

Visualisering af biologiske datasæt - 2022

Sarah Rennie

Last updated: 2022-04-18

Contents

1 Grundlæggende R	7
1.1 Inledning til kapitel	7
1.2 RStudio	7
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	28
2 Introduktion til R Markdown	33
2.1 Hvad er R Markdown?	33
2.2 Installere R Markdown	33
2.3 Videodemonstrationer	34
2.4 Lave et nyt dokument i R Markdown	34
2.5 Skrive baseret tekst	37
2.6 Kode indenfor teksten ('Inline chunks')	38
2.7 Kode chunks	39
2.8 Knit kode	40
2.9 Matematik	40
2.10 Problemstillinger	40
2.11 Slut for ugen	41
2.12 Ekstra links	42
3 Visualisering - ggplot2 dag 1	43
3.1 Inledning og videoer	43
3.2 Transition fra base R til ggplot2	45
3.3 Vores første ggplot	47
3.4 Lidt om ggplot2	49
3.5 Specifcere axse tekst og titel	51
3.6 Ændre farver	53
3.7 Ændre temaet	54

3.8	Forskellige type plots	55
3.9	Troubleshooting	69
3.10	Problemstillinger	70
3.11	Næste gang	76
4	Visualisering - ggplot2 dag 2	77
4.1	Indledning og videoer	77
4.2	Koordinat systemer	78
4.3	Mere om farver og punkt former	84
4.4	Annotations	91
4.5	Opdele plots (<code>facet_grid/facet_wrap</code>)	97
4.6	Problemstillinger	99
4.7	Workshop opgave (OBS Fredag)	107
4.8	Ekstra notat: gemme dit plot	107
4.9	Ekstra links	107
4.10	Slut for ugen	108
5	Bearbejdning dag 1	109
5.1	Inledning og læringsmålene	109
5.2	Video ressourcer	109
5.3	Hvad er Tidyverse?	110
5.4	Lidt om <code>tibbles</code>	112
5.5	Transition fra base til tidyverse	113
5.6	Bearbejdning af data: <code>dplyr</code>	117
5.7	Ændre data med <code>recode()</code>	124
5.8	Visualisering: bruge som input i ggplot2	125
5.9	Problemstillinger	127
5.10	Kommentarer	129
6	Bearbejdning dag 2	131
6.1	Indledning og læringsmålene	131
6.2	<code>dplyr: group_by()</code> med <code>summarise()</code>	132
6.3	Tidyr pakke - Wide og Long data	138
6.4	Eksempel: Titanic summary statistics	142
6.5	Problemstillinger	146
6.6	Opgave (fredag workshop om tidyverse)	149
6.7	Ekstra links	149
6.8	Sidste kommentarer	149
7	Forbine tables, iterations og funktioner	151
7.1	Inledning og læringsmålene	151
7.2	Tilføje sample oplysninger med <code>left_join()</code>	152
7.3	Iterativ processer med <code>map()</code> funktioner	155
7.4	Custom functions	159
7.5	Nesting <code>nest()</code>	163
7.6	Problemstillinger	168

CONTENTS	5
7.7 Ekstra notater og næste gang	172
8 Visualising trends	173
8.1 Indledning og læringsmålene	173
8.2 <code>nest()</code> og <code>map()</code> : eksempel med korrelation	174
8.3 Visualisering af lineær regression med ggplot2	179
8.4 Plot linear regresion estimates	185
8.5 Problemstillinger	192
8.6 Næste uge	195
9 Clustering	197
9.1 Indledning og læringsmålene	197
9.2 K-means clustering	198
9.3 Hvor mange clusters skal der være?	205
9.4 Hierarchical clustering	209
9.5 Problemstillinger	212
10 Principal component analysis (PCA)	217
10.1 Indledning og læringsmålene	217
10.2 Hvad er principal component analysis (PCA)?	218
10.3 Fit PCA to data in R	223
10.4 Integrere PCA resultater med broom-pakke	224
10.5 Quiz og problemstillinger	231
10.6 Ekstra læsning	233
11 Emner fra eksperimental design	235
11.1 Inledning og læringsmålene	235
11.2 Baserende princip af eksperimental design	236
11.3 Case studies: Simpson's paradox	239
11.4 Case studies: Anscombe's quartet	244
11.5 Using PCA to find batch effects	247
11.6 Problemstillinger	253
12 Tidymodels og introduktion til maskin læring	257
12.1 Læringsmålene	258
12.2 Purpose of chapter	258
12.3 Testing and training sets	258
12.4 Regression in the tidy model framework	258
12.5 Classification in the tidy model framework	261
12.6 How good is the fit?	264
12.7 Cross validation	265
13 Presætning af datasæt over for andre	269
13.1 Introduktion til chapter og læringsmålene	269
13.2 Video ressourcer	269
13.3 Interaktiv plots	270
13.4 Shiny	270

13.5 Ekstra generelle råd	285
13.6 Andre muligheder/advanceret topics	286
13.7 Problemstillinger	286

Chapter 1

Grundlæggende R

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 udover 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 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 tekste 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] -1.44298135 -1.29490882 -0.18766910 -0.06217884  0.15052664 -0.01777189
## [7]  2.04844656 -0.44290878  0.35968202  0.23619413 -1.13732791  1.53681532
## [13]  1.10508885  0.06586895 -0.43349239  0.44762130 -0.34788940  0.16963958
## [19] -1.24747685  0.43346455 -0.14446346  0.55452115  0.35108834  3.26144336
## [25]  1.11562923
```

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] -1.44298135 -1.29490882 -0.18766910 -0.06217884  0.15052664 -0.01777189
tail(x) #sidste 6
## [1]  0.4334645 -0.1444635  0.5545211  0.3510883  3.2614434  1.1156292
x[1] #første værdi
## [1] -1.442981
x[length(x)] #sidste data point
## [1] 1.115629
```

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] 0.2030784 0.1505266 3.2614434 -1.4429814 1.1193251 1.0579816

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.
## -1.4430 -0.3479  0.1505  0.2031  0.4476  3.2614
```

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 (μ) 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
```

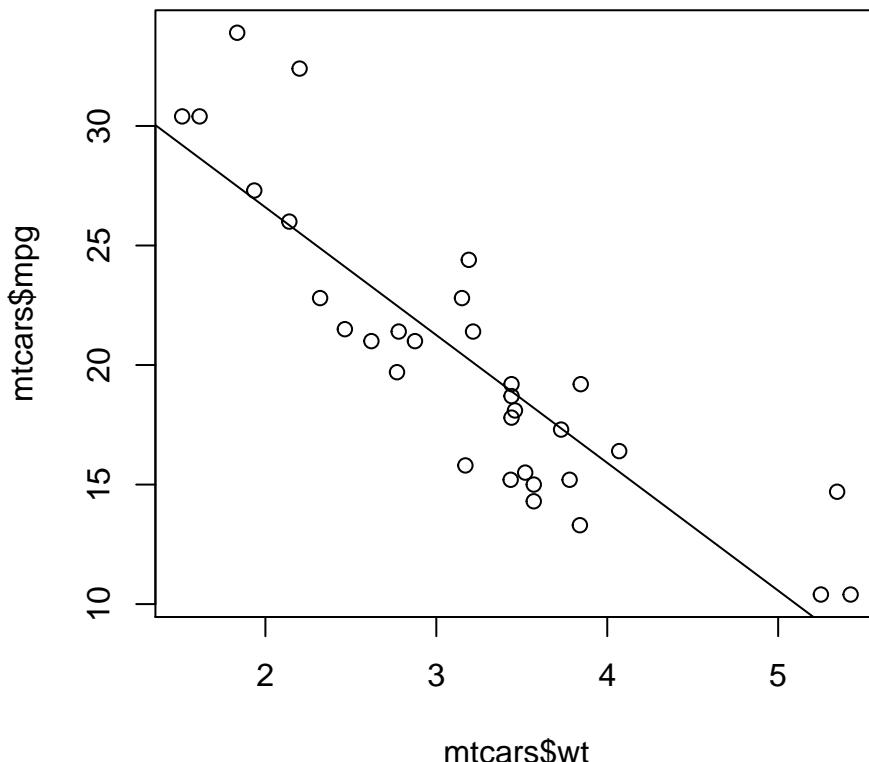
P-værdien er (<0.05), så nulhypotesen er forkastet til fordel af alternativhypotesen, altså modellen, hvor hver gruppe har sin egen middelværdi. Bemærk at det er til trods af, at to af de tre grupper kommer fra en normal fordeling med præcis de samme middelværdier (det er nok, at den tredje gruppe har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

1.9.7 Lineær regression

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`.

**Best fit line for
predicting mpg from weight**



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” (cofficeint 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
```

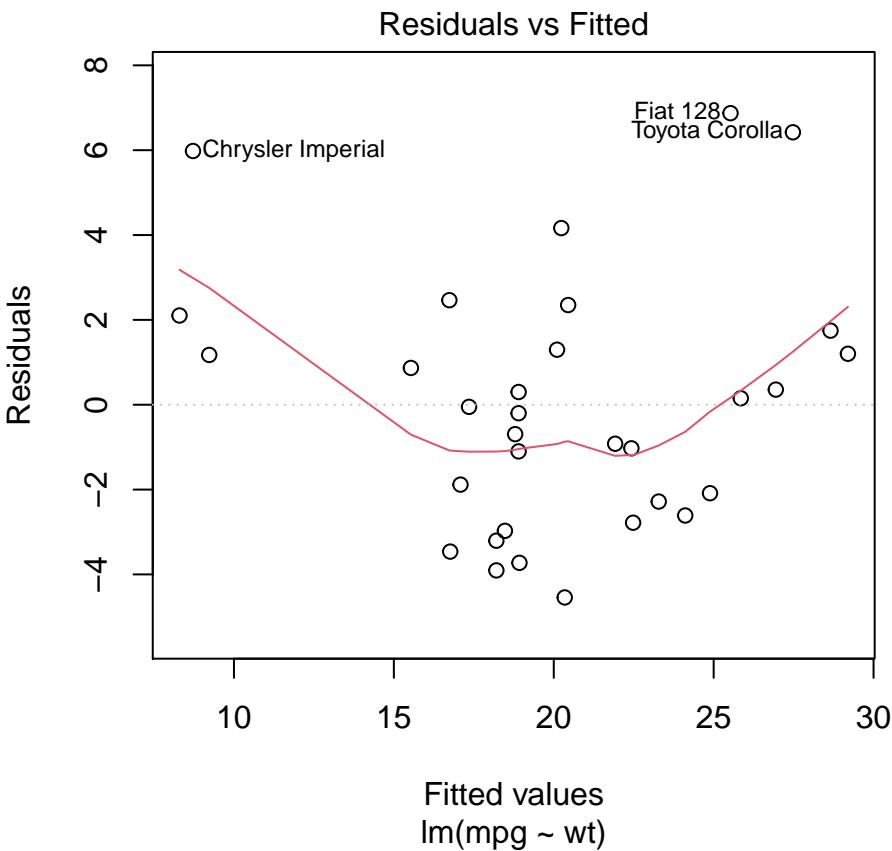
Det fortæller os, at $R^2 = 0.7528$.

1.9.9 Antagelser - lineær regression

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

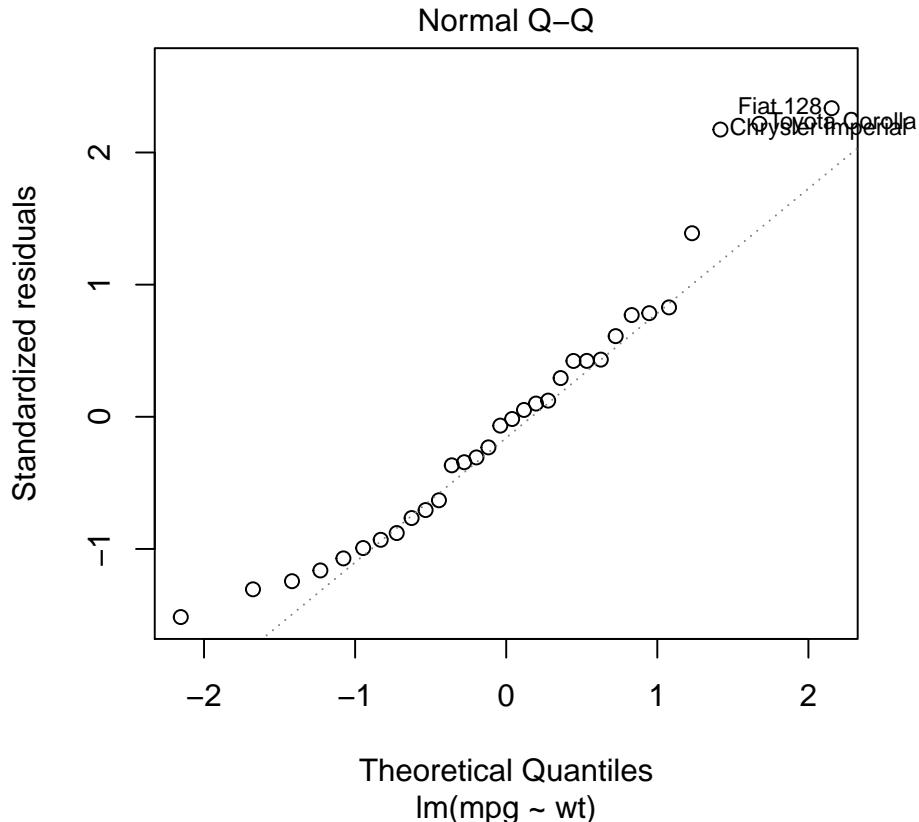
Koden `plot(mylm, which=c(1))` angiver residualer vs predikterede (fitted) værdier - de skal være tilfældigt fordele over plottet og prikkernes varians skal være nogenlunde konstant langt x-aksen (det giver, at den røde linje er flade).

```
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         -0.01773   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 (hvor 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

Alle bør lave quizzen men ellers vælg øvelser efter egen erfaring:

- 2-7 er meget grundlæggende og de fleste kan springer over hvis nogenlunde tryg med base-R
- 8-14 anbefaler jeg som en god måde at tjekke viden på
- 15-18 øver hvordan man laver variansanalyse/regression i R - regression kommer jeg ind på igen senere men det hjælper meget hvis du er tryg med brugen af funktionen `lm` til at lave modeller i ANOVA/simpel lineær regression.

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)
mean(y)
sd(y)
```

- 3) (helt baserende viden) Køre følgende kode til at åbne nogle af de indbygget datasæt, som vi bruger i kurset.

- Prøve `head()`, `summary()` osv.
- Prøve også fk. `?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) Lad os lave plotter i ‘baseR’. Jeg giver nogle muligheder for datasættet “iris”. Afprøve dem 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 påenere ved at give dem en titel/aksen-navne osv. Prøve `?plot` for at se nogle muligheder, og tilføj `ylab`, `xlab`, `main` (titel) i én af plotterne. Leg også med `col` (farver).

- 5) (dataframes) Brug datasættet `cars` til at:

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

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

- Beregn gennemsnitsværdien af variablen `dist` for hurtige biler og ikke-hurtige biler hver for sig (brug funktionen `tapply`). Gem resultatet.
- Brug `barplot` til at lave et plot af den gennemsnitlige `dist` for hurtige og ikke-hurtige biler.

- 5) (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(...), ...)
dim(mydf) # fire række og tre kolonner
mydf
```

- 6) (**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 <- #??
```

- 7) (**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 og `supp` er “VC”.
- Beregn den gennemsnitlige længde for observationerne i delmængden.

1.10.3 Kort analyse med reaktionstider

- 8) (**indlæse data**) Åbn en fil, der sidder i Absalon og hedder “reactions.txt” ved at bruge funktionen `read.table()` (giv objektet et navn, e.g. `data`). Husk at tjekke, om filen har en ‘header’ og bruge således `header=T` hvis nødvendigt.

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

- 9) (**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, fk.

```
data$subject <- as.factor(data$subject) #gør subject til en faktor
## gör den samme her for time:
data$...
```

- 10) (**delmængde af dataframe**)

Lav to delmængder af ovenstående datasæt -

- én til alle observationer fra tidspunktet “before” (`time == "before"`) og

- én til alle observationer fra tidspunktet “after”.

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

- 11) (**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 til at gøre den samme med mindre kode.

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

- 12) (**beregn forskellen og mean**)

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

- Lav en vector `diff`, der er ændringen i reaktionstiderne mellem “before” og “after”.
- Beregn den gennemsnitlige forskel i reaktionstiderne over de 10 personer.

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

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

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

Find følgende i outputtet fra R:

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

- 14) Skriv en kort sætning med din konklusion.

1.10.4 Øvelser med ANOVA/regression

- 15) (**ANOVA**) 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_h0 <- lm(Sepal.Width ~ 1, data=iris)
model_h1 <- lm(Sepal.Width ~ Species, data=iris)

anova(model_h0,model_h1)
```

- Er der en signifikant forskel i den gennemsnitlige `Sepal.Width` efter de forskellige `Species`?
- 16) (**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)
#skriv kode herfra
```

- 17) (**Lineær regression**) 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"` (default)

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

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

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

Brug `summary` på din model for at se:

- Hvad er `r.squared`? (multiple eller adjusted)
- Er hældningen signifikant?
- Hvad er ligningen på den bedste rette linje?

- 18) (**Multiple lineær regression**) I ovenstående model tilføj variablen `Height` som en ekstra prediktør variabel i modellen med en "+" tegn.

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

Brug `summary` for at finde ud af:

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

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

Chapter 2

Introduktion til R Markdown

2.1 Hvad er R Markdown?

R Markdown kan bruges som en nem 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. Man anvender R Markdown til følgende:

- Skrive, gemme og køre R kode
- Få direkte adgang til datasæt
- Lave rapporter som kan deles med andre
- En nem tilgang til at forstå andres kode, for eks. gennem eksemplerne og opgaverne fra dette kursus

Du kommer til at bruge R markdown gennem hele kurset, og jeg anbefaler, at du bruger det udelukkende til alle opgaverne.

2.2 Installere R Markdown

R Markdown er ligesom R gratis og ‘open source’. Den fungerer indenfor RStudio, så

Den kan installeres indenfor R ved at bruge den følgende kommando :

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

2.3 Videodemonstrationer

Jeg har lavet to korte videoer som kan ses her:

Video 1:

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

Link her hvis det ikke virker nedenunder: <https://vimeo.com/541035944>

Video 2:

- Jeg viser en kort analyse med et datasæt
- Jeg indlæser de data og laver en forløbig undersøgelse
- Jeg laver en paired t-test og skriver konklusioner
- Jeg tjekker antagelserne omkring den normale fordeling

Link her hvis det ikke virker nedenunder: <https://vimeo.com/541232690>

2.4 Lave et nyt dokument i R Markdown

Husk at indlæse pakken i RStudio:

```
library(rmarkdown)
```

```
## Warning: pakke 'rmarkdown' blev bygget under R version 4.0.5
```

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

I de fleste tilfælde arbejder vi med HTML dokumenter, men man har også andre muligheder (PDF/Word/Shiny osv...).

I mange tilfælde er det bekvemt at se resultaterne af din R kode inline, det vil sige, indenfor din .Rmd fil indenfor det R studio vindue. Til længere opgaver er det god praktisk at sikre, at man få altid en HTML rapport, der viser resultaterne. Til eksamen forventer jeg, at du afleverer en HTML rapport til mig. Det er derfor dit ansvar til at sikre, at din kode fungerer og du kan dermed succesfuldt producere en fil med din løsninger.

With HTML, you can easily view it in a web browser. compiling an HTML document is generally faster than generating a PDF or other format.

2.4.1 YAML

YAML Header: Controls certain output settings that apply to the entire document.

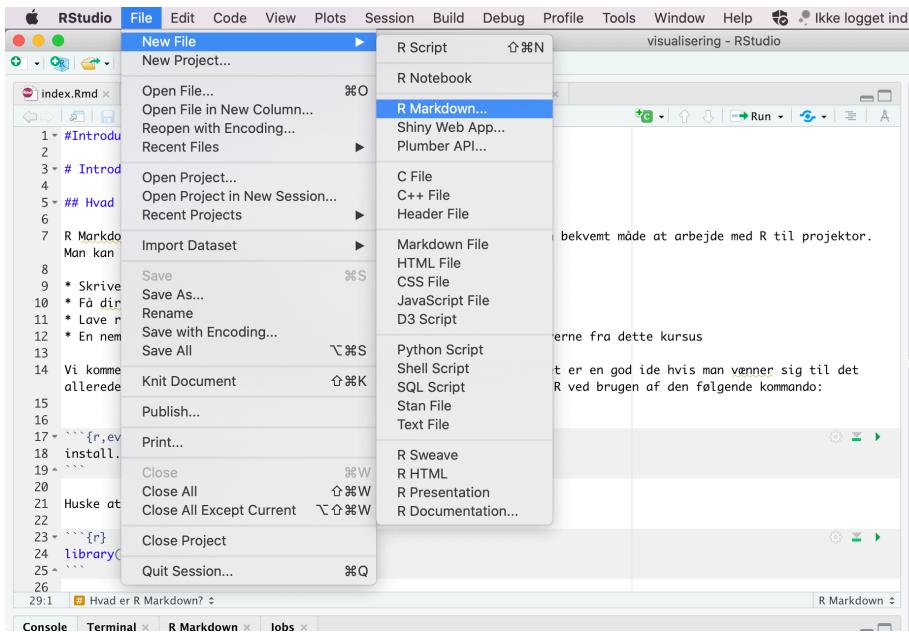


Figure 2.1: Hvordan man åbner et nyt R Markdown dokument

Den første sektion hedder ‘YAML’. (Dette står for ‘YAML Ain’t Markup Language’).

```

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

```

Det indeholder oplysninger om dokumentet, og her kan man specificere forskellige muligheder - f.eks. titel, forfatter, output-type (f.eks. HTML eller PDF), dato, osv.

I de fleste tilfælde kan vi bare nøjes med at bruge standard indstillinger, men hvis man gerne vil lære mere om de forskellige muligheder med YAML, kan man læse her:

<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>

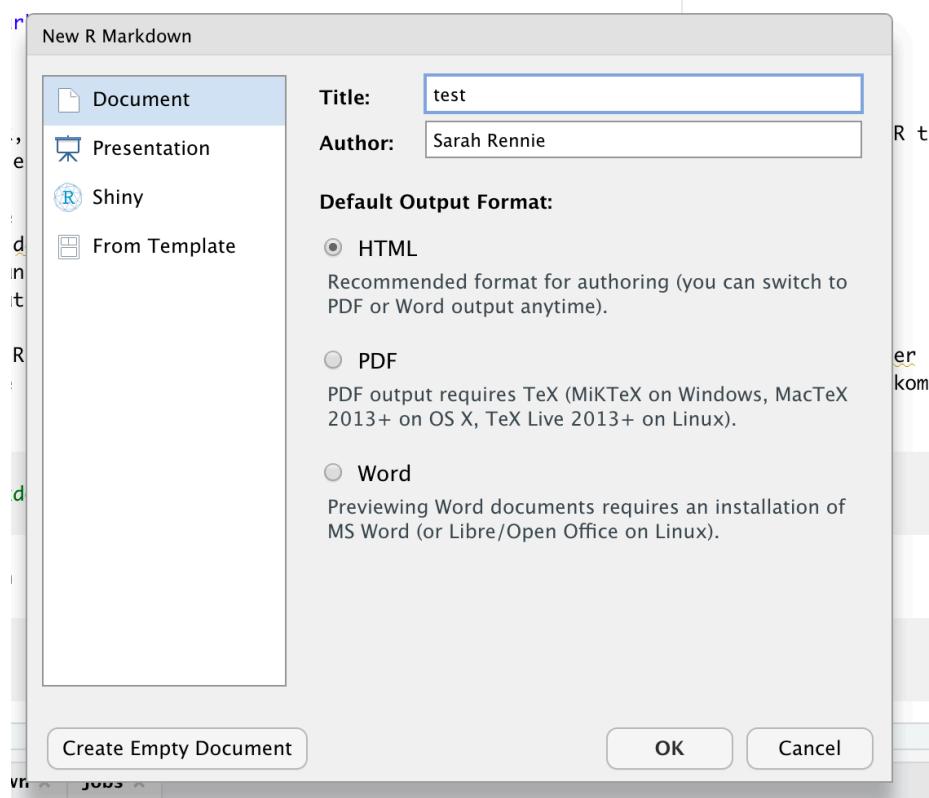
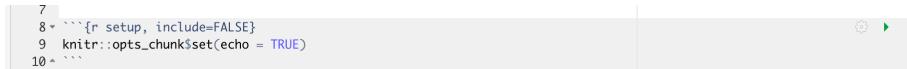


Figure 2.2: Hvordan man åbne et nyt R Markdown dokument

2.4.2 Globale options

Bemærke at der også er tekst som ser ud som følgende:



```

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

```

Figure 2.3: Hvordan man åbner et nyt R Markdown dokument

Med funktionen `opts_chunk$set()` kan man specificere de globale indstillinger, man gerne vil have, 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 andet angivet. Hvis `echo` er `TRUE`, så betyder det, at når man kører sine kode og kompilerer dokumentet, så kan man også se den kode, der er kørt, ligesom dens output, som en del af den færdige HTML, der fremvises.

2.5 Skrive baseret tekst

Her er nogle nyttige måder, man kan skrive tekst på, i opgaverne eller rapporter.

italic* **bold*

italic __bold__

italic bold

italic bold

2.5.1 Headers

Man kan også lave sektioner:

Header 1

Header 2

Header 3

2.5.2 liste

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

- Item 1
- Item 2

Chapter 3 Header 1

3.1 Header 2

3.1.1 Header 3

Figure 2.4: Caption for the picture.

- Item 2a
- Item 2b

2.6 Kode indenfor teksten ('Inline chunks')

This is useful when you want to include information about your data in the written summary. We'll add a few examples of inline code to our R Markdown Guide to illustrate how it works.

De fleste koder skrives indenfor såkaldte 'chunks' som vi kommer til nedenfor. Men nogle gange kan det være nyttigt at skrive kode direkte indenfor teksten. Dette gøres ved at skrive

Her er min `kode`

som ser sådan ud indenfor teksten:

Her er min kode

I dette tilfælde, bliver koden ikke kørt. Hvis man vil køre koden indenfor teksten, kan man skrive (for eksempel):

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

som ser sådan ud indenfor teksten:

De gennemsnitlige antal af observationer er 4.6666667

Og 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))
```

2.7 Kode chunks

Man kan oprette en ny chunk, enten ved at trykke på de *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).

Tryk på den grønne pile, der hedder *Run current chunk* for at køre hele den chunk. Resultatet kan ses lige nedenunder. Man kan også trykke på den grønne pile som ligger øverst til højre i den kode chunk.

Det er ofte hurtigere at bruge *Run current chunk* i stedet for at *Knit* (se nedenfor) hver gang man vil køre kode, fordi her kører man kun den enkelte chunk, man er interessenst i, i stedet for det hele dokument (som er tilfældet med *Knit*).

2.7.1 Chunk options

One of the great things about R Markdown is that you have many options to control how each chunk of code is evaluated and presented. This allows you to build presentations and reports from the ground up — including code, plots, tables, and images — while only presenting the essential information to the intended audience. For example, you can include a plot of your results without showing the code used to generate it.

Mastering code chunk options is essential to becoming a proficient R Markdown user. The best way to learn chunk options is to try them as you need them in your reports, so don’t worry about memorizing all of this now. Here are the key chunk options to learn:

For eksempel, en chunk med en ‘option’ nævnt 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

Do not put all your R commands into one big R chunk. Instead, split it into well-defined smaller chunks, which you may even give names. Investing effort in choosing good chunk names will pay off in terms of structuring your R code.

2.8 Knit kode

Man bruger *Knit* for at gengive filen i HTML form. Når man trykker på *Knit*, bliver alle kode i filen kørt og et HTML dokument fremvises. Man kan også bruge *Preview*, der kører ikke kode chunks, men bruger kun koden som er blevet kørt før.

2.9 Matematik

For example, the inline code $\int_0^5 x^2 dx$ will typeset as $\int_0^5 x^2 dx$:

2.10 Problemstillinger

- 1) Jeg har lavet en kort **quiz** i Absalon, som hedder “Quiz - R Markdown”.
- 2) Lave et nyt R Markdown dokument i R Studio. Prøve at lave en liste og nogle overskrifter af forskellige størrelser.
- 3) Tryk på **Knit** og tjekke at et html-dokument fremvises som forventet.
- 4) Edit the title of the markdown document to My first Markdown document, and click Knit again. Notice how the title of the output document changes
- 5) Tilføj en ny R chunk, med noget kode. e.g.

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

Øve med at trykke enten på den grønne pile, eller på knit. Prøve også at køre linjene en ad gangen med Ctrl + Enter.

Bemærk, at det tager længere til at `knit` hver eneste gang, end at bare køre chunks individ indenfor dit dokument.

- 6) *Kør en chunk* Trykke på den hjule i øverste hjørn af chunket og ændre på de forskellige chunk options. Tryk på ‘knit’ for at se, hvad der sker.
- 7) *Skriv matematik* Bruge $\$ \$$ til at skrive en lindring ind i teksten, \bar{x} . Hint: $\$\\sum_{i=1}^n\$$ and $\${x_i}\$$.
- 8) *Ændr dokument type* Hver gang du `knit`, du lave en HTML dokument. Prøve at lave en andet type dokument i stedet for - erstætte `html_document` med `word_document`.

The code in Rmd-file must be self-contained in the sense that you cannot use datasets (or other objects) that you have imported “outside” the Rmd-file. You therefore have to include the commands for data import in the file.

- 9) *Vigtig at husk* Tilføj følgende chunk til dit dokument og trykke på “knit”. Få du en fejlmeddelse?

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

Bemærk, at du får en fejlmeddelse. Det er 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 følgende **øverst i ovenstående chunk**, som du tilføjede i dit dokument. Nu bør dit dokument `knit`.

```
library(tidyverse)
```

- 10) Ind på Absalon har jeg lagt en R Markdown fil som hedder “R Markdown opgave”, som I kan bruge til at starte med at arbejde lidt med R Markdown baserede opgaver. Det kombinerer koncepter fra de forudgående kapitel om de grundlæggende ting i R og statistik.

2.11 Slut for ugen

Huske at sende mig eventuelle spørgsmål, som jeg kan svare på i Zoom rummet (sende gerne via mail eller chat).

Næste gang begynder vi at arbejde vi med `ggplot2`.

2.12 Ekstra links

https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf?_ga=2.49295910.1034302809.1602760608-739985330.1601281773

Her er en ‘quick tour’ som kan være nyttig (valgfri) https://rmarkdown.rstudio.com/authoring_quick_tour.html

Bonus: 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 i hvordan man kan visualisere data med R med 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_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 anvendte funktionen **plot()**, der er den standard base-R funktion til at lave et plot på. Man kan godt kan blive ved med at lave adskillige plotter i base-pakken, men det er ofte meget tidskrævende så snart man gerne vil lave noget mere indviklet eller pænere.

ggplot2 er den mest populær pakke fra **tidyverse**, og giver en alternativ måde at lave plotter på i R. Som vi kommer til at se i dette kapitel, har den **ggplot2** løsning en ret logisk tilgang, hvor man bygger et plot op i forskellige komponenter. Det kan virke uoverskueligt i først 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 man selv vil afprøve dem. Jeg anbefaler at bruge kursusnotaterne som en reference gennem kurset når man arbejder på øvelserne (hvor effektiv læring sker), og vær også opmærksom på, at jeg ofte introducerer nye ting i selve øvelserne.

3.1.4 Video ressourcer

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/544299069>

- 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/543945046>

- I video 2 dækker vi boxplots.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/543989990>

OBS: jeg sagde i videoen at `alpha=0.5` gøre punkterne mere markant, men jeg mente selvfølgelig mindre markant ;)

- I video 3 demonstrerer jeg barplots.

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/544033975>

- Video 5: Histogram og density plots

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/544188160>

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`).



Man kan 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()`:

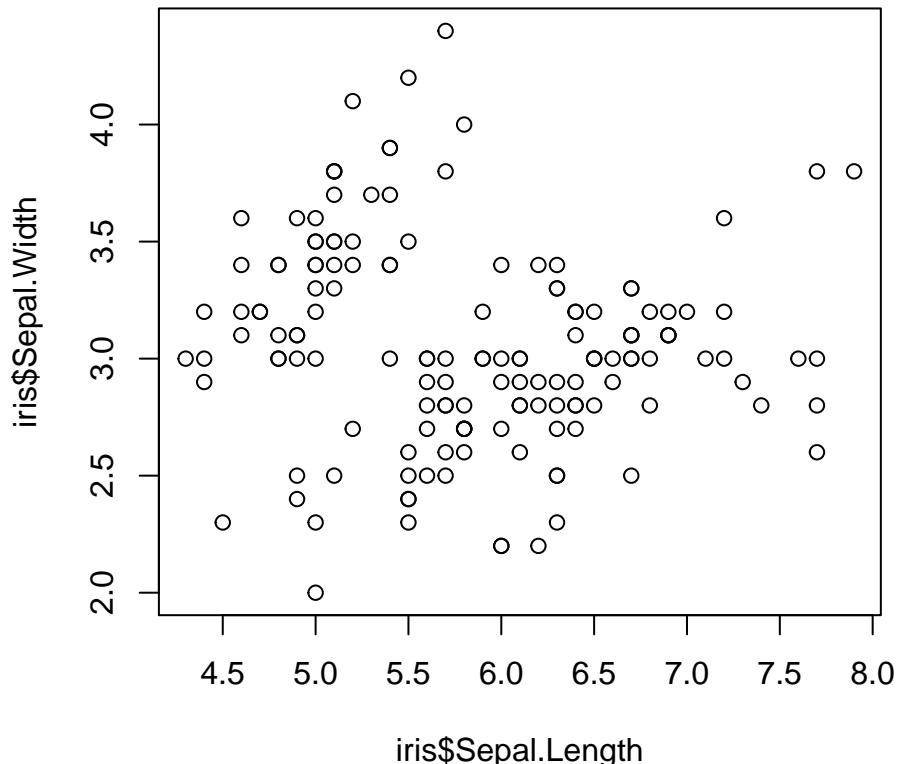
```
summary(iris)
```

```
##   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.

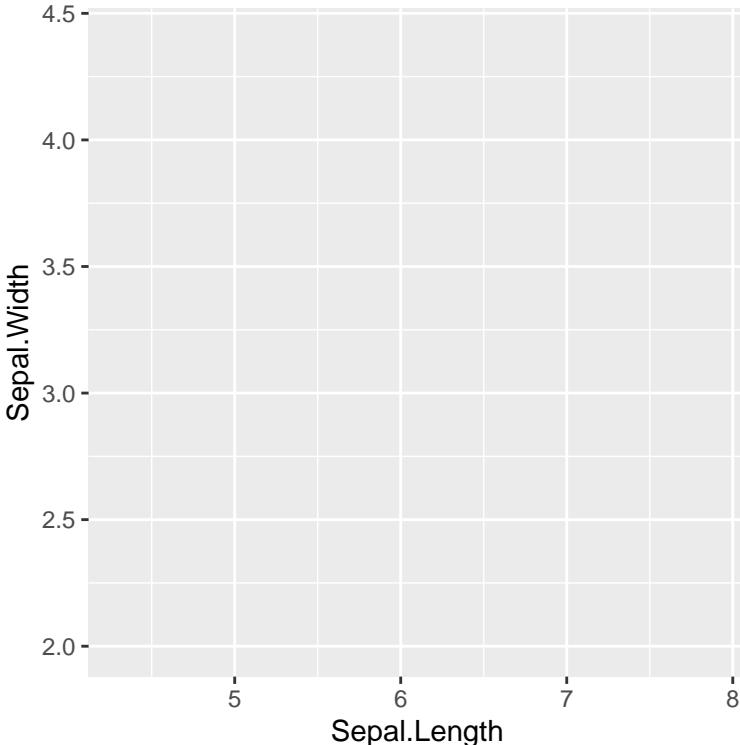
3.3 Vores første ggplot

Vi vil imidlertid fokusere på at lave et lignende plot med `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))
```



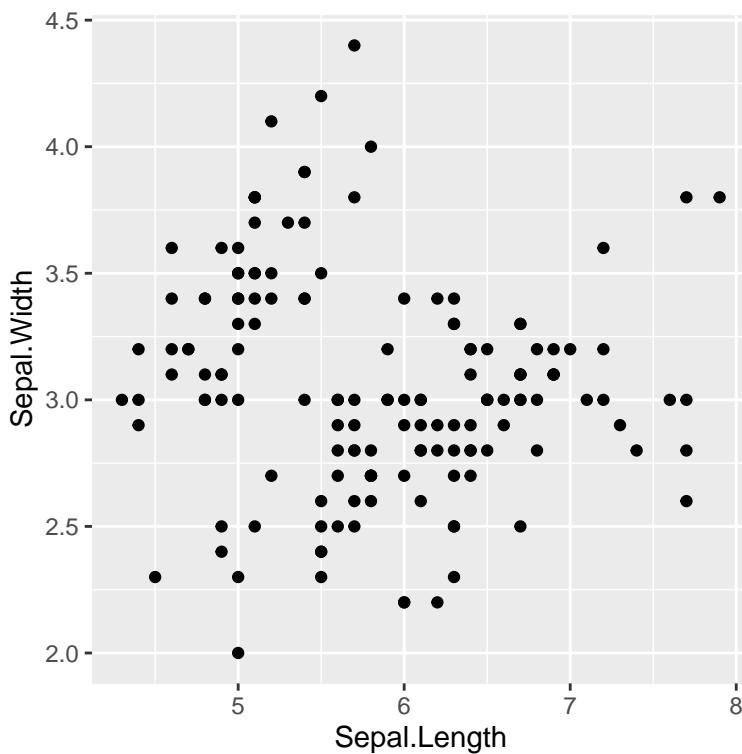
Koden fungerer, men bemærk at plottet er helt blank og derfor ikke særligt brugbart. Men der er blevet lavet et grundlag (se aksenavne osv.).

Det er blank fordi vi endnu ikke har fortalt, hvilket plot type det skal være - for eksempel søjediagram/bar plot, histogram, punktplot/scatter plot (og jeg benytter de engelske begreber hermed). Her vil vi gerne bruge et scatter plot, som betegnes af funktionen `geom_point()` i `ggplot2`. Vi forbinder derfor funk-

tionen `geom_point()` til den `ggplot()` funktion, vi allerede har specifiseret. Husk altid, at man bruger `+` til at forbinde de to “komponenter” (altså `ggplot()` og `geom_point()`) af plottet (ellers få vi et blank plot).

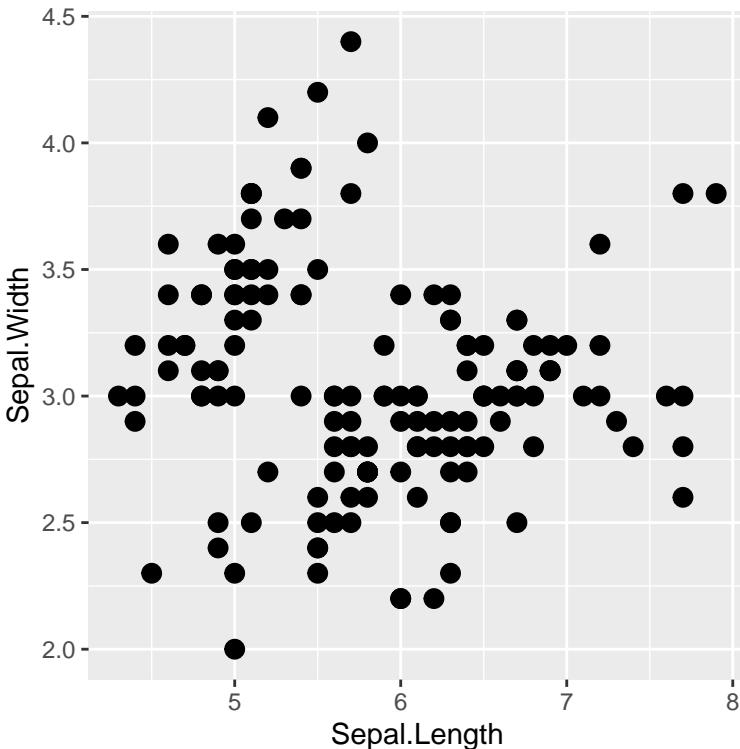
Koden for et meget simpelt scatter plot er således:

```
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 standard (prøve at tjekke `?geom_point()` for at se en liste overfor de mulige parametre, man kan ændre 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*, som vi kunne præsentere overfor andre. Her i **ggplot2** gøres det ved at tilføje flere komponenter ovenpå, med brugen af `+`, ligesom vi gjorde da vi tilføjede `geom_point()` til `ggplot()`. Det vil vi sikkert komme til at uddybe, men først 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 dataramme, der vi specificerer, alle de data vi skal bruge til at få lavet vores færdige figur. Til at gøre det til noget mere konkret, lad os sammenligne kodet mellem base-pakken og `ggplot()` til vores `iris` data. Her kan man bemærke, at i base-pakken specificerede vi direkte vektorer `iris$Sepal.Length` og `iris$Sepal.Width` som parametre `x` og `y`, der tager henholdsvis første og andens-plads i funktionen. Til gengæld i `ggplot()`, specificerer vi 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.

Efter vi har angivet vores dataframe, angiver vi indenfor funktionen `aes()`, hvilke variabler i datarammen, vi gerne vil visualisere. `aes()` er forkortelse af ‘aesthetic’ og det er her at vi fortæller `ggplot2` om de variabler der skal være i plottet - for eksempel her kan man specificere hvilken variable skal være på x-aksen og hvilken skal være på y-aksen.

Hvis vi eksempelvis har en dataramme med en `variabel1` og `variabel2`, kan specifikationen se ud som følger:

```
#not run, bare for at vise hvordan man bruger ggplot:
ggplot(dataramme, aes(x=variabel1, y=variabel2))
```

Her har vi således angivet, at vi er interesseret i `variabel1` på x-aksen og `variabel2` på y-aksen, samt at de er kolonner af en dataramme som hedder `dataramme`.

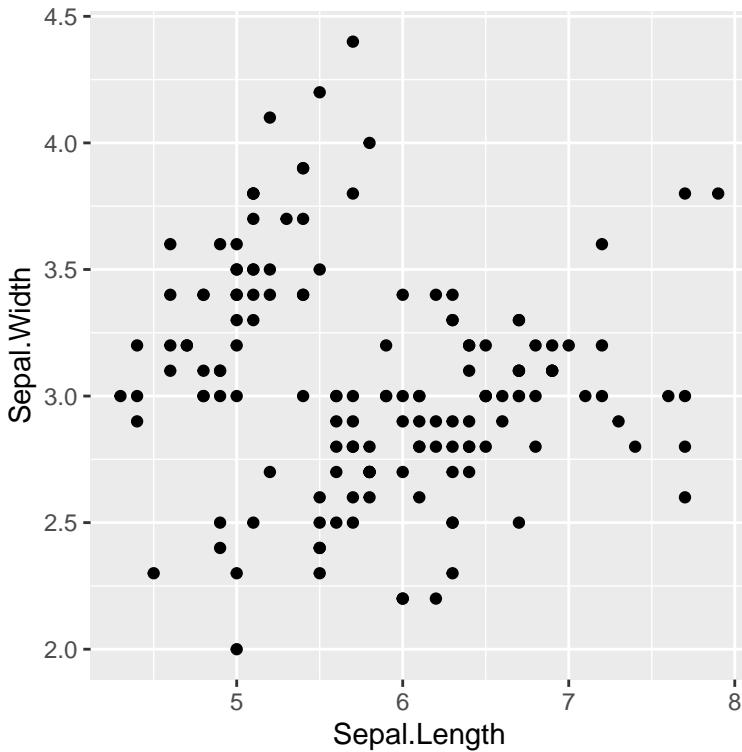
Næste kan kan vi tilføje hvilket plot type, vi ønsker at lave (i dette tilfælde et scatterplot), og forbinder med + tegn:

```
#not run, bare for at vise hvordan man bruger ggplot:
ggplot(dataramme, aes(x=variabel1, y=variabel2)) +
  geom_point()
```

3.4.2 Globale versus lokale æstetik

De fleste gange bruger man funktionen `aes()` indenfor `ggplot()`, med betydningen, at de variabler specificeret indenfor `aes()` gælder globalt over alle komponenter i plottet. Man kan faktisk også skrive `aes()` lokale indenfor `geom_point()` (eller andet geom funktion):

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

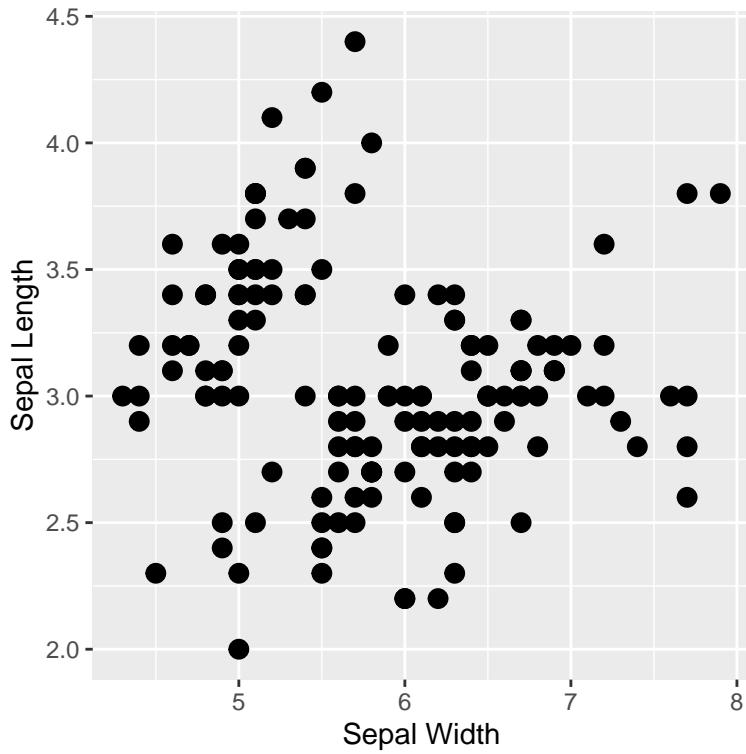


Vi får det samme plot som før, men det er kun `geom_point()` der er påvirket af specificeringen indenfor `aes()`. I simpel situationer som ovenpå er der ingen forskel, men når man har mange forskellige komponenter i spil, så kan det give mening at bruge lokale æstetik.

3.5 Specificere axse tekst og titel

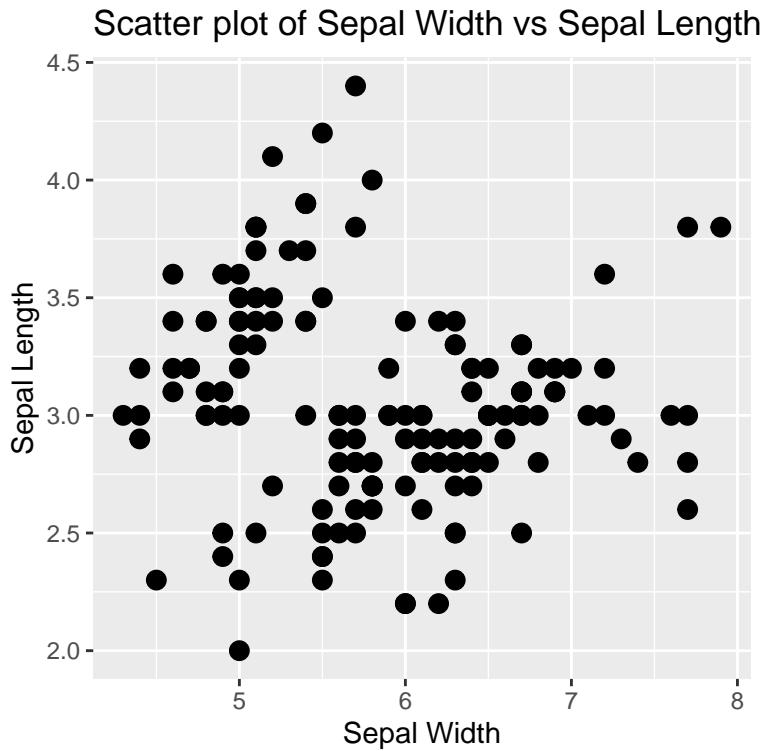
Vi tager udgangspunkt i plottet, vi lavet i ovenstående og prøver at gøre det bedre, ved at tilføje nye etiketter og en titel. I ggplot kan man opdatere y-akse og x-akse etiketter ved at bruge henholdsvis `ylab` og `xlab`:

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



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

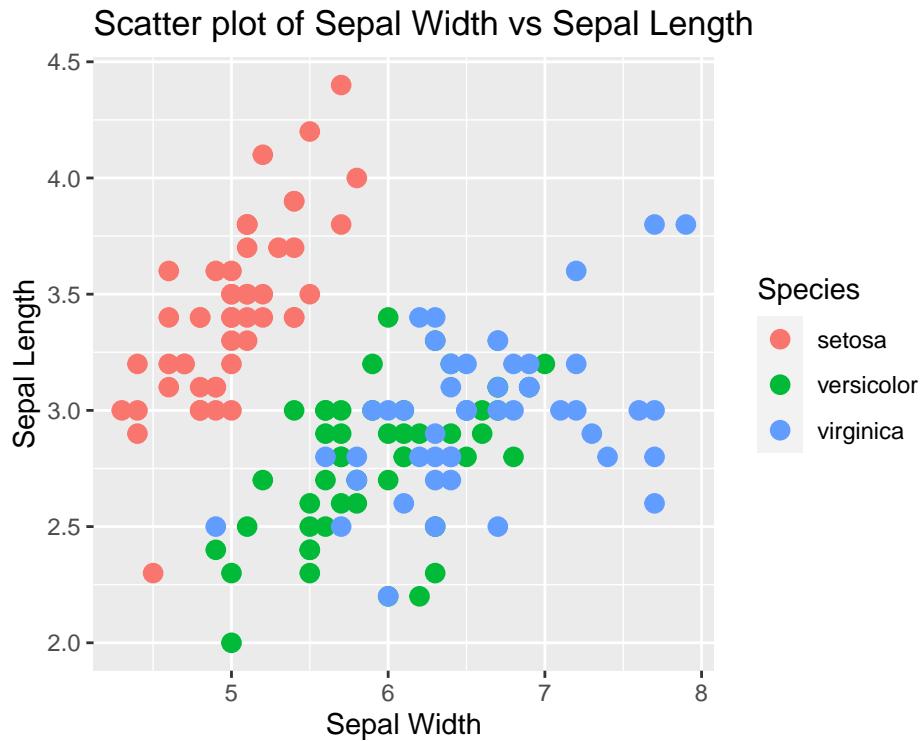
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point(size=3) +  
  ylab("Sepal Length") +  
  xlab("Sepal Width") +  
  ggtitle("Scatter plot of Sepal Width vs Sepal Length")
```



3.6 Ændre farver

I `ggplot2` kan man specificere “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, 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.

