

# Introduktion til R

Erik Gahner Larsen

Oktober 27, 2017

# Indhold

<b>Kapitel 1: Introduktion</b>	<b>1</b>
1.1 Hvorfor R? Fordele og ulemper ved R	2
1.2 Installation af R og RStudio	3
<b>Kapitel 2: Fundamentet</b>	<b>5</b>
2.1 Objekter og funktioner	5
2.2 Tal i objekter	6
2.3 Logiske operatoren	6
2.4 Manglende værdier (NA)	10
2.5 Tekst i objekter	11
2.6 Datarammer	12
2.7 Import og eksport af datarammer	19
2.8 Installation af pakker	20
2.9 Objekter i hukommelsen	21
<b>Kapitel 3: Databehandling</b>	<b>23</b>
3.1 Behandling af datarammer med dplyr	25
3.2 Vælg bestemte variable med select()	26
3.3 Vælg bestemte observationer med filter()	27
3.4 Vælg rækkefølgen på observationer med arrange()	28
3.5 Skift navnet på en variabel med rename()	29
3.6 Tilføj variabel med mutate()	29
3.7 Kombination af funktioner med pipe operatoren	30
3.8 Kør funktioner på variable med apply()	31
3.9 Aggreger variable med summarize()	32
<b>Kapitel 4: Visualisering</b>	<b>34</b>

4.1	Introducerende eksempel med ggplot2 . . . . .	36
4.2	Specifikationer med ggplot2 . . . . .	40
4.3	Eksempel: Venstre i meningsmålingerne i 2015 . . . . .	43
4.4	Afsluttende bemærkninger . . . . .	49
<b>Kapitel 5: OLS regression . . . . .</b>		<b>51</b>
5.1	Bivariat analyse . . . . .	53
5.2	Multivariat analyse . . . . .	56
5.3	Forudsætningstests . . . . .	59
5.3.1	Linearitet . . . . .	59
5.3.2	Outliers og indflydelsesrige observationer . . . . .	61
5.3.3	Normalfordelte fejllæd . . . . .	64
5.3.4	Heteroskedasticitet . . . . .	66
5.3.5	Multikollinearitet . . . . .	69
<b>Kapitel 6: Interaktionsanalyse . . . . .</b>		<b>71</b>
<b>Kapitel 7: Logistisk regression . . . . .</b>		<b>72</b>
<b>Kapitel 8: Propensity Score Matching . . . . .</b>		<b>73</b>
<b>Kapitel 9: Regressionsdiskontinuitetsdesigns . . . . .</b>		<b>80</b>
<b>Konklusion . . . . .</b>		<b>81</b>
4.1	Pakkeanbefaling: swirl . . . . .	81
<b>Bilag A: Funktioner . . . . .</b>		<b>83</b>
A.1	Loops . . . . .	85
<b>Bilag B: Genveje og udvalgte funktioner . . . . .</b>		<b>87</b>
B.1	Funktioner . . . . .	87
B.2	Genveje i RStudio . . . . .	88
<b>Bilag C: Anbefalede pakker . . . . .</b>		<b>89</b>
<b>Bilag D: Eksport af tabeller . . . . .</b>		<b>90</b>
<b>Referencer . . . . .</b>		<b>91</b>

# Kapitel 1

## Introduktion

Denne bog giver dig en indføring i, hvordan man bruger R til statistiske analyser. Kort fortalt er der ingen grænser for, hvad man kan bruge R til, så nærværende introduktion er på ingen måde udtømmende, men skal blot ses som et grundlag for, at kunne gennemføre bestemte analyser. R er *ikke* nemt at lære, men det har væsentlige styrker, der gør, at det er bedre end alternativerne (Excel, SPSS, Stata, SAS m.v.).

Bogens ambition er at give en introduktion til tre vigtige stadier i statistiske analyser. For det første skal vi bearbejde data. Dette kan blandt andet være ved at konstruere variable med bestemte informationer, men også ved at downloade data og importere disse. For det andet skal der gennemføres analyser af ens datamateriale. Dette kan være en simpel test for forskelle i to gennemsnit, men også mere komplicerede analyser. For det tredje skal resultaterne af ens analyser præsenteres på den mest pædagogiske og informative måde, dette være sig enten i tabeller eller figurer.

Der findes talrige bøger og ressourcer på nettet, der giver en introduktion til R. En del af dette materiale er uden tvivl mere dybdegående, end det du vil finde her. Der vil af samme grund blive henvist til en del anbefalsværdigt eksternt materiale i løbet af bogen. Ambitionen med denne bog er udelukkende at give en pædagogisk introduktion til især statskundskabsstuderende, der ønsker en letlæselig introduktion til R på dansk.

I dette kapitel gives en introduktion til R. Dette sker ved først at beskrive nogle af styrkerne og svaghederne ved R, hvorefter det gennemgås, hvordan R installeres og ser ud. Til sidst introduceres logikken bag R, der klæder os på til at bruge R i de kommende kapitler. De efterfølgende kapitler vil blandt andet fokusere på, hvordan man bearbejder og visualiserer sine data, gennemfører lineære regressionsanalyser samt andre statistiske analyser.

Materialet der anvendes i bogen, kan hentes på GitHub: <https://github.com/erikgahner/Rguide>. Den nyeste version af bogen kan findes her:

- Online: <http://erikgahner.dk/Rguide/>
- PDF: <http://erikgahner.dk/Rguide/Rguide.pdf>
- EPUB: <http://erikgahner.dk/Rguide/Rguide.epub>

Hvis du finder fejl og mangler i bogen, må du meget gerne oprette et issue på GitHub eller sende en mail til [erikgahner@gmail.com](mailto:erikgahner@gmail.com). Har du en idé eller et forslag til, hvad der vil kunne styrke bogen, er du også meget velkommen til at kontakte mig på nævnte mail.

## 1.1 Hvorfor R? Fordele og ulemper ved R

R hjælper dig effektivt fra A til B, men som det også blev beskrevet indledningsvist: R kan være svært - og det tager tid at lære. Der findes masser af statistikprogrammer på markedet, der kan gennemføre statistiske analyser, og mange af disse er nemmere at lære end R. Skal man udelukkende bruge et program til at lave lagkagediagrammer, er det ikke din tid værd at lære R. Med andre ord: Hvis distancen fra A til B er at betegne som gåafstand, giver det ingen mening at lære at køre en Lamborghini (i dette tilfælde R).

Hvorfor så bruge R? For det første er det gratis. Ja, *gratis*. Det er muligt, at du allerede har “gratis” adgang til Stata eller SPSS gennem dit universitet eller arbejde, men dette er ikke det samme som, at du vil have adgang for evigt. Tværtimod. Når du bruger et gratis program er du fri for at tænke på, hvilken licens du bruger og hvor stor din pengepung er. Ikke bare nu, men også i fremtiden.

For det andet giver R adgang til en række muligheder, der kun i begrænset omfang er muligt at gennemføre i andre programmer. Dette både hvad angår bearbejdning, analyse og præsentation af data. R er eksempelvis nemt at anvende til at indsamle og analysere forskellige typer af data fra internettet, herunder også fra sociale medier som twitter. R giver med sin objektorienterede struktur (der bliver introduceret i næste kapitel) således mulighed for at arbejde med data på en anden måde end Stata gør. Hvor man i Stata kun kan have ét datasæt åbent af gangen, giver R rig mulighed for at arbejde med og kombinere flere forskellige datasæt.

For det tredje giver R mulighed for at præsentere og visualisere data og analyser på langt pænere måder end andre programmer. Hvis du ser en flot figur i en videnskabelig artikel, vil det bedste gæt være, at den er lavet i R. Flere store medievirksomheder bruger således R - og især en pakke ved navn `ggplot2`, når de skal lave flotte visualiseringer.

For det fjerde er der et stort *community* af brugere, der meget gerne står til rådighed og hjælper, hvis du møder et problem. Den gode nyhed er, at du ikke er den første (eller den sidste), der skal til at lære R, hvorfor der er mange brugere, der har haft de samme problemer, som du

kommer til at have. Hvis du derfor får en fejlmeddelelse (og tro mig - det gør du!), kommer du som regel langt ved blot at *google* fejlmeddelelsen, hvor du kan finde information omkring, hvad der er galt og hvordan problemet som regel kan løses.

For det femte er det nemmere at lære et nyt statistikprogram, hvis man kan R, end omvendt. Hvis man bliver bekendt med, hvordan statistiske analyser gennemføres i R, er det relativt nemt at lære at bruge SPSS, Stata og andre programmer. Det samme er ikke nødvendigvis tilfældet, hvis man først lærer eksempelvis SPSS, hvor der kan være mange dårlige vaner, man først skal vænne sig af med. Hvis du lærte statistik med SPSS er du potentielt set begrænset af den menulinje, SPSS benytter sig af.

For det sjette faciliterer brugen af R et øget fokus på reproducerbarheden af ens resultater. Når man laver noget i R, gør man det som regel gennem funktioner, altså kommandoer (i et *script*), der er let at dokumentere. Dette gør, at man nemt kan sende sit datasæt og R-script til en kollega eller ven, der kan køre *samme* script på det *samme* data og (forhåbentlig) få de *samme* resultater. Det samme er muligt med både SPSS og Stata, men disse programmer giver også rig mulighed for, at man nemt kan omkode variable og gennemføre analyser, uden at man nødvendigvis husker at dokumentere processen heraf.

På baggrund af ovenstående liste (der ikke er udtømmende), burde det være åbenlyst, at R er værd at bruge tid på. Når dette er sagt, er der ikke desto mindre nogle ulemper forbundet med R. For det første er det, som beskrevet, svært at lære at bruge R. For det andet er der ikke en officiel supportlinje, man kan ringe til, hvis man støder ind i problemer. Dette ændrer dog ikke på, at fordelene ved at lære R langt overstiger ulemperne, så næste afsnit viser, hvordan du installerer R.

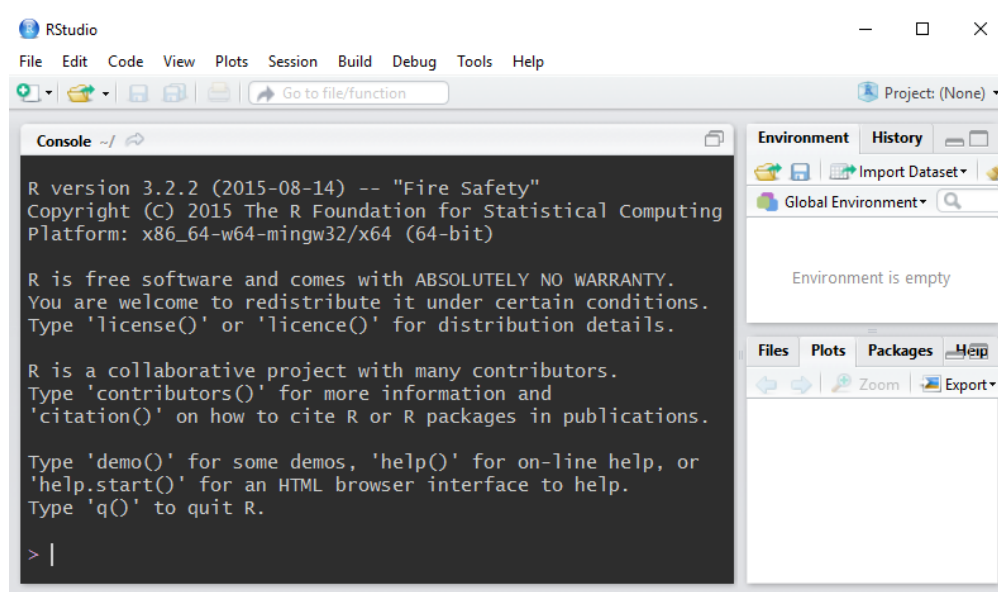
## 1.2 Installation af R og RStudio

Der er to programmer, du skal installere. Det første er R, som er basispakken, der *skal* installeres. Det andet er RStudio, der er det program, du kommer til at anvende. RStudio er et integreret udviklingsmiljøbrugerflade (*integrated development environment*, IDE), der gør R nemmere at anvende. RStudio er ikke alene langt pænere end basispakken, men rummer også en lang række værktøjer, der gør R nemmere at lære og bruge. Her er de step, du kan følge:

1. Gå ind på [r-project.org](http://r-project.org) og klik på menupunktet CRAN under *Download*.
2. Du er nu på en side med titlen *CRAN Mirrors*. Vælg en af hjemmesiderne, hvorfra du vil hente R.
3. Vælg hvilket styresystem, du vil hente R til (Windows, Linux eller Mac).

4. Hvis du bruger Windows, så klik først på *base* og derefter det link, der indeholder *download*. Hvis du bruger Mac, så klik på den øverste .pkg fil.
5. Når filen er downloadet, kan du følge installationsguiden og dermed installere R.
6. Efter installationen går du ind på RStudios side, hvor RStudio kan downloades: [rstudio.com/products/rstudio/download/](http://rstudio.com/products/rstudio/download/).
7. Klik på den fil under *Installers for Supported Platforms*, der matcher dit styresystem.
8. Når filen er downloadet, kan du følge installationsguiden og dermed installere RStudio.

Ovenstående bør ikke medføre nogle problemer. Hvis der af en eller anden grund skulle komme en fejlmeddelelse, kan det varmt anbefales blot at *google* denne. Du har nu installeret R og RStudio og kan åbne RStudio, der med nogle visuelle forskelle (farverne og evt. styresystem), ligner skærbilledet i Figur 1.1.



Figur 1.1: Det grafiske interface i RStudio

Du er nu i R. Når du ikke har et script åbent (i en *editor*, beskrevet i næste kapitel), er der tre vinduer. For det første er der konsollen, hvor du kan skrive kommandoer. Det er ligeledes her mange resultater og lignende, vil blive vist. For det andet er der dit miljø (*environment*), hvor det bliver synligt, hvilke ting du arbejder med samt hvad du tidligere har gjort. For det tredje har du et *output* vindue, hvor blandt andet grafer, hjælpedokumenter og lignende vil blive præsenteret.

Når du har installeret R, handler det først og fremmest om at lære det helt grundlæggende. I næste kapitel tager vi fat på dette.

# Kapitel 2

## Fundamentet

### 2.1 Objekter og funktioner

Alt du laver i R, kan skrives som kommandoer. Dette sikrer, at du altid kan dokumentere dit arbejde, modsat hvis du eksempelvis bruger menulinjer, hvor det ikke altid er klart, hvilke analyser, der er gennemført. Nederst i programmet ser du en prompt (>), hvor du kan skrive, hvad R skal gøre. Prøv at skrive 2+2 og tryk ENTER. Dette burde gerne resultere i følgende:

```
2+2
```

```
[1] 4
```

Ovenstående viser hvilken kommando, der er kørt, samt resultatet heraf<sup>1</sup>. Da du kommer til at køre mange forskellige kommandoer, hvor mange skal køres i en bestemt rækkefølge, er det godt allerede nu at begynde at dokumentere, hvad du gør. Den bedste måde at gøre dette er i en script-fil (R), også kaldet et R-script. Åbn et nyt R-script ved i menuen at vælge File → New File → R Script.

Sørg for allerede nu at dokumentere dit arbejde. Det vil sige, at alle kommandoer du bruger, kan skrives ind i dit R-script. Sørg desuden for at skrive kommentarer i R-scriptet, så du og andre kan se beskrivelser af, hvad der gøres. Kommentarer begynder med # (for at fortælle R, at den ikke skal læse teksten som kode), og kan tilføjes på deres egne linjer eller i forlængelse af en kommando<sup>2</sup>.

Når du har indtastet noget kode i dit R-script, kan du køre det i konsollen ved at markere koden og bruge tastaturgenvejen CTRL+R (Windows) eller CMD+ENTER (Mac). Forsøg at indtaste

---

<sup>1</sup>Dette svarer til at skrive `display 2+2` i Stata.

<sup>2</sup>Dette svarer til `*` og `//` i Stata.



nedenstående kode, marker det hele og kør det.

```
50*149
3**2      # 3^2
2**3      # 2^3
sqrt(81)   # 81^0.5
```

## 2.2 Tal i objekter

Du er nu i stand til at bruge R som en lommeregner. Det næste vi skal have styr på er objekter. Kort fortalt er alt hvad vi vil bruge i R, gemt i objekter. Dette være sig lige fra ét tal til hele datasæt. Fordelen ved dette er, at vi kan have flere datasæt åbne i hukommelsen på samme tid gemt som hvert deres objekt (modsat eksempelvis Stata). Med andre ord kan *alt* vi arbejder med i R gemmes i objekter. Lad os forsøge at gemme tallet 2 i objektet x.

```
x <- 2
```

Når du kører ovenstående kommando, gemmer du tallet 2 i objektet x. Du kan nu bruge x i stedet for 2. Lad os prøve en række forskellige simple operationer. Indtast dem i dit R-script og kør dem én efter én.

```
x
x * 2
x * x
x + x
```

Når du kører disse linjer, burde du gerne få værdierne 2, 4, 4 og 4. Hvis du ændrer x til at have værdien 3, vil du kunne køre linjerne igen og få andre værdier<sup>3</sup>. Generelt, når du laver scripts, må du gerne arbejde på at få så mange informationer til at være i objekter, så du er fri for at ændre tal mere end én gang, hvis du skal lave ændringer<sup>4</sup>.

## 2.3 Logiske operatører

En stor del af det vi skal lave i R, bygger på logiske operatører. Med en logisk operator tester vi sandhedsværdien af et udsagn, der kan være enten sand eller falsk. Dette bliver især brugbart

---

<sup>3</sup>Helt præcist 3, 6, 9 og 6.

<sup>4</sup>En anden fordel er, at du på denne måde reducerer sandsynligheden for, at lave fejl ved at have forskellige informationer flere steder.

når vi skal lave omkodninger og kun bruge bestemte værdier i et objekt. I R er en logisk operator TRUE (sand) eller FALSE (falsk). Kør nedenstående kode og se, hvad de respektive kommandoer returnerer.

```
x == 2      # "==" betyder "lig med"
x == 3
x != 2      # "!=" betyder "ikke lig med"
x < 1       # "<" betyder "mindre end"
x > 1       # ">" betyder "større end"
x <= 2      # "<=" betyder "mindre eller lig med"
x >= 2.01   # ">=" betyder "større eller lig med"
```

Hvis x er 2, vil værdierne være hhv. TRUE, FALSE, FALSE, FALSE, TRUE, TRUE og FALSE. Hvis x ændres til 3 og scriptet køres igen, vil andre sandhedsværdier returneres.

Objekter kan videre bruges til at skabe andre objekter. I følgende eksempel laver vi et nyt objekt y, der giver os summen af x og 7. Bemærk desuden at hele kommandoen er skrevet i en parentes, der gør, at vi også får værdien af y returneret. Hvis vi ikke gør dette, laver vi objektet y, men uden at få det vist med det samme.

```
(y <- x + 7)
```

```
[1] 9
```

I vores objekter er vi heller ikke begrænset til kun at have ét tal. Tværtimod vil de fleste objekter vi arbejder med have mere end én værdi. Nedenstående giver således en talrække med tallene fra 1 til 10.

```
1:10
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

Denne talrække kan vi gemme i et objekt (med <-), men også bruge direkte uden at have den i et objekt. Vi kan eksempelvis tage hvert tal i talrækken og addere 2 til hvert tal i rækken.

```
1:10 + 2
```

```
[1] 3 4 5 6 7 8 9 10 11 12
```

Når der skal arbejdes med flere tal, kan vi ikke blot skrive en talrække. Først skal R vide, at det er en talrække, der arbejdes med. Til dette bruger vi `c()`, der fortæller R, at vi arbejder med vektorer<sup>5</sup>. Funktionen `c()` står for *concatenate* eller *combine*<sup>6</sup>. Alt der sker i R, sker med funktioner. En vektor kan således se ud som følger.

```
c(2, 2, 2)
```

```
[1] 2 2 2
```

Ovenstående er en numerisk vektor. En vektor er således en samling af værdier af samme type af data. Hvis vi gerne vil gemme vektoren i et objekt, kan det også lade sig gøre uden problemer. I nedenstående gemmer vi fire tal (14, 6, 23, 2) i objektet `x`.

```
x <- c(14, 6, 23, 2)
x
```

```
[1] 14 6 23 2
```

Denne vektor kan vi behandle efter forgodtbefindende, eksempelvis ved at få alle værdierne i vektoren multipliceret med 2.

```
x * 2
```

```
[1] 28 12 46 4
```

Vi kan ligeledes hente information ud af vektoren. Til at gøre dette skal vi bruge de firkantede parenteser, altså `[ ]`, som placeres i forlængelse af objektet. Ved at placere tallet 3 i den firkantede parentes (på engelsk kaldet *brackets*), får vi det tredje tal i vektoren.

```
x[3]
```

```
[1] 23
```

---

<sup>5</sup>I eksemplet med `1:10` svarer det til at vi skriver `c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`. I `1:10` er der således en skjult funktion, `c()`.

<sup>6</sup>`c()` opretter således en vektor med alle elementer i parentesen. Da en vektor kun kan indeholde én type data, og ikke eksempelvis både numre og karakterer (der bliver introduceret i næste afsnit), vil `c()` også sikre, at de værdier der gives, reduceres til det maksimale niveau. Er der således blot én værdi der er karakterbaseret, vil alle andre værdier i vektoren også blive det.

På samme måde kan vi også få alle værdier i vektoren med undtagelse af et bestemt tal, blot ved at tilføje et minustegn til parentesen. I nedenstående eksempel får vi vektoren uden tal nummer to ved at skrive -2. Bemærk også, at vores objekt `x` ikke ændres, da vi ikke overskriver vores objekt (med `<-`).

```
x[-2]
```

```
[1] 14 23 2
```

Med udgangspunkt i vores objekt, kan vi bruge en række funktioner til at få nogle informationer ud omkring vores objekt, som gennemsnittet af værdierne, den mindste værdi med videre.

```
length(x)      # antallet af numre i vektoren
min(x)         # minimumsværdien
max(x)         # maksimumværdien
median(x)      # medianen
sum(x)         # summen
mean(x)        # gennemsnittet
var(x)         # variansen
sd(x)          # standardafvigelsen
```

Ovenstående skulle gerne returnere værdierne 4, 2, 23, 10, 45, 11.25, 86.25 og 9.287088. Vi kan bruge resultaterne fra de forskellige funktioner til at undersøge om eksempelvis kvadratroden af variansen er lig standardafvigelsen. Dette er, som man lærer i introducerende statistikfag, tilfældet.

```
sqrt(var(x)) == sd(x)
```

```
[1] TRUE
```

Hvis vi har glemt at tilføje et tal til vores vektor, er der heldigvis en nem måde at opdatere vores vektor og gemme det i objektet. Dette kan vi gøre ved at overskrive vores objekt (eller lave et nyt), med en ny vektor der består af vores objekt og en ekstra værdi:

```
x <- c(x, 5)
```

```
x
```

```
[1] 14  6 23  2  5
```

Som det kan ses, er der nu fem værdier i vores vektor i stedet for fire. Værdien 5, der blev tilføjet, har den sidste placering i vektoren, da vi placerede den til sidst, da vi lavede et nyt objekt. Hvis vi havde placeret 5 før x, ville vi have værdien 5 i begyndelsen af den nye vektor. Vi kan eksempelvis forsøge at tage gennemsnittet af vores opdaterede objekt.

