

Analyse og visualisering af biologiske datasæt - 2022

Sarah Rennie

Last updated: 2022-05-02

Contents

1 Grundlæggende R	5
1.1 Inledning til kapitel	5
1.2 RStudio	5
1.3 Working directory	6
1.4 R pakker	7
1.5 Hvor kommer vores data fra?	8
1.6 Beregninger i R	9
1.7 Dataframes	11
1.8 Descriptive statistics	13
1.9 Statistiske tester	15
1.10 Problemstillinger	27
2 Introduktion til R Markdown	35
2.1 Hvad er R Markdown?	35
2.2 Installere R Markdown	35
2.3 Videodemonstrationer	36
2.4 Oprette et nyt dokument i R Markdown	36
2.5 Skrive baseret tekst	37
2.6 Knitte kode	40
2.7 Kode chunks	40
2.8 R beregninger indenfor teksten i dokument ('inline code')	42
2.9 Working directory	42
2.10 Matematik	43
2.11 Problemstillinger	43
2.12 Færdig for i dag og næste gang	44
2.13 Ekstra links	44
3 Visualisering - ggplot2 dag 1	47
3.1 Inledning og videoer	47
3.2 Transition fra base R til ggplot2	49
3.3 Vores første ggplot	50
3.4 Lidt om ggplot2	53
3.5 Specificere etiketter og titel	55
3.6 Ændre farver	57

3.7	Ændre tema	58
3.8	Forskellige geoms	59
3.9	Troubleshooting	74
3.10	Problemstillinger	75
3.11	Næste gang	83
4	Visualisering - ggplot2 dag 2	85
4.1	Indledning og videoer	85
4.2	Koordinat systemer	86
4.3	Mere om farver og punkt former	93
4.4	Annotations (<code>geom_text</code>)	100
4.5	Adskille plots med facets (<code>facet_grid/facet_wrap</code>)	106
4.6	Gemme dit plot	113
4.7	Problemstillinger	113
4.8	Ekstra links	121

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 selv følgelig gerne springe sektionen over, hvis du allerede har meget erfaring med base R, men det kan være værd at tjekke, om der noget ting, der lige skal gennemgås. En god tilgang er bare at arbejde gennem problemstillingerne nedenfor, og bruger følgende notater som en reference.

1.6.1 Vectorer

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

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

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

Man er ikke begrænset til tal:

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

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

1.6.2 datatyper

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

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

Her er en liste overfor nogle af de vigtigste datatyper:

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

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

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

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

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

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

1.7 Dataframes

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

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

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

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

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

```
mydf$personID
```

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

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

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

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

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

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

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

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

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

1.7.1 Delmægder af dataframes

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

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

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

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

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

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

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

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

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

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

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

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

Jeg kan også angive et variabelnavn direkte:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.8 Descriptive statistics

1.8.1 Simulere data fra den normale fordeling

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

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

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

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

## [1] 0.12055193 0.69772093 1.07580565 -0.23546727 0.71866700 -1.32677754
## [7] 0.03419063 -0.09919032 1.71477185 -0.26985550 -1.10938022 1.68083572
## [13] 0.62768517 -0.97259026 0.66536629 -1.55877236 -0.17774910 1.89879786
## [19] -0.22021773 -0.80990979 -0.87608150 -1.18278748 1.59573737 0.25318056
## [25] -0.02744638
```

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

```
head(x) #første 6
## [1] 0.1205519 0.6977209 1.0758056 -0.2354673 0.7186670 -1.3267775
tail(x) #sidste 6
## [1] -0.80990979 -0.87608150 -1.18278748 1.59573737 0.25318056 -0.02744638
x[1] #første værdi
## [1] 0.1205519
x[length(x)] #sidste data point
## [1] -0.02744638
```

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] 0.08868342 -0.02744638 1.89879786 -1.55877236 1.00859065 1.00428614

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.
## -1.55877 -0.80991 -0.02745  0.08868  0.69772  1.89880
```

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).

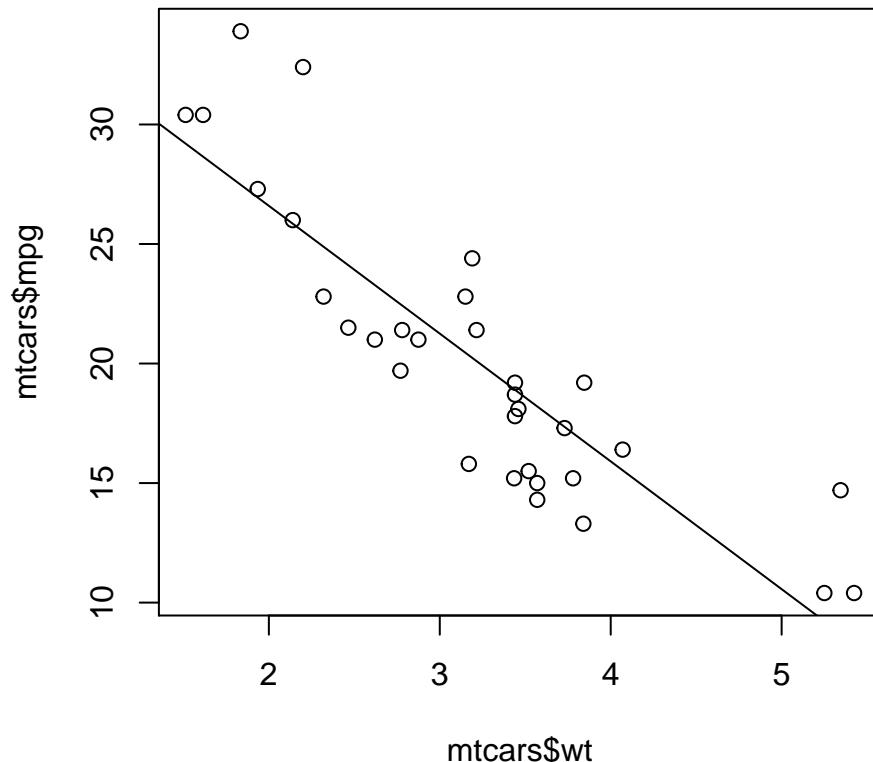
1.9.7 Lineær regression

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

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

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

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

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

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

```
summary(mylm)

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

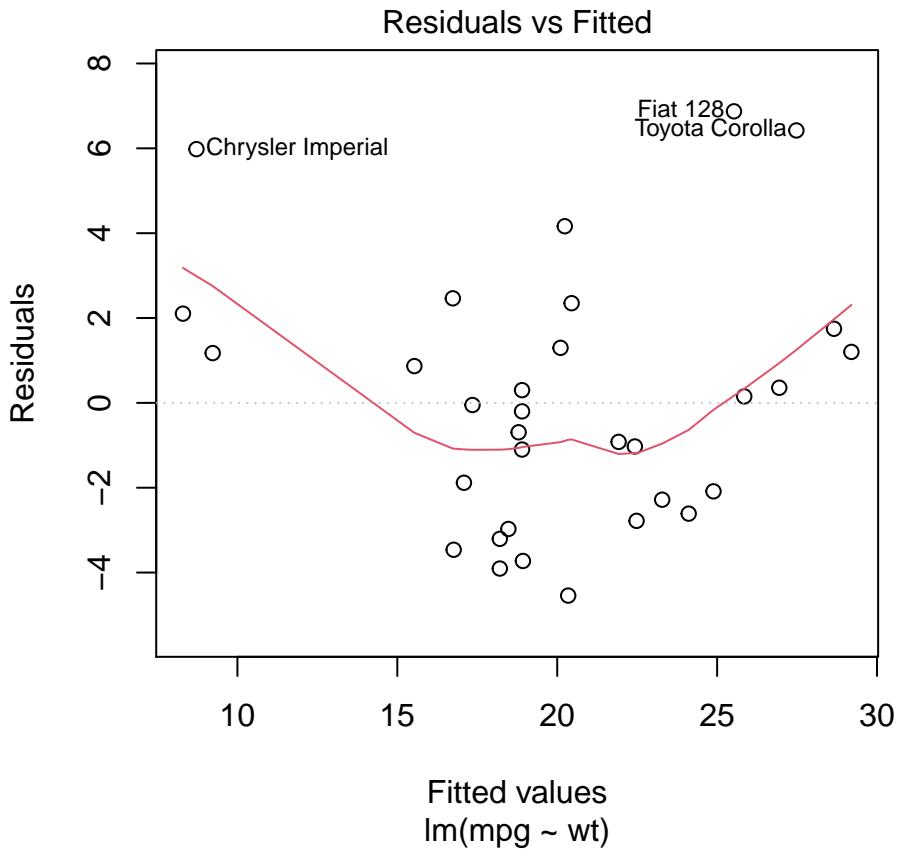
Det fortæller os, at $R^2 = 0.7528$.

1.9.9 Antigelser - lineær regression

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

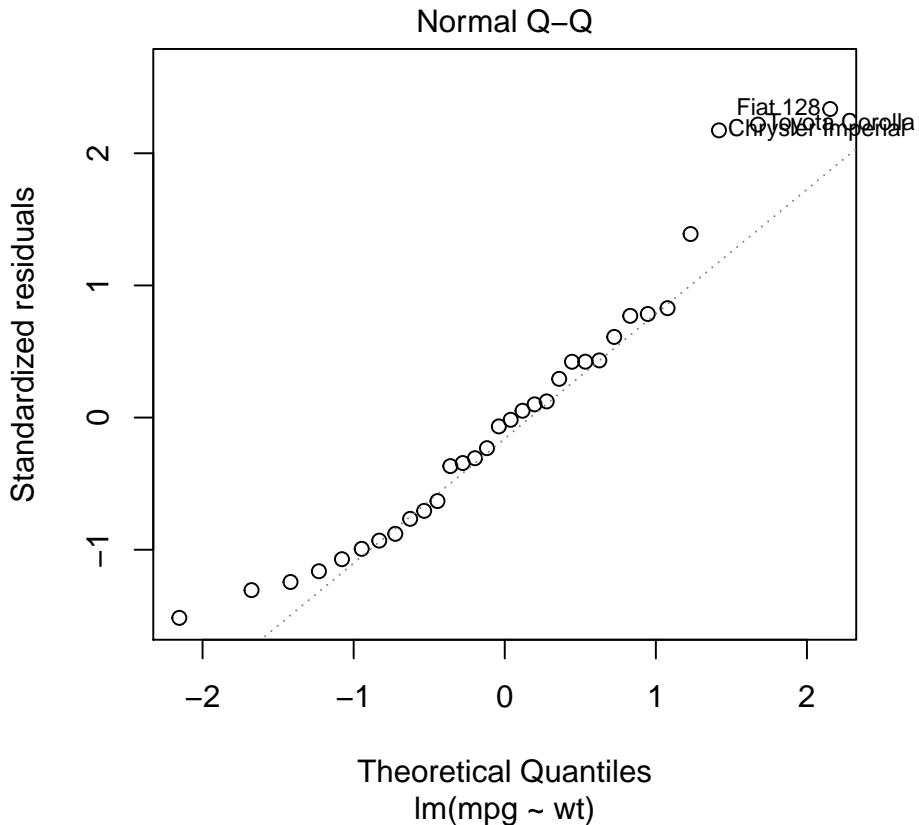
Koden `plot(mylm,which=c(1))` angiver residualer vs predikterede (fitted) værdier - de skal være tilfældigt fordelt over plottet og prikkernes varians skal være nogenlunde konstant langt x-aksen (det giver, at den røde linje er flade).

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



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

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



1.9.10 Multiple lineær regression

Her kan man tilføje flere variabler i vores model formel.

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

```
##
## Call:
## lm(formula = mpg ~ wt + disp, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.4087 -2.3243 -0.7683  1.7721  6.3484
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.96055   2.16454 16.151 4.91e-16 ***
## wt          -3.35082   1.16413 -2.878  0.00743 **
## disp        -0.01234   0.00833 -1.477  0.14284
```

```

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

```

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

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

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

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

```

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

```

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

```
anova(mylm, mylm_cyl)

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

Så kan man se, at cyl er signifikant.

1.10 Problemstillinger

Lav quizzen og ellers vælg øvelser efter egen erfaring:

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

1.10.1 Quiz - Basics

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

1.10.2 Grundlæggende R

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.10.3 Kort analyse med reaktionstider

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

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

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

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

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

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

11) (delmængde af dataframe)

Lav to delmængder af ovenstående datasæt -

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

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

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

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

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

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

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

- Er reaktionstiderne blevet hurtigere eller langsommere i gennemsnit?

13) (beregn forskellen og mean)

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

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

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

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

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

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

Find følgende i outputtet fra R:

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

15) Skriv en kort sætning med din konklusion.