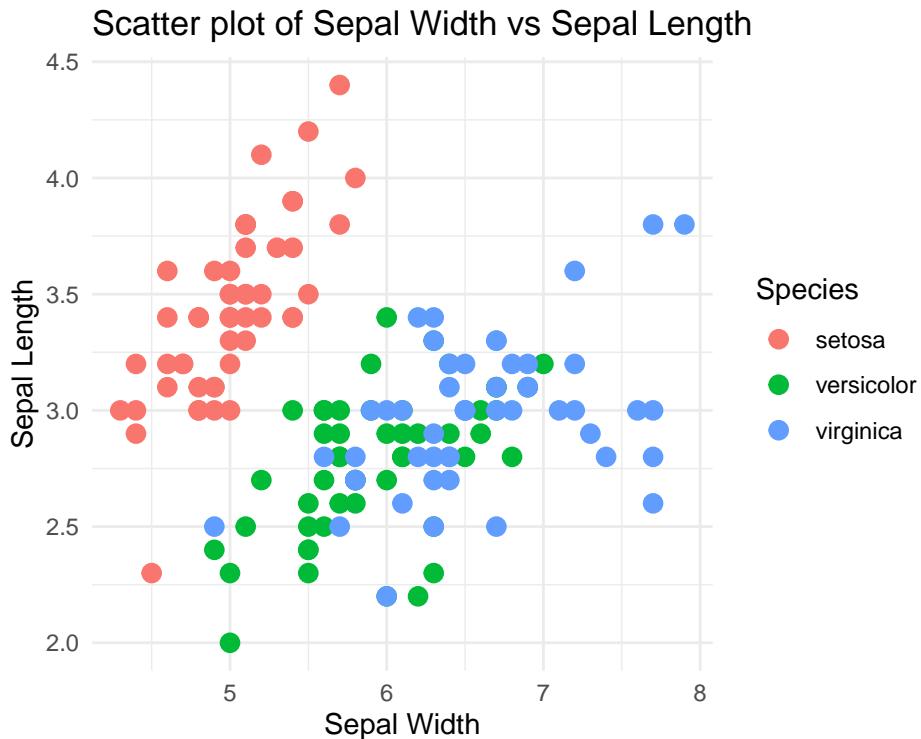
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size=3) +
  ylab("Sepal Length") +
  xlab("Sepal Width") +
  ggtitle("Scatter plot of Sepal Width vs Sepal Length")
```



3.7 Ændre temaet

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) +
  ylab("Sepal Length") +
  xlab("Sepal Width") +
  ggtitle("Scatter plot of Sepal Width vs Sepal Length") +
  theme_minimal()
```



Her er nogle ekslempel på mulige temae du kan bruge i dine plots (det er dog generelt op til dig).

theme
 theme_grey()
 theme_classic()
 theme_bw()
 theme_dark()
 theme_minimal()
 theme_light()

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

3.8 Forskellige type plots

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":

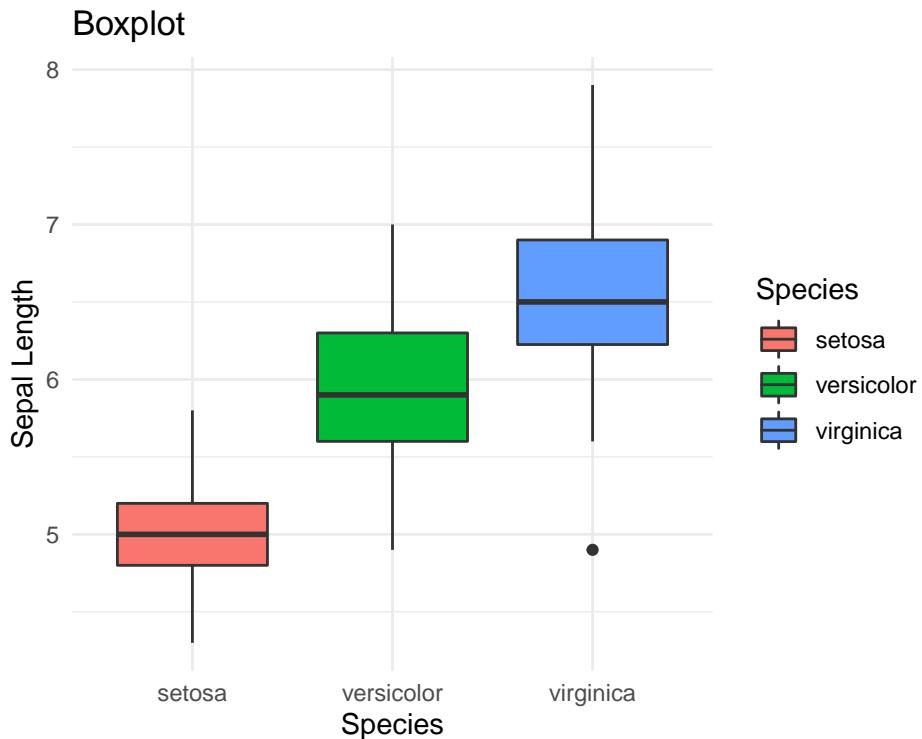
geom	plot
<code>geom_point()</code>	scatter plot
<code>geom_bar()</code>	barplot
<code>geom_boxplot()</code>	boxplot
<code>geom_histogram()</code>	histogram
<code>geom_density()</code>	density

For at lave disse plot typer, skal man tillægge funktionen til den `ggplot()` kommando med `+`, ligesom vi gjorde med `geom_point()`. Der kan dog være for nogle plot-type specifikke overvejelser som er værd at vide, indenfor man selv prøver dem. Jeg anbefaler at du første går i gang med problemstillingerne og refererer tilbage til den plot type at du bliver bedt om lave i kursusnotaterne.

3.8.1 Boxplot (`geom_box`)

For at lave et boxplot og `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`.

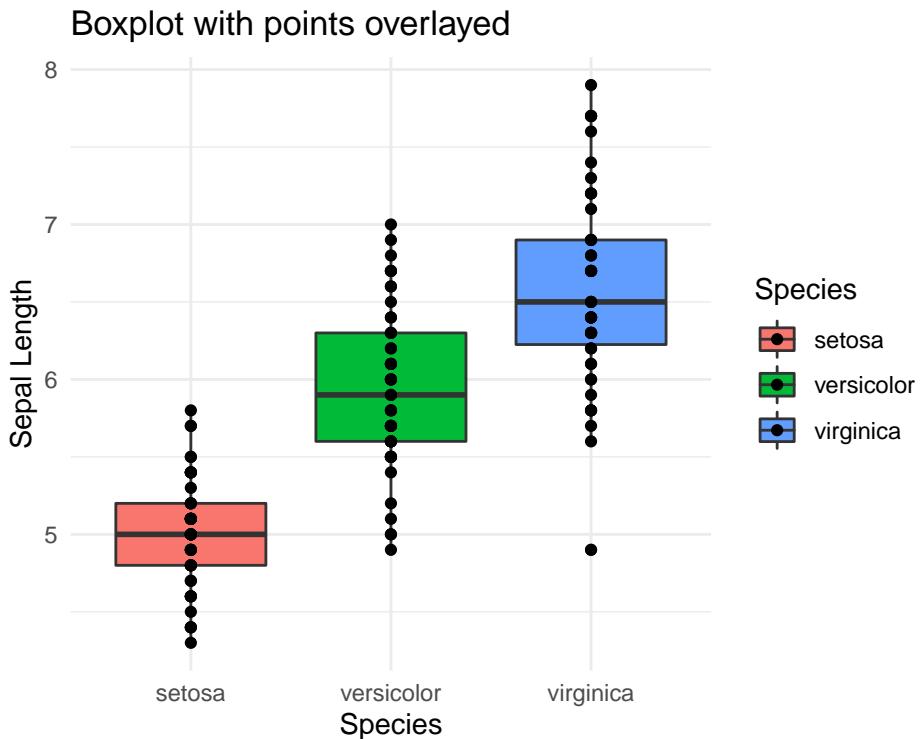
```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  ylab("Sepal Length") +
  ggtitle("Boxplot") +
  theme_minimal()
```



Lave punkter ovenpå

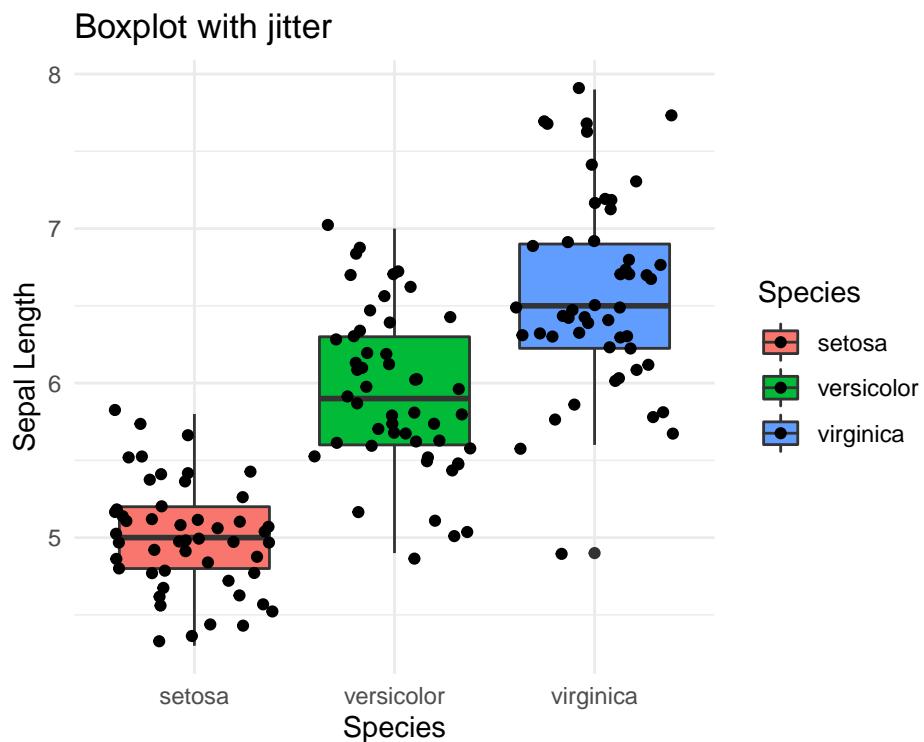
Det kan ofte være nyttigt at plotte de egentlige data punkter ovenpå boxplottet, så I kan se både fordelingen i de data samt de rå data. En løsning er at benytte `geom_point()` ved at tilføje det som komponent over vores eksisterende kode.

```
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()
```



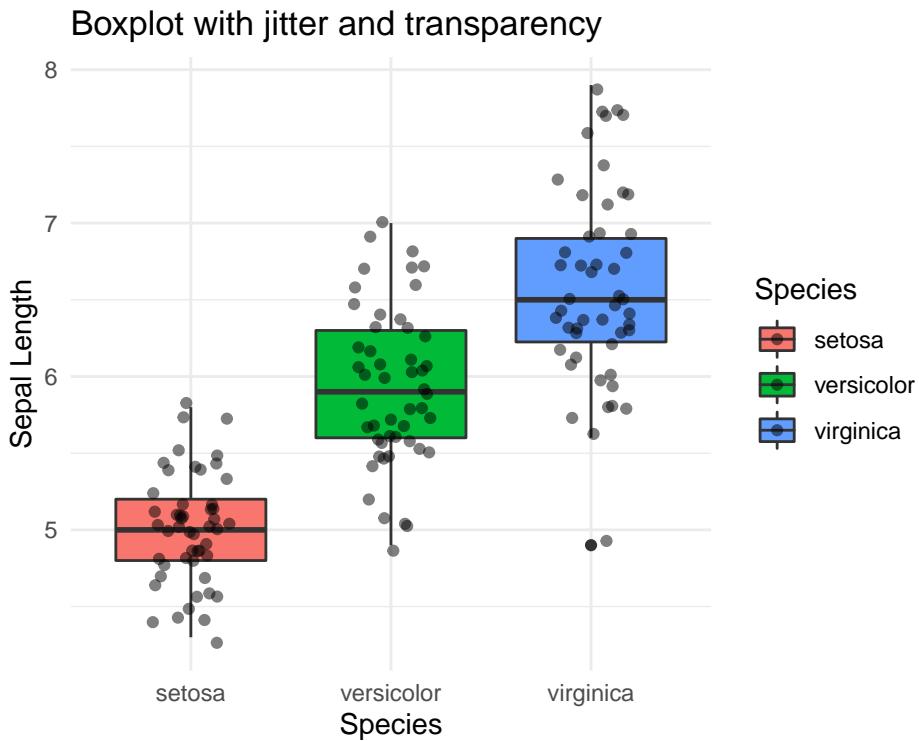
Man kan dog se, at det ikke er særlig informativ, da alle punkter er på den samme lodrette linje. Hvis vi har mange punkter med samme eller næsten samme værdier, så kan vi ikke se de fleste af dem i plottet. En bedre løsning er at indføre noget tilfældighed i punkterne langt x-aksen, så at man mere tydelige 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.

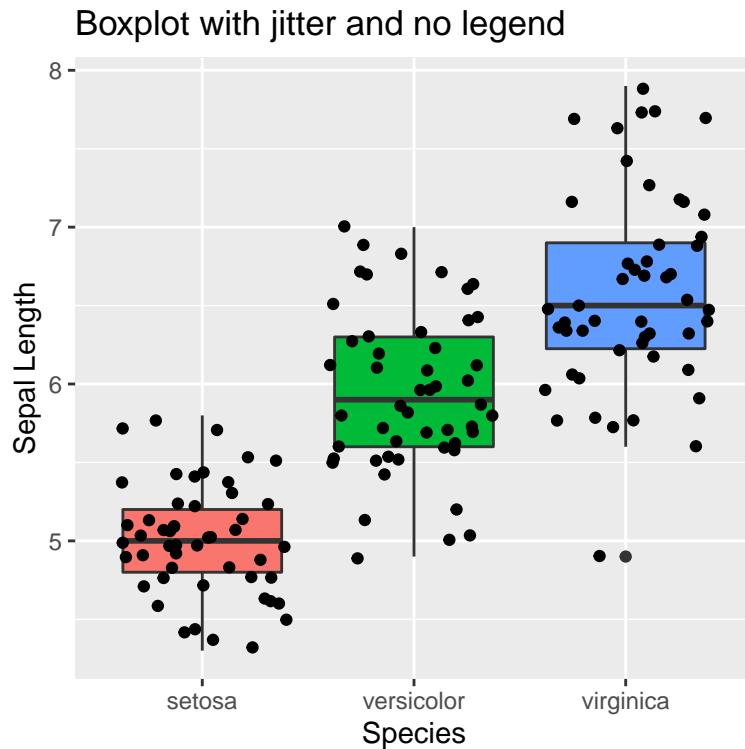
```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  geom_jitter(alpha=0.5, width=0.2) +
  ylab("Sepal Length") +
  ggtitle("Boxplot with jitter and transparency") +
  theme_minimal()
```



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")
```

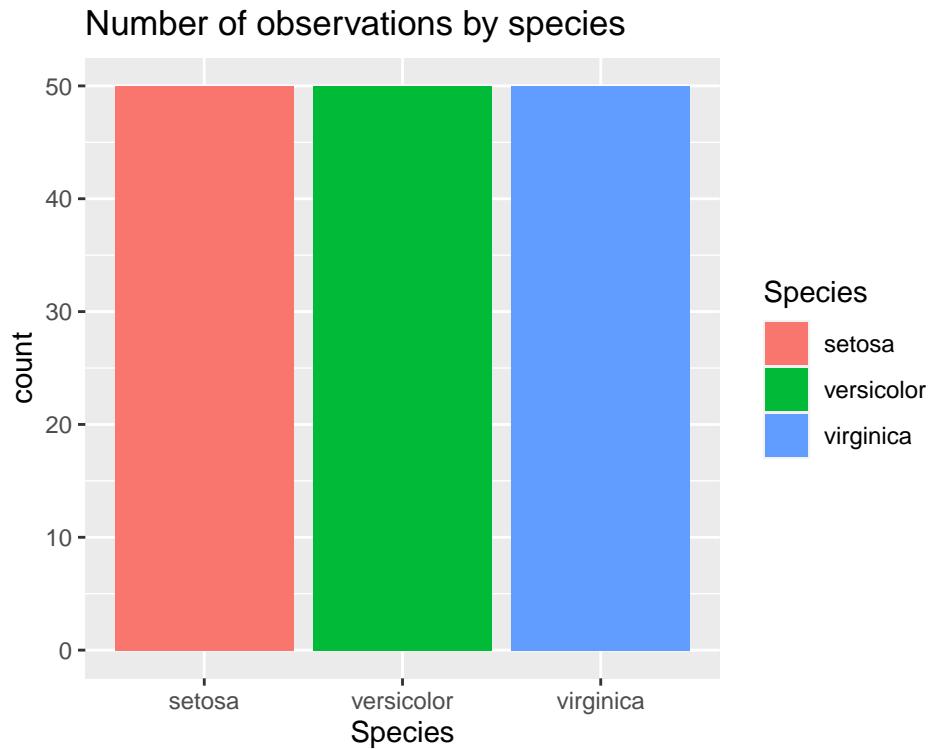


3.8.2 Barplot (geom_bar)

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 visualerer dem således som sjæler (eller bare "bars") Indenfor `geom_bar()` specificerer vi således `stat="count"`, som er den mest almindelige brugt mulighed.

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

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



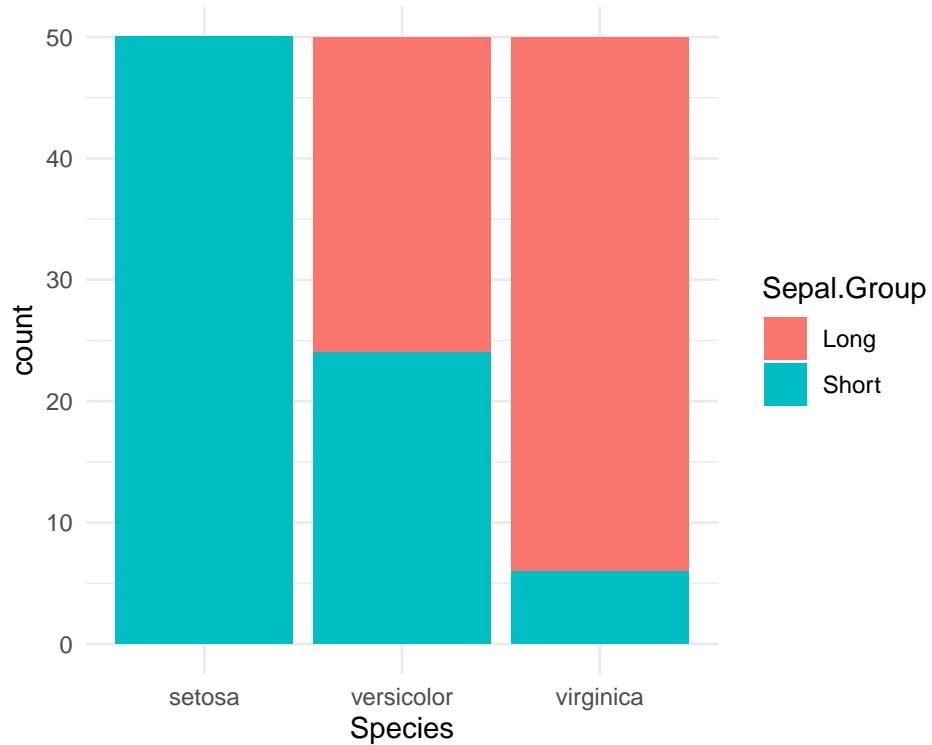
Barplot: stack vs dodge

Hvis man har flere gruppe variabler (altså `Species` og måske en anden), kan man lave bar plotte på forskellige måder. Da vi ikke har en ekstra gruppe variabel, jeg laver 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 `Long`, ellers er det `Short`.

Hvis jeg lave en barplot med de to variabler, kan jeg tilføje `Sepal.Group` som `fill`, og `ggplot` splitter de antal observationer efter `Sepal.Group` med farver som repræsenterer `Sepal.Group` og tilføjer en tilsvarende legende.

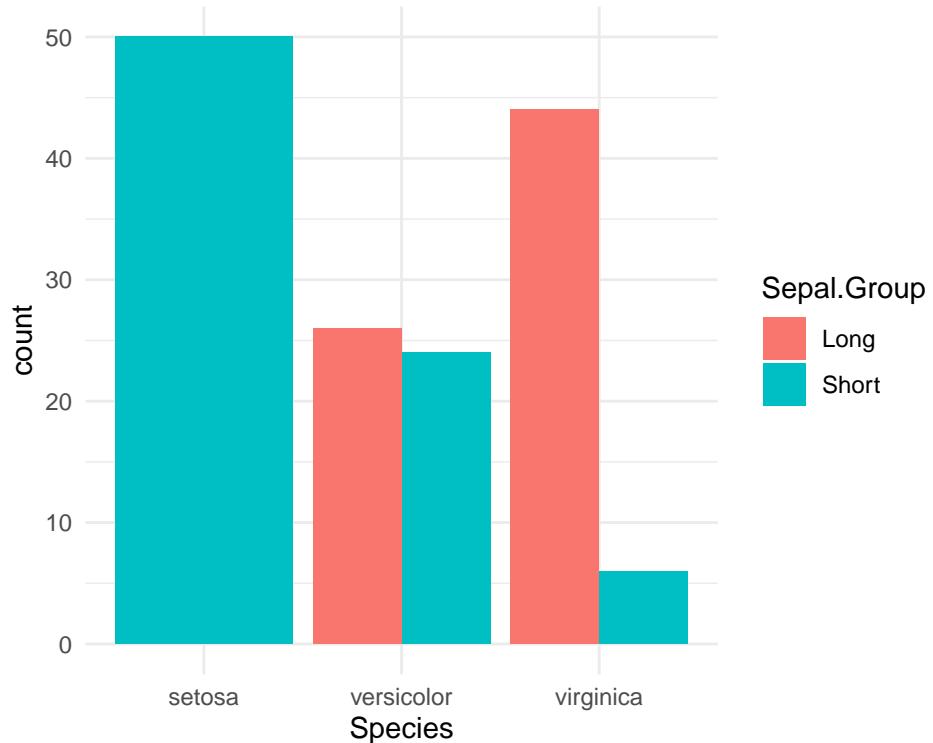
```
iris$Sepal.Group <- ifelse(iris$Sepal.Length>mean(iris$Sepal.Length), "Long", "Short")

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



Mange gange vil vi hellere få bars som stå ved siden af hinanden. Det kan vi specificere med blot at tilføje `position="dodge"` ind i `geom_bar()`.

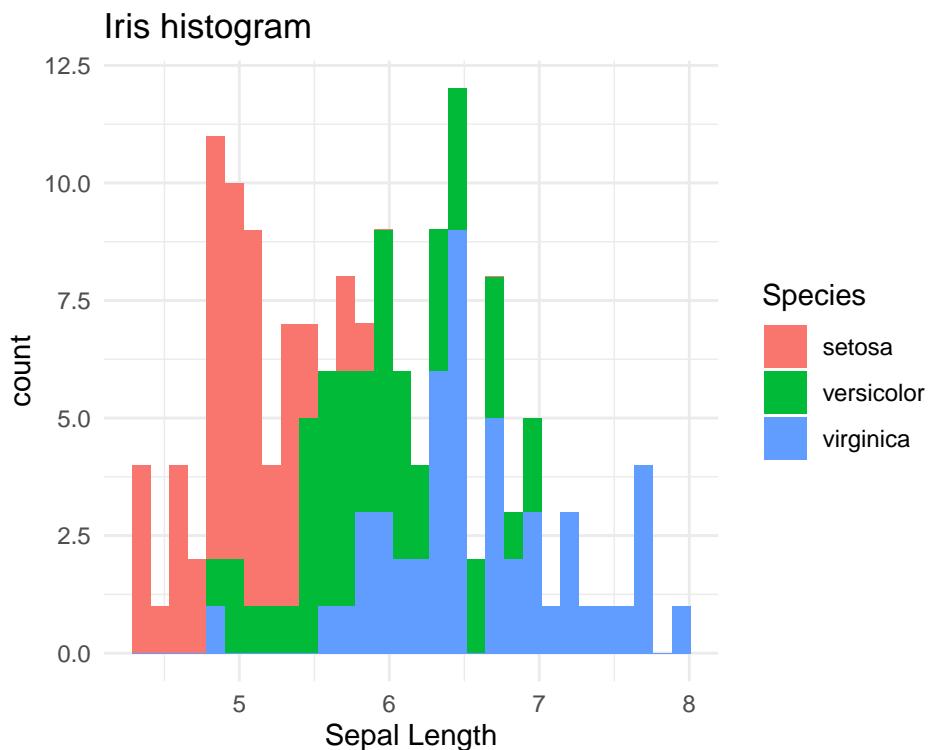
```
ggplot(iris,aes(x=Species,fill=Sepal.Group)) +  
  geom_bar(stat="count",position="dodge") +  
  theme_minimal()
```



3.8.3 Histogram (geom_histogram)

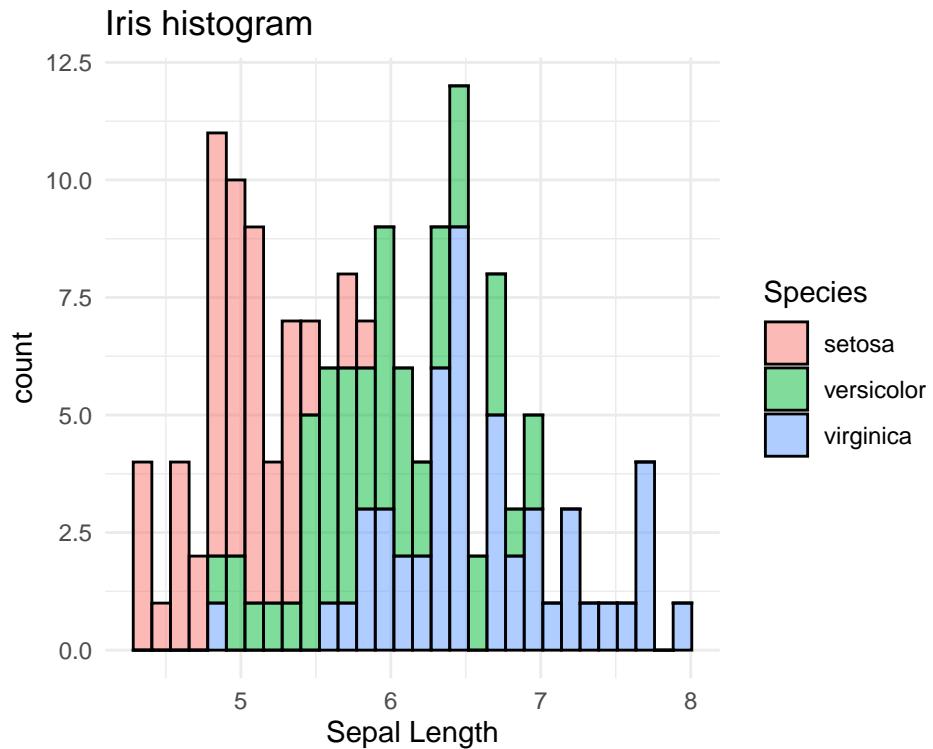
En histogram bruges til at få overblik over hvordan data fordeler sig. I ggplot2 kan man lave en histogram med `geom_histogram()`. Den x-akse variable skal være en ‘continuous’ variable. Her specificerer vi, at vi gerne vil have et histogram for hver Species.

```
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()`.

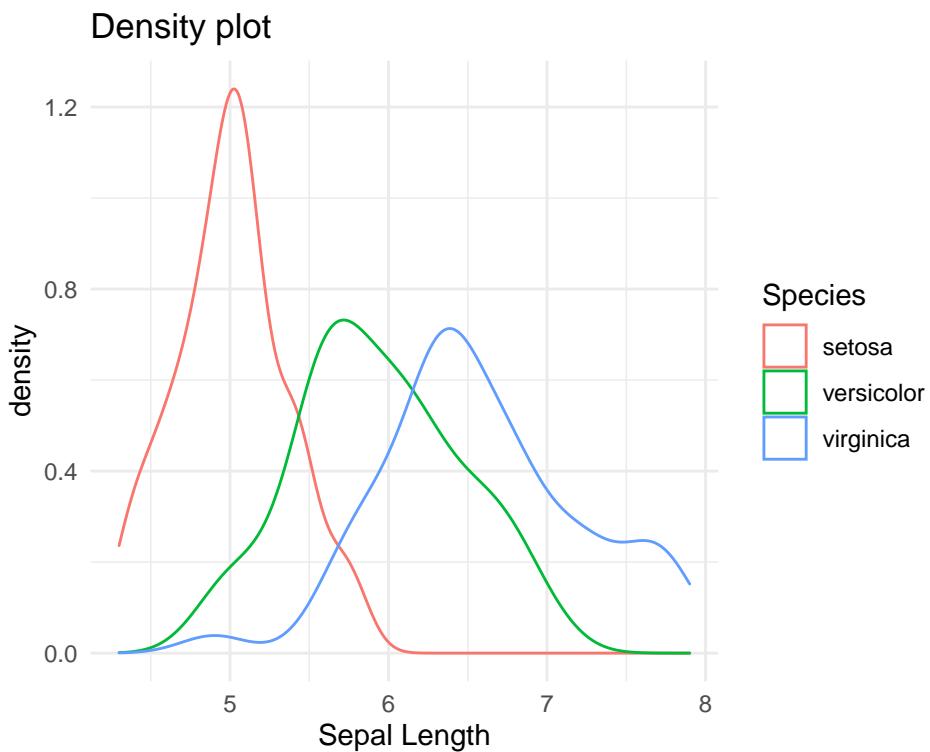
```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram(alpha=0.5, color="black") +
  xlab("Sepal Length") +
  ggtitle("Iris histogram") +
  theme_minimal()
```



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()
```

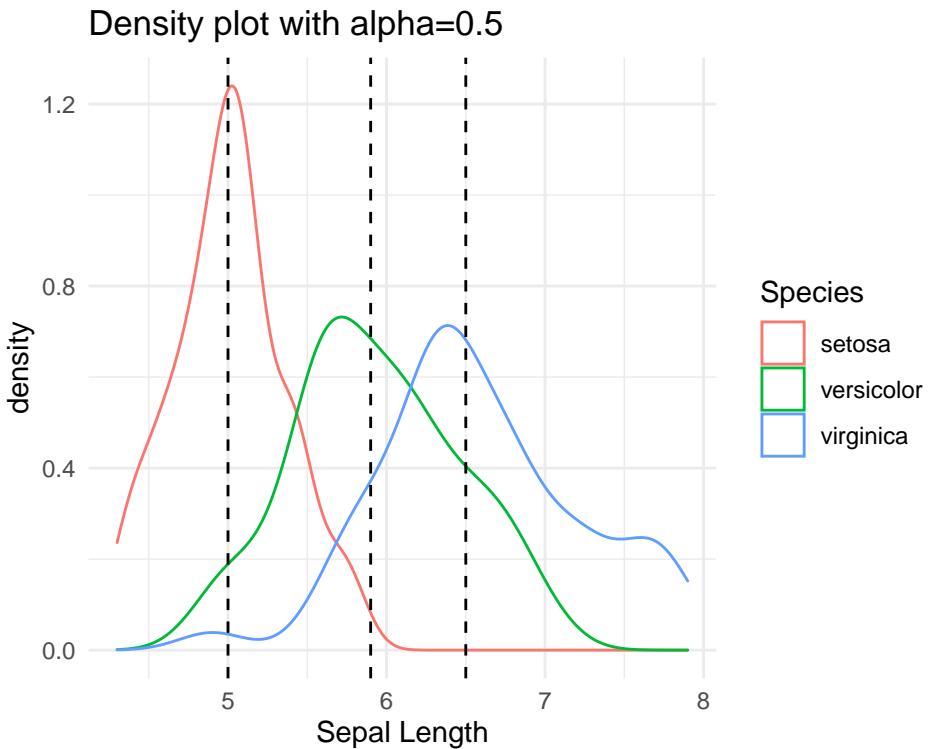


Tilføje middelværdi linjer

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 to 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.
- Den forkerte 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 fk. bars eller regioner med farver, og `colour` referere til farven af linjer eller punkter.

3.10 Problemstillinger

1) Quiz på Absalon - det 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 afdleverer 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 over også med at lave de tre plot typer, med labeller og titler. Vi bruger datasættet der hedder `diamonds`. Huske at først indlæse de data:*

```
data(diamonds)
```

Her er beskrivelsen af `diamonds`:

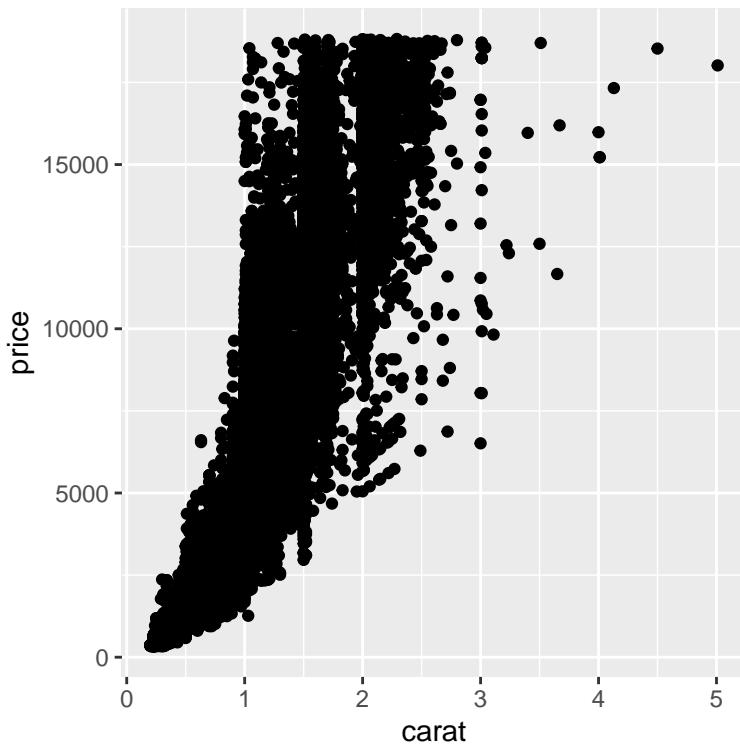
Prices of over 50,000 round cut diamonds: a dataset containing the prices and other attributes of almost 54,000 diamonds.

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 I har noget at sammenligne med, skal det se sådan ud:

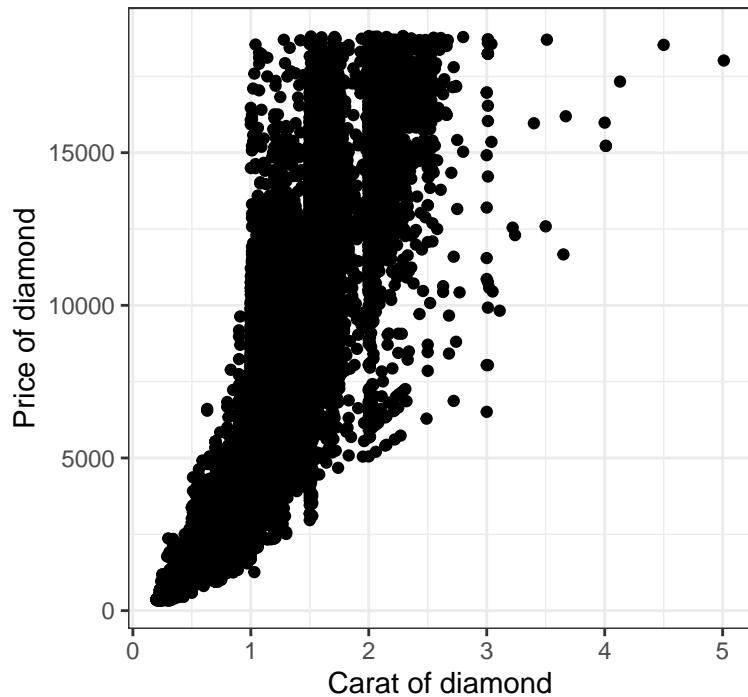


3) Tilføj nogle komponenter til dit plot fra 2).

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

Det skal se sådan ud:

Scatter plot of caret and price



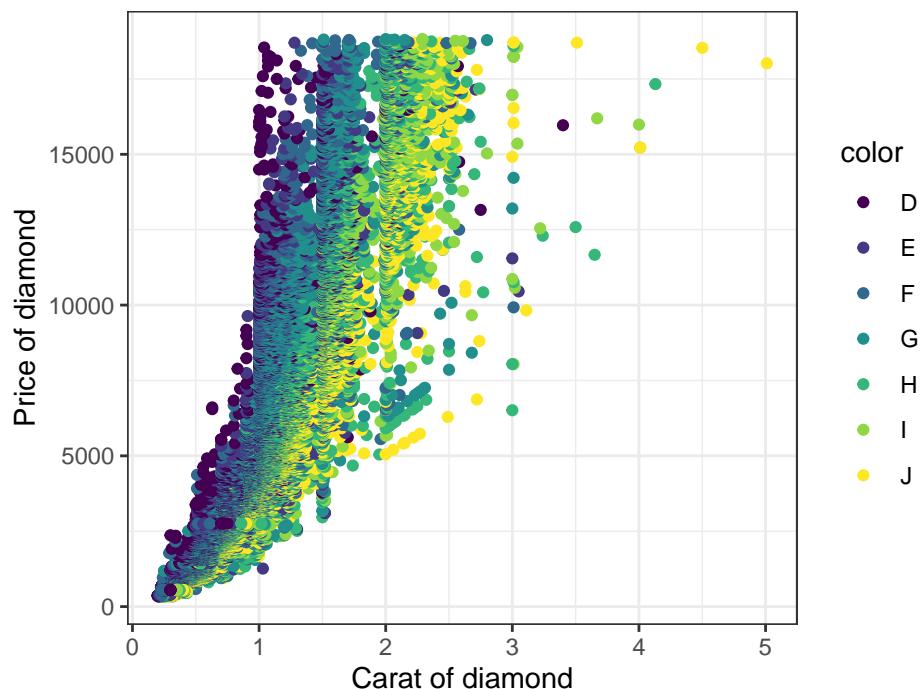
- 4) Ændre temaet til `theme_classic()` eller `theme_minimal()` i stedet for `theme_bw()` og kig på resultatet.

Ekstra: Hvis man (måske ved uheld) skriver ind `to` temaeer på samme tid (for eksempel `+ theme_bw() + theme_classic()`) - hvilke tema får man så i plotten?

Valgfri ekstra: her er nogle flere tema man kan prøve: <https://ggplot2.tidyverse.org/reference/ggtheme.html>

- 5) Lave det samme plot som i 4), med `color=color` indenfor `aes()`. Den første `color` refererer til punkt farver og den anden til en variable i datarammen der hedder `color`.

Scatter plot of carat and price

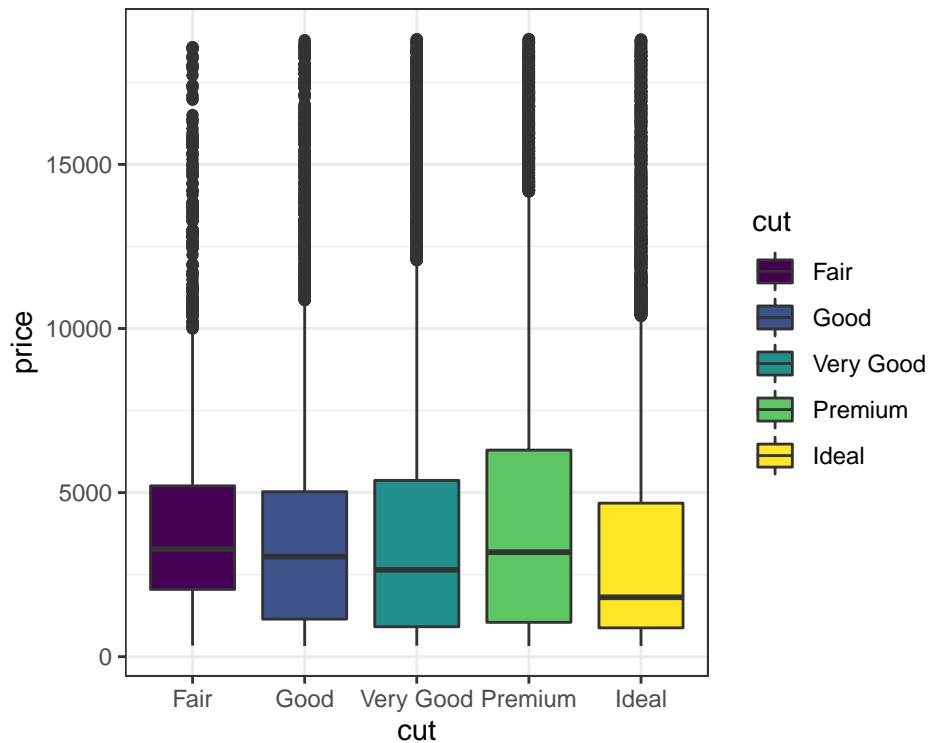


Det skal se sådan ud:

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:

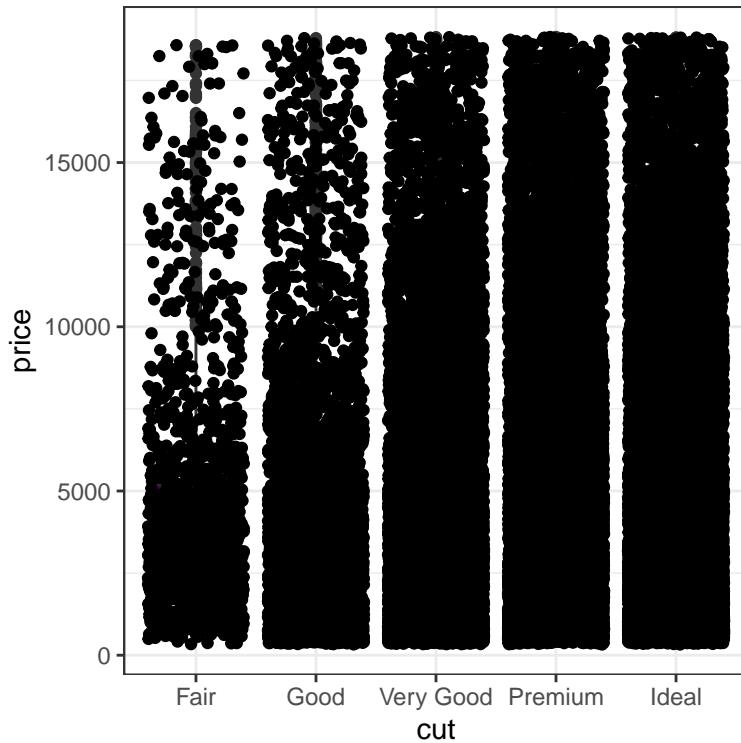


Ekstra: hvordan ser det ud, hvis man bruger `color` i stedet for `fill`?

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

- Tilføj `geom_jitter()` til din boxplot
- fjern de legend (se notaterne).

Det skal se sådan ud:



Eksstra 1: 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 bare fjerne dem).

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

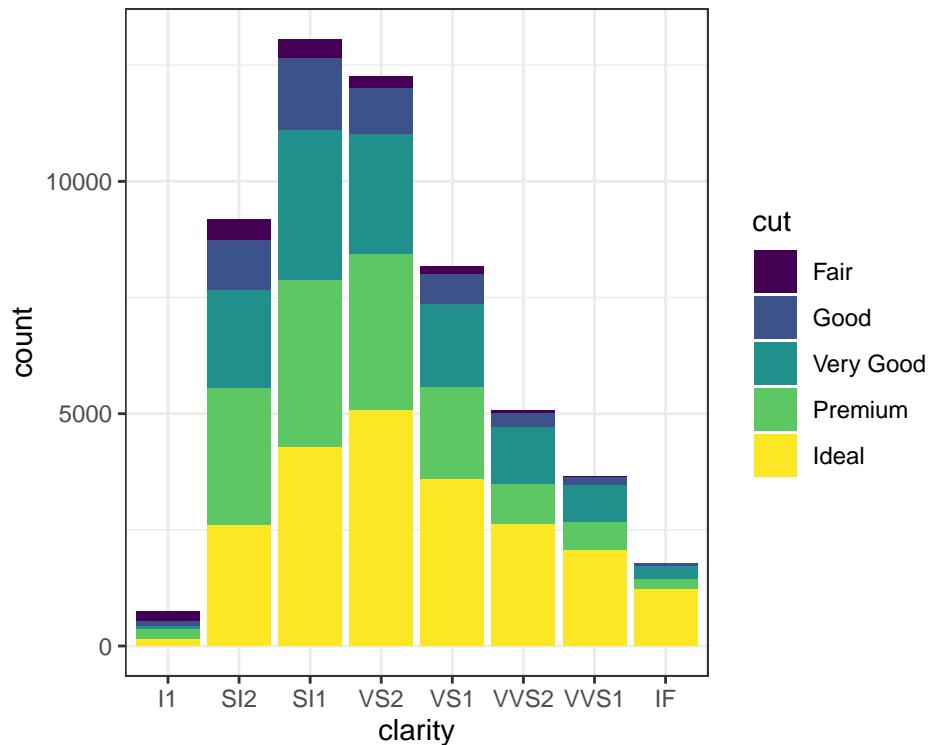
Alpha refers to the opacity of a geom. Values of alpha range from 0 to 1, with lower values corresponding to more transparent colors. https://ggplot2.tidyverse.org/reference/aes_colour_fill_alpha.html

Eksstra 2: Prøve også at skifte rækkefølgerne af `geom_jitter()` og `geom_boxplot()` i den plot kode og se - - gøre det en forskel til hvordan plottet ser ud?

8)

Lave en bar plot med `stat="count"`:

- Variable `clarity` på x-aksen
- Forskellige farver til den gruppe-variable `cut`
- Specifierer `position="dodge"` for at få bars ved siden af hinanden
- Tilføj et tema



9) 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`.

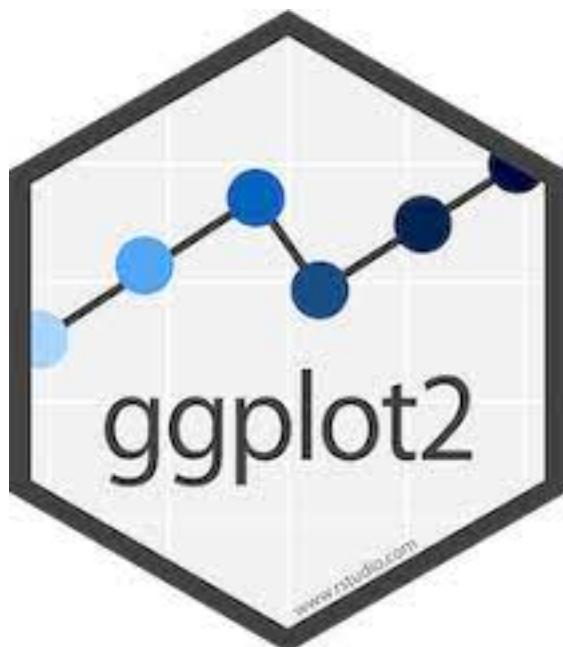
I alle tilfælde tilføje akse-labs, en titel, et tema osv.

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 fx. sætte farver manuelt.

Chapter 4

Visualisering - ggplot2 dag 2



4.1 Indledning og videoer

I dag udvider vi værktøjskassen af kommandoer i **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 lave yderligere opdeling af plots.
- Gemme dit færdigt plot i en fil.

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

4.1.2 Video ressourcer

- Video 2: Koordinat systemer

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

- Video 3: Farver og punkt former

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

- Video 4: Labels - `geom_text()` og `geom_text_repel()`

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

4.2 Koordinat systemer

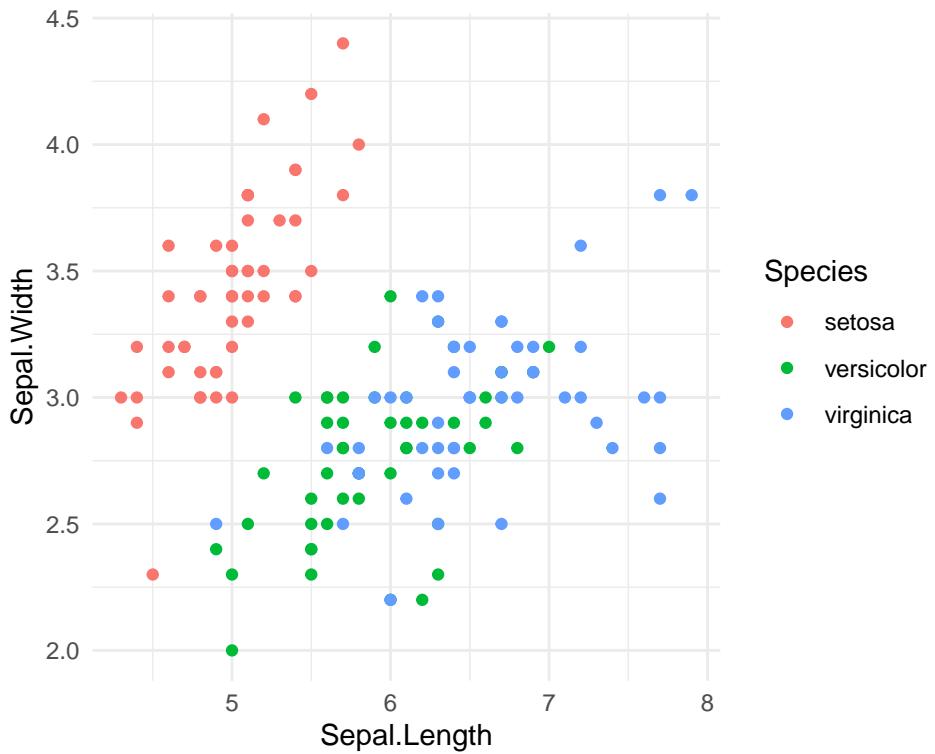
Her arbejder vi videre med koordinater i pakken `ggplot2`.

4.2.1 Zoom på plottet (`coord_cartesian()`)

Man kan bruge funktionen `coord_cartesian()` til at zoome ind på et bestemt område i plottet. Indenfor `coord_cartesian()` angiver man `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, og så får man en advarsel. Med `coord_cartesian()` beholder man til gengæld samtlige data, og man får således ikke en advarsel.

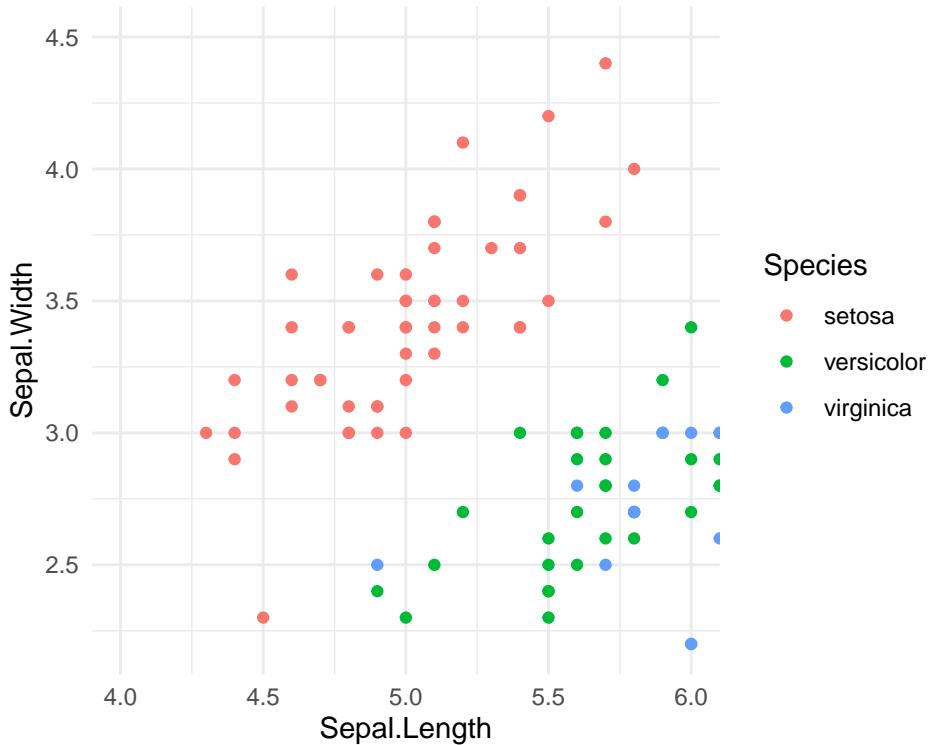
Her ses vores oprindeligt scatter plot:

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



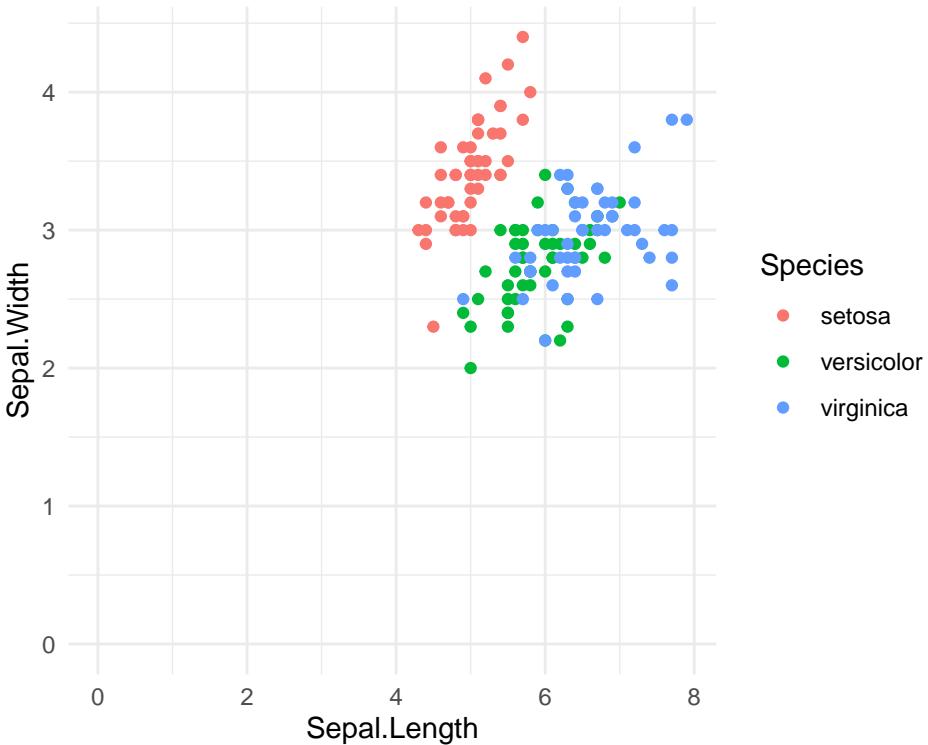
Og her bruger vi funktionen `coord_cartesian()` med `xlim()` og `ylim()` indenfor til at zoome ind på et ønsket område på plottet.

```
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()
```



Hvis man har lyst til det kan man 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 på plottet:

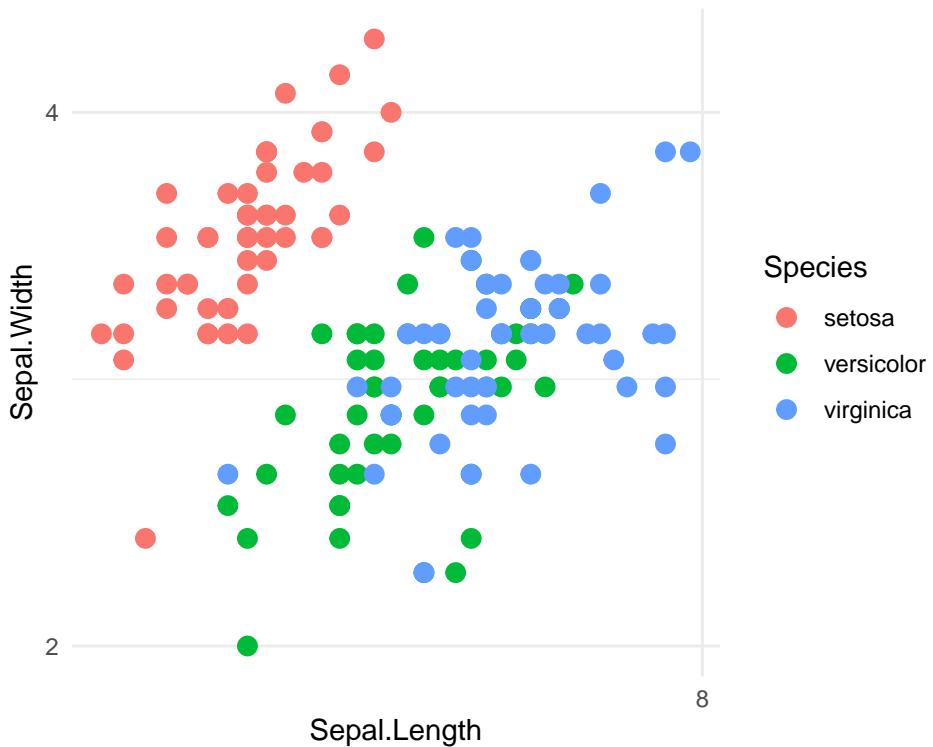
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point() +
  expand_limits(x = 0, y = 0) +
  theme_minimal()
```



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 `log` og `sqrt` på x-aksen og y-aksen. Vi kan bruge disse til at transformere skaleringen på akserne, 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()
```



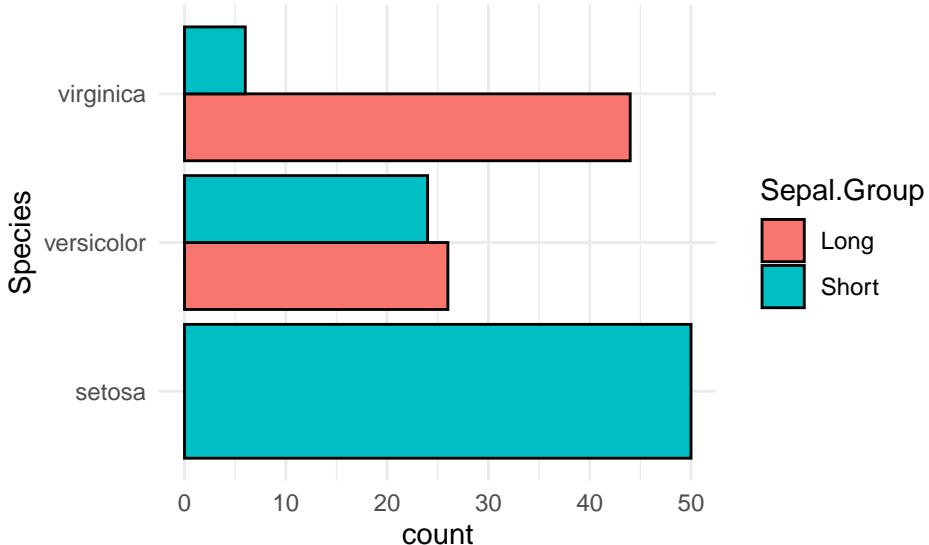
Man kan også prøve fx. "sqrt" i stedet for "log2". Formålet er, at hvis de data fordeler sig mere 'normalt', kan man nemmere visualisere det i et plot - en måde til at gøre der er ved at transformere de data med "sqrt" eller "log2".