```
mean(x)
```

```
[1] 10
```

Nu er gennemsnittet 10 (før vi tilføjede værdien 5 var gennemsnittet 11.25).

## 2.4 Manglende værdier (NA)

Heldigvis har alle værdier, vi har arbejdet med til nu, været numeriske og nemme at arbejde med. I de fleste af de data, vi arbejder med, er der dog også manglende værdier, altså værdier, vi ikke ved hvad er. Hvis man arbejder med større datasæt vil der sjældent være objekter, der ikke rummer manglende værdier. I Stata betegnes manglende værdier med et punktum (.), hvor der i R bruges NA. Lad os tilføje en manglende værdi til vores objekt x og tage gennemsnittet af det nye objekt.

```
x <- c(x, NA)
```

```
mean(x)
```

```
[1] NA
```

Som det kan ses, får vi nu ikke et gennemsnit, men blot NA. Dette skyldes, at R ikke kan finde gennemsnittet af en vektor, hvor NA er med. Heldigvis kan vi tilføje en ekstra specifikation til `mean()`, der fortæller, at den skal fjerne manglende værdier, før den tager gennemsnittet.

```
mean(x, na.rm=TRUE)
```

```
[1] 10
```

Her får vi gennemsnittet 10 (ligesom ovenfor, før vi tilføjede NA). Bemærk at der er tilføjet et komma og `na.rm=TRUE`. De fleste funktioner i R har en lang række af ekstra specifikationer, man kan tilføje. Som standard er `na.rm` sat til `FALSE`, hvorfor det kræver, at man ændrer dette, hvis man har manglende værdier i sine data.

## 2.5 Tekst i objekter

Foruden tal kan vi også arbejde med tekst. Tekst i R adskiller sig fra tal ved, at tekst pakkes ind i citationstegn<sup>7</sup>. Som eksempel kan vi lave et objekt `z`, der indeholder partierne Venstre og Socialdemokraterne.

```
z <- c("Venstre", "Socialdemokraterne")
```

```
z
```

```
[1] "Venstre"          "Socialdemokraterne"
```

For at se hvilken type data, vi har i `z`, kan vi bruge funktionen `class()`. Denne funktion returnerer, hvilken type data vi har at gøre med i et objekt. Hvis vi bruger funktionen på vores objekt, ser vi, at det pågældende objekt med tekst indeholder karakterer (altså *“character”*).

```
class(z)
```

```
[1] "character"
```

Til sammenligning kan vi gøre det samme med vores objekt `x`, der som bekendt kun har numeriske værdier. Her ser vi, at funktionen `class()` for `x` returnerer `"numeric"`. De forskellige klasser en vektor kan have er hhv. `character` (tekst), `numeric` (numeriske tal), `integer` (hele tal), `factor` (kategorier) og `logical` (logisk).

---

<sup>7</sup> Alternativt kan man også bruge `'` i stedet for `"`.

```
class(x)
```

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

Hvis vi vil vide, om vores objekt er numerisk, kan vi bruge funktionen `is.numeric()`, der returnerer TRUE, hvis objektet er numerisk. På samme måde kan man også bruge funktionen `is.character()`. I eksemplet returnerer de hhv. TRUE og FALSE.

```
is.numeric(x)
```

```
is.character(x)
```

Prøv gerne at bruge funktionerne `is.numeric()` og `is.character()` på objektet `z`. Med vores objekt `z` og de resterende partinavne repræsenteret i Folketinget i 2016, kan vi lave et objekt med navnet `party`. I scriptet vil `z` automatisk blive erstattet med Venstre og Socialdemokraterne, som vi tildelte til `z` i ovenstående (med andre ord kan vi også lave nye objekter med vores eksisterende objekter, når det kommer til tekst). Bemærk at placeringen på navnet i objektet, når vi får vist alle partinavnene, er angivet i tallene i de firkantede parenteser.

```
party <- c(z, "Enhedslisten", "SF", "Radikale", "Konservative",  
          "Dansk Folkeparti", "Liberal Alliance", "Alternativet")
```

```
party
```

```
[1] "Venstre"          "Socialdemokraterne" "Enhedslisten"  
[4] "SF"              "Radikale"          "Konservative"  
[7] "Dansk Folkeparti" "Liberal Alliance"  "Alternativet"
```

## 2.6 Datarammer

For de partier vi arbejder med, vil vi gerne tilføje mere information. Derfor laver vi nogle ekstra objekter, der indeholder information om, hvorvidt det er et højreorienteret parti (`rw`, forkortelse for *right-wing*), hvor mange stemmer det fik ved folketingsvalget i 2015 (`vote`) og hvor mange mandater partiet fik (`seat`). Disse objekter laver vi med nedenstående kode. Bemærk at rækkefølgen af værdierne er afgørende, og skal matche rækkefølgen af partierne i `party` (så vi begynder med Venstre og ender med Alternativet).

```
rw <- c(1, 0, 0, 0, 0, 1, 1, 1, 0)
vote <- c(19.5, 26.3, 7.8, 4.2, 4.6, 3.4, 21.1, 7.5, 4.8)
seat <- c(34, 47, 14, 7, 8, 6, 37, 13, 9)
```

Det næste vi skal gøre er at samle disse objekter til ét objekt. Dette gør vi i en dataramme (*data frame*), der kan sammenlignes med et datasæt i Stata. En dataramme er kort fortalt en samling af forskellige vektorer, der har den samme længde, og derved kan sættes sammen som kolonner. Bare rolig - hvis du forsøger at sætte forskellige vektorer sammen, der ikke har samme længde, skal R nok give dig en fejlmeddelelse.

I en dataramme kan vi have forskellige typer af variable, der kan gennemføres analyser på. Der findes andre typer af objekter i R, eksempelvis også matricer, men vi vil for nu udelukkende fokusere på datarammer. Til dette bruger vi funktionen `data.frame()` og gemmer det i objektet `pol`.

```
pol <- data.frame(party, vote, seat, rw)
```

Nu kan vi bruge `class()` til at vise, at `pol` er en dataramme:

```
class(pol)
```

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

Hvis vi gerne vil vide, hvilken *class* de enkelte variable i vores dataramme er, kan vi bruge funktionen `sapply()`. Funktionen gør det muligt at applicere en funktion på en liste eller en vektor, hvor vi i nedenstående applicerer `class()` på alle de enkelte variable i objektet `pol`:

```
sapply(pol, class)
```

```
party      vote      seat      rw
"factor" "numeric" "numeric" "numeric"
```

Her kan vi se, at vi har én factor (partinavnene) og tre numeriske variable. Lignende informationer om vores variable i datarammen kan vi få ved at bruge `str()`, der giver data på strukturen i datarammen:



```
str(pol)
```

```
'data.frame':  9 obs. of  4 variables:
 $ party: Factor w/ 9 levels "Alternativet",...: 9 8 3 7 6 4 2 5 1
 $ vote : num  19.5 26.3 7.8 4.2 4.6 3.4 21.1 7.5 4.8
 $ seat : num   34 47 14 7 8 6 37 13 9
 $ rw   : num   1 0 0 0 0 1 1 1 0
```

Datarammen består således af 9 observationer og 4 variable. Hvis rækkerne har navne, kan disse findes ved hjælp af funktionen `rownames()`. Navnene på kolonnerne, altså de respektive variable i vores dataramme, kan findes ved hjælp af `colnames()`:

```
colnames(pol)
```

```
[1] "party" "vote"  "seat"  "rw"
```

Hvis vi gerne vil have antallet af kolonner og rækker i vores dataramme, kan de findes ved at bruge hhv. `ncol()` og `nrow()`:

```
ncol(pol)
```

```
[1] 4
```

```
nrow(pol)
```

```
[1] 9
```

Her kan vi ligeledes se, at der er fire kolonner og ni rækker. Hele datarammen kan vi få vist ved blot at skrive navnet på objektet, `pol`:

```
pol
```

	party	vote	seat	rw
1	Venstre	19.5	34	1
2	Socialdemokraterne	26.3	47	0
3	Enhedslisten	7.8	14	0
4	SF	4.2	7	0

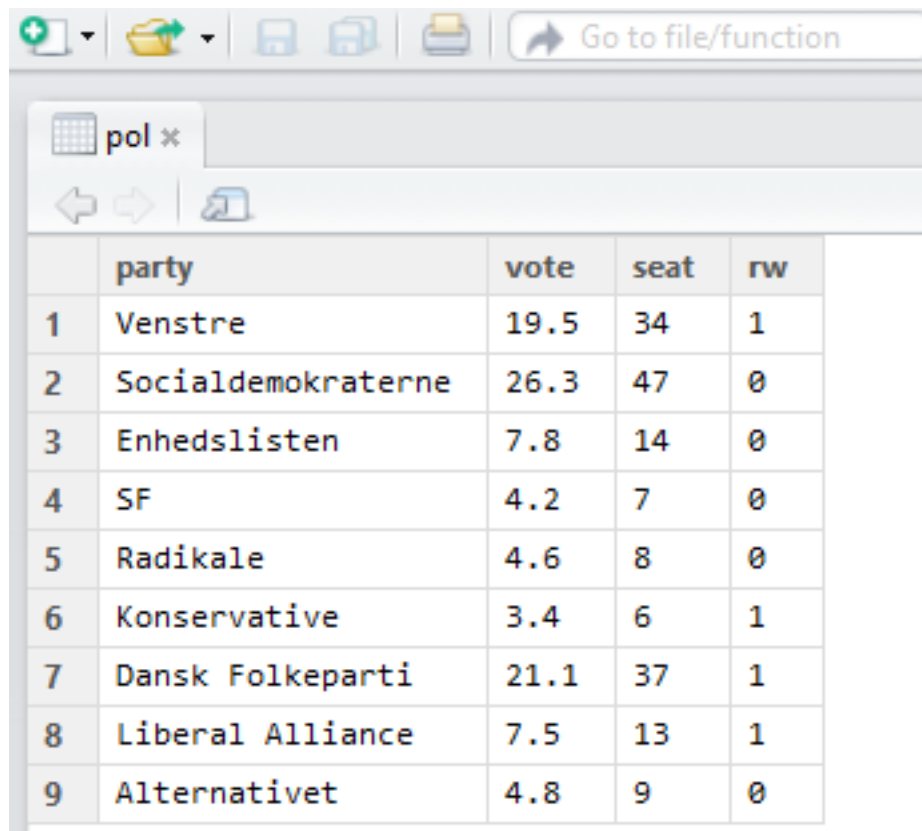
5	Radikale	4.6	8	0
6	Konservative	3.4	6	1
7	Dansk Folkeparti	21.1	37	1
8	Liberal Alliance	7.5	13	1
9	Alternativet	4.8	9	0

Dette er overkommeligt at vise i dette tilfælde, men når man arbejder med større datarammer, oftest med flere tusinde eller millioner af observationer, bliver det hurtigt uoverskueligt at vise hele datarammer. Heldigvis har R flere funktioner, der gør det let at få et overblik over, hvilke variable, vi har i vores dataramme. Med funktionen `head()` kan man få vist de seks første observationer i ens dataramme (altså de første seks rækker), og man kan tilføje et tal som argument efter ens objekt, hvis man gerne vil have vist et præcist antal observationer. Skulle man have lyst til at se de sidste observationer i ens dataramme, kan man bruge `tail()`.

```
head(pol)      # viser de første seks rækker
head(pol, 3)   # viser de første tre rækker
tail(pol)      # viser de sidste seks rækker
```

Det er ligeledes muligt at få vist ens dataramme i et nyt vindue, ligesom med browse i Stata, ved at bruge funktionen `View()` (bemærk det store V - ikke v).

```
View(pol)
```



	party	vote	seat	rw
1	Venstre	19.5	34	1
2	Socialdemokraterne	26.3	47	0
3	Enhedslisten	7.8	14	0
4	SF	4.2	7	0
5	Radikale	4.6	8	0
6	Konservative	3.4	6	1
7	Dansk Folkeparti	21.1	37	1
8	Liberal Alliance	7.5	13	1
9	Alternativet	4.8	9	0

Figur 2.1: Dataramme vist med View(), RStudio

Når man arbejder med datarammer vil man som regel arbejde med specifikke variable heri. Måden hvorpå man angiver bestemte variable i en dataramme er med `$` (altså et dollartegn, i dette tilfælde brugt som en *component selector*). Hvis vi eksempelvis gerne vil have alle stemmetallene ud fra `pol`, skriver vi:

```
pol$vote
```

```
[1] 19.5 26.3 7.8 4.2 4.6 3.4 21.1 7.5 4.8
```

Modsat ved en vektor, der er én dimension, har vi to dimensioner i en dataramme, altså rækker og kolonner (horisontalt og vertikalt). Her skal vi ligeledes bruge de firkantede parenteser, `[ ]`, som placeres i forlængelse af objektet, hvor vi blot skal tilføje to argumenter, mere specifikt i forhold til både hvilke rækker og kolonner, vi er interesseret i. Er vi eksempelvis interesseret i hele den første række, kan vi bruge `[1, ]` i forlængelse af objektet, hvor kommaet adskiller informationen, og den manglende information efter kommaet indikerer, at vi er interesseret i alle kolonner for den specifikke række.

```
pol[1,] # første række
```

```
      party vote seat rw
1 Venstre 19.5   34   1
```

Havde vi også tilføjet et tal ved kolonnen, ville vi få information ud for den pågældende række og kolonne. I nedenstående eksempel tilføjer vi 1 efter kommaet, for at fortælle, at vi ikke alene er interesseret i første række, men også i informationen i første kolonne (i dette eksempel under party).

```
pol[1,1] # første række, første kolonne
```

```
[1] Venstre
9 Levels: Alternativet Dansk Folkeparti Enhedslisten ... Venstre
```

Som det kan ses er værdien på første række i første kolonne Venstre. Hvis vi er interesseret i at få alle partierne, altså informationen gemt i første kolonne, kan vi fjerne argumentet om, at vi kun vil have første række.

```
pol[,1] # første kolonne
```

```
[1] Venstre          Socialdemokraterne Enhedslisten
[4] SF               Radikale           Konservative
[7] Dansk Folkeparti Liberal Alliance Alternativet
9 Levels: Alternativet Dansk Folkeparti Enhedslisten ... Venstre
```

Vi kan stadig bruge de funktioner, vi har gennemgået til nu, på vores datarammer. En nyttig funktion, der vil blive brugt til at få et overblik over informationen i et objekt, er `summary()`. For en dataramme giver `summary()` deskriptiv statistik for alle elementerne i vores dataramme (for de numeriske variable er dette minimum, første kvartil, medianen, gennemsnit, tredje kvartil og maksimum).

```
summary(pol)
```

```
      party      vote      seat      rw
Alternativet :1  Min.   : 3.40  Min.   : 6.00  Min.   :0.0000
```

Dansk Folkeparti:	1	1st Qu.:	4.60	1st Qu.:	8.00	1st Qu.:	0.0000
Enhedslisten	:1	Median	: 7.50	Median	:13.00	Median	:0.0000
Konservative	:1	Mean	:11.02	Mean	:19.44	Mean	:0.4444
Liberal Alliance:	1	3rd Qu.:	19.50	3rd Qu.:	34.00	3rd Qu.:	1.0000
Radikale	:1	Max.	:26.30	Max.	:47.00	Max.	:1.0000
(Other)	:3						

Hvis vi blot ønsker at få værdien ud på det maksimale antal stemmer givet til et parti, kan vi bruge `max()` funktionen.

```
max(pol$vote)
```

```
[1] 26.3
```

Hvis vi gerne vil have værdien på en bestemt række i en variabel i vores dataramme, kan vi bruge både `$` og `[ ]`. I nedenstående får vi værdien 2 i `party`.

```
pol$party[2]
```

```
[1] Socialdemokraterne
```

```
9 Levels: Alternativet Dansk Folkeparti Enhedslisten ... Venstre
```

Med det vi har lært til nu kan vi hente information ud om, hvilket parti der har fået flest stemmer (eller partier, hvis der er to partier, der har fået lige mange stemmer). Til dette specificerer vi, at vi gerne vil have information om variabelen `party`, for de partier for hvem tilfældet er, at deres stemmetal er lig (`==`) det maksimale antal stemmer (`max(pol$vote)`).

```
pol$party[pol$vote == max(pol$vote)]
```

```
[1] Socialdemokraterne
```

```
9 Levels: Alternativet Dansk Folkeparti Enhedslisten ... Venstre
```

Det var dermed Socialdemokraterne, der fik flest stemmer ved folketingsvalget i 2015. Samme procedure kan vi bruge med funktionen `min()` for at finde det parti, der fik færrest stemmer. Det var således, jævnfør nedenstående, de Konservative, der fik færrest stemmer ved folketingsvalget i 2015.

```
pol$party[pol$vote == min(pol$vote)]
```

```
[1] Konservative
```

```
9 Levels: Alternativet Dansk Folkeparti Enhedslisten ... Venstre
```

Der er ingen begrænsninger for, hvad vi kan lave med denne og lignende datarammer, herunder også statistiske analyser. For blot at give et eksempel, kan vi finde korrelationen mellem, hvor mange stemmer et parti har fået ved valget og antallet af mandater i Folketinget.

```
cor(pol$vote, pol$seat)
```

```
[1] 0.9997078
```

Nu har du styr på, hvad en dataramme er.

## 2.7 Import og eksport af datarammer

De fleste datarammer vil ikke blive lavet fra bunden i R, men tværtimod importeret fra andre filer. Det er heldigvis nemt at importere forskellige typer af data (også fra Stata og SPSS), men et af de mest anvendte og nyttige formater er kommaseparerede filer (.csv). Til at eksportere og importere datarammer fra og til R, bruger vi hhv. `write.csv()` og `read.csv()`.

Før vi arbejder med dette, er det vigtigt at have styr på, hvor man gemmer sine data til. I R arbejder man med et *working directory*, og ved at skrive `getwd()`, kan man se, hvor ens data vil blive gemt til.

```
getwd()
```

Hvis jeg gerne vil ændre dette, eksempelvis hvis jeg har en mappe på mit Skrivebord ved navn Rguide, jeg hellere vil have som mit *working directory*, kan jeg bruge funktionen `setwd()`.

```
setwd("C:/Users/Erik/Desktop/Rguide")
```

En nem måde at have styr på sit *working directory* er ved at åbne RStudio gennem sit R-script, hvorved ens *working directory* automatisk bliver det sted, hvor ens R-script ligger. Når vi har styr på, hvor vores data vil blive gemt, kan vi begynde at gemme dem. Her vil vi gemme vores dataramme `pol` til en fil (`ft2015.csv`). Til dette bruger vi `write.csv()`.

```
write.csv(pol, "ft2015.csv")
```

Bemærk at vi først angiver i funktionen, hvilken dataramme vi vil gemme, og derefter hvad filen skal hedde. Det er desuden vigtigt, at filen angives i citationstegn. Når vi har gemt filen kan vi først undersøge manuelt, om filen er gemt i vores *working directory*<sup>8</sup>. Hvis vi omvendt gerne vil importere et datasæt, kan vi bruge funktionen `read.csv()` og gemme datasættet i en dataramme.

```
pol <- read.csv("ft2015.csv")
```

Som med alt i R er der som regel flere pakker, der kan håndtere ting, herunder også især import og eksport af filer. Blandt de nævneværdige er pakkerne `foreign` (R Core Team, 2015), `rio` (C. Chan, Chan, & Leeper, 2016) og `readr` (H. Wickham & Francois, 2015) (installation af pakker gennemgås i næste sektion).

## 2.8 Installation af pakker

Pakker er noget af det, der gør R fantastisk. Der er ingen grænser for, hvad man kan bruge R til, og dette skyldes især de talrige pakker, der er lavet til R. Der er to funktioner, der skal bruges i denne sammenhæng. For det første en funktion til at installere pakker, `install.packages()`, og en funktion til at åbne en pakke, `library()` (alternativt kan man også bruge `require()`).

En pakke skal kun installeres én gang. Det vil sige, at når du har brugt `install.packages()` til at installere en pakke, er du fri for at gøre det igen. I dette eksempel vil vi bruge funktionen til at installere pakken `ggplot2`, som vi vil bruge til at lave figurer senere. Bemærk desuden citationstegnene, der også er nødvendige i denne sammenhæng.

```
install.packages("ggplot2")
```

Når en pakke er installeret skal den hentes ind i R. Du kan have utallige pakker installeret på din computer, men der er ingen grund til at R skal bruge alle pakker, hver gang du åbner R. Derfor skal du hver gang, du bruger en bestemt pakke, bruge `library()` til at hente pakken. Det er med andre ord en god idé at begynde dine scripts med, at åbne alle de pakker, du får brug for i analyserne.

---

<sup>8</sup>I R kan man evt. bruge funktionen `file.exists()` til at se, om filen eksisterer.

```
library("ggplot2")
```

Når du installerer `ggplot2` vil du desuden opdage, at R også installerer en række andre pakker. Dette fordi `ggplot2` anvender andre pakker, der også skal installeres, for at pakken fungerer hensigtsmæssigt. Disse pakker åbnes også automatisk, hver gang du bruger `library()` til at åbne `ggplot2`.

Der er et hav af forskellige pakker til R, og hvilke der er relevante at bruge afhænger af, hvad man ønsker at bruge R til. Ikke desto mindre er der en lille oversigt i Bilag C, hvor en række anbefalelsesværdige pakker nævnes.

## 2.9 Objekter i hukommelsen

Vi har nu arbejdet med en lang række af objekter. For at se hvilke objekter, vi har gang i - og evt. for at fjerne nogle af dem - er der et par enkelte nyttige funktioner, du bør kende til. Den første er `ls()`, der viser, hvilke objekter vi har i hukommelsen (*ls* står for *list objects*). Som det kan ses har vi otte objekter i hukommelsen.

```
ls()
```

```
[1] "party" "pol"   "rw"    "seat"  "vote"  "x"     "y"     "z"
```

Hvis vi gerne vil fjerne et objekt fra hukommelsen, kan vi bruge funktionen `rm()` (*rm* er en forkortelse af *remove*). I nedenstående eksempel bruger vi først `rm()` til at fjerne objektet `x` og dernæst `ls()` til at se, om `x` er fjernet.

```
rm(x)
```

```
ls()
```

```
[1] "party" "pol"   "rw"    "seat"  "vote"  "y"     "z"
```

Hvis vi gerne vil fjerne *alt* i hukommelsen, kan vi bruge `ls()` i kombination med `rm()`.

```
rm(list = ls())
```

```
ls()
```



`character()`

Ligeledes kan man i RStudio se hvilke objekter, funktioner m.v. man har åbent under 'Environment' (i boksen hvor der også er 'History', hvor man kan se en historik over de kommandoer, man har kørt i ens nuværende session).

# Kapitel 3

## Databehandling

Der er flere måder hvorpå man i R kan behandle datarammer på og herunder få lavet nye variable i datarammen. I dette kapitel vil der blive vist forskellige måder til at behandle datarammer, og der vil især blive fokuseret på behandlingen af variable i en dataramme. Her vil der også blive givet et par eksempler på, hvordan vi kan lave en ny variabel baseret på værdierne på en anden variabel.

Lad os tage udgangspunkt i, at vi gerne vil have lavet en binær variabel, der antager værdien 1, hvis et parti i Folketinget har fået mere end 20 procent af stemmerne, og værdien 0 hvis ikke. Vi vil gerne kalde denne variabel for `big`. For at gøre dette henter vi først de data, vi lavede i forrige kapitel.