1.10.4 Ekstra øvelser med statistik tests

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

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

```
mytable
```

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

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

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

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

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

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

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

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

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

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

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

```
data(iris)
```

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

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

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

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

Kig på outputtet:

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

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

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

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

```
data(chickwts)
```

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

20) (Lineær regression)

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

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

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

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

21) (Kort intro til multiple lineær regression)

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

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

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

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

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

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

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

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

Chapter 2

Introduktion til R Markdown

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

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

2.1 Hvad er R Markdown?

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

2.2 Installere R Markdown

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

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

2.3 Videodemonstrationer

Jeg har lavet to videoer som kan ses her:

Video 1:

Jeg viser

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

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

Video 2:

Jeg viser en kort lineær regression analyse:

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

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

2.4 Oprette et nyt dokument i R Markdown

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

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

2.4.1 YAML

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

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

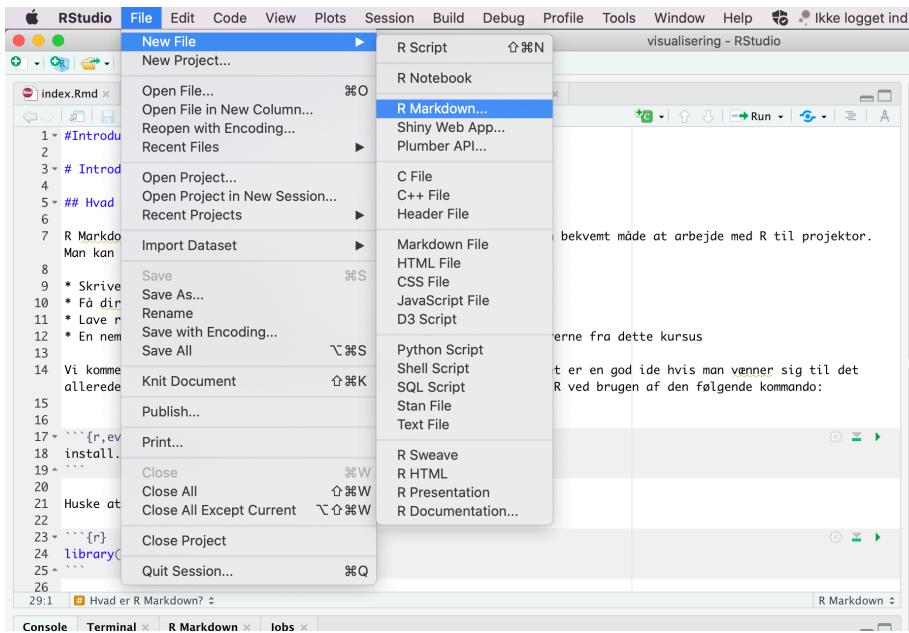


Figure 2.1: Hvordan man åbner et nyt R Markdown dokument

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

eller se en liste af muligheder her på dette cheatsheet:

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

2.4.2 Globale options

Der er også tekst som ser ud som følgende:

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

2.5 Skrive baseret tekst

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

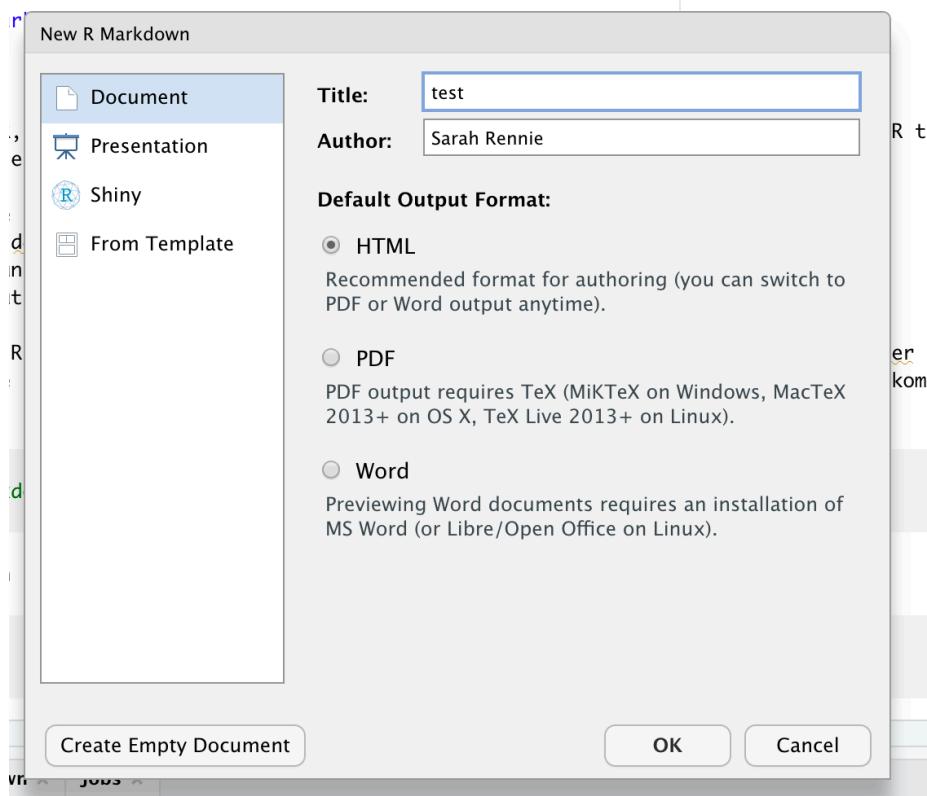


Figure 2.2: Hvordan man åbner et nyt R Markdown dokument

```

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

Figure 2.3: Hvordan man åbner et nyt R Markdown dokument

```

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

  
```

Figure 2.4: Hvordan man åbner et nyt R Markdown dokument

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

```
_italic_ __bold__
```

```
italic bold
```

```
italic bold
```

2.5.1 Headers

Man kan også lave sektioner:

```
# Header 1
```

```
## Header 2
```

```
### Header 3
```

Chapter 3 Header 1

3.1 Header 2

3.1.1 Header 3

Figure 2.5: Caption for the picture.

2.5.2 liste

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

- Item 1

- Item 2
 - Item 2a
 - Item 2b

2.6 Knitte kode

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

2.7 Kode chunks

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

Her er et eksempel af en chunk:

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

## [1] 2
```

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

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

2.7.1 Et godt råd når man arbejder med chunks

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

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

2.7.2 Chunk indstillinger

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

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

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

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

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

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

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

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

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

Ovenstående ser sådan ud når skrevet direkte indenfor teksten:

Her er min kode

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

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

Ovenstående ser sådan ud når skrevet direkte indenfor teksten:

De gennemsnitlige antal af observationer er 4.6666667

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

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

giver:

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

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

2.9 Working directory

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

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

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

2.10 Matematik

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

2.11 Problemstillinger

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

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

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

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

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

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

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

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

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

Knit og se hvad, der sker.

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

2.12 Færdig for i dag og næste gang

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

2.13 Ekstra links

- Her er en ‘quick tour’ https://rmarkdown.rstudio.com/authoring_quick_tour.html

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

Chapter 3

Visualisering - ggplot2 dag 1



3.1 Inledning og videoer

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

3.1.1 Læringsmålene for dag 1

I skal være i stand til at:

- Forstå hvad “Grammar of Graphics” betyder og sammenhængen med den **ggplot2**-pakke
- Lære at bruge funktionen **ggplot** og den relevante **geoms** (**geom_point()**, **geom_bar()**, **geom_histogram()**, **geom_boxplot()**, **geom_density()**)
- Lave en ‘færdig’ figur med en titel og korrekte etiketter på akserne
- Begynde at arbejde med farver og temaer

3.1.2 Hvad er ggplot2?

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

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

3.1.3 Brugen af materialerne

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

3.1.4 Video ressourcer

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

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

- I video 2 dækker vi boxplots.

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

- I video 3 demonstrerer jeg barplots.

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

- Video 4: Histogram og density plots

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

3.2 Transition fra base R til ggplot2

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



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

```
data(iris)
```

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

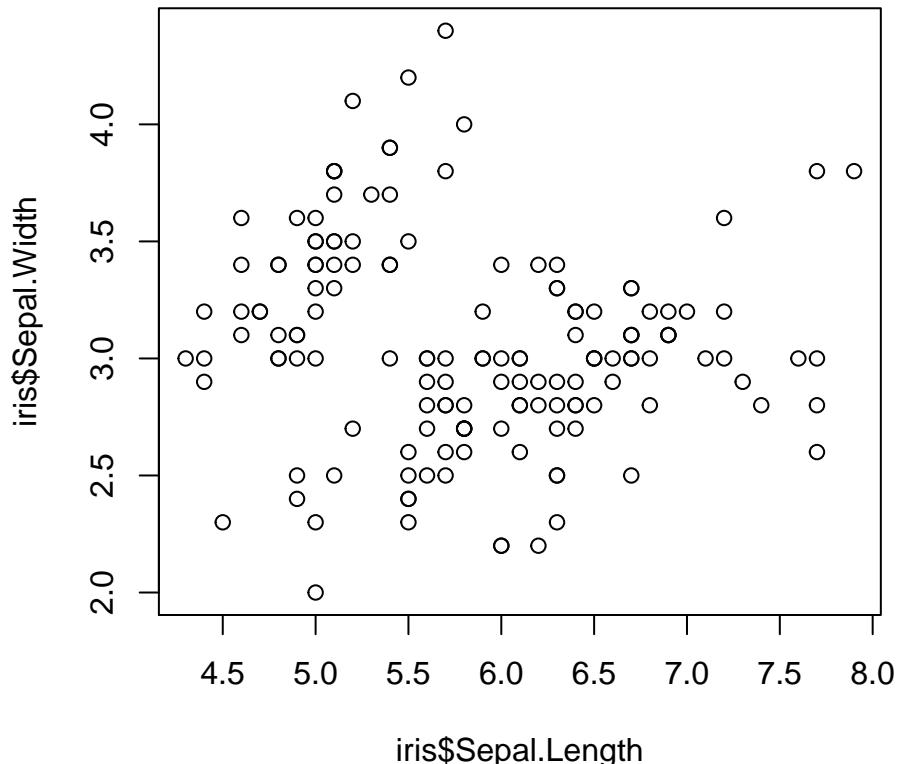
```
summary(iris)
```

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

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

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

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



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

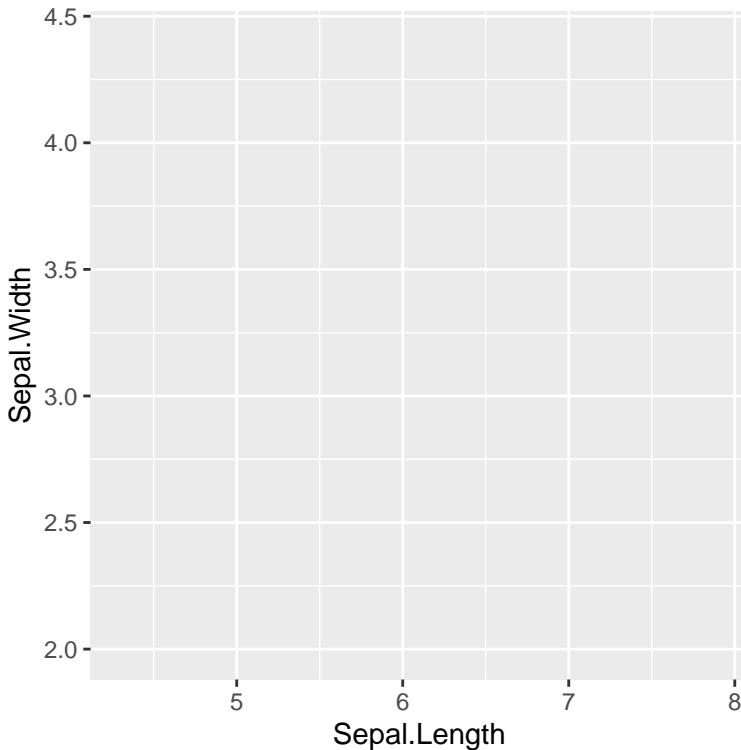
3.3 Vores første ggplot

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

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

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

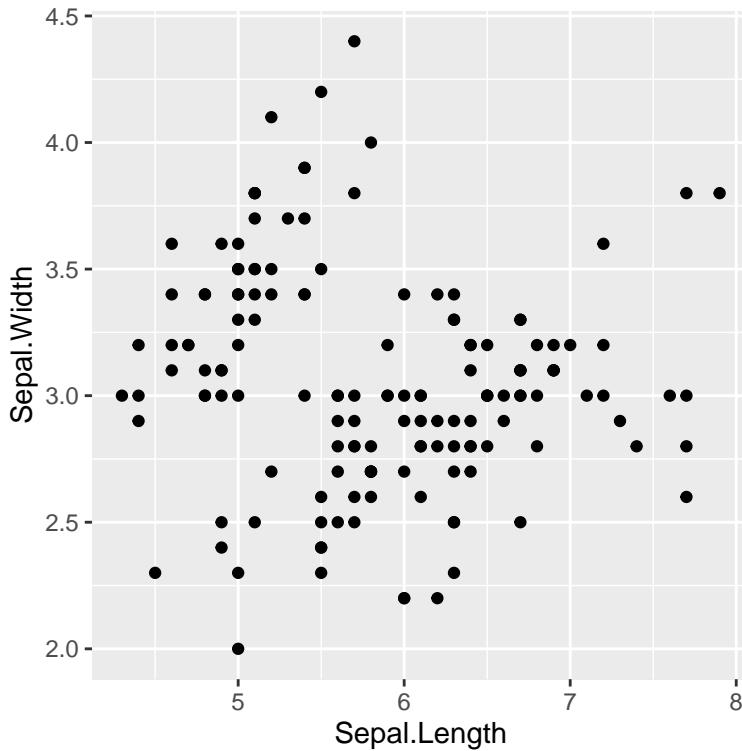
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width))
```



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

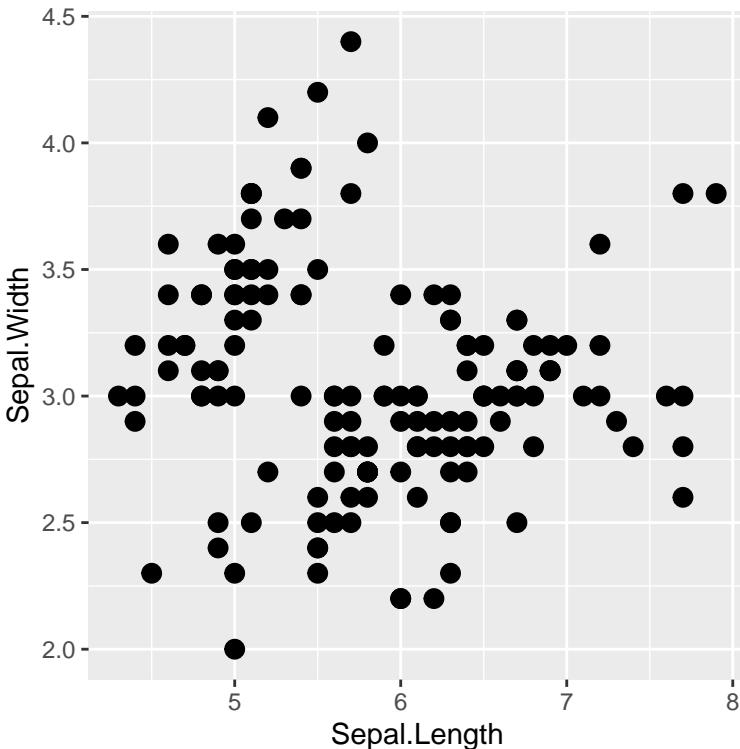
Koden er således:

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



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

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



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

3.4 Lidt om ggplot2

3.4.1 Syntax

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

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

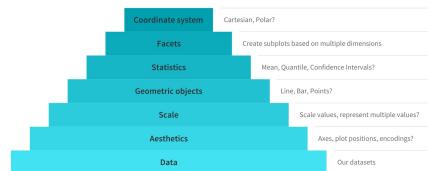
#ggplot2 solution
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point()
```

