

# Analyse og visualisering af biologiske datasæt - 2022

Sarah Rennie

Last updated: 2022-05-29



# Contents

<b>1 Grundlæggende R</b>	<b>7</b>
1.1 Inledning til kapitel . . . . .	7
1.2 RStudio . . . . .	8
1.3 Working directory . . . . .	8
1.4 R pakker . . . . .	9
1.5 Hvor kommer vores data fra? . . . . .	10
1.6 Beregninger i R . . . . .	11
1.7 Dataframes . . . . .	13
1.8 Descriptive statistics . . . . .	15
1.9 Statistiske tester . . . . .	17
1.10 Problemstillinger . . . . .	29
<b>2 Introduktion til R Markdown</b>	<b>37</b>
2.1 Hvad er R Markdown? . . . . .	37
2.2 Installere R Markdown . . . . .	37
2.3 Videodemonstrationer . . . . .	38
2.4 Oprette et nyt dokument i R Markdown . . . . .	38
2.5 Skrive baseret tekst . . . . .	39
2.6 Knitte kode . . . . .	42
2.7 Kode chunks . . . . .	42
2.8 R beregninger indenfor teksten i dokument ('inline code') . . . . .	44
2.9 Working directory . . . . .	44
2.10 Matematik . . . . .	45
2.11 Problemstillinger . . . . .	45
2.12 Færdig for i dag og næste gang . . . . .	46
2.13 Ekstra links . . . . .	46
<b>3 Visualisering - ggplot2 dag 1</b>	<b>49</b>
3.1 Inledning og videoer . . . . .	49
3.2 Transition fra base R til ggplot2 . . . . .	51
3.3 Vores første ggplot . . . . .	52
3.4 Lidt om ggplot2 . . . . .	55
3.5 Specificere etiketter og titel . . . . .	57
3.6 Ændre farver . . . . .	59

3.7	Ændre tema . . . . .	60
3.8	Forskellige geoms . . . . .	61
3.9	Troubleshooting . . . . .	76
3.10	Problemstillinger . . . . .	77
3.11	Næste gang . . . . .	85
<b>4</b>	<b>Visualisering - ggplot2 dag 2</b>	<b>87</b>
4.1	Indledning og videoer . . . . .	87
4.2	Koordinat systemer . . . . .	88
4.3	Mere om farver og punkt former . . . . .	95
4.4	Annotations ( <code>geom_text</code> ) . . . . .	102
4.5	Adskille plots med facets ( <code>facet_grid/facet_wrap</code> ) . . . . .	108
4.6	Gemme dit plot . . . . .	115
4.7	Problemstillinger . . . . .	115
4.8	Ekstra links . . . . .	123
<b>5</b>	<b>Bearbejdning dag 1</b>	<b>125</b>
5.1	Hvad er Tidyverse? . . . . .	125
5.2	Video ressourcer . . . . .	126
5.3	Oversigt over pakker . . . . .	126
5.4	Principper med ‘tidy data’ . . . . .	127
5.5	Lidt om <code>tibbles</code> . . . . .	128
5.6	Transition fra base til tidyverse . . . . .	130
5.7	Bearbejdning af data: <code>dplyr</code> . . . . .	134
5.8	Visualisering: bruge som input i ggplot2 . . . . .	143
5.9	Misc funktioner som er nyttige at vide . . . . .	145
5.10	Problemstillinger . . . . .	146
5.11	Kommentarer . . . . .	150
<b>6</b>	<b>Bearbejdning dag 2</b>	<b>153</b>
6.1	Indledning og læringsmålene . . . . .	153
6.2	<code>group_by()</code> med <code>summarise()</code> i dplyr-pakken . . . . .	154
6.3	<code>pivot_longer()</code> / <code>pivot_wider()</code> med Tidyr-pakken . . . . .	161
6.4	Eksempel: Titanic summary statistics . . . . .	166
6.5	<code>left_join()</code> : forbinde dataframes . . . . .	170
6.6	Problemstillinger . . . . .	174
6.7	Ekstra links . . . . .	181
<b>7</b>	<b>Functional programming med purrr-pakken</b>	<b>183</b>
7.1	Inledning og læringsmålene . . . . .	183
7.2	Iterativ processer med <code>map()</code> funktioner . . . . .	184
7.3	Custom funktioner . . . . .	188
7.4	Nesting <code>nest()</code> . . . . .	192
7.5	Andre brugbar purrr . . . . .	197
7.6	Problemstillinger . . . . .	200
7.7	Ekstra notater og næste gang . . . . .	203

CONTENTS	5
----------	---

<b>8 Visualisering af trends</b>	<b>205</b>
8.1 Indledning og læringsmålene . . . . .	205
8.2 <code>nest()</code> og <code>map()</code> : eksempel med korrelation . . . . .	207
8.3 Lineær regression - visualisering . . . . .	212
8.4 Plot linear regresion estimates . . . . .	218
8.5 Multiple regression and model comparison . . . . .	222
8.6 Problemstillinger . . . . .	227
8.7 Ekstra . . . . .	230
<b>9 Clustering</b>	<b>231</b>
9.1 Indledning og læringsmålene . . . . .	231
9.2 Method 1: K-means clustering . . . . .	232
9.3 Kmeans: hvor mange clusters? . . . . .	239
9.4 Method 2: Hierarchical clustering . . . . .	248
9.5 Problemstillinger . . . . .	252
<b>10 Principal component analysis (PCA)</b>	<b>255</b>
10.1 Indledning og læringsmålene . . . . .	255
10.2 Hvad er principal component analysis (PCA)? . . . . .	256
10.3 Fit PCA to data in R . . . . .	261
10.4 Integrere PCA resultater med broom-pakke . . . . .	262
10.5 Problemstillinger . . . . .	269
10.6 Ekstra læsning . . . . .	272
<b>11 Emner fra eksperimental design</b>	<b>273</b>
11.1 Inledning og læringsmålene . . . . .	273
11.2 Grundlæggende principper i eksperimental design . . . . .	274
11.3 Case studies: Simpson's paradox . . . . .	277
11.4 Case studies: Anscombe's quartet . . . . .	282
11.5 Undersøgelse af "batch-effects" . . . . .	285
11.6 Problemstillinger . . . . .	291
11.7 Yderligere læsning . . . . .	294
<b>12 Maskinslæring i tidyverse</b>	<b>295</b>
12.1 Indledning og læringsmålene . . . . .	295
12.2 Video ressourcer . . . . .	296
12.3 Regression models . . . . .	296
12.4 Classification models . . . . .	301
12.5 Problemstillinger . . . . .	312
12.6 Yderligere kommentarer og pakker . . . . .	315



# Chapter 1

## Grundlæggende R

“Det er ikke, fordi noget er svært, at vi ikke tør, det er, fordi vi ikke tør, at noget er svært” - Seneca

### 1.1 Inledning til kapitel

Her opsummerer jeg nogle grundlæggende R og statistik, der betragtes som forudsætninger i det nuværende kursus. Selvom vi i kurset skifter hurtigt over til den tidyverse-pakke løsning, som erstatter meget af funktionaliteten fra base-R, er det stadig vigtigt at have et grundlæggende kendskab til hvordan tingene fungerer i base-R - derfor hvis du har meget lidt erfaring med base-R anbefaler jeg, at du også bruger noget ekstra tid uddover den første mødegange til at komme op på niveauet.

For at bestå kurset er det ikke forventningen, at du kender til alle detaljer og teori bag de statistiske metoder, men at du kan anvende dem hensigtsmæssigt i praksis i R, samt fortolke resultaterne. Jeg giver masser af muligheder for at øve dig med at lave statistik hele vejen gennem kurset, og i selve eksamen stiller jeg ikke spørgsmål om metoder, der ikke bliver dækkede blandt de forskellige øvelser (herunder workshop opgaver). Jeg kommer også ind på lineær regression igen senere gennem forelæsningerne så vær ikke bekymret hvis du ikke har set det hele før.

Se gerne også “Quiz - grundlæggende” på Absalon for at tjekke din forståelse og udfylde eventuelle huller i din viden (OBS: Quizzen er tilgængelig lidt inden starten af kurset).

## 1.2 RStudio

Vi kommer fremadrettet til at være afhængig af RStudio til at lave blandt andet R Markdown dokumenter. Kendskab til R Markdown er emnet i vores næste lektion og jeg antager, at du ikke har benyttet det før.

Det allerførste du skulle gør, hvis du ikke har installeret RStudio på din computer, er at downloade det gratis på nettet:

<https://www.rstudio.com/products/rstudio/download/#download>

Følg venligst RStudios egne anvisninger til at få det installeret. Bemærk, at installering af RStudio er ikke den samme som at have R installeret på din computer - man skal installere dem begge to (man kan bruge R uden RStudio men ikke omvendt).

### 1.2.1 De forskellige vinduer i RStudio

Du kan læse følgende for at lære de fire forskellige vinduer i RStudio at kende:

<https://bookdown.org/ndphillips/YaRrr/the-four-rstudio-windows.html>

Her er et kort oversigt:

- Man skriver kode i **Source** (øverst til venstre)
- Man kører kode ved at tryk CMD+ENTER (eller WIN-KEY+ENTER)
- Koder køres ind i **Console** (som plejer at være nederst til venstre, selvom det er øverst til højere i billedet). Man kan også skrive koder direkte i Console, men det ikke anbefales generelt, når koden ikke bliver gemt.
- **Environment** - her kan man se blandt andet, alle objekter i Workspace.

## 1.3 Working directory

Når man arbejder på et projekt, er det ofte nyttigt at vide, den *working directory* som R arbejder fra - det er den mappe, hvor R forsøger at åbne eller gemme filer fra, medmindre man angiver et andet sted.

```
getwd() #se nuværende working directory
list.dirs(path = ".", recursive = FALSE) #se mappe indenfor working directory
setwd("~/Documents/") #sætte en ny working directory (C:/Users/myname/Documents hvis man er på Windows)
```

Hvis man bruger Windows, husk at man kan skrive en path på følgende måde:

```
#notrun
setwd("C:/Users/myname/Documents") #enten med /
setwd("C:\\\\Users\\\\myname\\\\Documents") #eller med \\
```

**OBS:** jeg bruger Mac, så hvis der er et vigtigt ting at man skal huske hvis man bruger en Windows computer, kan jeg også tilføje det her. Bemærk dog, at de

allerfleste ting ved R programmering og tidyverse er ens uanset om man bruger Windows eller Mac.

## 1.4 R pakker

R pakker er simpelthen en samling af funktioner (eller datasæt i nogle tilfælde), der udvider hvad der er tilgængelige i base-R (den R man få, uden at indlæse en pakke). I R er der mange tusind R pakker (op mod 100,000), der er tilgængelige på **CRAN** (<https://cran.r-project.org/>). Indenfor det biologiske fag er der også mange flere pakker på **Bioconductor** (<https://www.bioconductor.org/>), og i nogle tilfælde kan R pakker også installeres direkte fra **Github**.

I dette kursus arbejder vi rigtig meget med en pakke der hedder **tidyverse**. **tidyverse** er faktisk en samling af otte R pakker, som indlæses på en gang. Inden du indlæser pakken, skal du først sikre dig, at pakken er installeret på systemet ved følgende kommando:

```
install.packages("tidyverse")
```

Alle pakker på **CRAN** er installeret på samme måde. Når du faktisk gerne vil bruge en R pakke, skal du først indlæse den ved at bruge `library()`:

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --
## v ggplot2 3.3.5     v purrr  0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyverse 1.2.0    v stringr 1.4.0
## v readr   2.1.2     vforcats 0.5.1

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
```

Vi kommer til at arbejde med **tidyverse** pakker fra kapitel tre (vi starter med `ggplot2` og så nogle af de andre pakke fra `tidyverse` fra kapitel fire), **så det er en god idé at har `tidyverse` installeret allerede nu**, når det nogle gange kan tage lidt tid til at installere eller opdatere de mange andre mulige pakker, der `tidyverse` er afhængig af.

Vær opmærksom på, at der nogle gange opstår konflikter når det samme funktionnavn findes i flere pakker - for eksempel, funktionen `filter()` findes indenfor to forskellige pakker, nemlig `dplyr` og `stats`. Når du skriver `filter()` så ved R ikke, hvilke pakker du mener. I dette tilfælde kan du være gennemskueligt overfor den pakke, du gerne vil bruge ved at skrive `dplyr::filter()` eller `stats::filter()` i stedet for bare `filter()`.

Som sidste kommentar, er det god praksis at indlæse alle pakker, der du benytter

sig af, på toppen af din script, så at du hurtigt kan få overblik over, hvilke pakker, der skal indlæses til at få dine koder til at fungere.

## 1.5 Hvor kommer vores data fra?

De forskellige datasæt, vi kommer til at arbejde med i kurset stammer fra mange forskellige steder.

### 1.5.1 Indbyggede datasæt

I R er der mange indbygget datasæt som er meget brugbart for at vise koncepter, hvilket gøre dem især populært i undervisningsmateriale. Indbyggede datasæt er ofte tilgængeligt indenfor mange pakker, men `library(datasets)` er den mest brugt (der er også mange indenfor `library(ggplot2)`). For eksempel, for at indlæse datasættet, der hedder ‘iris’, kan man bruge `data()`:

```
library(datasets)
data(iris)
```

Så er en *dataframe*, der hedder ‘iris’ tilgængelige som en *objekt* i *workspacen* - se den “Environment” fane på højere side i RStudio, eller indtaste `ls()`, så bør du kunne se et objekt med navnet ‘iris’. Man kan kun arbejde med objekter som er en del af workspacen.

### 1.5.2 Importering af data fra .txt fil

Det er meget hyppigt, at man har sin data i formen af en .txt fil eller .xlsx fil på sin computer. Den nemmeste måde at få åbnet en .txt fil er ved at bruge `read.table()`, som i nedenstående:

```
data <- read.table("mydata.txt") #indlæse data filen mydata.txt som er i working direc
head(data)
```

Hvis datasættet har kolonner navne, der er skrevet ind i filen, så skal man huske at bruge `header=T` for at undgå, at den første række i datasættet bliver disse tekster i stedet for virkelige observationer.

```
data <- read.table("mydata.txt",header=T) #indlæse data filen mydata.txt som er i work
head(data)
```

### 1.5.3 Importering af data fra Excel

Der findes også en hjælpsom pakke, som hedder `readxl`, der kan indlæse Excel-ark direkte ind i R:

```
library(readxl)
data <- read_excel("data.xlsx")
data
```

### 1.5.4 Kaggle

Hvis du gerne vil øve dig med statistiske analyser (udover nuværende kursus), er Kaggle en fantastisk ressource til at finde forskellige datasæt. I rigtige mange tilfælde kan man også finde analyser som andre har lavet i R (også Python), hvilket kan inspirere jeres egen læring.

Link hvis interesseret: <https://www.kaggle.com/>

## 1.6 Beregninger i R

Her er nogle helt grundlæggende koncepter når man arbejder med R. Du må selvfolgtlig gerne springe sektionen over, hvis du allerede har meget erfaring med base R, men det kan være værd at tjekke, om der noget ting, der lige skal gennemgås. En god tilgang er bare at arbejde gennem problemstillingerne nedenfor, og bruger følgende notater som en reference.

### 1.6.1 Vectorer

I R laver man en vector med `c()`, hvor man adskiller de forskellige elementer med en komma, som i nedenstående eksempel:

```
a <- c(1,2,3,4,5) #sæt objektet 'a' til at være en vector af tal
a
```

```
## [1] 1 2 3 4 5
```

Man er ikke begrænset til tal:

```
c <- c("cat", "mouse", "horse", "sheep", "dog")
c

## [1] "cat"    "mouse"   "horse"   "sheep"   "dog"
```

### 1.6.2 datatyper

Nar vi kommer til at arbejde med visualiseringer og data bearbejdning er det vigtigt at have styr på datatyper i datasættet. For eksempel har vectoren `c` ovenpå typen `character` (forkortet `chr`) og ikke `numeric` (forkortet `num`):

```
is.numeric(c)
## [1] FALSE
is.character(c)
## [1] TRUE
```

Her er en liste overfor nogle af de vigtigste datatyper:

Datatype	Navn	Beskrivelse
<code>int</code>	<code>integer</code>	kun hel tal <code>c(-1,0,1,2,3)</code>
<code>lgl</code>	<code>logical</code>	<code>TRUE</code> <code>TRUE</code> <code>FALSE</code> <code>TRUE</code> <code>FALSE</code>
<code>chr</code>	<code>character</code>	<code>c("Bob","Sally","Brian",...)</code>
<code>fct</code>	<code>factor</code>	bestemte niveauer e.g. <code>Species</code> : <code>c("setosa","versicolor")</code>
<code>dbl</code>	<code>double</code>	Tal fk. <code>c(4.3902, 3.12, 4.5)</code>
<code>lst</code>	<code>list</code>	blande forskellige data typer og specificere elementer med <code>[[i]]</code> <code>[[1]]</code> <code>[1]</code> <code>c("red","blue")</code> <code>[[2]]</code> <code>[1]</code> <code>TRUE</code> <code>[[3]]</code> <code>[1]</code> <code>c(3,2.3,1.459)</code>

En datatype, der bør få særlig opmærksomhed er `fct` (factor). I følgende vector `tea_coffee` har vi tekst, men blandt de fem elementer er der kun to bestemte niveauer (nemlig “tea” og “coffee”).

```
tea_coffee <- c("tea", "tea", "coffee", "coffee", "tea")
is.factor(tea_coffee)
## [1] FALSE
tea_coffee
## [1] "tea"     "tea"     "coffee"  "coffee"  "tea"
```

Vi vil derfor gerne fortælle R, at `tea_coffee` er ikke bare nogle tilfældig tekst men at der er en struktur med, så vi bruger funktionen `as.factor` for at lave den om til datatypen `fct`.

```
tea_coffee <- as.factor(tea_coffee)
is.factor(tea_coffee)
## [1] TRUE
tea_coffee
## [1] tea     tea     coffee  coffee  tea
## Levels: coffee tea
```

Den ‘ekstra’ oplysninger man har ved at sige, at en variabel betragtes som factor bliver vigtigt når man arbejder med visualiseringer - for eksempel, hvis vi gerne vil lave et barplot hvor man gerne vil adskille sjælerne efter de to niveauer “tea” og “coffee” (visualiseringer er emnet fra kapitel 3).

## 1.7 Dataframes

<http://www.r-tutor.com/r-introduction/data-frame>

Mange af de ting, som vi laver i R tager udgangspunkten i dataframes (eller datarammer).

```
mydf <- data.frame("personID"=1:5, "height"=c(140,187,154,132,165), "age"=c(34,31,25,43,29))
mydf
```

```
##   personID height age
## 1         1     140  34
## 2         2     187  31
## 3         3     154  25
## 4         4     132  43
## 5         5     165  29
```

Man kan få adgang til variabler i en dataframe ved at bruge det dollar tegn \$. For eksempel giver følgende variablen personID fra dataframen mydf:

```
mydf$personID
```

```
## [1] 1 2 3 4 5
```

Husk, at vores dataframe, ligesom et matrix (i R: `matrix()`) har to dimensioner - række og kolonner. Forskellen mellem en matrix og en dataramme er, at datarammer kan indeholde mange forskellige data typer (herunder numeriske, faktorer, karakterer osv.), men matrix indeholder kun numeriske data. For eksempel i tilfældet af ovenstående dataframen er alle variabler numeriske, men vi kan godt tilføje en variabel som er ikke-numeriske:

```
mydf$colour <- c("red", "blue", "green", "orange", "purple") #make new variable which is non-numeric
mydf
```

```
##   personID height age colour
## 1         1     140  34    red
## 2         2     187  31   blue
## 3         3     154  25  green
## 4         4     132  43 orange
## 5         5     165  29 purple
```

Nu er mydf er en dataframe, der blander forskellige datatyper, men følgende er en matrix

```
matrix(c(1, 2, 3, 4, 5, 6),
      nrow=3,
      ncol=2)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
```

```
## [3,]    3    6
```

og kan kun indeholde numeriske data, som kan bruges til at lave matematik operationer (matrix multiplikation osv.). I dette kursus beskæftiger os primært med dataframes (som bliver kaldt for tibbles i **tidyverse**).

### 1.7.1 Delmægder af dataframes

Selvom vi kommer til at redefinere hvordan man laver delmængde når vi kommer til at arbejde med pakken **tidyverse**, er det alligevel vigtigt at forstå, hvordan man laver en delmængde i base-R, og det er et område, der ofte skaber forvirring blandt de uerfarne.

Når man vil gerne har en bestemt delmængde af en vector, bruger man firkantet paranteser [ ]. Følgende kode giver mig de første to værdier fra vectoren a:

```
a[1:2]
```

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

Bemærk, at mens vectorer har kun en dimension, **har dataframes to dimensioner**. Når man skal lave en delmængde af en dataframe, skal man derfor fortælle R, hvilke række og hvilke kolonner skal være med.

```
mydf[række indeks, kolonner indeks] #not run
```

For eksempel, hvis vi gerne vil have den første to observationer med, samt kun den anden variabel, skriver man følgende:

```
mydf[1:2, 2] #first two rows (observations), second column (variable) only
```

```
## [1] 140 187
```

Hvis vi vil beholde den første to observationer og samtlige variabler, kan den anden plads være tom:

```
mydf[1:2, ] #first two rows, all columns
```

```
##   personID height age colour
## 1         1    140  34    red
## 2         2    187  31   blue
```

Jeg kan også angive et variabelnavn direkte:

```
mydf[1:2, "height"]
```

```
## [1] 140 187
```

Man kan kigge på en subset af rækkerne i de data ved at

```
mydf[mydf$height>=165,] #alle rækker i datarammen med height = 165 eller over
```

```
##   personID height age colour
```

```
## 2      2     187 31   blue
## 5      5     165 29 purple
```

Her er en tabel af comparitiver, og jeg gengiver samme tabel når I kommer til at lave delmængde i **tidyverse**:

comparativ	beskrivelse
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to
&	and
%in%	in
	or
!	not

Jeg mener, at `%in%` er særlig brugbart og er værd at lære:

```
mydf[mydf$personID %in% c(1,3,5),] #alle personer med personID 1,3 eller 5

##   personID height age colour
## 1         1    140  34   red
## 3         3    154  25  green
## 5         5    165  29 purple
```

Her er et eksempel på, hvordan man bruger udråbstegnet: personer med personID, der ikke er 1,3 eller 5:

```
mydf[!(mydf$personID %in% c(1,3,5)),] #alle personer med personID 2 eller 4

##   personID height age colour
## 2         2    187  31   blue
## 4         4    132  43 orange
```

## 1.8 Descriptive statistics

### 1.8.1 Simulere data fra den normale fordeling

Hvis du har bruge for at vide mere om den normale fordeling: <http://www.r-tutor.com/elementary-statistics/probability-distributions/normal-distribution>

Man kan nemt lave sin egne ‘fake’ data ved at simulere det fra en fordeling, der vil typiske være den normale fordeling, idet den normale fordeling opstår mest hyppigt i den virkelige verden (husk den klassiske klokke-form). I R kan man bruge funktionen `rnorm` til at simulere data - først angiver man, hvor mange

observationer man vil have, og dernæst den mean og standard deviation (sd), som er de to nødvendige parametre for at beskrive en normal fordeling

```
x <- rnorm(25,mean=0,sd=1) #standard normal distribution
x #så har vi 25 værdier fra en normal distribution med mean=0 og standard deviation=1.

## [1] 0.11061220 0.26194036 -0.74804948 0.91450709 0.34908502 -1.77144593
## [7] -0.94136236 -0.46445096 0.02651141 -1.00082963 -2.00938552 2.04089381
## [13] 0.08820498 0.24437671 1.31476257 0.07886485 -0.03996859 0.30734122
## [19] 1.28519058 1.43438077 -0.12288067 -0.85668870 1.28325074 0.86359650
## [25] 0.77279565
```

I stedet for at kigge på alle værdier på én gang, vil vi måske hellere kigge kun på de første (eller sidste) værdier:

```
head(x) #første 6
## [1] 0.1106122 0.2619404 -0.7480495 0.9145071 0.3490850 -1.7714459
tail(x) #sidste 6
## [1] 1.4343808 -0.1228807 -0.8566887 1.2832507 0.8635965 0.7727957
x[1] #første værdi
## [1] 0.1106122
x[length(x)] #sidste data point
## [1] 0.7727957
```

Bemærk, at til forskellen af Python og mange andre programmering sprog, R bruger 1-baserende indicer - det betyder, at den første værdi er `x[1]` og ikke `x[0]` som i Python.

### 1.8.2 Measures of central tendency

function	Description
<code>mean()</code>	mean $\bar{x}_i = \frac{1}{n} \sum_{i=1}^n x_i$
<code>median()</code>	median value
<code>max()</code>	maximum value
<code>min()</code>	minimum value
<code>var()</code>	variance $s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x}_i)^2$
<code>sd()</code>	standard deviation $s$

Lad os afprøve dem på vores simulerede data:

```
my_mean <- mean(x)
my_median <- median(x)
my_max <- max(x)
my_min <- min(x)
my_var <- var(x)
my_sd <- sd(x)
```

```
c(my_mean,my_median,my_max,my_min,my_var,my_sd) #print results

## [1] 0.1368501 0.1106122 2.0408938 -2.0093855 0.9967437 0.9983705

Man kan også lave et summary af dataen, som består af mange af de statistiker navnt ovenpå:

summary(x)

##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.
## -2.0094 -0.4645  0.1106  0.1369  0.8636  2.0409
```

### 1.8.3 tapply()

En meget brugbar funktion, som er værd at vide, er `tapply()`.

```
data(iris)
tapply(iris$Sepal.Length,iris$Species,mean) # ovenstående i kun en linje

##      setosa versicolor virginica
##      5.006      5.936      6.588
```

Her tager vi en variabel der hedder `Sepal.Length`, opdeler den efter `Species`, og beregner `mean` for enhver af de tre arter i `Species` (setosa, versicolor og virginica). Man kan opnå det samme resultat ved at beregne `mean` for de tre `Species` hver for sig (en tilgang, der ikke opskaleres særlig godt!):

```
# gennemsnit Sepal Length for Species setosa
mean_setosa <- mean(iris$Sepal.Length[iris$Species=="setosa"])

# gennemsnit Sepal Length for Species versicolor
mean_versi <- mean(iris$Sepal.Length[iris$Species=="versicolor"])

# gennemsnit Sepal Length for Species virginica
mean_virgin <- mean(iris$Sepal.Length[iris$Species=="virginica"])

c(mean_setosa,mean_versi,mean_virgin)
```

```
## [1] 5.006 5.936 6.588
```

Det er også værd at ved koncepten, fordi vi kommer til lære en lignende koncept i `tidyverse` (med `group_by` og `summarise`).

## 1.9 Statistiske tester

Her giver jeg et oversigt over nogle af de baserende tests man kan lave på data i R - det giver noget, du kan referere til senere hvis der er brug for det. Jeg går ikke i detaljer eller teorien af testerne (se dit tidligere kursus), men jeg forventer at I er i stand til at bruge dem på en hensigtsmæssigt måde i R, og

fortolker resultaterne. Vær ikke bekymret hvis du ikke har set de hele før, jeg giver masser a muligheder for at øve statistik gennem forløbet.

### 1.9.1 Korrelation

Måler sammenhængen mellem to normalfordelte variabler:

- $> 0$  betyder, at der er en positiv sammenhæng
- $< 0$  betyder, at der er en negativ sammenhæng
- $= 0$  betyder, at der er ingen sammenhængen mellem de to variabler

```
data(cars)
cor(cars$speed, cars$dist)
```

```
## [1] 0.8068949
```

Man kan teste om korrelationen er signifikant ved at bruge `cor.test()`

```
cor.test(cars$speed, cars$dist)
```

```
##
## Pearson's product-moment correlation
##
## data: cars$speed and cars$dist
## t = 9.464, df = 48, p-value = 1.49e-12
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6816422 0.8862036
## sample estimates:
##      cor
## 0.8068949
```

Så kan man se, at p-værdien er 0, der er under 0.05, så konkludere man, at der er en signifikant korrelation mellem de to variabler.

### 1.9.2 Test for uafhængighed (chi-sq test)

Her undersøger man, om der er en sammenhæng mellem antal observationer i to forskellige kategorier. Se for eksempel følgende tabel, der viser antal kopi af en gen variant og to forskellige farver som phenotype (farve på en type blomst):

	0	1	2
red	29	31	16
pink	11	16	24

Vi vil gerne vide, om phenotype er afhængig af genotype:

- $H_0$  : antal gen copi og phenotype er uafhængig af hinanden VS

- $H_1$  : antal gen copi og phenotype er afhængie af hinanden

Testen går ud på, at man beregner forventede værdier (baserende på de totals under nullhypotesen af de er uafhængige) og sammenligne forventede værdier med observerede værdier. Man laver testen i R ved at benytte funktionen `chisq.test`:

```
chisq.test(dat)

##
## Pearson's Chi-squared test
##
## data: dat
## X-squared = 9.9516, df = 2, p-value = 0.006903
```

Her er p-værdien = 0.006903 < 0.05, så vi forkaster nulhypotesen og konkluderer, at der er en afhængighed mellem de to variabler. Man kan også se fra rådatasættet, at der er langt flere røde blomster, der har ingen kopi af genet end der er røde blomster, der har to kopier af genet, og mønstret er omvendt i tilfældet af de lyserøde blomster.

### 1.9.3 1 sample t-test

For at vise en 1-sample t-test, simulerer jeg noget data fra den normal fordeling med `mean = 3`.

```
set.seed(290223) # bare for at få den samme resultat hver gang
x <- rnorm(10, mean = 3, sd = 1)
```

Forestil dig, at du ikke helt stoler på funktionen `rnorm()` og gerne vil teste, om `x` virkelig kommer fra en normal fordeling med et gennemsnit ( $\mu$ ) på tre. Nulhypotesen og alternativ hypotesen (2-sidet test) er således:

- $H_0 : \mu = 3$ , VS
- $H_1 : \mu \neq 3$

For at lave testen i R, bruger man funktionen `t.test()` og angiver `mu = 3` for at reflektere vores hypoteser:

```
t.test(x, mu = 3)

##
## One Sample t-test
##
## data: x
## t = -1.1448, df = 9, p-value = 0.2818
## alternative hypothesis: true mean is not equal to 3
## 95 percent confidence interval:
##  2.169968 3.272231
## sample estimates:
```

```
## mean of x
## 2.721099
```

Fra resultatet kan man se, at p-værdien er estimeret som 0.2818, og da den er  $> 0.05$  forkaster vi ikke nulhypotesen, og konkluderer at  $\mu = 3$ .

**Bemærkning:** da vi simulerede vores data fra en normal fordeling med et gennemsnit på tre, vidste vi i forvejen at det korrekte svar er, at beholde nulhypotesen. Havde vi forkastet nulhypotesen, havde vi lavet en **type I fejl** - det vil sige, at vi forkaster nulhypotesen når det faktisk er sandt.

#### 1.9.4 2-sample t-test

Undersøger om der er en forskel i de gennemsnitlige værdier mellem to grupper - kan de to grupper betragtes til at stammer fra den samme normale fordeling? Hypoteserne er således (to-sidet):

- $H_0 : \mu_1 = \mu_2$ , VS
- $H_1 : \mu_1 \neq \mu_2$

I følgende kode simulere jeg to stikprøver, der kommer fra en normal fordeling med forskellige gennemsnitte og bruger funktionen `t.test`. Man kan angive at de to stikprøver har samme variance ved at skrive `var.equal = T` indenfor funktionen `t.test`:

```
x <- rnorm(10,3,1)
y <- rnorm(10,5,1)

t.test(x,y,var.equal = T)
```

```
##
##  Two Sample t-test
##
## data: x and y
## t = -5.4258, df = 18, p-value = 3.729e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -2.700858 -1.193081
## sample estimates:
## mean of x mean of y
## 2.783056 4.730025
```

Hvis man til gengæld ikke kan antage, at variansen er den samme i de to grupper:

```
x <- rnorm(10,3,1)
y <- rnorm(10,5,3) #større variance

t.test(x,y,var.equal = F) #var.equal=F er 'default' så man behøver ikke at specifere

##
```

```

## Welch Two Sample t-test
##
## data: x and y
## t = -2.0238, df = 11.77, p-value = 0.0663
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -3.9077927 0.1483728
## sample estimates:
## mean of x mean of y
## 2.757436 4.637146

```

Bemærk at hvis man kan antage at variancen er den samme, så har man mere **power** (kræft) til at kalde en virkelig forskel for signifikant.

### 1.9.5 Paired t-test

En paired t-test bruges når man for eksempel har målinger for den samme sæt personer i hver stikprøve, og man gerne vil teste om forskellen i værdier mellem de to stikprøver er signifikant. For eksempel hvis vi har “before” og “after” målinger for den samme 10 individer:

```

set.seed(320)
before <- rnorm(10,3,1)
after <- rnorm(10,6,2)

t.test(before,after,paired=T) #specify paired data

##
## Paired t-test
##
## data: before and after
## t = -9.3296, df = 9, p-value = 6.356e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -5.415186 -3.301613
## sample estimates:
## mean of the differences
## -4.358399

t.test(before-after,mu=0) #exactly the same result

##
## One Sample t-test
##
## data: before - after
## t = -9.3296, df = 9, p-value = 6.356e-06
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:

```

```
## -5.415186 -3.301613
## sample estimates:
## mean of x
## -4.358399
```

### 1.9.6 ANOVA (variansanalyse)

Har man flere grupper i stedet for to, kan man bruge ANOVA (analysis of variance eller variansanalyse). For en kategorisk variabel med  $k$  grupper, er nul/alternativhypotesen:

- $H_0 : \mu_1 = \mu_2 = \dots = \mu_k$
- $H_1 : \text{ikke alle middelværdier er enes}$

```
#simulere data til 3 forskellige grupper fra den normale fordeling med standard afvige
group1 <- rnorm(50,10,3)
group2 <- rnorm(55,10,3)
group3 <- rnorm(48,5,3)

#data må være i en dataramme, med den ene kolon = vores værdier, og den anden kolon =
y <- c(group1,group2,group3)
x <- c(rep("G1",50),rep("G2",55),rep("G3",48))
mydf <- data.frame("group"=x,"value"=y)
```

Til at udføre testen bruger man funktionen `lm`. Det er en forkortelse for “linear model” og kan bruges til at bygge op forskellige modeller. Her angiver vi en model, således at hver group (G1, G2 og G3 fra variablen `x`) har sin egen middelværdi (variablen `value`), hvilket er modellen under alternativhypotesen:

```
mylm <- lm(value~group,data=mydf) #H1 model
```

Under nullhypotesen har alle grupper den samme middelværdi og vi behøver derfor ikke at have variablen `group` en del af modellen. Vi betegner situationen i modellen ved at skrive 1, der betyder at de forventede værdier for den afhængige variabel `value` er bare dens middelværdi:

```
mylm_null <- lm(value~1,data=mydf) #H0 model
```

For at sammenligne de to modeller benytter vi funktionen `anova` (etter analysis of variance):

```
anova(mylm_null,mylm)
```

```
## Analysis of Variance Table
##
## Model 1: value ~ 1
## Model 2: value ~ group
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     152 2215.4
```

```
## 2      150 1509.9  2     705.55 35.047 3.245e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

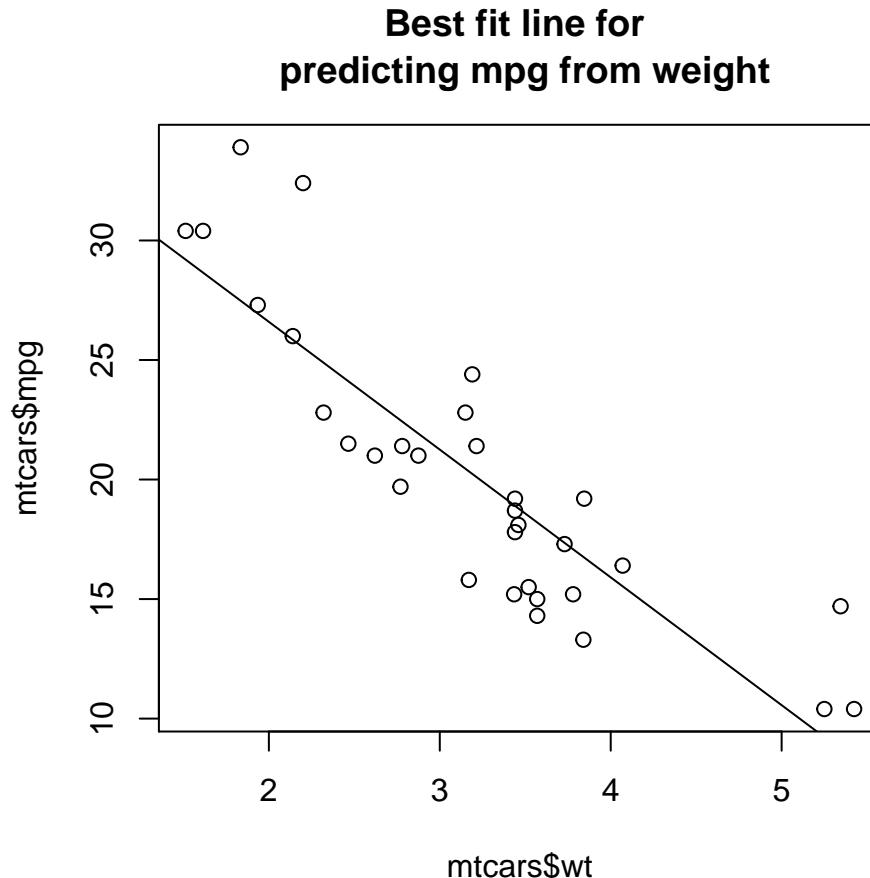
P-værdien er ( $<0.05$ ), så nulhypotesen er forkastet til fordel af alternativhypotesen, altså modellen, hvor hver gruppe har sin egen middelværdi. Bemærk at det er til trods af, at to af de tre grupper kommer fra en normal fordeling med præcis de samme middelværdier (det er nok, at den tredje gruppe har en ænderledes middelværdi).

### 1.9.7 Lineær regression

*OBS: se også video i forbindelse med Rmarkdown (næste emne), hvor jeg gennemgår lineær regression med R*

Formål: mäter (en retningsbestemt) relation mellem to kontinuerte variabler. I simpel lineær regression svarer det til, at man gerne vil finde den rette linje gennem punkterne, der bedste beskriver relationen.

Eksempel - datasættet `mtcars`, response (afgængig) variabel er `mpg` og predictor (uafhængig) variabel er `wt`.



Man skriver relationen i R som `mpg ~ wt` og benytter `lm()`(`lm(mpg~wt, data=mtcars)`):

```
mylm <- lm(mpg ~ wt, data=mtcars) # build linear regression model
mylm
```

```
## 
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
## 
## Coefficients:
## (Intercept)          wt
##       37.285        -5.344
```

Vores "Coefficients" beskriver den bedste rette linje:

- Skæringen (intercept): 37.285
- Hældningskoefficient (slope): -5.344

Det betyder, at hvis vægten `wt` af en bil stiger med 1, så stiger `mpg` ved -5.344 (det vil sige at `mpg` reduceres med 5.344).

### 1.9.8 R-squared coefficient of determination

Den  $R^2$  eller “forklaringsgraden” (coefficeint of determination) har til formål at forklare, hvor godt vores lineær model passer til de data. For eksempel hvor meget af variansen i `mpg` forklares af variablen `wt`?

- Hvis det er tæt på 1 - så er der en meget tæt relation (hvis man kender vægten, så vide man også `mpg` med stor sikkerhed)
- Hvis det er tæt på 0 - så er relationen svag - høj sandsynlighed for, at der er andre variabler der bedre kan forklare variansen i `mpg`.

I ovenstående model, kan man se den  $R^2$  værdi med `summary(mylm)`.

```
summary(mylm)

##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##   Min     1Q Median     3Q    Max 
## -4.5432 -2.3647 -0.1252  1.4096  6.8727 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 37.2851    1.8776 19.858 < 2e-16 ***
## wt          -5.3445    0.5591 -9.559 1.29e-10 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446 
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

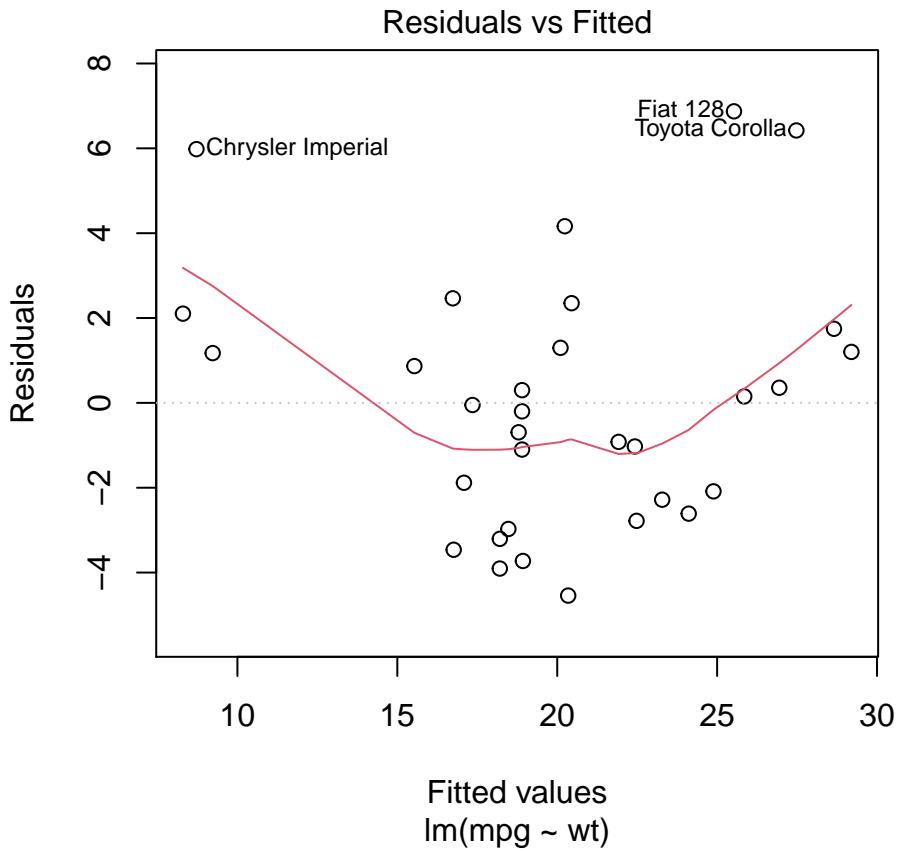
Det fortæller os, at  $R^2 = 0.7528$ .

### 1.9.9 Antigelser - lineær regression

- Normalfordelte residualer
- Residualer har samme spredning (varianshomogenitet)
- Uafhængighed
- Fit er linæer

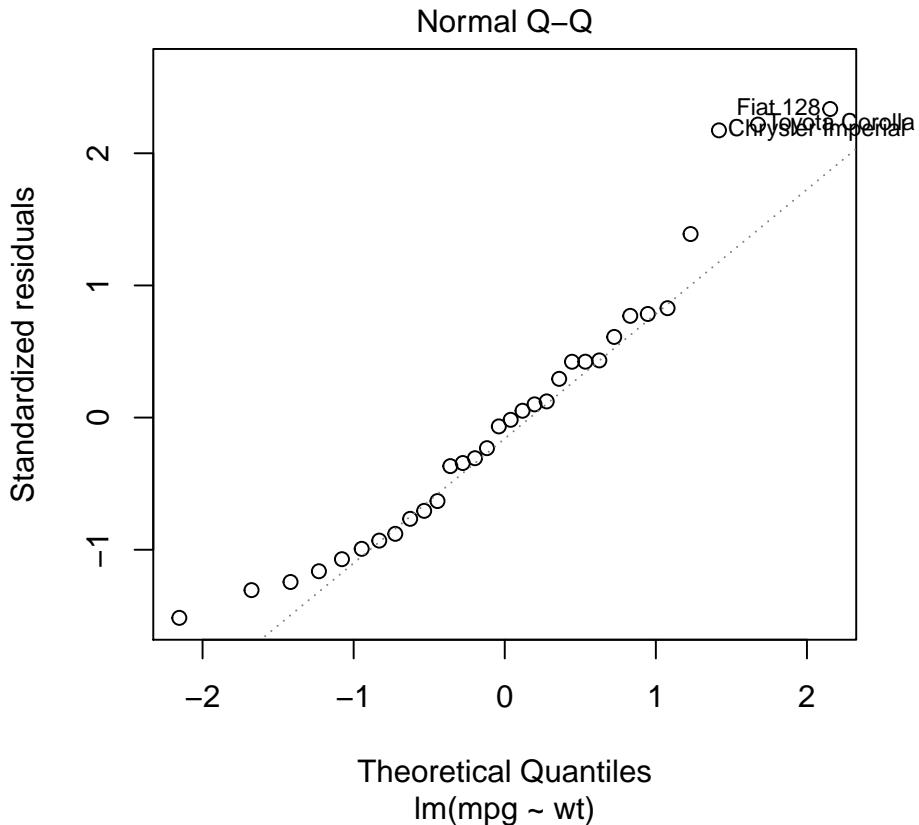
Koden `plot(mylm,which=c(1))` angiver residualer vs predikterede (fitted) værdier - de skal være tilfældigt fordelt over plottet og prikkernes varians skal være nogenlunde konstant langt x-aksen (det giver, at den røde linje er flade).

```
plot(mylm,which=c(1))
```



Med koden `plot(mylm,which=c(2))` kan man tjekke antagelsen på en normal fordeling. Punkterne skal være nogenlunde tæt på den diagonale linje.

```
plot(mylm,which=c(2))
```



### 1.9.10 Multiple lineær regression

Her kan man tilføje flere variabler i vores model formel.

```
mylm_disp <- lm(mpg ~ wt + disp, data=mtcars) # build linear regression model
summary(mylm_disp)
```

```
##
## Call:
## lm(formula = mpg ~ wt + disp, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.4087 -2.3243 -0.7683  1.7721  6.3484
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.96055   2.16454 16.151 4.91e-16 ***
## wt          -3.35082   1.16413 -2.878  0.00743 **
## disp        -0.01234   0.00833 -1.477  0.14294
```

```

## disp      -0.01773   0.00919  -1.929  0.06362 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.917 on 29 degrees of freedom
## Multiple R-squared:  0.7809, Adjusted R-squared:  0.7658
## F-statistic: 51.69 on 2 and 29 DF,  p-value: 2.744e-10

```

Her kan man se, at med tilføjelsen af variablen `disp`, er  $R^2$  steget til 0.7809. Bemærk, at jo flere variabler man tilføjer til modellen, jo større bliver  $R^2$ -værdien. Den adjusted  $R^2$  værdi er lavere fordi den prøver at tage højde for kompleksiteten af modellen (hvordan mange parametre der er).

Variablen `disp` er faktisk ikke selv signifikant når der er taget højde for variablen `wt` (p-værdien 0.0636 - tjek, at du selv kan finde værdien i resultatet).

Hvis en af de uafhængige variabler er kategorisk bruger man funktionen `anova` til at teste den overordnet effekt af den variabel. For eksempel har variablen `cyl` 3 mulige værdier (niveauer) - 4, 6 og 8. Vi kan inddrage variablen i vores model: ->

```
mylm_cyl <- lm(mpg ~ wt + factor(cyl), data=mtcars) # build linear regression model
summary(mylm_cyl)
```

```

##
## Call:
## lm(formula = mpg ~ wt + factor(cyl), data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.5890 -1.2357 -0.5159  1.3845  5.7915
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 33.9908   1.8878  18.006 < 2e-16 ***
## wt          -3.2056   0.7539  -4.252 0.000213 ***
## factor(cyl)6 -4.2556   1.3861  -3.070 0.004718 ** 
## factor(cyl)8 -6.0709   1.6523  -3.674 0.000999 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.557 on 28 degrees of freedom
## Multiple R-squared:  0.8374, Adjusted R-squared:  0.82
## F-statistic: 48.08 on 3 and 28 DF,  p-value: 3.594e-11

```

Man kan ikke se den overordnet effekt af `cyl` fra den ovenstående `summary` men man kan teste den med `anova`:

```
anova(mylm, mylm_cyl)

## Analysis of Variance Table
##
## Model 1: mpg ~ wt
## Model 2: mpg ~ wt + factor(cyl)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     30 278.32
## 2     28 183.06  2     95.263 7.2856 0.002835 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Så kan man se, at cyl er signifikant.

## 1.10 Problemstillinger

Lav quizzen og ellers vælg øvelser efter egen erfaring:

- 2-8 er meget grundlæggende og de fleste kan springer over hvis nogenlunde tryg med base-R
- 9-15 anbefaler jeg til alle som en god måde at tjekke viden på
- 16-20 øver hvordan man andre statistiske teste i R - regression kommer jeg ind på igen senere men det hjælper hvis du er tryg med brugen af funktionen `lm` til at lave modeller i ANOVA/simpel lineær regression (se også video og problemstillingerne i morgen).

### 1.10.1 Quiz - Basics

1) Se quiz i Absalon, der hedder “Quiz - Basics”.

### 1.10.2 Grundlæggende R

2) (helt baserende viden) Åbn en ny fil i Rstudio ved at trykke på “File” > “New File” > “R script”. Køre følgende kode en linje ad gangen og tjek, du kan forstå outputtet.

Husk at den nemmeste måde at køre kode er ved at trykke CMD+ENTER (Mac) eller WIN-KEY+ENTER (Windows).

```
2+2
2*2
x <- 4
x <- x+2
sqrt(x)
sqrt(x)^2
rnorm(10,2,2)
log10(100)
```

```
y <- c(1,4,6,4,3)
class(y)
class(c("a","b","c"))
mean(y)
sd(y)
seq(1,13,by=3)
```

**3) (helt baserende viden)** Køre følgende kode til at åbne nogle af de indbygget datasæt, som vi bruger i kurset. \* Prøve `head()`, `nrow()`, `summary()` osv. \* Prøve også `?cars` for at se en beskrivelse.

```
data(iris)
data(cars)
data(ToothGrowth)
data(sleep)
head(chickwts)
data(trees)
#se her for andre:
library(help = "datasets")
```

**4) (baserende plots)** Jeg giver nogle muligheder for datasættet “iris”. Afprøve funktionerne for nogle af de andre ovenstående indbygget datasæt, som du indlæst.

```
plot(iris$Sepal.Length,iris$Sepal.Width)
hist(iris$Sepal.Width)
boxplot(iris$Sepal.Length~iris$Species)
```

Man kan også gøre plotterne lidt pænere ved at give dem en titel/aksen-navne osv. Prøve `?plot` for at se nogle muligheder, og tilføje `ylab`, `xlab`, `main` (titel) i én af plotterne. Leg også med `col` (farver). Bemærk dog, at vi kommer til at ændre måden at lave plotter på når vi starter `ggplot2`.

**5) (dataframes)** Brug datasættet `cars` (`data(cars)`) til at:

- Lav et scatter plot med speed på x-aksen og dist på y-aksen
- Tilføj en ny kolon med følgende kode:

```
cars$fast <- cars$speed>15
```

- Brug `mean` på den nye variabel til at finde ud af proportionen af biler, der er hurtige
- Beregn gennemsnitsværdien af variablen `dist` for hurtige biler og ikke-hurtige biler hver for sig (brug funktionen `tapply`). Gem resultatet med `<-`.
- Brug `barplot` til at lave et plot af den gennemsnitlige `dist` for hurtige og ikke-hurtige biler.

**6) (dataframes)** Lav en ny dataframe (funktionen `data.frame()`) med tre

kolonner som hedder "navn", "alder" og "yndlings\_farve" (find bare selv på værdierne). Sørge for, at den har 4 rækker.

```
mydf <- data.frame("navn"= c("alice", "freddy", ...), "alder" = c(...), ...) #not run, slette ...
dim(mydf) # fire række og tre kolonner
mydf
```

**7) (dataframes)** Tilføj en ny variabel `random` til ovenstående dataframe, hvor værdierne kommer fra en normal fordeling med et gennemsnit på 5 og sd på 1 (bruge funktionen `rnorm`).

```
mydf$random <- #??
```

**8) (delmængder af dataframes)** Åbn datasættet "ToothGrowth" med følgende kode:

```
data("ToothGrowth")
?ToothGrowth
```

- Find delmængden af datasættet således at diet (variablen `supp`) er "OJ" og længden (variablen `len`) er større end 15.

```
newdf <- ToothGrowth[#skrive her til at lave subset af observationerne,]
```

- Hvor mange rækker er der i den nye dataframe `newdf`?
- Hvor mange unikke værdier er der i variablen `dose` (brug funktionen `unique`) ?
- Find delmængden af datasættet `ToothGrowth`, hvor variablen `dose` er 0.5 eller 1.5 (hint: brug `%in%` eller `l`) og `supp` er "VC".
- Beregn den gennemsnitlige længde for observationerne i delmængden.

### 1.10.3 Kort analyse med reaktionstider

**9) (indlæse data)** Åbn en fil, der sidder i Absalon og hedder "reactions.txt" ved at bruge funktionen `read.table()` (kalde det for `data`). Husk at tjekke, om selve filen har variabelnavne og bruge således `header=T` hvis nødvendigt.

```
data <- ... #replace ...
```

**10) (factor variabler)** Variablerne `subject` og `time` indlæses som henholdsvis data type 'int' (heltal) og "chr" (character) men de skal hellere være 'factor' variabler. Lav dem om til faktor variabler.

```
#gør subject til en faktor
data$subject <- as.factor(data$subject)
```

```
## gör den samme her for time:
```

- Hvor mange niveauer er der i hver af de to variabler?

**11) (delmængde af dataframe)**

Lav to delmængder af ovenstående datasæt -

- én til alle observationer fra tidspunktet “before” (brug koden `data$time == "before"` til at udvælge rækkerne fra dataframe) og
- én til alle observationer fra tidspunktet “after”.

```
RT_before <- data[#skrive her, ]
RT_after <- #skrive her
```

- Brug `RT_before` og lav en delmængde der viser alle observationer fra tidspunktet “before” med en reaktionstid af mindst 800.  
– Hvor mange personer opfylder kriterien?

```
RT_before_mindst800 <- #skrive her
```

**12) (mean og tapply)** Benyt funktionen `mean` til at beregne den gennemsnitlige reaktionstid (variablen `RT`) til “before” og “after” hver for sig (brug ovenstående delmængder).

- Prøv også at anvende funktionen `tapply` på det oprindelige datasæt `data` til at beregne samme middelværdi med mindre kode.

```
tapply(#skrive her, #skrive her, #skrive her)
```

- Er reaktionstiderne blevet hurtigere eller langsommere i gennemsnit?

**13) (beregn forskellen og mean)**

Bemærk, at datasættet er ‘paired’ - målingerne er lavet på de præcis samme personer både “before” og “after”.

- Opret en vector `diff`, der er ændringen i reaktionstiderne mellem personerne “before” og “after”.
- Beregn den gennemsnitlige forskel i reaktionstiderne.

```
diff <- #change in reaction time between before and after
mean(diff)
```

- Tjek tegnet stemmer overens med din konklusion fra **11**) - hvis den er positiv, så betyder det, at reaktionstiderne er blevet langsmommere.

**14) (lav t-test i R)** Lav en t-test (funktionen `t.test`) for at teste hypotesen at den gennemsnitslige forskel i reaktionstiderne mellem “before” og “after” er anderledes end 0.

```
t.test(#skrive her...)
```

Find følgende i outputtet fra R:

- Hvor er test-statistik `t`?
- Hvor er p-værdien?
- Hvad er alternativhypotesen?

**15)** Skriv en kort sætning med din konklusion.

### 1.10.4 Ekstra øvelser med statistik tests

**16) (Chi-sq)** Kør følgende kode til at få en tabel (selve koden er ikke vigtigt):

```
mytable <- structure(c(80L, 97L, 372L, 136L, 87L, 119L), .Dim = 3:2, .Dimnames = structure(list(
    c("First", "Second", "Third"), c("Died", "Survived")), .Names = c("Class", "Survival")), clas
```

```
mytable
```

```
##           Survival
## Class     Died Survived
##   First     80     136
##   Second    97      87
##   Third    372     119
```

Tabellen omhandler personer ombord skibet ‘Titanic’ (der sank den 15. april 1912 efter et sammenstød med et isbjerg 600 km sydøst for Halifax, Nova Scotia i Canada). Tabellen angiver hvor mange passagerer tilhørte de tre klass (førsteklass, andenklass, trejdeklas), delte efter overlevelsesudfald (døde eller overlevede tragedien).

- Benyt funktionen `chisq.test()` på tabellen.
- Hvad er nulhypotesen?
  - Overlevelsesudfald er uafhængig af klasse
- Er testen signifikant?
  - p-value < 2.2e-16 - ja
- Er passagerernes klasse så uafhængige af deres chance for at overleve tragedien?
  - Jeg forkaster nulhypotesen og konkluderer de to variabler afhængige af hinanden
- Hvilken klasse havde den bedste chance for at overleve?
  - Førsteklasse = meget højere chance

Vi kommer til at arbejde meget mere med datasættet `Titanic` i emnet Tidyverse - dag 1!

**17) (Korrelation analyse)** Åbn datasættet `trees` og lav et scatter plot med variablen `Girth` på x-aksen og variablen `Volume` på y-aksen.

```
data(trees)
summary(trees)
```

- Anvend funktionen `cor.test` for at teste, om der er en signifikant korrelation mellem de to variabler. Brug `method = "pearson"` (det er dog faktisk default)

```
cor.test(???, ???, method="pearson")
```

- Hvad er korrelationen mellem `Girth` og `Volume`?
- Hvad er p-værdien? Er den signifikant?

**18) (ANOVA)** OBS: hvis du føler dig utryg med funktionen `lm()` - der kommer en video om det i morgen (i forbindelse med emnet Rmarkdown).

Kør følgende kode til at lave variansanalyse, der tester hulhypotesen hvor den gennemsnitlige værdi af variablen `Sepal.Width` er ens for hver af de tre arter (variablen `Species`) fra datasættet `iris`:

```
data(iris)
```

```
#model under H0: no difference according to group variable Species (1 just means "fit"
model_h0 <- lm(Sepal.Width ~ 1, data=iris)

#model under H1: each level of group variable Species has its own mean
model_h1 <- lm(Sepal.Width ~ Species, data=iris)

#compare two models - significant p-value equates to choosing H1 model
anova(model_h0, model_h1)
```

```
## Analysis of Variance Table
##
## Model 1: Sepal.Width ~ 1
## Model 2: Sepal.Width ~ Species
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     149 28.307
## 2     147 16.962  2     11.345 49.16 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Kig på outputtet:

- Hvilken model reflekterer nulhypotesen?
- Hvilken model reflekterer alternativhypotesen?
- Hvor er p-værdien?
- Er der en signifikant forskel i den gennemsnitlige `Sepal.Width` efter de forskellige `Species`?

Brug funktionen `tapply` for at finde ud af, hvad er den middelværdi `Sepal.Width` til hver af de tre arter.

**19) (ANOVA)** Lav en lignende analyse på datasættet `chickwts` for at svare på spørgsmålet:

- Er der en forskel i den gennemsnitlige vægt (variablen `weight`) efter fodertypen (variablen `feed`)? Med andre ord er vægt afhængig af fodertypen?

```
data(chickwts)
```

Valgfri - vi gennemgår også lineær regression i morgen og du kan altid komme tilbage senere hvis du har bruge for det

#### 20) (Lineær regression)

- Brug `lm` til at lave en simpel lineær regression, således at respons variablen `Volume` er afhængig af variablen `Girth` (datasætet `trees`).

```
mylm <- lm(???, data=trees)
```

Brug `summary` på din model for at finde følgende værdier:

- Hvad er r.squared? (multiple)
- Er variablen `Girth` signifikant?
- Hvad er ligningen på den bedste rette linje (husk formen  $y = ax + b$ )?

#### 21) (Kort intro til multiple lineær regression)

Tag ovenstående model og tilføj variablen `Height` som en ekstra prediktør (uafhængig) variabel i modellen med en "+" tegn:

```
mylm_height <- lm(??? ~ ??? + ???, data=trees)
summary(mylm_height)
```

Bemærk at det ikke betyder, at de to variabler skal lægges sammen, men at vi gerne vil have både variablerne i modellen som uafhængig variabler (med andre ord er `Volume` afhængig af både `Girth` og `Height`).

Benyt `summary` på modellen og prøv at finde følgende:

- Hvad er den den (multiple) r.squared værdi?
- Hvor meget ændre den (multiple) r.squared værdi i forhold til modellen med kun variablen `Girth`?
- Er `Volume` signifikant afhængig af `Height` (efter at man har taget højde for `Girth`)?

Brug funktionen `anova` til at sammenligne modellen uden `Height` med modellen med `Height`

```
anova(#model without height, #model with height)
```

Bemærk, at i dette tilfælde er p-værdien fra ANOVA samme p-værdi fra `summary(mylm_height)`.



## Chapter 2

# Introduktion til R Markdown

I dag begynder vi at arbejde med R Markdown. Selve emnet er relativt kort og er designet til, at du kan komme i gang med at bruge R Markdown i praksis, men notaterne refererer også til nogle ekstra muligheder så du kan indrette dit dokument efter eget ønske. Der er to videoer - én er en ‘quick-start guide’ for at komme i gang, og den anden viser en simpel lineær regression i R Markdown - har du ikke brug for lidt genopfriskning, kan man også se mere om funktionaliteten i R Markdown.

Efter quizzerne og problemstillinger er der en worksheet, hvor du kan øve dig videre med de typer opgaver vi kommer til at se i workshopperne. Fra næste gang (fredag) skifter vi emnet til visualiseringer i **ggplot2**, og vi arbejder naturligvis i R Markdown fremadrettet.

### 2.1 Hvad er R Markdown?

R Markdown er både en nem og fleksibel måde at arbejde med R til projekter på. Her kan du kombinere din R-kode, output og tekst i samme dokument, og derudover fremviser et pænt HTML dokument fra det, som potentielt kan deles med andre. Jeg anbefaler at du bruger R Markdown til alle opgaverne i kurset og **ved eksamen forventer jeg at du afleverer et HTML dokument** til mig med din “knittede” kode fra din analyse.

### 2.2 Installere R Markdown

R Markdown er, ligesom R, gratis og ‘open source’. Den fungerer indenfor RStudio and kan installeres ved at bruge den følgende kommando:

```
install.packages("rmarkdown")
```

## 2.3 Videodemonstrationer

Jeg har lavet to videoer som kan ses her:

Video 1:

Jeg viser

- hvordan man laver et nyt dokument i R Markdown
- hvordan man skriver tekst ind i dokumentet
- hvordan man bruger “knit” til at lave et HTML-dokument
- hvordan man opretter og kører kode chunks

Link her hvis det ikke virker nedenunder: <https://vimeo.com/702416505>

Video 2:

Jeg viser en kort lineær regression analyse:

- Indlæse datasæt og lave plot af datasættet
- Lynhurtig gennemgåelse af ligningen for rette linje
- Hvordan man anvender funktionen `lm()` til at fitte en lineær model
- Fortolkelse af resultaterne

Link her hvis det ikke virker nedenunder: <https://vimeo.com/701240044>

## 2.4 Oprette et nyt dokument i R Markdown

Man åbner et nyt rmarkdown dokument ved at trykke “New” > “New File” > “New R Markdown...”. Man kan også trykke på “+” knappen øverst til venstre hjørne.

Dernæst angiver man en titel (det kan ændres senere hvis der er bruge for det) og bekræfter, at outputtet kommer i HTML form. I kurset arbejdes der kun med HTML dokumenter, men man har også andre muligheder, som du er velkommen til at afprøve (PDF/Word/Shiny osv...).

### 2.4.1 YAML

Den første sektion af dokument skrives i hvad der kaldes for ‘YAML’. (Dette står for ‘YAML Ain’t Markup Language’).

Det indeholder oplysninger om dokumentet, og her kan man specificere forskellige muligheder - fk. titel, forfatter, output-type (fks. HTML eller PDF), dato, osv. I de fleste tilfælde nøjes vi med at bruge standard indstillinger, men hvis man gerne vil lære mere om de forskellige muligheder med YAML, kan man læse her:

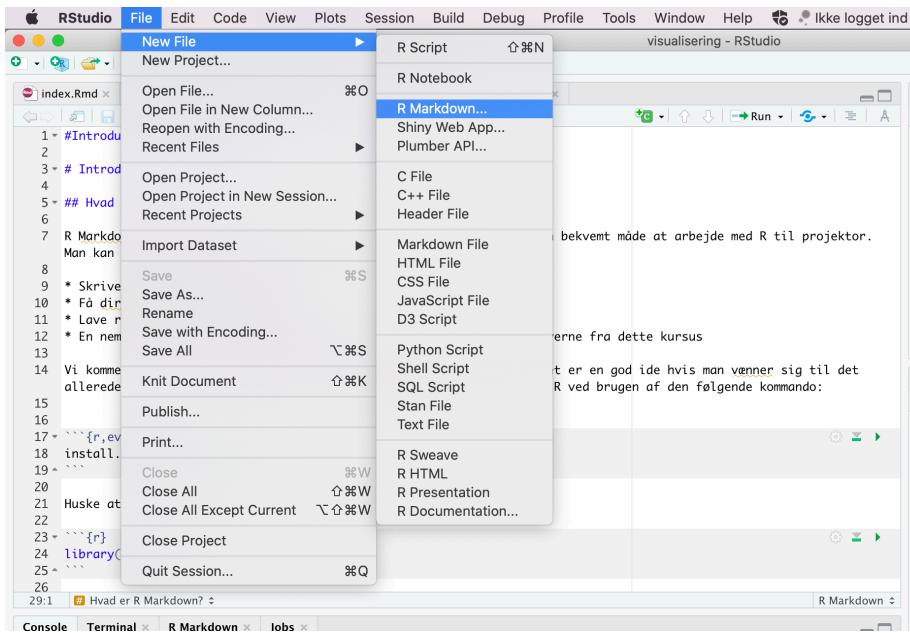


Figure 2.1: Hvordan man åbner et nyt R Markdown dokument

<https://bookdown.org/yihui/rmarkdown/html-document.html>

eller se en liste af muligheder her på dette cheatsheet:

<https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>

#### 2.4.2 Globale options

Der er også tekst som ser ud som følgende:

Med funktionen `opts_chunk$set()` kan man specificere de globale indstillinger, som styrer hvordan det færdige dokument ser ud. I dette tilfælde er de fleste parametre angivet som 'default' (da de ikke er nævnt eksplisit), og `echo` er den eneste der har noget angivet. Hvis `echo` er `TRUE`, så betyder det, at når man "knitter" sin kode (processen, der få et HTML dokument frem, se nedenfor), så kan man også se koden, der blev kørt, samt dens output, i det færdige HTML dokument, der kommer frem.

## 2.5 Skrive baseret tekst

Her er nogle brugbare muligheder for at skrive tekst i opgaverne eller rapporter:

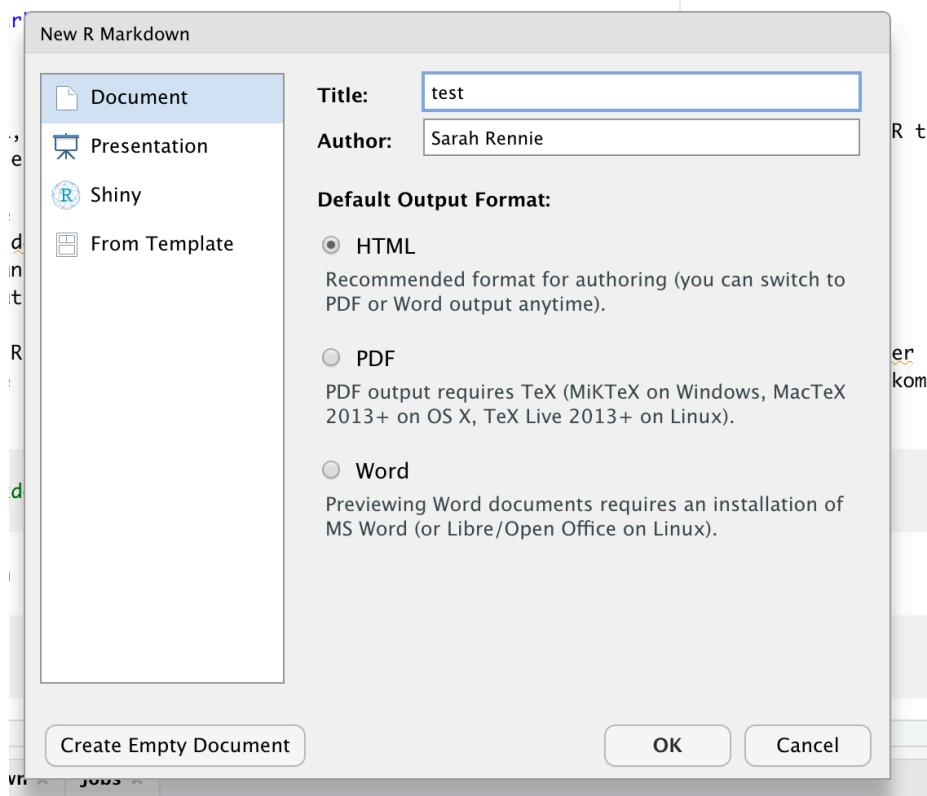


Figure 2.2: Hvordan man åbner et nyt R Markdown dokument

```

1 ---  
2 title: "test"  
3 author: "Sarah Rennie"  
4 date: "3/31/2021"  
5 output: html_document  
6 ---  
7

```

Figure 2.3: Hvordan man åbner et nyt R Markdown dokument

```

7  
8 ````{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ````  


```

Figure 2.4: Hvordan man åbner et nyt R Markdown dokument

```
*italic* **bold**
```

```
_italic_ __bold__
```

```
italic bold
```

```
italic bold
```

### 2.5.1 Headers

Man kan også lave sektioner:

```
# Header 1
```

```
## Header 2
```

```
### Header 3
```

# Chapter 3 Header 1

## 3.1 Header 2

### 3.1.1 Header 3

Figure 2.5: Caption for the picture.

### 2.5.2 liste

```
* Item 1  
* Item 2  
  + Item 2a  
  + Item 2b
```

- Item 1

- Item 2
  - Item 2a
  - Item 2b

## 2.6 Knitte kode

Man bruger *Knit* for at gengive filen i HTML form. Når man trykker på knappen *Knit*, bliver samtlige koder i filen kørt og et HTML dokument fremvises. Bemærk, at **koderne bliver kørt på ny hver gang man knitter**, uden hensyn til hvad du har i din nuværende workspace i RStudio. Det betyder, at hvis du eksempelvis har pakken `tidyverse` indlæst på din computer men har glemt at skrive `library(tidyverse)` eksplisit på toppen af dit dokument, så får du en fejlmeddeelse hvis du bruger tidyverse-baserede funktioner nogle steder.

## 2.7 Kode chunks

Man skriver selve R kode indenfor hvad der kaldes for “chunks”. Man kan oprette en ny chunk på flere måder - enten ved at trykke på den *Insert a new code chunk* knap ovenpå, eller ved at trykke *Cmd+Option+I* på tastaturet (hvis man bruger MAC) eller *Ctrl+Alt+I* (hvis man bruger Windows). Det er værd at huske den keyboard-shortcut - det sparer meget tid efter egen erfaring!

Her er et eksempel af en chunk:

```
# This is a chunk, let's write som R code
x <- 1
x + 1

## [1] 2
```

For at køre en chunk, trykker man på den grønne pile øverste i højre hjørne på selve chunk (der hedder *Run Current Chunk* når du holder musen over den). Resultatet kan ses nedenunder, som i ovenstående.

Bemærk, at når du arbejde med dit R Markdown dokument er det generelt hurtigere at bruge den grønne pile / *Run Current Chunk* i stedet for at knitte hele dokumentet hver gang man vil køre kode. Det er fordi her kører man kun den enkel chunk i stedet for hele dokumentet på ny (herunder indlæsning af pakker og eventuelle store filer), som er tilfældet med *Knit*.

### 2.7.1 Et godt råd når man arbejder med chunks

Til længere opgaver er det god praksis at sikre jævnligt, at man kan få et HTML dokument frem ved at knitte, selvom du kører din chunks lokalt mens du udvikler din kode - det vil sige, at du ikke får en alvorlig fejlmeddeelse, der forhindrer din koder at knitte. **Det er dit ansvar at sikre, at din kode fungerer som**

helhed og du kan dermed producere et HTML dokument med din løsninger.

### 2.7.2 Chunk indstillinger

I R Markdown er der mange muligheder for at styre hver eneste chunk i dit dokument - hvordan skal R håndtere koden med hensyn til evaluering og præsentering (især med hensyn til tabeller og plotter) af en bestemt chunk i dit dokument? Det kommer meget an på, hvem du gerne vil viser dit dokument til. For eksempel, i nuværende kursusnotater vil jeg gerne have generelt, at du ser alle min kode (en global indstilling), men nogle gange vil jeg foretrækker noget andet - en chunk som viser noget jeg ikke vil have kørt, eller ændre på størrelsen af et plotte i en bestemt chunk. For eksempel, en chunk med indstillingen `eval=FALSE` ser sådan ud (fjerne # symbol)

```
#``{r, eval=FALSE}
#
#````
```

Her er nogle muligheder (sektionen “Embed code with knitr syntax”):

<https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>

Her er seks populær muligheder som jeg har kopiret fra nettet:

- `include = FALSE`
  - prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.
- `echo = FALSE`
  - prevents code, but not the results from appearing in the finished file. This is a useful way to embed figures.
- `message = FALSE`
  - prevents messages that are generated by code from appearing in the finished file.
- `warning = FALSE`
  - prevents warnings that are generated by code from appearing in the finished.
- `fig.cap = "..."`
  - adds a caption to graphical results.
- `eval = FALSE`
  - does not evaluate the code

## 2.8 R beregninger indenfor teksten i dokument ('inline code')

I nogle tilfælde vil man køre R kode "inline", det vil sige, direkte indenfor teksten, eksempelvis indenfor en sætning. Dette gøres ved at skrive på følgende måde:

```
Her er min `kode`
```

Ovenstående ser sådan ud når skrevet direkte indenfor teksten:

Her er min kode

I dette tilfælde, er der ikke noget R kode som er blevet kørt. Hvis man vil køre R kode indenfor teksten skriver man (for eksempel):

```
De gennemsnitlige antal af observationer er `r mean(c(5,7,4,6,3,3))`
```

Ovenstående ser sådan ud når skrevet direkte indenfor teksten:

De gennemsnitlige antal af observationer er 4.6666667

Og bemærk, at hvis man glemmer 'r', så bliver koden ikke kørt:

```
De gennemsnitlige antal af observationer er `mean(c(5,7,4,6,3,3))`
```

giver:

De gennemsnitlige antal af observationer er `mean(c(5,7,4,6,3,3))`

Brugen af kode inline kan være en kæmpe fordel når man gerne vil skrive noget om en analyse, hvor man referere til forskellige statistik beregninger som man har beregnet i R (eksempelvis en middelværdi eller p-værdi). Hvis man skriver eller kopier et tal direkte og datasættet eller analysemetoden ændre sig på en eller anden grund, så bliver beregningerne indenfor teksten ikke opdateret, og så risikerer man at have en fejl i den endelige rapport. Bruger man inline code, så er beregningerne opdateret automatiske, uden at tænke over det.

## 2.9 Working directory

Bemærk at den måde man sætter en working directory er ændleredes i R Markdown i forhold til base-R. Hvis man bruger `setwd()` i en chunk, sætter man kun den working directory i den pågældende chunk og ikke i de efterfølgende chunks.

I R Markdown er standarden (default), at din working directory er mappen som du gemmer din .Rmd fil. Hvis du genre vil bruge noget andet, kan du tilføje `knitr::opts_knit$set(root.dir = '/tmp')` til din globale indstillinger chunk på toppen af din fil, hvor '/tmp' skal ændres til din ønskede mappe.

```
```{r, setup, include=FALSE}
knitr::opts_knit$set(root.dir = '/tmp')
```
```

## 2.10 Matematik

Man kan også skrive matematik (Latex) i R Markdown - eksempelvis  $\int_0^5 x^2 dx$  vil ser ud som  $\int_0^5 x^2 dx$  i dit HTML dokument. Jeg forventer ikke at du lære Latex men det er af og til brugbart - for eksempel en rette linje ligning er  $y = 3.4x + 2.1$  giver  $y = 3.4x + 2.1$  eller en hypotese:  $H_0: \mu = 0$  giver  $H_0: \mu = 0$ . Det er op til dig hvor meget du bruger matematik måde i dine egne dokumenter.

## 2.11 Problemstillinger

- 1) Der er en kort **quiz** i Absalon, som hedder “Quiz - R Markdown”.
- 2) Lav et nyt R Markdown dokument i RStudio. Prøve at lave en liste og nogle overskrifter i forskellige størrelser.
- 3) Nu tryk på **Knit** knappen og tjek at et HTML-dokument fremvises på din skærm.
- 4) Rediger på titlen (den er en del af din YAML-header oppe på toppen af din fil) - kald dit dokument for “My first R Markdown document”, og tryk på **Knit** igen for at se ændringen i dit HTML dokument.
- 5) Opret en ny R-chunk, og tilføj noget kode, eksempelvis

```
x <- rnorm(20,1,2) #make a sample of normally distributed data
plot(x)
```

- husk shortcut CMD+OPT+I eller CTRL+WIN+I når man oprette en chunk (det sparer tid)
  - tryk på den grønne pile
  - prøve også at køre en linjen ad gangen med CMD+Enter/CTRL+Enter
  - lav flere chunks med adskillige kode som du vælger
  - tryk på **knit** og bemærk, at det tager længere tid at **knit** hver eneste gang man ændre noget, end når man bare kører chunks individ indenfor dit dokument
- 6) Tryk på “hjulen”-knappen i øverste højre hjørne af en af din chunks og prøv at ændre på de forskellige chunk indstillinger. Tryk på ‘knit’ for at se, hvad der sker.
  - 7) Hver gang du knitter, du lave et HTML dokument. Nu prøv at lave en andet type dokument i stedet for - erstatte **html\_document** med

`word_document` i YAML (toppen af din .Rmd fil)

- Se her for endnu flere muligheder: <https://bookdown.org/yihui/rmarkdown/output-formats.html>
- 8) Tilføj følgende chunk til dit dokument og tryk på “knit”. Få du en fejlmeddelse?

```
data(mtcars)
mtcars %>% filter(cyl==6)
```

Bemærk, at du får en fejlmeddelse fordi, du endnu ikke har indlæst den påkrævet pakke til at få koden til at virke. Det kan ske, selvom du måske har indlæste pakken i Console eller i Packages tab.

- Først prøve at køre “library(tidyverse)” indenfor Console og dernæst prøve at knitte dit dokument igen - du får stadig en fejmeddelse.
- Tilføj `library(tidyverse)` øverst i din chunk. Nu bør dit dokument knitte.
- 9) Erstat linjen `output: html_document` med følgende i din YAML metadata oppe i toppen af din .Rmd fil:

```
output:
  html_document:
    code_folding: hide
```

Knit og se hvad, der sker.

- Erstat `hide` med `show` og kig på forskellen.
- 10) Brug `$ $` til at skrive en ligning ind i teksten i din .Rmd fil. Prøv for eksempel `$\bar{x}_i = \frac{1}{n} \sum_{i=1}^n x_i$` og knitte dit dokument for at tjekke, om du får formlen til middelværdien.
- 11) (**Worksheet**) Ind på Absalon har jeg lagt en R Markdown (.Rmd) fil som hedder “R Markdown opgave”, som du kan bruge til at starte med at arbejde med R Markdown baserede opgaver. Det kombinerer koncepter fra det forudgående kapitel om de grundlæggende ting i R og statistik.

## 2.12 Færdig for i dag og næste gang

Husk at sende mig eventuelle spørgsmål, som jeg kan svare på enten direkte eller i forelæsning næste gang. Næste gang begynder vi at arbejde vi med R-pakken `ggplot2`, der bruges til at lave høj kvalitet visualiseringer fra datasæt.

## 2.13 Ekstra links

- Her er en ‘quick tour’ [https://rmarkdown.rstudio.com/authoring\\_quick\\_tour.html](https://rmarkdown.rstudio.com/authoring_quick_tour.html)

- Handy R Markdown Cheatsheet: RStudio has published numerous cheatsheets for working with R, including a detailed cheatsheet on using R Markdown! The R Markdown cheatsheet can be accessed from within RStudio by selecting *Help > Cheatsheets > R Markdown Cheat Sheet*.



## Chapter 3

# Visualisering - ggplot2 dag 1



### 3.1 Inledning og videoer

Dette kapitel giver en introduktion til hvordan man visualiserer data med R-pakken **ggplot2**.

### 3.1.1 Læringsmålene for dag 1

I skal være i stand til at:

- Forstå hvad “Grammar of Graphics” betyder og sammenhængen med den **ggplot2**-pakke
- Lære at bruge funktionen **ggplot** og den relevante **geoms** (**geom\_point()**, **geom\_bar()**, **geom\_histogram()**, **geom\_boxplot()**, **geom\_density()**)
- Lave en ‘færdig’ figur med en titel og korrekte etiketter på akserne
- Begynde at arbejde med farver og temaer

### 3.1.2 Hvad er ggplot2?

De fleste i kurset har anvendt funktionen **plot()**, der er den standard base-R funktion til at lave et plot. Man kan godt blive ved med at lave plotter i base-pakken, men det er ofte meget tidskrævende så snart man gerne vil lave noget mere indviklet eller pånere.

En alternativ løsning er pakken **ggplot2**, som står for “grammar of graphics” (se nedenunder for nærmere forklaring). **ggplot2** er den mest populær pakke fra **tidyverse**, og som vi kommer til at se i dette kapitel, har den en ret logisk tilgang, hvor man opbygger et plot i forskellige komponenter. Det kan virke uoverskueligt i første omgang, men er faktisk meget intuitiv når man er vant til det. Det nyttige i at lære **ggplot2** kan også ses når man begynder at integrere de øvrige **tidyverse** pakker fra kapitel 4.

### 3.1.3 Brugen af materialerne

Jeg har optaget videoer hvor jeg viser nogle ‘quick-start’ type eksempler indenfor min RStudio. Videoerne er ikke designet til at indeholde alle detaljer, men til at fungere som udgangspunkt til at kunne komme i gang med øvelserne. Vær opmærksom på, at alle koder i videoerne findes også i kursusnotaterne, hvis du selv vil afprøve dem. Jeg anbefaler at du bruger kursusnotaterne som en reference gennem kurset når man arbejder på opgaverne, og vær også opmærksom på, at jeg nogen gange introducerer nye ting i selve øvelserne.

### 3.1.4 Video ressourcer

- I video 1 demonstrerer jeg, hvordan man lave sit første plot med **ggplot2**.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/701245598>

- I video 2 dækker vi boxplots.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/701245695>

- I video 3 demonstrerer jeg barplots.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/704025240>

- Video 4: Histogram og density plots

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/703699213>

## 3.2 Transition fra base R til ggplot2

Vi starter som udgangspunkt med base-R og viser, hvordan man laver et lignende plot med **ggplot2**. Til dette formål bruger vi det indbyggede datasæt, der hedder **iris**. Det er et meget berømt datasæt, og det er næsten sikkert, at du støder ind (eller har stødt ind) i det mange gange uden for dette kursus, enten på nettet eller i forbindelse med andre kurser som handler om R. Datasættet var oprindeligt samlet af statistikker og biologer Ronald Fisher i 1936 og indeholder 50 stikprøver, der dækker forskellige målinger, for hver af tre arter af planten iris (Iris setosa, Iris virginica og Iris Versicolor).



Som vi også så i grundlæggende R, kan man indlæse et indbyggede datasæt med hjælp af funktionen **data()**.

