logit2, newdata = nyedata, type = "response")
nyedata
```

	female	klasse89	age	pred_sannsynlighet
1	0	I Øvre serviceklasse	30	0.9288310
2	1	I Øvre serviceklasse	30	0.7212393
3	0	II Nedre serviceklasse	30	0.8377956
4	1	II Nedre serviceklasse	30	0.5059160
5	0	I Øvre serviceklasse	40	0.9439043
6	1	I Øvre serviceklasse	40	0.7693623
7	0	II Nedre serviceklasse	40	0.8694397

	Log-odds	Log-odds
(Intercept)	0.663 (0.047)	1.807 (0.240)
female	-1.486 (0.070)	-1.618 (0.094)
klasse89II Nedre serviceklasse		-0.927 (0.217)
klasse89III Rutinefunksjonærer		-2.752 (0.217)
klasse89V-VI Faglærte arbeidere		-2.625 (0.225)
klasse89VIIa Ufaglærte arbeidere		-2.991 (0.225)
age		0.025 (0.003)
Num.Obs.	3759	3680

```

8      1 II Nedre serviceklasse 40      0.5689974
9      0 I Øvre serviceklasse 50      0.9559366
10     1 I Øvre serviceklasse 50      0.8113507
11     0 II Nedre serviceklasse 50      0.8956791
12     1 II Nedre serviceklasse 50      0.6299164

```

Nå har vi estimerte sannsynligheter for høy lønn for ulike kombinasjoner av kjønn, klasse og alder. Merk at alle verdier ligger mellom 0 og 1, slik det skal være.

14.8 Presentere resultater med `modelsummary()`

Vi kan bruke `modelsummary()` for å lage penne tabeller, akkurat som for lineær regresjon. Her viser vi først koeffisienter på log-odds-skalaen:

```
modelsummary(list("Log-odds" = logit1, "Log-odds" = logit2),
            fmt = 3,
            estimate = "{estimate} ({std.error})",
            statistic = NULL,
            gofomit = 'DF|Deviance|AIC|BIC|Log.Lik.|F|RMSE')
```

Vi kan også vise odds-ratioer ved å bruke `exponentiate = TRUE`:

```
modelsummary(list("OR" = logit1, "OR" = logit2),
            exponentiate = TRUE,
            fmt = 2,
            estimate = "{estimate} ({std.error})",
            statistic = 'conf.int',
            gofomit = 'DF|Deviance|AIC|BIC|Log.Lik.|F|RMSE')
```

	OR	OR
(Intercept)	1.94 (0.09) [1.77, 2.13]	6.09 (1.46) [3.86, 9.89]
female	0.23 (0.02) [0.20, 0.26]	0.20 (0.02) [0.16, 0.24]
klasse89II Nedre serviceklasse		0.40 (0.09) [0.25, 0.60]
klasse89III Rutinefunksjonærer		0.06 (0.01) [0.04, 0.10]
klasse89V-VI Faglærte arbeidere		0.07 (0.02) [0.05, 0.11]
klasse89VIIa Ufaglærte arbeidere		0.05 (0.01) [0.03, 0.08]
age		1.03 (0.00) [1.02, 1.03]
Num.Obs.	3759	3680

Merk `exponentiate = TRUE` som gjør at koeffisientene vises som odds-ratioer. Det er som regel lurt å rapportere odds-ratioer fordi de er lettere å tolke.

For å eksportere til Word gjøres det på nøyaktig samme måte som for lineær regresjon, ved å sette `output = "filnavn.docx"`.

14.9 Modelltilpasning

I lineær regresjon har vi R^2 som mål på hvor godt modellen passer til dataene. I logistisk regresjon finnes det ingen perfekt ekvivalent, men det finnes flere varianter av pseudo- R^2 . Disse kan tolkes omrent på samme måte: et tall mellom 0 og 1 der høyere verdier betyr bedre tilpasning.

En annen tilnærming er å se på hvor godt modellen klassifiserer observasjonene. Vi kan sammenligne predikerte verdier med faktiske verdier:

```
abu89_pred <- abu89 %>%
  filter(!is.na(hoylenn), !is.na(klasse89), !is.na(age)) %>%
  mutate(pred_prob = predict(logit2, type = "response"),
        pred_klasse = ifelse(pred_prob > 0.5, 1, 0))
```

```
table(Faktisk = abu89_pred$hoylenn, Predikert = abu89_pred$pred_klasse)
```

		Predikert
Faktisk	0	1
0	1384	480
1	479	1337

Denne tabellen kalles en *forvirringsmatrise* (confusion matrix). Diagonalen viser korrekte prediksjoner, mens de andre cellene viser feilklassifiseringer. Andelen korrekte prediksjoner kan regnes ut slik:

```
mean(abu89_pred$hoylenn == abu89_pred$pred_klasse, na.rm = TRUE)
```

```
[1] 0.7394022
```

Dette gir oss en enkel indikasjon på hvor godt modellen predikerer. Husk imidlertid at hovedpoenget med logistisk regresjon i samfunnsvitenskap som regel er å forstå sammenhenger mellom variable, ikke nødvendigvis å predikere best mulig.

14.10 Oppsummering

Logistisk regresjon brukes når utfallsvariabelen er binær (0/1). De viktigste punktene er:

- Bruk `glm(y ~ x, family = binomial)` for å estimere modellen
- Koeffisientene er på log-odds-skalaen, konverter til odds-ratio med `exp()`
- Odds-ratio over 1 betyr høyere odds, under 1 betyr lavere odds
- Bruk `predict(modell, type = "response")` for å få predikerte sannsynligheter
- Bruk `modelsummary()` med `exponentiate = TRUE` for å vise odds-ratioer i tabeller
- Tolkningen av kontrollvariable er den samme som i lineær regresjon

15 Marginaleffekter

```
library(tidyverse)
library(haven)
library(marginaleffects)
```

I de foregående kapitlene har vi sett på lineær regresjon og logistisk regresjon. I lineær regresjon er regresjonskoeffisientene greie å tolke direkte: de uttrykker endring i gjennomsnittlig verdi på utfallsvariabelen per enhets endring i forklaringsvariabelen. Men i logistisk regresjon er koeffisientene på log-odds-skalaen, og det er ikke spesielt intuitivt. Vi kan eksponentiere til oddsratioer, men det er heller ikke alltid lett å forklare hva en oddsratio *betyr* i praksis.

Her kommer *marginaleffekter* inn i bildet. Marginaleffekter lar oss uttrykke effekten av en forklaringsvariabel som endring i sannsynlighet, noe som er langt mer intuitivt. Pakken `{marginaleffects}` gir oss et samlet rammeverk for å beregne marginaleffekter, predikerte verdier og sammenligninger mellom grupper. Hvis du ikke har installert pakken fra før:

```
install.packages("marginaleffects")
```

15.1 Eksempelmodeller

Vi trenger noen modeller å jobbe med. La oss estimere en lineær modell for timelønn og en logistisk modell for sannsynligheten for å ha høy lønn. Først lager vi en dummy for høy lønn, definert som timelønn over medianen.

```
abu89 <- abu89 %>%
  filter(!is.na(time89)) %>%
  mutate(hoy_lonn = ifelse(time89 > median(time89, na.rm = TRUE), 1, 0))
```

Så estimerer vi to modeller:

```
lm_mod <- lm(time89 ~ age + female + ed, data = abu89)
logit_mod <- glm(hoy_lonn ~ age + female + ed,
                  data = abu89, family = binomial)
```

I den lineære modellen kan vi lese koeffisientene direkte, men i den logistiske modellen er koeffisientene på log-odds-skalaen:

```
coef(logit_mod)
```

```
(Intercept)      age      female      ed
-1.84743222  0.03867057 -1.58421620  0.41064784
```

Det er her marginaleffekter kommer inn.

15.2 Gjennomsnittlige marginaleffekter (AME)

Den vanligste bruken er å beregne *Average Marginal Effects* (AME). Det betyr at vi beregner den marginale effekten for hver observasjon og tar gjennomsnittet. Funksjonen `avg_slopes()` gjør nettopp dette.

```
avg_slopes(logit_mod)
```

Term	Contrast	Estimate	Std. Error	z	Pr(> z)	S	2.5 %	97.5 %
age	dY/dX	0.00693	0.000538	12.9	<0.001	123.7	0.00588	0.00799
ed	dY/dX	0.07362	0.002552	28.8	<0.001	605.6	0.06862	0.07862
female	1 - 0	-0.30865	0.014233	-21.7	<0.001	344.0	-0.33655	-0.28076

Type: response

Nå er effektene uttrykt som endring i sannsynlighet. For eksempel betyr en effekt av `female` på -0.10 at kvinner i gjennomsnitt har 10 prosentpoeng lavere sannsynlighet for høy lønn sammenlignet med menn, alt annet likt.

For den lineære modellen gir `avg_slopes()` de samme verdiene som de vanlige regresjonskoeffisientene, fordi marginaleffekten er lik overalt i en lineær modell:

```
avg_slopes(lm_mod)
```

Term	Contrast	Estimate	Std. Error	z	Pr(> z)	S	2.5 %	97.5 %
age	dY/dX	0.542	0.0332	16.3	<0.001	196.6	0.477	0.607
ed	dY/dX	4.874	0.1626	30.0	<0.001	653.0	4.555	5.192
female	1 - 0	-17.601	0.8255	-21.3	<0.001	332.7	-19.219	-15.983

Type: response

15.3 Marginaleffekter ved bestemte verdier

Noen ganger vil man vite hva marginaleffekten er ved *bestemte* verdier av forklaringsvariablene. Funksjonen `slopes()` gjør dette. Med `datagrid()` kan vi lage et datasett med bestemte verdier, der variable som ikke spesifiseres holdes på gjennomsnittsverdier.

```
slopes(logit_mod,  
       variables = "ed",  
       newdata = datagrid(age = c(25, 40, 55)))
```

age	Estimate	Std. Error	z	Pr(> z)	S	2.5 %	97.5 %
25	0.0996	0.00458	21.7	<0.001	344.9	0.0906	0.1085
40	0.0833	0.00366	22.8	<0.001	379.2	0.0761	0.0904
55	0.0608	0.00327	18.6	<0.001	254.2	0.0544	0.0672

Term: ed
Type: response
Comparison: dY/dX

Her ser vi at effekten av utdanning varierer med alder. Det er fordi logistisk regresjon er ikke-lineær: den marginale effekten avhenger av *hvor* på fordelingen man befinner seg.

15.4 Predikerte verdier

I kapittelet om lineær regresjon brukte vi `predict()` for å beregne predikerte verdier. Pakken `{marginaleffects}` har tilsvarende funksjoner som gir ryddigere output med konfidensintervaller. Funksjonen `predictions()` gir predikerte verdier, mens `avg_predictions()` gir gjennomsnitt per gruppe.

```
predictions(logit_mod,  
            newdata = datagrid(age = c(25, 35, 45, 55),  
                               female = c(0, 1)))
```

age	female	Estimate	Pr(> z)	S	2.5 %	97.5 %
25	0	0.587	<0.001	22.6	0.555	0.618
25	1	0.226	<0.001	192.5	0.201	0.253
35	0	0.677	<0.001	134.7	0.653	0.700

```

35      1    0.300   <0.001 144.4 0.276  0.326
45      0    0.755   <0.001 258.0 0.732  0.776
45      1    0.387   <0.001 45.5 0.360  0.415
55      0    0.819   <0.001 267.6 0.795  0.841
55      1    0.482     0.331   1.6 0.445  0.518

```

Type: invlink(link)

Med `avg_predictions()` kan vi beregne gjennomsnittlig predikert sannsynlighet fordelt på grupper:

```
avg_predictions(logit_mod, by = "female")
```

female	Estimate	Std. Error	z	Pr(> z)	S	2.5 %	97.5 %
0	0.660	0.00968	68.2	<0.001	Inf	0.641	0.679
1	0.305	0.00984	31.0	<0.001	700.1	0.286	0.325

Type: response

15.5 Sammenligninger mellom grupper

Funksjonen `avg_comparisons()` beregner gjennomsnittlige forskjeller mellom grupper eller ved endring i en variabel. For eksempel forskjellen mellom menn og kvinner:

```
avg_comparisons(logit_mod, variables = "female")
```

	Estimate	Std. Error	z	Pr(> z)	S	2.5 %	97.5 %
	-0.309	0.0142	-21.7	<0.001	344.0	-0.337	-0.281

Term: female

Type: response

Comparison: 1 - 0

Vi kan også se på effekten av en bestemt endring i en kontinuerlig variabel, for eksempel fra 10 til 15 års utdanning:

```
avg_comparisons(logit_mod,
                  variables = list(ed = c(10, 15)))
```

Estimate	Std. Error	z	Pr(> z)	S	2.5 %	97.5 %
0.0547	0.00619	8.83	<0.001	59.8	0.0426	0.0668

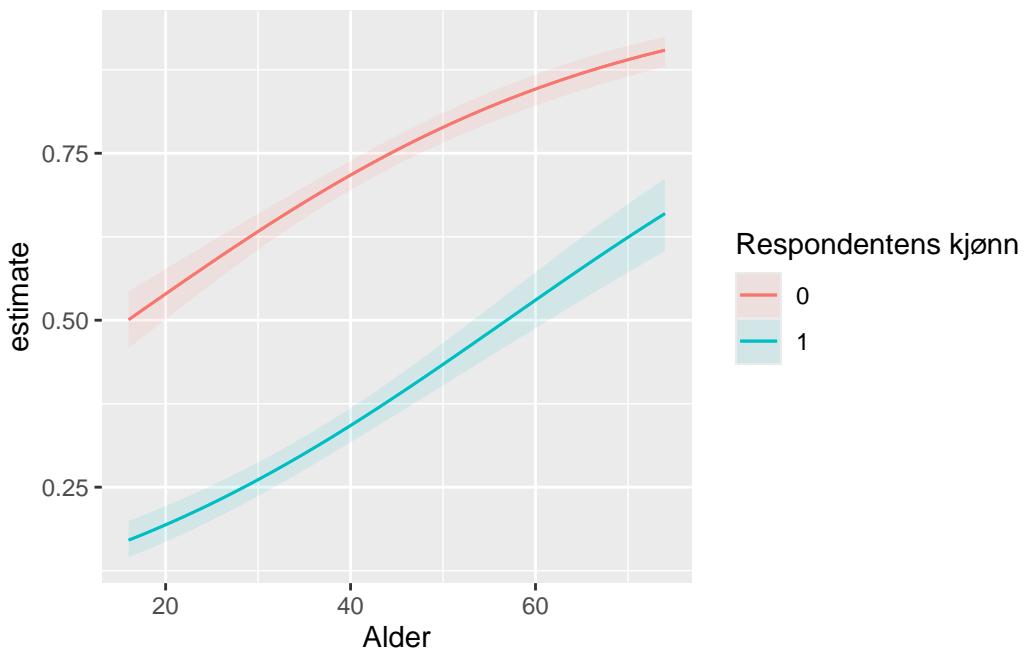
Term: ed
 Type: response
 Comparison: 15 - 10

15.6 Plotting av marginaleffekter

En stor styrke med `{marginaleffects}` er enkle og fine plot. Funksjonen `plot_predictions()` viser predikerte verdier, mens `plot_slopes()` viser hvordan marginaleffekten varierer.

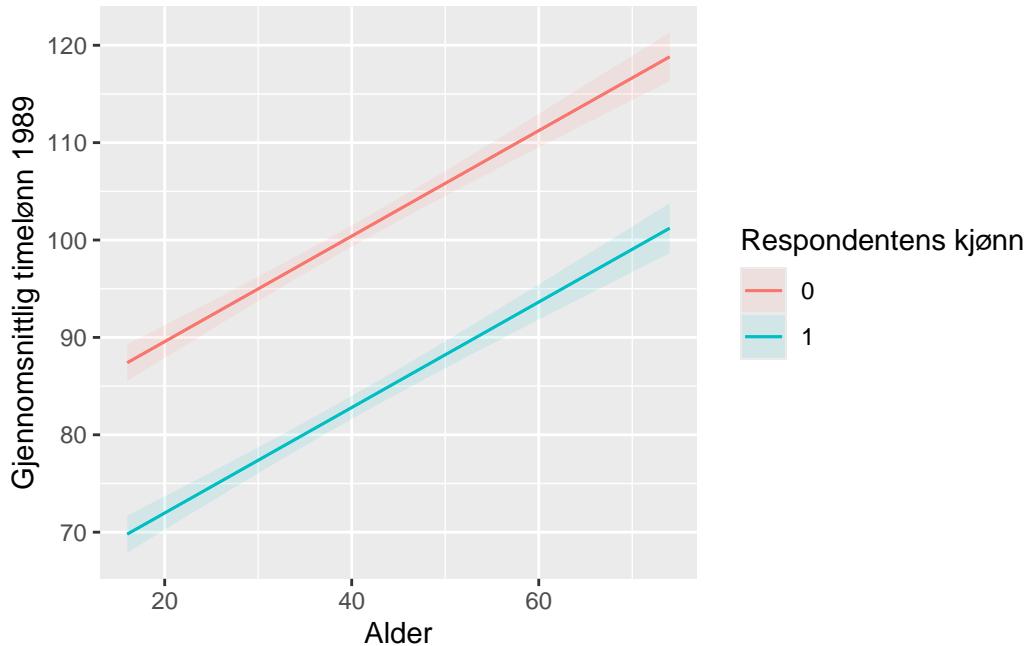
Her plotter vi predikert sannsynlighet for høy lønn som funksjon av alder, separat for menn og kvinner:

```
plot_predictions(logit_mod, condition = c("age", "female"))
```



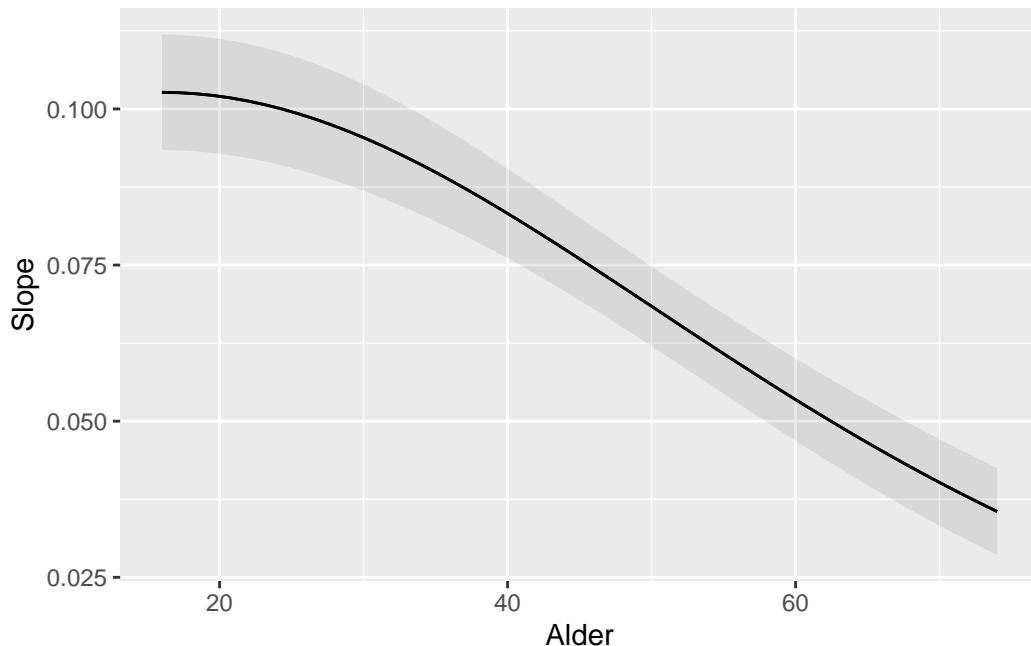
Den grå sonen viser konfidensintervallet. Vi ser den karakteristiske S-formen fra logistisk regresjon. For den lineære modellen blir linjene rette:

```
plot_predictions(lm_mod, condition = c("age", "female"))
```



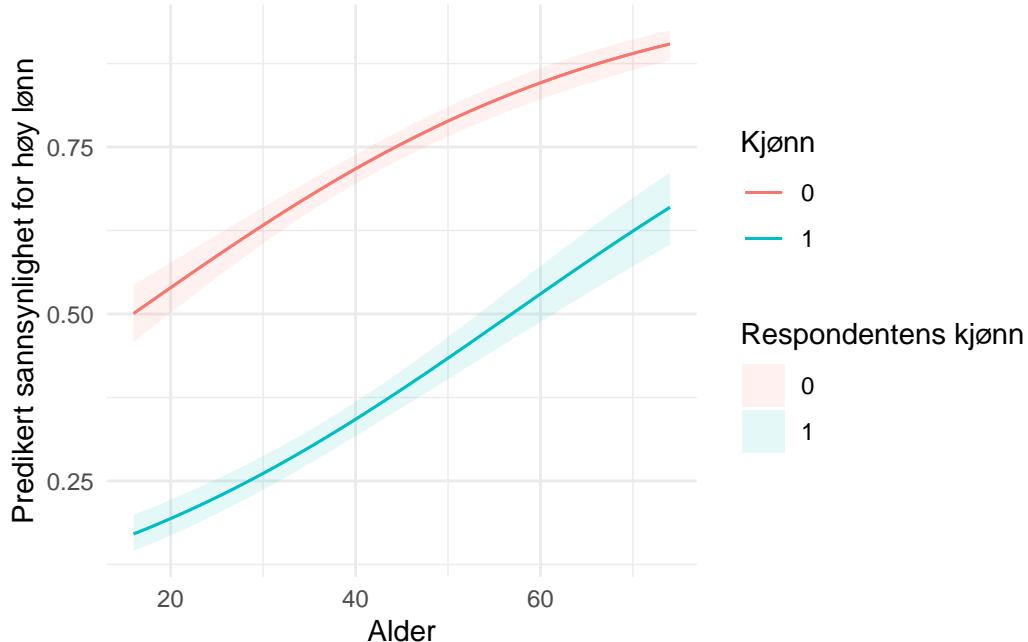
Med `plot_slopes()` kan vi visualisere hvordan marginaleffekten av en variabel varierer med en annen:

```
plot_slopes(logit_mod, variables = "ed", condition = "age")
```



Siden disse funksjonene returnerer ggplot-objekter, kan de tilpasses med vanlig ggplot-syntaks:

```
plot_predictions(logit_mod, condition = c("age", "female")) +  
  labs(x = "Alder",  
        y = "Predikert sannsynlighet for høy lønn",  
        color = "Kjønn") +  
  theme_minimal()
```



15.7 Sammenhengen med predict()

I kapittelet om lineær regresjon brukte vi `predict()` for å beregne predikerte verdier. Her er en kort sammenligning:

```
nyedata <- data.frame(age = 40, female = 0, ed = 12)