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

3.4.2 Hvad betyder egentlig grammar of graphics?

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

Major Components of the Grammar of Graphics



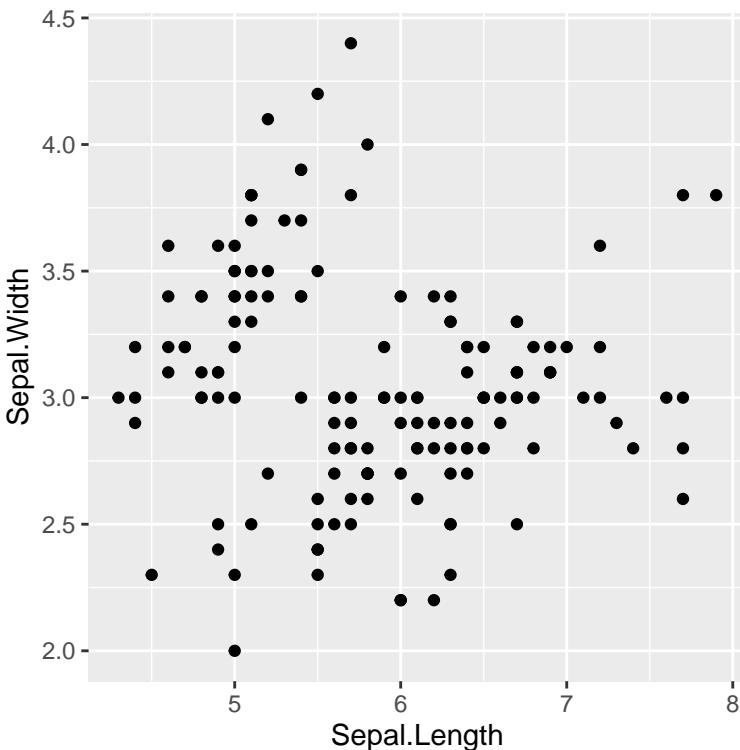
Her er en beskrivelse af de forskellige komponenter til at opbygge et plot:

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

3.4.3 Globale versus lokale æstetik

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

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

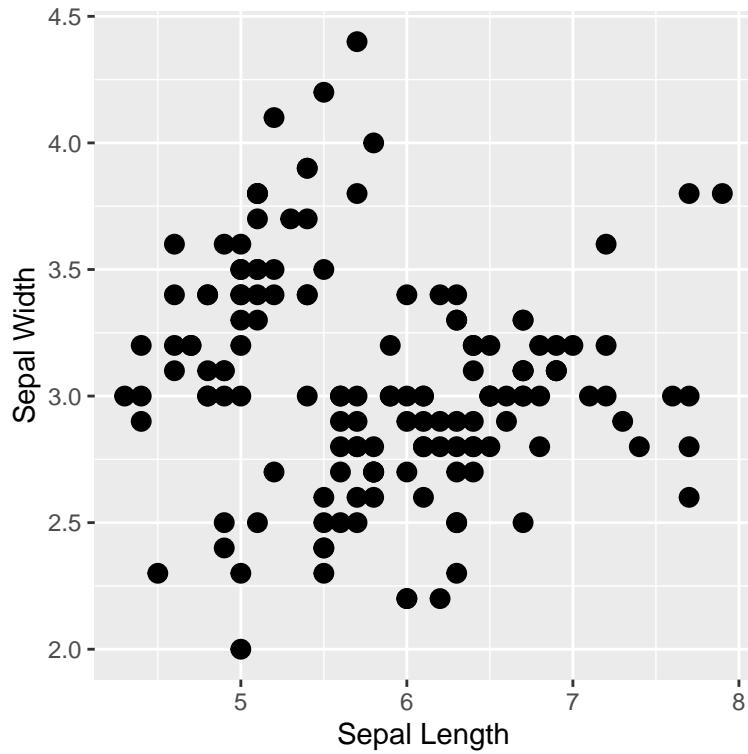


Vi får det samme plot som før, men det er kun `geom_point()` der er påvirket af specificeringen indenfor `aes()`. I simpel situationer som ovenpå er der ingen forskel, men når man har mange forskellige komponenter i spil, så kan det nogle gange give mening at bruge lokale æstetik.

3.5 Specificere etiketter og titel

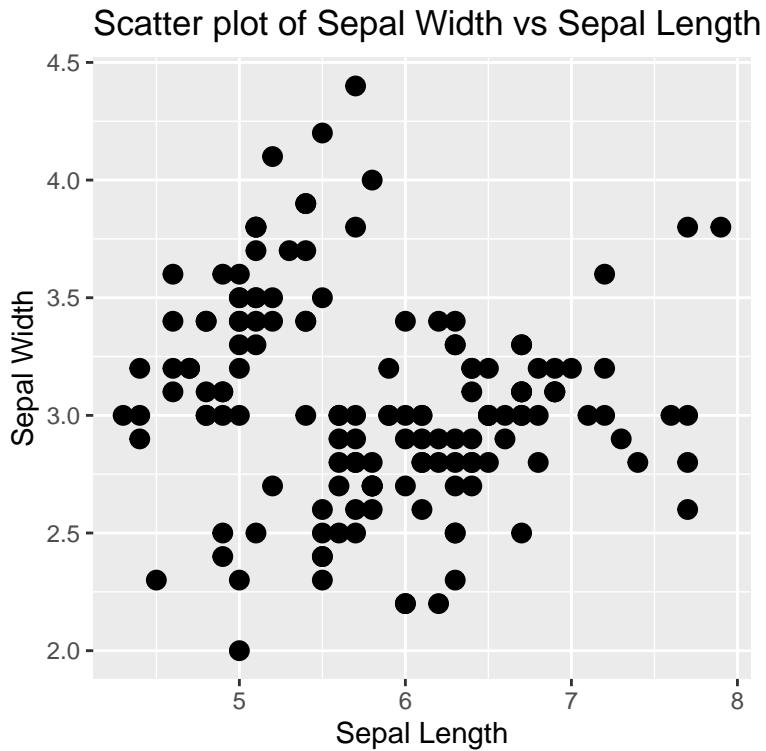
Vi tager udgangspunkt i plottet, vi lavet i ovenstående og prøver at gøre det bedre ved at tilføje nye etiketter og en titel. I `ggplot` kan man opdatere y-akse og x-akse etiketter ved at bruge henholdsvis `ylab` og `xlab`:

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



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

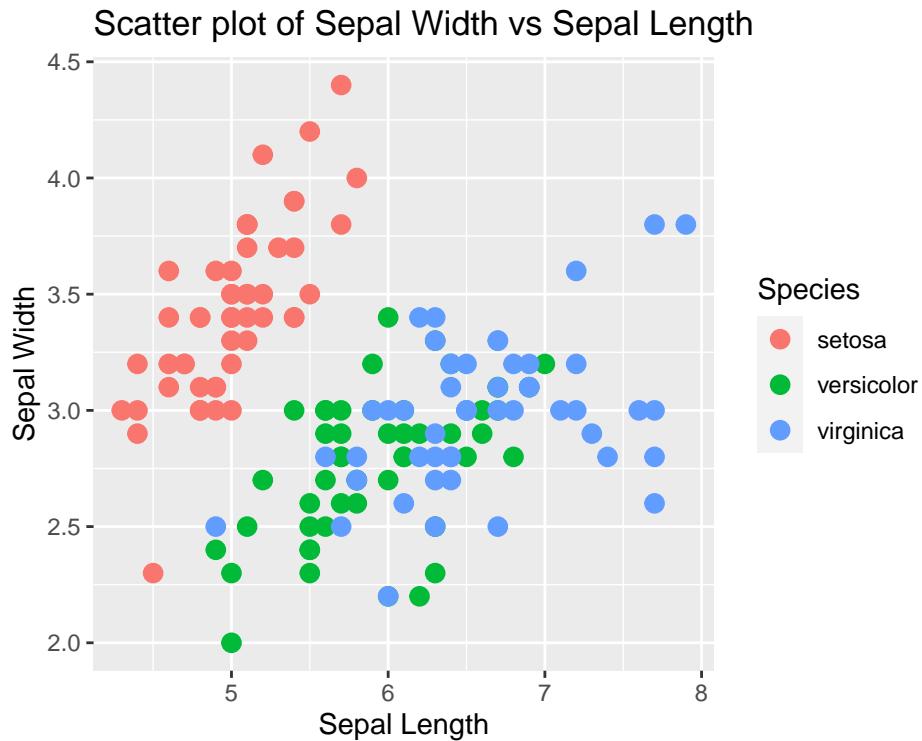
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point(size=3) +  
  xlab("Sepal Length") +  
  ylab("Sepal Width") +  
  ggtitle("Scatter plot of Sepal Width vs Sepal Length")
```



3.6 Ændre farver

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

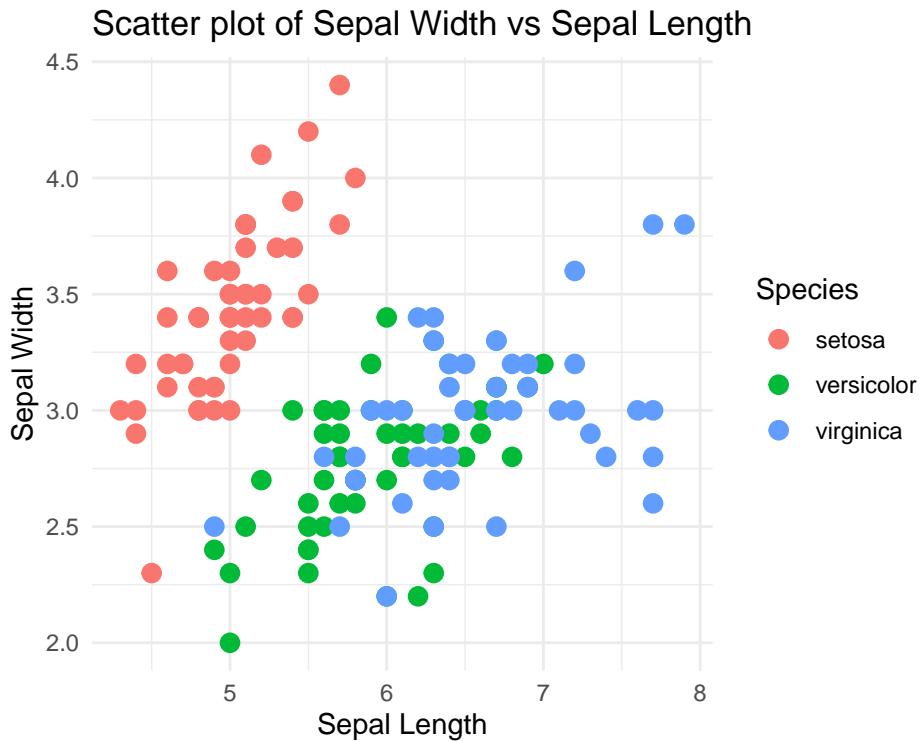
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size=3) +
  xlab("Sepal Length") +
  ylab("Sepal Width") +
  ggtitle("Scatter plot of Sepal Width vs Sepal Length")
```