```
data(iris)
```

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

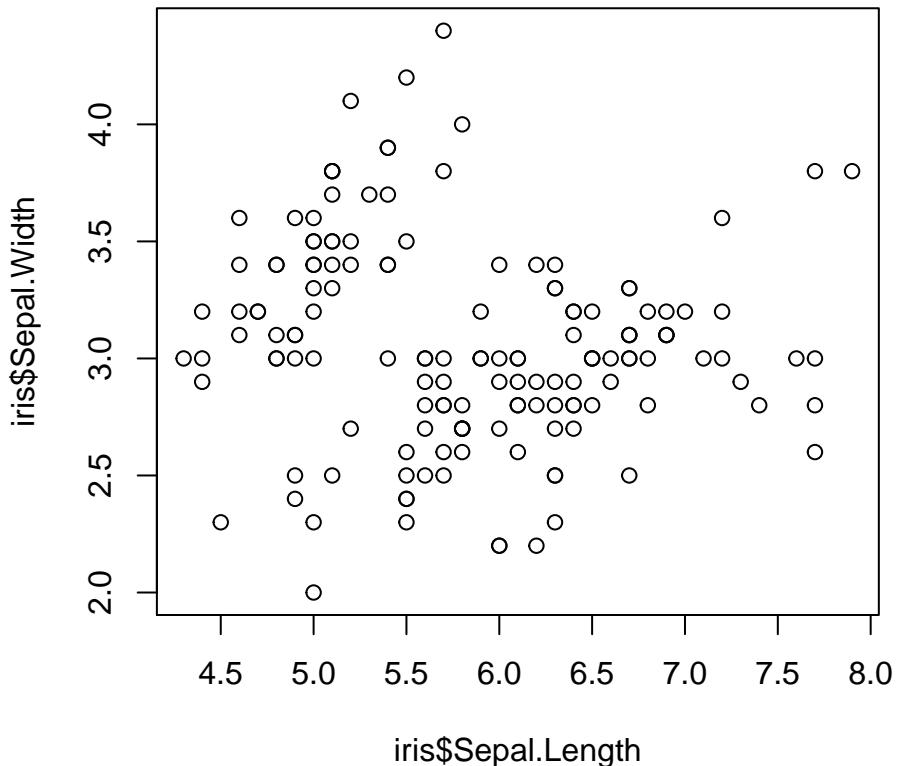
```
summary(iris)
```

```
##   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
##   Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##   Median :5.800   Median :3.000   Median :4.350   Median :1.300
##   Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##   Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##   Species
##   setosa   :50
```

```
##  versicolor:50
##  virginica :50
##
##
```

Forestil, at vi gerne vil lave et plot, som viser sammenhængen mellem længden og bredden af sepal (bægerblad), eller specifikt er vi interesseret i kolonnerne `iris$Sepal.Length` og `iris$Sepal.Width`. Lad os starte med at visualisere variablerne i base-R, ved at bruge `plot`:

```
plot(iris$Sepal.Length, iris$Sepal.Width)
```



Man kan gøre det meget pænere eksempelvis ved at bruge forskellige farver til at betegne de forskellige arter, eller ved at give en hensigtsmæssig overskrift eller aksenavne.

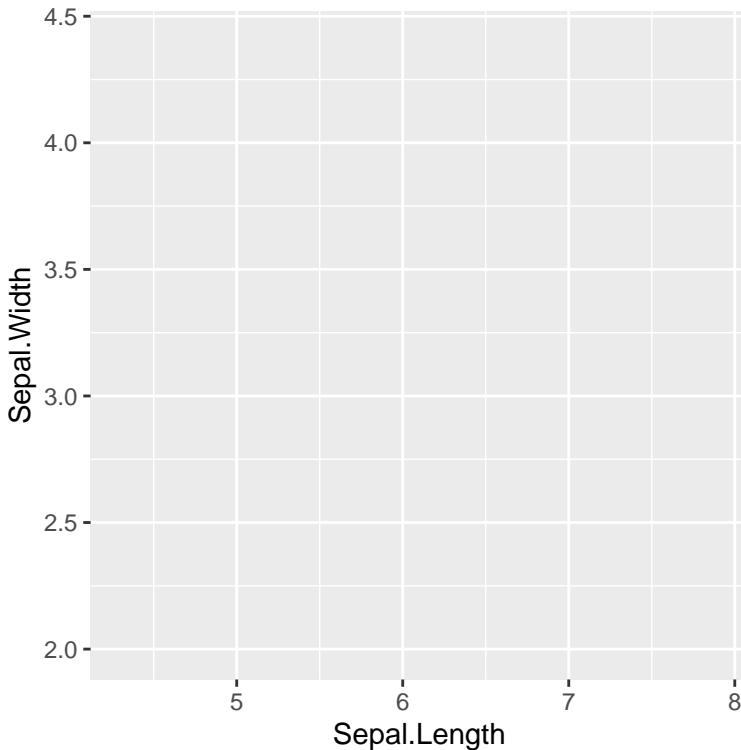
### 3.3 Vores første ggplot

Vi vil imidlertid fokusere på at lave et lignende plot med pakken `ggplot2`. Hvis man ikke allerede har gjort det, så husk at indlæse pakken i R for at få nedenstående koder til at virke.

```
#install.packages("ggplot2") #hvis ikke allerede installeret
library(ggplot2)
```

For at lave et plot med `ggplot2` tager man altid udgangspunkt i funktionen `ggplot()`. Først specificerer vi vores data - altså at vi gerne vil bruge dataframe `iris`. Dernæst angiver vi indenfor funktionen `aes()` (som sidder indenfor `ggplot()`), at x-aksen skal være `Sepal.Length` og y-aksen `Sepal.Width`. Det ser sådan ud:

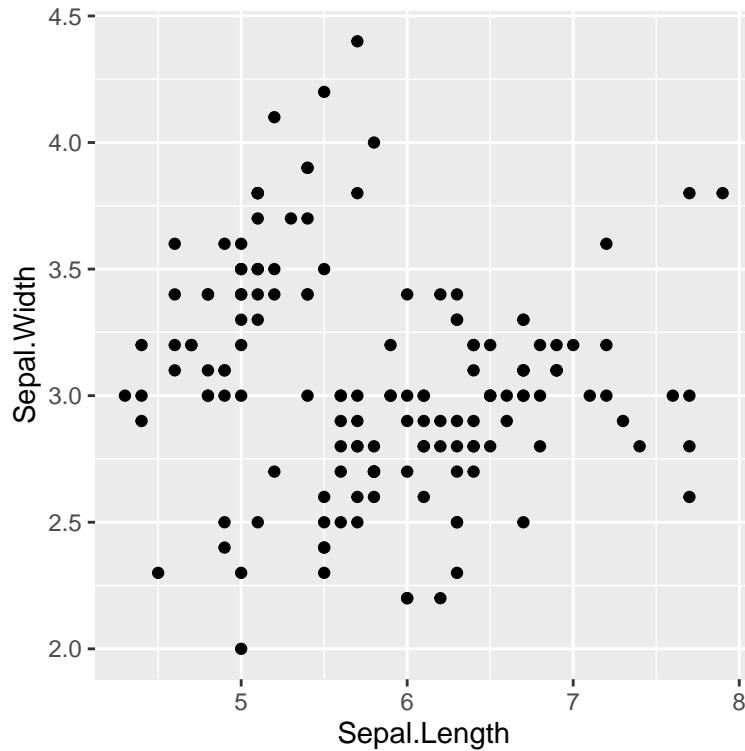
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width))
```



Koden fungerer, men bemærk at plottet er helt blank og derfor ikke særligt brugbart. Men der er blevet lavet et grundlag (se aksenenavne osv.). Det er blank fordi vi endnu ikke har fortalt, hvilken plot type det skal være - for eksempel søjlediagram/barplot, histogram, punktplot/scatter plot (jeg vælge de engelske begreber herfra for at skabe den bedste sammenhæng med kodden). Vi vil gerne bruge et scatter plot, som i `ggplot2` er angivet af funktionen `geom_point()`. Vi forbinder derfor funktionen `geom_point()` til den `ggplot()` funktion, vi allerede har specificeret. Husk altid, at man bruger `+` til at forbinde de to "komponenter" (altså `ggplot()` og `geom_point()`) af plottet (ellers få vi fortsat et blank plot).

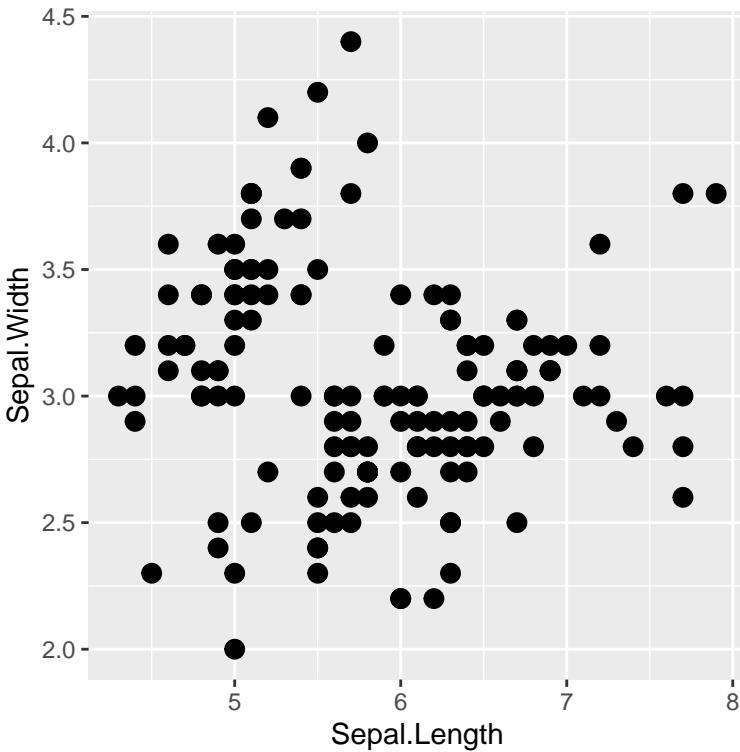
Koden er således:

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point()
```



Bemærk, at vi ikke har skrevet noget indeni de runde parenteser i funktionen `geom_point()`. Det betyder, at vi accepterer alle standard eller ‘default’ parametre, som funktionen tager. Hvis vi vil have noget andet end de standard parametre, kan vi godt specificere det. For eksempel kan vi gøre punkterne lidt større end ved standard (prøve at tjekke `?geom_point()` for at se en liste overfor de mulige parametre, man kan justere på):

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point(size=3)
```



Vi har nu et plot, som vi kan sammenligne med det ovenstående plot, vi lavet i base-pakken. Ligesom i base-pakken vil vi gerne tilføje nogle ting for at gøre vores plot til vores *færdige figur*.. Her i **ggplot2** gøres det ved at tilføje flere komponenter ovenpå, med brugen af `+`, ligesom vi gjorde da vi tilføjede `geom_point()` til `ggplot()`. Første vil jeg gerne skrive nogle ord om **ggplot2** generelt, og filosofien bag.

## 3.4 Lidt om ggplot2

### 3.4.1 Syntax

Som vi har lige set, `ggplot()` tager altid udgangspunkt i en dataframe, som vi specificerer først. I `ggplot()` indeholder den dataframe variablerne vi skal bruge til at få lavet figuren. Til at gøre det til noget mere konkret, lad os sammenligne koden mellem base-pakken og `ggplot()` til vores `iris` data. I base-R angav vi direkte vektorer `iris$Sepal.Length` og `iris$Sepal.Width` som parametre `x` og `y`, der tager henholdsvis første og andens-plads i funktionen `plot()`. Til gengæld i `ggplot()`, specificerer man først den hele dataramme i den første plads, og så bagefter med brugen af `aes()` angav vi hvordan x-aksen og y-aksen ser ud.

```
#baseplot solution
plot(iris$Sepal.Length, iris$Sepal.Width)

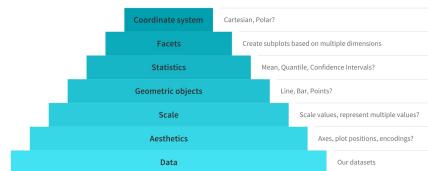
#ggplot2 solution
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point()
```

En anden fordel af `ggplot2()` er, at man kan blive ved med at forbedre plottet ved at tilføje ting ovenpå det plot, som vi allerede har lavet, i hvad man kan beskrive som en lagring tilgang. Det gøres intuitiv ved brugen af “+”. Man kan derfor starte med noget simpelt, og derefter opbygge det til noget mere kompleks. Dette er uafhængig af den type plot, vi laver.

### 3.4.2 Hvad betyder egentlig grammar of graphics?

Den `gg` i `ggplot2` står for *grammar of graphics*, og filosofien er at der skal defineres en *sætningsstruktur* til de figurer, man laver. Med andre ord består vores figur af forskellige komponenter, som man forbinder med “+”..

Major Components of the Grammar of Graphics



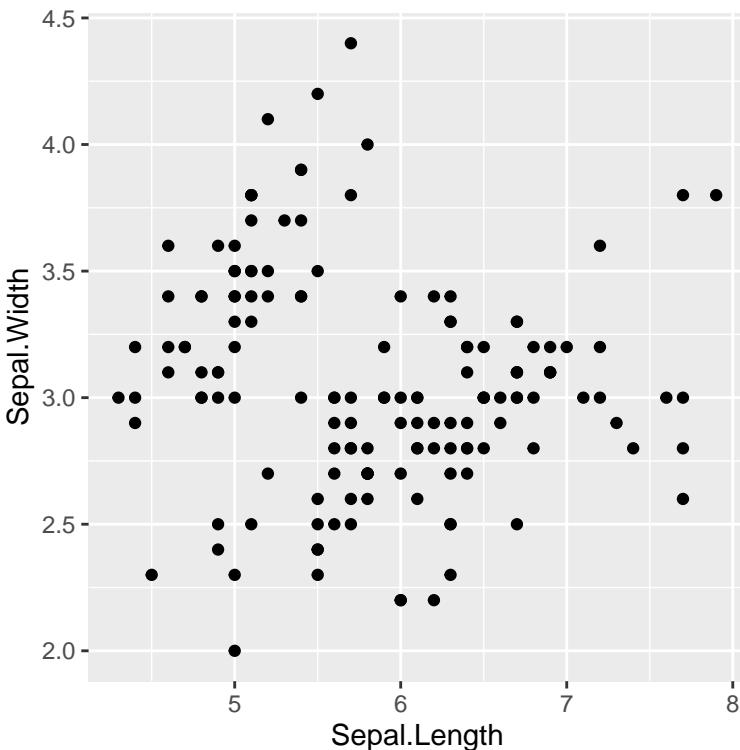
Her er en beskrivelse af de forskellige komponenter til at opbygge et plot:

- Data: Datarammer tager altid udgangspunkt
- Aesthetics: Variable til x-aksen eller y-aksen, farve, form eller størrelse
- Scale: Scalere værdier eller repræsentere flere værdier
- Geometries: Eller `geoms` - hvad type plot skal vi lave - fk. bars, points, lines osv.
- Statistics: Tilføj fk. mean, median, quartile som beskrive data
- Facets: Lave subplots baserende på flere dimensioner
- Coordinate system: Transformerer akser, ændrer afstanden for de viste data

### 3.4.3 Globale versus lokale æstetik

De fleste tilfældede bruger vil funktionen `aes()` indenfor `ggplot()`, med betydningen, at variablerne specifiseret indenfor `aes()` gælder globalt over alle komponenter i plottet. Man kan faktisk også skrive `aes()` lokale indenfor selve `geom` funktion, som i følgende:

```
ggplot(iris) +
  geom_point(aes(x=Sepal.Length, y=Sepal.Width))
```

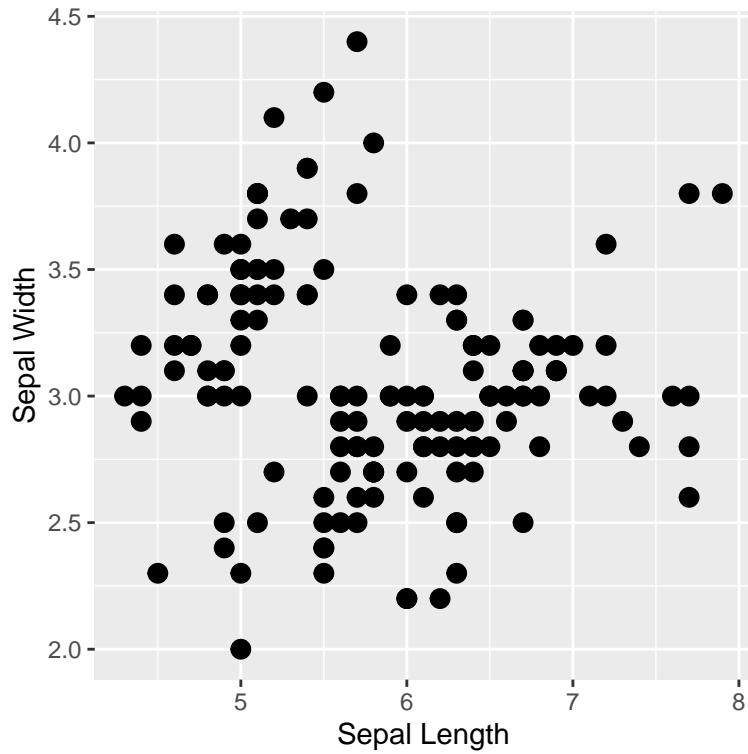


Vi får det samme plot som før, men det er kun `geom_point()` der er påvirket af specificeringen indenfor `aes()`. I simpel situationer som ovenpå er der ingen forskel, men når man har mange forskellige komponenter i spil, så kan det nogle gange give mening at bruge lokale æstetik.

## 3.5 Specificere etiketter og titel

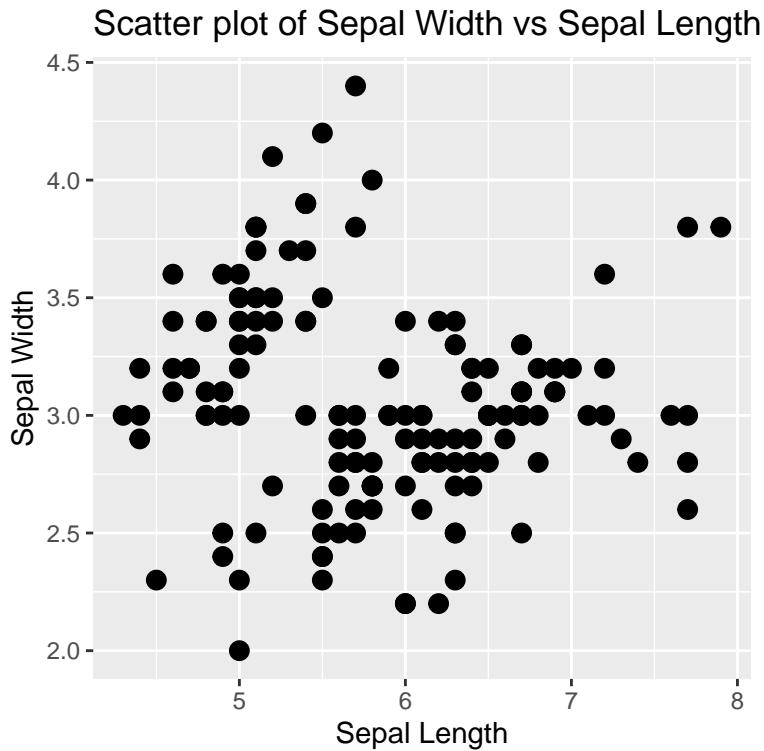
Vi tager udgangspunkt i plottet, vi lavet i ovenstående og prøver at gøre det bedre ved at tilføje nye etiketter og en titel. I `ggplot` kan man opdatere y-akse og x-akse etiketter ved at bruge henholdsvis `ylab` og `xlab`:

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point(size=3) +
  xlab("Sepal Length") +
  ylab("Sepal Width")
```



Vi tilføjer en titel med funktionen `ggtitle()`:

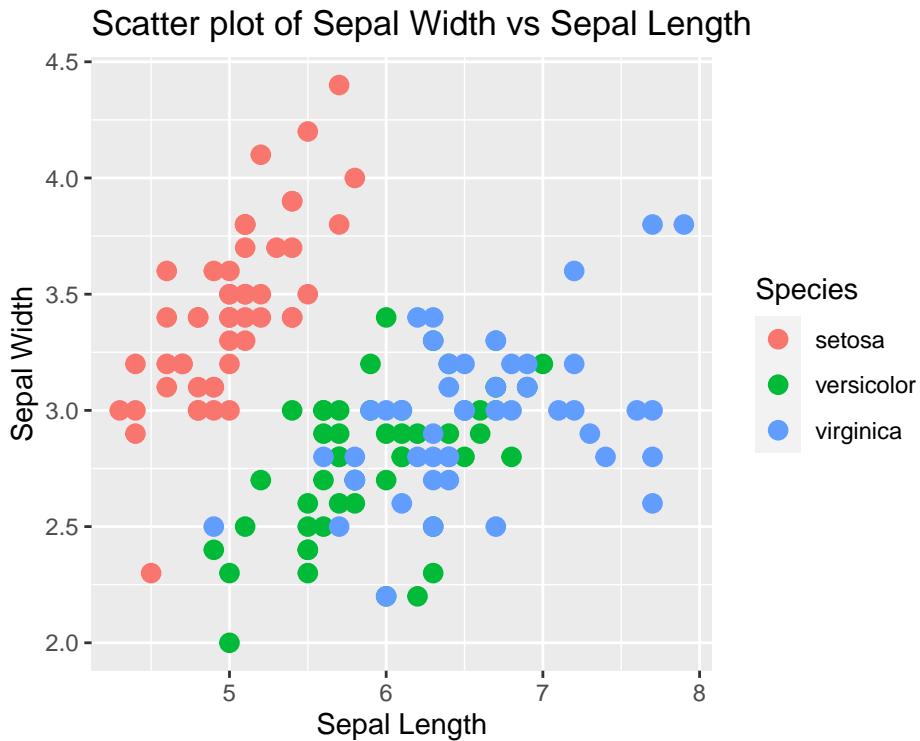
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point(size=3) +  
  xlab("Sepal Length") +  
  ylab("Sepal Width") +  
  ggtitle("Scatter plot of Sepal Width vs Sepal Length")
```



### 3.6 Ændre farver

I `ggplot2` kan man bruge "automatisk" farver for at skelne imellem de tre forskellige `Species` i datasættet `iris`. I næste lektion dækker jeg hvordan man kan være mere fleksibel ved at sætte farver manuelt, men ofte vil vi bare bruge som udgangspunkt den nemme løsning og eventuelle rette op på det bagefter med en ny komponent, hvis der er behov for det. Vi skriver `color=Species` indenfor `aes()`, som i følgende. Bemærk, at der kommer en 'legend' med, der fortæller os, hvilken art få hvilken farve.

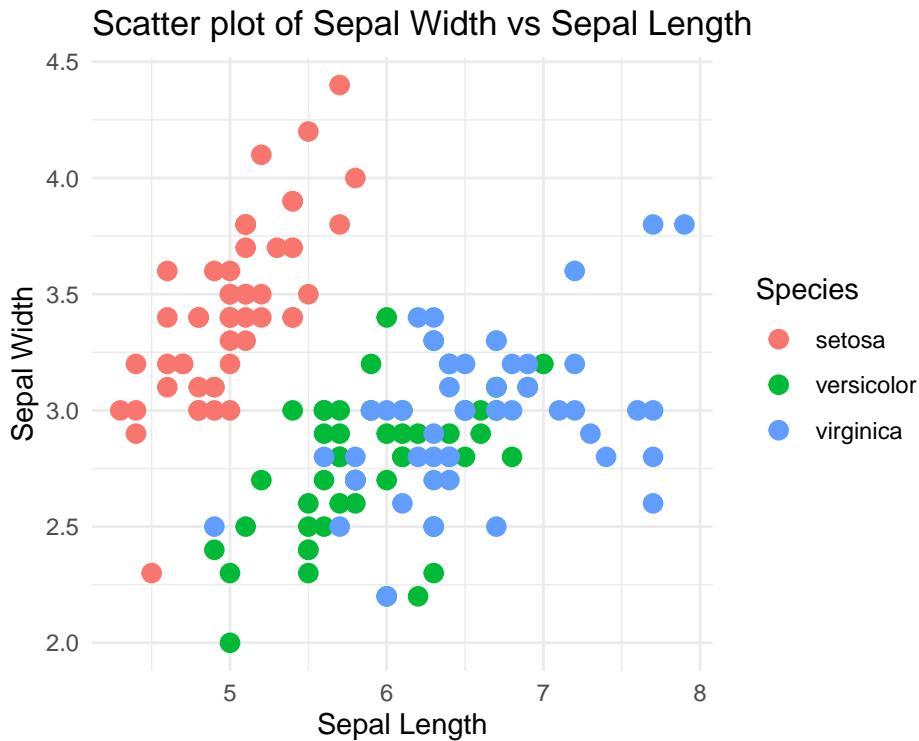
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size=3) +
  xlab("Sepal Length") +
  ylab("Sepal Width") +
  ggtitle("Scatter plot of Sepal Width vs Sepal Length")
```



### 3.7 ÅEndre tema

Det standard tema har en grå baggrund og “grid” linjer, men vi kan godt vælge noget andet. For eksempel kan man tilføje `theme_minimal()` som i nedenstående. Her får vi en hvid baggrund i stedet for, mens vi får stadig grid linjer. Man kan afprøve forskellige temae (for eksempel `theme_classic()`, `theme_bw()`), og se, hvilket tema fungerer bedst i det enkelt plot.

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size=3) +
  xlab("Sepal Length") +
  ylab("Sepal Width") +
  ggtitle("Scatter plot of Sepal Width vs Sepal Length") +
  theme_minimal()
```



Her er nogle eksempler på mulige temaeer du kan bruge i dine plotter (det er dog generelt op til dig).

|                 |
|-----------------|
| tema            |
| theme_grey()    |
| theme_classic() |
| theme_bw()      |
| theme_dark()    |
| theme_minimal() |
| theme_light()   |

Se også her hvis du er interesseret i flere temaeer: <https://r-charts.com/ggplot2/themes/>

## 3.8 Forskellige geoms

Indtil videre har vi kun arbejdet med `geom_point()` for at lave et scatter plot, men det kan være at vi gerne vil lave noget andet. Her gennemgår jeg følgende “geoms”:

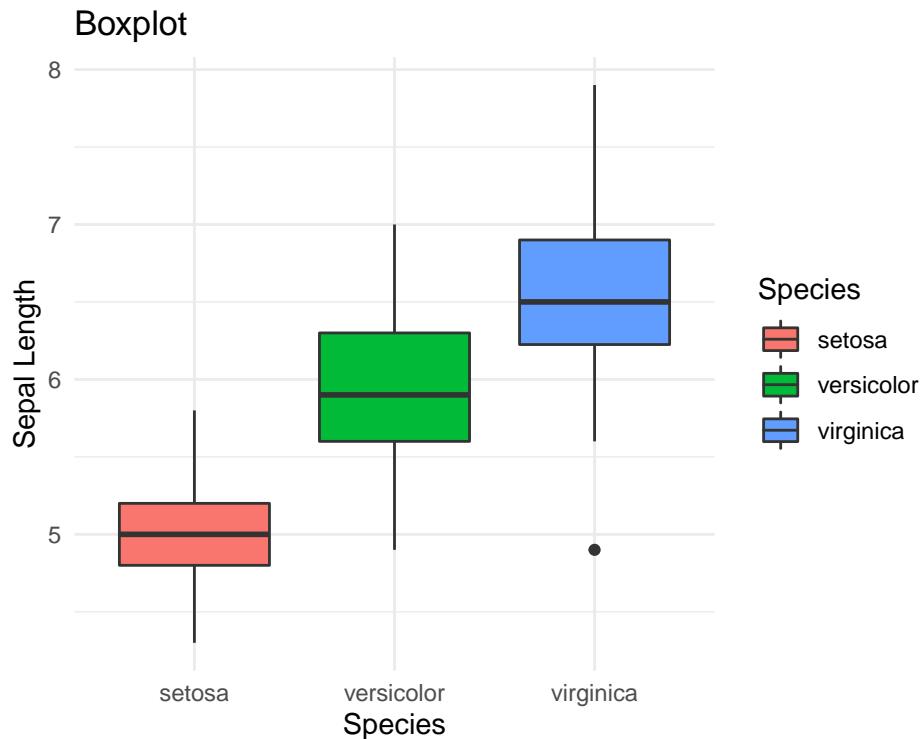
| geom                          | plot         |
|-------------------------------|--------------|
| <code>geom_point()</code>     | scatter plot |
| <code>geom_bar()</code>       | barplot      |
| <code>geom_boxplot()</code>   | boxplot      |
| <code>geom_histogram()</code> | histogram    |
| <code>geom_density()</code>   | density      |

For at lave disse geoms, skal man tillægge funktionen til den `ggplot()` kommando med `+`, ligesom vi gjorde med `geom_point()`. Der kan dog være for nogle plot-typer specifikke overvejelser, som er værd at vide inden man selv bruger dem.

### 3.8.1 Boxplot (`geom_box`)

For at lave et boxplot af `Sepal.Length` opdelt efter `Species`, angiver vi `Species` på x-aksen og `Sepal.Length` på y-aksen. Vi vil også have, at hver art få sin egen farve, så bruger vi `fill=Species`.

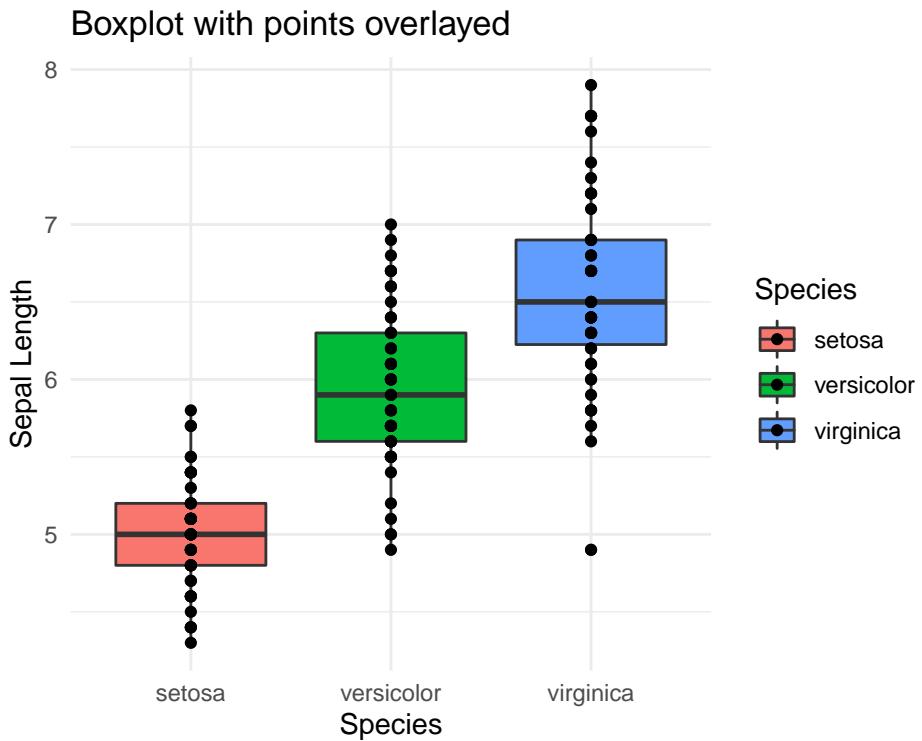
```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  ylab("Sepal Length") +
  ggtitle("Boxplot") +
  theme_minimal()
```



### Lave punkter ovenpå

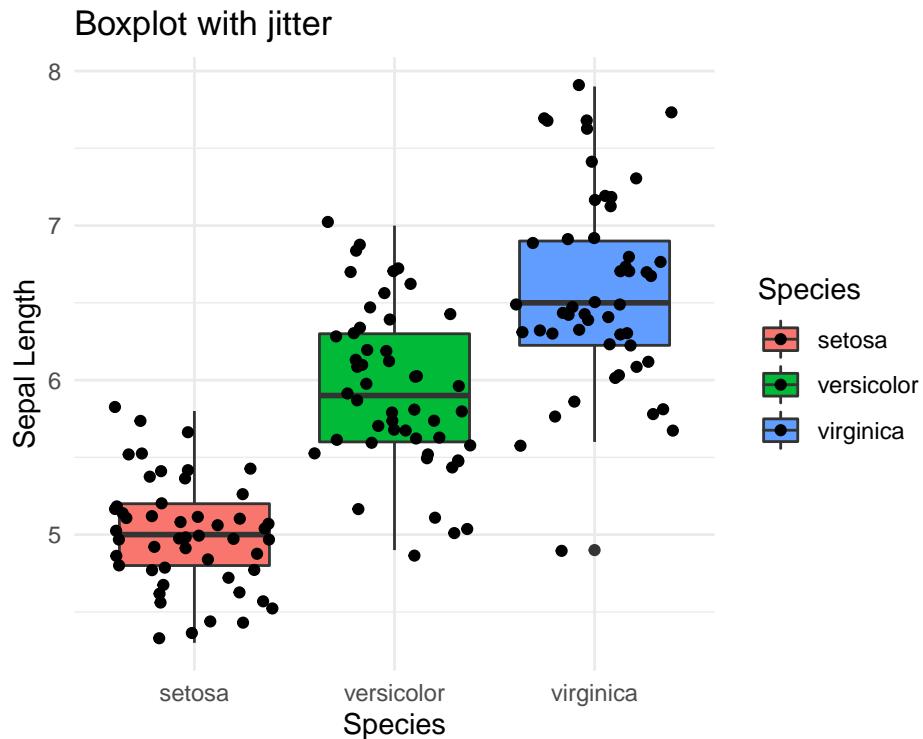
Det kan ofte være nyttigt at plotte de egentlige data punkter ovenpå boxplottet, så I kan se både fordelingen i de data samt de rå data. En løsning er at benytte `geom_point()` ved at tilføje det som komponent over vores eksisterende kode.

```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  geom_point() +
  ylab("Sepal Length") +
  ggtitle("Boxplot with points overlaid") +
  theme_minimal()
```



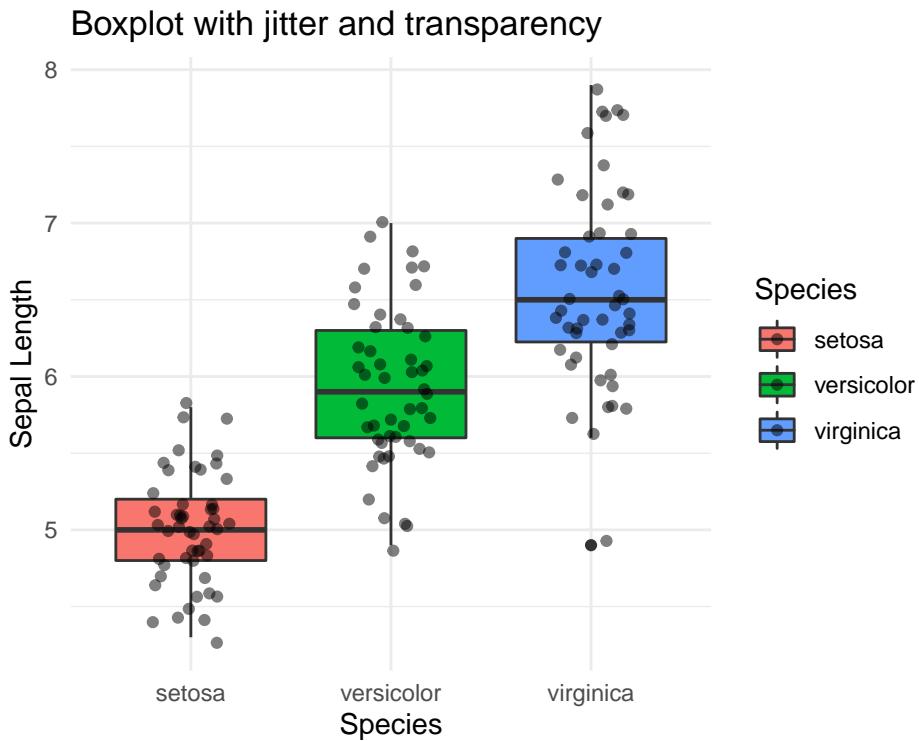
Man kan dog se, at det ikke er særlig informativ, da alle punkter er på den samme lodrette linje. Hvis vi har mange punkter med samme eller næsten samme værdier, så kan vi ikke se de fleste af dem i plottet. En bedre løsning er at indføre noget tilfældighed i punkterne langt x-aksen, så at man mere tydelige kan se dem. Det er kaldes for "jitter" og man specificerer jitter ved at bruge `geom_jitter` i stedet for `geom_point`.

```
gplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  geom_jitter() +
  ylab("Sepal Length") +
  ggtitle("Boxplot with jitter") +
  theme_minimal()
```



Jeg kan vi også specificere `alpha`, som indføre gøre punkterne gennemsigtige, for at gøre dem mindre markant. Man kan også ændre på `width` som kontrollerer deres spredning langt x-axsen.

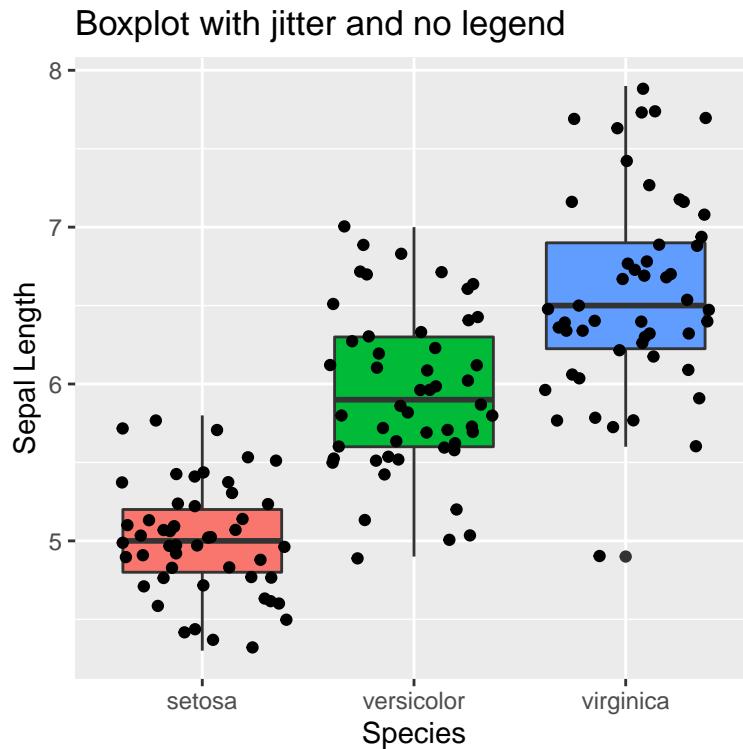
```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  geom_jitter(alpha=0.5, width=0.2) +
  ylab("Sepal Length") +
  ggtitle("Boxplot with jitter and transparency") +
  theme_minimal()
```



Fjerne legend hvis unødvendige

Man kan se, at når man specificerer farver, få man en legend på højre side af plotte. I dette tilfælde er det faktisk ikke nødvendige, da man kan se uden legend hvad de tre boxplots refererer til. Defor fjerner vi den fra plottet ved at bruge `theme(legend.position="none")`.

```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  geom_jitter() +
  ylab("Sepal Length") +
  ggtitle("Boxplot with jitter and no legend") +
  theme(legend.position="none")
```

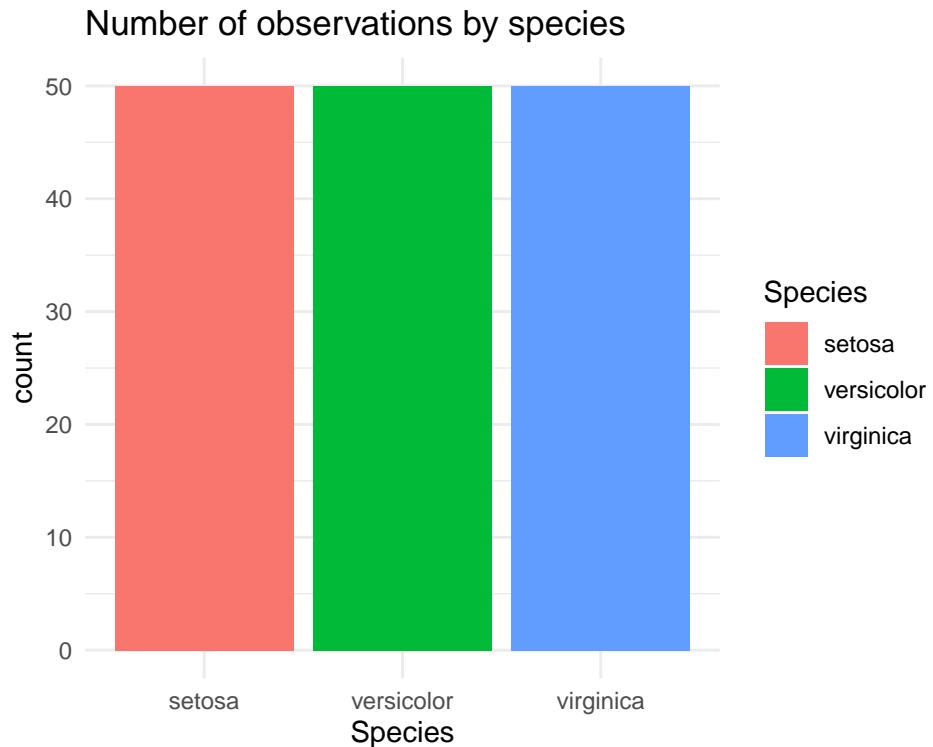


### 3.8.2 Barplot (geom\_bar)

Med `ggplot()` kan man representere data i et bar plot ved at bruge `geom_bar()`. I følgende vil vi gerne tælle op de antal observationer for hver art (variable `Species`), og visualiser dem således som søjler. Indenfor `geom_bar()` specificerer vi således `stat="count"`.

Vi bruger også `fill=Species` her for at lave en forskellige farve automatiske for hver af de tre arter. Bemærk, at det var `color=Species` i det forudgående plot når vi anvendte `geom_point()`. Det er fordi, `color` bruges for punkter og linjer, mens `fill` er til større regioner bliver udfyldt, såsom bars og histograms.

```
ggplot(iris, aes(x=Species, fill=Species)) +
  geom_bar(stat = "count") +
  ggtitle("Number of observations by species") +
  theme_minimal()
```



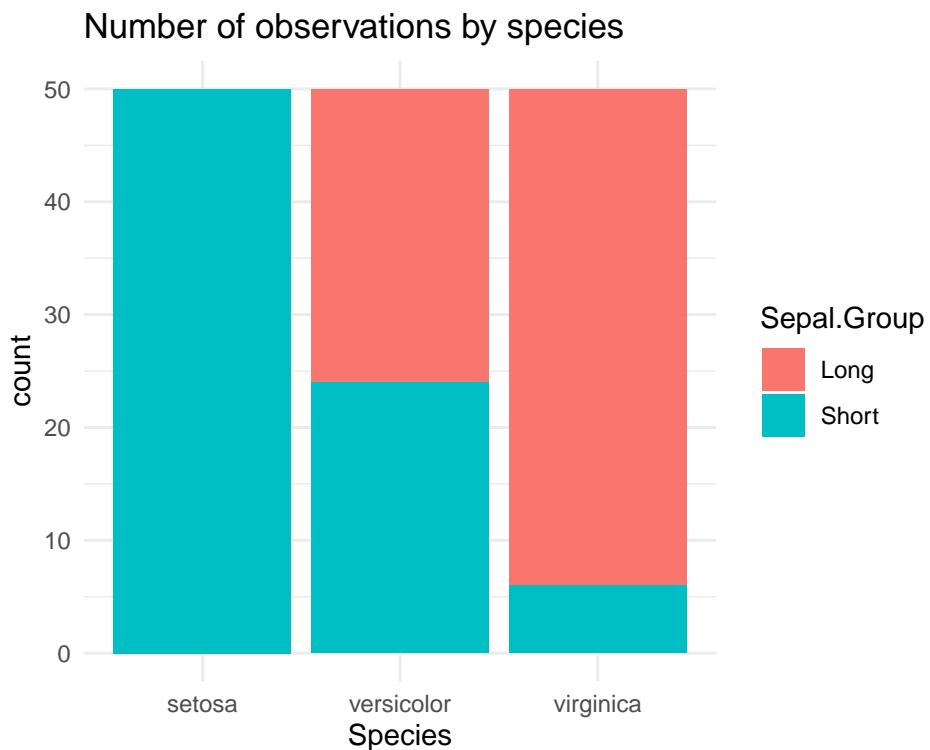
#### Barplot: stack vs dodge

Hvis man har flere katagoriske variabler, kan man lave barplots på forskellige måder. Da der er en ekstra katagorisk variabel i datasættet, laver jeg én, der hedder `Sepal.Group`, der skelne imellem `Long` og `Short` værdier af variablen `Sepal.Length`. Her specificerer jeg bare (med funktionen `ifelse()`), at hvis `Sepal.Length` er længere end den gennemsnitlige `Sepal.Length`, så er det betragtet `Long`, ellers er det `Short`, som i følgende:

```
iris$Sepal.Group <-
  ifelse(iris$Sepal.Length>mean(iris$Sepal.Length), #test
        "Long",                                         #if TRUE
        "Short")                                       #if FALSE
```

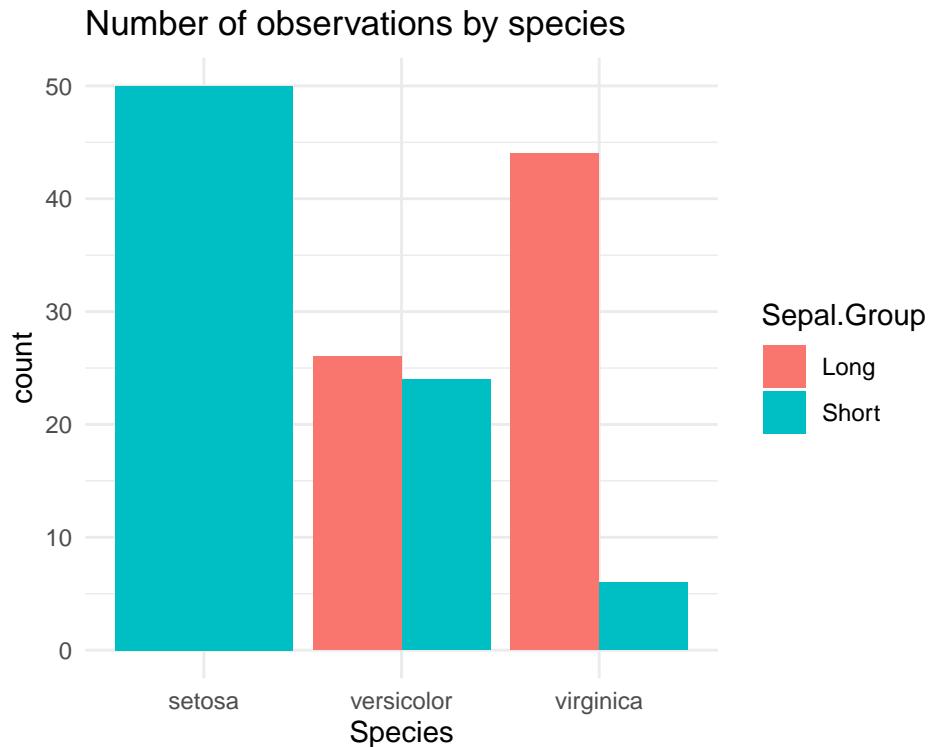
Når jeg laver en barplot med de to variabler, tilføjer jeg `Sepal.Group` med `fill`, og `ggplot` splitter antal observationer efter `Sepal.Group` med farver som repræsenterer `Sepal.Group`, og tilføjer en tilsvarende legend.

```
ggplot(iris, aes(x=Species, fill=Sepal.Group)) +
  geom_bar(stat = "count") +
  ggtitle("Number of observations by species") +
  theme_minimal()
```



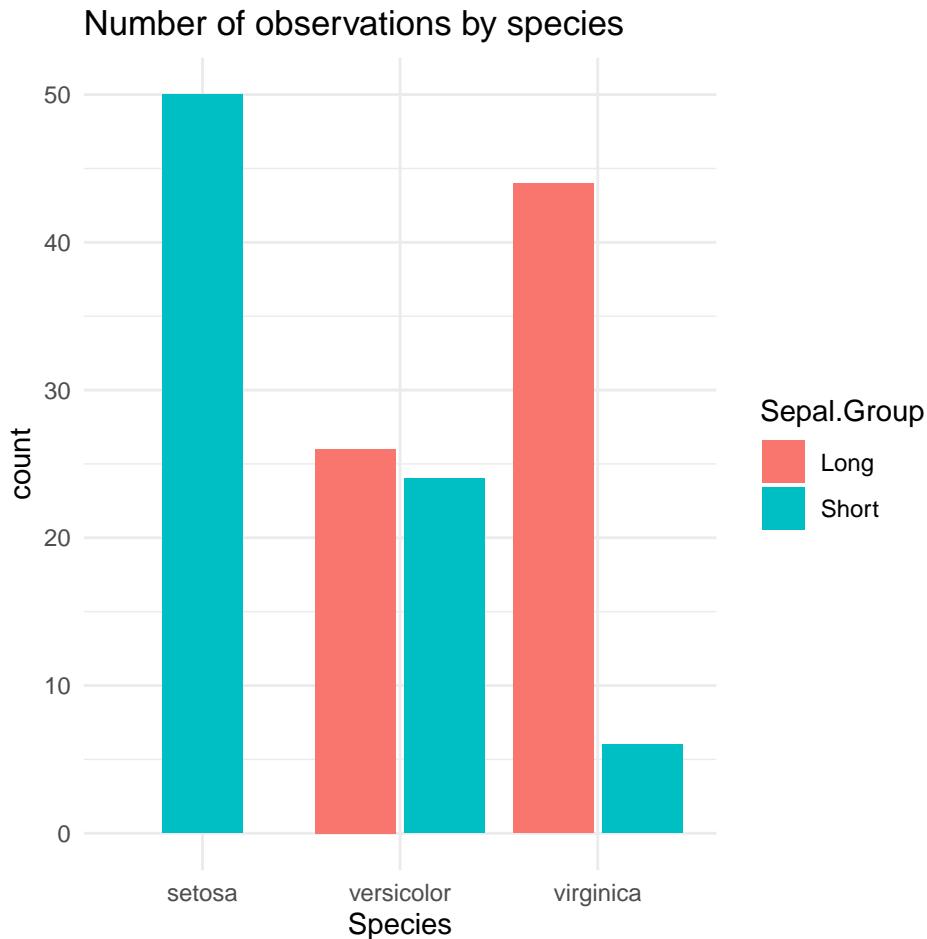
Mange gange foretrækker man at få bars som står ved siden af hinanden. Det kan vi specifcere med blot at tilføje `position = "dodge"` ind i `geom_bar()`.

```
ggplot(iris, aes(x=Species, fill=Sepal.Group)) +
  geom_bar(stat = "count", position = "dodge") +
  ggtitle("Number of observations by species") +
  theme_minimal()
```



Som ekstra for at vise fleksibiliten i pakken `ggplot2`: jeg kan ikke lide at bredden af søjlen til arten `setosa` i ovenstående plot har bredden som er to gange større end de andre søjler (fordi der er ingen observationer i `setosa` som har "Long" i variablen `Sepal.Group`). Jeg Gogglede mig frem til en løsning på det:

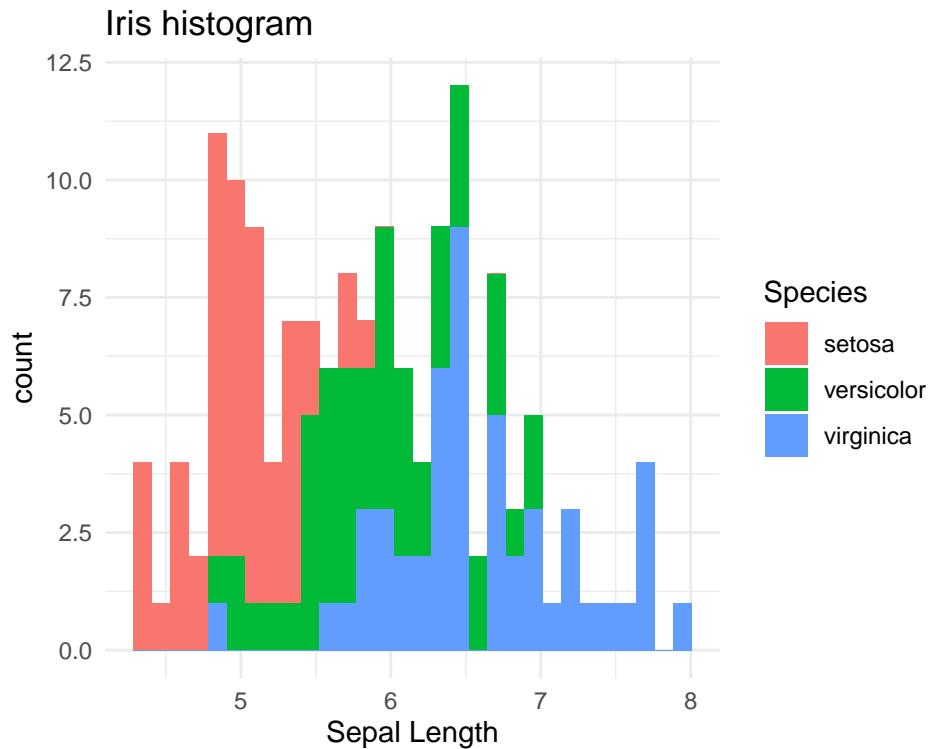
```
ggplot(iris, aes(x=Species, fill=Sepal.Group)) +
  geom_bar(stat = "count", position = position_dodge2(preserve = "single")) +
  ggtitle("Number of observations by species") +
  theme_minimal()
```



### 3.8.3 Histogram (geom\_histogram)

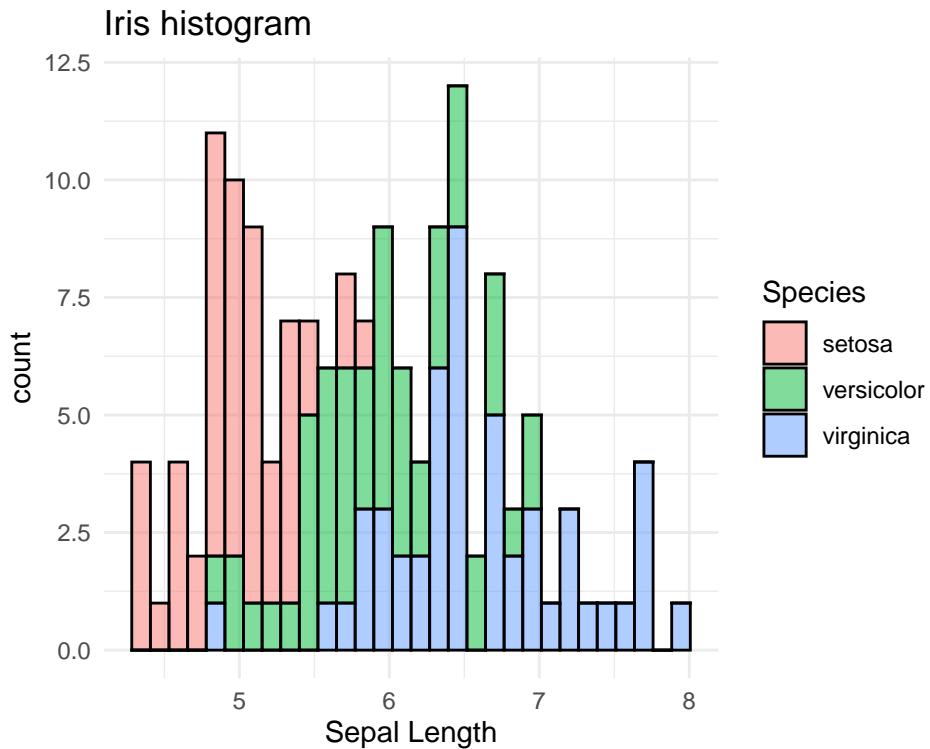
En histogram bruges til at få overblik over hvordan data fordeler sig. I `ggplot2` kan man lave en histogram med `geom_histogram()`. Den x-akse variabel skal være en ‘continuous’ variabel. Her specificerer vi, at vi gerne vil have et histogram for hver Species.

```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram() +
  xlab("Sepal Length") +
  ggtitle("Iris histogram") +
  theme_minimal()
```



Man kan også gøre det nemmere at skelne imellem de tre arter ved at sætte `alpha=0.5` indenfor `geom_histogram` og ved at angive en linje farve som mulighed inden for `geom_histogram()`.

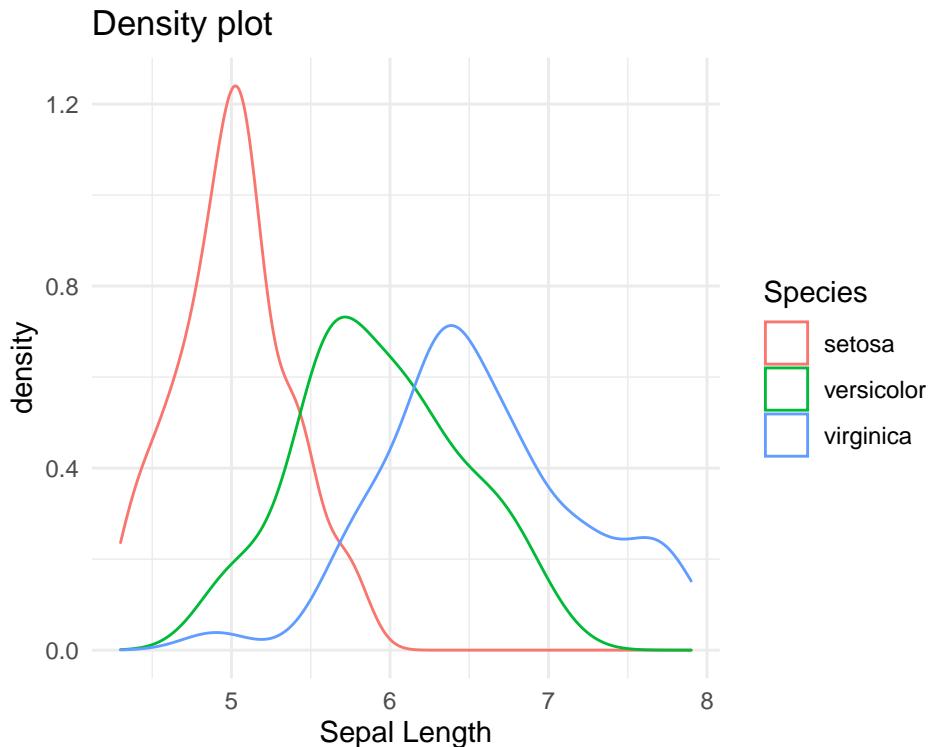
```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram(alpha=0.5, color="black") +
  xlab("Sepal Length") +
  ggtitle("Iris histogram") +
  theme_minimal()
```



### 3.8.4 Density (geom\_density)

Med en density plot, ligesom med en histogram, kan man se fordelingen, de data har, i formen af en glat (eller “smooth”) kurv.

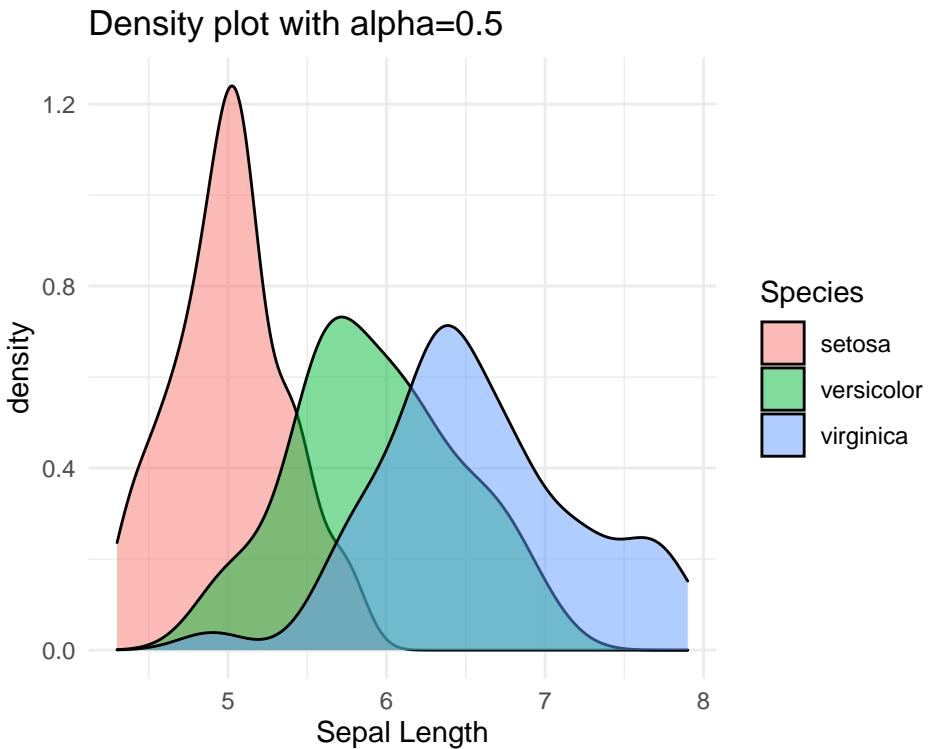
```
ggplot(data=iris, aes(x=Sepal.Length, color=Species)) +
  geom_density() +
  xlab("Sepal Length") +
  ggtitle("Density plot") +
  theme_minimal()
```



### Density plot med fill og gennemsigtig farver

Vi kan angive en værdi for `alpha` indenfor `geom_density()`. Den parameter `alpha` specificerer gennemsigtigheden af de density kurver i plottet.

```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_density(alpha=0.5) +
  xlab("Sepal Length") +
  ggtitle("Density plot with alpha=0.5") +
  theme_minimal()
```

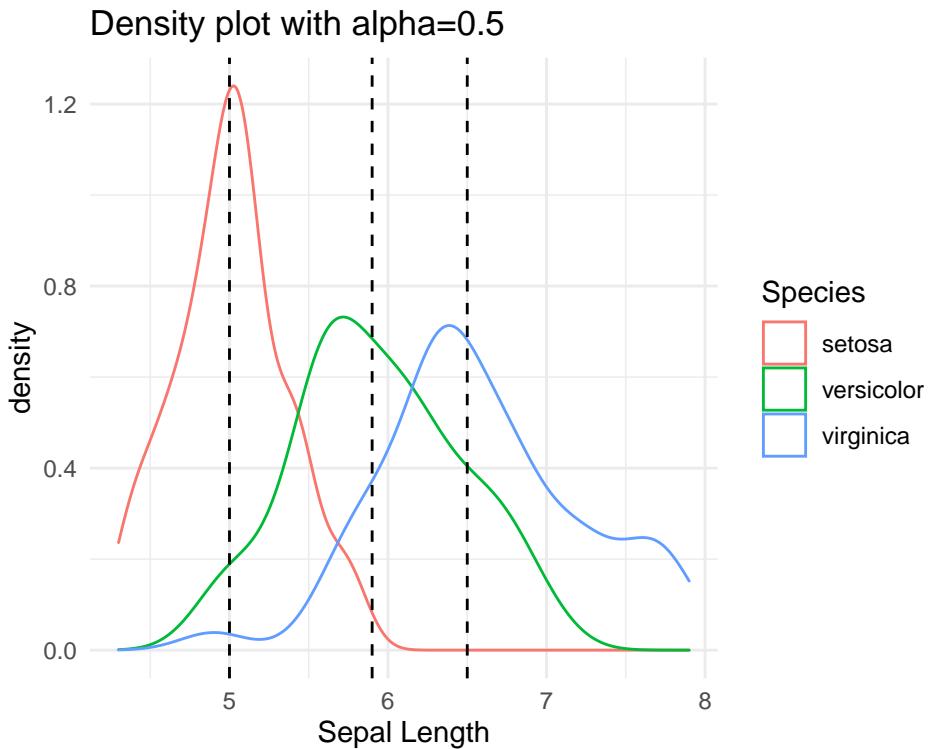


### Tilføje middelværdi linjer

Vi bruger funktionen `tapply()` til at beregne middelværdier af `Sepal.Length` for hver af de tre `Species`. Vi kan derefter tilføje dem som lodrette linjer til vores plot. Her bruger vi `geom_vline()` (OBS det er `geom_hline()` hvis man vil have en vandret linje) og fortæller, at `xintercept` skal være lig med de middelværdier, som vi har beregnet. Parameteren `lty=2` betyder, at vi vil gerne have en “dashed” linje.

```
means <- tapply(iris$Sepal.Length, iris$Species, median)

ggplot(data=iris, aes(x=Sepal.Length, color=Species)) +
  geom_density(alpha=0.5) +
  xlab("Sepal Length") +
  ggtitle("Density plot with alpha=0.5") +
  geom_vline(xintercept = means, lty=2) +
  theme_minimal()
```



### 3.9 Troubleshooting

Her er en lille liste over nogle ting, der forårsager en fejl når man kører kode med **ggplot2**. Jeg tilføjer også andre ting som opstår i vores lektion :).

- `ggplot(data=iris, aes(...))` : husk her `data=iris` er korrekt og ikke `Data=iris` (R skelner mellem store og små bogstaver). Man kan også undlade at bruge `data=` og skrive bare `iris` i stedet for.
- Forkert stavning - dobbelt tjek, at du har stavet variabler eller funktioner navne korrekt.
- Glemte + symbol - for at forbinde komponenterne i plottet, skal man huske at tilføje `+` i slutningen af en linje og så skrive de næste komponent bagefter (man behøver ikke at skrive hver komponent på en ny linje med det gøre det nemmere at læse koden).
- Skrev `%>%` symbol i stedet for `+` - de øvrige pakker fra **tidyverse** bruger `%>%`.
- Glemte parentes: her har man glemt den sidste parentes: skal være `fill=Species))` og ikke `fill=Species)`. Man får bare en `+` fordi R forventer at du fortsætter med at skrive mere kode.

```
> ggplot(data=iris, aes(x=Sepal.Length, fill=Species)
+
```

- **fill** og **colour** - indenfor **aes()** refererer **fill** til at man fylder fl. bars eller regioner med farver, og **colour** referere til farven af linjer eller punkter.

## 3.10 Problemstillinger

1) Quiz på Absalon - den hedder Quiz – `ggplot2 part 1`.

*OBS: Husk at lave følgende øvelser i R Markdown. Det er god praksis at sikre, at jeres dokument knitter - i selve eksamen afleverer du et html dokument.*

- Lav et nyt R Markdown dokument og fjern de eksempel koder. Husk at oprette en ny chunk ved at trykke på “Insert” new chunk”, eller bruge shortcut-kommandoen **CMD+ALT+I** eller **CTRL+ALT+I**. Jeg anbefaler, at I oprette en ny chunk for hver plot, I laver.

Vi bruger datasættet der hedder **diamonds**. Huske at først indlæse de data:\*

```
data(diamonds)
```

Her er beskrivelsen af **diamonds**:

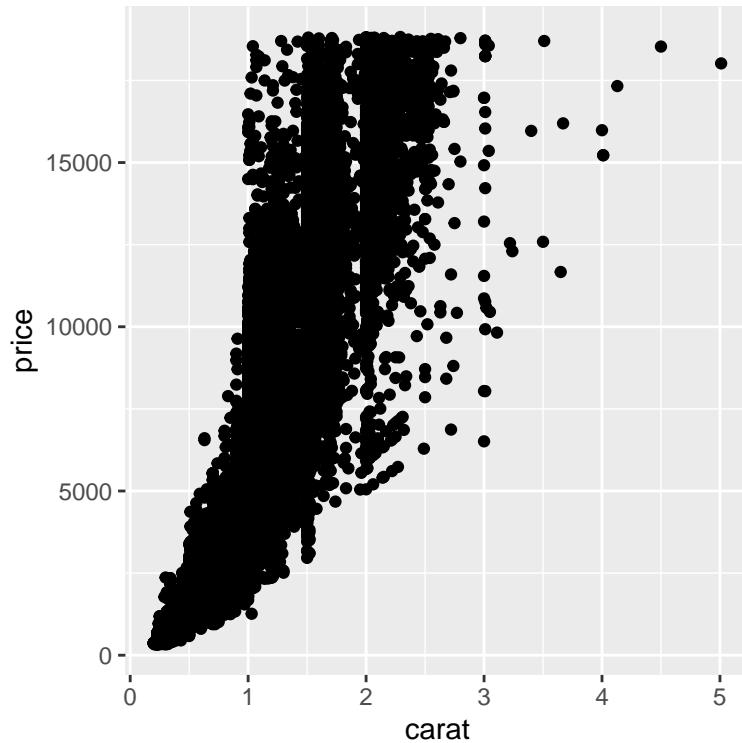
*Prices of over 50,000 round cut diamonds: a dataset containing the prices and other attributes of almost 54,000 diamonds.*

Se også `?diamonds` for en beskrivelse af de variabler.

2) Bruge datasættet **diamonds** til at lave et scatter plot (`geom_point()`):

- **carat** på x-aksen
- **price** på y-aksen

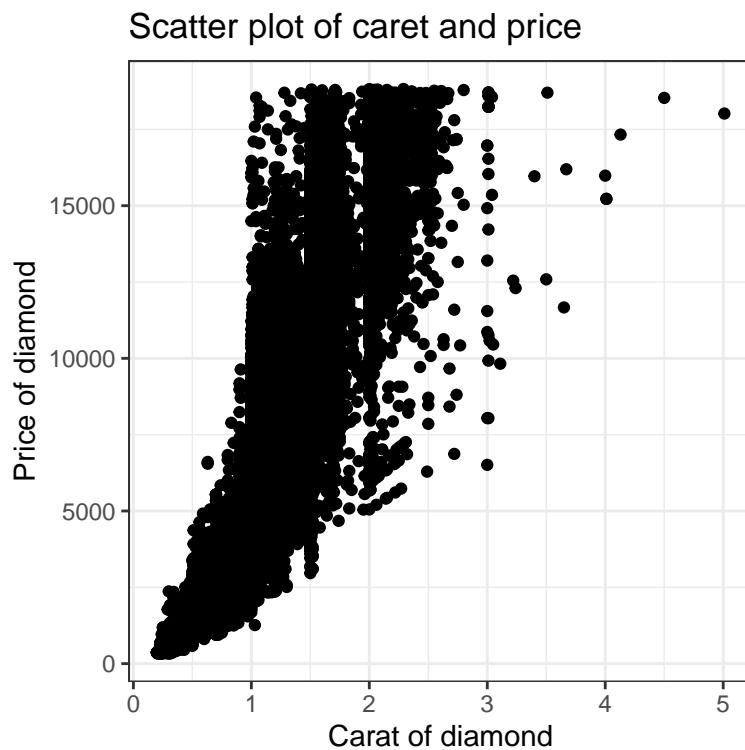
Så at du har noget at sammenligne med, skal dit plot se sådan ud:



3) Tilføj nogle komponenter til dit plot fra 2).

- En x-akse label (`xlab()`) og en y-akse label (`ylab()`)
- En titel (`ggtitle()`)
- Et tema som hedder `theme_bw()`
- Husk at forbinde komponenterne med + og skrive de nye komponenter på deres egen linje.

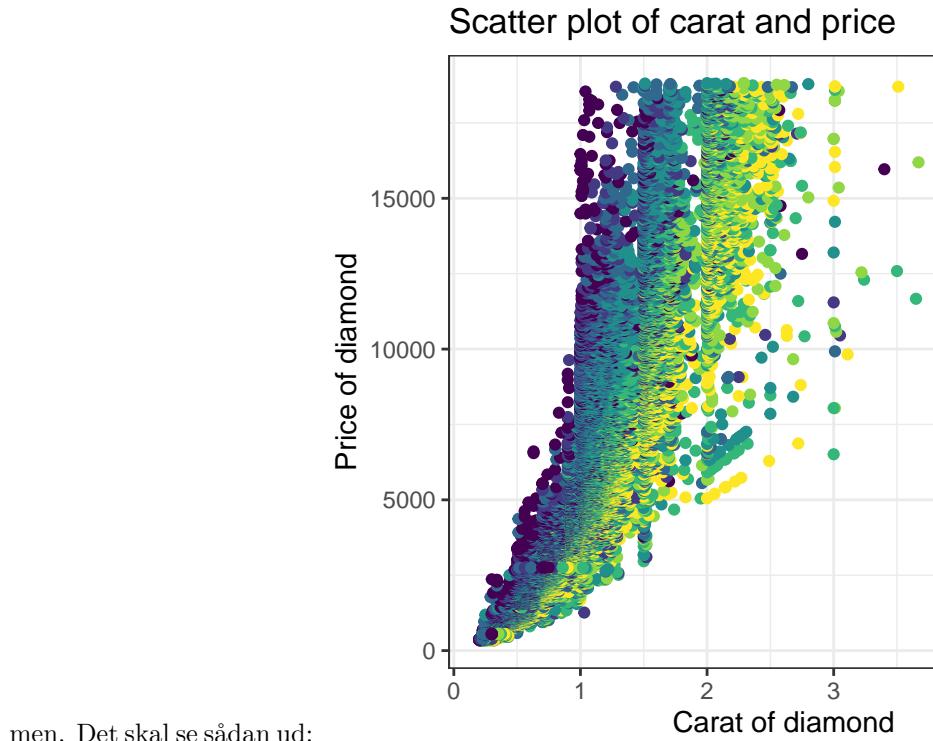
Det skal se sådan ud:



4) Ændre temaet af dit plot til `theme_classic()` eller `theme_minimal()` i stedet for `theme_bw()` og kig på resultatet.

- Hvis man (måske ved uheld) skriver ind `to` temaeer på samme tid (for eksempel `+ theme_bw() + theme_classic()`) - hvilket tema får man så i plottet?
- Valgfri ekstra: her er nogle flere tema man kan prøve: [https://ggplot2.tidyverse.org/reference/ggtheme.html\\*](https://ggplot2.tidyverse.org/reference/ggtheme.html)

5) Lav det samme plot som i 3), og skriv `color=color` indenfor `aes()`. Den første `color` refererer til punkt farver og den anden til variablen `color` i datafra-

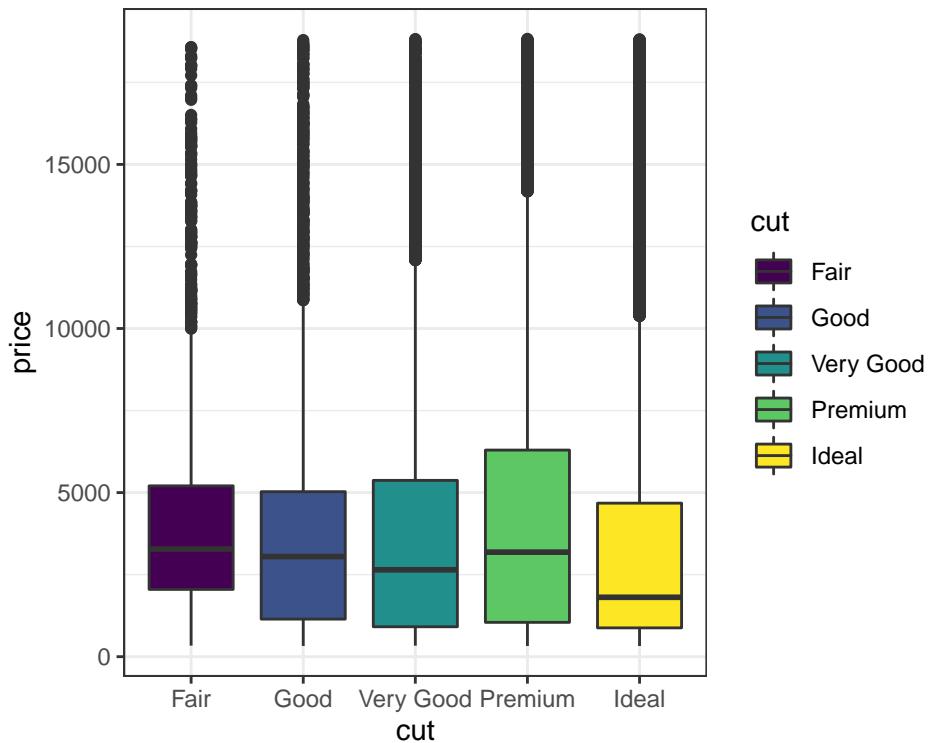


- Nu fjern `color=color` fra funktionen `aes()` og i stedet tilføj `aes(color=color)` i funktionen `geom_point()`. Får du samme resultat?
- Bemærk at det er lige meget om man bruger britisk eller amerikansk stavning i ggplot2 - fl. `colour` eller `color` indenfor `aes()` giiver samme resultat.

6) Brug stadig `diamonds`, til at lave et boxplot:

- `cut` på x-aksen (giv x-aksen label `Cut`)
- `price` på y-aksen (giv y-aksen label `Price of diamond`)
- bruge `fill` til at give forskellige farver til de mulige værdier af `cut`.
- bruge temaet `theme_bw()`

Det skal se sådan ud:

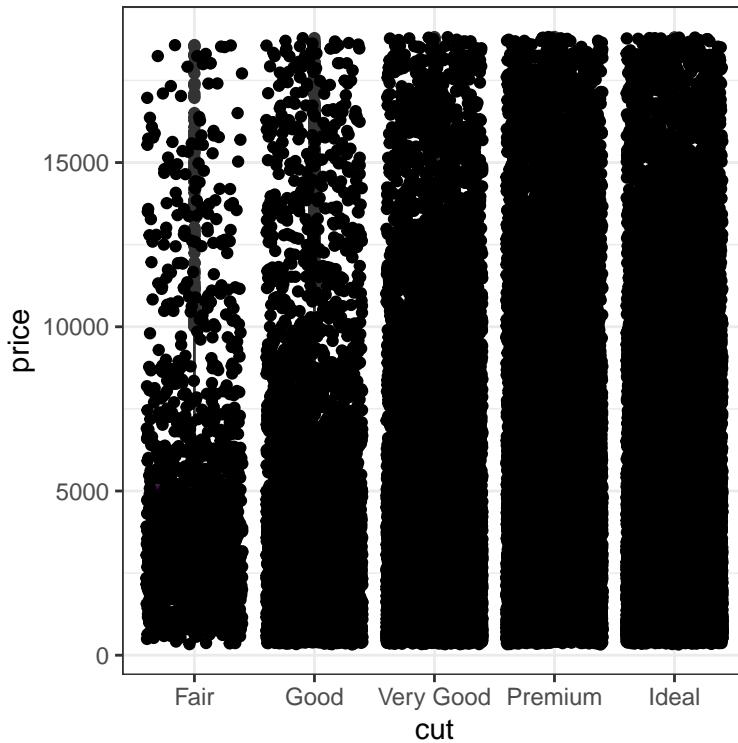


- Hvordan ser det ud, hvis man bruger `colour` i stedet for `fill`? Eller hvis man giver begge to?

7) Lav følgende ekstra ændringer til din boxplot fra ovenstående:

- Tilføj `geom_jitter()` til din boxplot
- fjern legend ved at tilføj `theme(legend.position="none")`
- Man kan også tilføj `show.legend=FALSE` til både `geom_boxplot()` og `geom_jitter()` i stedet for - prøv det i stedet for at bruge `theme(legend.position="none")`. Er det nok at tilføje `show.legend=FALSE` til kun én af de to geoms?

Det skal se sådan ud:



- Man kan også prøve at forbedre plottet ved at give nogle indstillinger ind i `geom_jitter()`, for eksempel kan man prøve `geom_jitter(size=.2,color="grey",alpha=0.5)` for at gøre punkter mindre overbelastende i plottet (eller kan man overveje at fjerne dem).

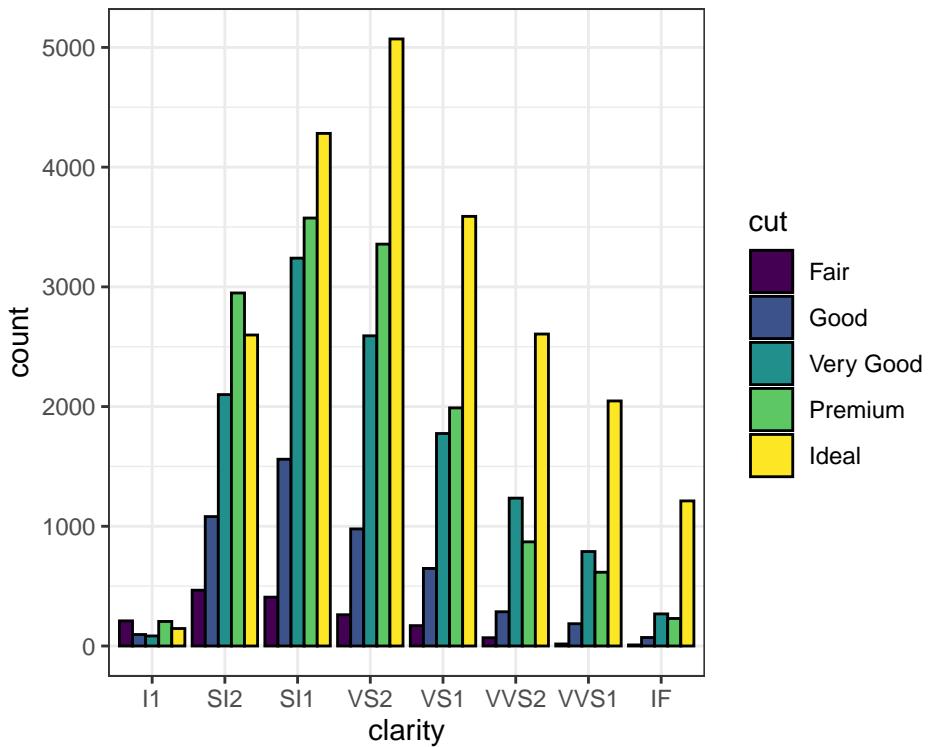
Leg med de tre indstilling `size`, `color` og `alpha` og ser på forskellen. Her er en note om `alpha`:

*Alpha refers to the opacity of a geom. Values of alpha range from 0 to 1, with lower values corresponding to more transparent colors.* [https://ggplot2.tidyverse.org/reference/aes\\_colour\\_fill\\_alpha.html](https://ggplot2.tidyverse.org/reference/aes_colour_fill_alpha.html)

- Prøv at skifte rækkefølgerne af `geom_jitter()` og `geom_boxplot()` i dit plot kommando og se - gøre det en forskel til hvordan plottet ser ud?

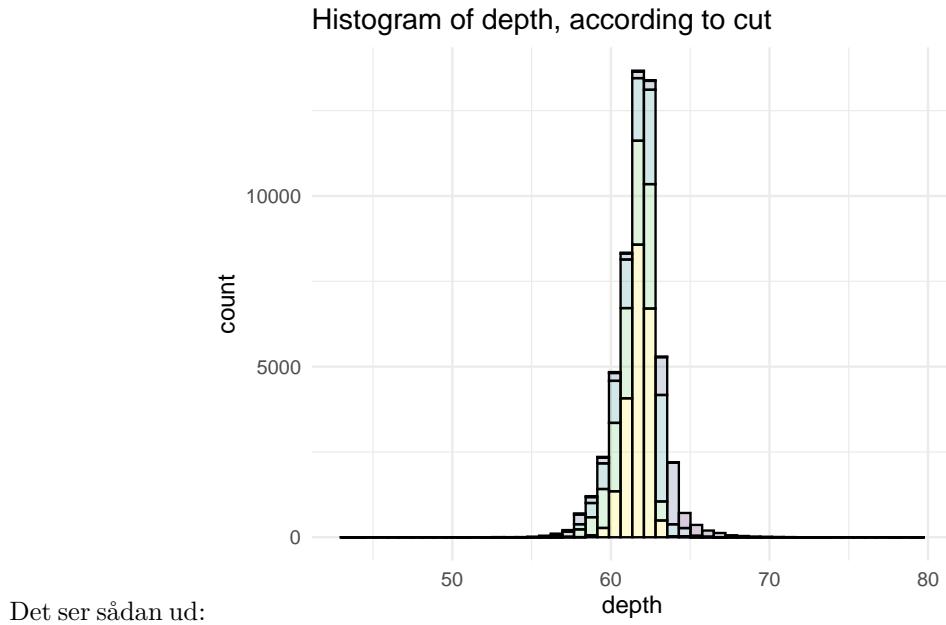
8) Lav en barplot med indstillingen `stat="count"`:

- Variable `clarity` på x-aksen
- Forskellige farver til den gruppe-variable `cut`
- Specifier `position="dodge"` for at få bars ved siden af hinanden
- Brug også indstillingen `colour="black"` og notater effekten
- Tilføj et tema



9) Lav en histogram

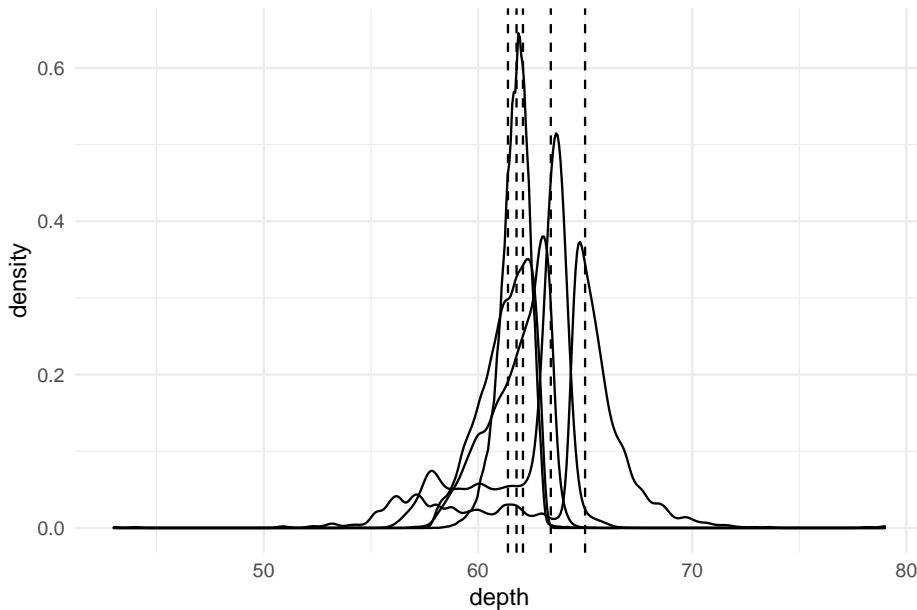
- Variable `depth` på x-aksen
- Forskellige farver efter `cut`
- Brug indstilling `alpha` til at ændre gennemsigtigheden af sørjerne
- Giv søjlerne en sort ramme
- Tilføj et tema osv.



- Nu får du en advarsel - gør hvad advarselen siger og ændre på indstillingen `bins` indenfor `geom_histogram()`.

### 10) Lav en density

- Man kan se, at det er svært at sammenligne fordelingerne i ovenstående histograms
- I din histogram kode fra 9) erstattede `geom_histogram` med `geom_density`
- Er det nu nemmere at sammenligne fordelingerne efter de forskellige niveauer af `cut`?
- Tilføj lodrette linjer med beregnede `median` værdier af variablen `depth` for hver af de cuts fra variablen `cut`.
  - Hint: `tapply` til at beregne `median`, `geom_vline` til at lave lodrette linjer



11) Bare ekstra øvelse: Lege frit med at lave andre plots fra diamonds med ggplot2. Eksempelvis

- Boxplots med `carat` opdelt efter `clarity`
- Barplots for de forskellige farver (variable `color`)
- Et scatter plot af `depth` vs `price`.

I alle tilfælde tilføje akse-labs, en titel, et tema osv.

### 3.11 Næste gang

Efter at have lavet de problemstillinger skal man kunne se, at der er rigtig meget fleksibilitet involveret med at lave et plot med `ggplot2`. I morgen går vi videre med andre plot typer, og hvordan man fk. sætte farver manuelt.