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).

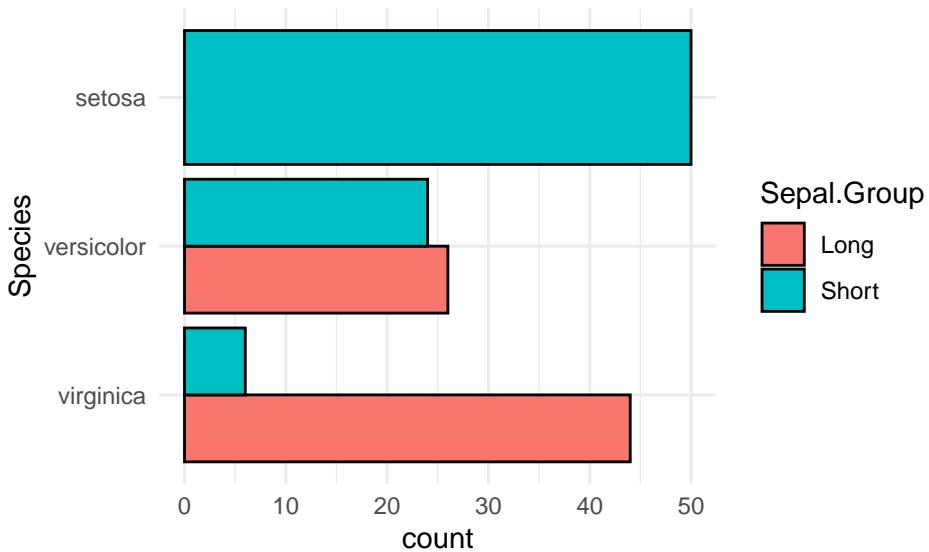
```
#Sepal.Group defineret som i går
iris$Sepal.Group <- ifelse(iris$Sepal.Length>mean(iris$Sepal.Length), "Long", "Short")

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 angive 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()
```



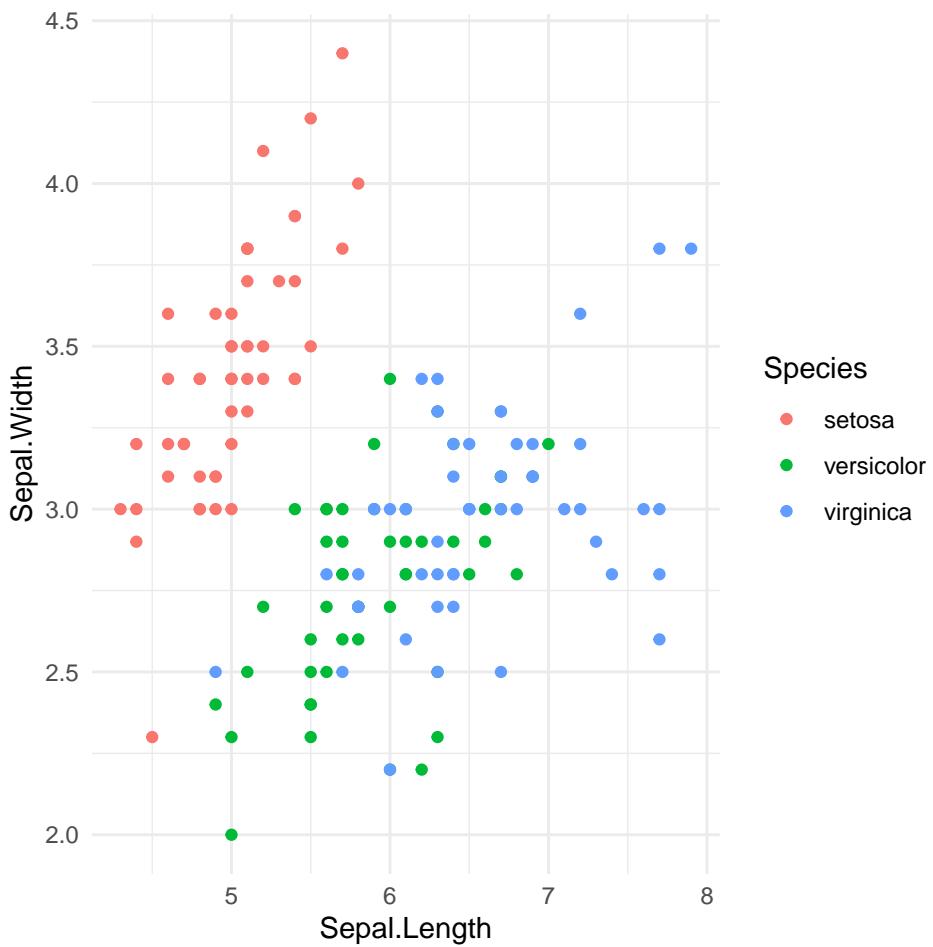
4.3 Mere om farver og punkt former

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

4.3.1 Automatisk farver

Man kan automatisk specificere at vi gerne vil have forskellige farver, ligesom vi gjorde i går med `colour=Species` indenfor `aes()` i ggplot funktion.

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

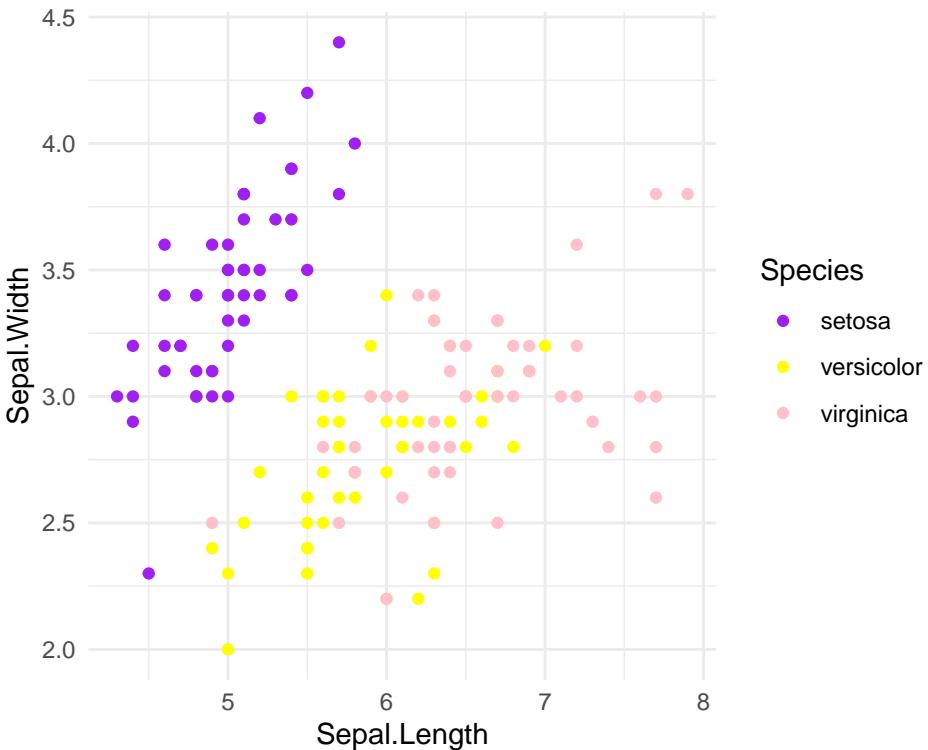


4.3.2 Farver manuelt

Hvis man foretrækker at specificere sine egne farver, kan man det ved at benytte `scale_colour_manual`.

#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()
```

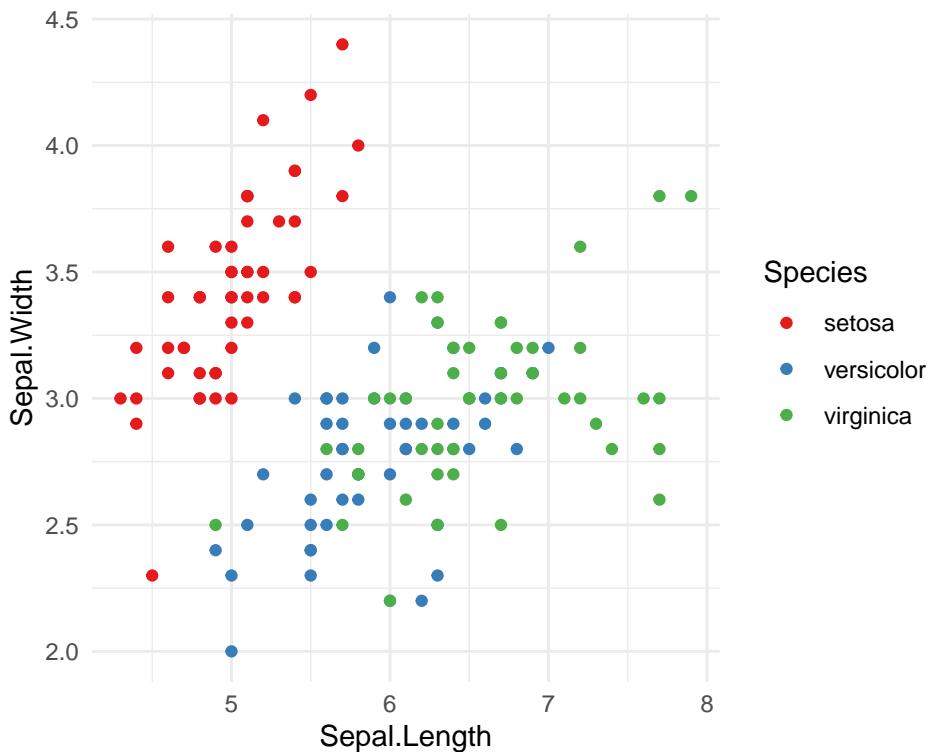


Man kan også bruge farver fra `RColorBrewer`-pakken. Pakken indeholder mange forskellige “colour palettes”, som er grupper af farver som passer godt med hinanden, så at man kan slippe for at vælge den bedste kombination af farver til deres plot. De palettes tager også i betragtning, f.eks. hvis man er farveblind, eller om man vil have en farvegradient eller et sæt diskrete farver som ikke ligner hinanden.

Der er andre pakke som også har andre colour palettes som man kan bruge, som jeg anbefaler at I tjekke ud på Google hvis interesseret.

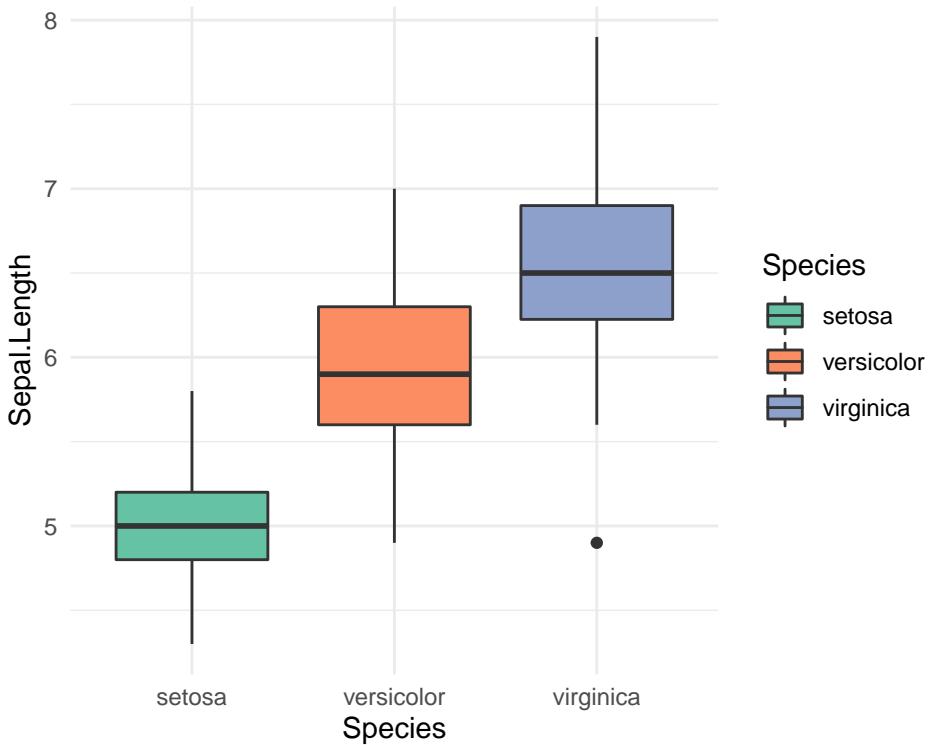
```
#install.packages("RColorBrewer")
library(RColorBrewer)
```

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



Bemærk, at `scale_color_brewer()` eller `scale_color_manual()` sætter farver af punkt og linjer, mens i en boxplot eller barplot sammenhænge, ligesom man specificerede `fill=Species` indenfor `aes()`, bruger man `scale_fill_manual()` eller `scale_fill_brewer()` i tilfældet af RColourBrewer.

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



Her er en tabel over de fire funktioner.

funktion	beskrivelse
scale_fill_manual(values=c("firebrick","darkblue"))	til boxplots og barplots osv.
scale_color_maual(values=c("darkorange","cyan"))	til punkter og linjer osv.
scale_fill_brewer(palette="Dark2")	RColourBrewer løsning til boxplots/barplots/osv.
scale_color_brewer(palette="Set1")	RColourBrewer løsning til punkter og linjer osv.

Der er også andre, for eksempel for continuous scala kan man google efter `scale_fill_gradient`.

Farver i RColourBrewer

Bare som reference for de forskellige farver tilgængelige i pakken **RColourBrewer**.

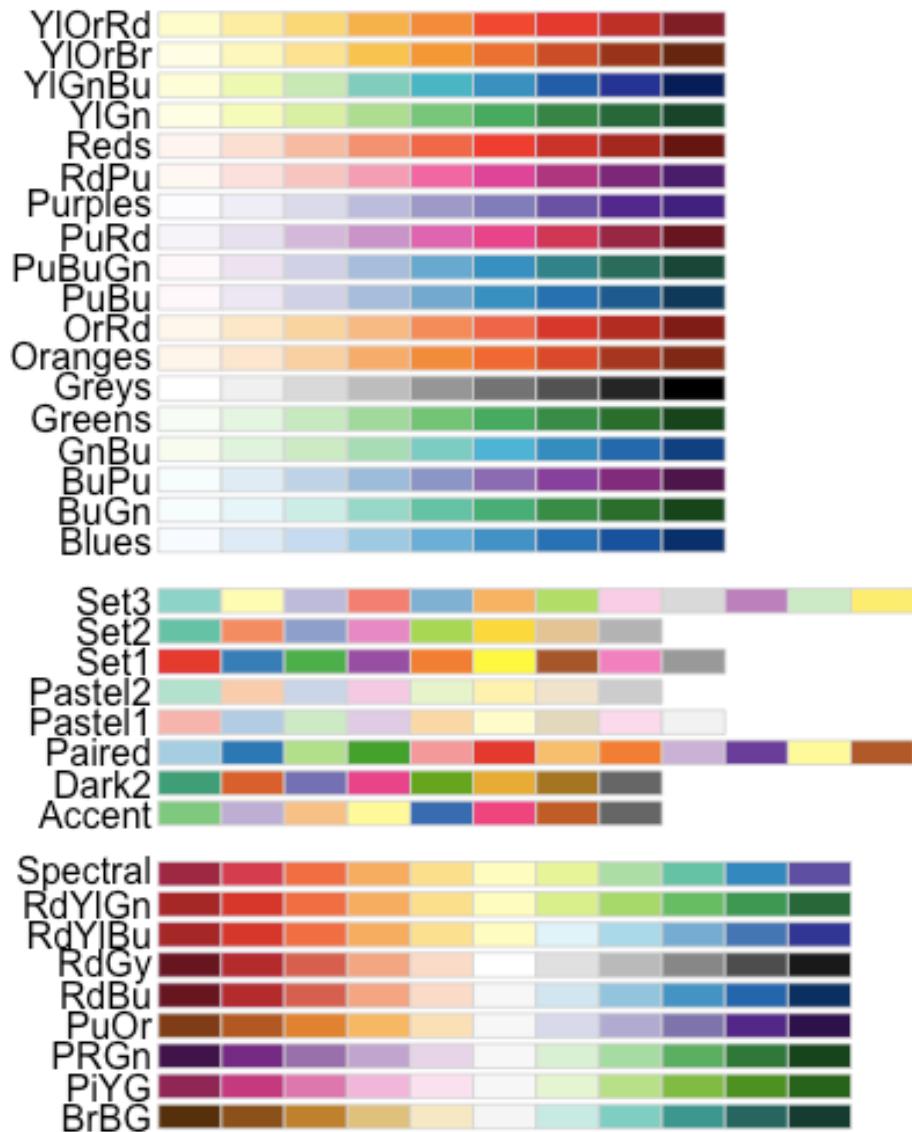


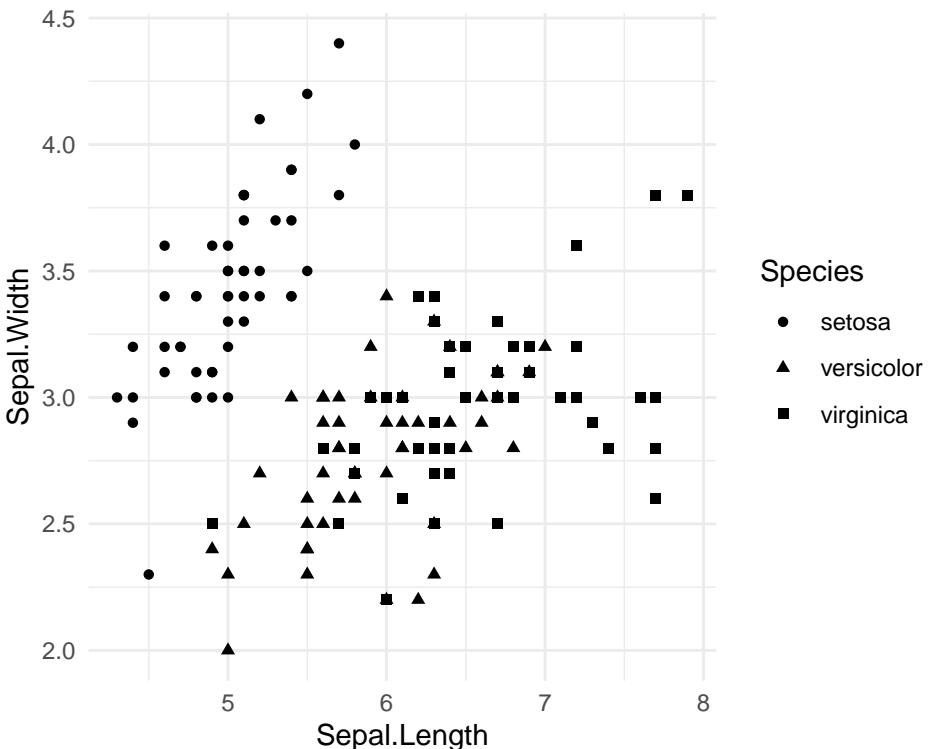
Figure 4.1: Mulige colour palettes tilgængelige i RColourBrewer

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 specificerer, skal vi skrive indenfor `aes()`. Her, da `shape` er en parameter som er meget specifik til `geom_point`, her vælger vi at sætte den indenfor en ny `aes()` indenfor `geom_point()` i stedet for indenfor `ggplot()`. Husk, at i funktionen `ggplot()` specificerer vi globale ting som gælder for hele plot, og i funktionen `geom_point()` ting som gælder kun for `geom_point()`.

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



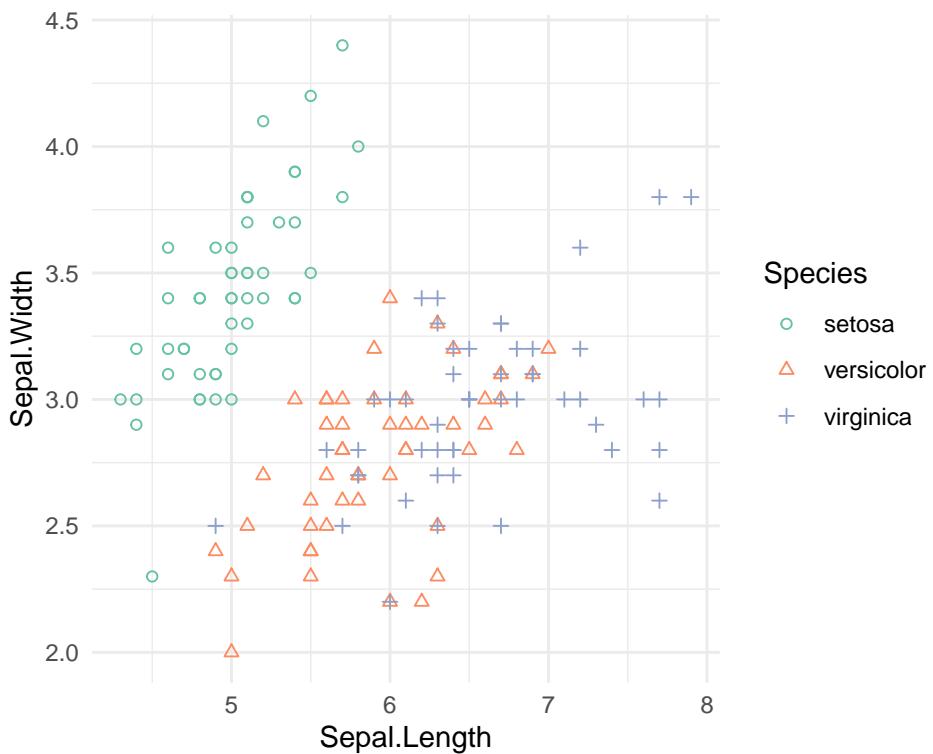
Så har vi fået en kombination af både forskellige farver og punkter til hver `Species`.

Sætte punkte former manuelt

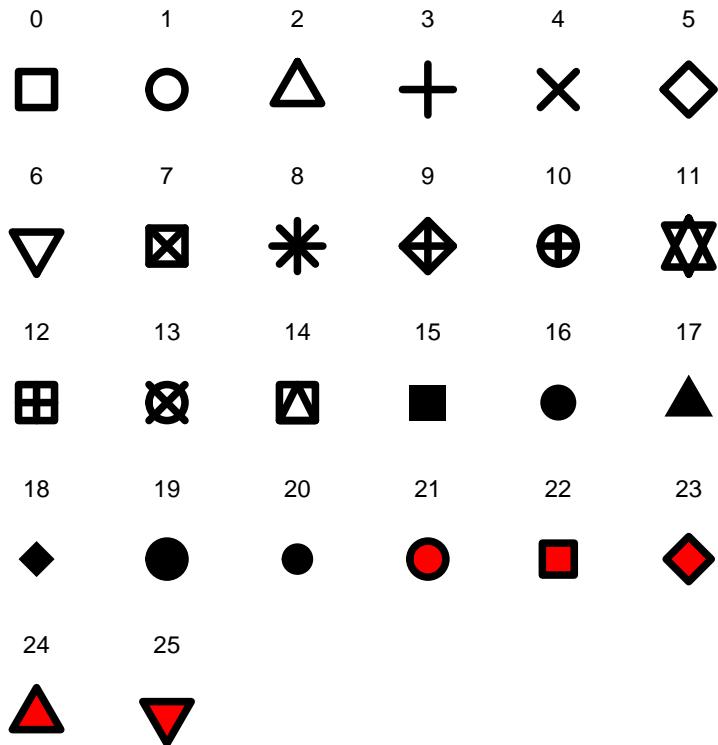
Hvis vi ikke kan lide de tre punkt former vi få automatisk, kan vi ændre dem ved at bruge `scale_shape_manual` - her vælger jeg `values=c(1,2,3)`, men der er en reference nedenunder, hvor I kan se, de mappings mellem de numeriske

tal og de punkt former, så at I kan vælger 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



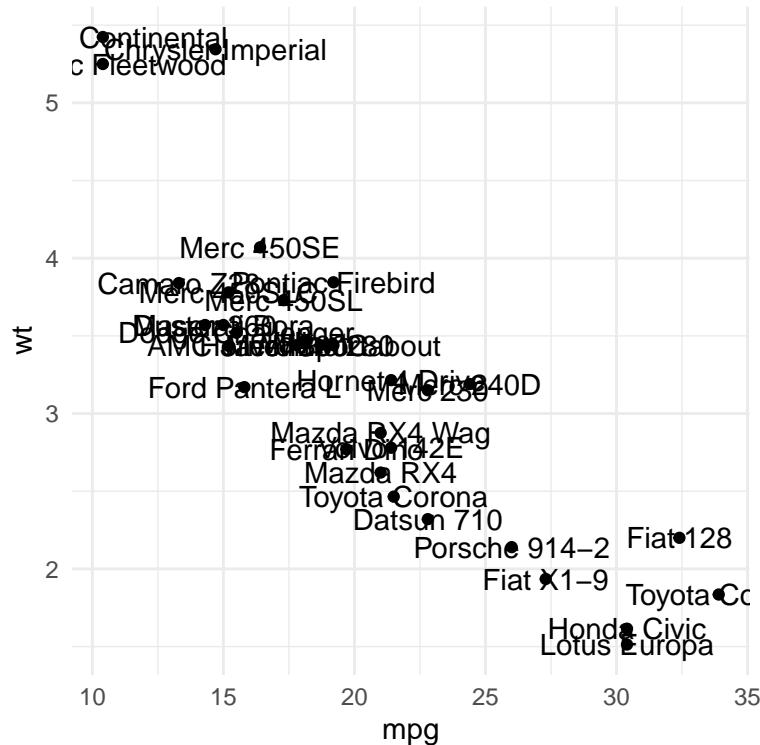
4.4 Annotations

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`.

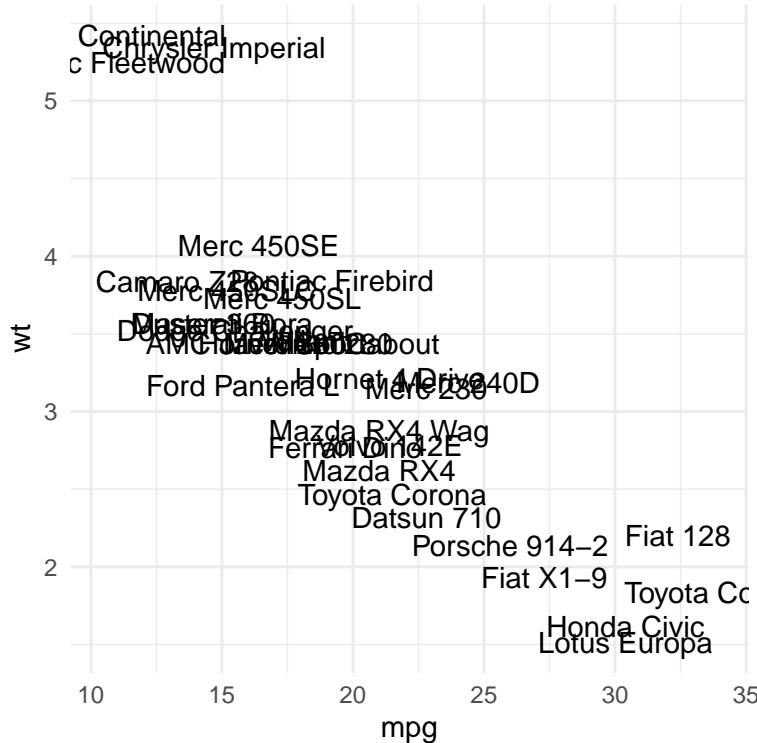
```
data(mtcars)

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



For at gøre det nemmere at læse kan man også fjerne de punkter:

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



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

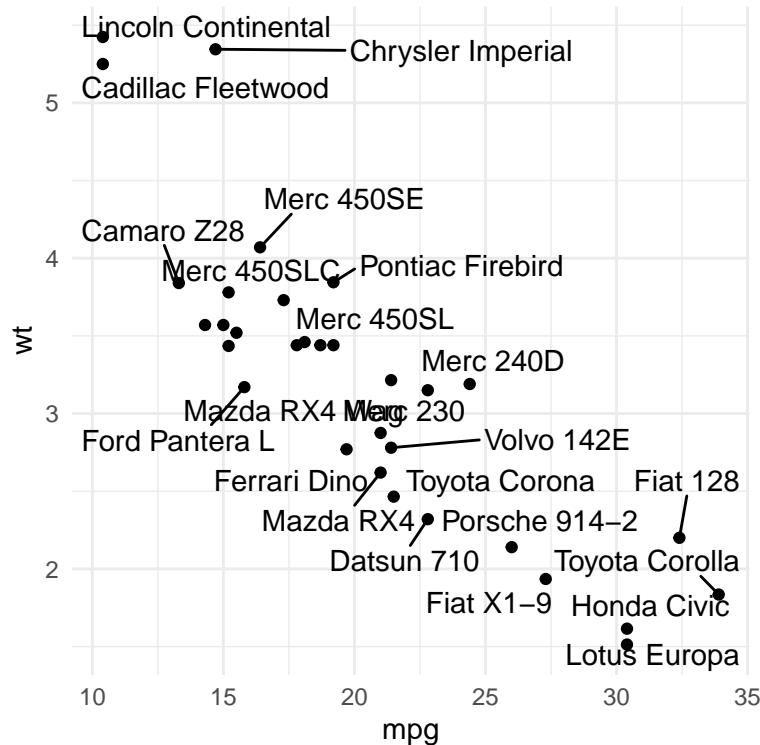
4.4.2 Pakken `ggrepel` for at tilføje tekst labeller

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

For at anvende pakken `ggrepel` for det `mtcars` datasæt. Man erstatter bare `geom_text()` med `geom_text_repel()`.

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

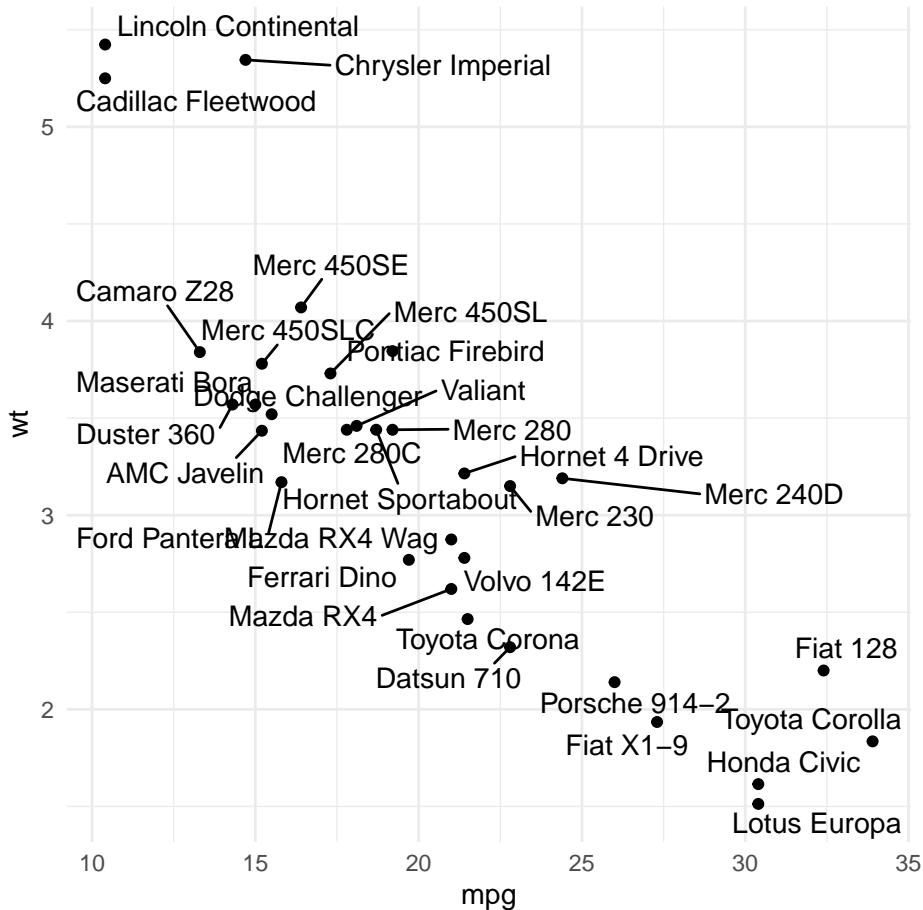
```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



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

Man kan se, at vi fik en advarsel her. Så vi kan prøve hvad vi er blevet bedt om her, og fortæl, at vi vil have max.overlaps måske 20.

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



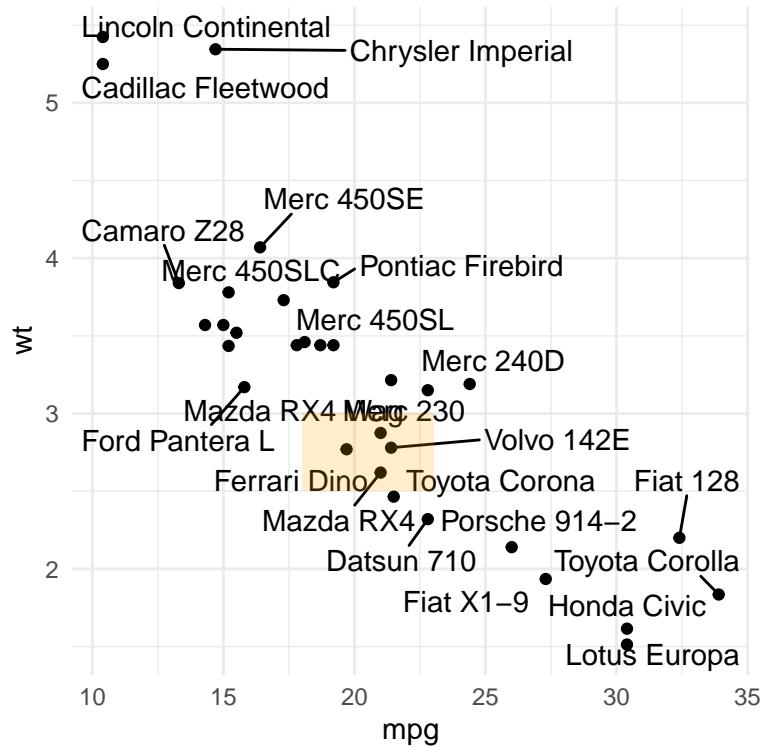
Så vi kan se, at vi ikke længere få en advarsel, og vi har tekst for alle vores punkter her.

4.4.3 Tilføje rektangler i regioner af interesse (annotate)

Ekstra: hvis man gerne vil fremhæv en bestemt region i plottet, kan man prøve `annotate`. Prøve at selv regne ud, hvad de muligheden indenfor `annotate` gå ud på.

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

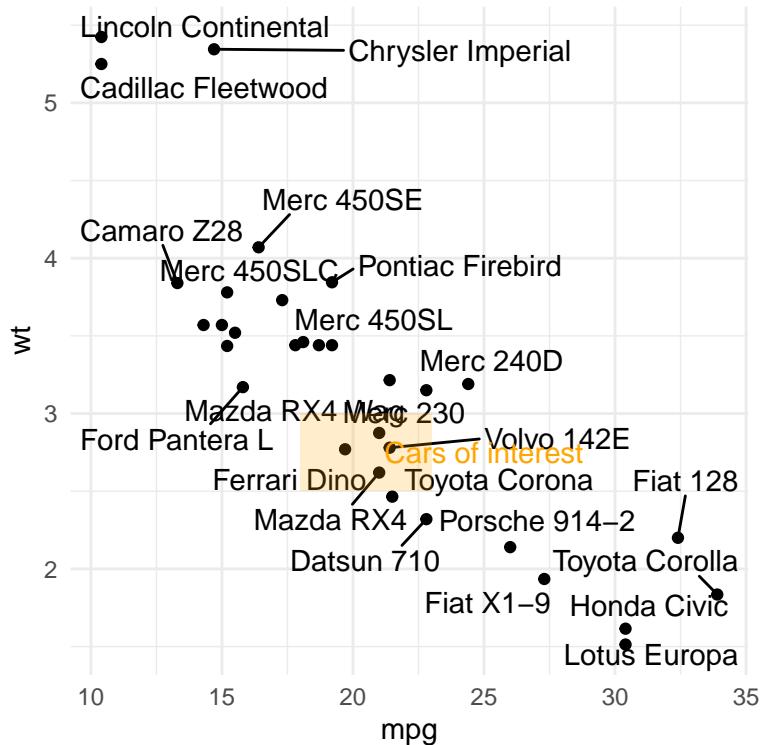
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Man kan også tilføje nogle tekster på en bestemt lokation med `annotate`:

```
ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text_repel(label=row.names(mtcars)) +
  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()
```

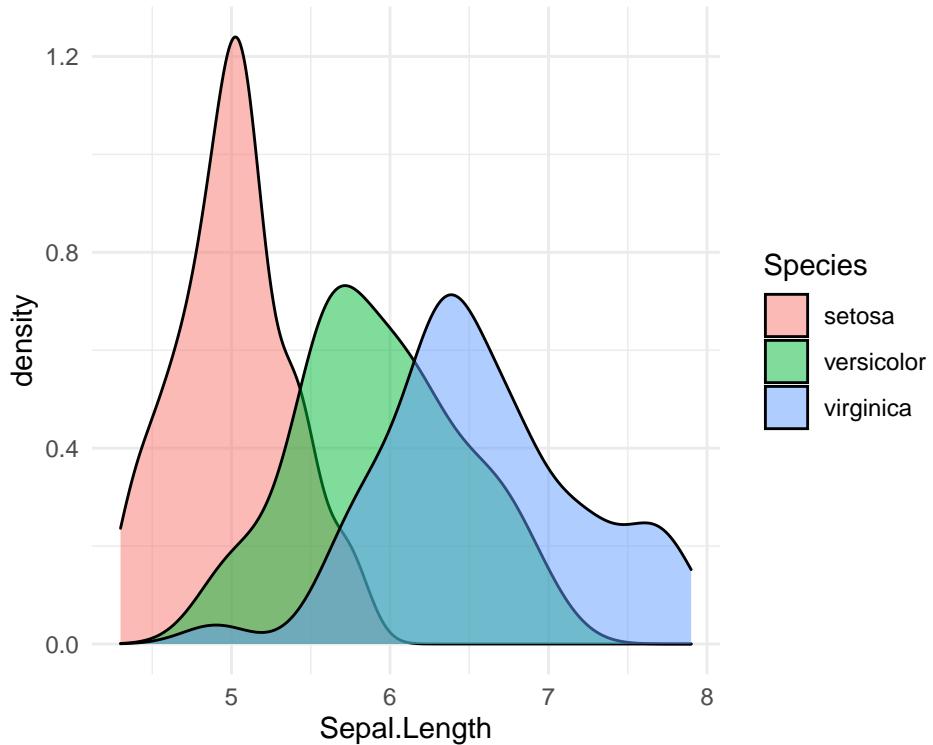
```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



4.5 Opdele plots (facet_grid/facet_wrap)

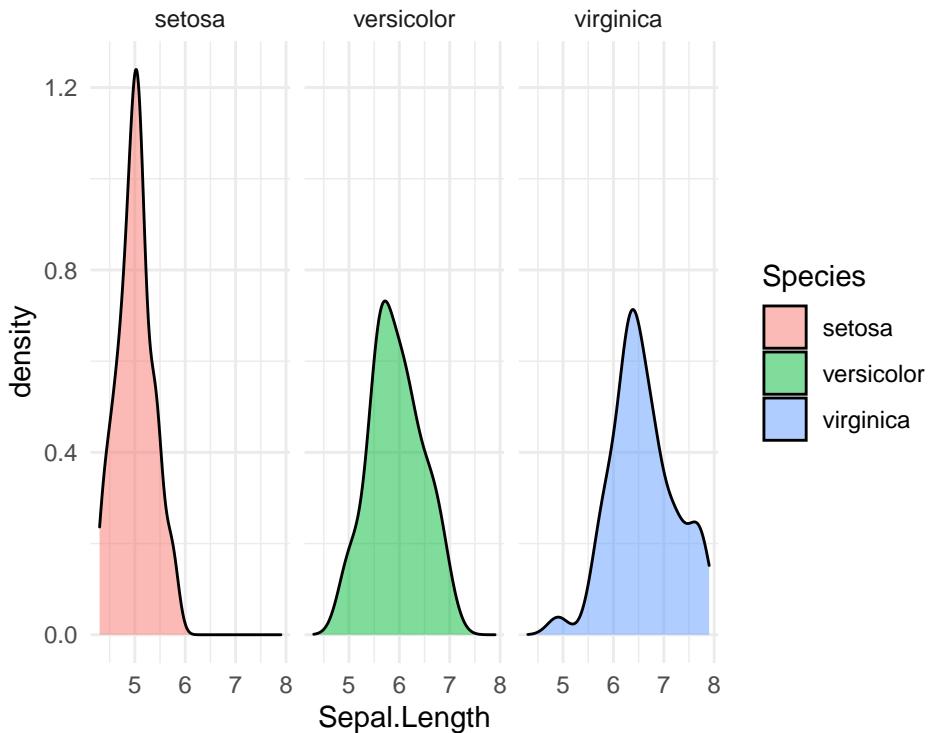
En stor fordele af at bruge `ggplot` er evne til at lave en `facet_grid` til at adskille grupperne i en variable over separate plotter. For eksempel:

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



bliver (ligger mærke til ~ her, som betyder at vi gerne vil opdele efter Species):

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



Vi leger lidt videre med `facet_grid()` i problemstillinger nedenfor.

4.6 Problemstillinger

- 1) Lave quiz - “Quiz - ggplot2 part 2”

Vi arbejder med **Palmer Penguins**.

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   Torge~         39.1         18.7       181      3750 male
FALSE  2 Adelie   Torge~         39.5         17.4       186      3800 fema~
FALSE  3 Adelie   Torge~         40.3          18        195      3250 fema~
FALSE  4 Adelie   Torge~          NA           NA        NA       NA <NA>
FALSE  5 Adelie   Torge~         36.7         19.3       193      3450 fema~
FALSE  6 Adelie   Torge~         39.3         20.6       190      3650 male
FALSE # ... with 1 more variable: year <int>
```

Man kan bruge `?penguins` for at se flere detaljer om variable navner.

I skal starte med at rydde op lidt med datasættet. Køre følgende for at fjerne række som har NA værdier:

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

2) Histogram

Lave en histogram:

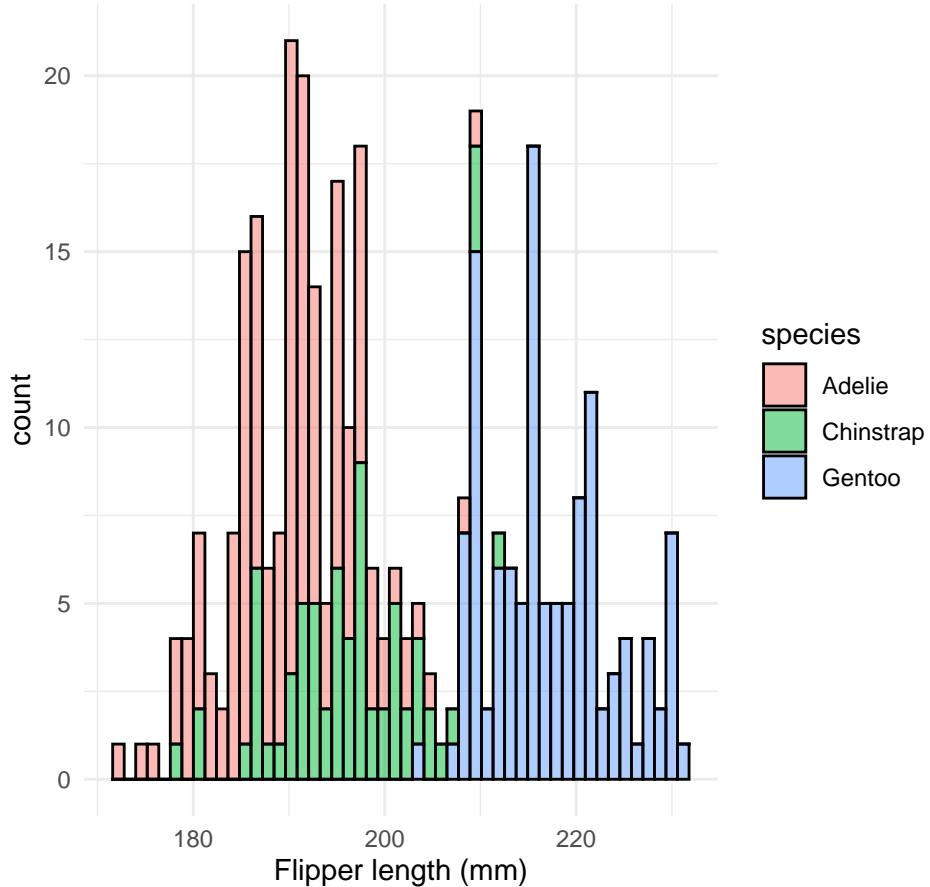
- `flipper_length_mm` på x-aksen
- brug `fill` til at opdele efter `species`
- specifiser sort linjer omkring de bars, så man mere tydeligt kan se dem
- specifiser hensigtsmæssige tekst (akserne, titel) og et tema

I få en advarsel: `stat_bin() using bins = 30. Pick better value with binwidth.`

- Prøve at ændre indstillingen `bins` til noget andet indenfor `geom_histogram()`.

Det skal ser sådan ud:

Histogram of flipper length according to species

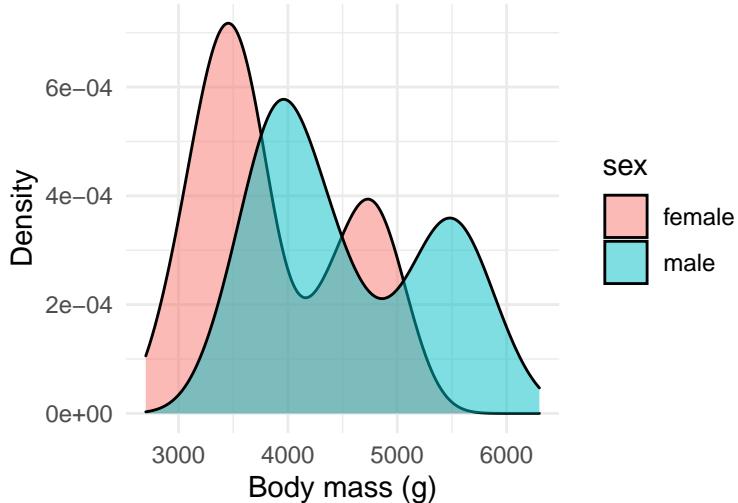


3) Density plots og introduktion til facet_grid()

a) Lave et density plot af body_mass_g.

- Bruge `fill` til at opdele efter `sex`
- Gøre dine density plots gennemsigtige
- Skrive en sætning om forskellen i `body_mass_g` mellem females og males.
- Hvorfor tror du, at de densities har flere toppe?

Density plots according to sex



b) Vi vil gerne adskille vores densities yderligere, efter `species`.

- Nu tilføj linjen `facet_grid(~species)` til dit plot og opdag, hvad der sker.
- Skriv endnu en sætning, som beskriver forskellen i `body_mass_g` mellem de to køn over de tre `species`.

4) Manuelt farver og punkter a) Lave en scatter plot med `ggplot`:

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

b) Lav følgende ændringer til det plot:

- Ændre farver manuelt - prøve 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 `species`.
- Prøv at vælge nogle punkt former fra listen og specificer dem manuelt.

5) Coordinate systemer

Tag overstående scatter plot fra 4) og

- bruge `coord_cartesian()` så at man fanger kun en bill længde mellem 40 og 50, og en bill depth mellem 16 og 19.
- som ekstra afprøve pakken `ggrepel` (husk at installere/indlæse) ved at tilføje de navne af de forskellige øer som labels direkte på plottet (valfrig opfordring: man kan lave en subset af de data og specificere det indenfor

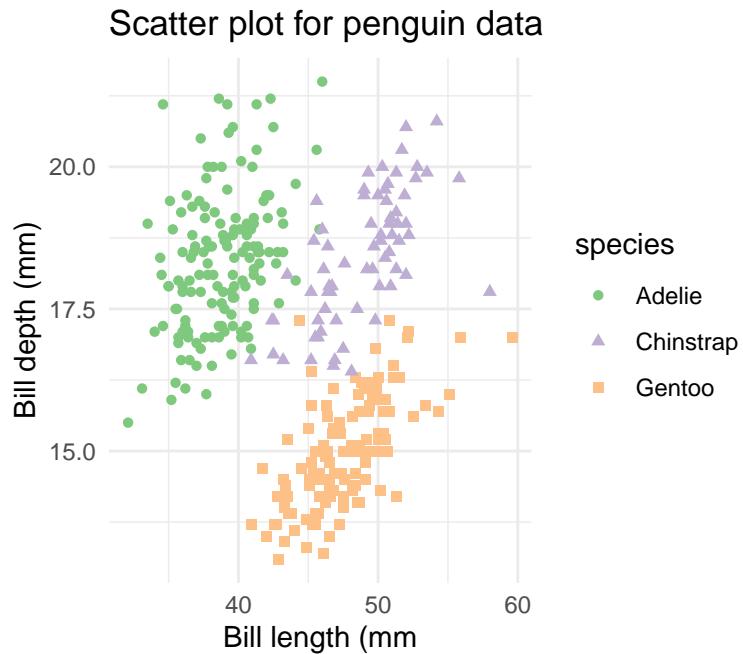


Figure 4.2: Min løsning

den relevant `ggrepel` funktion, for at undgå, at de labels bliver plottet for punkterne udenfor området angivet med `coord_cartesian()`).

6) Coordinate systemer

Lave en bar plot af counts for `species` opdelte efter `sex`.

- Anvende en ‘coordinate flip’ for at få den til at være horizontal.
- Vælge nogle manuelt farver.
- Vælge et tema som I godt kan lide.
- I vil have at de tre arter få rækkefølgen, således at den `species` med de meste observationer er på toppen og den `species` med den færrest er på bunden. Ændre rækkefølgen af de tre `species`.
- Prøve også `scale_y_reverse()` og kig på resultatet.

7) Lave boxplots af `body_mass_g` opdelte efter `species`.

- Tilføj punkter ovenpå.
- Specificere nogle farver manuelt for de boxes.
- Giv det en hensigtsmæssig titel og nogle akser-labels
- Adskille plotterne ved at opdele efter de forskellige `islands`.

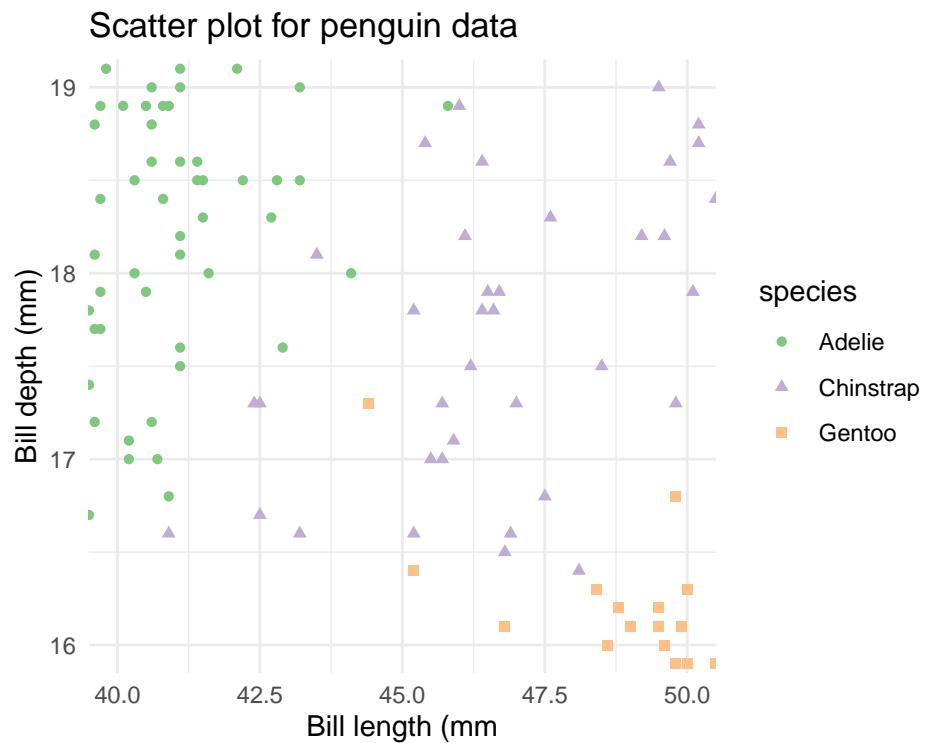


Figure 4.3: Min løsning

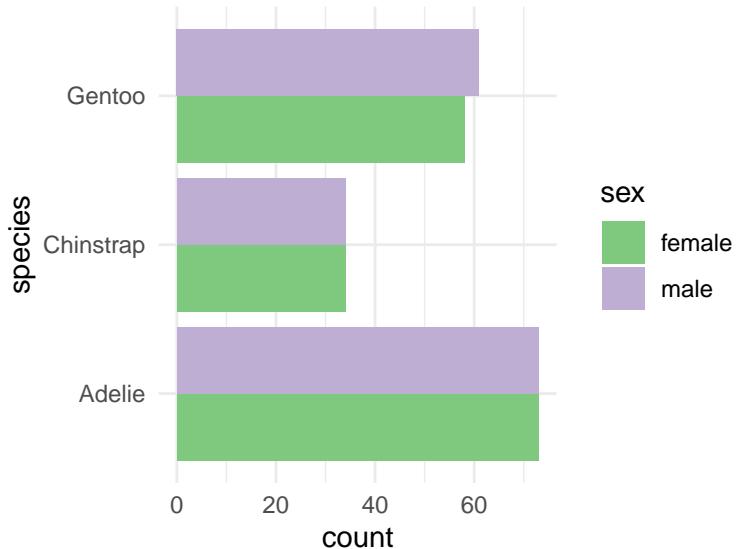
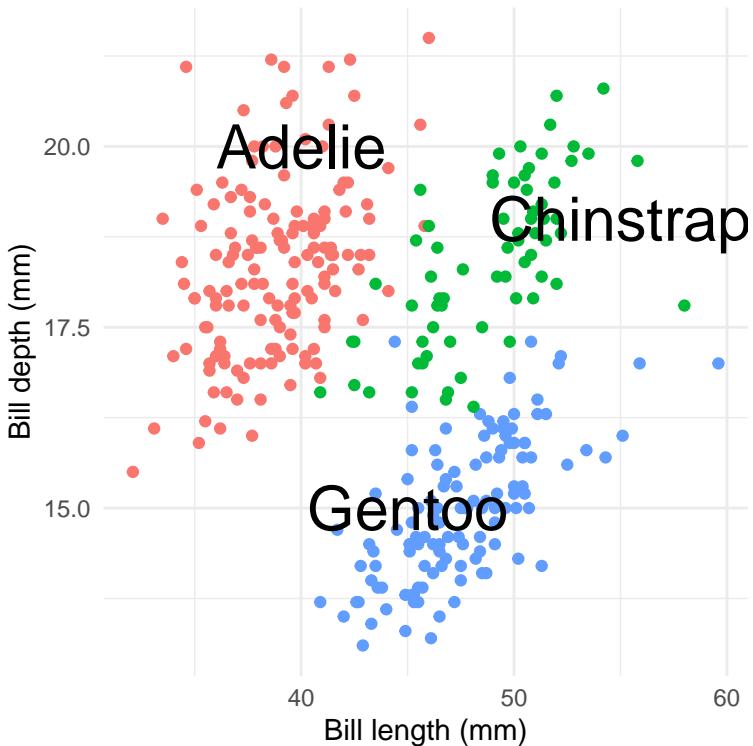


Figure 4.4: Min løsning

- Ekstra: skrive en sætning om `body_mass_g` ser forskellige ud for Adelie over de tre `islands`.
- Ekstra: udforsk `?geom_violin` som erstatning for `geom_boxplot`.

8) *Annotations.* 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`.
- Tjekke 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).
- Udforske hvordan man gøre teksten større, som jeg har gjort i min løsning.
- Fjerne den legend med `show.legend = FALSE` indenfor `geom_point()`
- Ekstra: bruge `annotate` igen til at lave et orange rektangel omkring alle de blå punkter (se kursusnotaterne).



b) Vi vil gerne tilføje noget lodrette og vandrette linjer til plottet som specifiser de middelværdier af de tre fordelinger.

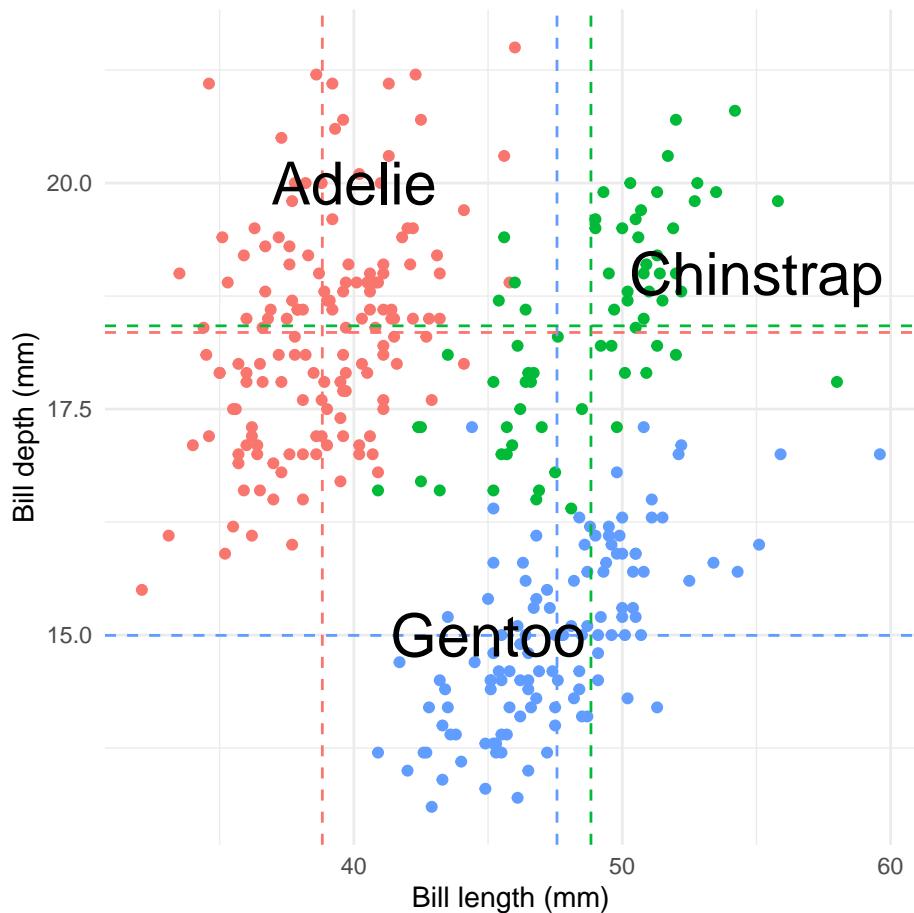
- Bruge `tapply` til at beregne de gennemsnitlige værdier for henholdsvis `bill_length_mm` og `bill_depth_mm` opdelte efter `species`.
- Lave en dataramme, med kolonner `species`, `mlength` og `mdepth`,

hvor `mlength` og `mdepth` er dine middelværdier for henholdsvis `bill_length_mm` og `bill_depth_mm`. Kalde det for `mydf`. Fk.