```
pol <- read.csv("data/ft2015.csv")
```

I nedenstående laver vi først et nyt element (en ny variabel) i vores dataramme med navnet `big`. Denne får værdierne `NA`. Dernæst angiver vi, at `pol$big` skal have værdien 1, men kun for de observationer, hvor `pol$vote` er større end eller lig med 20. Til sidst siger vi, at de observationer der fik mindre end 20 procent af stemmerne, skal værdien være 0. I dette eksempel kunne vi have undladt det sidste step og blot brugt værdien 0 i stedet for `NA`. Til sidst bruger vi funktionen `table()` til at få vist, hvilke partier der har værdien 1 på `pol$big`. Som det kan ses, er det hhv. Dansk Folkeparti og Socialdemokraterne.

```
pol$big <- NA
pol$big[pol$vote >= 20] <- 1
pol$big[pol$vote < 20] <- 0

table(pol$party, pol$big)
```

	0	1
Alternativet	1	0
Dansk Folkeparti	0	1
Enhedslisten	1	0
Konservative	1	0
Liberal Alliance	1	0
Radikale	1	0
SF	1	0
Socialdemokraterne	0	1
Venstre	1	0

Der er mange måder hvorpå man kan rekode variable. En anden mulighed er at bruge funktionen `ifelse()`, hvor vi fortæller R, at hvis hvis noget er sandt, skal den give en bestemt værdi, og en anden værdi til de resterende observationer. I nedenstående kode undersøger vi igen, om `vote` er større eller lig med 20, og hvis dette er tilfældet, gives værdien 1. Hvis ikke, gives værdien 0.

```
pol$big <- ifelse(pol$vote >= 20, 1, 0)

table(pol$party, pol$big)
```

	0	1
Alternativet	1	0
Dansk Folkeparti	0	1
Enhedslisten	1	0
Konservative	1	0
Liberal Alliance	1	0
Radikale	1	0
SF	1	0
Socialdemokraterne	0	1
Venstre	1	0

Hvis vi eksempelvis manuelt ville give Socialdemokraterne og Dansk Folkeparti værdien 1, uden at bruge information omkring deres stemmetal, kunne dette også gøres. Nedenstående

kode giver nogle eksempler herpå. Den første linje giver big værdien 1 hvis party er lig Socialdemokraterne *eller* Dansk Folkeparti. Den anden linje giver big værdien 1 hvis værdien i party er i (angivet ved %in%) vektoren med Socialdemokraterne og Dansk Folkeparti. De resterende linjer er eksempler dernæst på, hvor værdien 0 gives. Bemærk især den sidste linje, hvor funktionen `is.na()` bruges til at undersøge, om der er nogle manglende værdier, og hvis der er, gives værdien 0.

```
pol$big[pol$party == "Socialdemokraterne" |
        pol$party == "Dansk Folkeparti"] <- 1
pol$big[pol$party %in% c("Socialdemokraterne",
                        "Dansk Folkeparti")] <- 1

pol$big[pol$party != "Socialdemokraterne" &
        pol$party != "Dansk Folkeparti"] <- 0
pol$big[is.na(pol$big)] <- 0
```

## 3.1 Behandling af datarammer med dplyr

Der er forskellige pakker, der kan bruges til at bearbejde datarammer, men den bedste er uden sammenligning dplyr (Hadley Wickham & Francois, 2016). Et andet eksempel på en pakke, der kan bruges til at rekode variable i datarammer er pakken car (Fox & Weisberg, 2011), der har funktionen `recode()`, men vi vil fokusere på dplyr i resten af kapitlet. Det første vi gør - og forudsætningen for at resterende funktioner i kapitlet virker - er, at åbne dplyr<sup>1</sup>.

```
library("dplyr")
```

Pakken giver nogle helt basale funktioner, der gør det nemt at bearbejde datarammer. Funktionerne heri inkluderer `select()`, `filter()`, `arrange()`, `rename()`, `mutate()` og `summarize()`<sup>2</sup>. `select()` og `filter()` bruges til henholdsvis at udvælge kolonner (*variable*) og rækker (*observationer*). `mutate()` bruges til at tilføje eller ændre eksisterende kolonner. `arrange()` bruges til at sortere rækker. `rename()` bruges til at ændre navnet på en kolonne. `summarize()` bruges til at aggregere rækker.

Fælles for alle disse funktioner er, at de tager udgangspunkt i en dataramme. De lader sig dermed ikke applicere på andre typer af klasser end datarammer. De returnerer ligeledes alle

<sup>1</sup>Installation af pakker gennemgås i sektion 2.8.

<sup>2</sup>For en anden god introduktion til dplyr, se: [Managing Data Frames with the dplyr package](#).

en ny datarammer. Ved at kunne bruge disse funktioner og kombinere dem i ens arbejde, har man styr på hvad det kræver at bearbejde datarammer.

## 3.2 Vælg bestemte variable med `select()`

Hvis vi kun skal bruge bestemte variable i vores dataramme, eksempelvis partinavn (`party`) og om det er et højreorienteret parti (`rw`), kan vi anvende funktionen `select()`:

```
select(pol, party, rw)
```

	party	rw
1	Venstre	1
2	Socialdemokraterne	0
3	Enhedslisten	0
4	SF	0
5	Radikale	0
6	Konservative	1
7	Dansk Folkeparti	1
8	Liberal Alliance	1
9	Alternativet	0

Der er forskellige funktioner, der kan bruges til at finde de variable, vi gerne vil vælge. Hvis vi gerne vil vælge en eller flere variable, der indeholder bestemt tekst i variabelnavnet, kan vi bruge `contains()`:

```
select(pol, contains("par"))
```

	party
1	Venstre
2	Socialdemokraterne
3	Enhedslisten
4	SF
5	Radikale
6	Konservative
7	Dansk Folkeparti
8	Liberal Alliance
9	Alternativet

Foruden `contains()` er der mulighed for at bruge blandt andet `starts_with()`, `ends_with()`, `matches()`, `num_range()`, `one_of()` og `everything()`.

Hvis vi gerne vil fjerne en variabel fra en dataramme, kan vi bruge minustegnet. I dette eksempel fjerner vi `rw` fra datarammen:

```
select(pol, -rw)
```

	X	party	vote	seat	big
1	1	Venstre	19.5	34	0
2	2	Socialdemokraterne	26.3	47	1
3	3	Enhedslisten	7.8	14	0
4	4	SF	4.2	7	0
5	5	Radikale	4.6	8	0
6	6	Konservative	3.4	6	0
7	7	Dansk Folkeparti	21.1	37	1
8	8	Liberal Alliance	7.5	13	0
9	9	Alternativet	4.8	9	0

### 3.3 Vælg bestemte observationer med `filter()`

Hvis vi gerne vil have hele datarammen, altså alle kolonner, men blot for udvalgte rækker (observationer), kan vi bruge funktionen `filter()`. I nedenstående eksempel tager vi datarammen `pol` men viser kun rækkerne for de højreorienterede partier, altså der hvor `rw` er lig med 1.

```
filter(pol, rw == 1)
```

Warning: package 'bindrcpp' was built under R version 3.3.2

	X	party	vote	seat	rw	big
1	1	Venstre	19.5	34	1	0
2	6	Konservative	3.4	6	1	0
3	7	Dansk Folkeparti	21.1	37	1	1
4	8	Liberal Alliance	7.5	13	1	0

Det er vigtigt at nævne igen, at det at køre ovenstående kommando ikke ændrer noget i datarammen `pol`. Dette vil kun ske, hvis vi overskriver den eksisterende dataramme. Ligeledes

ville ovenstående nemt kunne gemmes i sin egen dataramme, evt. ved at lave en ny dataramme med navnet `pol.rw`.

```
pol.rw <- filter(pol, rw == 1)
pol.rw
```

	X	party	vote	seat	rw	big
1	1	Venstre	19.5	34	1	0
2	6	Konservative	3.4	6	1	0
3	7	Dansk Folkeparti	21.1	37	1	1
4	8	Liberal Alliance	7.5	13	1	0

### 3.4 Vælg rækkefølgen på observationer med `arrange()`

Hvis vi gerne vil have ændret rækkefølgen på vores rækker, kan vi bruge `arrange()`. I nedenstående eksempel sorterer vi observationerne i vores dataramme efter hvor mange stemmer de respektive partier har fået.

```
arrange(pol, vote)
```

	X	party	vote	seat	rw	big
1	6	Konservative	3.4	6	1	0
2	4	SF	4.2	7	0	0
3	5	Radikale	4.6	8	0	0
4	9	Alternativet	4.8	9	0	0
5	8	Liberal Alliance	7.5	13	1	0
6	3	Enhedslisten	7.8	14	0	0
7	1	Venstre	19.5	34	1	0
8	7	Dansk Folkeparti	21.1	37	1	1
9	2	Socialdemokraterne	26.3	47	0	1

Som det kan ses i ovenstående er det parti, der har fået færrest stemmer, placeret øverst. Hvis vi gerne vil have det således, at de partier, der har fået færrest stemmer, er nederst, kan vi angive dette med et minus (-) før variablen:

```
arrange(pol, -vote)
```

	X	party	vote	seat	rw	big
1 2	Socialdemokraterne	26.3	47	0	1	
2 7	Dansk Folkeparti	21.1	37	1	1	
3 1	Venstre	19.5	34	1	0	
4 3	Enhedslisten	7.8	14	0	0	
5 8	Liberal Alliance	7.5	13	1	0	
6 9	Alternativet	4.8	9	0	0	
7 5	Radikale	4.6	8	0	0	
8 4	SF	4.2	7	0	0	
9 6	Konservative	3.4	6	1	0	

### 3.5 Skift navnet på en variabel med `rename()`

Hvis man har en variabel, man gerne vil ændre navnet på, kan man anvende funktionen `rename()`. I nedenstående eksempel ændrer vi navnet på `party` til `partinavn`:

```
rename(pol, partinavn = party)
```

	X	partinavn	vote	seat	rw	big
1 1	Venstre	19.5	34	1	0	
2 2	Socialdemokraterne	26.3	47	0	1	
3 3	Enhedslisten	7.8	14	0	0	
4 4	SF	4.2	7	0	0	
5 5	Radikale	4.6	8	0	0	
6 6	Konservative	3.4	6	1	0	
7 7	Dansk Folkeparti	21.1	37	1	1	
8 8	Liberal Alliance	7.5	13	1	0	
9 9	Alternativet	4.8	9	0	0	

### 3.6 Tilføj variabel med `mutate()`

Hvis vi gerne vil tilføje en variabel til vores dataramme, kan vi bruge funktionen `mutate()`. I nedenstående eksempel tilføjer vi en variabel ved navn `vote.m`, der angiver hvor mange



procentpoint stemmer et parti ligger fra det gennemsnitlige antal stemmer, et parti fik (alså 11.02):

```
mutate(pol, vote.m = vote - mean(vote))
```

	X		party	vote	seat	rw	big	vote.m
1	1		Venstre	19.5	34	1	0	8.477778
2	2		Socialdemokraterne	26.3	47	0	1	15.277778
3	3		Enhedslisten	7.8	14	0	0	-3.222222
4	4		SF	4.2	7	0	0	-6.822222
5	5		Radikale	4.6	8	0	0	-6.422222
6	6		Konservative	3.4	6	1	0	-7.622222
7	7		Dansk Folkeparti	21.1	37	1	1	10.077778
8	8		Liberal Alliance	7.5	13	1	0	-3.522222
9	9		Alternativet	4.8	9	0	0	-6.222222

### 3.7 Kombination af funktioner med pipe operatoren

I mange sammenhænge er det ikke tilstrækkeligt blot at bruge én funktion. Det være sig eksempelvis hvis man både skal bruge bestemte kolonner og rækker. Der er heldigvis intet i vejen med at pakke en funktion ind i en anden funktion, som i nedenstående eksempel, hvor vi har `select()` pakket ind i `filter()`.

```
filter(select(pol, party, vote, rw), rw == 1)
```

		party	vote	rw
1		Venstre	19.5	1
2		Konservative	3.4	1
3		Dansk Folkeparti	21.1	1
4		Liberal Alliance	7.5	1

Problemet med dette er, at det kan blive meget kompliceret at læse, især hvis man tager flere funktioner i brug. Når kompleksiteten stiger, stiger sandsynligheden også for, at man laver en dum fejl, evt. med en ekstra parentes et forkert sted. I pakken `dplyr` er der en operator, der skal hjælpe med at gøre det nemmere at bearbejde datarammer, nemlig pipe operatoren `%>%`.

Denne operator anvender en trinvis logik i vores bearbejdning af datarammer. Logikken er, at vi først angiver hvilken dataramme, vi arbejder med, og så dernæst gør én ting ad gangen. I nedenstående eksempel bruger vi pipe operatoren til at gøre det samme som i ovenstående eksempel - bare over flere linjer, der er nemmere at læse og forstå.

```
pol %>%  
  select(party, vote, rw) %>%  
  filter(rw == 1)
```

	party	vote	rw
1	Venstre	19.5	1
2	Konservative	3.4	1
3	Dansk Folkeparti	21.1	1
4	Liberal Alliance	7.5	1

På den første linje viser vi, at vi skal bruge datarammen `pol`. På samme linje slutter vi med `%>%`, der fortæller, at vi skal bruge denne linje input i næste linje. På anden linje bruger vi `select()` og vælger variablene `party`, `vote` og `rw`. Her slutter vi også med `%>%`, så vi fortæller R, at dette skal bruges som input i næste linje. Tredje linje slutter af med at vælge de observationer, der har `rw` lig med 1 (altså er højreorienterede partier). Bemærk at denne linje *ikke* slutter med `%>%`, da vi ikke ønsker at gøre mere ved vores dataramme.

### 3.8 Kør funktioner på variable med `apply()`

Hvis vi gerne vil køre en funktion på nogle rækker eller en nogle kolonner i vores dataramme, kan vi bruge funktionen `apply()`. Hvis vi eksempelvis gerne vil have gennemsnittet af `vote` og `seat` for de højreorienterede partier, kan vi først angive, at vi vil arbejde med højreorienterede partier (vha. `filter()`) og så vælge de variable, vi ønsker information om. Vi bruger så `apply()` på denne dataramme, hvor vi først angiver, at vi er interesseret i at køre funktionen over kolonnerne (2 indikerer kolonner, 1 indikerer rækker), dernæst at vi er interesseret i at køre funktionen `mean`, og til sidst at eventuelle manglende værdier skal fjernes.

```
pol %>%  
  filter(rw == 1) %>%  
  select(vote, seat) %>%  
  apply(MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```

vote    seat
12.875  22.500

```

Ovenstående viser således, at vi kombinere de forskellige funktioner med pipe operatoren og nemt få informationer ud af en dataramme. Bemærk at vi ved `apply()` først angiver `MARGIN`, der som beskrevet skal være 1, hvis vi ønsker at applicere funktionen på nogle rækker, og 2 hvis vi ønsker at applicere funktionen på kolonner. Hvis man ønsker at gøre det på begge, skal man angive det med `c(1, 2)`. Med `FUN` angiver vi, hvilken funktion vi ønsker at applicere på vores dataramme. Til sidst kan man angive tilføjelser relateret til den funktion, man ønsker at applicere. Det er dog ikke nødvendigt at angive `MARGIN` eller `FUN`, hvis man blot har styr på rækkefølgen. I vores eksempel er der heller ingen manglende værdier, hvorfor en tilføjelse relateret til disse ej heller er nødvendig. Derfor kan vi simplificere vores kode til følgende:

```

pol %>%
  filter(rw == 1) %>%
  select(vote, seat) %>%
  apply(2, mean)

```

```

vote    seat
12.875  22.500

```

### 3.9 Aggreger variable med `summarize()`

Hvis vi gerne vil lave nye variable med aggregerede informationer, kan vi bruge funktionen `summarize()`. I nedenstående eksempel får vi informationer om henholdsvis antallet af partier i datarammen, det mindste antal stemmer et parti har fået, det maksimale antal stemmer et parti har fået samt antallet af højreorienterede partier.

```

pol %>%
  summarize(partier = n(), mindst = min(vote),
            stoerst = max(vote), hoejreorienteret = sum(rw))

```

```

partier mindst stoerst hoejreorienteret
1          9    3.4    26.3                4

```

Ved `partier` bruger vi funktionen `n()`. Denne er god at anvende, om ikke andet også bare for at sikre, at man har det korrekte antal af rækker i sine data. De andre funktioner, der anvendes,

blev introduceret i forrige kapitel. Vi kan ligeledes bruge funktionen `group_by()`, hvis vi gerne vil have informationer på baggrund af en anden variabel. I nedenstående eksempel får vi de samme typer af informationer, blot for hhv. højreorienterede og venstreorienterede partier.

```
pol %>%  
  group_by(rw) %>%  
  summarize(partier = n(), mindst = min(vote), stoerst = max(vote))
```

```
# A tibble: 2 x 4  
  rw partier mindst stoerst  
  <int>   <int>   <dbl>   <dbl>  
1     0     5     4.2    26.3  
2     1     4     3.4    21.1
```

# Kapitel 4

## Visualisering

Der er mange måder at præsentere data på. Dette kapitel vil give en introduktion i, hvordan man kan visualisere sine data med R med fokus på pakken ggplot2. Der er tungtvejende grunde til at fokusere på at visualisere sine data, og i samfundsvidenskaberne er der kommet fokus på vigtigheden af at præsentere sine resultater i figurer i stedet for tabeller (Healy & Moody, 2014, Kastelec & Leoni (2007), Schwabish (2014)).

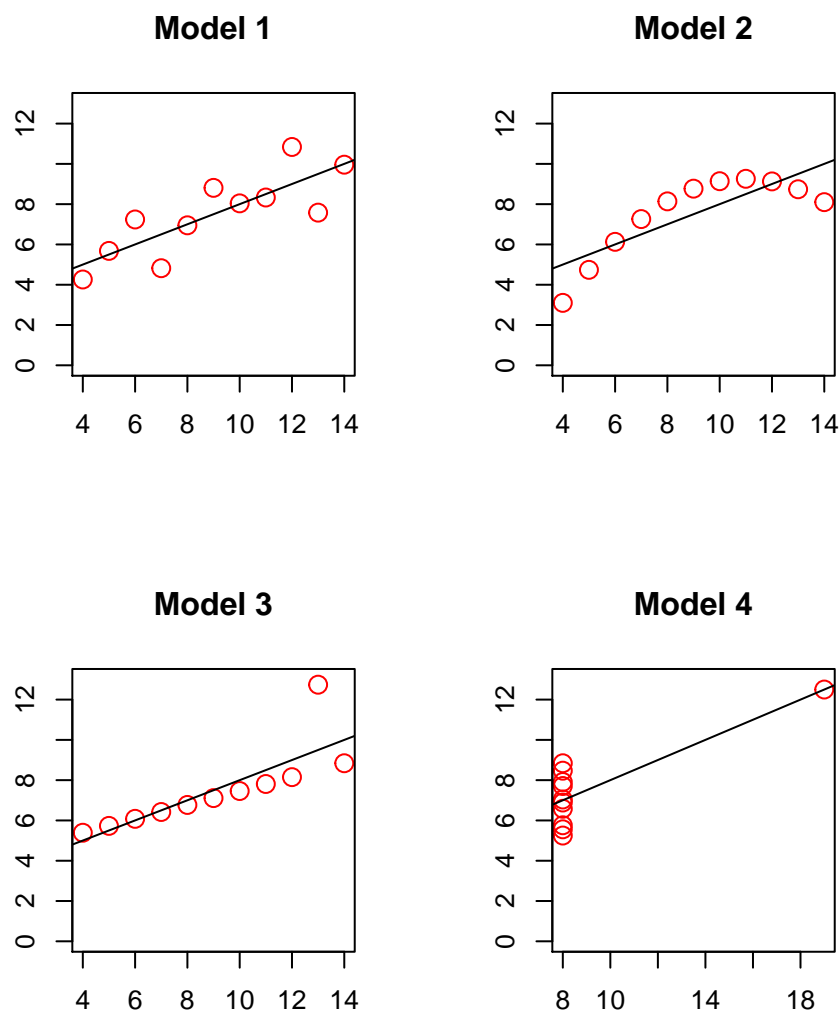
A. Field, Miles, & Field (2012) har beskrevet det som følger: “Data analysis is a bit like Internet dating (actually it’s not, but bear with me): you can scan through the vital statistics and find a perfect match (good IQ, tall, physically fit, likes arty French films, etc.) and you’ll think you have found the perfect answer to your question. However, if you haven’t looked at a picture, then you don’t really know how to interpret this information [...] Data analysis is much the same: inspect your data with a picture, see how it looks and only then think about interpreting the more vital statistics.” (side 117)

Et kendt eksempel på det nyttige i at visualisere sine data gives af Anscombe (1973). Tabel 4.1 viser udvalgte resultater fra fire forskellige bivariate regressionsmodeller. Som det kan ses, er resultaterne identiske for de fire modeller.

Tabel 4.1: Anscombe’s kvadrant, resultater

	Model 1	Model 2	Model 3	Model 4
$\beta_0$ (se)	3,00 (1,12)	3,00 (1,13)	3,00 (1,13)	3,00 (1,12)
$\beta_1$ (se)	0,50 (0,12)	0,50 (0,12)	0,50 (0,12)	0,50 (0,12)
$R^2$	0,63	0,63	0,63	0,63
$N$	11	11	11	11

Kigger vi omvendt på, hvordan forholdet er mellem den uafhængige og afhængige variabel i de fire modeller, som illustreret i Figur 4.1, ser vi, at der er nævneværdige forskelle på de forskellige modeller, som ikke kommer til syne ved at kigge på nogle af de estimer, vi som regel er mest interesserede i, når vi arbejder med regressionsanalyser<sup>1</sup>.



Figur 4.1: Anscombes kvadrant

Ovenstående figur er lavet med R uden brug af nogle pakker, og dette fungerer fint og kan fungere ganske godt i de fleste tilfælde. Det er dog ofte yderst besværligt (altså tidskrævende) at lave pæne figurer på denne måde, og det script man ligeledes bygger op kan være svært for andre at læse, hvis man laver mere komplicerede figurer. Heldigvis er der en fantastisk pakke til R kaldet `ggplot2` (H. Wickham, 2009), der gør det nemt at lave pænere figurer i R. Det er dog vigtigt at nævne, at `ggplot2` ikke nødvendigvis er bedre end så mange andre pakker til at lave figurer, hvorfor det delvist handler om ens personlige præference. Ligeledes er der

<sup>1</sup>For et andet godt eksempel, se indlægget [What data patterns can lie behind a correlation coefficient?](#).

også funktioner i ggplot2, som det vil blive anbefalet *ikke* at bruge, herunder især funktionen `qplot()`, der står for *quick plot*, men som er ganske overflødig. ggplot2 bruges dog af flere og flere, hvorfor der også er mange gode ressourcer, der gør det nemt at lave pæne figurer.

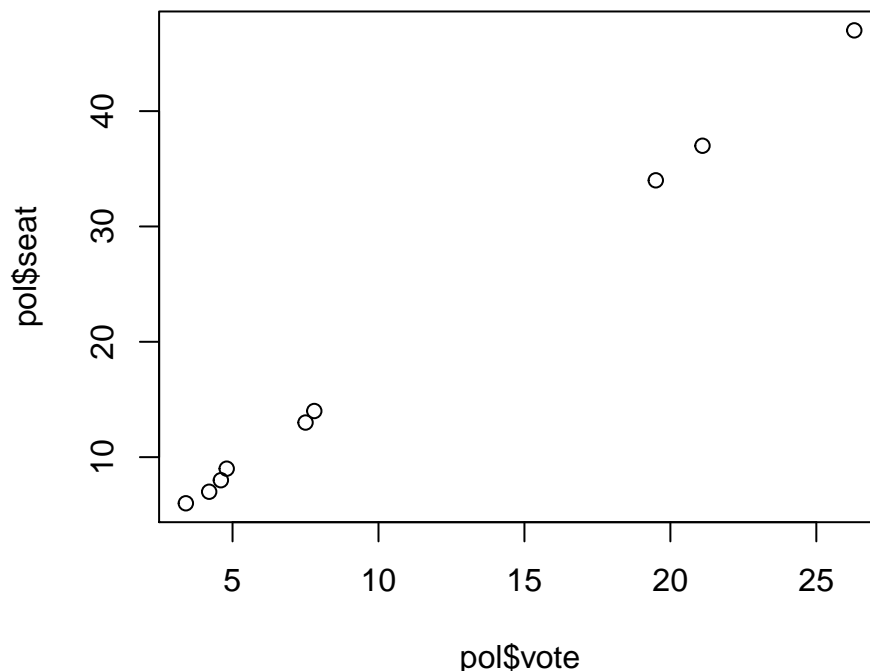
## 4.1 Introducerende eksempel med ggplot2

I nærværende kapitel vil vi først kigge på et simpelt eksempel, der viser, hvordan man kan lave figurer med ggplot2 i R<sup>2</sup>. Til dette formål vil vi bruge datasættet `ft2015.csv`, som vi genererede i forrige kapitel<sup>3</sup>. Vi indlæser datasættet med funktionen `read.csv()`.