## Chapter 4

# Visualisering - ggplot2 dag 2



### 4.1 Indledning og videoer

I nuværende emne udvider du værktøjskassen af kommandoer i pakken **ggplot2** for at tillade større fleksibilitet og appel i dine visualiseringer. Jeg anbefaler, at du ser videoerne inden undervisningstimerne og bruger notaterne som en slags

reference samtidig at du arbejder med problemstillingerne.

### 4.1.1 Læringsmålene

I skal være i stand til at:

- Arbejde fleksibelt med koordinat systemer - transformering, modificering og “flipping” af x- og y-aksen.
- Udvide brugen af farver og form.
- Tilføje tekst direkte på plottet med `geom_text()`.
- Bruge `facet_grid()` eller `facet_wrap()` til at adskille plots efter en katagorisk variabel.
- Gemme dit færdigt plot i en fil.

```
library(ggplot2) #husk
```

### 4.1.2 Video ressourcer

- Video 1: Koordinat systemer (2021)

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/544201985>

- Video 2: Farver og punkt former (2021)

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/544218153>

- Video 3: Labels - `geom_text()` og `geom_text_repel()` (2021)

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/544226498>

- Video 4 - Facets

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/704140333>

## 4.2 Koordinat systemer

Her arbejder vi videre med koordinater i pakken `ggplot2`.

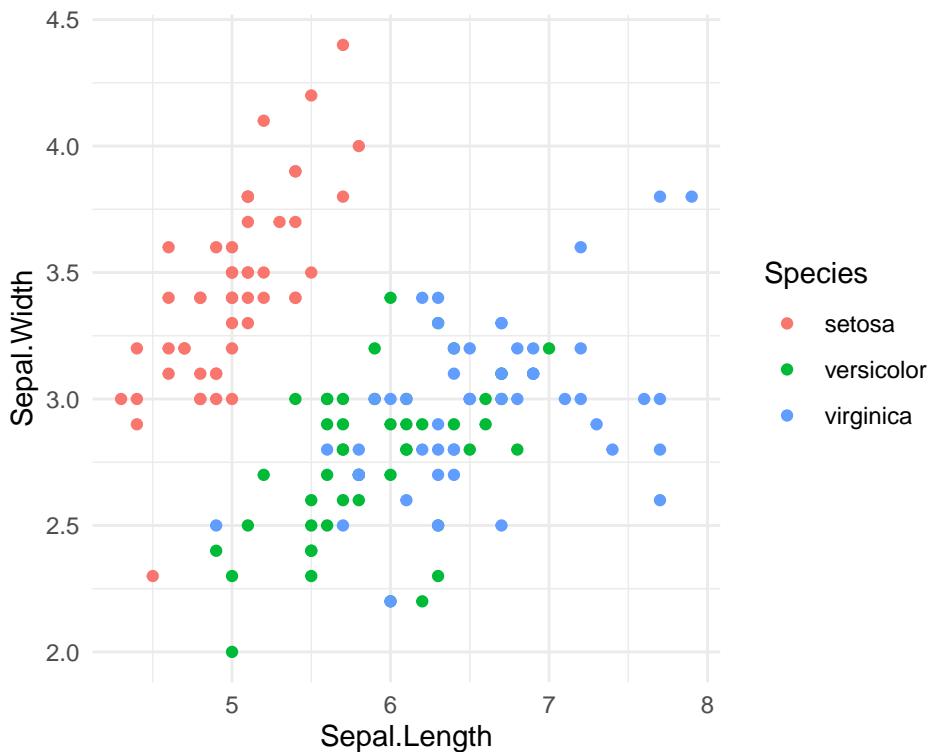
### 4.2.1 Zoom (`coord_cartesian()`, `expand_limits()`)

Man kan bruge funktionen `coord_cartesian()` til at zoome ind på et bestemt område i plottet. **Indenfor** `coord_cartesian()` angives `xlim()` og `ylim()`, som specificerer de øvre og nedre grænser langt henholdsvis x-aksen og y-aksen. Man kan også bruge `xlim()` og `ylim()` udenom `coord_cartesian()`, men i dette tilfælde bliver punkterne, som ikke kan ses i plottet (fordi deres koordinater ligger udenfor de angivne grænser), smidt væk (med en advarsel). Med

`coord_cartesian()` beholder man til gengæld samtlige data, og man får således ikke en advarsel.

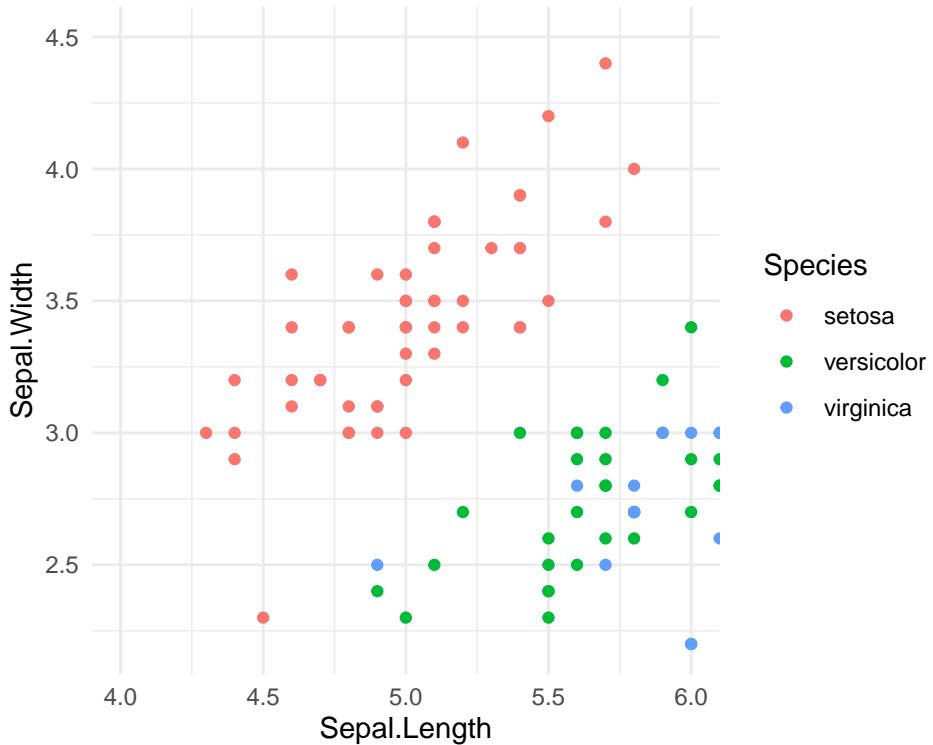
I følgende ses vores oprindeligt scatter plot:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() +
  theme_minimal()
```



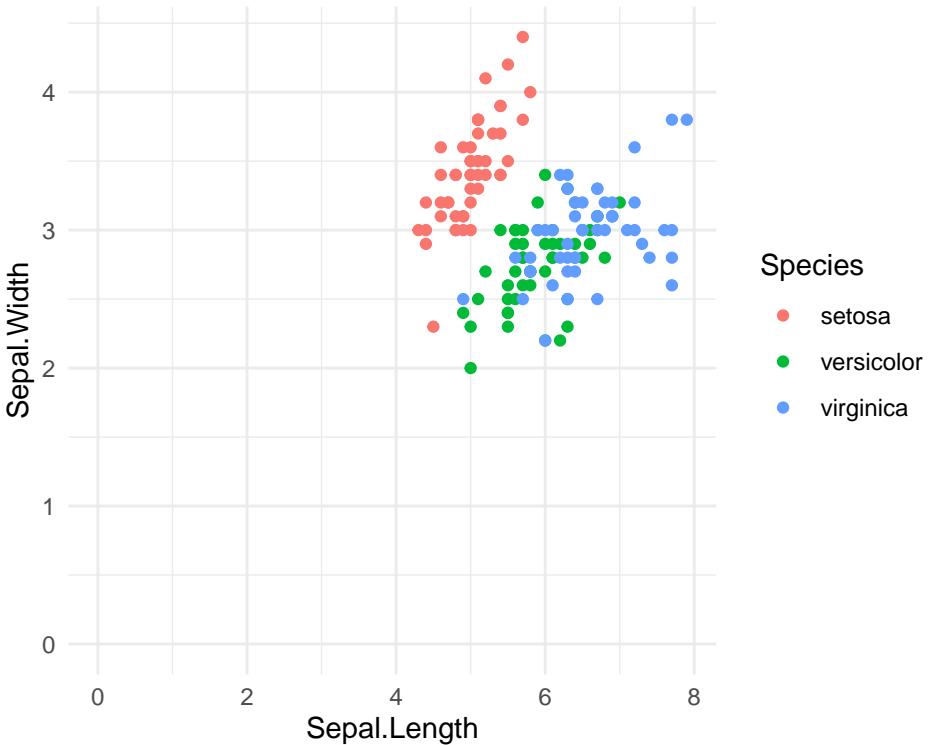
Og her anvender jeg funktionen `coord_cartesian()` med `xlim()` og `ylim()` indenfor til at zoome ind på et ønsket område på plottet.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() +
  coord_cartesian(xlim = c(4,6), ylim = c(2.2,4.5)) +
  theme_minimal()
```



Du kan også zoome ud ved at bruge `expand_limits()`. For eksempel hvis jeg gerne vil have punkterne  $x = 0$  og  $y = 0$  (`c(0,0)`, eller “origin”) med i selve plottet:

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point() +
  expand_limits(x = 0, y = 0) +
  theme_minimal()
```

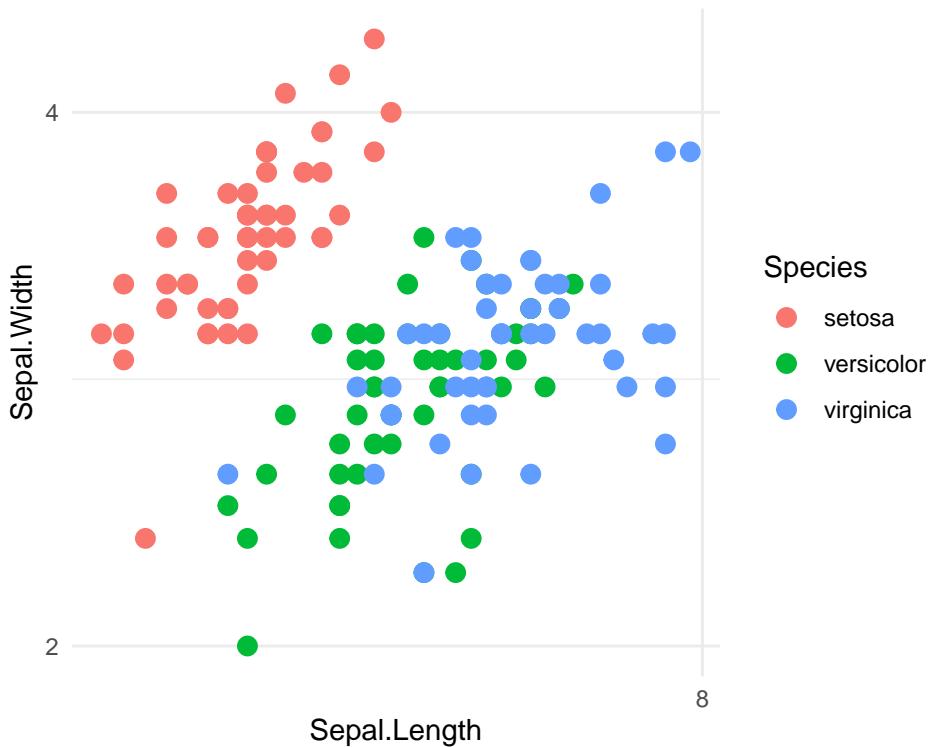


Det kan være brubart i situationer hvor man, eksempelvis har flere etiketter omkring punkterne i selve plottet, som bedre kan ses hvis man tillader lidt ekstra plads i plottets område.

#### 4.2.2 Transformering af akserne - log, sqrt osv (`scale_x_continuous`).

Nogle gange kan det være svært at visualisere nogle variabler på grund af deres fordeling. Er der mange outliers i variablen så er de fleste punkter samlede i et lille område i plottet. Transformering med enten `log` eller `sqrt` på x-aksen og/eller y-aksen er især en populær tilgang, så de data kan ses på en mere informativ måde.

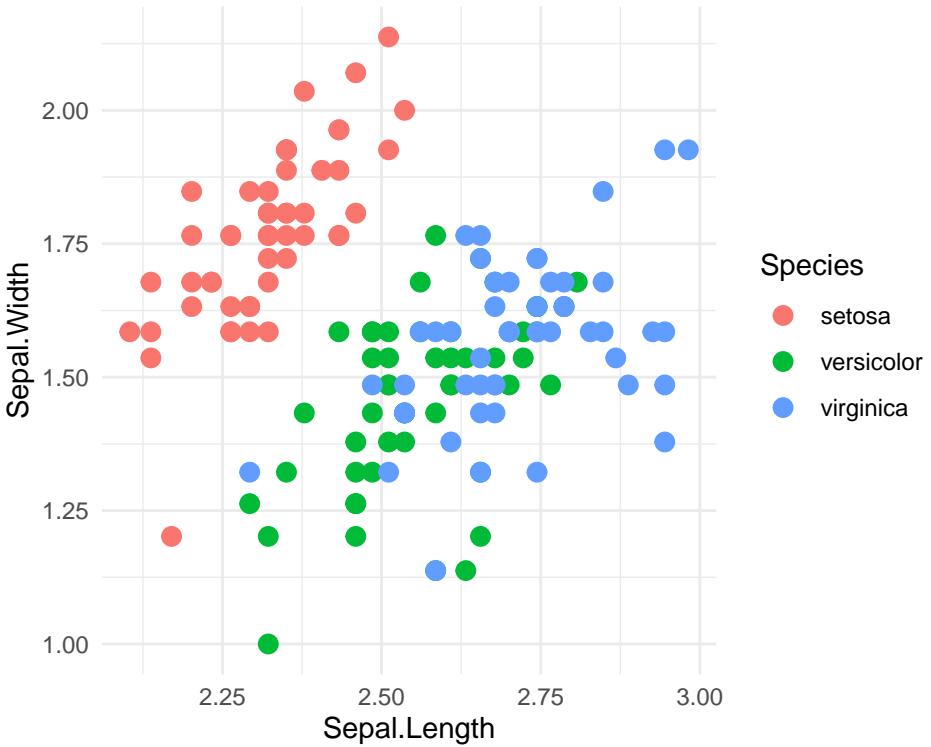
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point(size=3) +
  scale_x_continuous(trans = "log2") +
  scale_y_continuous(trans = "log2") +
  theme_minimal()
```



Man kan også prøve fx. "sqrt" i stedet for "log2". Formålet er, at hvis de data fordeler sig mere 'normalt', kan man nemmere visualisere det i et plot - en måde til at gøre der er ved at transformere de data med "sqrt" eller "log2".

Bemærk at det er til forskel fra at man transformere selve data som bruges i plottet. Jeg kan for eksempel få samme resultat ved at ændre på datasættet forud for at anvende ggplot2 - her behøver jeg ikke at bruge `scale_x_continuous(trans = "log2")` for at opnår samme resultat, men notater at tallerne på akserne reflektere de transformerede data og ikke de oprindelige værdier. Den beslutninger man tager her kommer an på, hvad man gerne vil opnå med analysering af de data.

```
iris$Sepal.Length <- log2(iris$Sepal.Length)
iris$Sepal.Width <- log2(iris$Sepal.Width)
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point(size=3) +
  theme_minimal()
```

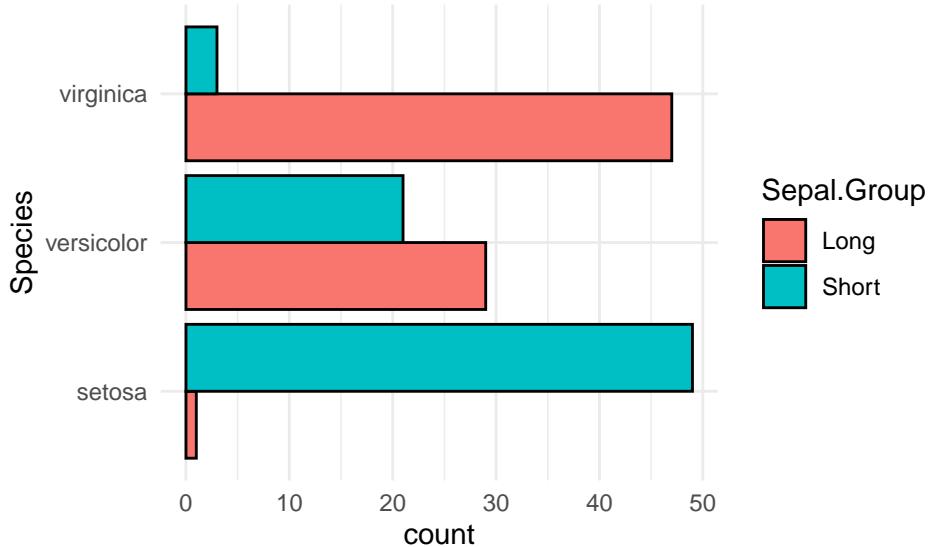


### 4.2.3 Flip coordinates (`coord_flip`)

Vi kan bruge `coord_flip()` til at spejler x-aksen på y-aksen og omvendt (det svarer til, at man drejer plottet ved 90 grader). Se følgende eksempel, hvor jeg først opretter variablen `Sepal.Group`, laver en barplot og anvender `coord_flip` for at få vandrettet søjler.

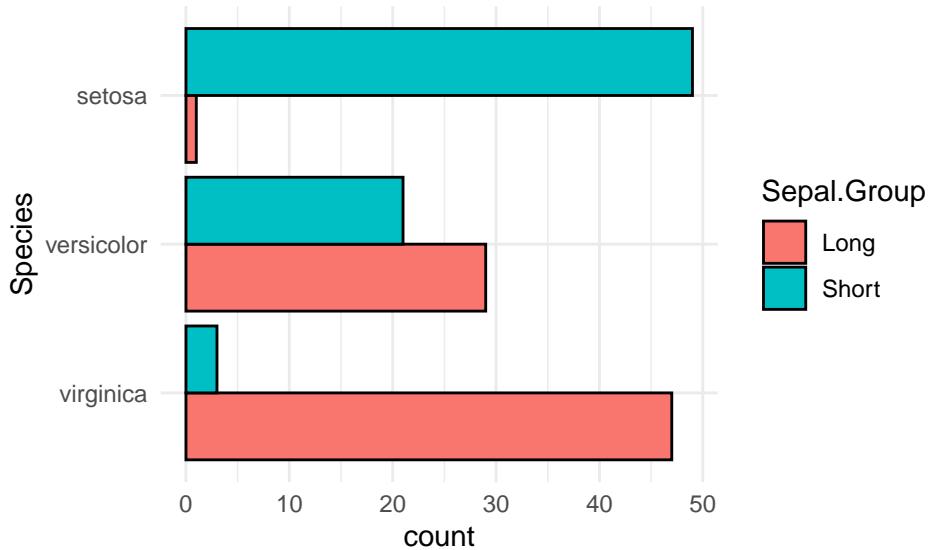
```
#Sepal.Group defineret som i går
iris$Sepal.Group <- ifelse(iris$Sepal.Length>mean(iris$Sepal.Length), "Long", "Short")

ggplot(iris,aes(x=Species,fill=Sepal.Group)) +
  geom_bar(stat="count",position="dodge",color="black") +
  coord_flip() +
  theme_minimal()
```



Man kan ændre på rækkefølgen af de tre `Species` ved at bruge funktionen `scale_x_discrete()` og angiver den nye rækkefølge med indstillingen `limits`:

```
ggplot(iris,aes(x=Species,fill=Sepal.Group)) +
  geom_bar(stat="count",position="dodge",color="black") +
  coord_flip() +
  scale_x_discrete(limits = c("virginica", "versicolor","setosa")) +
  theme_minimal()
```



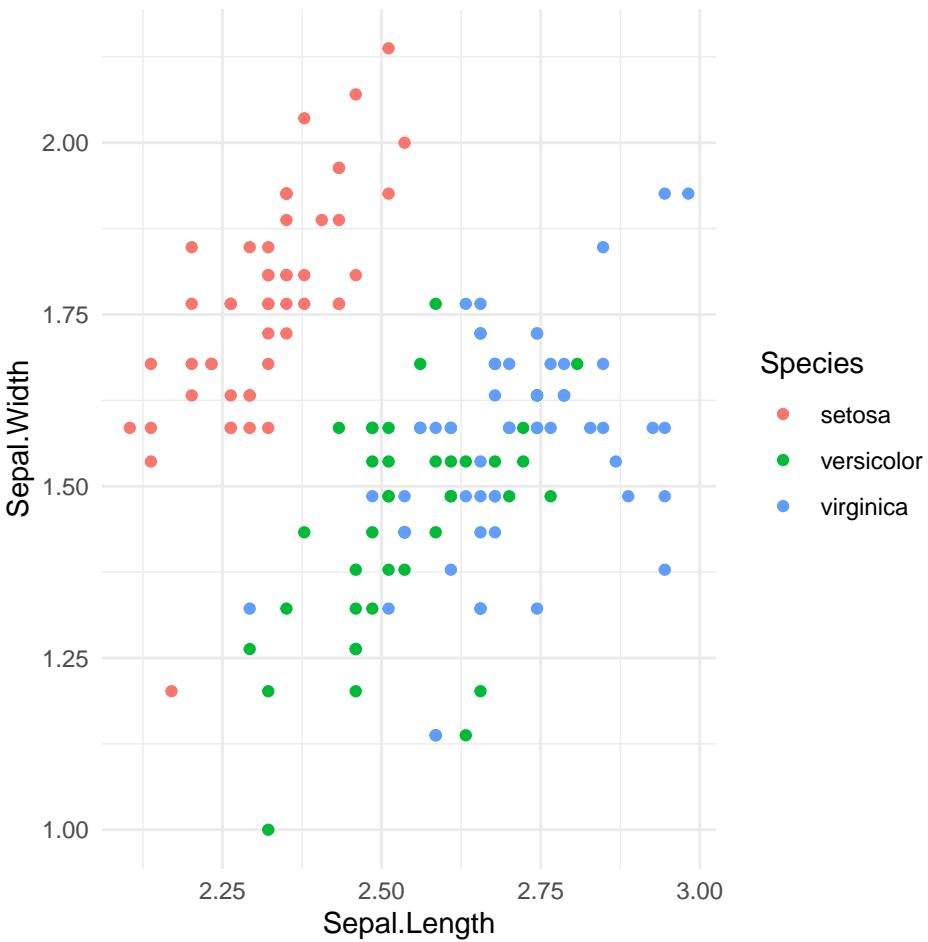
## 4.3 Mere om farver og punkt former

Der er flere måder at specificere farver på i ggplot2. Man kan nøjes med den automatiske løsning, som er hurtigt (og effektiv i mange situationer), eller man kan bruge den manuelle løsning, som tager lidt mere tid at indkode men er brugbar hvis man gerne vil lave et plot til at præsentere til andre.

### 4.3.1 Automatisk farver

Vi så sidste emnet at man automatisk kan bede om forskellige farver, ved at benytte `colour=Species` indenfor `aes()` i den `ggplot()` funktion.

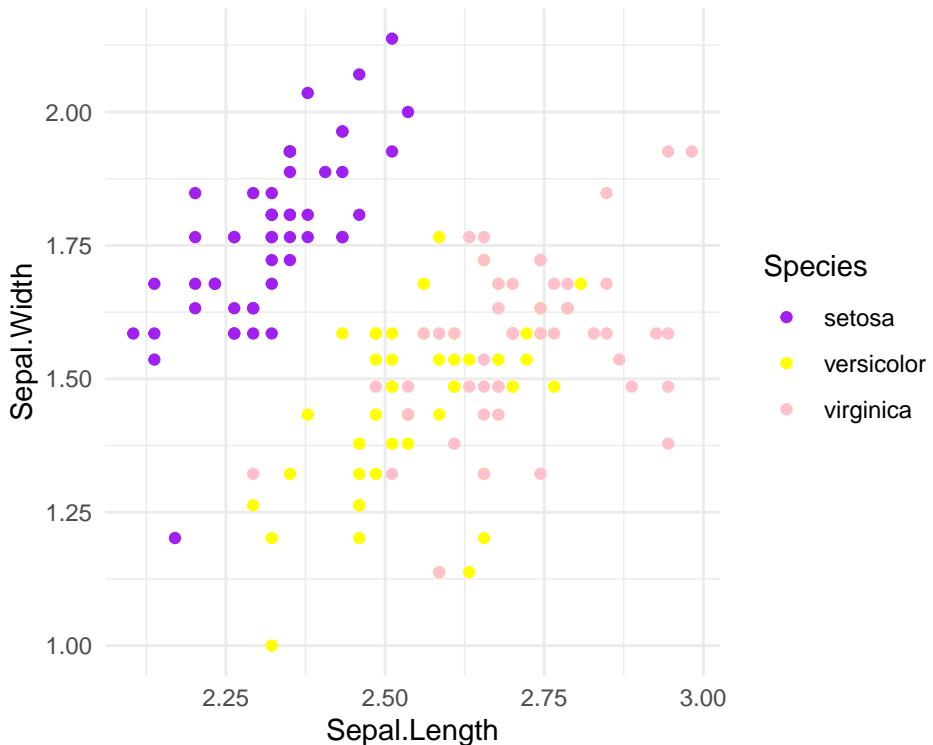
```
#automatisk løsning
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +
  geom_point() +
  theme_minimal()
```



### 4.3.2 Manuelle farver

Hvis man foretrækker at bruge sine egne farver, kan man det ved at benytte funktionen `scale_colour_manual()`. Her angiver jeg stadig `colour=Species` indenfor `aes()` men så angiver jeg hvilke bestemte farver de forskellige arter skal få indenfor `scale_colour_manual` med indstillingen `values`.

```
#manuelt løsning
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +
  scale_colour_manual(values=c("purple", "yellow", "pink")) +
  geom_point() +
  theme_minimal()
```

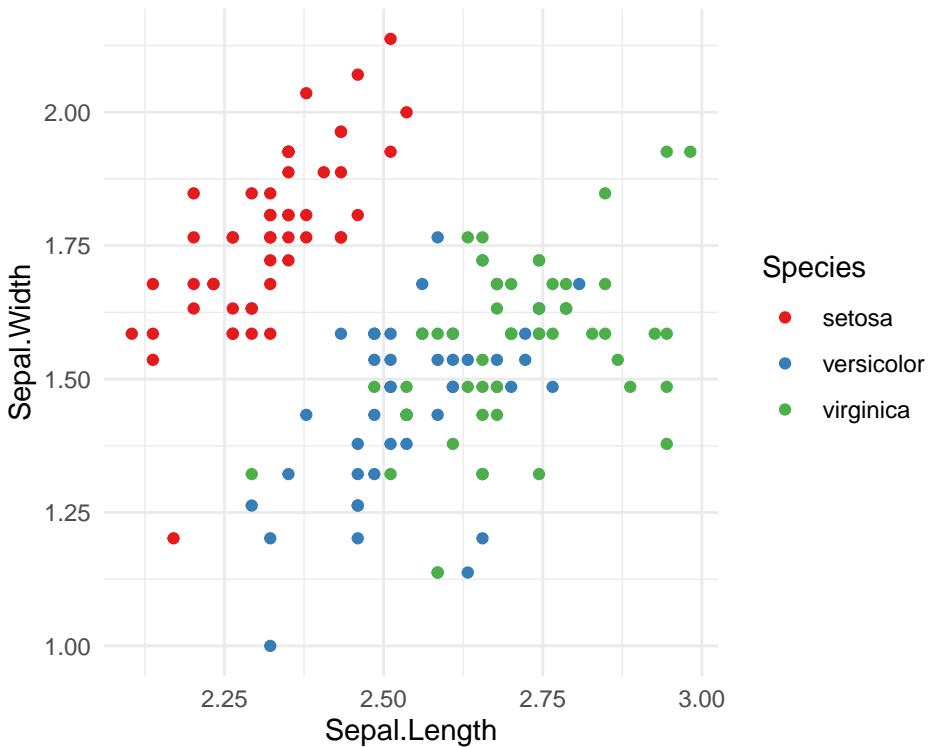


En faktastisk pakke er `RColorBrewer`. Pakken indeholder mange forskellige “colour palettes”, det vil sige grupper af farver, der passer godt med hinanden. Man kan således slippe for at selv samle et godt kombination der passer til plottet. Nogle af de colour palettes tager også i betragtning, hvis man er farveblind, eller om man vil have en farvegradient eller et sæt diskrete farver som ikke ligner hinanden.

I følgende indlæser jeg pakken `RColorBrewer` og anvender funktionen `scale_colour_brewer` med indstillingen `palette="Set1"`:

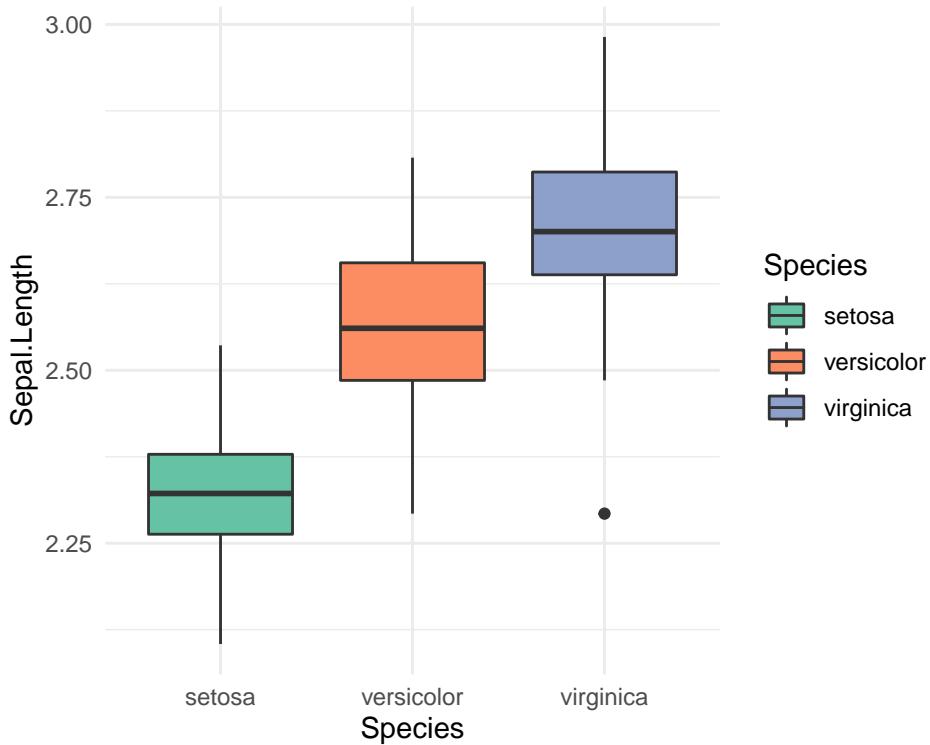
```
#install.packages("RColorBrewer")
library(RColorBrewer)

#manuelt løsning
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +
  scale_colour_brewer(palette="Set1") +
  geom_point() +
  theme_minimal()
```



Bemærk at både `scale_color_maual()` og `scale_color_brewer()` sætter farver af punkter og linjer, mens i en boxplot eller barplot sammenhænge, bruger man `scale_fill_manual()` eller `scale_fill_brewer()`. For eksempel i følgende vil jeg gerne sætte farver på de opfyldte områder i en boxplot:

```
ggplot(iris,aes(x=Species,y=Sepal.Length,fill=Species)) +
  geom_boxplot() +
  scale_fill_brewer(palette="Set2") +
  theme_minimal()
```



Her er en oversigt over de fire funktioner.

| funktion   | beskrivelse                                      |
|--|--|
| <code>scale_fill_manual(values=c("firebrick1", "steelblue1"))</code> | til boxplots og barplots osv.                    |
| <code>scale_color_manual(values=c("darkorange1", "cyan4"))</code>    | punkter og linjer osv.                           |
| <code>scale_fill_brewer(palette="Dark2")</code>                      | RColourBrewer løsning til boxplots/barplots/osv. |
| <code>scale_color_brewer(palette="Set1")</code>                      | RColourBrewer løsning til punkter og linjer osv. |

Der er også andre muligheder hvis man har behov for dem - for eksempel for kontinuitet data kan man prøve at google efter `scale_fill_gradient`.

#### *Farver i RColourBrewer*

Her er en nyttig reference, der viser de forskellige farver tilgængelige i pakken RColourBrewer.

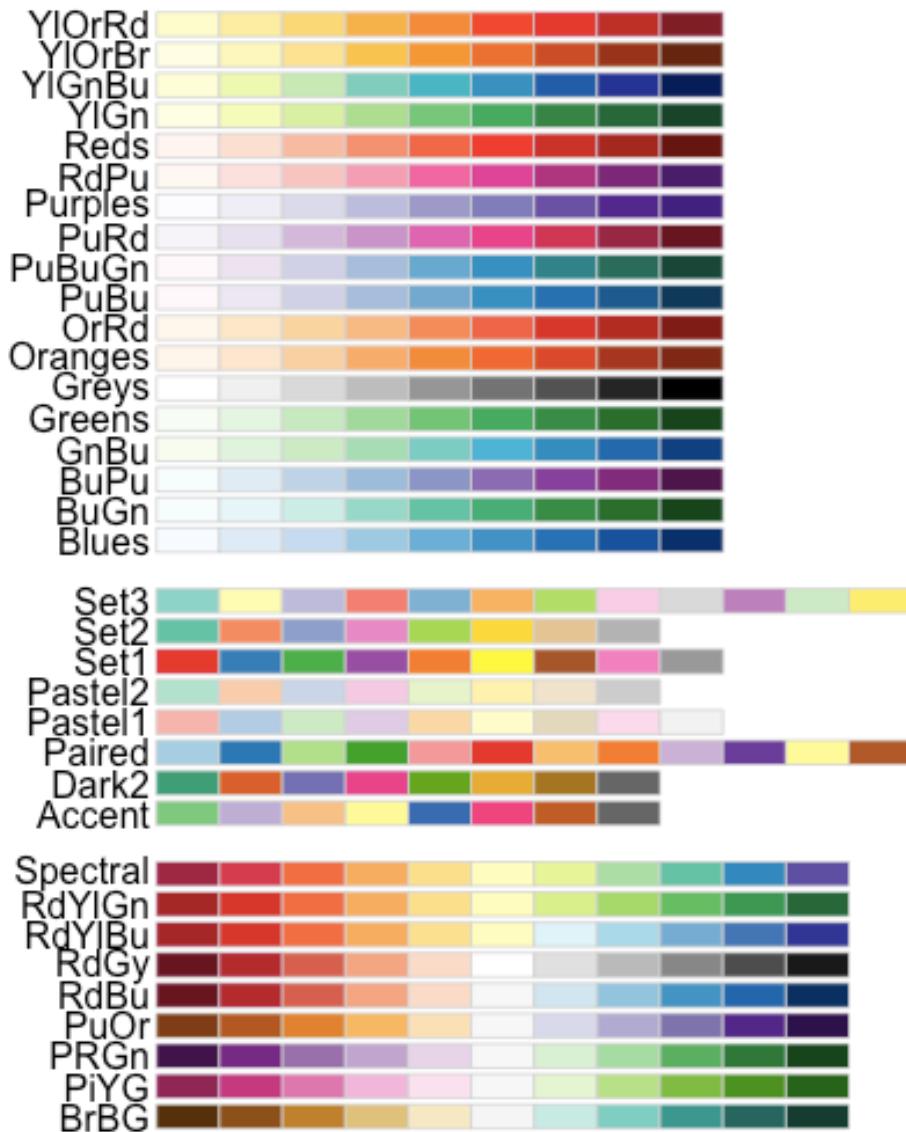
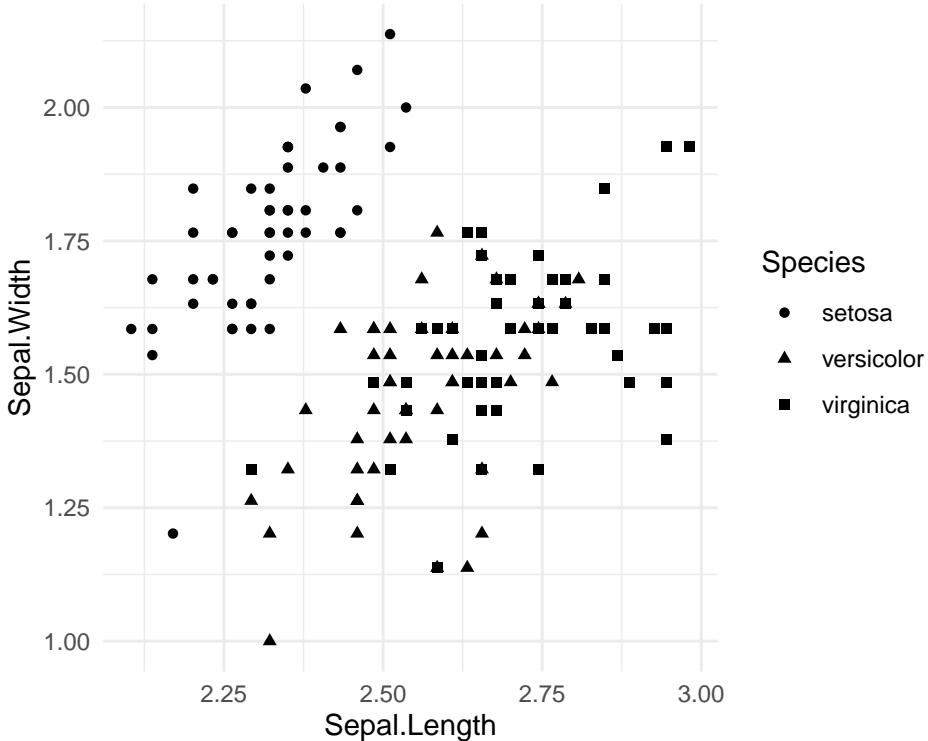


Figure 4.1: Mulige colour palettes tilgængelige i RColourBrewer

### 4.3.3 Punkt former

Ligesom man kan lave forskellige farver, kan man også lave forskellige punkt former. Vi starter med den automatiske løsning ligesom vi gjorde med farver. Når det er en variable vi angiver, skal variablenavnet skrives indenfor `aes()`. Her, da `shape` er en parameter som er meget specifik til `geom_point`, vælger jeg at skrive en ny `aes()` indenfor `geom_point()` i stedet for indenfor funktionen `ggplot()`. Husk, at i funktionen `ggplot()` specificerer man globale ting som gælder for hele plot, og i funktionen `geom_point()` angiver man ting som gælder kun for `geom_point()`. Se følgende eksempel:

```
ggplot(data=iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  scale_color_brewer(palette="Set2") +
  geom_point(aes(shape=Species)) +
  theme_minimal()
```

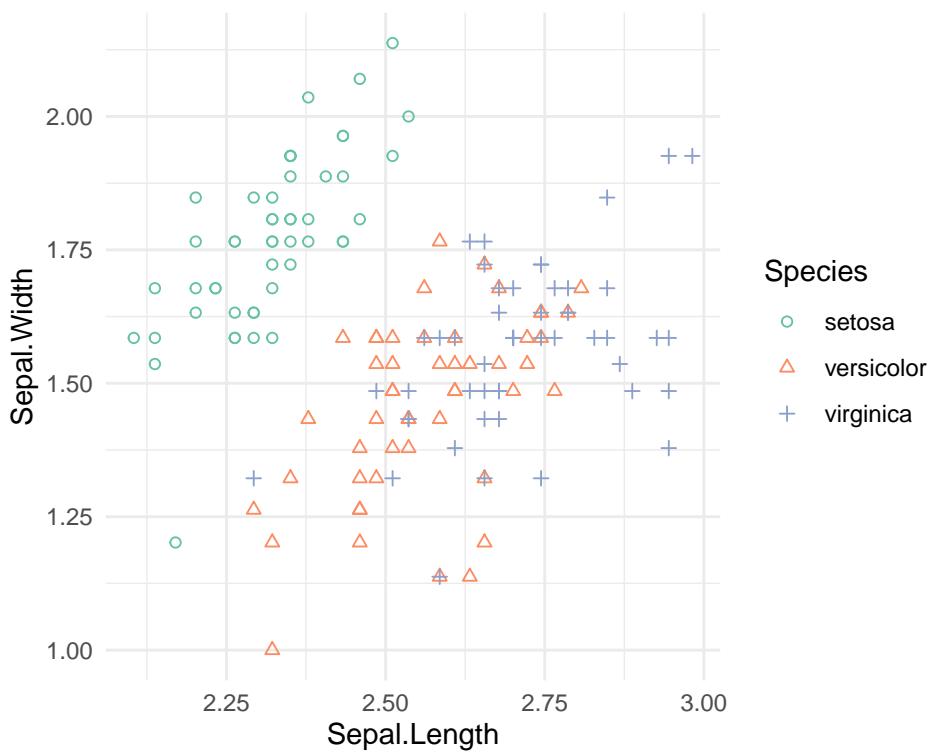


Så har jeg fået både en farve og en punkt form til hver art i variablen `Species`.

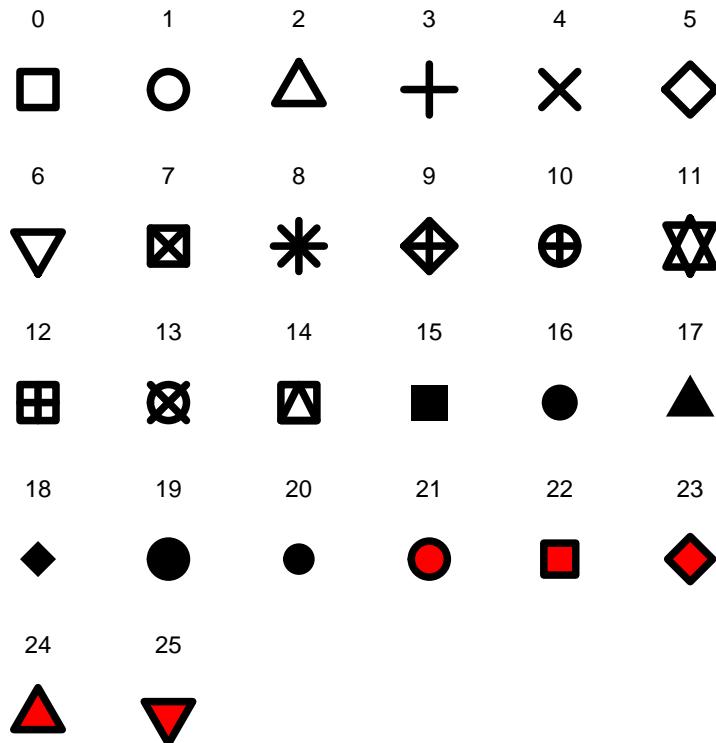
#### *Sætte punkt form manuelt*

Hvis vi ikke kan lide de tre punkt former vi få automatisk, kan vi ændre dem ved at bruge `scale_shape_manual` - her vælger jeg `values=c(1,2,3)`, men der er en reference nedenunder, hvor I kan se, de mappings mellem de numeriske tal og de punkt former, så at I kan vælge jeres egne.

```
ggplot(data=iris, aes(x = Sepal.Length, y = Sepal.Width, colour=Species)) +  
  geom_point(aes(shape=Species)) +  
  scale_color_brewer(palette="Set2") +  
  scale_shape_manual(values=c(1,2,3)) +  
  theme_minimal()
```



*Reference for punkt former*



## 4.4 Annotations (geom\_text)

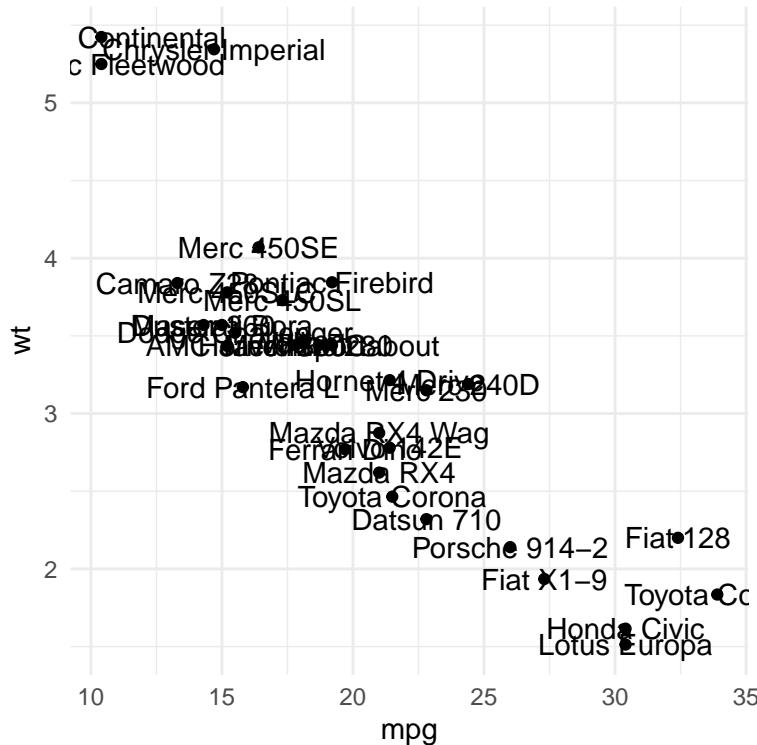
### 4.4.1 Tilføje labeller direkte på plottet.

Man kan bruge `geom_text()` til at tilføje tekst på punkterne direkte på plottet. Her skal man fortælle, hvad for nogle tekster skal være på plottet - her specificerer vi navne på biler fra datasættet `mtcars`. Plottet er en scatter plot mellem variabler `mpg` og `wt`.

```
data(mtcars)

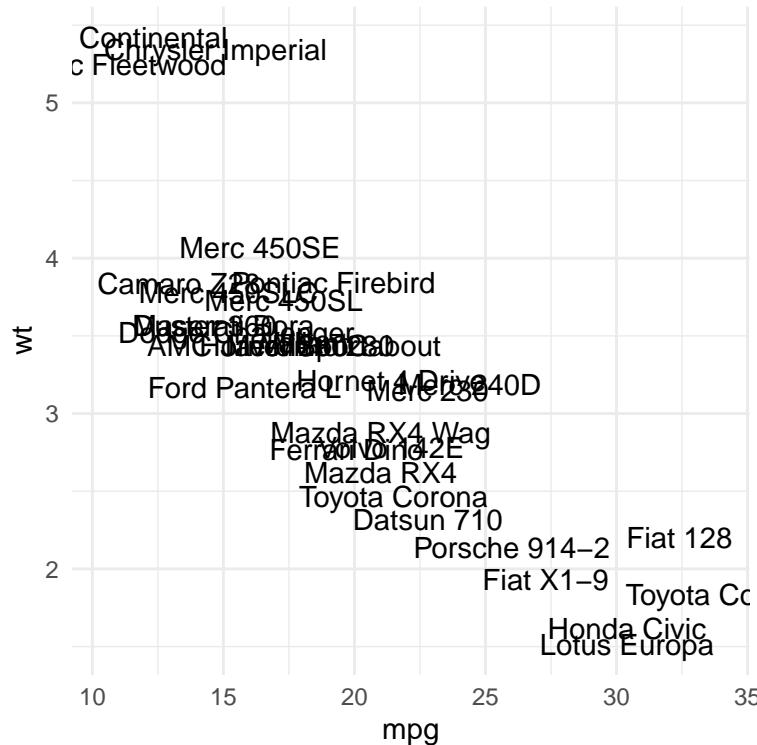
mtcars$my_labels <- row.names(mtcars) #take row names and set as a variable

ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text(aes(label=my_labels)) +
  theme_minimal()
```



For at gøre det nemmere at læse kan man også fjerne selve punkterne:

```
ggplot(mtcars,aes(x=mpg,y=wt)) +
  #geom_point() +
  geom_text(aes(label=my_labels)) +
  theme_minimal()
```



Teksten på plottet er stadig meget svært at læse. En god løsning kan være at bruge R-pakken `ggrepel`, som i følgende.

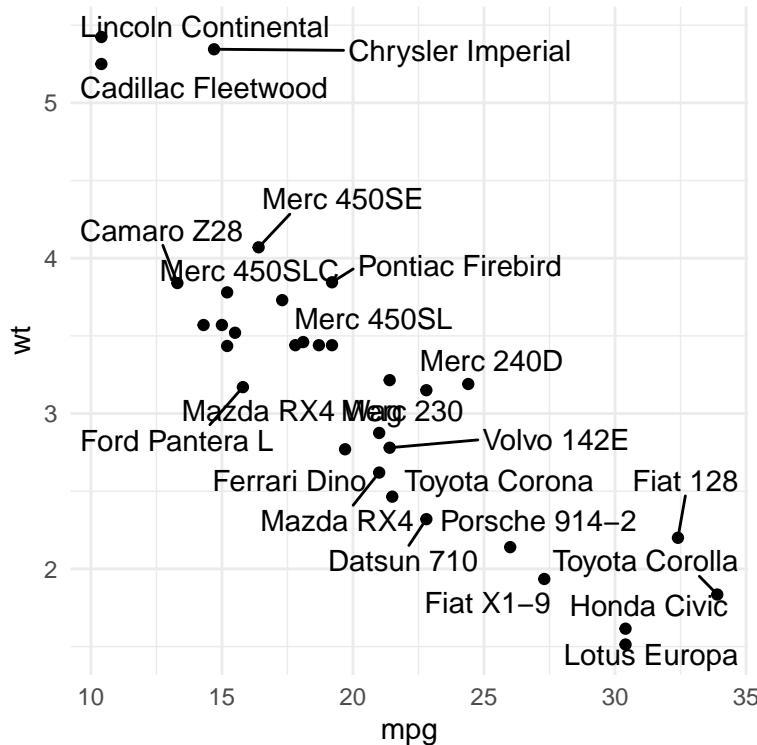
#### 4.4.2 Pakken `ggrepel` for at tilføje tekst labeller

```
#install.packages("ggrepel") #installere hvis nødvendigt
```

For at anvende pakken `ggrepel` for det `mtcars` datasæt, erstatter man bare `geom_text()` med `geom_text_repel()`:

```
library(ggrepel)
ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text_repel(aes(label=my_labels)) +
  theme_minimal()
```

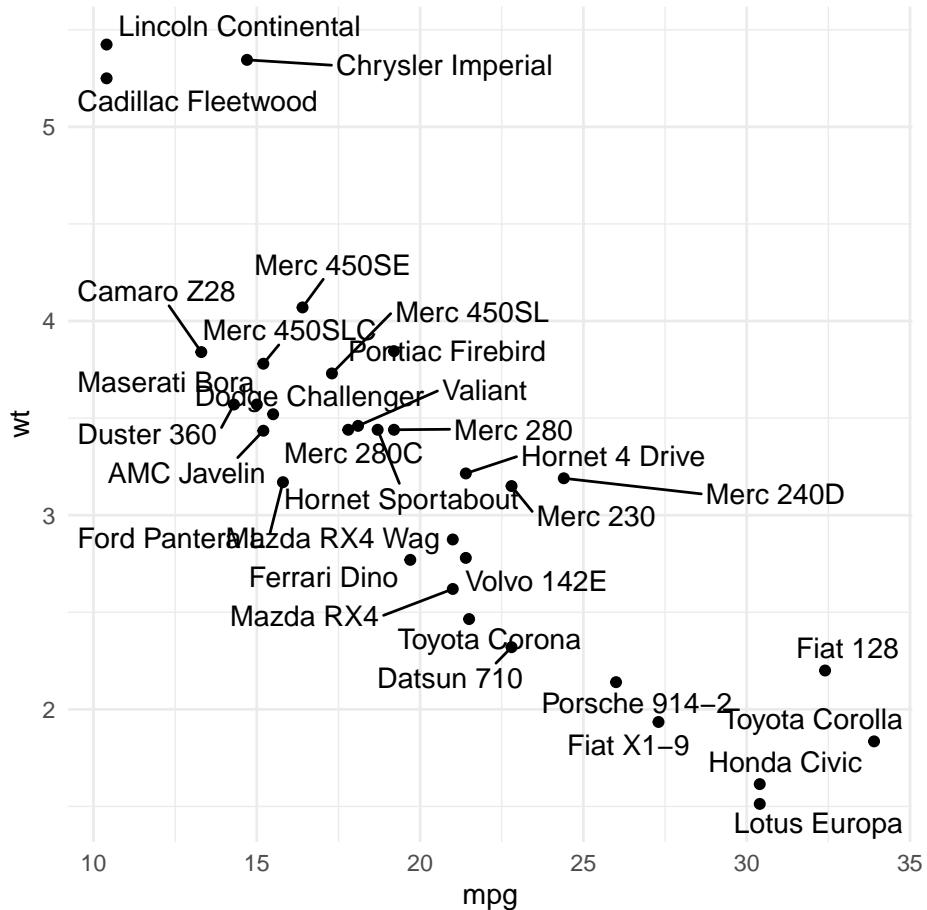
```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Så kan vi se, at nu er der ingen navne som sidder lige overfor hinanden, fordi `ggrepel()` har været dygtig nok til at placerer dem tæt på deres tilhørende punkter, og ikke ovenpå hinanden. Man kan også se her, at der er nogle punkter, hvor funktionen har tilføjet en linje for at gøre det klart, hvilken punkt teksten refererer til.

I ovenstående har jeg fået en advarsel. Jeg prøver hvad jeg er blevet bedt om - og fortæller, at jeg vil have `max.overlaps = 20`.

```
library(ggrepel)
ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text_repel(aes(label=my_labels),max.overlaps = 20) +
  theme_minimal()
```



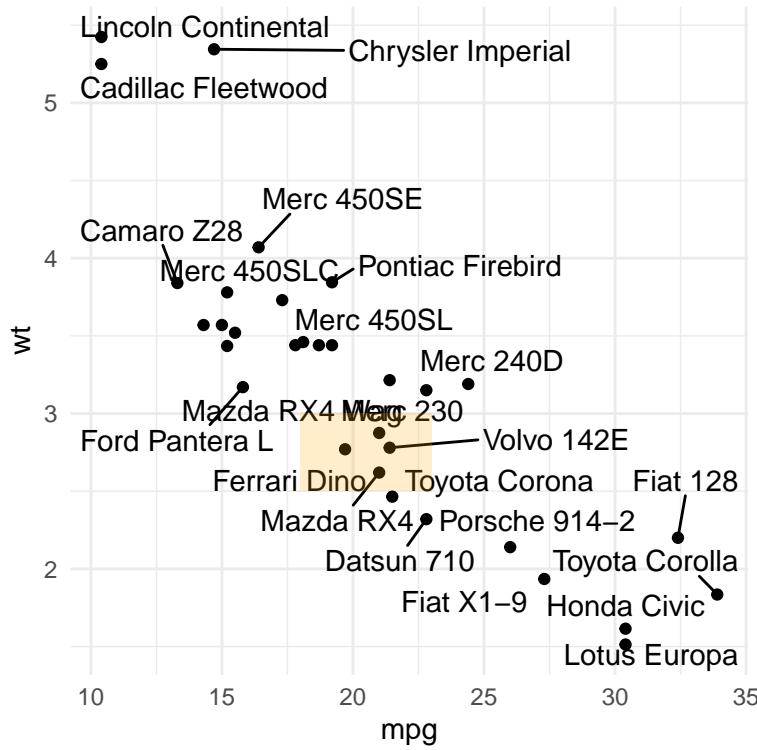
Så kan du se, at jeg ikke længere få en advarsel, og der tilhører tekst til alle punkterne nu.

#### 4.4.3 Tilføje rektangler i regioner af interesse (annotate)

Hvis man gerne vil fremhæve et bestemt område i plottet, kan man bruge funktionen `annotate()`. Prøve at selv regne ud, hvad de indstillinger indenfor `annotate()` går ud på i følgende eksempel:

```
ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text_repel(aes(label=my_labels)) +
  annotate("rect",xmin=18,xmax=23,ymin=2.5,ymax=3,alpha=0.2,fill="orange") +
  theme_minimal()

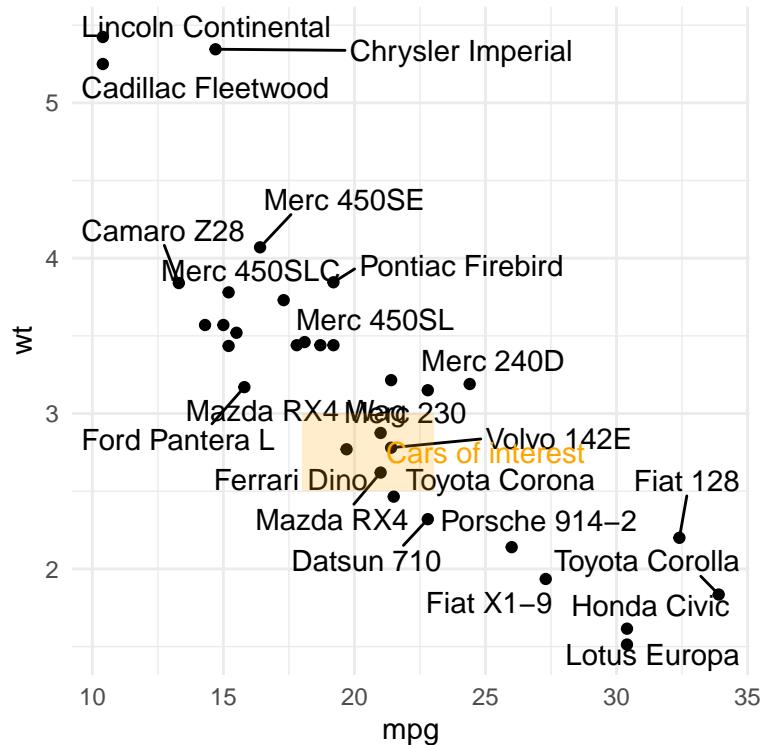
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Man kan også benytte den samme funktion til at tilføje nogle tekster på et bestemt sted:

```
ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text_repel(aes(label=my_labels)) +
  annotate("rect",xmin=18,xmax=23,ymin=2.5,ymax=3,alpha=0.2,fill="orange") +
  annotate("text",x=25,y=2.75,label="Cars of interest",col="orange") +
  theme_minimal()
```

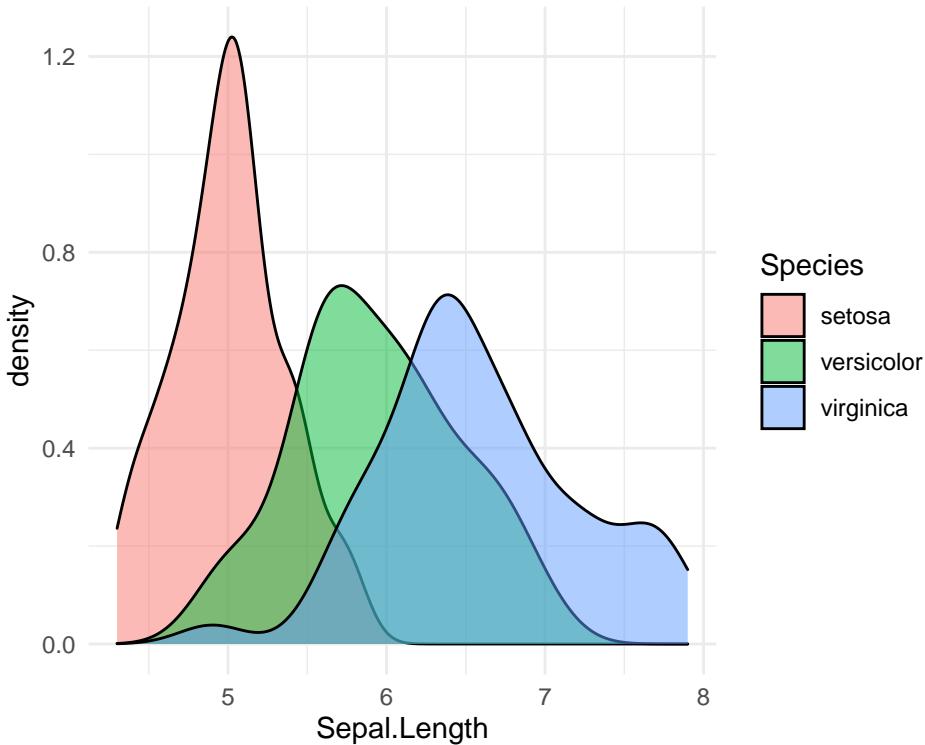
```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



## 4.5 Adskille plots med facets (facet\_grid/facet\_wrap)

En stor fordel af at bruge ggplot er evnen til at benytte funktionerne `facet_grid()` og `facet_wrap()` til at adskille efter en kategorisk variabel over flere plotter. I følgende kode viser jeg et density plot, hvor de tre kurver der tilhører de tre arter ligger oven på hinanden i det samme plot:

```
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  theme_minimal()
```



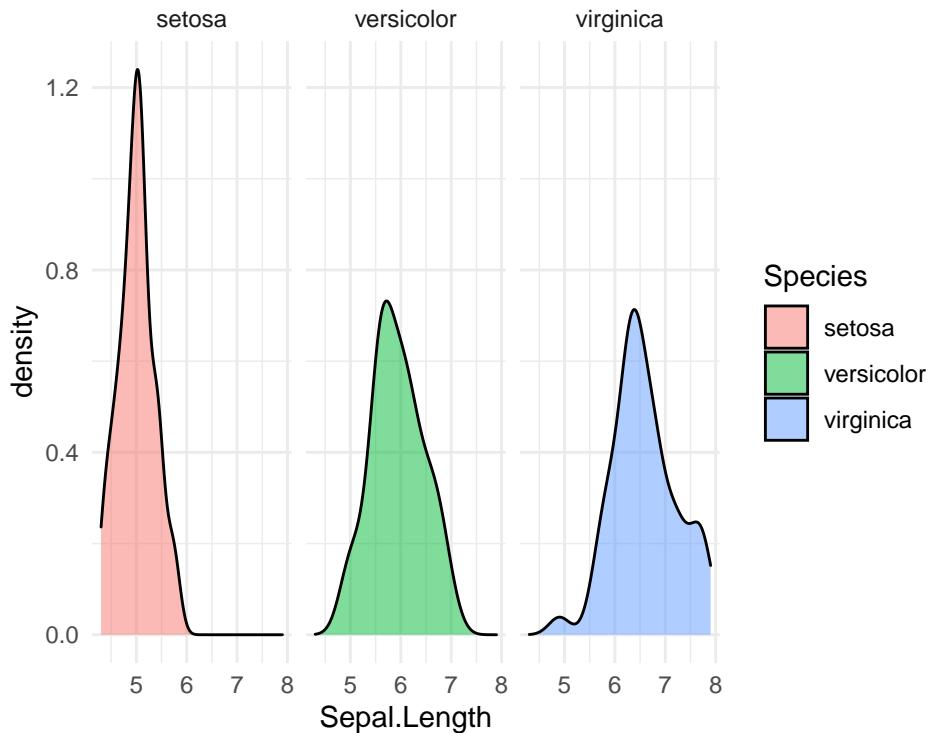
Med funktionen `facet_grid()` eller `facet_wrap()` bruger vi ~ (tilde) tegn til at angive hvordan vi gerne vil visualisere de forskellige plots - skal man opdele dem over rækker (variablerne venstre til ~) eller over kolonner (variabler højre til ~)?

**#notrun**

```
variable(s) to split into row-wise plots ~ variables(s) to split into column-wise plots
```

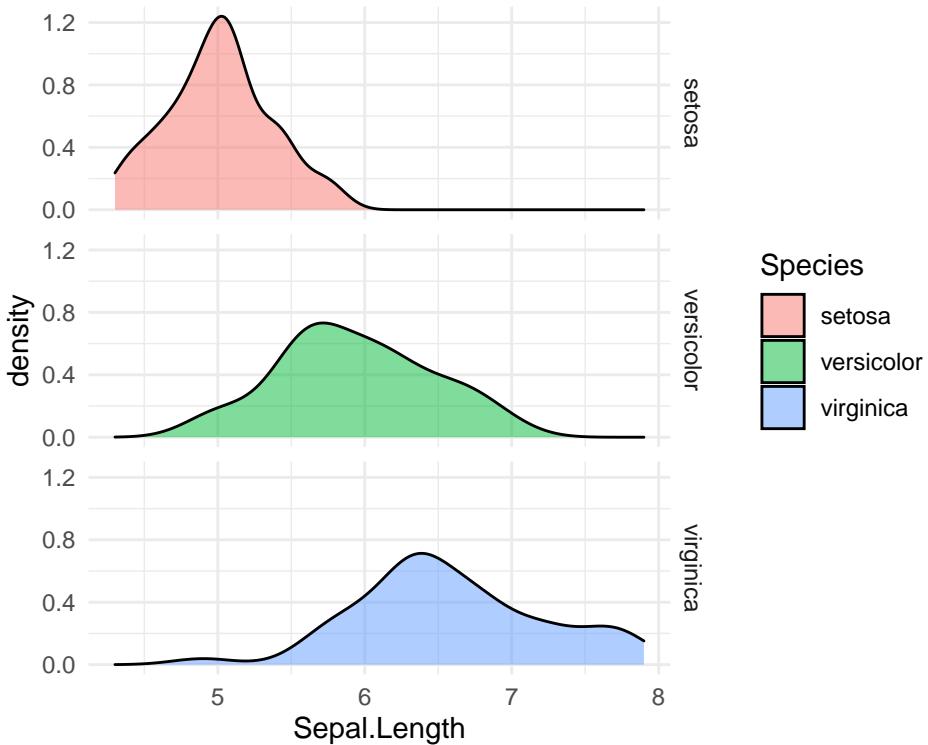
Ovenstående density plots af Sepal.Length kan adskilles på Species, således at man får tre plots med en kolon til hver af de tre arter:

```
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_grid(~Species) + #split Species into different column-wise plots
  theme_minimal()
```



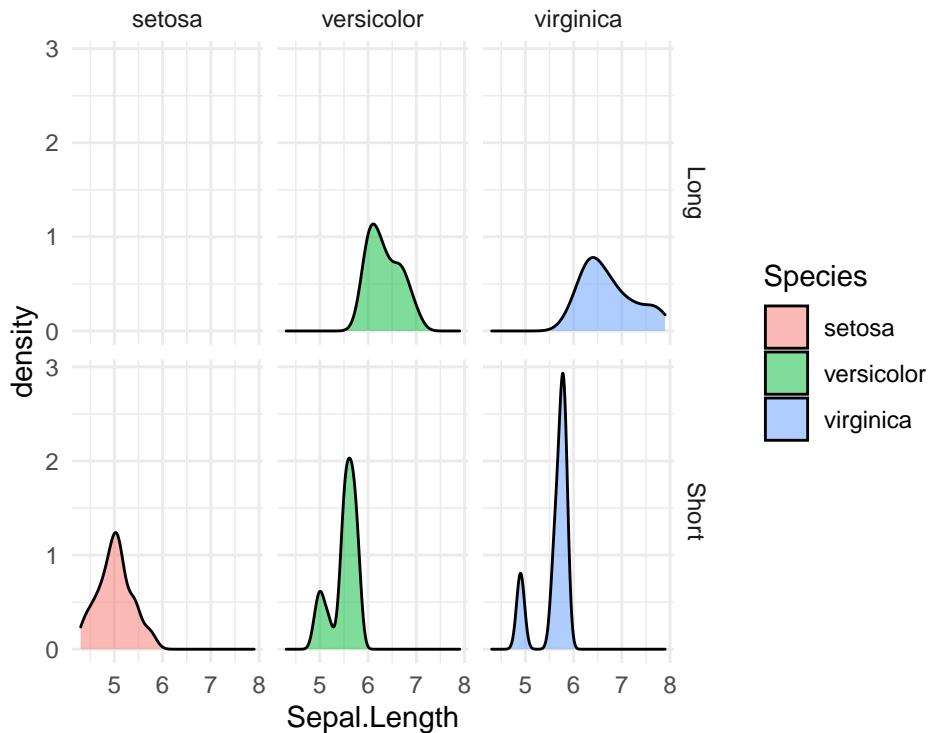
Man kan gøre det over rækkerne (man skal dog husk at bruge en “.” efter “~” for at betegne at man kun vil adskille plots over rækkerne, mens kan af en eller anden grund kan man dropper “.” hvis man kun vil adskiller over kolonner som i ovenstående).

```
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_grid(Species~.) + #split Species into different column-wise plots
  theme_minimal()
```



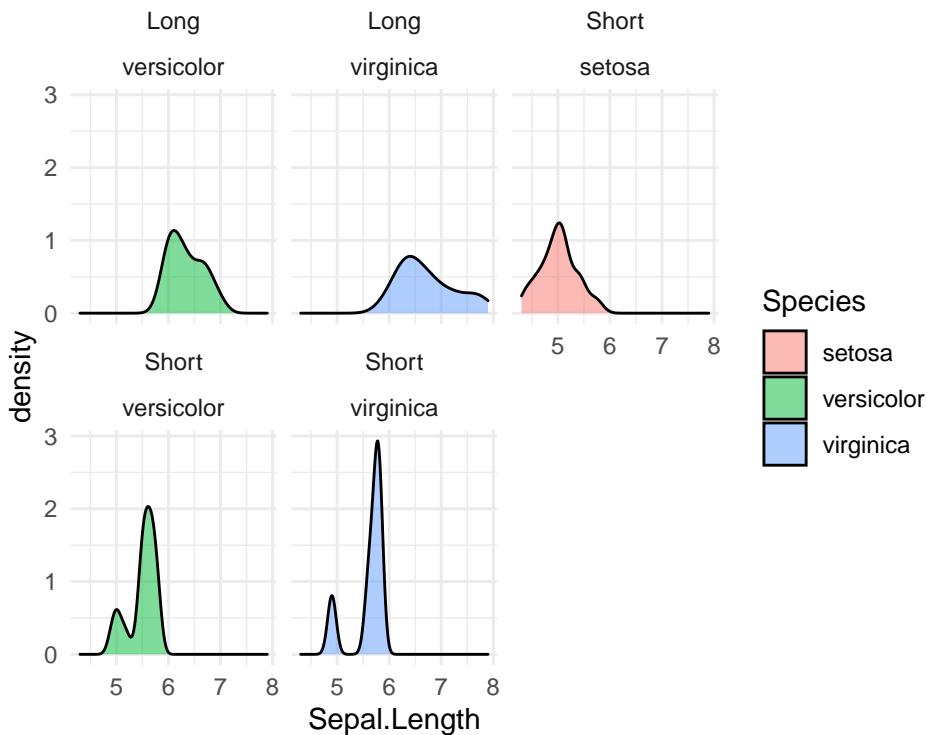
Her angives Sepal.Group~Species, der betyder, at plotterne bliver adskilt efter både Sepal.Group og Species - Sepal.Group over rækkerne, og Species over kolonnerne:

```
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_grid(Sepal.Group~Species) + #split Species into different column-wise plots
  theme_minimal()
```



Bemærk forskellen mellem `facet_grid()` og `facet_wrap()`:

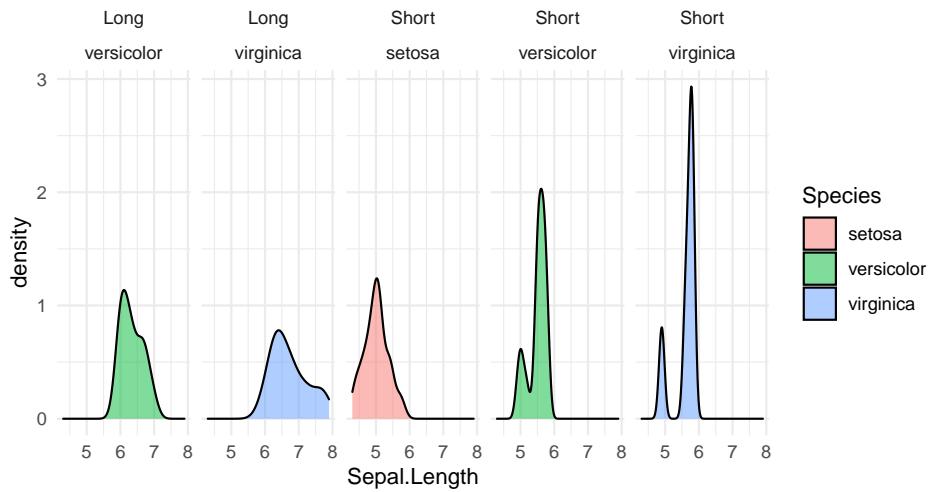
```
#same plot, replace facet_grid with facet_wrap
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_wrap(Sepal.Group~Species) +
  theme_minimal()
```



I `facet_grid()` bliver man tvunget til at få en “grid” layout. Vi har således 6 plotter i en  $2 \times 3$  grid (2 niveauer til variablen `Sepal.Group` og 3 niveauer til variablen `Species`), og det sker selvom den ene af dem har ikke noget data ind i - der findes altså ikke nogle observationer hvor `Species` er “Setosa” og `Sepal.Group` er “Long”, men vi får et plot alligevel for at bevare strukturen. Med `facet_wrap()` bliver plotterne uden data droppet og i dette tilfælde får man 5 plotter i hvad der kaldes for en “ribbon”.

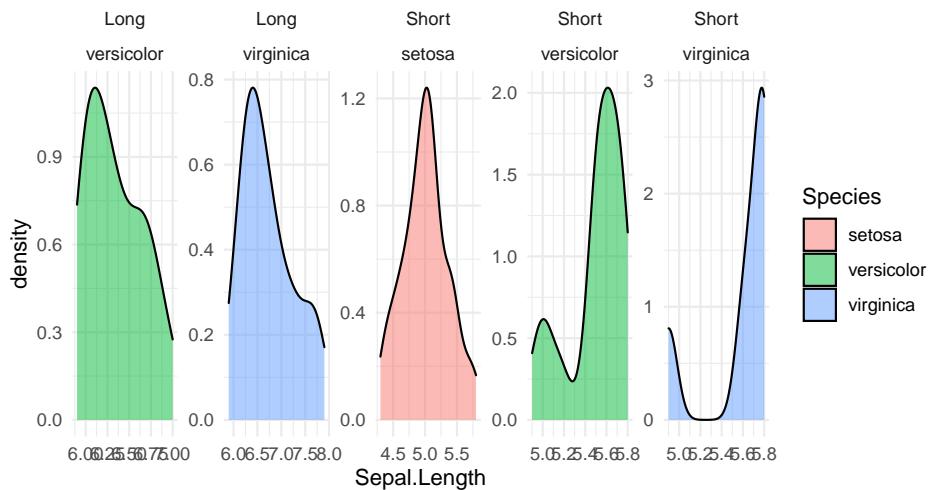
Med `facet_wrap()` kan man også fortælle at vi gerne vil have plotterne over 1 row (`nrow = 1` eller `ncol = 5`):

```
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_wrap(Sepal.Group~Species,nrow = 1) +
  theme_minimal()
```



Til sidste kan det være at jeg gerne vil ”befrie” skalen på y-akserne - på den måde har ikke alle plotter den samme maksimum y-værdi og de enkelte plotter benytter i stedet egne værdierne til at bestemme skalerne. Det kan være brugbart hvis man inddrager forskellige målinger men vær dog opmærksom på hvad der bedste giver mening - hvis man direkte vil sammenligne to af plotterne så er det bedre at de dele samme y-akse skale.

```
#same plot, replace facet_grid with facet_wrap
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_wrap(Sepal.Length~Species,ncol = 5,scales = "free") +
  theme_minimal()
```



Jeg håber at det er klart, at funktionerne er meget brugbart, og mens de opnår stort set samme ting, er der små forskel mellem dem, der er værd at husk.

## 4.6 Gemme dit plot

Her bruger vi R Markdown til at lave et rapport som indeholder vores plots, men det også kan være at man gerne vil gemme sit plot som fil på computeren. Til at gemme et plot kan man bruge kommandoen `ggsave()`:

```
ggsave(myplot, "myplot.pdf")
```

Figuren vil blive gemt i din *working directory* (eller den mappe, du har din .Rmd fil). Filtypen `.pdf` kan erstattes med andre formater, for eksempel `.png` eller `.jpeg` osv. Hvis man gerne vil tage sit plot og redigerer på det (fk. Adobe Illustrator eller Inkscape), vil jeg anbefaler, at du bruge `.pdf`.

Man må gerne ændre højden og bredden på det gemt plot med `width` og `height`:

```
ggsave(myplot, "myplot.pdf", width = 4, height = 4)
```

---

## 4.7 Problemstillinger

**Problem 1)** Lav quiz - “Quiz - ggplot2 part 2”

---

**Problem 2) (Factorer og plots)**

a) Åbn datsættet `mtcars` og lav en barplot:

- Brug variablen `cyl` på x-aksen og give forskellige farver efter den samme variabel.
- Virker din kode? Kig på x-aksen.
- Variablen er numerisk men skal fortolkes som en factor. Lav variablen om til en factor (eller bare skriv `as.factor(cyl)` i selve plottet) og lave dit plot igen.

b) Opdel søgerne ved at angive farver efter variablen `gear` i dit plot (søjlerne skal sidde ved siden af hinanden). Vær OBS om hvordan R fortolker variablen.

---

I følgende problemer arbejder vi med datasættet `Palmer Penguins`. Pakken `palmerpenguins` skal installeres hvis du ikke har brugt datasættet før.

Data beskrivelse: *The palmerpenguins data contains size measurements for three penguin species observed on three islands in the Palmer Archipelago, Antarctica.*



```
#install.packages("palmerpenguins") #kører hvis ikke allerede installeret
library(palmerpenguins)
```

```
library(ggplot2)
```

```
library(tidyverse)
```

```
head(penguins)
```

```
FALSE # A tibble: 6 x 8
FALSE   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
FALSE   <fct>    <fct>      <dbl>        <dbl>          <int>       <int> <fct>
FALSE 1 Adelie   Torgersen  39.1         18.7          181        3750 male
FALSE 2 Adelie   Torgersen  39.5         17.4          186        3800 female
FALSE 3 Adelie   Torgersen  40.3         18            195        3250 female
FALSE 4 Adelie   Torgersen  NA           NA            NA         NA <NA>
FALSE 5 Adelie   Torgersen  36.7         19.3          193        3450 female
FALSE 6 Adelie   Torgersen  39.3         20.6          190        3650 male
```

FALSE # ... with 1 more variable: year <int>

Man kan altid anvende `?penguins` for at se flere detaljer om variablenavner.

*Vi skal starte med at rydde op lidt i datasættet. Køre følgende for at fjerne alle rækker som har NA (manglerne) værdier:*

```
penguins <- drop_na(penguins)
```

### Problem 3) Manuelt farver og punkter

a) Lav en scatter plot med `ggplot`:

- `bill_length_mm` på x-aksen
- `bill_depth_mm` på y-aksen
- give hver `species` sin egen farve (automatisk løsning)
- sætte et tema

b) Lav følgende ændringer til det plot:

- Ændr farver manuelt - prøv både at angive farver med `scale_color_manual` og afprøve også løsningen med pakken `RColorBrewer` (husk at installere/indlæse pakken hvis nødvendigt).
- Angiv at der skal være forskellige punkt former for hver art i variablen `species`.
- Prøv også at vælge nogle punkt former fra listen og specifiser dem manuelt.

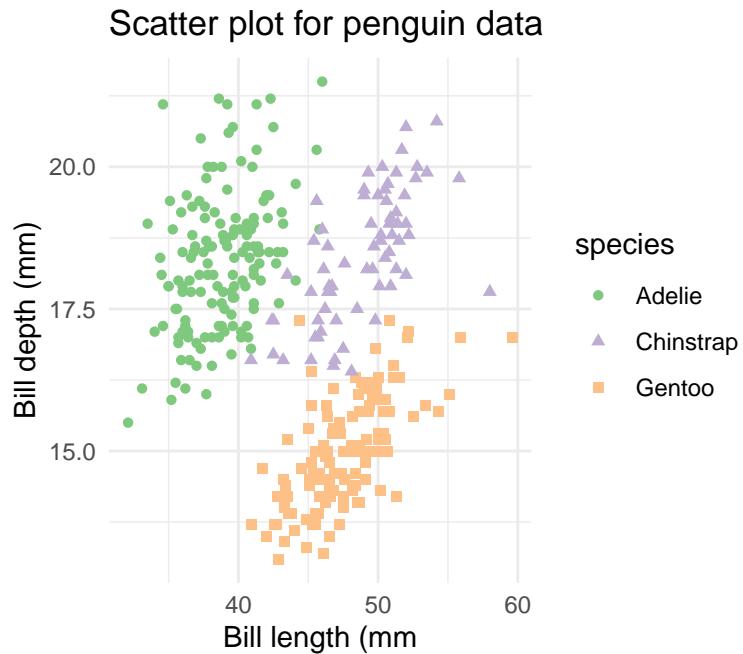


Figure 4.2: Min løsning

---

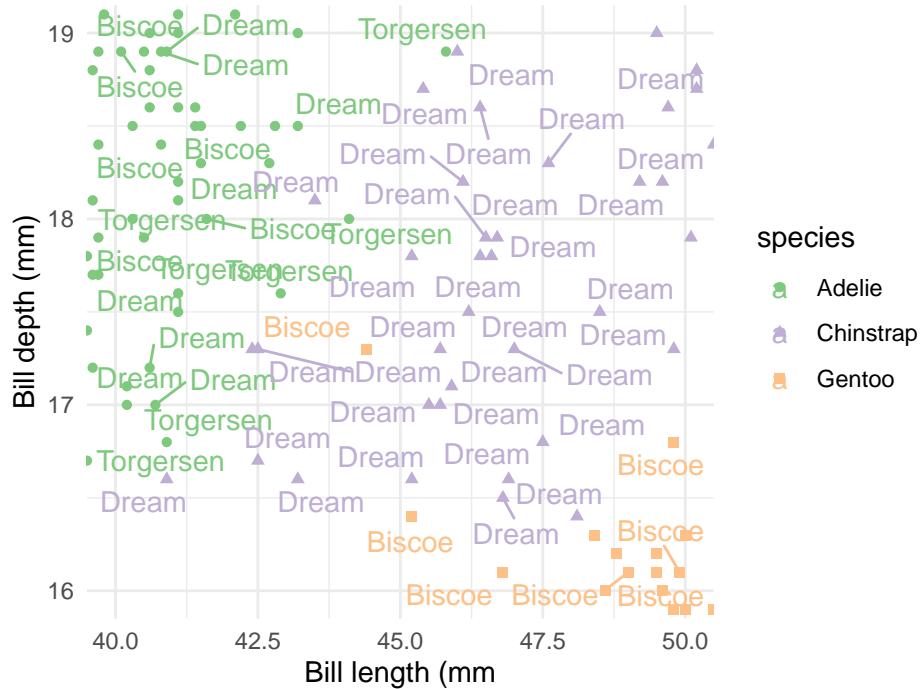
#### Problem 4) Coordinate systemer

Tag overstående scatter plot fra 3) og

- brug `coord_cartesian()` så at man fanger kun en bill længde (variablen `bill_length_mm`) mellem 40 og 50, og en bill depth (variablen `bill_depth_mm`) mellem 16 og 19.
- brug pakken `ggrepel` (husk at installere/inlæse) og tilføj navnerne af de forskellige øer som tekst direkte på plottet **c)** lav en delmængde af dataframen `penguins` efter samme kriterier som **a)** og specifiser din nye dataframe som parameteren `data` indenfor `geom_text_repel`-funktionen. Det undgår, at tekst bliver plottet for punkterne udenfor området angivet med `coord_cartesian()`.

```
## Warning: ggrepel: 14 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

### Scatter plot for penguin data



### Problem 5) Histogram med facets

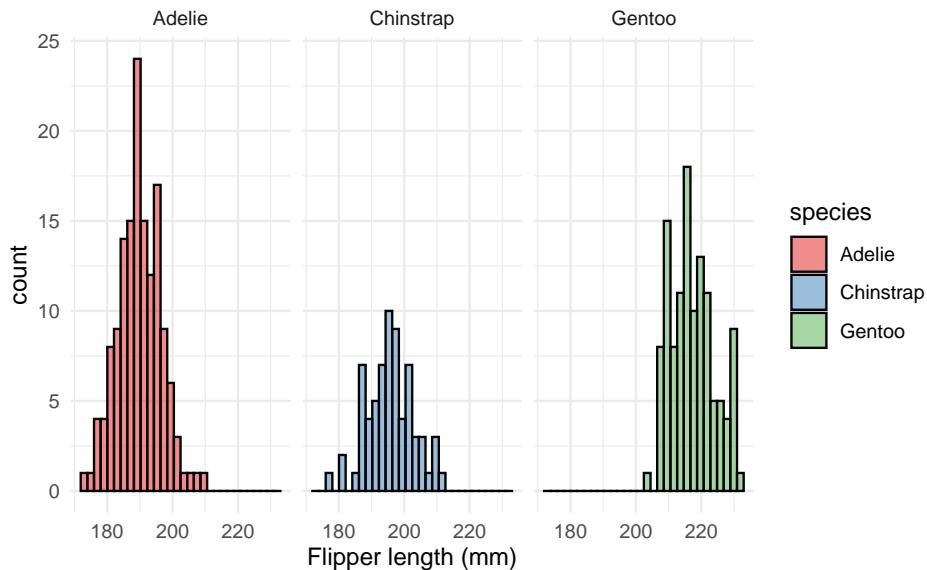
Lav en histogram:

- Variablen `flipper_length_mm` på x-aksen
- Anvend `facet_grid` for at adskille dit plot i tre efter variablen `species`
- Giv også en forskellige farve til hver art i `species`
- Hvis nødvendigt ændr parameteren `bins` til noget andet indenfor `geom_histogram()`.

Her er min løsning:

```
ggplot(data=penguins,aes(x=flipper_length_mm,fill=species)) +
  geom_histogram(bins = 30, alpha=0.5, colour="black") +
  scale_fill_brewer(palette = "Set1") +
  ggtitle("Histogram of flipper length according to species") +
  facet_grid(~species) +
  xlab("Flipper length (mm)") +
  theme_minimal()
```

### Histogram of flipper length according to species



**Problem 6)** a) Lave et density plot af `body_mass_g`.

- Anvend funktionen `facet_grid` til at opdele i til tre plots efter `species`
- Brug også `fill` til at opdele densities indenfor de tre plots efter variablen `sex`
- Gør dine density plots gennemsigtige
- Skrive en sætning om forskellen i `body_mass_g` mellem "females" og "males".

b) Nu udvikl din `facet_grid` kommando til at adskille plots yderligere således at du har en "grid" struktur med de forskellige øer (variablen `island`) på rækkerne og de tre arter (variablen `species`) på kolonnerne.