```
##           species mlength  mdepth
## Adelie     Adelie 38.82397 18.34726
```

- For de lodrette linjer anvende `geom_vline()` og specificere din ny dataramme indenfor med `data=mydf`. Tilføj også `xintercept=mlength` og `colour=species` (husk at da `mlength` og `species` er variabler fra `mydf` skal man bruge `aes()`)
- Tilføj de relevante vandrette linjer.
- Specifier “dashed” linjer

Plotten skal se sådan ud



9) Valgfri: Lav et scatter plot og inddrag `?scale_colour_gradient2` (se fk. <https://michaeltoth.me/a-detailed-guide-to-ggplot-colors.html>)

4.7 Workshop opgave (OBS Fredag)

- 1) I uge arbejder vi direkte på en skabalon som jeg har lavet i rmarkdown. Filen hedder `markdown_visualisering.Rmd` og er tilgængelige på Absalon.
- 2) Jeg kommer rundt i breakout rooms og tilbudsde min støtte.
- 3) Derefter poster jeg mine løsninger på problemer, som du måske kan bruge til at tage noget ekstra videre til næste gang.
- 4) På mandag går vi igennem nogle af de punkter som I havde mest svært ved.

4.8 Ekstra notat: gemme dit plot

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

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

Figuren vil blive gemt i den *working directory*. Filtypen `.pdf` kan erstattes med andre formater, for eksempel `.png` eller `.jpeg` osv. Hvis man gerne vil tage sit plot og redigerer det videre (fk. Adobe Illustrator eller Inkscape), vil jeg anbefalder at I bruge `.pdf`.

Det kan være når man har gemt sit plot, vil man gerne ændre højden og bredden på plottet. Det kan man ændre med `width` og `height`:

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

4.9 Ekstra links

ggthemes pakke:

<https://yutannihilation.github.io/allYourFigureAreBelongToUs/ggthemes/>

<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

R Graphics cookbook

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

Bar charts

<https://www.r-graph-gallery.com/218-basic-barplots-with-ggplot2.html>

4.10 Slut for ugen

Nærværende kapitel har givet en introduktion til nogle af de grundlæggende elementer i, hvordan man visualiserer datarammer med `ggplot2`-pakken i R.

Næste uge begynder vi med R-pakken tidyverse.

Chapter 5

Bearbejdning dag 1



5.1 Inledning og læringsmålene

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.

5.2 Video ressourcer

- Video 1 - introduktion til uge, hvad er den **tidyverse**-pakke? Hvad er en tibble?

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/547107062>

- Video 2 - rydder op den titanic datasæt med **select()** og **drop_na()**

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/546910795>

- Video 3 - introduktion til %>%

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/546910781>

- Video 4 - **tidyverse** verber: `select` og `filter`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/546910758>

- Video 5 - **tidyverse** verber: `mutate` og `arrange`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/546910712>

5.3 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.



Figure 5.1: Most common tidyverse packages

Lad os starte med at indlæse pakken **tidyverse**.

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

Du kan se, at det faktisk er ikke kun én, men otte pakker som blev indlæst. Her er nogle beskrivelser af pakkerne:

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>tidyr</code>	skifte imellem data forms (fk. ‘long’ > ‘wide’ format, eller omvendt)
<code>purrr</code>	for functional programming.
<code>dplyr</code>	manipulere tibbles (eller datarammer) - skabe nye variable, beregne oversigtsstatistikker osv.
<code>stringr</code>	manipulere strings
<code>forcats</code>	FOR CATegorical data (factors); this makes it easier to reorder and rename the levels in factor variables.

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)`.

5.3.1 Princippen med ‘tidy data’

Idéen bag **tidyverse** er, at hvis alle datasæt følger præcis den samme struktur, så er det ligefrem at bearbejde med vores data, 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.

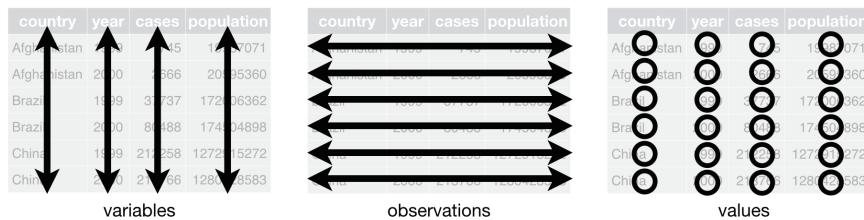


Figure 5.2: Principper af tidy data

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 **tidyr** 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 dog, at bar 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 filtrere eller rydde op i det. Pakken **dplyr** indeholder mange funktioner til at håndtere

5.4 Lidt om tibbles

En **tibble** er det **tidyverse** svar på en **data.frame** fra base-R. De ligner hinanden meget og derfor skal 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.

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
```

Den fleste **tidyverse** kode fungerer lige så godt uanset om man har en **tibble** eller en **data.frame**.

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
```

5.5 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 interessent, og sjovt at manipulere.

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

5.5.1 Om Titanic datasæt

Man kan godt downloade datasættet, der hedder `titanic_train`, direkte fra Kaggle, men der faktisk er en R-pakke, som gøre det mere bekvemt:

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

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 datasættet der hedder `titanic_train` - det hedder det fordi det bliver brugt i Kaggle til at train maskinelærings modeller (som bliver testet på `titanic_test` for at evaluere, hvor god modellen er).

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

```
## # A tibble: 6 x 12
##   PassengerId Survived Pclass Name      Sex     Age SibSp Parch Ticket  Fare Cabin
##       <int>     <int> <int> <chr>    <chr> <dbl> <int> <int> <chr> <dbl> <chr>
## 1         1       0     3 Braund~ male     22     1     0 A/5 2~  7.25  ""
## 2         2       1     1 Cuming~ fema~    38     1     0 PC 17~  71.3 "C85"
## 3         3       1     3 Heikki~ fema~    26     0     0 STON/~  7.92  ""
## 4         4       1     1 Futrel~ fema~    35     1     0 113803 53.1 "C12-
## 5         5       0     3 Allen,~ male    35     0     0 373450  8.05  ""
## 6         6       0     3 Moran,~ male    NA     0     0 330877  8.46  ""
## # ... with 1 more variable: Embarked <chr>
```

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).

5.5.2 Titanic: lidt oprydning

Der er faktisk nogle rengøring i datasættet vi skal tage os af, før vi gøre noget videre. Lad os lige tjekke hvor mange observationer vi har i datasættet:

```
nrow(titanic)
```

```
## [1] 891
```

For det første, har det fleste (687) passagerer ingenting for variabel `cabin`, og andre har mere end en cabin allokeret:

```
table(titanic$cabin)[1:40]
```

		A10	A14	A16	A19
##	687	1	1	1	1
##	A20	A23	A24	A26	A31
##	1	1	1	1	1
##	A32	A34	A36	A5	A6
##	1	1	1	1	1
##	A7	B101	B102	B18	B19
##	1	1	1	2	1
##	B20	B22	B28	B3	B30
##	2	2	2	1	1
##	B35	B37	B38	B39	B4
##	2	1	1	1	1
##	B41	B42	B49	B5	B50
##	1	1	2	2	1
##	B51 B53 B55 B57 B59 B63 B66		B58 B60	B69	B71
##	2	2	2	1	1

Det ser ikke særlig **tidy** ud, og man kan hellere ikke forestille sig at lave en fornuftig analyse fra den. Vi vælger derfor bare at fjerne hele kolon med funktionen `select()`:

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

`select()` er en af de core funktioner man bruger 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 kolonen der hedder `Cabin` med, men gerne vil beholde resten af kolonnerne. Prøv at køre `select(titanic, Cabin)` i stedet for - så får vi kun `Cabin` og fjerner resten af vores variable.

Næste kan vi tjekke for `NA` i datasættet. Man kan se, at `Age` faktisk har 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
```

Vi kan vælge at fjerne alle passagerer som har NA i stedet for deres alder. Her kan vi bare bruge `drop_na` som fjerner alle observationer, med 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.

```
nrow(titanic_clean)
```

```
## [1] 714
ncol(titanic_clean)
```

```
## [1] 11
```

Vi har stadig 714 observationer og 11 kolonner, og vores datasæt opfylder kriterien for at være **tidy**.

5.5.3 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 funktioner i samme kommando - linjen slutter med `%>%`, der fortæller, at vi skal bruge resultatet fra den linje som input i den næste linje. Logikken er således, at vi starter med en dataramme, og så dernæst gør én ting ad gangen.

Bemærk, at processen ligner den vi bruger i **ggplot2**, men forskellen er at man bruger `%>%` i stedet for `+` i denne ramme. Bemærk også her, at ligesom i **ggplot2**, skriver vi 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:

```
f(x)
x %>% f
```

I tilfældet af `x %>% f` starter vi med `x`, og så anvender vi funktionen `f` med `x` som argument - det er den **tidyverse** løsning.

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 (manglende værdi).

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

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

Vi kommer til at bruge den **tidyverse** løsning meget fremadrettet.

5.6 Bearbejdning af data: dplyr

Den meste nyttige pakke, der kan bruges til at bearbejde datarammer, er `dplyr`, som vi vil fokusere på i følgende sektioner.

<https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf>

```
library(dplyr) #behøves ikke hvis pakken tidyverse allerede er indlæst
```

Pakken giver nogle basale funktioner, der gør det nemt at bearbejde datarammer (`data.frame` eller `tibble`).

dplyr verbs	Description
<code>select()</code>	udvælge kolonner (<i>variable</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 navne på kolonner
<code>group_by()</code>	lave ‘split’ operationer over en gruppe variabel, som forudsætning til <code>summarise()</code>
<code>summarise()</code>	aggregere rækker

Med alle disse funktioner, at de tager man udgangspunkt i en dataramme, få man altid en ny dataramme som output. Ved at kunne bruge disse funktioner

og kombinere dem (ved hjælp af `%>%`), har man godt styr på bearbejdningen af datarammer.

5.6.1 dplyr verbs: `select()`

Med `select()` vælger man bestemte **variabler**. Vi kan vælge at beholde, fjerne eller andre rækkefølgen af variablerne i datarammen.

Som eksempel, hvis vi kun skal bruge bestemte variabler i vores dataramme, eksempelvis `Name` og `Age`:

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

```
## # A tibble: 6 x 2
##   Name                               Age
##   <chr>                             <dbl>
## 1 Braund, Mr. Owen Harris            22
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer)    38
## 3 Heikkinen, Miss. Laina             26
## 4 Futrelle, Mrs. Jacques Heath (Lily May Peel)           35
## 5 Allen, Mr. William Henry          35
## 6 McCarthy, Mr. Timothy J            54
```

Hvis vi gerne vil fjerne en variabel fra en dataramme, kan vi bruge en minus tegnet. I nedenstående fjerne vi `Name` og `Age` fra datarammen.

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

## # A tibble: 6 x 9
##   PassengerId Survived Pclass Sex     SibSp Parch Ticket      Fare Embarked
##   <int>      <int> <int> <chr>  <int> <int> <chr>      <dbl> <chr>
## 1         1         0     3 male    1      0 A/5 21171    7.25 S
## 2         2         1     1 female   1      0 PC 17599   71.3  C
## 3         3         1     3 female   0      0 STON/O2. 3101282  7.92 S
## 4         4         1     1 female   1      0 113803       53.1 S
## 5         5         0     3 male    0      0 373450        8.05 S
## 6         7         0     1 male    0      0 17463       51.9  S
```

5.6.2 dplyr verbs: `filter()`

Husk, at med `select()` vælger man bestemte **variabler**. Man anvender `filter()` til at vælge bestemte observationer (rækker) fra datarammen. I nedenstående tager vi kun rækkerne fra datarammen `titanic_clean`, hvor kolonnen `Age` er lig med 50. Bemærk, at vi bevarer alle kolonner i datarammen, og at det kun er rækkerne der bliver udvalgt.

```
titanic_clean %>%
  filter(Age == 50) %>%
  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        178      0     1 Isham, Miss~ fema~    50     0     0 PC 17~  28.7
## 2        260      1     2 Parrish, Mr~ fema~    50     0     1 230433  26
## 3        300      1     1 Baxter, Mrs~ fema~    50     0     1 PC 17~ 248.
## 4        435      0     1 Silvey, Mr.~ male    50     1     0 13507   55.9
## 5        459      1     2 Toomey, Mis~ fema~    50     0     0 F.C.C~ 10.5
## 6        483      0     3 Rouse, Mr. ~ male    50     0     0 A/5 3~  8.05
## # ... with 1 more variable: Embarked <chr>
```

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 kriterier fra forskellige kolonner, for eksempel i nedenstående vælger vi alle personer som er kvinder **og** som rejste ved første klasse **og** som overlevede.

```
titanic_clean %>%
  filter(Sex == 'female' & Pclass == 1 & Survived == 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 comparitiv symboler. For eksempel i nedenståenden vælger vi personer som er kvinder **og** som rejste i **enten** første eller anden klasse **og** som overlevede. Huske at tilføje round brackets omkring de to Pclass.

```
titanic_clean %>%
  filter(Sex == 'female' & (Pclass == 1 | Pclass == 2) & Survived == 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, Mr~ fema~ 38      1      0 PC 17~ 71.3
## 2          4      1      1 "Futrelle, M~ fema~ 35      1      0 113803 53.1
## 3         10      1      2 "Nasser, Mrs~ fema~ 14      1      0 237736 30.1
## 4         12      1      1 "Bonnell, Mi~ fema~ 58      0      0 113783 26.6
## 5         16      1      2 "Hewlett, Mr~ fema~ 55      0      0 248706 16
## 6         44      1      2 "Laroche, Mi~ fema~ 35      1      2 SC/Pa~ 41.6
## # ... with 1 more variable: Embarked <chr>
```

5.6.3 Comparitiver reference

Her er en tabel af comparitiver.

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

5.6.4 Kombinere `filter()` og `select()`

Man kan også kombinere både `filter()` og `select()` i samme kommando. For eksempel i nedenstående får vi kun kolonnerne `Name` og `Fare`, for alle passagerer som som er kvinder **og** som rejste ved første klasse **og** som overlevede.

```
titanic_clean %>%
  filter(Sex == "female" & Pclass == 1 & Survived == 1) %>%
```

```

  select(Name, Fare)  %>%
  head()

## # A tibble: 6 x 2
##   Name                               Fare
##   <chr>                             <dbl>
## 1 Cumings, Mrs. John Bradley (Florence Briggs Thayer)    71.3
## 2 Futrelle, Mrs. Jacques Heath (Lily May Peel)           53.1
## 3 Bonnell, Miss. Elizabeth                   26.6
## 4 Harper, Mrs. Henry Sleeper (Myra Haxtun)            76.7
## 5 Icard, Miss. Amelie                      80
## 6 Fortune, Miss. Mabel Helen                263

```

Bemærk at man bør passe på den rækkefølge, som man anvender de forskellige funktioner. For eksempel hvis man skifter `filter()` og `select()` i ovenstående, få man en advarsel. Det er fordi, hvis man første vælger at kun beholde variabler `Name` og `Age`, så findes de andre kolonner ikke mere i de resulterende dataramme. Man kan ikke derfor bruge den `filter()` funktion på variabler `Pclass`, `Sex` og `Survived`.

```

##virker ikke!!!!!!#####
titanic_clean %>%
  select(Name, Fare)  %>%
  filter(Pclass == 1 & Sex == "female" & Survived == 1) %>%
  head()

```

5.6.5 dplyr verbs: `mutate()`

Man kan avende funktionen `mutate()` til at tilføje en ny variable til en dataramme.

I nedenstående tilføjer vi en nye variabel ved navn `Adult`, der angiver om personen kan betragtes som en voksen (hvis de er mindst 18 år gammel).

Bemærke her, at vi gemme resultatet som en ny dataramme der hedder `titanic_with_Adult`, og derefter bruger vi `head()` for at se, hvordan vores nye kolon ser ud. I forudgående eksempler har vi ikke gemt resultatet (bare brugt `head()` for at se resultatet på skærmen).

```

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

head(titanic_with_Adult$Adult)

## [1] TRUE TRUE TRUE TRUE TRUE TRUE
summary(titanic_with_Adult$Adult)

##      Mode   FALSE     TRUE

```

```
## logical      113      601
```

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

Man kan lave samme kolon mere informativ end bare TRUE eller FALSE med en `ifelse` statement. Vi tilføjer også en `select()` funktion, bare så vi kan se variablen i nedenstående table i dette dokument.

```
titanic_clean %>%
  mutate(Adult = ifelse(Age>=18, "Adult", "Child")) %>%
  select(Name, Age, Sex, Pclass, Adult) %>%
  head()

## # A tibble: 6 x 5
##   Name                               Age Sex Pclass Adult
##   <chr>                             <dbl> <chr> <int> <chr>
## 1 Braund, Mr. Owen Harris           22   male     3   Adult
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) 38   female    1   Adult
## 3 Heikkinen, Miss. Laina          26   female    3   Adult
## 4 Futrelle, Mrs. Jacques Heath (Lily May Peel)        35   female    1   Adult
## 5 Allen, Mr. William Henry         35   male     3   Adult
## 6 McCarthy, Mr. Timothy J          54   male     1   Adult
```

Så er kolonnen lidt mere informativ end før.

5.6.6 `rename()`

Man kan bruge `rename()` til at ændre navnet på en eller flere kolonner i datasættet. Lad os måske bruge `rename()` til at give en kolon navnet ‘Years’ i stedet for ‘Age’.

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

## # A tibble: 6 x 11
##   PassengerId Survived Pclass Name                           Sex   Years SibSp Parch Ticket Fare
##   <int>       <int> <int> <chr>                         <chr> <dbl> <int> <int> <chr> <dbl>
## 1          1       0     3 Braund, Mr. ~ male             22     1     0 A/5 2~  7.25
## 2          2       1     1 Cumings, Mrs~ fema~            38     1     0 PC 17~  71.3
## 3          3       1     3 Heikkinen, M~ fema~            26     0     0 STON/~  7.92
## 4          4       1     1 Futrelle, Mr~ fema~            35     1     0 113803 53.1
## 5          5       0     3 Allen, Mr. W~ male             35     0     0 373450  8.05
## 6          7       0     1 McCarthy, Mr~ male            54     0     0 17463   51.9
## # ... with 1 more variable: Embarked <chr>
```

Så vi kan se at `Age` ikke findes, og vi har `Years` i stedet for.

Man kan også andre navne på flere kolonner på en gang. For eksempel, vi kan måske lave nogle oversættelse arbejde.

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

Så vi kan se vi har ændrede naverne. Lad også kalde det for `titanic_clean_dansk`, så vi gemme vores danske version.

Vi kan også gøre så at vi har kun små bogstaver i vores navne. Lad os bruge vores danske version, og vi anvende `rename_with()` og specificerer `tolower`.

```
titanic_clean_dansk %>%
  rename_with(tolower) %>%
  head()

## # A tibble: 6 x 11
##   passengerid overlevede klasse navn       sex     age sibsp parch ticket fare
##       <int>      <int> <int> <chr>      <chr> <dbl> <int> <int> <chr> <dbl>
## 1          1          0     3 Braund, Mr~ male    22     1     0 A/5 2~  7.25
## 2          2          1     1 Cumings, M~ fema~    38     1     0 PC 17~ 71.3
## 3          3          1     3 Heikkinen, ~ fema~    26     0     0 STON/~  7.92
## 4          4          1     1 Futrelle, ~ fema~    35     1     0 113803 53.1
## 5          5          0     3 Allen, Mr.~ male    35     0     0 373450  8.05
## 6          7          0     1 McCarthy, ~ male    54     0     0 17463  51.9
## # ... with 1 more variable: embarked <chr>
```

Prøve også at erstatte `tolower` med `toupper`.

5.6.7 dplyr verbs: `arrange()`

Man anvender `arrange()` for at vælge rækkefølgen på observationer. 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) %>%
  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          1          0     3 Leonard, Mr.~ male    36     0     0 LINE     0
## 2          2          0     1 Harrison, Mr~ male    40     0     0 112059   0
## 3          3          1     3 Tornquist, M~ male    25     0     0 LINE     0
## 4          4          0     3 Johnson, Mr.~ male    19     0     0 LINE     0
## 5          5          0     3 Johnson, Mr.~ male    49     0     0 LINE     0
```

```
## 6      807      0      1 Andrews, Mr.~ male    39      0      0 112050      0
## # ... with 1 more variable: Embarked <chr>
```

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)) %>%
  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       259      1     1 Ward, Miss. ~ fema~    35     0     0 PC 17~ 512.
## 2       680      1     1 Cardeza, Mr.~ male   36     0     1 PC 17~ 512.
## 3       738      1     1 Lesurer, Mr.~ male   35     0     0 PC 17~ 512.
## 4        28      0     1 Fortune, Mr.~ male   19     3     2 19950 263
## 5        89      1     1 Fortune, Mis~ fema~   23     3     2 19950 263
## 6       342      1     1 Fortune, Mis~ fema~   24     3     2 19950 263
## # ... with 1 more variable: Embarked <chr>
```

5.7 Ændre data med `recode()`

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

```
titanic_clean %>%
  mutate(Sex = recode(Sex, "male" = 1, "female" = 0)) %>%
  select(PassengerId, Name, Survived, Sex) %>% head()
```

			Survived	Sex
			<int>	<dbl>
## 1	1	Braund, Mr. Owen Harris	0	1
## 2	2	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	1	0
## 3	3	Heikkinen, Miss. Laina	1	0
## 4	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	0
## 5	5	Allen, Mr. William Henry	0	1
## 6	7	McCarthy, Mr. Timothy J	0	1

Bemærk at vi benytte `recode()` indenfor funktionen `mutate`: vi lave en ny variable af samme navn, men med ændret værdier indenfor variablen.

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

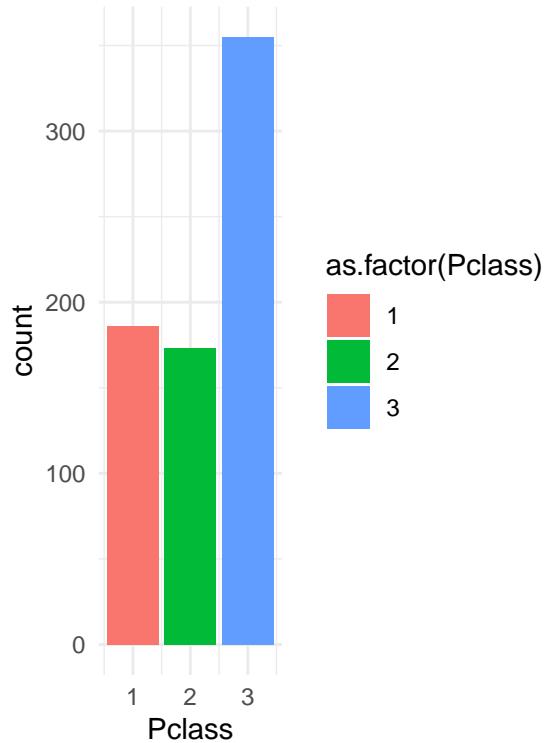
```
titanic_clean %>%
  mutate(Sex = recode(Sex, male = 1, female = 0)) %>%
  mutate(Sex = recode(Sex, `1` = "male", `0` = "female")) %>%
  select(PassengerId, Name, Survived, Sex) %>% head()
```

```
## # A tibble: 6 x 4
##   PassengerId Name                               Survived Sex
##       <int> <chr>                               <int> <chr>
## 1          1 Braund, Mr. Owen Harris             0 male
## 2          2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) 1 fema-
## 3          3 Heikkinen, Miss. Laina              1 fema-
## 4          4 Futrelle, Mrs. Jacques Heath (Lily May Peel)        1 fema-
## 5          5 Allen, Mr. William Henry            0 male
## 6          7 McCarthy, Mr. Timothy J             0 male
```

5.8 Visualisering: bruge som input i ggplot2

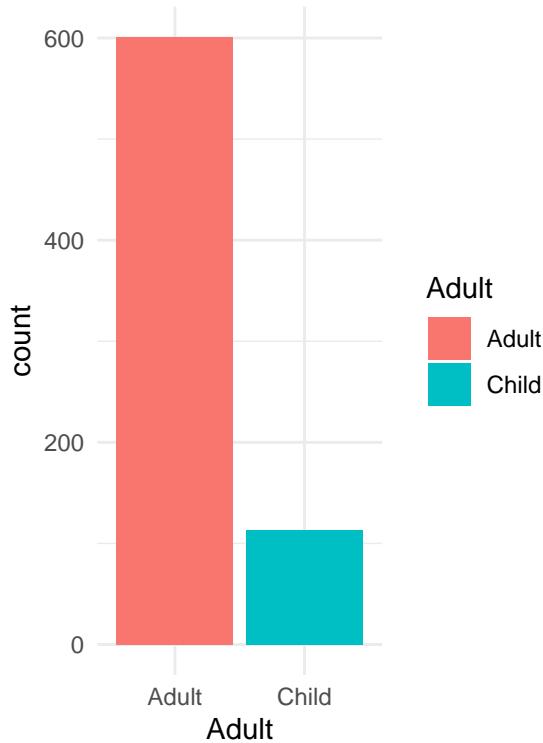
Man kan lave nogle bearbejdning med en `tidyverse` kommando, og så specificerer de resulterende dataramme som data i et `ggplot`. I `tidyverse` stil kan man bruge `%>%` til at forbinde den `tidyverse` kommando med den `ggplot2` kode. I dette tilfælde specificerer man ikke den data indenfor funktionen `ggplot`. I nedenstående eksempel tager vi `titanic_clean` og så lave et barplot af 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()
```



Lad os gøre det lidt mere kompliceret, ved at tage `titanic_clean`, lave en ny kolon der hedder `Adult`, og så bruge den resulterende dataramme i et `ggplot`, hvor vi lave et plot med `Adult` på x-aksen for at tælle op antallet af adults og children.

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



Vi kan se, at der var 600 Adults og lidt over 100 Children ombord skibet.

5.9 Problemstillinger

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

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()
```

2) `select()`. Fjerne variablen `Name` fra `titanic_clean`.

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

- Tilføj også `head()` for at få kun de første 6 rækker

- Prøv at erstatte `head()` med `glimpse()` - det er bare en anden måde at kigge på datarammen.

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

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

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

- Prøve også `contains("ar")`

5) filter(). Lave en ny dataramme fra `titanic_clean` med alle personer som er male og over 30 år gammel.

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

- Hvor mange observationer er der i den ny dataramme?
- Prøve at tilføje `%>% count()` til kommandoen - få man så samme tal?

7) filter() og select() : kombinering med %>%

Lave en ny dataramme fra `titanic_clean` med alle passagerer som er male og survived, og angiv kun kolonner `Name`, `Age` og `Fare`.

8) 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()` funktioner omvendt her? Hvorfor?

9) Mutate(). Lave en ny dataramme fra `titanic_clean` som hedder `FareRounded` som viser `Fare` rundet til det nærmest integar (hint: benytte funktionen `round()`).

10) Mutate(). 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 intuitive - "Family" og "Not family".

11) 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?

12) 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 (S for variablen Embarked), arrangere de resulterende observationer efter Fare (højeste på toppen) og angiv kun kolonnerne Name, Age og Fare.

13) *Rename.* Fra titanic_clean angiv kun variabler Survived,Ticket, og Name og ændre deres navne til Overlevede, Billet og Navn.

- Gøre variabler navne til store bogstaver ved at anvende rename_with().

14) *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. Bruge følgende struktur for koden:

```
titanic_clean %>%
  filter(...) %>% #rediger linjen
  ggplot(aes(...)) + .... #tilføj plot
```

15) *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.

5.10 Kommentarer

I morgen arbejder vi videre med tidyverse.

Chapter 6

Bearbejdning dag 2



6.1 Indledning og læringsmålene

I dag skal vi arbejde videre med `tidyverse`, især på pakken `dplyr` og `tidyr`, som kan bruges til at andre på strukturen af de data, således at det passer til den struktur, som kræves for at lave plots med `ggplot2`.

6.1.1 Læringsmålene

I skal være i stand til at

- Benytte kombinationen af `group_by()` og `summarise()`.
- Forbinde `tidyverse` kode og `ggplot2` kode sammen for at svare på spørgsmål om datasættet.
- Forstå forskellen mellem `wide` og `long` data og bruge `pivot_longer()` til at facilitere plotting

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/546910660>

- Video 3 - eksempel med titanic summary statistics og `facet_wrap()`

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/547096274>

6.2 dplyr: `group_by()` med `summarise()`

Man kan lave summary statistics med funktionen `summarise()`. Man plejer at kombinere `summarise()` med `group_by()`, som anvendes til at opdele datasættet efter en eller flere variabler. Her kan vi begynde at stille spørgsmål omkring vores data. For eksempel: 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: her opdeles vi kolonnen `Survived` efter kolonnen `Sex` og tager middelværdien, som resultater i proportionen der overlevede efter køn (da `Survived` er kodet sådan at 1 betyder at man overlevede og 0 betyder at man ikke overlevede).

```
#tapply løsning
tapply(titanic_clean$Survived,titanic_clean$Sex,mean)
```

```
##      female      male
## 0.7547893 0.2052980
```

Lad os skifte 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 specificeret. Vi er interesseret i proportionen, der overlevede, så vi behøver at tage 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
```

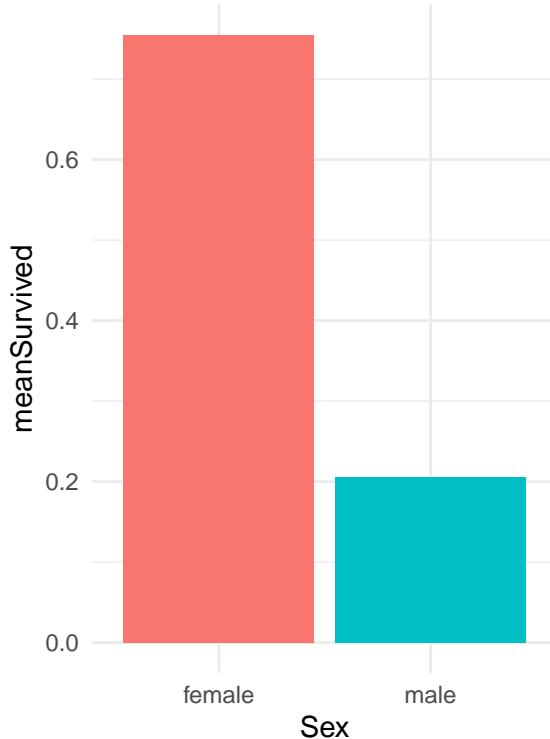
Få at svare på spørgsmålet er vi også nødt til at opdele efter kolonnen `Sex`. Vi kan bruge den kombinerende af `group_by()` og `summarise()` - vi opdele efter `Sex` ved at anvende funktionen `group_by()` og derefter bruger `summarise()` til at oprette en kolon der hedder `meanSurvived`, der viser proportionen 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 `summary()` funktioner

Nogle funktioner man ofte bruge med `summary()` (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`.

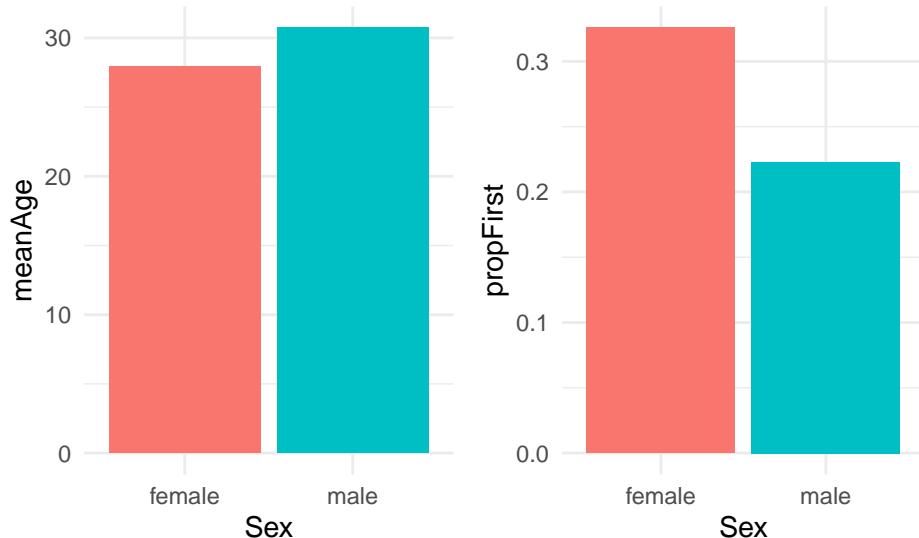
```

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`.

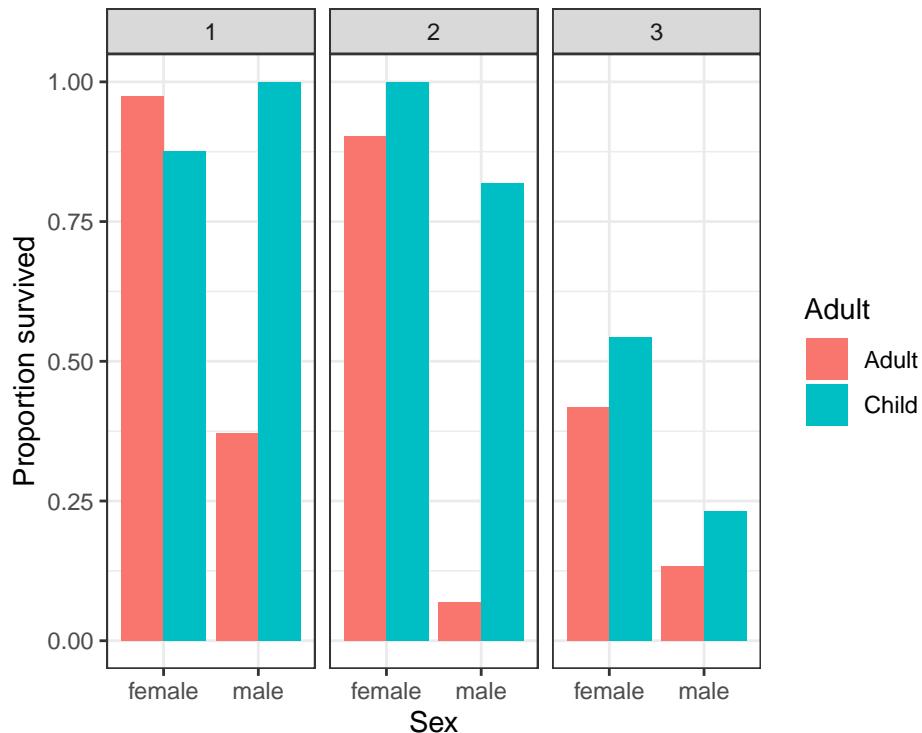
```
titanic_clean_summary_survived <- titanic_clean %>%
  mutate(Adult = ifelse(Age>=18,"Adult","Child")) %>%
  group_by(Adult,Sex,Pclass) %>%
  summarise(meanSurvived = mean(Survived))
```

```
## `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()
```



6.3 Tidyr pakke - Wide og Long data

Tidy data findes i to former: wide data og long data. Som vi vil se, kan det være nyttigt at transformere de data fra den ene form til den anden, for fx. at facilitere et plot med `ggplot2`. Indenfor R-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 variable 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.

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 transformerer data fra wide til long eller omvendt, kaldes det for **reshaping**.

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/>

6.3.1 Tidyr pakke - oversigt

Her er en oversigt over de fire vigtigste funktioner fra R-pakken `tidyr`.

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

Vi fokuserer på funktionen `pivot_longer()`, da den er mest brugbar, men nævne kort de andre funktioner.

6.3.2 Wide -> Long med `pivot_longer()`

Lad os arbejde med datasættet `Iris`. Man få Iris' i long form med følgende kommando:

```
iris %>% pivot_longer(cols = -Species)
```

WIDE					LONG									
> head(as_tibble(iris))					> iris %>% pivot_longer(-Species)									
# A tibble: 6 x 5					# A tibble: 600 x 3									
Sepal.Length Sepal.Width Petal.Length Petal.Width Species					Species name value									
<dbl> <dbl> <dbl> <dbl> <fct>					<fct> <chr> <dbl>									
1 5.1 3.5 1.4 0.2 setosa					1 setosa Sepal.Length 5.1									
2 4.9 3 1.4 0.2 setosa					2 setosa Sepal.Width 3.5									
3 4.7 3.2 1.3 0.2 setosa					3 setosa Petal.Length 1.4									
4 4.6 3.1 1.5 0.2 setosa					4 setosa Petal.Width 0.2									
5 5 3.6 1.4 0.2 setosa					5 setosa Sepal.Length 4.9									
6 5.4 3.9 1.7 0.4 setosa					6 setosa Sepal.Width 3									
Measurements over 4 columns →														
Measurements in 1 column														

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

default: for eksempel i nedenstående skal målingerne hedde `measurements` og nøglen hedde `trait`.

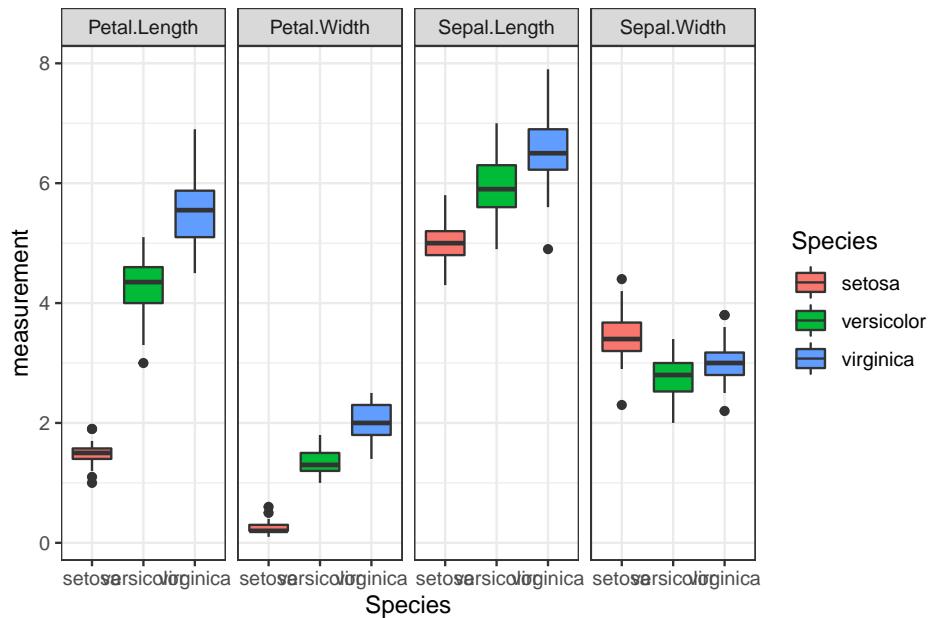
```
iris.long <- iris %>% pivot_longer(cols = -Species,
                                         names_to = "trait",
                                         values_to = "measurement")
```

WIDE				LONG		
> head(as_tibble(iris)) # A tibble: 6 × 5 Sepal.Length Sepal.Width Petal.Length Petal.Width Species				> iris %>% pivot_longer(cols = -Species, + names_to = "trait", + values_to = "measurement") # A tibble: 600 × 3 Species trait measurement		
1 5.1	3.5	1.4	0.2	setosa	1 setosa	Sepal.Length 5.1
2 4.9	3	1.4	0.2	setosa	2 setosa	Sepal.Width 3.5
3 4.7	3.2	1.3	0.2	setosa	3 setosa	Petal.Length 1.4
4 4.6	3.1	1.5	0.2	setosa	4 setosa	Petal.Width 0.2
5 5	3.6	1.4	0.2	setosa	5 setosa	Sepal.Length 4.9
6 5.4	3.9	1.7	0.4	setosa	6 setosa	Sepal.Width 3

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.

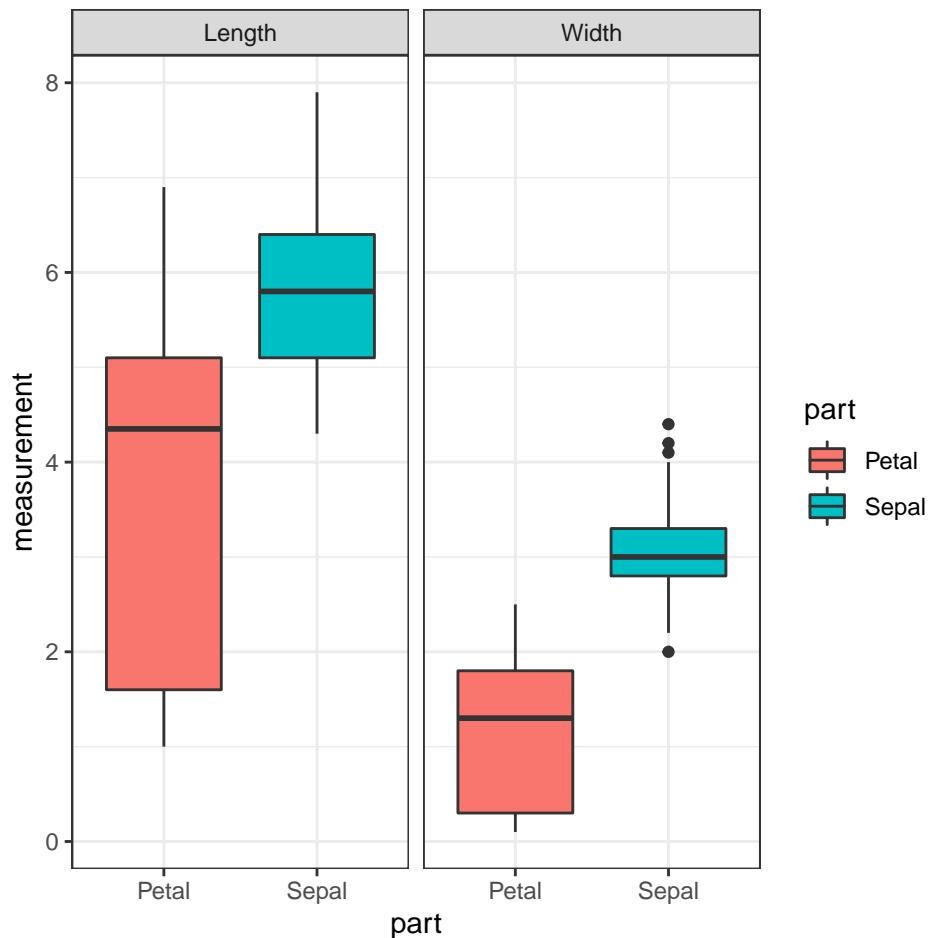
Samme resultat:

```
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 de modsatte til `separate()`.

6.4 Eksempel: Titanic summary statistics

Lad også tage vores Titanic summary statistic eksempel hvor man anvender de forskellige koncepter fra ovenstående.

- `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())
```

- `pivot_longer()`

Vi transformerer eller **reshape** datarammen fra wide data til long data. Vi vil få kun de summary statistics samlet i en enkel kolon, så variablen **Sex** ikke skal med.

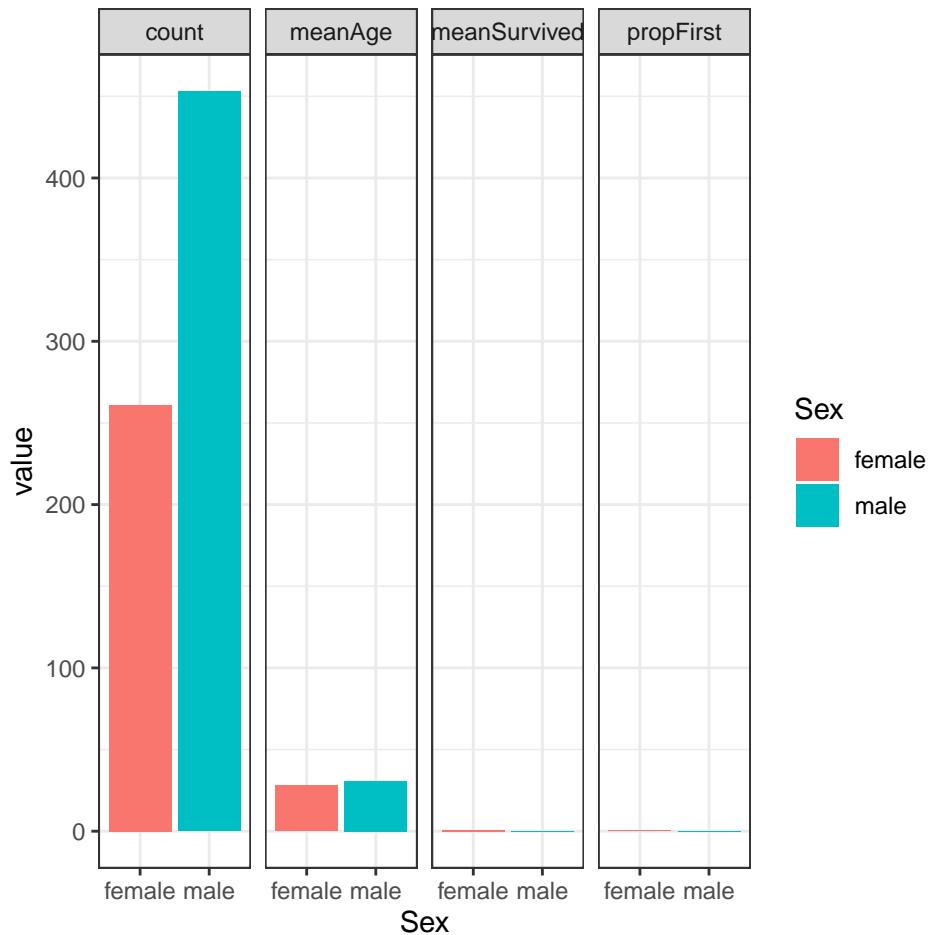
```
titanic_clean_summary_by_sex %>% pivot_longer(cols=-Sex)
```

```
## # 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.