# Med predict():
predict(logit_mod, newdata = nyedata, type = "response")
```

```
1
0.9903123
```

```
# Med predictions():
predictions(logit_mod, newdata = nyedata)
```

```
Estimate Pr(>|z|)      S 2.5 % 97.5 %
0.99    <0.001 408.6 0.986  0.993
```

Type: `invlink(link)`

Begge gir samme predikerte verdi, men `predictions()` gir i tillegg konfidensintervaller og standardfeil. Funksjonen `predict()` er innebygd i R og fungerer alltid, men `{marginaleffects}` gir ryddigere output og er enklere for mer avanserte ting.

15.8 Oppsummering

Funksjon	Hva den gjør
<code>avg_slopes()</code>	Gjennomsnittlige marginaleffekter (AME)
<code>slopes()</code>	Marginaleffekter ved bestemte verdier
<code>predictions()</code>	Predikerte verdier for angitte observasjoner
<code>avg_predictions()</code>	Gjennomsnittlige predikerte verdier per gruppe
<code>avg_comparisons()</code>	Gjennomsnittlige sammenligninger mellom grupper
<code>plot_predictions()</code>	Plot av predikerte verdier
<code>plot_slopes()</code>	Plot av marginale effekter

Hovedpoenget er at marginaleffekter lar oss tolke resultater fra ikke-lineære modeller (som logistisk regresjon) på en intuitiv skala. I stedet for log-odds eller oddsratioer kan vi snakke om endring i sannsynlighet, noe som er langt lettere å forstå.

16 Prediksjon og maskinlæring

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

I tidligere kapitler har vi brukt regresjonsmodeller til å beskrive sammenhenger mellom variable. Vi har tolket regresjonskoeffisienter som uttrykk for forskjeller mellom grupper. Men regresjonsmodeller kan også brukes til noe annet: *prediksjon*. Her gir vi en kort introduksjon til hvordan man tenker på prediksjon og hva som skiller det fra den vanlige tilnærmingen i samfunnsvitenskap.

16.1 Forklaring vs. prediksjon

I samfunnsvitenskap er vi vanligvis opptatt av *forklaring*: vi vil forstå *hvorfor* ting henger sammen. Da er vi mest interessert i regresjonskoeffisientene. Hva er sammenhengen mellom utdanning og inntekt? Hva er kjønnsforskjellen i timelønn?

Men noen ganger er vi mer opptatt av å *predikere* utfall. Da bryr vi oss ikke nødvendigvis om hvorfor modellen virker, men om den gir gode gjettninger. Eksempler kan være:

- Hvilke pasienter har høy risiko for tilbakefall?
- Hvilke elever har høy risiko for å falle fra videregående?
- Hva blir arbeidsledigheten neste kvartal?

I slike tilfeller er det *utfallsvariabelen* vi er mest opptatt av, ikke forklaringsvariablene. Vi vil ha en modell som gir gode prediksjoner for nye observasjoner vi ikke har sett ennå. Dette er kjernen i det som ofte kalles *maskinlæring*.

16.2 Overtilpasning

Et sentralt problem i prediksjon er *overtilpasning* (engelsk: *overfitting*). En modell som er veldig kompleks kan tilpasses seg alle særegenheterne i dataene vi har, inkludert tilfeldig støy. Da får modellen veldig god “score” på treningsdataene, men dårlige prediksjoner for nye data.

Tenk deg at du pugger gamle eksamensoppgaver ord for ord. Du kan gjengi alle svarene perfekt – men bare for akkurat de oppgavene. Får du en ny oppgave som er litt annerledes, hjelper det lite. Det er det samme som skjer med en overtilpasset modell.

En enkel modell med få variable vil kanskje ikke fange opp alt i treningsdataene, men kan likevel gi bedre prediksjoner for nye data fordi den har lært det generelle mønsteret i stedet for støyen. Balansen mellom for enkel og for kompleks modell er det viktigste spørsmålet i prediksjon.

16.3 Trening og test: dele opp dataene

Hvordan vet vi om modellen vår gir gode prediksjoner for *nye* data? En enkel strategi er å dele opp dataene i to deler:

- **Treningsdata:** Brukes til å estimere modellen
- **Testdata:** Brukes til å evaluere hvor gode prediksjonene er

Vi estimerer modellen kun på treningsdataene, og deretter ser vi hvor godt modellen predikrer utfallsvariabelen itestdataene – data modellen aldri har “sett” under estimeringen.

La oss prøve dette med abu89-datasettet. Vi deler tilfeldig dataene i 80% trening og 20% test.

```
set.seed(42)
n <- nrow(abu89)
trenings_indeks <- sample(1:n, size = round(0.8 * n))

trening <- abu89[trenings_indeks, ]
test     <- abu89[-trenings_indeks, ]
```

Funksjonen `set.seed()` sørger for at den tilfeldige oppdelingen blir den samme hver gang koden kjøres. Det er god praksis for reproducertbarhet.

16.4 En enkel prediksjonsmodell

Vi bruker helt vanlig lineær regresjon som prediksjonsmodell. La oss predikere timelønn (`time89`) basert på alder, kjønn, utdanning og yrkeserfaring.

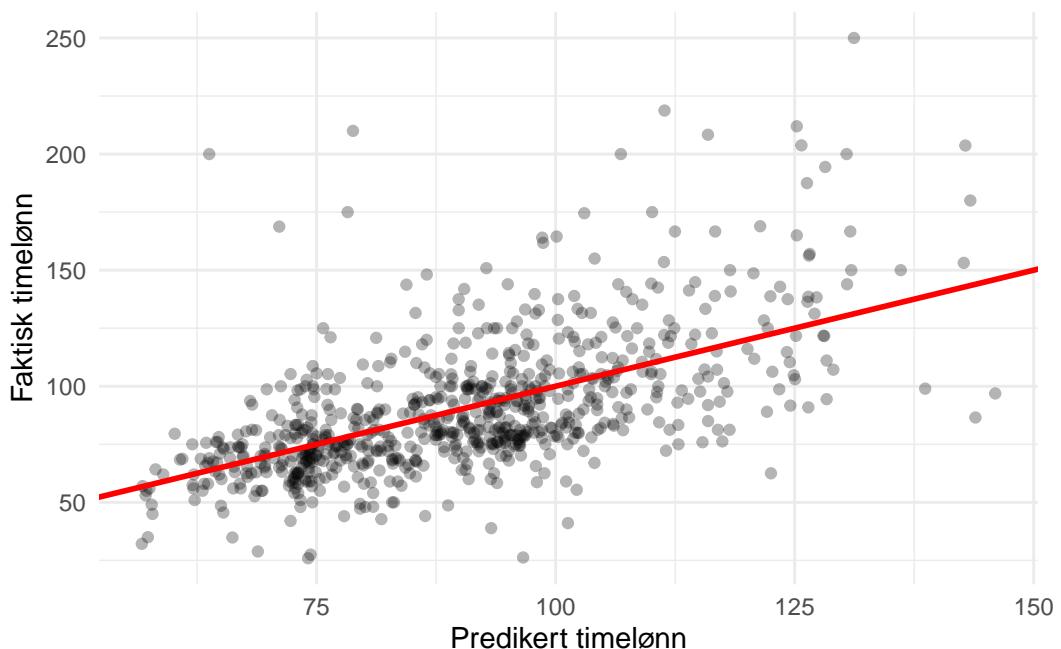
```
mod <- lm(time89 ~ age + female + ed + fexp, data = trening)
```

Nå bruker vi `predict()` til å lage prediksjoner for testdataene:

```
test <- test %>%
  mutate(predikert = predict(mod, newdata = test))
```

Vi kan visualisere hvor godt modellen treffer ved å plotte predikert mot faktisk timelønn:

```
ggplot(test, aes(x = predikert, y = time89)) +
  geom_point(alpha = 0.3) +
  geom_abline(intercept = 0, slope = 1, col = "red", linewidth = 1) +
  labs(x = "Predikert timelønn", y = "Faktisk timelønn") +
  theme_minimal()
```



Den røde linjen viser hvor punktene ville ligget hvis modellen predikerte perfekt. Vi ser at modellen fanger opp den generelle trenden, men det er mye variasjon den ikke klarer å forklare. Det er helt normalt for denne typen data.

16.5 Mål på prediksjonsevne

For å tallfeste hvor gode prediksjonene er bruker vi gjerne to mål:

RMSE (*Root Mean Squared Error*) måler gjennomsnittlig avvik mellom predikert og faktisk verdi, i samme enhet som utfallsvariabelen. Lavere er bedre.

R-kvadrat (R^2) måler hvor stor andel av variasjonen i utfallsvariabelen modellen forklarer. Verdien ligger mellom 0 og 1, der 1 betyr perfekt prediksjon.

```
residualer <- test$time89 - test$predikert

rmse <- sqrt(mean(residualer^2))
r2   <- 1 - sum(residualer^2) / sum((test$time89 - mean(test$time89))^2)

data.frame(RMSE = round(rmse, 1),
           R2    = round(r2, 3))
```

```
RMSE      R2
1 23.6 0.366
```

Her er RMSE uttrykt i kroner, altså det gjennomsnittlige avviket i predikert timelønn. R-kvadrat forteller oss hvor mye av variasjonen i timelønn modellen fanger opp.

16.6 Kryssvalidering

Oppdelingen i trening og test er litt tilfeldig. Kanskje fikk vi et uheldig utvalg? En mer robust tilnærming er *kryssvalidering* (engelsk: *cross-validation*). Prinsippet er at man deler dataene i k deler (f.eks. 10), og så bruker man 9 deler til trening og 1 del til testing. Dette gjentas slik at hver del brukes som testdata nøyaktig en gang. Til slutt regner man ut gjennomsnittet av prediksionsfeilen over alle rundene.

Vi går ikke nærmere inn på implementeringen her, men det er viktig å vite at dette er standardmetoden for å evaluere prediksjonsmodeller. Pakker som `caret` og `tidymodels` i R gjør dette enkelt i praksis.

16.7 Regularisering: LASSO

Når man har mange mulige forklaringsvariable, kan det være fristende å inkludere alle sammen. Men det kan føre til overtilpasning. En teknikk som hjelper med dette er *regularisering*. Den mest brukte varianten kalles **LASSO** (Least Absolute Shrinkage and Selection Operator).

LASSO fungerer som vanlig regresjon, men legger til en “straff” for store koeffisienter. Resultatet er at noen koeffisienter krympes mot null, og noen settes til nøyaktig null. Det betyr at LASSO automatisk velger bort variable som ikke bidrar nok til prediksjon. Dette er veldig nyttig når man har mange potensielle forklaringsvariable.

Vi viser ikke koden for LASSO her, men pakken `glmnet` er standardverktøyet i R. Poenget er at dette er et naturlig neste steg etter at man har forstått prediksjon med vanlig regresjon.

16.8 Klassifisering

Så langt har vi predikert en kontinuerlig variabel (timelønn). Men ofte vil vi predikere en *kategorisk* variabel. For eksempel: vil en person bli arbeidsledig eller ikke? Er en e-post spam eller ikke? Dette kalles *klassifisering*.

For klassifisering bruker vi gjerne logistisk regresjon (som vi har sett i et annet kapittel) i stedet for lineær regresjon. Evalueringsmålene er da litt annerledes: i stedet for RMSE ser vi på *andelen riktig klassifiserte* (accuracy) og andre mål som presisjon og recall.

16.9 Når er prediksjon nyttig i samfunnsvitenskap?

Selv om samfunnsvitenskap tradisjonelt er mest opptatt av forklaring, er det flere områder der prediksjon er sentralt:

- **Risikoscoring:** Identifisere individer med høy risiko for f.eks. tilbakefall til kriminalitet, frafall fra skolen, eller helseproblemer
- **Klassifisering av tekst:** Automatisk kategorisering av store mengder tekst, f.eks. avisartikler eller stortingsdebatter
- **Prognoser:** Forutsi utvikling i arbeidsledighet, kriminalitet eller demografiske trender
- **Variabelseleksjon:** Bruke maskinlæring til å identifisere hvilke variable som er viktigst, som utgangspunkt for videre analyse

Det er verdt å merke seg at mange av maskinlæringsmetodene (tilfeldige skoger, nevrale nettverk osv.) bygger på de samme grunnprinsippene vi har gjennomgått her: dele opp data, evaluere prediksjonsevne, og balansere modellkompleksitet mot overtilpasning.

16.10 Videre lesning

Dette kapittelet har gitt en helt grunnleggende smakebit på prediksjon og maskinlæring. Temaet er svært omfattende, og vi har bare skrapet i overflaten. For de som vil lære mer er [SOS2901](#) et introduksjonskurs i maskinlæring for samfunnsvititere ved UiO som tar utgangspunkt i nettopp disse konseptene. En god bok for videre lesning er *An Introduction to Statistical Learning* av James, Witten, Hastie og Tibshirani, som også er tilgjengelig gratis på nett.

Part IV

Del IV: Statistisk tolkning

17 Statistisk tolkning

Det vi omtaler som *statistisk tolkning* eller *statistisk inferens* handler om å skille systematikk fra støy. Altså: håndtering av usikkerhet ved estimeringen.

- 1) "Bekvemmelighetsutvalg": Dataene er ikke et tilfeldig utvalg fra noen bestemt populasjon, og det er ikke randomisert til f.eks. kontrollgruppe. I slike utvalg er det ikke den typen usikkerhet ved estimeringen at vi kan regne på usikkerhet på meningsfull måte. Statistisk tolkning er lite relevant.
- 2) Data er et tilfeldig utvalg fra en veldefinert populasjon. Altså: det vi vanligvis mener med "tilfeldig utvalg". Hvis ikke populasjonen er veldefinert spørs det om det heller er et utvalg av type 2). Usikkerheten er knyttet til tilfeldigheter i hvem som ble med i utvalget. Statistisk tolkning for generalisering gjelder.
- 3) Data er fra et eksperiment eller kvasieksperiment der det er tilfeldig fordeling av hvem som er i behandlingsgruppen og kontrollgruppen (evt. flere grupper). Usikkerhet er knyttet til måling av kausaleffekter, og statistisk tolkning gjelder.

På nivå 2 handler det om å *generalisere* til en veldefinert populasjon, mens det på nivå 3 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 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?

¹Hvis dette er ukjent stoff for deg kan du se f.eks. Moore, Notz, and Fligner (2021), kapittel 15, etterfulgt av forlengelsen til konfidensintervall og statistiske tester i kapittel 16 og 17.

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 utvalgstørrelsen. 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.

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.

17.1 Stokastiske variabler og sannsynlighetsfordelinger

En stokastisk variabel er en funksjon som tilordner en numerisk verdi til utfallet av et eksperiment. For eksempel, et terningkast er et eksperiment med et utfallsrom som omfatter tallene 1 til 6. Det er usikkerhet knyttet til utfallet av eksperimentet (vi vet ikke på forhånd hva utfallet er, altså om vi kaster 2 eller 5 eller noe annet). La oss si at jeg kaster terningen 100 ganger og teller antall ganger jeg får 6. Vi kan se på denne prosessen som en stokastisk variabel som mäter antall 6'ere (den gir utfallet en numerisk verdi).

Hvis vi kaster en vanlig terning, har alle mulige utfall den samme sannsynligheten ($1/6$). I andre tilfeller, dvs. for andre stokastiske variabler, kan det hende at noen utfall er mer sannsynlige enn andre. Alle stokastiske variabler er derfor forbundet med en tilsvarende sannsynlighetsfordeling. En sannsynlighetsfordeling er en funksjon som tilordner en sannsynlighetsverdi (et tall mellom 0 og 1) til alle mulige verdier av den stokastiske variablene. Et terningkast følger en binomisk fordeling. Andre fordelinger du kanskje har hørt om er Bernoulli-fordelingen, uniformfordelingen, Poisson-fordelingen og selsvsagt normalfordelingen.

Mini-oppgave: Kan du tenke på og forklare hvordan en survey er et stokastisk utfall (en realisering) av et eksperiment?

17.1.1 Estimater som stokastiske variabler

Når vi produserer statistiske estimater, er vi som oftest avhengige av et (tilfeldig) utvalg fra en større populasjon (vi mäter som regel utdanningsnivået til et utvalg mennesker, ikke for hele befolkningen). Dette betyr at et estimat kan variere fra ett utvalg til et annet: det er usikkerhet knyttet til det. Med andre ord, kan selve estimatet anses som en stokastisk variabel. Med nok observasjoner, forteller sentralgrenseteoremet at denne stokastiske variablene følger en

normalfordeling med to viktige parametere: en gjennomsnittlig (forventet) verdi og en varians. Denne variansen kvantifiserer hvor mye estimatene våre kan variere dersom vi trekker et nytt (hypotetisk) utvalg fra populasjonen. I praksis, forholder vi oss som regel til standardavviket og ikke variansen til denne utvalgsfordelingen. Standardavviket til denne fordelingen kalles *standardfeilen* (se under).

Standardfeilen forbundet med et estimat kan brukes til å bygge et konfidensintervall (se under). Dette er et intervall som med en gitt sannsynlighet inneholder den sanne verdien av det vi prøver å estimere (gitt antakelsene som vår statistiske modell legger til grunn). Et 95 prosent konfidensintervall er definert som et intervall som gjennom hypotetisk repeterete datasamlinger, "fanger" denne sanne verdien med 95 prosent sannsynlighet. Som en røff tommelfingerregel, tilsvarer et slikt intervall estimatet ± 2 ganger standardfeilen.

Mini-oppgave: Gitt det du vet om normalfordelingen, forklar hvorfor vi er 95 prosent sikre på at den sanne verdien til det vi estimerer faller innenfor ca. 2 ganger standardfeilen.

17.2 Estimater og feilmarginer

17.2.1 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*.

17.2.2 Standardfeil

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

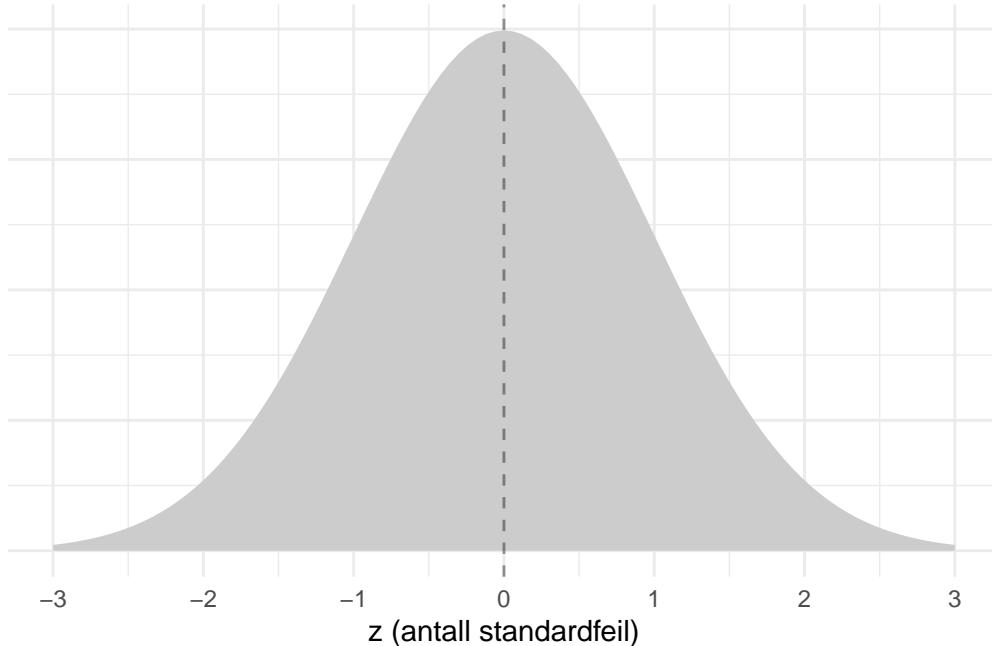
Ordet *standardfeil* er lett å blande sammen med *standardavvik* i mer generell forstand, så la oss ta det med det samme. Standardavviket beskriver som regel variasjon i observerte data, f.eks. hvis man vil beskrive hvordan personers inntekt varierer rundt gjennomsnittet. Standardavviket beskriver også variasjon i *data*.

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

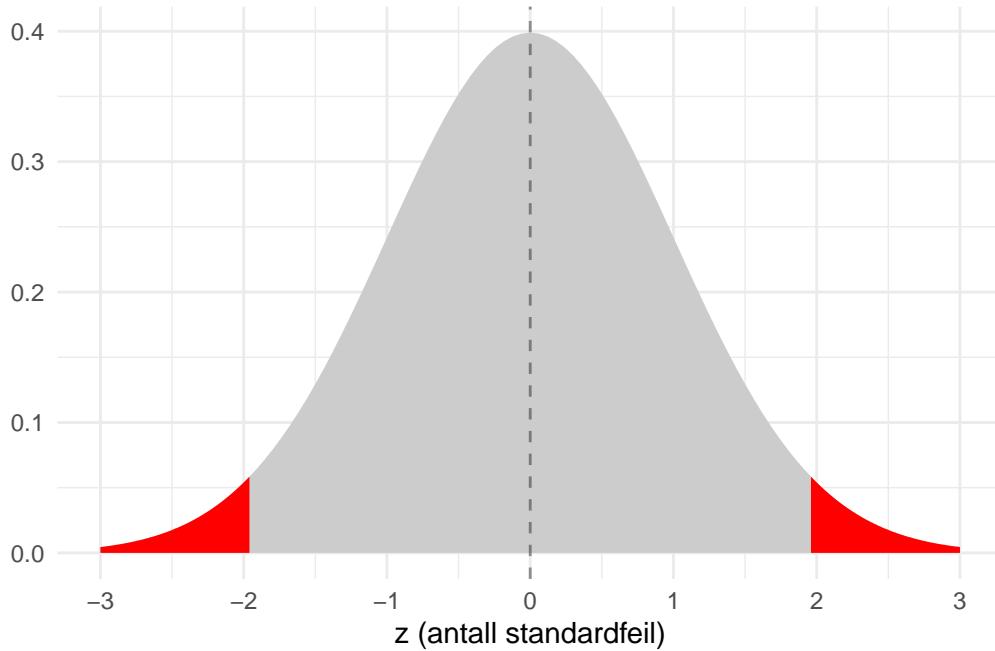
Sentralgrenseteoremet sier at estimatet på et gjennomsnitt vil ha tilfeldige feil som er *normalfordelt*, og dermed kan vi bruke normalfordelingen til å si noe om usikkerheten ved estimatet. Dette er forsøkt illustrert nedenfor der x-aksen viser hvor mye estimatet kan avvike fra den sanne verdien, mens kurven viser hvor sannsynlighetsfordelingen til avviket. Den stiplete 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 nærmere 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 estimat innenfor intervallet mellom -1.96 og 1.96 tilsvarende 95%. Dette er grunnlaget for det vi kaller *konfidensintervall*.



17.2.3 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:

$$90.2 \pm 1.96 \times 0.5 = [89.2, 91.1]$$

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.

17.2.3.1 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 nærmere 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 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.

17.2.4 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 skyldes tilfeldigheter. 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).

17.2.4.1 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:

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

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.

17.2.4.2 P-verdi

Tolkningen av p-verdien er i hvilken grad det er sannsynlig å få det observerte resultatet *ved en tilfeldighet* hvis NULL-hipotesen 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-hipotesen 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.⁷(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 hipotesen.

17.3 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!

17.4 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.

18 Statistikk i praksis

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

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.

18.1 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*.⁷ (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 ¹	Menn N = 2,193 ¹	Difference ²	95% CI ²
Gjennomsnittlig timelønn 1989	79 (24)	100 (32)	-21	-23, -19
År utdanning	2.38 (2.40)	2.96 (2.66)	-0.58	-0.74, -0.43
Bedriftserfaring	0.83 (0.81)	1.05 (0.97)	-0.22	-0.27, -0.16
Alder	40 (13)	40 (12)	-0.17	-0.93, 0.58

¹Mean (SD)

²Welch Two Sample t-test

Abbreviation: CI = Confidence Interval

Characteristic	Kvinner N = 1,934 ¹	Menn N = 2,193 ¹	p-value ²
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

¹Mean (SD)

²Welch Two Sample t-test

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()
```

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

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

Characteristic	Kvinner N = 1,934 ¹	Menn N = 2,193 ¹	p-value ²
Klasse			<0.001
I Øvre serviceklasse	74 (3.9%)	254 (12%)	
II Nedre serviceklasser	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%)	

¹n (%)

²Pearson's Chi-squared test

```
abu89 %>%
  select(female, klasse89, promot, private) %>%
  mutate(female = ifelse(female == 0, "Menn", "Kvinner")) %>%
 tbl_summary(by = female,
            label = list(klasse89 = "Klasse"),
            missing = "no") %>%
add_p()
```

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()
```

Characteristic	Overall N = 4,127 ¹	Kvinner N = 1,934 ¹	Menn N = 2,193 ¹	p-v
Gjennomsnittlig timelønn 1989	90 (30)	79 (24)	100 (32)	<
År utdanning	2.69 (2.56)	2.38 (2.40)	2.96 (2.66)	<
Alder	40 (12)	40 (13)	40 (12)	<
Klasse				<
I Øvre serviceklasse	328 (8.1%)	74 (3.9%)	254 (12%)	
II Nedre serviceklasser	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				<
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)	<
Privat sektor				<
Public	1,602 (39%)	1,016 (53%)	586 (27%)	
Private	2,525 (61%)	918 (47%)	1,607 (73%)	

¹Mean (SD); n (%)

²Welch Two Sample t-test; Pearson's Chi-squared test

18.2 Regresjon

For regresjon er det i prinsippet det samme: regresjonskoeffisientene er estimater med usikkerhet som uttrykkes med standardfeil og tilhørende konfidenstintervaller og p-verdier. Merk at p-verdiene er resultatet av en helt ordinær t-test:

$$t = \frac{\beta}{se_{\beta}}$$

Husk at β er et estimat på en forskjell mellom grupper eller nivåer på en kontinuerlig variabel. Tolkningen er derfor lik som for t-test: kan denne *forskjellen* skyldes tilfeldig variasjon? Eller er forskjellen såpass stor i forhold til feilmarginen at vi velger å tolke det som en systematisk forskjell? Hvis p-verdien er høy (typisk: større enn 0.05), så er vi ikke tilstrekkelig sikker på at det ikke bare er tilfeldig støy.