3.7 Ændre tema

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

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



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

tema

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

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

3.8 Forskellige geoms

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

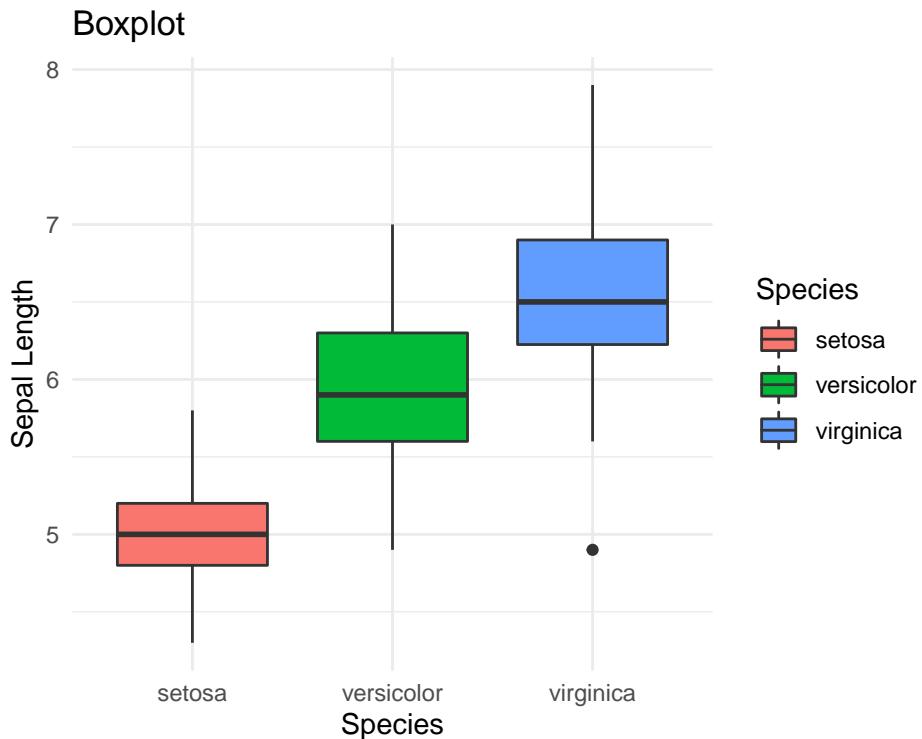
| geom | plot |
|-------------------------------|--------------|
| <code>geom_point()</code> | scatter plot |
| <code>geom_bar()</code> | barplot |
| <code>geom_boxplot()</code> | boxplot |
| <code>geom_histogram()</code> | histogram |
| <code>geom_density()</code> | density |

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

3.8.1 Boxplot (`geom_box`)

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

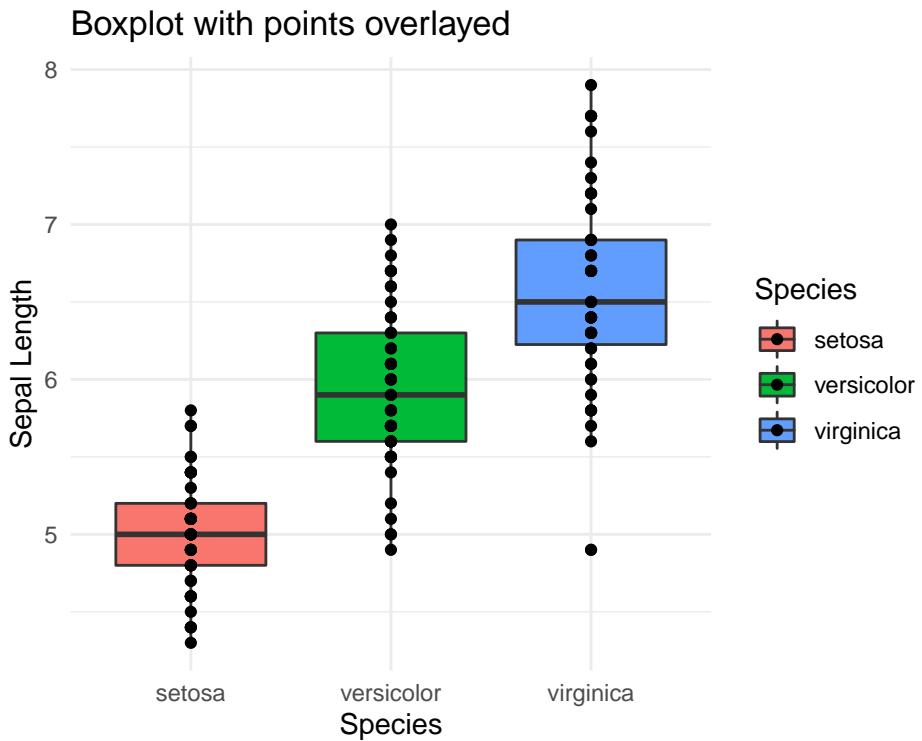
```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  ylab("Sepal Length") +
  ggtitle("Boxplot") +
  theme_minimal()
```



Lave punkter ovenpå

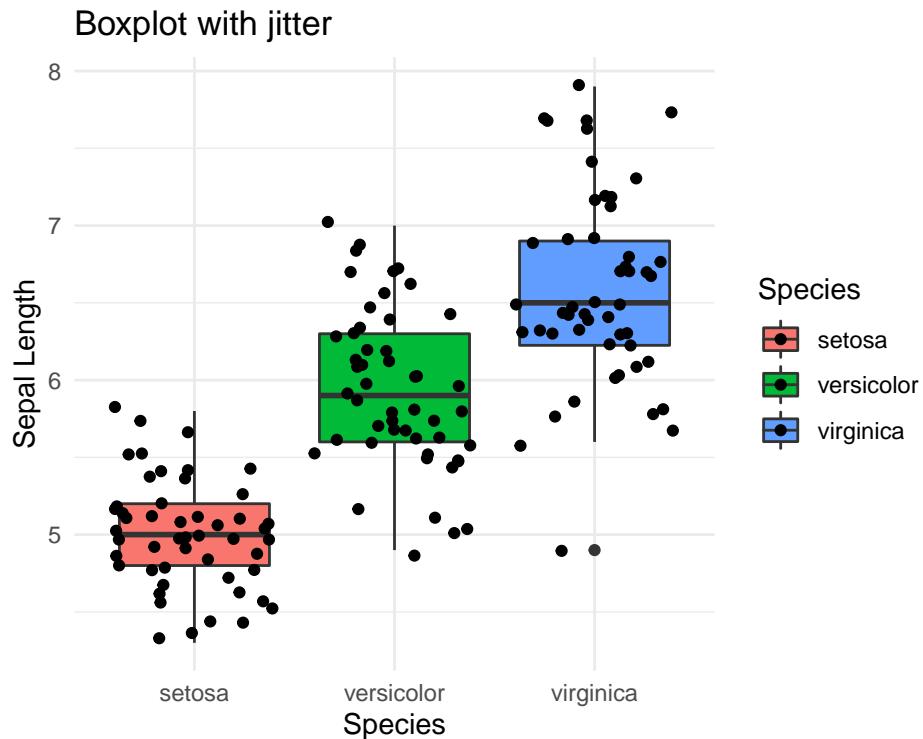
Det kan ofte være nyttigt at plotte de egentlige data punkter ovenpå boxplottet, så I kan se både fordelingen i de data samt de rå data. En løsning er at benytte `geom_point()` ved at tilføje det som komponent over vores eksisterende kode.

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



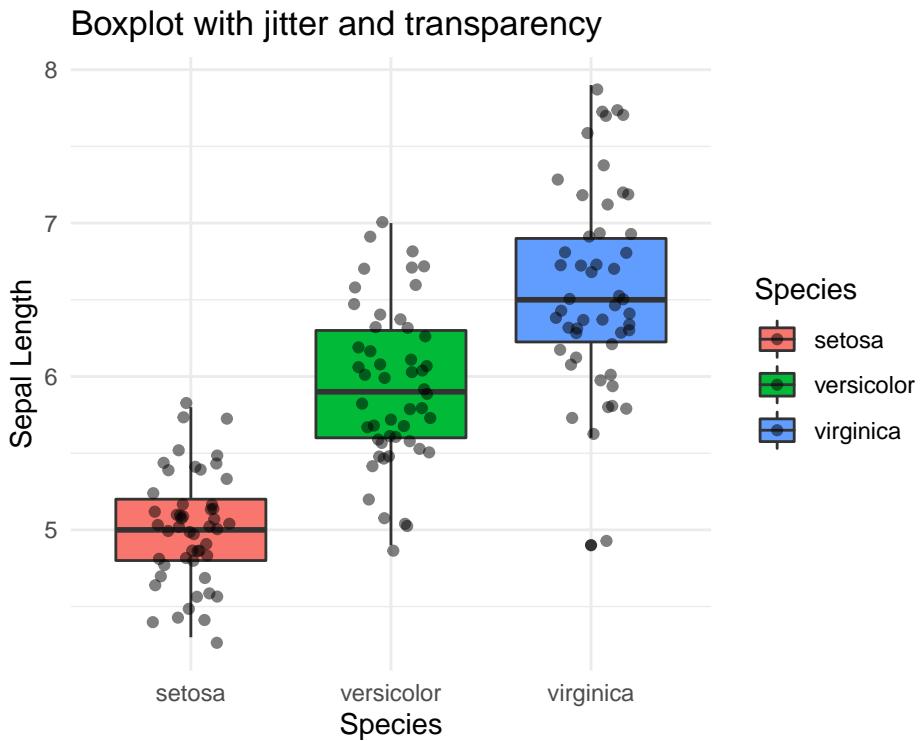
Man kan dog se, at det ikke er særlig informativ, da alle punkter er på den samme lodrette linje. Hvis vi har mange punkter med samme eller næsten samme værdier, så kan vi ikke se de fleste af dem i plottet. En bedre løsning er at indføre noget tilfældighed i punkterne langt x-aksen, så at man mere tydelige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

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



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

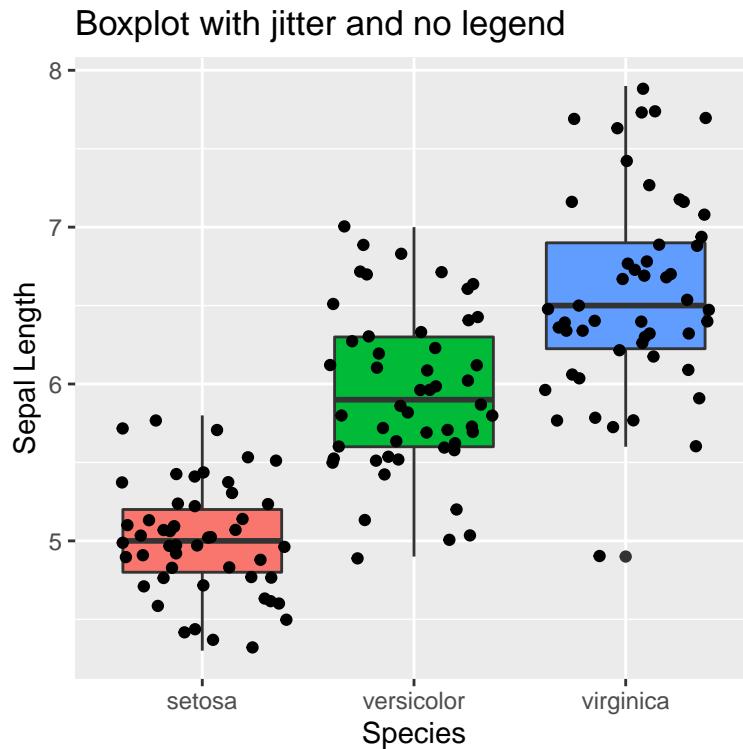
```
ggplot(data=iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_boxplot() +
  geom_jitter(alpha=0.5, width=0.2) +
  ylab("Sepal Length") +
  ggtitle("Boxplot with jitter and transparency") +
  theme_minimal()
```