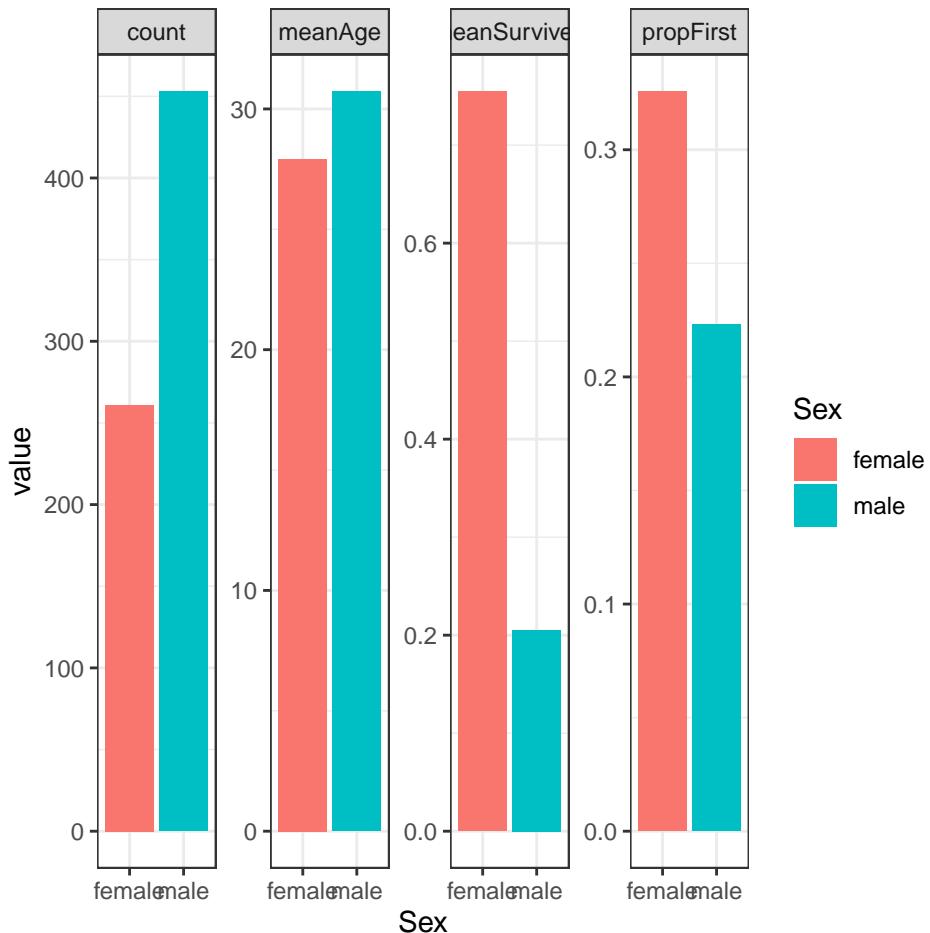
```
titanic_clean_summary_by_sex %>%
  pivot_longer(cols=-Sex) %>%
  ggplot(aes(x=Sex,y=value,fill=Sex)) +
  geom_bar(stat="identity") +
  facet_grid(~name) +
  theme_bw()
```



- `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()

- Wide -> Long

```
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

• Wide -> Long -> Wide

titanic_clean_summary_by_sex %>%
  pivot_longer(cols=-Sex) %>%
  pivot_wider()

## # 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

```

6.5 Problemstillinger

- 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()

```

- 2) *summarise()*. Fra *titanic_clean* beregne den median alder af alle passagerer ombord skibet.

```

titanic_clean %>%
  summarise(...) #rediger her

```

- I samme kommando beregne også den maksimum alder og minimum alder, samt med den standard afvigelse af alder. Datarammen skal ses sådan ud:

```

## # A tibble: 1 x 4
##   mean_alder max_alder min_alder sd_alder
##   <dbl>       <dbl>      <dbl>     <dbl>
## 1     29.7       80        0.42     14.5

```

- 3) *group_by()* og *summarise()*. Beregne samme summary statistics som i 2) men anvende *group_by()* til at først opdelt efter variablen Pclass.

- Lave et barplot med *stat="identity"* som viser den gennemsnitlige alder på y-aksesen opdelt efter Pclass på x-aksen.

- OBS: prøve at tilføje `fill=Pclass` til dit plot og kigge på farverne/legend. Hvad er skete? Kigg på datarammen `titanic_clean` (som skulle være en `tibble` her) og tjekke datatypen.
- Gør variablen til en “factor” variable indenfor plottet.

4) `group_by()` og `summarise()`. Beregne samme summary statistics som i **2)** men anvende `group_by()` til at først opdele efter både variablerne `Pclass` og `Sex`. Man få en advarsel “`summarise()` has grouped output by ‘Pclass’ ...” som du kan se bort fra.

- Lave et barplot med `stat=="identity"` som viser den gennemsnitlige alder opdelt efter `Pclass`, adskilte efter `Sex` (`facet_grid()`)

5) Ekstra øvelse med `group_by()` og `summarise()`. Med `titanic_clean` som udgangspunkt angiv både `Pclass` og `Embarked` i `group_by()` og brug `summarise()` til at beregne de gennemsnitlige `Fare` til passagerene.

- OBS: Der er et par observationer med en blank værdi for “`Embarked`” - bruge `filter()` i linjen før man anvender `group_by()` for at først få dem fjernet (Hint: benyt `!=` tegn).
- Lav et barplot med `Embarked` på x-aksen og den gennemsnitlige `Fare` på y-aksen og adskille efter `Pclass`.
- Hvilke gruppe betalte mest i gennemsnit for deres billet?

6) `group_by()` med tre variabler og `summarise()`. Afprøve en kombination med tre forskellige variabler indenfor `group_by()` og bruge `summarise()` til at beregne middelværdien for `Age`.

- Leg med at lave et plot for at visualisere de data. Idé: som mulighed kan man tilføje variabler til `facet_grid()` - for eksempel `facet_grid(~Pclass + Sex)`.

6 `recode()` Tjekk kursus notaterne og se funktionen `recode()`. Afpøve den ved at ændre værdierne i variablen `Embarked` således at man få de fulde navne af de steder folk gik ombord skibet, i stedet for kun den første bogstav. Fra data beskrivelsen:

- Embarked: Where passengers boarded the titanic. C = Cherbourg, Q = Queenstown, S = Southampton).

7) `pivot_longer()`

- Lave en ny dataramme fra `titanic_clean` der viser tre summary statistics opdelt efter `Survived`: den gennemsnitlige alder, den gennemsnitlige `Fare` og proportionen som er “male”.
- Benyt `pivot_longer()` til at få de resulterende dataramme i Long form. Husk at angiv `-Survived` da vi kun vil have vores beregnet værdier med.
- Bruge resultatet til at lav et barplot med `Survived` på x-aksen og `value` på y-aksen (husk at angiv i plot, at `Survived` er en factor).

- Anvend `facet_wrap()` til at adskille efter `name` og bruger indstillingen `scales="free"`.

8) Mere øvelse med `pivot_longer()`

Her er nogle fiktiv data om hjerterytmene for seks forskellige patienter, efter at have taget fire forskellige lægemidler:

```
heart_rate <- tibble("patient"=c("George", "Sally", "Henry", "Peter", "Charlotte", "Jason"),
heart_rate
```

```
## # A tibble: 6 x 5
##   patient    drugA drugB drugC drugD
##   <chr>     <dbl> <dbl> <dbl> <dbl>
## 1 George      72    74    80    68
## 2 Sally       84    84    88    76
## 3 Henry       64    66    68    64
## 4 Peter        60    58    64    58
## 5 Charlotte    74    72    78    70
## 6 Jason        88    87    88    72
```

- Bruge `pivot_longer()` til at få datasættet i Long form. Er der nogle kolonner som ikke skal med i den enkel kolon med værdier? Kalde den kolon med værdier “heartrate” og den nøgle kolon med variable-navne “drug”.
- Bruge `group_by()` og `summarise()` til at beregne den gennemsnitlige heartrate efter drug.
- Lave et plot som viser den gennemsnitlige heartrate efter drug.

9) `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 `names_from` og `values_from` indenfor `pivot_wider()` - prøv at angiv de relevante kolonner.

10) `Separate()` øvelse

- Benytt funktionen `Separate()` til at opdele variablen `Name` ind til to variabler, “Surname” og “Rest” (Hint: bruge `sep=" "`, ” for at undgå, at man

få en 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: husk at bruge “\\” foran en punktum).
- Optæll hvor mange der er i hver af de forskellige “Titles” og beregne også den maksimum og den minimum alder for hver “Title”.

11) Valgfri ekstra hvis man er færdig: lave en ny dataramme med alle passagerer, der hedder “Alice” eller “Elizabeth” (brug Google her).

6.6 Opgave (fredag workshop om tidyverse)

Fredag opgave bliver tilgængelige fredag morgen.

6.7 Ekstra links

Cheatsheet: <https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf>

6.8 Sidste kommentarer

->

Chapter 7

Forbine tables, iterations og funktioner



7.1 Inledning og læringsmålene

7.1.1 Læringsmålene

I skal være i stand til at:

- Benytte `left_join()` til at tilføje sample information til datasættet.
- Anvende `map()` - funktioner til at udføre beregninger iterativt over flere kolonner og `nest()` til at lave analyser over forskellige dele af datasættet.
- Kombinere `map()` med custom funktioner til at forbedre reproducerbarhed i analyser.

7.1.2 Introduktion til Chapter

Det er ofte tilfældet indenfor biologi, at man har sine data i den ene dataramme og nogle ekstra sample oplysninger i den anden dataramme. Derfor vil vi gerne have en måde, at integrere de to datarammer i R, som gøre, at vi kan inddrage de ekstra oplysninger når vi lave plots af de data.

Det er også ofte tilfældet indenfor biologi, at man har sine data værdier over forskellige kolonner som refererer til rigtige mange samples, replikater eller kon-

ditioner. Vi beskæftiger os med de `map()` funktioner, som kan benyttes til at lave iterativ baserende analyser i R.

7.1.3 Video ressourcer

- Video 1: left_join of tables with extra sample information and plot

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630870>

- Video 2: Introduction to map functions for iterating over columns

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630848>

- Video 3: Introduction to custom functions and combining them with map

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630825>

- Video 4: Introduction to nest functions for breaking data into sections

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/549630798>

7.2 Tilføje sample oplysninger med `left_join()`

For at bedste demonstrere scenariet, har jeg lavet nogle fiktiv data fra et eksperiment, hvor man indhenter målinger over 100 tidspunkter, for to konditioner (treatment og control). Der er tre replikater til hver kondition. Jeg har også lavet en table, som viser forskellige oplysninger om de samples, som ikke er med i de egentlige data.

```
data_exper <- read.table("https://www.dropbox.com/s/hb7m63agz4jti6w/fictive_left_join.txt")
samples <- c("control_rep1", "control_rep2", "control_rep3", "treat_rep1", "treat_rep2", "treat_rep3")
condition <- gsub("(.)_rep[1|2|3]", "\\\1", samples)
replicate <- gsub(".+_rep([1|2|3])", "\\\1", samples)
batch <- c("A", "B", "A", "B", "A", "B")
sample_info <- data.frame("sample"=samples, "condition"=condition, "replicate"=replicate)
```

Lad os kigge på datasættet `data_exper`:

```
head(data_exper)
```

```
##      x control_rep1 control_rep2 control_rep3 treat_rep1 treat_rep2 treat_rep3
## 1  2    -0.6222110    12.112904     8.624800  10.504877 -5.793916  22.26163
## 2  4    -5.7463250    26.082811    -8.929131 -13.287492   8.068806 11.50306
## 3  6     2.7762770     3.834661    14.663878  16.328472  18.107530 10.11893
## 4  8     5.4739947     0.344840    -6.184393   7.770677  7.311946 20.78667
```

```
## 5 10 -9.1930518 10.890655 6.855960 6.299163 21.456005 23.81832
## 6 12 0.2531807 13.079257 -5.423093 8.472125 27.178203 29.94170
```

Lad os også kigge på de sample oplysninger, som kan være nyttige at inddrage i vores analyse/plotter for at undersøge eventuelle batch effekter osv.

```
head(sample_info)

##           sample condition replicate batch
## 1 control_rep1   control        1     A
## 2 control_rep2   control        2     B
## 3 control_rep3   control        3     A
## 4 treat_rep1     treat         1     B
## 5 treat_rep2     treat         2     A
## 6 treat_rep3     treat         3     B
```

Som man kan se, har vi en kolon som viser hvilke kondition og replikate vores samples kommer fra (det er også oplagt fra sample navne her, men det er ikke altid), samt den batch, hver sample kommer fra. For at integrere de to tables, skal vi første have de data i **Long form**.

```
data_long <- data_exper %>% pivot_longer(cols= -x,
                                             names_to="sample",
                                             values_to="measurement")
head(data_long)

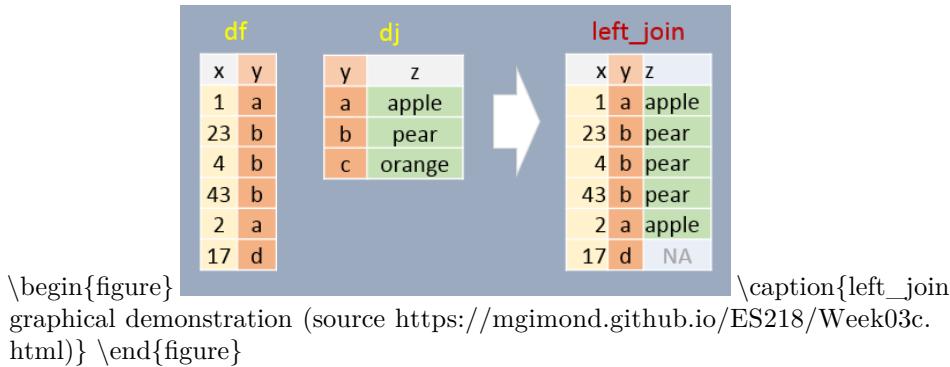
## # A tibble: 6 x 3
##       x sample      measurement
##   <int> <chr>          <dbl>
## 1     2 control_rep1    -0.622
## 2     2 control_rep2     12.1
## 3     2 control_rep3      8.62
## 4     2 treat_rep1      10.5
## 5     2 treat_rep2     -5.79
## 6     2 treat_rep3     22.3
```

7.2.1 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. Her er en meget kort beskrivelse af de fire hoved `join` funktioner.

funktion	Beskrivelse
<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. Her er en grafiske demonstration af `left_join()`:



7.2.2 Anvende `left_join()` for vores fiktiv dataset.

For at bedste forstå funktionen `left_join()` skal vi afprøve den med vores datasæt. Her tager vi udgangspunkt i `data_long` og så tilføjer de 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_long %>% left_join(sample_info, by = "sample")
data_long
```

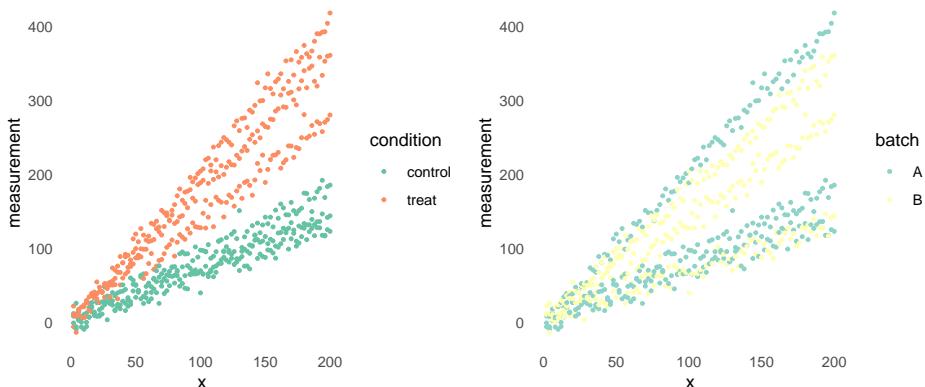
```
## # A tibble: 600 x 6
##       sample      measurement condition replicate batch
##   <int> <chr>          <dbl> <chr>     <chr>    <chr>
## 1 control_rep1 -0.622 control    1        A
## 2 control_rep2  12.1   control    2        B
## 3 control_rep3  8.62   control    3        A
## 4 treat_rep1    10.5   treat     1        B
## 5 treat_rep2   -5.79   treat     2        A
## 6 treat_rep3   22.3   treat     3        B
## 7 control_rep1 -5.75   control    1        A
## 8 control_rep2  26.1   control    2        B
## 9 control_rep3 -8.93   control    3        A
## 10 treat_rep1   -13.3  treat     1        B
## # ... with 590 more rows
```

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 kondition og et plot med en farve til hver batch.

```
gg1 <- ggplot(data_long_join,aes(y=measurement,x=x,colour=condition)) +
  geom_point(size=0.75) +
  theme_minimal() +
  scale_color_brewer(palette = "Set2") +
  theme(panel.grid = element_blank())

gg2 <- ggplot(data_long_join,aes(y=measurement,x=x,colour=batch)) +
  geom_point(size=0.75) +
  theme_minimal() +
  scale_color_brewer(palette = "Set3") +
  theme(panel.grid = element_blank())

library(gridExtra)
grid.arrange(gg1,gg2,ncol=2)
```



Vi kan se, at man kan godt kan skelne imellem de målingerne for de to konditioner, men ikke så meget for de to batches. Det betyder, at vores kondition effekt er stærkere end den batch effekt, som er en god tegn for vores analyse af datasættet.

7.3 Iterativ processer med `map()` funktioner

Når man lave en interaktiv 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. Indenfor biologi er det et meget realistisk scenarie, for eksempel hvis man har mange replikater, konditioner eller tidspunkter og gerne vil beregne noget på dem alle sammen - det kan være at man gerne vil normalisere ekspressionsniveauerne over forskellige gener, osv.

I resten af dette kapitel, lad os beskæftige os med en datasæt der hedder `eukaryotes`, som indeholder meget oplysninger om forskellige organismer som hører til eukaryotes - for eksempel deres navne, gruppe, sub-gruppe, antal proteiner/genes, genom størrelse og så videre. Man kan få de data indlæste med

156 CHAPTER 7. FORBINE TABLES, ITERATIONS OG FUNKTIONER

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")

## # A tibble: 11508 × 19
## # ... with 19 variables:
## #   organism_name     <chr>          center      <chr>
## #   taxid            <dbl>           bioproject_id <chr>
## #   size_mb          <dbl>           gc          <dbl>
## #   scaffolds         <dbl>           genes       <dbl>
## #   proteins          <dbl>           release_date <date>
## #   modify_date       <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 enkel, har jeg brugt `select()` til at kun får de fire variabler `organism_name`, `center`, `group` og `subgroup` i en dataramme.

```
#eukaryotes_full <- eukaryotes
eukaryotes_subset <- eukaryotes %>% select(organism_name, center, group, subgroup)
eukaryotes_subset %>% head()
```

			group	subgroup
## # A tibble: 6 × 4			<chr>	<chr> <chr>
## organism_name		center		Other Other
## <chr>		<chr>		Prot~ Other P~
## 1 Pyropia yezoensis	Ocean University			Plan~ Land Pl~
## 2 Emiliania huxleyi CCMP1516	JGI			Plan~ Land Pl~
## 3 Arabidopsis thaliana	The Arabidopsis Information Resourc~			Plan~ Land Pl~
## 4 Glycine max	US DOE Joint Genome Institute (JGI)			Plan~ Land Pl~
## 5 Medicago truncatula	International Medicago Genome Annot~			Plan~ Land Pl~
## 6 Solanum lycopersicum	Solanaceae Genomics Project			Plan~ Land Pl~

Lad os forestille os, at vi er interesseret i 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).

7.3.1 Introduktion til map() funktioner

Den **tidyverse** løsning til at lave iterativ processer er såkaldte **map()** funktioner, som er en del af pakken **purrr** og er stadig relative nye. 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 lave flere iterativ processer).

`map()` er det *tidyverse* svar til en for loop (eller `apply` hvis man har kendskab til det). 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 fået 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

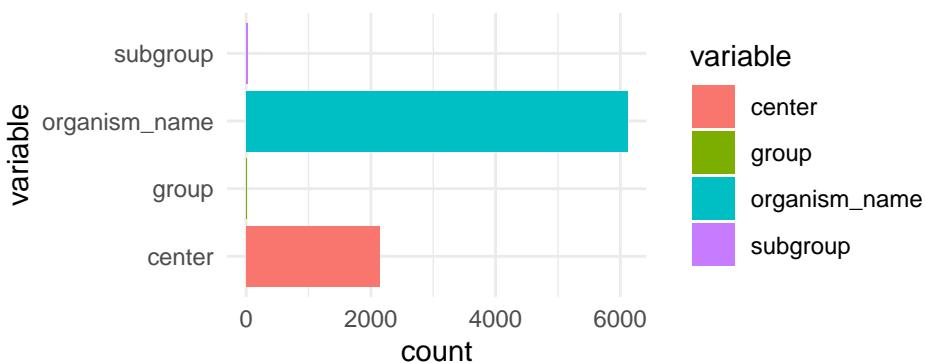
Man kan også bruge `map_df()` for at få en dataramme (`tibble`) tilbage - det er særligt nyttigt for os, fordi vi tager altid udgangspunkt i en dataramme når vi skal få lavet et plot.

```
# 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()
```



7.3.2 Reference for the different map functions

Funktion	Beskrivelse
<code>map_lgl()</code>	returns a logical
<code>map_int()</code>	returns an integer vector
<code>map_dbl()</code>	returns a double vector
<code>map_chr()</code>	returns a character vector
<code>map_df()</code>	returns a data frame

7.4 Custom functions

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.

7.4.1 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
my_function_name	funktion navn
<- function(.x)	fortæl R, at vi lave en funktion med nogle data .x
sum(.x)/length(.x)	brug data .x til at beregne middelværdi
return()	hvad 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
```

7.4.2 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>
```

```

##           <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

```

eller når `nth` is a **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(~n_distinct(.x,na.rm = TRUE))

## # 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

```

```
iris %>%
  map_df(~.x %>% n_distinct(na.rm = TRUE)) #fordi 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 #en lang funktion

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

```

7.5 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 opelde 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 the Iris dataset using the `nest()` function. On the left, the original wide-format dataset is shown as a table with columns: Species, S.L, S.W, P.L, and P.W. The rows represent different observations for three species: setosa, versi, and virginii. An arrow points to the right, where three narrow-format datasets are shown for each species. Each narrow-format dataset has two columns: Species and Measurements. The Measurements column contains a tibble with four columns: S.L, S.W, P.L, and P.W. The values in the Measurements column correspond to the values in the original wide-format table for each species.

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
virginii	6.3	3.3	6.0	2.5
virginii	5.8	2.7	5.1	1.9
virginii	7.1	3.0	5.9	2.1
virginii	6.3	2.9	5.6	1.8
virginii	6.5	3.0	5.8	2.2

Species	Measurements																								
setosa	<table border="1"> <thead> <tr> <th>S.L</th> <th>S.W</th> <th>P.L</th> <th>P.W</th> </tr> </thead> <tbody> <tr><td>5.1</td><td>3.5</td><td>1.4</td><td>0.2</td></tr> <tr><td>4.9</td><td>3.0</td><td>1.4</td><td>0.2</td></tr> <tr><td>4.7</td><td>3.2</td><td>1.3</td><td>0.2</td></tr> <tr><td>4.6</td><td>3.1</td><td>1.5</td><td>0.2</td></tr> <tr><td>5.0</td><td>3.6</td><td>1.4</td><td>0.2</td></tr> </tbody> </table>	S.L	S.W	P.L	P.W	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
S.L	S.W	P.L	P.W																						
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	<table border="1"> <thead> <tr> <th>S.L</th> <th>S.W</th> <th>P.L</th> <th>P.W</th> </tr> </thead> <tbody> <tr><td>7.0</td><td>3.2</td><td>4.7</td><td>1.4</td></tr> <tr><td>6.4</td><td>3.2</td><td>4.5</td><td>1.5</td></tr> <tr><td>6.9</td><td>3.1</td><td>4.9</td><td>1.5</td></tr> <tr><td>5.5</td><td>2.3</td><td>4.0</td><td>1.3</td></tr> <tr><td>6.5</td><td>2.8</td><td>4.6</td><td>1.5</td></tr> </tbody> </table>	S.L	S.W	P.L	P.W	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
S.L	S.W	P.L	P.W																						
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																						
virginii	<table border="1"> <thead> <tr> <th>S.L</th> <th>S.W</th> <th>P.L</th> <th>P.W</th> </tr> </thead> <tbody> <tr><td>6.3</td><td>3.3</td><td>6.0</td><td>2.5</td></tr> <tr><td>5.8</td><td>2.7</td><td>5.1</td><td>1.9</td></tr> <tr><td>7.1</td><td>3.0</td><td>5.9</td><td>2.1</td></tr> <tr><td>6.3</td><td>2.9</td><td>5.6</td><td>1.8</td></tr> <tr><td>6.5</td><td>3.0</td><td>5.8</td><td>2.2</td></tr> </tbody> </table>	S.L	S.W	P.L	P.W	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
S.L	S.W	P.L	P.W																						
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																						

Species	Measurements
setosa	<tibble [50x4]>
versi	<tibble [50x4]>
virginii	<tibble [50x4]>

i.e.,

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
## 5 Galdieria sulphuraria     Galdieria sulphuraria Genome P~ Other
## 6 Phaeodactylum tricornutum CCAP 1055/1 Diatom Consortium Other
```

Hvis vi gerne vil tilbage til vores oprindeligt datasæt, kan vi brug unnest() og specificer 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
## 5 Other Galdieria sulphuraria     Galdieria sulphuraria Ge~ Other
## 6 Other Phaeodactylum tricornutum CCAP 1055/1 Diatom Consortium Other
```

Spørgsmålet er: hvordan kan vi inddrage “nested” data indenfor vores analyser?

7.5.1 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 med
  median(.x,na.rm=T) # `na.rm` fjerne `NA` værdier)

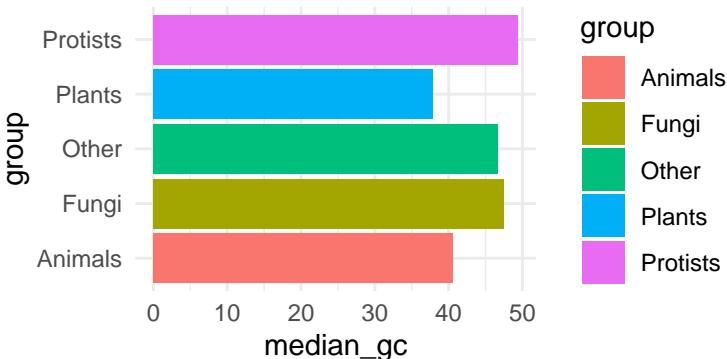
eukaryotes_gc_by_group <- eukaryotes %>%
  group_by(group) %>%
  nest() %>%
  mutate("median_gc"=map_dbl(data, func_gc))
eukaryotes_gc_by_group

## # 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:

```
eukaryotes_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 <- ~ .x %>% pull(genes)      %>% median(.x,na.rm=T)
func_proteins <- ~ .x %>% pull(proteins)  %>% median(.x,na.rm=T)
func_size <- ~ .x %>% pull(size_mb)       %>% median(.x,na.rm=T)
```

Anvende nest():

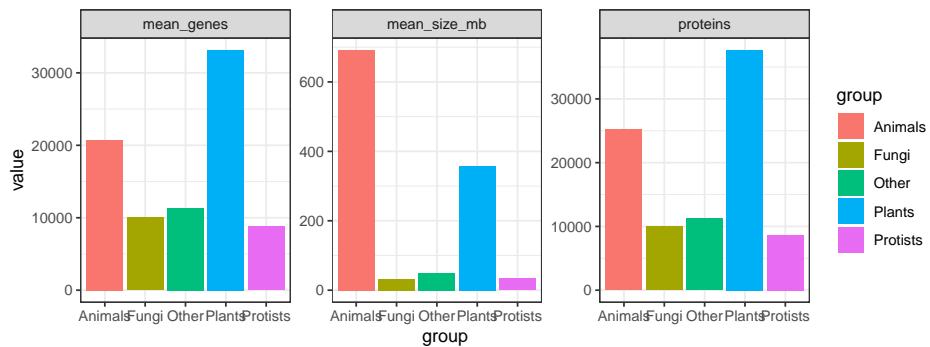
```
eukaryotes_nested <- eukaryotes %>%
  group_by(group) %>%
  nest()
```

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()
```



7.6 Problemstillinger

1) Lave Quiz på Absalon “Quiz - tables, maps and functions”

2) `left_join()` øvelse. Kør følgende kode:

```
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 tables - `superheroes` og `publishers`. Hvilke kolon kan man bruge til at forbinde de to tables? Brug `left_join()` til at tilføje oplysninger fra `publishers` til datarammen `superheroes`.

- Få man alle observationer fra `superheroes` med i resultatet?
- Benyt `inner_join()` til at forbinde `publishers` til `superheroes` - få man så nu alle observationer med?
- Benyt `full_join()` til at forbinde `publishers` til `superheroes` - hvor mange observationer får man med nu? Hvorfor?

3) `left_join()` øvelse.

```
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"
)

head(iris2)

## # A tibble: 6 x 5
##   sample1 sample2 sample3 sample4 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

samp_table

## # A tibble: 4 x 3
##   sample part  measure
##   <chr>  <chr> <chr>
## 1 sample1 Sepal Length
## 2 sample2 Sepal Width
## 3 sample3 Petal Length
## 4 sample4 Sepal Width
```

Man kan se, at vi har to tables - `iris2` og `sample_table`. `iris2` er ikke særlig informativ med hensyn til hvad de samples er, men oplysningerne står

i `sample_table`. Bruge `left_join()` til at tilføje `sample_table` til `iris2` for at få en dataramme som indeholder både de data og de samples oplysninger.

4) `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:

- Select variabler `carat`, `depth`, `table` og `price` og beregne den median værdi til hver. Resultatet skulle være en list (anvende default `map()` funktion).
- Select variabler `cut`, `color` og `clarity` og beregne antallet af distinkt værdier til hver. Resultatet skal være en double.
- Select alle variabler og return de datatyper (funktionen `typeof()`). Resultatet skal være en dataramme.

5) `map()` øvelse med 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)) #non-default funktion
```

- Afprøve følgende kode linjer og beskrive hvad der sker.

```
diamonds %>% select(carat, depth, price) %>% map_df(~(.x-mean(.x)) )
diamonds %>% select(carat, depth, price) %>% map_df(~ifelse(.x>mean(.x), "big_value", "small_value"))
diamonds %>% filter(cut=="Ideal") %>% select("color","clarity") %>% map(~sum(.x==nth(.x)))
```

Brug funktioner indenfor `map()` til at beregne følgende:

- Select alle variabler og output den 100th observation. Resultatet skal være en list.
- Select variabler `carat`, `depth`, `table` og `price` og for hver kolon tilføj tre og så tag square ($\wedge 2$). Resultatet skal være en dataramme.

- Select variabler `carat`, `depth`, `table` og `price` og angiv `TRUE` hvis den første værdi er større en den median værdi, ellers `FALSE`. Resultatet skal være en `logical`.
- Select variabler `carat`, `depth`, `table` og `price` og beregne den `log2` transformering til være. Resultatet skal være en dataramme. Brug resulterende dataramme til at lave et scatter plot af `log2(carat)` på x-aksen og `log2(price)` på y-aksen.

6) nest øvelse

a) For datasættet `iris`, anvend `group_by(Species)` og tilføj dernæst `nest()`, og kigger på resultatet.

- tilføj `pull(data)` og se på resultatet
- tilføj prøve også `pluck("data", 1)` 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_cor = map_dbl(data, ~cor(.x$Sepal.Width, .x$Sepal.Length)))`
 - `mutate(new_column_sum_SW = map_dbl(data, ~.x %>% pull(Sepal.Width) %>% sum))`
- Bemærk at vi har brugt `map_dbl` til at få tal som vi nemt kan læse i kolonner - prøve bare `map()` i stedet for og se resultatet. Man er nødt til at bruge `unnest()` til at se resultatet i dette tilfælde.

b)

- Afprøve følgende funktioner indenfor samme ramme og angiv hensigtsmæsigt kolon navne i `mutate()`.

```
my_func_1 <- ~.x %>%
  pull(Petal.Length, Petal.Width) %>%
  sum

my_func_2 <- ~cor(.x$Sepal.Width, .x$Sepal.Length)

my_func_3 <- ~ t.test(.x$Petal.Width, .x$Sepal.Length)$statistic
```

c)

- Indenfor samme ramme tilføj selv kode linjer som beregner til hver af de tre datasæt:
 - den maksimum værdi
 - den maksimum værdi for `Petal.Width`
 - den gennemsnitlige værdi af `Petal.Width/Petal.Length`
 - den gennemsnitlige værdi af `Sepal.Width>3`

d)

- Lav et barplot for nogle dine beregninger, adskilte efter de forskellige statistikker.

– Husk at få fjernet kolonnen `data` før man anvende `pivot_long()`.

7) Introduktion til næste lektion

For at se værdien af at bruge `group_by()` og `nest()` 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()` som i sidste spørgsmål
- tilføj `mutate()` til at lave en ny kolon som hedder `t_test` og bruge funktionen indenfor `map()`.
- tilføj `pull(t_test)` - man får de tre t-test 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 gøre dem tidy først.

-
- Nu installer R-pakken `broom` (`install.packages("broom")`)
 - Lav samme som ovenstående men bruge følgende funktion i stedet for.
 - `glance()` få de statistik fra `t.test()` ind i en pæn form (`tidy`)

```
library(broom)
```

```
## Warning: pakke 'broom' blev bygget under R version 4.0.5
my_func <- ~ t.test(.x$Petal.Width,.x$Sepal.Length) %>% glance()
```

- Tilføj `pull` eller `unnest` som før og se på resultatet.
- Man får en pæn dataramme frem med alle de forskellige statistik fra `t.test()`.

7.7 Ekstra notater og næste gang

<https://r4ds.had.co.nz/iteration.html> <https://sanderwuyts.com/en/blog/purrr-tutorial/>

Chapter 8

Visualising trends



```
#load packages
library(ggplot2)
library(tidyverse)
library(gridExtra)
library(broom)
library(glue)
```

```
FALSE Warning: pakke 'glue' blev bygget under R version 4.0.5
```

8.1 Indledning og læringsmålene

8.1.1 Læringsmålene

I 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 trend linjer.
- Kombinere `map()` og `lm()` til at beregne regression statistikke for flere lineær regression modeller og tilføje dem til plottet til at gøre det mere informativ.

8.1.2 Introduktion til chapter

I dette chapter demonstrerer jeg hvordan man anvende den `nest()` og `map()` struktur som vi så sidste gange til at lave statistiske analyser med korrelation og lineær regression. Vi lærer hvordan vi kan visualisere trends og korrelations og tilføj relevante statistikker til plots til at gøre dem endnu mere informativ.

8.1.3 Video ressourcer

(OBS: kunne desværre ikke nå videoerne til i dag pga. forkølelse/travlhed men kan eventuelle lave dem senere hvis der er efterspørgsel)

- Video 0: Korrelation koefficient med `nest()` og `map()`
- Video 1: Lineær regression linjer med `ggplot2`
- Video 2: Lineær regression med `nest()` og `map()`
- Video 3: Tilføj labels med `lm()` statistik på plottet

8.2 `nest()` og `map()`: eksempel med korrelation

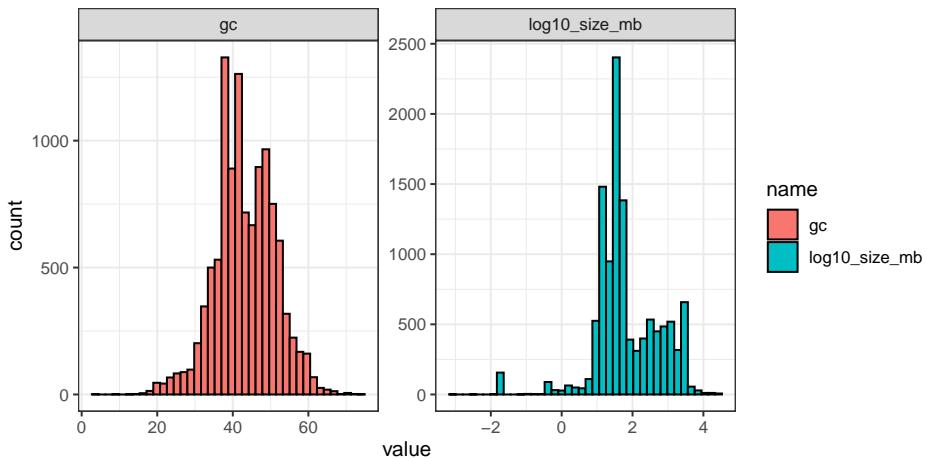
Vi kigger på korrelation analyse først men gentage samme struktur med `nest()` og `map()` når vi gennemgå lineær regression.

8.2.1 Korrelation analyse i R

Man kan lave en korrelation analyse i R ved at anvende `cor.test()`. For eksempel, forestille os at vi gerne vil finde ud af korrelationen mellem gc content (variablen `gc`) og genes (variablen `genes`) for de data `eukaryotes` fra sidste lektion. Vi plotter en histogram og beslutter os for at lave korrelation mellem `gc` og den transformerede variable `log10(genes)`.

```
eukaryotes %>%
  mutate(log10_size_mb = log10(size_mb)) %>%
  select(log10_size_mb,gc) %>%
  pivot_longer(everything()) %>%
  ggplot(aes(x=value,fill=name)) +
  geom_histogram(bins=40,colour="black") +
  facet_wrap(~name,scales="free") +
  theme_bw()

## Warning: Removed 388 rows containing non-finite values (stat_bin).
```



Vi mistænker, at der kan være nogle sub-strukturer indenfor de data - for eksempel over de forskellige organismer grupper i variablen `Group`. Vi benytter alligevel `cor.test()` til at teste for korrelation mellem `gc` og `log10(size_mb)` over hele datasæt:

```
my_cor_test <- cor.test(eukaryotes$gc, log10(eukaryotes$size_mb))
my_cor_test

##
## Pearson's product-moment correlation
##
## data: eukaryotes$gc and log10(eukaryotes$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
```

Her vil jeg også gerne introducerer en funktion der hedder `glance()` som findes i R-pakken `broom`. Funktionen `glance()` anvendes til at tage den output fra en statistiske test (fk. `cor.test()`) og lave det om til et `tidy` dataramme. Det gøre det nemmere for eksempel til at lave et plot, eller samler op statistikker fra forskellige tests.

```
library(broom)
glance(my_cor_test)

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
```

Vi kan se at over hele datasæt, er der en signifikant negativ korrelation (estimate -0.147 og p-værdi 8.25054×10^{-55}) mellem de to variabler, men vi er dog stadig mistænksom overfor eventuelle forskelligheder blandt de fem grupper fra variablen `group`.

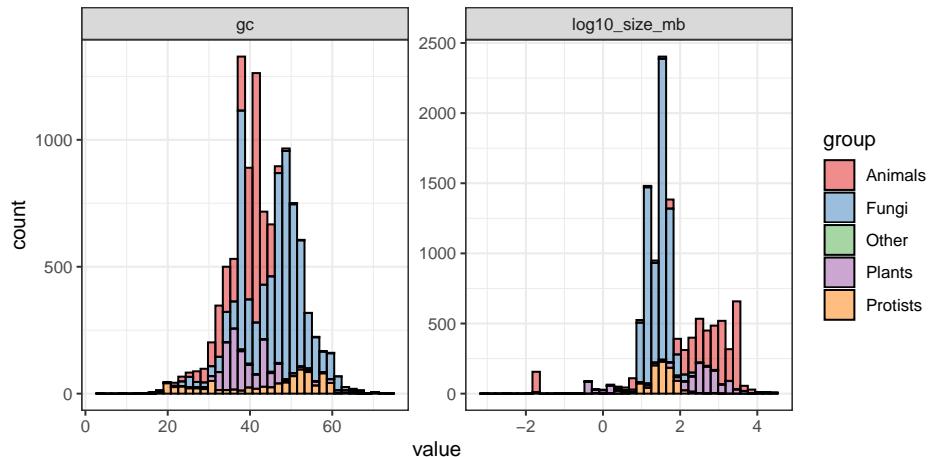
Vi vil gerne gentage vores nalyse for hver af de fem grupper fra `group`. En god tilgang til at undersøge det er at bruge den ramme med `group_by()` og `nest()` som vi lært sidste gange.

8.2.2 Korrelation over flere datasæt på en gang

Lad os først tjekke fordelingen af de to variabler opdelt efter variablen `group`. Bemærk at når de to variabler er i forskellige kolonner er vi nødt til at først få vores data i long form med `pivot_longer()`.

```
eukaryotes %>%
  mutate(log10_size_mb = log10(size_mb)) %>%
  select(log10_size_mb, gc, group) %>%
  pivot_longer(-group) %>%
  ggplot(aes(x=value, fill=group)) +
  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_bin).
```



Vi kan se, at der er forskelligheder blandt de fem grupper og der kan forekomme forskellige sammenhænge mellem de to variabler. Vi benytter i følgende den `group_by() + nest()` ramme som blev introducerede sidste lektion.

Definere korrelation funktion

Lad os definere den korrelation test mellem `gc` og `size_mb` i en funktion:

```
cor_test <- ~cor.test(.x$gc,log10(.x$size_mb))
```

Husk:

- Brug `~` liget i starten for at fortælle R, at vi arbejde med en funktion.
- Specifier det indiv datasæt indenfor `cor.test()` med `.x` (husk at `map()` her fungere over en liste af datasæt, så indenfor et datasæt specificerer vi kolonner `gc` og `size_mb` ved `.x$gc` og `.x$size_mb`).

Vi vil gerne få statistikker fra `cor.test()` i en pæn form så vi tilføjer `glance()` i funktionen:

```
cor_test <- ~cor.test(.x$gc,log10(.x$size_mb)) %>% glance()
```

Benytte `group_by()` + `nest()`

For at lave et `cor.test()` til hver af de fem grupper i variablen `group()`, skal vi først anvende `nest()`:

```
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 18]>
## 2 Protists <tibble [888 x 18]>
## 3 Plants    <tibble [1,304 x 18]>
## 4 Fungi     <tibble [6,064 x 18]>
## 5 Animals   <tibble [3,201 x 18]>
```

Vi har fået vores fem sub-datasæt i en liste der hedder `data`.

Bruge `map()` på de nested datasæt

Nu lad os køre vores funktion på den nested data. Vi bruger `map()` til at lave samme funktion for hver af de fem datasæt. Vi bruger `map` indenfor `mutate` til at lave en ny kolonne der hedder `test_stats`, hvor resultaterne for hver af de fem tests kan lagres.

```
eukaryotes_cor <- eukaryotes_nest %>%
  mutate(
    test_stats=map(data,cor_test),
  )
eukaryotes_cor
```

```
## # A tibble: 5 x 3
```

```
## # Groups:   group [5]
##   group    data           test_stats
##   <chr>    <list>         <list>
## 1 Other    <tibble [51 x 18]>  <tibble [1 x 8]>
## 2 Protists <tibble [888 x 18]>  <tibble [1 x 8]>
## 3 Plants   <tibble [1,304 x 18]> <tibble [1 x 8]>
## 4 Fungi    <tibble [6,064 x 18]> <tibble [1 x 8]>
## 5 Animals  <tibble [3,201 x 18]> <tibble [1 x 8]>
```

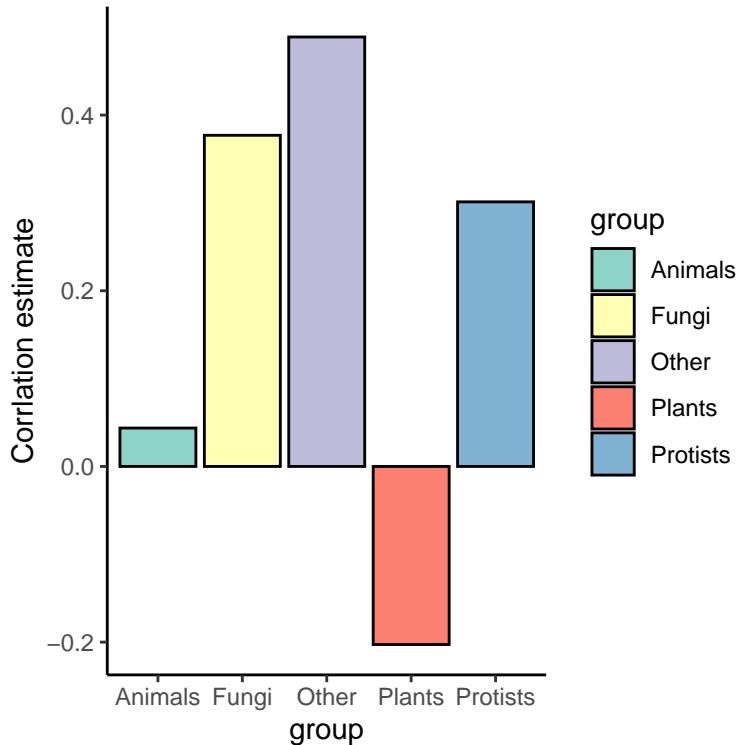
For at kunne se de statistics skal man bruge funktionen `unnest()` på variablen `test_stats`:

```
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>
```

Vi kan bruge den direkte i et plot. Lad os fokusere på den korrelaton koefficient i kolonnen `estimate` og omsætte den til et plot:

```
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
```



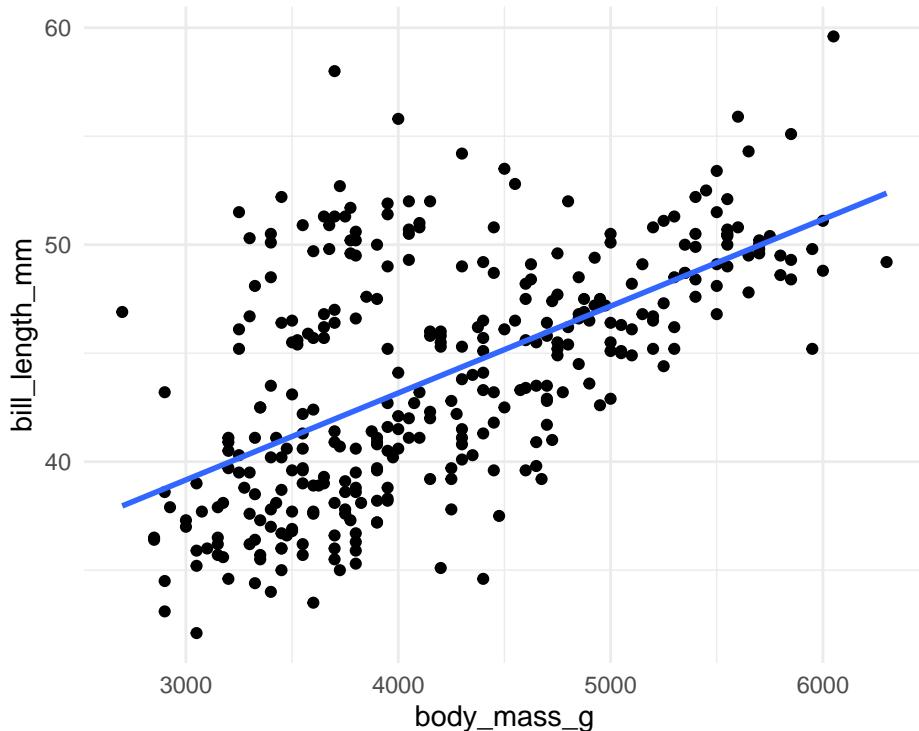
Vi kan gå videre med vores plot og tilføje nogle labels efter p-værdien for at gøre det mere informativ - lad os først gå videre til at lave en lignende analyse med lineær regression.

