

Introduktion til R

Erik Gahner Larsen

04/09/2016

Indhold

Kapitel 1: Introduktion	1
1.1 Hvorfor R? Fordele og ulemper ved R	1
1.2 Installation af R og RStudio	3
Kapitel 2: Det grundlæggende	5
2.1 Objekter og funktioner	5
2.2 Rekodninger	15
2.3 Import og eksport af datasæt	16
2.4 Installation af pakker	17
2.5 Objekter i hukommelsen	18
Kapitel 3: Visualisering	19
Konklusion	20
Bilag A: Genveje og funktioner	21
A.1 Funktioner	21
A.2 Genveje i RStudio	21
Bilag B: Anbefalede pakker	22
Referencer	23

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 dog ingenlunde nemt at lære, men det har væsentlige styrker, der gør, at det er bedre end alternativerne (SPSS, Stata, SAS m.v.).

Bogens fokus er på at give en introduktion til tre vigtige stadier i statistiske analyser. For det første skal vi bearbejde de data, vi skal bruge. 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 betydeligt 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.

I dette kapitel gives en introduktion til R. Dette sker ved først at gennemgå 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 visualiserer sine data, gennemfører lineære regressionsanalyser samt matching.

Alt materiale, der anvendes i nærværende bog (herunder datasæt og kode), kan hentes på GitHub: <https://github.com/erikgahner/Rguide>. Hvis du finder fejl og mangler i bogen, må du meget gerne oprette et issue på GitHub eller sende en mail til egl@sam.sdu.dk, og det samme gør sig selvfølgelig gældende, hvis du har en idé eller et forslag til, hvad der vil kunne styrke bogen.

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 statistiske programmer på markedet, der kan gennemføre statistiske analyser. Herunder også programmer der er nemmere at lære end R. Skal man udelukkende bruge et program til at lave lagkagedia-

grammer, 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 Ferrari (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. Hvad angår præsentationen af analyser, giver R betydeligt bedre muligheder for at lave pæne figurer end de andre programmer på markedet.

For det tredje 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 fjerde 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 tilfældet, hvis man først lærer eksempelvis SPSS, hvor der kan være mange dårlige vaner, man først skal vænne sig af med.

For det femte 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, 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 på at dokumentere processen heraf.

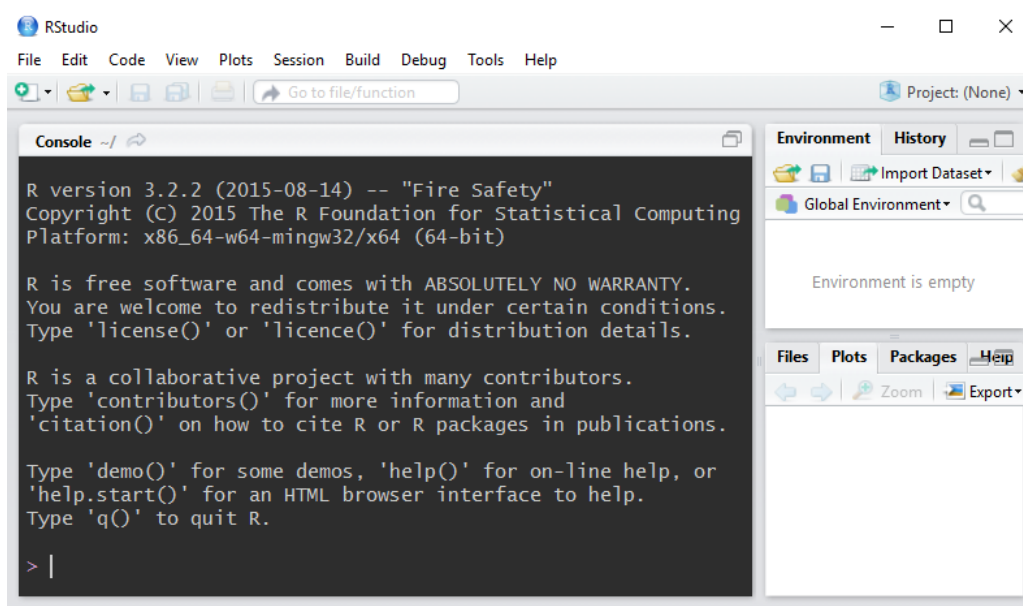
På baggrund af ovenstående liste (der ikke er udtømmende), burde det være åbenlyst for de fleste, at R er værd at bruge tid på. Når dette er sagt, er der ikke desto mindre nogle ulemper forbundet med R. For det første er det, som beskrevet, svært at lære at bruge R. For det andet er der ikke en officiel supportlinje, man kan ringe til, hvis man støder ind i problemer. Dette ændrer 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. Her er de step, du kan følge:

1. Gå ind på 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/.
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 skærmbilledet 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 nedenfor), er der tre vinduer. For det første er der konsollen, hvor du kan skrive kommandoer. For det andet er der dit miljø (*environment*), hvor det bliver synligt, hvilke ting du arbejder med. For det tredje har du et *output* vindue, hvor blandt andet grafer, hjælpedokumenter og lignende vil blive præsenteret.

Kapitel 2

Det grundlæggende

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 heraf¹. 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 efter noget kode².

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+R` (Mac) (Se Appendix for flere genveje). Forsøg at indtaste nedenstående kode, marker det hele og kør det.

¹Dette svarer til at skrive `display 2+2` i Stata.

²Dette svarer til `*` og `//` i Stata.

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

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, modsat eksempelvis Stata, at vi kan have flere datasæt åbne i hukommelsen på samme tid gemt som hvert deres objekt. 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 forsøge med 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 værdier³. Forsøg gerne 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 ændringer⁴.

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 når vi skal lave rekodninger 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
x == 3      # "==" betyder "lig med"
x != 2      # "!=" betyder "ikke lig med"
x < 1
x > 1
x <= 2
x >= 2.01
```

³Helt præcist 3, 6, 9 og 6.

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

Hvis x er 2, vil værdierne være hhv. TRUE, FALSE, FALSE, FALSE, TRUE, TRUE og FALSE. Igen, 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.

```
(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.

```
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 vektorer⁵. Funktionen $c()$ står for *concatenate* eller *combine*⁶. 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 .

⁵I 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 desuden et skjult $c()$.

⁶ $c()$ opretter således en vektor med alle elementer i parentes. Da en vektor kun kan indeholde én type data, og ikke eksempelvis både numre og karakterer, 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.

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

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

Denne vektor kan vi behandle efter forgodtbefindende. Først 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
```

På samme måde kan vi også få alle værdier i vektoren med undtagelse af et bestemt tal, blot ved at tilføje et `'-'`-tegn til parentesen. I eksemplet her får vi vektoren uden tal nummer to. 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 eksempelvis gennemsnittet af værdierne.

<code>length(x)</code>	<i># antallet af numre i vektoren</i>
<code>min(x)</code>	<i># minimumsværdien</i>
<code>max(x)</code>	<i># maksimumsværdien</i>
<code>median(x)</code>	<i># medianen</i>
<code>sum(x)</code>	<i># summen</i>
<code>mean(x)</code>	<i># gennemsnittet</i>
<code>var(x)</code>	<i># variansen</i>
<code>sd(x)</code>	<i># standardafvigelsen</i>

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 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.

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

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

Som det kan ses, er der nu fem værdier i vores vektor. Værdien 5, der blev tilføjet, har den sidste placering i vektoren, da vi placerede den til sidst, da vi lavede et nyt objekt. Vi kan så 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). 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. 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.

```
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.

Foruden tal kan vi også arbejde med tekst. Tekst i R adskiller sig fra tal ved, at tekst pakkes ind i citationstegn⁷. 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()`. Hvis vi bruger denne funktion 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”*⁸.

```
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)
```

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.

⁷Alternativt kan man også bruge `'` i stedet for `“`.

⁸De forskellige klasser en vektor kan have er hhv. `character` (tekst), `numeric` (numeriske tal), `integer` (hele tal), `factor` (kategorier) og `logical` (logisk).

```
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"
```

For disse partier 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. 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 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. 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)
pol
```

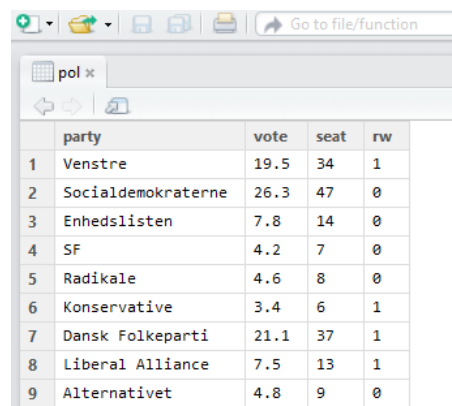
	party	vote	seat	rw
1	Venstre	19.5	34	1
2	Socialdemokraterne	26.3	47	0
3	Enhedslisten	7.8	14	0
4	SF	4.2	7	0
5	Radikale	4.6	8	0
6	Konservative	3.4	6	1
7	Dansk Folkeparti	21.1	37	1
8	Liberal Alliance	7.5	13	1
9	Alternativet	4.8	9	0