Fjerne legend hvis unødvendige

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

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

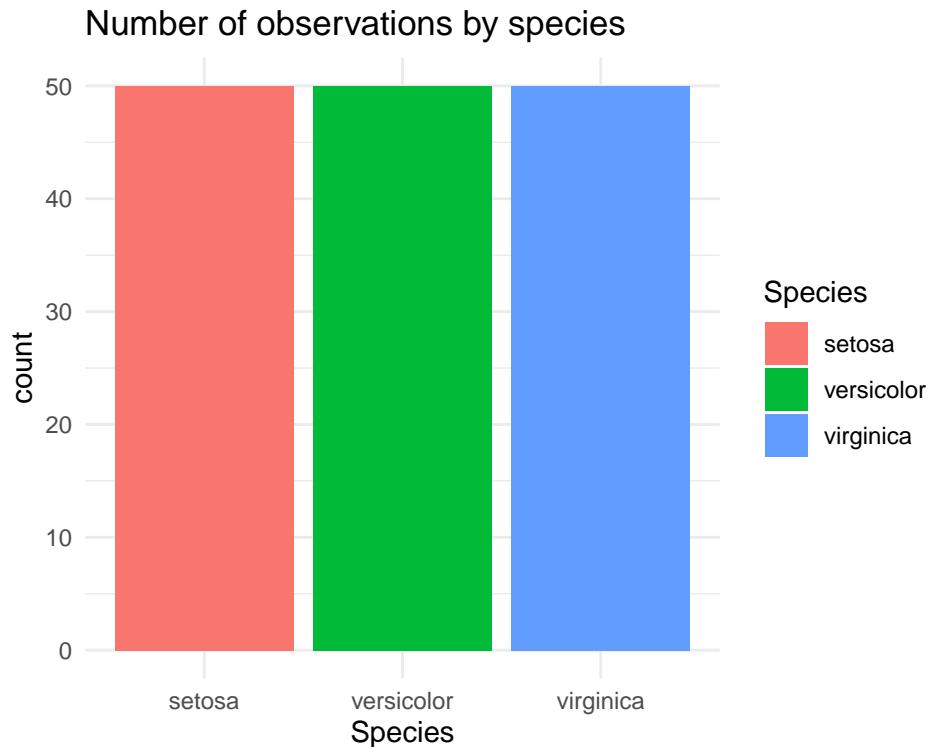


3.8.2 Barplot (geom_bar)

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

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

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



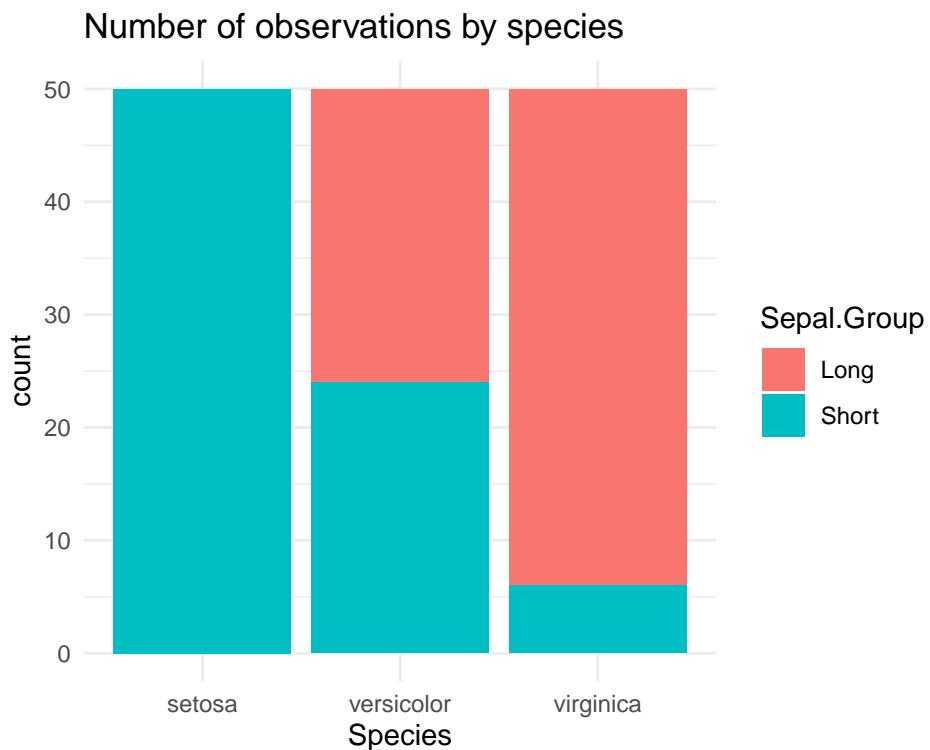
Barplot: stack vs dodge

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

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

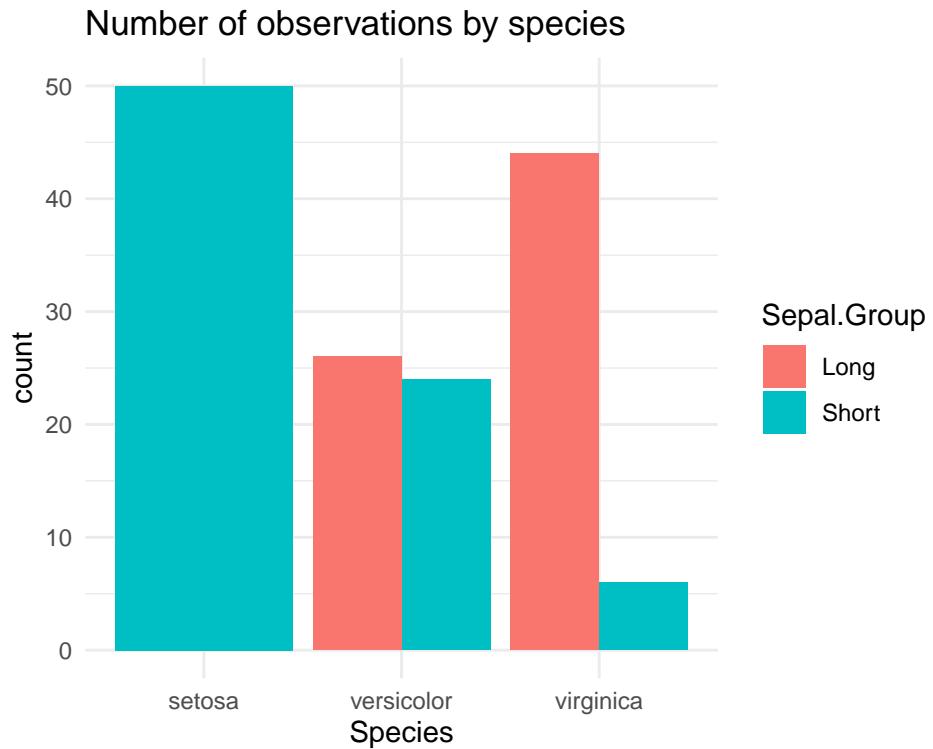
Når jeg laver en barplot med de to variabler, tilføjer jeg `Sepal.Group` med `fill`, og `ggplot` splitter antal observationer efter `Sepal.Group` med farver som repræsenterer `Sepal.Group`, og tilføjer en tilsvarende legend.

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



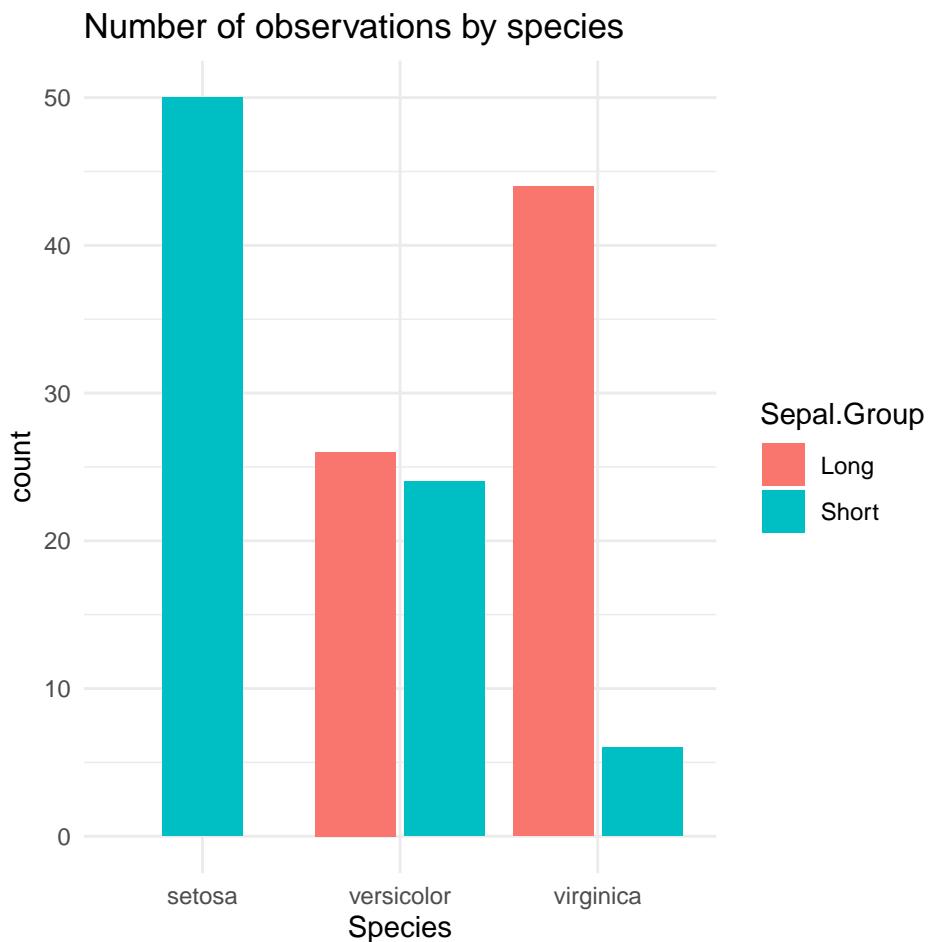
Mange gange foretrækker man at få bars som står ved siden af hinanden. Det kan vi specificere med blot at tilføje `position="dodge"` ind i `geom_bar()`.

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



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

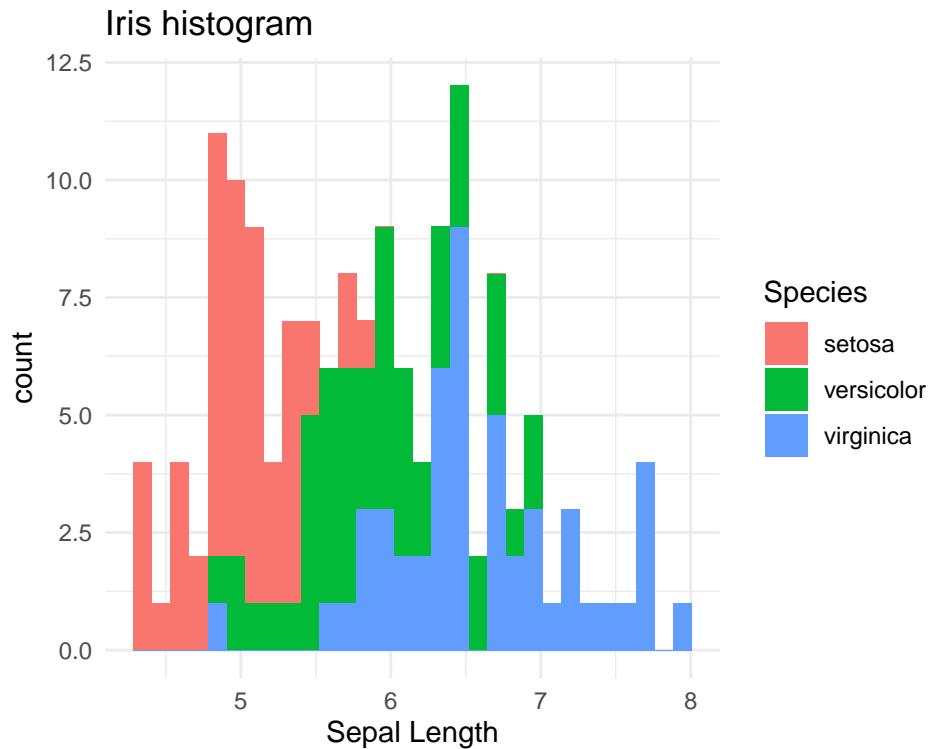
```
ggplot(iris, aes(x=Species, fill=Sepal.Group)) +
  geom_bar(stat = "count", position = position_dodge2(preserve = "single")) +
  ggtitle("Number of observations by species") +
  theme_minimal()
```



3.8.3 Histogram (geom_histogram)

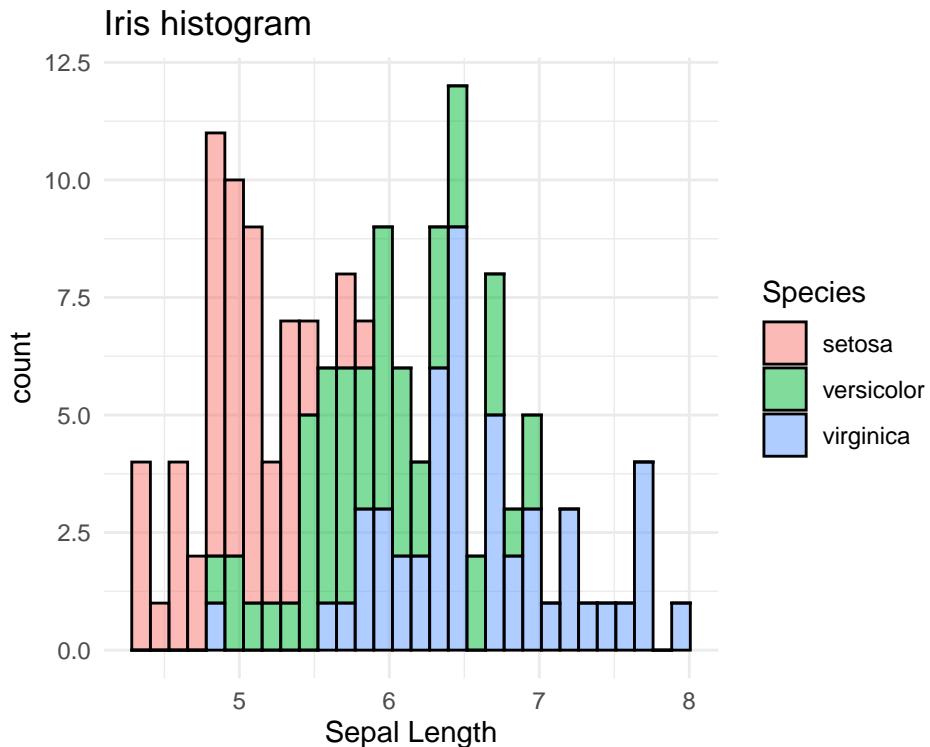
En histogram bruges til at få overblik over hvordan data fordeler sig. I `ggplot2` kan man lave en histogram med `geom_histogram()`. Den x-akse variabel skal være en ‘continuous’ variabel. Her specificerer vi, at vi gerne vil have et histogram for hver Species.