8.3 Visualisering af lineær regression med ggplot2

8.3.1 Lineær trends

Man kan benytte lineær regression til at visualisere trends i de data. For eksempel kan man se i følgende scatter plot mellem `body_mass_g` og `bill_length_mm` i datasættet `penguins` 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'
```



Den bedste rette linje har en formel $y = a + bx$, hvor a er den “intercept” og b er den “slope” (hælde) af linjen. Idéen med simpel lineær regression er, at man gerne vil finde den 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, kan man bruger `lm` indenfor R til at finde den bedste rette linje. For eksempel:

```
lm(bill_length_mm~body_mass_g, data=penguins)
```

```
## 
## Call:
## lm(formula = bill_length_mm ~ body_mass_g, data = penguins)
## 
## Coefficients:
## (Intercept) body_mass_g
##   27.150722     0.004003
```

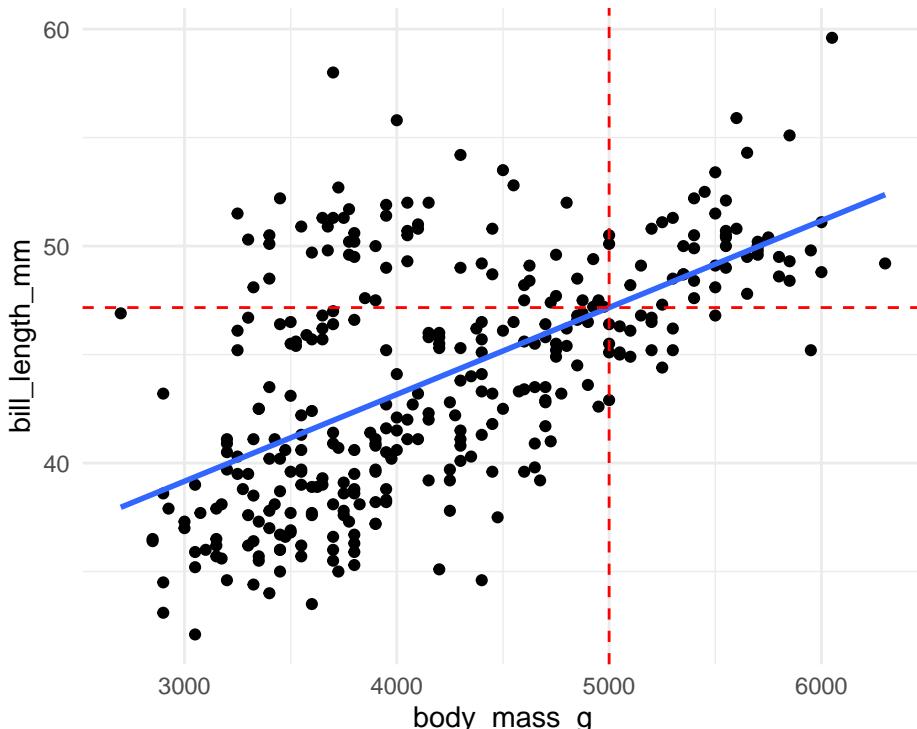
Her kan man se, at den intercept er 27.15 og den slope er 0.004 - det betyder, at hvis `body_mass_g` stiger ved 1, så ville den forventet `bill_length_mm` stige ved 0.004. Man kan således bruge linjen til at lave forudsigelser. For eksempel, hvis jeg vejede en ny pingvin og fandt ud af, at den vejede 5000 gramms, kunne jeg bruge min linje som den bedste gætte på dens bill længde:

```
y <- 27.150722 + 0.004003 * 5000
y
```

```
## [1] 47.16572
```

Så kan man se, jeg forventer en pingvin af 5000 g til at have en bill length af omkring 47.2 mm. Den kan også visualiseres i et plot:

```
## `geom_smooth()` using formula 'y ~ x'
```

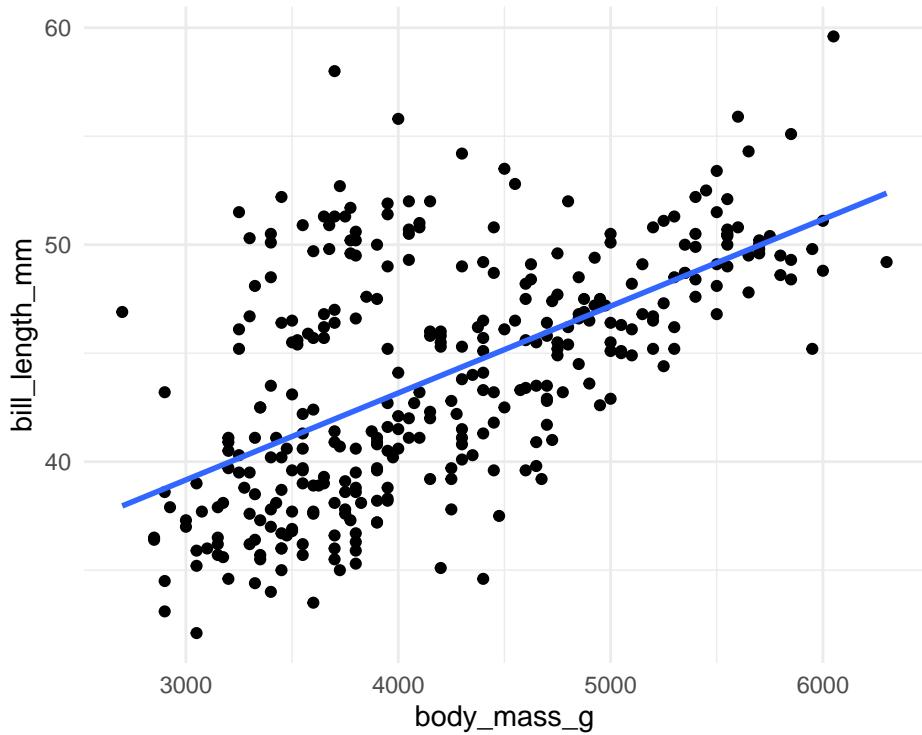


8.3.2 plotting lineær trend lines with `geom_smooth()`

Indbygget i ggplot2 er en funktion der hedder `geom_smooth()` som kan bruges til at tilføje den bedste rette linje til plottet. Man benytte den simpelthen ved at specificere `+ geom_smooth(method="lm")` indenfor `plot` kommando:

```
ggplot(penguins,aes(x=body_mass_g,y=bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm",se=FALSE)
```

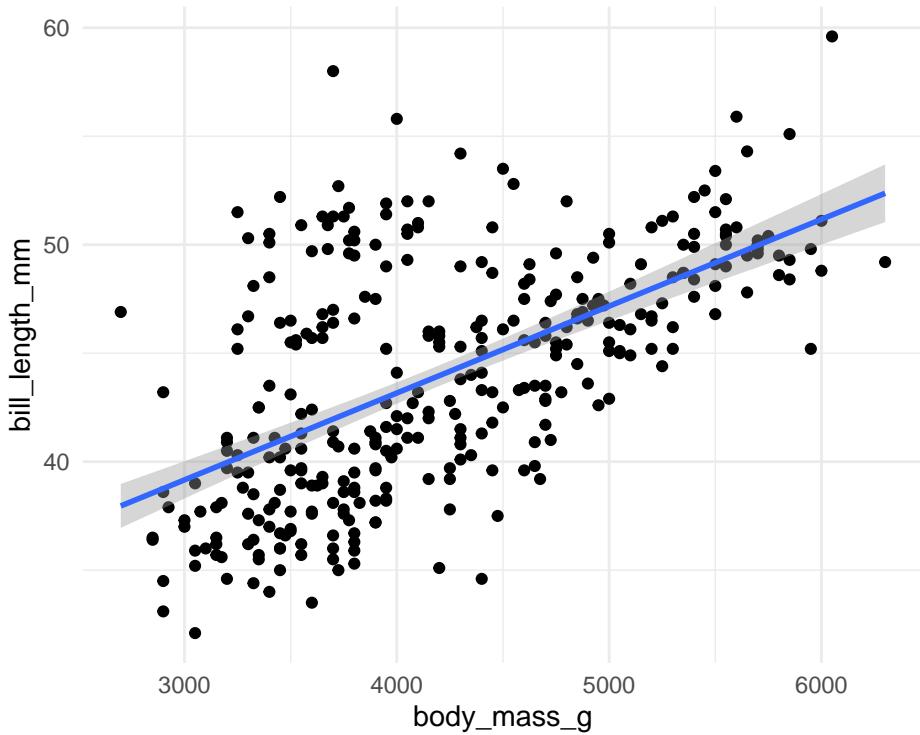
```
## `geom_smooth()` using formula 'y ~ x'
```



Det er nemt at bruge (bare tilføj en kode linje) og at man kan få en konfidensinterval med, hvis man gerne vil have den: i ovenstående plot specificeret jeg `se=FALSE` men hvis jeg angiv `se=TRUE` (default), får jeg følgende plot:

```
ggplot(penguins,aes(x=body_mass_g,y=bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm", se=TRUE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

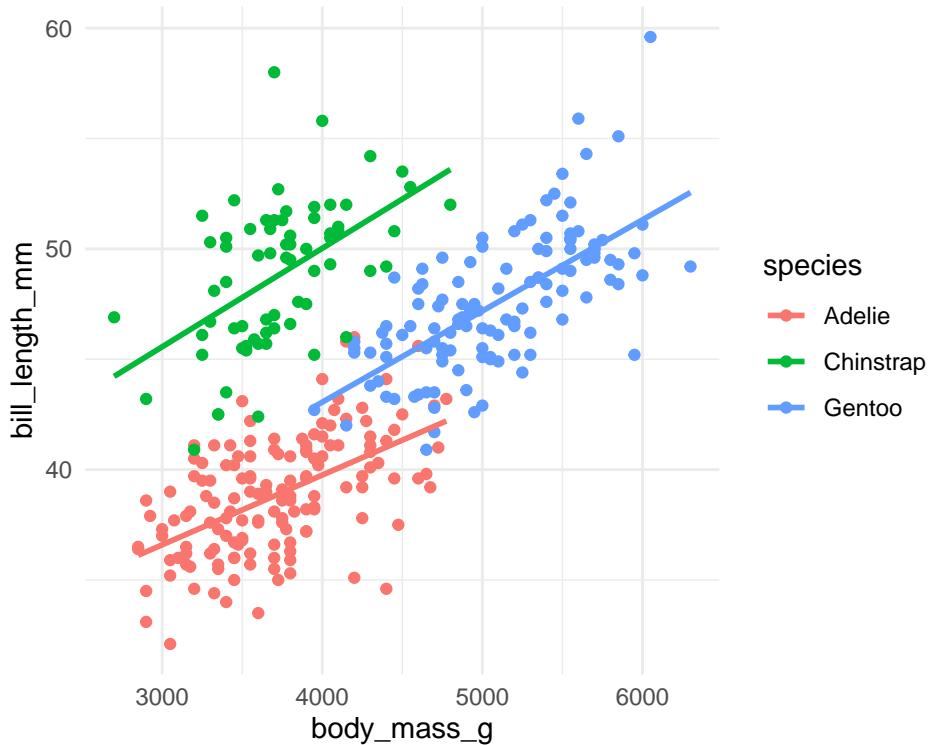


8.3.3 plotting multiple lineær trend lines with geom_smooth()

For at tilføje en bedste rette linje til hver af de tre `species` i stedet for samtlige data, er det meget nemt i ggplot: man angiver bare `colour=species`:

```
ggplot(penguins,aes(x=body_mass_g,y=bill_length_mm,colour=species)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm",se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



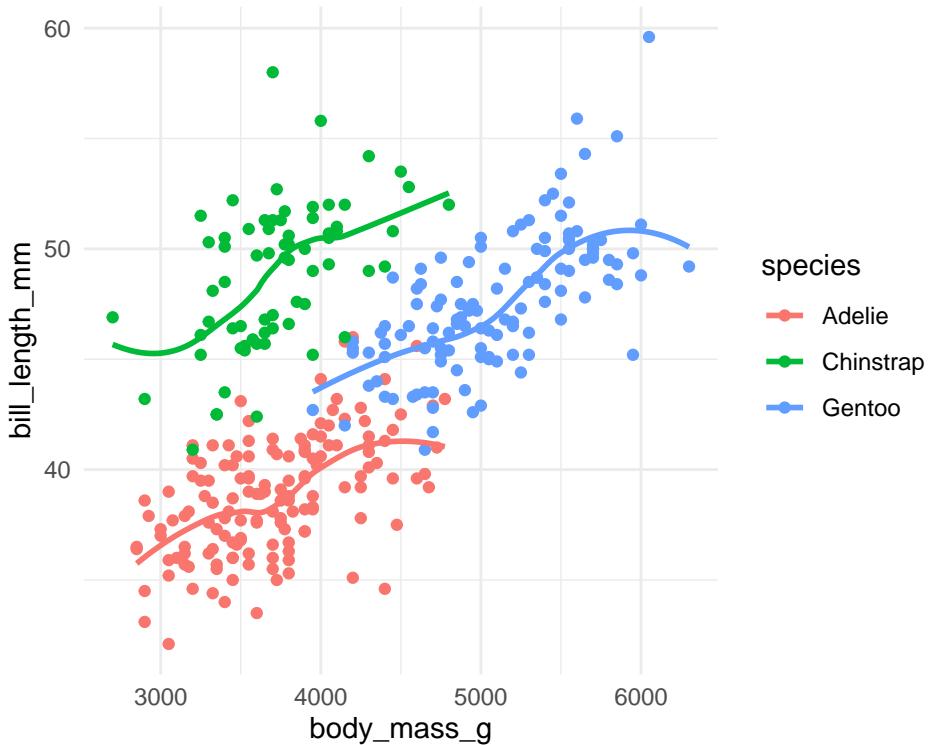
Så kan vi se, at der faktisk er tre forskellige trends her, så det giver god mening at bruge de tre forskellige linjer i stedet for kun en.

8.3.4 Plot trends med `method=="loess"` i ggplot.

I ggplot er vi ikke begrænset til `method="lm"` indenfor `geom_smooth()`. Lad os afprøve i stedet `method="loess"`:

```
library(palmerpenguins)
penguins <- na.omit(penguins)
ggplot(penguins,aes(x=body_mass_g,y=bill_length_mm,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 det er mere ligefrem at beskrive en lineær trend og bruge den til at beregne de nødvendige statistik til at støtte vores observationer.

8.4 Plot linear regresion estimates

For at finde vores estimates og tjekke signifikansen af de lineær trend, skal man arbejde direkte med den lineær model funktion `lm` som i ovenstående:

```
my.lm <- lm(bill_length_mm~body_mass_g, data=penguins)
my.lm
```

```
##
## Call:
## lm(formula = bill_length_mm ~ body_mass_g, data = penguins)
##
## Coefficients:
## (Intercept)  body_mass_g
##   27.150722    0.004003
```

For at finde ud af den signifikans kan man kigge på `summary`

```
summary(my.lm)

##
## Call:
## lm(formula = bill_length_mm ~ body_mass_g, data = penguins)
##
## Residuals:
##    Min     1Q   Median     3Q    Max 
## -10.1652 -3.0664 -0.7672  2.2356 16.0371 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 2.715e+01 1.292e+00 21.02   <2e-16 ***
## body_mass_g 4.003e-03 3.016e-04 13.28   <2e-16 ***  
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
##
## Residual standard error: 4.424 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
```

De tal, som er vigtige her:

- Den p-værdi: <2e-16 - den trend er statistiske signifikant.
- Den R-squared værdi - det viser den proportion af variansen i `bill_length_mm` som `body_mass_g` forklarer. Det er svarende til den korrelation koefficient squared og kan fortolkes således -
 - 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 Iteratively estimating multiple trends

Vi kan benytte de koncepter som vi lært i sidste lektion, og som er samme som ovenpå i vores korrelation analyse.

Vi vil gerne lave en model med `lm` som ovenpå, og vi tilføje `glance()` til at få de model statistikker i en pæn form.

```
lm_model_func <- ~lm(bill_length_mm~body_mass_g,data=.x) %>% glance()
```

Vi bruge `group_by` til at opdele efter de tre `species` og så nest de tre datarammer:

```
penguins %>%
  group_by(species) %>%
  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]>
```

Vi lave en lineær model på hver af de tre datasæt med `map` og ved at specifie funktion som vi defineret ovenpå. Vi bruger `mutate` ligesom før til at tilføje resulterende statistikker som en ny kolon der hedder `lm_stats`:

```
penguins_lm <- penguins %>%
  group_by(species) %>%
  nest() %>%
  mutate(
    lm_stats=map(data,lm_model_func),
  )
penguins_lm

## # A tibble: 3 x 3
## # Groups:   species [3]
##   species   data      lm_stats
##   <fct>    <tibble [146 x 7]> <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     2.24     60.6 1.24e-12     1 -324.
## 2 Gentoo   <tibble>    0.445      0.440     2.32     93.6 1.26e-16     1 -268.
## 3 Chinstrap <tibble>    0.264      0.253     2.89     23.7 7.48e- 6     1 -168.
## # ... with 5 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
## #   df.residual <int>, nobs <int>
```

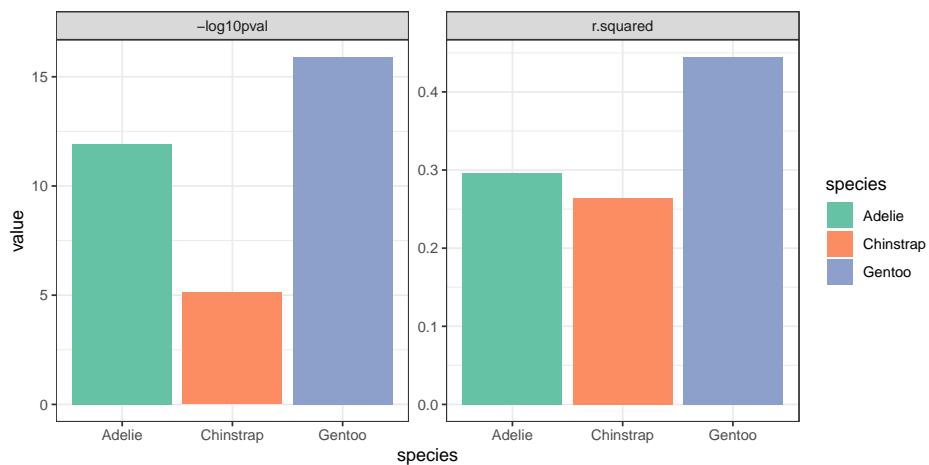
Så kan vi se, at vi har fået en dataramme med vores lineær model statistics. Lad os omsætte dem til et plot for at sammenligne dem over de tre `species` af pingvin.

```
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) +
theme_bw()

```



8.4.2 Lave og tilføje labels til dit regression plot

Som nævnt tidligere, kan det være nyttig at tilføje nogle labels til vores plots med de statistikker, vi lige har beregnet. I tilfældet af vores lineær regression trend lines, vil vi gerne tilføje de r-squared værdier og de p-værdier.

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

```

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

```

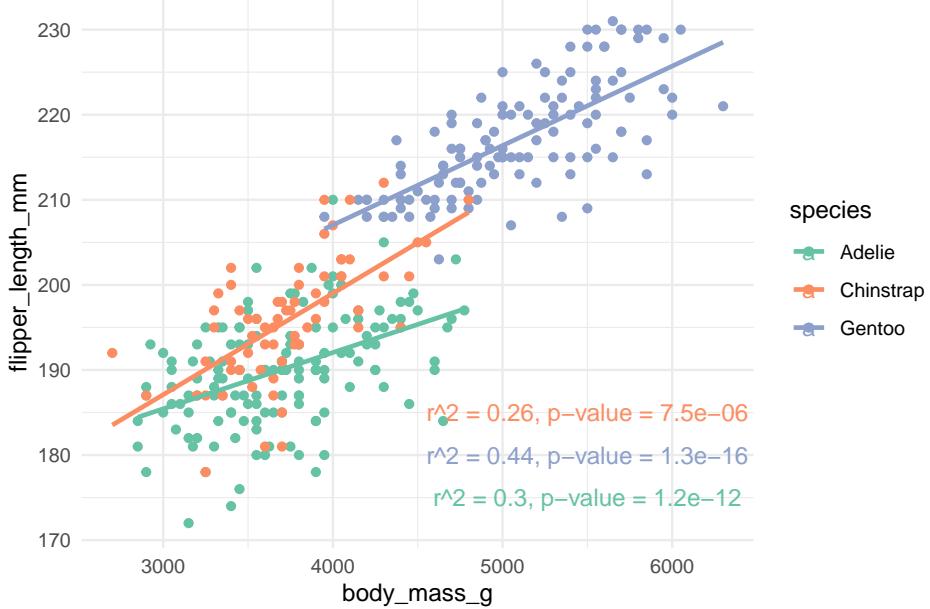
Vi kan tilføje vores label data indenfor `geom_text()`. `x` og `y` specificere hvor i plottet teksten skal være, og husk at specificere `data=label_data` og `label=label` skal stå indenfor `aes()` når det handler om en variable i `label_data`.

```

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()

```

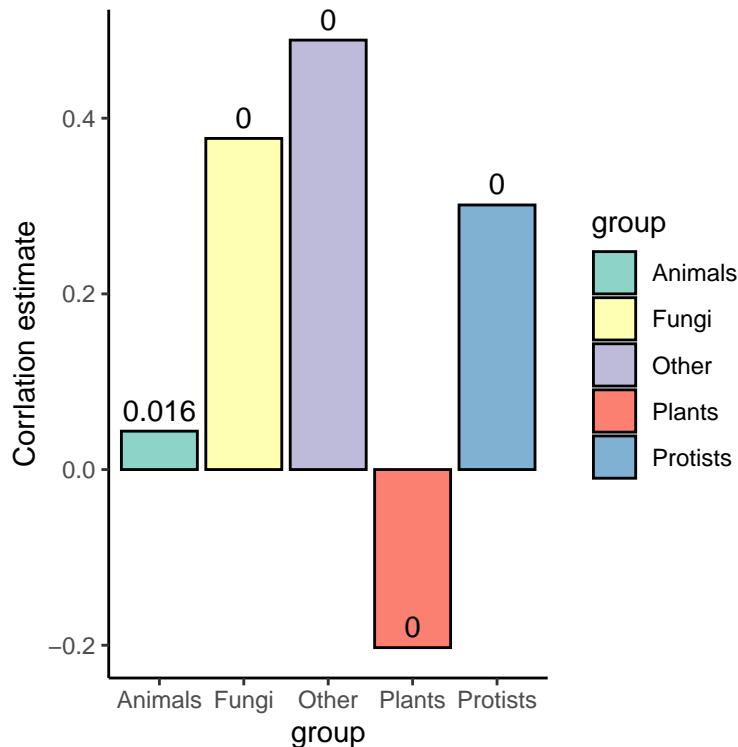
`## `geom_smooth()` using formula 'y ~ x'`



8.4.3 `case_when()`: adding bar plot text

Det vil også være rart at tilføje vores p-værdier i vores barplot fra ovenstående korrelation analyse. Den kan laves igen med `geom_text`. Husk at vi bruge lokale aethetics funktion `aes()` indenfor `geom_text()`. Her specificerer vi at parameteren `label` skal være vores p-værdier fra vores datasæt. Parameteren `vjust` bruges til at få de tal lidt ovenpå søjlerne.

```
cor_plot +
  geom_text(aes(label = round(p.value,3)), vjust = -0.5)
```



Lad os transformere de p-værdi kolon til en der viser signifikans i formen af stjerne (det ses meget ofte indenfor præsentationer og publikationer). Her kan man bruge `case_when` - det ligner `ifelse` men kan benyttes i tilfældet hvor der er flere en to muligheder. Man angiver de forskellige muligheder og separerer hvad sker hvis `TRUE` med tilde (for eksempel hvis p-værdien er mindre end 0.001 så få kolonnen `sig` tre stjerne). Her er `p.value < 0.001` skrev i linjen før `p.value < 0.01` så dette tilfælde få prioritet (hvis p-værdien er mindre end 0.001 få `sig` så tre stjerner i stedet for to).

```
eukaryotes_cor_sig <- eukaryotes_cor %>%
  select(-data) %>%
  mutate(sig = case_when(p.value < 0.001 ~ "***",
```

```

    p.value < 0.01 ~ "**",
    p.value < 0.05 ~ "*",
    p.value < 0.1 ~ ".",
    p.value >= 0.1 ~ "n.s"))
eukaryotes_cor_sig %>% select(group,sig) #viser hvordan det ser ud

## # A tibble: 5 x 2
## # Groups:   group [5]
##   group     sig
##   <chr>    <chr>
## 1 Other     ***
## 2 Protists ***
## 3 Plants    ***
## 4 Fungi     ***
## 5 Animals   *

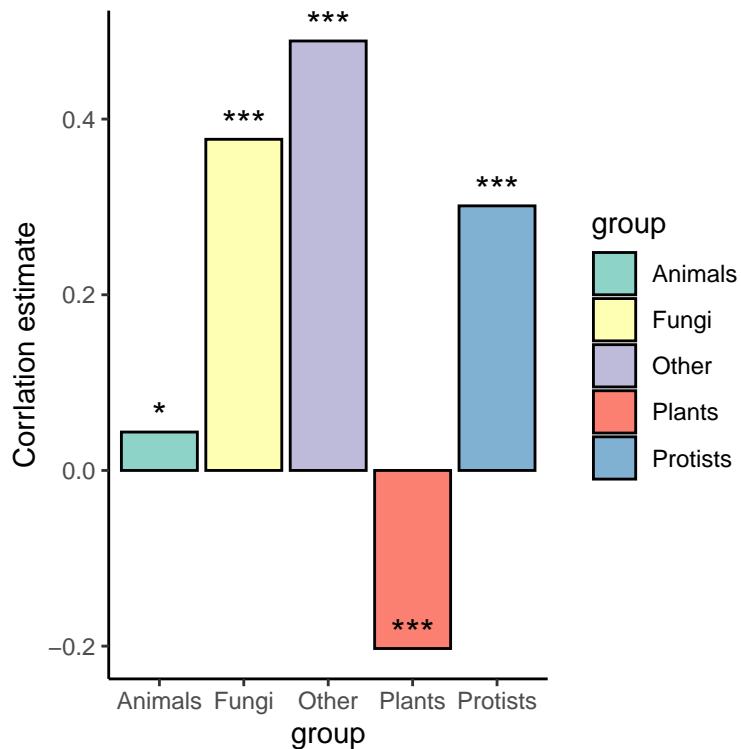
```

Nu at vi har fået en ny kolon der hedder `sig` kan det anvendes indenfor `geom_text` som i forudgående plot.

```

cor_plot <- eukaryotes_cor_sig %>%
  ggplot(aes(x=group,y=estimate,fill=group)) +
  geom_bar(stat="identity",colour="black") +
  scale_fill_brewer(palette = "Set3") +
  ylab("Correlation estimate") +
  theme_classic() +
  geom_text(aes(label = sig),size=5, vjust = -0.1)
cor_plot

```



Mere om adding labels to barplots: <https://r-graphics.org/recipe-bar-graph-labels>

8.5 Problemstillinger

0) Quizzen på Absalon.

Husk at have indlæste følgende:

```
library(tidyverse)
library(broom)
data(mtcars)
data(iris)
```

1) 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. Kan du genkende de statistikker fra `cor.test()` i den resulterende dataramme?

2) Korrelation øvelse

Vælg `Species setosa` fra `iris` (data(`iris`)) og brug `cor.test()` til at beregne korrelationen mellem `Sepal.Width` og `Sepal.Length`.

- Angiv resultaterne i **tidy** form.

3) Nesting øvelse

For datasættet `mtcars` anvende `group_by()` og `nest()` til at få en nested datasæt opdelt efter variablen `cyl`.

- Hvor mange datasæt har du fået indenfor kolonnen `data`?
- Tilføj en ny kolon med `mutate` der hedder `n_rows` som beregner antallet af række i hver af de tre datasæt - brug følgende struktur (først angiv kolonnen som indeholder vores list af datasæt, og så angive funktionen som beregner antallet af række).

```
mtcars %>%
  group_by(cyl) %>%
  nest() %>%
  mutate("n_rows" = map(???, ???)) #erstatte ??? her
```

- Hvad få du indenfor din ny kolonne? Prøve at andre `map` til `map dbl` og se hvad der sker.
- Beholde `map` og tilføj `%>% unnest(n_rows)` i stedet for og se hvad der sker.

4) Multiple korrelation

Vi vil gerne beregne den korrelation mellem variablerne `qsec` og `drat` til hver af de tre datasæt.

- Tilpasser 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(.x$qsec, ???) #erstatte ??? og tilføj glance funktion
```

- Brug `map` i nedenstående med din funktion til at beregne de korrelation til hver af de tre datasæt.
- Huske at `unnest` kolonnen med dine korrelation statistikker bagefter.

```
mtcars %>%
  group_by(cyl) %>%
  nest() %>%
  mutate(cor_stats = ???) #erstatte ??? her og husk unnest
```

- Lave et barplot af `estimate` med den resulterende dataramme

5) Multiple korrelation øvelse

Lave samme analyse som **4)** men

- nest de data efter `gear`
- beregne korrelationen mellem `wt` og `drat`

6) Linear regression øvelse

Brug `data(iris)` og anvende `lm()` til at finde den forventet `Petal.Length` med hensyn til `Petal.Width`.

- Hvad er den intercept og slope af den beregnet linje?
- Tilføj funktionen `glance()` og angive værdier `r.squared` og `p.value`.

```
lm(??? ~ ???, data=???) #erstatte ???
```

7) Lave et scatter plot af Petal.Width på x-aksen og Petal.Length på y-aksen.

- Tilføj linjen `geom_smooth(method="smooth")`
- Ændre linjen til `geom_smooth(method="lm")`
- Ændre linjen til `geom_smooth(method="lm", se=FALSE)`
- Nu specifiser en forskellige farve efter Species. Er der forskellige trends?

Vi vil gerne lave samme analyse for hver af de tre Species.

8) Lineær regression øvelse over multiple datasæt

Vi vil gerne udføre lineær regression med `Petal.Length` og `Petal.Width` som i **6)**, men opdelt efter de tre Species. * Lave en funktion ved at tilpasse følgende kode. Passe på hvad `data` skal være lig med her (når vi skriver en funktion). * Tilføj `glance()`.

```
lm_model_func <- ??? lm(??? ~ ???, data=???) #erstat ???
```

- Anvende `group_by()` og `nest()`
- Anvende `map()` med din funktion indenfor `mutate()` til at tilføj en ny kolon som hedder `lm_stats` til din dataramme.
- Husk at `unnest` kolonnen `lm_statstil` at kunne se statistikker.

9) Tilføj statistik til plottet

Vi vil gerne tilføj de statistikker vi lige har beregnet i **8)** til plottet vi lavet i **7)**.

Kopiere koden fra sektionen “Lave og tilføj labels til dit regression plot” og tilpasse den til din analyse med `iris`.

10) Tilføj statistik til plottet

Vi vil gerne tilføj de statistikker vi lige har beregnet i **4)** til et bar plot af vores korrelation estimates.

Kopiere koden fra sektionen “`case_when()`: adding bar plot text” og tilpasse den til din analyse med `mtcars`.

8.6 Næste uge

Clustering af data ind i forskellige grupper - kmeans/hierarchical clustering (kun undervisning på tirsdag og workshop på fredag).

Chapter 9

Clustering

9.1 Indledning og læringsmålene

9.1.1 Læringsmålene

I skal være i stand til at

- Beskrive hvad k-means clustering går ud på
- Anvende `kmeans` og output resultatet på en `tidy` måde
- Iterate 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 der til formål at dele et datasæt op i forskellige grupper (clusters eller klynges på dansk) af observationer, der mest ligner hinanden. Det øger indsigten i datasættet ved at f.eks. bedre forstår strukturen. De spørgsmål som vi kan prøve at give svar på er bla.:

- Hvor mange forskellige clusters er repræsenteret i mit datasæt?
- Hvilke individuelle observationer tilhører hvilken cluster?

I dette kapitel ser vi hvordan vi kan implementere både k-means clustering og hierarchical clustering i R (indenfor den tidyverse ramme), og bruge dem til at tage beslutninger om de ovenstående spørgsmål.

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

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>

9.2 K-means clustering

```
library(palmerpenguins)
library(tidyverse)
library(broom)
```

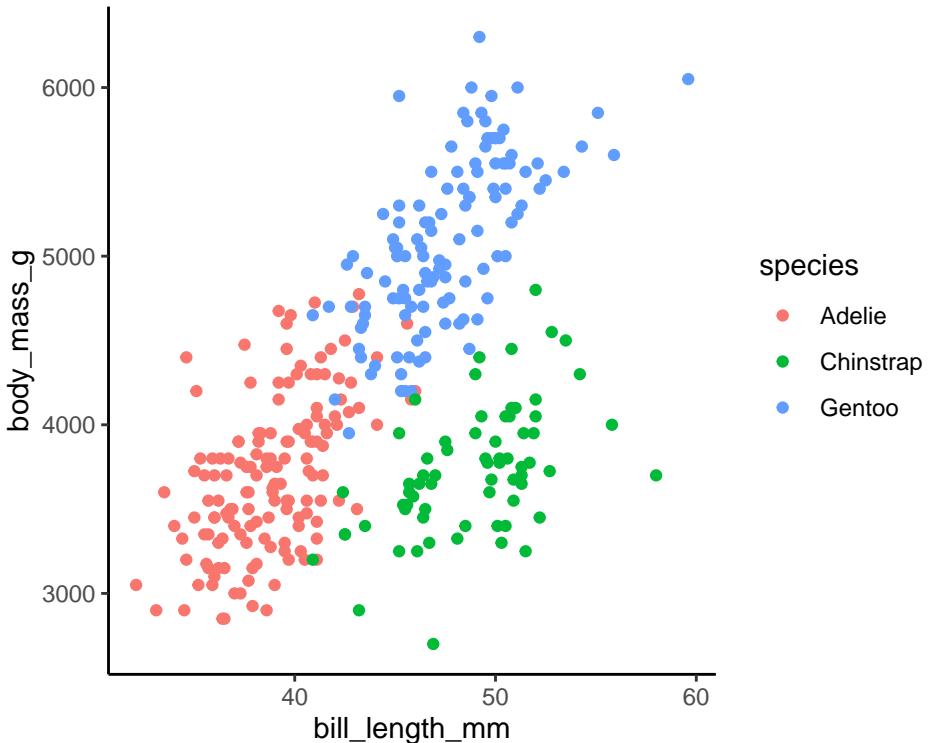
I k-means er alle observationer eller datapunkter tilknyttet til den nærmeste cluster, efter deres nærhed til hinanden. Man specificere i forvejen antallet af clusters som data skal være delt op ind i. Derfor, skal der være nogle undersøgelserne arbejde for at vælge den bedste antal clusters som passer til problemstillingen/ bedste repræsenterer de data.

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 de datasæt, og derefter kan det være nyttigt at sammenligne de clusters vi få med de tre species - hvor gode er de clusters til at skelne i mellem de forskellige species, eller fanger de noget andet struktur i de data (for eksempel kønnet eller øen, de bor)?

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 observationer ved tilfælde og kalde dem for de cluster middelværdierne eller "centroids". Man tilknytter alle observationer til en af de tre clusters (efter de nærmeste af de tre centroids), og så beregner en ny middelværdi/centroid for at hver cluster. Man tilknytter observationer til den nye cluster centroids og så gentager man processen flere gange.

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.

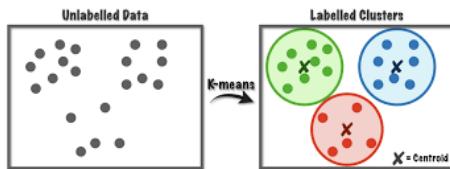


Figure 9.1: source: <https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c>

9.2.2 Run k-means i R

K-means fungerer kun på numeriske data, som vi kan vælge fra datasættet med `select` - man kan specificere `where(is.numeric)` indenfor `select()` for at slippe for at manuelt indtaste de relevante variable navne. Vi bruger også `scale()` på de data her. Det 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 ind 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** - de svarer til de centroids markerede med \mathbf{x} i den ovenst  ende figur - bem  rk at her er de 5-dimensionel da vi brugt 5 variabler til at beregne resultatet.
 - **Clustering vector** - hvilke cluster er hver observation blevet tilknyttet til.
 - **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 n  rmeste centroid).

9.2.3 Tidy up k-means resultaterne med pakken broom

Fra pakken `broom` har vi beskæftiget os med `glance` som vi benyttede til at få enkel-linje baserede summary statistics fra flere modeller sammen i én dataramme, for at facilitere et plot/labels osv. Der er også to andre funktioner vi tager i bruge her. Her er en beskrivelse af de tre.

Funktion	Beskrivelse
glance	single line summary
augment	Connect information from the model to the original dataset
tidy	multi-line summary

For at lave et plot af de clusters kan det 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 kunne 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 3        Adelie  Torgersen      39.1            18.7            181
```

```

##  2 3      Adelie Torgersen    39.5    17.4    186
##  3 3      Adelie Torgersen    40.3     18      195
##  4 3      Adelie Torgersen    36.7    19.3    193
##  5 3      Adelie Torgersen    39.3    20.6    190
##  6 3      Adelie Torgersen    38.9    17.8    181
##  7 3      Adelie Torgersen    39.2    19.6    195
##  8 3      Adelie Torgersen    41.1    17.6    182
##  9 3      Adelie Torgersen    38.6    21.2    191
## 10 3     Adelie Torgersen    34.6    21.1    198
## # ... with 323 more rows, and 3 more variables: body_mass_g <int>, sex <fct>,
## #   year <fct>

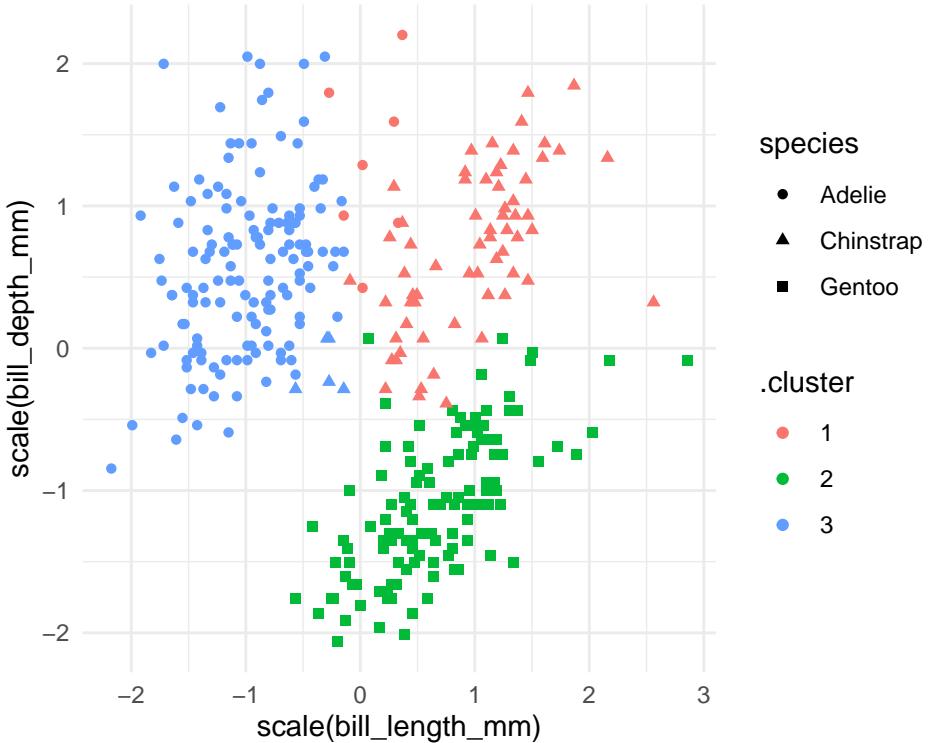
```

Nu lad os benytte vores datasæt som vi har fået med `augment` 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 species. 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 f. k. tælle op hvor mange af tre species vi få 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     7
## 2 1       Chinstrap  63
## 3 2       Gentoo    119
## 4 3       Adelie    139
## 5 3       Chinstrap  5
```

Lad os også kigge på resultatet af tidy. Her har vi fået en pæn dataramme med middelværdierne (centroids) af de tre clusters over de fire variabler som blev brugt i beregningerne.

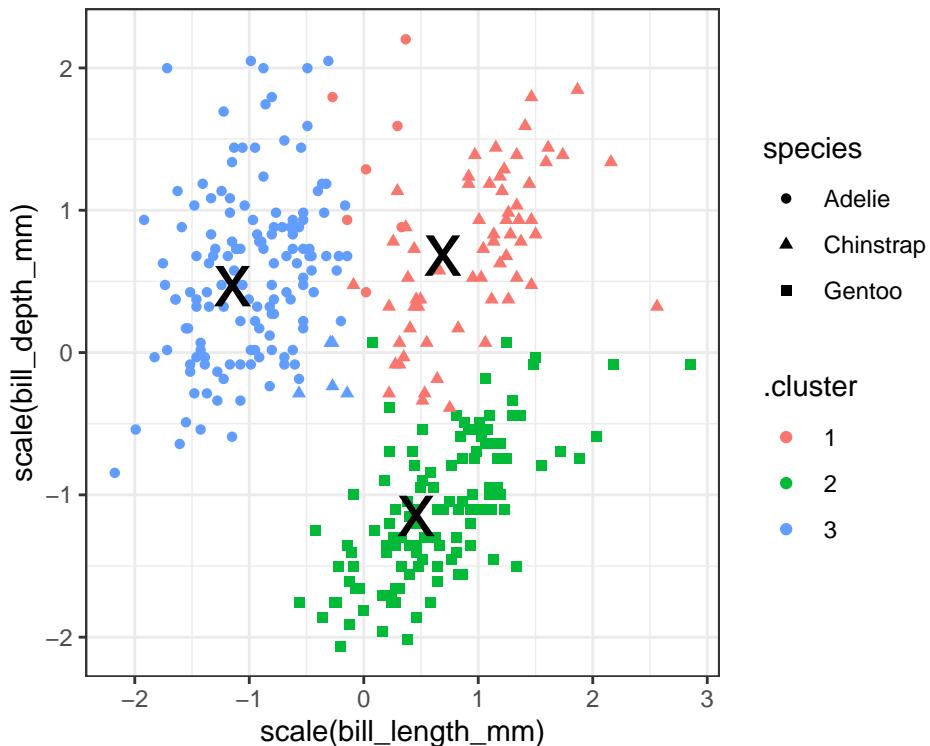
```
kclust_tidy <- tidy(kclust)
kclust_tidy
```

```
## # 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          0.891         0.759        -0.304       -0.469     70    79.0
## 2          0.654        -1.10         1.16        1.10      119   139.
## 3         -0.973         0.541        -0.811      -0.681     144   152.
## # ... with 1 more variable: cluster <fct>
```

Lad os benytte `kclust_tidy` som et datasæt i vores ovenstående plot indenfor en anden `geom_point()` til at tilføje en `x` form i de centre af de tre clusters (bemærk 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),
                 y = scale(bill_depth_mm)) +
  geom_point(aes(color = .cluster, shape = species)) +
  geom_point(data = kclust_tidy,
             size = 10, shape = "x", show.legend = FALSE) +
  theme_bw()
```



Vi kan se at vores clusters fanger ikke de samme gruppe som `species` perfekt - der er forskelligheder. Det kan være at vi også har fanget nogle oplysninger om fk. øen de kommer fra eller kønnet af pingvinerne.

9.3 Hvor mange clusters skal der være?

Vi gættede på 3 clusters i ovenstående analyse (da vi havde oplysninger om de species i forvejen) men det kunne være, at et andet antal clusters passer bedre med de data. Vi kan anvende vores statistikker over de forskellige antal cluster til at tage en beslutning om, hvor mange clusters vi gerne vil beholde i vores færdig clustering resultat. Det er vigtigt at kan finde frem til en hensigtsmæssigt antal clusters - for mange clusters kan resultatet i over-fitting, hvor vi har for mange til at fortolke eller giver mening, og for få kan betyde at vi mangler indsigt i strukturen eller trends i de data.

9.3.1 Få statistikker for antal clusters fra 1 til 9

I nedenstående iterater vi over vectoren 1:9, som vi angive i `kmeans` for at fortæl hvor mange clusters vi gerne vil beregne. For hver af de integraler 1 til 9, benytter vi således `kmeans` med hjælp af funktionen `my_func` (bemærk at vores input data `.x` er antallet af clusters men de datasæt er den samme hver gang). Dernæst benytte vi `tidy`, `glance` og `augment` på vores resultaterne fra `kmeans`:

```
my_func <- ~kmeans(penguins %>% select(where(is.numeric)) %>% scale(),
                     centers = .x)

kclusts <-
  tibble(k = 1:9) %>%
  mutate( kclust = map(k, my_func),
         tidied = map(kclust, tidy),
         glanced = map(kclust, glance),
         augmented = map(kclust, augment, penguins)
  )
```

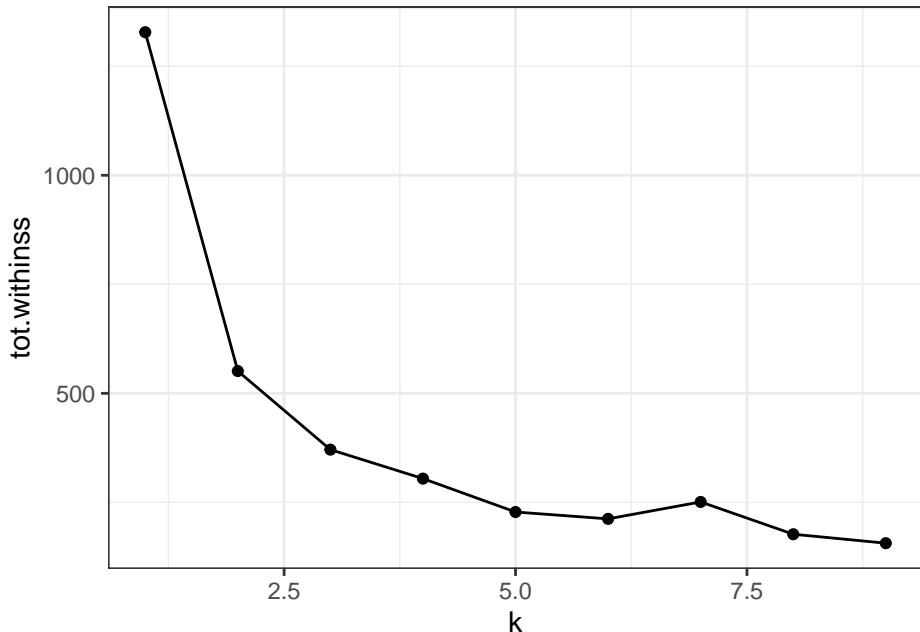
Husk at for at få frem resultaterne i de forskellige former fra `tidy`, `glance` og `augment` er vi nødt til at anvende `unnest`:

```
kclusts_tidy    <- kclusts %>% unnest(cols = c(tidied))
kclusts_augment <- kclusts %>% unnest(cols = c(augmented))
kclusts_glance <- kclusts %>% unnest(cols = c(glanced))
```

9.3.2 Manuelt beslutning med elbow plot.

Vi bruger `tot.withinss` fra vores `glance` output. Det måler hvor meget observationerne indenfor samme cluster ligner hinanden og er den totale afstand af observationerne fra deres nærmeste centroid. Jo flere clusters, jo mindre statistikken, men vi kan se, at efter 2-3 clusters, er der ikke meget gevinst med at bruge flere clusters. Derfor vælger man enten 2-3. Der er ofte kaldes for en ‘elbow’ plot - man vælge de tal på den ‘elbow’, hvor der ikke er meget gevinst med at have flere clusters i de data.

```
kclusts_glance %>%
  ggplot(aes(x = k, y = tot.withinss)) +
  geom_line() +
  geom_point() +
  theme_bw()
```



9.3.3 Automatistke beslutning med pakken NbClust

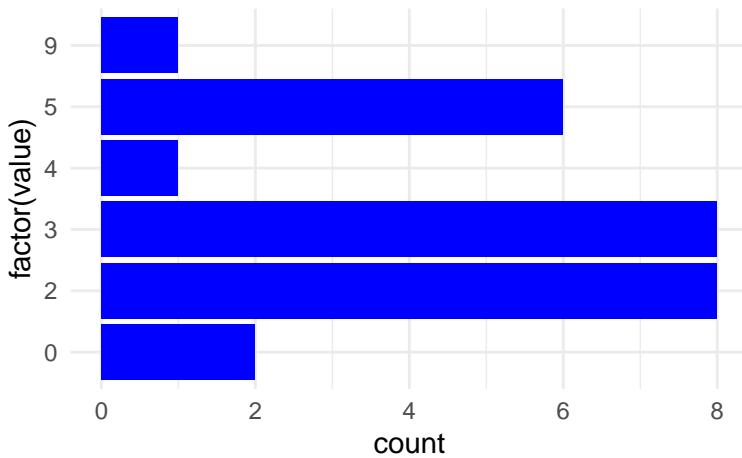
Hvis man synes at de er svært at vælge et bestemt cluster tal fra den elbow plot, kan man også prøve noget mere automatisk. For eksempel pakken NbClust lave 30 forskellige clustering algoritme af de data fra antal clusters = 2 til antal cluster = 9 og for hver af de 30 tage en beslutning om de bedste antal clusters. Man kan således se hvilket antal clusters blev valgt af de mest algoritmer.

```
library(NbClust)
cluster_30_indexes <- NbClust(data = penguins %>% select(where(is.numeric)) %>% scale,
                                distance = "euclidean",
                                min.nc = 2,
                                max.nc = 9,
                                method = "complete")
```

Man kan se her, at enten 2 eller 3 er optimelt, som passer sammen med den elbow plot metode.

```
as_tibble(cluster_30_indexes$Best.nc[1,]) %>%
  ggplot(aes(x=factor(value))) +
```

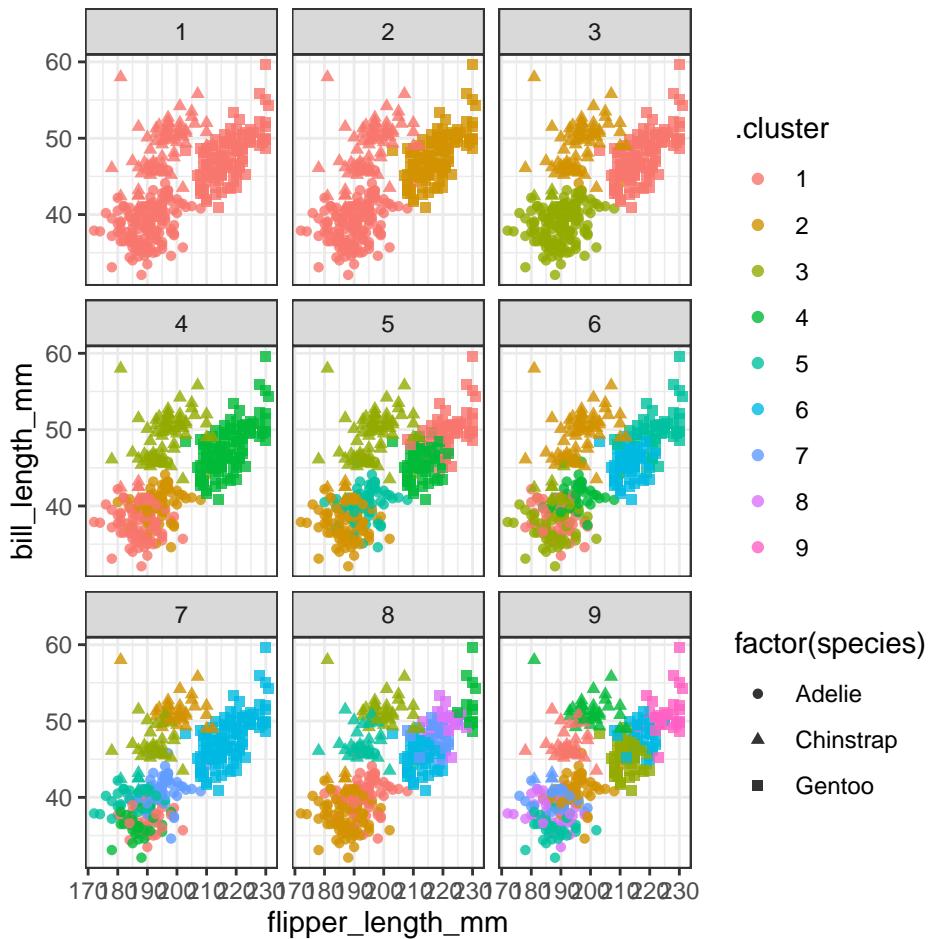
```
geom_bar(stat="count", fill="blue") +
coord_flip() +
theme_minimal()
```



9.3.4 Plot de forskellige antal clusters

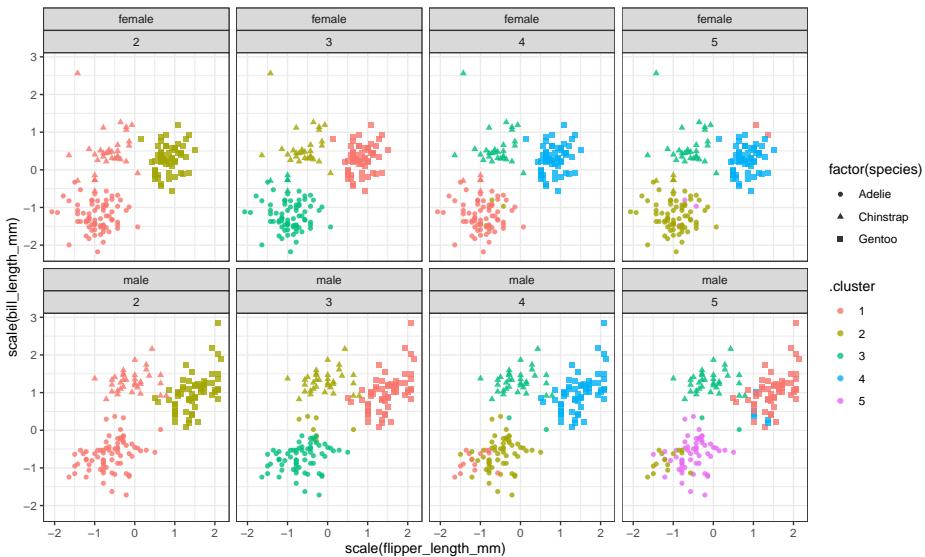
Vi kan også visualisere hvordan de forskellige antal clusters ser ud. Her kan vi bruge vores resultater fra funktionen `augment` til hver af de 9 clusterings. Her har vi har plottet `flipper_length_mm` vs `bill_length_mm`.

```
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()
```



Her introducerer jeg `sex` som en ekstra variable i de plot. Husk at variablen `sex` ikke blev brugt i k-means, men det kan være, at der er nogle aspekter af de fire varierende, som kan fortælle os nogle om kønnet af pingvingerne. 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_wrap(~ sex+k, nrow=2) + theme_bw()
```



9.4 Hierarchical clustering

K-means er en meget populær måde at lave clusters i de data på, men der er mange andre metoder til at lave clusters. Jeg nævne kort hierarchical clustering. Derfor vil jeg også navne her hierarchical clustering. Vi skifter over til `mtcars`, og ligesom i `kmeans` skal vi 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. De 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 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 de data. For eksempel, hvis man gerne vil have 4 clusters, bruger man `k = 4`. Jeg specificerer

`order_clusters_as_data = FALSE` for at få de clusters i de rækkefølger, som passer til plottet (dendrogram) vi lave (man skal have ovenstående pakker installeret for at få den til at fungere).

9.4.2 Lav et pænt plot af dendrogram med ggplot2

Første vi anvende `dendro_data` til at udtrække de dendrogram oplysninger fra de `hclust` resultater.

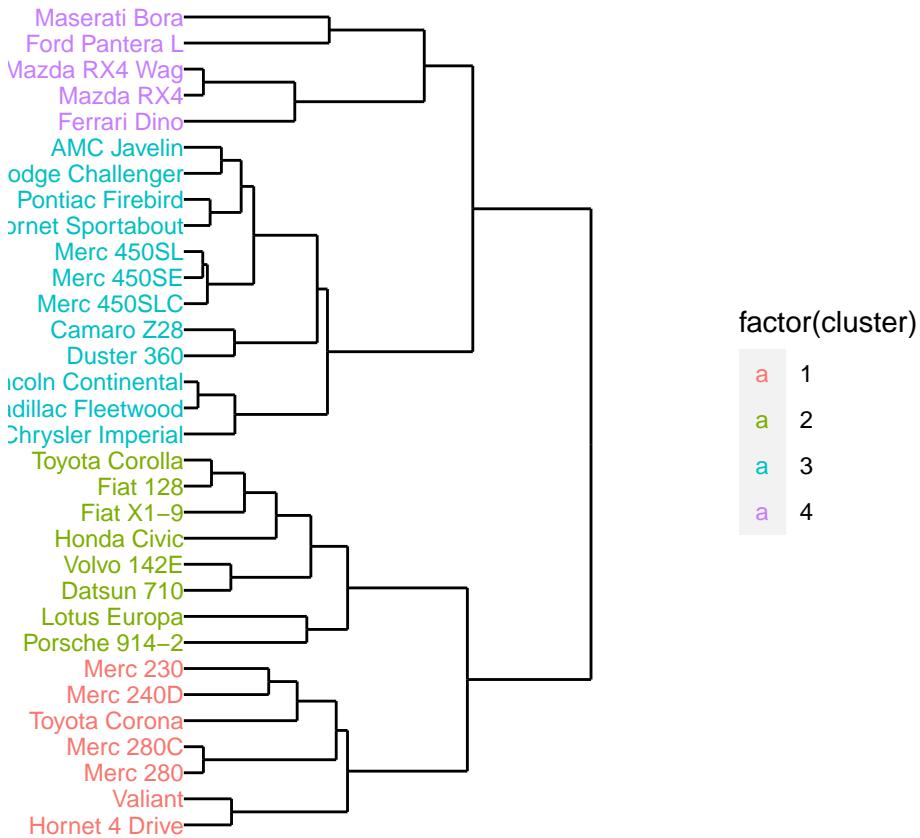
```
library(ggdendro)
dend_data <- dendro_data(mtcars_hc %>% as.dendrogram, type = "rectangle")
```

Vi tilføjer vores clusters som vi beregnede ovenpå (det er derfor vi sikrede 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()
```



9.4.3 Ekstra: afprøve andre metoder

Her giver jeg nogle valgfri ekstra kode til at afprøve de fire metoder - “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:

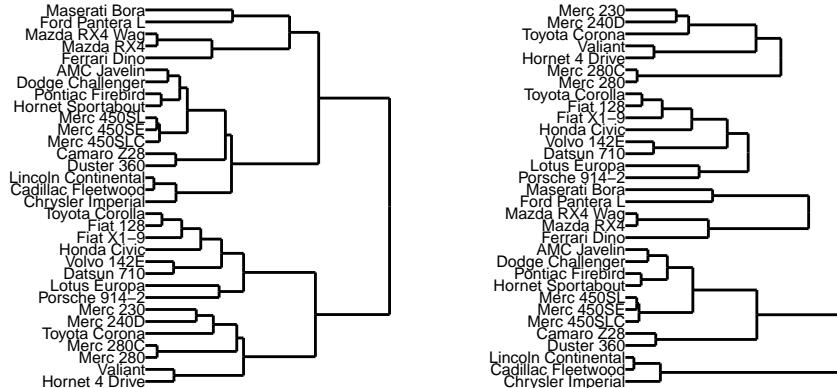
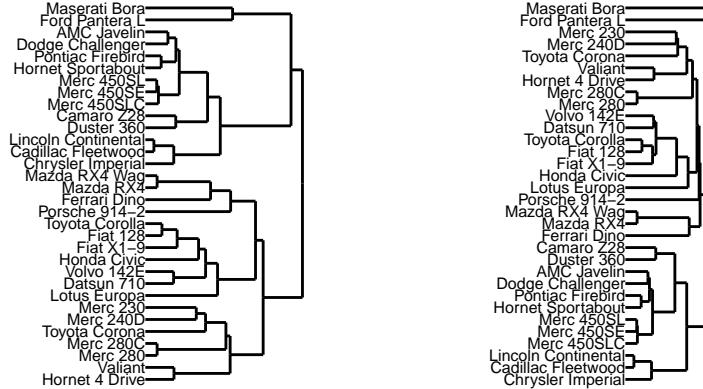
```
# fire metoder:
m <- c( "average", "single", "complete", "ward.D")
```

```

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

0) Quiz - Clustering

- 1) *Funktionen kmeans.* I ovenstående anvendt vi `mtcars` i hierarchical clustering, men lad os se, hvordan det ser ud med k-means. Man kan tilpasse den ovenstående kode for det `penguins` datasæt:

- a) Benyt `kmeans` til at finde 2 clusters i de data
 - husk at vælge kun de numeriske kolonner og scale de data i forvejen
 - gem din clustering som `my_clusters`.
 - hvor mange observationer er der i hver af de to clusters?
- b) Anvend `augment` til at forbinde de datasæt til de clusters fra `my_clusters` (husk at din clustering resultat skrives i første plads i funktionen og så de data i anden plads).
- c) Brug din augmented datasæt til at lave et scatter plot mellem to af de numeriske variabler i de data og give dem farver efter de clusters du har beregnet.
 - hvis du har forbundet det oprindeligt data (der ikke var scaled) i `augment`, husk at scale de data i plottet.
 - prøve også to andre numeriske variabler
- d) Anvend `tidy` til at finde ud af de middelværdier/centroids af hver af de 2 clusters
 - tilpas min kode fra notaterne til at tilføj dem til plottet som ‘x’.

2) *k-means for forskellige k*

Åbn LungCapData fra nedenstående link

```
LungCapData <- read.csv("https://www.dropbox.com/s/ke27fs5d37ks1hm/LungCapData.csv?dl=1")
LungCapData <- as_tibble(LungCapData)
head(LungCapData) #se variabler navne
```

```
## # A tibble: 6 x 6
##   LungCap    Age Height Smoke Gender Caesarean
##   <dbl> <int>  <dbl> <chr> <chr>   <chr>
## 1    6.48     6    62.1 no    male    no
## 2   10.1      18   74.7 yes   female  no
## 3    9.55     16   69.7 no    female  yes
## 4   11.1      14    71   no    male    no
## 5    4.8       5    56.9 no    male    no
## 6   6.22      11   58.7 no    female  no
```

- a) Anvend `kmeans` på LungCapData
 - Vælg de numeriske variabler og scale
 - Angiv `centers = 3`
 - Benytte `augment` til at forbinde resultaterne til LungCapData og gemme resultatet.
- b) Lav et scatter plot mellem LungCap og Age og giv farver efter din beregnet clusters.
- c) Lav et scatter plot mellem LungCap og Height og giv farver efter Smoke.

- Angiv en forskellige form efter Smoke
- d) Man kan også bruge `my_clusters_augment` til at beregne middelværdier med `group_by` og `summarise`.
 - group efter `.cluster` og `smoke` og beregner den gennemsnitlige `LungCap` og `Height` for hver kombination.
 - Angiv datarammen af resultaterne middelværdier indenfor plottet med `geom_point` for at få dem som “X” punkter i plottet.

Lad os undersøge om, 3 var en gode antal clusters til at beregne i de data.

- d) Tilpas koden nedenstående (baserende på den jeg præsenterede i notaterne) til at beregne 9 clusterings for det `LungCapData` datasæt.