```
lm_est1 <- lm(time89 ~ female , data = abu89)
modelsummary(lm_est1)
```

	(1)
(Intercept)	99.844 (0.637)
female	-20.752 (0.932)
Num.Obs.	3759
R2	0.117
R2 Adj.	0.116
AIC	35 854.9
BIC	35 873.6
Log.Lik.	-17 924.434
F	496.278
RMSE	28.49

18.2.1 Multippel regresjon

```
lm_est2 <- lm(time89 ~ female + age , data = abu89)
modelsummary(lm_est2)
```

18.2.2 Interaksjonsledd

```
lm_est3 <- lm(time89 ~ female + age + female * age , data = abu89)
modelsummary(lm_est3)
```

```
modelsummary(list(lm_est1, lm_est2, lm_est3))
```

	(1)
(Intercept)	81.101 (1.585)
female	-20.625 (0.912)
age	0.474 (0.037)
Num.Obs.	3759
R2	0.154
R2 Adj.	0.153
AIC	35 694.9
BIC	35 719.8
Log.Lik.	-17 843.437
F	341.699
RMSE	27.88
	(1)
(Intercept)	75.854 (2.126)
female	-9.905 (3.040)
age	0.606 (0.051)
female × age	-0.272 (0.074)
Num.Obs.	3759
R2	0.157
R2 Adj.	0.156
AIC	35 683.2
BIC	35 714.4
Log.Lik.	-17 836.611
F	233.121
RMSE	27.83

	(1)	(2)	(3)
(Intercept)	99.844 (0.637)	81.101 (1.585)	75.854 (2.126)
female	-20.752 (0.932)	-20.625 (0.912)	-9.905 (3.040)
age		0.474 (0.037)	0.606 (0.051)
female × age			-0.272 (0.074)
Num.Obs.	3759	3759	3759
R2	0.117	0.154	0.157
R2 Adj.	0.116	0.153	0.156
AIC	35 854.9	35 694.9	35 683.2
BIC	35 873.6	35 719.8	35 714.4
Log.Lik.	-17 924.434	-17 843.437	-17 836.611
F	496.278	341.699	233.121
RMSE	28.49	27.88	27.83

Part V

Del V: Tolkning og vitenskapelig praksis

19 Forskningsdesign og tolkning

Denne boken handler primært om *hvordan* du gjør ting i R. Men det er like viktig å forstå *hva* du faktisk gjør og *hvorfor*. Statistiske analyser gir bare mening i lys av forskningsdesignet og den substansielle teorien du jobber med. I dette kapittelet diskuterer vi noen viktige prinsippelle poenger som bør ligge i bakhodet når du gjør dataanalyse.

19.1 Observasjonsdata vs. eksperimentelle data

I samfunnsvitenskap jobber vi nesten alltid med *observasjonsdata*. Det vil si at vi observerer verden slik den er, uten å ha manipulert noe. I et *eksperiment* kan forskeren derimot kontrollere hvem som får en “behandling” og hvem som ikke får det.

Denne forskjellen har store konsekvenser for hva vi kan konkludere. I et eksperiment med tilfeldig fordeling av behandling kan vi si at forskjeller mellom gruppene *skyldes* behandlingen. Med observasjonsdata kan vi som regel ikke trekke slike slutninger like enkelt, fordi gruppene kan være forskjellige på mange andre måter enn det vi er interessert i.

Det betyr ikke at observasjonsdata er ubrukelige - det er tvert imot svært mye nyttig kunnskap vi kan trekke ut. Men vi må være bevisste på begrensningene og tolke resultatene deretter.

19.2 Korrelasjon er ikke kausalitet

Du har garantert hørt dette utsagnet før. Men hva betyr det egentlig i praksis?

Når vi finner en statistisk sammenheng mellom to variable, betyr det at de *samvarierer*. Men det betyr ikke nødvendigvis at den ene *forårsaker* den andre. Det finnes flere grunner til at to variable kan samvariere:

1. **Kausalitet:** X forårsaker faktisk Y. For eksempel: røyking forårsaker lungekreft.
2. **Omvendt kausalitet:** Y forårsaker X. For eksempel: dårlig helse (Y) kan føre til lavere inntekt (X), ikke bare omvendt.
3. **Felles bakenforliggende årsak (confounding):** En tredje variabel Z påvirker både X og Y. For eksempel: fysisk aktivitet (X) og god helse (Y) kan begge påvirkes av sosioøkonomisk bakgrunn (Z).
4. **Tilfeldig samvariasjon:** Med nok variable vil noen alltid korrelere tilfeldig.

I observasjonsstudier er det spesielt *confounding* som er den store utfordringen. Det er dette vi prøver å håndtere med kontrollvariable i regresjonsanalyser (se kapittelet om kontrollvariable), men det er begrenset hva vi kan oppnå med statistiske metoder alene.

19.3 Seleksjonsskjevhets

Seleksjonsskjevhets oppstår når utvalget vi studerer ikke er representativt for populasjonen vi er interessert i. Det kan skje på mange måter:

- **Utvalgsskjevhets:** De som svarer på en spørreundersøkelse er kanskje systematisk forskjellige fra de som ikke svarer.
- **Selvseleksjon:** Folk velger selv om de deltar i et program eller en aktivitet. De som velger å delta kan være systematisk forskjellige fra de som ikke deltar.
- **Frafall:** Deltakere faller ut av en studie, og frafallet er kanskje ikke tilfeldig.

I praksis betyr dette at vi alltid bør tenke over: hvem er det egentlig vi studerer? Og kan resultatene generaliseres til en bredere befolkning?

19.4 Teoriens rolle

Statistiske analyser kan fortelle oss *om* det finnes en sammenheng mellom variable, og *hvor sterkt* sammenhengen er. Men statistikken kan ikke fortelle oss *hvorfor* sammenhengen finnes. Det er teoriens jobb.

Teori er viktig for dataanalyse på flere måter:

1. **Velge variable:** Teori hjelper oss å bestemme hvilke variable som bør inkluderes i analysen og hvilke som bør holdes utenfor.
2. **Bestemme modellstruktur:** Bør vi inkludere interaksjonsledd? Ikke-lineære termer? Det avhenger av hva teorien sier om sammenhengen.
3. **Tolke resultater:** Et regresjonsestimat er bare et tall inntil vi legger en substansiell tolkning på det.
4. **Identifisere confounders:** Teori forteller oss hvilke bakenforliggende variable som kan forstyrre analysen.

Poenget er at god dataanalyse krever faglig forståelse av det du studerer. R er et verktøy, men det er *du* som må bruke det klokt.

19.5 DAGer: Rettet asyklistisk graf

Et nyttig verktøy for å tenke om kausale sammenhenger er *DAGer* (Directed Acyclic Graphs, eller rettede asyklistiske grafer). En DAG er et diagram som viser antatte kausale sammenhenger mellom variable.

```
library(ggdag)

dag <- dagify(
  Y ~ X + Z,
  X ~ Z,
  labels = c("Y" = "Lønn",
             "X" = "Utdanning",
             "Z" = "Familiebakgrunn")
)

ggdag(dag, text = FALSE, use_labels = "label") +
  theme_dag()
```

I denne DAGen ser vi at:

- Familiebakgrunn påvirker både utdanning og lønn
- Utdanning påvirker lønn
- Familiebakgrunn er en *confounding*-variabel som vi bør kontrollere for

DAGer tvinger oss til å være eksplisitte om hvilke kausale antakelser vi gjør. Det kan avsløre problemer vi ikke hadde tenkt på. Det kan også hjelpe oss å unngå å kontrollere for variable vi *ikke* bør kontrollere for (se kapittelet om kontrollvariable, avsnittet om “dårlige kontrollvariable”).

For mer om DAGer anbefales å utforske pakken `dagitty` og nettressursen [daggity.net](#). Der kan du tegne DAGer interaktivt og få hjelp til å identifisere hvilke variable du bør kontrollere for.

19.6 Trusler mot validitet

Validitet handler om hvorvidt vi faktisk måler og konkluderer det vi tror vi gjør. Det er vanlig å skille mellom:

19.6.1 Intern validitet

Intern validitet handler om hvorvidt den observerte sammenhengen mellom X og Y faktisk reflekterer en kausal effekt. Trusler mot intern validitet inkluderer:

- **Confounding:** Uobserverte variable som påvirker både X og Y
- **Omvendt kausalitet:** Y påvirker X, ikke bare omvendt
- **Seleksjonsskjevhet:** Systematiske forskjeller mellom grupper
- **Målefeil:** Variablene måler ikke det vi tror de måler

19.6.2 Ekstern validitet

Ekstern validitet handler om hvorvidt resultatene kan generaliseres til andre kontekster. Selv om en effekt er godt identifisert i én studie, er det ikke sikkert den gjelder i en annen kontekst, tidsperiode eller populasjon.

19.7 Kvasi-eksperimentelle design

I mange tilfeller kan vi komme nærmere kausale konklusjoner selv med observasjonsdata ved å bruke smarte forskningsdesign. Dette er et stort felt, men her nevnes bare kort noen av de viktigste tilnærmingene:

- **Difference-in-differences (DiD):** Sammenligner endringer over tid mellom en behandlingsgruppe og en kontrollgruppe. Krever antakelse om parallele trender.
- **Regression discontinuity (RDD):** Utnytter at en behandling tildeles basert på en terskelverdi (f.eks. aldersgrenser, poenggrenser).
- **Instrumentvariabler (IV):** Bruker en ekstern variabel som påvirker X, men ikke Y direkte, for å estimere kausal effekt.
- **Matching:** Sammenligner individer som er like på observerbare kjennetegn, men forskjellige på behandlingsvariablene.

Disse metodene går langt utover denne boken, men det er nyttig å vite at de finnes. Hvis du skal gjøre kausale analyser med observasjonsdata, bør du sette deg inn i minst én av disse.

19.8 Praktiske råd

Her er noen konkrete råd for å tenke godt om forskningsdesign i din egen analyse:

1. **Vær eksplisitt om antakelser:** Skriv ned hva du antar om kausale sammenhenger. Tegn gjerne en DAG.

2. **Tenk over confounders:** Hva kan påvirke *både* X og Y? Har du kontrollert for det?
3. **Ikke overfortolke:** En statistisk signifikant sammenheng er ikke det samme som en kausal effekt.
4. **Diskuter begrensninger:** Enhver analyse har begrensninger. Det er bedre å diskutere dem åpent enn å ignorere dem.
5. **Les andres forskning:** Se hvordan andre har håndtert lignende problemstillinger. Hvilke metoder har de brukt? Hvilke kontrollvariable?
6. **Bruk teori aktivt:** La faglig teori veilede valgene dine, ikke bare statistiske kriterier.

19.9 Oppsummering

- Observasjonsdata gir sjeldent grunnlag for å trekke kausale konklusjoner direkte
- Korrelasjon skyldes ikke nødvendigvis kausalitet - confounding er en hovedutfordring
- Teori er nødvendig for å velge variable, spesifisere modeller og tolke resultater
- DAGer er et nyttig verktøy for å tenke eksplisitt om kausale sammenhenger
- Det finnes kvasi-eksperimentelle metoder som kan styrke kausale slutninger
- God dataanalyse krever faglig forståelse - R er verktøyet, men du må bruke det klokt

20 Reproduserbarhet

Forskning handler om å finne ut noe om verden, og det er viktig at andre kan etterprøve det vi har gjort. Reproduserbarhet betyr ganske enkelt at noen andre skal kunne ta dine data og din kode, kjøre analysen på nytt, og få nøyaktig samme resultat. Det høres kanskje selvfølgelig ut, men det er overraskende vanskelig i praksis.

En del av grunnen til at vi bruker R og skriver kode er nettopp dette: alt vi gjør er dokumentert i et script. Det er ingen skjulte klikk i menyer som ingen husker etterpå. Men det kreves litt disiplin for å gjøre det ordentlig. Dette kapittelet handler om praktiske grep du kan ta for å gjøre analysene dine reproducerbare.

20.1 Replikasjonskrisen

I løpet av de siste årene har det blitt avdekket at overraskende mange forskningsresultater ikke lar seg reproduksere. Dette omtales ofte som “replikasjonskrisen” og har særlig rammet psykologi og andre samfunnsvitenskapelige fag. Studier som forsøkte å gjenta klassiske eksperimenter fant at bare omtrent halvparten ga tilsvarende resultater som det opprinnelige studiet. Årsakene er sammensatte og handler om mye mer enn kode og data, men en viktig del av løsningen er åpenhet om hvordan analyser er gjennomført. Det starter med at analysekoden er tilgjengelig og at andre faktisk kan kjøre den.

20.2 Bruk script – ikke pek-og-klikk

Det mest grunnleggende grepene for reproducibilitet er å gjøre *alt* i et script. Alt du gjør i analysen skal stå i koden. Du skal ikke gjøre noe manuelt som ikke er dokumentert. Det betyr for eksempel at du ikke skal sortere data i Excel, ikke redigere datafiler for hånd, og ikke kopiere tall fra R-output og lime inn i et Word-dokument manuelt.

Grunnen er enkel: hvis du gjør noe manuelt, så er det ingen garanti for at du (eller noen andre) kan gjøre nøyaktig det samme om igjen. Og du vil garantert glemme hva du gjorde etter noen måneder.

20.3 R-prosjekter og filstier

En av de vanligste kildene til problemer med reproducertbarhet er filstier. Hvis koden din inneholder noe slikt:

```
data <- read.csv("C:/Users/torbjorn/Mine dokumenter/masteroppgave/data/survey.csv")
```

...så vil denne koden bare fungere på akkurat din datamaskin. Ingen andre har den samme mappestien. Løsningen er å bruke **R-prosjekter** (.Rproj-filer) og **relative filstier**.

Når du åpner et R-prosjekt i RStudio, settes arbeidskatalogen automatisk til mappen der .Rproj-filen ligger. Da kan du skrive:

```
data <- read.csv("data/survey.csv")
```

Dette fungerer for alle som har prosjektmappen, uansett hvor på datamaskinen den ligger. Opprett et nytt prosjekt via *File > New Project* i RStudio.

20.3.1 Aldri bruk setwd()

Du har kanskje sett at noen bruker `setwd()` for å sette arbeidskatalogen. Ikke gjør det. Problemet er det samme som med absolutte filstier: `setwd("C:/Users/torbjorn/prosjekt")` fungerer bare på din maskin. I tillegg gjør det at koden blir avhengig av *rekkefølgen* ting kjøres i, noe som gjør det vanskeligere å feilsøke. Bruk R-prosjekter i stedet, så slipper du `setwd()` helt.

20.3.2 here-pakken for filstier

Selv med R-prosjekter kan det noen ganger være litt klønnete med filstier, særlig i Quarto-dokumenter der arbeidskatalogen er mappen der .qmd-filen ligger, ikke prosjektets rotmappe. Pakken `here` løser dette elegant:

```
library(here)
data <- read.csv(here("data", "survey.csv"))
```

Funksjonen `here()` finner alltid prosjektets rotmappe (der .Rproj-filen ligger) og bygger filstien derfra. Det fungerer uansett hvor i prosjektmappen scriptet ditt befinner seg.

20.4 Kommenter koden din

God kode er kode som andre kan lese og forstå. Og med “andre” mener vi også deg selv om seks måneder. Bruk kommentarer (#) for å forklare *hvorfor* du gjør noe, ikke bare *hva* du gjør. R ignorerer alt som står etter # på en linje.

```
# Fjerner observasjoner med manglende inntektsdata  
# fordi disse er kodet som -1 i rådataene  
data <- data %>%  
  filter(inntekt >= 0)
```

Du trenger ikke kommentere hver eneste linje, men de stegene der det ikke er opplagt hva som skjer eller *hvorfor* du gjør det, bør ha en kommentar.

20.5 Quarto og R Markdown for rapporter

Denne boken er skrevet i Quarto, og det er et godt eksempel på hvordan man kan kombinere tekst og kode i ett dokument. Når du skriver en oppgave eller rapport i Quarto, ligger all kode og alle resultater i samme fil. Det betyr at du aldri trenger å kopiere tall fra R-output og lime inn i Word. Resultatene genereres direkte fra koden, og hvis data endres, oppdateres alt automatisk.

For semesteroppgaver og masteroppgaver er det veldig lurt å skrive i Quarto. Du slipper mye rot med å holde styr på hvilke tall som hører til hvilken analyse, og du unngår feil som oppstår ved manuell kopiering av resultater.

20.6 Versjonskontroll med Git

Har du noen gang hatt filer som heter `analyse_v2.R`, `analyse_v3_endelig.R`, `analyse_v3_endelig_VIRKELIG.R`. Da trenger du versjonskontroll.

Git er et system for å holde styr på endringer i filer over tid. Hver gang du gjør en meningsfull endring, lager du et “commit” – et slags øyeblikksbilde av filene dine. Du kan alltid gå tilbake til en tidligere versjon, og du kan se nøyaktig hva som ble endret og når.

Git brukes mest via kommandolinjen, men RStudio har innebygd støtte for Git slik at du kan gjøre de vanligste operasjonene direkte i RStudio. For å komme i gang med Git trenger du å installere det og koble det til RStudio. Det er litt oppsett første gangen, men når det er på plass er det veldig nyttig. For de fleste studentprosjekter er det nok å lære seg de grunnleggende operasjonene: `commit`, `push` og `pull`.

Vi går ikke i detalj om Git her, men det er verdt å vite at det finnes og at det er standard verktøy for versjonskontroll i de fleste fagmiljøer.

20.7 Pakkeversjoner med `renv`

R-pakker oppdateres hele tiden, og noen ganger kan en ny versjon av en pakke gi andre resultater enn den gamle. For å sikre at analysen din gir samme resultat om et år, kan du bruke pakken `renv` til å “fryse” pakkene i prosjektet ditt.

```
# Initialiser renv i prosjektet
renv::init()

# Etter at du har installert alle pakkene du trenger:
renv::snapshot()
```

Kommandoen `renv::snapshot()` lager en fil (`renv.lock`) som inneholder en oversikt over alle pakkene du bruker og nøyaktig hvilke versjoner som er installert. Hvis noen andre skal kjøre koden din, kan de bruke `renv::restore()` for å installere nøyaktig de samme versjonene.

For en masteroppgave eller et forskningsprosjekt er dette et veldig godt grep. For vanlige semesteroppgaver er det kanskje ikke strengt nødvendig, men det er greit å vite at muligheten finnes.

20.8 Dele data og kode

Reproduserbarhet handler også om at andre faktisk har *tilgang* til data og kode. To vanlige plattformer for dette er:

- **GitHub:** Et nettsted for å dele kode og samarbeide om prosjekter. GitHub er tett integrert med Git og er standard plattform i mange fagmiljøer. Du kan opprette en konto gratis på github.com.
- **Open Science Framework (OSF):** En plattform laget spesifikt for forskning, der du kan dele data, kode, og annen dokumentasjon. OSF er mye brukt i samfunnsvitenskap og er tilgjengelig på osf.io.

For en masteroppgave kan det være veldig nyttig å legge kode og data (hvis dataene kan deles) på en av disse plattformene. Det viser at du tar reproducertbarhet på alvor, og det gjør det lett for veileder å se på koden din.

20.9 Dokumenter R-miljøet ditt

En enkel ting du kan gjøre helt til slutt i ethvert prosjekt er å dokumentere hvilken versjon av R og hvilke pakker du har brukt. Funksjonen `sessionInfo()` gir deg alt du trenger:

```
sessionInfo()
```

Dette skriver ut R-versjon, operativsystem, og alle pakkene som er lastet inn med versjonsnumre. Hvis noen har problemer med å reproduksjonere resultatene dine, er dette det første stedet å se etter forskjeller. Legg gjerne til `sessionInfo()` helt sist i Quarto-dokumentet ditt.

20.10 Sjekkliste for reproducerte prosjekter

Her er en praktisk sjekkliste du kan bruke for å sjekke at prosjektet ditt er rimelig reproducerebart:

- All analyse er gjort i script (ingen manuelle steg)
- Prosjektet bruker en `.Rproj`-fil
- Alle filstier er relative (ingen `setwd()`, ingen absolutte stier)
- Koden er kommentert der det trengs
- Rapporten er skrevet i Quarto eller R Markdown
- `sessionInfo()` er inkludert i dokumentet
- Data og kode er organisert i en logisk mappestruktur
- Prosjektet kan kjøres fra start til slutt uten feil på en annen maskin

Du trenger ikke krysse av alt for en vanlig semesteroppgave, men jo flere punkter du får med, desto bedre. For en masteroppgave bør du sikte på å få med alt.

Part VI

Del VI: Importere og utforske data

21 Innlesning av data

Vi skal bruke følgende pakker i dette kapittelet

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

21.1 Generelt om ulike dataformat

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.

21.1.1 rds

Rds-formatet er et format særlig egnet for R. Her er et eksempel med et utdrag fra datasettet `wagepan` (paneldata om lønn fra pakken `wooldridge`)

```
wagepan_rds <- readRDS("data/wagepan_eksempel.rds")
glimpse(wagepan_rds)
```

```
Rows: 4,360
Columns: 10
$ nr      <int> 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~
$ year    <int> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
$ hours   <int> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~
$ educ    <int> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13~
$ black   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ hisp    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ married <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~
$ union   <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ exper   <int> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

21.1.2 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.