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



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

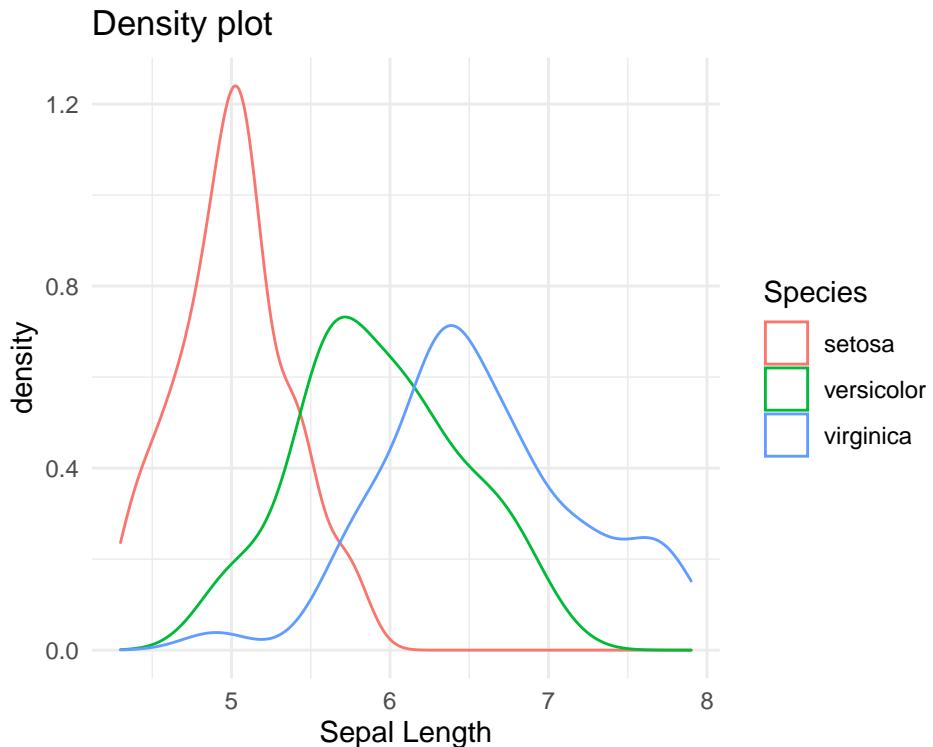
```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram(alpha=0.5, color="black") +
  xlab("Sepal Length") +
  ggtitle("Iris histogram") +
  theme_minimal()
```



3.8.4 Density (geom_density)

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

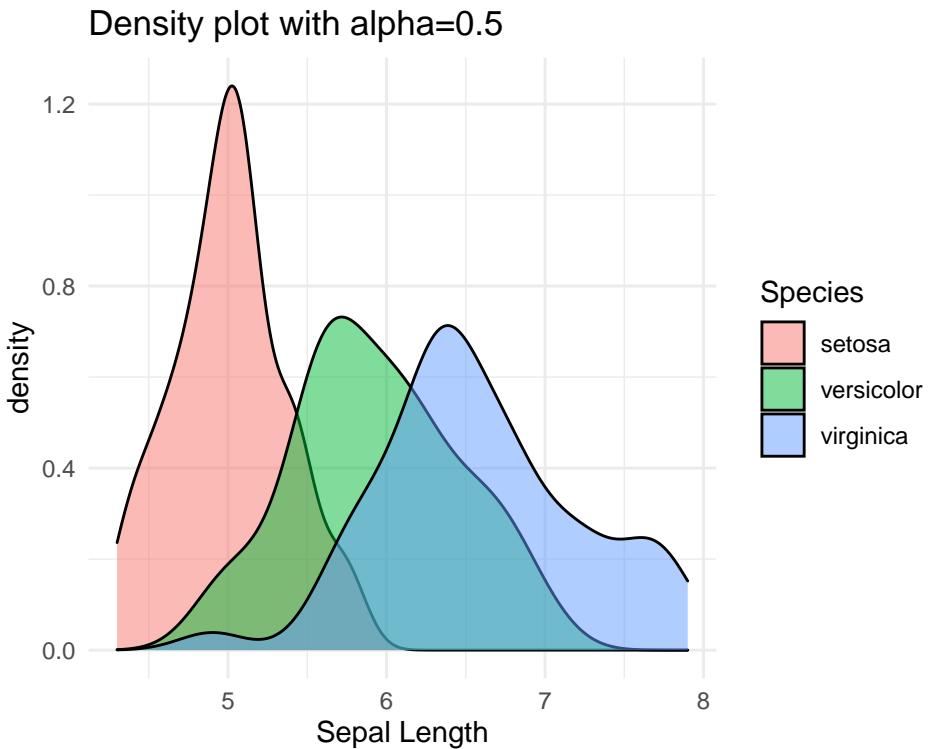
```
ggplot(data=iris, aes(x=Sepal.Length, color=Species)) +
  geom_density() +
  xlab("Sepal Length") +
  ggtitle("Density plot") +
  theme_minimal()
```



Density plot med fill og gennemsigtig farver

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

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

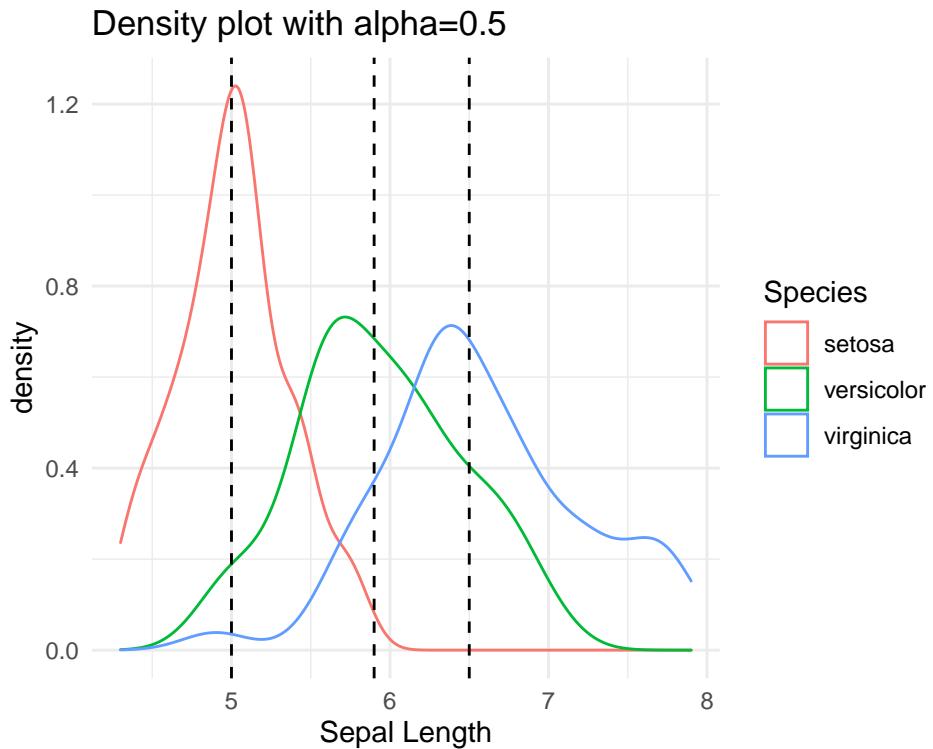


Tilføje middelværdi linjer

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

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

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



3.9 Troubleshooting

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

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

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

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

3.10 Problemstillinger

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

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

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

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

```
data(diamonds)
```

Her er beskrivelsen af **diamonds**:

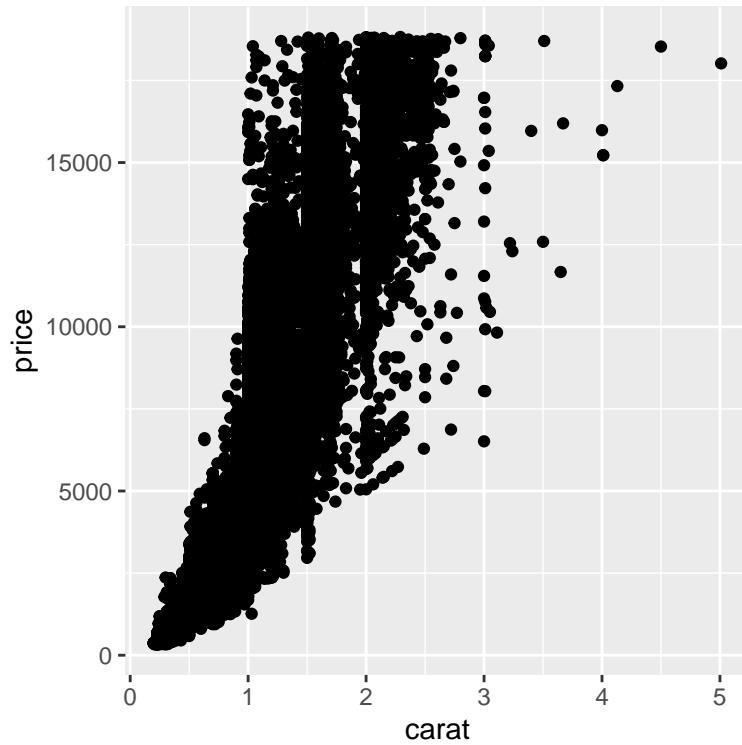
Prices of over 50,000 round cut diamonds: a dataset containing the prices and other attributes of almost 54,000 diamonds.

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

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

- **caret** på x-aksen
- **price** på y-aksen

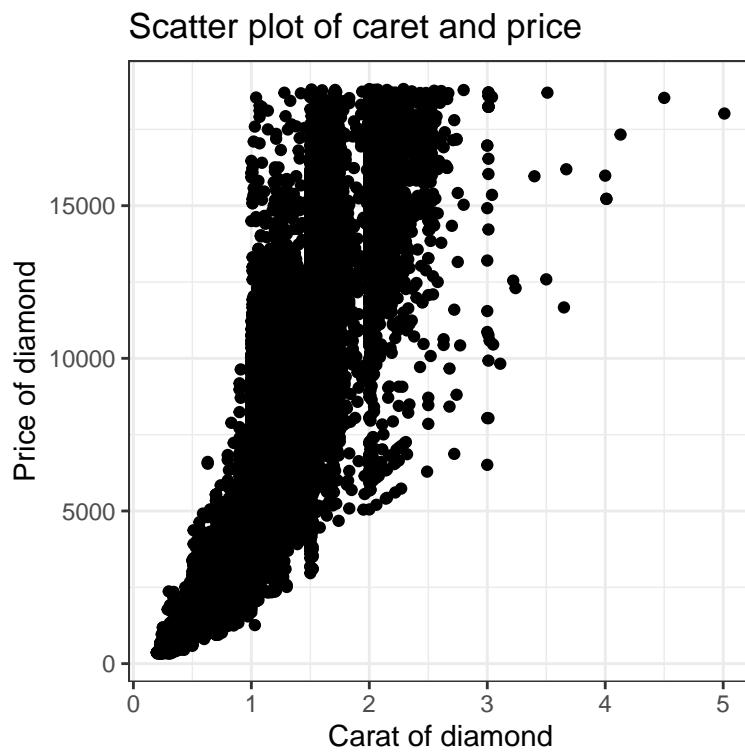
Så at du har noget at sammenligne med, skal dit plot se sådan ud:



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

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

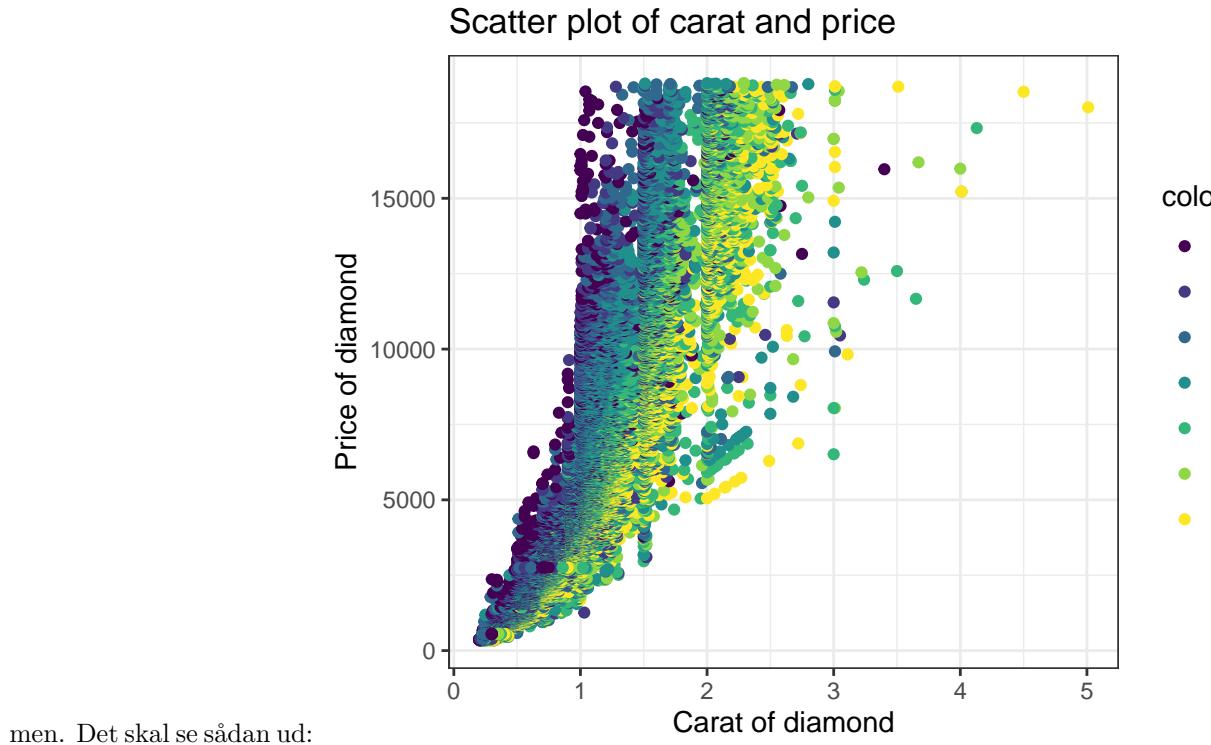
Det skal se sådan ud:



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

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

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

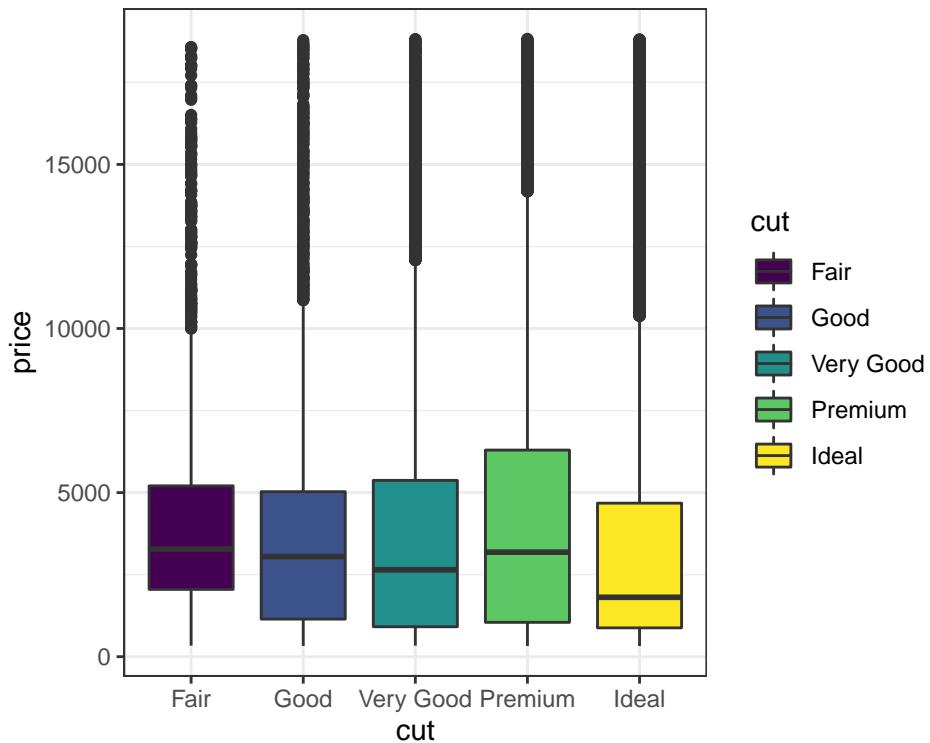


- Nu fjern `color=color` fra funktionen `aes()` og i stedet tilføj `aes(color=color)` i funktionen `geom_point()`. Får du samme resultat?
- Bemærk at det er lige meget om man bruger britisk eller amerikansk stavning i ggplot2 - fl. `colour` eller `color` indenfor `aes()` giiver samme resultat.

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

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

Det skal se sådan ud:

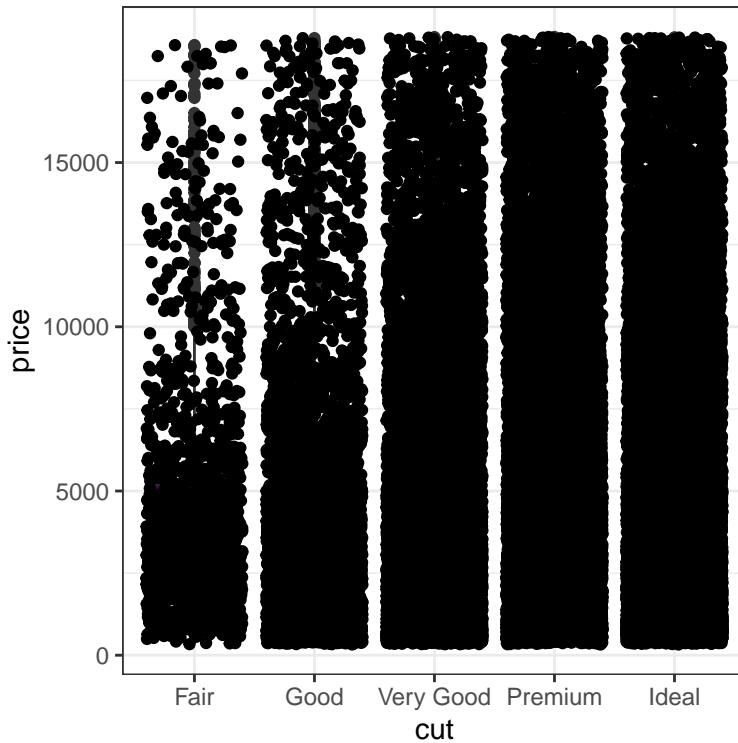


- Hvordan ser det ud, hvis man bruger `colour` i stedet for `fill`? Eller hvis man giver begge to?

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

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

Det skal se sådan ud:



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

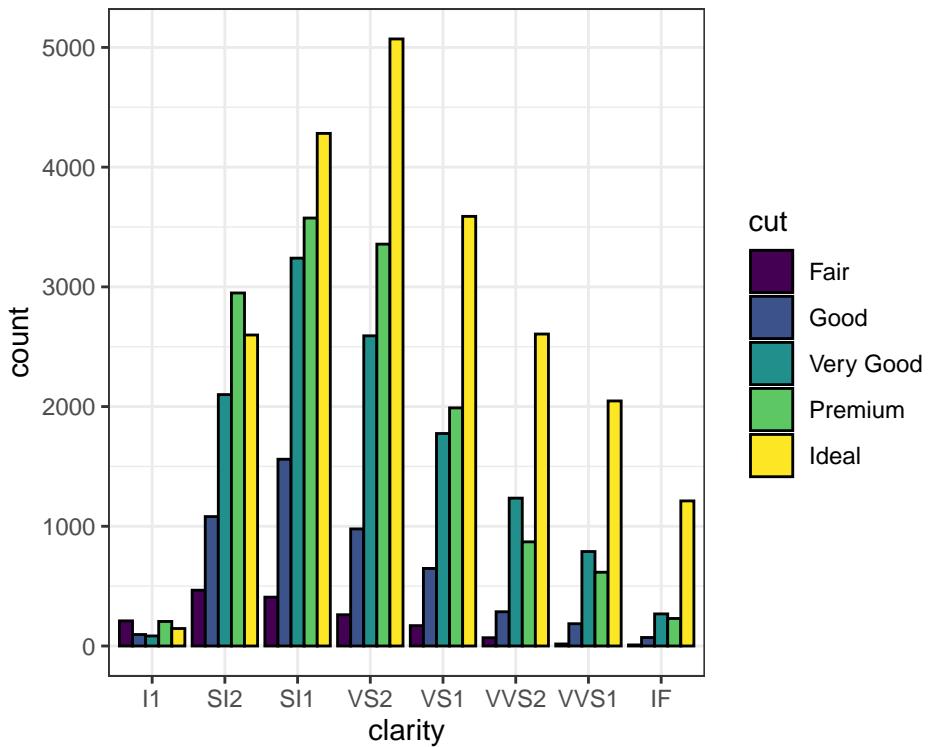
Leg med de tre indstilling `size`, `color` og `alpha` og ser på forskellen. Her er en note om `alpha`:

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

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

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

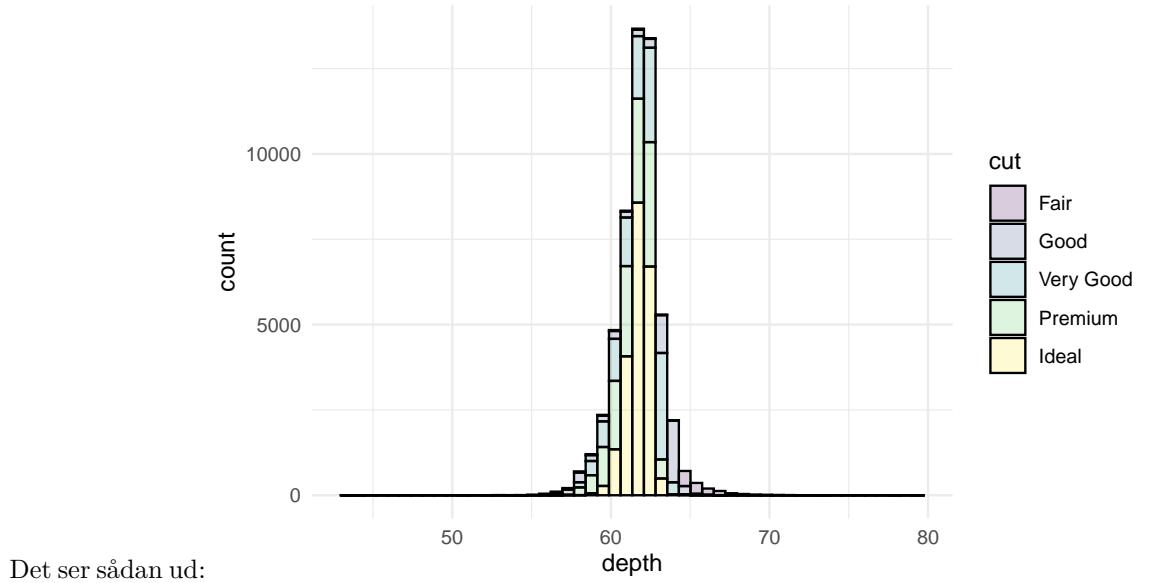
- Variable `clarity` på x-aksen
- Forskellige farver til den gruppe-variable `cut`
- Specifier `position="dodge"` for at få bars ved siden af hinanden
- Brug også indstillingen `colour="black"` og notater effekten
- Tilføj et tema



9) Lav en histogram

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

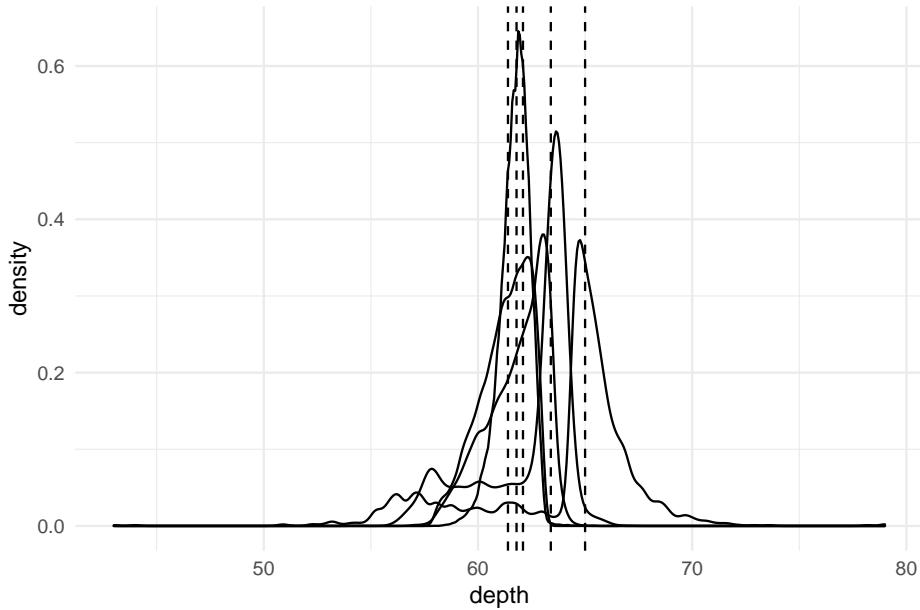
Histogram of depth, according to cut



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

10) Lav en density

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



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

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

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

3.11 Næste gang

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

Chapter 4

Visualisering - ggplot2 dag 2



4.1 Indledning og videoer

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

reference samtidig at du arbejder med problemstillingerne.

4.1.1 Læringsmålene

I skal være i stand til at:

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

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

4.1.2 Video ressourcer

- Video 1: Koordinat systemer (2021)

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

- Video 2: Farver og punkt former (2021)

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

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

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

- Video 4 - Facets

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

4.2 Koordinat systemer

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

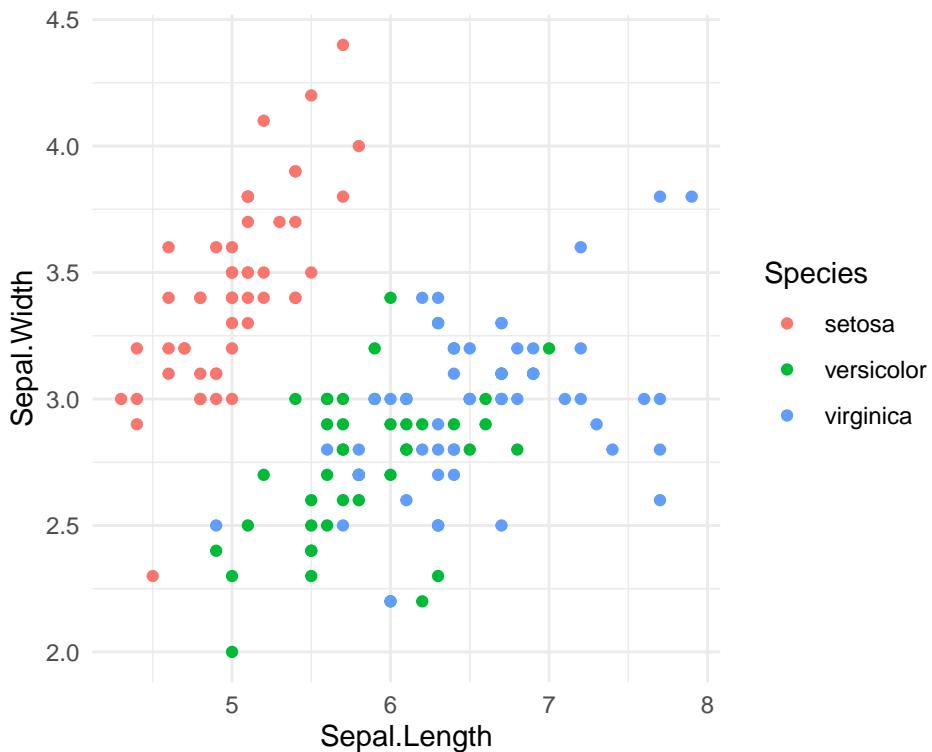
4.2.1 Zoom (`coord_cartesian()`, `expand_limits()`)