Som det kan ses har vi 9 observationer⁹. Dette er overkommeligt at vise, men når man arbejder med større datarammer, oftest med flere tusinde 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 `head()` kan man således få vist de seks første observationer i ens dataramme (altså de seks første 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)
head(pol, 3)
tail(pol)
```

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
```

⁹Brug evt. `class()` til at vise, at `pol` er en dataramme.

Modsat en vektor 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 tilfælde 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, 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 den 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 `[]`.

```
pol$party[1]
```

```
[1] Venstre
```

```
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 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
```

2.2 Rekodninger

Der er flere måder hvorpå man i R kan rekode variable og dermed danne nye variable i sin dataramme. Kort fortalt kan man med udgangspunkt i det der er gennemgået i ovenstående, lave nye variable. Her vil der blive givet et par eksempler på, hvordan vi kan lave en ny variabel baseret på værdierne på en anden variabel. Mere specifikt vil vi gerne have lavet en binær variabel, der antager værdien 1, hvis et parti har fået mere end 20 procent af stemmerne, og værdien 0 hvis ikke.

Vi vil gerne have en variabel med navnet `big`. For at gøre dette 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 det pågældende 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. Hvis vi eksempelvis manuelt ville give Socialdemokraterne og Dansk Folkeparti værdien 1, uden at bruge information omkring deres stemmetal, kunne dette også gøres. Ligeledes kunne vi have sagt, at de værdier der var NA efter at værdien 1 var blevet tildelt til de store partier, skulle have værdien 0. Nedenstående kode giver nogle eksempler herpå.

```
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
```

Der er også udviklet pakker, der har specifikke funktioner til at rekode variable i datarammer. Et eksempel herpå er at finde i pakken *car* (Fox & Weisberg, 2011), der har funktionen `recode()`¹⁰. En anden pakke, der er yderst effektiv er *dplyr* (Hadley Wickham & Francois, 2016), der gør det nemt at bearbejde datarammer. Funktionerne heri inkluderer `select()`, `filter()`, `arrange()`, `rename()` og `mutate()`, der gør det nemt at lave nye variable, vælge bestemte variable ud m.v.¹¹.

2.3 Import og eksport af datasæt

Det meste af det data vi kommer til at arbejde med i R, er noget vi importerer. Det er heldigvis nemt at importere forskellige typer af data (også fra Stata og SPSS), men det meste format er og bliver 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 således 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()`.

¹⁰ Installation af pakker gennemgås i sektion 2.4.

¹¹ For en god introduktion til *dplyr*, se [dette kapitel](#) i *Programming for Data Science*.

```
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")
```

Når vi har gemt filen kan vi først undersøge manuelt, om filen er gemt i vores *working directory*¹². 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 næste sektion).

2.4 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 køre 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 fro at gøre det igen. I dette eksempel vil vi bruge funktionen til at installere pakken `ggplot2`, som vi vil bruge til at lave figurer. Bemærk desuden citationstegnene, der 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.

¹²I 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().

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 B, hvor en række anbefalelsesværdige pakker nævnes.

2.5 Objekter i hukommelsen

Før vi slutter af er der et par enkelte nyttige funktioner, du bør kende til. Den første er ls(), der viser, hvilke objekter vi har i hukommelsen. 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(). 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(0)
```

Kapitel 3

Visualisering

Der er mange måder hvorpå man kan præsentere data. I dette kapitel vil der blive givet en introduktion til, hvordan man kan visualisere sine data med R. 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 eksempelvis tabeller (Healy & Moody, 2014, Kastellec & Leoni (2007), Schwabish (2014)).

Der er gode grunde til at visualisere sine data. 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)

Konklusion

Kommer senere.

Der er en lang række af statistikbøger, der 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

Bilag A

Genveje og funktioner

A.1 Funktioner

Funktion	Beskrivelse
<code>abs()</code>	Numerisk værdi
<code>cor()</code>	Korrelation

A.2 Genveje i RStudio

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

Bilag B

Anbefalede pakker

Kommer.

Referencer

- 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.
- 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.
- 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>