Her bruker man `load` som laster dette speilbildet og objektet med dataene i beholder det navnet de hadde da de ble laget. Se i fanen “Environment” i Rstudio om det har dukket opp noe nytt der, for å finne navnet hvis du ikke vet det fra før. I dette eksempelet er dataene lagret i et objekt som “wagepan_eksempel_rdata” som altså er lagret i en fil som heter “wagepan_eksempel.Rdata”. Ved lasting av filen dukker objektet opp under “Enviroment”-fanen, men du får ikke noen melding av noe slag. Men er altså tilgjengelig i minnet i R. Her er koden:

```
load("data/wagepan_eksempel.Rdata")
glimpse(wagepan_eksempel_rdata)
```

```
Rows: 4,360
Columns: 10
$ nr      <int> 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~
$ year    <int> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
```

```
$ hours    <int> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~  
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~  
$ educ     <int> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13~  
$ black    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ hisp     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ married   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~  
$ union    <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ exper    <int> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

21.1.3 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. Hvis du får feilmeldinger og det ser skikkelig rart ut, så åpen filen i Notepad (eller annet ren-tekst program) og sjekk. I koden nedenfor er det spesifisert komma som skilletegn, men hvis det er semikolon endrer du det til `sep =", "`. I utgangspunktet forventer `read.csv` at det er kommaseparert, så koden vil funke her uten den delen.

```
wagepan_eksempel_csv <- read.csv("data/wagepan_eksempel.csv", sep =",")  
glimpse(wagepan_eksempel_csv)
```

```
Rows: 4,360  
Columns: 10  
$ nr          <int> 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~  
$ year        <int> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~  
$ hours       <int> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~  
$ lwage        <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~  
$ educ         <int> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13~  
$ black        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ hisp         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ married      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~  
$ union        <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ exper        <int> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

21.1.4 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. Det finnes også andre pakker for å håndtere Excel-filer, men hvis formålet bare er å lese inn data, så gjør denne pakken jobben. Husk å laste pakken først. Her er et eksempel:

```
library(readxl)
wagepan_xlsx <- read_excel("data/wagepan_eksempel.xlsx")
glimpse(wagepan_xlsx)
```

```
Rows: 4,360
Columns: 10
$ nr      <dbl> 13, 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~
$ year    <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
$ hours   <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~
$ educ    <dbl> 14, 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13~
$ black   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ hisp    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ married <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, ~
$ union   <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ exper   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

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.

21.1.5 Proprietære format: Stata, SPSS og SAS

21.1.5.1 Stata

```
wagepan_dta <- read_stata("data/wagepan_eksempel.dta")
glimpse(wagepan_dta)
```

```
Rows: 4,360  
Columns: 10  
$ nr      <dbl> 13, 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~  
$ year    <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~  
$ hours   <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~  
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~
```

```

$ educ      <dbl> 14, 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13~
$ black     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ hisp      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ married   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~
$ union     <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ exper     <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~

```

Legg merke til at den andre kolonnen her viser hva slags variabeltype det er. `<dbl>` betyr at det er numerisk variabel⁷(Det finnes flere typer numeriske variable som vi for praktiske analyser sjeldent 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 variablene. *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.

Neste kapittel går nærmere inn på å håndtere data fra Stata og SPSS, inkludert hvordan man effektivt gjør om labler. Hvordan dette gjøres i praksis 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 appendikset.

21.1.5.2 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:

```
wagepan_sav <- read_spss("data/wagepan_eksempel.sav")
glimpse(wagepan_sav)
```

Her er eksempel for innlesning av SAS-fil:

```
wagepan_sas <- read_sas("data/wagepan_eksempel.sas7bdat")
glimpse(wagepan_sas)
```

```
Rows: 4,360
Columns: 10
$ nr      <dbl> 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~
$ year    <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
$ hours   <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~
```

```
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~  
$ educ     <dbl> 14, 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13~  
$ black    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ hisp     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ married   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~  
$ union    <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  
$ exper    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

21.1.6 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 er uopraktisk 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.

22 Import fra Stata og SPSS

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

I forrige kapittel så vi kort hvordan man leser inn data fra Stata og SPSS. Det fungerer greit nok for de fleste formål, men det er noen utfordringer knyttet til *labelled* data som krever litt mer arbeid. Dette kapittelet går dypere inn i dette.

Hvis du jobber med data som opprinnelig er laget i Stata eller SPSS, vil du nesten garantert støte på det som kalles *labler*. Det er en spesiell måte å lagre informasjon om variablene på. R kan håndtere dette, men det er viktig å forstå hva som skjer og hvordan du bør rydde opp.

22.1 Hva er labelled data?

I Stata og SPSS er det vanlig å bruke to typer labler:

- **Value labels** (verdilabel): Knytter en tekst til en tallverdi. For eksempel kan verdien 0 ha labelen “Not married” og 1 ha labelen “Married”.
- **Variable labels** (variabellabel): En kort beskrivelse av hva variablen inneholder. For eksempel kan variabelen `married` ha variabellabelen “Sivilstatus”.

Når du leser inn en Stata- eller SPSS-fil med `haven`, beholdes denne informasjonen. Men R behandler den annerledes enn det Stata og SPSS gjør. I R er disse verdiene fortsatt tall, men med labler vedlagt som metadata.

22.2 Lese inn data

La oss starte med å lese inn en Stata-fil:

```
wagepan <- read_stata("data/wagepan_eksempel.dta")
glimpse(wagepan)
```

```

Rows: 4,360
Columns: 10
$ nr      <dbl> 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17~
$ year    <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
$ hours   <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~
$ educ    <dbl> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13~
$ black   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ hisp    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ married <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~
$ union   <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ exper   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~

```

Legg merke til variabeltypene. Der det står `<dbl+lbl>` betyr det at variabelen er numerisk (`dbl` = double), men at det finnes labler knyttet til verdiene. Dette er altså *labelled* variable.

22.3 Inspisere labler

Vi kan se på lablene til en enkelt variabel med `val_labels()`:

```
val_labels(wagepan$married)
```

NULL

Vi kan også se variabellabelen (beskrivelsen av variabelen) med `var_label()`:

```
var_label(wagepan$married)
```

[1] "Married"

For å få oversikt over alle variable med labler kan vi bruke `look_for()` fra `labelled`-pakken:

```
look_for(wagepan)
```

pos	variable	label	col_type	missing	values
1	nr	Person identifier	dbl	0	
2	year	Year	dbl	0	
3	hours	Annual hours worked	dbl	0	

```

4   lwage    Log hourly wage      dbl      0
5   educ     Years of education dbl      0
6   black    Black              dbl      0
7   hisp     Hispanic          dbl      0
8   married  Married           dbl      0
9   union    Union member      dbl      0
10  exper    Years of experience dbl      0

```

Denne funksjonen gir en ryddig oversikt over variabelnavn, variabellabler, verdier og verdilabel. Det er veldig nyttig for å utforske datasett fra Stata og SPSS.

22.4 Problemet med labelled data i R

Labelled data fungerer fint for å inspirere dataene, men de kan skape problemer når du skal gjøre analyser. Mange R-funksjoner forventer enten numeriske variable eller factor-variable, ikke labelled-variable. Du kan for eksempel oppleve at:

- `ggplot` viser tallverdier i stedet for meningsfulle kategorinavn
- Tabeller viser koder i stedet for tekst
- Noen funksjoner gir uventede resultater eller feilmeldinger

Derfor er det lurt å konvertere labelled variable til factor-variable tidlig i arbeidsprosessen.

22.5 Konvertere til factor

Den enkleste måten å konvertere en labelled variabel til factor er med `as_factor()` fra `haven`:

```
wagepan <- wagepan %>%
  mutate(married_factor = as_factor(married))

table(wagepan$married_factor)
```

0	1
2446	1914

Men hvis du har mange labelled variable, er det mye mer effektivt å konvertere alle på en gang med `across()`:

```
wagepan_clean <- wagepan %>%
  mutate(across(where(is.labelled), ~as_factor(.)))

glimpse(wagepan_clean)
```

```
Rows: 4,360
Columns: 11
$ nr          <dbl> 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, ~
$ year        <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1~
$ hours       <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2~
$ lwage        <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, ~
$ educ         <dbl> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, ~
$ black        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ hisp         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ married      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1~
$ union        <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ exper        <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, ~
$ married_factor <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1~
```

Denne ene linjen konverterer *alle* labelled variable i datasettet til factor-variable. Det er den anbefalte metoden.

Noen ganger vil du også fjerne ubrukte factor-nivåer som kan oppstå etter filtrering:

```
wagepan_clean <- wagepan %>%
  mutate(across(where(is.labelled), ~as_factor(.)),
        across(where(is.factor), ~fct_drop(.)))
```

22.6 Brukerdefinerte missing-verdier

I Stata og SPSS er det vanlig å bruke spesielle verdier for å markere ulike typer missing. For eksempel kan -9 bety “ikke besvart” og -8 bety “ikke relevant”. Disse verdiene er ofte dokumentert med labler.

Pakken `haven` håndterer dette med *tagged NA*-verdier. Du kan sjekke om det finnes slike verdier:

```
# Se brukerefinerte missing-verdier
na_values(wagepan$hours)
```

I praksis er det ofte enklest å bare konvertere til factor (som gjort ovenfor) og deretter filtrere ut de kategoriene du ikke trenger. Alternativt kan du bruke `zap_labels()` for å fjerne alle labler og beholde bare tallverdiene:

```
wagepan_tall <- wagepan %>%
  mutate(across(where(is.labelled), ~zap_labels(.)))
```

Men vær forsiktig med dette - da mister du all informasjon om hva verdiene betyr!

22.7 Lese inn SPSS-filer

Innlesning av SPSS-filer fungerer på tilsvarende måte:

```
wagepan_spss <- read_spss("data/wagepan_eksempel.sav")
glimpse(wagepan_spss)
```

Konvertering til factor gjøres på nøyaktig samme måte:

```
wagepan_spss_clean <- wagepan_spss %>%
  mutate(across(where(is.labelled), ~as_factor(.)),
        across(where(is.factor), ~fct_drop(.)))
```

22.8 Tegnsett og encoding

Et vanlig problem med nordiske data er tegnsettproblemer. Norske bokstaver (æ, ø, å) kan vises feil hvis filen er lagret med et annet tegnsett enn det R forventer. Hvis du ser rare tegn i stedet for æøå, kan du prøve å spesifisere tegnsett ved innlesning:

```
# Prøv med ulike tegnsett
wagepan <- read_stata("data/wagepan_eksempel.dta", encoding = "latin1")
```

De vanligste tegnsettene er "UTF-8" og "latin1" (også kalt ISO-8859-1). Prøv begge hvis du har problemer.

22.9 Anbefalt arbeidsflyt

Her er en anbefalt arbeidsflyt for å jobbe med data fra Stata eller SPSS:

1. **Les inn** datafilen med `read_stata()` eller `read_spss()`
2. **Inspiser** dataene med `glimpse()` og `look_for()`
3. **Konverter** labelled variable til factor med `across(where(is.labelled), ~as_factor(.))`
4. **Velg** de variablene du trenger med `select()`
5. **Lagre** den ryddede versjonen som `.rds` for videre bruk

```
# Komplett arbeidsflyt
wagepan_ferdig <- read_stata("data/wagepan_eksempel.dta") %>%
  mutate(across(where(is.labelled), ~as_factor(.)),
        across(where(is.factor), ~fct_drop(.))) %>%
  select(nr, year, hours, lwage, educ, married, union)

# Lagre til .rds for videre bruk
saveRDS(wagepan_ferdig, "data/wagepan_ferdig.rds")
```

Fordelen med å lagre som `.rds` er at du slipper å gjøre denne konverteringen hver gang. Neste gang du trenger dataene kan du bare laste inn `.rds`-filen direkte.

22.10 Oppsummering

- Data fra Stata og SPSS har labler som R behandler som *labelled* data
- Konverter til factor med `mutate(across(where(is.labelled), ~as_factor(.)))`
- Bruk `look_for()` for å utforske variabelnavn og labler
- Lagre ryddede data som `.rds` for effektiv videre bruk
- Pass på tegnsettproblemer med nordiske bokstaver

23 Import av Excel-filer

```
library(tidyverse)
library(readxl)
```

Forbløffende mye data finnes i Excel-format. Selv om Excel ikke er designet for statistisk analyse, er det slik at svært mange organisasjoner, kommuner og offentlige etater lagrer og distribuerer data i Excel. Du vil nesten garantert støte på Excel-filer i løpet av studiene eller i arbeidslivet, så det er greit å vite hvordan du håndterer dem i R.

Excel-filer kan være litt mer kronglete enn csv-filer fordi de kan inneholde flere ark, sammenslitte celler, formatering og formler. Men R har gode verktøy for å håndtere dette.

23.1 Lese inn Excel-filer med `readxl`

Pakken `readxl` er den anbefalte pakken for å lese inn Excel-filer i R. Den er en del av tidyverse-familien og håndterer både `.xls` (gammelt format) og `.xlsx` (nytt format). Den enkleste bruken er helt rett frem:

```
wagepan_xlsx <- read_excel("data/wagepan_eksempel.xlsx")
glimpse(wagepan_xlsx)
```

```
Rows: 4,360
Columns: 10
$ nr      <dbl> 13, 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~
$ year    <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
$ hours   <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~
$ educ    <dbl> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13~
$ black   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ hisp    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ married <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~
$ union   <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ exper   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

Funksjonen `read_excel()` gjetter selv om det er `.xls` eller `.xlsx` basert på filendelsen. Du kan også bruke `read_xlsx()` eller `read.xls()` direkte hvis du vil være eksplisitt.

23.2 Velge ark med sheet

En Excel-fil kan inneholde flere ark (sheets). Som standard leser `read_excel()` inn det første arket. For å se hvilke ark som finnes i en fil kan du bruke `excel_sheets()`:

```
excel_sheets("data/wagepan_eksempel.xlsx")
```

```
[1] "Sheet 1"
```

Du kan velge ark enten med navn eller nummer:

```
# Med navn
df <- read_excel("data/wagepan_eksempel.xlsx", sheet = "Sheet1")

# Med nummer (første ark)
df <- read_excel("data/wagepan_eksempel.xlsx", sheet = 1)
```

Hvis du trenger data fra flere ark kan du lese dem inn hver for seg og eventuelt koble dem sammen etterpå.

23.3 Håndtere overskriftsader og hoppe over rader

I praksis ser Excel-filer sjeldne så ryddige ut som man skulle ønske. Det er vanlig at de første radene inneholder titler eller merknader som ikke er en del av dataene. Med `skip` hopper du over et gitt antall rader fra toppen:

```
# Hopp over de 3 første radene
df <- read_excel("data/wagepan_eksempel.xlsx", skip = 3)
```

Noen ganger har ikke filen variabelnavn i det hele tatt. Da kan du bruke `col_names`:

```
# Ingen overskriftsrad - R lager egne navn (X1, X2, ...)
df <- read_excel("data/wagepan_eksempel.xlsx", col_names = FALSE)

# Sett egne variabelnavn
df <- read_excel("data/wagepan_eksempel.xlsx",
                  col_names = c("id", "alder", "kjonn", "inntekt"))
```

23.4 Spesifisere celleområde med range

Hvis dataene bare ligger i en del av regnearket, kan du spesifisere nøyaktig hvilke celler som skal leses inn.

```
# Les bare cellene B2 til F100
df <- read_excel("data/wagepan_eksempel.xlsx", range = "B2:F100")

# Du kan også spesifisere ark og celleområde samtidig
df <- read_excel("data/wagepan_eksempel.xlsx", range = "Sheet1!B2:F100")
```

Du kan også bruke `cell_rows()` og `cell_cols()` for å spesifisere bare rader eller kolonner:

```
# Bare radene 5 til 50
df <- read_excel("data/wagepan_eksempel.xlsx", range = cell_rows(5:50))

# Bare kolonnene A til D
df <- read_excel("data/wagepan_eksempel.xlsx", range = cell_cols("A:D"))
```

23.5 Vanlige problemer med Excel-filer

23.5.1 Datoer

Excel lagrer datoer som tall internt, og det kan noen ganger bli krøll ved innlesning. Hvis datoene ser rare ut (f.eks. som femsifrede tall), kan du konvertere dem:

```
# Excel bruker 1899-12-30 som nullpunkt
excel_tall <- 44927
as.Date(excel_tall, origin = "1899-12-30")
```

```
[1] "2023-01-01"
```

Som regel håndterer `readxl` datoer automatisk, men det er greit å vite om dette.

23.5.2 Sammenslåtte celler

Sammenslåtte celler er en gjenganger. Ved innlesning får bare den første cellen verdien, resten blir NA. Løsningen er å fylle nedover med `tidy::fill()`:

```
# Eksempel: Fyll NA-verdier nedover (typisk etter sammenslåtte celler)
demo <- tibble(
  kategori = c("Menn", NA, NA, "Kvinner", NA, NA),
  alder = c("18-29", "30-49", "50+", "18-29", "30-49", "50+"),
  antall = c(120, 340, 280, 135, 360, 295)
)

demo %>%
  fill(kategori, .direction = "down")
```

```
# A tibble: 6 x 3
  kategori alder antall
  <chr>    <chr>   <dbl>
1 Menn     18-29     120
2 Menn     30-49     340
3 Menn     50+      280
4 Kvinner  18-29     135
5 Kvinner  30-49     360
6 Kvinner  50+      295
```

23.5.3 Flere tabeller i samme ark

Noen bruker ett Excel-ark til å plassere flere tabeller ved siden av hverandre eller under hverandre. Da er `range`-argumentet din beste venn. Les inn hver tabell for seg med hvert sitt celleområde.

23.5.4 Tekst i tallkolonner

Hvis noen har skrevet en kommentar eller merknad i en celle som ellers inneholder tall, vil hele kolonnen kunne bli lest inn som tekst. Sjekk variabeltypene med `glimpse()` og konverter om nødvendig med `as.numeric()`.

23.6 Skrive Excel-filer

Noen ganger trenger du å lagre data som Excel-fil, f.eks. fordi noen du samarbeider med foretrekker det. Pakken `openxlsx` er et godt alternativ:

```

library(openxlsx)

# Enkel eksport
write.xlsx(wagepan_xlsx, file = "data/eksempel_ut.xlsx")

# Med flere ark i samme fil
write.xlsx(list("Datasett1" = wagepan_xlsx,
                "Oppsummering" = summary(wagepan_xlsx) %>% as.data.frame()),
           file = "data/eksempel_flere_ark.xlsx")

```

Et enklere alternativ er pakken `writexl` som ikke har noen avhengigheter til Java eller andre systemer:

```

library(writexl)
write_xlsx(wagepan_xlsx, path = "data/eksempel_ut.xlsx")

```

Begge fungerer fint for enkel eksport. `openxlsx` gir deg mer kontroll over formatering, mens `writexl` er mer lettvektig og enklere å installere.

23.7 Tips for rotete Excel-filer

I virkeligheten er Excel-filer ofte rotete. Her er noen praktiske tips:

1. **Åpne filen i Excel først** og se på strukturen. Legg merke til hvilke rader og kolonner dataene faktisk ligger i.
2. **Bruk `excel_sheets()`** for å se hvilke ark som finnes.
3. **Start med `range`** hvis dataene ikke begynner i celle A1.
4. **Sjekk variabeltypene** med `glimpse()` rett etter innlesning. Hvis en tallkolonne er blitt tekst, er det gjerne en kommentar eller et spesialtegn som lurer seg inn.
5. **Bruk `fill()`** etter innlesning dersom det har vært sammenslattet celler.
6. **Unngå å redigere Excel-filen manuelt** for å “rydde” den. Skriv heller R-kode som håndterer rotet. Da er arbeidet reproducerbart og du slipper å gjøre det om igjen neste gang du får oppdaterte data.

Det siste punktet er kanskje det viktigste. Det er fristende å fikse ting manuelt i Excel, men da mister du sporbarhet. Gjør du alt i R-kode, kan du og andre kjøre koden på nytt og få nøyaktig samme resultat.

24 Data fra SSBs statistikkbank

All statistikk fra SSB publiseres (også) i en statistikkbank der man kan hente ut f.eks. tidsserier over flere årganger. Du kan ta en titt på Statistikkbanken på [SSB sine sider](#).

SSB har laget et r-pakke for å hente data direkte fra SSBs statistikkbank. Denne kan du installere direkte fra CRAN via R slik:

```
install.packages("PxWebApiData")
```

Vi skal altså bruke følgende pakker i dette kapittelet:

```
library(PxWebApiData)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.2.0     v readr     2.1.5
v forcats   1.0.1     v stringr   1.6.0
v ggplot2   4.0.2     v tibble    3.3.1
v lubridate  1.9.4     v tidyr    1.3.1
v purrr    1.2.1
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()   masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become non-conflicting
```

```
# Befolknings
meta<- ApiData("http://data.ssb.no/api/v0/en/table/07459", returnMetaFrames = TRUE)
names(meta)
```

```
[1] "Region"        "Kjonn"         "Alder"          "ContentsCode"  "Tid"
```

```
meta$Kjonn
```

```

values valueTexts
1      2    Females
2      1    Males

## Anmeldte lovbrudd
meta<- ApiData("http://data.ssb.no/api/v0/en/table/08487",  returnMetaFrames = TRUE)

kommuner <- meta$Gjerningssted %>%
  filter(nchar(values) == 4 & !(values %in% c("Ialt", "0000"))) %>%
  pull(values)

grupper <- meta$LovbruddKrim %>%
  filter(nchar(values) > 5 & valueTexts != "All groups of offences" ) %>%
  pull(values)

## Pull data from API
anm_list <- ApiData("http://data.ssb.no/api/v0/en/table/08487",
                      ContentsCode = "AnmLovbrPer1000",
                      Gjerningssted = kommuner,
                      LovbruddKrim = grupper,
                      Tid = TRUE,
                      makeNAstatus = FALSE)

# rename
names(anm_list[[1]]) <- c("kommune", "lovbruddsgruppe", "content", "year", "lovbrudd_per1000")

# Combine and tidy up
anm <- cbind(anm_list[[2]][1], anm_list[[1]]) %>%
  select(-content, -NAstatus) %>%
  mutate(lovbruddsgruppe = str_sub(lovbruddsgruppe, 3, nchar(lovbruddsgruppe))) %>%
  mutate(year = as.numeric(str_sub(year, 1, 4))) %>%
  filter(!is.na(lovbrudd_per1000)) %>%
  mutate(lovbruddsgruppe = case_when(str_sub(lovbruddsgruppe, 1, 4) == "Prop" ~ "vinningskrimi",
                                     str_sub(lovbruddsgruppe, 1, 4) == "Viol" ~ "voldskriminalitet",
                                     str_sub(lovbruddsgruppe, 1, 4) == "Drug" ~ "nark_alko_krimi",
                                     str_sub(lovbruddsgruppe, 1, 4) == "Publ" ~ "ordenslovbrudd",
                                     str_sub(lovbruddsgruppe, 1, 4) == "Traf" ~ "trafikklovbrudd",
                                     str_sub(lovbruddsgruppe, 1, 4) == "Othe" ~ "andre_lovbrudd")
  # pivot_wider(values_from = lovbrudd_per1000, names_from = lovbruddsgruppe) %>%
  # rename(kommune_nr = Gjerningssted) %>%

```