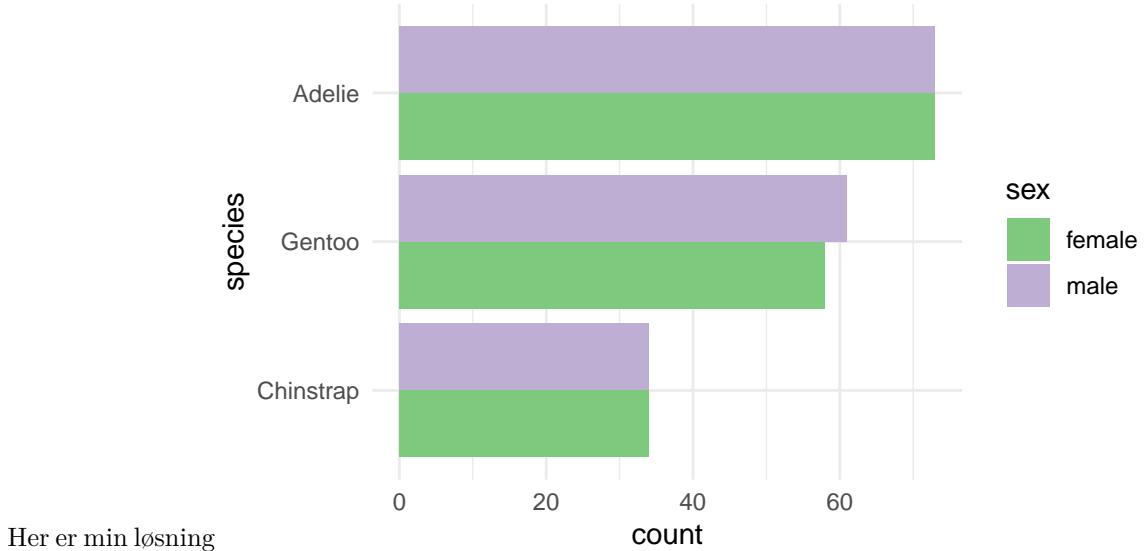
c) Kan du forklare hvorfor der er blank plotter i din grid? Eksperimenter med `facet_wrap` i stedet for `facet_grid`.

**Problem 7)** *Coordinate systemer*

Lav en barplot af counts for `species` opdelte efter `sex`.

- Anvend en 'coordinate flip' for at få den til at være vandrette/horizontal.
- Vælg nogle farver - jeg benytter `palette = "Accent"` fra den `RColorBrewer` løsning
- Ændr rækkefølgen af de tre søjler, således at arten med de meste observationer er på toppen og arten med den færrest er på bunden.

- Prøv også `scale_y_reverse()` og kig på resultatet.



**Problem 8)** Lav boxplots af `body_mass_g` opdelt efter `species`.

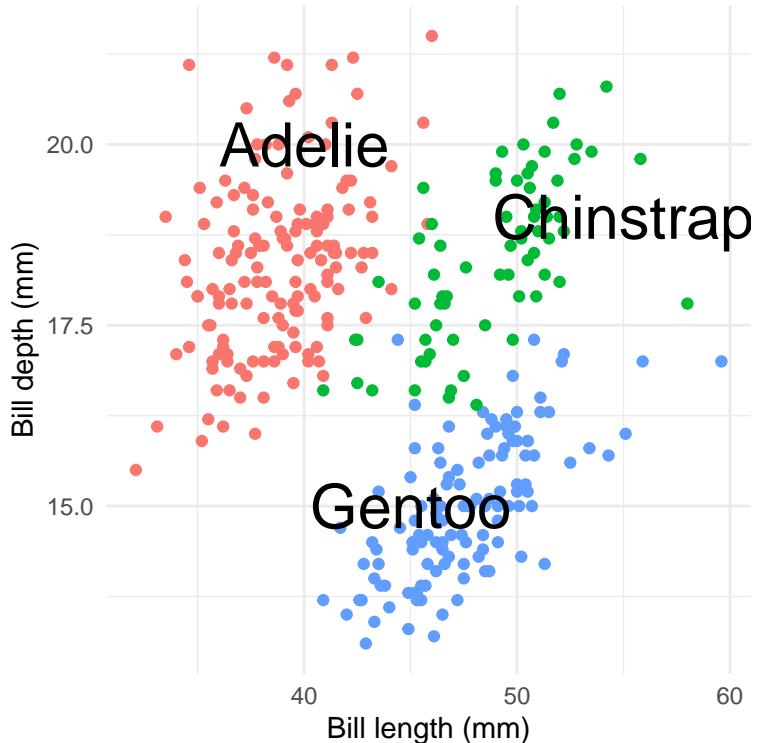
- Tilføj “jitter” punkter ovenpå boxplots.
- Specifier nogle farver manuelt for både boxes og punkterne (en farve til hver art)
- Giv det en hensigtsmæssig titel og nogle akser-labels
- Tilføj en ny variabel `island_binary` til `penguins`, som er “Biscoe” hvis `island` er ‘Biscoe’ og “not Biscoe” hvis ikke.
- Adskille plotterne ved at opdele efter `island_binary`.
- Ekstra: prøv `?geom_violin` som erstatning for `geom_boxplot`.

**Problem 9)** *Annotations og linjer.*

a) Lav et scatter plot af `bill_length_mm` vs `bill_depth_mm`.

- Anvend hensigtsmæssigt titel/labels/tema
- Anvend forskellige farver for de tre `species`.
- Tjek funktionen `?annotate` og bruge den med `geom="text"` og hensigtsmæssigt x- og y-akse værdier til at tilføje `species` navne som tekst direkte på plottet (se eksempel nedenfor for at se, hvad jeg mener).
- Udforsk hvordan man gøre teksten større, som jeg har gjort i min løsning.
- Fjern legend med `show.legend = FALSE` indenfor `geom_point()`

Her er min løsning:



b) Vi vil gerne tilføje nogle lodrette og vandrette linjer til plottet, som viser middelværdierne af variablerne for de tre arter

- Først brug `tapply` til at beregne de gennemsnitlige værdier for henholdsvis `bill_length_mm` og `bill_depth_mm` opdelte efter `species` (gem dem som henholdsvis `mean_length` og `mean_depth`)
- Brug `mean_length` og `mean_depth` til at tilføje linjer til plottet med den relevante funktion.

c) **Udfordring** Kan du gøre linjerne til samme farver som punkterne af deres pågældende art (se min løsning nedenunder)?

- Hint: tage udgangspunkt i følgende dataframe, der bruger din beregnede værdier:

```
mydf <- data.frame("species"=names(mean_length), "mlength"=mean_length, "mdepth"=mean_depth)
mydf
```

```
##           species   mlength   mdepth
## Adelie      Adelie 38.82397 18.34726
## Chinstrap   Chinstrap 48.83382 18.42059
## Gentoo      Gentoo 47.56807 14.99664
```

- Angiv parameteren `data` til at være ovenstående dataframe i `geom_vline()` og brug lokal aethestiks (`aes()`) til at angive parametre til linjerne.
- Gør samme for `geom_hline()`
- Specifier også “dashed” linjer

Her er min løsning:

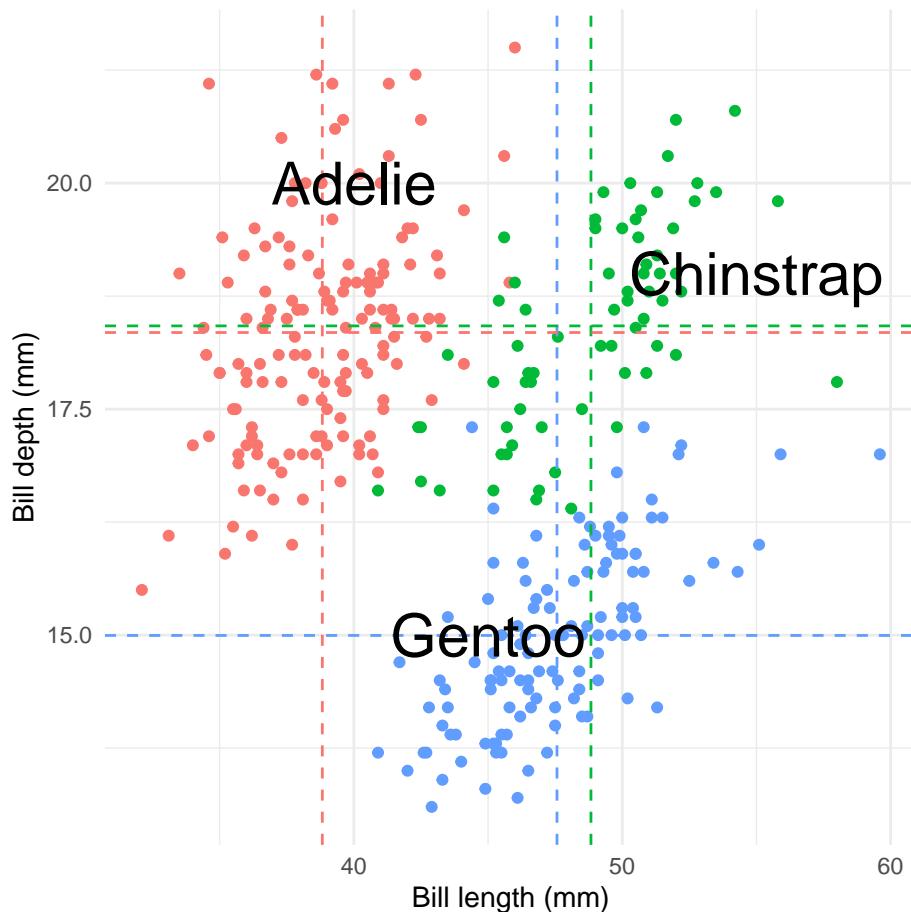


Figure 4.3: min løsning

---

**Problem 10) Ekstra.** Kig på “cheatsheet” til ggplot2 (Tryk på “Help” > “Cheatsheets” og vælg en til ggplot2) og afprøve nogle af de ting, som ikke var dækket i kurset indtil videre! Gerne lad mig høre hvis du synes der er eventuelle noget meget nyttig for dig, som er ellers blevet glemt i notaterne.

## 4.8 Ekstra links

R Graphics cookbook

<https://r-graphics.org/>



# Chapter 5

## Bearbejdning dag 1



### 5.1 Hvad er Tidyverse?

**Tidyverse** er en samling af pakker i R, som man bruger til at bearbejde datasæt. Formålet er ikke nødvendigvis at erstatte funktionaliteten af base-pakken, men til at bygge på den. Som vi kommer til at se i detaljer, **tidyverse** deler faktisk meget af den samme tankegang bag **ggplot2** - men i stedet for at bruge + til at bygge komponenter op i et plot, bruger man `%>%` (udtales ‘pipe’) til at tilknytte de forskellige funktioner til hinanden.

#### Læringsmålene til i dag

I skal være i stand til at:

- Beskrive generelle hvad R-pakken **Tidyverse** kan benyttes til.
- Beskrive en tibble og genkende når et datasæt er betragtet som “tidy”.
- Benytte nogle vigtige **Tidyverse**-verbs til at bearbejde data (`filter()`,`select()`,`mutate()`,`rename()`,`arrange()`,`recode()`).
- Bruge `%>%` til at forbinde **Tidyverse**-verber sammen og at overføre data til et plot.



Figure 5.1: Most common tidyverse packages

## 5.2 Video ressourcer

- Begynd ved at læse “Principper med ‘tidy data’” og ‘Lidt om tibbles’ nedenunder, og så se følgende videoer.
- **Video 1** - rydde op i datasættet `titanic` med `select()` og `drop_na()`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/706266697>

- **Video 2** - tidyverse verber: `select` og `filter`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/705136725>

- **Video 3** - flere tidyverse verber
  - Lave en ny kolon med `mutate()`
  - Ændre variabelnavne med `rename()`
  - Ændre på værdierne med `recode()`
  - Ændre rækkefølgen af observationerne med `arrange()`
  - Bruge tidyverse kommandoer som input i `ggplot2()`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/706266885>

## 5.3 Oversigt over pakker

Lad os starte med at indlæse pakken `tidyverse`. Vær opmærksom på, at har du ikke pakken på din computer, kan det tage lidt tid at installere - det er pga. de mange pakke der `tidyverse` er afhængig af, der enten skal også installeres eller opdateres. **Hvis har pakken installeret men oplever problemer tjek om du har det seneste versioner af pakkerne og R på dit system.**

```
#install.packages("tidyverse")
library(tidyverse)
```

Du kan se, at det faktisk er ikke kun én, men otte pakke som er blevet indlæst. Man kan godt indlæse alle pakke individuelt ved at bruge fk. `library(dplyr)`, men det er meget bekvemt at indlæse alle på samme tid med brugen af `library(tidyverse)`. Her er nogle beskrivelser af de pakker:

| pakke                  | korte beskrivelse  |
|------------------------|--|
| <code>readr</code>     | indlæse data   |
| <code>ggplot2</code>   | plot data  |
| <code>tibble</code>    | lave “tibbles” - <i>tidyverse</i> ’s svar på datarammer (data.frame).                            |
| <code>tidyverse</code> | skifte imellem data forms (fk. ‘long’ > ‘wide’ format, eller omvendt)                            |
| <code>purrr</code>     | functional programming, gentalelse   |
| <code>dplyr</code>     | manipulere tibbles - beholde delmængder, skabe nye variabler, beregne oversigtsstatistikker osv. |
| <code>stringr</code>   | manipulere strings (ikke brugt i dette kursus)   |
| <code>forcats</code>   | FOR CATegorical data (factors); håndtere faktor variabler  |

Man kan også indlæse alle pakke individuelt ved at bruge fk. `library(dplyr)`, men det er meget bekvemt bare at indlæse alle på samme tid med brugen af `library(tidyverse)`.

## 5.4 Principper med ‘tidy data’

Idéen bag **tidyverse** er, at hvis alle datasæt følger præcis den samme struktur, så er det enkelt datasæt ligefrem at bearbejde til præcis som vi gerne vil have det. Datasæt som har den struktur hedder “tidy data”. For at betragte et datasæt som “tidy”, må det opfylde tre kriterie:

- Hver variabel i datasættet har sin egen kolonne
- Hver observation i datasættet har sin egen række
- Hver værdi i datasættet få sin egen cell

Iris er et godt eksempel af **tidy data**:

```
data(iris)
head(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
```

```
## 1      5.1      3.5      1.4      0.2  setosa
## 2      4.9      3.0      1.4      0.2  setosa
## 3      4.7      3.2      1.3      0.2  setosa
## 4      4.6      3.1      1.5      0.2  setosa
## 5      5.0      3.6      1.4      0.2  setosa
## 6      5.4      3.9      1.7      0.4  setosa
```

Hver variabel (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width` og `Species`) har sin egen kolon, og hver observation (e.g. 1,2,3, osv.) har sin egen række. Derudover har hver cell sin egen data værdi og det er dermed meget klart at læse og forstå dataframen ved øjnene.

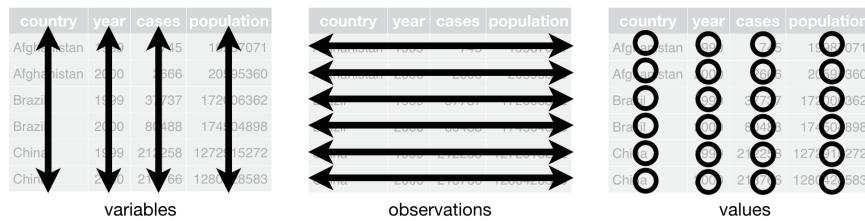


Figure 5.2: Principper af tidy data

Det er tilfældet, at de fleste af datasæt i dette kursus hører til “tidy data”, især i disse notater, hvor vi benytter en del af indbygget datasæt. Nogle gange kan det dog være, at vi er nødt til at gøre noget, til at lave et datasæt om til en “tidy datasæt”. R-pakker `dplyr` og `tidyverse` er velegnet til at hjælpe med at transformere et datasæt til en, der er “tidy”, og bagefter kan man forsætte i den sædvanlige måde med at analyse datasættet. Bemærk at bare fordi et datasæt er “tidy”, betyder det ikke nødvendigvis, at det er klart til at analysere, for der kan godt være, at man har bruge for at bearbejde videre på det første - igen med pakkerne `dplyr` og `tidyverse`.

## 5.5 Lidt om `tibbles`

En `tibble` er det `tidyverse` svar på en `data.frame` fra base-R. De ligner hinanden meget og derfor behøver man ikke tænk for meget over forskellen, men der er nogle opdateret aspekter i en `tibble` - for eksempel bruger en `tibble` ikke `row.names`, og når man visualiserer en `tibble` i R Markdown, få man lidt ekstra oplysninger, såsom dimensioner og data typer. Bemærk, at de fleste `tidyverse` funktioner fungerer lige så godt uanset om man har en `tibble` eller en `data.frame`. Bemærk, at jeg vedligeholder ordet ‘dataframe’ indenfor almindelig tekst.

Man kan lave sin egen `tibble` på samme måde som en `data.frame`.

```
tibble(x=1:3,y=c("a","b","c"))

## # A tibble: 3 x 2
##       x     y
##   <int> <chr>
## 1     1     a
## 2     2     b
## 3     3     c
```

Man kan også lave en `tribble`, som er den samme som en `tibble` men har en lidt anderledes måde at indsætte data på. For eksempel er følgende tilsvarende til den overstående tibble:

```
tribble(~x, ~y,
       1, "a",
       2, "b",
       3, "c")
```

```
## # A tibble: 3 x 2
##       x     y
##   <dbl> <chr>
## 1     1     a
## 2     2     b
## 3     3     c
```

Man kan lave en `data.frame` om til en `tibble` som i følgende:

```
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1         5.1       3.5       1.4      0.2  setosa
## 2         4.9       3.0       1.4      0.2  setosa
## 3         4.7       3.2       1.3      0.2  setosa
## 4         4.6       3.1       1.5      0.2  setosa
## 5         5.0       3.6       1.4      0.2  setosa
## 6         5.4       3.9       1.7      0.4  setosa
## 7         4.6       3.4       1.4      0.3  setosa
## 8         5.0       3.4       1.5      0.2  setosa
## 9         4.4       2.9       1.4      0.2  setosa
## 10        4.9       3.1       1.5      0.1  setosa
## # ... with 140 more rows
```

De fleste tidyverse koder fungerer lige så godt uanset om man har en `tibble` eller en `data.frame`.

## 5.6 Transition fra base til tidyverse

Jeg introducerer **tidyverse** gennem et meget berømt datasæt som hedder **titanic** - det er ikke biologiske data men er stadigvæk ret interessant, og sjovt at manipulere på.

**titanic** er brugt som en del af en åben konkurrence på hjemmesiden Kaggle, hvor mindst 31.000 personer indtil videre har arbejdet på at lave den bedste model til at forudsige, hvem der overlever katastrofen - linket er her, hvor du kan også læse om baggrunden til datasættet <https://www.kaggle.com/c/titanic>.

### 5.6.1 Om Titanic datasæt

Man kan downloade datasættet, der hedder **titanic\_train**, direkte fra Kaggle, men der er faktisk en R-pakke, der hedder **titanic** som gøre det mere bekvemt:

```
#install.packages("titanic") #hvis ikke allerede installerede
library(titanic)
```

Her er beskrivelsen for pakken:

*titanic is an R package containing data sets providing information on the fate of passengers on the fatal maiden voyage of the ocean liner “Titanic”, summarized according to economic status (class), sex, age and survival. These data sets are often used as an introduction to machine learning on Kaggle.*

Vi vil gerne bruge **titanic\_train** fordi det er datasættet, der bliver brugt på Kaggle til at træne maskinelærings modeller (som bliver testet på **titanic\_test** for at evaluere, hvor god modellen er). Til at gøre tingene nemmere, lad os bare omdøb **titanic\_train** til **titanic** og anvende **glimpse**, der er fra pakken **dplyr**, på datasættet.

```
titanic <- as_tibble(titanic_train)
glimpse(titanic)
```

```
## Rows: 891
## Columns: 12
## $ PassengerId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ~
## $ Survived <int> 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1~
## $ Pclass <int> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 2, 3, 3~
## $ Name <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (Fl~
## $ Sex <chr> "male", "female", "female", "male", "male", "mal~
## $ Age <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, 39, 14, ~
## $ SibSp <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 4, 0, 1, 0~
## $ Parch <int> 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0, 0, 0~
## $ Ticket <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "113803", "37~
## $ Fare <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51.8625, ~
## $ Cabin <chr> "", "C85", "", "C123", "", "", "E46", "", "", "", "G6", "C~
## $ Embarked <chr> "S", "C", "S", "S", "Q", "S", "S", "C", "S", "S", "C", "S", "S"~
```

Jeg har også kopieret de variable beskrivelser her:

- PassengerId: unique index for each passenger
- Survived: Whether or not the passenger survived. 0 = No, 1 = Yes.
- Pclass: Ticket class: 1 = 1st Class, 2 = 2nd Class, 3 = 3rd Class.
- Name: A character string containing the name of each passenger.
- Sex: Character strings for passenger sex (“male”/ “female”).
- Age: Age in years.
- SibSp: The number of siblings/spouses aboard the titanic with the passenger
- Parch: The number of parents/children aboard the titanic with the passenger
- Ticket: Another character string containing the ticket ID of the passenger.
- Fare: The price paid for tickets in pounds Sterling (Keep in mind that unskilled workers made around 1 pound a week - these were expensive tickets!)
- Cabin: The cabin number of the passengers (character).
- Embarked: Where passengers boarded the titanic. C = Cherbourg, Q = Queenstown, S = Southampton).

### 5.6.2 Titanic: oprydning

Der er faktisk nogle rengøring i datasættet vi skal tage os af, før vi kan komme videre med analysen. Vi kan se fra `glimpse(titanic)` ovenpå at der er 891 observationer. De fleste (687) passagerer har faktisk ingenting for variabel Cabin:

```
sum(titanic$Cabin=="") #ingenting for variabelen 'cabin'
```

```
## [1] 687
```

Andre har mere end én cabin. Det ser ikke særlig **tidy** ud, og man kan hellere ikke forestille sig at få meget insight fra variablen, så vi vælger at fjerne hele kolon med funktionen `select()`:

```
titanic_no_cabin <- select(titanic, -Cabin)
```

`select()` er en af de kerne funktioner i **tidyverse** - her angiver vi, hvilke kolonner vi gerne vil beholde eller fjerne fra datasættet. I dette tilfælde har vi specificeret `-Cabin`, som betyder, at vi *ikke* vil have variablen Cabin med, men gerne vil beholde resten af kolonnerne. Prøv selv at køre `select(titanic, Cabin)` i stedet for - så får vi kun Cabin og fjerner resten af vores variabler.

```
glimpse(titanic_no_cabin)
```

```
## Rows: 891
## Columns: 11
## $ PassengerId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ~
## $ Survived    <int> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1~
## $ Pclass      <int> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 2, 3, 3~
```

```

## $ Name      <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (F1-
## $ Sex       <chr> "male", "female", "female", "male", "male", "mal-
## $ Age        <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, 39, 14, ~
## $ SibSp     <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 0, 1, 0-
## $ Parch      <int> 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0, 0, 0-
## $ Ticket     <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "113803", "37-
## $ Fare       <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51.8625, ~
## $ Embarked   <chr> "S", "C", "S", "S", "Q", "S", "S", "C", "S", "S"-

```

Næste tjekker vi efter NA i datasættet. NA er hvordan R betegner manglende værdier. Man kan se i følgende, mens de fleste variabler ikke har NA værdier, har variablen Age 177 NA.

```
colSums(is.na(titanic_no_cabin))
```

|    | PassengerId | Survived | Pclass | Name | Sex      | Age |
|----|-------------|----------|--------|------|----------|-----|
| ## | 0           | 0        | 0      | 0    | 0        | 177 |
| ## | SibSp       | Parch    | Ticket | Fare | Embarked |     |
| ## | 0           | 0        | 0      | 0    | 0        |     |

I dette tilfælde vælger jeg at fjerne alle passagerer som har NA i stedet for deres alder. Til dette formål bruger jeg funktionen `drop_na`, som fjerner alle observationer, der har NA i mindst én variabel.

```
titanic_clean <- drop_na(titanic_no_cabin)
colSums(is.na(titanic_clean))
```

|    | PassengerId | Survived | Pclass | Name | Sex      | Age |
|----|-------------|----------|--------|------|----------|-----|
| ## | 0           | 0        | 0      | 0    | 0        | 0   |
| ## | SibSp       | Parch    | Ticket | Fare | Embarked |     |
| ## | 0           | 0        | 0      | 0    | 0        |     |

Nu kan vi tjekke igen, hvor mange observationer og variabel vi har tilbage.

```
glimpse(titanic_clean)
```

```

## Rows: 714
## Columns: 11
## $ PassengerId <int> 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19-
## $ Survived    <int> 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1-
## $ Pclass      <int> 3, 1, 3, 1, 3, 3, 2, 3, 1, 3, 3, 2, 3, 3, 2, 2, 3-
## $ Name        <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (F1-
## $ Sex         <chr> "male", "female", "female", "male", "male", "mal-
## $ Age         <dbl> 22, 38, 26, 35, 35, 54, 2, 27, 14, 4, 58, 20, 39, 14, 55, ~
## $ SibSp       <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 1, 0, 0, 0-
## $ Parch       <int> 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0, 0, 0, 0-
## $ Ticket      <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "113803", "37-
## $ Fare        <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51.8625, 21.0750-
## $ Embarked    <chr> "S", "C", "S", "S", "Q", "S", "S", "C", "S", "S"-

```

Vi har beholdt 714 observationer og 11 variabler, og datasættet opfylder kriteren for at være **tidy**.

### 5.6.3 Pipe

Man kan faktisk gøre den samme som i ovenstående ved at bruge pipe `%>%`:

```
titanic_clean <- titanic %>% # we take the titanic dataset
  select(-Cabin) %>% # select the bits we want
  drop_na() # then remove the NAs
```

Man bruger pipe `%>%` til at kombinere adskillige tidyverse funktioner i den samme kommando - linjen slutter med `%>%`, der fortæller, at vi skal bruge resultatet fra den linje som inputtet i den næste linje. Logikken er således, at vi starter med en dataframe, gør dernæst én ting ad gangen, og så slutter med en ny dataframe (som vi kan gemme med `<-`).

Bemærk, at processen ligner den, man bruger i **ggplot2**, men forskellen er at man bruger `%>%` i stedet for `+` i denne ramme. Bemærk også her, at ligesom i **ggplot2**, skriver jeg koden over flere linjer. Det er ikke et krav men det gøre det nemmere at læse og forstå koden.

For at illustrerer logikken, kan man se, at følgende to linjer er tilsvarende:

```
#take x and apply some function f
f(x)      #traditional approach
x %>% f #tidyverse approach
```

I begge tilfælde starter vi med `x`, og så anvender vi funktionen `f` med `x` som argument - en kæmpe fordel med den tidyverse løsning er, at når man har flere funktioner, slipper man for at anvende mange parenteser, og rækkefølgen man skriver funktionerne læses fra venstre til højre og ikke omvendt, se for eksempel følgende:

```
#take x, apply f, then apply g, then apply h
h(g(f(x)))          #traditional approach
x %>% f %>% g %>% h #tidyverse approach
```

På sammen måde i vores titanic oprydning kan man både pakke funktionen `select()` ind i funktionen `drop_na()`, eller bruge den **tidyverse** løsning, ligesom i nedenstående - de to giver det tilsvarende resultat: første bruger vi `select()` til at fjerne kolonnen `Cabin`, og så bruger vi `drop_na()` til at fjerne alle række med mindste én NA .

```
titanic_clean <- drop_na(select(titanic,-Cabin))

titanic_clean <- titanic %>%
  select(-Cabin) %>%
  drop_na()
```

## 5.7 Bearbejdning af data: `dplyr`

Pakken `dplyr` er nok den mest brugbare pakke til at bearbejde dataframes. Jeg gennemgår nogle af de mest almindelige muligheder med pakken, og der er også en “cheatsheet” som du kan downloade som reference: <https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf>. Jeg tager afsæt i følgende funktioner, og dækker flere gennem de forskellige øvelese og øvrige emner.

| dplyr verbs              | beskrivelse  |
|--------------------------|--|
| <code>select()</code>    | udvælge kolonner ( <i>variabler</i> )                                |
| <code>filter()</code>    | udvælge rækker ( <i>observationer</i> )                              |
| <code>arrange()</code>   | sortere rækker   |
| <code>mutate()</code>    | tilføje eller ændre eksisterende kolonner                            |
| <code>rename()</code>    | ændre variabler navne  |
| <code>recode()</code>    | ændre selve data   |
| <code>group_by()</code>  | dele datasættet op efter en variabel                                 |
| <code>summarise()</code> | aggregere rækker, findes ofte tilknyttet til <code>group_by()</code> |

Bemærk, at alle disse funktioner tager udgangspunkt i en dataframe, og man får altid en ny dataframe som outputtet. Ved at kunne bruge disse funktioner og kombinere dem (ved hjælp af `%>%`), har man godt styr på bearbejdningen af datarammer.

### 5.7.1 dplyr verbs: `select()`

Som vi lige har set i ovenstående, med `select()` udvælger man bestemte **variabler**. Vi kan vælge at beholde, fjerne eller andre rækkefølgen af variablerne i dataframe. Som eksempel, her beholder vi kun variablerne `Name` og `Age`:

```
titanic_clean %>%
  select(Name, Age) %>%
  glimpse()
```

```
## Rows: 714
## Columns: 2
## $ Name <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (Florence ~
## $ Age   <dbl> 22, 38, 26, 35, 35, 54, 2, 27, 14, 4, 58, 20, 39, 14, 55, 2, 31, ~
```

Hvis vi gerne vil fjerne en variabel fra en dataframe, kan vi bruge et minustegn. I nedenstående fjerner vi `Name` og `Age` fra dataframe:

```
titanic_clean %>%
  select(-Name, -Age) %>%
  glimpse()
```

```
## Rows: 714
```

```
## Columns: 9
## $ PassengerId <int> 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19~
## $ Survived <int> 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1~
## $ Pclass <int> 3, 1, 3, 1, 3, 1, 3, 2, 3, 1, 3, 3, 3, 2, 3, 3, 2, 2, 3~
## $ Sex <chr> "male", "female", "female", "female", "male", "male", "mal~
## $ SibSp <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 1, 0, 0, 0~
## $ Parch <int> 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0, 0, 0, 0~
## $ Ticket <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "113803", "37~
## $ Fare <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 51.8625, 21.0750~
## $ Embarked <chr> "S", "C", "S", "S", "S", "S", "S", "C", "S", "S", "S"~
```

### 5.7.1.1 Hjælper funktioner til select()

Hjælper funktioner til funktionen `select()` kan være brugbare hvis du gerne vil udvælge bestemte variabler efter nogle kriterier. Jeg har samlet nogle (men ikke alle mulige!) funktioner nedenfor og inddrager eksempler i problemstillingerne.

| select helper              | beskrivelse   |
|----------------------------|---|
| <code>starts_with()</code> | starts with a prefix  |
| <code>ends_with()</code>   | ends with a prefix  |
| <code>contains()</code>    | contains a literal string   |
| <code>matches()</code>     | matches a regular expression  |
| <code>num_range()</code>   | a numerical range like x01, x02, x03.   |
| <code>one_of()</code>      | variables in character vector.  |
| <code>everything()</code>  | all variables.  |
| <code>where()</code>       | fk. takes a function and returns all variables for which the function returns TRUE: |

For eksempel:

```
titanic_clean %>% select(starts_with("P"))
```

```
## # A tibble: 714 x 3
##   PassengerId Pclass Parch
##       <int> <int> <int>
## 1            1     3     0
## 2            2     1     0
## 3            3     3     0
## 4            4     1     0
## 5            5     3     0
## 6            6     7     1
## 7            7     8     3
## 8            8     9     3
## 9            9    10     2
```

```
## 10          11      3      1
## # ... with 704 more rows
```

Særligt brugbar i statistik statistisk metoder der kræver kun numeriske variabler er `where()` når kombinerede med `is.numeric`. Eksempelvis i følgende kode udvælger man kun numeriske variabler fra datasættet `titanic_clean`:

```
titanic_clean %>% select(where(is.numeric))

## # A tibble: 714 x 7
##   PassengerId Survived Pclass     Age SibSp Parch   Fare
##       <int>     <int> <int> <dbl> <int> <int> <dbl>
## 1           1         0     3    22     1     0    7.25
## 2           2         1     1    38     1     0   71.3
## 3           3         1     3    26     0     0    7.92
## 4           4         1     1    35     1     0   53.1
## 5           5         0     3    35     0     0    8.05
## 6           7         0     1    54     0     0   51.9
## 7           8         0     3    2   3     1   21.1
## 8           9         1     3    27     0     2  11.1
## 9          10         1     2    14     1     0  30.1
## 10          11         1     3     4     1     1 16.7
## # ... with 704 more rows
```

### 5.7.2 dplyr verbs: filter()

Med funktionen `select()` udvælger man bestemte variabler. Man anvender til gengæld funktionen `filter()` til at udvælge bestemte **observationer** (rækker) fra dataframe. I nedenstående beholder jeg rækkerne, hvor variablen `Age` er lig med 50. Bemærk, at vi bevarer alle variabler i dataframe.

```
titanic_clean %>%
  filter(Age == 50) %>%
  glimpse()

## #> #> Rows: 10
## #> #> Columns: 11
## #> #> $ PassengerId <int> 178, 260, 300, 435, 459, 483, 527, 545, 661, 724
## #> #> $ Survived <int> 0, 1, 1, 0, 1, 0, 1, 0, 1, 0
## #> #> $ Pclass <int> 1, 2, 1, 1, 2, 3, 2, 1, 1, 2
## #> #> $ Name <chr> "Isham, Miss. Ann Elizabeth", "Parrish, Mrs. (Lutie Davis)~"
## #> #> $ Sex <chr> "female", "female", "female", "male", "female", "male", "f~"
## #> #> $ Age <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50, 50
## #> #> $ SibSp <int> 0, 0, 0, 1, 0, 0, 0, 1, 2, 0
## #> #> $ Parch <int> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0
## #> #> $ Ticket <chr> "PC 17595", "230433", "PC 17558", "13507", "F.C.C. 13531", ~
## #> #> $ Fare <dbl> 28.7125, 26.0000, 247.5208, 55.9000, 10.5000, 8.0500, 10.5~
## #> #> $ Embarked <chr> "C", "S", "C", "S", "S", "S", "C", "S", "S"
```

Man kan også vælge intervaller - for eksempel hvis man vil vælge alle som er i halvtredserne.

```
titanic_clean %>%
  filter(Age >= 50 & Age < 60) %>%
  head()

## # A tibble: 6 x 11
##   PassengerId Survived Pclass Name                 Sex   Age SibSp Parch Ticket  Fare
##       <int>     <int> <int> <chr>             <chr> <dbl> <int> <int> <chr> <dbl>
## 1          7       0     1 "McCarthy, M~ male    54     0     0  17463  51.9
## 2         12      1     1 "Bonnell, Mi~ fema~  58     0     0 113783  26.6
## 3         16      1     2 "Hewlett, Mr~ fema~  55     0     0 248706   16
## 4         95      0     3 "Coxon, Mr. ~ male   59     0     0 364500  7.25
## 5        125      0     1 "White, Mr. ~ male   54     0     1 35281  77.3
## 6        151      0     2 "Bateman, Re~ male   51     0     0 S.O.P~ 12.5
## # ... with 1 more variable: Embarked <chr>
```

Man kan også kombinere betingelser fra forskellige kolonner, for eksempel i nedenstående vælger vi alle personer som er kvinder **og** som rejste ved første klasse.

```
titanic_clean %>%
  filter(Sex == 'female' & Pclass == 1) %>%
  head()

## # A tibble: 6 x 11
##   PassengerId Survived Pclass Name           Sex     Age SibSp Parch Ticket Fare
##       <int>     <int> <int> <chr>        <chr> <dbl> <int> <int> <chr> <dbl>
## 1          2       1     1 Cumings, Mrs~ fema~    38     1     0 PC 17~  71.3
## 2          4       1     1 Futrelle, Mr~ fema~    35     1     0 113803  53.1
## 3         12       1     1 Bonnell, Mis~ fema~    58     0     0 113783  26.6
## 4         53       1     1 Harper, Mrs.~ fema~    49     1     0 PC 17~  76.7
## 5         62       1     1 Icard, Miss.~ fema~    38     0     0 113572   80
## 6         89       1     1 Fortune, Mis~ fema~    23     3     2 19950   263
## # ... with 1 more variable: Embarked <chr>
```

Vi kan også inddrage flere symboler. For eksempel i nedenståenden vælger vi personer som er kvinder **og** som rejste i **enten** første eller anden klasse **og** som er **i** trediverne. Huske at tilføje runde parenteser omkring de to **Pclass** - prøv selv at fjerne dem og se, hvad der sker.

```
titanic_clean %>%
  filter(Sex == 'female' & (Pclass == 1 | Pclass == 2) & Age %in% c(30:39)) %>%
  glimpse()

## #> #> #> #> #>
```

### 5.7.3 Comparitive reference

Her er en tabel af comparitiver (kopirede fra grundlæggende R) - de er brugbare i både `filter()` og i baseR (fordi koncepten bag er den samme, bare tilgang er ænderedes).

| comparitive | beskrivelse              |
|-------------|--------------------------|
| <           | less than                |
| >           | greater than             |
| <=          | less than or equal to    |
| >=          | greater than or equal to |
| ==          | equal to                 |
| !=          | not equal to             |
| &           | and                      |
| %in%        | in                       |
|             | or                       |

#### 5.7.4 Kombinere filter() og select()

Man kan også kombinere både `filter()` og `select()` i samme kommando, som i følgende:

```
titanic_clean %>%
  filter(Sex == 'female' & (Pclass == 1 | Pclass == 2) & Age %in% c(30:39)) %>%
  select(Name, Fare)  %>%
  glimpse()
```

```
## Rows: 43  
## Columns: 2  
## $ Name <chr> "Cumings, Mrs. John Bradley (Florence Briggs Thayer)", "Futrelle,~  
## $ Fare <dbl> 71.2833, 53.1000, 80.0000, 23.0000, 13.0000, 21.0000, 113.2750, 7~
```

Bemærk at man bør passe på rækkefølgen, som man anvender de forskellige funktioner. For eksempel hvis man bytter rundt `filter()` og `select()` i ovenstående, få man en advarsel - prøv selv at køre følgende:

```
##virker ikke!!!!#####
titanic_clean %>%
  select(Name, Fare) %>%
  filter(Sex == 'female' & (Pclass == 1 | Pclass == 2) & Age %in% c(30:39)) %>%
  glimpse()
```

Det er fordi, hvis man første vælger at beholde variablerne `Name` og `Age`, så findes de andre variabler ikke mere i de resulterende dataframe, som bliver dernæst brugt i funktionen `filter()` - man kan derfor ikke benytte funktionen `filter()` på variablerne `Pclass`, `Sex` og `Age`.

### 5.7.5 dplyr verbs: `mutate()`

Man kan avende funktionen `mutate()` til at tilføje en ny variable til en dataframe. I nedenstående tilføjer jeg en ny variabel med navnet `Adult`, der angiver om personen kan betragtes som en voksen (hvis vedkommende er mindst 18 år gammel).

```
titanic_with_Adult <- titanic_clean %>%
  mutate(Adult = Age>=18)

titanic_with_Adult %>% select(Adult) %>% glimpse

## Rows: 714
## Columns: 1
## $ Adult <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, FALSE, T~
```

#Så kan man se, at der er 601 voksne og 113 børn som passagerere på skibet.

Bemærk her, at jeg gemmer resultatet som en ny dataframe, der hedder `titanic_with_Adult`, og derefter bruger jeg `glimpse()` på det nye objekt `titanic_with_Adult` for at se, hvordan min nye dataframe ser ud. I forudgående eksempler havde jeg ikke gemt resultatet - bare brugt `glimpse()` for at se resultatet på skærmen. **Hvis du gerne vil bruge din resulterende dataframe videre, så skal du husk at gemme den (med brugen af <- tegn)**

#### funktionen `ifelse()` indenfor `mutate()`

Jeg kan oprette variablen `Adult` sådan at den er mere informativ end bare `TRUE` eller `FALSE`. Jeg anvender funktionen `ifelse()`, der giver mulighed for at angive, at jeg gerne vil have teksten "adult" hvis udsagnet `Age>=18` er `TRUE`, og hvis `FALSE` vil jeg have teksten "child":

```
ifelse(Age>=18, "adult", "child")
```

Funktionen `ifelse()` bruges indenfor funktionen `mutate()`, fordi vi er i gang med at oprette en ny variable `Adult` - `ifelse()` giver bare mulighed for at fortælle, hvordan den nye variablen skal ser ud.

```
titanic_clean %>%
  mutate(Adult = ifelse(Age>=18,"adult","child")) %>%
  select(Age,Adult) %>%
  glimpse()

## Rows: 714
## Columns: 2
## $ Age    <dbl> 22, 38, 26, 35, 35, 54, 2, 27, 14, 4, 58, 20, 39, 14, 55, 2, 31, ~
## $ Adult   <chr> "adult", "adult", "adult", "adult", "adult", "adult", "child", "~
```

Så er variablen lidt mere informativ end før.

### 5.7.6 `rename()`

Man kan bruge `rename()` til at ændre navnet på en eller flere variabler i datasættet. Som eksempel bruger jeg `rename()` til at give en variable navnet `Years` i stedet for `Age` (bemærk at variablen `Age` findes ikke længere).

```
titanic_clean %>%
  rename(Years = Age) %>%
  glimpse()

## Rows: 714
## Columns: 11
## $ PassengerId <int> 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19~
## $ Survived     <int> 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1~
## $ Pclass       <int> 3, 1, 3, 1, 3, 3, 2, 3, 1, 3, 3, 2, 3, 3, 2, 2, 3~
## $ Name         <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (Fl~
## $ Sex          <chr> "male", "female", "female", "male", "male", "mal~
## $ Years        <dbl> 22, 38, 26, 35, 35, 54, 2, 27, 14, 4, 58, 20, 39, 14, 55, ~
## $ SibSp        <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 1, 0, 0~
## $ Parch        <int> 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0, 0, 0~
## $ Ticket       <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "113803", "37~
## $ Fare         <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 51.8625, 21.0750~
## $ Embarked     <chr> "S", "C", "S", "S", "S", "S", "S", "C", "S", "S", "S"~
```

Man kan også ændre navne på flere kolonner samtidigt - for eksempel, i følgende laver jeg nogle oversættelsesarbejde:

```
titanic_clean_dansk <- titanic_clean %>%
  rename(Overlevede = Survived,
        Navn = Name,
        Klasse = Pclass)
```

Så du kan se, at jeg har ændrede de variabler navne. Jeg kalder den nye dataframe for `titanic_clean_dansk`, så jeg min danske udgave er blevet gemt et sted.

Man kan også gøre sådan, at man har kun små bogstaver i de variabler navne. Jeg benytter den danske version, og Jeg anvender `rename_with()` og specificerer `to_lower`.

```
titanic_clean_dansk %>%
  rename_with(tolower)  %>% #all variable names are lower case only
  glimpse()

## Rows: 714
## Columns: 11
## $ passengerid <int> 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19-
## $ overlevede <int> 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1-
## $ klasse      <int> 3, 1, 3, 1, 3, 1, 3, 2, 3, 1, 3, 3, 3, 2, 3, 3, 2, 2, 3-
## $ navn        <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (Fl-
## $ sex         <chr> "male", "female", "female", "female", "male", "male", "male-
## $ age         <dbl> 22, 38, 26, 35, 35, 54, 2, 27, 14, 4, 58, 20, 39, 14, 55, ~
## $ sibsp       <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 1, 0, 0, 0-
## $ parch       <int> 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0, 0, 0, 0-
## $ ticket      <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "113803", "37-
## $ fare        <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 51.8625, 21.0750-
## $ embarked    <chr> "S", "C", "S", "S", "S", "S", "S", "S", "C", "S", "S", "S"-
```

Prøv også at erstatte `tolower` med `toupper`

### 5.7.7 dplyr verbs: `recode()`

Med `recode()` kan man ændre hvordan en variable ser ud - f.eks. male/female kan ændres til 0/1, som i følgende.

Bemærk den måde, at funktionen `recode()` er blevet brugt indenfor funktionen `mutate()`: jeg lavede en ny variable af samme navn, men med ændret værdier indenfor variablen.

Hvis vi gerne vil skifter omvendt fra 0/1 til male/female er vi nødt til at skrive 1 / 0 for at specificie at vi har værdier som er tal, og vi gerne vil kalde dem for nogle andet ("male"/"female" i dette tilfælde):

```
#recodes variable Sex and then recodes it back to original form again  
titanic_clean %>%
```

```
mutate(Sex = recode(Sex, male = 1, female = 0)) %>%
  mutate(Sex = recode(Sex, `1` = "male", `0` = "female")) %>% #note use of `` in the
  select(PassengerId,Name,Sex) %>% glimpse()

#> #> Rows: 714
#> #> Columns: 3
#> #> $ PassengerId <int> 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19~
#> #> $ Name      <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bradley (Fl~
#> #> $ Sex       <chr> "male", "female", "female", "female", "male", "male", "mal~
```

### 5.7.8 dplyr verbs: `arrange()`

Man anvender `arrange()` for at vælge rækkefølgen på observationerne. I nedenstående tager vi datarammen `titanic_clean` og arrangerer observationer efter variablen `Fare`. Det sker således at, personer som betalt mindst er på toppen af de resultarende dataramme, og personer som betalt mest er på bunden.

```
# Arrange by increasing Fare
titanic_clean %>%
  arrange(Fare) %>%
  glimpse()
```

Hvis man gerne vil få det omvendt - at personer som betalt mest er på toppen af datarammen, kan man bruge `desc()` omkring `Fare`, som i nedenstående:

```
# Arrange by decreasing Fare
titanic_clean %>%
  arrange(desc(Fare)) %>%
  glimpse()
```

```
## Rows: 714  
## Columns: 11  
## $ PassengerId <int> 259, 680, 738, 28, 89, 342, 439, 312, 743, 119, 300, 381, ~
```

```

## $ Survived    <int> 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1~
## $ Pclass      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ Name        <chr> "Ward, Miss. Anna", "Cardeza, Mr. Thomas Drake Martinez", ~
## $ Sex          <chr> "female", "male", "male", "male", "female", "female", "mal~
## $ Age          <dbl> 35.00, 36.00, 35.00, 19.00, 23.00, 24.00, 64.00, 18.00, 21~
## $ SibSp        <int> 0, 0, 0, 3, 3, 3, 1, 2, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1~
## $ Parch        <int> 0, 1, 0, 2, 2, 2, 4, 2, 2, 1, 1, 0, 0, 0, 2, 1, 0, 1, 2, 1~
## $ Ticket       <chr> "PC 17755", "PC 17755", "PC 17755", "19950", "19950", "199~
## $ Fare          <dbl> 512.3292, 512.3292, 512.3292, 263.0000, 263.0000, 263.0000~
## $ Embarked     <chr> "C", "C", "C", "S", "S", "S", "C", "C", "C", "C", "C"~

```

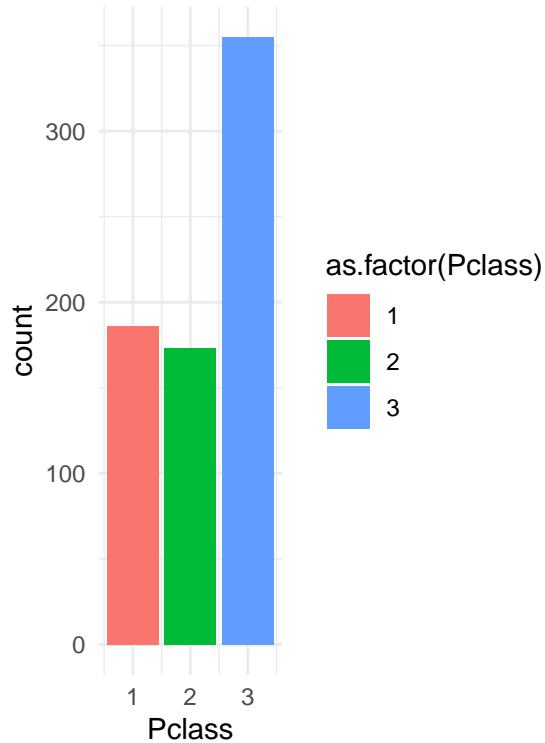
## 5.8 Visualisering: bruge som input i ggplot2

Efter man har lavet bearbejdning med tidyverse kommandoer, kan man specifice de resulterende dataframe som data i funktionen `ggplot()`. Man benytter `%>%` til at forbinde de `dplyr` kommandoer med den `ggplot` funktion, og i dette tilfælde behøver man ikke at angive navnet på datasættet indenfor funktionen `ggplot`. I nedenstående eksempel tager jeg udgangspunkt i `titanic_clean` og så laver jeg et barplot som viser antallet af passagerer som rejste i hver af de tre klass.

```

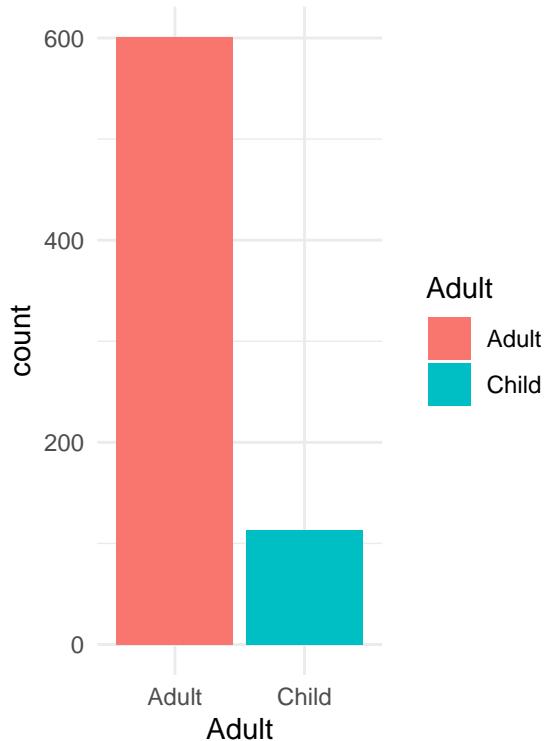
titanic_clean %>%
  ggplot(aes(x=Pclass, fill=as.factor(Pclass))) +
  geom_bar(stat="count") +
  theme_minimal()

```



Jeg gør det lidt mere kompliceret i følgende, ved at tage `titanic_clean`, lave en ny kolon der hedder `Adult`, og så bruge den resulterende dataframe med funktionen `ggplot`, hvor jeg laver et plot med `Adult` på x-aksen for at tælle op antallet af voksne og børn.

```
titanic_clean %>%
  mutate(Adult = ifelse(Age>=18, "Adult", "Child")) %>%
  ggplot(aes(x=Adult, fill=Adult)) +
  geom_bar(stat="count") +
  theme_minimal()
```



Så viser det, at der var 600 Adults og lidt over 100 Children ombord skibet.

## 5.9 Misc funktioner som er nyttige at vide

### 5.9.1 Pull

I tidyverse arbejder vi meget med dataframes - tilgagen er således at man tager udgangspunkt i en dataframe, får en dataframe som resultat og så arbejde videre på den dataframe. Nogle gange kan det dog være at man gerne vil udtrække en variabel som vector fra en dataframe, f.eks. hvis man gerne vil bruge den i en bestemt statistik metode.

Se følgende eksempel, hvor man udtrækker variable `Age` for "male" og "female" (variablen `Sex`) og bruge resulterende vectorer i en t-test sammenhæng:

```
ages_male <- titanic_clean %>% filter(Sex=="male") %>% pull(Age)
ages_female <- titanic_clean %>% filter(Sex=="female") %>% pull(Age)
t.test(ages_male,ages_female)

##
## Welch Two Sample t-test
##
## data: ages_male and ages_female
```

```
## t = 2.5259, df = 560.05, p-value = 0.01181
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## 0.6250732 4.9967983
## sample estimates:
## mean of x mean of y
## 30.72664 27.91571
```

Så kan man se at mænd og kvinder har signifikant forskellige alder i gennemsnit (hvor mændene er ældre en kvinderne).

### 5.9.2 Slice

Med funktionen `slice` kan man kigge på nogle bestemte observationer, for eksempel, viser følgende de to passagerer der betalt mest for billeten (variable `Fare`).

```
titanic %>%
  arrange(desc(Fare)) %>%
  select(Name, Age) %>%
  slice(1, 2)

## # A tibble: 2 x 2
##   Name           Age
##   <chr>          <dbl>
## 1 Ward, Miss. Anna     35
## 2 Cardeza, Mr. Thomas Drake Martinez 36
```

Se udvidet muligheder her: <https://dplyr.tidyverse.org/reference/slice.html>

## 5.10 Problemstillinger

**Problem 1)** Lav quizzen på Absalon - “Quiz - tidyverse - part 1”

*Vi øver os med titanic. Inlæs datasættet og lav overstående oprydningen med følgende kode:*

```
library(tidyverse)
library(titanic)
titanic <- as_tibble(titanic_train)

titanic_clean <- titanic %>%
  select(-Cabin) %>%
  drop_na() %>%
  mutate(Adult = ifelse(Age>=18,"adult","child")) %>%
  mutate(Survived = recode(Survived, `1` = "yes", `0` = "no"))
```

```
glimpse(titanic_clean) #take a look!
```

---

**Problem 2)** `select()`. Fjern variablen `Name` fra `titanic_clean` (du behøver ikke at gemme din nye dataframe).

```
titanic_clean %>%
  select(...) #redigere her
```

- Tilføj også `glimpse()` for at se et overblik (man kan også bruge `head()`)
- 

**Problem 3)** `select()`. Lave en ny dataframe fra `titanic_clean` med kun variabler `Name`, `Pclass` og `Fare`.

- Gør det en forskel, hvilke rækkefølger man skriver `Name`, `Pclass` og `Fare`?
- 

**Problem 4)** `select()` og hjælper funktioner. I stedet for at specificere bestemt kolonner navn, skriv `starts_with("S")` indenfor `select()`. Hvad sker der?

- Prøv også `contains("ar")`
  - Prøv også `-any_of(c("Survived", "Pclass", "FavouriteColour"))` og `-all_of(c("Survived", "Pclass", "FavouriteColour"))`
    - i tilfældet af `all_of` skal alle variablerne i vectoren `c("Survived", "Pclass", "FavouriteColour")` være i datasættet, ellers får man en advarsel.
    - i tilfældet af `any_of` gælder det alle variabler fra vectoren `c("Survived", "Pclass", "FavouriteColour")` der er i datasættet, og resten bliver ignoreret.
  - Prøv også `matches("^S[i|u] ")` - kan du gisner på hvad det betyder (se nedenunder)?
- 

**Problem 5)** `filter()`. Lave en ny dataframe fra `titanic_clean` med alle passagerer som er mellem 10 og 15 og rejst enten første eller anden klasse.

- Prøv at tilføje `%>% count()` til kommandoen - Hvor mange observationer er der i den nye dataframe?
- 

**Problem 6)** `filter()` og `select()` : kombinering med `%>%`

Lave en ny dataramme fra `titanic_clean` med alle passagerer som er “male” og overlevede (variablen `Survived` er “yes”), og udvælg kun kolonner `Name`, `Age` og `Fare`.

---

**Problem 7) filter() og select() kombinering med %>%**

Lave en ny dataramme fra `titanic_clean` med kun variabler `Name` og `Age` og dernæst specificere kun de passagerer som er over 60.

- Få man så den samme sæt observationer hvis du skriver dine `select()` og `filter()` funktionerne omvendt her? Hvorfor?
- 

**Problem 8) Mutate().** Lave en ny dataframe fra `titanic_clean` som hedder `FareRounded` og viser `Fare` rundet til det nærmest integer (hint: benyt funktionen `round()`).

**Problem 9) Mutate() og ifelse().** Lave en ny dataramme fra `titanic_clean` med en ny kolon som hedder `Family` som angiver TRUE hvis `Parch` er ikke nul, ellers FALSE.

- Anvende `ifelse` til at gøre variablen mere intuitiv - “Family” og “Not family”.
- 

**Problem 10) Mutate() og ifelse()**

Kig en gang til på beskrivelsen af følgende to variabler i datasættet:

*SibSp: The number of siblings/spouses aboard the titanic with the passenger*

*Parch: The number of parents/children aboard the titanic with the passenger*

- Lav en ny variabel `Solo` som viser “Yes” hvis passageren rejste alene, og “No” hvis passageren rejste med andre.
- Brug `mutate` igen til at lave den nye variabel om til at være en factor.
- Gem også dit output (som `titanic_clean` igen) så du kan bruge din nye variable videre i næste spørgsmål.

```
titanic_clean <- titanic_clean %>% ...
```

---

**Problem 11) pull() og t.test()** Betalte passagererne der rejste alene (variablen `Solo` fra sidste problem) den samme i gennemsnit for deres billet (variablen `Fare`) end passagererne der ikke rejste alene? Lav et `t.test` (anvend `pull()` til at udtrække hensigtsmæssige vectorer - se også eksempel i kursusnotaterne)

```
t.test(titanic_clean %>% filter(Solo=="Yes") %>% pull(Fare),
       titanic_clean %>% filter(Solo=="No") %>% pull(Fare) )
```

```
##  
## Welch Two Sample t-test
```

```

## 
## data: titanic_clean %>% filter(Solo == "Yes") %>% pull(Fare) and titanic_clean %>% filter(Solo == "No")
## t = -6.9703, df = 573.64, p-value = 8.724e-12
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -35.57536 -19.93377
## sample estimates:
## mean of x mean of y
## 22.64421 50.39878

```

**Problem 12)** *Recode()* I variablen Embarked:

- C står for Cherbourg
- Q står for Queenstown
- S står for Southampton

- a) Anvend `recode` (indenfor `mutate`) til at ændre værdierne i variablen `Embarked` således at man får de fulde navne af de steder folk gik ombord skibet, i stedet for kun den første bogstav. Gem også dit output (som `titanic_clean` igen) så du kan bruge din nye variable videre.
- b) Erstat `recode` med `recode_factor` og sammenlign datatypen af variablen `Embarked` i din nye dataframe.
- c) Prøv at tilføje funktionen `count()` for at tælle op hvor mange gik om bord i de forskellige steder.

- Prøv også med to variabler indenfor `count()` - Solo og `Embarked`

Resultatet ser sådan ud:

```

## # A tibble: 7 x 3
##   Solo   Embarked      n
##   <fct> <fct>     <int>
## 1 No    "Southampton" 229
## 2 No    "Queenstown"   9
## 3 No    "Cherbourg"    72
## 4 Yes   "Southampton" 325
## 5 Yes   "Queenstown"  19
## 6 Yes   "Cherbourg"    58
## 7 Yes   ""              2

```

- d) Man kan se, at der er to passagerer hvor der ikke er noget skrevet i `Embarked`. + Rejste de alene? + Lav en ny dataframe med de to passagerer fjernet fra datasættet.

**Problem 13)** *Arrange()*. Lave en ny dataramme fra `titanic_clean` med observationerne arrangerede således at de yngst er på toppen og ældste er på

bunden. Kig på resultatet - hvad kan du fortælle om den yngste passager ombord skibet Titanic?

- Hvad kan du fortælle om den ældste passager ombord skibet? Overlevede de? Hvad med de andre ældste passagerer?
- 

**Problem 14)** *Arrange() og kombinering med andre verber.* Lave en ny dataramme fra `titanic_clean` med kun personer med `SibSp>0` og som gik ombord skibet i Southampton, arrangere de resulterende observationer efter `Fare` (højeste på toppen) og udvælg kun kolonnerne `Name`, `Age` og `Fare`.

---

**Problem 15)** *Rename.* Fra `titanic_clean` udvælg kun variabler `Survived`, `Ticket`, og `Name` og ændre deres navne til `Overlevede`, `Billet` og `Navn`.

- Gør variabler navne til store bogstaver ved at anvende `rename_with()`.
- 

**Problem 16)** *Lave et plot.* Fra `titanic_clean` bruge `filter()` til at lave en ny dataramme kun med personer under 30 og bruge den til at lave et barplot som viser antallet af personer opdelt efter `Pclass`. Brug følgende struktur for koden:

```
titanic_clean %>%
  filter(...) %>% #rediger linjen
  ggplot(aes(...)) + .... #tilføj plot
```

---

**Problem 17)** *Lave et plot.* Fra `titanic_clean`, bruge `mutate()` til at lave et nyt kolon der hedder `with_siblings_spouses` der er TRUE hvis `SibSp` ikke er nul. Brug den til at lave boxplots som viser `Fare` på y-aksen og `with_siblings_spouses` på x-aksen.

- Ekstra: Ændre skalen på y-aksen for at gøre plottet klarer at fortolke.

## 5.11 Kommentarer

- `matches("S[i|u]")` betyder
  - `S` variabel navn skal starte med en `S`
  - `[i|u]` den næste bogstav i variabel navnet skal være enten `i` eller `u`
- OBS det er ikke vigtigt at lære pattern matching i kurset men det er meget brugbart i andre sammenhænge!

Næste gange arbejder vi videre med tidyverse.

- `Group_by` kombinerede med `Summarise`

- Pivot\_Longer/Pivot\_Wider
- Join funktionerne



# Chapter 6

## Bearbejdning dag 2



### 6.1 Indledning og læringsmålene

I dag arbejder du videre med `tidyverse`, især på pakken `dplyr` og `tidyr`, som kan bruges til at ændre på strukturen af et datasæt, således at det passer til den struktur, som kræves for at blandt andet lave plots med `ggplot2`.

Det er ofte tilfældet indenfor biologi, at man har sit data i et dataframe og nogle ekstra sample oplysninger i en anden dataframe. Derfor vil vi gerne have en måde, at forbinde de to dataframes i R, som gør, at vi kan inddrage de ekstra oplysninger når vi lave plots af de data.

#### 6.1.1 Læringsmålene

Du skal være i stand til at

- Benytte kombinationen af `group_by()` og `summarise()`.
- Forstå forskellen mellem `wide` og `long` data og bruge `pivot_longer()` til at facilitere plotting
- Benytte `left_join()` eller øvrige join funktioner til at tilføje sample information til datasættet.

### 6.1.2 Videoer

- Video 1 - vi skal kig lidt nærmere på `group_by()` + `summarise()` og forbinde `tidyverse` kode og `ggplot2` kode sammen med `%>%/+`.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/546910681>

---

- Video 2 - wide/long data forms og `pivot_longer()` og bruge den i `ggplot2`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/707081191>

---

- Video 3 - eksempel med titanic summary statistics

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/707223997>

---

- Video 4: `left_join()` of tables with extra sample information and plot

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/707082269>

---

## 6.2 `group_by()` med `summarise()` i `dplyr`-pakken

Med kombination af `group_by()` med `summarise()` kan man finde numeriske svar på spørgsmålet: havde mænd eller kvinder en højre sandsynlighed for at overleve tragedien?

Lad os starte med løsningen med `tapply` til at udregne proportionen af mænd og kvinder der overlevede: følgende kode svarer til, at man opdeler variablen `Survived` efter den katagoriske variable `Sex` og tager middelværdien. Det giver dermed proportionen der overlevede efter køn (da `Survived` er kodet sådan at 1 betyder at man overlevede og 0 betyder at man ikke overlevede).

```
titanic_clean <- titanic %>%
  select(-Cabin) %>%
  drop_na()

#tapply løsning
tapply(titanic_clean$Survived, titanic_clean$Sex, mean)

##      female      male
## 0.7547893 0.2052980
```

Lad os skifter over til den `tidyverse` løsning. Lad os tage udgangspunkt i `summarise()`: som et eksempel af hvordan man bruger funktionen, vil vi beregne en variable der hedder "medianFare" som er lig med `median(fare)`.

```
titanic_clean %>%
  summarise("medianFare" = median(Fare))

## # A tibble: 1 x 1
##   medianFare
##       <dbl>
## 1      15.7
```

Vi får faktisk en ny dataramme her, med kun variablen som vi lige har specifiseret. Vi er interesseret i proportionen, der overlevede, så vi tager middelværdien af variablen `Survived`. Lad os gøre det med `summarise()`:

```
titanic_clean %>%
  summarise(meanSurvived = mean(Survived))

## # A tibble: 1 x 1
##   meanSurvived
##       <dbl>
## 1      0.406
```

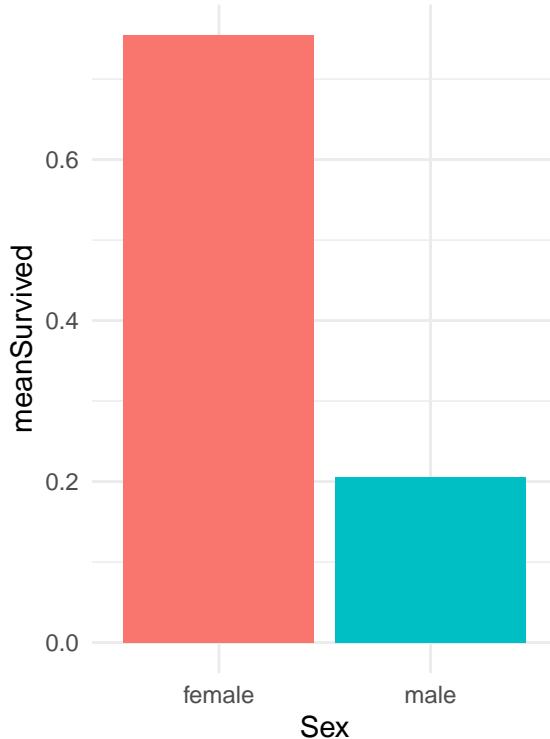
Få at svare på spørgsmålet er vi også nødt til at opdele efter kolonnen `Sex`. Vi kan bruge den kombinerende af `group_by()` og `summarise()` - vi opdele efter `Sex` ved at anvende funktionen `group_by()` og derefter bruger `summarise()` til at oprette en kolon der hedder `meanSurvived`, der viser proportionen der overlevede for female and male.

```
#tidyverse løsning
titanic_clean %>%
  group_by(Sex) %>%
  summarise(meanSurvived = mean(Survived))

## # A tibble: 2 x 2
##   Sex   meanSurvived
##   <chr>     <dbl>
## 1 female    0.755
## 2 male      0.205
```

Lad os tage resultatet fra ovenpå og visualiserer det i et barplot, som i nedenstående:

```
titanic_clean %>%
  group_by(Sex) %>%
  summarise(meanSurvived = mean(Survived)) %>%
  ggplot(aes(x=Sex,y=meanSurvived,fill=Sex)) +
  geom_bar(stat="identity",show.legend = FALSE) + theme_minimal()
```



### 6.2.1 Reference af `summarise()` funktioner

Nogle funktioner man ofte bruge med `summarise()` (der er mange andre muligheder).

| funktion             | beskrivelse  |
|----------------------|--|
| <code>mean()</code>  | to give us the mean value of a variable.                           |
| <code>sd()</code>    | to give us the standard deviation of a variable.                   |
| <code>min()</code>   | giving us the lowest value of a variable.                          |
| <code>max()</code>   | giving us the highest value of a variable.                         |
| <code>n()</code>     | giving us the number of observations in a variable. and many more. |
| <code>first()</code> | first values   |

### 6.2.2 Flere summary statistic på én gang

Vi kan også lave flere summary statistics på én gang. For eksempel, lad os anvende funktionen `group_by` med `Sex` igen, men beregner flere forskellige summary statistics:

```
titanic_clean_summary_by_sex <- titanic_clean %>%
  group_by(Sex) %>%
```

```
summarise(count = n(),                                     #count
          meanSurvived = mean(Survived),      #middelværdi survived
          meanAge = mean(Age),                #middelværdi age
          propFirst = sum(Pclass==1)/n())    #proportionen i første klasse
titanic_clean_summary_by_sex
```

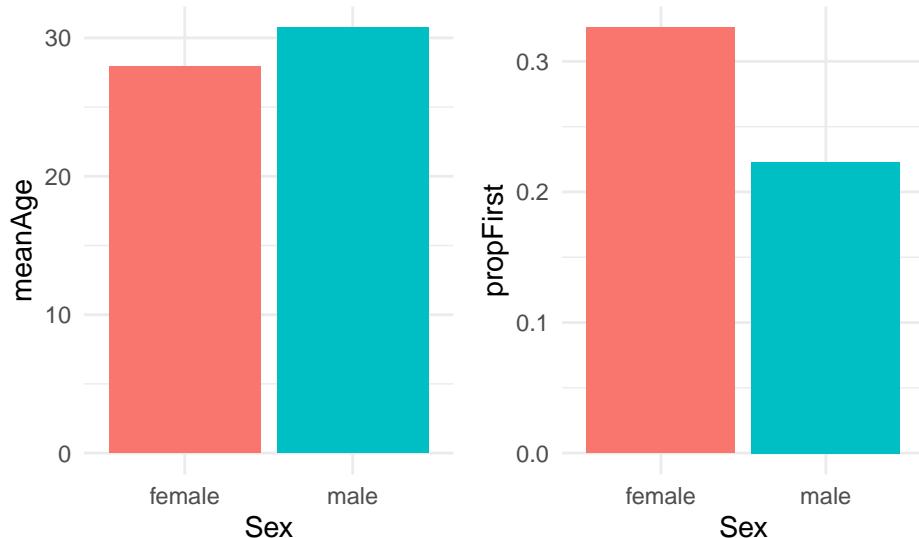
```
## # A tibble: 2 x 5
##   Sex     count meanSurvived meanAge propFirst
##   <chr>   <int>      <dbl>     <dbl>      <dbl>
## 1 female    261       0.755     27.9      0.326
## 2 male      453       0.205     30.7      0.223
```

Igen kan denne summary table bruges som et datasæt til at lave et plot med ggplot2. Bemærk at her bruger vi `stat="identity"`, fordi vi skal ikke tælle observationerne op, men bare plot præcis de tal som er i datarammen på y-aksen. I nedenstående laver vi barplots for `meanAge` og `propFirst` - de er plottet ved at bruge to forskellige ggplot kommandoer og bemærk, at det er plottet ved siden af hinanden med en funktion der hedder `grid.arrange()` fra R-pakken `gridExtra`.

```
plotA <- ggplot(data=titanic_clean_summary_by_sex,aes(x=Sex,y=meanAge,fill=Sex)) +
  geom_bar(stat="identity",show.legend = FALSE) +
  theme_minimal()

plotB <- ggplot(data=titanic_clean_summary_by_sex,aes(x=Sex,y=propFirst,fill=Sex)) +
  geom_bar(stat="identity",show.legend = FALSE) +
  theme_minimal()

library(gridExtra)
grid.arrange(plotA,plotB,ncol=2) #plot both together
```



Vi kan se, at females var i gennemsnit lidt yngere end males, og havde en højere sandsynlighed for at være i første klasse. Et interessant spørgsmål er, hvordan man kan lave ovenstående plots uden at bruge to forskellige `ggplot` kommandoer - altså, en automatiske løsning hvor vi kan plotte flere summary statistiks med kun én `ggplot` kommando. Vi kommer til at se hvordan man gøre det med at første lave datasættet om til long form.

### 6.2.3 Mere kompliceret `group_by()`

Lad os også beregne hvor mange passagerer der var efter både deres klasse, og hvor de gik ombord skibet:

```
titanic_clean %>%
  group_by(Embarked, Pclass) %>% # group by multiple variables...
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'Embarked'. You can override using the
## `.` groups` argument.

## # A tibble: 10 x 3
## # Groups:   Embarked [4]
##   Embarked Pclass count
##   <chr>     <int> <int>
## 1 ""        1      2
## 2 "C"       1      74
## 3 "C"       2      15
## 4 "C"       3      41
## 5 "Q"       1      2
## 6 "Q"       2      2
## 7 "Q"       3     24
```

```
## 8 "S"           1   108
## 9 "S"           2   156
## 10 "S"          3   290
```

Man kan se at de flest gik om bord i Southampton (S), men der var også forholdsvis mange første klasse passagerer der gik om bord i Cherbourg (C). Lad os gå videre med vores `Survived` eksempel og beregne proportionen der overlevede efter de tre variabler `Adult`, `Sex` og `Pclass`.

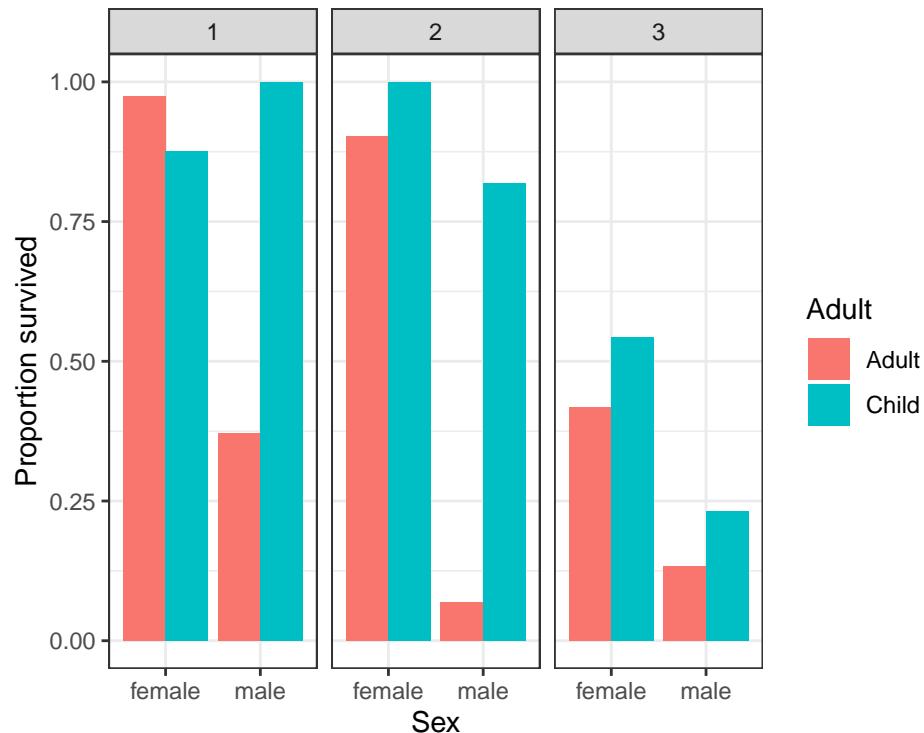
```
titanic_clean_summary_survived <- titanic_clean %>%
  mutate(Adult = ifelse(Age>=18,"Adult","Child")) %>%
  group_by(Adult,Sex,Pclass) %>%
  summarise(meanSurvived = mean(Survived))
```

```
## `summarise()` has grouped output by 'Adult', 'Sex'. You can override using the
## `.`groups` argument.
titanic_clean_summary_survived
```

```
## # A tibble: 12 x 4
## # Groups:   Adult, Sex [4]
##   Adult Sex     Pclass meanSurvived
##   <chr> <chr>   <int>        <dbl>
## 1 Adult female    1      0.974
## 2 Adult female    2      0.903
## 3 Adult female    3      0.418
## 4 Adult male      1      0.371
## 5 Adult male      2      0.0682
## 6 Adult male      3      0.133
## 7 Child female    1      0.875
## 8 Child female    2      1
## 9 Child female    3      0.543
## 10 Child male     1      1
## 11 Child male     2      0.818
## 12 Child male     3      0.233
```

Og så kan vi også bruge resultatet ind i en `ggplot`, hvor vi kombinerer de tre variabler og adskiller efter `Pclass`:

```
ggplot(titanic_clean_summary_survived,aes(x=Sex,y=meanSurvived,fill=Adult)) +
  geom_bar(stat="identity",position = "dodge") +
  facet_grid(~Pclass) +
  ylab("Proportion survived") +
  theme_bw()
```



#### 6.2.4 Funktionen ungroup

Nogle gange når man er færdig med en proces, men gerne vil arbejde videre på et dataframe, er det nyttigt at anvende `ungroup()` på datasættet igen. Det er mest relevant i længere projektor men som eksempel kig på følgende kode og bemærk at der står "Groups: Adult [2]" på toppen af den nye dataframe med summary statistics:

```
titanic_clean_summary <- titanic_clean %>%
  mutate(Adult = ifelse(Age>=18, "Adult", "Child")) %>%
  group_by(Adult, Sex) %>%
  summarise(meanSurvived = mean(Survived))

## `summarise()` has grouped output by 'Adult'. You can override using the
## `.groups` argument.

titanic_clean_summary

## # A tibble: 4 x 3
## # Groups:   Adult [2]
##   Adult   Sex   meanSurvived
##   <chr>  <chr>      <dbl>
## 1 Adult female      0.772
```

```
## 2 Adult male      0.177
## 3 Child female    0.691
## 4 Child male      0.397
```

Bemærk at vi først anvendte `group_by` på både `Adult` og `Sex`, men hver gange man lave en beregning bliver én opdeling fjernet - i dette tilfælde opdeler man ikke længere efter `Sex` men man stadig opdeler efter `Adult`. Det er ikke et problem hvis vi ikke vil arbejde videre med dataframen. Men forstil at vi gerne vil vide, hvad den maksimum chance for survival er ud fra de fire tal beregnede. Der vi ikke vil adskille efter en kategorisk variabel dropper vi `group_by()`:

```
titanic_clean_summary %>%
  summarise("maxChance" = max(meanSurvived))
```

```
## # A tibble: 2 x 2
##   Adult maxChance
##   <chr>     <dbl>
## 1 Adult     0.772
## 2 Child     0.691
```

Man kan dog se, at outputtet er blevet adskilt efter variablen `Adult`. For at undgå disse bør man første anvende `ungroup` for at få fjernet effekten af `group_by()`.

```
titanic_clean_summary %>%
  ungroup() %>%
  summarise("maxChance" = max(meanSurvived))
```

```
## # A tibble: 1 x 1
##   maxChance
##       <dbl>
## 1     0.772
```

## 6.3 pivot\_longer() / pivot\_wider() med Tidyr-pakken

**Tidy data** findes i to former: wide data og long data. Det kan være nyttigt at transformere dataframen fra den ene form til den anden, for fx. at lave et bemstegt plot med `ggplot2`-pakken. Indenfor pakken `tidyverse` er der funktioner som kan bruges til at lave disse transformeringer.

Inden vi begynder at kigge lidt nærmere på `tidyverse` skal vi beskrive, hvad betyder long data og wide data.

**Wide data:** Her har man en kolon til hver variabel og en række til hver observation. Det gør de data nem at forstå og denne data type findes ofte indenfor biologi - for eksempel hvis man har forskellige samples (treatments, controls, conditions osv.) som variabler.

|    | wide |   |   | long |     |     |
|----|------|---|---|------|-----|-----|
| id | x    | y | z | id   | key | val |
| 1  | a    | c | e | 1    | x   | a   |
| 2  | b    | d | f | 2    | x   | b   |
|    |      |   |   | 1    | y   | c   |
|    |      |   |   | 2    | y   | d   |
|    |      |   |   | 1    | z   | e   |
|    |      |   |   | 2    | z   | f   |

Figure 6.1: source: <https://www.garrickadenbuie.com/project/tidyexplain/>

**Long data:** Med long data har man værdier samlet i en enkel kolon og en kolon som en slags nøgle, som fortæller også hvilken variable hver værdi hørte til i den wide format. Datasættet er stadig betragtet som **tidy** men informationen opbevares på en anden måde. Det er lidt sværer at læse men nemmere at arbejde med når man analyser de data.

Når man transformer data fra wide til long eller omvendt, kaldes det for **reshaping**.

### 6.3.1 Tidyr pakke - oversigt

Her er en oversigt over de fire vigtigste funktioner fra R-pakken **tidyr**. Vi fokuserer mest på **pivot** funktionerne men det kan være nyttigt at bruge **separate** og **unite** en gang i mellem.

| tidr funktion               | Beskrivelse                                    |
|-----------------------------|--|
| <code>pivot_longer()</code> | short til long                                 |
| <code>pivot_wider()</code>  | long til short                                 |
| <code>separate()</code>     | opdele strings fra en kolon til to             |
| <code>unite()</code>        | tilføje strings sammen ind fra to til én kolon |

### 6.3.2 Wide -> Long med `pivot_longer()`

Lad os arbejde med datasættet **Iris**. Man få Iris i long form med følgende kommando. Her vil man gerne tage alle numeriske kolonner og placerer deres værdier i en enkel kolon `value` (med en nøgle kolon `name` til at skelne imellem de forskellige variabler).

```
iris %>% pivot_longer(cols = where(is.numeric))
```

```
## # A tibble: 600 x 3
##   Species name      value
##   <fct>   <chr>     <dbl>
## 1 setosa  Sepal.Length 5.1
## 2 setosa  Sepal.Width  3.5
## 3 setosa  Petal.Length 1.4
## 4 setosa  Petal.Width  0.2
## 5 setosa  Sepal.Length 4.9
## # ... with 595 more rows, and 1 more variable:
## #   Petal.Length: num  1.4
```

variablen `Species` med i den enkel kolon:

```
iris %>%
  pivot_longer(cols = -Species)

## # A tibble: 600 x 3
##   Species name      value
##   <fct>   <chr>     <dbl>
## 1 setosa  Sepal.Length 5.1
## 2 setosa  Sepal.Width  3.5
## 3 setosa  Petal.Length 1.4
## 4 setosa  Petal.Width  0.2
## 5 setosa  Sepal.Length 4.9
## 6 setosa  Sepal.Width  3
## 7 setosa  Petal.Length 1.4
## 8 setosa  Petal.Width  0.2
## 9 setosa  Sepal.Length 4.7
## 10 setosa Sepal.Width  3.2
## # ... with 590 more rows
```

Her er et billede der illustrerer wide og long form med datasættet `iris`:

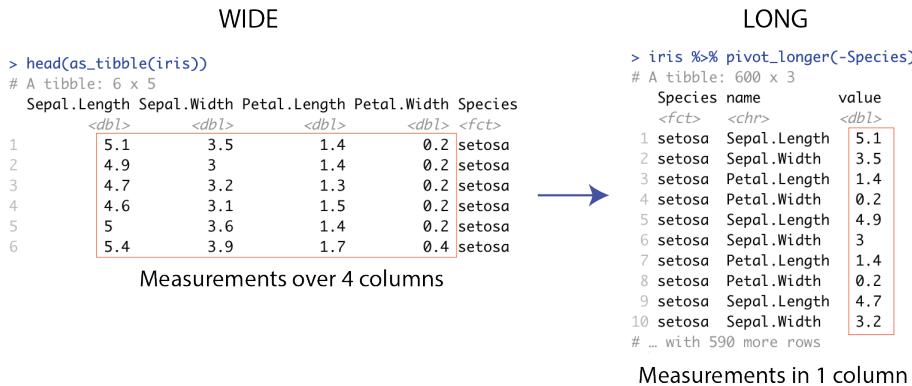


Figure 6.2: wide til long med Iris

Til venstre har vi målingerne i datasættet over fire forskellige kolonner som hedder `Sepal.Length`, `Sepal.Width`, `Petal.Length` og `Petal.Width`, og en ekstra kolon der skelne imellem de tre `Species`. Til højre har vi fået alle målingerne ind i en enkel kolon der hedder `values`, og så bruger man en anden ‘nøgle’ kolon der hedder `name` til at fortælle os om det er en måling for `Sepal.Length` eller `Sepal.Width` osv.

Jeg kan kalde de kolonner navne for målingerne og nøglen til nogle andre en default: for eksempel i nedenstående skal målingerne hedde `measurements` og nøglen hedde `trait`.

The diagram illustrates the transformation of the iris dataset from a wide format to a long format using the `pivot_longer` function.

**WIDE**

```
> head(as_tibble(iris))
# A tibble: 6 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>       <dbl>      <dbl>       <dbl>   <fct>
 1       5.1        3.5       1.4        0.2  setosa
 2       4.9        3.0       1.4        0.2  setosa
 3       4.7        3.2       1.3        0.2  setosa
 4       4.6        3.1       1.5        0.2  setosa
 5       5.0        3.6       1.4        0.2  setosa
 6       5.4        3.9       1.7        0.4  setosa
```

**LONG**

```
> iris %>% pivot_longer(cols = -Species,
+                         names_to = "trait",
+                         values_to = "measurement")
# A tibble: 600 x 3
  Species trait     measurement
    <fct>   <chr>       <dbl>
 1 setosa  Sepal.Length 5.1
 2 setosa  Sepal.Width  3.5
 3 setosa  Petal.Length 1.4
 4 setosa  Petal.Width  0.2
 5 setosa  Sepal.Length 4.9
 6 setosa  Sepal.Width  3.0
 7 setosa  Petal.Length 1.3
 8 setosa  Petal.Width  0.2
 9 setosa  Sepal.Length 4.7
10 setosa  Sepal.Width  3.2
# ... with 590 more rows
```

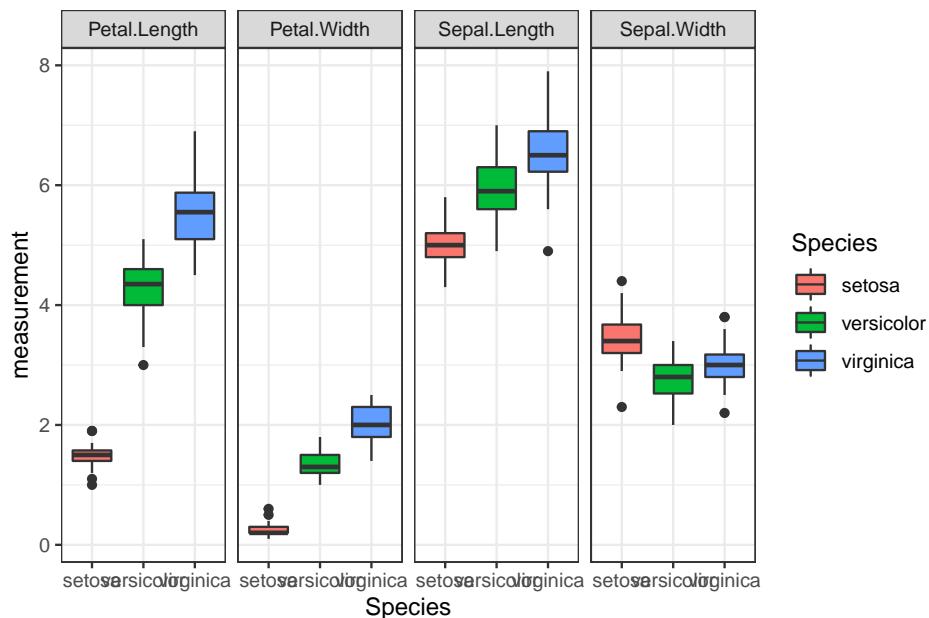
A red arrow points from the WIDE section to the LONG section, indicating the transformation process.