```
pol <- read.csv("data/ft2015.csv")
```

Hvis vi gerne vil vise relationen mellem hvor mange stemmer et parti har fået ved folketingsvalget og dets mandater i Folketinget, kan vi bruge funktionen `plot()` i R, der er en del af basispakken og dermed ikke kræver at en pakke indlæses.

```
plot(pol$vote, pol$seat)
```



Et lignende plot kan vi lave med ggplot2, hvorfor vi først åbner pakken i R med `library()`.

<sup>2</sup>For installation af pakken, se sektion 2.8.

<sup>3</sup>De datasæt vi arbejder med i løbet af bogen kan ligeledes findes på GitHub: [github.com/erikgahner/Rguide/tree/master/data](https://github.com/erikgahner/Rguide/tree/master/data)

```
library("ggplot2")
```

Til at lave et plot ala ovenstående med ggplot2 bruger vi funktionen `ggplot()`. Med `ggplot()` specificerer vi først hvilken dataramme, vi ønsker at anvende, samt hvilke variable i datarammen, der skal visualiseres (bemærk at `ggplot()` *altid* tager udgangspunkt i en dataramme). Det er med andre ord her, at vi angiver, hvilke variable der skal visualiseres samt deres rolle (evt. om variabelen skal være på x-aksen eller y-aksen). Hvis vi eksempelvis har en dataramme med en `variabel1` og `variabel2`, kan specifikationen se ud som følger:

```
ggplot(dataramme, aes(x=variabel1, y=variabel2))
```

Her har vi således angivet, at vi er interesseret i `variabel1` og `variabel2`, samt at førstnævnte er en x-akse variabel og sidstnævnte er en y-akse variabel. Vi har endnu ikke specificeret, hvilken type figur, vi ønsker at lave. Dette kan vi tilføje ved først at lave et `+` tegn, og så angive, hvilken figur vi ønsker at lave (i dette tilfælde et punktdiagram):

```
ggplot(dataramme, aes(x=variabel1, y=variabel2)) +  
  geom_point()
```

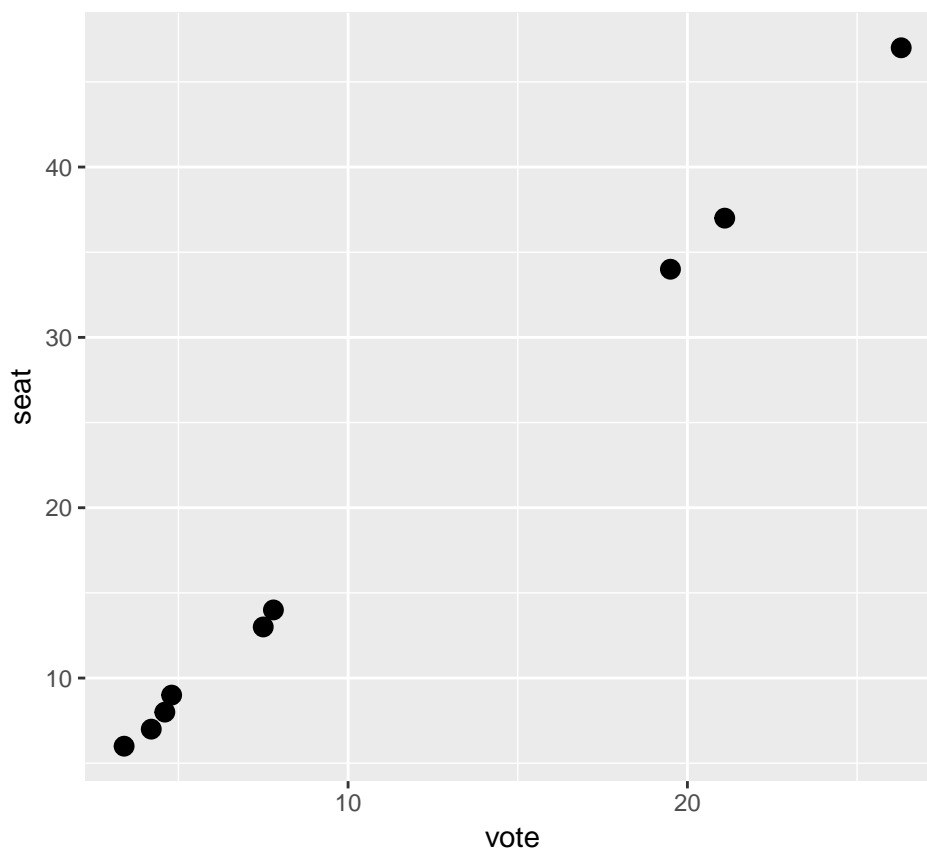
Dette er grundideen med ggplot2. Først angiver vi, hvilke data vi er interesseret i at arbejde med, og så tilføjer vi ekstra specifikationer efter forgodtbefindende. Vi kan således tilføje information om, hvad der skal stå på akserne, hvor store punkterne skal være og så videre. En af fordelene ved denne tilgang er, at du først kan lave noget meget simpelt, og så derefter redigere i dette, til du til sidst opnår det ønskede resultat. Dette er uanset om du vælger at lave et søjlediagram, en punktdiagram eller noget helt tredje.

Dette kan virke uoverskueligt, men der er en mening med galskaben (en mening der kun bliver mere klar, jo mere man arbejder med det). De to g'er (altså gg) i ggplot2 står for *grammar of graphics*, og det er hele filosofien bag, altså at der skal være en sætningsstruktur til de figurer, man laver. Med andre ord består vores figur af forskellige komponenter. Har vi først lavet én figur, kan vi bygge videre på denne, eksempelvis ved at tilføje linjer og ændre farvetema, tekst med videre.

Vi kan således bruge ggplot2 til at lave en figur som ovenfor. Først specificerer vi, at vi er interesseret i at bruge datarammen `pol`, hvor x-aksen skal være `vote` og y-aksen `seat`. Dernæst siger vi at det skal være punkter (`geom_point`), vi gerne vil have visualiseret.

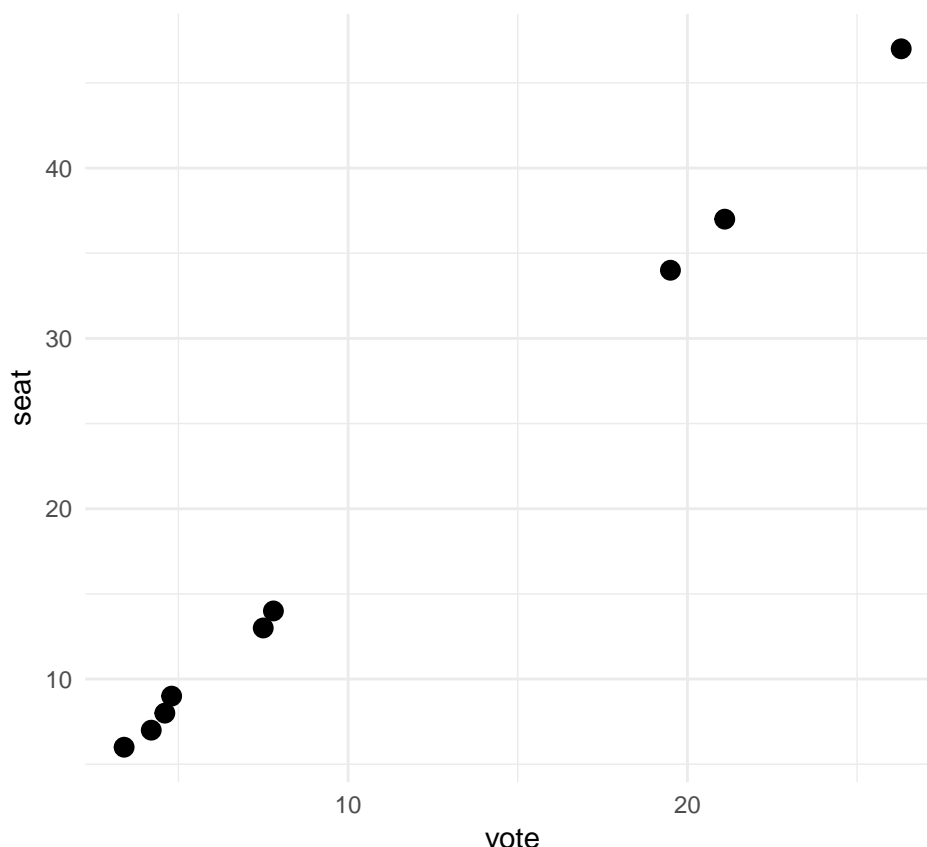


```
ggplot(pol, aes(x=vote, y=seat)) +  
  geom_point(size=3)
```



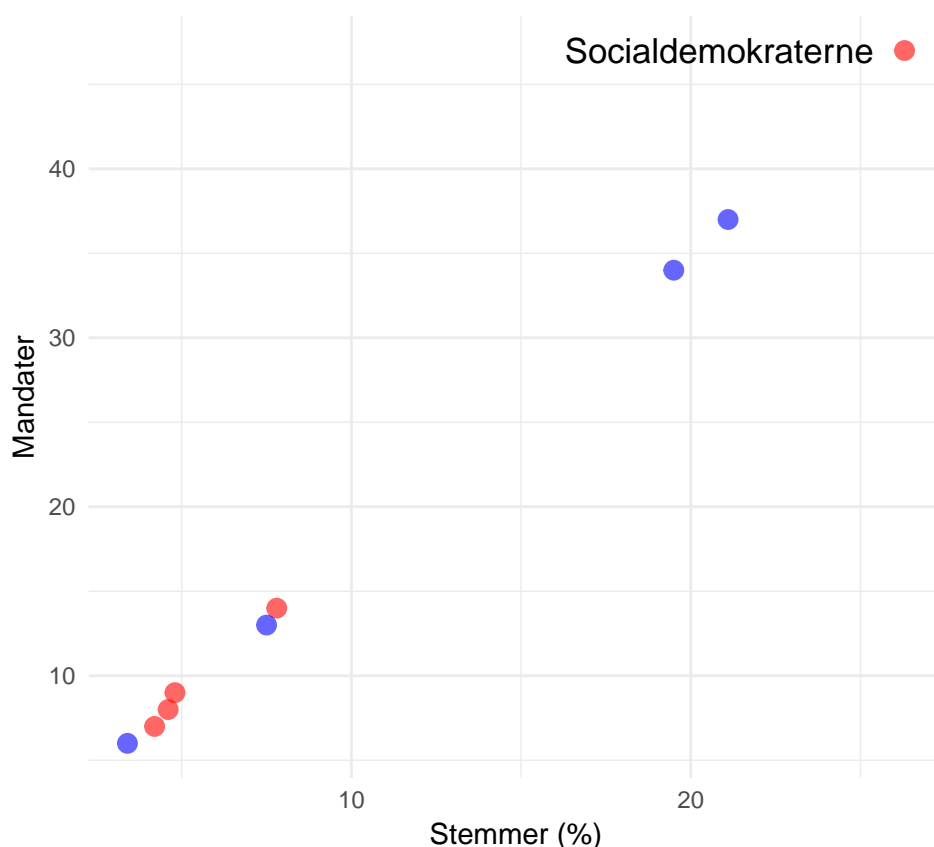
Dette er dog ikke nødvendigvis vores færdige figur. Tværtimod kan vi som nævnt tilføje flere komponenter. Lad os begynde med blot at ændre temaet, så vi får et mere minimalistisk *look*. Dette gøres med tilføjelsen `theme_minimal()`. Bemærk desuden at de forskellige komponenter adskilles med `+`, men at det ikke er et krav, at de er på en ny linje.

```
ggplot(pol, aes(x=vote, y=seat)) +  
  geom_point(size=3) +  
  theme_minimal()
```



Dette er blot for at vise logikken bag ggplot2. Det kan virke kompliceret, men undlad at give op. Jo flere figurer du laver, gerne helt fra bunden, desto nemmere bliver det med tiden. Husk desuden også, at Google er en fantastisk ressource i denne sammenhæng, og hvis du søger efter ggplot2 og det du gerne vil lave, vil der i de fleste tilfælde komme resultater op, der vil hjælpe dig. Der er som sagt ingen grænse for, hvad man kan bygge på og ændre i, og i nedenstående kan det ses, at der er tilføjet en række ekstra linjer, der hver især tjener et formål i forhold til figuren.

```
ggplot(pol, aes(x=vote, y=seat)) +
  scale_colour_manual(values=c("red", "blue")) +
  geom_point(aes(col=as.factor(rw)), alpha=0.6, size=3) +
  theme_minimal() + theme(legend.position="none") +
  ylab("Mandater") +
  xlab("Stemmer (%)") +
  geom_text(aes(label=ifelse(party == "Socialdemokraterne",
                             "Socialdemokraterne", "")),
            hjust=1.1, vjust=0.5), size=4.5)
```



Ovenstående plot illustrerer blot den måde hvorpå ggplot2 fungerer. I nærværende tilfælde ville det ikke være nødvendigt at visualisere forholdet mellem stemmer og mandater, da det er begrænset hvor meget anden information der kommer ud af figuren, end ved blot at formidle korrelationen.

Det bedste råd der kan gives i forhold til brugen af ggplot2 pakken er, at lære det grundlæggende (altså hvordan man laver et histogram og andre figurer) og derfra så gradvist lære at bygge ovenpå (eksempelvis gennem Google), når der er problemer eller bestemte ønsker.

## 4.2 Specifikationer med ggplot2

I dette afsnit introduceres vigtig terminologi, og mere specifikt hvordan ggplot2 fungerer med geometriske objekter og æstetiske tilføjelser. Alle plots lavet med ggplot2 laves med funktionen `ggplot()`. I denne funktion angives det først, som beskrevet i forrige afsnit, hvilken dataramme, man anvender, hvorefter det angives hvilke variable, man er interesseret i. Ønsker man blot at vise en distribution af én variabel, er det selvsagt tilstrækkeligt blot at angive én variabel. De variable der skal visualiseres *skal* være en del af en dataramme.

Der er et utal af muligheder med ggplot2, og således også flere ting, der kan specificeres.

Alt efter hvilken type figur, vi ønsker at lave, skal vi vælge en bestemt geom, der er et geometrisk objekt. Det er denne specifikation, der fortæller, om vi er interesseret i et histogram, et punktdiagram eller noget helt tredje.

Tabel 4.2 viser udvalgte geometriske objekter, der ofte anvendes. Foruden objektnavn er der angivet et link til hvert objekt, hvor man kan læse mere om de respektive objekter og se illustrative eksempler på, hvordan de fungerer.

Tabel 4.2: Udvalgte geometriske objekter i `ggplot2`

Navn	Funktion	Side: Cookbook for R
Bar plot	<code>geom_bar()</code>	<a href="#">Bar and line graphs</a>
Boxplot	<code>geom_boxplot()</code>	<a href="#">Plotting distributions</a>
Densitetsplot	<code>geom_density()</code>	<a href="#">Plotting distributions</a>
Histogram	<code>geom_histogram()</code>	<a href="#">Plotting distributions</a>
Punktdiagram	<code>geom_point()</code>	<a href="#">Scatterplots</a>

Det næste vi skal have styr på, er det æstetiske. Her anvendes `aes()` (forkortelse af *aesthetic*). Her specificeres det, hvilke variable vi skal visualisere m.v. Det er således også her det angives, hvis observationer skal have forskellige farver.

Som det kunne ses ovenfor, er der et standardtema, som `ggplot2` anvender. Der er tale om en karakteristisk grå baggrund, som ikke er decideret grim, men nemt kan ændres, hvis man vil have noget mere simpelt (eller bare noget andet).

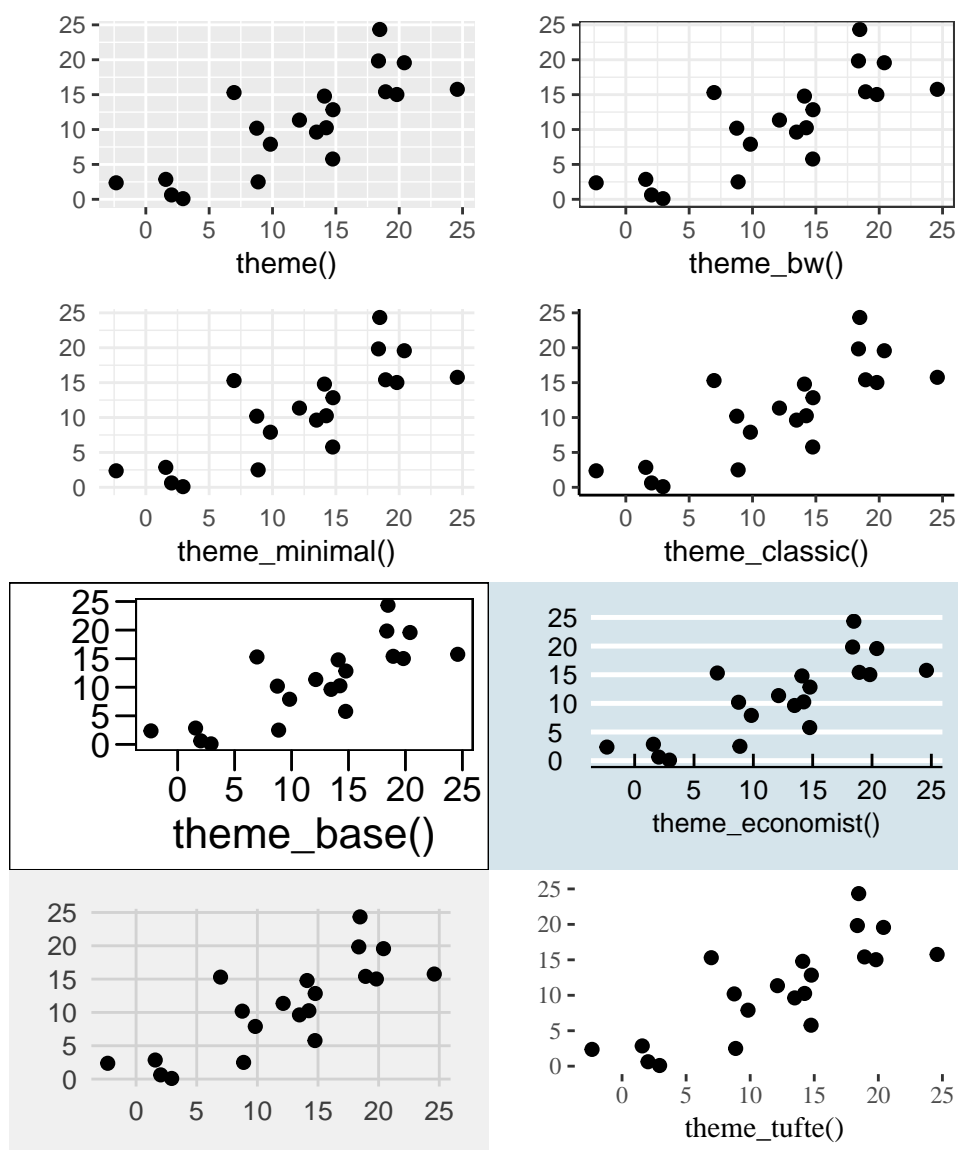
Tabel 4.3 viser en række udvalgte temaer der er at finde i hhv. `ggplot2` og [ggthemes](#). Det er kun udvalgte temaer, og `ggthemes` har eksempelvis også `theme_stata()`, og det siger sig selv, at dette tema *aldrig* skal anvendes.

Tabel 4.3: Udvalgte temaer til i `ggplot2`

Funktion	Pakke	Beskrivelse
<code>theme_bw()</code>	<code>ggplot2</code>	Mørke elementer på hvid baggrund
<code>theme_minimal()</code>	<code>ggplot2</code>	Minimalistisk tema
<code>theme_classic()</code>	<code>ggplot2</code>	Tema uden gitterlinjer
<code>theme_base()</code>	<code>ggthemes</code>	Kopi af base tema i R
<code>theme_economist()</code>	<code>ggthemes</code>	The Economist tema
<code>theme_fivethirtyeight()</code>	<code>ggthemes</code>	FiveThirtyEight tema
<code>theme_tufte()</code>	<code>ggthemes</code>	Tufte (1983) tema

Funktion	Pakke	Beskrivelse
----------	-------	-------------

Figur 4.2 viser hvordan de forskellige *themes* ser ud. Rækkefølgen er: Standard, `theme_bw()`, `theme_minimal()`, `theme_classic()`, `theme_base()`, `theme_economist()`, `theme_fivethirtyeight()`, `theme_tufte()`.



Figur 4.2: Otte forskellige temaer

Der findes flere ressourcer online, der beskæftiger sig med temaer til ggplot2, og foruden ovenstående kan [ggthemr](#) og [ggplot2 extensions](#) anbefales.

## 4.3 Eksempel: Venstre i meningsmålingerne i 2015

I dette afsnit gives flere eksempler på, hvordan man kan bygge figurer op. Idéen er ikke, at du efter at have læst dette afsnit kan - eller skal kunne - lave lignende figurer, men at du har en klar idé om hvordan figurer laves og kan finde inspiration i nedenstående, når du laver dine egne figurer.

Til dette vil vi bruge datasættet `polls.csv`, der indeholder data på en lang række meningsmålinger, der viser opbakningen til de partier, der enten er i Folketinget eller arbejder på at komme det. Vi bruger igen kommandoen `read.csv()` til at indlæse vores datasæt. Vi gemmer det i objektet `polls`:

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

Først skal vi have et overblik over datasættet. Til at gøre dette bruger vi `summary()`:

```
summary(polls)
```

id	pollingfirm	year	month
Min. : 1	Voxmeter:266	Min. :2010	Min. : 1.000
1st Qu.: 254	Gallup :132	1st Qu.:2011	1st Qu.: 4.000
Median : 507	Megafoon :123	Median :2013	Median : 6.000
Mean : 507	YouGov :107	Mean :2013	Mean : 6.563
3rd Qu.: 760	Rambøll :106	3rd Qu.:2015	3rd Qu.: 9.000
Max. :1013	Epinion :105	Max. :2016	Max. :12.000
	(Other) :174		

day	party_a	party_b	party_c
Min. : 1.00	Min. :14.40	Min. : 3.300	Min. : 2.100
1st Qu.: 8.00	1st Qu.:20.30	1st Qu.: 5.500	1st Qu.: 3.500
Median :13.00	Median :24.20	Median : 7.600	Median : 4.200
Mean :15.13	Mean :23.29	Mean : 7.277	Mean : 4.509
3rd Qu.:23.00	3rd Qu.:26.10	3rd Qu.: 8.725	3rd Qu.: 5.225
Max. :31.00	Max. :31.50	Max. :11.700	Max. :12.700
	NA's :1	NA's :1	NA's :1

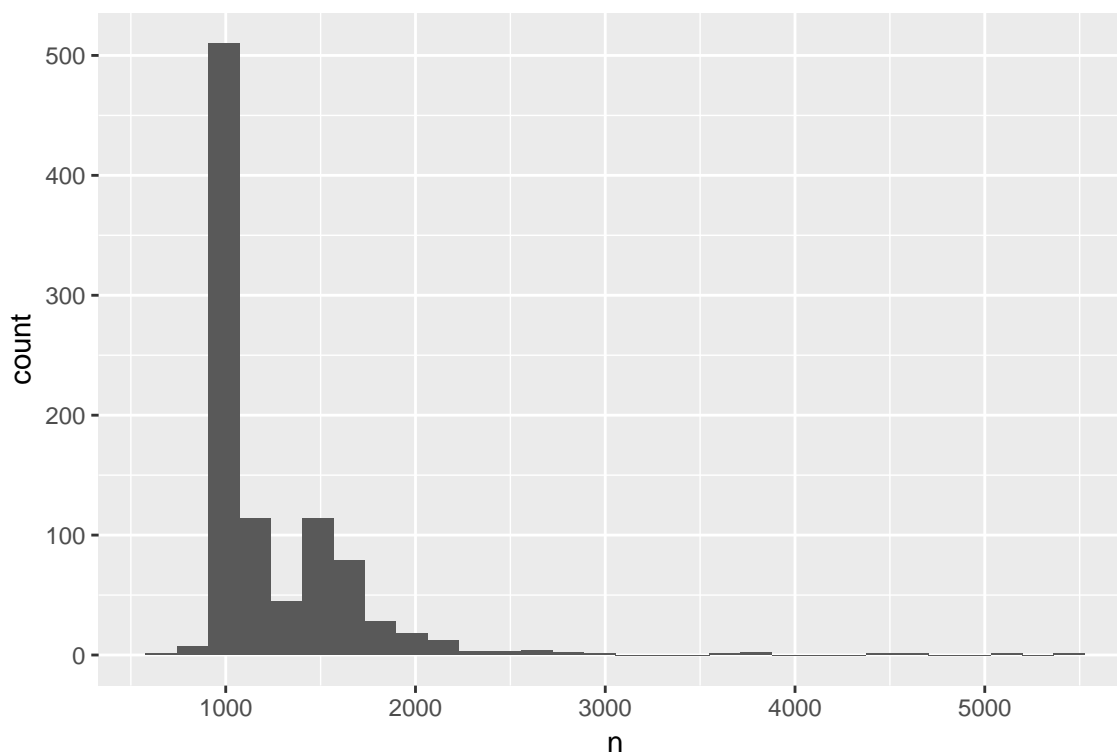
party_d	party_f	party_i	party_k
Min. :1.000	Min. : 2.400	Min. :0.200	Min. :0.0000
1st Qu.:1.800	1st Qu.: 4.775	1st Qu.:4.700	1st Qu.:0.5000

Median :2.700	Median : 6.100	Median :5.500	Median :0.7000
Mean :2.792	Mean : 7.256	Mean :5.682	Mean :0.7072
3rd Qu.:3.600	3rd Qu.: 7.900	3rd Qu.:6.900	3rd Qu.:0.9000
Max. :5.600	Max. :19.400	Max. :9.500	Max. :1.5000
NA's :988	NA's :1	NA's :1	NA's :624
party_o	party_v	party_oe	party_aa
Min. : 9.90	Min. :14.50	Min. : 2.100	Min. :0.900
1st Qu.:13.40	1st Qu.:20.60	1st Qu.: 7.200	1st Qu.:3.600
Median :17.10	Median :23.50	Median : 8.600	Median :4.800
Mean :16.73	Mean :24.51	Mean : 8.349	Mean :4.709
3rd Qu.:19.60	3rd Qu.:28.40	3rd Qu.: 9.800	3rd Qu.:6.000
Max. :24.50	Max. :36.40	Max. :14.900	Max. :9.300
NA's :1	NA's :1	NA's :1	NA's :689
n			source
Min. : 717			:908
1st Qu.:1016	<a href="http://voxmeter.dk/index.php/meningsmalinger/">http://voxmeter.dk/index.php/meningsmalinger/</a>		: 41
Median :1050	<a href="http://www.politiko.dk/barometeret">http://www.politiko.dk/barometeret</a>		: 17
Mean :1259	<a href="http://www.b.dk/politiko/barometeret">http://www.b.dk/politiko/barometeret</a>		: 2
3rd Qu.:1500	<a href="http://www.mx.dk/nyheder/danmark/story/17762091">http://www.mx.dk/nyheder/danmark/story/17762091</a>		: 2
Max. :5503	<a href="http://www.mx.dk/nyheder/danmark/story/20170441">http://www.mx.dk/nyheder/danmark/story/20170441</a>		: 2
NA's :65	(Other)		: 41

Som det kan ses har vi en lang række af variable i vores dataramme `polls`. `id`, der giver hver observation et id. `pollingfirm`, der angiver hvilket analyseinstitut, der har foretaget meningsmålingen. `year`, `month` og `day` angiver henholdsvis årstal, måned og dag for, hvornår dataindsamlingen for meningsmålingen blev fuldført. I vores datasæt har vi, som det kan ses, observationer i perioden fra 2010 til 2016. De næste variable angiver opbakningen til de forskellige partier, hvor de begynder med præfikset `party_` og dernæst partibogstavet. `party_v` er således partiet Venstres opbakning i meningsmålingen. Sidst har vi variabelen `n`, der angiver hvor mange der er blevet spurgt, og til sidst `source`, der angiver, hvor informationerne er fra.

Vi kan begynde med at lave et histogram over, hvor mange respondenter der normalt bliver spurgt i meningsmålingerne.

```
ggplot(polls, aes(x=n)) +  
  geom_histogram()
```

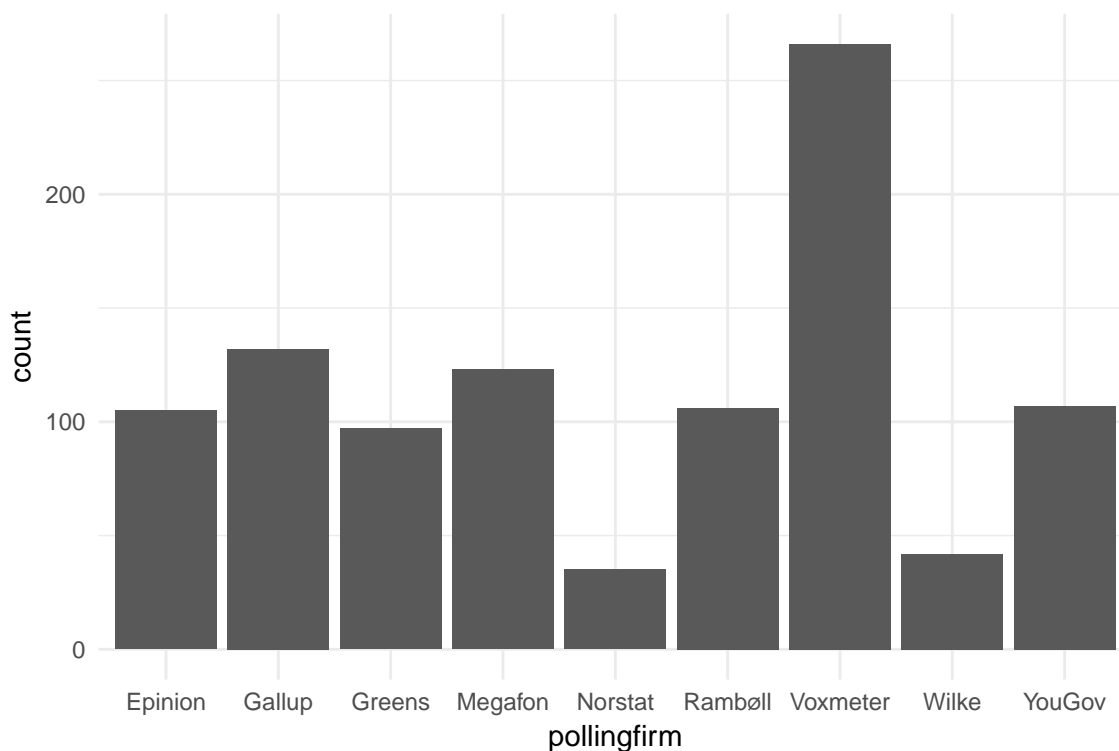


Som vi kan se spørger de fleste meningsmålinger omkring 1000 personer. Dette harmonerer fint med vores deskriptive statistik i ovenstående, hvor vi kunne se, at medianen var 1050.

Som det næste kan vi se på, hvilke institutter der har gennemført meningsmålingerne i vores datasæt. Til dette bruger vi `geom_bar()` i stedet for `geom_histogram`, og så tilføjer vi `theme_minimal()`, blot for at gøre figuren lidt pænere:

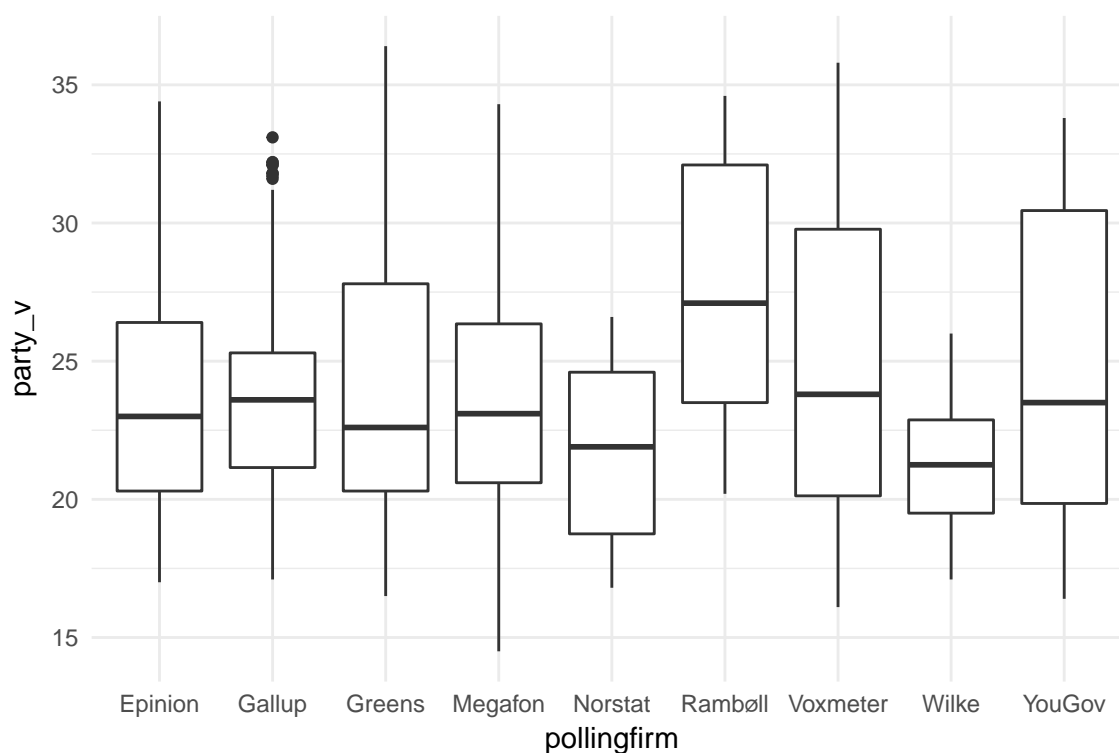
```
ggplot(polls, aes(x=pollingfirm)) +  
  geom_bar() +  
  theme_minimal()
```





Som det kan ses gennemfører Voxmeter flest meningsmålinger i denne periode (hvilket skyldes, at Voxmeter normalt foretager en meningsmåling om ugen). Vi kan også se på, hvordan partierne i hele perioden ligger hos de respektive analyseinstitutter. I nedenstående laver vi et boxplot med information om, hvordan partiet Venstre normalt ligger.

```
ggplot(polls, aes(x=pollingfirm, y=party_v)) +  
  geom_boxplot() +  
  theme_minimal()
```



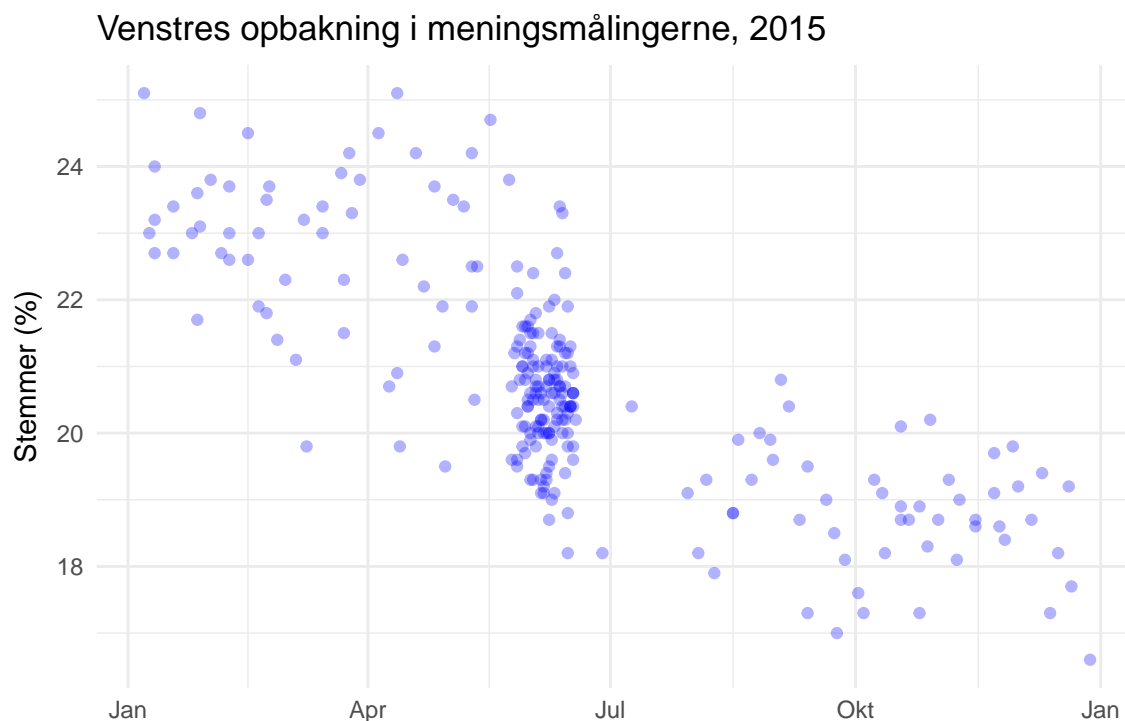
Det kan ses, at især Rambøll har Venstre højere oppe. Det skyldes dog ikke, at Rambøll overvurderer Venstre, men at meningsmålingerne fra Rambøll er fra tidligt i perioden (hvor Venstre lå højere i meningsmålingerne). Derfor er vi også interesseret i at kigge på Venstres opbakning over tid, hvor vi i nærværende tilfælde indskrænker os til 2015. For at gøre dette laver vi først en ny variabel, `date`, der samler information fra `year`, `month` og `day` og så indskrænker datasættet i objektet `polls` til kun at have meningsmålingerne fra 2015.

```
polls$date <- format(as.Date(c(paste(polls$year,
                                     polls$month,
                                     polls$day,
                                     sep="-")), by = "days"))

polls.2015 <- polls[polls$date > as.Date("2015-01-01") &
                    polls$date < as.Date("2015-12-31"),]
```

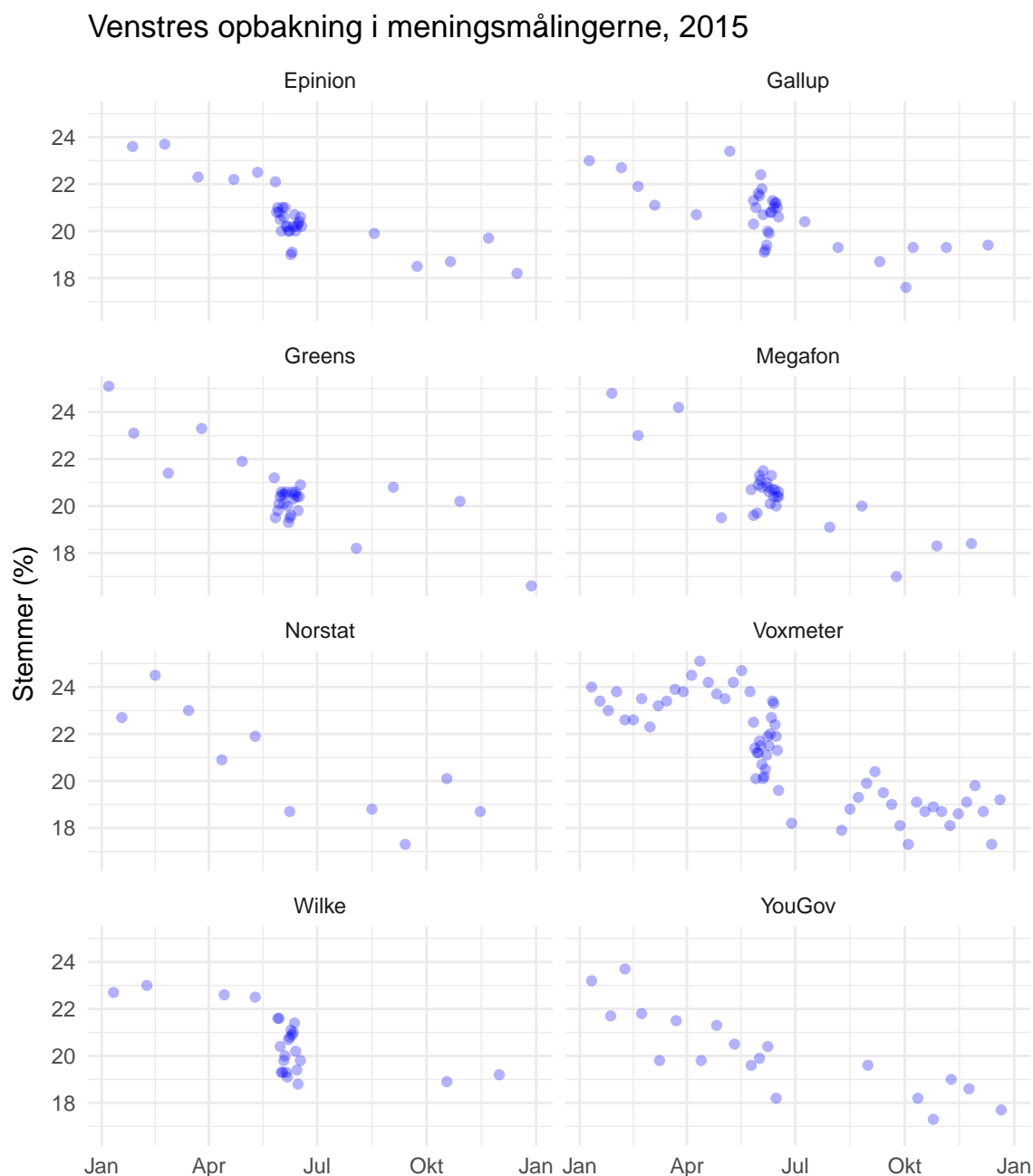
Dette kan vi illustrere med nedenstående kode. Bemærk at vi her også angiver, at `date` er en datovariabel (med `as.Date()`). Ligeledes tilføjer vi nogle andre specifikationer, der gør figuren lidt pænere og informativ.