```
select(-kommune)
```

```
glimpse(anm)
```

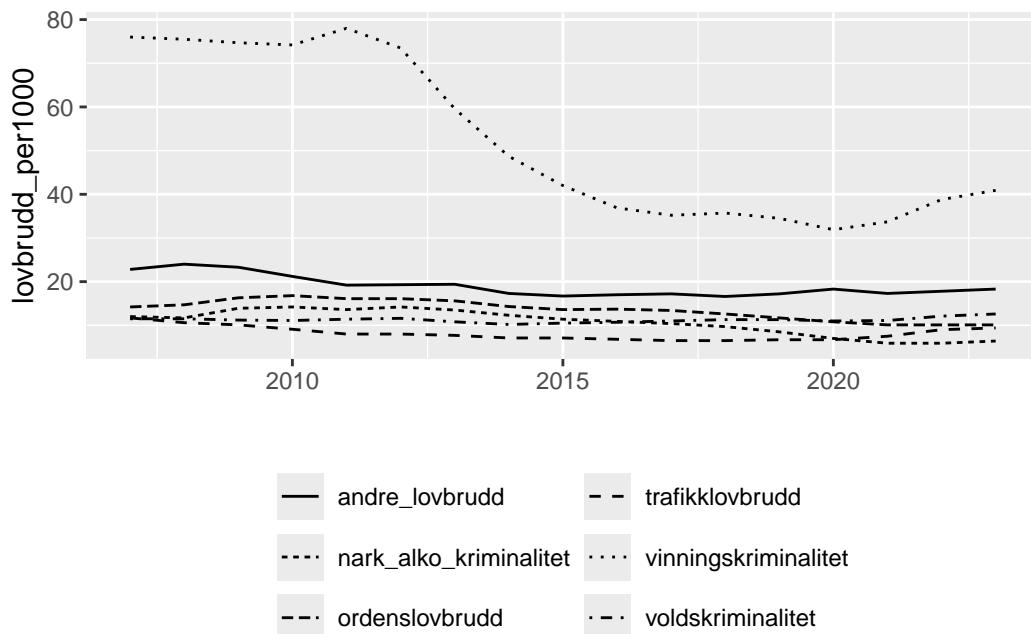
Rows: 36,866

Columns: 4

```
$ kommune_nr      <chr> "3101", "3101", "3101", "3101", "3101", "3101", "3103~  
$ lovbruddsgruppe <chr> "vinningskriminalitet", "voldskriminalitet", "nark_alk~  
$ year           <dbl> 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, ~  
$ lovbrudd_per1000 <dbl> 12.6, 11.6, 16.8, 13.2, 13.8, 18.0, 21.8, 7.4, 6.3, 5~
```

```
#Check
```

```
ggplot( filter(anm, kommune_nr == "0301"),  
       aes(x = year, y = lovbrudd_per1000, group = lovbruddsgruppe, linetype = lovbruddsgrup~  
         geom_line() +  
         theme(legend.position = "bottom", legend.title = element_blank()) +  
         guides(linetype = guide_legend(ncol = 2)) +  
         xlab("") )
```



```
# meta$Alder  
# meta$Kjonn
```

```

# meta$ContentsCode
# meta$Region %>% head()

kommuner_bef <- meta$Region %>%
  filter(nchar(values) == 4 ) %>%
  pull(values)
head(kommuner_bef)

befolk <- list(length(meta$Tid$values))
for(i in 1:length(meta$Tid$values)){
  #print(i)
  bef_list <- ApiData("http://data.ssb.no/api/v0/en/table/07459",
                       ContentsCode = TRUE,
                       Region = kommuner_bef,
                       Kjonn = TRUE,
                       Alder = TRUE,
                       Tid = i)
  befolk[[i]] <- cbind(bef_list[[2]][1], bef_list[[1]])

  #rm(bef_list)
}

befolkning <- bind_rows(befolk)

# rename
names(befolkning) <- c("kommune_nr", "kommune", "kjonn", "alder", "contents", "year", "bef_an")

# Combine and tidy up
befolkning2 <- befolkning %>%
  select(-contents) %>%
  mutate(year = as.numeric(year)) %>%
  mutate(age = str_extract(alder, "(\\d+)")) %>%
  mutate(age_gr = case_when(age < 18 ~ "under 18",
                            age <= 25 ~ "18-25",
                            age <= 35 ~ "26-35",
                            age <= 67 ~ "36-67",
                            age > 67 ~ "over 67"))

bef_gruppe <- befolkning2 %>%
  group_by(kommune_nr, kommune, year, age_gr, kjonn) %>%

```

```

summarise(n = sum(bef_antall)) %>%
ungroup() %>%
pivot_wider(values_from = n, names_from = age_gr) %>%
rename(bef_18_25 = `18-25`, bef_26_35 = `26-35`, bef_36_67 = `36-67`,
       bef_67plus = `over 67`, bef_18min = `under 18`) %>%
filter(year >= 2010)

bef_menn <- filter(bef_gruppe, kjonn == "Males") %>%
  rename_all(str_replace_all, "bef_", "menn_") %>%
  select(-kjonn, -kommune)

bef_kvinner <- filter(bef_gruppe, kjonn == "Females") %>%
  rename_all(str_replace_all, "bef_", "kvinner_") %>%
  select(-kjonn, -kommune)

head(bef_kvinner)

bef_tot <- befolkning2 %>%
  group_by(kommune_nr, kommune, year, age_gr) %>%
  summarise(n = sum(bef_antall)) %>%
  ungroup() %>%
  pivot_wider(values_from = n, names_from = age_gr) %>%
  rename(bef_18_25 = `18-25`, bef_26_35 = `26-35`, bef_36_67 = `36-67`,
         bef_67plus = `over 67`, bef_18min = `under 18`) %>%
  filter(year >= 2010) %>%
  rowwise() %>%
  mutate(bef_totalt = sum(across(bef_18_25:bef_18min))) %>%
  select(1:3, 8, 4, 5, 9)

glimpse(bef_tot)

```

```

ggplot(bef, aes(x = year, y = value, group = region, col = region)) +
  geom_line()

(ApiData("http://data.ssb.no/api/v0/no/table/07459", returnMetaFrames = TRUE))

```

```

# bef_agg <- befolkning2 %>%
#   group_by( year, age_gr) %>%
#   summarise(n = sum(bef_antall)) %>%
#   ungroup()
#
# glimpse(bef_agg)
#
# ggplot(bef_agg, aes(x = year, y = n, linetype = age_gr, group = age_gr)) +
#   geom_line() +
#   ylim(0,NA)
#
## Inntekt

meta<- ApiData("http://data.ssb.no/api/v0/en/table/06944", returnMetaFrames = TRUE)
names(meta)
meta$Tid
meta$HusholdType
meta$ContentsCode

kommuner_innt <- meta$Region %>%
  filter(nchar(values) == 4 ) %>%
  pull(values)
head(kommuner_innt)

innt_list <- ApiData("http://data.ssb.no/api/v0/en/table/06944",
                      ContentsCode = TRUE,
                      Region = kommuner_innt,
                      HusholdType = TRUE,
                      Tid = TRUE)

glimpse(innt_list[[1]])

inntekt <- cbind(innt_list[[2]][1], innt_list[[1]]) %>%
  mutate(year = as.numeric(year)) %>%
  select(-NAstatus) %>%
  filter(!is.na(value)) %>%
  rename(kommune = region, kommune_nr = Region,
         hushold = `type of household`) %>%

```

```

  mutate(contents = case_when(str_sub(contents,1,5) == "Total" ~ "inntekt_totalt_median",
                               str_sub(contents,1,6) == "Income" ~ "inntekt_eskatt_median",
                               str_sub(contents,1,6) == "Number" ~ "ant_husholdninger")) %>%
  pivot_wider(values_from = value, names_from = contents)

glimpse(inntekt)

hh_inntekt <- filter(inntekt, hushold == "All households") %>%
  select(-hushold, -kommune)
head(hh_inntekt)

## SOSIALHJELP

meta<- ApiData("http://data.ssb.no/api/v0/en/table/12210", returnMetaFrames = TRUE)
names(meta)
meta$ContentsCode
kokk <- meta$KOKkommuneregion0000 %>%
  filter(!(values %in% c("EAK", "EAKUO"))) ) %>%
  pull(values)

glimpse(kokk)

shj_list <- ApiData("http://data.ssb.no/api/v0/en/table/12210",
                     ContentsCode = TRUE,
                     KOKkommuneregion0000 = kokk,
                     Tid = TRUE)

shj <- cbind(shj_list[[2]], shj_list[[1]][1]) %>%
  filter(!is.na(value)) %>%
  pivot_wider(values_from = value, names_from = ContentsCode) %>%
  rename(kommune_nr = KOKkommuneregion0000,
         year = Tid,
         shj_klienter = KOSsosantkliente0000 ,
         shj_unge = KOSant18240000) %>%

```

```

select(kommune_nr, year, shj_klienter, shj_unge) %>%
  mutate(year = as.numeric(year))

glimpse(shj)

## Sample

samlet <- left_join(bef_tot, bef_menn, by = c("kommune_nr", "year")) %>%
  left_join(bef_kvinner, by = c("kommune_nr", "year")) %>%
  left_join(hh_inntekt, by = c("kommune_nr", "year")) %>%
  left_join( shj, by = c("kommune_nr", "year")) %>%
  left_join( anm, by = c("kommune_nr", "year")) %>%
  filter(year >= 2015) %>%
  data.frame() %>%
  filter(complete.cases(.))

glimpse(samlet)

saveRDS(samlet , "data/kommunedata.rds")

```

25 Få oversikt over datasettet

Når man jobber med datasett kan det være mange variable, og det er viktig å ha en oversikt over datasettet. Bare det å finne riktig variabel kan være en utfordring i større datasett. Det vil normalt følge med et dokumentasjonsnotat eller -rapport med oversikt over alle variable. Ofte vil 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 bruker vi datasettet `wagepan` som er paneldata om lønn fra pakken `wooldridge`. Datasettet inneholder informasjon om arbeidstakere over flere år.

25.1 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 `wagepan`:

```
class(wagepan)
```



```
[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(wagepan)
```

```
[1] 4360 10
```

Her får vi vite at det er 4360 rader (dvs. observasjoner) og 10 kollonner (dvs. variable).

25.1.1 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(wagepan)
```

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.

25.1.2 Bruke head()

Funksjonen `head()` skriver de første 6 observasjonene 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 mange variable er ikke dette veldig nyttig, men for datasett med noen få variable fungerer det greit.

```
head(wagepan)
```

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.

25.1.3 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(wagepan[, 1:5])
```

```
# A tibble: 6 x 5
  nr   year hours lwage  educ
  <dbl> <dbl> <dbl> <dbl> <dbl>
1    13  1980  2672  1.20    14
2    13  1981  2320  1.85    14
3    13  1982  2940  1.34    14
4    13  1983  2960  1.43    14
5    13  1984  3071  1.57    14
6    13  1985  2864  1.70    14
```

Vi kan altså også angi både rader og kollonner på denne måten. Her er eksempel som viser første 3 rader og variabelnummer 5 til 8.

```
head(wagepan[1:3, 5:8])
```

```
# A tibble: 3 x 4
  educ black hisp married
  <dbl> <dbl> <dbl>   <dbl>
1    14     0     0      0
2    14     0     0      0
3    14     0     0      0
```

Legg merke til at under hvert variabelnavn står det en liten tekst, f.eks. `<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.

25.1.4 Bruke ‘glimpse()’

I tidligere kurs skal dere ha lært å bruke funksjonen `glimpse()`. Her er et eksempel:

glimpse(wagepan)

```
Rows: 4,360
Columns: 10
$ nr      <dbl> 13, 13, 13, 13, 13, 13, 13, 13, 17, 17, 17, 17, 17, 17, 17, 17~
$ year    <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1980, 1981, 19~
$ hours   <dbl> 2672, 2320, 2940, 2960, 3071, 2864, 2994, 2640, 2484, 2804, 25~
$ lwage    <dbl> 1.1975402, 1.8530600, 1.3444617, 1.4332134, 1.5681251, 1.69989~
$ educ    <dbl> 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13~
$ black   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ hisp    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ married <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, ~
$ union   <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ exper   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 4, 5, 6, 7, ~
```

I denne output'en er den første kolumnen 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.

25.1.5 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(wagepan$married)
```

```
=====
```

```
wagepan$married 'Married'
```

```
Storage mode: double
```

```
Min: 0.0000000  
Max: 1.0000000  
Mean: 0.4389908  
Std.Dev.: 0.4962639
```

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

25.2 Søke i datasettet etter variable

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

```
look_for(wagepan, "married")
```

```
pos variable label col_type missing values  
8 married Married dbl 0
```

I output fremgår det at variabelen `married` finnes i datasettet, hva slags type den er, og eventuelle labler.

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

```
var_label(wagepan$married)
```

```
[1] "Married"
```

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

```
val_labels(wagepan$married)
```

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. For datasett fra R-pakker finnes dokumentasjonen i hjelpesiden, som du finner med `?wagepan` eller `help(wagepan)`.

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 variableneavn, verdier og labler. Her er et eksempel for hvordan finne variabler som inneholder ordet "wage". Du kan også søke på kortere eller lengre tekststrenger.

```
look_for(wagepan, "wage")
```

pos	variable	label	col_type	missing	values
4	lwage	Log hourly wage	dbl	0	

Her finner vi variabelen `lwage` som inneholder logaritmen av timelønn. Vi kan se nærmere på variabellabelen slik:

```
var_label(wagepan$lwage)
```

[1] "Log hourly wage"

Part VII

Del VI: Datahåndtering

26 Datahåndtering med tidyverse

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

I et tidligere kapittel ble det nevnt at R har ulike “dialekter”. I denne boken bruker vi **tidyverse** konsekvent, og dette kapittelet går grundigere inn i hvordan man bruker tidyverse til datahåndtering. Datahåndtering er alt man gjør med dataene *før* man analyserer dem: lage nye variable, velge ut variable, filtrere observasjoner, sortere, gruppere og oppsummere.

Tidyverse er en samling pakker som deler en felles filosofi og syntaks. Kjernepakken for datahåndtering er **dplyr**, som lastes automatisk når du laster **tidyverse**. Poenget med tidyverse er at koden skal være lesbar og logisk, nesten som å lese setninger.

Vi bruker datasettet *abu89* i eksemplene som følger. Dette datasettet inneholder informasjon om lønn, alder, utdanning, kjønn, klasse og sektor for et utvalg arbeidstakere. La oss først se på dataene:

```
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, 1, 0, 0, 1, 1, 1~
$ klasse89 <fct> III Rutinefunksjonærer, VIIa Ufaglærte arbeidere, II Nedre se~
$ promot    <fct> NEI, JA, JA, NEI, NEI, NEI, NEI, JA, NEI, JA, NEI, 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~
```

26.1 Pipe-operatoren: %>%

Det viktigste konseptet i tidyverse er “pipen” `%>%`. Den betyr rett og slett “ta dette, og gjør deretter...”. Pipen sender resultatet fra venstre side videre som input til funksjonen på høyre side.

Hurtigtasten er **Ctrl + Shift + M** (du kommer til å bruke denne mye!).

Uten pipe ville vi skrevet noe slikt:

```
head(abu89)
```

Med pipe kan vi skrive det slik:

```
abu89 %>% head()
```

I dette enkle eksempelet er det ingen stor forskjell. Men pipen blir veldig nyttig når man skal gjøre *flere ting etter hverandre*. Uten pipe ender man opp med enten svært uleselig nøsting av funksjoner eller mange mellomliggende objekter. Med pipe kan man kjede sammen operasjoner slik:

```
abu89 %>%  
  filtrer noe %>%  
  lag ny variabel %>%  
  oppsummer
```

Dette er pseudokode, men illustrerer poenget: man leser koden fra topp til bunn, steg for steg. Merk at det finnes en nyere pipe i base R: `|>`. Den fungerer omtrent likt for de aller fleste formål, og du vil se begge brukt i kodeeksempler på nettet. I denne boken bruker vi `%>%` konsekvent.

26.2 mutate() – lage nye variable

Funksjonen `mutate()` brukes til å lage nye variable eller endre eksisterende. Nye variable legges til som en ny kolonne i datasettet.

26.2.1 Enkel beregning

La oss si vi ønsker å lage en variabel som inneholder alderen i måneder:

```
abu89 %>%
  mutate(alder_mnd = age * 12) %>%
  select(io_nr, age, alder_mnd) %>%
  head()
```

```
# A tibble: 6 x 3
  io_nr   age  alder_mnd
  <dbl> <dbl>    <dbl>
1     3     58      696
2     4     24      288
3     5     44      528
4     8     46      552
5    11     40      480
6    12     36      432
```

Merk at vi ikke har endret det opprinnelige datasettet. For å lagre endringen må vi legge resultatet tilbake i et objekt:

```
abu89 <- abu89 %>%
  mutate(alder_mnd = age * 12)
```

26.2.2 Lage aldersgrupper med `case_when()`

Et svært vanlig behov er å lage kategorier basert på en kontinuerlig variabel. Det gjøres med `case_when()` inni `mutate()`:

```
abu89 <- abu89 %>%
  mutate(aldersgruppe = case_when(
    age < 30           ~ "Under 30",
    age >= 30 & age < 40 ~ "30-39",
    age >= 40 & age < 50 ~ "40-49",
    age >= 50           ~ "50+"
  ))
```

```
abu89 %>%
  select(age, aldersgruppe) %>%
  head(10)
```

```
# A tibble: 10 x 2
  age  aldersgruppe
  <dbl> <chr>
1     3  Under 30
2     4  Under 30
3     5  Under 30
4     8  Under 30
5    11  Under 30
6    12  Under 30
7    13  Under 30
8    14  Under 30
9    15  Under 30
10   16  Under 30
```

```

<dbl> <chr>
1   58 50+
2   24 Under 30
3   44 40-49
4   46 40-49
5   40 40-49
6   36 30-39
7   31 30-39
8   31 30-39
9   26 Under 30
10  29 Under 30

```

`case_when()` går gjennom betingelsene fra topp til bunn og tildeler verdien på høyre side av ~ for den første betingelsen som er oppfylt. Rekkefølgen har altså betydning.

26.2.3 Lage en binær variabel med `ifelse()`

For enklere tilfeller med bare to kategorier kan man bruke `ifelse()`:

```

abu89 <- abu89 %>%
  mutate(hoy_lønn = ifelse(time89 > 150, "Høy lønn", "Lav/middels lønn"))

abu89 %>%
  select(time89, hoy_lønn) %>%
  head()

```

```

# A tibble: 6 x 2
  time89 hoy_lønn
<dbl> <chr>
1   62   Lav/middels lønn
2   NA   <NA>
3   91.3 Lav/middels lønn
4   84.2 Lav/middels lønn
5   90.4 Lav/middels lønn
6  103.  Lav/middels lønn

```

26.3 `select()` – velge variable

Funksjonen `select()` brukes til å velge ut variable (kolonner) du ønsker å beholde – eller fjerne de du ikke trenger.

26.3.1 Velge variable

```
abu89 %>%
  select(io_nr, age, time89, female) %>%
  head()
```

```
# A tibble: 6 x 4
  io_nr    age time89 female
  <dbl> <dbl>   <dbl>   <dbl>
1     3     58     62      1
2     4     24     NA      0
3     5     44    91.3     1
4     8     46    84.2     1
5    11     40    90.4     0
6    12     36   103.      0
```

26.3.2 Fjerne variable

Bruker man minus-tegn foran variabelnavnet fjernes den:

```
abu89 %>%
  select(-io_nr, -alder_mnd, -aldersgruppe, -hoy_lonn) %>%
  head()
```

```
# A tibble: 6 x 8
  time89    ed    age female klasse89          promot fexp private
  <dbl> <dbl>   <dbl>   <dbl> <fct>        <fct> <dbl> <fct>
1    62      0     58      1 III Rutinefunksjonærer NEI      1 Public
2    NA      1     24      0 VIIa Ufaglærte arbeidere JA       0.3 Private
3   91.3     3     44      1 II Nedre serviceklasse JA       1.9 Private
4   84.2     5     46      1 II Nedre serviceklasse NEI      0.3 Public
5   90.4     3     40      0 II Nedre serviceklasse NEI      1 Private
6  103.      1     36      0 II Nedre serviceklasse NEI      1.2 Public
```

26.3.3 Hjelpefunksjoner i select()

Det finnes nyttige hjelpefunksjoner for å velge variable basert på navnemønster:

```
# Velg alle variable som starter med en bestemt tekst
abu89 %>% select(starts_with("time"))

# Velg alle variable som inneholder en bestemt tekst
abu89 %>% select(contains("89"))

# Velg alle numeriske variable
abu89 %>% select(where(is.numeric))
```

Disse er spesielt nyttige når man jobber med datasett som har mange variable.

26.4 filter() – filtrere rader

Mens `select()` velger kolonner, velger `filter()` rader basert på betingelser. Her brukes logiske operatorer som `==`, `!=`, `>`, `<`, `>=`, `<=`, `&` (og), `|` (eller).

26.4.1 Filtrere på én betingelse

```
abu89 %>%
  filter(age > 50) %>%
  head()

# A tibble: 6 x 12
  io_nr time89    ed    age female klasse89      promot fexp private alder_mnd
  <dbl>   <dbl> <dbl> <dbl> <dbl> <fct>       <fct>  <dbl> <fct>       <dbl>
1     3     62     0     58     1 III Rutinefunk~ NEI      1 Public     696
2    18    89.9    0     54     1 III Rutinefunk~ JA      3.5 Public    648
3    19     NA     9     58     0 I Øvre service~ JA      3.1 Public    696
4    22     NA     9     56     0 I Øvre service~ NEI     0.3 Public    672
5    23   112.     3     54     0 III Rutinefunk~ JA      2 Public     648
6    38     70     0     65     1 V-VI Faglærte ~ NEI     0.2 Private   780
# i 2 more variables: aldersgruppe <chr>, hoy_lonn <chr>
```

26.4.2 Filtrere på flere betingelser

```
abu89 %>%
  filter(age > 40, female == 1) %>%
  select(io_nr, age, female, time89) %>%
  head()
```

```
# A tibble: 6 x 4
  io_nr    age  female time89
  <dbl> <dbl>   <dbl>   <dbl>
1     3     58      1     62
2     5     44      1    91.3
3     8     46      1    84.2
4    18     54      1    89.9
5    38     65      1     70
6    39     65      1    73.9
```

Når man skriver flere betingelser adskilt med komma inni `filter()` tolkes det som “og” (&). Alternativt kan man bruke | for “eller”:

```
abu89 %>%
  filter(age < 25 | age > 60) %>%
  select(io_nr, age, time89) %>%
  head()
```

```
# A tibble: 6 x 3
  io_nr    age time89
  <dbl> <dbl>   <dbl>
1     4     24     NA
2    25     17     41
3    36     21     NA
4    38     65     70
5    39     65    73.9
6    42     61   118.
```

26.4.3 Fjerne manglende verdier

En vanlig bruk av `filter()` er å fjerne rader med manglende verdier (NA) på en bestemt variabel:

```
abu89_komplett <- abu89 %>%
  filter(!is.na(time89))
```

Her betyr `!is.na()` “er ikke NA”.

i Note

Merk forskjellen: `select()` velger *kolonner* (variable), mens `filter()` velger *rader* (observasjoner). Denne distinksjonen er viktig!