**Annotations:**

- Sepal.Length, Sepal.Width, Petal.Length, Petal.Width move into single column “trait” and their values move into single column “measurement”.
- Species column retained.

Man kan for eksempel bruge den long form den til at visualisere samtlige mulige boxplots opdelt efter Species og trait på samme plot:

```
ggplot(iris.long,aes(y=measurement,x=Species,fill=Species)) +  
  geom_boxplot() +  
  facet_grid(~trait) +  
  theme_bw()
```



### 6.3.3 separate()

Funktionen `separate()` fra pakken `tidyverse` kan bruges til at opdele to forskellige dele som eksisterer i samme kolon. For eksempel, i `iris` har vi variabler med navne `Sepal.Width`, `Sepal.Length` osv. - man kan forestille sig, at opdele disse navne over to kolonner i stedet for en - f.eks. "Sepal" og "Width" i tilfældet af `Sepal.Width`. I nedenstående kan man se, hvordan man anvender `separate()`.

```
iris %>%
  pivot_longer(cols = -Species, names_to = "trait", values_to = "measurement") %>%
  separate(col = trait, into = c("part", "measure"), sep = "\\.") %>%
  head()

## # A tibble: 6 x 4
##   Species part  measure measurement
##   <fct>   <chr> <chr>     <dbl>
## 1 setosa   Sepal Length      5.1
## 2 setosa   Sepal Width       3.5
## 3 setosa   Petal Length     1.4
## 4 setosa   Petal Width      0.2
## 5 setosa   Sepal Length     4.9
## 6 setosa   Sepal Width       3
```

Man specificerer variablen `trait`, og at det skal opdeles til to variabler `part` og `measure`. Vi angiver `sep = "\\."` som betyder, at vi gerne vil have `part` som delen af `trait` foran ‘`‘` og `measure` som delen af `trait` efter .. Vi bruger “`\.`” til at fortælle, at vi er interesseret i punktum og ikke en “anonym character”, som punktum plejer at betyde i “string”-sprog. Man behøver faktisk ikke at specificere `sep = "\\."` i dette tilfælde - som standard kigger funktionen efter ‘non-character’ tegn og bruger dem til at lave opdelingen.

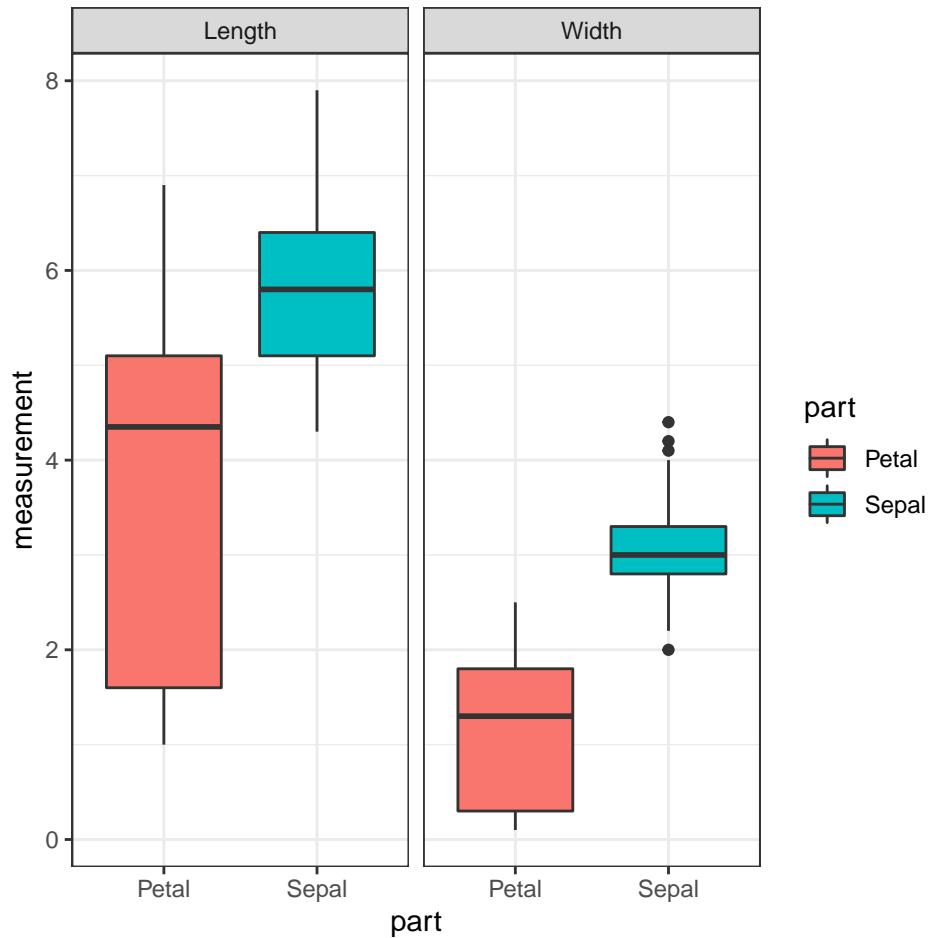
Samme resultat:

```
iris %>%
  pivot_longer(cols = -Species, names_to = "trait", values_to = "measurement") %>%
  separate(col = trait, into = c("part", "measure")) %>%
  head()

## # A tibble: 6 x 4
##   Species part  measure measurement
##   <fct>   <chr> <chr>     <dbl>
## 1 setosa   Sepal Length      5.1
## 2 setosa   Sepal Width       3.5
## 3 setosa   Petal Length     1.4
## 4 setosa   Petal Width      0.2
## 5 setosa   Sepal Length     4.9
## 6 setosa   Sepal Width       3
```

Bruger resultatet i et plot:

```
iris %>%
  pivot_longer(cols = -Species, names_to = "trait", values_to = "measurement") %>%
  separate(col = trait, into = c("part", "measure")) %>%
  ggplot(aes(y=measurement,x=part,fill=part)) +
  geom_boxplot() +
  facet_grid(~measure) +
  theme_bw()
```



Se også `unite()` som gøre det modsatte til `separate()`.

## 6.4 Eksempel: Titanic summary statistics

Her er et eksempel med datasættet `titanic` der inddrager mange af de tidyverse koncepter vi har lært indtil videre.

- `group_by()` og `summarise()`

Vi laver vores summary statistics som i ovenstående.

```
titanic_clean_summary_by_sex <- titanic_clean %>%
  group_by(Sex) %>%
  summarise(count = n(),
            meanSurvived = mean(Survived),
            meanAge = mean(Age),
            propFirst = sum(Pclass==1)/n())
titanic_clean_summary_by_sex
```

```
## # A tibble: 2 x 5
##   Sex     count meanSurvived meanAge propFirst
##   <chr>    <int>      <dbl>     <dbl>      <dbl>
## 1 female    261       0.755     27.9      0.326
## 2 male      453       0.205     30.7      0.223
```

- `pivot_longer()`

Vi transformerer eller `reshape` datarammen fra wide data til long data. Vi vil få kun de numeriske summary statistics samlede i en enkel kolonne, så variablen `Sex` skal indgå i den enkel kolonne.

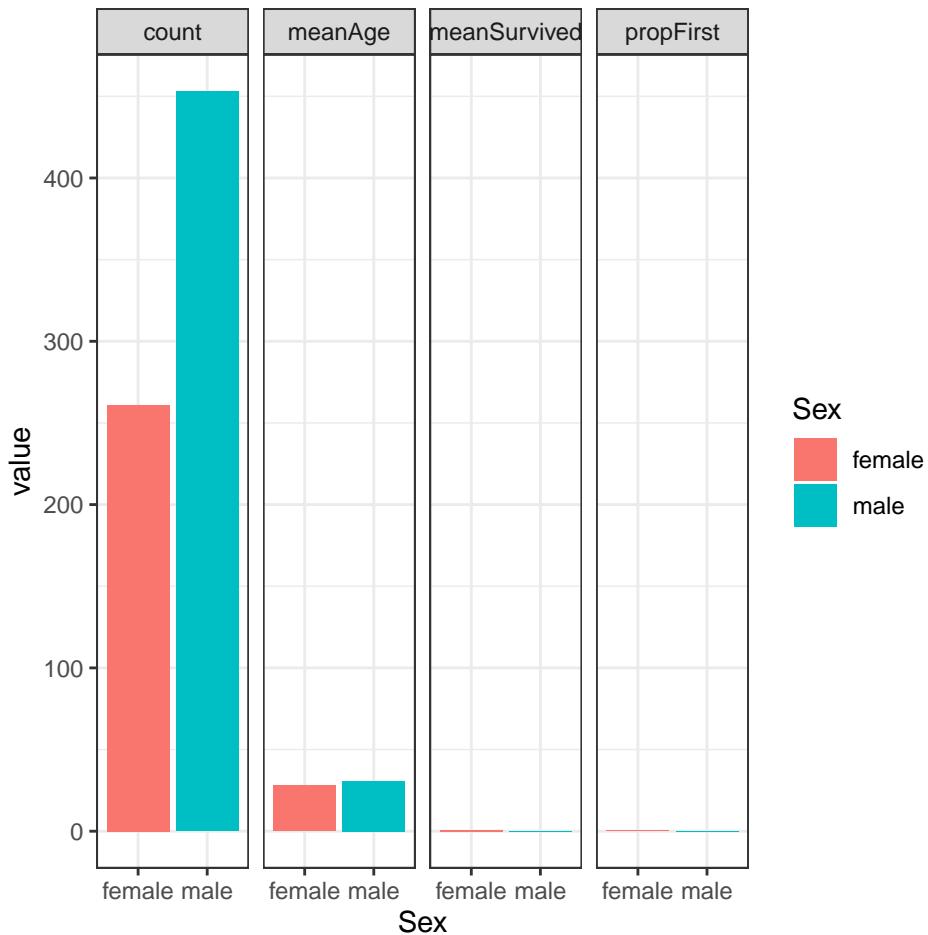
```
titanic_clean_summary_by_sex %>% pivot_longer(cols=-Sex)
```

```
## # A tibble: 8 x 3
##   Sex     name      value
##   <chr>   <chr>     <dbl>
## 1 female  count      261
## 2 female  meanSurvived  0.755
## 3 female  meanAge      27.9
## 4 female  propFirst     0.326
## 5 male    count      453
## 6 male    meanSurvived  0.205
## 7 male    meanAge      30.7
## 8 male    propFirst     0.223
```

- `ggplot()` med `facet_grid()`

Vi kombinerer `pivot_longer()` med et plot af vores summary statistics og benytte `facet_grid()` til at separere ved de forskellige statistiker.

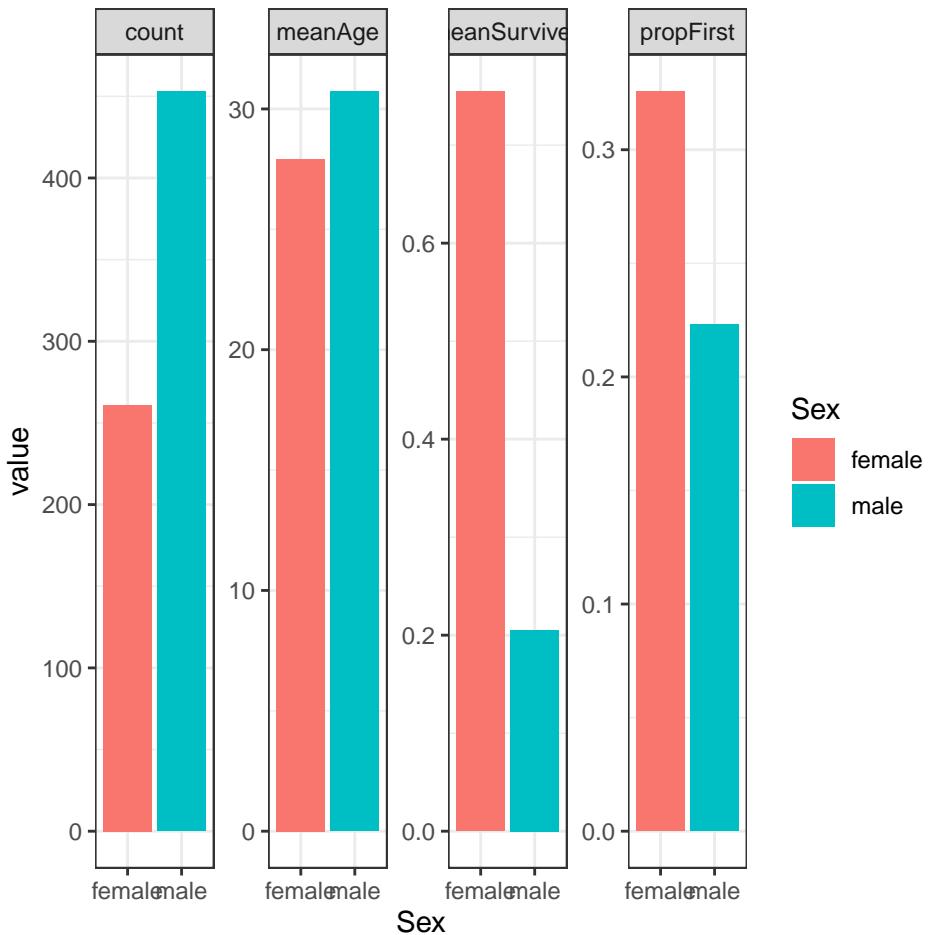
```
titanic_clean_summary_by_sex %>%
  pivot_longer(cols=-Sex) %>%
  ggplot(aes(x=Sex,y=value,fill=Sex)) +
  geom_bar(stat="identity") +
  facet_grid(~name) +
  theme_bw()
```



- `facet_wrap()`

Vi laver den sammen som ovenstående men specifiserer `facet_wrap()` i stedet for `facet_grid()` - indenfor `facet_wrap()` kan man bruge indstillingen `scales="free"` som gøre, at de fire plots få hver deres egne akse limits.

```
titanic_clean_summary_by_sex %>%
  pivot_longer(cols=-Sex) %>%
  ggplot(aes(x=Sex,y=value,fill=Sex)) +
  geom_bar(stat="identity") +
  facet_wrap(~name,scales="free",ncol=4) +
  theme_bw()
```



#### 6.4.1 Demonstration af pivot\_wider()

Det er også brugbart at kende måden at man skifter fra long form til wide form.

- Wide -> Long

```
titanic_summary_long <- titanic_clean_summary_by_sex %>%
  pivot_longer(cols=-Sex)
```

- Long -> Wide

```
titanic_summary_long %>%
  pivot_wider(names_from = "name", values_from = "value")
```

```
## # A tibble: 2 x 5
##   Sex     count meanSurvived meanAge propFirst
##   <chr>    <dbl>      <dbl>    <dbl>     <dbl>
```

```
## 1 female    261      0.755    27.9     0.326
## 2 male      453      0.205    30.7     0.223
```

Parametre er:

- `names_from` - nøglekolon som skal udgør flere kolonner i den nye dataframe
- `values_from` - selve værdier, som skal være i de nye kolonner i den wide form

## 6.5 `left_join()`: forbinde dataframes

Vi tager udgangspunkt i følgende to dataframes:

```
gene_table <- as_tibble(read.table("https://www.dropbox.com/s/6118ezrskly8joi/mouse_2g
coldata <- as_tibble(read.table("https://www.dropbox.com/s/jlrszakmqlnmu2m/bottomly_ph
```

Lad os kigge først på datasættet `gene_table`, som viser genekspression målinger over forskellige samples i mus.

```
gene_table
```

```
## # A tibble: 3 x 22
##   gene      SRX033480 SRX033488 SRX033481 SRX033489 SRX033482 SRX033490 SRX033483
##   <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 ENSMUSG~    158.     182.     119.     155.     167.     164.     180.
## 2 ENSMUSG~    143.     118.     91.6     106.     157.     95.1     131.
## 3 ENSMUSG~    132.     117.     100.     116.     88.1     125.     124.
## # ... with 14 more variables: SRX033476 <dbl>, SRX033478 <dbl>,
## #   SRX033479 <dbl>, SRX033472 <dbl>, SRX033473 <dbl>, SRX033474 <dbl>,
## #   SRX033475 <dbl>, SRX033491 <dbl>, SRX033484 <dbl>, SRX033492 <dbl>,
## #   SRX033485 <dbl>, SRX033493 <dbl>, SRX033486 <dbl>, SRX033494 <dbl>
```

Man kan se, at der er 22 kolonner i datasættet - én der refererer til et gen navn og 21 der er forskellige samples fra eksperimentet. Men det ikke er klart, hvad den enkel sample egentlig er. Lad os derfor kigge på de sample oplysninger, som kan være nyttige at inddrage i vores analyse/plotter for at undersøge eventuelle batch effekter osv.

```
coldata
```

```
## # A tibble: 21 x 5
##   sample  num.tech.reps strain  batch lane.number
##   <chr>        <int> <chr>    <int>        <int>
## 1 SRX033480          1 C57BL.6J     6         1
## 2 SRX033488          1 C57BL.6J     7         1
## 3 SRX033481          1 C57BL.6J     6         2
## 4 SRX033489          1 C57BL.6J     7         2
## 5 SRX033482          1 C57BL.6J     6         3
## 6 SRX033490          1 C57BL.6J     7         3
```

```
## 7 SRX033483      1 C57BL.6J    6      5
## 8 SRX033476      1 C57BL.6J    4      6
## 9 SRX033478      1 C57BL.6J    4      7
## 10 SRX033479     1 C57BL.6J   4      8
## # ... with 11 more rows
```

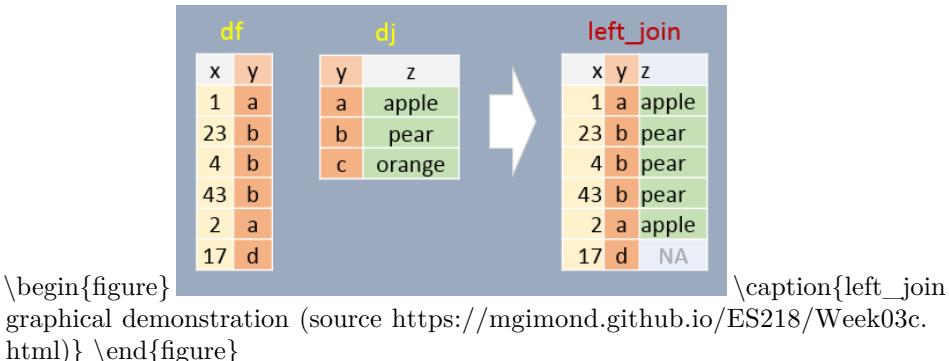
Man kan se forskellige oplysninger om de 21 samples, blandt andet den strain af mus hver sample stammer fra og den batch. Her refererer `batch` på de forskellige omstændigheder eller tidspunkter de samples var blevet samlet. Hvis man er interesseret i om der er en forskel i ekspressionsniveau mellem de to strains, kan det være, at man er nødt til at kontrollere efter batch for at sikre at forskellen skyldes `strain` og ikke tekniske effekter pga. `batch`.

### 6.5.1 Funktionen `left_join()` fra dplyr-pakken

Funktionen `left_join()` er en del af pakken `dplyr` som vi har arbejdet meget med indtil videre i kurset.

| funktion                  | Beskrivelse (kopirer)                                 |
|---------------------------|---|
| <code>left_join()</code>  | Join matching rows from second table to the first     |
| <code>right_join()</code> | Join matching rows from the first table to the second |
| <code>inner_join()</code> | Join two tables, returning all rows present in both   |
| <code>full_join()</code>  | Join data with all possible rows present              |

Vi fokuserer her på funktionen `left_join()` fordi den er den mest brugbart i biologiske data analyser, men vi kigger også på de øvrige funktioner gennem problemstillingerne nedenunder. Her er en grafiske demonstration af `left_join()`:



Det særligt med `left_join` i forhold til de andre funktioner, er at `left_join` bevarer samtlige data i dataframen man tager udgangspunkt i - det vil sige `df` i ovenstående billede, selvom `d` matchet ikke med en frugt i `dj`. I ovenstående genekspression eksempel betyder det, at man bevarer alle målinger i `gene_table`, uanset om der er oplysninger om deres pågældende samples.

### 6.5.2 Anvende `left_join()` for vores dataset.

Ligesom man matcher kolonnen `y` i `df` og `dj` i ovenstående eksempel, skal vi også have en kolon vi kan matcher. Vi vil gerne bruge kolonnen `sample` fra `sample_info` til at sammenligne med de forskellige sample navne i `gene_table`, men først er vi nødt til at lave `gene_table` om til long-form, således at sample navne fremgår i en enkel kolon, `sample` (der kan bruges i `left_join`).

```
gene_table_long <- gene_table %>%
  pivot_longer(cols = -gene,
               names_to = "sample",
               values_to = "expression")
gene_table_long

## # A tibble: 63 x 3
##   gene           sample     expression
##   <chr>          <chr>        <dbl>
## 1 ENSMUSG00000006517 SRX033480    158.
## 2 ENSMUSG00000006517 SRX033488    182.
## 3 ENSMUSG00000006517 SRX033481    119.
## 4 ENSMUSG00000006517 SRX033489    155.
## 5 ENSMUSG00000006517 SRX033482    167.
## 6 ENSMUSG00000006517 SRX033490    164.
## 7 ENSMUSG00000006517 SRX033483    180.
## 8 ENSMUSG00000006517 SRX033476    263.
## 9 ENSMUSG00000006517 SRX033478    276.
## 10 ENSMUSG00000006517 SRX033479   328.
## # ... with 53 more rows
```

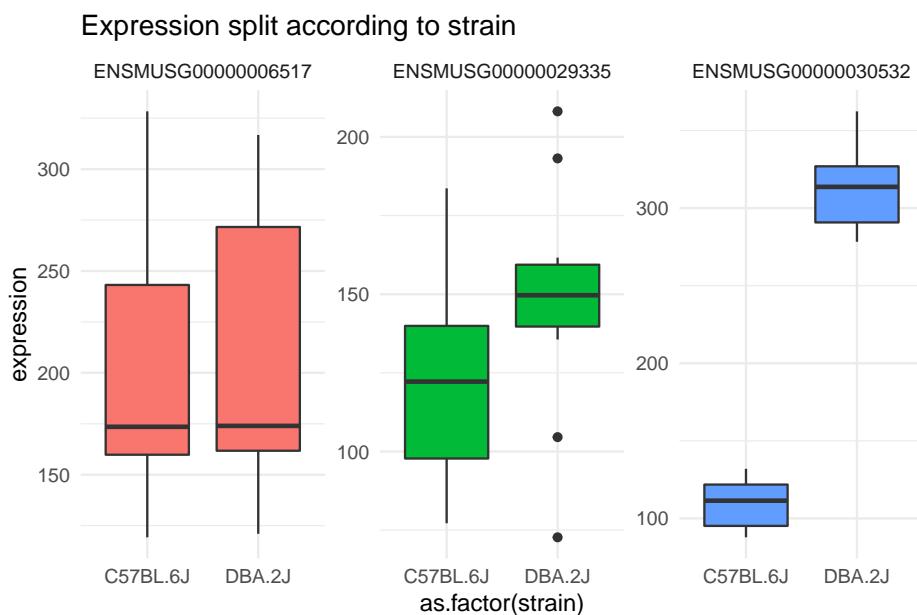
Dernæst kan vi tilføje oplysninger data fra `sample_info`. Her angiver vi `by = "sample"` fordi det er navnet til kolonnen som vi gerne vil bruge til at forbinde de to datarammer - altså, det er med i begge to datarammer, så `left_join()` kan bruge den som en slags nøgle til at vide, hvor alle de forskellige oplysninger skal tilføjes.

```
data_join <- gene_table_long %>% left_join(coldata, by="sample")
```

Nu at vi har fået forbundet de to datarammer, kan man inddrage de ekstra oplysninger vi har fået i et plot. Her laver vi et plot med en farve til hver strain og et plot med en farve til hver batch.

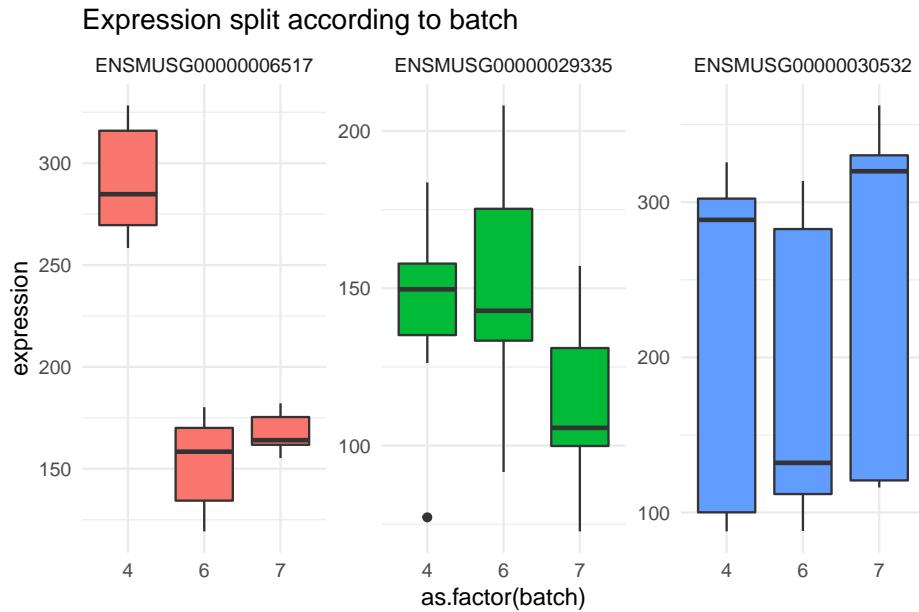
```
gg2 <- data_join %>%
  ggplot(aes(y=expression, x=as.factor(strain), fill=gene)) +
  geom_boxplot() +
  facet_wrap(~gene, scales="free") +
  theme_minimal() +
  theme(legend.position = "none") +
  ggtitle("Expression split according to strain")
```

```
gg2
```



```
gg1 <- data_join %>%
  ggplot(aes(y=expression,x=as.factor(batch),fill=gene)) +
  geom_boxplot() +
  facet_wrap(~gene,scales="free") +
  theme_minimal() +
  theme(legend.position = "none") +
  ggtitle("Expression split according to batch")
```

```
gg1
```



## 6.6 Problemstillinger

**Problem 1)** Lav quizzen - “Quiz - tidyverse - part 2”.

---

*Vi øver os med titanic. Inlæs de data og lave oprydningen med følgende kode:*

```
library(tidyverse)
library(titanic)
titanic <- as_tibble(titanic_train)

titanic_clean <- titanic %>%
  select(-Cabin) %>%
  drop_na()
```

---

**Problem 2)** Fra titanic\_clean beregn den gennemsnitlige alder af alle passagerer ombord skibet.

```
titanic_clean %>%
  summarise(...) #rediger her
```

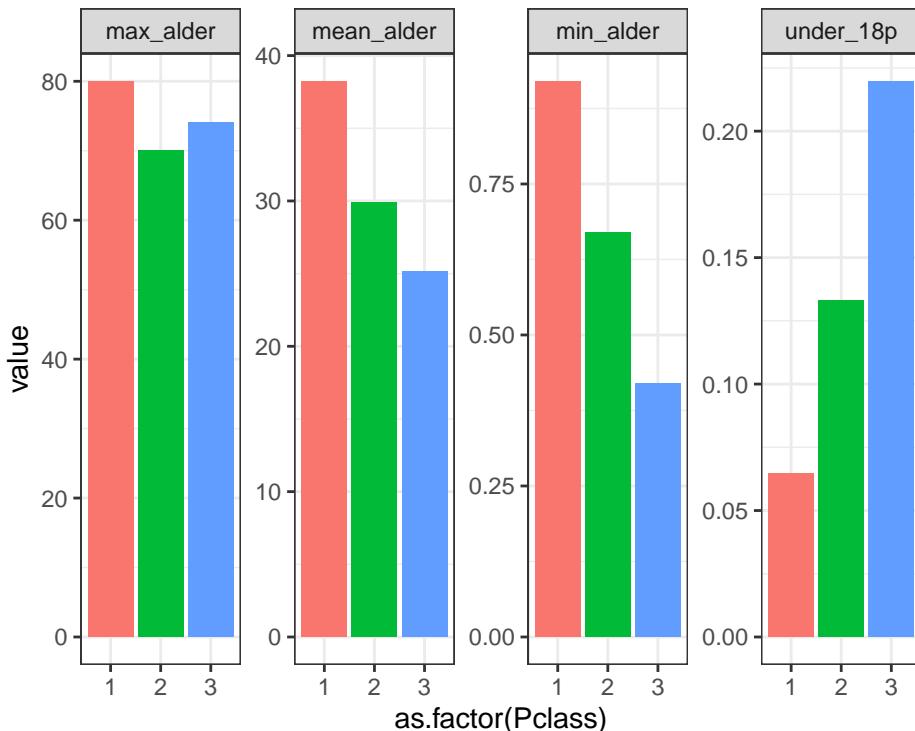
- I samme kommando beregne også den maksimum alder og minimum alder, samt proportionen af passagerer, der er under 18 (for den sidste se mit eksempel med Pclass oven på). Dataframen skal se sådan ud:

```
## # A tibble: 1 x 4
##   mean_alder max_alder min_alder under_18p
##       <dbl>      <dbl>     <dbl>      <dbl>
## 1        29.7        80     0.42     0.158
```

---

**Problem 3)**

- a) Beregne samme summary statistics som i sidste problem men anvende `group_by()` til at først opdeler efter variablen `Pclass`.
- b) Brug din nye summary statistikker dataframe til at lave et barplot med `stat="identity"` som viser den gennemsnitlige alder på y-aksen opdelt efter `Pclass` på x-aksen (tænke lidt over data typen på `Pclass`)
- c) Anvend `pivot_longer()` på din summary statistikker dataframe (brug indstilling `cols = -Pclass`)
- d) Brug din long-form dataframe af summary statistikker til at lave plots af samtlige summary statistikker med én `ggplot` kommando (adskil dem ved at benytte facet og opdele efter `Pclass` indenfor hvert plot, ligesom i følgende).

**Problem 4)**

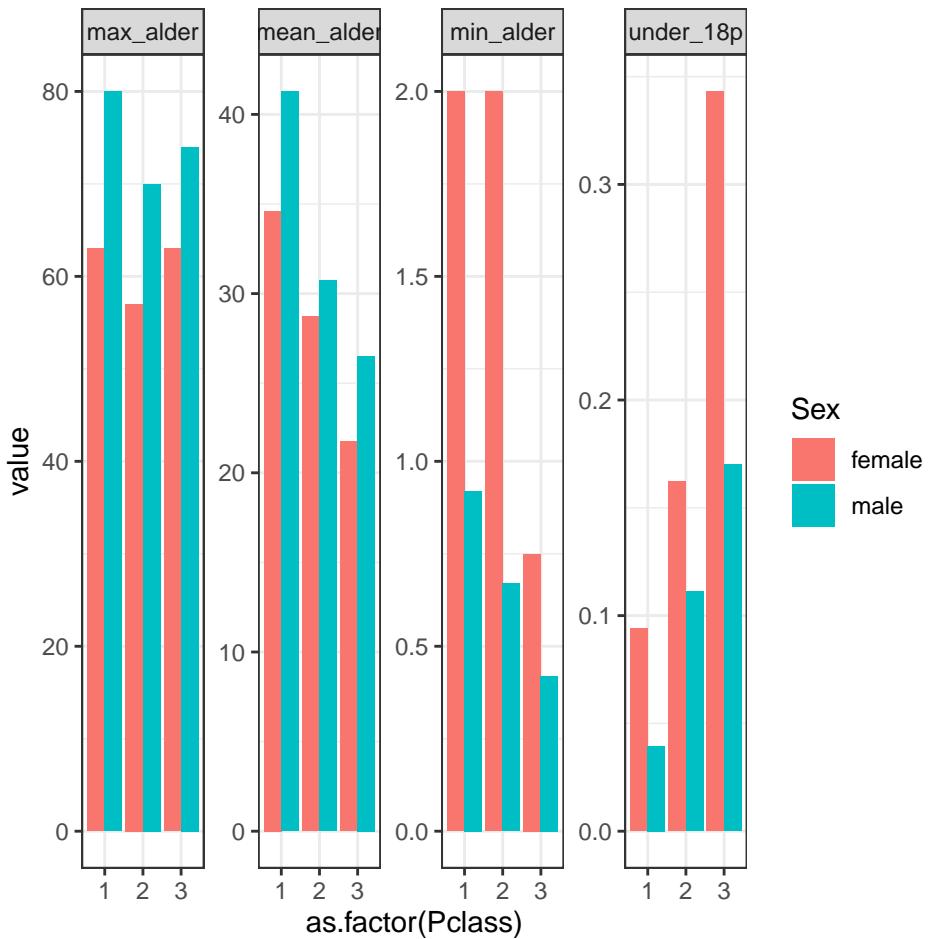
a) Beregne samme summary statistics som i 2) men anvende `group_by()` til at først opdele efter både variablerne `Pclass` og `Sex`.

- OBS: Man får en advarsel “`summarise()` has grouped output by ‘Pclass’ ...” fordi din dataframe er stadig betragtet som opdelte efter `Pclass`, som du skal tage i betragtning hvis du laver flere beregninger på den.
- Brug til sidste `ungroup()` på din nye dataframe for at være sikker på, at den ikke længere er opdelt efter en variabel.

b) Brug `pivot_longer` til at få datasættet i long form (tænk over hvilke variabler skal være i indstillingen `cols` - det kan hjælp at skrive dem i en vector med notationen `c()`). Nøglekolonnen skal hed `stat` og kolonnen med værdierne skal hed `values`.

```
## `summarise()` has grouped output by 'Pclass'. You can override using the
## ` `.groups` argument.
```

c) Lav et plot af samtlige summary statistikker, som er i long form og ser ud som følgende plot.




---

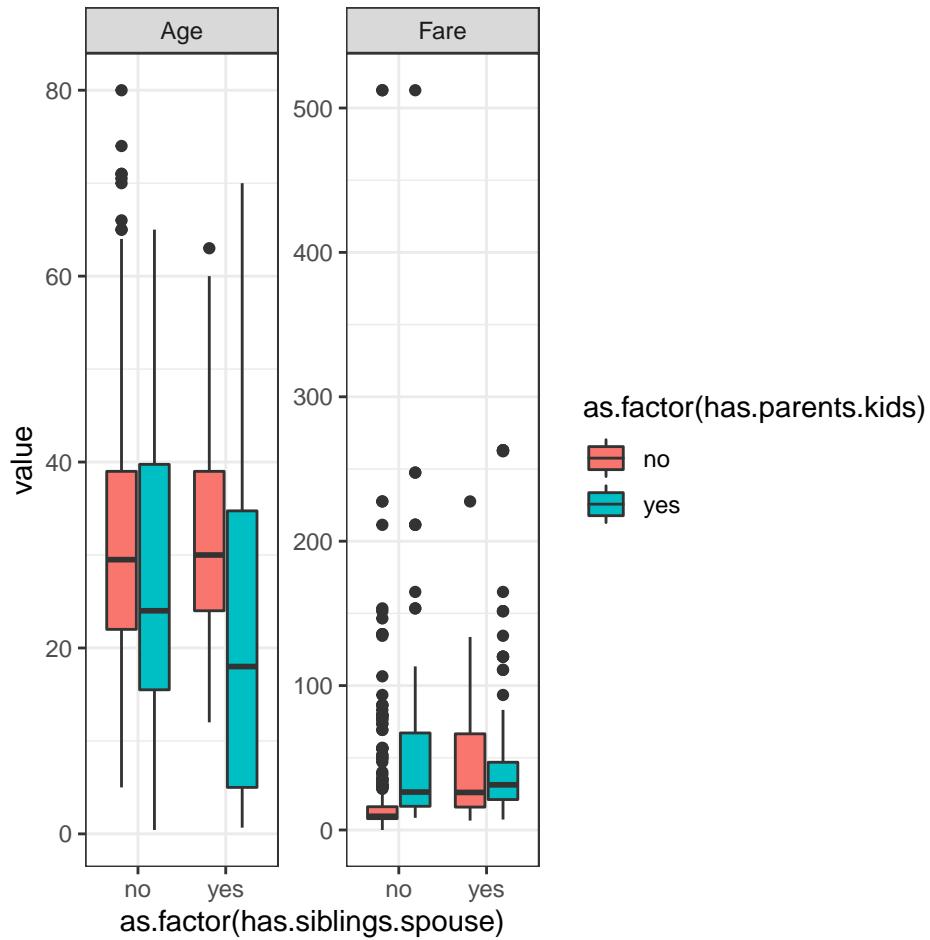
**Problem 5)** `group_by()` med tre variabler og `summarise()`. Afprøv en kombination med tre forskellige variabler (vælg selv) indenfor `group_by()` og bruge `summarise()` til at beregne middelværdien for `Fare`.

- Anvend `ungroup()` når du er færdig med `summarise`
  - Lave et plot for at visualisere `meanFare`. Idé: som mulighed kan man tilføje variabler til `facet_grid()` - for eksempel `facet_grid(~Var1 + Var2)`.
- 

**Problem 6)** `pivot_longer()` Lav følgende plot

- Først lave to nye variabler fra `SibSp` og `Parch`, hvor der står "yes" hvis værdien er større end 0
- `select` nødvendige variabler

- Lave om til long form (tænk over hvilke variabler skal være i en enkel kolon)
- Brug din long form dataframe til at lave plottet




---

**Problem 7)** `Pivot_wider()` Vi har en tribble som jeg har kopiret fra <https://r4ds.had.co.nz/index.html>.

```
people <- tribble(
  ~name, ~names, ~values,
  #-----/-----/-----
  "Phillip Woods", "age", 45,
  "Phillip Woods", "height", 186,
  "Jessica Cordero", "age", 37,
  "Jessica Cordero", "height", 156,
  "Brady Smith", "age", 23,
```

```
"Brady Smith",      "height",    177
)
```

Brug `pivot_wider()` på `people`. Vi er nødt til at specifiser som minimum `names_from` og `values_from` indenfor `pivot_wider()` - prøv at angiv de relevante variabler

**Problem 8) `left_join()` øvelse.** Kør følgende kode med to tribbles:

```
superheroes <- tribble(
  ~name, ~alignment, ~gender, ~publisher,
  "Magneto",     "bad",   "male",      "Marvel",
  "Storm",       "good",  "female",    "Marvel",
  "Mystique",    "bad",   "female",    "Marvel",
  "Batman",      "good",  "male",      "DC",
  "Joker",       "bad",   "male",      "DC",
  "Catwoman",    "bad",   "female",    "DC",
  "Hellboy",     "good",  "male",      "Dark Horse Comics"
)

publishers <- tribble(
  ~publisher, ~yr Founded,
  "DC",        1934L,
  "Marvel",    1939L,
  "Image",     1992L
)
```

Vi har to dataframes - `superheroes` og `publishers`. Hvilken kolon kan man bruge til at forbinde de to dataframes? Brug `left_join()` til at tilføje oplysninger fra `publishers` til datarammen `superheroes`.

- Få man alle observationerne fra dataframen `superheroes` med i din nye dataframe?
- Benyt `inner_join()` til at forbinde `publishers` til `superheroes` - få man så nu alle observationer med denne gang?
- Benyt `full_join()` til at forbinde `publishers` til `superheroes` - hvor mange observationer få man med nu? Hvorfor?

**Problem 9) `left_join()` øvelse.**

Køre nedenstående kode, hvor der er to dataframes - `iris2` og `sample_table`. Dataframen `iris2` er ikke særlig informativ med hensyn til hvad de forskellige samples egentlige er, men oplysningerne om dem står i `sample_table`. Brug `left_join()` til at tilføje `sample_table` til `iris2` for at få en dataramme som indeholder både de data og de samples oplysninger.

```

data(iris)
iris2 <- as_tibble(iris)
names(iris2) <- c("sample1", "sample2", "sample3", "sample4", "Species")

samp_table <- tribble(
  ~sample, ~part, ~measure,
  #-----/-----/-----#
  "sample1", "Sepal", "Length",
  "sample2", "Sepal", "Width",
  "sample3", "Petal", "Length",
  "sample4", "Sepal", "Width"
)

iris2 %>% glimpse()

## Rows: 150
## Columns: 5
## $ sample1 <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.8, 4.-
## $ sample2 <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3.-
## $ sample3 <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.6, 1.-
## $ sample4 <dbl> 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.2, 0.2, 0.-
## $ Species <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, setosa-
samp_table %>% glimpse()

## Rows: 4
## Columns: 3
## $ sample <chr> "sample1", "sample2", "sample3", "sample4"
## $ part   <chr> "Sepal", "Sepal", "Petal", "Sepal"
## $ measure <chr> "Length", "Width", "Length", "Width"

```

### Problem 10) *Separate()* øvelse

- Tag udgangspunkt i datasættet `titanic_clean` og benyt funktionen `Separate()` til at opdele variablen `Name` ind til to variabler, “Surname” og “Rest” (Godt råd: brug `sep=" "` for at undgå, at man få en unødvendig mellemrum lige før “Rest”).
- Anvend `Separate()` en gang til, men for at opdele variablen `Rest` into to variabler, “Title” og “Names”. Hvad bruger man som `sep?` (Hint: brug “\\” foran en punktum).
- Beregn summary statistikker for hver “Title” - mange passagerer, gennemsnits alder, proportionen der overlevede, og proportionen der rejste i første klasse.
- Arrange din ny dataframe efter hvor mange personer der er for hver “Title”

- mest på toppen og mindst på bunden.

---

**Problem 11)** Valgfri ekstra: lav en ny dataramme med alle passagerer, der hedder "Alice" eller "Elizabeth" (brug Google her).

---

**Problem 12)** Analyse og visualisering af biologiske datasæt konkurrence!

- Når du færdig med ovenstående husk at få lavet dit plot til konkurrencen!!
- Se siden på Absalon for yderligere detaljer

## 6.7 Ekstra links

Cheatsheet: <https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf>



# Chapter 7

## Functional programming med purrr-pakken



### 7.1 Inledning og læringsmålene

Emnet handler om hvordan man kan inddrage funktioner for at øge reproducebarhed og gennemskuelighed i dine analyser. Det er ofte tilfældet i biologi, at man har flere datasæts eller variabler, der referer f.eks. til forskellige samples, replikator eller batches, og man gerne vil lave præcis samme proces på dem alle sammen samtidig.

I dette emne beskæftiger du dig med især pakken **Purrr** og `map()` funktioner, som kan benyttes til at lave gentagne baserende analyser i R.

#### 7.1.1 Læringsmålene

I skal være i stand til at:

- Anvende `map()` - funktioner til at udføre beregninger iterativt over flere kolonner
- `group_by()` og `nest()` til at lave reproducerbar analyser over forskellige dele af datasættet.
- Kombinere `map()` og `map2()` med custom funktioner for at øge fleksibilitet i analyserne.

### 7.1.2 Video ressourcer

- Video 1: Introduction to map functions for iterating over columns

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630848>

- Video 2: Introduction to custom functions and combining them with map

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630825>

- Video 3: Introduction to nest functions for breaking data into sections

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630798>

## 7.2 Iterativ processer med `map()` funktioner

Når man lave en interativ proces, vil man gerne lave samme ting gentagne gange. Det kan være for eksempel, at vi har ti variabler og vi gerne vil beregne middelværdien for hver variable. Vi arbejder med datasættet `eukaryotes`, som indeholder oplysninger om forskellige organismer som hører til eukaryotes - for eksempel deres navne, gruppe, sub-gruppe, antal proteins/genes, genom størrelse og så videre. Man kan få de data indlæste med følgende kommando og se en list over for de forskellige kolon navne nedenfor.

```
eukaryotes <- read_tsv("https://www.dropbox.com/s/3u4nuj039itzg81/eukaryotes.tsv?dl=1")
```

```
## Rows: 11508 Columns: 19
## -- Column specification -----
## Delimiter: "\t"
## chr (10): organism_name, bioproject_accession, group, subgroup, assembly_ac...
## dbl (7): taxid, bioproject_id, size_mb, gc, scaffolds, genes, proteins
## date (2): release_date, modify_date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Vi tager udgangspunkt i kun fire variabler, så for at gøre tingene mere overskuelige, har jeg brugt `select()` til at kun får de fire variabler `organism_name`, `center`, `group` og `subgroup` i en dataframe

```
#eukaryotes_full <- eukaryotes
eukaryotes_subset <- eukaryotes %>% select(organism_name, center, group, subgroup)
eukaryotes_subset %>% glimpse()
```

```
## Rows: 11,508
## Columns: 4
## $ organism_name <chr> "Pyropia yezoensis", "Emiliania huxleyi CCMP1516", "Arab-
```

```
## $ center      <chr> "Ocean University", "JGI", "The Arabidopsis Information ~
## $ group       <chr> "Other", "Protists", "Plants", "Plants", "Plants", "Plan~
## $ subgroup    <chr> "Other", "Other Protists", "Land Plants", "Land Plants", ~
```

Lad os forestille os, at vi gerne vil beregne antallet af unikke organismer (variablen `organism_name`). Der er en funktion der hedder `n_distinct` som beregner antallet af unikke værdier i en vector/variable. Her vælger vi `organism_name` og så tilføjer `n_distinct()`.

```
eukaryotes_subset %>%
  select(organism_name) %>%
  n_distinct()
```

```
## [1] 6111
```

Lad os forestille os, at vi også er interesseret i antallet af unikke værdier i variablerne `center`, `group` og `subgroup` - som er de tre andre kolonner i datasættet. Vi har forskellige muligheder:

- Skrive dem ud - men hvad nu hvis vi havde 100 variabler at håndtere?

```
eukaryotes_subset %>% select(organism_name) %>% n_distinct()
eukaryotes_subset %>% select(center) %>% n_distinct()
eukaryotes_subset %>% select(group) %>% n_distinct()
eukaryotes_subset %>% select(subgroup) %>% n_distinct()
```

```
## [1] 6111
## [1] 2137
## [1] 5
## [1] 19
```

- Vi kræver en mere automatiske løsning på det. Vi bruger ikke tid på det her, men der er den traditionelle programmering løsning: for loop, som fungerer også i R:

```
col_names <- names(eukaryotes_subset)

for(column_name in col_names)
{
  print(eukaryotes_subset %>%
        select(column_name) %>%
        n_distinct())
}

## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(column_name)` instead of `column_name` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

## [1] 6111
```

```
## [1] 2137
## [1] 5
## [1] 19
```

Man i teorien kan holde sig til for loops men jeg vil gerne præsentere den **tidyverse** løsning, som bliver mere intuitiv og nemmere for ændre at læse koden når man er vant til det (det integrerer også bedre med de andre **tidyverse** pakker).

### 7.2.1 Introduktion til `map()` funktioner

Den **tidyverse** løsning er såkaldte de `map()` funktioner, som er en del af pakken **purrr**. Jeg introducerer dem her frem for de base-R løsninger ikke bare fordi de er **tidyverse**, men fordi de er en meget fleksibel og nemt at forstå tilgang, når man vænner sig til dem.

Jeg viser hvordan de fungere igennem `eukaryotes` og bagefter introducerer dem i konteksten af custom funktioner og `nest()` som kan bruges til at opdele datasættet indtil forskellige dele (ovenpå hvori man kan gentage samme process).

Man anvender `map()` ved at angiv funktionen navn `n_distinct` indenfor `map()`, og `map()` beregner `n_distinct()` for hver kolon i datasættet.

```
eukaryotes_subset %>% map(n_distinct) #do 'n_distinct' for every single column
```

```
## $organism_name
## [1] 6111
##
## $center
## [1] 2137
##
## $group
## [1] 5
##
## $subgroup
## [1] 19
```

Så kan man se, at vi har fået en `list` tilbage, med en tal som viser antallet af unikke værdier til hver af de fire kolonner. Det fungerer lidt som den base-R funktion `apply`, men med `apply` skal man bruge 2 i anden plads til at fortælle, at vi gerne vil iterate over kolonnerne.

```
apply(eukaryotes_subset, 2, n_distinct)
```

| ## organism_name | center | group | subgroup |
|------------------|--------|-------|----------|
| ## 6111          | 2137   | 5     | 19       |

Bemærk at vi har fået her en vector af tal tilbage, men vi fået en `list` med `map`. Der er faktisk andre varianter af `map` som kan benyttes til at give resultatet som

andre data typer. For eksempel, kan man bruge `map_dbl()` til at få en double `dbl` tilbage - en vector af tal ligesom vi fæt med `apply` i ovenstående.

```
# Apply n_distinct to all variables, returning a double
eukaryotes_subset %>% map_dbl(n_distinct)
```

```
## organism_name      center      group      subgroup
##           6111       2137        5          19
```

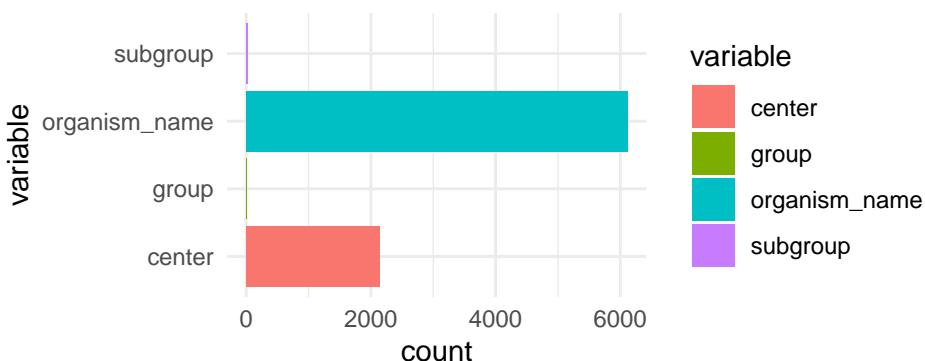
Man kan også bruge `map_df()` for at få en dataramme (`tibble`) tilbage - det er særligt nyttigt for os, fordi vi tager altid udgangspunkt i en dataramme når vi skal få lavet et plot.

```
# Apply n_distinct to all variables, returning a dataframe
eukaryotes_subset %>% map_df(n_distinct)
```

```
## # A tibble: 1 x 4
##   organism_name center group subgroup
##       <int>    <int> <int>     <int>
## 1       6111     2137     5         19
```

For eksempel, kan man tilføje de tal fra `map_df` direkte ind i et ggplot.

```
eukaryotes_subset %>%
  map_df(n_distinct) %>%
  pivot_longer(everything(), names_to = "variable", values_to = "count") %>%
  ggplot(aes(x = variable, y = count, fill = variable)) +
  geom_col() +
  coord_flip() +
  theme_minimal()
```



### 7.2.2 Reference for the different map functions

| Funktion               | Beskrivelse               |
|------------------------|---------------------------|
| <code>map_lgl()</code> | returns a logical         |
| <code>map_int()</code> | returns an integer vector |

| Funktion               | Beskrivelse                |
|------------------------|----------------------------|
| <code>map dbl()</code> | returns a double vector    |
| <code>map chr()</code> | returns a character vector |
| <code>map df()</code>  | returns a data frame       |

## 7.3 Custom funktioner

Vi kan lave vores egne funktioner og betnytter dem indenfor map til at yderligere øge fleksibiliteten i R. For eksempel, kan det være at vi har en bestemt idé overfor, hvordan vi gerne vil normalisere vores data, og der eksisterer ikke en relevant funktion indenfor R i forvejen.

### 7.3.1 Simple functions

Vi starter med en simpel funktion fra base-R og så forklare den i den table bagefter. Vi bruger mest en anden form af funktioner i **tidyverse** som vi kigger på næste, men koncepten er den samme.

```
my_function <- function(.x)
{
  return(sum(.x)/length(.x))
}
```

| Kode                            | Beskrivelse  |
|---------------------------------|--|
| <code>my_function_name</code>   | funktion navn                                      |
| <code>&lt;- function(.x)</code> | fortæl R, at vi lave en funktion med nogle data .x |
| <code>sum(.x)/length(.x)</code> | brug data .x til at beregne middelværdi            |
| <code>return()</code>           | håd funktionen skal output - her middelværdi       |

Lad også afprøve vores nye funktion ved at beregne den gennemsnitlige værdi for `Sepal.Length` i `iris`.

```
my_function(iris$Sepal.Length)
mean(iris$Sepal.Length)
```

```
## [1] 5.843333
## [1] 5.843333
```

### 7.3.2 Custom functions with mapping

Indenfor den **tidyverse** bruger man en lidt anden måde at skrive samme funktion på.

```
my_function <- ~ sum(.x)/length(.x)
```

- ~ betyder at vi definere en funktion
- .x betyder de data, der vi angiver funktionen (for eksempel variablen Sepal.Length fra iris). Man bruger den symbol .x hver gang og R ved automatiske hvad det betyder.

Vi kan bruge my\_function indenfor map() for at beregne den gennemsnitlige værdi for alle variabler (uden Species), og vi kan se at vi få tilsvarende resultat til funktionen mean():

```
iris %>%
  select(-Species) %>%
  map_df(my_function)

iris %>%
  select(-Species) %>%
  map_df(mean)

## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##       <dbl>      <dbl>      <dbl>      <dbl>
## 1       5.84     3.06      3.76      1.20

## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##       <dbl>      <dbl>      <dbl>      <dbl>
## 1       5.84     3.06      3.76      1.20
```

Man kan også placere funktionen direkte indenfor map\_df i stedet for at kalde den for nogle (fk. my\_funktion):

```
iris %>%
  select(-Species) %>%
  map_df(~ sum(.x)/length(.x)) #for each data column, compute the sum and divide by the length

## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##       <dbl>      <dbl>      <dbl>      <dbl>
## 1       5.84     3.06      3.76      1.20
```

Vi kan godt specificere andre funktioner.

```
iris %>%
  map_df(~nth(.x, 10)) #tag hver kolon, kalde det for .x og finde 10. værdi

## # A tibble: 1 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl>    <fct>
## 1       4.9       3.1       1.5       0.1  setosa
```

## 190 CHAPTER 7. FUNCTIONAL PROGRAMMING MED PURRR-PAKKEN

eller når `nth` is en **tidyverse** funktion kan vi bruge `%>%`:

```
iris %>%
  map_df(~.x %>% nth(10)) #tag hver kolon, kalde det for .x og finde 10. værdi

## # A tibble: 1 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1         4.9       3.1       1.5       0.1  setosa
```

Antallet af distinkt værdier som ikke er NA:

```
#tag hver kolon, kalde det for .x og beregne n_distinct
iris %>%
  map_df(~.x %>% n_distinct(na.rm = TRUE)) #n_distinct er fra tidyverse

## # A tibble: 1 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <int>      <int>      <int>      <int> <int>
## 1         35       23        43        22       3
```

Bemærk at hvis det er en indbygget funktion og vi benytter default parametre (altså `na.rm = FALSE` i ovenstående) kan man bare skrive:

```
iris %>%
  map_df(n_distinct)

## # A tibble: 1 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <int>      <int>      <int>      <int> <int>
## 1         35       23        43        22       3
```

Et andet eksempel: tilføje 3 og square:

```
iris %>%
  select(-Species) %>%
  map_df(~(.x + 3)^2) %>% head()

## # A tibble: 6 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##       <dbl>      <dbl>      <dbl>      <dbl>
## 1       65.6      42.2      19.4      10.2
## 2       62.4      36.0      19.4      10.2
## 3       59.3      38.4      18.5      10.2
## 4       57.8      37.2      20.2      10.2
## 5       64.0      43.6      19.4      10.2
## 6       70.6      47.6      22.1      11.6
```

Jo mere funktionen bliver indviklet, jo mere mening det giver at specificere den udenfor den `map()` funktion:

```

my_function <- ~(.x - mean(.x))^2 + 0.5*(.x - sd(.x))^2 #a long pointless function

iris %>%
  select(-Species) %>%
  map_df(my_function) #beregne my_function for hver kolon og output en dataramme

## # A tibble: 150 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <dbl>       <dbl>       <dbl>       <dbl>
## 1 9.68        4.89        5.63        1.16 
## 2 9.18        3.29        5.63        1.16 
## 3 8.80        3.84        6.15        1.16 
## 4 8.66        3.55        5.13        1.16 
## 5 9.41        5.30        5.63        1.16 
## 6 10.6         6.71        4.24        0.705
## 7 8.66        4.51        5.63        0.916
## 8 9.41        4.51        5.13        1.16 
## 9 8.46        3.06        5.63        1.16 
## 10 9.18        3.55        5.13        1.43 
## # ... with 140 more rows

```

### 7.3.3 Effekten af `map` på andre data typer

I ovenstående fokuser jeg på `map` i forhold til dataframes. I `group_by` + `nest` i nedenstående anvender man `map()` på en liste af dataframes, der hedder `data`, der tillader os at arbejde med datasæt hvert for sig. Det er derfor værd at tage lidt tid på at se hvordan `map()` behandler forskellige data typer.

#### input: vector

- `.x` referes til en værdi i vectoren. Hvis man tager integer `1:10` og anvender `map`, så tager man hvert tal hver for sig og beregner en funktion med det - i følgende simulere man et tal fra den normale fordeling med indstillingen `mean=.x`:

```

c(1:10) %>% map_dbl(~rnorm(1,mean=.x))

## [1] 1.982341 1.460212 5.566752 4.225571 6.115275 5.748129 9.289555
## [8] 7.149355 7.900811 11.354384

```

#### input: dataframe

- `.x` refereres til en variable fra dataframe. I ovenstående er første variable `int1` og i `map` tager man den første element med funktionen `pluck(1)`.

```
tibble("int1"=1:10,"int2"=21:30) %>% map(~.x %>% pluck(1))
```

```

## $int1
## [1] 1

```

```
##  
## $int2  
## [1] 21
```

Med nest kigger vi på den mulighed vore man laver map over en list, som bliver lavet med funktionen `nest()`.

#### input: list

- `.x` referes til et list element - i nedenstående er den første element `c(1,2)` så hvis man anvender funktionen `max` så tager man maksimum værdi (2 i dette tilfælde).

```
list(c(1,2),c(2,3),c(3,4)) %>% map(~max(.x))
```

```
## [[1]]  
## [1] 2  
##  
## [[2]]  
## [1] 3  
##  
## [[3]]  
## [1] 4
```

Bemærk at der der kommer kun et tal som resultat, kan man `map dbl` i stedet for `map` - så får man en vector som outputtet, selvom inputtet er en list.

```
list(c(1,2),c(2,3),c(3,4)) %>% map_dbl(~max(.x))
```

```
## [1] 2 3 4
```

#### list af dataframe

- `.x` referes til et datasæt - så kan man referer til de forskellige variabler i `.x` som man plejer i tidyverse.

Følgende er ligesom i tilfælde med koncepten “nesting” i næste sektion. Man tager en list af tibbles og vælger det første tal fra variablen `int`.

```
list(tibble("int"=1:10),tibble("int"=1:10),tibble("int"=1:10)) %>%  
  map_int(~.x %>% pluck("int",1))
```

```
## [1] 1 1 1
```

## 7.4 Nesting `nest()`

Vi kommer til at se i næste lektion, at det er meget nyttige at bruge funktionen `nest()` for at få svar på adskillige statistiske spørgsmål. Det kan være for eksempel:

- Vi har lavet 10 eksperimental under lidt forskellige konditioner, og gerne vil lave præcis samme analyse på alle 10.
- Vi har 5 forskellige type bakterier med 3 replikater til hver, og gerne vil transformere de data på samme måde efter bakterien og replikat.

Funktionen `nest()` kan virke lidt abstract i starten men koncepten er faktisk ret simpelt. Vi kan opdele vores datasæt (som indeholder vores forskellige konditioner/replikats etc.) med `group_by()` og så bruge `nest()` til at gemme de opdelt "sub" datasæt i en liste. De bliver gemt indenfor en kolon i en `tibble`, og det gøre det bekvemt at arbejde med de forskellige datasæt på samme tid (med hjælp af `map()`).

The diagram illustrates the transformation of a wide-format data frame into a nested long-format data frame using the `nest()` function. On the left, a wide-format data frame is shown with columns `Species`, `S.L`, `S.W`, `P.L`, and `P.W`. The data consists of five rows for each species: `setosa`, `versi`, and `virgini`. An arrow points from this wide-format frame to the right, where the same data is shown in a nested long-format structure. The nested structure is organized by `Species` (rows) and `Measurements` (columns). For each species, there is a separate `tibble` containing the measurements. The `Measurements` tibble has columns `S.L`, `S.W`, `P.L`, and `P.W`, with values corresponding to the original wide-format frame. The entire process is labeled "i.e.,".

| Species | S.L | S.W | P.L | P.W |
|---------|-----|-----|-----|-----|
| setosa  | 5.1 | 3.5 | 1.4 | 0.2 |
| setosa  | 4.9 | 3.0 | 1.4 | 0.2 |
| setosa  | 4.7 | 3.2 | 1.3 | 0.2 |
| setosa  | 4.6 | 3.1 | 1.5 | 0.2 |
| setosa  | 5.0 | 3.6 | 1.4 | 0.2 |
| versi   | 7.0 | 3.2 | 4.7 | 1.4 |
| versi   | 6.4 | 3.2 | 4.5 | 1.5 |
| versi   | 6.9 | 3.1 | 4.9 | 1.5 |
| versi   | 5.5 | 2.3 | 4.0 | 1.3 |
| versi   | 6.5 | 2.8 | 4.6 | 1.5 |
| virgini | 6.3 | 3.3 | 6.0 | 2.5 |
| virgini | 5.8 | 2.7 | 5.1 | 1.9 |
| virgini | 7.1 | 3.0 | 5.9 | 2.1 |
| virgini | 6.3 | 2.9 | 5.6 | 1.8 |
| virgini | 6.5 | 3.0 | 5.8 | 2.2 |

| Species | Measurements |     |     |     |
|---------|--------------|-----|-----|-----|
|         | S.L          | S.W | P.L | P.W |
| setosa  | 5.1          | 3.5 | 1.4 | 0.2 |
|         | 4.9          | 3.0 | 1.4 | 0.2 |
|         | 4.7          | 3.2 | 1.3 | 0.2 |
|         | 4.6          | 3.1 | 1.5 | 0.2 |
|         | 5.0          | 3.6 | 1.4 | 0.2 |
| versi   | 7.0          | 3.2 | 4.7 | 1.4 |
|         | 6.4          | 3.2 | 4.5 | 1.5 |
|         | 6.9          | 3.1 | 4.9 | 1.5 |
|         | 5.5          | 2.3 | 4.0 | 1.3 |
|         | 6.5          | 2.8 | 4.6 | 1.5 |
| virgini | 6.3          | 3.3 | 6.0 | 2.5 |
|         | 5.8          | 2.7 | 5.1 | 1.9 |
|         | 7.1          | 3.0 | 5.9 | 2.1 |
|         | 6.3          | 2.9 | 5.6 | 1.8 |
|         | 6.5          | 3.0 | 5.8 | 2.2 |

Lad os opdele `eukaryotes_subset` efter variablen 'group' og anvende `nest()`:

```
eukaryotes_subset_nested <- eukaryotes_subset %>%
  group_by(group) %>%
  nest()
```

```
eukaryotes_subset_nested
```

```
## # A tibble: 5 x 2
## # Groups:   group [5]
##   group    data
##   <chr>    <list>
## 1 Other    <tibble [51 x 3]>
## 2 Protists <tibble [888 x 3]>
## 3 Plants   <tibble [1,304 x 3]>
## 4 Fungi   <tibble [6,064 x 3]>
```

```
## 5 Animals <tibble [3,201 x 3]>
```

Vi kan se at vi har to variabler - `group` og `data`. Variablen `data` er indeholde faktisk fem dataramme (tibble), for eksempel den første datasæt har kun observationerne hvor `group` er lig med "Other", den anden dataset har kun observationerne hvor `group` er lig med "Protists" osv.

Vi kan tjekke ved at kig på den første datasæt: her er to måder at gøre det på:

```
first_dataset <- eukaryotes_subset_nested$data[[1]]
first_dataset <- eukaryotes_subset_nested %>% pluck("data", 1)
first_dataset %>% head()
```

| ## # A tibble: 6 x 3                       |                                 |       |          |
|--|---------------------------------|-------|----------|
| ## organism_name                           | center                          |       | subgroup |
| ## <chr>                                   | <chr>                           |       | <chr>    |
| ## 1 Pyropia yezoensis                     | Ocean University                |       | Other    |
| ## 2 Thalassiosira pseudonana CCMP1335     | Diatom Consortium               |       | Other    |
| ## 3 Guillardia theta CCMP2712             | JGI                             |       | Other    |
| ## 4 Cyanidioschyzon merolae strain 10D    | National Institute of Genetics~ | Other | Other    |
| ## 5 Galdieria sulphuraria                 | Galdieria sulphuraria Genome P~ | Other | Other    |
| ## 6 Phaeodactylum tricornutum CCAP 1055/1 | Diatom Consortium               |       | Other    |

Hvis vi gerne vil tilbage til vores oprindeligt datasæt, kan vi brug `unnest()` og specificerer kolonnen `data`:

```
eukaryotes_subset_nested %>%
  unnest(data) %>%
  head()
```

| ## # A tibble: 6 x 4                             |                           |       |          |
|--|---------------------------|-------|----------|
| ## # Groups: group [1]                           |                           |       |          |
| ## group organism_name                           | center                    |       | subgroup |
| ## <chr> <chr>                                   | <chr>                     |       | <chr>    |
| ## 1 Other Pyropia yezoensis                     | Ocean University          |       | Other    |
| ## 2 Other Thalassiosira pseudonana CCMP1335     | Diatom Consortium         |       | Other    |
| ## 3 Other Guillardia theta CCMP2712             | JGI                       |       | Other    |
| ## 4 Other Cyanidioschyzon merolae strain 10D    | National Institute of Ge~ | Other | Other    |
| ## 5 Other Galdieria sulphuraria                 | Galdieria sulphuraria Ge~ | Other | Other    |
| ## 6 Other Phaeodactylum tricornutum CCAP 1055/1 | Diatom Consortium         |       | Other    |

Spørgsmålet er: hvordan kan vi inddrage "nested" data indenfor vores analyser?

#### 7.4.1 Anvende `map()` med nested data

De fleste gange vi arbejder med nested data, er fordi vi gerne vil lave samme ting på hver af de "sub" datasæt. Derfor hænger det sammen med funktionen `map()`. Den typiske process er:

- Tag nested datasæt

- Tilføj en ny kolon med `mutate()`, hvor vi:
- Tag hver datasæt fra kolonnen `data` og brug `map()`, i nedenstående tilfælde til at finde antallet af rækkerne.

```
eukaryotes_subset_nested %>%
  mutate(n_row = map_dbl(data,nrow))

## # A tibble: 5 x 3
## # Groups:   group [5]
##   group     data           n_row
##   <chr>    <list>          <dbl>
## 1 Other    <tibble [51 x 3]>     51
## 2 Protists <tibble [888 x 3]>    888
## 3 Plants   <tibble [1,304 x 3]>  1304
## 4 Fungi    <tibble [6,064 x 3]> 6064
## 5 Animals  <tibble [3,201 x 3]> 3201
```

Vi kan også bruge en custom funktion. I nedenstående beregne man antallet af unikke organisme fra variablen `organism_name` i datasættet. Husk:

- ~ betyder at vi lave en funktion, som kommer til at fungere for alle de fem datasæt.
- Tag et datasæt og kalde det for `.x` - det referer til en bestemt datasæt fra en af de fem datasæt som hører under kolonnen `data` i den `nest()` data.
- Vælg variablen `organism_name` fra `.x`
- Beregn `n_distinct`

```
n_distinct_organisms <- ~ .x %>% #take data
  select(organism_name) %>% #select organism name
  n_distinct #give back distinct

#repeat function for each of the five datasets:
eukaryotes_subset_nested %>%
  mutate(n_organisms = map_dbl(data, n_distinct_organisms))
```

```
## # A tibble: 5 x 3
## # Groups:   group [5]
##   group     data           n_organisms
##   <chr>    <list>          <dbl>
## 1 Other    <tibble [51 x 3]>     35
## 2 Protists <tibble [888 x 3]>    490
## 3 Plants   <tibble [1,304 x 3]>  673
## 4 Fungi    <tibble [6,064 x 3]> 2926
## 5 Animals  <tibble [3,201 x 3]> 1987
```

Her er en anden eksempel. Her handler det om de `eukaryotes` data (ikke den subset), som har oplysninger om fk. GC-content med variablen `gc`. Her bruger vi `pull` i stedet for `select` - det er næsten den samme men med `pull()` få vi en vector som fungerer med `median` som er en base-R funktion.

```
# # func_gc <- ~ .x %>%
#   pull(gc) %>%           # ligesom select men vi har bruge for en vector for at beregne medie
#   median(.x,na.rm=T) # `na.rm` fjerne `NA` værdier)

func_gc <- ~median(.x %>% pull(gc),na.rm=T)

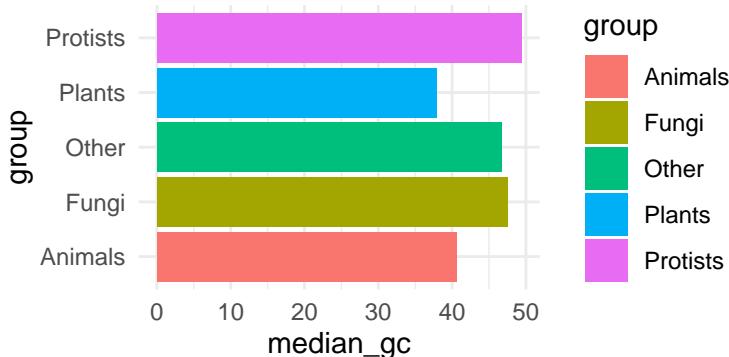
ekaryotes_gc_by_group <- eukaryotes %>%
  group_by(group) %>%
  nest() %>%
  mutate("median_gc"=map_dbl(data, func_gc))

ekaryotes_gc_by_group
```

## # A tibble: 5 x 3  
## # Groups: group [5]  
## group data median\_gc  
## <chr> <list> <dbl>  
## 1 Other <tibble [51 x 18]> 46.7  
## 2 Protists <tibble [888 x 18]> 49.4  
## 3 Plants <tibble [1,304 x 18]> 37.9  
## 4 Fungi <tibble [6,064 x 18]> 47.5  
## 5 Animals <tibble [3,201 x 18]> 40.6

Og jeg kan bruge resultatet ind i et plot ligesom vi plejer:

```
ekaryotes_gc_by_group %>%
  ggplot(aes(x=group,y=median_gc,fill=group)) +
  geom_bar(stat="identity") +
  coord_flip() +
  theme_minimal()
```



flere statistik på en gang

Lave funktionerne:

```
func_genes <- ~median(.x %>% pull(genes), na.rm=T)
func_proteins <- ~median(.x %>% pull(proteins),na.rm=T)
```

```
func_size <- ~median(.x %>% pull(size_mb), na.rm=T)
```

Anvende `nest()`:

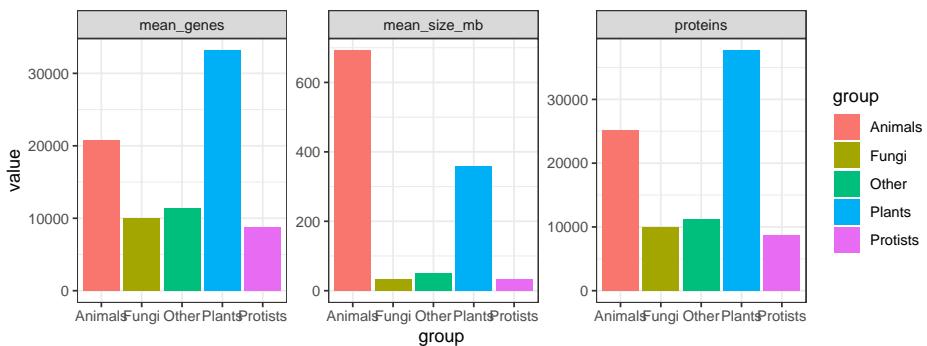
```
eukaryotes_nested <- eukaryotes %>%
  group_by(group) %>%
  nest()
```

Tilføje resultatet over de fem datasæt med `mutate()`:

```
eukaryotes_stats <- eukaryotes_nested %>%
  mutate(mean_genes = map_dbl(data, func_genes),
        proteins = map_dbl(data, func_proteins),
        mean_size_mb = map_dbl(data, func_size))
```

Husk at fjerne kolonnen `data` før man anvende `pivot_longer()` (ellers får man en advarsel):

```
eukaryotes_stats %>%
  select(-data) %>%
  pivot_longer(-group) %>%
  ggplot(aes(x=group, y=value, fill=group)) +
  geom_bar(stat="identity") +
  facet_wrap(~name, scales="free", ncol=4) +
  theme_bw()
```



## 7.5 Andre brugbar purrr

### 7.5.1 `map2()` function for flere inputs

Funktionen `map2()` kan bruges ligesom `map()` men tager 2 "input" i stedet for kun én. I følgende eksempel angiver jeg to kolonner fra datasættet `eukaryotes_stats`, `mean_genes` og `proteins` og beregner `sum`, som bliver lagret takket være funktionen `mutate` som colstat.

```
eukaryotes_stats %>% mutate(colstat = map2_dbl(mean_genes, proteins, sum))
```

```
## # A tibble: 5 x 6
## # Groups:   group [5]
##   group    data      mean_genes  proteins mean_size_mb colstat
##   <chr>   <list>        <dbl>     <dbl>       <dbl>    <dbl>
## 1 Other    <tibble [51 x 18]>    11354.    11240.      49.7  22594
## 2 Protists <tibble [888 x 18]>    8813     8628.      33.5  17442.
## 3 Plants   <tibble [1,304 x 18]>   33146.    37660.      358.  70806.
## 4 Fungi    <tibble [6,064 x 18]>   10069    10034.      32.2  20103
## 5 Animals  <tibble [3,201 x 18]>   20733    25161.      692.  45894
```

Bemærk at præcis samme resultat kan fås ved at bare lægger de to kolonner sammen:

```
eukaryotes_stats %>% mutate(colstat2 = mean_genes + proteins)
```

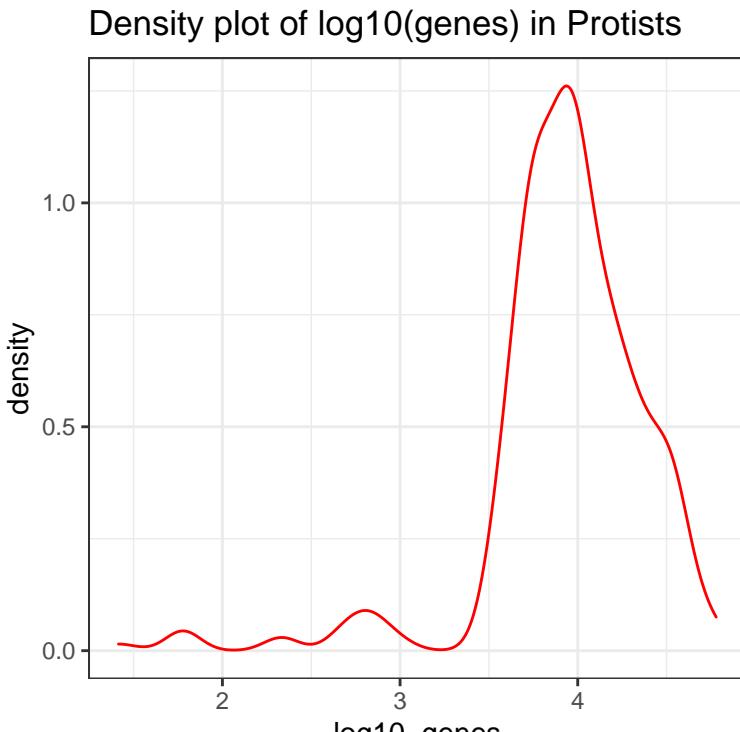
```
## # A tibble: 5 x 6
## # Groups:   group [5]
##   group    data      mean_genes  proteins mean_size_mb colstat2
##   <chr>   <list>        <dbl>     <dbl>       <dbl>    <dbl>
## 1 Other    <tibble [51 x 18]>    11354.    11240.      49.7  22594
## 2 Protists <tibble [888 x 18]>    8813     8628.      33.5  17442.
## 3 Plants   <tibble [1,304 x 18]>   33146.    37660.      358.  70806.
## 4 Fungi    <tibble [6,064 x 18]>   10069    10034.      32.2  20103
## 5 Animals  <tibble [3,201 x 18]>   20733    25161.      692.  45894
```

Der er dog mange mere indviklet situationer hvor man ikke kan gå udenom map2:

```
eukaryotes_stats_with_plots <- eukaryotes_stats %>%
  mutate(log10_genes = map(data, ~log10(.x %>% pull(genes)))) %>%
  mutate(myplots = map2(group, log10_genes,
    ~ tibble("log10_genes"=y) %>%
      ggplot(aes(x=log10_genes)) +
      geom_density(colour="red", alpha=0.3) +
      theme_bw() + ggtitle(paste("Density plot of log10(genes) in",
      group))))
```

```
eukaryotes_stats_with_plots %>% pluck("myplots", 2)
```

```
## Warning: Removed 613 rows containing non-finite values (stat_density).
```



Der er flere `map` funktioner der tager flere input fra. `pmap` - jeg har ikke tid til at dække dem men du kan godt læse om dem hvis du har brug for dem: <https://purrr.tidyverse.org/reference/map2.html>

### 7.5.2 Transformer numeriske variabler

Som sidste bemærk, her er en brugbar variant af `map` - her kan man lave noget på kun bestemte variabler - for eksempel i følgende anvender jeg funktionen `log2()` på alle numeriske variabler. Bemærk at man er nødt til at anvende `as_tibble()` igen bagefter (som kun fungerer hvis alle variabler har stadig samme længde efter `map_if()`):

```
eukaryotes %>% map_if(is.numeric, ~log2(.x)) %>% as_tibble()
```

```
## # A tibble: 11,508 x 19
##   organism_name      taxid bioproject_acce~ bioproject_id group subgroup size_mb
##   <chr>             <dbl> <chr>           <dbl> <chr> <chr> <dbl>
## 1 Pyropia yezoensis 11.4 PRJNA589917     19.2 Other Other    6.75
## 2 Emiliania huxley~ 18.1 PRJNA77753      16.2 Prot~ Other P~    7.39
## 3 Arabidopsis thal~ 11.9 PRJNA10719      13.4 Plan~ Land Pl~    6.90
## 4 Glycine max        11.9 PRJNA19861      14.3 Plan~ Land Pl~    9.94
## 5 Medicago truncat~ 11.9 PRJNA10791      13.4 Plan~ Land Pl~    8.69
## 6 Solanum lycopers~ 12.0 PRJNA119       6.89 Plan~ Land Pl~    9.69
```

```

## 7 Hordeum vulgare ~ 16.8 PRJEB34217          19.1 Plan~ Land Pl~ 12.1
## 8 Oryza sativa Jap~ 15.3 PRJNA12269          13.6 Plan~ Land Pl~ 8.55
## 9 Triticum aestivum 12.2 PRJNA392179         18.6 Plan~ Land Pl~ 13.9
## 10 Zea mays      12.2 PRJNA10769           13.4 Plan~ Land Pl~ 11.1
## # ... with 11,498 more rows, and 12 more variables: gc <dbl>,
## #   assembly_accession <chr>, replicons <chr>, wgs <chr>, scaffolds <dbl>,
## #   genes <dbl>, proteins <dbl>, release_date <date>, modify_date <date>,
## #   status <chr>, center <chr>, biosample_accession <chr>

```

## 7.6 Problemstillinger

**Problem 1)** Lave Quiz på Absalon “Quiz - functional programming”

---

**Problem 2) *map()* øvelse**

Eksempel:

```
diamonds %>% select(cut,color,depth) %>% map_df(n_distinct)
```

```

## # A tibble: 1 x 3
##       cut   color depth
##   <int> <int> <int>
## 1     5     7    184

```

Husk også referencen med de forskellige varianter af *map()* som kan bruges for at få en anden output type.

Indlæse *diamonds* med *data(diamonds)*. Brug *map()* funktioner til at beregne følgende:

- a) Select variabler *cut*, *color* og *clarity* og beregne antallet af distinkt værdier til hver (anvende funktionen *n\_distinct*). Resultatet skulle være en list (anvende default *map()* funktion).
  - b) Select variabler som er numeriske og beregne den median værdi til hver. Resultatet skal være en double.
  - c) Brug alle variabler og return de datatyper (funktion en *typeof*). Resultatet skal være en dataramme.
- 

**Problem 3) *map()* øvelse med custom funktioner**

Indlæse *diamonds* med *data(diamonds)*.

Husk at når man inddrager nogle data *.x*, for eksempel når man vil bruge en custom funktion eller specificerer non-default indstillinger såsom *na.rm=TRUE* (for at fjerne NA værdier i beregningen) i funktionen, skal man angiv *~* i starten:

```
diamonds %>% map_df(n_distinct) #specificere funktion under instillinger
diamonds %>% map_df(~n_distinct(.x,na.rm = TRUE)) #custom funktion
```

a) Afprøve følgende kode linjer og beskrive hvad der sker.

```
diamonds %>% select(carat, depth, price) %>% map_df(~mean(.x,na.rm=T))
diamonds %>% filter(cut=="Ideal") %>% select("color","clarity") %>% map(~nth(.x,100))
diamonds %>% select(carat, depth, price) %>% map_df(~ifelse(.x>mean(.x),"big_value","small_value"))
```

Brug custom funktioner indenfor `map()` til at beregne følgende:

- Udvælg variabler `carat`, `depth`, `table` og `price` og for hver kolonne tilføj tre og så dernæst tag square ( $\wedge 2$ ). Resultatet skal være en dataramme.
  - Udvælg numeriske variabler og angiv `TRUE` hvis den først værdi i kolonnen (betegnet med `nth(.x,1)`) er større en den median værdi i kolonnen, ellers `FALSE`. Resultatet skal være en logical.
- 

#### Problem 4) `map` på forskellige datatyper

Vi har set, at når vi kører `map` på en dataframe, laver man en funktion over hver kolonne. Lad os se hvad der forgår hvis man har andre datatyper:

a) (En vector af tal)

- Kør følgende - hvad sker der?

```
c(1:10) %>% map_dbl(~rnorm(1,mean=.x, sd=1))
```

- Tag udgangspunkt i `c(1:10)` og anvend `map_dbl` til at beregne den `log()` af hvert tal i vektoren (angiv indstillingen `base=5`):

```
c(1:10) %>% ???
```

b) (En list af tibbles)

```
my_list_of_tibbles <- list( tibble("letter" = c("a","a","a"), "number" = c(1,3,3)),
                           tibble("letter" = c("a","b","b"), "number" = c(3,3,1)),
                           tibble("letter" = c("a","b","a"), "number" = c(2,3,3)))
```

- Kør følgende kode, der tager den første `tibble` udvælger observationer hvor `letter` = "a" og dernæst beregner den middelværdi af "number".

```
my_list_of_tibbles %>% pluck(1) %>% filter(letter=="a") %>% summarise(mn = mean(number))
```

- Nu at jeg kan se, at min kode virker til den første `tibble`, kan jeg ændre den til en funktion og anvende den med `map()` til at lave samme process på alle dataframes - prøve at køre følgende.