```
ggplot(polls.2015, aes(x=as.Date(date), y=party_v)) +
  geom_point(colour="blue", alpha=.3) +
  ggtitle("Venstres opbakning i meningsmålingerne, 2015") +
  ylab("Stemmer (%)") +
  scale_x_date("", date_labels = c("%b")) +
  theme_minimal()
```



Figuren viser alle meningsmålingerne i 2015. Hvis vi gerne vil vise dem for de enkelte analyseinstitutter, kan vi bruge funktionen `facet_wrap()` og angive, at vi skal have en figur for hver kategori i `pollingfirm` og vil have dem vist i to kolonner:

```
ggplot(polls.2015, aes(x=as.Date(date), y=party_v)) +
  geom_point(colour="blue", alpha=.3) +
  ggtitle("Venstres opbakning i meningsmålingerne, 2015") +
  ylab("Stemmer (%)") +
  scale_x_date("", date_labels = c("%b")) +
  theme_minimal() +
  facet_wrap(~ pollingfirm, ncol=2)
```



## 4.4 Afsluttende bemærkninger

Nærværende kapitel har givet en introduktion til nogle af de grundlæggende elementer i, hvordan man visualiserer datarammer med `ggplot2`-pakken i R. Når du har lavet en figur, er det ofte også nødvendigt at eksportere den, altså få den gemt, så du kan inkludere den i dit arbejde. Til at gemme et plot bruger vi kommandoen `ggsave()`:

```
ggsave("filnavn.png")
```

Figuren vil blive gemt i dit *working directory*. Filtypen `.png` kan erstattes med andre formater, du skulle have interesse i. Hvis du har gemt figuren i et objekt, eksempelvis objektet `plot.1`, kan du også angive dette før selve angivelsen af filen:

```
ggsave(plot.1, "filnavn.png")
```

Når du har gemt din figur, vil du i mange tilfælde se, at du ikke er helt tilfreds med højden og bredden på figuren. Dette kan du ændre ved hjælp af specifikationerne `width` og `height`:

```
ggsave(plot.1, "filnavn.png", width = 4, height = 4)
```

Der er to pointer, der er vigtige at afslutte nærværende kapitel med. For det første, overordnet set, er det vigtigt at huske på, at man i en visualisering ikke oversælger sine data (eller modelestimater). Tufte (1983) opererer med begrebet løgnfaktor, der er effektstørrelsen i ens grafik divideret med effektstørrelsen i ens data. Der skal i ens præsentation gerne være en løgnfaktor på 1, altså en retmæssig præsentation af, hvad ens data rent faktisk viser.

For det andet har hensigten med dette kapitel været at give et indblik i, hvad man kan med `ggplot2`. Kapitlet er på ingen måde udtømmende, og der findes talrige guides på nettet, der mindst lige så godt som dette kapitel, klæder dig på til at visualisere data med `ggplot2` i R. Disse inkluderer blandt andet:

- [Introduction to R Graphics with ggplot2](#)
- [Building a ggplot2 step by step](#)
- [An Introduction on How to Make Beautiful Charts With R and ggplot2](#)
- [Scatter plots \(ggplot2 way\)](#)

Som den sidste ressource kan officielle dokumentation anbefales, der har en oversigt over alle *geoms*, *statistics*, *scales*, *aesthetics* osv.: [docs.ggplot2.org/current/](https://docs.ggplot2.org/current/)

# Kapitel 5

## OLS regression

OLS (eller *mindste kvadraters metode* på dansk) regression er et af de mest anvendte redskaber i den politologiske værktøjskasse. Den simple lineære regressionsanalyse kan skrives som:

$$Y_i = \alpha + \beta X_i + \epsilon_i$$

Her er  $Y_i$  den afhængige variabel, vi ønsker at forklare forskelle i (eksempelvis hvor højreorienterede forskellige vælgere er).  $\alpha$  er konstanten, en parameter der angiver værdien på den afhængige variabel, når den uafhængige variabel (eller de uafhængige variable) er 0.  $\beta$  angiver ændringen i  $Y$  når  $X$  stiger med én enhed. Det er oftest  $\beta$ , vi er interesseret i at undersøge effekten af på  $Y_i$ .  $\epsilon_i$  er fejlleddet.

Vi vil i dette kapitel anvende den danske del af *European Social Survey* fra 2014. Bemærk at dette ikke er det fulde datasæt, så det er ikke alle observationer (*rækker*) såvel som variable (*kolonner*), der er med. Det fulde datasæt kan hentes i forskellige formater hos [europeansocialsurvey.org](https://europeansocialsurvey.org). Det første vi gør er at indlæse vores datasæt i objektet `ess`.

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

For at få et indblik i de inkluderede variable i datarammen og observationerne deri, bruger vi først `head()`-funktionen:

```
head(ess)
```

```
  male age edu inc union lrscale
1    0  66  6  4     0         4
2    1  57  5  9     1         7
```

```

3    0  56   6   6    1    5
4    0  74   3   2    0    5
5    0  49   4   9    1    8
6    1  58   3   3    1    5

```

Som det kan ses er der seks variable. De er alle numeriske variable. `male` er køn, hvor 1 er mand og 0 er kvinde. `age` er alder i år. `edu` er uddannelse (i ISCED kategorier). `inc` er indkomst angivet i indkomstdecil (hvorfor der er 10 værdier). `union` angiver om man er medlem af en fagforening eller ej. `lrscale` er politisk orientering målt på en venstre-højre skala (hvor 0 er meget venstreorienteret og 10 er meget højreorienteret). Vi bruger `summary()` til at få deskriptiv statistik på de respektive variable:

```
summary(ess)
```

```

      male      age      edu      inc
Min.   :0.0000  Min.   :15.00  Min.   :1.000  Min.   : 1.000
1st Qu.:0.0000  1st Qu.:35.00  1st Qu.:3.000  1st Qu.: 4.000
Median :1.0000  Median :49.00  Median :4.000  Median : 6.000
Mean   :0.5339  Mean   :49.27  Mean   :4.263  Mean   : 5.842
3rd Qu.:1.0000  3rd Qu.:63.00  3rd Qu.:6.000  3rd Qu.: 8.000
Max.   :1.0000  Max.   :95.00  Max.   :7.000  Max.   :10.000

      union      lrscale
Min.   :0.0000  Min.   : 0.000
1st Qu.:0.0000  1st Qu.: 4.000
Median :1.0000  Median : 5.000
Mean   :0.6179  Mean   : 5.481
3rd Qu.:1.0000  3rd Qu.: 7.000
Max.   :1.0000  Max.   :10.000

```

Alternativt kan vi også bruge funktionen `glimpse()`, der kan bruges, hvis du har åbnet pakken `dplyr`. Denne viser også antallet af observationer, der er i datarammen.

```
glimpse(ess)
```

```

Observations: 1,285
Variables: 6

```

```
$ male    <int> 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ age     <int> 66, 57, 56, 74, 49, 58, 21, 65, 37, 27, 65, 27, 17, 74...
$ edu     <int> 6, 5, 6, 3, 4, 3, 2, 3, 7, 6, 3, 7, 2, 3, 3, 3, 7, 7, ...
$ inc     <int> 4, 9, 6, 2, 9, 3, 2, 6, 7, 9, 3, 7, 8, 5, 9, 6, 9, 10,...
$ union   <int> 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, ...
$ lrscale <int> 4, 7, 5, 5, 8, 5, 10, 8, 6, 7, 5, 7, 8, 2, 9, 7, 7, 8,...
```

I den resterende del af kapitlet gives først en introduktion til bivariate regressioner (altså med én uafhængig variabel), dernæst multivariate regressioner og til sidst introduceres en række relevante forudsætningstests.

## 5.1 Bivariat analyse

For at lave en OLS regression bruger vi funktionen `lm()`, der står for *linear model*. Denne funktion er en del af basispakken i R og kræver dermed ikke, at du åbner en bestemt pakke. For at lave en simpel OLS regression med en afhængig variabel og én uafhængig variabel, angiver vi den afhængige variabel før `~` i funktionen og den uafhængige variabel efter. Sidst angiver vi datasættet. I nedenstående ønsker vi at undersøge om folk der har en højere indkomst, er mere højreorienteret (politisk orientering er dermed den afhængige variabel). Dette ønsker vi at undersøge med datarammen `ess`, hvilket vi gemmer i objektet `reg.lr`.

```
reg.lr <- lm(lrscale ~ inc, data=ess)
```

Når vi har kørt ovenstående funktion får vi et objekt, der, når man bruger funktionen `class()` giver `lm`. Dette gør blandt andet, at når vi bruger funktioner på vores objekt, vil der blive taget højde for, at det er en lineær model (eksempelvis vil funktionen `plot()` på objektet kalde funktionen `plot.lm()`). For at se resultaterne af regressionen kan vi prøve at se, hvad der er i objektet `reg.lr`.

```
reg.lr
```

Call:

```
lm(formula = lrscale ~ inc, data = ess)
```

Coefficients:



```
(Intercept)      inc
      5.13282      0.05959
```

Som det kan ses af ovenstående, får vi ikke anden information frem end koefficienterne i modellen. Dette betyder i dette tilfælde konstanten (altså værdien på den afhængige variabel når `inc` er 0) og koefficienten for `inc`. Konstanten er 5.13282, hvilket betyder, at når `inc` er 0, er værdien på den afhængige variabel 5,13 (skæringen med y-aksen). Koefficienten for indkomst er 0.05959, hvilket betyder, at når indkomst stiger med én enhed, stiger politisk orientering med 0,0596. Disse informationer er generelt ikke tilstrækkelige, hvorfor det anbefales, at man bruger eksempelvis `summary()`, når man skal se resultaterne fra ens regression.

```
summary(reg.lr)
```

Call:

```
lm(formula = lrscale ~ inc, data = ess)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-5.729 -1.609 -0.252  1.688  4.808
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.13282     0.14661   35.01  < 2e-16 ***
inc          0.05959     0.02249    2.65  0.00816 **
```

---

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

Residual standard error: 2.332 on 1283 degrees of freedom

Multiple R-squared: 0.005442, Adjusted R-squared: 0.004667

F-statistic: 7.02 on 1 and 1283 DF, p-value: 0.008158

Her får vi et meget større *output*, der også indeholder signifikanstests, som vi ofte er interesseret i. Under `Estimate` ses koefficienterne (som ligeledes er angivet ovenfor). Under `Std. Error` får vi standardfejlene. `t value` viser t værdien og `Pr(>|t|)` giver p-værdien

(og ved siden af disse er der, såfremt der er tale om et statistisk signifikant estimat, en indikator herfor). Nedenunder er en lang række af modelestimater, herunder frihedsgrader, determinationskoefficient, F-test m.v.

Hvis man finder ovenstående uoverskueligt, er der andre måder, at få præsenteret resultaterne i R. En metode er at anvende pakken `stargazer`, der ofte anvendes til at eksportere tabeller. Først indlæser vi pakken (husk at installere den først, såfremt du ikke har den):

```
library("stargazer")
```

Med pakken kan vi bruge funktionen `stargazer()`. Her angiver vi først, at vi er interesseret i objektet `reg.lr` og ønsker at få det præsenteret som tekst (standard er LaTeX-kode).

```
stargazer(reg.lr, type="text")
```

```
=====
                        Dependent variable:
                        -----
                                lrscale
-----
inc                                0.060***
                                (0.022)

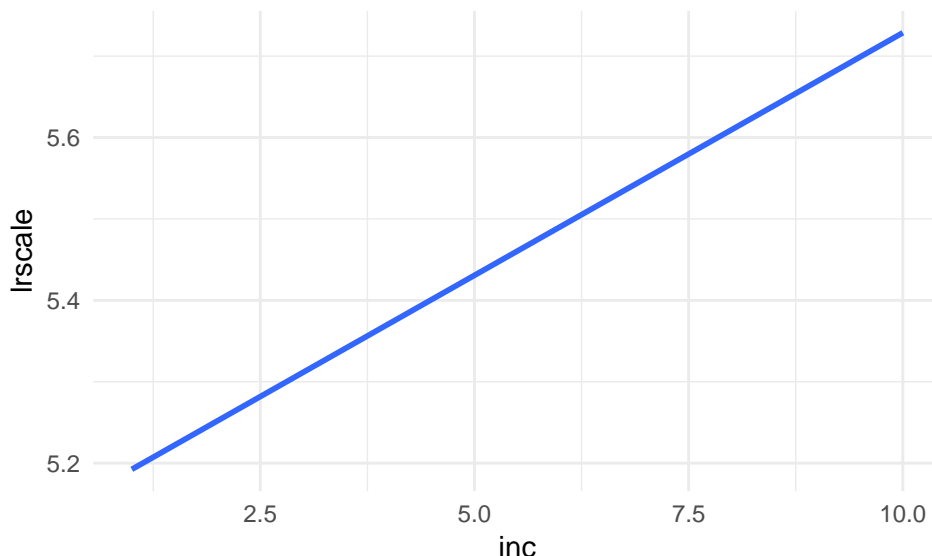
Constant                            5.133***
                                (0.147)

-----
Observations                        1,285
R2                                  0.005
Adjusted R2                         0.005
Residual Std. Error      2.332 (df = 1283)
F Statistic                7.020*** (df = 1; 1283)
=====
Note:                *p<0.1; **p<0.05; ***p<0.01
```

Her ses et mere brugervenligt *output*, der i format ligner det, man vil finde i artikler og bøger. Her kan vi ligeledes nemt se antallet af observationer i vores model (hvilket ikke var angivet

eksplicit, da vi brugte `summary()`). Hvis vi gerne vil illustrere den lineære regressionslinje, kan vi bruge `ggplot2` med tilføjelsen `geom_smooth()` og specificere, at det skal være en lineær model, der skal vises:

```
ggplot(ess, aes(x=inc, y=lrscale)) +  
  geom_smooth(method="lm", se=FALSE) +  
  theme_minimal()
```



Her kan det også ses, at når `inc` er 0, er værdien på y-aksen 5,13.

## 5.2 Multivariat analyse

Til nu har vi blot kørt en bivariat regressionsanalyse. Det er heldigvis nemt at udvide denne med flere uafhængige variable, der giver mulighed for at kontrollere for andre variable:

$$Y_i = \alpha + \beta_1 X_i + \beta_2 Z_i + \epsilon_i$$

For at gøre dette i R tilføjer vi et plus (+) efter den uafhængige variabel og derefter navnet på endnu en variabel, der skal inkluderes i modellen. Dette kan man fortsætte med at gøre, til ens model er korrekt specificeret. I nedenstående er indkomst, køn, alder, uddannelse og fagforeningsmedlemskab uafhængige variable.

```
reg.lr.c <- lm(lrscale ~ inc + male + age + edu + union, data=ess)
```

Dette er gemt i objektet `reg.lr.c`. Ligesom i den bivariate analyse kan vi få resultaterne af modellen frem ved hjælp af funktionen `summary()`:

```
summary(reg.lr.c )
```

Call:

```
lm(formula = lrscale ~ inc + male + age + edu + union, data = ess)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-6.2257	-1.5882	0.0025	1.8217	5.1478

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	5.278791	0.292423	18.052	< 2e-16 ***
inc	0.118345	0.024878	4.757	2.19e-06 ***
male	0.452107	0.129368	3.495	0.000491 ***
age	0.003823	0.003684	1.038	0.299628
edu	-0.176360	0.037958	-4.646	3.73e-06 ***
union	-0.270446	0.140951	-1.919	0.055241 .

---

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

Residual standard error: 2.294 on 1279 degrees of freedom

Multiple R-squared: 0.04013, Adjusted R-squared: 0.03637

F-statistic: 10.69 on 5 and 1279 DF, p-value: 4.383e-10

*Outputtet* følger samme struktur som i den bivariate analyse. Den eneste forskel er, at der nu er tilføjet fire ekstra variable og dermed fire ekstra estimater og dertilhørende standardfejl og statistiske tests. Hvis vi gerne vil sammenligne resultaterne i denne regression med den bivariate analyse, kan vi bruge `stargazer()` til at vise resultaterne fra begge modeller i én tabel. Dette gør vi ved at tilføje begge modeller til funktionen, adskilt af et komma:

```
stargazer(reg.lmr, reg.lmr.c, type="text")
```

Dependent variable:		
	lrscale	
	(1)	(2)
inc	0.060*** (0.022)	0.118*** (0.025)
male		0.452*** (0.129)
age		0.004 (0.004)
edu		-0.176*** (0.038)
union		-0.270* (0.141)
Constant	5.133*** (0.147)	5.279*** (0.292)
Observations	1,285	1,285
R2	0.005	0.040
Adjusted R2	0.005	0.036
Residual Std. Error	2.332 (df = 1283)	2.294 (df = 1279)
F Statistic	7.020*** (df = 1; 1283)	10.693*** (df = 5; 1279)

Tolkningen af resultaterne er anderledes i den multivariate regression. Koefficienten for `inc` skal nu tolkes partielt, og altså som korrelationen mellem `inc` og `lrscale`, når vi kontrollerer for de andre variable (køn, alder, uddannelse og fagforeningsmedlemsskab). Konstanten er i denne sammenhæng værdien på den afhængige variabel, når alle andre variable har værdien 0.

## 5.3 Forudsætningstests

I Model 4 i Figur 4.1 kan det ses, at én observation gør det muligt at lave en regressionslinje. Dette viser vigtigheden af at kende sine data, og især om de modeller der estimeres, bygger på realistiske antagelser. Som med alle modeller vi estimerer, bygger disse på bestemte antagelser. Disse antagelser kan være meget heroiske i forhold til strukturen af vores data, og dermed blive brudt. Nogle af disse antagelser kan heldigvis belyses empirisk, hvor andre kræver viden omkring selve den proces, der har genereret vores data.

Det er for det første vigtigt, at vi har et godt forskningsdesign. Dette holder for *alt* vi laver - ikke kun lineære regressionsanalyser. Som det siges: *You can't fix by analysis what you bungled by design.*

Det er for det andet vigtigt, at vi bruger de rigtige variable - og hverken færre eller flere. Med andre ord skal vi sikre os, at vi har den korrekte model, og ikke en fejlspecificeret model. Hvis vores model er underspecificeret, altså vi undlader vigtige variable, får vi *biased* estimater. Det samme kan vi få, hvis vi overspecificerer vores model, eksempelvis ved at smide alle variable ind i vores model. Det er derfor vigtigt, at vi bruger vores teoretiske viden, når vi specificerer vores regression.

I den resterende del af kapitlet vil forskellige forudsætningstests blive gennemgået. Til at gøre dette vil vi bruge to forskellige pakker, `car` og `MASS`. Det første vi gør er at åbne pakkerne (husk at installere dem først, hvis du ikke har dem).

```
library("car")  
library("MASS")
```

### 5.3.1 Linearitet

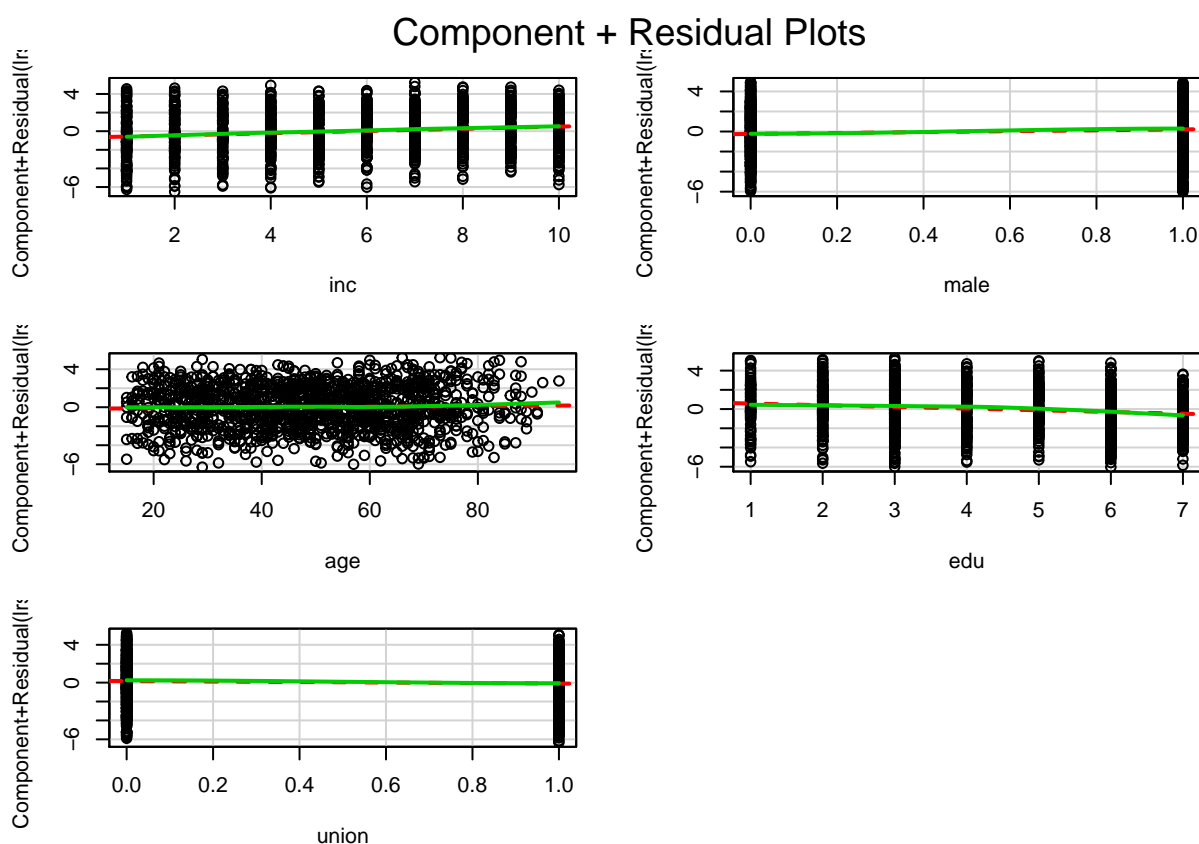
Det første vi skal undersøge empirisk er, om der et lineært forhold mellem vores uafhængige og afhængige variable. Hvis der ikke er dette, vil vi få *biased* estimater, og dermed kan vi ikke tro på resultaterne. Vi kunne gøre dette ved at kigge på punktdiagrammer for den afhængige variabel og hver af de uafhængige variable, men dette vil ikke tage højde for, at relationen

mellem den afhængige variabel og én uafhængig variabel, kan ændres, når man tager højde for de andre uafhængige variable. Derfor er vi nødt til at kigge på partielle korrelationer.

Til at gøre dette vil vi bruge funktionen `crp()`, der er en forkortelse for *component + residual plot*. Disse plots giver os en grafisk illustration af, om der er et lineært forhold mellem hver af den afhængige variable og hver af de uafhængige variable, når der tages højde for de andre variable.

Det gode ved disse plots er, at de også tilføjer en linje, der bedst passer til data (den grønne linje). Den røde linje er den lineære relation. Vi vil gerne have, at der er et klart overlap mellem de to linjer, så der rent faktisk er et lineært forhold mellem vores variable. Her kører vi funktionen `crp()` på vores objekt med den multiple lineære regression, vi estimerede i forrige sektion.

```
crp(reg.lr.c)
```



Der er et pænt overlap i de figurer, hvor det er muligt at estimere begge linjer. Dette er ikke muligt i alle tilfælde, og mere specifikt ikke for de uafhængige variable, der er binære.

Hvis der er problemer med ovenstående, skal man overveje en alternativ modelspecifikation. Dette kan være ved at inkludere variable, der giver et mere lineært forhold mellem de respektive

variable i modellen, eller transformere de eksisterende variable i modellen (eksempelvis via en logaritmisk transformation).

### 5.3.2 Outliers og indflydelsesrige observationer

Det næste vi skal belyse er forekomsten af outliers og indflydelsesrige observationer. En outlier er en observation, der ligger langt fra den lineære tendenslinje og dermed ikke følger den trend, man burde forvente med udgangspunkt i de andre observationer. For at undersøge dette skal vi teste residualen for en observation. Til at gøre dette estimerer vi studentiserede residualer, der fortæller os, hvor overraskende et punkt er relativt til tendenslinjen uden det pågældende punkt. Her bruger vi funktionen `outlierTest()`.

```
outlierTest(reg.lm.c)
```

```
No Studentized residuals with Bonferonni p < 0.05
```

```
Largest |rstudent|:
```

	rstudent	unadjusted	p-value	Bonferonni	p
597	-2.72712		0.0064764		NA

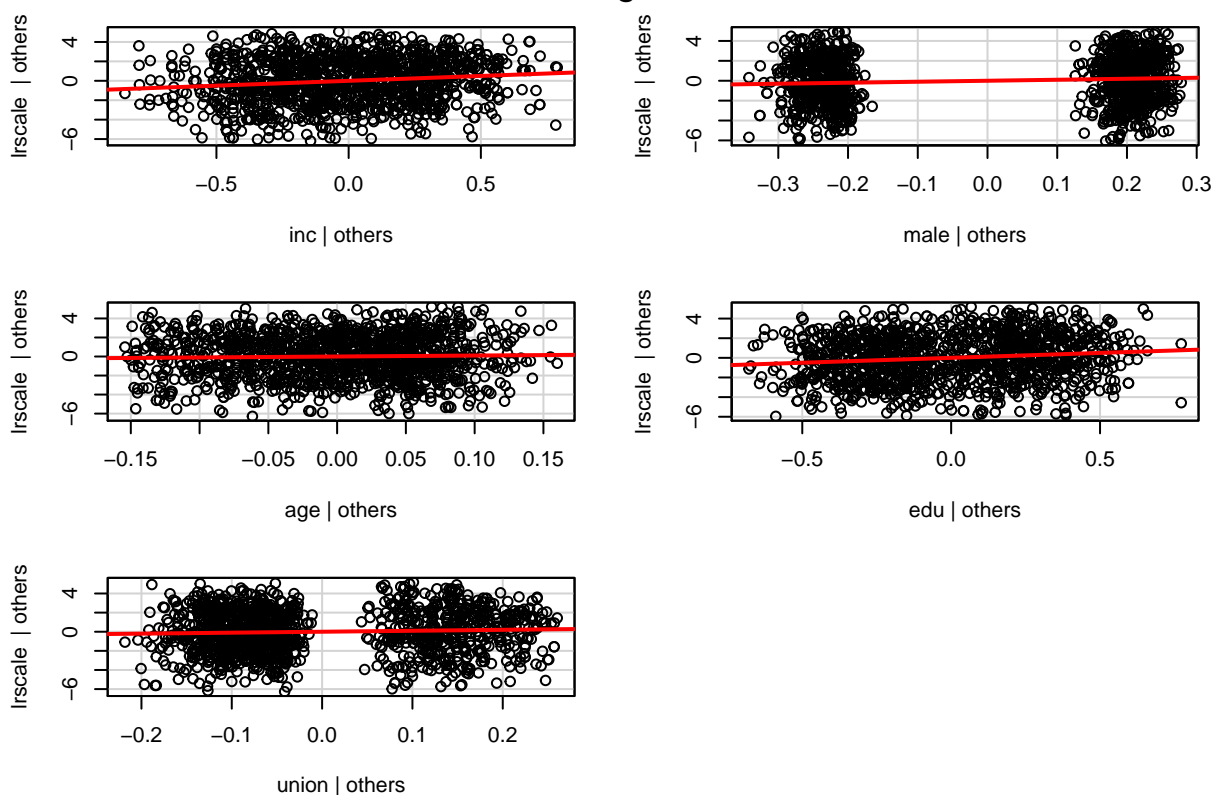
I outputtet er vi interesserede i Bonferonni p-værdier, og konkret vil vi gerne have, at ingen af disse er signifikante. I outputtet kan vi i ovenstående se, at der ikke er studentiserede residualer med Bonferonni p-værdier lavere end 0,05, hvorfor der ikke er et problem med ekstreme observationer.