26.5 `summarise()` – oppsummere data

Funksjonen `summarise()` (eller `summarize()`, begge stavemåter fungerer) beregner oppsummerende statistikk og returnerer en ny, liten tabell.

```
abu89 %>%
  summarise(
    snitt_lonn = mean(time89, na.rm = TRUE),
    sd_lonn    = sd(time89, na.rm = TRUE),
    snitt_alder = mean(age, na.rm = TRUE),
    antall      = n()
  )

# A tibble: 1 x 4
  snitt_lonn sd_lonn snitt_alder antall
  <dbl>     <dbl>       <dbl>   <int>
1     90.1     30.3       39.7    4127
```

Merk at `n()` gir antall observasjoner. Argumentet `na.rm = TRUE` sier at manglende verdier skal ignoreres i beregningen. Uten dette argumentet vil resultatet bli NA hvis det finnes noen manglende verdier.

26.6 `group_by()` – grupperte operasjoner

`group_by()` gjør egentlig ingenting synlig i seg selv, men endrer hvordan etterfølgende funksjoner oppfører seg. Når man bruker `group_by()` etterfulgt av `summarise()` beregnes statistikken *per gruppe*.

```
abu89 %>%
  group_by(klasse89) %>%
  summarise(
    snitt_lonn = mean(time89, na.rm = TRUE),
```

```

    antall      = n()
)

# A tibble: 6 x 3
#> #> #> klasse89          snitt_lonn antall
#> #> #> <fct>            <dbl>   <int>
#> #> #> 1 I Øvre serviceklasse 118.     328
#> #> #> 2 II Nedre serviceklasse 104.    1181
#> #> #> 3 III Rutinefunksjonærer 75.5    1248
#> #> #> 4 V-VI Faglærte arbeidere 88.7    648
#> #> #> 5 VIIa Ufaglærte arbeidere 81.4    637
#> #> #> 6 <NA>              92.1    85

```

Man kan også gruppere etter flere variable:

```

abu89 %>%
  group_by(klasse89, female) %>%
  summarise(
    snitt_lonn = mean(time89, na.rm = TRUE),
    antall      = n()
  )

# A tibble: 12 x 4
# Groups:   klasse89 [6]
#> #> #> klasse89          female snitt_lonn antall
#> #> #> <fct>            <dbl>   <dbl>   <int>
#> #> #> 1 I Øvre serviceklasse 0       123.    254
#> #> #> 2 I Øvre serviceklasse 1       102.     74
#> #> #> 3 II Nedre serviceklasse 0       113.    626
#> #> #> 4 II Nedre serviceklasse 1       93.6    555
#> #> #> 5 III Rutinefunksjonærer 0       90.5    262
#> #> #> 6 III Rutinefunksjonærer 1       71.6    986
#> #> #> 7 V-VI Faglærte arbeidere 0       89.4    602
#> #> #> 8 V-VI Faglærte arbeidere 1       79.3     46
#> #> #> 9 VIIa Ufaglærte arbeidere 0       88.1    393
#> #> #> 10 VIIa Ufaglærte arbeidere 1       71.1    244
#> #> #> 11 <NA>              0       96.4     56
#> #> #> 12 <NA>              1       84.1     29

```

💡 Tip

Husk å bruke `ungroup()` etter at du er ferdig med grupperingen, spesielt hvis du skal gjøre flere operasjoner etterpå. Ellers kan grupperingen påvirke senere beregninger på uventede måter.

`group_by()` kan også brukes sammen med `mutate()`. Da legges den nye variabelen til i datasettet, men beregningen gjøres per gruppe. Et typisk eksempel er å beregne gruppegenomsnittet og legge det til som en variabel:

```
abu89 %>%
  group_by(klasse89) %>%
  mutate(snitt_lonn_klasse = mean(time89, na.rm = TRUE)) %>%
  ungroup() %>%
  select(io_nr, klasse89, time89, snitt_lonn_klasse) %>%
  head(10)
```

```
# A tibble: 10 x 4
  io_nr klasse89                  time89 snitt_lonn_klasse
  <dbl> <fct>                    <dbl>      <dbl>
1     3 III Rutinefunksjonærer    62        75.5
2     4 VIIa Ufaglærte arbeidere NA        81.4
3     5 II Nedre serviceklasse   91.3      104.
4     8 II Nedre serviceklasse   84.2      104.
5    11 II Nedre serviceklasse   90.4      104.
6    12 II Nedre serviceklasse  103.       104.
7    13 VIIa Ufaglærte arbeidere 75        81.4
8    14 I Øvre serviceklasse    110.      118.
9    16 V-VI Faglærte arbeidere  79        88.7
10   17 I Øvre serviceklasse    112.      118.
```

26.7 `arrange()` – sortere data

Funksjonen `arrange()` sorterer datasettet etter en eller flere variable. Som standard sorteres det i stigende rekkefølge.

```
abu89 %>%
  select(io_nr, age, time89) %>%
  arrange(time89) %>%
  head()
```

```
# A tibble: 6 x 3
  io_nr    age time89
  <dbl> <dbl> <dbl>
1   148     48   25
2  4107     17   25
3  3316     41  25.9
4  1685     37  26.3
5  2126     21  26.8
6   272     18  27.5
```

For synkende rekkefølge brukes `desc()`:

```
abu89 %>%
  select(io_nr, age, time89) %>%
  arrange(desc(time89)) %>%
  head()
```

```
# A tibble: 6 x 3
  io_nr    age time89
  <dbl> <dbl> <dbl>
1  2557     36  344.
2  3276     35  325
3  1188     63  300
4  1997     44  300
5  3757     42  300
6  3751     52  278.
```

26.8 Kombinere funksjoner med pipe

Den virkelige styrken med tidyverse er at man kan kombinere alle disse funksjonene i én sammenhengende arbeidsflyt. Her er et eksempel som gjør flere ting:

```
abu89 %>%
  filter(!is.na(time89)) %>%                                # Fjern manglende verdier
  mutate(ed_aar = as.numeric(as.character(ed))) %>%      # Gjør utdanning numerisk
  group_by(klasse89) %>%                                    # Grupper etter klasse
  summarise(
    snitt_lonn = mean(time89, na.rm = TRUE),                 # Gjennomsnittslønn
    snitt_alder = mean(age, na.rm = TRUE),                   # Gjennomsnittsalder
    antall = n()                                              # Antall per gruppe
```

```
) %>%
arrange(desc(snitt_lonn)) # Sorter etter lønn
```

```
# A tibble: 6 x 4
klasse89          snitt_lonn snitt_alder antall
<fct>            <dbl>      <dbl>    <int>
1 I Øvre serviceklasse 118.       40.8     292
2 II Nedre serviceklasse 104.       40.7    1044
3 <NA>              92.1       36.5     79
4 V-VI Faglærte arbeidere 88.7       38.5     607
5 VIIa Ufaglærte arbeidere 81.4       40.1     598
6 III Rutinefunksjonærer 75.5       38.2    1139
```

Denne koden leses som: "Ta datasettet abu89, fjern deretter rader med manglende lønn, lag deretter en numerisk utdanningsvariabel, grupper deretter etter klasse, beregn deretter gjennomsnitt per klasse, og sorter til slutt etter lønn."

Det er god praksis å legge til kommentarer i koden med `#` slik at du husker hva hvert steg gjør.

26.9 `across()` – samme operasjon på flere variable

Noen ganger ønsker man å gjøre det samme med flere variable samtidig. Funksjonen `across()` brukes inni `mutate()` eller `summarise()` for å anvende en funksjon på flere kolonner.

Her er et eksempel der vi beregner gjennomsnittet av alle numeriske variable per klasse:

```
abu89 %>%
group_by(klasse89) %>%
summarise(across(where(is.numeric), ~mean(.x, na.rm = TRUE))) %>%
head()
```

```
# A tibble: 6 x 8
klasse89          io_nr time89   ed   age female  fexp alder_mnd
<fct>            <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>      <dbl>
1 I Øvre serviceklasse 3112. 118.  6.23  41.1  0.226  0.990    494.
2 II Nedre serviceklasse 3095. 104.  4.27  40.9  0.470  1.03     491.
3 III Rutinefunksjonærer 3139.  75.5 1.67  38.5  0.790  0.798    462.
4 V-VI Faglærte arbeidere 3205.  88.7 1.73  38.9  0.0710 1.03     467.
5 VIIa Ufaglærte arbeidere 3044.  81.4 0.948 40.1  0.383  0.977    481.
6 <NA>              2388.  92.1 2.42  36.6  0.341  0.826    439.
```

Syntaksen `~mean(.x, na.rm = TRUE)` er en forkortet måte å skrive en funksjon på der `.x` er et plassholdernavn for variabelen. `where(is.numeric)` velger alle numeriske variable. Man kan også angi variabelnavn direkte:

```
abu89 %>%
  summarise(across(c(age, time89),
    list(snitt = ~mean(.x, na.rm = TRUE),
      sd = ~sd(.x, na.rm = TRUE))))
```

```
# A tibble: 1 x 4
  age_snitt age_sd time89_snitt time89_sd
  <dbl>     <dbl>       <dbl>      <dbl>
1     39.7     12.4       90.1      30.3
```

Her beregnes både gjennomsnitt og standardavvik for variablene `age` og `time89`. Resultatnavnene settes automatisk sammen av variabelnavn og funksjonsnavnet angitt i `list()`.

`across()` er svært nyttig for å unngå repetitiv kode, men kan virke litt kryptisk i starten. Det er helt greit å skrive ut variabel for variabel til man er komfortabel med `across()`.

26.10 Oppsummering

Her er en oversikt over de viktigste funksjonene i `dplyr`:

Funksjon	Hva den gjør
<code>mutate()</code>	Lager nye variable / endrer eksisterende
<code>select()</code>	Velger (eller fjerner) kolonner
<code>filter()</code>	Filtrerer rader basert på betingelser
<code>summarise()</code>	Beregner oppsummerende statistikk
<code>group_by()</code>	Grupperer data for etterfølgende operasjoner
<code>arrange()</code>	Sorterer rader
<code>across()</code>	Anvender funksjoner på flere kolonner

Alle disse kan kombineres med `%>%` for å lage ryddige, lesebare arbeidsflyter.

26.11 Oppgaver

Exercise 26.1. Bruk datasettet abu89 og lag en ny variabel som angir om en person er over eller under 40 år. Bruk `mutate()` og `ifelse()`.

Exercise 26.2. Filtrer datasettet slik at du bare har personer som jobber i privat sektor, og beregn gjennomsnittslønn og gjennomsnittsalder for dette utvalget med `summarise()`.

Exercise 26.3. Bruk `group_by()` og `summarise()` til å beregne gjennomsnittlig timelønn for hver klasse, fordelt på kjønn. Sorter resultatet etter synkende lønn med `arrange()`.

Exercise 26.4. Skriv en sammenhengende pipe som filtrerer bort manglende verdier på timelønn, grupperer etter aldersgruppe (som du lager med `case_when()`), og beregner gjennomsnitt, standardavvik og antall per gruppe.

27 Omkoding av variable

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

Å omkode variable er noe du vil gjøre veldig ofte. Noen ganger trenger du å slå sammen kategorier, andre ganger trenger du å lage helt nye variable basert på eksisterende. Det kan handle om å lage aldersgrupper fra en kontinuerlig aldersvariabel, slå sammen utdanningskategorier, eller gjøre om en variabel til en annen type. I dette kapittelet går vi gjennom de viktigste teknikkene.

27.1 Factor-variable

Factor-variable er Rs måte å håndtere kategoriske data på. En factor har definerte *nivåer* (levels) som angir de mulige verdiene. La oss se på et eksempel:

```
class(abu89$klasse89)
[1] "factor"

levels(abu89$klasse89)
[1] "I Øvre serviceklasse"      "II Nedre serviceklasse"
[3] "III Rutinefunksjonærer"   "V-VI Faglærte arbeidere"
[5] "VIIa Ufaglærte arbeidere"
```

Vi kan se at variabelen `klasse89` er en factor med fem nivåer. Rekkefølgen på nivåene har betydning, blant annet for hvilken kategori som blir referansekategori i regresjonsanalyser.

27.1.1 Lage factor fra tall

Hvis du har en numerisk variabel som egentlig er kategorisk, kan du gjøre den om til factor:

```
abu89 <- abu89 %>%
  mutate(kjonn = factor(ifelse(female == 1, "Kvinne", "Mann"),
                        levels = c("Mann", "Kvinne")))

table(abu89$kjonn)
```

	Mann	Kvinne
2193	1934	

Med `levels` = bestemmer du rekkefølgen. Den første kategorien blir referansekategori i regresjon.

27.2 Omkoding med `ifelse()`

Den enkleste formen for omkoding er `ifelse()`. Den tester en betingelse og gir én verdi hvis betingen er sann og en annen hvis den er usann:

```
abu89 <- abu89 %>%
  mutate(hoylønn = ifelse(time89 > 100, "Høy lønn", "Lav lønn"))

table(abu89$hoylønn)
```

	Høy lønn	Lav lønn
1007	2752	

Her er `time89 > 100` betingen. Alle som har timelønn over 100 får verdien “Høy lønn”, resten får “Lav lønn”.

27.3 Omkoding med `case_when()`

Når du har mer enn to kategorier, er `case_when()` mye mer oversiktlig enn nestede `ifelse()`-kall:

```
abu89 <- abu89 %>%
  mutate(aldersgruppe = case_when(
    age < 25 ~ "Under 25",
    age < 35 ~ "25-34",
    age < 45 ~ "35-44",
    age < 55 ~ "45-54",
    age >= 55 ~ "55 og over"
  ))
  
table(abu89$aldersgruppe)
```

25-34	35-44	45-54	55 og over	Under 25
1061	1179	790	595	502

Merk at `case_when()` evaluerer betingelsene *ovenfra og ned*. Den stopper ved den første betingelsen som er sann. Derfor trenger vi ikke skrive `age >= 25 & age < 35` for den andre linjen - alle under 25 er allerede fanget opp av første linje.



Tip

Hvis ingen av betingelsene er sanne, får verdien NA. Du kan legge til `.default = "Annet"` som siste argument for å fange opp alt som ikke matcher.

Resultatet av `case_when()` er en tekstvariabel. Hvis du vil ha den som factor med bestemt rekkefølge, kan du pakke det inn i `factor()`:

```
abu89 <- abu89 %>%
  mutate(aldersgruppe = factor(
    case_when(
      age < 25 ~ "Under 25",
      age < 35 ~ "25-34",
      age < 45 ~ "35-44",
      age < 55 ~ "45-54",
      age >= 55 ~ "55 og over"
    ),
  ))
```

```

  levels = c("Under 25", "25-34", "35-44", "45-54", "55 og over")
))

levels(abu89$aldersgruppe)

[1] "Under 25"    "25-34"      "35-44"      "45-54"      "55 og over"

```

27.4 Lage grupper med `cut()`

For å dele en kontinuerlig variabel inn i grupper er `cut()` et hendig alternativ:

```

abu89 <- abu89 %>%
  mutate(aldersgruppe2 = cut(age,
                             breaks = c(0, 25, 35, 45, 55, 100),
                             labels = c("Under 25", "25-34", "35-44",
                                       "45-54", "55+")))

table(abu89$aldersgruppe2)

```

Under 25	25-34	35-44	45-54	55+
587	1092	1171	745	532

`breaks` angir grensene og `labels` angir navnene på gruppene. Merk at det alltid er én label mindre enn antall grenseverdier.

27.5 Endre factor-nivåer med `forcats`

Pakken `forcats` (en del av tidyverse) har en rekke nyttige funksjoner for å jobbe med factor-variable.

27.5.1 Slå sammen kategorier med `fct_collapse()`

Hvis du har for mange kategorier og vil slå noen sammen:

```
abu89 <- abu89 %>%
  mutate(klasse_enkel = fct_collapse(klasse89,
    "Service" = c("I: Øvre serviceklasse", "II: Nedre serviceklasse"),
    "Rutine" = c("III: Rutinefunksjonærer"),
    "Arbeider" = c("V-VI: Faglærte arbeidere", "VII: Ufaglærte arbeidere")
  ))
```

Warning: There was 1 warning in `mutate()`.
 i In argument: `klasse_enkel = fct_collapse(...)`.
 Caused by warning:
 ! Unknown levels in `f`: I: Øvre serviceklasse, II: Nedre serviceklasse, III: Rutinefunksjonærer

```
table(abu89$klasse_enkel)
```

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

27.5.2 Gi nye navn med fct_recode()

Noen ganger vil du bare endre navnene uten å slå sammen:

```
abu89 <- abu89 %>%
  mutate(klasse_kort = fct_recode(klasse89,
    "I" = "I: Øvre serviceklasse",
    "II" = "II: Nedre serviceklasse",
    "III" = "III: Rutinefunksjonærer",
    "V-VI" = "V-VI: Faglærte arbeidere",
    "VII" = "VII: Ufaglærte arbeidere"
  ))
```

Warning: There was 1 warning in `mutate()`.
 i In argument: `klasse_kort = fct_recode(...)`.
 Caused by warning:
 ! Unknown levels in `f`: I: Øvre serviceklasse, II: Nedre serviceklasse, III: Rutinefunksjonærer

```
table(abu89$klasse_kort)
```

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

Merk syntaksen: "nytt navn" = "gammelt navn".

27.5.3 Endre rekkefølge med fct_relevel()

For å endre hvilken kategori som er først (og dermed referansekategori i regresjon):

```
abu89 <- abu89 %>%
  mutate(klasse_omvendt = fct_relevel(klasse89, "VII: Ufaglærte arbeidere"))
```

```
Warning: There was 1 warning in `mutate()` .
i In argument: `klasse_omvendt = fct_relevel(klasse89, "VII: Ufaglærte
  arbeidere")` .
Caused by warning:
! 1 unknown level in `f`: VII: Ufaglærte arbeidere
```

```
levels(abu89$klasse_omvendt)
```

```
[1] "I Øvre serviceklasse"      "II Nedre serviceklasse"
[3] "III Rutinefunksjonærer"    "V-VI Faglærte arbeidere"
[5] "VIIa Ufaglærte arbeidere"
```

Nå er "VII: Ufaglærte arbeidere" flyttet til første posisjon og vil bli referansekategori.

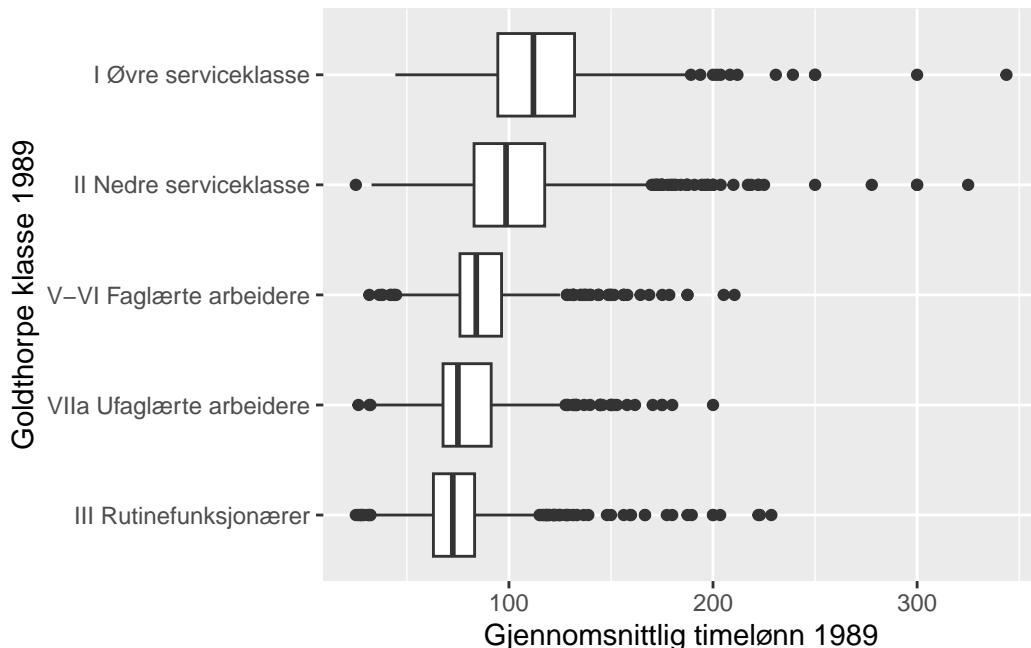
27.5.4 Sortere etter en annen variabel med fct_reorder()

Noen ganger vil du sortere kategoriene etter en annen variabel, for eksempel for å lage pene grafer:

```

abu89 %>%
  filter(!is.na(klasse89), !is.na(time89)) %>%
  mutate(klasse89 = fct_reorder(klasse89, time89, .fun = mean)) %>%
  ggplot(aes(x = klasse89, y = time89)) +
  geom_boxplot() +
  coord_flip()

```



Her sorteres klassekategoriene etter gjennomsnittlig timelønn, slik at den klassen med lavest lønn kommer først.

27.5.5 Fjerne ubrukte nivåer med `fct_drop()`

Etter filtrering kan du sitte igjen med factor-nivåer som ikke lenger har observasjoner:

```

abu89_lite <- abu89 %>%
  filter(klasse89 %in% c("I: Øvre serviceklasse", "II: Nedre serviceklasse"))

# Ubrukte nivåer er fortsatt der
levels(abu89_lite$klasse89)

```

```
[1] "I Øvre serviceklasse"      "II Nedre serviceklasse"
```

```
[3] "III Rutinefunksjonærer"    "V-VI Faglærte arbeidere"  
[5] "VIIa Ufaglærte arbeidere"
```

```
# Fjern dem  
abu89_lite <- abu89_lite %>%  
  mutate(klasse89 = fct_drop(klasse89))  
  
levels(abu89_lite$klasse89)  
  
character(0)
```

27.6 Jobbe med tekst

Noen ganger trenger du å jobbe med tekstverdier. Pakken **stringr** (en del av tidyverse) har nyttige funksjoner:

```
# Finne tekst i en variabel  
abu89 %>%  
  filter(str_detect(as.character(klasse89), "service")) %>%  
  head(3) %>%  
  select(klasse89, time89)  
  
# A tibble: 3 x 2  
klasse89          time89  
<fct>            <dbl>  
1 II Nedre serviceklasse  91.3  
2 II Nedre serviceklasse  84.2  
3 II Nedre serviceklasse  90.4  
  
# Erstatte tekst  
abu89 %>%  
  mutate(klasse_ny = str_replace(as.character(klasse89), "klasse", "kl."))
```

27.7 Praktisk eksempel: komplett omkoding

Her er et eksempel som viser en typisk omkodingsprosess der vi forbereder data til analyse:

```

abu89_analyse <- abu89 %>%
  mutate(
    # Lage kjønnsvariabel som factor
    kjonn = factor(ifelse(female == 1, "Kvinne", "Mann"),
                  levels = c("Mann", "Kvinne")),
    # Lage aldersgrupper
    alder_gr = cut(age, breaks = c(0, 30, 40, 50, 100),
                  labels = c("Under 30", "30-39", "40-49", "50+")),
    # Forenkle klassevariabel
    klasse = fct_collapse(klasse89,
      "Serviceklasse" = c("I: Øvre serviceklasse", "II: Nedre serviceklasse"),
      "Rutine" = "III: Rutinefunksjonærer",
      "Arbeiderklasse" = c("V-VI: Faglærte arbeidere", "VII: Ufaglærte arbeidere")
    ),
    # Lage dummy for høy lønn
    hoylønn = ifelse(time89 > median(time89, na.rm = TRUE), 1, 0)
  ) %>%
  select(time89, hoylønn, kjonn, age, alder_gr, klasse)

glimpse(abu89_analyse)

```

Rows: 4,127
 Columns: 6

Oppgave	Funksjon	Eksempel
To kategorier	<code>ifelse()</code>	<code>ifelse(x > 10, "Høy", "Lav")</code>
Flere kategorier	<code>case_when()</code>	<code>case_when(x < 10 ~ "Lav", ...)</code>
Kutte kontinuerlig	<code>cut()</code>	<code>cut(x, breaks = c(0,10,20))</code>

27.8 Oppsummering

Oppgave	Funksjon	Eksempel
To kategorier	<code>ifelse()</code>	<code>ifelse(x > 10, "Høy", "Lav")</code>
Flere kategorier	<code>case_when()</code>	<code>case_when(x < 10 ~ "Lav", ...)</code>
Kutte kontinuerlig	<code>cut()</code>	<code>cut(x, breaks = c(0,10,20))</code>

Oppgave	Funksjon	Eksempel
Slå sammen nivåer	<code>fct_collapse()</code>	<code>fct_collapse(x, A = c("a", "b"))</code>
Endre navn	<code>fct_recode()</code>	<code>fct_recode(x, "Ny" = "Gml")</code>
Endre rekkefølge	<code>fct_relevel()</code>	<code>fct_relevel(x, "Sist")</code>
Sortere etter verdi	<code>fct_reorder()</code>	<code>fct_reorder(x, y, mean)</code>
Fjerne tomme nivåer	<code>fct_drop()</code>	<code>fct_drop(x)</code>

28 Haandtering av missing-verdier

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

I nesten alle datasett fra sporreskjemaundersokelser vil det vaere observasjoner der vi mangler informasjon paa noen variable. Noen har ikke svart paa alle sporsmaal, noen har hoppet over deler av skjemaet, og noen ganger er det feil i registreringen. Slike manglende verdier – eller *missing values* – er noe du maa forholde deg til i praksis. R representerer manglende verdier med den spesielle verdien `NA` (Not Available), og det er viktig aa forstaa hvordan dette fungerer for aa unngaa feil i analysene.