```
my_func <- ~kmeans(??? %>% select(???) %>% ???,
                    centers = ???)

kclusts <-
  tibble(k = ???) %>%
  mutate( kclust = map(k, my_func),
         tidied = ???,
         glanced = ???,
         augmented = ???
       )

kclusts_tidy    <- kclusts %>% unnest(?)
kclusts_augment <- ???
kclusts_glance <- ???
```

- e) Lav et “elbow”-plot fra din beregnet clusterings
 - Anvend `kclusts_glance` og plot `tot.withinss` på y-aksen.
- e) Afpøv automatiske method (tilpas min kode fra kursus notaterne)
- f) Visualiser de forskellige antal clusters i et plot
 - Tilpas min kode og benytter `kclusts_augment`
 - Husk at bruge `facet_wrap` til at adskille efter k.

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 til at lave følgende skridt:

- **a)** Benyt funktioner `dist` og så `hclust` på datasættet `msleep_scaled`.
- **b)** Benyt `cuttree` for at finde 5 clusters i de data, og kalde det for `clusters`. Husk at anvende `order_clusters_as_data = FALSE` så at vi har den korrekt rækkefølge for et plot (man skal indlæse pakken `dendextend`)
- **c)** Benyt `dendro_data` til at udtrække de dendrogram fra resultaterne
 - Tilføj `clusters` til `dend_data$labels`
- **d)** Lav et plot af de dendrogram
 - Tilpas koden for `mtcars` eksempel for nuværende data

Chapter 10

Principal component analysis (PCA)

```
library(tidyverse)
library(broom)
```

10.1 Indledning og læringsmålene

10.1.1 Læringsmålene

- Forstå koncepten bag PCA
- Benytte PCA i R og lave et plot af de data i to dimensioner
- Vurdere den relative variance forklarede af de forskellige components
- Anvende PCA til at vurdere variabernes bidrag til principal components

10.1.2 Introduktion til chapter

Principal component analysis er en meget populær og benyttet metode og kan anvendes til bla. at visualisere data med en høj antal dimensioner i et enkelt scatter plot med to dimensioner. Det er meget nyttige for at se de underliggende struktur i de data og indenfor biologi er det meget brugt til at visualisere hvor de forskellige samples sidder relative til hinanden - for eksempel for at se, om de controls og treatment samples fremgå i samme steder i et plot.

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 vores 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
```

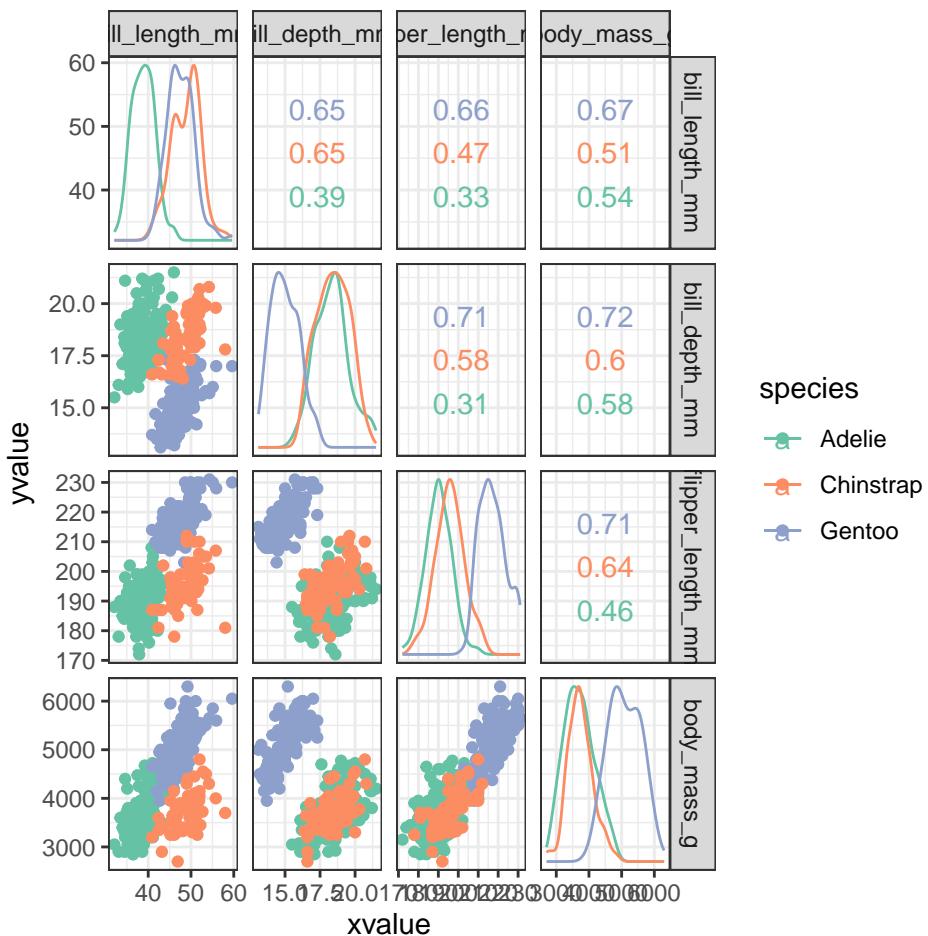
Når man lave et plot af de data for at vise de forskellige clusters, få man et problem - hvilke to variable skal plottes? Man kan plotte hver eneste pair af variabler. For eksempel kan man prøve en pakke der hedder `GGally`, som automatiske kan plotte de forskellige pairs af numeriske variabler og beregner 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()
```



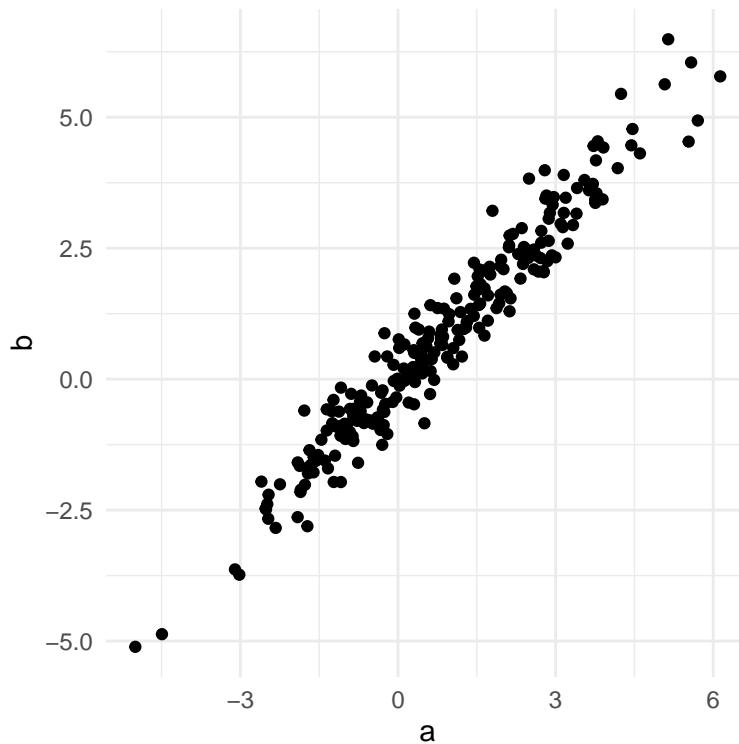
Problemet er, at så snart antallet af dimensioner bliver større end 4, bliver det alt for kompleks og plads krævende.

En løsning til problemmet er at “projekt” de data 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, få man repræsenteret det hele datasæt i stedet for kun to udvalgt variabler. Metoden for at lave disse såkaldte ‘projektion’ kaldes for ‘princial component analysis’.

10.2.1 Simpel eksempel med to dimensioner

Man kan prøve at forstå hvordan PCA fungere ved at kigge på en simpel eksempel med 2 dimensioner:

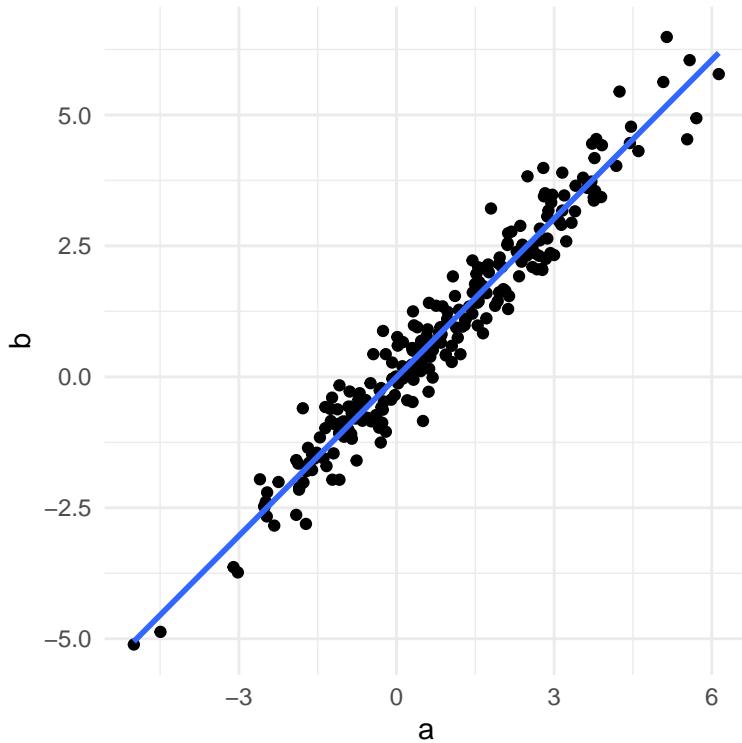
```
#simulere data med en høj korrelation
a <- rnorm(250,1,2)
b <- a + rnorm(250,0,.5)
df <- tibble(a,b)
ggplot(df,aes(a,b)) +
  geom_point() +
  theme_minimal()
```



Vi kan se her, at der er en meget stor korrelation mellem a og b. Selvom de data er plottet i 2 dimensioner kan de næsten forklaries af én bedste rette linje.

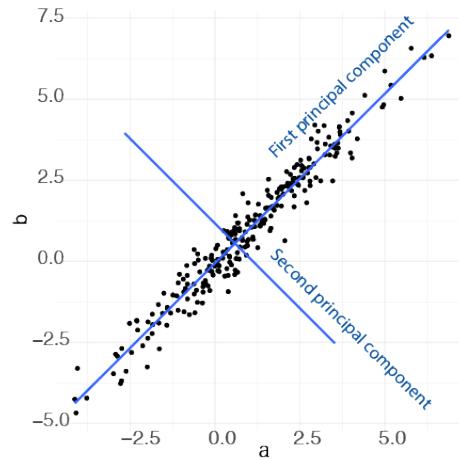
```
df <- tibble(a,b)
ggplot(df,aes(a,b)) +
  geom_point() +
  theme_minimal() +
  geom_smooth(method="lm", se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Med andre ord kan vi næsten forklare de data i kun en dimension - punkternes afstand langt linjen. Når man tager alle punkterne og beskriver dem langt en linje som bedste beskrive variancen i de data, kaldes den linje for den første principal component (PC1). Man kan dernæst beskrive en anden linje som er vinkelret til PC1 som bedste forklarer variancen i de data som ikke var fanget af PC1 - det kaldes for den anden principal component (PC2).

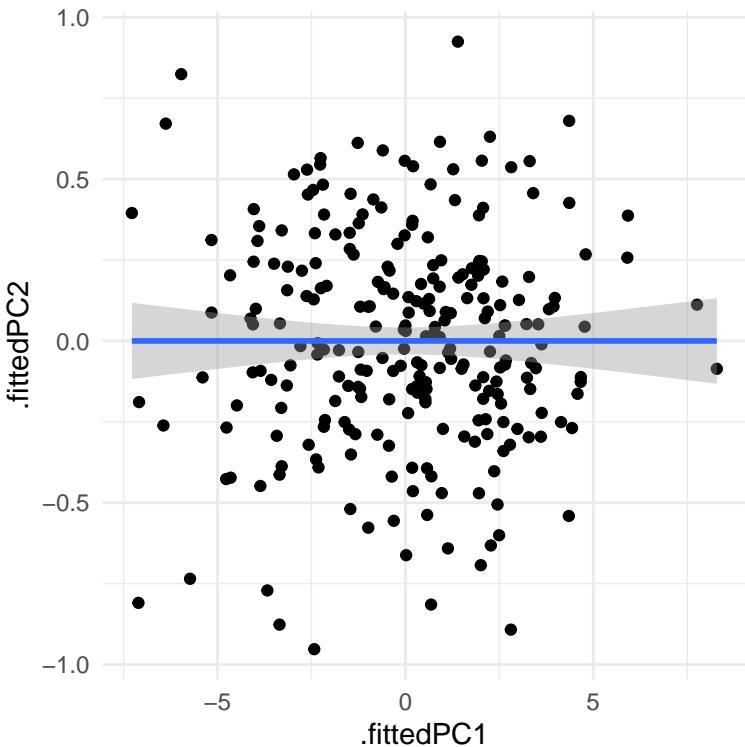
Vi kan se her PC1 og PC2 plottet:



Når vi tager PC1 and PC2 og plotter dem som henholdsvis x-aksen og y-aksen, svarer det til in 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 i de plot bedre en før (og bemærk at de askse skala er blev meget mindre i de ny y-akse, da de data spreder sig meget mindre langt PC2 i forhold til PC1.)

Det her er kun en 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 de data med de første to principal components
- Se for meget af variancen i de data blev forklaret af de forskellige components
- Bruge den rotation matrix til at se, hvordan variabler sidder med hensyn til hinanden

For at få lavet vores plot af de principal komponenter kan vi benytter `augment` ligesom vi gjorde i vores sidste lektion med k-means clustering. Her få vi værdierne til hver af de fire principal components med 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>

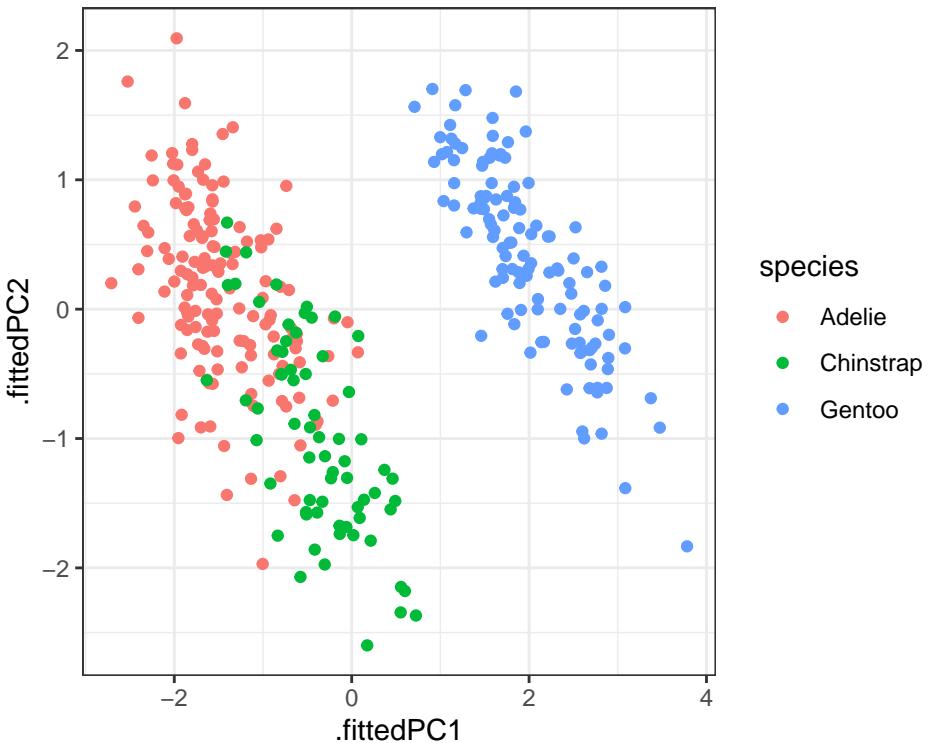
```

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()

```



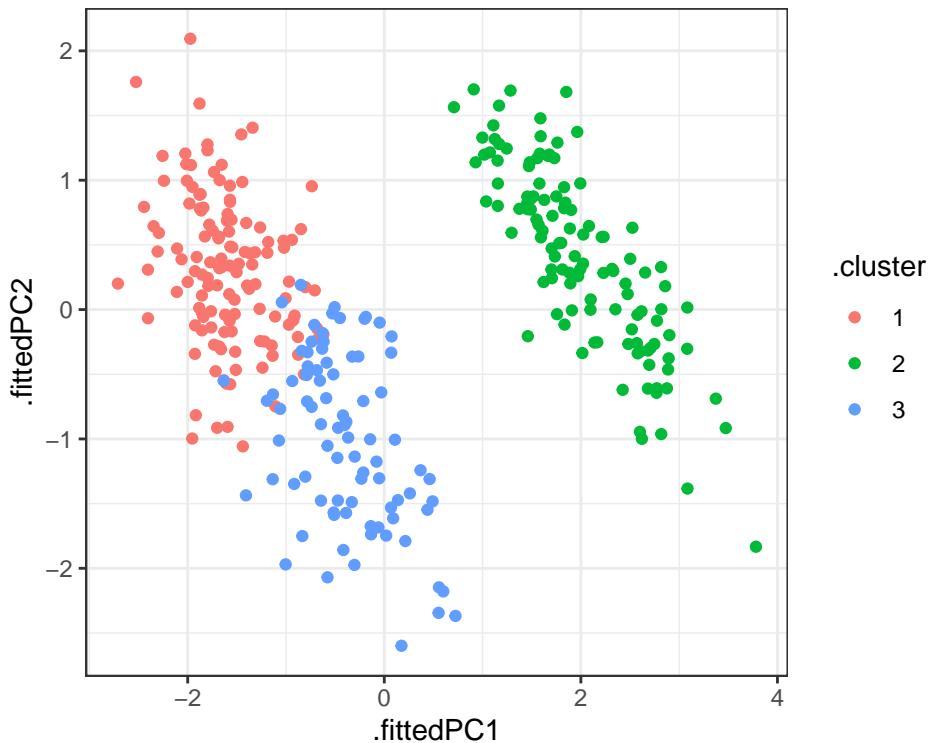
Vi kan også integrere de clusters som vi fik fra funktionen `kmeans` i vores PCA ved at anvende `augment` på resultaterne fra `kmeans` og vores data som allerede

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 kan vi kigge på variancen i de data som er fanget af hver af de forskellige componentes. Vi 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 - altså det beskriver sammenhængen eller korrelationen mellem de forskellige variabler. Man bruger denne covariance matrix til at beregne de såkaldte eigenvalues og

deres tilsvarende eigenvectors.

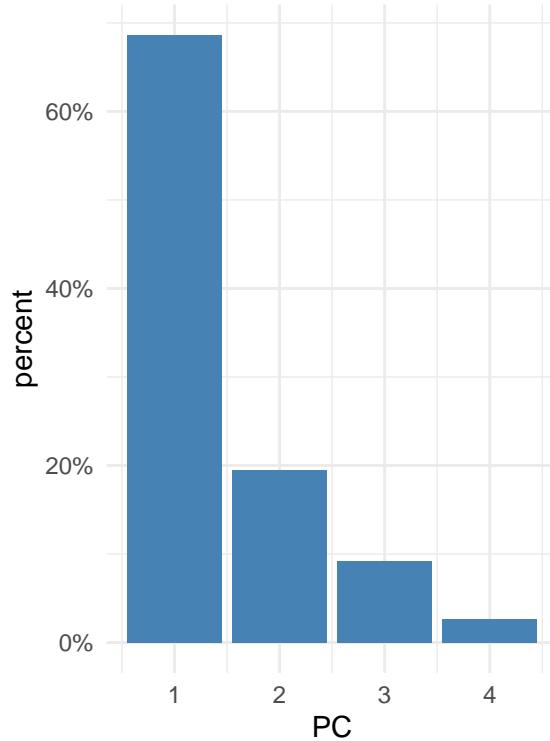
Det er faktisk den største eigenvalue som fortæller os om den første principal component - det fortæller os hvor meget af variancen i de data den første principal component fanger - jo større det er relativ til de andre eigenvalues, jo mere variancen man forklarer med den første principal component. 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 ved de første components (for eksempel den første og den anden tilsammen), betyder det at der er en del redundans i de data, på grund af at mange af de variabler har en tæt sammenhæng med hinanden. På den anden side hvis der er meget lille andel af variancen som er forklaret af de første components, betyder det at det er svært at beskrive de data i mindre dimensioner (fordi der næsten er ingen sammenhæng mellem variablerne) - i dette tilfælde er PCA mindre effektiv.

10.4.1 Rotation matrix to extract variable contributions

De eigenvalues kan anvendes til at find ud af variancen i de data, 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 pivot_wide 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")
```

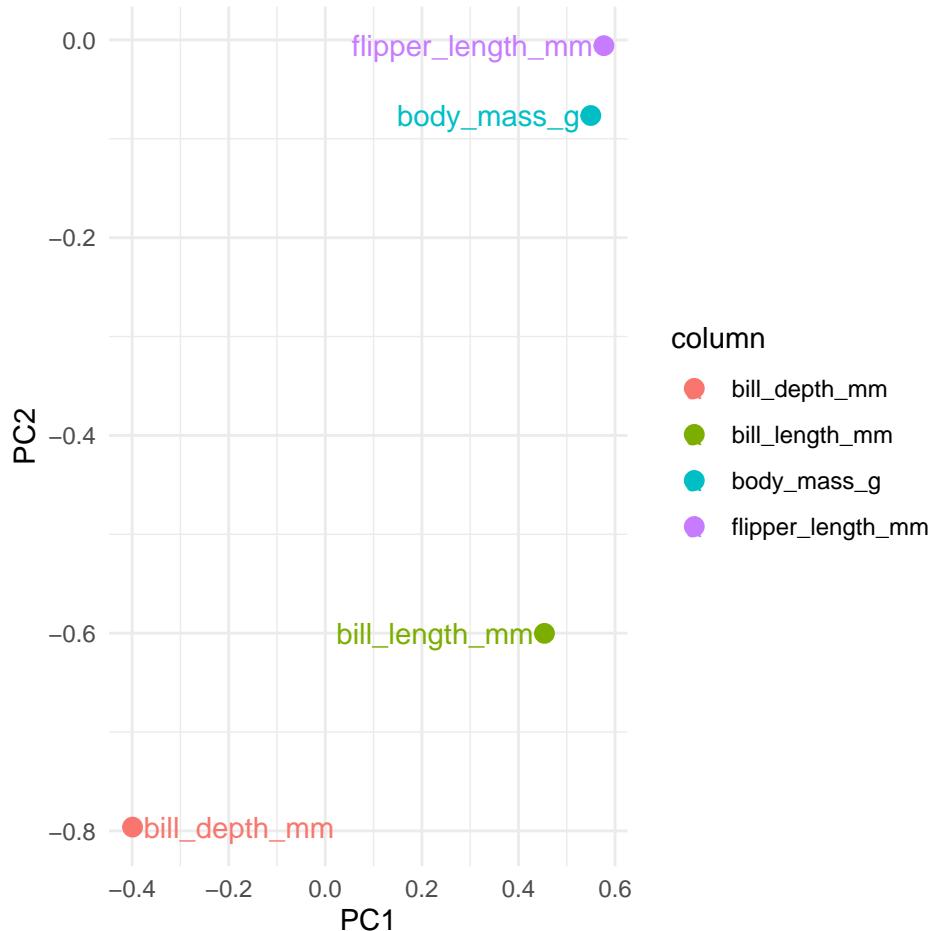
```
pca_fit_rotate
```

```
## # 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 andvende den rotation matrix til at se hvordan de forskellige variabler relatere til hinanden. Variablerne som er tæt på hinanden i plottet ligner hinanden. Vi kan se at flipper_length og body_mass ligner hinhanen ret meget i vores datasæt, mens bill_depth 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()
```



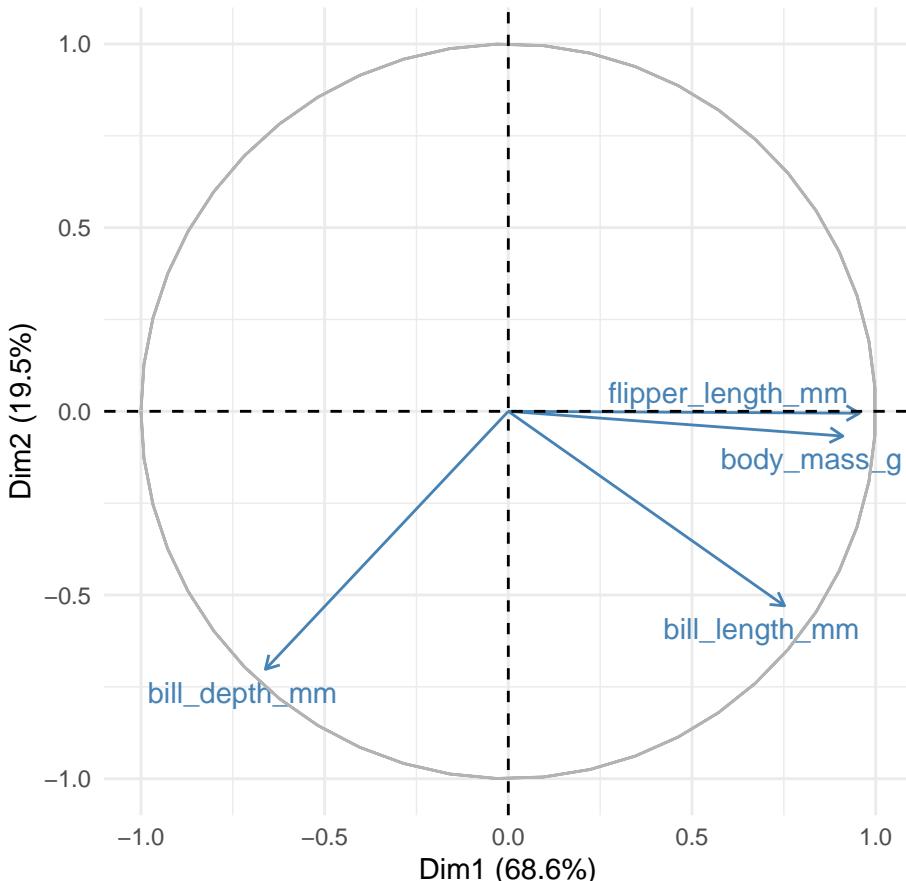
10.4.2 Pakken factoextra

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()
```

Variables – PCA



10.5 Quiz og problemstillinger

- 0) Quiz på Absalon
- 1) Download følgende datasæt ved at køre følgende kode chunk:

```
# the column names of the dataset
names <- c('id_number', 'diagnosis', 'radius_mean',
          'texture_mean', 'perimeter_mean', 'area_mean',
          'smoothness_mean', 'compactness_mean',
          'concavity_mean', 'concave_points_mean',
          'symmetry_mean', 'fractal_dimension_mean',
          'radius_se', 'texture_se', 'perimeter_se',
          'area_se', 'smoothness_se', 'compactness_se',
          'concavity_se', 'concave_points_se',
          'symmetry_se', 'fractal_dimension_se',
```

```

'radius_worst', 'texture_worst',
'perimeter_worst', 'area_worst',
'smoothness_worst', 'compactness_worst',
'concavity_worst', 'concave_points_worst',
'symmetry_worst', 'fractal_dimension_worst')

# get the data from the URL and assign the column names
cancer <- read.csv(url("https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin.csv"))
cancer <- as_tibble(cancer)
cancer <- as.data.frame(cancer %>% select(-id_number))

```

1) 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 `ggscatmat`.
- Give farver efter factor variablen `diagnosis` og vælger “person” som `corMethod`.
- Opfatter du, at der er en del redundans i de data (altså er der høje korrelationer mellem de forskellige variabler)?

2) Benyt funktionen `prcomp` til at beregne en principal component analysis af de data.

- Husk at det skal kun være numeriske variabler og angiv `scale=TRUE` indenfor funktionen.
- Lav et `summary` af resultaterne. Hvad er proportionen af variancen forklaret af den første principal component?
- Hvad er proportionen af variancen forklaaret af den første to principal components tilsammen?

3) *Augment og plot* Anvend `augment` til at tilføje de rådata til ovenstående resultater fra `prcomp`.

- Brug den til at lave et scatter plot af de første to principal components
- Giv farver efter `diagnosis`

4) *Integrere kmeans clustering* Lav et clustering med `kmeans` på de data, med to clusters.

- Augment resultaterne til din rå data (med de `prcomp` resultater allerede tilføjet).
- Lav et plot og give farver efter `.cluster` og forme efter `diagnosis`.
- Sammenligne de clusters med `diagnosis`.

5) *tidy form og variancen* Anvende `tidy(matrix = "eigenvalues")` til at få den bidrage af de forskellige components til variancen i de data.

- Lav et barplot som viser de components på x-aksen og `percent` på y-aksen.

6) *tidy form og rotation matrix* Anvende `tidy(matrix = "rotation")` til at få den rotation matrix.

- Anvend `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

7) *Ekstra øvelse* tilpas kode på andre datasæt

- `iris`
- `mtcars`
- `decathlon2` (indbygget data fra pakken `factoextra`)

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)
```

11.1 Inledning og læringsmålene

11.1.1 Læringsmålene

I skal være i stand til at

- Beskrive randomimisation, 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 chapter er - hvordan kan vi anvende de værktøj 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 vigtigheden af at lave hensigsmæssige visualiseringer.

Forståelse af hvordan batch effekts påvirker en analyse er særligt vigtigt indenfor biologi, når mange store sekvensering projektor involverer data samlede eller sekvenseret over forskellige batches, sekvensering maskiner eller forskellige library forberelses metoder.

11.1.3 Video ressourcer

- Part 1: randomisation, replikation, blocking + confounding
 - Læse 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

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/556581521>

11.2 Baserende princip af eksperimental design

11.2.1 Randomisation and replication

Man laver et **eksperiment** for at få svar på et bestemt spørgsmål, eller hypoteze. Og man designer eksperimentet ud fra princip som gøre det gyldigt at fortolke resultaterne fra analysen af de resulterende data. For et eksperiment at være gyldigt skal det kunne demonstrere hensigtsmæssige **replikation** og **randomisation**.

Randomisation

Resultaterne kan skyldes variabiliteten i en faktor som ikke direkte er interessant i eksperimentet. Derfor bruger man randomisation til at få disse forskellige 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 undgår forskelsbehandling som kan påvirke resultaterne.

Replikation

Når man gentage eksperimentet flere gange - for eksempel ved at have flere patienter i hver treatment gruppe Det tillader os til at kunne beregne variabiliteten i de data, som er nødvendige 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.

I ovenstående figur er der 6 replikater i hver gruppe "treatment" eller "control". I tilfældet "Good randomisation" er genstande som er taget ved tilfældet 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

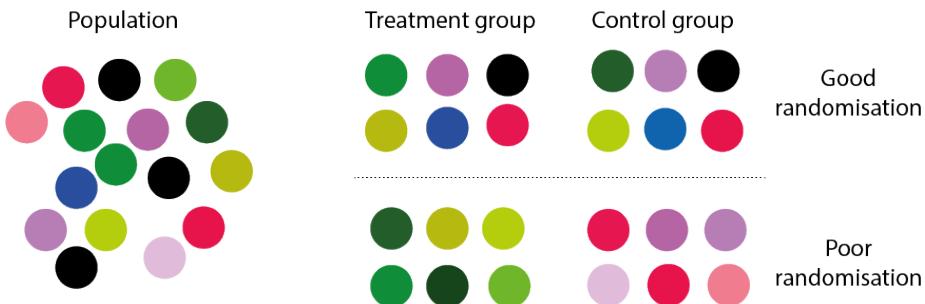
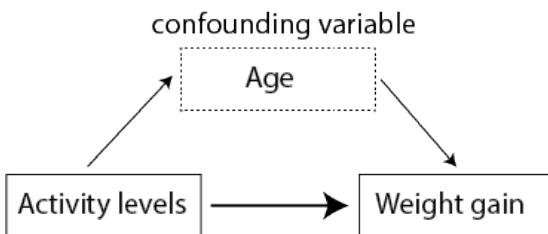


Figure 11.1: randomisation og replikation

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 variable i et eksperiment hvor man prøver at forstå sammenhæng mellem aktivitet niveau og vægtøgning. Der ser umiddelbart ud til at være, at lave aktivitet niveauer (dependent variable) forklarer vægtøgning (indenpendent variable) men man er nødt til at tage ændre variabler ind i betragtning for at sikre, at sammenhængen ikke skyldes noget andet. For eksempel, gruppen med de høj aktivitetsniveau kunne bestå af yngre mennesker end gruppen med de lav aktivitetsniveauer, og deres alder kan påvirke deres vægtøgning (måske på grund af forskelligheder 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 stor 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 fx. 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

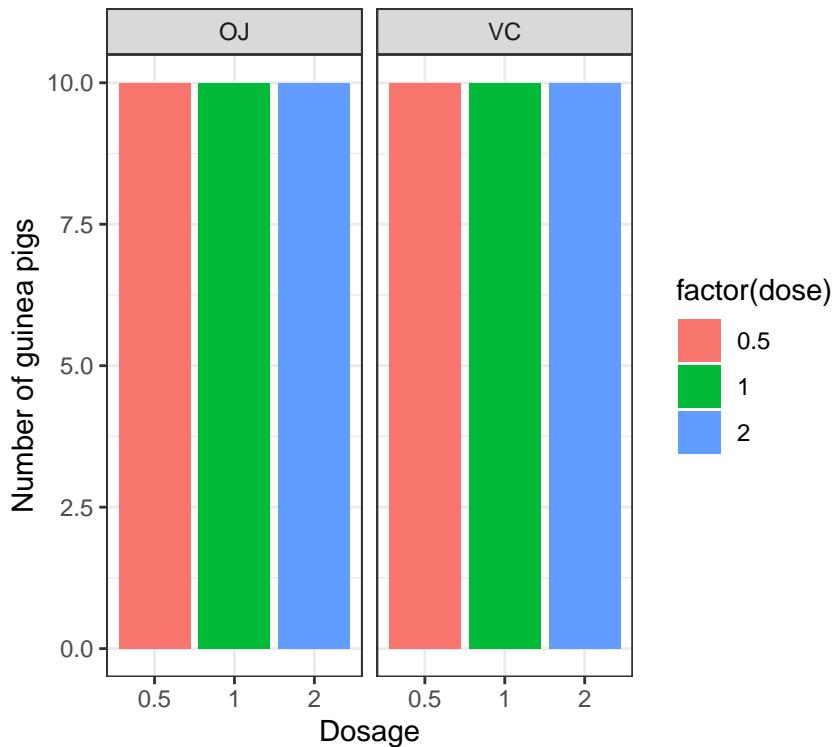
Et god eksempel på en god eksperimental design er datasættet `ToothGrowth`, som er baserende på marsvin - de fik forskellige kosttilskud og doser og så fik målte deres tænder.

```
data(ToothGrowth)
ToothGrowth <- as_tibble(ToothGrowth)
ToothGrowth <- ToothGrowth %>% 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          2  :20
##  Mean   :18.81
##  3rd Qu.:25.27
##  Max.   :33.90
```

Her kan man se, at for hver grupper (efter `supp` og `dose`) er der 10 marsvin - vi har således replikation over de grupper, og hver `supp` (supplement) har hver af de tre 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.

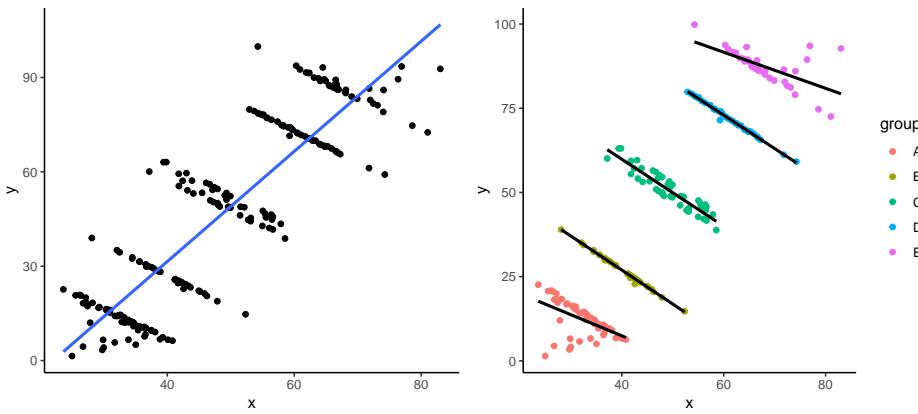
```

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

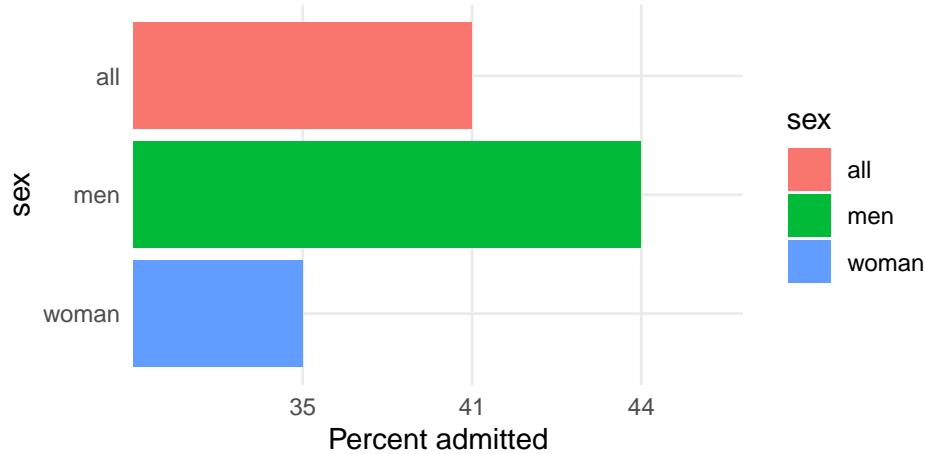
Det meste berømte eksempel af Simpson's Paradox drejer sig om optagelsen til universitetet i Berkeley i 1973. Følgende tabel fra Wikipedia viser statistikker om antallet som ansøgt samt procenttallet som blev optaget overordnet i universitet efter kønnet.

	All		Men		Women	
	Applicants	Admitted	Applicants	Admitted	Applicants	Admitted
Total	12,763	41%	8442	44%	4321	35%

Hvis vi lave et barplot af tallet, kan man se, at der er en højere procenttal af mænd end kvinder som blev optaget på universitet (som resulterede i en retsag mod universitet).

```
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()
```

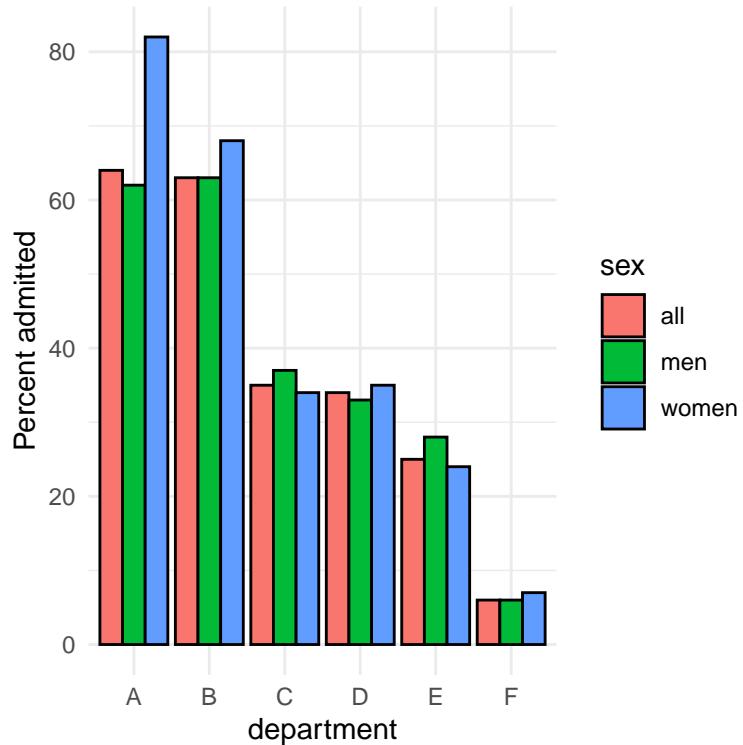


Da man dog kiggede lidt nærmere på de samme tal, men opdelt efter de forskellige afdelinger i universitet, fik man en anderledes billede af situationen. I følgende table har vi optagelses tallene for mænd og kvinder i hver af de forskellige afdelinger (A til F).

```
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 de afdelinger, er der ikke en signifikant forskel mellem mænd og kvinder, og i nogen tilfælde havde kvinder faktisk en større chance for at blive optaget.

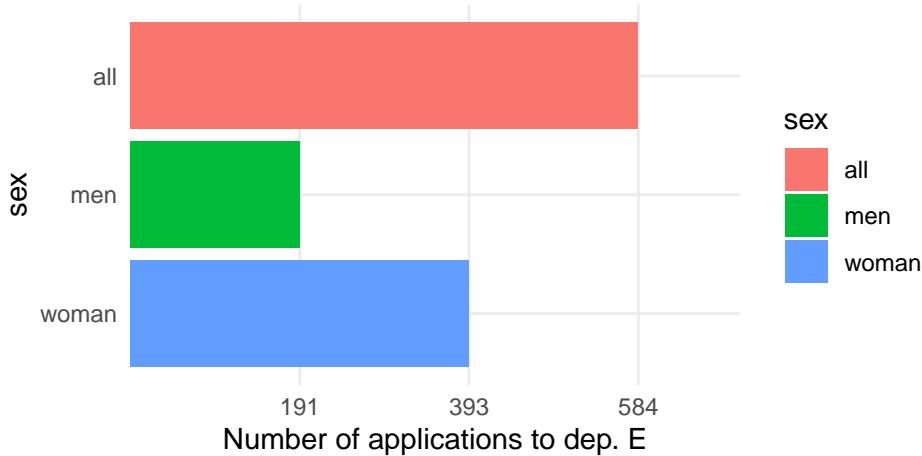
```
admissions_separate %>%
  pivot_longer(-department, names_to="sex", values_to="admitted") %>%
  ggplot(aes(x=department, y=admitted, fill=sex)) +
  ylab("Percent admitted") +
  geom_bar(stat="identity", position = "dodge", colour="black") +
  theme_minimal()
```



Hvad skyldes det her? Det viste sig, at kvinder havde en tendens til, at ansøge indenfor de afdelinger, som var sværreste at komme ind i. For eksempel, kan man se her, at department E har en relativt lav optagelses procent. Den samme afdelinger 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()
```



11.4 Case studies: Anscombe's quartet

(Se også videoressourcer Part 3).

Anscombes quartet (se også https://en.wikipedia.org/wiki/Anscombe%27s_quartet) er en meget nyttige 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> <dbl> <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 
## 10    1      7  4.82
## # ... with 34 more rows
```

Formålet med de data 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 de data 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)

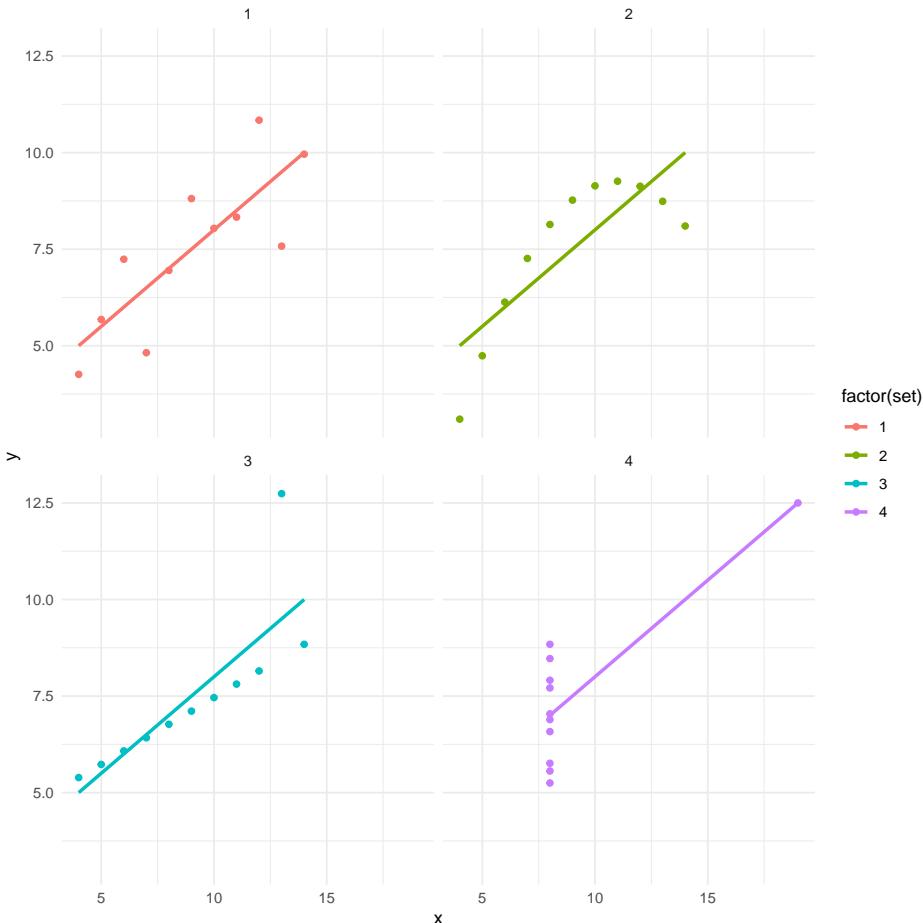
anscombe %>%
```

```
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 den samme
beste rette linje, men de data er slet ikke den samme. Den første datasæt ser
egnet til en lineær regression analyse men vi kan se i nummer to at der ikke
engang er en linær sammenhæng. Og de andre to har en outlier værdi, som
gøre at den bedste rette linje passer ikke særlig godt til de data.

11.5 Using PCA to find batch effects

(Se også videoressourcer Part 4).

Man kan også anvende visualisering til at kigge lidt nærmere på eventuelle batch
effekter i de data. Det er især vigtige i store eksperimenter, hvor dele af de data
var samlet 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 genekspresion sekvensering data lavet i mus.

```
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. Her benytter jeg `map_df` over alle de numeriske kolonner - et strategi er at gemme kolonnen `gene`, fjerne den fra de data, anvende `map_df` for at lave de transformering og så tilføj `gene` som en kolon igen bagefter (OBS: der er mere effektiv måder at gøre det på - for eksempel ved at anvende `map_at`, som I er velkommen til at undersøge).

```
#normalise and filter the data
norm.cts <- norm.cts %>%
  filter(rowSums(norm.cts %>% select(-gene))>50)

genes <- norm.cts %>% pull(gene)

norm.cts <- norm.cts %>%
  select(-gene) %>%
  map_df(~log(.x+1)) %>%
  mutate(gene=genes, .before=1)

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 spredes sig over kolonnerne. Vi har også nogle sample oplysninger - der er to forskellige strains af mus og også forskellige batches.

```
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 kigger på hvor mange af hver stain og batch vi har i de data:

```
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 vil vi bare gerne kigger efter batch effekts og ikke fjerner dem.

For at kigger på de batch effekts kan man via principal component analysis. Husk at når man lave en principal component analysis, få men hvad der kaldes for den rotation matrix - det anvendes til at se hvor de forskellige samples ligger relative til hinanden i de forskellige principal components - samples der ligner hinanden vises på samme sted på plottet.

```
pca_fit <- norm.cts %>%
  select(where(is.numeric)) %>% # retain only numeric columns
  prcomp(scale = TRUE) # do PCA on scaled data
```

For at analysere sammenhænge mellem de forskellige variabler kan man kigger på den rotation matrix 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 datarammer har en kolon som hedder `column` som referer til de sample navne, så jeg forbinde efter `column` her.