```
my_list_of_tibbles %>% map(~.x %>% filter(letter=="a") %>% summarise(mn = mean(number)))
```

- Tag udgangspunkt i `my_list_of_tibbles` og for hver tibble select alle tilfælde hvor variablen `number` er 3 og dernæst optæl hvor mange `a` er tilbage i variablen `letter` (man kan afprøve på den første `tibble` hvis det hjælper).
- 

**Problem 5) `group_by/nest` øvelse**

a) For datasættet `iris`, anvend `group_by(Species)` og tilføj dernæst `nest()`, og kigger på resultatet.

b)

- tilføj `pull(data)` og se på resultatet
- fjern `pull(data)` og tilføj `pluck("data", 1)` i stedet for, for at se den første dataramme.
- tilføj `unnest(data)` i stedet for og se på resultatet
- tilføj følgende i stedet for og prøve at forstå hvad der sker:
  - `mutate(new_column_nrow = map_dbl(data, nrow))`
  - `mutate(new_column_ratio = map(data, ~(.x$Sepal.Width/.x$Sepal.Length)))`

b) *Skift mellem long og wide form*

- Afprøv også følgende funktion med `map()`-funktionen på kolonnen `data` (angiv ny kolonnenavne `new_column_long` i `mutate()`).

*#add an id and put data into long form*

```
my_func_longer <- ~.x %>% mutate(id=1:50) %>% pivot_longer(cols=-id)
```

- Skriv en funktion og anvend den på kolonnen `new_column_long`, der laver hvert datasæt i kolonnen om til wide-form igen (lav ny kolonne "new\_column\_wide").

*my\_func\_wider <- ??? #lave long data om til wide form (husk at angiv names\_from,values\_*

```
iris_nested_stats %>% ???#anvend din funktion
```

- Anvend `pluck` på kolonnerne `new_column_long` og `new_column_wide` til at kigge på den første datasæt for at se, om du har outputtet i long/wide form som forventede.

---

**Problem 6) `group_by/nest + map` på datasættet `mtcars`**

a) Åbn `mtcars` med `data(mtcars)` og anvende `group_by()/nest()` for at opdele datasættet i tre efter variablen `cyl`.

b) Lave nye kolonner med `map` og inddrage hensigtsmæssige funktioner, som beskriver dine tre datasæt, der er lagret i kolonne `data` (outputtet skal være `dbl`):

- Antal observationer
  - Korrelationskoefficient mellem `wt` og `drat` (funktionen `cor`)
  - Antal unikke værdier fra variablen `gear`
- c) Omsætte dine statistikke til et plot for at sammenligne de tre datasæt.
- 

**Problem 7)** *Forberedelse til næste lektion*

For at bedre kunne se værdien af at bruge `group_by()`/`nest()` + `map()` kan vi gennemgå en simpel eksampel som indledning til vores næste lektion.

Tag funktionen:

```
my_func <- ~ t.test(.x$Petal.Width,.x$Sepal.Length)
```

- anvend `group_by(Species)` og dernæst `nest()` på datasættet `iris`.
  - tilføj `mutate()` til at lave en ny kolon som hedder `t_test` og bruge funktionen indenfor `map()` på kolonnen `data`.
  - tilføj `pull(t_test)` til din kommando - man får de tre t-tests frem, som man lige har beregnet.
  - prøv `unnest(t_test)` i stedet for `pull(t_test)` - man får en advarsel fordi de t-test resultater ikke er i en god form til at vise indenfor en dataramme. Vi vil gerne ændre deres output til tidy-form først.
- 

- Nu installer R-pakken `broom` (`install.packages("broom")`)
- Lav samme som ovenstående men bruge følgende funktion i stedet for `(glance())` få statistikkerne fra `t.test()` ind i en pæn form (`tidy()`)

```
library(broom)
my_func <- ~ t.test(.x$Petal.Width,.x$Sepal.Length) %>% glance()
```

- Tilføj `pull` eller `unnest` til din kommando som før og se på resultatet.
  - Man får en pæn dataramme frem med alle de forskellige statistikker fra `t.test()`.
  - Vælge en statistik og omsætte den til et plot.
- 

**Problem 8)** Hvis du færdig før tid kan du se videoer til næste emne i morgen :)

## 7.7 Ekstra notater og næste gang

<https://r4ds.had.co.nz/iteration.html> <https://sanderwuyts.com/en/blog/purrr-tutorial/>



# Chapter 8

## Visualisering af trends



```
#load following packages
library(ggplot2)
library(tidyverse)
library(broom)
library(glue)
library(ggsignif)
```

### 8.1 Indledning og læringsmålene

#### 8.1.1 Læringsmålene

Du skal være i stand til at

- Anvende `nest()` og `map()` strukturen til at gentage en korrelation analyse over flere forskellige datasæt.
- Bruge `ggplot` funktion `geom_smooth()` til at visualisere lineær regression eller loess-kurver.
- Kombinere `map()`/`nest()` og `lm()` til at beregne regression statistikker for flere lineær regression modeller på samme tid og sammenligne dem med `anova()`.

### 8.1.2 Introduktion til chapter

I dette kapitel viser jeg flere eksempler på processen, hvor man anvender `group_by()` og `nest()` og dernæst `map()`-funktioner for at lave reproducebar statistiske analyser. Vi fokuserer på eksempler med korrelationanalyse og lineær regression modeller, men den overordnede ramme kan anvendes i mange forskellige kontekster.

### 8.1.3 Video ressourcer

OBS der er mange videoer til i dag men de gentager samme process fra sidste emner med `group_by/nest` og `map` mange gange (med forskellige statistiske metoder).

- Video 1: Korrelation koefficient med `nest()` og `map()`
  - Jeg gennemgår processen langsomt med en korrelationsanalyse
  - Jeg introducerer `glance` til at lave outputtet fra statistikse methoder i tidy-form.

*OBS: Jeg sagde “antal gener” flere gange i videoen men variablen `log10_size_mb` er faktisk genomstørrelse i megabytes.*

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/709225323>

---

- Video 2: Lineær regression linjer med `ggplot2`
  - Jeg viser hvordan man tilføjer regression linjer på et plot
  - Jeg sammenligne linjen med resultatet fra `lm()`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/709225203>

---

- Video 3: Lineær regression med `nest()` og `map()`
  - Den proces igen fra Video 1 men anvendte på lineær regression

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/709225158>

---

- Video 4: Multiple lineær regression model
  - Sammen processen men med flere modeller og multiple uafhængige variabler

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/709225266>

---

- Video 5: anova+map (OBS muligvis mest udfordrende del i kurset)
  - Benytte funktionen `anova` for at sammenligne to modeller beregnede på datasættet `penguins` og få outputtet i “tidy”-form med funktionen `tidy()`
  - Lave en funktion med `anova`, der kan anvendes over alle arter med `map2()`
  - Omsætte p-værdier fra sammenligningerne til et plot og tilføj signifikans annotations

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/710108716>

## 8.2 nest() og map(): eksempel med korrelation

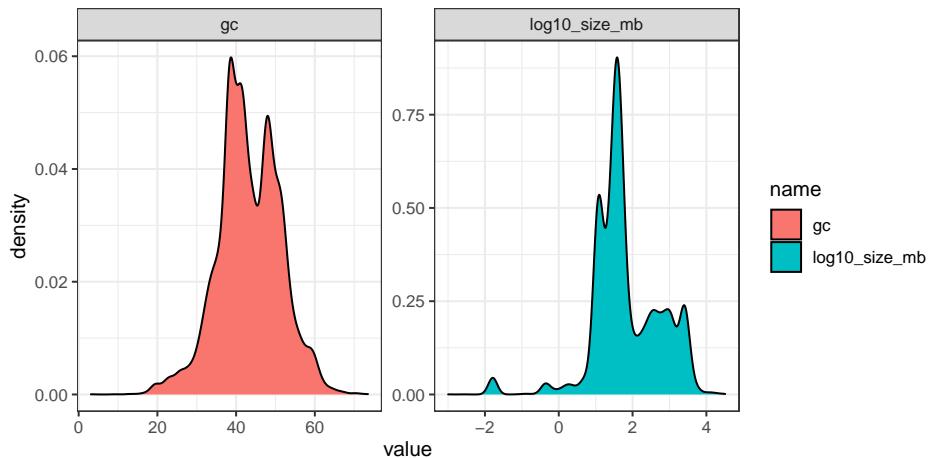
Man laver en korrelationsanalyse i R ved at benytte `cor.test()` (`cor()` virker også hvis du kun vil beregne koefficient og ikke signifikans). Forestille dig at du gerne vil finde ud af korrelationen mellem GC-indehold (variablen `gc`, procent G/C bases i genomet) og genomstørrelse (variablen `log10_size_mb`) i datasættet `eukaryotes` fra sidste lektion.

I følgende plotter jeg en density mellem `gc` og den transformerede variable `log10_size_mb` som er log10 genomstørrelse (ikke antal gener, som jeg sagde i videoen).

```
eukaryotes <- eukaryotes %>%
  mutate(log10_size_mb = log10(size_mb))

eukaryotes %>%
  mutate(log10_size_mb = log10(size_mb)) %>%
  select(log10_size_mb, gc) %>%
  pivot_longer(everything()) %>%
  ggplot(aes(x=value, fill=name)) +
  geom_density(colour="black") +
  facet_wrap(~name, scales="free") +
  theme_bw()

## Warning: Removed 388 rows containing non-finite values (stat_density).
```



Plottet ser ud til at have flere “peaks” og jeg mistænker, at der kan være nogle sub-strukturer indenfor de data - eksempelvis pga. de forskellige organismer grupper i variablen **Group** (Animals, Plants osv.). I følgende benytter jeg alligevel `cor.test()` til at teste for korrelation mellem `gc` og `log10_size_mb` over hele datasæt:

```
my_cor_test <- cor.test(eukaryotes %>% pull(gc),
                         eukaryotes %>% pull(log10_size_mb))

## 
## Pearson's product-moment correlation
##
## data: eukaryotes %>% pull(gc) and eukaryotes %>% pull(log10_size_mb)
## t = -15.678, df = 11118, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## -0.1652066 -0.1288369
## sample estimates:
##       cor
## -0.1470715
```

Outputtet fra `cor.test` (og mange andre metoder i R) er ikke særlig egnet til at bruge indenfor en dataframe, så jeg introducerer en funktion der hedder `glance()` som findes i R-pakken `broom`. Funktionen `glance()` anvendes til at tage outputtet fra en statistiske test (fl. `cor.test()` eller `lm()`) og lave det om til et `tidy` dataramme. Det gør det nemmere eksempelvis til at lave et plot, eller samler op statistikker fra forskellige tests.

```
library(broom)
my_cor_test %>% glance()

FALSE # A tibble: 1 x 8
```

```

FALSE   estimate statistic  p.value parameter conf.low conf.high method      alternative
FALSE     <dbl>     <dbl>     <dbl>     <int>     <dbl>     <dbl> <chr>      <chr>
FALSE 1   -0.147    -15.7 8.25e-55     11118    -0.165    -0.129 Pearson'~ two.sided

```

Man kan se, at over hele datasættet, er der en signifikant negativ korrelation (estimate  $-0.147$  og p-værdi  $8.25054 \times 10^{-55}$ ) mellem de to variabler. Men jeg er imidlertid stadig mistænksom overfor eventuelle forskelligheder blandt de fem grupper fra variablen `group`.

Jeg vil gerne gentage den samme analyse for de fem grupper fra variablen `group` hver for sig. En god tilgang til at undersøge det er at bruge den ramme med `group_by()` og `nest()` som vi lært sidste gange.

### 8.2.1 Korrelation over flere datasæt på en gang

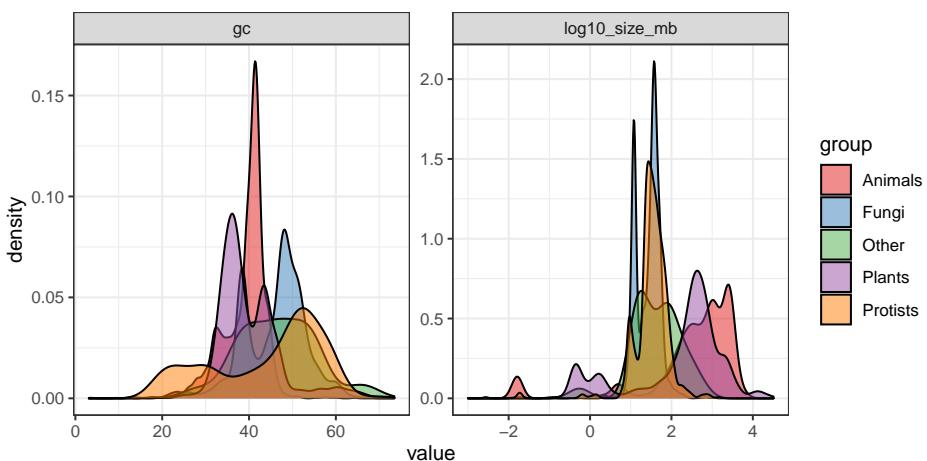
Jeg tjekker først fordelingen af de to variabler opdelt efter variablen `group`:

```

eukaryotes %>%
  select(log10_size_mb,gc,group) %>%
  pivot_longer(-group) %>%
  ggplot(aes(x=value,fill=group)) +
  geom_density(colour="black",alpha=0.5) +
  #geom_histogram(bins=40,alpha=0.5,colour="black") +
  scale_fill_brewer(palette = "Set1") +
  facet_wrap(~name,scales="free") +
  theme_bw()

## Warning: Removed 388 rows containing non-finite values (stat_density).

```



Man kan se, at der er forskelligheder blandt de fem grupper og der sagtens kan forekommer forskellige sammenhænge mellem de to variabler. Jeg benytter i følgende den `group_by()` + `nest()` ramme som blev introducerede sidste lektion.

### Step 1: Benytte `group_by()` + `nest()`

Jeg anvender `group_by()` på variablen `group` og så funktionen `nest()` for at adskille `eukaryotes` i fem forskellige datasæt (lagrede i samme dataframe i en kolonne der hedder `data`):

```
eukaryotes_nest <- eukaryotes %>%
  group_by(group) %>%
  nest()
eukaryotes_nest
```

```
## # A tibble: 5 x 2
## # Groups:   group [5]
##   group    data
##   <chr>    <list>
## 1 Other    <tibble [51 x 19]>
## 2 Protists <tibble [888 x 19]>
## 3 Plants   <tibble [1,304 x 19]>
## 4 Fungi    <tibble [6,064 x 19]>
## 5 Animals  <tibble [3,201 x 19]>
```

### Step 2: Definere korrelation funktion

Lad os definere den korrelation test mellem `gc` og `log10_size_mb` i en funktion.

- Brug `~` lige i starten for at fortælle R, at man arbejder med en funktion.
- Specifier et bestemt datasæt (som er en delmængde af `eukaryotes`) indenfor `cor.test()` med `.x`
- For det bestemt datasæt benytter jeg `.x %>% pull(gc)` og `.x %>% pull(size_mb)` til at udtrække de relevante vectorer for at udføre testen `cor.test`.

```
cor_test <- ~cor.test(.x %>% pull(gc),
                      .x %>% pull(log10_size_mb))
```

Vi vil gerne få statistikker fra `cor.test()` i en pæn form så vi tilføjer `glance()` til ovenstående funktion:

```
library(broom)
my_cor_test <- ~cor.test(.x$gc, log10(.x$size_mb)) %>% glance()
```

### Step 3: Bruge `map()` på det nested datasæt

Nu lad os køre vores funktion på den nested dataframe. Vi bruger `map()` til at lave funktionen `my_cor_test` for hvert af de fem datasæt. Det gøres ved at bruge funktionen `map()` indenfor funktionen `mutate()` til at oprette en ny kolonne, der hedder `test_stats`, hvor resultaterne for hver af de fem tests lagres.

```
eukaryotes_cor <- eukaryotes_nest %>%
  mutate(test_stats=map(data,my_cor_test))
```

```
eukaryotes_cor
```

```
## # A tibble: 5 x 3
## # Groups:   group [5]
##   group     data           test_stats
##   <chr>    <list>        <list>
## 1 Other    <tibble [51 x 19]>  <tibble [1 x 8]>
## 2 Protists <tibble [888 x 19]>  <tibble [1 x 8]>
## 3 Plants   <tibble [1,304 x 19]> <tibble [1 x 8]>
## 4 Fungi    <tibble [6,064 x 19]> <tibble [1 x 8]>
## 5 Animals  <tibble [3,201 x 19]> <tibble [1 x 8]>
```

#### Step 4: Anvende unnest() for at kunne se resultaterne

For at kunne se statistikerne bruger jeg funktionen `unnest()` på den nye variabel `test_stats`:

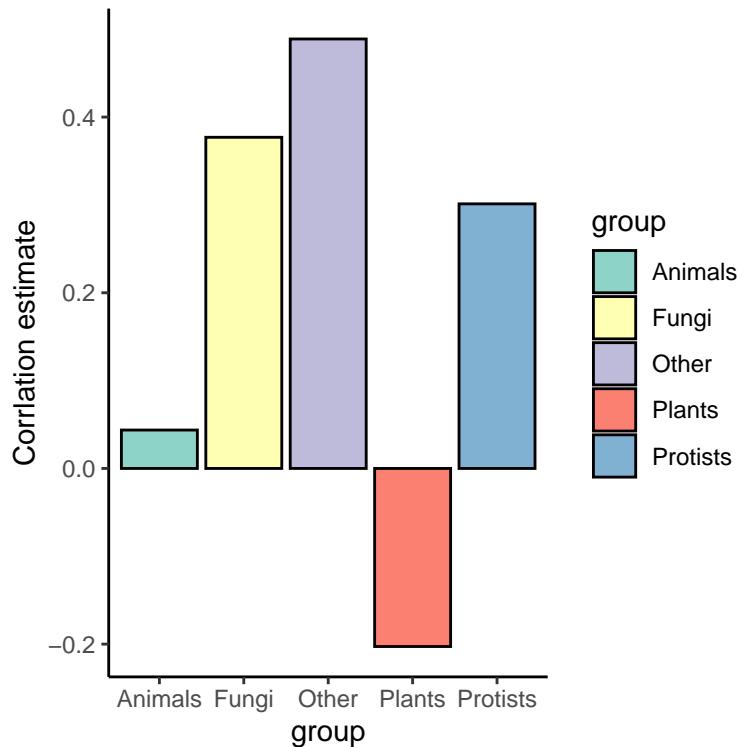
```
eukaryotes_cor <- eukaryotes_cor %>%
  unnest(test_stats)
eukaryotes_cor
```

```
## # A tibble: 5 x 10
## # Groups:   group [5]
##   group     data   estimate statistic  p.value parameter conf.low conf.high
##   <chr>    <tibble>    <dbl>    <dbl>    <dbl>    <int>    <dbl>    <dbl>
## 1 Other    <tibble>    0.489    3.80 4.22e- 4       46  0.238    0.679
## 2 Protists <tibble>    0.301    9.26 1.54e- 19      860  0.239    0.361
## 3 Plants   <tibble>   -0.203   -7.37 3.10e- 13     1267 -0.255   -0.149
## 4 Fungi    <tibble>    0.377    31.2 3.87e-198    5884  0.355    0.399
## 5 Animals  <tibble>    0.0437   2.42 1.57e- 2      3053  0.00825   0.0790
## # ... with 2 more variables: method <chr>, alternative <chr>
```

#### Step 5: Lave et plot fra statistikker

Vi kan bruge den direkte i et plot. Jeg fokuserer på den korrelaton koefficient i variablen `estimate` og omsætte den til et plot som i følgende:

```
cor_plot <- eukaryotes_cor %>%
  ggplot(aes(x=group,y=estimate,fill=group)) +
  geom_bar(stat="identity",colour="black") +
  scale_fill_brewer(palette = "Set3") +
  ylab("Correlation estimate") +
  theme_classic()
cor_plot
```



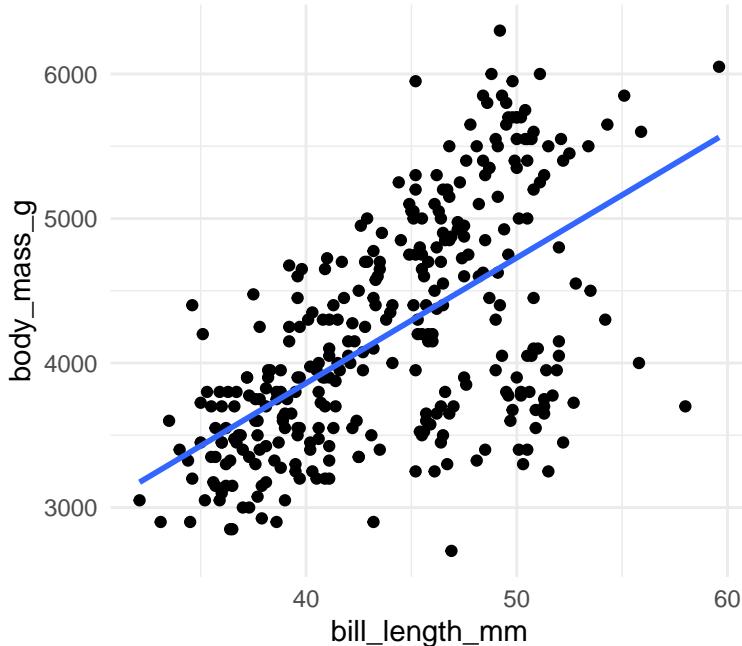
Bemærk at den overordnede process her med `cor.test` ligner processen hvis man anvender andre metoder såsom `t.test`, `lm` osv. Jeg gennemgår lidt om lineær regression og visualisering, og dernæst anvende processen på et eksempel med funktionen `lm()` og datasættet `penguins`.

## 8.3 Lineær regression - visualisering

### 8.3.1 Lineær trends

Vi skifter over til datasættet `penguins` som findes i pakken `palmerpenguins`. Man kan se i følgende scatter plot mellem `bill_length_mm` og `body_mass_g`, at der er plottet en bedste rette linje igennem punkterne, som viser, at der er en positiv sammenhæng mellem de to variabler.

```
## `geom_smooth()` using formula 'y ~ x'
```



Husk at den bedste rette linje har en formel  $y = a + bx$ , hvor  $a$  er den “intercept” (skæringspunktet) og  $b$  er den “slope” (hældningen) af linjen, og idéen med simpel lineær regression er, at man gerne vil finde de bedste mulige værdier for  $a$  og  $b$  for at plotte ovenstående linje således, at afstanden mellem linjen og punkterne bliver minimeret. Uden at gå i detaljer om hvordan det beregnes, husk at man bruger funktionen `lm()` som i følgende:

```
mylm <- lm(body_mass_g~bill_length_mm, data=penguins)
mylm
```

```
## 
## Call:
## lm(formula = body_mass_g ~ bill_length_mm, data = penguins)
## 
## Coefficients:
## (Intercept) bill_length_mm
##           388.85          86.79
```

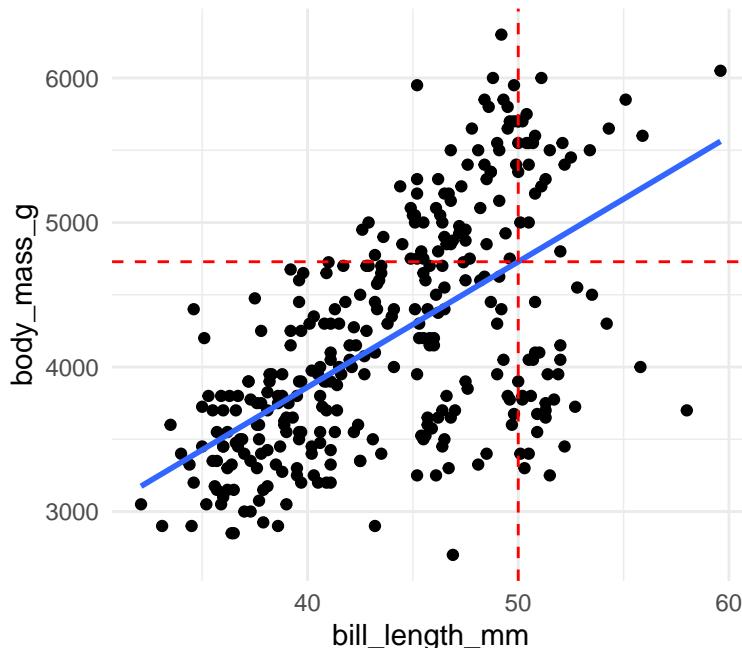
Intercept er således 388.85 og slope er 86.79 - det betyder, at hvis variablen `bill_length_mm` stiger ved 1, så ville den forventede `body_mass_g` stige ved 86.79. Man kan således bruge linjen til at lave forudsigelser. For eksempel, hvis jeg målet en ny pingvin og fandt ud af, at den havde en `bill_length_mm` af 50 mm, kunne jeg bruge min linje som den bedste gætte på dens `body_mass_g`:

```
y <- mylm$coefficients[1] + mylm$coefficients[2] * 50
y
```

```
## (Intercept)
##      4728.433
```

Jeg forventer derfor en pingvin med en bill længde af 50 mm til at have vægten omkring 4728.4331411 g:

```
## `geom_smooth()` using formula 'y ~ x'
```

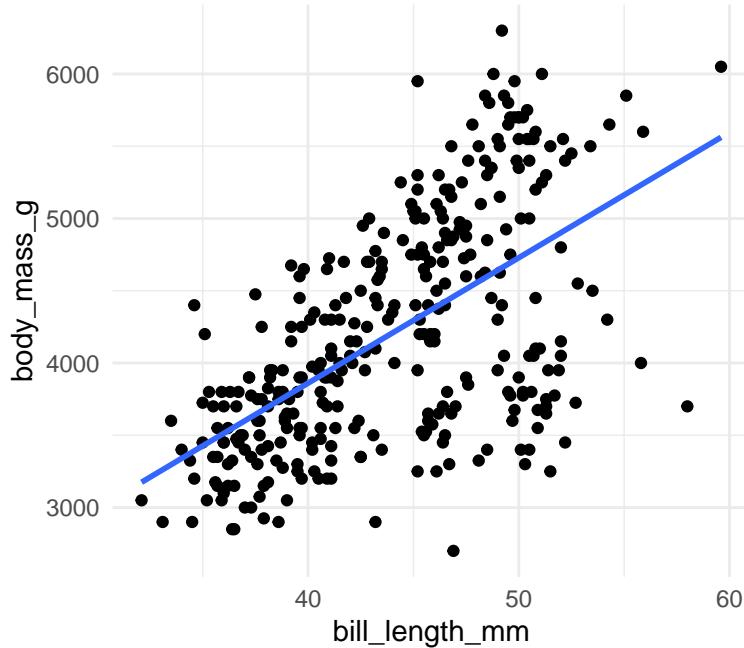


### 8.3.2 `geom_smooth()`: lm trendlinjer

Indbygget i ggplot2 er en funktion der hedder `geom_smooth()` som kan bruges til at tilføje den bedste rette linje til plottet. Man benytter den ved at specificere `+ geom_smooth(method="lm")` indenfor plot-kommandoen:

```
ggplot(penguins,aes(x=bill_length_mm,y=body_mass_g)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm",se=FALSE)
```

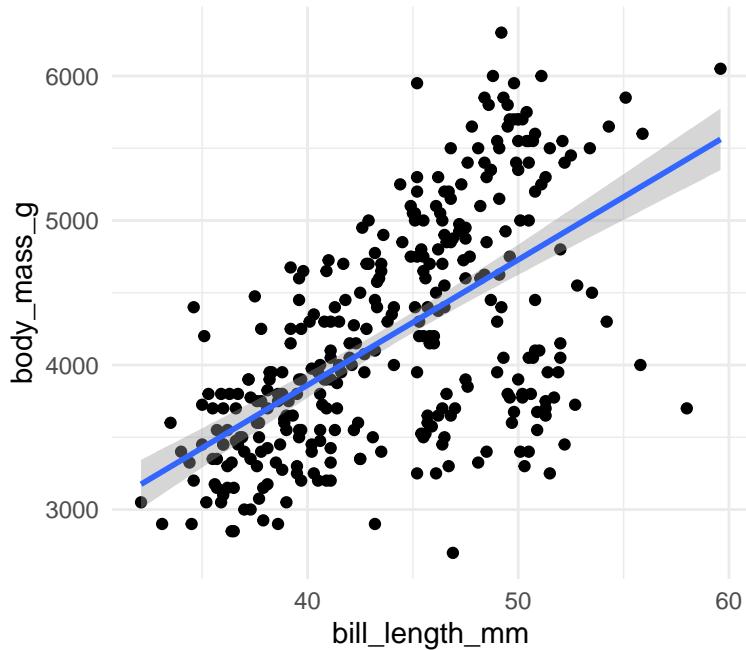
```
## `geom_smooth()` using formula 'y ~ x'
```



Det er nemt at bruge og at man kan få en konfidensinterval med, hvis man gerne vil have den: i ovenstående plot specificeret jeg `se=FALSE` men hvis jeg angiver `se=TRUE` (default), får jeg følgende plot:

```
ggplot(penguins,aes(x=bill_length_mm,y=body_mass_g)) +  
  geom_point() +  
  theme_minimal() +  
  geom_smooth(method="lm",se=TRUE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

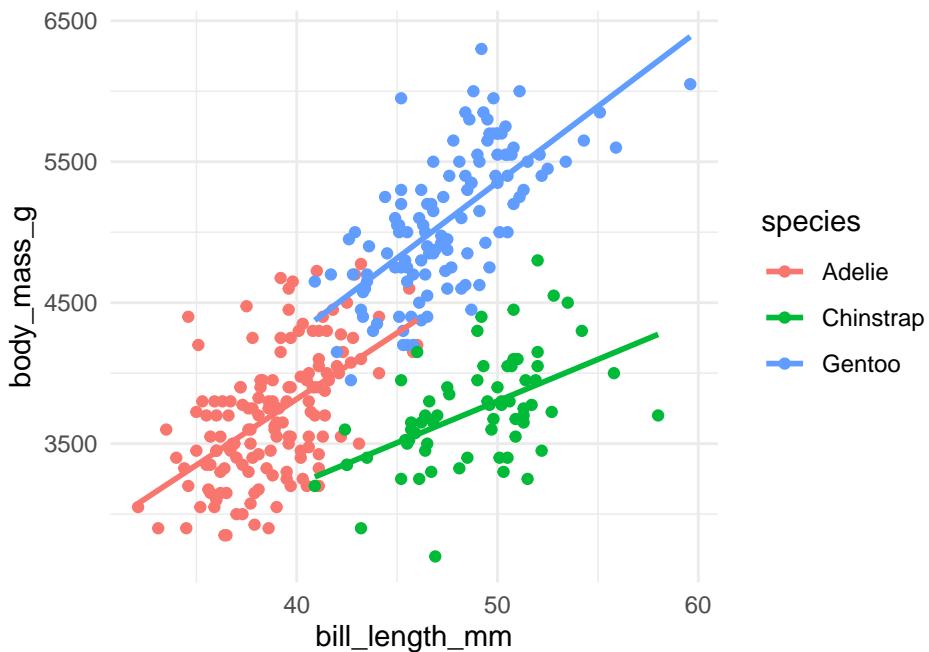


### 8.3.3 `geom_smooth()`: flere `lm` trendlinjer på samme plot

For at tilføje en bedste rette linje til de tre `species` hver for sig i stedet for samtlige data, er det meget nemt i `ggplot2`: man angiver bare `colour=species` indenfor aesthetics ('aes'):

```
ggplot(penguins,aes(x=bill_length_mm,y=body_mass_g,colour=species)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm",se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



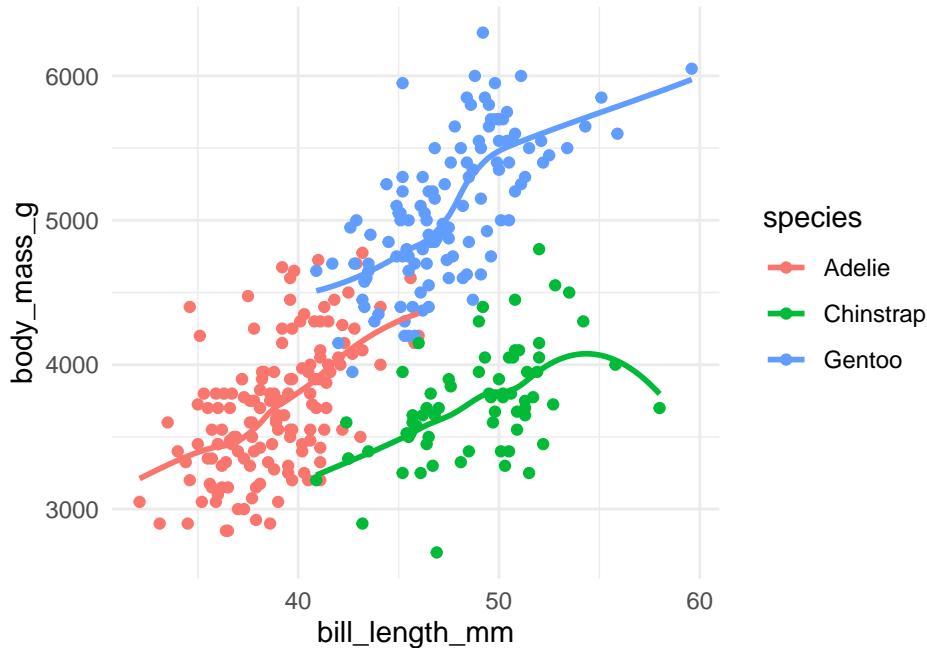
Så kan vi se, at der faktisk er tre forskellige trends her, så det giver god mening at bruge de tre forskellige linjer i stedet for kun en.

#### 8.3.4 Trendlinjer med `method=="loess"`

I `ggplot` er vi ikke begrænset til `method="lm"` indenfor `geom_smooth()`. Lad os afprøve i stedet `method="loess"`:

```
library(palmerpenguins)
penguins <- drop_na(penguins)
ggplot(penguins, aes(x=bill_length_mm, y=body_mass_g, colour=species)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="loess", se=FALSE)

## `geom_smooth()` using formula 'y ~ x'
```



Så kan man fange trends som ikke nødvendigvis er lineær - men bemærk at det er mere ligetil at beskrive og fortolke en lineær trend (og beregner udsigelser ud fra en lineær trend).

## 8.4 Plot linear regression estimates

For at finde vores estimates og tjekke signifikansen af en lineær trend, arbejder man direkte med den lineær model funktion `lm()`:

```
my_lm <- lm(body_mass_g~bill_length_mm, data=penguins)
summary(my_lm)
```

```
##
## Call:
## lm(formula = body_mass_g ~ bill_length_mm, data = penguins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1759.38  -468.82   27.79  464.20 1641.00
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 388.845   289.817   1.342   0.181
## bill_length_mm 86.792     6.538 13.276  <2e-16 ***
## ---
```

```
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 651.4 on 331 degrees of freedom
## Multiple R-squared: 0.3475, Adjusted R-squared: 0.3455
## F-statistic: 176.2 on 1 and 331 DF, p-value: < 2.2e-16
```

Husk de tal, som er vigtige her (se også emne 1 og 2):

- Den **p-værdi**: <2e-16 - uafhængige variablen `bill_length_mm` har en signifikant effekt/betydning for `body_mass_g`.
- Den **R-squared værdi**: - det viser den proportion af variancen i `body_mass_g` som `bill_length_mm` forklarer:
  - hvis R-squared er tæt på 1, så er der tæt på en perfekt korrespondens mellem `bill_length_mm` og `body_mass_g`.
  - hvis R-squared er tæt på 0, så er der nærmeste ingen korrespondens.

#### 8.4.1 Anvende `lm()` over nested datasæt

Vi kan benytte den samme proces som ovenpå i korrelation analysen. Vi bruge `group_by` til at opdele efter de tre `species` og så nest de tre datarammer:

```
penguins_nest <- penguins %>%
  group_by(species) %>%
  nest()
penguins_nest

## # A tibble: 3 x 2
## # Groups:   species [3]
##   species     data
##   <fct>    <list>
## 1 Adelie    <tibble [146 x 7]>
## 2 Gentoo    <tibble [119 x 7]>
## 3 Chinstrap <tibble [68 x 7]>
```

Jeg definerer en funktion hvor man lave lineær regression og tilføjer `glance()` til at få de model statistikker i en pån form.

```
#husk ~ og skriv .x for data og IKKE penguins
lm_model_func <- ~lm(body_mass_g~bill_length_mm,data=.x) %>% glance()
```

Vi kører en lineær model på hver af de tre datasæt med `map` og ved at specificere funktionen `lm_model_func` som vi definerede ovenpå. Vi bruger `mutate` ligesom før til at tilføje statistikkerne som en ny kolon der hedder `lm_stats`:

```
penguins_lm <- penguins_nest %>%
  mutate(lm_stats=map(data,lm_model_func))
penguins_lm

## # A tibble: 3 x 3
```

```
## # Groups:   species [3]
##   species    data           lm_stats
##   <fct>     <list>         <list>
## 1 Adelie    <tibble [146 x 7]> <tibble [1 x 12]>
## 2 Gentoo   <tibble [119 x 7]> <tibble [1 x 12]>
## 3 Chinstrap <tibble [68 x 7]>  <tibble [1 x 12]>
```

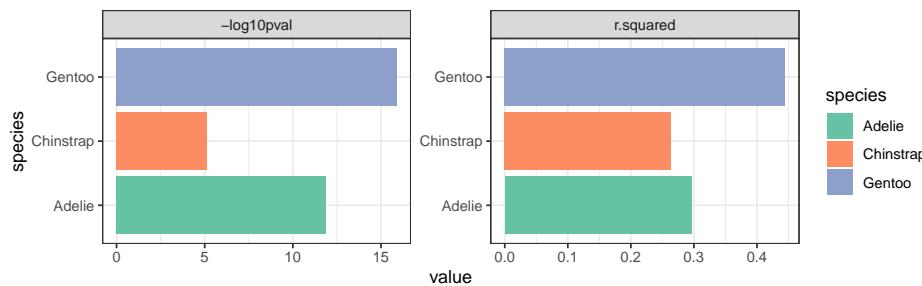
Til sidste bruger vi funktionen `unnest()` på vores statistikker:

```
penguins_lm <- penguins_lm %>%
  unnest(cols=lm_stats)
penguins_lm
```

```
## # A tibble: 3 x 14
## # Groups:   species [3]
##   species data      r.squared adj.r.squared sigma statistic  p.value    df logLik
##   <fct>   <tibble>    <dbl>        <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1 Adelie   <tibble>    0.296       0.291  386.    60.6 1.24e-12  1 -1076.
## 2 Gentoo   <tibble>    0.445       0.440  375.    93.6 1.26e-16  1 -873.
## 3 Chinstrap <tibble>    0.264       0.253  332.    23.7 7.48e- 6  1 -490.
## # ... with 5 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
## #   df.residual <int>, nobs <int>
```

Så kan vi se, at vi har fået en dataramme med vores lineær model statistikker. Jeg tager `r.squared` og `p.value` og omsætter dem til et plot for at sammenligne dem over de tre `species` af pingviner.

```
penguins_lm %>%
  select(species,r.squared,p.value) %>%
  mutate("-log10pval" = -log10(p.value)) %>%
  select(-p.value) %>%
  pivot_longer(-species) %>%
  ggplot(aes(x=species,y=value,fill=species)) +
  geom_bar(stat="identity") +
  scale_fill_brewer(palette = "Set2") +
  facet_wrap(~name,scale="free",ncol=4) +
  coord_flip() +
  theme_bw()
```



### 8.4.2 Funktion `glue()` for at tilføje labels

Det kan være nyttigt at tilføje nogle etiketter til vores plots med statistikkerne, vi lige har beregnet. Til at gøre det kan man bruge følgende kode. Vi tage vores datasæt `penguins_lm` med vores beregnet statistikker og bruge den til at lave en datasæt som kan bruges i `geom_text()` i vores trend plot. Funktionen `glue()` (fra pakken `glue`) er bare en nyttig måde at tilføj de `r.squared` og `p.value` værdier sammen i en “string”, som beskriver vores forskellige trends (lidt som `paste` fra base-R)

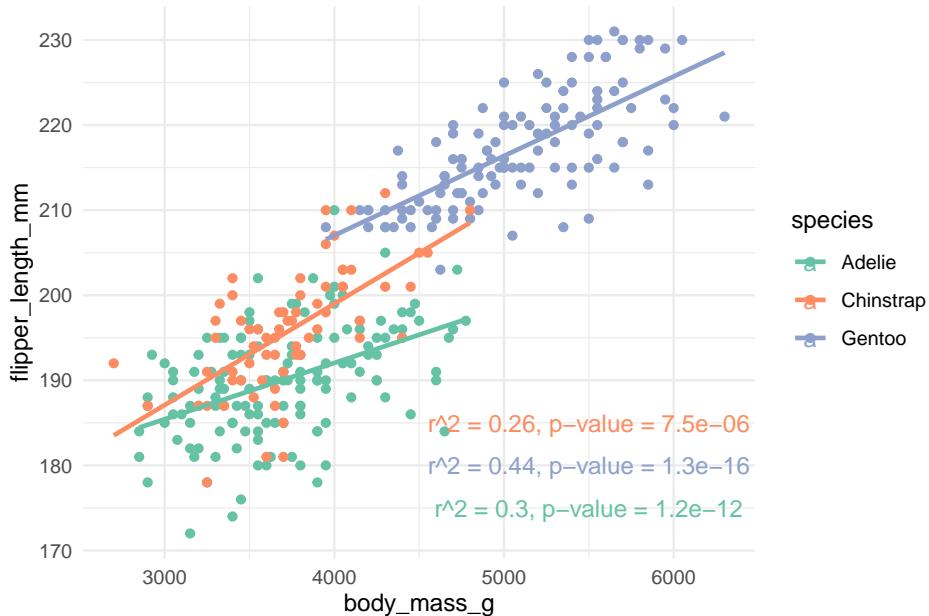
```
library(glue) # for putting the values together in a label
label_data <- penguins_lm %>%
  mutate(
    rsqr = signif(r.squared, 2), # round to 2 significant digits
    pval = signif(p.value, 2),
    label = glue("r^2 = {rsqr}, p-value = {pval}")
  ) %>%
  select(species, label)
label_data
```

FALSE # A tibble: 3 x 2  
 FALSE # Groups: species [3]  
 FALSE species label  
 FALSE <fct> <glue>  
 FALSE 1 Adelie r^2 = 0.3, p-value = 1.2e-12  
 FALSE 2 Gentoo r^2 = 0.44, p-value = 1.3e-16  
 FALSE 3 Chinstrap r^2 = 0.26, p-value = 7.5e-06

Vi kan tilføje vores label data indenfor `geom_text()`. `x` og `y` specificere hvor i plottet teksten skal være, og husk at specificere `data=label_data` og `label=label` skal stå indenfor `aes()` når det handler om en variable i `label_data`.

```
ggplot(penguins, aes(body_mass_g, flipper_length_mm, colour=species)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_text(
    x = 5500,
    y = c(175, 180, 185),
    data = label_data, aes(label = label), #specify label data from above
    size = 4
  ) +
  scale_color_brewer(palette = "Set2") +
  theme_minimal()

## `geom_smooth()` using formula 'y ~ x'
```



## 8.5 Multiple regression and model comparison

Man kan også bruge den samme ramme i ovenstående til at sammenligne forskellige modeller over samme tre datasæt - her definerer jeg `lm_model_func`, der har kun `sex` som uafhængige variabel og så bygger jeg op på den model ved at definere `lm_model_func2` og `lm_model_func3`, hvor jeg tilføjer ekstra uafhængige variabler `bill_length_mm` og `flipper_length_mm`. Jeg er interesseret i, hvor meget af variansen i `body_mass_g` de tre variabler forklarer tilsammen, og om der er forskelligheder efter de tre arter i `species`.

```
lm_model_func <- ~lm(body_mass_g ~ sex ,data=.x)
lm_model_func2 <- ~lm(body_mass_g ~ sex + bill_length_mm ,data=.x)
lm_model_func3 <- ~lm(body_mass_g ~ sex + bill_length_mm + flipper_length_mm ,data=.x)
```

Bemærk at jeg endnu ikke har tilføjet `glance()` her men jeg har tænkt mig at gøre det lidt senere i processen for at undgå, at jeg får alt for mange statistikker i min dataframe med mine resultater. Jeg anvender først `group_by()` efter `species` og `nest()`:

```
penguins_nest <- penguins %>%
  group_by(species) %>%
  nest()
penguins_nest

## # A tibble: 3 x 2
## # Groups:   species [3]
```

```
##   species   data
##   <fct>     <list>
## 1 Adelie    <tibble [146 x 7]>
## 2 Gentoo    <tibble [119 x 7]>
## 3 Chinstrap <tibble [68 x 7]>
```

Her bruger jeg map tre gange indenfor sammen `mutate`, for at bygge de tre modeller for hver art (ni modeller i alt).

```
penguins_nest_lm <- penguins_nest %>%
  mutate(
    model_sex = map(data, lm_model_func),
    model_sex_bill = map(data, lm_model_func2),
    model_sex_bill_flipper = map(data, lm_model_func3))
penguins_nest_lm

## # A tibble: 3 x 5
## # Groups:   species [3]
##   species   data      model_sex model_sex_bill model_sex_bill_flipper
##   <fct>     <list>    <list>    <list>    <list>
## 1 Adelie    <tibble [146 x 7]> <lm>      <lm>      <lm>
## 2 Gentoo    <tibble [119 x 7]> <lm>      <lm>      <lm>
## 3 Chinstrap <tibble [68 x 7]>  <lm>      <lm>      <lm>
```

Nu vil jeg gerne udtrækker nogle statistikkere fra modellerne så jeg kan sammenligne dem. Jeg vil gerne lave samme process på alle ni modeller - hvor jeg benytter funktionen `glance` til at få outputtet i tidy-form, og så udtrækker `r.squared` bagefter til at undgå, at der kommer alt for mange statistikker i mine nye dataframe.

```
get_r2_func <- ~.x %>% glance() %>% pull(r.squared)
```

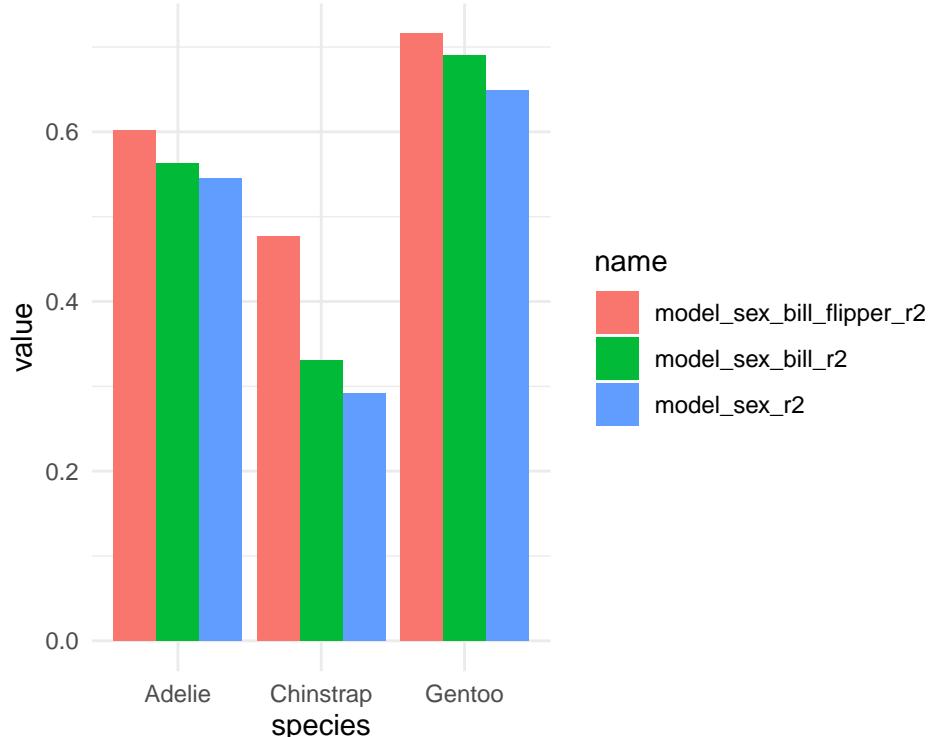
Nu gælder det om at køre ovenstående funktion på alle mine modeller, som er lagret i tre kolonner, `model_sex`, `model_sex_bill` og `model_sex_bill_flipper`. Jeg gøre det indenfor `map` så det bliver også kørt til hver af de tre arter.

```
penguins_nest_lm <- penguins_nest_lm %>%
  mutate(model_sex_r2 = map_dbl(model_sex, get_r2_func),
        model_sex_bill_r2 = map_dbl(model_sex_bill, get_r2_func),
        model_sex_bill_flipper_r2 = map_dbl(model_sex_bill_flipper, get_r2_func))
penguins_nest_lm %>% select(species, model_sex_r2, model_sex_bill_r2, model_sex_bill_flipper_r2)

## # A tibble: 3 x 4
## # Groups:   species [3]
##   species   model_sex_r2 model_sex_bill_r2 model_sex_bill_flipper_r2
##   <fct>       <dbl>          <dbl>            <dbl>
## 1 Adelie      0.545         0.563           0.602
## 2 Gentoo      0.649         0.691           0.716
## 3 Chinstrap   0.291         0.331           0.476
```

Omsætte til et plot:

```
penguins_nest_lm %>%
  pivot_longer(cols=c("model_sex_r2","model_sex_bill_r2","model_sex_bill_flipper_r2"))
  ggplot(aes(x=species,y=value,fill=name)) +
  geom_bar(stat="identity",position="dodge") +
  theme_minimal()
```



Man kan se i plottet, at `body_mass_g` i `species` "Gentoo" er bedste forklaret af de tre varibler, og den lavest `r.squared` er tilfælde hvor variablen `sex` er dene eneste uafhængig variable og `species` er "Chinstrap".

### 8.5.1 anova for at sammenligne de forskellige modeller

Grunden til, at jeg valgt at bruge `glance()` i en ny funktion for at udtrække `r.squared` værdier, var fordi jeg gerne ville bevare mine modeller i rå form, så de kan bruges indenfor `anova()`. Med `anova()` kan jeg sammenligne to modeller direkte og får således en p-værdi hvor man teste hypotesen, hvor den ekstra variabler i den ene model forklarer den afhængig variabel signifikant (når man tager højde for de variabler, der er fælles til både modeller).

I følgende skriver jeg en funktion hvor jeg kan sammenligne to modeller med `anova` og udtrækker p-værdien:

```
aov_func <- ~anova(.x,.y) %>% tidy() %>% pluck("p.value",2)
```

- ~ fordi det er en funktion (som jeg benytter for hver art og model sammenligning - 9 gange i alt!)
- `anova` for at sammenligne modellerne som er betegnet ved `.x` og `.y` (vi anvender `map2` som tager to input i stedet for én som i `map`)
- `tidy()` er ligesom `glance` men angiver sumamry statistikker og flere linjer - herunder p-værdien
- `pluck` - jeg vil have kun én statistik ("p.value") - og det er lagret i anden plads.

Se følgende kode for når man anvende `anova` og `tidy` på modellerne `model_sex` og `model_sex_bill` i species "Adelie" (da jeg benyttet `pluck` med "1" som betyder den først plads i listen):

```
myaov <- anova(penguins_nest_lm %>% pluck("model_sex",1),
                 penguins_nest_lm %>% pluck("model_sex_bill",1))
myaov %>% tidy #p.value for comparing the two models is in the second position
```

```
## # A tibble: 2 x 6
##   res.df      rss     df    sumsq statistic p.value
##   <dbl>      <dbl> <dbl>    <dbl>    <dbl>    <dbl>
## 1     144 13884760.     NA     NA      NA      NA
## 2     143 13332955.     1 551805.     5.92  0.0162
```

Man kan se, at p-værdien er 0.016 som er signifikant og betyder at den mere 'indviklet' model der også inddrager `bill_length_mm` er den model, vi accepterer (dvs. effekten af variablen `bill_length_mm` på `body_mass_g` er signifikant så vores 'final' m).

Man kan lave en lignende sammenligning mellem samtlige par modeller over de tre arter:

```
penguins_nest_lm <- penguins_nest_lm %>%
  mutate(model_sex_vs_model_sex_bill = map2_dbl(model_sex,model_sex_bill,aov_func),
         model_sex_vs_model_sex_bill_flipper = map2_dbl(model_sex,model_sex_bill_flipper,aov_func),
         model_sex_bill_vs_model_sex_bill_flipper = map2_dbl(model_sex_bill,model_sex_bill_flipper,aov_func))
penguins_nest_lm %>% select(species,model_sex_vs_model_sex_bill,model_sex_vs_model_sex_bill_flipper)

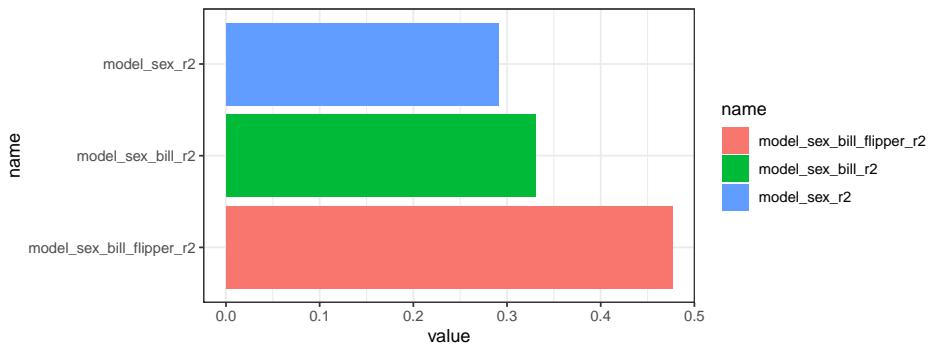
## # A tibble: 3 x 4
## # Groups:   species [3]
##   species  model_sex_vs_model_sex_bill model_sex_vs_model_sex_bill_flipper model_sex_bill_vs_
##   <fct>                <dbl>                  <dbl>                  <dbl>
## 1 Adelie        0.0162        0.0000730        0.000273
## 2 Gentoo       0.000142        0.00000592        0.00193
## 3 Chinstrap    0.0540        0.0000621        0.0000795
```

Det kunne være nyttigt at inddrage p-værdier i ovenstående plot med `r.squared` værdierne, til at se om, der er en signifikant effekt når man tilføjer flere variabler

til modellen samtidig at `r.squared` stiger. I følgende omsætter jeg `r.squared` statistikker til kun “Chunstrap” i et plot:

```
library(ggsignif)

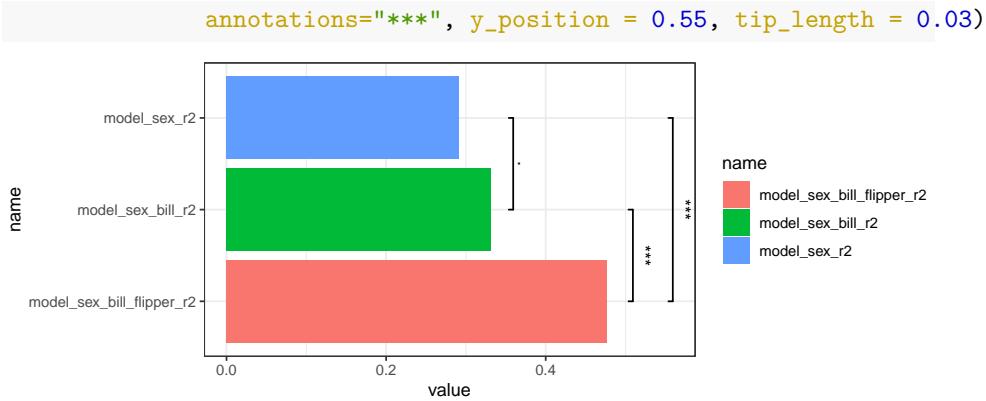
stats_plot <- penguins_nest_lm %>%
  filter(species=="Chinstrap") %>%
  pivot_longer(cols=c("model_sex_r2", "model_sex_bill_r2", "model_sex_bill_flipper_r2")) +
  ggplot(aes(x=name, y=value, fill=name)) +
  geom_bar(stat="identity", position="dodge") +
  coord_flip() +
  theme_bw()
stats_plot
```



I følgende tilføjer jeg funktionen `geom_signif` til plottet - det tillader, at jeg kan tilføje signifikans linjer/annotations til plottet - dvs. viser hvilke to modeller jeg sammenligner, og angiver stjerne efter beregnede p-værdierne. Du er velkommen til at kopier mine kode og tilpasse til egen behov.

- Når jeg sammenligner modellerne “model\_sex” og “model\_sex\_bill” i “Chinstrap”, er p-værdien over 0.05, så tilføjelsen af `bill_length_mm` i modellen var ikke signifikant - jeg giver ingen stjerner men skriver “?” til at matcher outputtet i `lm`.
- Når jeg sammenligner modellerne “model\_sex” og “model\_sex\_bill\_flipper” kan jeg se at p-værdien er under 0.05, så der er en signifikant effekt - `bill_length_mm` og `flipper_length_mm` forklarer den afhængige variabel `body_mass_g`, ud over variablen `sex`. Jeg angiver “\*\*\*” fordi p-værdien er under 0.001 (Se signif. codes i `lm` summary).
- Indstillingen `y_position` fortæller hvor jeg vil placerer linjerne.

```
stats_plot +
  geom_signif(comparisons = list(c("model_sex_r2", "model_sex_bill_r2")),
              annotations=". ", y_position = 0.35, tip_length = 0.03) +
  geom_signif(comparisons = list(c("model_sex_bill_r2", "model_sex_bill_flipper_r2")),
              annotations="***", y_position = 0.5, tip_length = 0.03) +
  geom_signif(comparisons = list(c("model_sex_r2", "model_sex_bill_flipper_r2")),
              annotations="?", y_position = 0.15, tip_length = 0.03)
```



## 8.6 Problemstillinger

**Problem 1)** Quizzen på Absalon.

---

Husk at have indlæste følgende:

```
library(tidyverse)
library(broom)
data(msleep)
data(iris)
```

**Problem 2) Korrelation øvelse**

- Brug `data(mtcars)` og `cor.test()` til at lave et test af korrelationen mellem variablerne `qsec` og `drat`.
  - Tip: hvis du foretrækker at undgå \$ i analysen til at specifie en kolon indenfor `cor.test()` kan du bruge `mtcars %>% pull(qsec)` i stedet for `mtcars$qsec`.
  - Tilføj funktionen `glance()` til din resultat fra `cor.test()` til at se de statistikker i `tidy` form (installer pakken `broom` hvis nødvendigt). Kan du genkende de statistikker fra `cor.test()` i den resulterende dataramme?
- 

**Problem 3) Nesting øvelse**

For datasættet `msleep`, anvende `group_by()` og `nest()` til at få en nested dataframe hvor datasættet er opdelt efter variablen `vore`. Kalder det for `msleep_nest`.

- Tilføj en kolon til `msleep_nest` med `mutate`, der hedder `n_rows` og viser antallet af rækker i hvert af de fire datasæt - husk følgende struktur:

```
msleep_nest %>%
  mutate("n_rows" = map(???, ???)) #erstatte ??? her
```

- I dette tilfælde kan man ændre `map` til `map dbl` - gør det.

#### Problem 4) Multiple korrelation

Vi vil gerne beregne den korrelation mellem variablerne `sleep_total` og `sleep_rm` til hver af de fire datasæt lagret i `msleep_nest`

- Tilpas følgende funktion så at vi teste korrelation mellem de to variabler.
- Tilføj `glance()` så at vi få vores data i `tidy` form.

```
cor_test <- ~cor.test(????, ???) #erstatte ??? og tilføj glance funktion
```

- Brug `map()` indenfor `mutate()` med din funktion for at beregne de korrelation statistikker til hver af de fire datasæt.
- Unnest din nye kolonne bagefter
- Lav barplots af `estimate` og `-log10(p.value)` med den resulterende dataramme
- Prøv også at tilføj `%>% pluck("estimate", 1)` til din `cor_test` funktion og kig på resultat

#### Problem 5) Linear regression øvelse

Åbn LungCapData (herunder `Age.Groups`):

```
LungCapData <- read.csv("https://www.dropbox.com/s/ke27fs5d37ks1hm/LungCapData.csv?dl=1")
glimpse(LungCapData) #se variabler navne
```

```
## Rows: 725
## Columns: 6
## $ LungCap    <dbl> 6.475, 10.125, 9.550, 11.125, 4.800, 6.225, 4.950, 7.325, 8.~
## $ Age        <int> 6, 18, 16, 14, 5, 11, 8, 11, 15, 11, 19, 17, 12, 10, 10, 13, ~
## $ Height     <dbl> 62.1, 74.7, 69.7, 71.0, 56.9, 58.7, 63.3, 70.4, 70.5, 59.2, ~
## $ Smoke      <chr> "no", "yes", "no", "no", "no", "no", "no", "no", "no", "no", ~
## $ Gender     <chr> "male", "female", "female", "male", "male", "female", "male", "ma~
## $ Caesarean <chr> "no", "no", "yes", "no", "no", "yes", "no", "no", "no", "no"~
```

- Anvende `lm()` med `LungCap` som afhængig variabel og `Age` som uafhængig variabel.
- Hvad er den intercept og slope af den beregnede linje?
- Prøv at tilføj funktionen `glance()` til din `lm` funktion og angive værdier `r.squared` og `p.value`.

---

**Problem 6)** Lave et scatter plot af Age på x-aksen og LungCap på y-aksen.

- Ændre linjen til `geom_smooth(method="lm")`
  - Ændre linjen til `geom_smooth(method="lm", se=FALSE)`
  - Nu specifiser en forskellige farve efter `Gender`. Hvordan er de to linje forskellige?
  - Nu specifiser en forskellige farve efter `Smoke`. Hvordan er de to linje forskellige?
- 

**Problem 7) Lineær regression øvelse over multiple datasæt**

Vi vil gerne udføre lineær regression med LungCap og Age men opdeler efter variablen `Smoke`. OBS: Vi følger sammen process som i kursus notaterne men med `LungCapData` i stedet for `Penguins` - tjek gerne kursusnotaterne for inspiration.

- a) Anvende `group_by()` og `nest()` for at opdele dit datsasæt efter `Smoke`
  - b) Lav en funktion, `lm_model_func`, som beregner en lineær regression med `LungCap` som afhængig variabel og `Age` som uafhængig variabel. Tilføj `glance()` til `lm_model_func`.
  - c) Anvende `map()` med din funktion indenfor `mutate()` til at tilføje en ny kolon som hedder `lm_stats` til din dataramme. Husk at `unnest` kolonnen `lm_stats` for at kunne se statistikker.
  - d) Fortolkning - er variablen `lungCap` bedre forklaret er variablen `Age` i rygere eller ikke-rygere?
- 

**Problem 8)**

I nedenstående er tre modeller, alle med `LungCap` som afhængig variabel alle som tager højde for `Age`:

```
my_lm_func1 <- ~lm(LungCap ~ Age, data=.x)
my_lm_func2 <- ~lm(LungCap ~ Age + Gender, data=.x)
my_lm_func3 <- ~lm(LungCap ~ Age + Gender + Height, data=.x)
```

- a) Anvend `map` til at lave tre nye kolonner i `LungCapData_nest`, en til hver af de tre modeller (uden `glance()` her, så vi kan brug vores `lm` objekts senere).
- b) Skriv en funktion `my_r2_func`, der udtrækker "r.squared" værdierne fra dine modeller (her refererer `.x` i funktionen ikke til en dataframe men til en beregnet model - hvad er det, der skal tilføjes?). Lav tre yderligere kolonner i `LungCapData_nest`, hvor du køre din funktion på dine modeller med `map` (outputtet skal være `dbl`).

```
my_r2_func <- ...

LungCapData_nest <- LungCapData_nest %>%
  mutate("Age_only_R2" = ...,
        "Age_Gender_R2" = ...,
        "Age_Gender_Height_R2"= ...)
```

- c) Omsæt dine beregnede r.squared værdier til et plot
- 

### Problem 9

- a) Skriv en funktion hvor man anvende `anova()` til at sammenligne to modeller, `.x` og `.y` og dernæst udtrækker p-værdien (det er den samme funktion som i kursusnotaterne).

```
my_aov_func <- ...
```

- b) Anvend din funktion med `map2` til at sammenligne de tre modeller fra sidste spørgsmål.

- c) Lav et plot med dine resultater.

- d) Tilføj signifikans annotations på plottet med funktionen `geom_signif()` (tilpas gerne kode fra kursusnotaterne).

## 8.7 Ekstra

<https://www.tidymodels.org/learn/statistics/tidy-analysis/>

# Chapter 9

## Clustering

### 9.1 Indledning og læringsmålene

#### 9.1.1 Læringsmålene

Du skal være i stand til at

- Beskrive hvad k-means clustering går ud på
- Anvende `kmeans` og output resultatet på en `tidy` måde
- Anvende `map` over forskellige antal clusters og vælge antallet som passer til de data
- Anvende funktionen `hclust` for at lave et simpel hierarchical clustering

#### 9.1.2 Inledning til chapter

I clustering er det til formål at dele observationerne i et datasæt op i forskellige grupper (clusters eller klynge på dansk), således at observationerne i sammen cluster ligner hinanden. Det øger indsigten i datasættet ved at fk. bedre forstår strukturen. Fk. hvor mange forskellige clusters er repræsenteret i mit datasæt? Og hvilke individuelle observationer tilhører hvilken cluster?

I dette kapitel ser vi hvordan vi kan implementere både **k-means clustering** og **hierarchical clustering** indenfor den tidyverse ramme.

#### 9.1.3 Video ressourcer

- Video 1: K-means clustering

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/553656150>

- Video 2: augment, glanced og tidy med K-means. OBS der er en lille fejl i koden omkring 6:00 - den anden `geom_point` skal være `geom_point(data = kclust_tidy, aes(x=bill_length,y=bill_depth),shape="x",colour="black")` fordi tallerne er allerede baserede på “scaled” data i `kclust_tidy` - se sektion 9.2.5 for uddybelse.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/553656139>

---

- Video 3: Hvor mange clusters skal man vælge?

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/553656129>

---

- Video 4: Hierarchical clustering

(OBS Video 4 mangler: se gerne kursusnotaterne og jeg laver videoen ASAP)

---

## 9.2 Method 1: K-means clustering

```
library(palmerpenguins)
library(tidyverse)
library(broom)
```

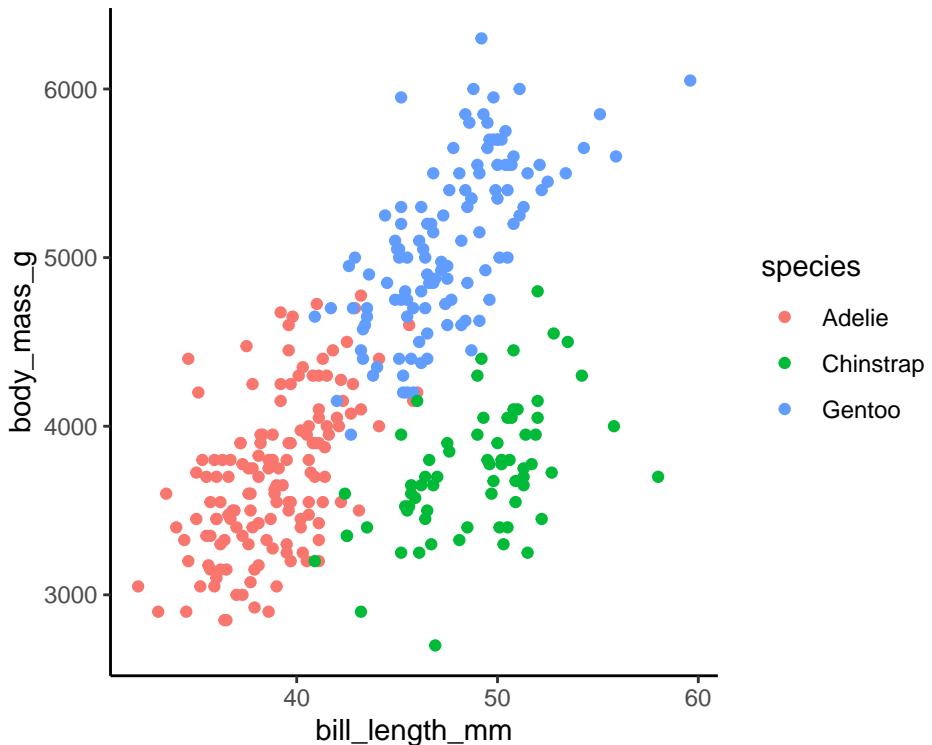
I k-means clustering bliver samtlige observationer tilknyttet den nærmeste cluster “centroid” (se “hvordan fungerer kmeans?” nedenunder). I k-means er man nødt til at specificere antallet af clusters, som observationerne skal være delt op i, i forvejen. Derfor skal der være nogle undersøgelsesarbejde for at vælge den bedste antal clusters som passer til problemstillingen, eller som bedste repræsentanter datasættet.

Lad os tage udgangspunkt i datasættet `penguins`. Vi begynder med at få fjernet observationerne med `NA` i mindst én variable med funktionen `drop_na` og ved at specificere at `year` skal være en faktor (for at skelne den fra de andre numeriske kolonner):

```
data(penguins)
penguins <- penguins %>%
  mutate(year=as.factor(year)) %>%
  drop_na()
```

Vi vide allerede i forvejen, at der er 3 `species` med i de data, som vi plotter her med forskellige farver.

```
penguins %>% ggplot(aes(x=bill_length_mm, y=body_mass_g, colour=species)) +
  geom_point() +
  theme_classic()
```



Vi vil gerne bruge k-means clustering på de numeriske variabler i datasættet, og beregne 3 clusters ud fra dem. Derefter kan det være interessant at sammenligne de clusters vi få med de tre arter af pingvin - hvor gode er de clusters til at skelne i mellem de forskellige species, eller fanger de noget anden struktur i datasættet (for eksempel kønnet eller øen, de bor på)?

### 9.2.1 Hvordan fungere kmeans?

K-means er en iterativ process. Lad os forestille os at vi gerne vil have tre clusters i de data. Man starter med tre tilfældige observationer og kalder dem for de cluster middelværdier eller "centroids". Man tilknytter alle observationer til én af de tre clusters (efter den nærmeste af de tre centroids), og så beregner en ny middelværdi/centroid for at hver cluster. Man tilknytter samtlige observationer igen efter den nærmeste af de tre nye cluster centroids, og så gentager man processen flere gange. Efter flere gange konvergere de tre centroids til nogle faste værdier, der ikke længere ændre sig meget hver gang med gentager processen. Disse tre centroids defininere de tre endelige clusters og samtlige observationer er tilknyttet én af de tre.

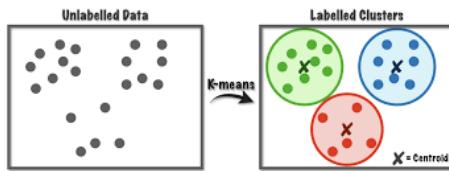


Figure 9.1: source: <https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c>

Jeg spørger ikke efter detaljerne i metoden men der er mange videoer på Youtube som bedre foreklarer hvordan k-means fungerer, for eksempel: <https://www.youtube.com/watch?v=4b5d3muPQmA>

Bemærk, at der er noget **tilfældighed** indbygget i algoritmen. Det betyder, at hver gang man anvende k-means, få man en lidt anderledes resultat.

### 9.2.2 Within/between sum of squares

Man kan forestille sig, at hvis man lave en gode clustering af datasæt, så ligner observationerne indenfor den sammen cluster hinanden meget, og til gengæld er observationerne i forskellige clusters meget forskellige fra hinanden. Med andre ord, er afstanden mellem observationerne i samme cluster så mindre som muligt og afstanden mellem observationerne i forskellige clusters er så stor som muligt. For at måle det kan man beregne følgende:

- **total within sum of squares** - den totale squared afstand af observationerne fra deres nærmeste centroid.
- **total between sum of squares** - den totale afstand af centroids til samtlige andre centroids. Det skal være så stor som muligt.

### 9.2.3 Run k-means i R

K-means *fungerer kun på numeriske data*, som vi kan vælge fra datasættet med `select()` sammen med hjælper `where(is.numeric)`. Vi bruger også `scale()`, som betyder, at alle variabler få den samme skala og det undgår, at der er nogle som få mere indflydelse end andre i de færdige resultater.

```
penguins_scaled <- penguins %>%
  select(where(is.numeric)) %>%
  scale()
```

Man er også nødt til at fortælle i forvejen hvor mange clusters at opdele datasættet i, så lad os sige `centers=3` indenfor funktionen `kmeans()` her og beregner vores clusters:

```
kclust <- kmeans(penguins_scaled, centers = 3)
kclust
```

Man få forskellige ting frem, for eksempel:

- **Cluster means** - det svarer til de centroids markerede med  $\mathbf{x}$  i den ovenstående figur - bemærk at her er de 4-dimensionelle da vi brugt 4 variabler til at beregne resultatet.
  - **Clustering vector** - hvilke cluster er hver observation blevet tilknyttet.
  - **Within cluster sum of squares** - Jo mindre, jo bedre - hvor meget observationerne indenfor samme cluster ligner hinanden (den totale squared afstand af observationerne fra deres nærmeste centroid).

## 9.2.4 Tidy up k-means resultaterne med pakken broom

Fra pakken **broom** har vi mest beskæftiget os med **glance()** indstil videre. Med **glance()** få man enkel-linje baserede summary statistikker fra én eller flere modeller sammen i én dataramme, for at facilitete et plot/labels osv. Der er også to andre funktioner vi tager i bruge her. Her er en beskrivelse af de tre.

---

| Broom verb | Beskrivelse   |
|------------|---|
| glance()   | single line summary - make elbow plot                       |
| augment()  | Append dataset to clusters - make plots coloured by cluster |
| tidy()     | Multi-line summary - extract centroids                      |

---

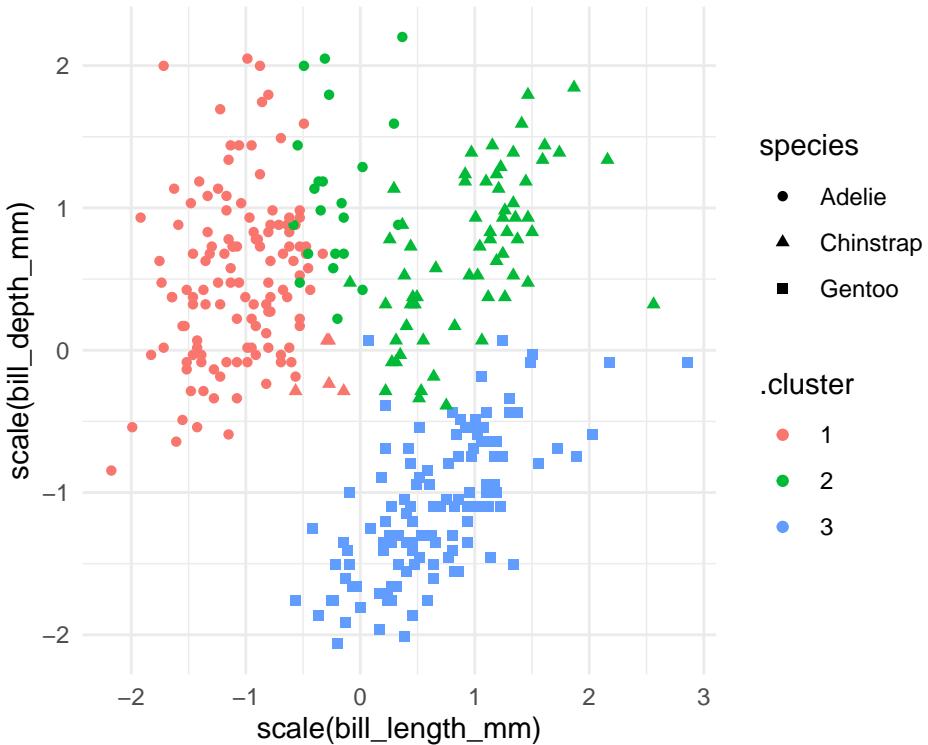
For at lave et plot af de clusters kan det især være nyttigt at benytte `augment`. Her kan man se, at vi har fået en kolon der hedder `.cluster` med i den oprindelige dataramme (jeg flyttet kolonen til første plads i følgende kode så man kan se den i de output af kursusnotater).

```
kc1 <- augment(kclust, penguins) #clustering = første plads, data = anden plads
kc1 %>% select(.cluster, all_of(names(penguins)))
```

```
## # A tibble: 333 x 9
##   .cluster species island   bill_length_mm bill_depth_mm flipper_length_mm
##   <fct>    <fct>   <fct>        <dbl>        <dbl>            <int>
## 1 1       Adelie  Torgersen     39.1         18.7          181
## 2 1       Adelie  Torgersen     39.5         17.4          186
## 3 1       Adelie  Torgersen     40.3         18             195
## 4 1       Adelie  Torgersen     36.7         19.3          193
## 5 1       Adelie  Torgersen     39.3         20.6          190
## 6 1       Adelie  Torgersen     38.9         17.8          181
## 7 1       Adelie  Torgersen     39.2         19.6          195
## 8 1       Adelie  Torgersen     41.1         17.6          182
## 9 1       Adelie  Torgersen     38.6         21.2          191
## 10 1      Adelie  Torgersen     34.6         21.1          198
## # ... with 323 more rows, and 3 more variables: body_mass_g <int>, sex <fct>,
## #   year <fct>
```

Nu benytter vi `kc1` til at lave et plot. Her giver jeg en farve efter `.cluster` og shape efter `species` så at vi kan sammenligne vores beregnet clusters med de tre forskellige arter. Bemærk her, at jeg kun har to variabler i plottet, men der er faktisk fire variabler som blev brugt til at lave de clusters i med funktionen `kmeans`. En anden måde er at plotte de først to principal components i stedet for to af de fire variabler - det beskæftige vi os med næste gang.

```
ggplot(kc1, aes(x = scale(bill_length_mm),
                 y = scale(bill_depth_mm))) +
  geom_point(aes(color = .cluster, shape = species)) + theme_minimal()
```



Vi kan også fl. optælle hvor mange af de tre arter vi får i hver af vores tre clusters, hvor vi kan se, at Adelie og Chinstrap er blevet mere blandet blandt to af de tre clusters end Gentoo.

```
kc1 %>% count(.cluster, species)
```

```
## # A tibble: 5 x 3
##   .cluster species     n
##   <fct>    <fct>    <int>
## 1 1        Adelie    124
## 2 1        Chinstrap  5
## 3 2        Adelie    22
## 4 2        Chinstrap 63
## 5 3        Gentoo   119
```

### 9.2.5 Plot cluster centroids

Næste kigger vi på resultatet af funktionen `tidy()` fra `broom`-pakken. Her har vi fået en pæn dataramme med middelværdierne (centroids) af de tre clusters over de fire variabler som var brugt i beregningerne.

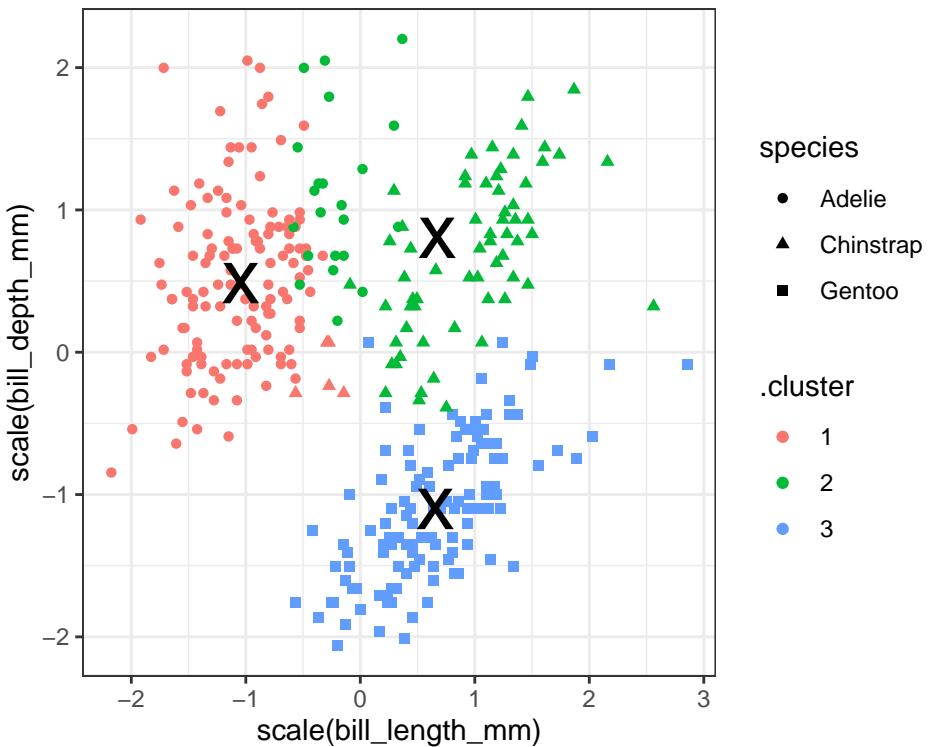
```
kclust_tidy <- kclust %>% tidy()
kclust_tidy
```

```
## # A tibble: 3 x 7
##   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g size withinss
##       <dbl>          <dbl>          <dbl>      <dbl> <int>     <dbl>
## 1      -1.05         0.486        -0.880     -0.762    129     121.
## 2       0.671         0.804        -0.289     -0.384     85      109.
## 3       0.654        -1.10         1.16      1.10      119     139.
## # ... with 1 more variable: cluster <fct>
```

I følgende benytter jeg `kclust_tidy` som et ekstra datasæt i ovenstående plot, men indenfor en anden `geom_point()` for at tilføje en `x` form på midten af de tre clusters - se følgende tre punkter, der forklarer nogle detaljer i koden:

- Jeg bruger funktionen `scale()` på `bill_length_mm` og `bill_depth_mm`, fordi min centroids, som også skal med i plottet, var beregnet på skaleret data.
- Jeg behøver ikke at anvende `scale()` på min centroids lagrede i `kclust_tidy` så jeg angiver bare akser-variablene i `aes()` uden at anvende `scale()`.
- Jeg har brugt `color` og `shape` som lokale aesthetics i den første `geom_point()` her, der de ikke eksistere som kolonner i `kclust_tidy`.

```
ggplot(kc1, aes(x = scale(bill_length_mm), #need to scale the original data
                 y = scale(bill_depth_mm))) +
  geom_point(aes(color = .cluster, shape = species)) +
  geom_point(data = kclust_tidy,
             aes(x = bill_length_mm, #don't need to scale again
                 y = bill_depth_mm),
             size = 10, shape = "x", show.legend = FALSE) +
  theme_bw()
```



Vi kan se at vores clusters ikke fanger de samme tre gruppe som variablen **species** præcist - der er forskelligheder. Det kan være at vi også har fået nogle oplysninger om fk. øen pingviner bor på, eller deres køn.

### 9.3 Kmeans: hvor mange clusters?

Vi gættede på 3 clusters i ovenstående analyse (da vi havde oplysninger om arter i forvejen) men det godt kunne være, at et andet antal clusters passer bedre til datasættet. Vi kan beregne flere clusterings og angive forskellige antal clusters, og dernæst bruge outputterne fra resultaterne til at tage en beslutning om, hvor mange clusters vi gerne vil angiv i vores færdig clustering.

Det er vigtigt at kunne finde frem til en hensigtsmæssigt antal clusters -

- For mange clusters kan resultatere i over-fitting, hvor vi har for mange til at fortolke eller give mening,
- For få kan betyde at vi mangler indsigt i strukturen eller vigtige trends i datasættet.

### 9.3.1 Få Broom output for forskellige antal clusters

I følgende laver jeg en custom funktion, der laver en clustering på datasættet `penguins_scaled` og hvor jeg angiver, at antal beregnede clusters skal være `.x`, der er en integer (fk. 1,3,99 osv.). Bemærk derfor, at selve data er samme hver gang jeg anvender funktionen - det er bare antal clusters jeg beregner, der kan variere.

```
my_func <- ~kmeans(penguins_scaled, centers = .x)
```

Dernæst laver jeg en `tibble` med variablen `k` som indeholder heltal fra 1 op til 9. Når man anvender funktionen `map` på kolonnen `k` med ovenstående funktion `my_func`, svarer det til, at jeg anvender `kmeans` ni gange, med antal clusters fra 1 til 9. Jeg gemmer clustering resultaterne i en kolon der hedder `kclust`, og så anvende `tidy`, `glance` og `augment` til at få de forskellige outputter fra mine clusterings.

```
kclusts <-
  tibble(k = 1:9) %>%
  mutate( kclust = map(k, my_func),
         tidied = map(kclust, tidy),
         glanced = map(kclust, glance),
         augmented = map(kclust, ~.x %>% augment(penguins))
  )
```

Husk at for at få frem resultaterne i de forskellige former fra `tidy`, `glance` og `augment` er vi nødt til at anvende funktionen `unnest()` - her gemme jeg resultaterne i tre nye dataframes, som vi kan referere til efterfølgende.

```
kclusts_tidy <- kclusts %>% unnest(tidied)
kclusts_augment <- kclusts %>% unnest(augmented)
kclusts_glance <- kclusts %>% unnest(glanced)
```

### 9.3.2 Elbow plot (glance)

Vi bruger `tot.withinss` fra outputtet fra `glance()` (dataframen `kclusts_glance`). Det giver målinger for den totale afstand af observationerne fra deres nærmeste centroid (within sum of squares).

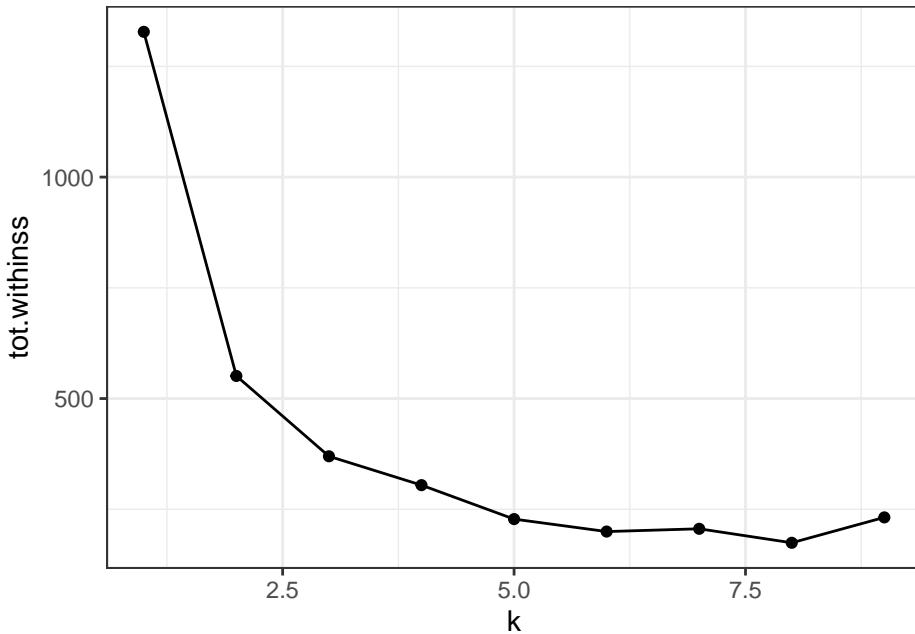
```
kclusts_glance
```

```
## # A tibble: 9 x 8
##       k kclust    tidied      totss tot.withinss betweenss   iter augmented
##   <int> <kmeans> <tibble [1 x 7]>  <dbl>      <dbl>     <dbl> <int> <list>
## 1     1 <kmeans> <tibble [1 x 7]> 1328.    1328.  9.09e-13     1 <tibble>
## 2     2 <kmeans> <tibble [2 x 7]> 1328.    551.  7.77e+ 2     1 <tibble>
## 3     3 <kmeans> <tibble [3 x 7]> 1328.    370.  9.58e+ 2     3 <tibble>
## 4     4 <kmeans> <tibble [4 x 7]> 1328.    304.  1.02e+ 3     2 <tibble>
## 5     5 <kmeans> <tibble [5 x 7]> 1328.    228.  1.10e+ 3     3 <tibble>
```

```
## 6     6 <kmeans> <tibble [6 x 7]> 1328      200.  1.13e+ 3    3 <tibble>
## 7     7 <kmeans> <tibble [7 x 7]> 1328      206.  1.12e+ 3    4 <tibble>
## 8     8 <kmeans> <tibble [8 x 7]> 1328      174.  1.15e+ 3    5 <tibble>
## 9     9 <kmeans> <tibble [9 x 7]> 1328      232.  1.10e+ 3    4 <tibble>
```

Jo flere clusters, jo mindre statistikken `tot.withinss` er som regel, men vi kan se i følgende plot, at efter 2 eller 3 clusters, er der ikke meget gevinst ved at bruge flere clusters. Derfor vælger man enten 2 eller 3. Plottet er ofte kaldes for en ‘elbow’ plot - man vælger de tal på den ‘elbow’, hvor der ikke er meget gevinst med at have flere clusters i datasættet (men det er selvfølgelig meget subjektiv, det tal man vælger til sidste).