Det næste vi kigger på er såkaldte *leverage plots*, der skal belyse om der er problemer med ekstreme værdier forudsagt af hver af de uafhængige variable. Til at få dette, bruger vi funktionen `leveragePlots()`. Her finder vi en lige fordeling omkring de horisontale linjer, hvilket indikerer, at der ikke er problemer med outliers.

```
leveragePlots(reg.lm.c)
```



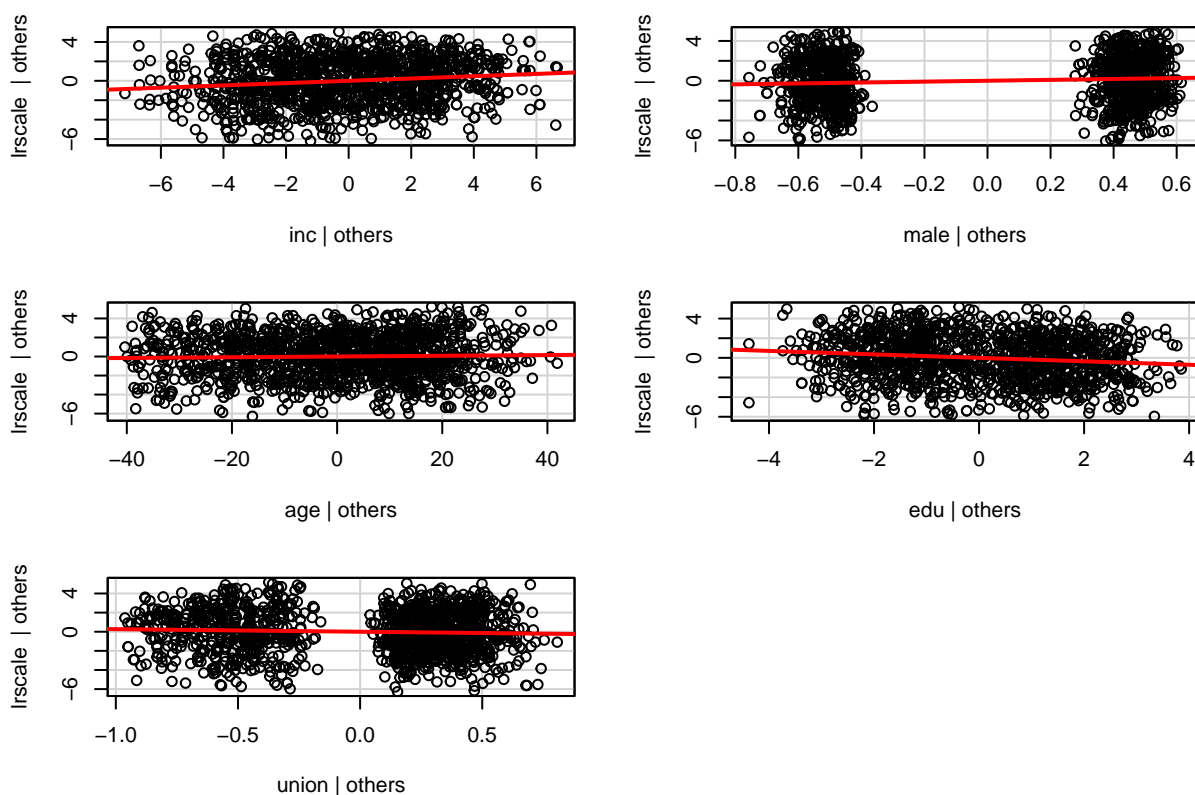
## Leverage Plots



Lignende plots der bruges til at kigge på indflydelsesrige observationer, er *added-variable plots*, der laves med funktionen `avPlots()`. Disse undersøger effekten af de uafhængige variable på den afhængige variabel, hvor vi i nedenstående ikke finder evidens for, at enkelte observationer afviger betydeligt og ekstremt fra andre.

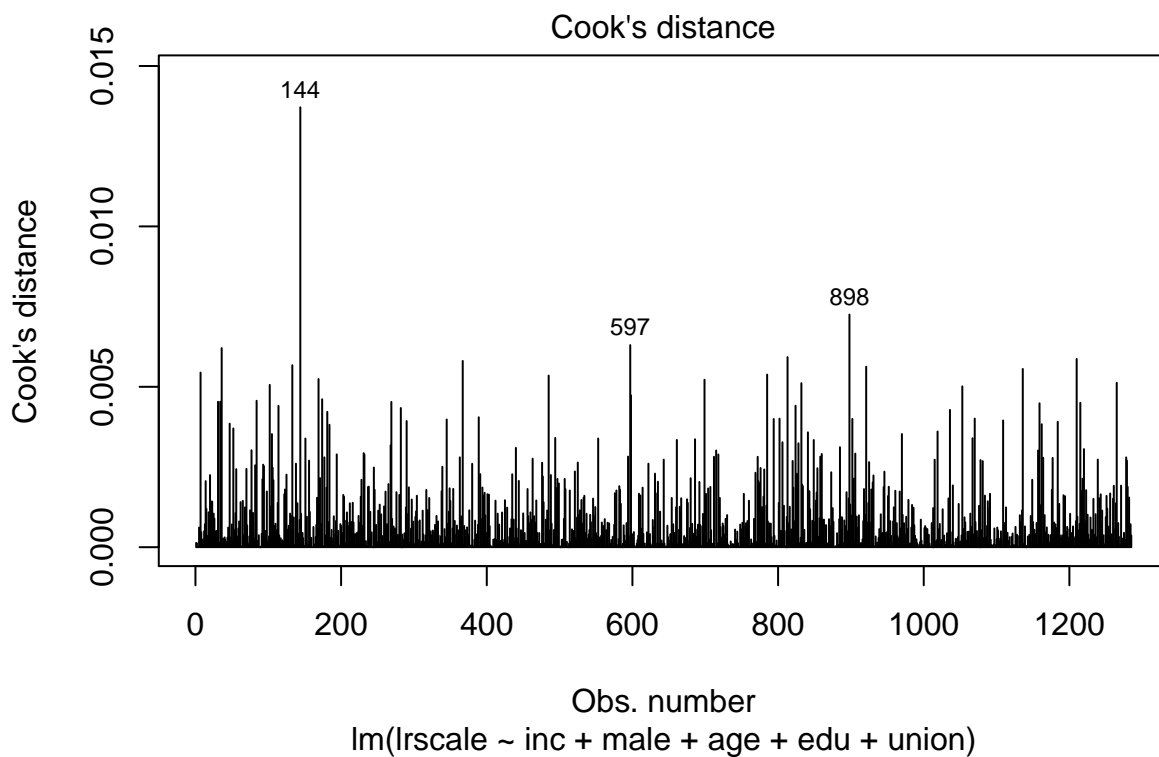
```
avPlots(reg.lr.c)
```

## Added-Variable Plots



Som det næste kan vi kigge på Cook's D, hvor skal se hvilke observationer der har Cook's D værdier der er større end  $4/(n-k-1)$ , hvor  $n$  er antallet af observationer og  $k$  er antallet af variable i vores model.

```
cutoff <- 4 / (( nrow(ess) - length(reg.lm.c$coefficients) - 1 - 1 ))
plot(reg.lm.c, which=4, cook.levels=cutoff)
```

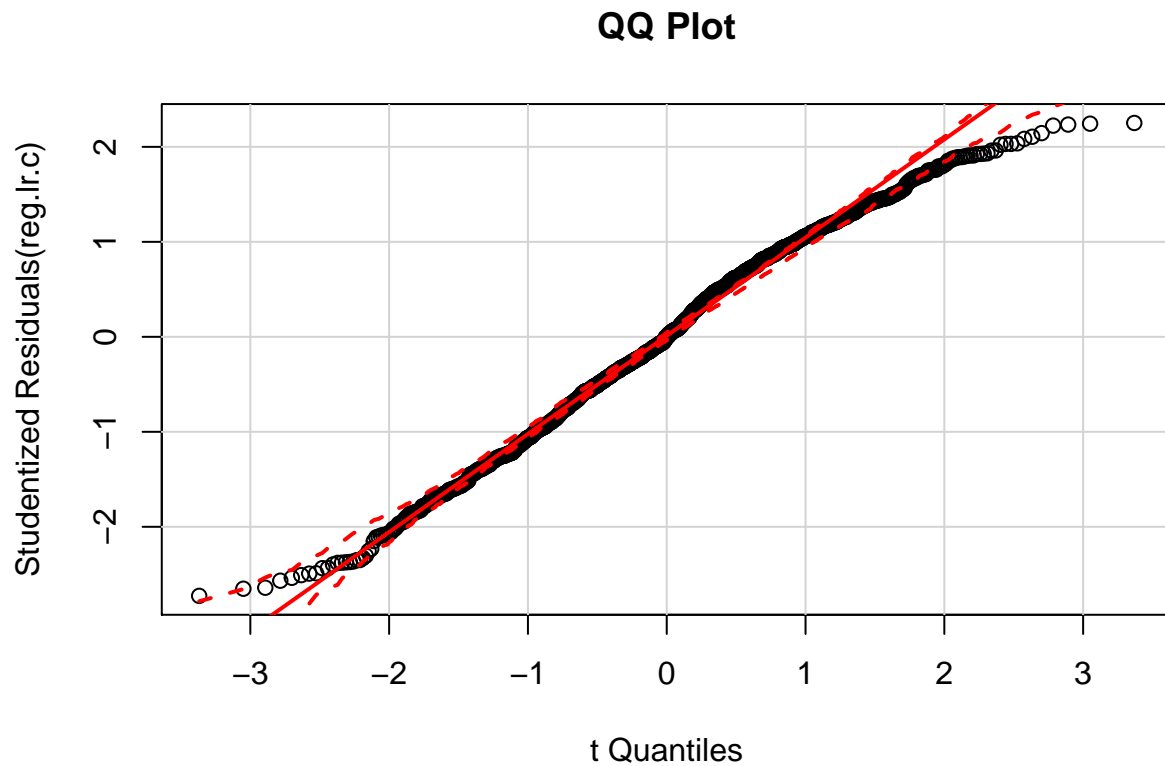


Vores *cutoff* er 0.0031323, og i figuren vises de observationer, der overskrider denne værdi. Det næste vi kan gøre er at kigge på et

### 5.3.3 Normalfordelte fejllid

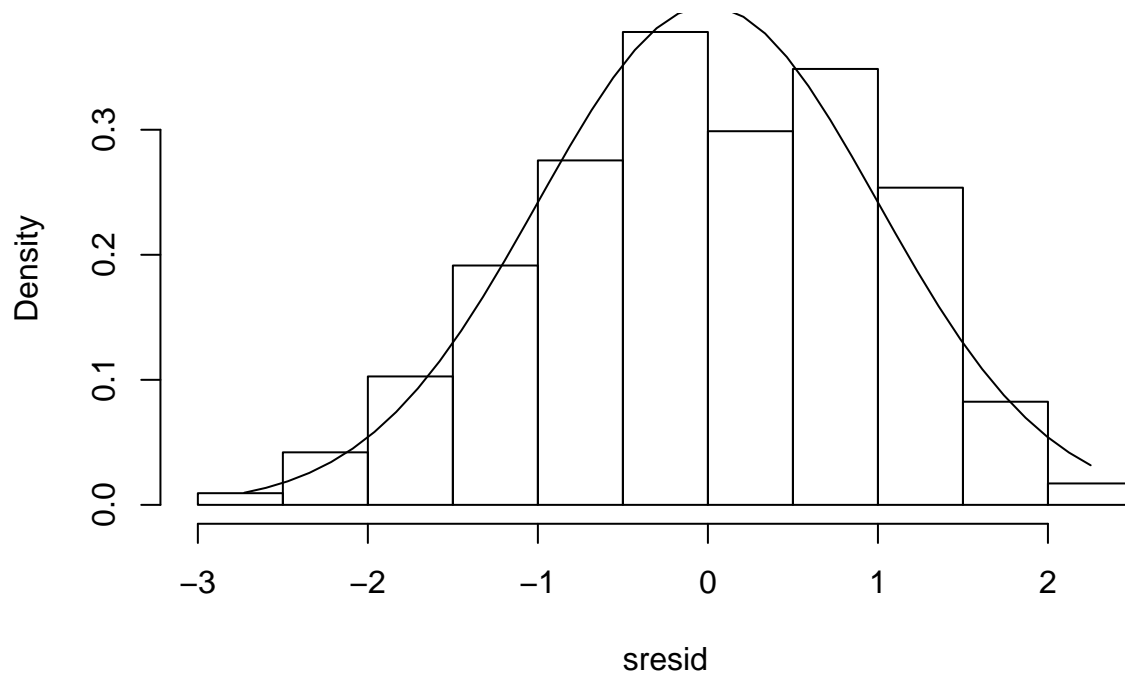
Bliver tilføjet senere.

```
qqPlot(reg.lm.c, main="QQ Plot")
```



```
sresid <- studres(reg.lm.c)
hist(sresid, freq=FALSE,
     main="Distribution of Studentized Residuals")
xfit<-seq(min(sresid),max(sresid),length=40)
yfit<-dnorm(xfit)
lines(xfit, yfit)
```

### Distribution of Studentized Residuals



-

#### 5.3.4 Heteroskedasticitet

*Bliver tilføjet senere.*

```
ncvTest(reg.lr.c)
```

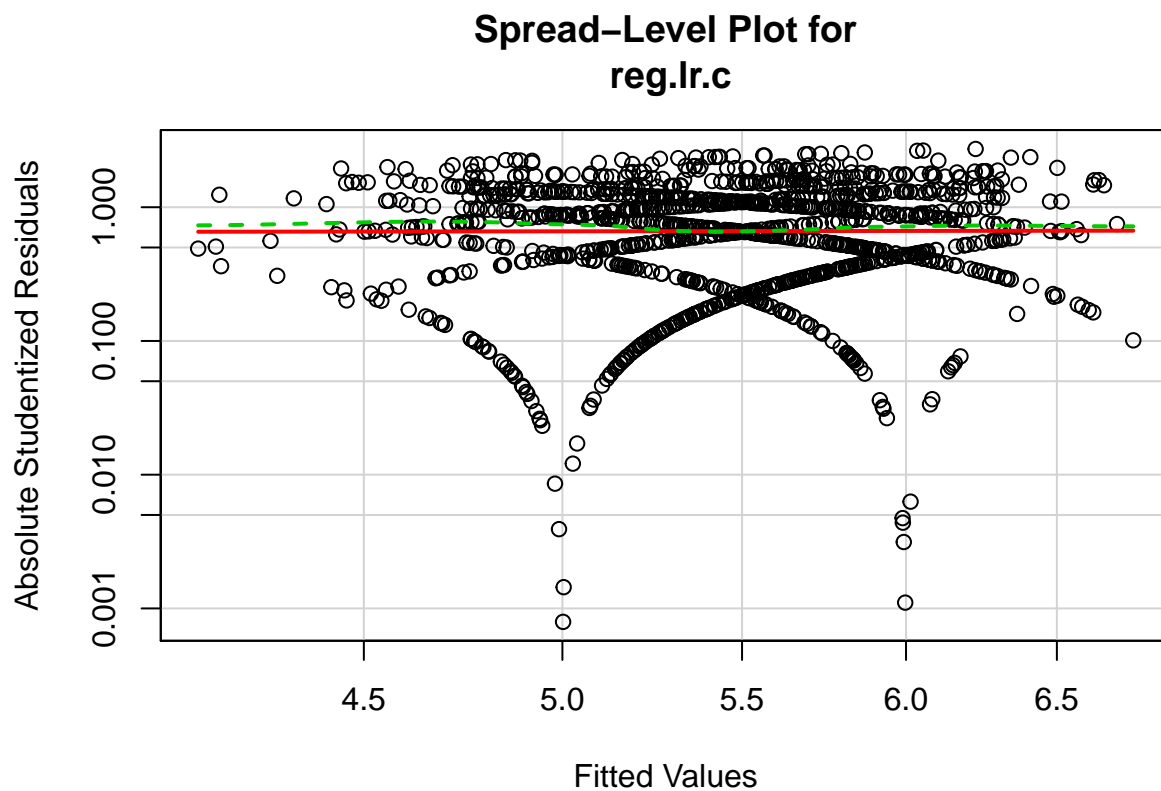
Non-constant Variance Score Test

Variance formula: ~ fitted.values

Chisquare = 0.06267413      Df = 1      p = 0.8023182

-

```
spreadLevelPlot(reg.lr.c)
```



Suggested power transformation: 0.9625135

–

```
library("lmtest")
```

Warning: package 'lmtest' was built under R version 3.3.2

Loading required package: zoo

Warning: package 'zoo' was built under R version 3.3.2

Attaching package: 'zoo'

The following objects are masked from 'package:base':

as.Date, as.Date.numeric

```
bptest(reg.lr.c, studentize = FALSE)
```

Breusch-Pagan test

data: reg.lr.c

BP = 14.36, df = 5, p-value = 0.01348

-

```
library("sandwich")
```

Warning: package 'sandwich' was built under R version 3.3.2

```
coeftest(reg.lr.c)
```

t test of coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	5.2787906	0.2924231	18.0519	< 2.2e-16 ***
inc	0.1183454	0.0248777	4.7571	2.187e-06 ***
male	0.4521067	0.1293676	3.4947	0.0004907 ***
age	0.0038226	0.0036839	1.0377	0.2996278
edu	-0.1763603	0.0379581	-4.6462	3.731e-06 ***
union	-0.2704461	0.1409507	-1.9187	0.0552411 .

---

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

```
coeftest(reg.lr.c,vcov=vcovHC(reg.lr.c,type="HCO"))
```

t test of coefficients:

Estimate	Std. Error	t value	Pr(> t )
----------	------------	---------	----------

```

(Intercept)  5.2787906  0.3055163 17.2783 < 2.2e-16 ***
inc          0.1183454  0.0248774  4.7571 2.187e-06 ***
male         0.4521067  0.1285771  3.5162 0.000453 ***
age          0.0038226  0.0038626  0.9896 0.322536
edu         -0.1763603  0.0376846 -4.6799 3.175e-06 ***
union       -0.2704461  0.1405988 -1.9235 0.054635 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```
coeftest(reg.lr.c,vcov=vcovHC(reg.lr.c,type="HC1"))
```

t test of coefficients:

```

              Estimate Std. Error t value  Pr(>|t|)
(Intercept)  5.2787906  0.3062321 17.2379 < 2.2e-16 ***
inc          0.1183454  0.0249357  4.7460 2.308e-06 ***
male         0.4521067  0.1288783  3.5080 0.0004671 ***
age          0.0038226  0.0038717  0.9873 0.3236674
edu         -0.1763603  0.0377729 -4.6690 3.346e-06 ***
union       -0.2704461  0.1409282 -1.9190 0.0552024 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

### 5.3.5 Multikollinearitet

*Bliver tilføjet senere.*

```
vif(reg.lr.c)
```

```

      inc      male      age      edu      union
1.263889 1.016578 1.048884 1.190281 1.144920

```



```
sqrt(vif(reg.lm.c)) > 2
```

```
inc  male  age  edu union  
FALSE FALSE FALSE FALSE FALSE
```

—

# Kapitel 6

## Interaktionsanalyse

*Bliver tilføjet senere.*

$$Y_i = \alpha + \beta_1 X_i + \beta_2 Z_i + \beta_3 (X_i \times Z_i) + \epsilon_i$$

–

```
library("margins")  
library("interplot")  
library("interflex")
```

–

# Kapitel 7

## Logistisk regression

*Bliver tilføjet senere.*

# Kapitel 8

## Propensity Score Matching

I dette kapitel gives en introduktion til, hvordan man gennemfører analyser med matching. Der vil - som i de andre kapitler - ikke blive givet en introduktion til statistikken bag, så for de læsere, der skulle have en interesse i at læse et par introduktionsartikler, kan Ho, Imai, King, & Stuart (2007) og Sekhon (2009) anbefales.

Vi vil gøre brug af tre pakker i dette kapitel. Den primære pakke, der vil gøre det meste af arbejdet, er `MatchIt` (Ho, Imai, King, & Stuart, 2011). Der findes forskellige pakker, der alle indeholder tilsvarende funktioner, men `MatchIt` er blandt de nemmeste at anvende, hvorfor den også vil blive brugt her. De to andre pakker vi skal bruge, er `ggplot2` (til at visualisere data) og `RITools` (til at undersøge balancen mellem ikke-matched og matched data). Først indlæser vi pakkerne:

```
library("ggplot2")
library("MatchIt")
library("RITools")
```

Vi vil igen gøre brug af den danske del af European Social Survey fra 2014. For en nærmere beskrivelse af disse data og de respektive variable, henvises der til kapitlet om OLS regression.

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

I dette eksempel ønsker vi at undersøge, om personer, der er medlem af en fagforening, er mere højreorienterede end folk, der ikke er medlem. Vi har dog en idé om, at køn, alder, uddannelse og indkomst, kan være med til at forklare forskelle mellem dem, der er medlem af en fagforening og dem der ikke er. Med andre ord er de to grupper (hhv. gruppen af medlemmer

og ikke-medlemmer) ikke sammenlignelige. Det første vi gør her er at specificere, at fagforeningsmedlemsskab er relateret til køn, alder, uddannelse og indkomst. Denne information gemmer vi i objektet `treat.f`:

```
treat.f <- union ~ male + age + edu + inc
```

Det første vi skal gøre er at have *propensity scores*, altså sandsynlighedsværdier, for, at en person er medlem af en fagforening eller ej, som en funktion af de respektkive uafhængige variable. Til at gøre dette estimerer vi først en logistisk regression med funktionen `glm()`, der står for *generalized linear model*. Dette gemmer vi i objektet `fit`, hvorefter vi bruger funktionen `predict()` til at få sandsynlighedsværdien for hver respondent, og denne gemmes i datarammen `ess` i variabelen `pscores`.