28.1 Hva er NA?

`NA` er Rs maate aa si “vi vet ikke”. Det er ikke det samme som null eller en tom tekststreng. Det betyr rett og slett at verdien mangler. Du kan tenke paa det som et tomt felt i et regneark – det er ikke fylt ut, og vi vet ikke hva svaret ville vaert.

```
x <- c(3, 7, NA, 12, NA, 5)
x
```

```
[1] 3 7 NA 12 NA 5
```

Legg merke til at `NA` ikke har anførselstegn rundt seg. Det er fordi `NA` er en spesiell verdi i R, ikke en tekststreng.

28.2 Sjekke for missing-verdier

For aa finne ut hvilke verdier som er `NA` bruker vi funksjonen `is.na()`. Den returnerer `TRUE` for hver verdi som er `NA` og `FALSE` for resten.

```
is.na(x)
```

```
[1] FALSE FALSE TRUE FALSE TRUE FALSE
```

Merk at du *ikke* kan bruke vanlig likhetstegn for aa sjekke om noe er NA. Uttrykket `x == NA` gir nemlig NA tilbake – ikke TRUE eller FALSE som du kanskje forventer. Det er fordi R tenker at “vi vet ikke om en ukjent verdi er lik en annen ukjent verdi”. Bruk alltid `is.na()`.

For et helt datasett kan vi bruke `complete.cases()` for aa finne radene som ikke har noen manglende verdier overhodet:

```
sum(complete.cases(abu89))
```

```
[1] 3680
```

Dette forteller oss hvor mange rader i datasettet som har gyldige verdier paa *alle* variable.

28.3 Oppsummere missing i datasettet

Det er lurt aa skaffe seg en oversikt over hvor mye missing det er i datasettet for man begynner aa analysere. En rask maate er aa kombinere `colSums()` og `is.na()`:

```
colSums(is.na(abu89))
```

io_nr	time89	ed	age	female	klasse89	promot	fexp
0	368	0	0	0	85	0	0
private							
0							

Dette gir antall NA per variabel. Hvis det er mange variable kan det vaere nyttig aa sortere resultatet:

```
sort(colSums(is.na(abu89)), decreasing = TRUE)
```

time89	klasse89	io_nr	ed	age	female	promot	fexp
368	85	0	0	0	0	0	0
private							
0							

For en visuell oversikt kan du bruke pakken **naniar**:

```
library(naniar)
vis_miss(abu89)
```

Denne lager et plott der du ser monstre i missing-verdiene. Det kan for eksempel avsløre om det er bestemte variable som har mye missing, eller om det er grupper av observasjoner som mangler paa mange variable samtidig.

28.4 Funksjoner og NA: argumentet na.rm

Mange funksjoner i R gir **NA** som resultat hvis det er missing-verdier i dataene. Det er egentlig fornuftig: R sier at “resultatet er ukjent fordi noen av verdiene er ukjente”. Men i praksis vil man som oftest beregne resultatet basert paa de verdiene man faktisk har.

```
x <- c(3, 7, NA, 12, 5)
mean(x)
```

```
[1] NA
```

For aa faa gjennomsnittet av de verdiene som finnes, bruker vi argumentet **na.rm = TRUE** (som staar for “NA remove”):

```
mean(x, na.rm = TRUE)
```

```
[1] 6.75
```

Dette gjelder for en rekke funksjoner som **sum()**, **sd()**, **median()**, **min()**, **max()** og andre. Det er et veldig vanlig argument som du kommer til aa bruke ofte.

28.5 Filtrere ut missing

Naar du jobber med datasett i tidyverse kan du filtrere bort rader med missing paa bestemte variable. Det gjor du med **filter()** og **is.na()**:

```
abu89 %>%
  filter(!is.na(time89)) %>%
  nrow()
```

```
[1] 3759
```

Her beholder vi bare radene der `time89` ikke er `NA`. Utropstegnet ! betyr “ikke”.

Hvis du vil fjerne alle rader som har `NA` paa en eller flere bestemte variable kan du bruke `drop_na()` fra `tidyverse`:

```
abu89 %>%
  drop_na(time89, ed) %>%
  nrow()
```

```
[1] 3759
```

Uten argumenter fjerner `drop_na()` alle rader med missing paa *noen som helst* variabel. Det kan vaere ganske drastisk, saa det er som regel lurt aa spesifisere hvilke variable du vil fjerne missing paa.

28.6 Omkode verdier til NA

I en del datasett er missing-verdier kodet som bestemte tall i stedet for ekte `NA`. For eksempel kan -9, 99 eller 999 bety “ikke svart” eller “vet ikke”. Slike verdier maa du kode om til `NA` selv.

Med `mutate()` og `na_if()` er dette enkelt:

```
datasett <- datasett %>%
  mutate(variabel = na_if(variabel, -9))
```

Dette gjor at alle verdier som er lik -9 paa variabelen blir til `NA`. Du kan ogsaa bruke `case_when()` for mer avansert omkoding der flere verdier skal bli `NA`:

```
datasett <- datasett %>%
  mutate(variabel = case_when(
    variabel %in% c(-9, -8, 999) ~ NA_real_,
    TRUE ~ variabel
  ))
```

28.7 Erstatte NA med verdier

Noen ganger vil du erstatte `NA` med en bestemt verdi. For eksempel kan det vaere fornuftig aa anta at manglende verdier paa en variabel for antall barn betyr at personen ikke har barn, altsa null.

```
y <- c(2, NA, 1, NA, 3)
replace_na(y, 0)
```

```
[1] 2 0 1 0 3
```

I en pipe med `mutate()`:

```
datasett <- datasett %>%
  mutate(ant_barn = replace_na(ant_barn, 0))
```

Funksjonen `coalesce()` er nyttig hvis du har flere variable og vil bruke den forste som ikke er `NA`:

```
a <- c(NA, 2, NA, 4)
b <- c(10, NA, 30, NA)
coalesce(a, b)
```

```
[1] 10 2 30 4
```

Denne funksjonen velger den forste ikke-manglende verdien fra venstre til hoyre. Det kan vaere nyttig naar du har informasjon fra flere kilder og vil fylle inn der det mangler.

28.8 Missing i regresjonsanalyser

Naar du kjører en regresjonsmodell med `lm()` i R, saa haandterer funksjonen `missing` automatisk ved aa fjerne alle observasjoner som har `NA` paa noen av variablene i modellen. Dette kalles *listwise deletion* (eller *complete case analysis*).

```
mod <- lm(time89 ~ ed + age, data = abu89)
nobs(mod)
```

```
[1] 3759
```

```
nrow(abu89)
```

```
[1] 4127
```

Legg merke til at antall observasjoner i modellen (`nobs()`) kan vaere lavere enn antall rader i datasettet. Det er fordi rader med `NA` paa noen av variablene i modellen er fjernet. Det er viktig aa vaere oppmerksom paa dette, spesielt naar du sammenligner modeller med ulike variable. Hvis en variabel har mye missing, kan det endre utvalget betraktelig fra en modell til en annen.

Et godt tips er aa lage et analysesdatasett der du fjerner missing paa de relevante variablene først, slik at alle modeller estimeres paa det samme utvalget:

```
abu89_analyse <- abu89 %>%
  drop_na(time89, ed, age)

mod1 <- lm(time89 ~ ed, data = abu89_analyse)
mod2 <- lm(time89 ~ ed + age, data = abu89_analyse)

nobs(mod1)
```

```
[1] 3759
```

```
nobs(mod2)
```

```
[1] 3759
```

Naa bruker begge modellene noyaktig samme utvalg, og eventuelle forskjeller skyldes modellspesifikasjonen og ikke at ulike observasjoner er med.

28.9 Multippel imputering

Listwise deletion er den enkleste tilnaermingen, men den har en ulempe: du mister data. Hvis mange observasjoner har missing paa minst en variabel, kan utvalget bli vesentlig mindre. I tillegg kan resultatene bli skjeve hvis dataene ikke mangler helt tilfeldig.

En mer avansert tilnaerming er *multippel imputering*, der man estimerer hva de manglende verdiene sannsynligvis ville vaert basert paa den informasjonen man faktisk har. Pakken `mice` i R er det mest brukte verktoyet for dette. Vi gaar ikke naermere inn paa dette her, men det er greit aa vite at det finnes. Hvis du har mye missing i dataene dine og det er viktig for resultatene, bor du sette deg inn i dette temaet.

28.10 Vanlige fallgruver og tips

Her er noen praktiske rad for aa haandtere missing:

- **Sjekk alltid for missing for du begynner aa analysere.** Bruk `summary()` eller `colSums(is.na())` for aa faa en oversikt. Missing-verdier som du ikke er klar over kan gi misvisende resultater.
- **Bruk `na.rm = TRUE` bevisst.** Det er lett aa bare legge til `na.rm = TRUE` overalt, men tenk igjennom *hvorfor* det er missing. Hvis mange verdier mangler, bør du undersøke dette naermere for du fjerner dem.
- **Ikke bruk `x == NA`.** Bruk alltid `is.na(x)`. Dette er en av de vanligste nybegynnerfeilene i R.
- **Pass paa at missing betyr det du tror.** I noen datasett er missing-verdier kodet som -9, 999 eller lignende. Sjekk dokumentasjonen for datasettet og kod om til NA der det trengs.
- **Vær forsiktig med `drop_na()` uten argumenter.** Uten argumenter fjerner den alle rader med missing paa *hvilk* variabel, noe som kan gi et veldig lite datasett.
- **Lag et analysesdatasett.** Når du kjører regresjoner, lag et eget datasett der du har fjernet missing paa de relevante variablene, slik at alle modeller bruker samme utvalg.

29 Koble sammen og omforme data

```
library(tidyverse)
```

I samfunnsvitenskapelig forskning vil man ofte ha behov for data fra ulike kilder. Kanskje har du et datasett med survey-data om individer og et annet med registerdata om kommunene de bor i. Eller du har data fra to ulike tidspunkter som skal kobles sammen. Da trenger du verktøy for å koble datasett sammen.

I tillegg vil du noen ganger oppleve at dataene er organisert på en måte som ikke passer til det du skal gjøre. Data kan være i “bredt” format der hvert tidspunkt har sin egen kolonne, mens du trenger det i “langt” format der hvert tidspunkt er en egen rad – eller omvendt. Da trenger du verktøy for å omforme data.

Dette kapittelet handler om begge deler: å koble datasett og å omforme data mellom bredt og langt format.

29.1 Koble datasett med join-funksjoner

For å illustrere hvordan join-funksjoner fungerer lager vi to små eksempldatasett. Tenk deg at du har et datasett med karakterer fra et kurs og et annet med bakgrunnsinformasjon om studentene.

```
karakterer <- tibble(  
  id = c(1, 2, 3, 4),  
  navn = c("Anna", "Bjørn", "Cecilie", "David"),  
  karakter = c("A", "B", "C", "B")  
)  
  
bakgrunn <- tibble(  
  id = c(1, 2, 3, 5),  
  studieprogram = c("Sosiologi", "Statsvitenskap", "Økonomi", "Sosiologi"),  
  kjonn = c("K", "M", "K", "M")  
)
```

Legg merke til at de to datasettene ikke har helt de samme id-verdiene. Student 4 (David) finnes bare i karakterdatasettet, mens student 5 finnes bare i bakgrunnsdatasettet. Dette er helt vanlig i praksis og det er nettopp derfor det er viktig å forstå de ulike typene joins.

La oss se på de to datasettene:

```
karakterer
```

```
# A tibble: 4 x 3
  id navn   karakter
  <dbl> <chr>  <chr>
1     1 Anna    A
2     2 Bjørn   B
3     3 Cecilie C
4     4 David   B
```

```
bakgrunn
```

```
# A tibble: 4 x 3
  id studieprogram kjonn
  <dbl> <chr>      <chr>
1     1 Sosiologi K
2     2 Statsvitenskap M
3     3 Økonomi K
4     5 Sosiologi M
```

29.1.1 left_join: Behold alt fra venstre datasett

Den klart vanligste join-funksjonen er `left_join()`. Den beholder alle radene fra det første (venstre) datasettet og legger til informasjon fra det andre datasettet der det finnes en match.

```
left_join(karakterer, bakgrunn, by = "id")
```

```
# A tibble: 4 x 5
  id navn   karakter studieprogram kjonn
  <dbl> <chr>  <chr>      <chr>      <chr>
1     1 Anna    A         Sosiologi   K
2     2 Bjørn   B         Statsvitenskap M
3     3 Cecilie C         Økonomi    K
4     4 David   B         <NA>       <NA>
```

Her ser vi at alle fire studentene fra `karakterer` er med. Anna, Bjørn og Cecilie har fått påkoblet informasjon om studieprogram og kjønn. Men David (id = 4) finnes ikke i `bakgrunn`, så han får NA på de nye variablene. Student 5 fra `bakgrunn` er ikke med fordi vedkommende ikke fantes i `karakterer`.

Dette er den vanligste situasjonen: du har et hoveddatasett og vil legge til tilleggsinformasjon fra en annen kilde.

29.1.2 `right_join`: Behold alt fra høyre datasett

`right_join()` gjør det motsatte: beholder alt fra det andre (høyre) datasettet.

```
right_join(karakterer, bakgrunn, by = "id")
```

```
# A tibble: 4 x 5
  id navn karakter studieprogram kjonn
  <dbl> <chr>  <chr>    <chr>      <chr>
1     1 Anna    A        Sosiologi   K
2     2 Bjørn   B        Statsvitenskap M
3     3 Cecilie C        Økonomi    K
4     5 <NA>    <NA>     Sosiologi  M
```

Nå er student 5 med (fra `bakgrunn`), men David (id = 4) er borte. I praksis brukes `right_join()` sjeldent fordi man like gjerne kan bytte rekkefølgen på datasettene og bruke `left_join()`.

29.1.3 `inner_join`: Bare rader som finnes i begge

`inner_join()` beholder bare radene som har en match i begge datasettene.

```
inner_join(karakterer, bakgrunn, by = "id")
```

```
# A tibble: 3 x 5
  id navn karakter studieprogram kjonn
  <dbl> <chr>  <chr>    <chr>      <chr>
1     1 Anna    A        Sosiologi   K
2     2 Bjørn   B        Statsvitenskap M
3     3 Cecilie C        Økonomi    K
```

Her er bare id 1, 2 og 3 med – altså de som finnes i begge datasett. Både David (id = 4) og student 5 er borte. Dette kan være nyttig når du bare vil ha komplette observasjoner, men vær oppmerksom på at du kan miste data uten å være klar over det.

29.1.4 full_join: Behold alt fra begge datasett

`full_join()` beholder alle rader fra begge datasettene.

```
full_join(karakterer, bakgrunn, by = "id")
```

```
# A tibble: 5 x 5
  id navn karakter studieprogram kjonn
  <dbl> <chr> <chr>   <chr>      <chr>
1     1 Anna    A        Sosiologi    K
2     2 Bjørn   B        Statsvitenskap M
3     3 Cecilie C        Økonomi     K
4     4 David   B        <NA>       <NA>
5     5 <NA>    <NA>    Sosiologi    M
```

Nå er alle med. David får NA på studieprogram og kjønn, og student 5 får NA på navn og karakter. Dette gir deg all tilgjengelig informasjon, men du må håndtere de manglende verdiene etterpå.

29.1.5 Spesifisere koblingsnøkkelen med `by`-argumentet

I eksemplene ovenfor hadde begge datasettene en variabel som het `id`, og vi koblet på den. Men hva om variablene heter forskjellige ting i de to datasettene? Da må du spesifisere dette eksplisitt.

```
karakterer2 <- tibble(
  student_id = c(1, 2, 3, 4),
  navn = c("Anna", "Bjørn", "Cecilie", "David"),
  karakter = c("A", "B", "C", "B")
)

bakgrunn2 <- tibble(
  id_nr = c(1, 2, 3, 5),
  studieprogram = c("Sosiologi", "Statsvitenskap", "Økonomi", "Sosiologi")
)

left_join(karakterer2, bakgrunn2, by = c("student_id" = "id_nr"))
```

```
# A tibble: 4 x 4
  student_id navn    karakter studieprogram
```

```

<dbl> <chr>  <chr>   <chr>
1      1 Anna    A        Sosiologi
2      2 Bjørn   B        Statsvitenskap
3      3 Cecilie C        Økonomi
4      4 David   B        <NA>

```

Her forteller vi R at `student_id` i det første datasettet tilsvarer `id_nr` i det andre.

Du kan også koble på flere variable samtidig. For eksempel hvis du har paneldata med både person-id og år:

```

survey <- tibble(
  id = c(1, 1, 2, 2),
  aar = c(2020, 2023, 2020, 2023),
  tilfredshet = c(7, 8, 5, 6)
)

register <- tibble(
  id = c(1, 1, 2, 2),
  aar = c(2020, 2023, 2020, 2023),
  inntekt = c(450000, 510000, 380000, 420000)
)

left_join(survey, register, by = c("id", "aar"))

```

```

# A tibble: 4 x 4
  id   aar tilfredshet inntekt
<dbl> <dbl>       <dbl>    <dbl>
1     1  2020         7  450000
2     1  2023         8  510000
3     2  2020         5  380000
4     2  2023         6  420000

```

29.1.6 Hva kan gå galt?

Det vanligste problemet er at koblingen gir flere rader enn forventet. Det skjer når det er duplikater i koblingsnøklene. Hvis en id forekommer flere ganger i det ene datasettet, får du en rad for hver kombinasjon. Sjekk derfor alltid antall rader før og etter en join for å forsikre deg om at resultatet er som forventet:

```
resultat <- left_join(karakterer, bakgrunn, by = "id")  
nrow(karakterer)
```

```
[1] 4
```

```
nrow(resultat)
```

```
[1] 4
```

Hvis `nrow(resultat)` er større enn `nrow(karakterer)` etter en `left_join()`, så har du sannsynligvis duplikater i koblingsnøklene og bør undersøke dette nærmere.

29.2 Omforme data: bredt og langt format

I samfunnsvitenskap jobber vi ofte med paneldata – altså data der vi har målinger på samme enhet over tid. Slike data kan organiseres på to måter.

I **bredt format** har hver enhet en rad, og hvert tidspunkt har sin egen kolonne:

```
bred <- tibble(  
  kommune = c("Oslo", "Bergen", "Trondheim"),  
  arbeidsledighet_2020 = c(4.2, 3.8, 3.5),  
  arbeidsledighet_2021 = c(5.1, 4.5, 4.0),  
  arbeidsledighet_2022 = c(3.9, 3.6, 3.2)  
)  
bred
```

```
# A tibble: 3 x 4  
  kommune    arbeidsledighet_2020 arbeidsledighet_2021 arbeidsledighet_2022  
  <chr>          <dbl>            <dbl>            <dbl>  
1 Oslo           4.2              5.1              3.9  
2 Bergen         3.8              4.5              3.6  
3 Trondheim      3.5              4                 3.2
```

I **langt format** har hver måling sin egen rad:

```

lang <- tibble(
  kommune = rep(c("Oslo", "Bergen", "Trondheim"), each = 3),
  aar = rep(c(2020, 2021, 2022), 3),
  arbeidsledighet = c(4.2, 5.1, 3.9, 3.8, 4.5, 3.6, 3.5, 4.0, 3.2)
)
lang

```

```

# A tibble: 9 x 3
  kommune     aar  arbeidsledighet
  <chr>      <dbl>        <dbl>
1 Oslo        2020         4.2
2 Oslo        2021         5.1
3 Oslo        2022         3.9
4 Bergen      2020         3.8
5 Bergen      2021         4.5
6 Bergen      2022         3.6
7 Trondheim   2020         3.5
8 Trondheim   2021         4
9 Trondheim   2022         3.2

```

Begge formatene inneholder nøyaktig samme informasjon. Men til analyser og plotting i R vil du som oftest trenge data i langt format. Data fra SSB og andre kilder kommer imidlertid ofte i bredt format. Heldigvis er det enkelt å omforme mellom de to formatene.

29.2.1 Fra bredt til langt med pivot_longer()

pivot_longer() gjør data lengre ved å samle flere kolonner til en.

```

bred %>%
  pivot_longer(
    cols = starts_with("arbeidsledighet"),
    names_to = "aar",
    values_to = "arbeidsledighet",
    names_prefix = "arbeidsledighet_"
  )

```

```

# A tibble: 9 x 3
  kommune     aar  arbeidsledighet
  <chr>      <chr>        <dbl>
1 Oslo       2020         4.2

```

2	Oslo	2021	5.1
3	Oslo	2022	3.9
4	Bergen	2020	3.8
5	Bergen	2021	4.5
6	Bergen	2022	3.6
7	Trondheim	2020	3.5
8	Trondheim	2021	4
9	Trondheim	2022	3.2

Her angir vi:

- `cols`: hvilke kolonner som skal omformes (her: alle som starter med “arbeidsledighet”)
- `names_to`: hva den nye variabelen med kolonnenavnene skal hete
- `values_to`: hva den nye variabelen med verdiene skal hete
- `names_prefix`: en tekst som skal fjernes fra kolonnenavnene (slik at vi får “2020” i stedet for “arbeidsledighet_2020”)

Merk at `aar` her blir en tekstvariabel. Hvis du trenger den som numerisk kan du legge til `names_transform = list(aar = as.numeric)` eller gjøre det i et eget steg etterpå.