```
kclusts_glance %>%
  ggplot(aes(x = k, y = tot.withinss)) +
  geom_line() +
  geom_point() +
  theme_bw()
```



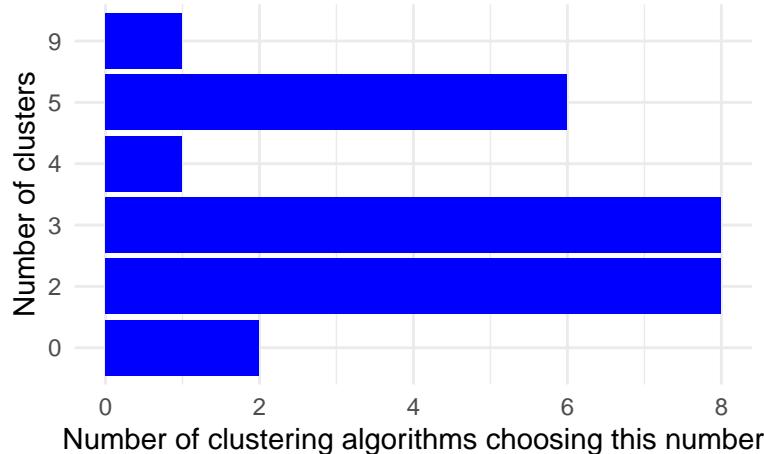
### 9.3.3 Automatistiske beslutning med pakken NbClust

Man kan man også overvejer at prøve noget mere automatisk. For eksempel pakken NbClust lave 30 forskellige clustering algoritme på datasættet fra antal clusters = 2 op til til antal cluster = 9 og for hver af de 30 tager en beslutning om de bedste antal clusters. Man kan således se hvilket antal clusters blev valgt fleste gange af de forskellige algoritme.

```
library(NbClust)
set.seed(24) #fordi outputt af NbClust har indbygget tilfældighed
cluster_30_indexes <- NbClust(data = penguins_scaled,
                               distance = "euclidean",
                               min.nc = 2,
                               max.nc = 9,
                               method = "complete")
```

Man kan se i følgende, at enten 2 eller 3 er optimelt, som passer sammen med den elbow plot methode.

```
as_tibble(cluster_30_indexes$Best.nc[1,]) %>%
  ggplot(aes(x=factor(value))) +
  geom_bar(stat="count", fill="blue") +
  xlab("Number of clusters") + ylab("Number of clustering algorithms choosing this number of clusters") +
  coord_flip() +
  theme_minimal()
```



#### 9.3.4 Plot de forskellige antal clusters (augment)

Vi kan også visualisere hvordan de forskellige antal clusters ser ud. Her kan vi bruge vores resultater fra funktionen `augment` (`kclusts_augment`), som indeholder tilknytninger af observationerne til clusters for hver af de 9 clusterings. Læg mærke til at `kclusts_argument` har 2997 observationer, der svarer til 9 (antal clusterings) x 333 (antal observationer i `penguins`), fordi vi brugt `unnest` til at lægge samtlige resultaterne sammen.

```
kclusts_augment %>% glimpse()
```

```
## Rows: 2,997  
## Columns: 13  
## $ k <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
```

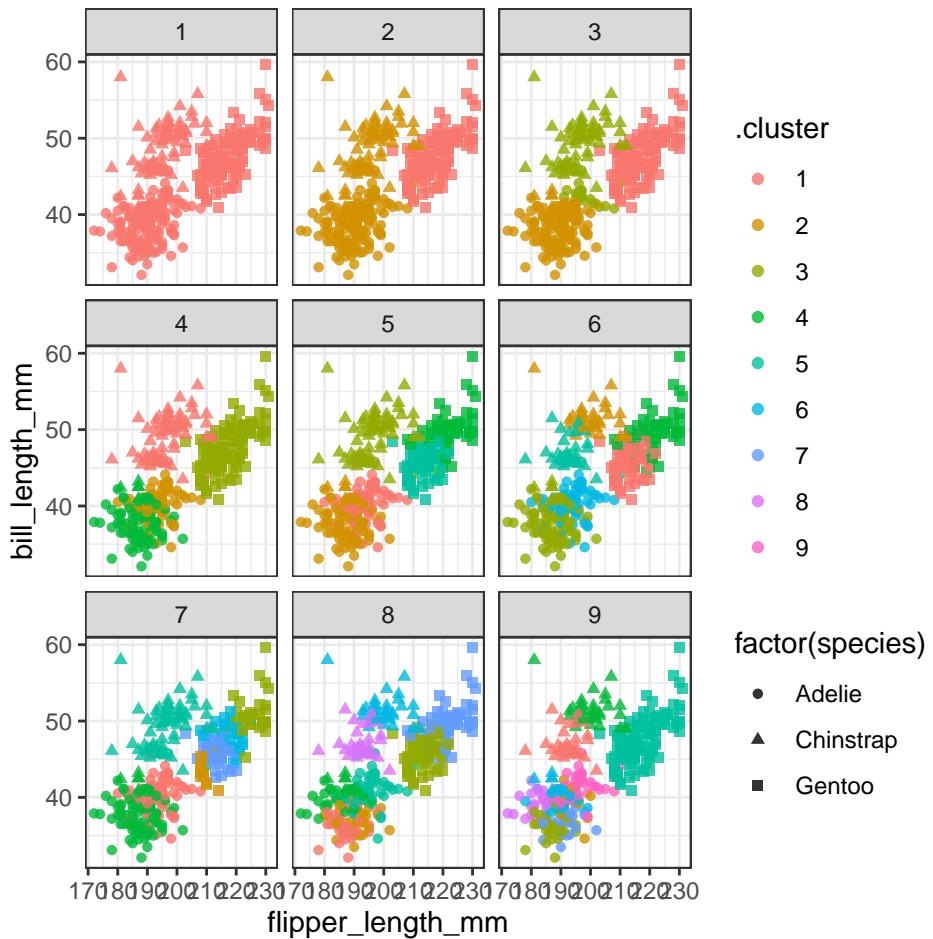
```

## $ kclust          <list> [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ tidied         <list> [<tbl_df[1 x 7]>], [<tbl_df[1 x 7]>], [<tbl_df[1 x ~
## $ glanced        <list> [<tbl_df[1 x 4]>], [<tbl_df[1 x 4]>], [<tbl_df[1 x ~
## $ species         <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
## $ island          <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgers~
## $ bill_length_mm   <dbl> 39.1, 39.5, 40.3, 36.7, 39.3, 38.9, 39.2, 41.1, 38.6~
## $ bill_depth_mm    <dbl> 18.7, 17.4, 18.0, 19.3, 20.6, 17.8, 19.6, 17.6, 21.2~
## $ flipper_length_mm <int> 181, 186, 195, 193, 190, 181, 195, 182, 191, 198, 18~
## $ body_mass_g       <int> 3750, 3800, 3250, 3450, 3650, 3625, 4675, 3200, 3800~
## $ sex              <fct> male, female, female, female, male, female, male, fe~
## $ year              <fct> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
## $ .cluster          <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
```

I følgende kode plotter jeg `flipper_length_mm` vs `bill_length_mm`, og anvende `facet_wrap` så at hver clustering får sit eget plot (så der er 333 observationer pr. plot).

```

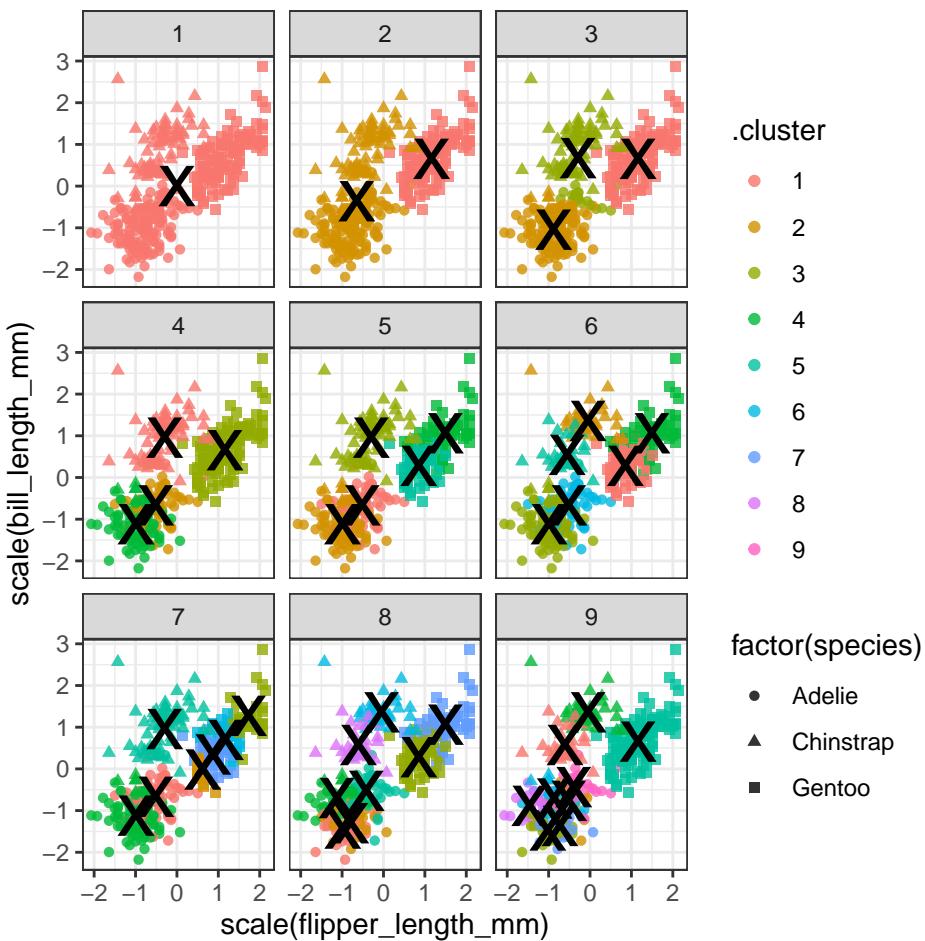
kclusts_augment %>%
  ggplot(aes(x = flipper_length_mm, y = bill_length_mm, colour=.cluster)) +
  geom_point(aes(shape=factor(species)), alpha = 0.8) +
  facet_wrap(~ k) +
  theme_bw()
```



Vi kan nemt inddrage `kclusts_tidy()` og lave "X"-former, bare ved at tilføje en ekstra `geom_point` og angive `kclusts_tidy`. Jeg anvender først funktionen `rename` så at variablen `cluster` fra `kclusts_tidy` matcher til `.cluster` fra `kclusts_augment`.

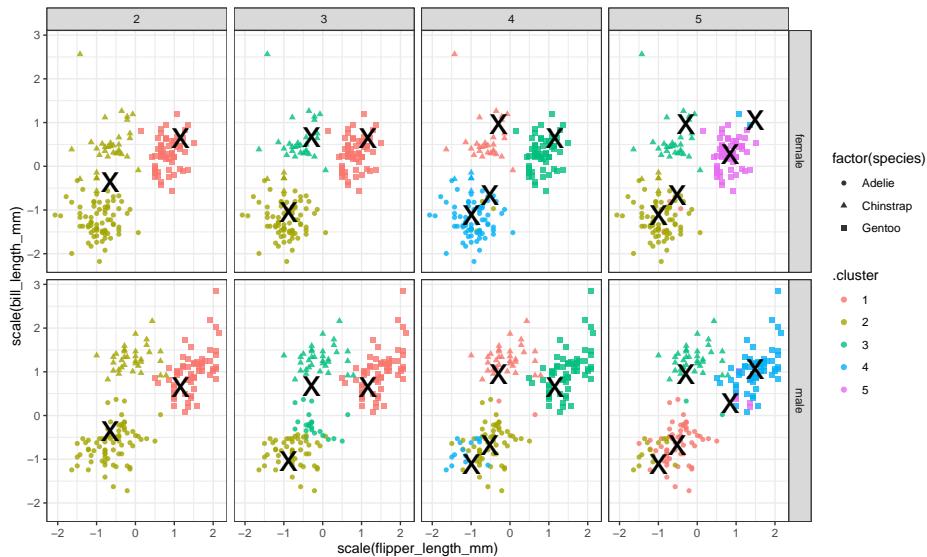
```
kclusts_tidy <- kclusts_tidy %>% rename(.cluster=cluster)

kclusts_augment %>%
  ggplot(aes(x = scale(flipper_length_mm), y = scale(bill_length_mm), colour=.cluster))
    geom_point(aes(shape=factor(species)), alpha = 0.8) +
    facet_wrap(~ k) +
    geom_point(data = kclusts_tidy,
      aes(x=flipper_length_mm,y=bill_length_mm), #already based on scaled
      size = 10, shape = "x", col="black", show.legend = FALSE) +
    theme_bw()
```



Vi kan prøve at kigge endnu dyber ind i resultaterne - her introducerer jeg `sex` som en ekstra variabel i plottet. Husk at variablen `sex` var ikke blevet brugt i vores k-means clusterings, men det kan være, at der er nogle aspekter af de fire variabler, som kan fortælle os nogle om kønnet af pingvinerne. For at spare plads, har jeg kun plottet antal clusters fra 2 til 5.

```
kclusts_augment %>% filter(k %in% 2:5) %>%
  ggplot(aes(x = scale(flipper_length_mm), y = scale(bill_length_mm), colour=.cluster)) +
  geom_point(aes(shape=factor(species)), alpha = 0.8) +
  facet_grid(sex ~ k) +
  geom_point(data = kclusts_tidy %>% filter(k %in% 2:5),
             aes(x = flipper_length_mm,
                 y = bill_length_mm),
             size = 10, shape = "x", colour = "black", show.legend = FALSE) +
  theme_bw()
```



### 9.3.5 Nest/map ramme fra sidste gange

Som sidste bemærk med k-means, kan man også lave en clustering til de tre arter hver for sig. I følgende opretter jeg en nested dataframe, som indeholder 3 datasæt (`penguins` opdelt efter variablen `species`), og jeg anvender den custom funktion `scale_me` til at udvælge de numeriske variabler og anvende `scale()` i hvert datasæt.

```
scale_me <- ~.x %>% select(where(is.numeric)) %>% scale

penguins_nest <- penguins %>%
  group_by(species) %>%
  nest() %>%
  mutate("data_scaled" = map(data,scale_me))
```

Næste laver jeg en custom funktion til at lave en clustering på datasættet `.x`, og angiver at antal clusters skal være 3. Bemærk at i ovenstående sektion varierede vi på antal clusters (indstilling `centers`), men her fastlægger vi antal clusters og så variere selve datasæt i stedet for.

```
cluster_me <- ~.x %>% kmeans(centers=3)
```

Jeg anvender `cluster_me` på mine skaleret datasæts, og så anvender `glance`, `augment` og `tidy` på clustering resultater ligesom i ovenpå (bemærk brugen af `map` til at `augment` de opdelte datasæt).

```
penguins_nest <- penguins_nest %>%
  mutate(clusters = map(data_scaled,cluster_me),
        clusters_glance = map(clusters,glance),
```

```

clusters_augment = map2(clusters,data_scaled,~.x %>% augment(.y)), #I augment the scaled
clusters_tidy = map(clusters,tidy)

nested_clusters_augment <- penguins_nest %>% unnest(clusters_augment)
nested_clusters_tidy <- penguins_nest %>% unnest(clusters_tidy)

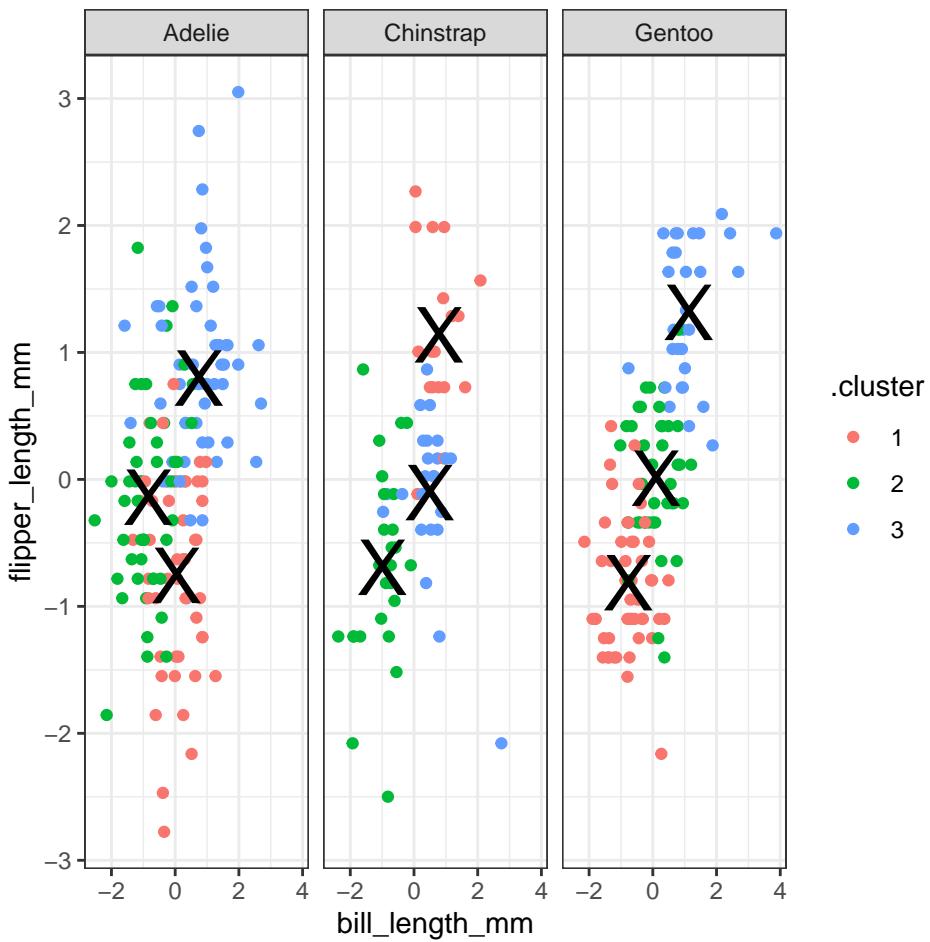
```

Til sidste laver jeg en plot af resultaterne:

```

nested_clusters_augment %>%
  ggplot(aes(x=bill_length_mm,y=flipper_length_mm,colour=.cluster)) + #data already scaled
  geom_point() +
  facet_grid(~species) +
  geom_point(data=nested_clusters_tidy,,,
             shape="X",colour="black",
             size = 10) +
  theme_bw()

```



## 9.4 Method 2: Hierarchical clustering

K-means er en meget populær metode til at lave clustering, men der er mange andre metoder, f.eks. hierarchical clustering. Vi skifter over til `mtcars`, og ligesom i `kmeans` skal vi først bruge `scale` på de numeriske kolonner i de data.

```
mtcars_scaled <- mtcars %>% select(where(is.numeric)) %>% scale()
```

I modsætning til k-means, for at lave hierarchical clustering skal man først beregne afstanden mellem alle de observationer i de data. Det gør man med funktionen `dist()` (som bruger den Euclidean distance som default):

```
d <- dist(mtcars_scaled)
```

For at lave en hierarchical clustering anvender man funktionen `hclust()`. Metoden `complete` er default men man kan afprøve de andre methoder (der er ikke en fast regel over for, hvilken metode man skal bruge).

```
mtcars_hc <- hclust(d, method = "complete")
# Metoder: "average", "single", "complete", "ward.D"
```

I følgende arbejder vi lidt med `mtcars_hc` til at få nogle clusters frem, og til at lave et plot.

### 9.4.1 Vælge ønsket antal clusters

Funktionen `cutree` anvendes til at få clusters fra resultaterne af funktionen `hclust`. For eksempel hvis man gerne vil have 4 clusters, bruger man `k = 4`. Jeg specificerer `order_clusters_as_data = FALSE` for at få clusters i rækkefølgen, som passer til plottet (dendrogram) vi laver (bemærk at man skal have pakken `dendextend` installeret for at få den til at fungere).

```
library(dendextend)

clusters <- cutree(mtcars_hc, k = 4, order_clusters_as_data = FALSE)
```

Her laver jeg et overblik over, hvor mange observationer fra `mtcars` er i hver cluster:

```
tibble("cluster"=clusters) %>% group_by(cluster) %>% summarise(n())

FALSE # A tibble: 4 x 2
FALSE   cluster `n()`
FALSE     <int> <int>
FALSE 1      1      7
FALSE 2      2      8
FALSE 3      3     12
FALSE 4      4      5
```

### 9.4.2 Lav et påent plot af dendrogram med ggplot2

Første anvender jeg funktionen `dendro_data()` til at udtrække den “dendrogram” fra de `hclust()` resultater.

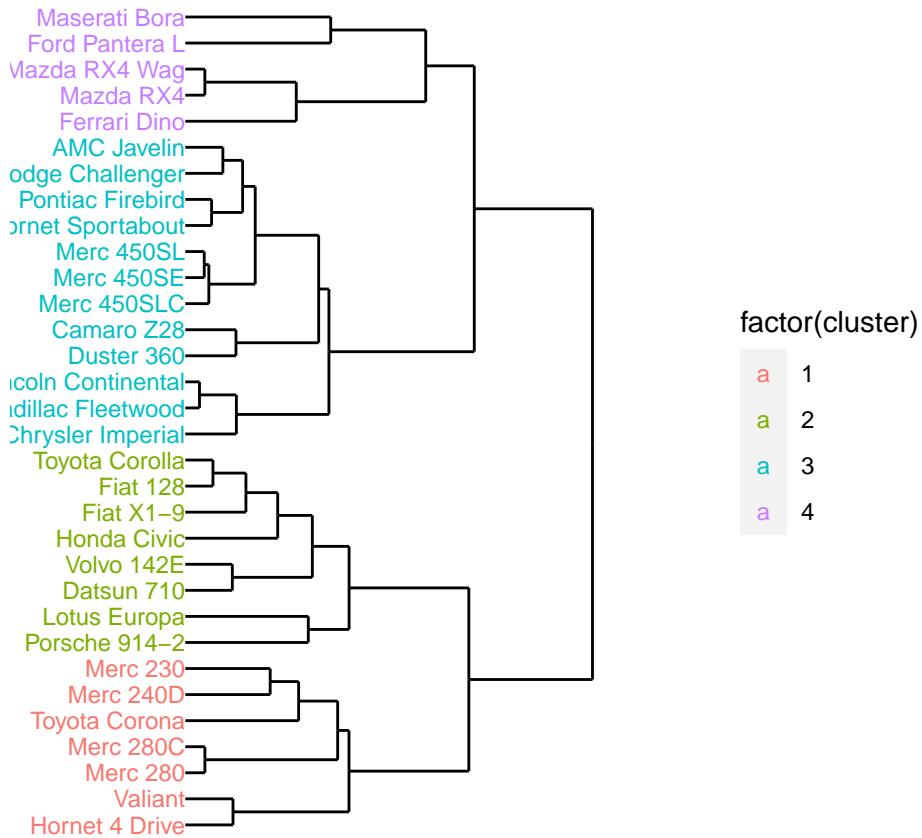
```
library(ggdendro)
dend_data <- dendro_data(mtcars_hc %>% as.dendrogram, type = "rectangle")
```

Vi tilføjer vores clusters som vi beregnede ovenpå (det er derfor vi sikret rækkefølgen af de clusters):

```
dend_data$labels <- dend_data$labels %>%
  mutate(cluster = clusters)
```

Vi benytter `dend_data$segments` og `dend_data$labels` til at lave et informativt plot af de data i `ggplot2`.

```
ggplot(dend_data$segments) +
  geom_segment(aes(x = x, y = y, xend = xend, yend = yend)) +
  coord_flip() +
  geom_text(data = dend_data$labels,
            aes(x, y, label = label, col = factor(cluster)),
            hjust = 1, size = 3) +
  ylim(-3, 10) +
  theme_dendro()
```



Så kan man se, der er fire clusters i dengrammet, og biler der er tætest på hinanden ligner hinanden mest - fk. Merc 280C og Merc 280 må være meget éns, og er som forventet lige ved siden af hinanden i plottet.

Man kan godt tilpasse ovenstående kode til et andet datasæt - se problemstillinger, men man må også gerne udvide plottet med de forskellige viden vi har om ggplot2.

#### 9.4.3 Ekstra (valgfri): afprøve andre metoder på hierarchical clustering

Valfri ekstra hvis du vil afprøve de fire metoder i hclust - “average”, “single”, “complete” og “ward.D”.

```
# samme ggplot kommando som ovenpå lavet til en funktion
den_plot <- ~ggplot(.x$segments) +
  geom_segment(aes(x = x, y = y, xend = xend, yend = yend)) +
  coord_flip() +
  geom_text(data = .x$labels,
            aes(x, y, label = label),
```

```

  hjust=1, size=2) +
  ylim(-4, 10) + theme_dendro()

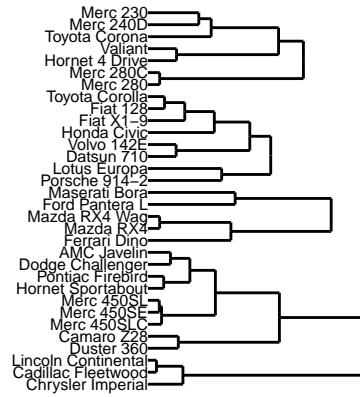
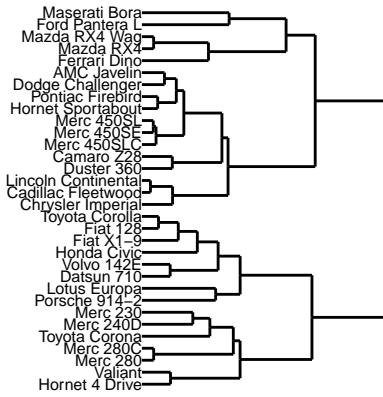
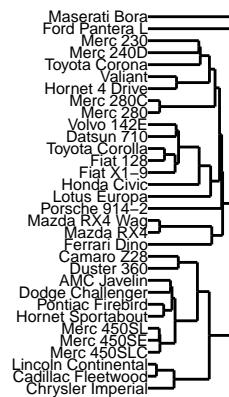
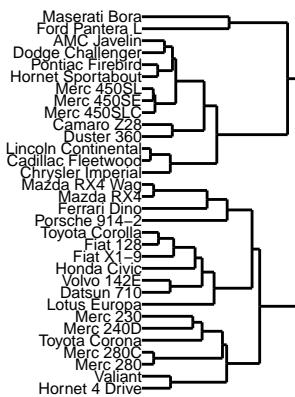
```

Vi iterate over de fire metoder og lave samme process som ovenpå med map. Derefter kan man lave et plot fk. med grid.arrange:

```
# fire metoder:
m <- c("average", "single", "complete", "ward.D")

hc_results <-
  tibble(method = m) %>%
  mutate( kclust = map(method, ~hclust(d, method = .x)),
         dendrogram = map(kclust, as.dendrogram),
         den_dat = map(dendrogram, ~dendro_data(.x, type="rectangle")),
         plot = map(den_dat, den_plot))

library(gridExtra)
grid.arrange(grobs = hc_results %>% pull(plot), ncol=2)
```



## 9.5 Problemstillinger

### Problem 1) Quiz - Clustering

---

**Problem 2)** Funktionen `kmeans`. I ovenstående anvendt vi `mtcars` i hierarchical clustering, men lad os se, hvordan det ser ud med `k-means`. Du er velkommen til at tilpasse min ovenstående kode fa det `penguins` datasæt:

- a) Benyt `kmeans` til at finde 2 clusters i datasættet `mtcars`:
    - husk at vælge kun de numeriske kolonner og scale datasættet i forvejen
    - gem din clustering som `my_clusters`.
    - hvor mange observationer er der i hver af de to clusters?
  - b) Anvend funktionen `augment` til at forbinde det oprindelige datasæt til dine clusters fra `my_clusters` (skriv `mtcars` indenfor funktionen `augment`).
  - c) Brug dit “augmented” datasæt til at lave et scatter plot mellem to af de numeriske variabler (vælg selv) i datasættet og giv dem farver efter din clusters, som du har beregnet. Da du har forbundet det oprindeligt datasæt (der ikke var scaled) i `augment`, scale din variabler i plottet.
  - d) Tilføj `tidy` til at få fat i de middelværdier/centroids af hver af de 2 clusters og tilpas min kode fra notaterne (sektion 9.2.5) til at tilføje dem til plottet som ‘x’ (husk at din “centers”/centroids er allerede baserede på scaled data så du behøver ikke at anvende `scale` på deres værdier).
- 

### Problem 3) Hierarchical clustering øvelse

Vi laver en analyse af det `msleep` datasæt. Jeg har lavet oprydningen og scaling for jer:

```
data(msleep)
msleep_clean <- msleep %>% select(name,where(is.numeric)) %>% drop_na()
msleep_scaled <- msleep_clean %>% select(-name) %>% scale
row.names(msleep_scaled) <- msleep_clean$name
```

Tilpas min kode fra kursusnotaterne (sektion 9.4) til at lave følgende:

- a) Benyt funktioner `dist` og dernæst `hclust` på datasættet `msleep_scaled`.
- b) Benyt `cutree` for at finde 5 clusters fra dine `hclust`-resultater, og kalde det for `clusters`. Husk at anvende `order_clusters_as_data = FALSE` så at vi har den korrekt rækkefølge for et plot (*OBS man skal installere/indlæse pakken dendextend*)
- c) Benyt `dendro_data` til at udtrække de dendrogram fra resultaterne og tilføj `clusters` til `dend_data$labels` (kopier kode fra 9.4.2).

- d) Lav et dendrogram plot: igen tilpas koden (9.4.2) for `mtcars` eksempel for nuværende data
- 

#### Problem 4)

Inlæs data

```
wholesale <- read.csv("https://www.dropbox.com/s/7nb5pkruqt4fqn4/Wholesale%20customers%20data.csv")
```

- a) Ændre på datasættet efter følgende instruks:
- Channel - anvend `recode` for at ændre til navne
    - 1 = horeca
    - 2 = retail
  - Region - anvend `recode` for at ændre til navne
    - 1 = Lisnon
    - 2 = Oporto
    - 3 = Other
  - Anvend `map_if` til at transformere samtlige numeriske variabler med log  
(kode er: `wholesale <- wholesale %>% map_if(is.numeric, log) %>% as_tibble()`).
- b) Udvælg de numeriske variabler fra dit datasæt og anvende `scale()` - kalde dit nye datasæt for `wholesale_scale`
- c) Tilpas min kode fra sektion 9.3.1 til at lave 10 clusterings ( $k=1:10$ ) på `wholesale_scale` og gem dem i en dataframe, sammen med din clusterings resultater i “tidy”, “glance” og “augment” form.
- d) Lav et elbow plot fra dit output fra `glance` (sektion 9.3.2)
- e) Udvælg clusterings hvor  $k$  er fra 2 til 7 fra dit output fra `augment` og lav scatter plots af variabler `Frozen` VS `Fresh`, hvor du:
- Giv farve efter `.cluster`
  - Adskil plots efter `k`
  - Prøv bagefter at også adskil dit plots yderligere efter `Channel`.
- f) Tilpas koden fra 9.3.5 til at lave en analyse for “hoerca” og “retail” (variablen `Channel`) hver for sig. Angiv 4 clusters i din analyse.
- g) Lav et plot af din clustering (adskilt efter variablen `Channel`) og få “x” på plotterne til at vise din cluster middelværdier for `Frozen` og `Fresh`.



# Chapter 10

## Principal component analysis (PCA)

```
library(tidyverse)
library(broom)
```

### 10.1 Indledning og læringsmålene

#### 10.1.1 Læringsmålene

Du skal være i stand til at

- Forstå koncepten bag principal component analysis (PCA)
- Benytte PCA i R og lave et plot af et datasæt i to dimensioner
- Vurdere den relative varians forklarede af de forskellige components
- Anvende PCA til at vurdere variabernes bidrag til de principal components

#### 10.1.2 Introduktion til chapter

Principal component analysis er en meget populær og benyttet statistiske metode og kan anvendes til bla. at visualisere data med et højt antal dimensioner i et enkelt scatter plot med to dimensioner. Det er meget nyttigt for at se den underliggende struktur i datasættet og indenfor biologi er det meget brugt til at blandt andet visualisere hvor de forskellige samples eller replikates sidder relativt til hinanden - for eksempel for at se, om de controls samples og treatment samples fremgår i samme steder på plottet (som indikerer at de ligner hinanden).

### 10.1.3 Video ressourcer

- Video 1 - hvad er PCA?

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556581604>

---

- Video 2 - hvordan man lave PCA i R og få output i tidy form

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556581588>

---

- Video 3 - hvordan man visualisere de data (principal components, rotation matrix)

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556787141>

## 10.2 Hvad er principal component analysis (PCA)?

I sidste lektion arbejdede vi med `penguins`, hvor vi så at der faktisk var fire numeriske variabler - altså fire dimensioner - som blev brugt til at lave k-means clustering.

```
library(palmerpenguins)
penguins <- penguins %>%
  drop_na() %>%
  mutate(year=as.factor(year))

penguins %>% select(where(is.numeric)) %>% head()

## # A tibble: 6 x 4
##   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##       <dbl>        <dbl>          <int>        <int>
## 1         39.1        18.7          181        3750
## 2         39.5        17.4          186        3800
## 3         40.3         18            195        3250
## 4         36.7        19.3          193        3450
## 5         39.3        20.6          190        3650
## 6         38.9        17.8          181        3625
```

Når man laver et plot for at vise de forskellige clusters, får man et problem - hvilke to variable skal plottes? Man kan plotte hver eneste pair af variabler. For eksempel kan man prøve en pakke der hedder `GGally`, som automatiske

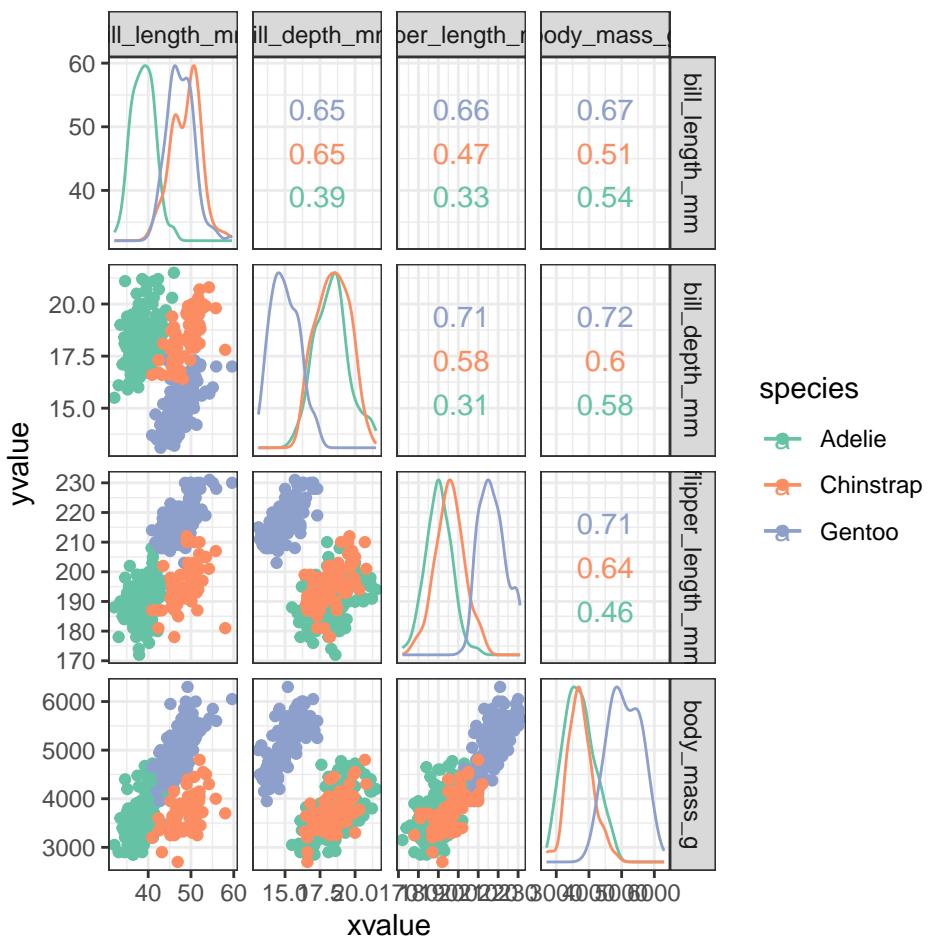
kan plotte de forskellige pairs af numeriske variabler og beregner korrelationen mellem variablerne.

```
require(GGally)

## Indlæser krævet pakke: GGally

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

penguins %>%
  ggscatmat(columns = 3:6 ,color = "species", corMethod = "pearson") +
  scale_color_brewer(palette = "Set2") +
  theme_bw()
```



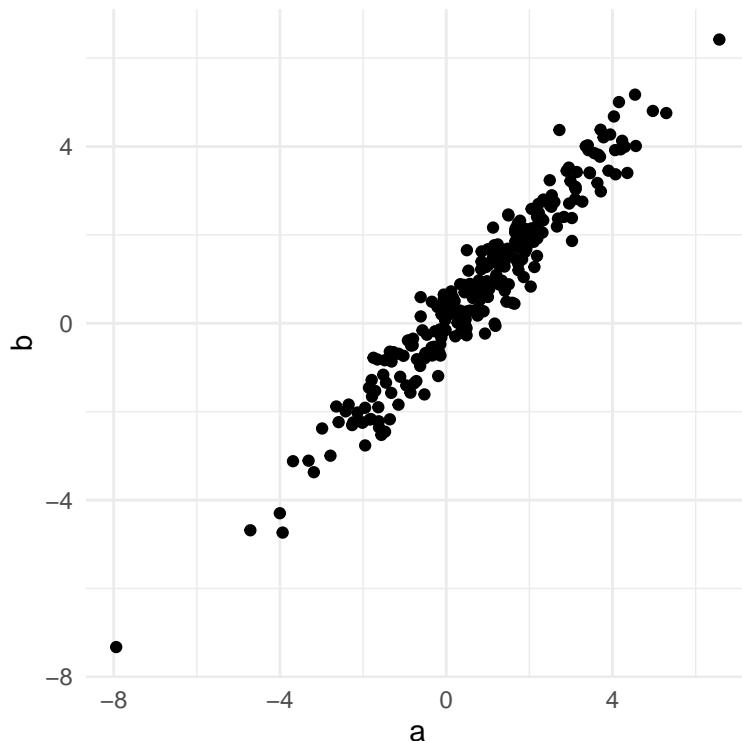
Problemet er, at så snart antallet af dimensioner i datasættet bliver større end 4, bliver plottet alt for kompleks og pladskrævende.

En løsning til problemmet er at projektere datasættet ned indtil et mindre antal dimensioner (fk. kun 2 dimensioner). Disse dimensioner fanger oplysninger fra alle variablerne i datasættet, og derfor når man lave et scatter plot, får man repræsenteret det hele datasæt i stedet for kun to udvalgte variabler. Metoden for at lave disse såkaldte ‘projektion’ kaldes for ‘principal component analysis’.

### 10.2.1 Simpel eksempel med to dimensioner

Man kan prøve at forstå hvordan PCA fungerer ved at kigge på et simpelt eksempel med 2 dimensioner:

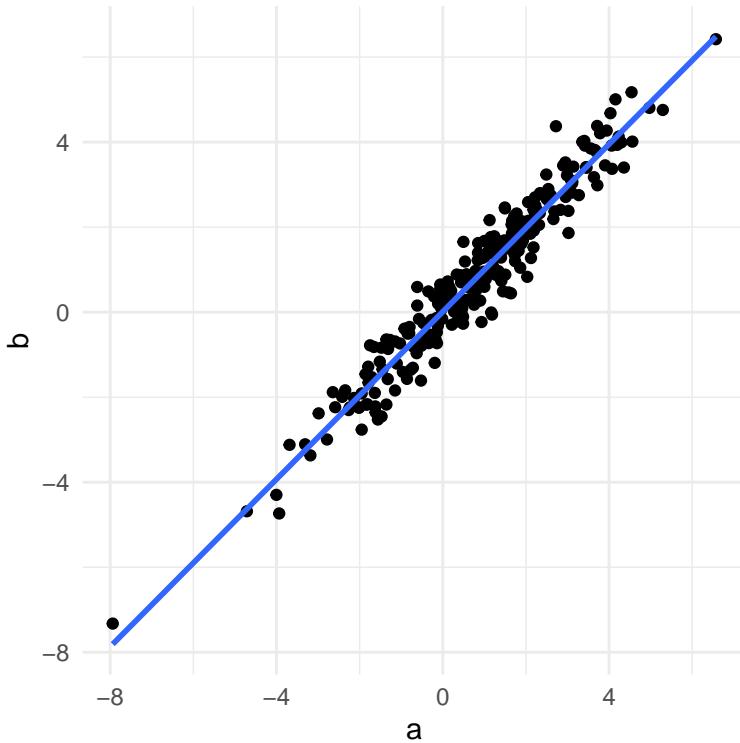
```
#simulere data med en høj korrelation
a <- rnorm(250,1,2)
b <- a + rnorm(250,0,.5)
df <- tibble(a,b)
ggplot(df,aes(a,b)) +
  geom_point() +
  theme_minimal()
```



Vi kan se her, at der er en meget stor korrelation mellem a og b. Selvom datasættet er plottet i 2 dimensioner kan de næsten forklares af én linje - en såkaldte bedste rette linje der passer gennem punkterne.

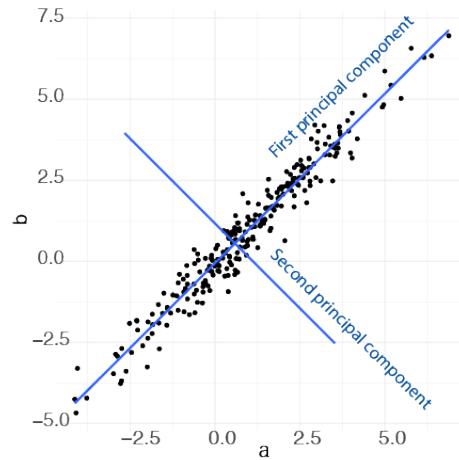
```
df <- tibble(a,b)
ggplot(df,aes(a,b)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm",se=FALSE)

## `geom_smooth()` using formula 'y ~ x'
```



Med andre ord kan vi næsten forklare datasættet i blot én dimension - punkternes afstand langt linjen. Når man tager alle punkterne og beskriver dem langt én linje som bedste beskrive variansen i datasættet, kaldes den linje for den første principal component (PC1). Man kan dernæst beskrive en anden linje som er vinkelrettet til PC1 som bedste forklarer variancen i de data som ikke var fanget af PC1 - det kaldes for den anden principal component (PC2).

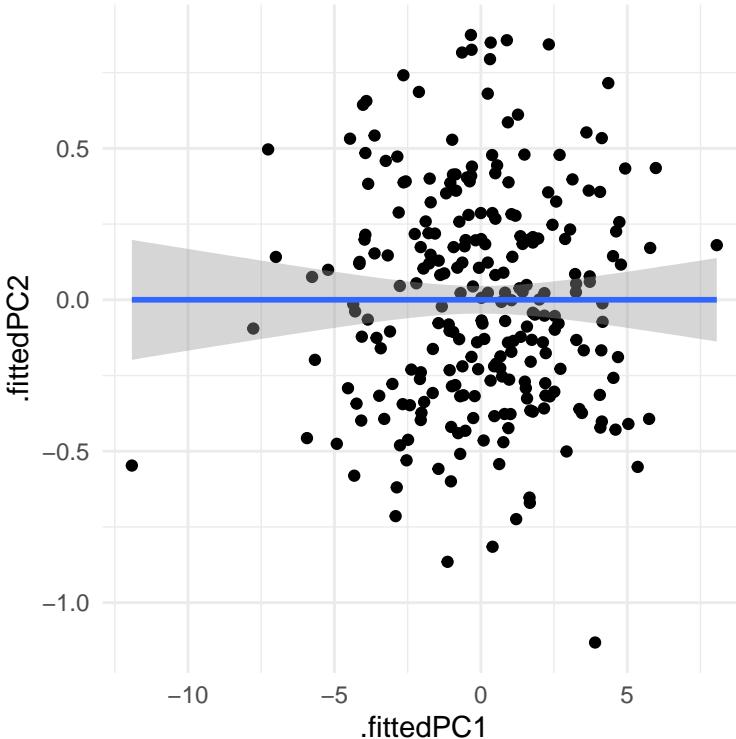
Vi kan se her PC1 og PC2 plottet:



Når vi tager PC1 and PC2 og plotter dem som henholdsvis x-aksen og y-aksen, svarer det til en drejning af akserne i plottet (vi finder PC1 og PC2 fra funktionen `prcomp` som jeg forklarer i næste sektion):

```
dat <- augment(prcomp(df),df)
ggplot(dat,aes(x=.fittedPC1,y=.fittedPC2)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Vi kan se her, at de data flyder de plads på plottet bedre en før (og bemærk at de askse skala er blevet meget mindre i den nye y-aksen, da de data spreder sig meget mindre langt PC2 i forhold til PC1.)

Det her er kun et eksempel hvor vores oprindelige data ligger i to dimensions (to variabler), for at gøre det nemt at visualisere dem i et plot, men de fleste datasæt (fk penguins, iris osv.) har flere end to dimensioner. Vi kan godt lave samme process, hvor vi definerer PC1 som forklarer så meget af variancen i de data som muligt, og dernæst PC2 som forklarer nogle af variancen ikke fanget af PC1, og dernæst PC3 osv., efter hvor mange dimensioner de data har. I mange praktisk situationer vælger man de første to komponenter, som er mest vigtige, da de forklarer mest af variancen i de data i forhold til de andre komponenter.

“So to sum up, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.” <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>

## 10.3 Fit PCA to data in R

```
library(broom)
```

Lad os skifte tilbage til nogle virkelige data for at benytte `prcomp`: datasættet

penguins. Med `prcomp` fokuserer vi kun på numeriske variabler, så vi bruger `select` med `where(is.numeric)` og så anvender scaling ved at specificere `scale = TRUE` indenfor funktionen `prcomp`.

```
pca_fit <- penguins %>%
  select(where(is.numeric)) %>% # retain only numeric columns
  prcomp(scale = TRUE) # do PCA on scaled data

summary(pca_fit)

## Importance of components:
##                 PC1     PC2     PC3     PC4
## Standard deviation 1.6569 0.8821 0.60716 0.32846
## Proportion of Variance 0.6863 0.1945 0.09216 0.02697
## Cumulative Proportion 0.6863 0.8809 0.97303 1.00000
```

`Proportion of Variance` indikerer hvor meget af variancen i de data blev forklaret af de forskellige komponenter. Vi kan se, at PC1 forklaret omkring 69% og de første to komponenter sammen forklarer 88% af variancen i de data. Derfor hvis vi viser et plot af de første to komponenter ved vi, at vi har fanget rigtig meget oplysninger om de fire variabler i datasættet.

## 10.4 Integrere PCA resultater med broom-pakke

Der er flere ting som kan være nyttige at lave med vores PCA resultater:

- Lave et plot af datasættet ud fra de første to principal components
- Se for meget af variansen i datasættet er forklaret af de forskellige components
- Bruge den rotation matrix til at se, hvordan variabler sidder med relative til hinanden

For at få lavet vores plot af de principal components kan vi benytte funktionen `augment()` ligesom vi gjorde i vores sidste lektion med k-means clustering. Her få vi værdierne til hver af de fire principal components sammen med den oprindelige datasæt.

```
pca_fit_augment <- pca_fit %>%
  augment(penguins) # add original dataset back in

pca_fit_augment

## # A tibble: 333 x 13
##   .rownames species island bill_length_mm bill_depth_mm flipper_length_mm
##   <chr>     <fct>   <fct>        <dbl>        <dbl>            <int>
## 1 1         Adelie  Torgersen      39.1       18.7          181
```

```

##  2 2      Adelie  Torgersen    39.5    17.4    186
##  3 3      Adelie  Torgersen    40.3     18    195
##  4 4      Adelie  Torgersen    36.7    19.3    193
##  5 5      Adelie  Torgersen    39.3    20.6    190
##  6 6      Adelie  Torgersen    38.9    17.8    181
##  7 7      Adelie  Torgersen    39.2    19.6    195
##  8 8      Adelie  Torgersen    41.1    17.6    182
##  9 9      Adelie  Torgersen    38.6    21.2    191
## 10 10     Adelie  Torgersen    34.6    21.1    198
## # ... with 323 more rows, and 7 more variables: body_mass_g <int>, sex <fct>,
## #   year <fct>, .fittedPC1 <dbl>, .fittedPC2 <dbl>, .fittedPC3 <dbl>,
## #   .fittedPC4 <dbl>

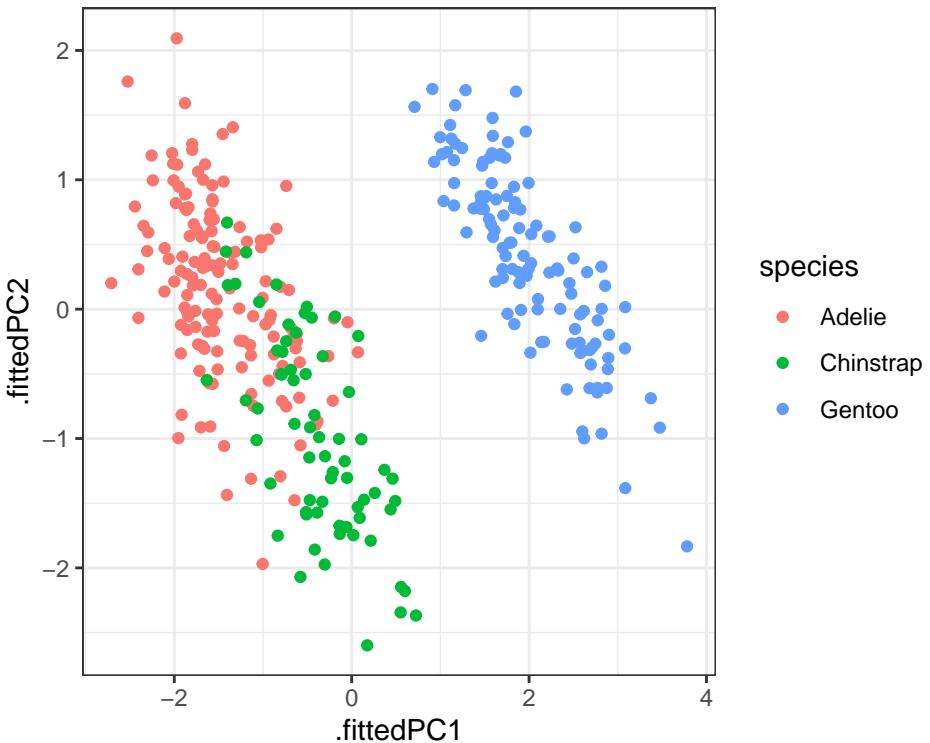
```

Vi kan tage `pca_fit_augment` og lave et plot af de første to principal components:

```

pca_fit_augment  %>%
  ggplot(aes(x=.fittedPC1, y=.fittedPC2, color = species)) +
  geom_point() +
  theme_bw()

```



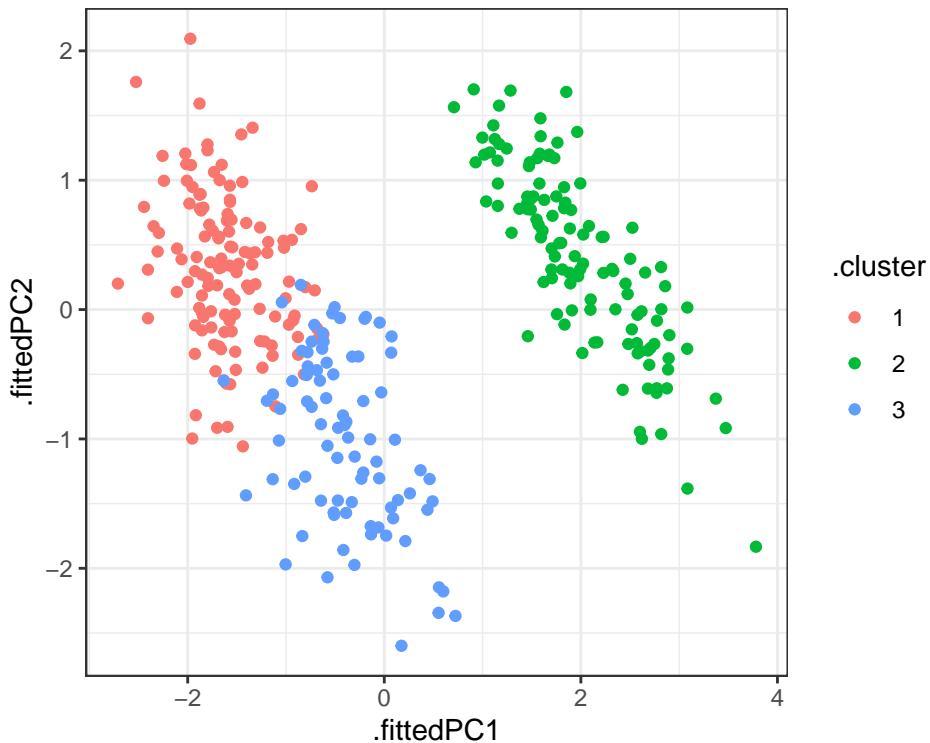
Vi kan også integrere de clusters som vi fik fra funktionen `kmeans()` i vores PCA ved at anvende funktionen `augment()` på resultaterne fra `kmeans` og vores

data som allerede har resultaterne fra `pca`. Da både PCA og k-means fanger oplysninger om strukturen af de data baserede på de fire numeriske variabler, kan man forventer en bedre sammenligning mellem de to (i forhold til at sammenligne de clusters med et plot med kun to af variablerne).

```
penguins_scaled <- penguins %>% select(where(is.numeric)) %>% scale

kclust <- kmeans(penguins_scaled, centers = 3)

kclust %>% augment(pca_fit_augment) %>%
  ggplot(aes(x=.fittedPC1, y=.fittedPC2, color = .cluster)) +
  geom_point() +
  theme_bw()
```



### Output med tidy

Næste kigger vi på variansen i datasættet som er blevet fanget af hver af de forskellige components. Man kan udtrække oplysningerne ved at benytte funktionen `tidy()` fra pakken `broom`, og ved at angiv `matrix = "eigenvalues"` indenfor `tidy`.

Det kaldes for “eigenvalues” fordi, hvis man kigger på matematikken bag principal component analysis, tager man udgangspunkt i en covariance matrix. En covariance matrix beskriver sammenhængen eller korrelationen mellem de forskel-

lige variabler. Man bruger denne covariance matrix til at beregne de såkaldte eigenvalues og deres tilsvarende eigenvectors.

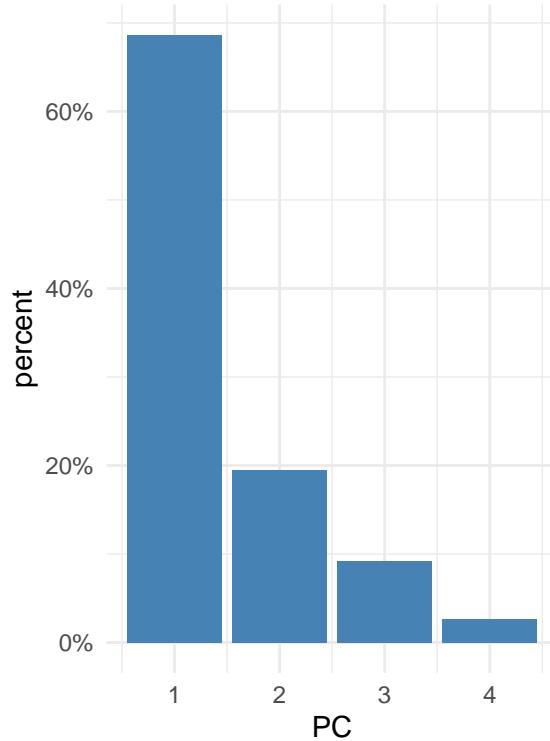
Det er faktisk den største eigenvalue som fortæller os om den første principal component - det fortæller os hvor meget af variansen i datasættet den første principal component fanger - jo større det er relativ til de andre eigenvalues, jo mere variansen man forklarer med den første principal component. Og den næste største fortæller os om den anden principal component og så videre.

```
pca_fit_tidy <- pca_fit %>%
  tidy(matrix = "eigenvalues")
pca_fit_tidy

## # A tibble: 4 x 4
##       PC std.dev percent cumulative
##   <dbl>    <dbl>    <dbl>    <dbl>
## 1     1     1.66    0.686    0.686
## 2     2     0.882   0.195    0.881
## 3     3     0.607   0.0922   0.973
## 4     4     0.328   0.0270   1
```

Lad os visualisere de tal her i procenttal, med at specificere `labels = scales::percent_format()` indenfor `scale_y_continuous` - så vi bare ændre på de tal som kan ses på y-aksen.

```
pca_fit_tidy %>%
  ggplot(aes(x = PC, y = percent)) +
  geom_bar(stat="identity", fill="steelblue") +
  scale_y_continuous(
    labels = scales::percent_format(), #convert labels to percent format
  ) +
  theme_minimal()
```



På den ene side hvis der er meget variance som er forklaret af de første components tilsammen, betyder det at der er en del redundans i datasættet, på grund af, at mange af de variabler har en tæt sammenhæng med hinanden. På den anden side hvis der er en meget lille andel af variancen som er forklaret af de første components tilsammen, betyder det at det er svært at beskrive datasættet i mindre dimensioner (fordi der næsten er ingen sammenhæng mellem variablerne) - i dette tilfælde, hvor datasættet er mere kompleks, er PCA mindre effektiv.

#### 10.4.1 Rotation matrix for at udtrækker bidraget af de forskellige variabler

De eigenvalues kan anvendes til at undersøge variancen i datasættet, men deres tilsvarende eigenvectors fortæller os om, hvordan de forskellige variabler er kombineret til at få de endelige principal component værdier, som vi bruger fk. i et scatter plot. De eigenvectors bruges til at lave et matrix som hedder ‘rotation matrix’.

Jeg anvender funktionen `pivot_wider` for at få vores matrix mere klart at se. Vi kan se at vi har variablerne her på rækkerne og de forskellige principal components i kolonnerne.

```
pca_fit_rotate <- pca_fit %>%
  tidy(matrix = "rotation") %>%
```

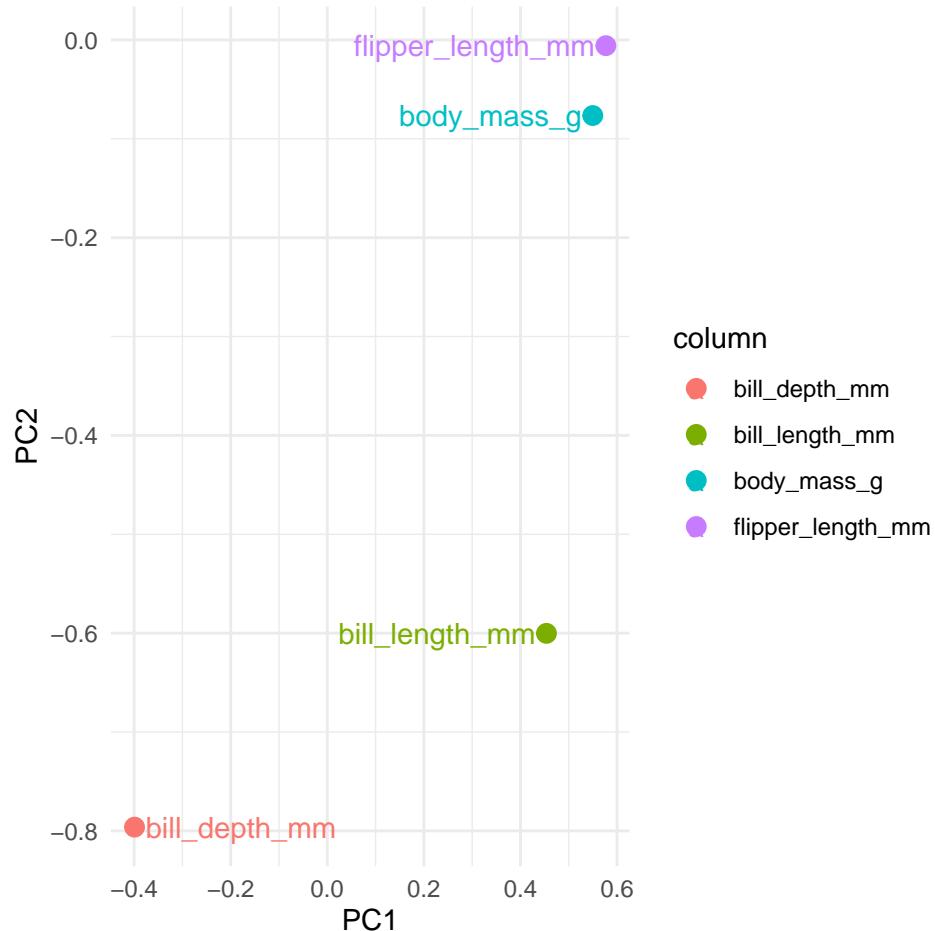
```
pivot_wider(names_from = "PC", names_prefix = "PC", values_from = "value")
pca_fit_rotate
```

```
## # A tibble: 4 x 5
##   column          PC1      PC2      PC3      PC4
##   <chr>       <dbl>    <dbl>    <dbl>    <dbl>
## 1 bill_length_mm 0.454 -0.600 -0.642  0.145
## 2 bill_depth_mm -0.399 -0.796  0.426 -0.160
## 3 flipper_length_mm 0.577 -0.00579 0.236 -0.782
## 4 body_mass_g     0.550 -0.0765  0.592  0.585
```

Den rotation matrix fortæller os hvordan man beregner værdierne af de principal components for alle observationer. For eksempel tager vi vores første observation, beregne 0.45 gange de bill length, og så minus 0.4 gang de bill depth, og så plus 0.58 x den flipper length og så plus 0.55 x body\_mass. Og så har vi værdien for observationen langt den første princip komponent.

Vi kan anvende den rotation matrix til at se hvordan de forskellige variabler relativt til hinanden. Variablerne som er tæt på hinanden i plottet ligner hinanden. Vi kan se at `flipper_length_mm` og `body_mass_g` ligner hinhanen ret meget i vores datasæt, mens `bill_depth_mm` sidder over til venstre langt den første principal component, så det måske indeholder nogle oplysninger om pingvinerne, der ikke kunne fanges i de andre variabler.

```
library(ggrepel)
pca_fit_rotate %>%
  ggplot(aes(x=PC1,y=PC2,colour=column)) +
  geom_point(size=3) +
  geom_text_repel(aes(label=column)) +
  theme_minimal()
```

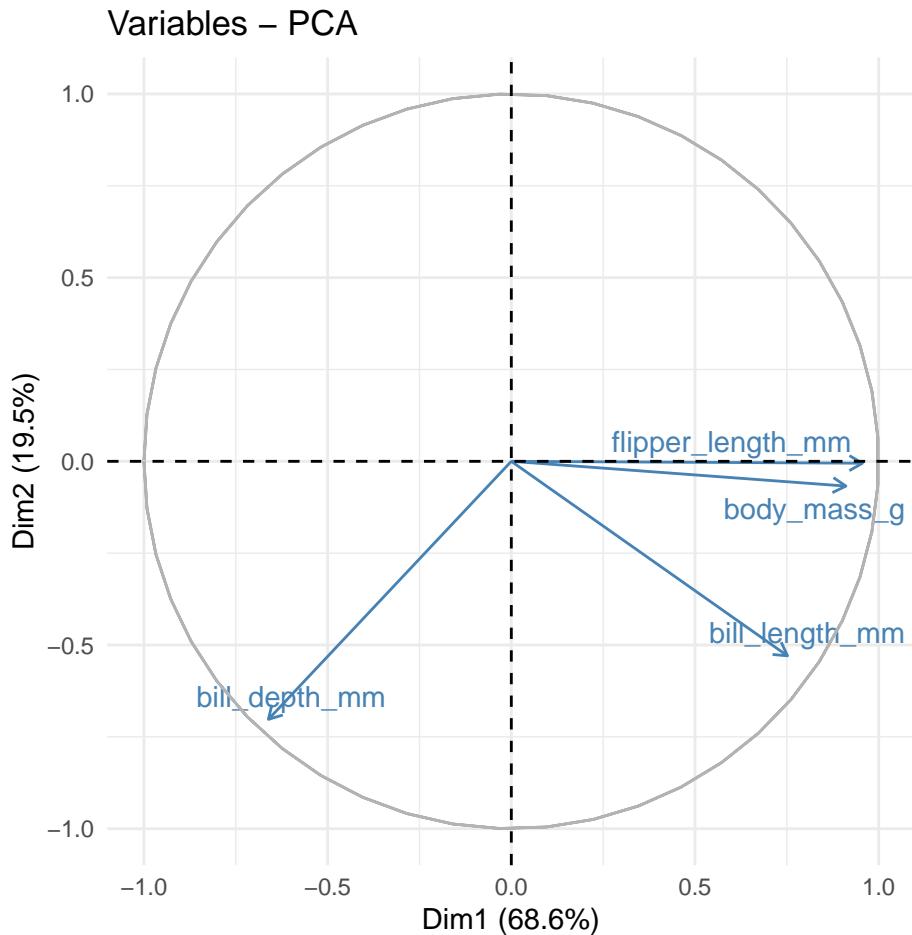


#### 10.4.2 Pakken factoextra

R-pakken **factoextra** kan avendes til at lave et lignende plot fra de rotation matrix automatiske, og den arbejder ovenpå **ggplot2** så man kan ændret temaet osv. Man kan se hvordan de fungere i følgende kode.

- Man få det varians procenttal på akserne.
- Lokationer af pilehovederne er fra den rotation matrix.
- Jo mindre vinklen mellem to linjer er, og tættere på de er til hinanden
- Jo nærmere til den cirkle pilehovedene er, jo mere indflydelse den variable har i de principal components.

```
library(factoextra)
fviz_pca_var(pca_fit, col.var="steelblue", repel = TRUE) +
  theme_minimal()
```



## 10.5 Problemstillinger

**Problem 1)** Quiz på Absalon

Download følgende datasæt ved at køre følgende kode chunk:

```
cancer <- read.csv(url("https://www.dropbox.com/s/4qa37itw9wtwtjg/breast-cancer.csv?dl=1")) %>%
cancer %>% glimpse()
```

```
## Rows: 569
## Columns: 31
## $ diagnosis          <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M", "~"
## $ radius_mean        <dbl> 17.990, 20.570, 19.690, 11.420, 20.290, 12.450~
## $ texture_mean       <dbl> 10.38, 17.77, 21.25, 20.38, 14.34, 15.70, 19.9~
```

```

## $ perimeter_mean          <dbl> 122.80, 132.90, 130.00, 77.58, 135.10, 82.57, ~
## $ area_mean               <dbl> 1001.0, 1326.0, 1203.0, 386.1, 1297.0, 477.1, ~
## $ smoothness_mean         <dbl> 0.11840, 0.08474, 0.10960, 0.14250, 0.10030, 0-
## $ compactness_mean        <dbl> 0.27760, 0.07864, 0.15990, 0.28390, 0.13280, 0-
## $ concavity_mean          <dbl> 0.30010, 0.08690, 0.19740, 0.24140, 0.19800, 0-
## $ concave.points_mean     <dbl> 0.14710, 0.07017, 0.12790, 0.10520, 0.10430, 0-
## $ symmetry_mean           <dbl> 0.2419, 0.1812, 0.2069, 0.2597, 0.1809, 0.2087-
## $ fractal_dimension_mean   <dbl> 0.07871, 0.05667, 0.05999, 0.09744, 0.05883, 0-
## $ radius_se                <dbl> 1.0950, 0.5435, 0.7456, 0.4956, 0.7572, 0.3345-
## $ texture_se                <dbl> 0.9053, 0.7339, 0.7869, 1.1560, 0.7813, 0.8902-
## $ perimeter_se              <dbl> 8.589, 3.398, 4.585, 3.445, 5.438, 2.217, 3.18-
## $ area_se                   <dbl> 153.40, 74.08, 94.03, 27.23, 94.44, 27.19, 53.-
## $ smoothness_se             <dbl> 0.006399, 0.005225, 0.006150, 0.009110, 0.0114-
## $ compactness_se            <dbl> 0.049040, 0.013080, 0.040060, 0.074580, 0.0246-
## $ concavity_se              <dbl> 0.05373, 0.01860, 0.03832, 0.05661, 0.05688, 0-
## $ concave.points_se         <dbl> 0.015870, 0.013400, 0.020580, 0.018670, 0.0188-
## $ symmetry_se               <dbl> 0.03003, 0.01389, 0.02250, 0.05963, 0.01756, 0-
## $ fractal_dimension_se      <dbl> 0.006193, 0.003532, 0.004571, 0.009208, 0.0051-
## $ radius_worst              <dbl> 25.38, 24.99, 23.57, 14.91, 22.54, 15.47, 22.8-
## $ texture_worst              <dbl> 17.33, 23.41, 25.53, 26.50, 16.67, 23.75, 27.6-
## $ perimeter_worst            <dbl> 184.60, 158.80, 152.50, 98.87, 152.20, 103.40, ~
## $ area_worst                 <dbl> 2019.0, 1956.0, 1709.0, 567.7, 1575.0, 741.6, ~
## $ smoothness_worst           <dbl> 0.1622, 0.1238, 0.1444, 0.2098, 0.1374, 0.1791-
## $ compactness_worst          <dbl> 0.6656, 0.1866, 0.4245, 0.8663, 0.2050, 0.5249-
## $ concavity_worst            <dbl> 0.71190, 0.24160, 0.45040, 0.68690, 0.40000, 0-
## $ concave.points_worst       <dbl> 0.26540, 0.18600, 0.24300, 0.25750, 0.16250, 0-
## $ symmetry_worst              <dbl> 0.4601, 0.2750, 0.3613, 0.6638, 0.2364, 0.3985-
## $ fractal_dimension_worst    <dbl> 0.11890, 0.08902, 0.08758, 0.17300, 0.07678, 0-

```

**Problem 2)** Anvend funktionen `ggscatmat` fra pakken `GGally` til at lave et plot hvor man sammenligne fem af de variabler.

- Man kan lave en tilfældig sample af fem variabler med at angive `columns = sample(2:31,5)` indenfor funktionen `ggscatmat`(husk at installere `GGally`-pakken).
- Give farver efter factor variablen `diagnosis` og vælger “pearson” som `corMethod`.
- Opfatter du, at der er en del redundans i datasættet (dvs. er der stærke korrelationer mellem de forskellige variabler)?

**Problem 3)** Benyt funktionen `prcomp` til at beregne en principal component analysis af datasættet.

- Husk at det skal kun være numeriske variabler og angiv `scale=TRUE` indenfor selve funktion.
- Lav et `summary` af resultaterne. Hvad er proportionen af variansen, som

- er forklaret af den første principal component?
- Hvad er proportionen af variansen, som er forklaret af de første to principal components tilsammen?
- 

**Problem 4)** *Augment og plot* Anvend `augment` til at tilføje dit rå datasæt til ovenstående resultater fra `prcomp`.

- Brug den til at lave et scatter plot af de første to principal components
  - Giv farver efter `diagnosis`
  - Skriv kort om man kan skelne imellem "M" og "B" fra variablen `diagnosis` ud fra de første to principal components.
- 

**Problem 5)** *Integrere kmeans clustering.* Lav et clustering med `kmeans` på datasættet, med to clusters (husk at udvælge numeriske variabler og scale inden du anvender funktionen `kmeans`).

- Augment resultaterne af `kmeans` til dit datasæt, der allerede har `prcomp` resultater tilføjet.
  - Lav et plot og give farver efter `.cluster` og former efter `diagnosis`.
  - Sammenligne dine to clusters med `diagnosis`.
- 

**Problem 6)** *tidy form og variansen* Anvende `tidy(matrix = "eigenvalues")` på din PCA resultater til at få bidraget af de forskellige components til den overordnet varianse i datasættet.

- Lav et barplot som viser de components på x-aksen og `percent` på y-aksen.
- 

**Problem 7)** *tidy form og rotation matrix* Anvende `tidy(matrix = "rotation")` til at få den rotation matrix.

- Anvend funktionen `pivot_wider` til at få den til wide form
  - Lav et scatter plot som viser de forskellige variabler relativ til hinanden
  - Anvend `geom_text_repel` til at give labels til de variabler (kan være en god idé at anvend `show.legend=F`)
- 

**Problem 8)** *Ekstra Udvidelse af Problem 5):* Fra din augmented resultater med både dine principal components og clusters: Beregne middelværdierne af din første to principal components for hver af de to clusters. Tilføj dine beregnede middelværdierne til plottet som "x".

## 10.6 Ekstra læsning

Step by step explanation: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>

PCA tidyverse style fra claus wilke: <https://clauswilke.com/blog/2020/09/07/pca-tidyverse-style/>

More PCA in tidyverse framework: <https://tbradley1013.github.io/2018/02/01/pca-in-a-tidy-verse-framework/>

# Chapter 11

## Emner fra eksperimental design

```
library(tidyverse)
library(broom)
```

“Amatører sidder og venter på inspiration, resten af os står bare op og går på arbejde.” - Stephen King

### 11.1 Inledning og læringsmålene

#### 11.1.1 Læringsmålene

I skal være i stand til at

- Beskrive randomisation, replication and blocking
- Beskrive Simpson’s paradox
- Beskrive Anscombes quartet
- Tjekke efter batch effects med PCA

#### 11.1.2 Inledning til chapter

Formålet med dette kapitel er at spørge: hvordan kan vi anvende værktøjerne som vi har lært i kurset til at kigge nærmere på forskellige emner i eksperimental design? Det er slet ikke en grundig introduktion til eksperimental design, men nogle nyttige og også interesseret emner som godt viser hvorfor det er vigtigt at lave hensigtsmæssige visualiseringer af dit data. Forståelsen af hvordan batch effekts påvirker en analyse er særlig vigtig indenfor biologi-fag, hvor mange store sekvensering projektor involverer data samlede eller sekvenseret over forskellige batches, sekvensering maskiner eller forskellige forberedelsesmetoder.

### 11.1.3 Video ressourcer

- Part 1: randomisation, replikation, blocking + confounding
    - Ingen video: læs gerne notaterne nedenfor
- 

- Part 2: Simpson's paradox

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556581563>

---

- Part 3: Anscombe's quartet

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556581540>

---

- Part 4: Batch effects and principal component analysis.

*OBS Man kan selvfølgelige også anvende map\_if() til at log-transformere.*

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556581521>

---

## 11.2 Grundlæggende principper i eksperimental design

### 11.2.1 Randomisation and replication

Man laver et **eksperiment** for at få svar på et bestemt spørgsmål, eller hypotese. Og man designer eksperimentet ud fra principper som gøre det gyldigt at fortolke resultaterne fra analysen af datasættet bagefter. For et eksperiment at være gyldigt skal det kunne demonstrere hensigtsmæssige **replikation** og **randomisation**.

#### Randomisation

Man vil gerne udelukke, at konklusionerne kan bare skyldes variansen pga. en faktor som ikke direkte er interessant i eksperimentet. Derfor bruger man randomisation til at få disse faktorer fordele over de forskellige treatment grupper. Et eksempel kan være 'double-blinding' i kliniske eksperimenter - både lægen og patienten har ikke kenskab til, hvem der hører til de forskellige grupper, og så kan man undgå forskelsbehandling indenfor grupperne, som kan påvirke de endelige resultater.

#### Replikation

Når man gentager et eksperiment flere gange - for eksempel ved at have flere patienter i hver treatment gruppe. Det tillader os at kunne beregne variabiliteten i data, som er nødvendig for at konkludere om der er en forskel mellem de grupper. Man kan altså ikke generalisere resultater som er blevet målt på kun én person.

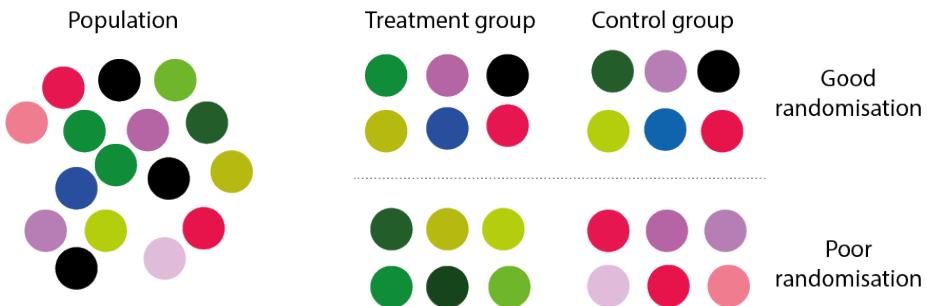
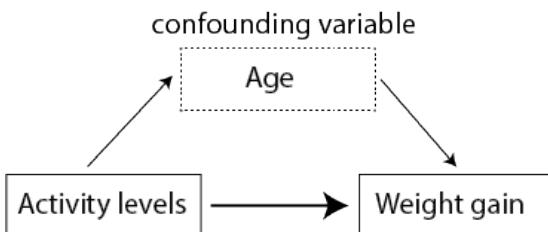


Figure 11.1: randomisation og replikation

I ovenstående figur er der 6 replikater i hver gruppe, som er enten "treatment" eller "control". I tilfældet "Good randomisation" er genstande som er taget tilfældige fra populationen vel matchet mellem de to grupper, mens i andet tilfældet "Poor randomisation", kan man se, at farverne af genstandene er vel matchet indenfor samme grupper. Det gøre det derfor umuligt at fortæl, om en eventuelle forskel mellem "treatment" og "control" er i virkeligheden resultatet af farven i stedet for målingerne, at man gerne vil sammenligne.

### Confounding

Figuren nedenfor illustrerer age som confounding variabel i et eksperiment hvor man prøver at forstå sammenhæng mellem aktivitet niveau og vægtøgning. Det kan være, at det ser umiddelbart ud til at være, at et lavt aktivitetsniveau (afhængig variabel) forklarer vægtøgning (unaghængig variabel), men man er nødt til at tage ændre variabler i betragtning for at sikre, at sammenhængen ikke skyldes noget andet. For eksempel, gruppen med det højt aktivitetsniveau kunne bestå af personer, der er yngre end personerne i gruppen med det lavt aktivitetsniveau, og deres alder kan påvirke deres vægtøgning (måske på grund af forskellige heder i stress niveauer, kost osv.).



### Blocking

Man kan prøve at kontrollere for ekstra variabler som vi ikke er interesseret i gennem “blocking”. Man laver “blocking” ved først at identificere grupper af individuelle som ligner hinanden så meget som muligt. Det kan være for eksempel at tre forskellige forsker var med til at lave et stort eksperiment med mange patienter og forskellige treatment grupper. Vi er interesseret i om der er forskellen mellem de treatment grupper, men ikke om der er en forskel mellem forskernes behandling af patienterne. Derfor vil vi gerne ‘block’ efter forsker - kontrollere for dem som en “batch” effect. Man kan også block efter f.eks. sex, for at sikre at forskellen i treatment grupper skyldes ikke forskelligheder mellem mænd og kvinder. Man laver “blocking” som del af en lineær model efter data er samlet, men det er nyttige at tænke over det fra starten.

### 11.2.2 Eksempel med datasættet ToothGrowth

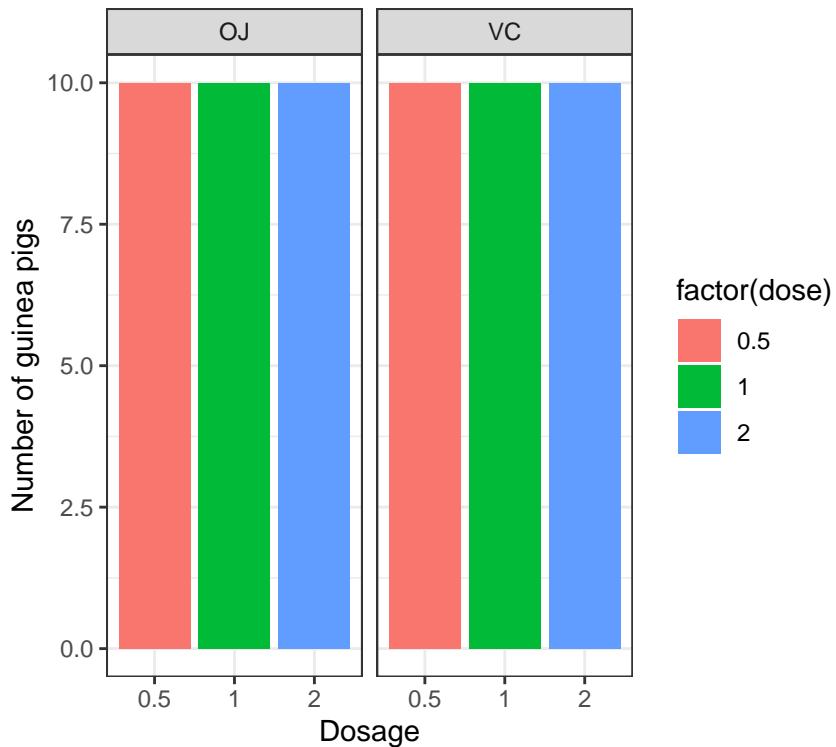
Et god eksempel på et godt eksperimental design er datasættet **ToothGrowth**, som er baserende på marsvin - de få forskellige kosttilskud og doser og så få de målte længden af deres tænder.

```
data(ToothGrowth)
ToothGrowth <- ToothGrowth %>% as_tibble() %>% mutate(dose = as.factor(dose))
summary(ToothGrowth)

##      len      supp      dose
##  Min.   : 4.20   OJ:30   0.5:20
##  1st Qu.:13.07  VC:30    1  :20
##  Median :19.25
##  Mean   :18.81
##  3rd Qu.:25.27
##  Max.   :33.90
```

Her kan man se, at for hver gruppe (efter **supp** og **dose**) er der 10 marsvin - vi har således replikation over de grupper, og hver **supp** (supplement) har hver af de tre mulige værdier for “dose”. Hvis vi for eksempel ikke var interesseret i **supp** men kun dose, så kan man ‘block’ efter **supp** for at afbøde forskelligheder i effekten af de to supplements i **supp**.

```
ToothGrowth %>% dplyr::count(supp,dose) %>%
  ggplot(aes(x=factor(dose),y=n,fill=factor(dose))) +
  geom_bar(stat="identity") +
  ylab("Number of guinea pigs") +
  xlab("Dosage") +
  facet_grid(~supp) +
  theme_bw()
```



Man må dog passe på, fordi vi vide ikke om, hvordan de marsvin blev tilknyttet til de forskellige grupper. For eksempel, hvis male og female marsvin er ikke tilknyttet ved tilfælde, kan det opstå, at supp "OJ" og dose "0.5" har kun male guinea pigs og supp "OJ" med dose "1.0" har kun female guninea pigs. Så kunne vi ikke fortæl, om forskellen i dose "0.5" vs "1.0" er resultatet af de dose eller kønnet.

### 11.3 Case studies: Simpson's paradox

(Se også videoressourcer Part 2).

Simpson's paradox opstår når man drager to modsætte konklusioner fra det samme datasæt - på den ene side når man kigger på de data samlede, og på den anden side når man tager nogle grupper i betragtning. Vi kan visualisere Simpson's paradox gennem eksemplet nedenfor - her har vi to variabler  $x$  og  $y$  som vi kan anvende til at lave et scatter plot, samt nogle forskellige grupper indenfor variablen  $group$ .

```
#library(datasauRus)
simpsons_paradox <- read.table("https://www.dropbox.com/s/ysh3qpc7qv0ceut/simpsons_paradox_groups.csv")
simpsons_paradox <- simpsons_paradox %>% as_tibble(simpsons_paradox)
simpsons_paradox
```

```

FALSE # A tibble: 222 x 3
FALSE     x     y group
FALSE   <dbl> <dbl> <chr>
FALSE 1 62.2 70.6 D
FALSE 2 52.3 14.7 B
FALSE 3 56.4 46.4 C
FALSE 4 66.8 66.2 D
FALSE 5 66.5 89.2 E
FALSE 6 62.4 91.5 E
FALSE 7 38.9 6.76 A
FALSE 8 39.4 63.1 C
FALSE 9 60.9 92.6 E
FALSE 10 56.6 45.8 C
FALSE # ... with 212 more rows

```

Hvis vi bare ignorere `group` og ser på de data samlet, kan vi se at der er en stærk positiv sammenhæng mellem `x` og `y`. Men når vi opdele efter de forskellige grupper, ved at skrive `colour = group`, få vi faktisk en negativ sammenhæng indenfor hver af de grupper.