```
rot_matrix <- rot_matrix %>%
  left_join(coldata, by = "column")
```

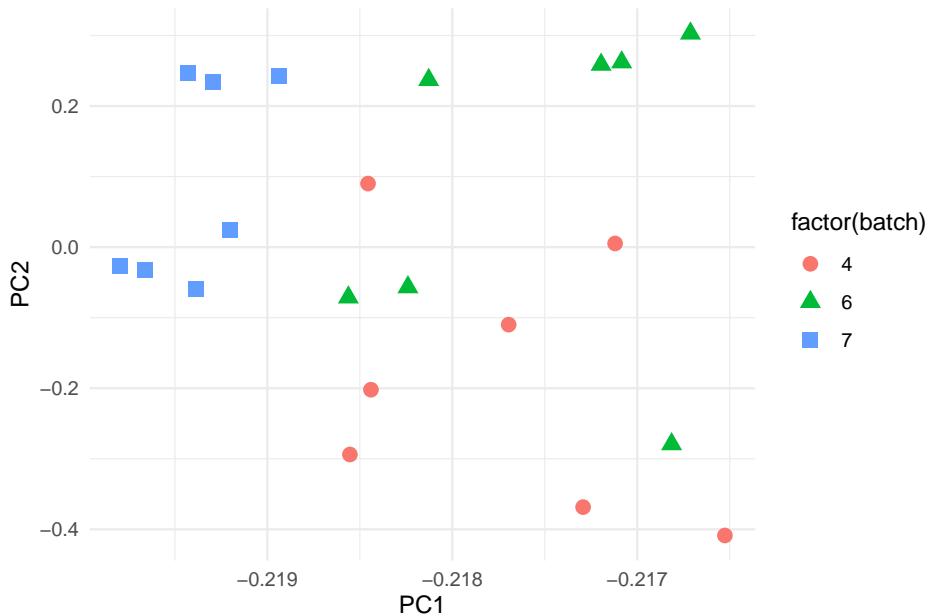
Anvende `pivot_wider` til at få den i wide form:

```
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>        <dbl>    <dbl>    <dbl>    <dbl>
## 1 SRX03~           1 C57BL~     6      -0.217  0.262  0.0200 -0.508
## 2 SRX03~           1 C57BL~     7      -0.219  0.243  0.00586 0.166
## 3 SRX03~           1 C57BL~     6      -0.217  0.303  0.0153 -0.237
## 4 SRX03~           1 C57BL~     7      -0.219  0.246  0.00799 0.178
## 5 SRX03~           1 C57BL~     6      -0.217  0.259  0.0974 -0.113
## 6 SRX03~           1 C57BL~     7      -0.219  0.234  0.00385 0.209
## 7 SRX03~           1 C57BL~     6      -0.218  0.237  0.0303 -0.179
## 8 SRX03~           1 C57BL~     4      -0.217  0.00524 0.463  0.376
## 9 SRX03~           1 C57BL~     4      -0.218  0.0901 0.305  0.0651
## 10 SRX03~          1 C57BL~     4      -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>
```

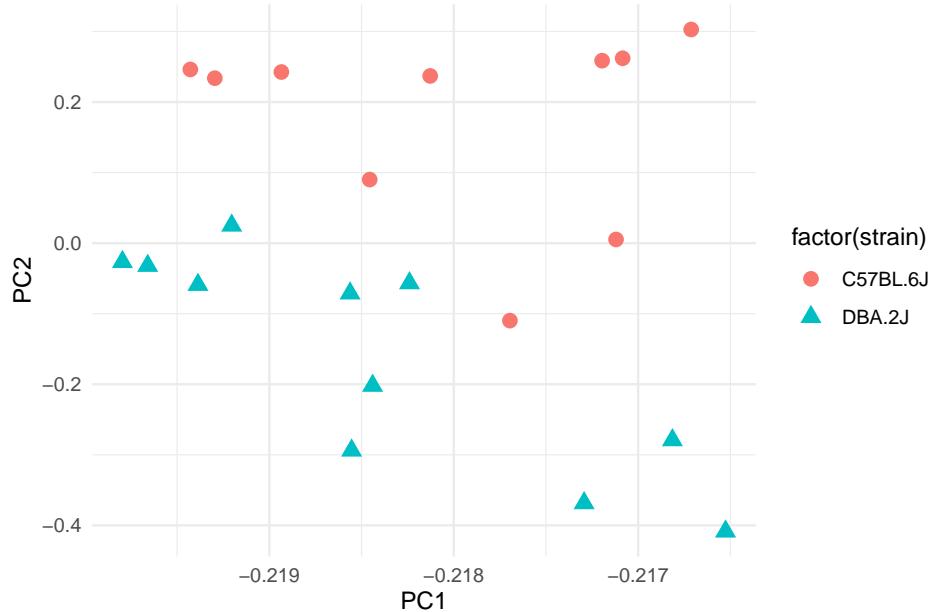
Vi laver et plot af de første to principal components og giv farve og form efter de tre batches i de samples. Vi kan se, at vi har fået alle samples fra batch nummer 2 på samme sted i plottet.

```
rot_matrix_wide %>%
  ggplot(aes(PC1, PC2, shape = factor(batch), colour = factor(batch))) +
  geom_point(size = 3) +
  theme_minimal()
```



Vi kan også give farver efter strain, hvor vi 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()
```

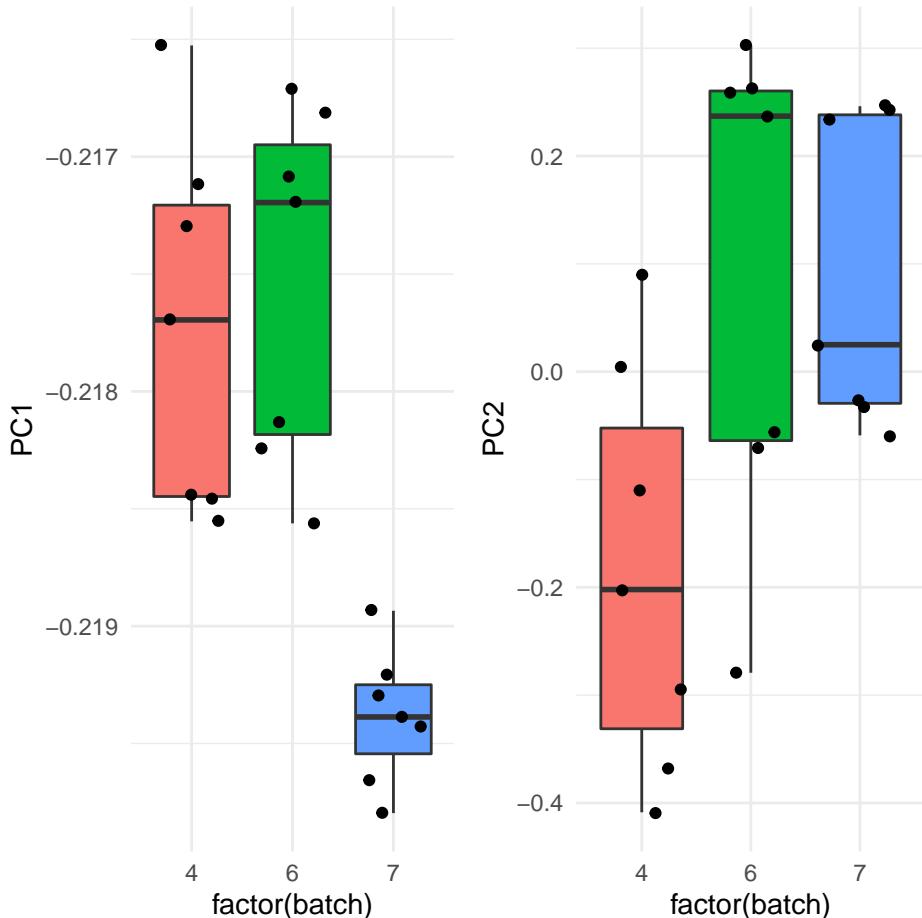


Man kan også se de data på en anden måde ved at lave boxplots for to første to principal componerter opdelt efter batch. Vi får bekræftet vores observation at der er en stærk forskel mellem batch 7 og de andre to batches langt den første principal component, og det er et problem som muligvis skal korrigeres 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)
```



Nogle ide for hvordan man korrigere batch effekts (men ikke dækket i kurset): https://en.wikipedia.org/wiki/Batch_effect#Correction

11.6 Problemstillinger

0) Quiz på Absalon - experimental

1) *Eksperimental design*

Jeg laver et eksperiment hvor patienter få en af tre forskellige kosttilskud. Der er 5 patienter i hver gruppe og jeg vil gerne se, om patienters energi niveau i gennemsnit er forskellige mellem de tre grupper. Alderne af de patienter 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 de alders 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 eksperiment design for at løse problemet?

2) Simpson's paradox Lung Cap data revisited

Indlæse LungCapData:

```
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+")
```

- Lav boxplots med `smoke` på x-aksen og `LungCap` på y-aksen.
– Notere hvilke gruppe den højeste lungkapacitet.
- Lav samme plot men adskilte efter `Age.Group`.
- Beskriv, hvordan det er et eksempel på Simpson's Paradoks.
- Lav et boxplot med `Age` på y-aksen og `Smoke` på x-aksen for at støtte hvorfor man ser Simpson's Paradoks i de data.

3) Anscombes analyse Gentage Anscombes analyse med dinosaurus datasæt:

```
library(datasauRus)
data_dzen <- datasauRus::datasaurus_dozen
```

- Fit en lineær regression model for hver af de datasæt (anvende `group_by`, `nest` og `map` ramme), hvor man finde den forventet y afhængig om x.
- Anvende `tidy` og `glance` på resultaterne.
- Anvende resultaterne fra `tidy` til at kigge på den slope og intercept for de forskellige modeller.
- Anvende også resultaterne fra `glance` til at kigge på den `r.squared` værdi og p-værdien.
- Er de de samme datasæt? Lave et scatter plot adskilte efter de forskellige datasæt.
– Hvordan ser de bedste rette linjer ud på de plots?

4) Vi vil gerne tjekke for batch effects i følgende datasæt. Det er simuleret "single cell" sekvensering count data `cse50` samt med `batches` som angiver en batch for hver af de 500 cells i de datasæt.

```
cse50 <- read.table("https://www.dropbox.com/s/o0wz0jpcsekeg6z/cell_mix_50_counts.txt?")
batches <- read.table("https://www.dropbox.com/s/4t382bfgr46ka5/cell_mix_50_batches.txt?")
```

```
batches <- as_tibble(batches)
cse50 <- as_tibble(cse50)
```

- **a)** Anvend `map_df` til at få transformeret de data til log scale (plus 1 først og tag log bagefter).
- **b)** Lav PCA på det transformeret datasæt.
- **c)** Anvend din PCA resultater til at få den rotation matrix
- **c)** Forbinde oplysningerne fra datarammen `batches` med `left_join` til din rotation matrix.
 - Først tilføj en ny kolon der hedder “column”, som er lig med `names(cse50)`.
- **d)** Anvend `pivot_wider` og lave et plot af den første to principal components, og angive farve efter `batch`.
- **e)** Lav også boxplots for de første to principal components fordelt efter `batch`.
 - Kommentere på eventuelle batch effekts i de data.

Chapter 12

Tidymodels og introduktion til maskin læring

```
library(tidymodels) # metapackage for ML

FALSE -- Attaching packages ----- tidymodels 0.2.0 --
FALSE v dials      0.1.1    v rsample     0.1.1
FALSE v infer       1.0.0    v tune        0.2.0
FALSE v modeldata   0.1.1    v workflows   0.2.6
FALSE v parsnip     0.2.1    v workflowsets 0.2.1
FALSE v recipes     0.2.0    v yardstick   0.0.9

FALSE -- Conflicts ----- tidymodels_conflicts() --
FALSE x gridExtra::combine() masks dplyr::combine()
FALSE x scales::discard()   masks purrr::discard()
FALSE x dplyr::filter()     masks stats::filter()
FALSE x recipes::fixed()   masks stringr::fixed()
FALSE x dplyr::lag()       masks stats::lag()
FALSE x dials::prune()     masks dendextend::prune()
FALSE x yardstick::spec()  masks readr::spec()
FALSE x recipes::step()    masks stats::step()
FALSE * Use suppressPackageStartupMessages() to eliminate package startup messages

library(tidyverse) # metapackage for data manipulation and visulaisation
library(ranger) #install this, random forest
library(palmerpenguins)
library(kernlab)

FALSE
FALSE Vedhæfter pakke: 'kernlab'
```

```

FALSE Det følgende objekt er maskeret fra 'package:scales':
FALSE
FALSE      alpha

FALSE Det følgende objekt er maskeret fra 'package:purrr':
FALSE
FALSE      cross

FALSE Det følgende objekt er maskeret fra 'package:ggplot2':
FALSE
FALSE      alpha
library(ISLR)

```

12.1 Læringsmålene

- introducerer testing/training koncept
- bruger tidymodel workflow til at lave linær regression (parsnip)
- bruge samme workflow til at lave RF (tune parameter)
- tage RF og lave en AUC plot til at vurdere resultatet (yardstick)

12.2 Purpose of chapter

Not to be a comprehensive guide to machine learning

The idea is to introduce a new workflow which you can use for modelling.

The workflow is a framework which is easy to expand as you learn new modelling methodohtolgies.

It may seem like overkill for linear regression but it works particularly well for machine learning set-ups, where we can cut out a lot of steps (which are done internally by the package)

12.3 Testing and training sets

Introduce what testing and training is, and what the purpose is

12.4 Regression in the tidy model framework

Introduce the tidymodel framework with a very simple linear regression

```

#mt cars
set.seed(7834)

# Create a split object

```

```
mtcars_split <- initial_split(penguins %>% drop_na(), prop = 0.75)

# Build training data set
mtcars_training <- mtcars_split %>%
  training()

# Build testing data set
mtcars_test <- mtcars_split %>%
  testing()
```

Make a recipe

```
mtcars_recipe <- recipe(body_mass_g ~ ., data = mtcars_training) %>%
  step_YeoJohnson(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), - all_outcomes())
```

select a model

```
lm_model <- linear_reg() %>%
  set_engine('lm') %>%
  set_mode('regression')
```

Make a workflow

```
mtcars_workflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(mtcars_recipe)
```

Run

```
mtcars_fit <- mtcars_workflow %>%
  last_fit(split = mtcars_split)
```

How good is the fit?

```
mtcars_fit %>% collect_metrics()

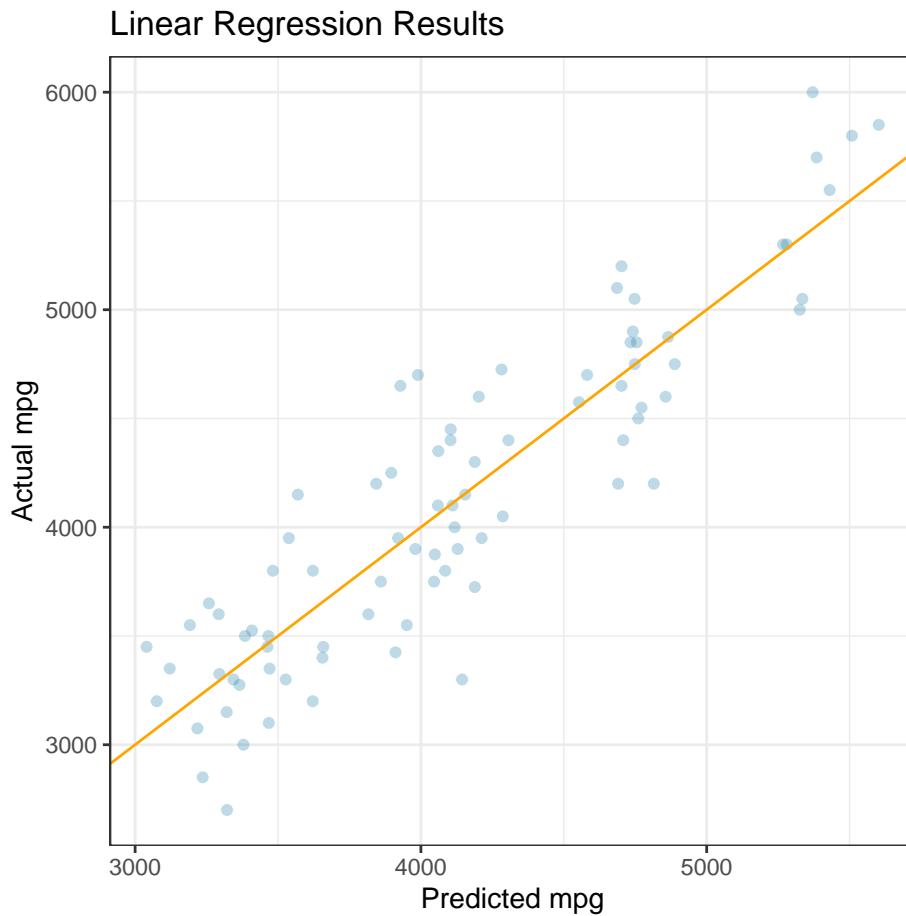
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>       <dbl> <chr>
## 1 rmse    standard     319.  Preprocessor1_Model1
## 2 rsq     standard      0.820 Preprocessor1_Model1

# Obtain test set predictions data frame
mtcars_results <- mtcars_fit %>%
  collect_predictions()

# View results
mtcars_results
```

260CHAPTER 12. TIDYMODELS OG INTRODUKTION TIL MASKIN LÆRING

```
## # A tibble: 84 x 5
##   id          .pred   .row body_mass_g .config
##   <chr>      <dbl> <int>      <int> <chr>
## 1 train/test split 4085.     9        3800 Preprocessor1_Model1
## 2 train/test split 4104.    10        4400 Preprocessor1_Model1
## 3 train/test split 3658.    12        3450 Preprocessor1_Model1
## 4 train/test split 3295.    14        3325 Preprocessor1_Model1
## 5 train/test split 3816.    17        3600 Preprocessor1_Model1
## 6 train/test split 3622.    18        3800 Preprocessor1_Model1
## 7 train/test split 3951.    22        3550 Preprocessor1_Model1
## 8 train/test split 3622.    23        3200 Preprocessor1_Model1
## 9 train/test split 3928.    35        4650 Preprocessor1_Model1
## 10 train/test split 4129.   37        3900 Preprocessor1_Model1
## # ... with 74 more rows
ggplot(data = mtcars_results,
       mapping = aes(x = .pred, y = body_mass_g)) +
  geom_point(color = '#006EA1', alpha = 0.25) +
  geom_abline(intercept = 0, slope = 1, color = 'orange') +
  labs(title = 'Linear Regression Results',
       x = 'Predicted mpg',
       y = 'Actual mpg') + theme_bw()
```



12.5 Classification in the tidy model framework

Try fitting a random forest for classification

Can use the titanic dataset for this - to predict survival

Don't go into details (and give disclaimer), but to give the framework

Make an AUC curve

12.5.1 Titanic survival

Let's start with the famous Titanic dataset. We need to predict if a passenger survived the sinking of the Titanic (1) or not (0). A dataset is provided for training our models (train.csv). Another dataset is provided (test.csv) for which we do not know the answer. We will predict survival for each passenger, submit our answer to Kaggle and see how well we did compared to other folks. The

metric for comparison is the percentage of passengers we correctly predict – aka as accuracy.

First things first, let's load some packages to get us started.

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

titanic <- titanic_clean

titanic <- titanic %>% mutate(survived = as_factor(if_else(Survived == 1, "yes", "no")))
  mutate(survived = relevel(survived, ref = "yes")) %>% # first event is survived
  mutate(class = case_when(Pclass == 1 ~ "first",
    Pclass == 2 ~ "second",
    Pclass == 3 ~ "third"),
    class = as_factor(class),
    gender = factor(Sex),
    port = factor(Embarked),
    age = Age,
    fare = Fare,
    alone = if_else(SibSp + Parch == 0, "yes", "no")) %>%
  select(survived, class, gender, age, alone, port, fare)
```

12.5.2 Fit workflow on titanic data

```
set.seed(7834)

# Create a split object
mtcars_split <- initial_split(titanic, prop = 0.75)

# Build training data set
mtcars_training <- mtcars_split %>%
  training()

# Build testing data set
mtcars_test <- mtcars_split %>%
  testing()

mtcars_recipe <- recipe(survived ~ ., data = mtcars_training) %>%
  step_YeoJohnson(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), - all_outcomes())
```

- Let's select a model

Here are two possible classification models, you can just use these interchangeably/pick one. Notice with the first there is an extra parameter. It is this step that has the largest difference according to which model you want to use.

```
my_model_RF <- rand_forest(trees = 100) %>%
  set_engine("ranger") %>%
  set_mode("classification")

my_model_GLM <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

my_model_SVM <- svm_rbf() %>%
  set_mode("classification") %>%
  set_engine("kernlab")
```

Beware that I just give the models without going into the details - they are just to show how you can classify data with tidymodels. I suggest looking into glm and random forest a little more on your own before using it with your own data (but I don't expect you to know details for this course).

Let's run with both models, we can compare them later:

- Make a workflow

```
mtcars_workflow_RF <- workflow() %>%
  add_model(my_model_RF) %>%
  add_recipe(mtcars_recipe)

mtcars_workflow_GLM <- workflow() %>%
  add_model(my_model_GLM) %>%
  add_recipe(mtcars_recipe)

mtcars_workflow_SVM <- workflow() %>%
  add_model(my_model_SVM) %>%
  add_recipe(mtcars_recipe)
```

Run

```
mtcars_fit_RF <- mtcars_workflow_RF %>%
  last_fit(split = mtcars_split)

mtcars_fit_GLM <- mtcars_workflow_GLM %>%
  last_fit(split = mtcars_split)

mtcars_fit_SVM <- mtcars_workflow_SVM %>%
  last_fit(split = mtcars_split)
```

12.6 How good is the fit?

```

mtcars_fit_RF %>% collect_metrics()

## # A tibble: 2 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>       <dbl> <chr>
## 1 accuracy  binary      0.799 Preprocessor1_Model1
## 2 roc_auc   binary      0.857 Preprocessor1_Model1

mtcars_fit_GLM %>% collect_metrics()

## # A tibble: 2 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>       <dbl> <chr>
## 1 accuracy  binary      0.760 Preprocessor1_Model1
## 2 roc_auc   binary      0.857 Preprocessor1_Model1

mtcars_fit_SVM %>% collect_metrics()

## # A tibble: 2 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>       <dbl> <chr>
## 1 accuracy  binary      0.782 Preprocessor1_Model1
## 2 roc_auc   binary      0.825 Preprocessor1_Model1

# Obtain test set predictions data frame
mtcars_results <- mtcars_fit_RF %>%
  collect_predictions()

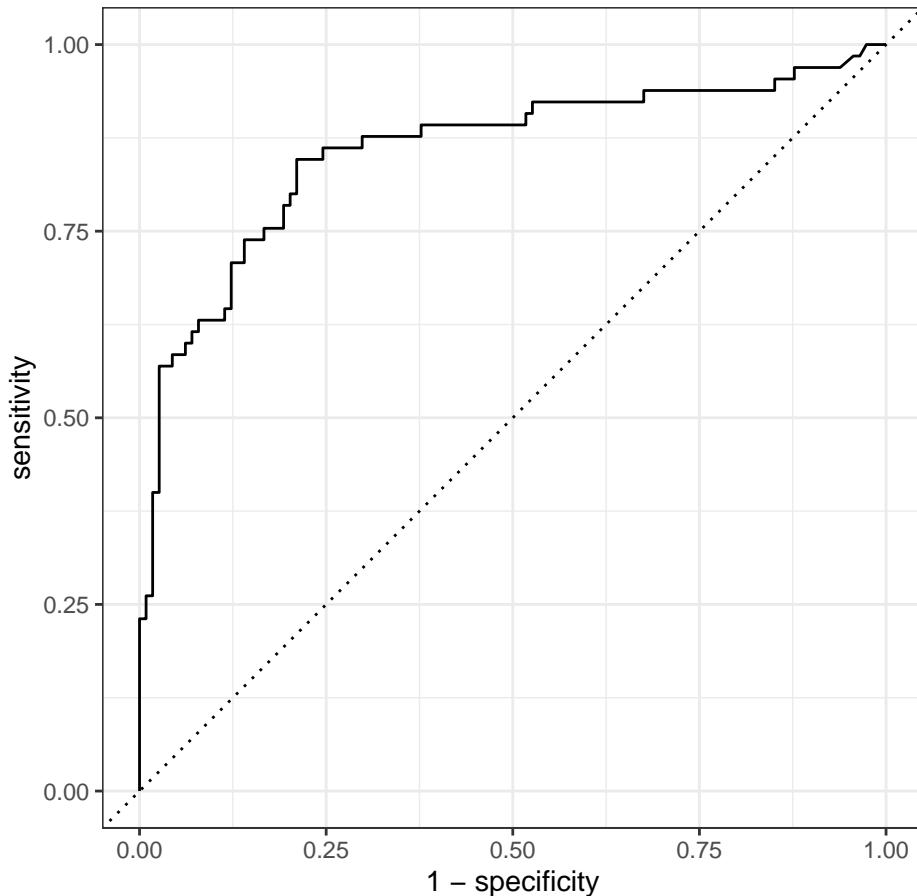
# View results
mtcars_results

## # A tibble: 179 x 7
##   id      .pred_yes .pred_no  .row .pred_class survived .config
##   <chr>     <dbl>    <dbl> <int> <fct>    <fct>   <chr>
## 1 train/test split  0.700    0.300    11 yes      yes    Preprocessor1-
## 2 train/test split  0.0756   0.924    19 no       yes    Preprocessor1-
## 3 train/test split  0.253    0.747    23 no       yes    Preprocessor1-
## 4 train/test split  0.510    0.490    24 yes      no     Preprocessor1-
## 5 train/test split  0.485    0.515    25 no       no     Preprocessor1-
## 6 train/test split  0.613    0.387    27 yes      no     Preprocessor1-
## 7 train/test split  0.134    0.866    29 no       no     Preprocessor1-
## 8 train/test split  0.124    0.876    38 no       no     Preprocessor1-
## 9 train/test split  0.936    0.0638   40 yes      yes    Preprocessor1-
## 10 train/test split 0.848    0.152    42 yes      yes    Preprocessor1-
## # ... with 169 more rows

```

```
dagaa.roc = mtcars_results %>%
  yardstick::roc_curve(survived,
                        .pred_yes) %>% autoplot()

dagaa.roc
```



```
#dagaa.roc %>% ggplot(aes(x=1-specificity,y=sensitivity)) + geom_point() + geom_abline(slope=1,intercept=0)
```

12.7 Cross validation

```
#cross_val_tbl <- vfold_cv(train_tbl, v = 10)
```

12.7.1 Khan data (maybe for workshop)

```

Khan_train <- bind_cols(
  y = factor(Khan$ytrain),
  as_tibble(Khan$xtrain)
)

## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
## Using compatibility `name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated

Khan_test <- bind_cols(
  y = factor(Khan$ytest),
  as_tibble(Khan$xtest)
)

mtcars_recipe <- recipe(y ~ ., data = Khan_train) %>%
  step_YeoJohnson(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), - all_outcomes())

iris_ranger <- rand_forest(trees = 100) %>%
  set_engine("ranger") %>%
  set_mode("classification")

mtcars_workflow <- workflow() %>%
  add_model(iris_ranger) %>%
  add_recipe(mtcars_recipe)

mtcars_fit <- iris_ranger %>% fit(y ~ ., data=Khan_train)

mtcars_fit %>% augment(Khan_test) %>% conf_mat(truth = y, estimate = .pred_class)

##          Truth
## Prediction 1 2 3 4
##           1 3 0 0 0
##           2 0 6 0 0
##           3 0 0 5 0
##           4 0 0 1 5

mtcars_fit %>% augment(Khan_train) %>% conf_mat(truth = y, estimate = .pred_class)

##          Truth
## Prediction 1 2 3 4
##           1 8 0 0 0
##           2 0 23 0 0
##           3 0 0 12 0

```

```
##          4 0 0 0 20
```


Chapter 13

Presæntering af datasæt over for andre



13.1 Introduktion til chapter og læringsmålene

I dette chapter besæftiger vi os med at lave interaktiv visualisering, som vi måske kan vise overfor andre, f.eks. som en del af en præsentation. Til sidste vil der være general råd til at lave plots f.eks. i en powerpoint præsentation.

13.1.1 Læringsmålene

- Interaktiv plotter med pakken `plot_ly`.
- R pakken `Shiny` - lave en simpel app og gør den interaktiv
- Bygger op på appen - tilføj forskellige inputs, plot typer og panels.
- Generelle råd om præsentering af data gennem visualiseringer

Husk evaluering!

13.2 Video ressourcer

- Video 1 - Introduktion til Shiny

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/559363250>

- Video 2 - Introduktion til Shiny - interaktiv plots

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/559558820>

- Video 3 - Introduktion til Shiny - ui layout

Link her hvis det ikke virker nedenunder: <https://player.vimeo.com/video/559562153>

13.3 Interactiv plots

Der er en nyttig pakke som hedder `plot_ly` som man kan anvende til at lave interaktiv plotte. Kig for eksempel på følgende plot og afprøve de forskellige interaktiv muligheder.

```
library(plotly)
data(diamonds)
plot_ly(diamonds,
        x = ~cut)
```

Her er et scatter plot lavet med `plot_ly`:

```
diamonds %>% slice_sample(n = 1000) %>%
  plot_ly(x = ~carat,
          y = ~price) %>%
  add_markers(color = ~color)
```

Bemærk at vi har specificeret `slice_sample` her, som vælger 1000 tilfældige rækker fra de data. Det er fordi `diamonds` er en meget stor datasæt og hvis man forsøger at plotte alle observationer på en gang med `plot_ly` kan det være at den crasher eller kører for langsom.

Der er faktisk en nyttig funktion indenfor `plot_ly` som hedder `ggplotly`- man kan tage et plot som var lavet i `ggplot2` og så anvende `ggplotly` til at gøre den interaktiv. Her er samme plot:

```
my_plot <- diamonds %>% slice_sample(n = 1000) %>%
  ggplot(aes(x=carat,y=price,colour=color)) +
  geom_point() +
  theme_minimal()

ggplotly(my_plot)
```

13.4 Shiny

Man kan anvende Shiny til at lave en interaktiv program. Det er nemt at få din første app op at køre, så vi lave en simple app hvor vi lave et plot af noget af

de data som vi har arbejdet med de sidste uger.

13.4.1 Eksempler som viser Shiny

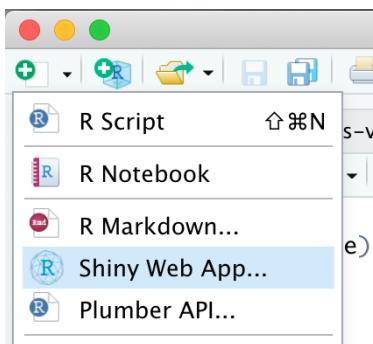
Indenfor pakken **shiny** er der nogle eksempler som viser nogle forskellige Shiny apps. Prøve at køre nogle af følgende kode linjer.

```
library(shiny)
runExample("01_hello")      # a histogram
runExample("02_text")        # tables and data frames
runExample("03_reactivity")  # a reactive expression
runExample("04_mpg")         # global variables
runExample("05_sliders")     # slider bars
runExample("06_tabssets")    # tabbed panels
runExample("07_widgets")     # help text and submit buttons
runExample("08_html")        # Shiny app built from HTML
runExample("09_upload")      # file upload wizard
runExample("10_download")    # file download wizard
runExample("11_timer")       # an automated timer
```

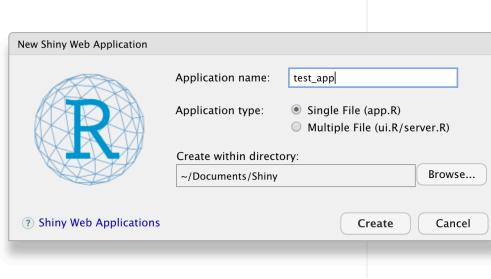
I eksempel 1 er der en histogram som viser nogle waiting times, med en slider hvor man kan specificere antallet af bins i det histogram. Man kan således ændre nogle parametre, og så plottet bliver ændret automatiske. Eksempel 2 har en table hvor man kan specificere hvor mange række man gerne vil have, og så kan man også vælger imellem forskellige datasæt.

Disse eksempler har fordelen af, at man bare kan kopiere koden og så lave egen app og ændre den efter sin egne data.

13.4.2 Create new app



- Man vælger en mappe, og "Single File". Indenfor mappen skal ligge en skript der hedder "app.R" (det er vigtigt at man ikke ændre navnet af "app.R").



- Man kan trykke på "Run App" indenfor R Studio for at få den at køre

```

app.R x
Run App
1 |#
2 # This is a Shiny web application. You can run the application by clicking
3 # the 'Run App' button above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 #   http://shiny.rstudio.com/
8 #
9 #
10 library(shiny)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins", ""))
22   )
23 )
24
25 shinyApp(ui = ui, server = server)
1:1 (Top Level) R Script

```

13.4.3 Struktur af en Shiny App

Her er den grundlæggende struktur af en Shiny app:

```

library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)

```

Indenfor "app.R" er der tre komponenter som kan være opmærksom på. Vi ændrer kun to af de tre komponenter.

Del	Beskrivelse
ui <- fluidPage()	"user interface" object: det fortæller Shiny hvordan appen ser ud.

Dele	Beskrivelse
<code>server <- function(input,output){}</code>	her skrives R kode, fk til et plot, som skal være en del af den R objekt som hedder <code>output</code> . <code>input</code> er for eksempel antallet af bins som man specificerer med en slider som i det histogram eksempel.
<code>shinyApp(ui = ui, server = server)</code>	er altid den sidste linje - den ændre vi ikke. Den få appen til at køre.

13.4.4 Minimal example (men ikke interaktiv)

Jeg starter med at lave et meget simpelt “minimal” eksempel af hvordan man kan lave et program med **Shiny**:

- **ui komponent:** Vi vil gerne vise et plot som hedder “my_plot” - så skriver jeg `plotOutput("my_plot")`

```
ui <- fluidPage(  
  plotOutput("my_plot") #fortælle, at vi vil fremvise "my_plot".  
)
```

- **server komponent:** angiver vi koden til `my_plot`.
 - Plottet skal være en del af vores `output`, så vi skriver `output$my_plot <-`.
 - `renderPlot({ #plot kode })` fortæller den at det vi lave er et plot og vi skriver koden indenfor.

```
server <- function(input, output) {  
  #definere "my_plot"  
  output$my_plot <- renderPlot({  
    #Skriv plot kode her  
  })  
}
```

Her er appen med ovenstående kode sæt ind: man godt kan køre den men når vi ikke har nogle data eller plot kode er den meget kedelig.

```
library(shiny)  
library(tidyverse)  
  
ui <- fluidPage(  
  plotOutput("my_plot") #fortælle, at vi vil fremvise "my_plot".  
)  
  
server <- function(input, output) {  
  #definere "my_plot"
```

```

    output$my_plot <- renderPlot({
      #Skriv plot kode her
    })
  }
  # Run the application
  shinyApp(ui = ui, server = server)
}

```

Tilføje databaseættet `iris` og lave et plot

Lad os tilføje nogle data til vores plot i formen af `iris`, og skriv nogle kode til et scatter plot:

```

library(shiny)
library(tidyverse)
data(iris)

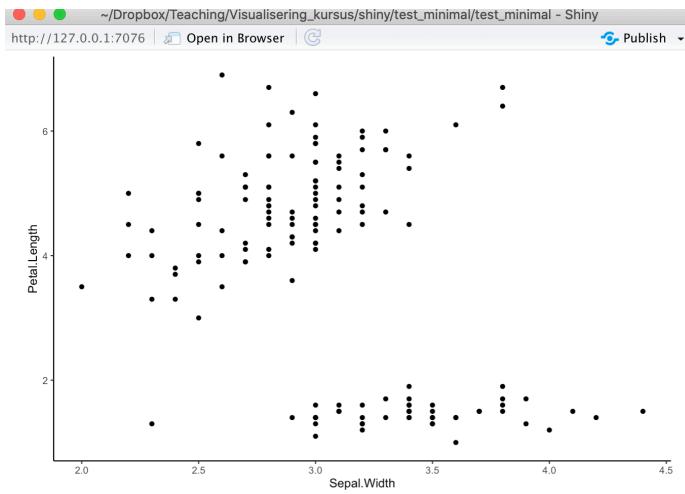
#ui: output plottet 'my_plot'
ui <- fluidPage(
  plotOutput("my_plot")
)

#server: lav plot og kalde den for output$my_plot
server <- function(input, output) {
  output$my_plot <- renderPlot({
    iris %>%
      ggplot(aes(x=Sepal.Width,y=Petal.Length)) +
      geom_point() +
      theme_classic()
  })
}

# køre appen
shinyApp(ui = ui, server = server)

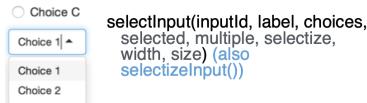
```

Hvis vi køre koden kan man se at vi har et plot frem. Appen er dog ikke interaktiv.



13.4.5 Minimal example - interaktiv.

Vi vil gerne gøre vores app mere fleksibelt - lad os gøre det interaktiv ved at plotte et subset af Iris for en af de tre species, som vælges fra en “drop-down” box - vi kan anvende en funktion som hedder `selectInput`:



Her er koden for vores `selectInput`, som skal tilføjes indenfor `fluidPage`.

```
selectInput(inputId = "Species", #giv en id
           choices = iris %>% distinct(Species) %>% pull(Species), #setosa, virginica eller
           selected = "setosa", #default species
           label = "Choose species") #label på plottet
```

Vi skal også ændre koden på plottet så at det reagerer på vores `selectInput`:

- Vi laver en subset `iris_subset` af de data efter “Species” vi valgt i vores “drop-down box” ved at angiv `Species==input$Species` indenfor funktionen `filter`.
- Vi laver et scatter plot med `iris_subset`:

```
iris_subset <- iris %>%
  filter(Species==input$Species)

iris_subset %>%
  ggplot(aes(x=Sepal.Width, y=Petal.Length)) +
  geom_point() +
  theme_classic()
```

Her tilføjer jeg de kode chunks fra ovenpå til vores program:

```
library(shiny)
library(tidyverse)
data(iris)

ui <- fluidPage(
  selectInput(inputId = "Species",
              choices = iris %>% distinct(Species) %>% pull(Species),
              selected = "setosa",
              label = "Choose species"),

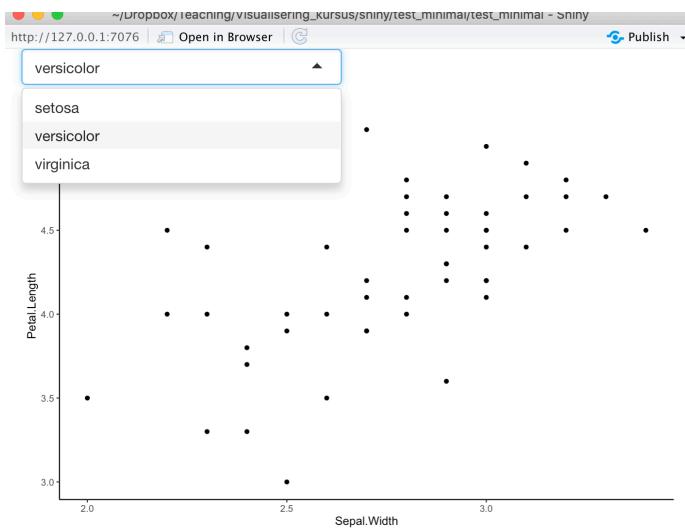
  plotOutput("my_plot")
)

server <- function(input, output) {
  output$my_plot <- renderPlot({
    iris_subset <- iris %>%
      filter(Species==input$Species)

    iris_subset %>%
      ggplot(aes(x=Sepal.Width,y=Petal.Length)) +
      geom_point() +
      theme_classic()
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Som man kan se i screenshot nedenunder, har vi en “drop-down box” hvor vi kan vælge imellem de tre **Species**.



13.4.6 Flere plots på samme Shiny app

- Vi laver to plotter, et scatter plot og et histogram, som reagere på `selectInput` som fortæller de subset af de data efter `Species` som skal plottes.
- Vi kalder dem for `p1` og `p2` og plotte dem ved siden af hinanden med `grid.arrange`:

```
grid.arrange(p1,p2,ncol=2)

library(shiny)
library(tidyverse)
library(gridExtra)
data(iris)

ui <- fluidPage(
  selectInput(inputId = "Species",
              choices = iris %>% distinct(Species) %>% pull(Species),
              selected = "setosa",
              label = "Choose species"),

  plotOutput("my_plot")
)

server <- function(input, output) {
  output$my_plot <- renderPlot({
    iris_subset <- iris %>%
      filter(Species == input$Species)
  })
}
```

```

    filter(Species==input$Species)

    p1 <- iris_subset %>%
      ggplot(aes(x=Sepal.Width,y=Petal.Length)) +
      geom_point(colour="steel blue") +
      ggtitle(input$Species) +
      theme_classic()

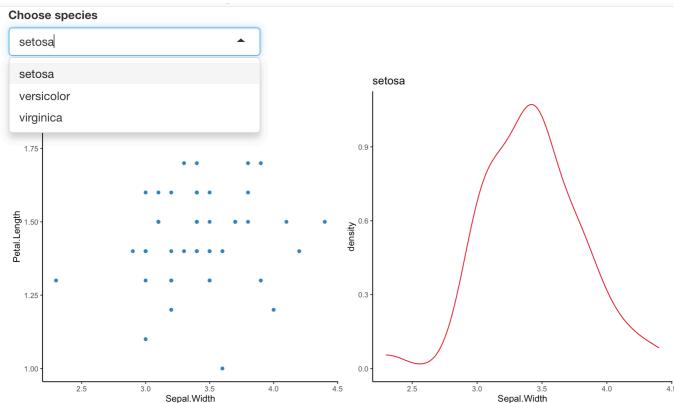
    p2 <- iris_subset %>%
      ggplot(aes(x=Sepal.Width)) +
      geom_density(colour="firebrick3") +
      ggtitle(input$Species) +
      theme_classic()

    grid.arrange(p1,p2,ncol=2)

  }
}

# Run the application
shinyApp(ui = ui, server = server)

```



13.4.7 Involvere en table

Vi vil gerne tilføj en table så vi kan se vores data i appen. Jeg vil gerne specificere hvor mange rækker jeg skal vise frem i appen - vi kan gøre den interaktiv ved at giv en `numericInput` funktion:

```
1 numericInput(inputId, label, value,
               min, max, step)
```

Her er koden for vores `numericInput`. Bemærk, at man skriver `tableOutput("my_table")`

for at fortælle at vi vil viser "my_table" i appen.

```
numericInput(inputId = "num_rows",
            label = "Number of observations to view:",
            value = 5), #giv id num_rows
#label på selve plottet
#default værdi

tableOutput("my_table") #fortæl, at vi vil output "my_table"
```

Her er koden for `my_table`. Bemærk at vi anvender funktion `renderTable` i stedet for `renderPlot`, og der er en indstilling indenfor `head` hvor vi kan specificere hvor mange række vi skal have (med `n = input$num_rows` - altså tallet som vi skriver i appen).

```
output$my_table <- renderTable({
  iris_subset <- iris %>%
    filter(Species==input$Species)
  head(iris_subset, n = input$num_rows)
})
```

Jeg tilføjer de to kode chunks til vores app i følgende (som kan kopireres og blive kørte):

```
library(shiny)
library(tidyverse)
library(gridExtra)
data(iris)

ui <- fluidPage(
  selectInput(inputId = "Species",
              choices = iris %>% distinct(Species) %>% pull(Species),
              selected = "setosa",
              label = "Choose species"),

  numericInput(inputId = "num_rows",
              label = "Number of observations to view:",
              value = 5),

  plotOutput("my_plot"),
  tableOutput("my_table")

)

server <- function(input, output) {

  output$my_plot <- renderPlot({

    iris_subset <- iris %>%
      filter(Species==input$Species)
```

```

p1 <- iris_subset %>%
  ggplot(aes(x=Sepal.Width,y=Petal.Length)) +
  geom_point(colour="steel blue") +
  ggtitle(input$Species) +
  theme_classic()

p2 <- iris_subset %>%
  ggplot(aes(x=Sepal.Width)) +
  geom_density(colour="firebrick3") +
  ggtitle(input$Species) +
  theme_classic()

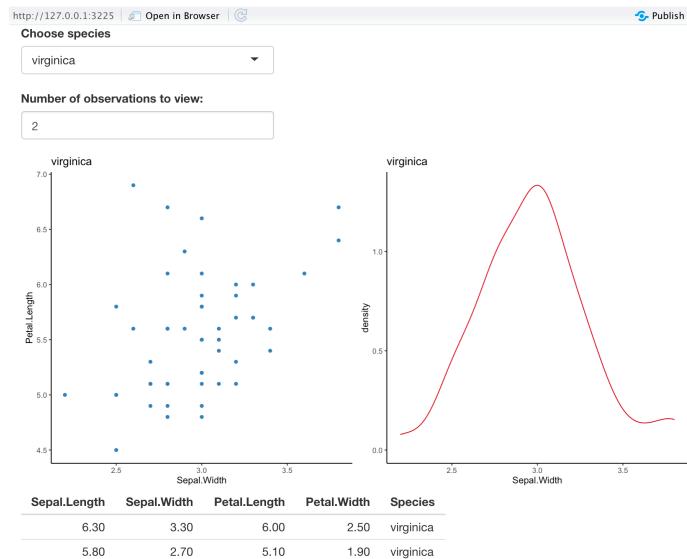
grid.arrange(p1,p2,ncol=2)

})

output$my_table <- renderTable({
  iris_subset <- iris %>%
    filter(Species==input$Species)
  head(iris_subset, n = input$num_rows)
})
}

# Run the application
shinyApp(ui = ui, server = server)

```



13.4.8 Add a slider

- Lad os tilføj en `sliderInput` for at styr punkt eller linje størrelse i plottet.
Den ser sådan ud:



```
sliderInput(inputId, label, min,
           max, value, step, round, format,
           locale, ticks, animate, width, sep,
           pre, post)
```

- Vores kode ser sådan ud. Vi lave en slider af integer fra 1 til 10.

```
sliderInput(inputId = "Point",      #giv den id Point
            label = "Point size", #label på selve plottet
            min = 1,                #min værdi
            max = 10,               #max værdi
            step = 1,                #step size
            value = 1),             #default værdi
```

- Vi referer vores point størrelse ind i plottet med `input$Point`:

```
....+ geom_point(size = input$Point, colour="steel blue") + ....
....+ geom_density(colour="firebrick3", lwd=input$Point) + .....
```

- Vi tilføj ovenstående kode til vores plot:

```
library(shiny)
library(tidyverse)
library(gridExtra)
data(iris)

ui <- fluidPage(
  selectInput(inputId = "Species",
              choices = iris %>% distinct(Species) %>% pull(Species),
              selected = "setosa",
              label = "Choose species"),

  numericInput(inputId = "num_rows",
              label = "Number of observations to view:",
              value = 5),

  sliderInput(inputId = "Point",
              label = "Point size",
              min = 1,
              max = 10,
              step = 1,
              value = 1),

  plotOutput("my_plot"),
  tableOutput("my_table")
```

```

)
server <- function(input, output) {
  output$my_plot <- renderPlot({
    iris_subset <- iris %>%
      filter(Species==input$Species)

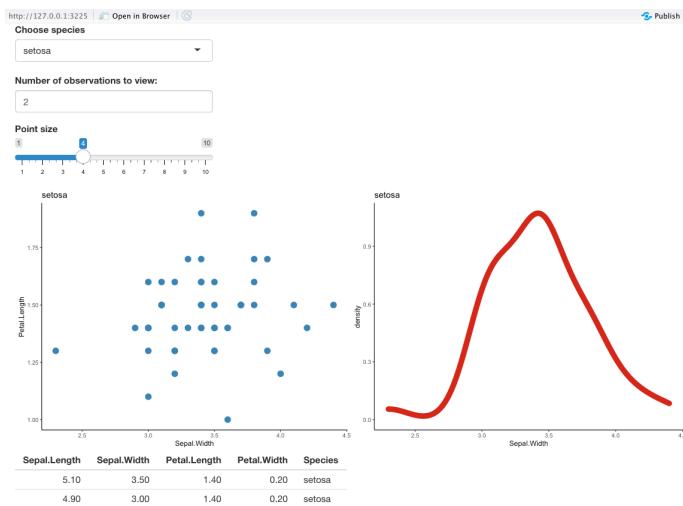
    p1 <- iris_subset %>%
      ggplot(aes(x=Sepal.Width,y=Petal.Length)) +
      geom_point(size = input$Point, colour="steel blue") +
      ggtitle(input$Species) +
      theme_classic()

    p2 <- iris_subset %>%
      ggplot(aes(x=Sepal.Width)) +
      geom_density(size = input$Point,colour="firebrick3") +
      ggtitle(input$Species) +
      theme_classic()

    grid.arrange(p1,p2,ncol=2)
  })
  output$my_table <- renderTable({
    iris_subset <- iris %>%
      filter(Species==input$Species)
    head(iris_subset, n = input$num_rows)
  })
}

# Run the application
shinyApp(ui = ui, server = server)

```



13.4.9 Ekstras som man kan tilføje til ui

Her er nogle ekstra tekst som vi kan tilføj - for eksempel en title. Vi kan også lave nogle forskellige panels for at fremvise de forskellige dele af vores app - for eksempel en side panel og en main panel.

Title

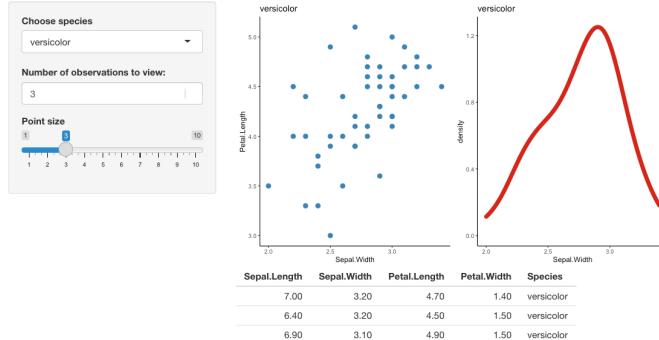
```
titlePanel("Shiny Text")
```

Specificier sidebarLayout

```
sidebarLayout(
  sidebarPanel(),
  mainPanel()
)
```

Her er hvordan appen ser ud med titlen samt en sidebarPanel og mainPanel

My Iris app!



Her er fuld kode som man kan copy/paste ind i app.R:

```
library(shiny)
library(tidyverse)
library(gridExtra)
data(iris)

ui <- fluidPage(
  titlePanel("My Iris app!"),
  sidebarPanel(
    selectInput(inputId = "Species",
               choices = iris %>% distinct(Species) %>% pull(Species),
               selected = "setosa",
               label = "Choose species"),
    numericInput(inputId = "num_rows",
                label = "Number of observations to view:",
                value = 5),
    sliderInput(inputId = "Point",
                label = "Point size",
                min = 1,
                max = 10,
                step = 1,
                value = 1)
  ),
  mainPanel(
    plotOutput("my_plot"),
    tableOutput("my_table")
  )
)

server <- function(input, output) {
  output$my_plot <- renderPlot({
    iris_subset <- iris %>%
      filter(Species==input$Species)

    p1 <- iris_subset %>%
      ggplot(aes(x=Sepal.Width,y=Petal.Length)) +
      geom_point(size = input$Point, colour="steel blue") +
      ggtitle(input$Species) +
      theme_classic()
  })
}
```

```

p2 <- iris_subset %>%
  ggplot(aes(x=Sepal.Width)) +
  geom_density(size = input$Point, colour="firebrick3") +
  ggtitle(input$Species) +
  theme_classic()

grid.arrange(p1,p2,ncol=2)

})

output$my_table <- renderTable({
  iris_subset <- iris %>%
    filter(Species==input$Species)
  head(iris_subset, n = input$num_rows)
})
}

# Run the application
shinyApp(ui = ui, server = server)

```

13.5 Ekstra generelle råd

- Tilføj hensigtsmæssige title/akse labels etc. og gøre dem til en størrelse som er nem at læse (tænk i PowerPoint - personer bagest i rummet skal kunne læse teksten).
- Fjerne unødvendige legends.
- Anvende colour palettes som er designet til at fungere godt for de fleste (for eksempel undgå grøn og rød ved siden af hinanden).
- Viser kun hvad er nødvendige for at fortælle den historie, du gerne vil videregive til andre. For eksempel undgår ekstra tekst som ikke tilfører noget til beskeden.
- En historie på en slide hvis man laver en PowerPoint præsentation.
- Andvende hensigtsmæssige akse transformeringer som giver mest mening til de data
- Undgå piecharts, 3D plots osv. medmindre de absolut tilføj noget ekstra til visualisering (meget sjældent).
- Hvis man tilfører et plot til en PowerPoint, anbefaler jeg at bruge PDFs og ikke PNG/JPG, når en PDF har en højere kvalitet.
- Interaktiv plots kan være sjovt og interessant (men igen sikre at de tilføj nogle ekstra til din præsentation/rapport).

13.6 Andre muligheder/advanceret topics

- Lave præsentations i PowerPoint direkte fra RStudio: <https://support.rstudio.com/hc/en-us/articles/360004672913-Rendering-PowerPoint-Presentations-with-RStudio>
- Shiny cheatsheet: <https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf>
- Debugging with RStudio: <https://support.rstudio.com/hc/en-us/articles/200713843-Debugging-with-RStudio>
- Advanced R: <https://adv-r.hadley.nz/index.html>

13.7 Problemstillinger

0) Lav survey

1) Lav interaktiv plots

Anvend funktionen `ggplotly` fra pakken `plotly` til at lave nogle interaktive plots:

- Et barplot som viser antallet af biler for de forskellige antal cylinders i variablen `cyl` i datasættet `mtcars`.
- Et scatter plot som viser `wt` på x-aksen og `qsec` på y-aksen og med farver efter variablen `gear`.
 - tilføj lineær trend linjer til plottet.

2) Shiny

- Inlæs pakken og se på de forskellige eksempler fk. `runExample("05_sliders")`.
- Lav et nyt app, køre den default app.

3) Shiny

- Slet default tekst og kopi koden fra ovenpå (med slider og to plotte) - køre appen.
- Tilpas koden til at andre de datasæt fra `Iris` til `Penguins`.

4) Shiny

- Lav et app som viser den første n række af datarammen `mtcars` (`numericInput`)
- Tilføj også en drop-down box med funktion `selectInput` for at vise en subset af datarammen efter de forskellige mulige værdier i variablen `gear`.

5) Shiny

- Lav et app med en `sliderInput` som styr antallet af clusters med funktionen `kmeans` i datasættet `penguins`.

- Vis clusters som forskellige farver indenfor et scatter plot (det kan være fk. to variabler eller den første to principal components).
- Tilføj også en `selectInput` for at visualisere dine clusterings i de data subset efter variablen `island`.
- Anvend `sidebarPanel` og `mainPanel` til at separere de inputs fra de plots.
- Tilføj en app title samt plot title/akse labels osv. for at bedste viser din app over for andre.