29.2.2 Fra langt til bredt med `pivot_wider()`

`pivot_wider()` gjør det motsatte: sprer rader utover i kolonner.

```
lang %>%
  pivot_wider(
    names_from = aar,
    values_from = arbeidsledighet,
    names_prefix = "arbeidsledighet_"
  )

# A tibble: 3 x 4
  kommune    arbeidsledighet_2020 arbeidsledighet_2021 arbeidsledighet_2022
  <chr>          <dbl>            <dbl>            <dbl>
1 Oslo           4.2              5.1              3.9
2 Bergen         3.8              4.5              3.6
3 Trondheim     3.5              4                 3.2
```

Her angir vi:

- `names_from`: hvilken variabel som skal gi de nye kolonnenavnene
- `values_from`: hvilken variabel verdiene skal hentes fra
- `names_prefix`: en tekst som legges foran de nye kolonnenavnene

29.2.3 Når trenger du hva?

Noen tomelfingerregler:

- **Langt format** trengs for de fleste analyser i R, inkludert ggplot, regresjonsmodeller og gruppert deskriptiv statistikk. Hvis du skal lage et plott med `ggplot()` der du vil vise utvikling over tid, må dataene være i langt format.
- **Bredt format** er nyttig for å presentere data i tabeller, og noen spesifikke analyser krever bredt format. Data fra SSB og en del andre kilder leveres typisk i bredt format.

I praksis er det vanligst at du må gjøre om fra bredt til langt format, altså bruke `pivot_longer()`. Men det er greit å vite om begge deler.

Disse funksjonene kan virke litt forvirrende i starten, men blir raskt naturlige når man har brukt dem noen ganger. Det beste tipset er rett og slett å prøve selv på egne data.

30 Håndtering av store datasett

```
library(tidyverse)
```

I de foregående kapitlene har vi jobbet med datasett som er relativt små og håndterlige. Men hva skjer når datasettet blir stort? I dette kapittelet ser vi på noen praktiske strategier for når ting begynner å gå tregt – eller når datasettet rett og slett ikke får plass i minnet på datamaskinen din.

30.1 Når blir data “store”?

For de fleste samfunnsvitenskapelige datasett vil du aldri oppleve problemer med størrelsen. Et spørreundersøkelse med noen tusen respondenter og noen hundre variable er egentlig ganske lite i dataverdenen. Men det finnes situasjoner der ting kan bli tyngre:

- Registerdata med millioner av observasjoner over mange år
- Tekstdata, f.eks. fra sosiale medier, med millioner av poster
- Geodata med svært detaljert oppløsning
- Koblede datasett der mange registre er koblet sammen

R holder alle data i minnet (RAM) på datamaskinen. En typisk laptop har 8-16 GB RAM, og operativsystemet bruker en del av dette. Når datasettet nærmer seg størrelsen på tilgjengelig RAM, begynner ting å gå veldig tregt – og til slutt krasjer R.

30.2 Sjekke størrelsen på datasett

Det er nyttig å vite hvor mye plass et datasett tar. Funksjonen `object.size()` gir deg en enkel oversikt:

```
# Lager et eksempel-datasett
eksempel <- data.frame(
  id = 1:100000,
  verdi = rnorm(100000),
  kategori = sample(letters[1:5], 100000, replace = TRUE)
)

object.size(eksempel)
```

2001272 bytes

```
print(object.size(eksempel), units = "Mb")
```

1.9 Mb

For en mer nøyaktig måling kan du bruke `lobstr::obj_size()` som tar hensyn til delte objekter i minnet:

```
library(lobstr)
obj_size(eksempel)
```

Som en tommelfingerregel: et datasett på noen hundre megabyte er uproblematisk. Når det nærmer seg noen gigabyte bør du begynne å tenke på alternativer.

30.3 Effektive filformater: Parquet

Csv-filer er enkle og universelle, men de er trege å lese inn og tar mye plass. For større datasett finnes det langt bedre alternativer. De to viktigste er *feather* og *parquet*, som begge håndteres av pakken `arrow`.

Parquet-formatet er spesielt nyttig fordi det:

- Tar vesentlig mindre plass enn csv (ofte 5-10 ganger mindre)
- Er mye raskere å lese inn
- Bevarer datatyper (du slipper problemer med at tall blir lest inn som tekst)
- Kan leses av både R, Python og mange andre verktøy

Her er et eksempel på hvordan du skriver og leser parquet-filer:

```

library(arrows)

# Skrive til parquet
write_parquet(eksempel, "data/eksempel.parquet")

# Lese fra parquet
eksempel_parquet <- read_parquet("data/eksempel.parquet")

```

Forskjellen i hastighet merkes godt på store filer. En csv-fil som tar 30 sekunder å lese inn kan ta under ett sekund som parquet. Hvis du jobber med store datasett og leser inn de samme dataene flere ganger, er det absolutt verdt å konvertere til parquet.

30.4 data.table: Et raskt alternativ

Gjennom hele denne boken har vi brukt tidyverse og dplyr for datahåndtering. Det fungerer utmerket for de aller fleste formål. Men hvis du jobber med svært store datasett og synes ting går tregt, finnes det et alternativ: pakken `data.table`.

`data.table` er designet for hastighet og minneeffektivitet. Syntaksen er ganske annerledes enn tidyverse, men for noen operasjoner kan den være mange ganger raskere. Her er en liten snakebit:

```

library(data.table)

# Konvertere til data.table
dt <- as.data.table(eksempel)

# Filtrere og aggregere (tilsvarer filter + group_by + summarise)
dt[kategori == "a", .(gjennomsnitt = mean(verdi)), by = kategori]

```

Merk at `data.table` bruker en kompakt syntax med klammeparenteser: `dt[i, j, by]` der `i` er radfiltrering, `j` er hva du vil gjøre med kolonnene, og `by` er gruppering. Det er effektivt, men kan være vanskeligere å lese for nybegynnere.

For de fleste studenter er tidyverse absolutt å anbefale. Men det er greit å vite at `data.table` finnes dersom du en gang jobber med data der hastighet er kritisk. Det går også an å kombinere de to tilnærmingene med pakken `dtplyr`, som lar deg skrive dplyr-kode som kjøres med `data.table` i bakgrunnen.

30.5 Lazy evaluation med Arrow

Hva gjør du når datasettet er *større* enn minnet på datamaskinen? Da kan du bruke `arrow` til å jobbe med data uten å laste alt inn i minnet. Dette kalles *lazy evaluation*: du bygger opp en serie operasjoner, og arrow utfører dem først når du eksplisitt ber om resultatet.

```
library(arrow)

# Åpne datasett uten å laste det i minnet
stort_datasett <- open_dataset("data/stor_mappe/")

# Bygg opp operasjoner (ingenting kjøres ennå)
resultat <- stort_datasett |>
  filter(aar >= 2015) |>
  select(id, aar, inntekt, kommune) |>
  group_by(kommune) |>
  summarise(snitt_inntekt = mean(inntekt))

# Hent resultatet inn i minnet
resultat_df <- collect(resultat)
```

Her er poenget at `filter()`, `select()` og `summarise()` ikke utføres med en gang. Det er først når du kaller `collect()` at arrow faktisk leser og prosesserer dataene – og da leser den bare de delene den trenger. Syntaksen er altså helt vanlig dplyr-kode, men den kjøres på en smartere måte i bakgrunnen.

Dette fungerer spesielt godt med parquet-filer som er delt opp i flere filer i en mappe (såkalt *partisjonert* data).

30.6 Databasetilkoblinger

For virkelig store data, eller data som oppdateres kontinuerlig, er det vanlig å bruke databaser. R kan koble seg til de fleste typer databaser via pakkene `DBI` og `dbplyr`. Med `dbplyr` kan du skrive vanlig dplyr-kode, men i bakgrunnen oversettes det til SQL-spøringer mot databasen.

```
library(DBI)
library(dbplyr)

# Koble til en database (eksempel med SQLite)
con <- dbConnect(RSQLite::SQLite(), "min_database.sqlite")
```

```

# Referere til en tabell i databasen
tabell <- tbl(con, "min_tabell")

# Bruke vanlig dplyr-syntax
resultat <- tabell |>
  filter(aar == 2020) |>
  group_by(kommune) |>
  summarise(antall = n()) |>
  collect()

# Lukke tilkoblingen
dbDisconnect(con)

```

De fleste studenter vil ikke trenge dette, men det er godt å vite at muligheten finnes. Noen forskningsprosjekter benytter databaser som PostgreSQL, MySQL eller Oracle for lagring av store datamengder, og da er dette veien å gå.

30.7 Praktiske tips for store datasett

Uansett hvilke verktøy du bruker, er det noen enkle grep som hjelper:

- Velg kun de kolonnene du trenger.** Ikke les inn 500 variable hvis du bare skal bruke 10. Bruk `select()` så tidlig som mulig.
- Filtrer tidlig.** Hvis du bare trenger data fra et bestemt år eller en bestemt kommune, filtrer *før* du gjør tunge beregninger.
- Bruk effektive datatyper.** Tekstvariable tar mer plass enn numeriske. Konverter kategoriske variable til factor med `as.factor()` for å spare plass.
- Lagre mellomresultater.** Hvis du har gjort en tung bearbeiding, lagre resultatet som en rds- eller parquet-fil slik at du slipper å gjøre det på nytt.
- Unngå unødvendige kopier.** Hver gang du lager et nytt objekt i R, brukes mer minne. Fjern objekter du ikke trenger lenger med `rm()` og frigjør minne med `gc()`.

```

# Fjerne et objekt og frigjøre minne
rm(stort_objekt)
gc()

```

30.8 Når R ikke er nok

I noen sjeldne tilfeller kan det hende at R rett og slett ikke strekker til. Dette gjelder typisk for svært store data (mange titalls gigabyte eller mer) eller oppgaver som krever distribuert beregning. Da kan det være aktuelt å se på:

- **Python** med pandas eller polars for datahåndtering, eventuelt via pakken `reticulate` som lar deg kjøre Python-kode direkte fra R
- **Apache Spark** for distribuert databehandling, tilgjengelig fra R via pakken `sparklyr`

Men for de aller, aller fleste samfunnsvitenskapelige analyser er R mer enn kraftig nok. Og med verktøyene vi har sett på i dette kapittelet – parquet-filer, arrow og eventuelt data.table – kommer du veldig langt.

Referanser

- Moore, David S., Notz William I, and Michael Fligner. 2021. *The Basic Practice of Statistics*. W.H.Freeman & Co Ltd.
- Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science*. Beijing, China: <https://r4ds.had.co.nz/index.html>; O'Reilley.

A Rstudio addins

Rstudio er det mulig å installere såkalte “addins”. Dette gir ikke økt funksjonalitet til R, men til Rstudio. De fleste addins er imidlertid laget med tanke på helt andre brukere enn studenter og kan fremstå nokså kryptiske. Nedenfor er det omtalt tre pakker som kan være nyttige for nybegynnere i R-programmering ved å tilby en “pek-og-klikk” funksjonalitet til å skrive kode. 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”. For en liste over addins, se [hjelpesiden til pakken addinslist](#).

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 for å skrive kode i det lange løp. De som nevnes nedenfor genererer kode for deg og du bør så lime den koden inn i scriptet ditt!

A.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](#)

A.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](#).

A.3 Questionr - omkode factor

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

OBS! Questionr genererer 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. Dessuten funker det, jo.

Du kan se nærmere på [vignetten til questionr](#).

B 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 Sikt¹

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 omrent 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

B.0.1 Hvorfor så vanskelig?

Nedenfor vil det vises en god del komplisert kode bare for å få datasettet over i et håndterbart R-format. Mye styr her. Dette gjør at du lett kan få inntrykk av at R er lite egnet til å håndtere slike data når det trengs så mye jobb. **Men:** Hvis dataene var lagret på en annen måte ville det vært vesentlig enklere. La oss derfor vise hvordan koden ville vært hvis NorLAG var lagret i Stata-format med følgende forutsetninger:

- Ingen rare bruker-definerte missing-verdier
- Alle variable er *enten* kun kontinuerlig *eller* kun kategorisk (dvs. ikke noen spesielle verdier på en annen skala, f.eks. missing-verdier)

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

- Alle data er samlet i én fil

Hvis dataene er konsekvent kodet på denne måten kunne en innlesning se omtrent slik ut:

```
norlag <- read_stata("data/norlag.dta") %>%
  unlabelled()
```

Første linje importerer dataene. Andre linje gjør om alle variable av typen “labelled” til factor-variable der lablene blir omgjort til “factor-levels”. Dette er da alt som trengs.

Men i den virkelige verden er det sjeldent så enkelt. Alle datasett har en del mikk-makk ved seg av både gode og dårlige grunner. Hvordan dataene har blitt til og hvem som har lagt til rette for videre bruk for andre er de to store avgjørende faktorene her.

B.1 Håndtering av user-NAs

For datasettet NorLAG 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.

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.²

²Basert på kode fra <https://tim-tiefenbach.de/post/2023-recode-columns/>

B.2 innlesning av data

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

# data

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

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

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

Når dataene er fordelt på flere filer, som i NorLAG, må de slås sammen. Dette gjøres ved å bruke `merge()`-funksjonen i R. I NorLAG er det en del variable som er i faste og en del som er i intervju. Så er det et eget datasett med variable hentet fra registre. Vi vil ha alle variable i samme datasett.

```
norlag_lbl1 <- merge(faste, lang, by = c("ref_nr"), all.x = 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))
) %>%
  mutate(year = iointervjuar)

norlag_lbl <- merge(norlag_lbl1, register, all.x = TRUE, by = c("ref_nr", "year"))
```

I neste steg benyttes katalogen som ble laget tidligere til å omkode alle variable som har spesifikke missing-verdier til `NA`. Dette gjøres for alle variable i hele datasettet.

```
# 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 tillegg 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. Dette gjøres noe mer manuelt da det bare gjelder akkurat disse variablene.

```

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

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" ~ "Yrkesaktiv",
                                       str_detect(hovedaktivitet, "arbeidsledig") ~ "Trygdet/arbeidsledig",
                                       str_detect(hovedaktivitet, "student") ~ "Trygdet/arbeide",
                                       str_detect(hovedaktivitet, "trygd") ~ "Trygdet/arbeidstrygd",
                                       str_detect(hovedaktivitet, "annet") ~ "Trygdet/arbeidsannet",
                                       str_sub(hovedaktivitet,1,6) == "hjemme" ~ "hjemneværende",
                                       str_detect(hovedaktivitet, "pensjonist") ~ "pensjonist",
                                       is.na(hovedaktivitet) ~ "Trygdet/arbeidsledig/stud/annet"))

```

I NorLAG er det en del variable som omhandler inntekt og er beløp i kroner. I NorLAG har de valgt å legge på en label på noen av disse verdiene. Bildet nedenfor viser hvordan variabelen `inwies` ser ut.

inwies: Inntekt etter skatt NorLAG longitudinell		
Values and categories:		
	-5000	Value <0 >-5000
	5000	Value >0 <5000
	999999999	Mangler data

Det er altså to verdier som er spesifisert som “Value <0 >-5000” og “Value >0 <5000”. Dette er altså en måte å spesifisere at verdien er en verdi som ligger innenfor et intervall. Dette skaper problemer fordi det gjør at R tolker disse som factor-variable i stedet for numeriske variable slik de er ment å være.

Det kan virke praktisk å legge slike labler inn i datasettet slik at man lett ser f.eks. at verdien “-5000” faktisk viser til et intervall og ikke den nøyaktige verdien. Men i dette tilfellet står jo dette allerede i dokumentasjonen og i en konkret analyse vil man jo måtte tolke verdien numerisk uansett. Det kan også bemerkes at de numeriske verdiene er avrundet til intervaller på 10.000 slik at det er ganske logisk at det også gjelder for “-5000” og “5000”. Vi velger derfor å omkode disse til numeriske verdier og slette lablene.

Men det er en hel rekke variable som har labler på tilsvarende måte. Alle like hensiktsløst. Vi kan derfor bruke `across()` for å kode om på samme måte for alle variable samtidig.

Heldigvis ligger disse inntektsvariablene i rekkefølge i datasettet slik at vi kan spesifisere variablene med “fra:til”. Den første variabelen er `inarbled` og den siste er `inwyrkinnt` og det kan da skrives som `inarbled:inwyrkinnt`.

Her er koden som gjør dette:

```
norlag <- norlag %>%
  mutate(across(inarbled:inwyrkinnt,
    ~ case_when(. == "Value <0 >-5000" ~ -5000,
                . == "Value >0 <5000" ~ 5000,
                TRUE ~ as.numeric(as.character(.)))
    )
  )
```

Og dermed har vi et datasett i et svært så ryddig R-format. Eller i hvert fall det aller meste. Det vil alltid være behov for å gjøre ytterligere omkoder for en spesifikk analyse, og det kan være andre generelle ting som ikke har blitt tatt hånd om her.

B.3 Hvordan fungerer koden ovenfor?? En intro til mer avansert databehandling

Koden ovenfor er ganske avansert. Det er en del ting som er litt mer avansert enn det vi har gått gjennom tidligere. Vi skal her gå gjennom de viktigste elementene i koden.

B.3.1 `across()`

`across()` er en funksjon som brukes for å gjøre noe på tvers av kolonner. Det er en del av `dplyr`-pakken. Den brukes for å gjøre noe på tvers av kolonner. I koden ovenfor brukes den for å gjøre om alle variable som er inntektsvariable til numeriske variable.

B.3.2 case_when()

`case_when()` er en funksjon som brukes for å gjøre en rekke sammenligninger og returnere en verdi basert på disse sammenligningene. Den brukes i koden ovenfor for å gjøre om lablene på inntektsvariablene til numeriske verdier.

B.3.3 fra factor til numerisk

I koden ovenfor gjøres det om fra factor til numerisk ved å bruke `as.numeric(as.character(..))`. Dette er fordi en factor ikke kan gjøres om til numerisk direkte.

C 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 ha 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
```

C.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).

```
# read html file

cb <- read_html("data/Kodebok.html")
```

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 ``. Tilsvarende koder finnes for tabeller og andre elementer. Disse delene har er oftest gitt et navn som man kan identifiseres og brukes til å 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)
```

C.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]), split = ":" )[[1]][1],
             spm = strsplit(as.character(.[,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")
```

D Empirisk eksempel

Artikkelen “[Human Values and Retirement Experiences: a Longitudinal Analysis of Norwegian Data](#)” bruker data fra NorLAG, runde 2 og 3. I det som følger vil hovedresultatene fra denne studien replikeres. Stor takk til Morten Blekesaune som var meget velvillig til å dele script slik at reprodusering er praktisk mulig.

D.1 Utvalg av variable

Alle variable er dokumentert på NorLAG sine sider, og i egen fil som følger med utdelt datasett “kodebok.html”.

```
library(tidyverse)

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.2.0     v readr     2.1.5
v forcats   1.0.1     v stringr   1.6.0
v ggplot2   4.0.2     v tibble    3.3.1
v lubridate 1.9.4     v tidyr    1.3.1
v purrr    1.2.1

-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become non-conflicting
```

```
library(haven)

infilbane <- "C:/Users/torbskar/OneDrive - Universitetet i Oslo/Dokumenter/Undervisning/SOS4000/tilgangsfil/variables.dta"

# Leser inn dta-fil, velger aktuelle variable og koder om missing-verdier til NA.
# drop_na() sletter observasjoner med NA-verdier. Merk at dette gjøres før de variablene som
register <- read_stata( paste0(infilbane, "NorLAG-lengde-register.dta"), encoding = "utf-8")
select(ref_nr, year,
```

```
    inpgivinnt, inwoverfor, inftryg, inwyrkinnt, inwoverfor,    # pensj_ft overf_ft pensj
    inkode217, inkode218 ) %>%
mutate(inpgivinnt = ifelse(inpgivinnt == 999999999, NA, inpgivinnt),
       inwoverfor = ifelse(inwoverfor == 999999999, NA, inwoverfor),
       inftryg = ifelse(inftryg == 999999999, NA, inftryg)) %>%
drop_na()
```

E Leser inn data. Bruker labelled for enklere oversettelse av stata-kode

```
norlag <- readRDS("data/norlag_labelled.rds") %>% select(ref_nr, round, iodeltakelse,
iointervjuar, # ref_nr round io_intervjuyr
iofodselsaar, ioalder, iokjonn, # io_ioalder
wb007, wr004, # wb007 wr004 hcPCS12, starts_with("vahv") # hcPCS12 va_bhv*
#vahvssen, vahvsstr, vahvsotc, vahvscon ) %>% filter(round > 1, iodeltakelse %in% c(1,3))
%>% mutate(alder = as.numeric(ioalder), satis = as.numeric(wb007)) %>% mutate(ald10
= alder/10, ald2 = ald10^2, ald3 = ald10^3, year = iointervjuar) %>% filter(alder < 86,
satis < 15, wr004 <= 5, !is.na(wr004), !is.na(satis)) %>% arrange(ref_nr, round, year) %>%
filter(complete.cases(.))
```

F Kobler på register, beholder bare treff på begge.

```
norlagreg <- merge(norlag, register, by = c("ref_nr", "year") ) %>% mutate(hcPCS12 = hcPCS12/10, fyshel1 = hcPCS12) %>% group_by(ref_nr) %>% mutate(antobs = n()) %>% # Filtrerer på de som har akkurat to observasjoner. Altså: deltar i begge runder. filter(antobs == 2) %>% select(-antobs) %>% group_by(ref_nr) %>% mutate(pensj_ft = mean(inftyg), #“Pensjoner folketrygden” overf_ft = mean(inwoverfor), #“Overføringer i alt” penal_ft = mean(inkode217), # “Alderspensjon” penuf_ft = mean(inkode218), # “Uførepensjon” penin_ft = mean(inpgivinnt) #“Pensjonsgivende inntekt” )  
dim(norlagreg)  
summary(norlagreg$alder)
```

G Utvalgsstørrelse og personer

```
norlagreg %>% group_by(round) %>% ungroup() %>% summarise(personer = n_distinct(ref_nr),  
observasjoner = n())  
table(is.na(norlagreg$satis))
```

G.1 Etablering av utvalg for analyse

G.2 Deskriptiv statistikk

G.3 En kommentar om reproducertbarhet

Det er svært viktig at forskning lar seg reproduksjon. I denne sammenhengen gjelder det å kunne reproduksjon nøyaktig samme resultat på samme data, som er viktig for uavhengig ettergåelse og kvalitetkontroll. At reproducertbart script kan gjøres tilgjengelig er helt avgjørende, men også at data er tilgjengelig. I dette tilfellet har Morten Blekesaune vært meget velvillig delt script etter forespørsel. Han svarte på epost samme dag, noe som er et tydelig tegn på at han også hadde orden i sakene og visst godt hvor scriptet var. Sånn skal det være!

Data var vanskeligere. NorLAG er ikke åpne data og er ikke opp til forskeren å dele. Vi er derfor avhengig av at også Sikt har orden i sakene. NorLAG er publisert i flere versjoner der det har vært noen endringer underveis. Det ble derfor praktisk vanskelig å få etablert nøyaktig samme utvalg uten å også få nøyaktig samme datauttrekk utlevert. Hvis ikke dataleverandør kan gi nøyaktig samme data, så blir heller ikke scriptet nøyaktig reproducert.

En viktig side ved å dele scriptet er at det gjøres en del små valg i den faktiske analysen som typisk ikke fremgår med tilstrekkelig detaljgrad i forskningsartikkelen.

G.3.1 Replikering på uavhengige data

Mer generelt er det også viktig å kunne replikere *tilsvarende resultater* på andre data, men det har et litt annet formål. For slik replisering er det imidlertid også viktig å ha nøyaktig informasjon om hva som ble gjort i den opprinnelige studien og data er godt dokumentert.