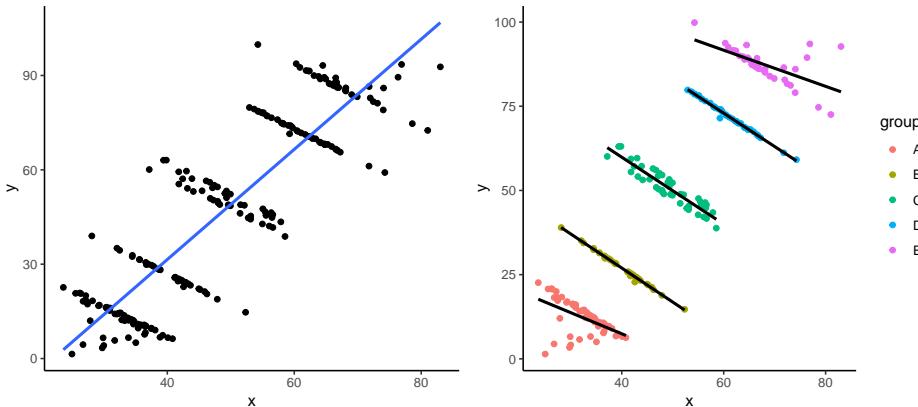
```

p1 <- simpsons_paradox %>%
  ggplot(aes(x,y)) +
  geom_point() +
  geom_smooth(method="lm",se=FALSE) +
  theme_classic()

p2 <- simpsons_paradox %>%
  ggplot(aes(x,y, colour=group)) +
  geom_point() +
  geom_smooth(method="lm", aes(group=group), colour="black", se=FALSE) +
  theme_classic()

library(gridExtra)
grid.arrange(p1,p2,ncol=2)

```



Simpson's paradox sker mere ofte end man skulle tror, og derfor er det vigtigt at tænke over hvad for nogen ændre variabler man også er nødt til at tage i betragtning.

### 11.3.1 Berkely admissions

Det mest berømte eksempel af Simpson's Paradox drejer sig om optagelsen til universitetet i Berkeley i 1973. Følgende tabel fra Wikipedia ([https://en.wikipedia.org/wiki/Simpson%27s\\_paradox](https://en.wikipedia.org/wiki/Simpson%27s_paradox)) viser statistikker om antallet som ansøgt samt procenttallet som blev optaget overordnet i universitet efter kønnet.

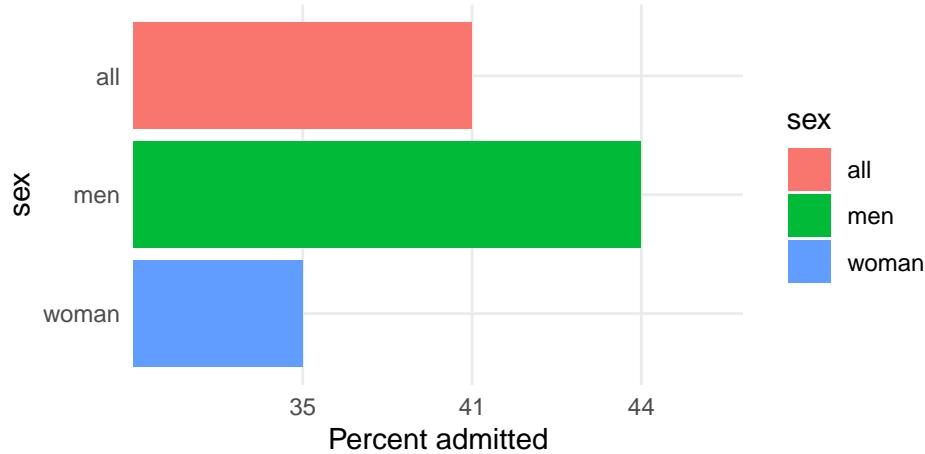
|       | All        |          | Men        |          | Women      |          |
|-------|------------|----------|------------|----------|------------|----------|
|       | Applicants | Admitted | Applicants | Admitted | Applicants | Admitted |
| Total | 12,763     | 41%      | 8442       | 44%      | 4321       | 35%      |

Figure 11.2: source: wikipedia

Hvis vi lave et barplot af tallerne, kan man se, at der er en højere procenttal af mænd end kvinder som var blevet optaget på universitet (sagen medførte en retsag mod universitetet).

```
admissions_all <- tibble("sex"=c("all","men","woman"),admitted=c("41","44","35"))

admissions_all %>% ggplot(aes(x=sex,y=admitted,fill=sex)) +
  geom_bar(stat="identity") +
  theme_minimal() +
  ylab("Percent admitted") +
  scale_x_discrete(limits = c("woman","men","all")) +
  coord_flip()
```

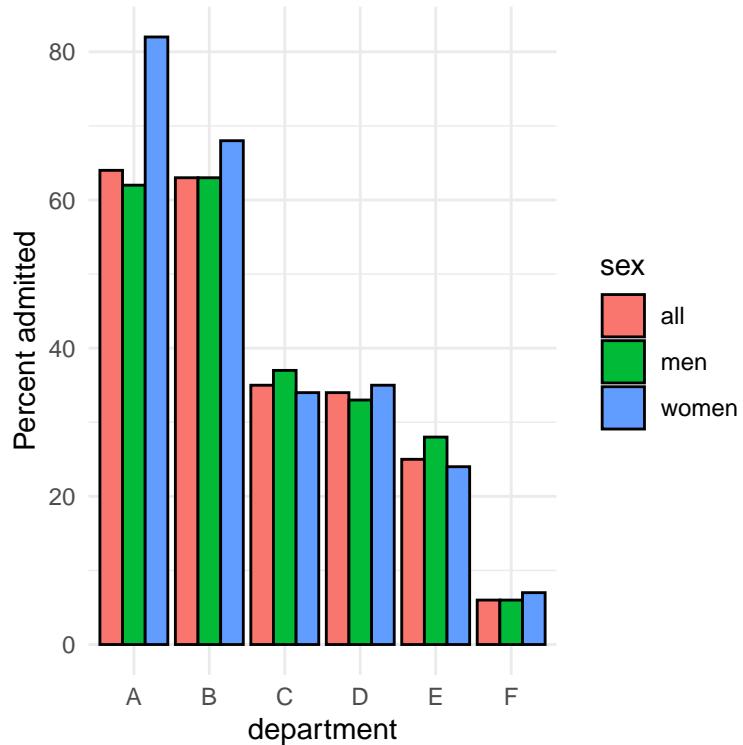


Da man dog kiggede lidt nærmere på de samme tal, men opdelte efter de forskellige afdelinger i universitet, fik man en anderledes billede af situationen. I følgende tabel har vi optagelsestallene for mænd og kvinder i hver af de forskellige afdelinger (A til F).

```
admissions_separate <- tribble(
  ~department, ~all, ~men, ~women,
  #-----/-----/-----/-----#
  "A",       64,     62,     82,
  "B",       63,     63,     68,
  "C",       35,     37,     34,
  "D",       34,     33,     35,
  "E",       25,     28,     24,
  "F",        6,      6,      7
)
```

Man kan man se, at for de fleste af afdelingerne, er der ikke en signifikant forskel mellem mænd og kvinder, og i nogle tilfælde havde kvinder faktisk en større chance for at blive optaget.

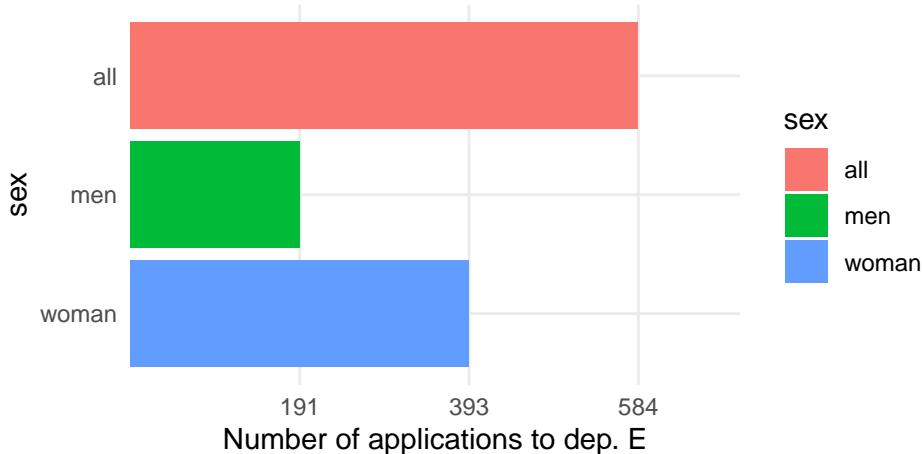
```
admissions_separate %>%
  pivot_longer(-department, names_to="sex", values_to="admitted") %>%
  ggplot(aes(x=department, y=admitted, fill=sex)) +
  ylab("Percent admitted") +
  geom_bar(stat="identity", position = "dodge", colour="black") +
  theme_minimal()
```



Hvad skyldes sammenhængen? Det viste sig, at kvinder havde en tendens til, at ansøge indenfor afdelingerne, som var sværreste at komme ind i. For eksempel, kan man se her, at department E har en relativt lav optagelses procent. Den samme afdeling var dog en af dem, hvor langt flere kvinder ansøgt end mænd.

```
applications_E <- tibble("sex"=c("all","men","woman"),applications=c("584","191","393"))

applications_E %>% ggplot(aes(x=sex,y=applications,fill=sex)) +
  geom_bar(stat="identity") +
  theme_minimal() +
  ylab("Number of applications to dep. E") +
  scale_x_discrete(limits = c("woman","men","all")) +
  coord_flip()
```



Derfor, selvom kvinder ikke havde en lavere sandsynlighed af at blive optaget end mænd i deres fortrukne fag, var antallet af kvinder der blev optaget i helhed over alle afdelinger faktisk lavere end antallet af mænd.

## 11.4 Case studies: Anscombe's quartet

(Se også videoressourcer Part 3).

Anscombes quartet (se også [https://en.wikipedia.org/wiki/Anscombe%27s\\_quartet](https://en.wikipedia.org/wiki/Anscombe%27s_quartet)) er et meget nyttigt og berømt eksempel, som stammer fra 1973, og som forklarer vigtigheden af at få lavet en visualisering af datasættet. Vi kan få åbnet de data fra linket nedenfor - man har x værdier og y værdier som kan anvendes til at lave et scatter plot, og der er også `set`, det refererer til fire forskellige datasæt (derfor 'quartet').

```
anscombe <- read.table("https://www.dropbox.com/s/mlt7crdik3eih9a/anscombe_long_format.csv")
anscombe <- as_tibble(anscombe)
anscombe
```

```
## # A tibble: 44 x 3
##       set     x     y
##   <int> <int> <dbl>
## 1     1     10  8.04
## 2     1      8  6.95
## 3     1     13  7.58
## 4     1      9  8.81
## 5     1     11  8.33
## 6     1     14  9.96
## 7     1      6  7.24
## 8     1      4  4.26
## 9     1     12 10.8
```

```
## # ... with 34 more rows
```

Formålet med datasættet er, at man gerne vil fitte en lineær regression model for at finde den forventet  $y$  afhængig om  $x$  (husk `lm(y ~ x)`). Da vi har fire datasæt, kan man således opdele datasættet efter `set` og benytter rammen `nest` og `map` (se Chapter 7) til at fitte de fire lineær regression modeller. Vi anvender også `tidy` og `glance` for at få summary statistikker fra de fire modeller:

```
my_func <- ~lm(y ~ x, data = .x)

tidy_anscombe_models <- anscombe %>%
  group_nest(set) %>%
  mutate(fit = map(data, my_func),
        tidy = map(fit, tidy),
        glance = map(fit, glance))
```

Man kan anvende `unnest` på den output fra `tidy` og kigge på den intercept og den slope af de fire modeller. Man kan se, at de to parametre er næsten identiske for de fire modeller:

```
tidy_anscombe_models %>% unnest("tidy") %>%
  pivot_wider(id_cols = "set", names_from = "term", values_from="estimate")
```

```
## # A tibble: 4 x 3
##       set `(Intercept)`      x
##   <int>     <dbl> <dbl>
## 1     1         3.00 0.500
## 2     2         3.00 0.5
## 3     3         3.00 0.500
## 4     4         3.00 0.500
```

Hvad med de andre parametre fra modellen - lad os kigge for eksempel på `r.squared` og `p.value` fra modellerne, som kan findes i output fra `glance`. Her kan vi igen se, at de er næsten identiske.

```
tidy_anscombe_models %>%
  unnest(cols = c(glance)) %>%
  select(set, r.squared,p.value)

## # A tibble: 4 x 3
##       set r.squared p.value
##   <int>     <dbl>    <dbl>
## 1     1     0.667 0.00217
## 2     2     0.666 0.00218
## 3     3     0.666 0.00218
## 4     4     0.667 0.00216
```

Hvad med korrelation? Også næsten den samme:

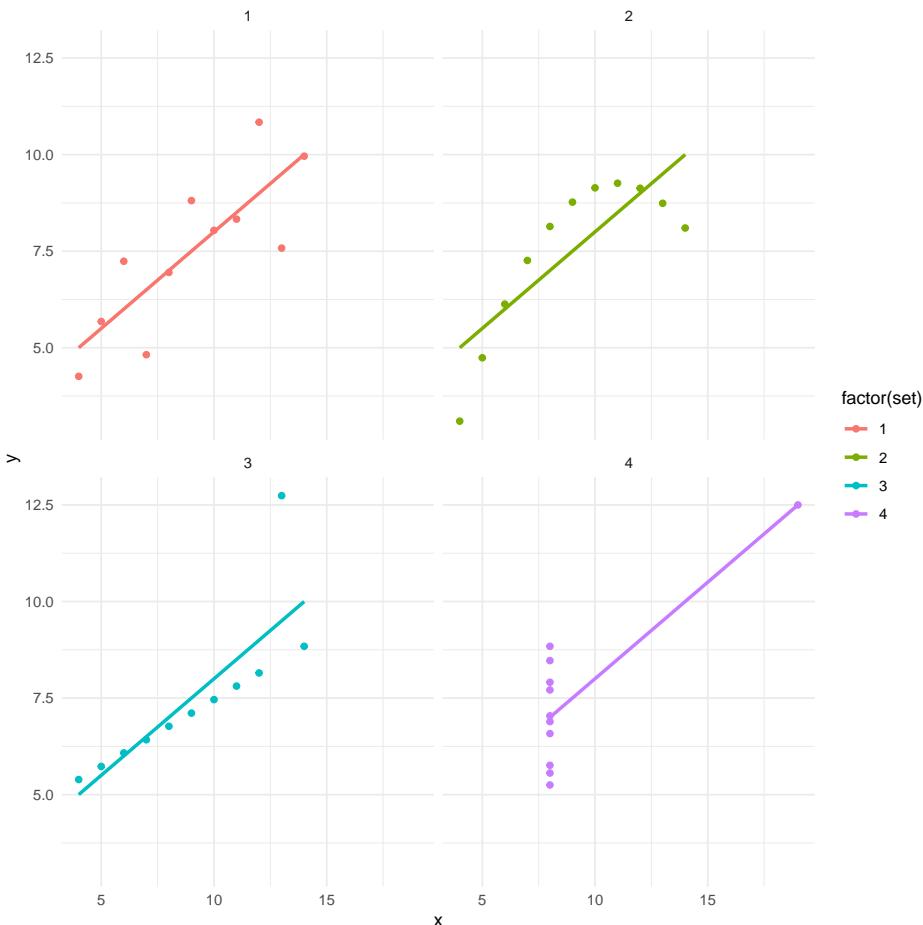
```
my_func <- ~cor(.x$x, .x$y)

anscombe %>%
  group_nest(set) %>%
  mutate(cor = map(data, my_func)) %>%
  unnest(cor) %>%
  select(-data)
```

```
## # A tibble: 4 x 2
##       set     cor
##   <int> <dbl>
## 1     1  0.816
## 2     2  0.816
## 3     3  0.816
## 4     4  0.817
```

Kan man så konkludere, at de fire datasæt som underligger de forskellige modeller, er identiske? Vi laver et scatter plot af de fire datasæt (som vi faktisk skulle have gjort i starten af vores analyse).

```
anscombe %>%
  ggplot(aes(x = x, y = y, colour=factor(set))) +
  geom_point() +
  facet_wrap(~set) +
  geom_smooth(method = "lm", se = FALSE) +
  theme_minimal()
```



De fire datasæt er meget forskellige. Vi ved, at de har alle samme de samme  
beste rette linjer, men underliggende data er slet ikke de samme. Den første  
datasæt ser egnert til en lineær regression analyse men vi kan se i datasæt nummer to, at der ikke engang er en lineær sammenhæng. Og de andre to har outlier  
værdier, som gør, at den bedste rette linje passer ikke særlig godt til punkterne.

## 11.5 Undersøgelse af “batch-effects”

(Se også videoressourcer Part 4).

Man kan også anvende visualiseringer til at kigge lidt nærmere på eventuelle batch effekter eller confounding variabler i datasættet. Det er især vigtigt i store eksperimenter, hvor forskellige samples eller dele af datasættet bliver samlede på forskellige tidspunkter, lokationer, eller af forskellige personer. Det er ofte tilfældet i sekvensering-baserede datasæt, at man ser batch effects, og det kan skyldes mange ting, bla.:

- Sekvensering dybelse
- Grupper samples lavet på forskellige tidspunkter af forskellige individuelle
- Sekvensering maskiner - samples sekvenserne på forskellige maskiner eller ‘lanes’.

Lad os tage udgangspunkt i nogle geneekspression sekvensering data lavet i mus (vi så også samme datasæt når vi lært `pivot_longer` kombineret med `left_join`).

```
norm.cts <- read.table("https://www.dropbox.com/s/3vhwnsnhzsy35nd/bottomly_count_table")
coldata <- read.table("https://www.dropbox.com/s/e13sm9ncvzbq6xf/bottomly_phenodata.txt")
coldata <- as_tibble(coldata)
norm.cts <- as_tibble(norm.cts, rownames = "gene")
coldata <- as_tibble(coldata)
```

Jeg begynder ved at vælge kun rækker som har mindst 50 counts, for at undgå gener med lave ekspressionsniveauer. Det næste jeg gør er at transformere de data til log form (funktion `map_if()`) for at opnå en bedre fordeling i datasættet.

```
#normalise and filter the data
norm.cts <- norm.cts %>%
  filter(rowSums(norm.cts) %>% select(-gene)) > 50) %>%
  map_if(is.numeric, ~log(.x + 1)) %>% as_tibble()
```

```
norm.cts
```

```
## # A tibble: 10,193 x 22
##   gene    SRX033480 SRX033488 SRX033481 SRX033489 SRX033482 SRX033490 SRX033483
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 ENSMUS~    6.35      6.32      6.21      6.29      6.31      6.27      6.30
## 2 ENSMUS~    3.51      3.56      3.57      3.27      2.99      3.61      3.56
## 3 ENSMUS~    3.19      3.50      3.08      3.21      3.14      3.09      3.22
## 4 ENSMUS~    6.69      6.48      6.38      6.35      6.39      6.34      6.50
## 5 ENSMUS~    6.05      6.37      6.17      6.26      6.16      6.06      6.13
## 6 ENSMUS~    2.89      2.94      3.16      3.21      3.77      3.30      3.11
## 7 ENSMUS~    3.42      3.12      3.86      4.36      3.77      3.99      4.24
## 8 ENSMUS~    3.42      2.94      3.52      3.41      3.57      3.60      2.99
## 9 ENSMUS~    5.02      4.98      4.49      4.27      4.67      4.35      4.81
## 10 ENSMUS~   5.13      4.88      4.97      4.76      4.82      4.79      4.96
## # ... with 10,183 more rows, and 14 more variables: SRX033476 <dbl>,
## #   SRX033478 <dbl>, SRX033479 <dbl>, SRX033472 <dbl>, SRX033473 <dbl>,
## #   SRX033474 <dbl>, SRX033475 <dbl>, SRX033491 <dbl>, SRX033484 <dbl>,
## #   SRX033492 <dbl>, SRX033485 <dbl>, SRX033493 <dbl>, SRX033486 <dbl>,
## #   SRX033494 <dbl>
```

Så der er omkring 10,000 genes i rækkerne og så 21 forskellige samples som spredet sig over kolonnerne. Vi har også nogle sample oplysninger - der er to forskellige strains af mus og også forskellige batches, som vi gerne vil kigge nærmere

på.

```
coldata
```

```
## # A tibble: 21 x 5
##   column    num.tech.reps strain  batch lane.number
##   <chr>        <int> <chr>    <int>      <int>
## 1 SRX033480          1 C57BL.6J     6         1
## 2 SRX033488          1 C57BL.6J     7         1
## 3 SRX033481          1 C57BL.6J     6         2
## 4 SRX033489          1 C57BL.6J     7         2
## 5 SRX033482          1 C57BL.6J     6         3
## 6 SRX033490          1 C57BL.6J     7         3
## 7 SRX033483          1 C57BL.6J     6         5
## 8 SRX033476          1 C57BL.6J     4         6
## 9 SRX033478          1 C57BL.6J     4         7
## 10 SRX033479         1 C57BL.6J    4         8
## # ... with 11 more rows
```

Vi kan kigge på hvor mange samples efter hver kombination af stain og batch vi har i datasættet:

```
table(coldata$strain, coldata$batch)
```

```
##
##           4 6 7
##   C57BL.6J 3 4 3
##   DBA.2J   4 3 4
```

Så kan man se, at både strain har repræsenteret tre eller fire samples i hver af de tre batches. Der er derfor *replication* og da vi har fået repræsenteret hver kombination af strain og batch, kan man eventuelle **block** efter batch for at få fjernet dens effekt. Her har vi ikke tid til at kigge på metoder for at fjerne batch-effekts, men det er vigtigt at vi er i stand til at opdage dem.

### 11.5.1 Principal component tilgang

Man kan undersøge mulige batch effekter via prinicpal component analysis.

```
pca_fit <- norm.cts %>%
  select(where(is.numeric)) %>% # retain only numeric columns
  prcomp(scale = TRUE) # do PCA on scaled data
```

Husk at når man lave en principal compononent analysis, kan man får den rotation matrix, der anvendes til at se hvor de forskellige samples ligger relative til hinanden over de forskellige principal components - dvs. at samples der ligner hinanden vises på samme sted på plottet. Den rotation matrix udtrækkes med funktionen `tidy()`:

```
rot_matrix <- pca_fit %>%
  tidy(matrix = "rotation")
```

Vi vil gerne lave et plot af de rotation matrix, men første vil vi gerne tilføje de sample oplysninger med `left_join`, så at vi kan se de forskellige batches eller strains. Både dataframes har en kolon som hedder `column`, der refereres til de sample navne, så jeg forbinde efter `column` her.

```
rot_matrix <- rot_matrix %>%
  left_join(coldata, by = "column")
```

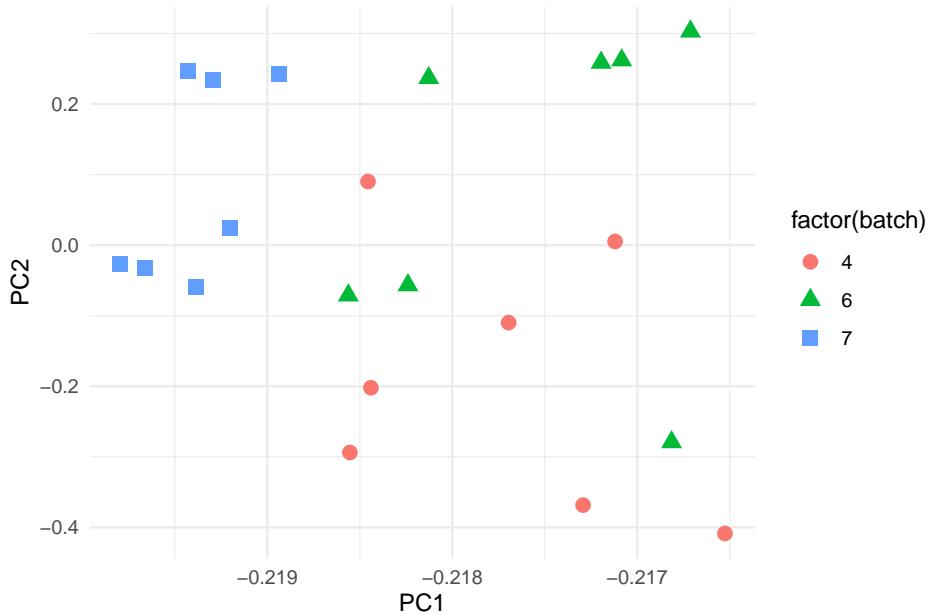
Anvende `pivot_wider()` til at få den i wide form, så vi kan plotte “PC1” og “PC2” i et scatter plot:

```
rot_matrix_wide <- rot_matrix %>%
  pivot_wider(names_from = "PC", names_prefix = "PC", values_from = "value")
rot_matrix_wide
```

```
## # A tibble: 21 x 26
##   column num.tech.reps strain batch lane.number    PC1      PC2      PC3      PC4
##   <chr>          <int> <chr>  <int>        <int>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 SRX03~           1 C57BL~     6       1 -0.217  0.262  0.0200 -0.508
## 2 SRX03~           1 C57BL~     7       1 -0.219  0.243  0.00586 0.166
## 3 SRX03~           1 C57BL~     6       2 -0.217  0.303  0.0153 -0.237
## 4 SRX03~           1 C57BL~     7       2 -0.219  0.246  0.00799 0.178
## 5 SRX03~           1 C57BL~     6       3 -0.217  0.259  0.0974 -0.113
## 6 SRX03~           1 C57BL~     7       3 -0.219  0.234  0.00385 0.209
## 7 SRX03~           1 C57BL~     6       5 -0.218  0.237  0.0303 -0.179
## 8 SRX03~           1 C57BL~     4       6 -0.217  0.00524 0.463  0.376
## 9 SRX03~           1 C57BL~     4       7 -0.218  0.0901 0.305  0.0651
## 10 SRX03~          1 C57BL~     4       8 -0.218 -0.110  0.434 -0.161
## # ... with 11 more rows, and 17 more variables: PC5 <dbl>, PC6 <dbl>,
## #   PC7 <dbl>, PC8 <dbl>, PC9 <dbl>, PC10 <dbl>, PC11 <dbl>, PC12 <dbl>,
## #   PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>, PC18 <dbl>,
## #   PC19 <dbl>, PC20 <dbl>, PC21 <dbl>
```

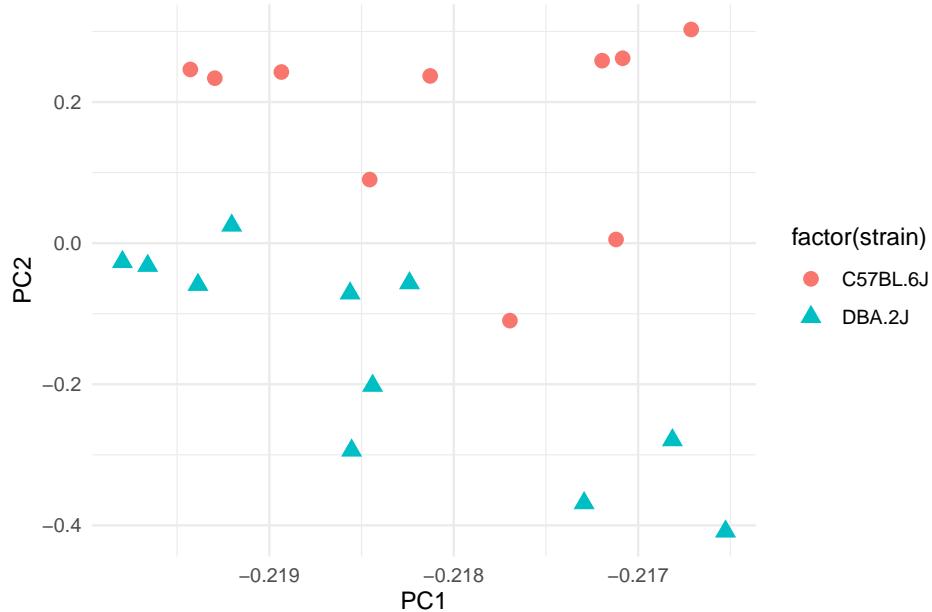
Jeg giver farver og former efter de tre batches. Man kan se, at jeg har fået alle samples fra batch nummer 2 på samme sted i plottet.

```
rot_matrix_wide %>%
  ggplot(aes(PC1, PC2, shape = factor(batch), colour = factor(batch))) +
  geom_point(size = 3) +
  theme_minimal()
```



Jeg kan også give farver efter strain, hvor man kan se at der nok er en forskellen mellem de to strains her.

```
rot_matrix_wide %>%
  ggplot(aes(PC1,PC2,shape=factor(strain),colour=factor(strain))) +
  geom_point(size=3) +
  theme_minimal()
```

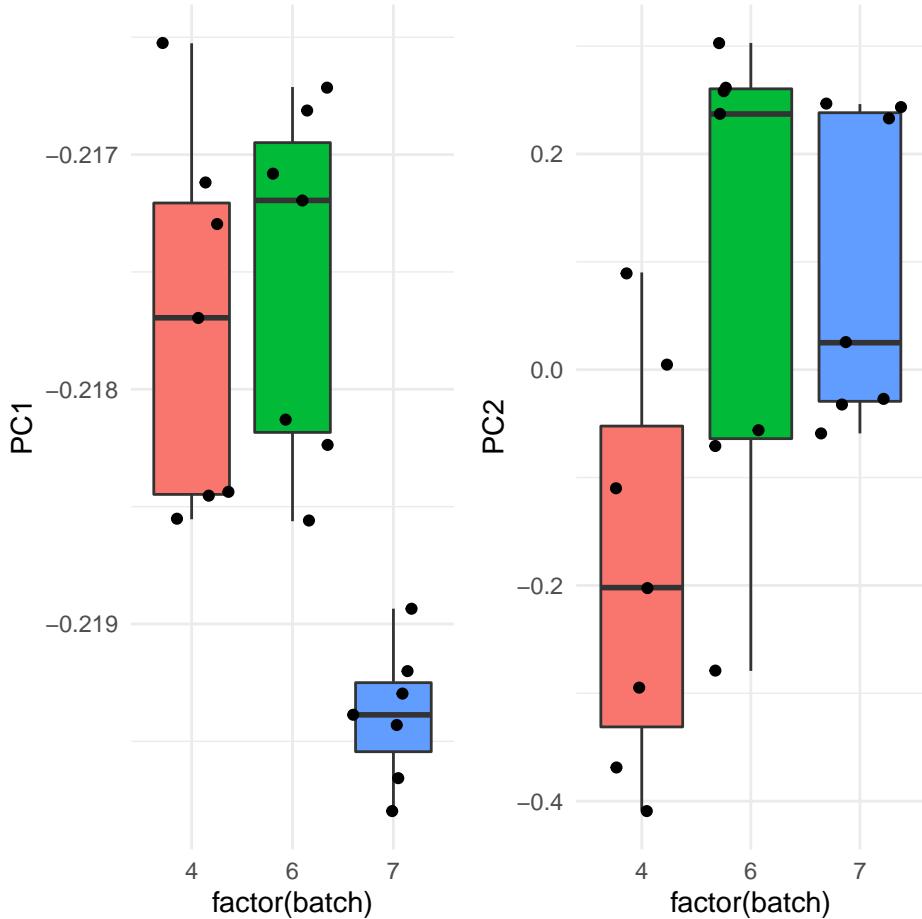


Man kan også se de data på en anden måde ved at lave boxplots for to første to principal compononets opdelte efter batch. Vi få bekræftet vores observation at der er en stærk forskel mellem match 7 og de andre to batches langt den første principal component, og det er et problem som muligvis skal korrigeres før man laver yderligere analyser på de data.

```
p1 <- rot_matrix_wide %>%
  ggplot(aes(x=factor(batch), y=PC1, fill=factor(batch))) + geom_boxplot(show.legend = FALSE)

p2 <- rot_matrix_wide %>%
  ggplot(aes(x=factor(batch), y=PC2, fill=factor(batch))) + geom_boxplot(show.legend = FALSE)

library(gridExtra)
grid.arrange(p1,p2, ncol=2)
```



## 11.6 Problemstillinger

**Problem 1)** Quiz på Absalon - experimental

**Problem 2)** Eksperimental design

Jeg laver et eksperiment hvor patienter få en af tre forskellige kosttilskud (Gruppe 1, 2 og 3). Der er 5 patienter i hver gruppe og jeg vil gerne se, om patienternes energi niveau i gennemsnit er forskellige mellem de tre grupper. Alderne af patienterne i hver af de tre grupper er:

Gruppe 1: 18, 23, 31, 25, 19

Gruppe 2: 24, 29, 35, 21, 30

Gruppe 3: 43, 52, 33, 39, 40

- Hvad er problemet her med det eksperimental design? Lav boxplots for at viser fordelingen af alderne for hver af de tre grupper.
  - Hvis man finder en signifikant forskel mellem de tre kosttilskud, kan man stoler på resultaterne?
  - Hvad andre variabler end alder kunne skyldes en eventuelle forskel mellem de tre kosttilskud, og som måske skulle tages i betragtning?
  - Hvad kan man gøre med den ovenstående eksperiment design for at løse problemet?
- 

**Problem 3) Simpson's paradoks Lung Cap data revisited**

Indlæse LungCapData og tilføj kategorisk variabel Age.Group:

```
LungCapData <- read.csv("https://www.dropbox.com/s/ke27fs5d37ks1hm/LungCapData.csv?dl=1")
LungCapData$Age.Group <- cut(LungCapData$Age, breaks=c(1, 13, 15, 17, 19), right=FALSE, include.lowest=TRUE)
levels(LungCapData$Age.Group) <- c("<13", "13-14", "15-16", "17+")
```

- a) Lav boxplots med smoke på x-aksen og LungCap på y-aksen. + Notere hvilke gruppe har den højeste lungkapacitet.
  - b) Lav samme plot men adskilte efter Age.Group og beskriv, hvordan det er et eksempel på Simpson's Paradoks.
  - c) Lav et boxplot med Age på y-aksen og Smoke på x-aksen for at støtte hvorfor man ser Simpson's Paradoks i datasættet.
- 

**Problem 4) Anscombes analyse** Gentage Anscombes analyse med dinosaurus datasæt:

```
library(datasauRus) #installer denne pakke
data_dozen <- datasauRus::datasaurus_dozen
data_dozen
```

```
FALSE # A tibble: 1,846 x 3
FALSE   dataset    x     y
FALSE   <chr>    <dbl> <dbl>
FALSE   1 dino    55.4  97.2
FALSE   2 dino    51.5  96.0
FALSE   3 dino    46.2  94.5
FALSE   4 dino    42.8  91.4
FALSE   5 dino    40.8  88.3
FALSE   6 dino    38.7  84.9
FALSE   7 dino    35.6  79.9
FALSE   8 dino    33.1  77.6
FALSE   9 dino    29.0  74.5
FALSE  10 dino   26.2  71.4
FALSE # ... with 1,836 more rows
```

- a) Fit en lineær regression model for hver af de datasæt (anvende `group_by`, `nest` og `map` ramme med en custom funktion), hvor man har `y` som afhængig variabel og `x` som uafhængig variabel. Anvend også `tidy` og `glance` på samtlige modeller
- b) Anvend resultaterne fra `tidy` til at kigge på den slope og intercept for de forskellige modeller - ligner de hinanden?
- c) Anvende også resultaterne fra `glance` til at kigge på `r.squared` og `p.value`.
- d) Er de alle det samme datasæt? Lav et scatter plot adskilte efter de forskellige datasæt. Hvordan ser de bedste rette linjer ud på plotterne?
- 

**Problem 5)** Vi vil gerne undersøge eventuelle batch effects i følgende datasæt. Det er simuleret “single cell” sekvensering count data (dataframen `cse50`) samt dataframen `batches`, som angiver hvilken batch hver af de 500 cells tilhører.

```
cse50 <- read.table("https://www.dropbox.com/s/o0wz0jpcsekeg6z/cell_mix_50_counts.txt?dl=1")
batches <- read.table("https://www.dropbox.com/s/4t382bfgro46ka5/cell_mix_50_batches.txt?dl=1")
batches <- as_tibble(batches)
cse50 <- as_tibble(cse50)
```

- a) Anvend `map_df` til at få transformerede data til log scale (plus 1 først og tag log bagefter).
- b) Lav PCA på det transformerede datasæt.
- c) Anvend din PCA resultater til at få den rotation matrix
- d) Forbinde oplysningerne fra dataframen `batches` til din rotation matrix (første tilføj en ny kolonne “column” til `batches` som er lig med `names(cse50)`).
- e) Anvend `pivot_wider` og lav et plot af den første to principal components, hvor du angiver farve efter `batch`.
- f) Lav også boxplots for de første to principal components fordelt efter `batch` og kommenter kort på eventuelle batch effekts i de data.
- 

### Problem 6 Mere Simpson’s Paradox

Kør følgende kode til at indlæs og bearbejde følgende datasæt `airlines`.

```
airlines <- read.table("http://www.utsc.utoronto.ca/~butler/d29/airlines.txt", header=T)

airlines <- airlines %>%
  pivot_longer(-airport) %>%
  separate(name, sep = "_", into = c("airline", "status")) %>%
  mutate(airline = recode(airline, aa = "Alaska", aw = "American")) %>%
  pivot_wider(names_from=status, values_from=value) %>%
```

```
mutate("ontime" = ontime + delayed) %>%
  rename(flights = ontime)
```

- a) Opsummer antal flights og antal delayed over de forskellige airports for at få et samlet tal til hver airline.
- b) Beregn også proportionen af flights som er delayed i hver airline (igen samlede over alle airports). Lav et barplot til at vise proportionerne.
- c) Denne gange opsummer over de to airlines for at få et samlet tal til hvert airline. Beregen også proportionen af flights som er delayed og lave et plot.
- d) Denne gange beregne proportionen af flights som er delayed til hver kombination af både airport og airline. Igen omsæt til et plot.
- e) Kan du forklare? Hint: tag for eksempel en kig på de rå data og bla. airport "Phoenix".

## 11.7 Yderligere læsning

Simpson's paradox og airlines: <http://ritsokiguess.site/docs/2018/04/07/simpson-s-paradox-log-linear-modelling-and-the-tidyverse/>

Batch effekt correction: [https://en.wikipedia.org/wiki/Batch\\_effect#Correction](https://en.wikipedia.org/wiki/Batch_effect#Correction)

## Chapter 12

# Maskinslæring i tidyverse



```
library(rsample) #install this
library(tidyverse)
library(ranger) #install this, random forest
library(palmerpenguins)
```

“Absorber hvad der er nyttigt, afvis det, der er ubrugeligt, tilføj det, der specifikt er dit eget.” - Bruce Lee

### 12.1 Indledning og læringsmålene

Her bliver en lynhurtig introduktion til maskinslæring koncepter - emnet er i virkeligheden kæmpe store og jeg springer over mange detaljer her. Men jeg håber, at det giver en god grundlag/forståelse af emnet, der motiverer én til at tage yderligere kurser i maskinslæringområdet, eller at øge forståelsen de mange artikler i biologiske område der benytter maskinslæring modeller.

I emnet introducerer jeg den grundlæggende ramme hvor man udvikler en model på træning data og cross-fold-validation og dernæst vurderer den model på testing data, som ikke var en del af selve træningsprocessen. Derudover arbejder vi stort set kun indenfor for emnerne vi har dækket i kurset. Processen virker måske lidt indviklet i starten, men generaliseres meget og nemt kan udvides i mange kontekster og situationer, der kræver forskellige modeller. I emnet introducerer jeg også en classification metode, som vi både optimerer (“tune”)

og bruger til at forudsige hvem, der overlevede tradgedien på titanic!

---

Du skal være i stand til at

- Kende anvendelse af testing og training datasæt og beskrive cross-fold-validation processen
- Anvende 5-fold-cross validation til at udvikle en model og beregner statistikker til at evaluere modellen
- Lave en classification model for at forudsige hvem der overlevede titanic
- Lave “tuning” af en parameter og bestemme om en “final” model

## 12.2 Video ressourcer

- Video 1: Data splitting and cross-fold validation i en regression kontekst
- Video 2: Cross-fold validation i en classification kontekst (random forest)
- Video 3: Tuning af parametre i classification og variabel importance

## 12.3 Regression models

Vi starter med at bygge processen op i en kontekst som vi godt kender i forvejen - en lineær regression model. Til det formål benytter vi endnu engang datasættet `penguins`, og målet er: hvor godt kan man forudsige variablen `body_mass_g` i de forskellige pingviner?

```
library(palmerpenguins)
penguins <- penguins %>% drop_na()
```

### 12.3.1 Oprette testing and training sets

Hvordan kan man vide, om en model er god til at forudsige hvad der sker når der kommer nye observationer? Det vil sige, fungerer modellen godt kun i forhold til observationerne som blev brugt til at opbygge selve modellen (så kaldt “overfitting”), eller kan modellen generalisere til andre, lignende situationer? For at vide det kan man overveje at splitte datasættet før man lave modellen - det vil sige et datasæt som man bruger til at opbygge modellen, og et datasæt som man kan bruge som “nye observationer”, hvor man kan teste, hvor godt modellen egentlig er (fordi i datasættet kender vi faktisk resultatet som vi kan sammenligne med vores forudsigelser fra modellen).

Pakken `rsample` fra `tidymodels` har nogle nyttige funktioner som kan bruges i konteksten af maskinslæring i R, fk. funktionen `initial_split`:



Funktionen `initial_split` opdeler datasættet således at f. k. 75% af observationerne hører til det “training” sæt og 25% hører til det “testing” sæt:

```
penguins_split <- initial_split(penguins, prop = 0.75)
penguins_split
```

```
## <Analysis/Assess/Total>
## <249/84/333>
```

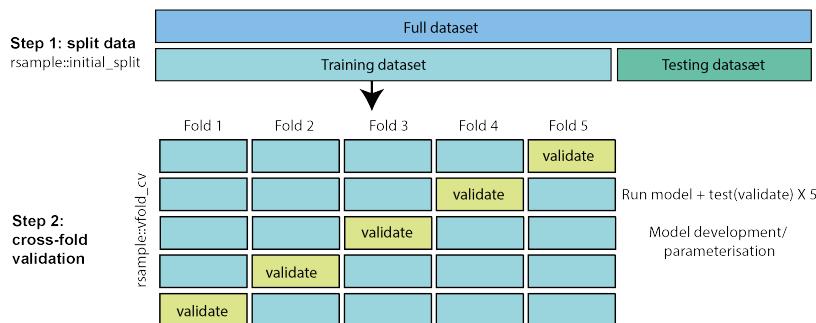
Det betyder, at ud af de 333 observationer, bliver 249 brugt som “training” og 84 er beholdt tilbage som en “testing” sæt. Jeg udtrækker bagefter de to sæt til at få “`penguins_training`” og “`penguins_test`” som i følgende:

```
penguins_training <- penguins_split %>% training()
penguins_test <- penguins_split %>% testing()
```

Jeg vil gerne anvende `penguins_training` til at opbygge modellen og så kun kigge på `penguins_test` senere for at evaluere modellens evne til at lave forudsigelser.

### 12.3.2 Cross-fold validation

For at udvikle min model med mit træning datasæt anvender jeg en proces, der hedder “cross-fold-validation”. I 5-fold-cross-validation splittet man et datasæt op i 5 stykker (“folds”). Man træner dernæst en model på 4 af de 5 stykker (folds), og tester på den sidste stykke (som ikke selv var en del af datasættet brugt til at fitte selve modellen). Man gentager model træning processen i alt fem gange, hvor hvert styk er brugt præcis én gang som “validation” sæt.



Tilgangen har mange fordele. Hvis der var for eksempel kun én testing sæt og ved tilfældighed har den sæt observationer som er især nemme at forudsige, så får vi en fejlagtig fornemmelse at modellen er bedre end den er i virkeligheden - vi undgår det vi at bruge 5 forskellige validation sæt som dækker hele træning sæt. Vi kan således lede efter en kommination af parametre, der konsekvent

giver en god model over samtlig validation sæt - vi kan være mere sikker på at få en model, der virker godt i vores testing sæt til sidste.

Der er en nem funktion man kan bruge i `rsample`-pakken, der hedder `vfold_cv` til at lave cross validation. Her angiver jeg `v=5` for at vise, jeg gerne vil have 5-fold cross validation.

```
cv_split <- vfold_cv(penguins_training, v = 5)
cv_split
```

```
## # 5-fold cross-validation
## # A tibble: 5 x 2
##   splits      id
##   <list>     <chr>
## 1 <split [199/50]> Fold1
## 2 <split [199/50]> Fold2
## 3 <split [199/50]> Fold3
## 4 <split [199/50]> Fold4
## 5 <split [200/49]> Fold5
```

Så kan man se, at vi får en dataframe med 5 datasæt som er lagret i kolonnen `splits`. Jeg vil gerne udtrække de forskellige træning og testing sæt ud i deres egne kolonner, som i følgende med `map` indenfor `mutate`:

```
cv_data <- cv_split %>%
  mutate(
    train = map(splits, ~.x %>% training), #extract training datasets
    validate = map(splits, ~.x %>% testing) #extract validation datasets
  )
cv_data

## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits      id   train      validate
##   <list>     <chr> <tibble [199 x 8]> <tibble [50 x 8]>
## 1 <split [199/50]> Fold1 <tibble [199 x 8]> <tibble [50 x 8]>
## 2 <split [199/50]> Fold2 <tibble [199 x 8]> <tibble [50 x 8]>
## 3 <split [199/50]> Fold3 <tibble [199 x 8]> <tibble [50 x 8]>
## 4 <split [199/50]> Fold4 <tibble [199 x 8]> <tibble [50 x 8]>
## 5 <split [200/49]> Fold5 <tibble [200 x 8]> <tibble [49 x 8]>
```

### 12.3.3 Bygge modellen og lave forudsigelser

Vi vil gerne lave en model på alle vores datasæt i kolonnen `train` hver for sig, og forudsigende `body_mass_g` i hvert tilknyttet validation sæt. Vi kan gøre det med en custom funktion med `purrr`-pakken.

```
my_lm_func <- ~lm(formula = body_mass_g ~ ., data = .x)
```

```
cv_models_lm <- cv_data %>%
  mutate(model = map(train, my_lm_func))

cv_models_lm

## # 5-fold cross-validation
## # A tibble: 5 x 5
##   splits      id    train      validate     model
##   <list>     <chr> <tibble`> <tibble`> <lm`>
## 1 <split [199/50]> Fold1 <tibble [199 x 8]> <tibble [50 x 8]> <lm>
## 2 <split [199/50]> Fold2 <tibble [199 x 8]> <tibble [50 x 8]> <lm>
## 3 <split [199/50]> Fold3 <tibble [199 x 8]> <tibble [50 x 8]> <lm>
## 4 <split [199/50]> Fold4 <tibble [199 x 8]> <tibble [50 x 8]> <lm>
## 5 <split [200/49]> Fold5 <tibble [200 x 8]> <tibble [49 x 8]> <lm>
```

Nu vil vi gerne bruge datasættet i kolonnen `validate` til at lave en forudsigelse til hver af vores modeller. Til det formål kan man bruge funktionen `predict()` - man tager en model samt et nye datasæt (dvs. tilsvarende datasæt i kolonne `validate`), og så få nogle forventede `body_mass_g` værdier ud fra modellen anvendte på de nye data. I følgende bliver disse forventede “predicted” værdier til `body_mass_g` lagret i kolonnen `validate_predicted`. Da vi faktisk vide de virkelige værdier til `body_mass_g` i vores validation datasæt laver jeg også en kolonne `validate_actual` som er disse virkelige værdier, som jeg kan sammenligne med mine forventede værdier.

```
cv_prep_lm <- cv_models_lm %>%
  mutate(
    actual_mass = map(validate, ~.x %>% pull(body_mass_g)), #actual body_mass_g values
    predicted_mass = map2(.x = model, .y = validate, ~predict(.x, .y)) #predicted body_mass_g va
  )
```

#### 12.3.4 Vurdere resultater på validation sæt

Jeg vil gerne sammenligne mine nye forudsigelser direkte med mine virkelige værdier til `body_mass_g`, hvilket kan give os en indikation på, hvor godt vores modeller egentlig er. I følgende tager jeg `id`, `actual_mass` og `predicted_mass` ud fra dataframen `cv_prep_lm` og gennem dem i en ny dataframe `prediction_df`:

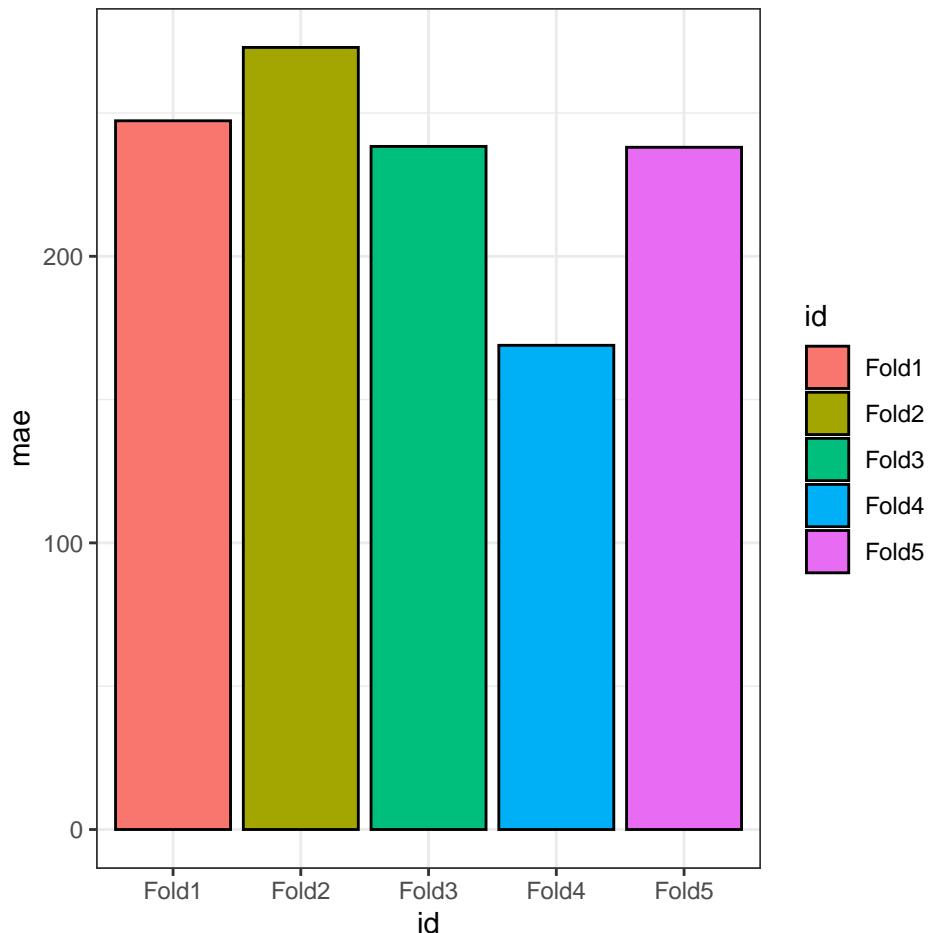
```
prediction_df <- cv_prep_lm %>% select(id,actual_mass,predicted_mass) %>% unnest(c(actual_mass,pr
```

Til hver af de fem folds bregener jeg statistiken “mean absolute error” der tager afstanden mellem de forventede værdier (`predicted_mass`) og virkelige værdier (`actual_mass`) og tager dernæst middelværdien (over deres absolute værdier dvs. uden minus tegn). Jo mindre “mae” er, jo bedre modellen ser ud.

```
#smaller mae = better model
mae_df <- prediction_df %>% group_by(id) %>% summarise("mae" = mean(abs(actual_mass-predicted_ma
```

Her er et plot af statistikken “mae” til de forskellige folds. I gennemsnit er det 233.0974471, der betyder at vores forudsigelser er ude/forkert ved knap 300 g i gennemsnit - ikke dårligt når den gennemsnits pingvin vejer 4207.06 g.

```
mae_df %>%
  ggplot(aes(y=mae,x=id,fill=id)) +
  geom_bar(stat="identity",colour="black") +
  theme_bw()
```



### 12.3.5 Teste og evaluere “final” model

Vi har lavet forudsigelser over de forskellige stykker i vores træning sæt, men vi har ikke endnu lavet forudsigelser på vores testing sæt `penguins_test`, som blev holdt ud af hele træningsprocessen. Efter den cross-validation proces skal man lave en “final” model, der bruger samtlige training data på én gang (dvs. ikke bare en enkel fold, der vi gerne vil opbygge den endelige model fra så meget

data, som muligt. Bemærk at når det er kun et datasæt, behøver vi ikke at bruge en custom funktion.

```
model_final <- lm(body_mass_g ~ ., data = penguins_training)
```

Nu kan vi endelige anvende vores testing data: her anvender jeg min “final model” til at forudsige `body_mass_g` på pingvinerne i datasættet `penguins_test`. Dernæst sammenligner vi vores forudsigelser med den virkelige værdier ligesom i ovenstående (men her har jeg ikke forskellige folds at tage i betragtning).

```
prediction_df <- tibble("actual_mass"=penguins_test %>% pull(body_mass_g),
                       "predicted_mass"=predict(model_final,penguins_test))

prediction_df %>% summarise("mae" = mean(abs(actual_mass-predicted_mass)))

## # A tibble: 1 x 1
##       mae
##   <dbl>
## 1 235.
```

Man kan se, at vores final model lave forudsigelser som er i gennemsnit 235.5g væk fra pingvinernes virkelige vægt (det vil sige, at forudsigelsen kan være over eller under den virkelige værdi, men i gennemsnit er afstanden fra den virkelige værdi 230g).

## 12.4 Classification models

I ovenstående proces lavede vi en regression model, hvor afhængig variabel er kontinuerlig (`body_mass_g`). I en classification model er den afhængig variabel kategorisk (fk. “yes” eller “no” i forhold til variablen `Survived` i datasættet `titanic`). Vi viser her at det er meget nemt at skifte fra den ene type model til en anden og bruger stort set samme kode (kun små ændringer).

Ovenstående ramme også giver os muligheden for at opbygge modeller, hvor der faktisk er forskellige parametre, der skal vælges. I ovenstående eksempel med penguins var der faktisk ikke meget “tuning” vi skulle lave for at udvikle modellen. Jeg viser her hvordan man kan “tune” en parameter indenfor samme ramme i vores classification model.

Jeg åbner først `titanic` datasæt. Datasættet i `titanic`-pakken er allerede blevet adskilt efter training og testing men lad os bare ignorere det og følger samme ramme som i ovenstående.

```
library(titanic)

titanic <- titanic_train
```

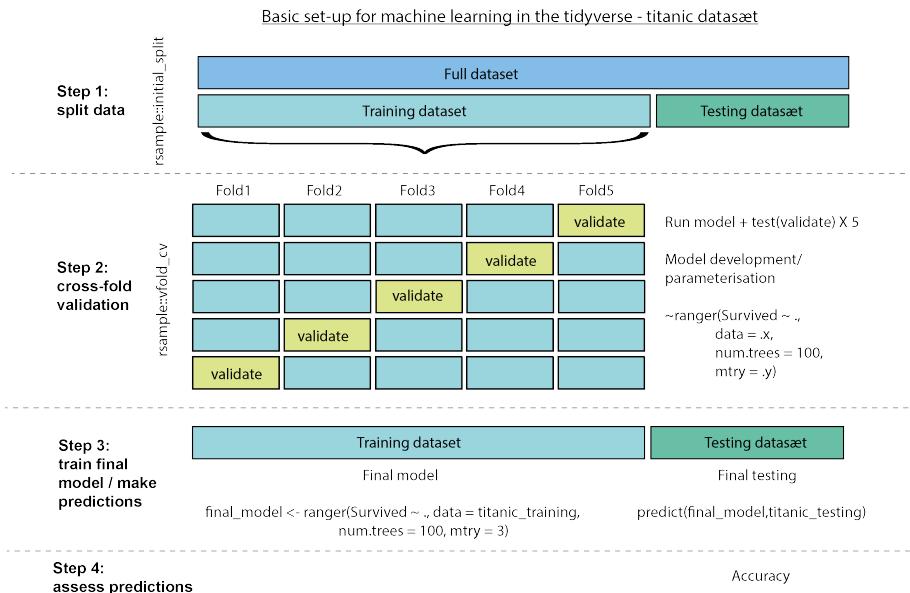
```
titanic <- titanic %>% drop_na() %>% as_tibble() %>%
  mutate(Survived = factor(if_else(Survived == 1, "yes", "no"), levels=c("yes", "no")),
  mutate(Pclass = recode(Pclass, `1`="first", `2`="second", `3`="third")) %>%
  mutate(Pclass = as_factor(Pclass),
  Sex = as_factor(Sex),
  Port = as_factor(Embarked),
  Solo = if_else(SibSp + Parch == 0, "yes", "no")) %>%
  select(Survived, Pclass, Sex, Age, Solo, Port, Fare)

titanic

## # A tibble: 714 x 7
##   Survived Pclass Sex     Age Solo Port   Fare
##   <fct>    <fct>  <fct> <dbl> <chr> <fct> <dbl>
## 1 no       third   male    22   no    S     7.25 
## 2 yes      first   female  38   no    C     71.3  
## 3 yes      third   female  26   yes   S     7.92 
## 4 yes      first   female  35   no    S     53.1  
## 5 no       third   male    35   yes   S     8.05 
## 6 no       first   male    54   yes   S     51.9  
## 7 no       third   male    2    no    S     21.1  
## 8 yes      third   female  27   no    S     11.1  
## 9 yes      second  female  14   no    C     30.1  
## 10 yes     third   female  4    no    S     16.7 
## # ... with 704 more rows
```

Her vil vi gerne forudsige hvem der overlevede, og hvem der ikke overlevede. Det er et eksempel på et “classification” problem, hvor der er to mulige svar (“yes” eller “no”) og man gerne vil beregne sandsynlighed for de to muligheder. Det er hvad der kaldes for et “supervised” problem, fordi når vi træner modellen, ved vi faktisk hvem der overlevede i datasættet. Det er til forskel for eksempelvis kmeans clustering, hvor vi ikke har svaret i forvejen (men bare kan gætte, at de forskellige clusters relatere til noget, for eksempel hvilke species en observation hører til).

Før vi går i gang, her er et overblik over processen med at opbygge modellen med titanic:



I følgende steps forklarer jeg processen med `titanic` i flere detaljer:

### 12.4.1 Splitte datasættet og lave cross-validation dataframe

```

set.seed(333)
titanic_split <- initial_split(titanic, prop = 0.75)
titanic_training <- titanic_split %>% training()
titanic_test <- titanic_split %>% testing()

```

Vi anvender 5-fold cross validation på datasættet `titanic_training` og udtrækker de træning og testing sæt til de forskellige folds (og lagrer dem i henholdsvis `train` og `validate` i dataframen `cv_data`):

```

set.seed(333)
cv_split <- vfold_cv(titanic_training, v = 5)

cv_data <- cv_split %>%
  mutate(
    train = map(splits, ~.x %>% training()),
    validate = map(splits, ~.x %>% testing())
  )

cv_data

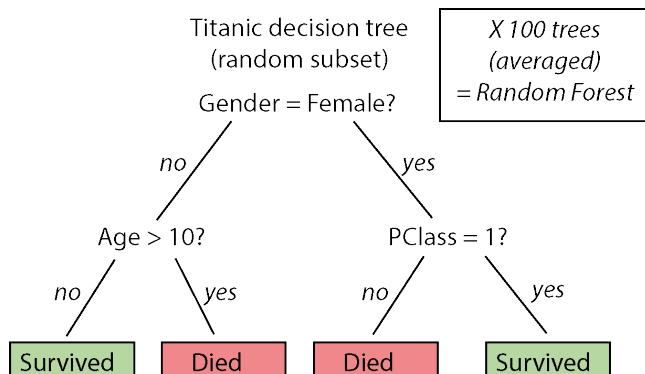
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits           id     train      validate

```

```
## <list> <chr> <list> <list>
## 1 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]>
## 2 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]>
## 3 <split [428/107]> Fold3 <tibble [428 x 7]> <tibble [107 x 7]>
## 4 <split [428/107]> Fold4 <tibble [428 x 7]> <tibble [107 x 7]>
## 5 <split [428/107]> Fold5 <tibble [428 x 7]> <tibble [107 x 7]>
```

### 12.4.2 Definere classification model

Næste vil vi gerne anvende en model over de fem training sæt. Et godt eksempel på en classifikation metode som er både nem at bruge og meget kræftig er en “random forest”. Her er det ikke nødvendeigt at forstå al detaljer, men helt kort - man opbygger beslutningstræer baserede på tilfældigt udvalgte subsets af data. I et beslutningstræ laver man “splits” over de forskellige variabler, som man bruger for at bestemme, om en passagerer overlevede eller ej (se billede). Når man kan gøre samme process flere gange, over eksempelvis 100 træer, der involverer 100 tilfældige subsets af data, kan man tage et gennemsnit, som udgør de endelige beslutninger (predictions eller forudsigelser).



I følgende anvender jeg funktionen `ranger()` (installer pakken `ranger`), der fitter en random forest i R på datasættet.

```
set.seed(333)
ranger_model <- ranger(formula = Survived ~ . , # . betyder alle variabler i datasættet
                        data = .x,
                        num.trees = 100)
```

Lidt om `ranger`-modellens parametre:

- Jeg angiver en formel - variablen `Survived` er afhængig og alle andre variabler er uafhængig (betegnet ved “.” for at spare tid).
- `data` er `.x` og referer til en træning sæt til én af vores “folds” i vores cross-fold validation datasæt.
- `num.trees` fortæller, hvor mange træer jeg vil lave (jo mere jo bedre, men der er en trade-off med hensyn til tid/energi)

### 12.4.3 Anvende modellen og lave predictions

Jeg laver mine modeller ved at oprette en ny kolonne `model` i dataframe `cv_data`, hvor jeg anvender min custom funktion med `map`:

```
cv_models_rf <- cv_data %>%
  mutate(model = map(train, ranger_model))
cv_models_rf

## # 5-fold cross-validation
## # A tibble: 5 x 5
##   splits           id    train      validate       model
##   <list>          <chr> <list>      <list>        <list>
## 1 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]> <ranger>
## 2 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]> <ranger>
## 3 <split [428/107]> Fold3 <tibble [428 x 7]> <tibble [107 x 7]> <ranger>
## 4 <split [428/107]> Fold4 <tibble [428 x 7]> <tibble [107 x 7]> <ranger>
## 5 <split [428/107]> Fold5 <tibble [428 x 7]> <tibble [107 x 7]> <ranger>
```

Næste laver jeg forudsigelser over de fem `validate` datasæt med funktionen `predict()` - det fungere på samme måde som i ovenstående regression model, men med `ranger` er man nødt til at udtrække selve forudsigelserne ved at tilføje `$predictions`, ligesom i følgende funktion `my_prediction_function`:

```
my_prediction_function <- ~predict(.x, .y)$predictions

cv_prep_rf <- cv_models_rf %>%
  mutate(
    actual = map(validate, ~.x %>% pull(Survived)), #virkelig "yes" eller "no"
    predicted = map2(.x = model, .y = validate, my_prediction_function) #forudsigelser af "yes" el
  )
```

Nu lad os kigge på vores forudsigelser (“yes” eller “no”) ved siden af de virkelige værdier (“yes” eller “no”) i variablen `Survived`, for hver af de fem validation sæt:

```
predictions_df <- cv_prep_rf %>% select(id, actual,predicted) %>% unnest(c(actual,predicted))
predictions_df

## # A tibble: 535 x 3
##   id    actual predicted
##   <chr> <fct>   <fct>
## 1 Fold1 yes     yes
## 2 Fold1 yes     yes
## 3 Fold1 no      no
## 4 Fold1 yes     yes
## 5 Fold1 no      no
## 6 Fold1 yes     yes
## 7 Fold1 yes     no
```

```
## 8 Fold1 no    no
## 9 Fold1 yes   yes
## 10 Fold1 yes  yes
## # ... with 525 more rows
```

#### 12.4.4 Beregne accuracy

Hvor mange fik vi rigtige? Vi beregner “accuracy” som måler proportionen af “rigtig” forudsigelser (både `actual` og `predicted` er enige om, hvem der overlevede eller ikke overlevede). Da jeg har forudsigelser til 5 forskellige datasæt (validation sæt for hver “fold”), anvender jeg `group_by(id)`, og dernæst `summary` til at beregne proportionen af tilfælde variablen `actual` er den samme som `predicted`.

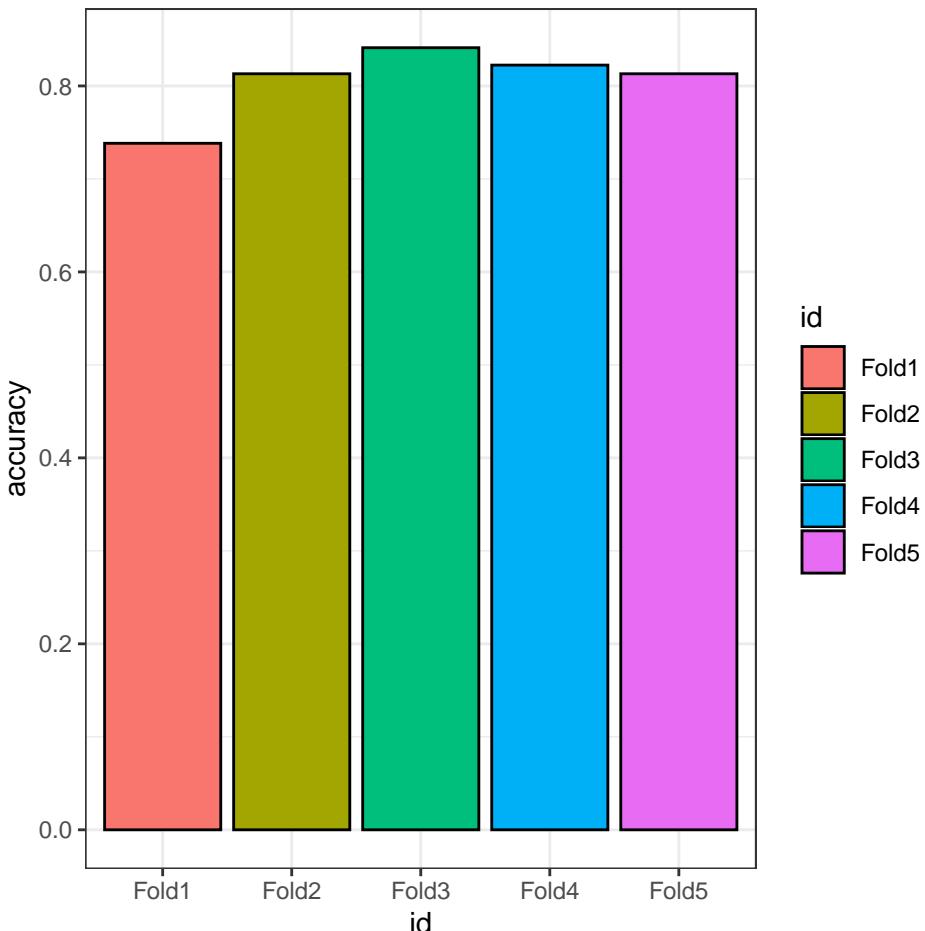
```
my_accuracies <- predictions_df %>%
  group_by(id) %>%
  summarise("accuracy" = sum(actual==predicted)/n())

my_accuracies
```

```
## # A tibble: 5 x 2
##   id     accuracy
##   <chr>    <dbl>
## 1 Fold1    0.738
## 2 Fold2    0.813
## 3 Fold3    0.841
## 4 Fold4    0.822
## 5 Fold5    0.813
```

Jeg kan omsætte tallene til et plot:

```
my_accuracies %>%
  ggplot(aes(x=id,y=accuracy,fill=id)) +
  geom_bar(stat="identity",colour="black") +
  theme_bw()
```



#### 12.4.5 Parameter tuning

I funktionen `ranger` var der faktisk nogle parametre som kan have indfyldelse over modellens predictions til sidste - én af dem er for eksempel `num.trees` - hvis man bruger f. k. 100 træer får man sandsynligvis bedre resultater end hvis man anvender kun et træ. En anden parameter, som vi angav som "default" i ovenstående analyse er `mtry`. Det fortæller hvor mange variabler vi skal inddrage i en enkel beregning i træerne (det inddrager mere tilfældighed i algoritmen og undgår, at 1-2 variabler har for meget indfyldelse på resulaterne). Vi vil gerne prøve de forskellige værdier til `mtry` i modellen og se, hvilken værdi giver bedste resultater. I `tidyverse`-pakken er der en nyttig funktion `crossing`, der laver kopier af de forskellige folds datasæt, én til hver mulig værdie af `mtry`:

```
# Prepare for tuning your cross validation folds by varying mtry
cv_tune <- cv_data %>%
  crossing(mtry = c(2:6))
```

```
cv_tune
```

```
## # A tibble: 25 x 5
##   splits      id    train     validate      mtry
##   <list>     <chr> <list>     <list>      <int>
## 1 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]>     2
## 2 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]>     3
## 3 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]>     4
## 4 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]>     5
## 5 <split [428/107]> Fold1 <tibble [428 x 7]> <tibble [107 x 7]>     6
## 6 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]>     2
## 7 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]>     3
## 8 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]>     4
## 9 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]>     5
## 10 <split [428/107]> Fold2 <tibble [428 x 7]> <tibble [107 x 7]>    6
## # ... with 15 more rows
```

Nu kan jeg bygge en model op til hver kombination af fold (variablen `id`) og mtry (variablen `mtry`). Bemærk at jeg bruger `map2` her, der jeg gerne vil bruge to kolonner af min næste dataframe - `mtry` (betegnet som `.y` i funktionen) og `train` (betegnet som `.x` i funktionen).

```
# Build a model for each fold & mtry combination
my_ranger_func <- ~ranger(formula = Survived~, data = .x,
                           mtry = .y,
                           num.trees = 100)

cv_models_rf <- cv_tune %>%
  mutate(model = map2(train, mtry, my_ranger_func))
```

Og utrækker predictions med `predict` ligesom i ovenstående:

```
cv_prep_rf <- cv_models_rf %>%
  mutate(
    actual = map(validate, ~.x$Survived),
    predicted = map2(.x = model, .y = validate, ~predict(.x, .y)$predictions)
  )
```

Nu laver jeg en dataframe ligesom i ovenstående som viser mine predictions - men denne gange har jeg predictions over både min folds og de forskellige værdier for "mtry":

```
predictions_df <- cv_prep_rf %>% select(id,mtry,actual,predicted) %>% unnest(cols=c(actual))
predictions_df

## # A tibble: 2,675 x 4
##   id      mtry actual predicted
##   <chr> <int> <fct>   <fct>
```

```
## 1 Fold1    2 yes    yes
## 2 Fold1    2 yes    yes
## 3 Fold1    2 no     no
## 4 Fold1    2 yes    yes
## 5 Fold1    2 no     no
## 6 Fold1    2 yes    yes
## 7 Fold1    2 yes    no
## 8 Fold1    2 no     no
## 9 Fold1    2 yes    yes
## 10 Fold1   2 yes    yes
## # ... with 2,665 more rows
```

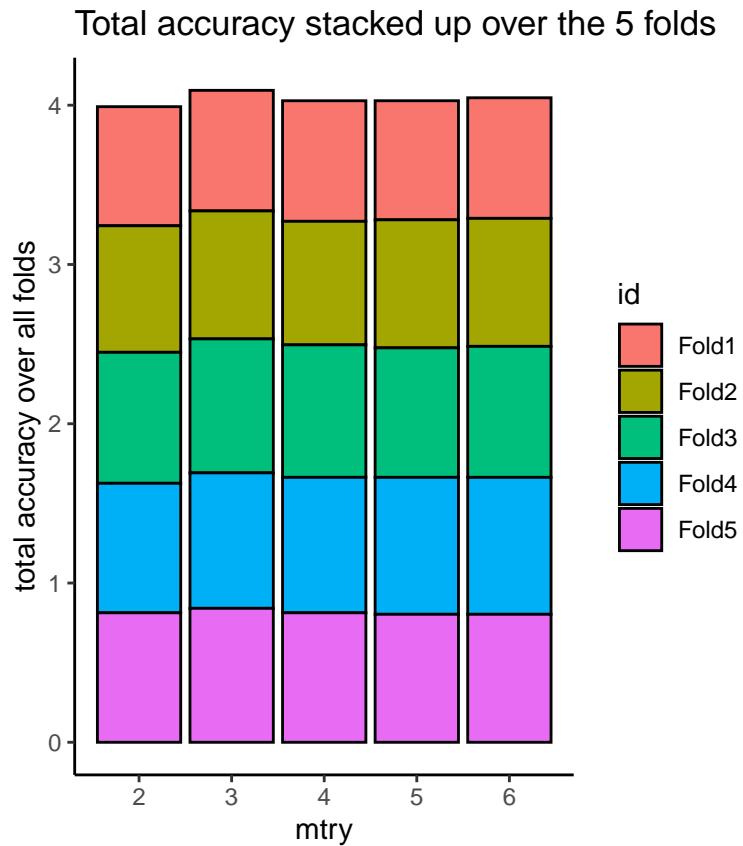
Jeg anvender `group_by` efter både `mtry` og `id` for at beregne accuracy:

```
my_accuracies <- predictions_df %>%
  group_by(mtry, id) %>%
  summarise("accuracy" = sum(actual==predicted)/n()) %>%
  ungroup(mtry)
```

```
## `summarise()` has grouped output by 'mtry'. You can override using the
## `.` argument.
```

I følgende plot visualiseres den sammenlagt accuracy over all folds, til hver mulig værdi af "mtry"-parametren. Man kan se, at det gør ikke meget forskel, hvilken værdie af "mtry" jeg vælger i min final model, min den højeste sammenlagt søjle hører til "mtry=3" - så jeg tager 3 som min "mtry" parameter i min "final" model.

```
my_accuracies %>% ggplot(aes(fill=id, x=mtry, y=accuracy)) +
  geom_bar(stat="identity", colour="black") +
  ylab("total accuracy over all folds") + ggtitle("Total accuracy stacked up over the 5 folds") +
  theme_classic()
```



#### 12.4.6 Lave final model med titanic datasæt

I min final model anvender jeg parametrene jeg valgt i min cross-validation proces, og træner jeg min model over hele `titanic_training` datasæt på én gang.

```
final_RF <- ranger(formula = Survived ~ . , data = titanic_training,
                     mtry = 3,
                     num.trees = 100)
```

Nu jeg kan endelige anvende modellen til at lave forudsigelser over min `titanic_test` datasæt. Så kan jeg få en “final” accuracy, som fortæller hvor god min model er at finde ud af hvem der overlevede tragedien, baserede på de andre variabler.

```
predictions <- tibble("actual"=titanic_test$Survived,
                      "predicted"=predict(final_RF,titanic_test)$predictions)

predictions %>% summarise("accuracy"=sum(actual==predicted)/n())
```

```
## # A tibble: 1 x 1
##   accuracy
##   <dbl>
## 1 0.821
```

Så kan man se, at vi er faktisk god til at forudsige hvem, der overlevde tragedien (se også de forskellige scores og analyser på Kaggle).

### 12.4.7 Variable importance

Den allersidste, som kan være nyttig at vide, er hvilke variabler var mest vigtig for at kunne lave en nøjagtig forudsigelse af hvem, der overlevede. Man kan tilføje en indstilling, der hedder `importance = "impurity"` til kommandoen i funktion `ranger`, som i følgende:

```
final_RF <- ranger(formula = Survived ~ . , data = titanic_training,
                     importance = "impurity",
                     mtry = 3,
                     num.trees = 100)
```

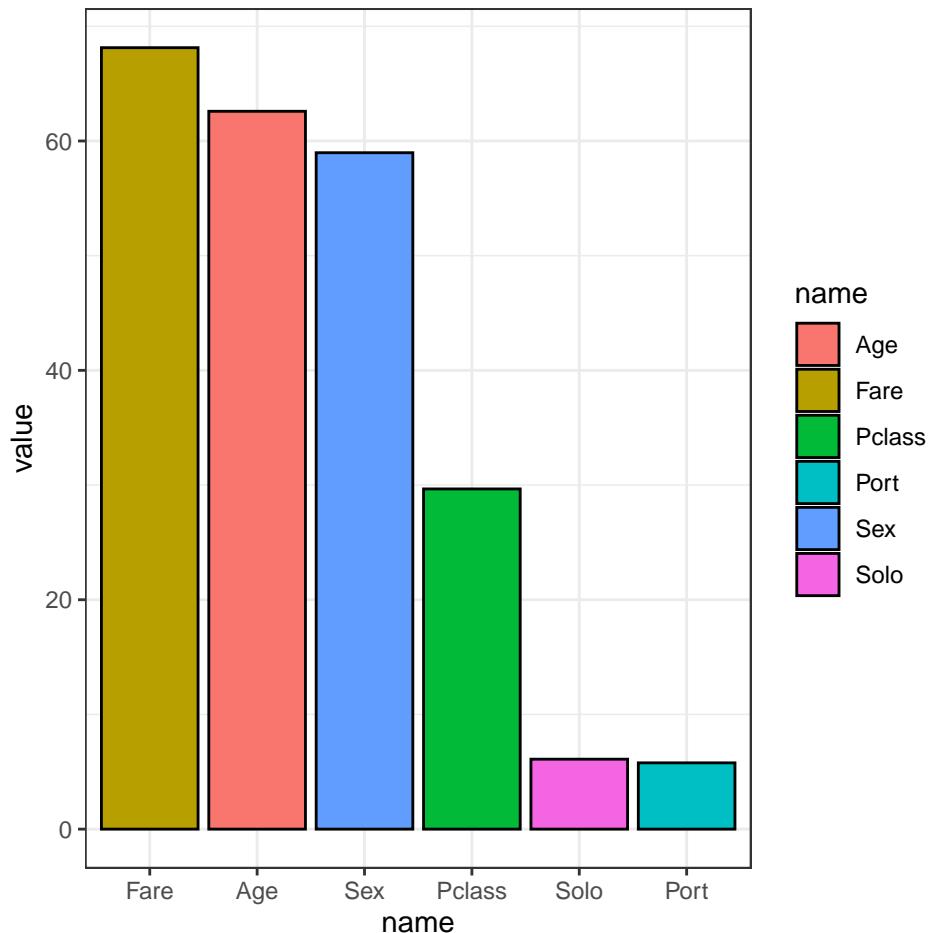
Sidste vil vil gerne vide, bidraget af de forskellige variabler i modellen:

```
mod <- t(final_RF$variable.importance) %>% as_tibble()
mod <- mod %>%
  pivot_longer(everything())
mod
```

```
## # A tibble: 6 x 2
##   name    value
##   <chr>   <dbl>
## 1 Pclass  29.7
## 2 Sex     59.0
## 3 Age     62.6
## 4 Solo    6.10
## 5 Port    5.78
## 6 Fare    68.1
```

Den højere værdien, jo vigtigere er variablen:

```
mod %>%
  ggplot(aes(x=name,y=value,fill=name)) +
  geom_bar(stat="identity",colour="black") +
  scale_x_discrete(limits=mod %>% arrange(desc(value)) %>% pull(name)) +
  theme_bw()
```



Så kan man se, at variablen **Sex** er meget vigtig for at udgøre hvem, der overlede, hvilket er i ovenenstemmelse med vors undersøgelser med titanic fra tidligere emner!

## 12.5 Problemstillinger

**Problem 1):** Quiz - maskinslæring

---

### Classification med penguins

I datasættet “penguins” vil vi gerne anvende en classification model for at forudsige “male” og “female” pengvinger (variablen **sex**) ud fra de andre variabler i datasættet. Første åbne datasættet:

```
library(palmerpenguins)
penguins <- penguins %>% drop_na()
```

OBS: Vi tilpasser koden tæt fra ovenstående eksempel med datasættet `titanic` - i følgende spørgsmål følg med koden fra Section 12.4

---

*Lad os begynde med at opdele datasættet i to og brug testing data til at lave en “nested” dataframe som vi kan bruge til at lave 5-fold-cross-validation.*

**Problem 2)** *Set up cross-validation framework*

- a) Opdel datasættet “penguins” i to, således at 80 procent af observationerne hører til det træning datasæt (og 20 procent hører til det testing datasæt). Kald dit resultat `penguins_split`. Hvor mange observationer er blevet allokeret til træning og testing datasæt?
- b) Udtræk de to datasæt fra `penguins_split` og kalder dem for `penguins_training` og `penguins_test`.

*Vi lægger datasættet `penguins_test` til siden (vi får bruge for det senere), og arbejder kun med datasættet `penguins_training` for at udvikle vores model.*

- c) Anvend `penguins_training` til at lave en 5-fold-cross-validation dataframe, `cv_split`. Udtrække de fem testing og training datasæt fra kolonnen `splits` til de forskellige folds (kald deres kolonner for henholdsvis `validate` og `train`) og kald din nye “nested” dataframe for `cv_data`.
- 

*Nu vil vi gerne lave samme model 5 gange - en gang til hvert datasæt i kolonnen `train` i vores nye `cv_data` dataframe. Den model skal være en ‘classification’ model, når vi gerne vil forudsige “male” eller “female” pingviner (vores afhængig variabel er `sex` som er kategorisk med to mulige værdier).*

**Problem 3)** *Set-up modellen*

- a) Lad os starte med det første datasæt og generaliser til en custom funktion i b). Anvend `pluck` for at få frem til det første datasæt i kolonnen `train`, og dernæst anvende funktionen `ranger` for at lave en classification of variablen `sex` som afhængig variable, med alle de andre variabler som uafhængige variabler (betegnet med `.` i modellen). Angiv `num.trees==100`.
  - b) Når vi har fået `ranger` til at virke på det første datasæt laver vi en custom funktion der kan anvendes over samtlige datasæt i kolonnen `train` - tilpas din `ranger` model fra a) til en custom funktion og kalder den for `ranger_function`.
  - c) Anvend `ranger_function` på samtlige datasæt i kolonne “train” (opret en ny kolonne `icv_data`, der hedder `model` hvor de fem modeller bliver lagret).
-

*Nu har vi lavet en classification model til hvert træning sæt! Vi vil gerne vide, hvor gode modellerne er til at forudsige `sex` i de fem datasæts i kolonnen `validate` (husk at `validate` indeholder de styk der ikke var brugt i træning til den relevant `id(fold)`).*

**Problem 4) Make predictions from model**

a) Brug `map/map2` til at lave to nye kolonner i `cv_data`:

- `actual` hvor du udtrækker de virkelige værdier for variablen `sex`
- `prediction` hvor du anvender `predict` med din models (betegnet som `.x`) og dine validate sæt (betegnet som `.y`)

b) Udvælg `id`, `actual` og `prediction`, unnest og beregne accuracy til hvert af de 5 folds.

c) Lav et barplot for at se din accuracies over de 5-folds.

*Nu har vi lavet 5-fold-cross validation til at træne en classification model. Vi kunne godt tænke os at udvide processen lidt, fordi der er nogle parametre der kan ændres på i funktionen `ranger`. Lad os prøve at ændre på parameten `mtry` for at se, om vi kan gøre modellen endnu bedre. Igen følger vi meget tæt på ovenstående kurusnotaterne - bare med et andet datasæt.*

**Problm 5) Tune parameteren `mtry`.**

a) Anvend funktionen `crossing` til at inddrage `mtry` (fra 2 til 6 som er antal uafhængig variabler) som kolonne i `cv_data`. Kald din nye dataframe for `cv_tune`.

b) Tilpas din custom ranger funktion fra **Spørgsmål 3** hvor du inddrager `mtry` (betegnet som `.y`) og anvende din model på de forskellige training sæt for hver mulige værdi af `mtry` (husk at bruge `map2` her).

c) Lav forudsigelser over alle modeller (der er 25!) og udtrækker dine “actual” og “predicted” værdier præcis ligesom før.

d) Beregn accuracy for alle 25 sæt, lav et stacked barplot og vælg den bedste mulige værdi for `mtry`.

*Nu at vi har valgt et værdi for `mtry` lad os går henne og lav en “final” model! Når vi har lavet den final model kan vi endelig kigger på vores `penguins_test` datasæt. Husk at `validate` sæts var brugt til at teste modellen undervejs i udviklingsprocessen, men var stadig en del af `penguins_training`. Man kigger på den test datasæt kun en gang til sidste for at spørge - hvor god er vores endelig model?*

**Problem 6) Lave en *final* model**

- a) Anvende funktionen `ranger` til at lave en classification model på datasættet `penguins_training`. Angiv `num.trees=100` og din valgt værdie for parameteren `mtry`.
  - b) Lav predictions med funktionen `predict` og angiv dit "penguins\_test" datasæt. Beregen accuracy fra din model baserede på din virkelige værdier for `sex` og din forventede værdier fra funktionen `predict`.
- 

*Nu at vi er færdig med at lave og bedømme vores "final" model er vi interesseret i - hvilke variabler havde den største indflydelse på vores forudsigelser af "male" og "female" pingviner?*

**Problem 7)** *Variable importance*

- a) Tag din "final" model og angiv indstillingen `importance = "impurity"` i funktionen. Kør modellen og udtræk din variable importance vector med `final_RF$variable.importance`.
  - b) Omsæt til et barplot hvor du angiv rækkefølgen af søjlerne efter vigtigheden af de forskellige variabler (den meste vigtig til venstre).
- 

**Problem 8)** Gentag samme process i et andet datasæt:

## 12.6 Yderligere kommentarer og pakker

Jeg spang over mange nyttige pakker og funktioner, der kan bruges til at lave maskinslæring processer i R

- `Yardstick`-pakken er til metrics - fk. accuracy/precesion/recall osv. man kan også lave ROC curves osv. Jeg anvendte kun accuracy i kurset (til at vise processen/iden men begrænser overordnet læs i emnet) men man plejer at anvende forskellige metrics i virkeligheden.
- `Tidymodels`-pakken er en generel pakke, hvor man få en "fælles" tilgang til at opbygge modeller der stammer fra mange forskellige pakke - man kan fk. bruge samme tilgang til at lave blandt andet regression eller bayesian modeller, random forest, support vector machines osv. - alle modeller har nu samme syntaksen i den `Tidymodels` tilgang.