```
fit <- glm(treat.f, family=binomial, data=ess)

ess$pscores <- predict(fit, type="response")
```

Det næste vi gør er at bruge *nearest neighbor* matching underlagt en 0,1 kaliber (for de forskellige metoder, se hjælpefilen til `matchit()`). Resultaterne heraf gemmer vi i objektet `m`:

```
m <- matchit(treat.f, method = "nearest", caliper=.1, data = ess)
m
```

Call:

```
matchit(formula = treat.f, data = ess, method = "nearest", caliper = 0.1)
```

Sample sizes:

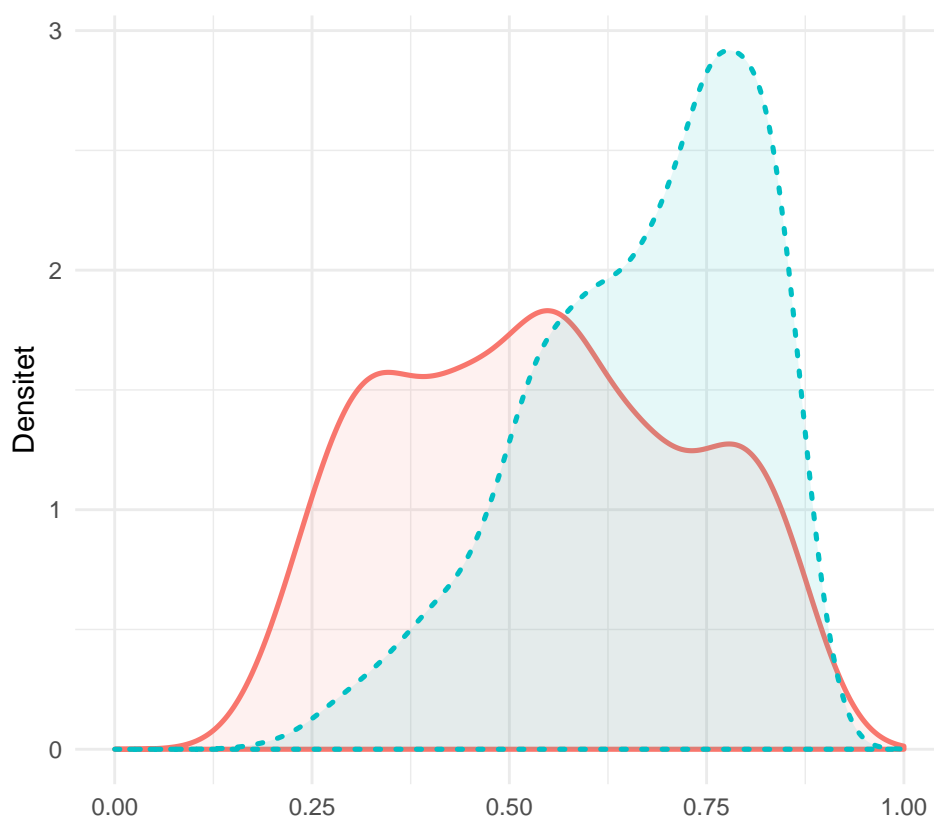
	Control	Treated
All	491	794
Matched	387	387
Unmatched	104	407
Discarded	0	0

I ovenstående kan vi se, at ikke alle cases kunne matches. Mere specifikt ender vi med 387 respondenter i hver gruppe. Vi bruger nu funktionen `match.data()` til at få en dataramme, for de respondenter, der er blevet matchet.

```
m.data <- match.data(m)
```

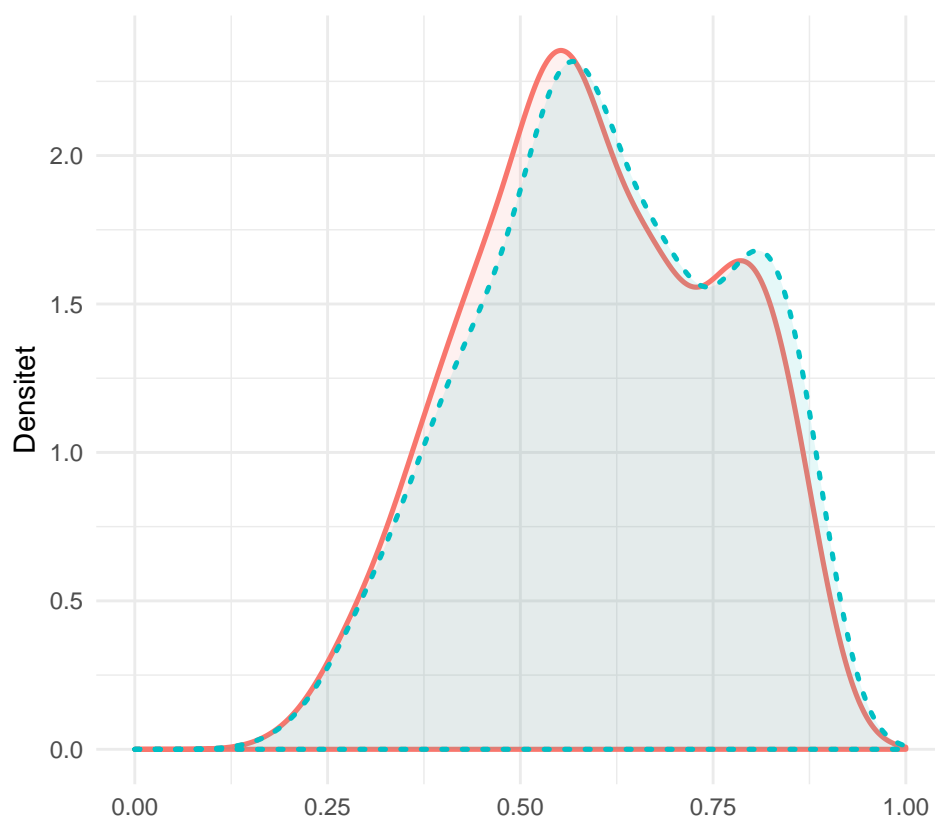
Nu kan vi så sammenligne sandsynlighedsværdierne for henholdsvis de ikke-matchede og matchede datasæt. Først kigger vi på sandsynlighedsværdierne for de ikke-matchede data:

```
ggplot(ess, aes(x=pscores, linetype=as.factor(union),
               fill = as.factor(union),
               colour = as.factor(union))) +
  geom_density(alpha = .1, size = .9) +
  ylab("Densitet") +
  theme_minimal() +
  theme(legend.position="none") +
  scale_x_continuous("", limits=c(0,1))
```



Her ses det tydeligt, at der ikke er et tilfredsstillende overlap mellem de to grupper. Som det næste kigger vi på vores overlap i de matchede data:

```
ggplot(m.data, aes(x=distance, linetype=as.factor(union),
                  fill = as.factor(union),
                  colour = as.factor(union))) +
  geom_density(alpha = .1, size = .9) +
  ylab("Densitet") +
  theme_minimal() +
  theme(legend.position="none") +
  scale_x_continuous("", limits=c(0,1))
```



Her ses det omvendt, at der er et langt mere tilfredsstillende overlap mellem de ikke-matchedede og de matchedede data. Næst ønsker vi at belyse, om matching proceduren også har skabt bedre balance mellem de respektive grupper på de inkluderede, observerede variable. Til at gøre dette bruger vi funktionen `xBalance()`, hvorefter vi laver en ny dataramme med de standardiserede forskelle på de inkluderede variable før og efter matching. Denne dataramme kalder vi `balance.df`:

```

xB.unmatched <- xBalance(treat.f, data=ess, report=c("all"))
xB.unmatched <- as.data.frame(xB.unmatched)
std.unmatched <- xB.unmatched[, "results.std.diff.unstrat"]
xB.matched <- xBalance(treat.f, data=m.data, report=c("all"))
xB.matched <- as.data.frame(xB.matched)
std.matched <- xB.matched[, "results.std.diff.unstrat"]
balance.df <- data.frame(covariate = row.names(xB.matched),
                        unmatched=
                            std.unmatched[row.names(xB.unmatched)
                                           %in% row.names(xB.matched)],
                        matched=std.matched)

```

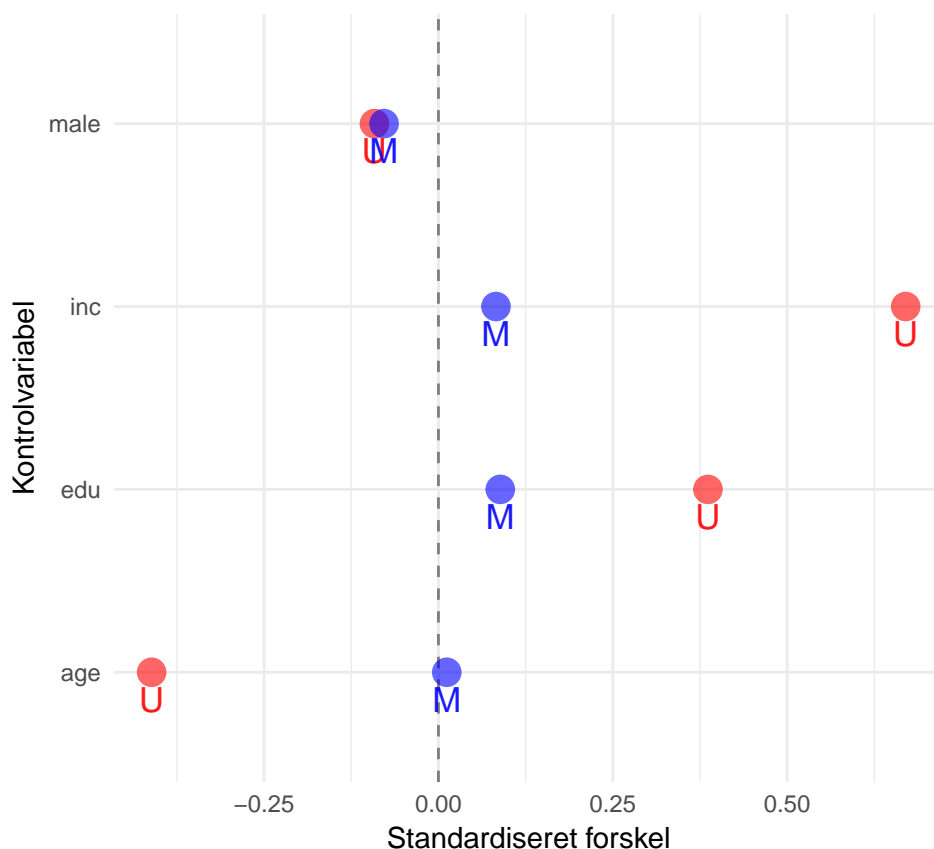
Resultaterne fra denne dataramme visualiserer vi så i nedenstående:

```

ggplot(balance.df, aes(x=covariate, y=unmatched)) +
  geom_hline(yintercept=0, linetype="dashed", colour="gray50") +
  geom_text(vjust = 1.6, aes(label="U", y=unmatched, size=.9),
            colour="red", alpha = 0.9) +
  geom_point(aes(y=unmatched, size=.9), colour="red",
             alpha = 0.6) +
  geom_text(vjust = 1.6, aes(label="M", y=matched, size=.9),
            colour="blue", alpha = 0.9) +
  geom_point(aes(y=matched, size=.9), colour="blue",
             alpha = 0.6) +
  coord_flip() +
  xlab("Kontrolvariabel") +
  ylab("Standardiseret forskel") +
  theme_minimal() +
  theme(legend.position="none")

```





Ovenstående viser, at de standardiserede forskelle er betydeligt mindre for de matchede data. Vi er således tilfredse med vores procedure, og kan nu køre vores analyser. Det første vi gør er at estimere en model på de ikke-matchede data. Dette gemmer vi i objektet `reg.unmatched`.

```
reg.unmatched <- lm(lrscale ~ union + male + age + edu + inc,
                    data=ess)
```

Dernæst kører vi en regression på de matchede data. Dette gemmer vi i objektet `reg.matched`:

```
reg.matched <- lm(lrscale ~ union + male + age + edu + inc,
                  data=m.data)
```

Vi kan så, forudsat at pakken `stargazer` er indlæst, se resultaterne med henholdsvis de ikke-matchede og de matchede data:

```
stargazer(reg.unmatched, reg.matched, type="text")
```

Dependent variable:		
	lrscale	
	(1)	(2)
union	-0.270* (0.141)	-0.316* (0.165)
male	0.452*** (0.129)	0.517*** (0.166)
age	0.004 (0.004)	0.002 (0.004)
edu	-0.176*** (0.038)	-0.175*** (0.047)
inc	0.118*** (0.025)	0.147*** (0.031)
Constant	5.279*** (0.292)	5.158*** (0.338)
Observations	1,285	774
R2	0.040	0.055
Adjusted R2	0.036	0.049
Residual Std. Error	2.294 (df = 1279)	2.284 (df = 768)
F Statistic	10.693*** (df = 5; 1279)	8.930*** (df = 5; 768)
Note: *p<0.1; **p<0.05; ***p<0.01		

# Kapitel 9

## Regressionsdiskontinuitetsdesigns

*Bliver tilføjet senere.*

# Konklusion

I de forudgående kapitler er givet en introduktion til grundlæggende funktioner og muligheder i R. Dette er stadig en bog under udarbejdelse, hvorfor der stadig mangler uddybende beskrivelser hist og her. Ligeledes vil der i fremtiden blive tilføjet nye kapitler, der introducerer flere andre funktioner i R.

Hvis du ikke kan vente på dette, er der heldigvis en lang række af statistikbøger, der også bruger og introducerer R. Tabel 4.1 giver et overblik over nogle af disse bøger, samt hvilket niveau de er på.

Tabel 4.1: Introduktionsbøger der anvender R

Bog	Titel	Niveau
A. Field et al. (2012)	Discovering Statistics Using R	Introducerende
Monogan III (2015)	Political Analysis Using R	Introducerende, middel
Owen (2010)	The R Guide	Introducerende, middel
H. Wickham (2014)	Advanced R	Middel, avanceret

H. Wickham (2014) kan findes gratis på <http://adv-r.hadley.nz/>.

## 4.1 Pakkeanbefaling: swirl

Foruden ovennævnte bøger kan pakken `swirl` også varmt anbefales. Når pakken er installeret (med `install.packages()`), kan den hentes med:

```
library("swirl")
```

Det smarte er, at du herfra blot skal bruge kommandoer og gennem outputtet lære at bruge R. For at komme igang skriver du:

```
swirl()
```

Det er muligt at hente flere forskellige kurser, der kan indlæses i `swirl`. Disse kan findes hos [Swirl Course Network](#).

# Bilag A

## Funktioner

R giver rig mulighed for at programmere egne funktioner. Det er muligt at lave det meste af det man skal i R uden at kende til funktioner, men når man eksempelvis skal lave mere specifikke arbejdsopgaver eller opgaver der skal gentages mange gange med enkelte modifikationer, er det ideelt at kunne lave sine egne funktioner. Ligeledes er det godt at kende til logikken bag funktioner, hvis man skal evaluere andres kode, hvor man ofte vil se funktioner.

Det meste af det man gør i R bruger eksisterende funktioner. En funktion anvendes på et objekt, der angives i en parentes, altså `funktion(objekt)`. Vi kan eventuelt se hvordan funktionen `sd()` fungerer ved blot at skrive `sd`:

```
sd

function (x, na.rm = FALSE)
sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
          na.rm = na.rm))
<bytecode: 0x7fcb4628f8d8>
<environment: namespace:stats>
```

Her kan vi se, at funktionen tager objektet, betegnet med `x`, og undersøger om det er en numerisk vektor, og såfremt den er det, tager funktionen kvadratroden af variansen i objektet. Der er i udgangspunktet således ingen grund til, at vi har funktionen `sd()`, da vi blot kan bruge:

```
sqrt(var(1:5))
```

```
[1] 1.581139
```

Det er dog blot nemmere at skrive (og læse), når man bruger funktionen `sd()`

```
sd(1:5)
```

```
[1] 1.581139
```

Funktionen tager dermed et objekt, i ovenstående eksempel et objekt med tallene i `c(1, 2, 3, 4, 5)`, og applicerer en kommando til objektet. Der er meget få begrænsninger på, hvad der kan laves i funktioner, og vi kan begynde med at lave vores helt egen meget simple funktion. Vi laver i nedenstående en funktion med navnet `plus2`, der tager et objekt og adderer 2:

```
plus2 <- function(x) x+2
```

Funktionen vil således tage vores nummer (`x`) og addere 2:

```
plus2(8)
```

```
[1] 10
```

I dette eksempel er der ét input, altså `x`, men det er muligt at lave funktioner, der kræver flere input. I nedenstående eksempel laver vi en ny funktion, der kræver to input, og med det første tal adderer 2 (som i ovenstående eksempel), og tager produktet af det andet input.

```
multif <- function(x, y) {  
  x <- x + 2  
  y <- y^2  
  c(x, y)  
}
```

I ovenstående angiver vi, at vi gerne vil have tallene ud i en vektor. I nedenstående eksempel bruger vi tallene 3 og 4, så funktionen returnerer  $3 + 2$  og  $4^2$ :

```
multif(3,4)
```

```
[1] 5 16
```

Dette *output* kan vi gemme i et nyt objekt, eller vi kan - som i nedenstående eksempel - lave andre operationer med det:

```
2 * multif(3,4)
```

```
[1] 10 32
```

## A.1 Loops

*Loops* er gode at anvende i en funktion, hvis man gentage en bestemt procedure for en række af tal. Hvis vi eksempelvis har tallene fra 1 til 6, og vi skal have det dobbelte af hvert af disse tal, kan vi lave en funktion, der for hvert tal i 1:6 tager tallet og multiplicerer det med 2 - og så viser tallet med print:

```
for (i in 1:6){  
  print(i * 2)  
}
```

```
[1] 2  
[1] 4  
[1] 6  
[1] 8  
[1] 10  
[1] 12
```

Disse *for loops* kan nemt anvendes sammen med andre funktioner, eksempelvis funktionen `plus2()`, vi lavede tidligere:

```
for (i in 1:6){  
  print(i * 2 * plus2(3))  
}
```

```
[1] 10  
[1] 20  
[1] 30  
[1] 40  
[1] 50  
[1] 60
```



Der er forskellige typer af *loops* i R, der ofte bruges i relation til *for loops*, herunder *if*, *else* og *while*. Det er ikke alle, der er lige store tilhængere af *for loops*, især da de kan blive komplicerede at skrive og læse, samt - for meget komplicerede opgaver, kan tage en del tid at fuldføre. Derfor bruger en del vektoriserede funktioner såsom `sapply()` og funktioner i pakker som `dplyr`.

# Bilag B

## Genveje og udvalgte funktioner

### B.1 Funktioner

Funktion	Beskrivelse
<code>abs()</code>	Numerisk værdi
<code>cor()</code>	Korrelation
<code>cov()</code>	Kovarians
<code>length()</code>	Længde på objekt
<code>log()</code>	Logaritmen
<code>max()</code>	Maksimum
<code>mean()</code>	Gennemsnit
<code>median()</code>	Median
<code>min()</code>	Minimum
<code>prod()</code>	Krydsprodukt
<code>quantile()</code>	Fem kvantiler
<code>read.csv()</code>	Indlæs .csv fil
<code>round()</code>	Afrunding
<code>sd()</code>	Standardafvigelse
<code>sqrt()</code>	Kvadratrod
<code>str()</code>	Struktur
<code>subset()</code>	Subsæt
<code>sum()</code>	Summering
<code>summary()</code>	Sammenfatning
<code>table()</code>	Krydstabel

Funktion	Beskrivelse
<code>unique()</code>	Unikke værdier
<code>var()</code>	Varians
<code>write.csv()</code>	Lav .csv fil

## B.2 Genveje i RStudio

Funktion	Windows	Mac
Kør markeret kode	CTRL+R	CMD+ENTER
Lav <i>assignment</i> operator (<-)	ALT+-	Option+-
Lav <i>pipe</i> operator (%>%)	CTRL+SHIFT+M	CMD+SHIFT+M

# Bilag C

## Anbefalede pakker

Navn	Formål	Nyttigt link
<a href="#">dplyr</a>	Databehandling	<a href="#">GitHub</a>
<a href="#">ggplot2</a>	Figurer, generelt	<a href="#">cookbook-r.com/Graphs/</a>
<a href="#">interplot</a>	Figurer, interaktion	<a href="#">Vignette: interplot</a>
<a href="#">MatchIt</a>	Matching	<a href="#">Documentation: MatchIt</a>
<a href="#">rddtools</a>	Regressionsdiskontinuitet	
<a href="#">rdrubust</a>	Regressionsdiskontinuitet	<a href="#">RD Software Packages</a>
<a href="#">rio</a>	Import/eksport af data	<a href="#">GitHub</a>
<a href="#">sem</a>	IV regression	<a href="#">2SLS in R</a>
<a href="#">stargazer</a>	Tabeller, eksport	<a href="#">A Stargazer Cheatsheet</a>

# Bilag D

## Eksport af tabeller

*Bliver tilføjet senere.*

Der er flere måder at eksportere tabeller fra R til ens tekstbehandlingsprogram. Her vil det i en fremtidig udgave af bogen blive vist, hvordan man gør det med pakken *stargazer* (Hlavac, 2015), der bruges i flere af bogens kapitler.

# Referencer

- Anscombe, F. J. (1973). Graphs in statistical analysis. *The American Statistician*, 27(1), 17–21.
- Chan, C., Chan, G. C. H., & Leeper, T. J. (2016). *Rio: A swiss-army knife for data file i/o*.
- Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using r*. London: SAGE Publications.
- Fox, J., & Weisberg, S. (2011). *An R companion to applied regression* (Second). Thousand Oaks CA: Sage. Retrieved from <http://socserv.socsci.mcmaster.ca/jfox/Books/Companion>
- Healy, K., & Moody, J. (2014). Data visualization in sociology. *Annual Review of Sociology*, 40, 105–128.
- Hlavac, M. (2015). *Stargazer: Well-formatted regression and summary statistics tables*. Cambridge, USA: Harvard University. Retrieved from <http://CRAN.R-project.org/package=stargazer>
- Ho, D. E., Imai, K., King, G., & Stuart, E. A. (2007). Matching as nonparametric preprocessing for reducing model dependence in parametric causal inference. *Political Analysis*, 15(3), 199–236.
- Ho, D. E., Imai, K., King, G., & Stuart, E. A. (2011). MatchIt: Nonparametric preprocessing for parametric causal inference. *Journal of Statistical Software*, 42(8), 1–28. Retrieved from <http://www.jstatsoft.org/v42/i08/>
- Kastellec, J. P., & Leoni, E. L. (2007). Using graphs instead of tables in political science. *Perspectives on Politics*, 5(4), 755–771.
- Monogan III, J. E. (2015). *Political analysis using r*. New York: Springer.
- Owen, W. J. (2010). *The r guide*. Retrieved from <http://CRAN.R-project.org/doc/contrib/>
- R Core Team. (2015). *Foreign: Read data stored by minitab, s, sas, spss, stata, systat, weka, dBase*,

- 
- . Retrieved from <http://CRAN.R-project.org/package=foreign>
- Schwabish, J. A. (2014). An economist's guide to visualizing data. *Journal of Economic Perspectives*, 28(1), 209–234.
- Sekhon, J. S. (2009). Opiates for the matches: Matching methods for causal inference. *Annual Review of Political Science*, 12, 487–508.
- Tufte, E. R. (1983). *The visual display of quantitative information*. Graphics Press.
- Wickham, H. (2009). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York.  
Retrieved from <http://ggplot2.org>
- Wickham, H. (2014). *Advanced r*. Chapman & Hall/CRC The R Series.
- Wickham, H., & Francois, R. (2015). *Readr: Read tabular data*. Retrieved from <http://CRAN.R-project.org/package=readr>
- Wickham, H., & Francois, R. (2016). *Dplyr: A grammar of data manipulation*. Retrieved from <http://CRAN.R-project.org/package=dplyr>