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

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

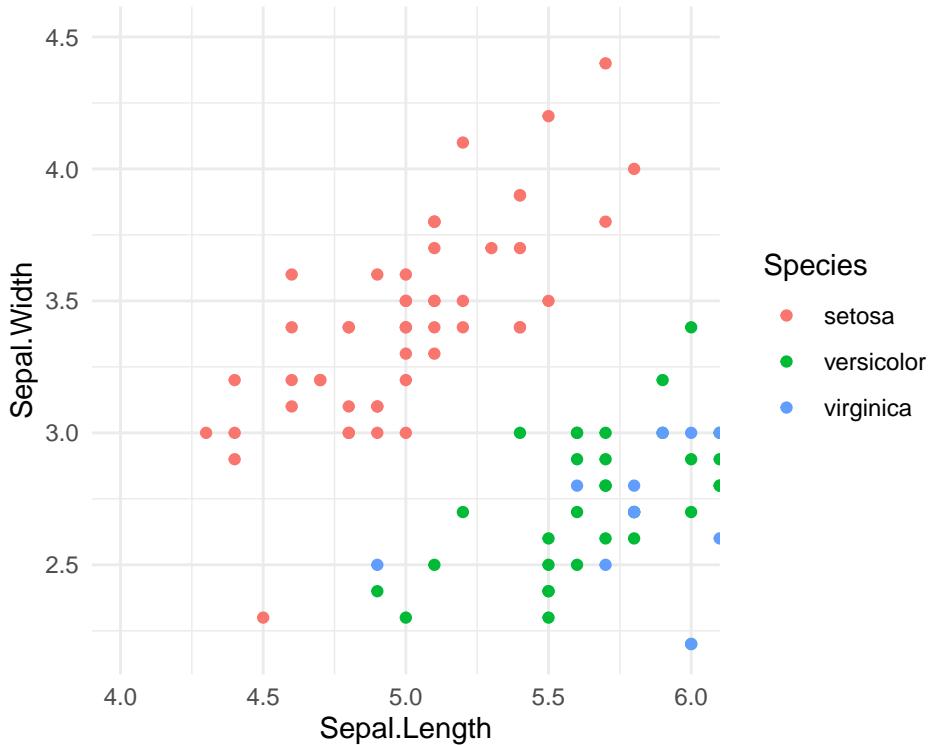
I følgende ses vores oprindeligt scatter plot:

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



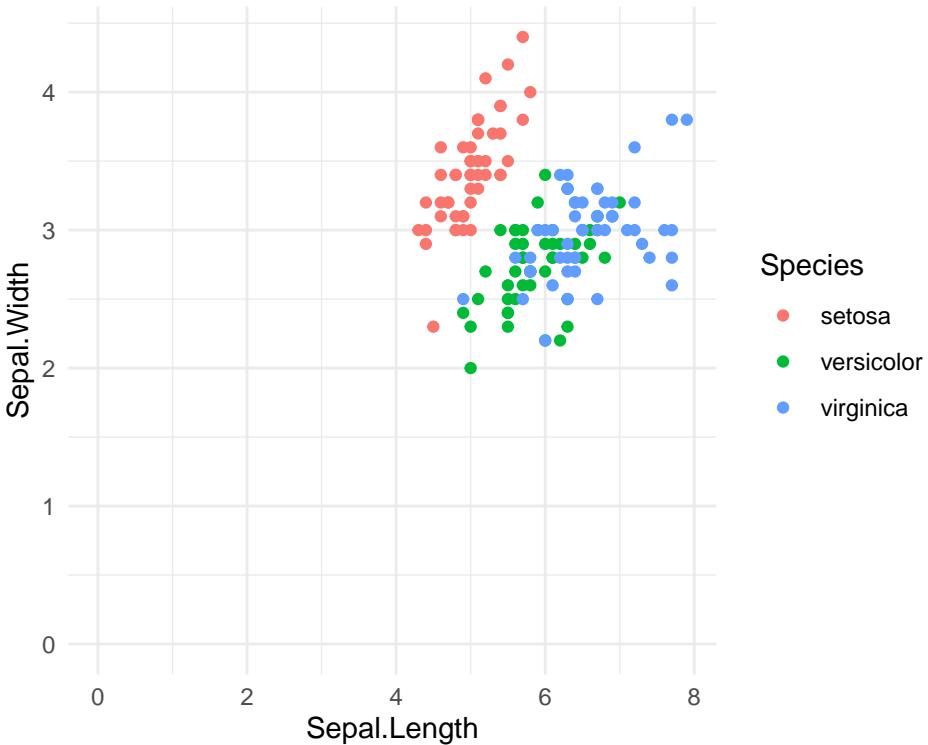
Og her anvender jeg funktionen `coord_cartesian()` med `xlim()` og `ylim()` indenfor til at zoome ind på et ønsket område på plottet.

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



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

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

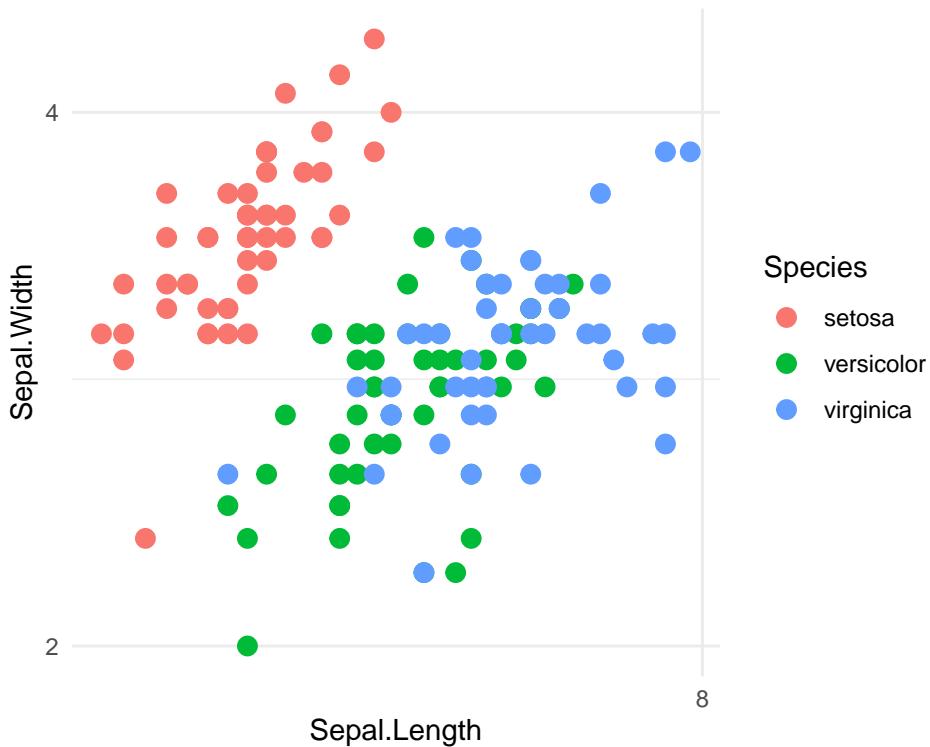


Det kan være brubart i situationer hvor man, eksempelvis har flere etiketter omkring punkterne i selve plottet, som bedre kan ses hvis man tillader lidt ekstra plads i plottets område.

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

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

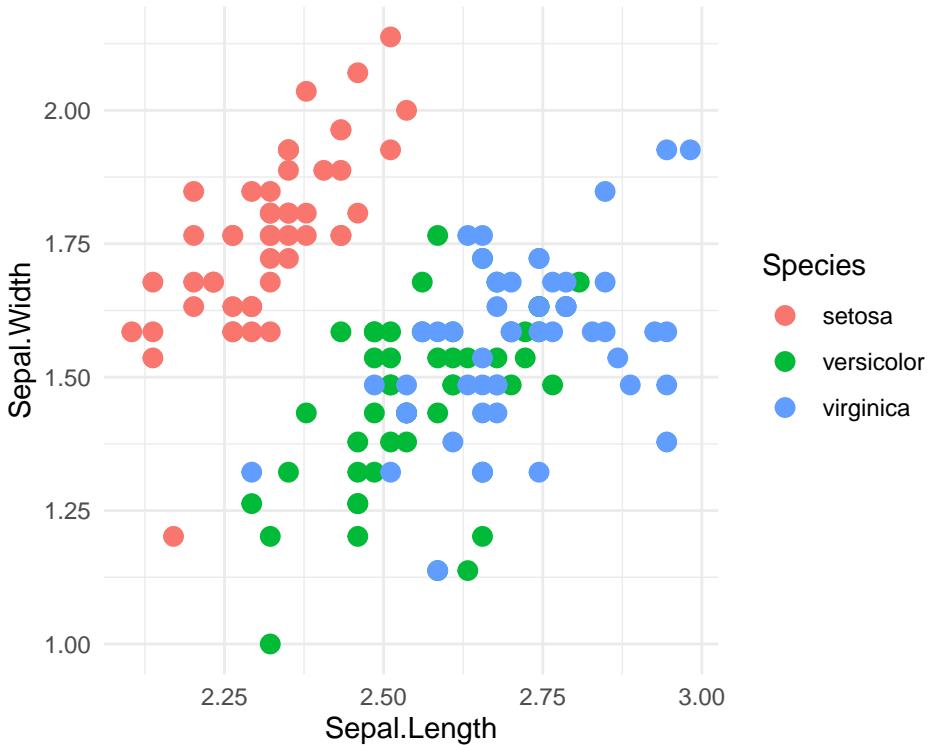
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point(size=3) +
  scale_x_continuous(trans = "log2") +
  scale_y_continuous(trans = "log2") +
  theme_minimal()
```



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

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

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

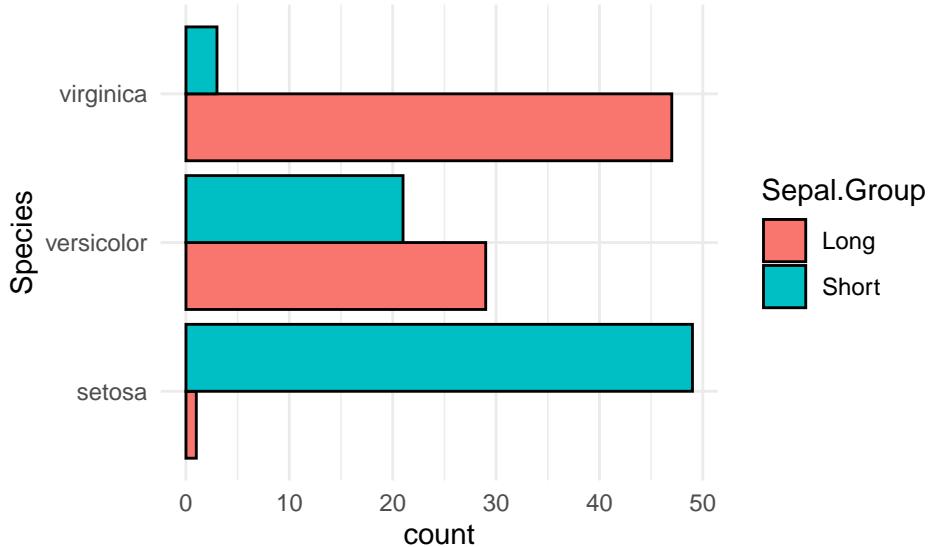


4.2.3 Flip coordinates (`coord_flip`)

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

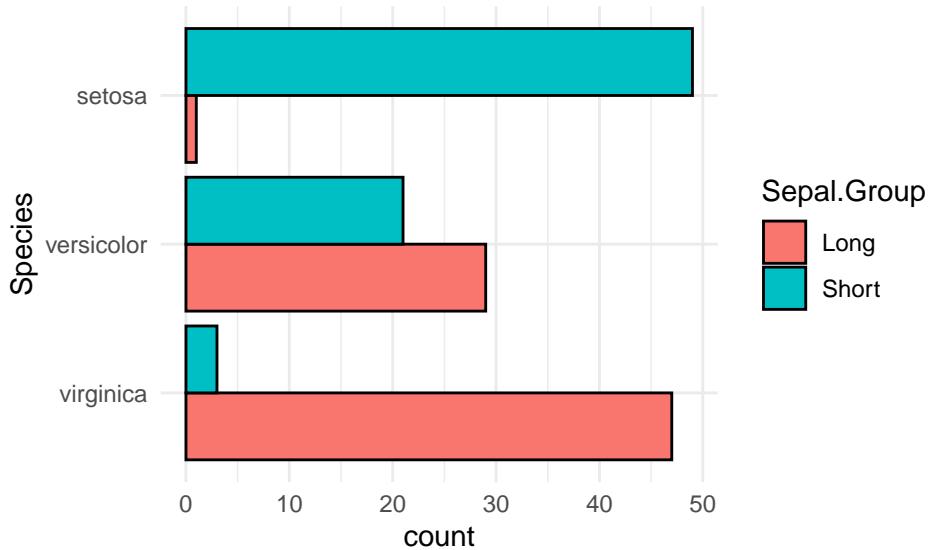
```
#Sepal.Group defineret som i går
iris$Sepal.Group <- ifelse(iris$Sepal.Length>mean(iris$Sepal.Length), "Long", "Short")

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



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

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



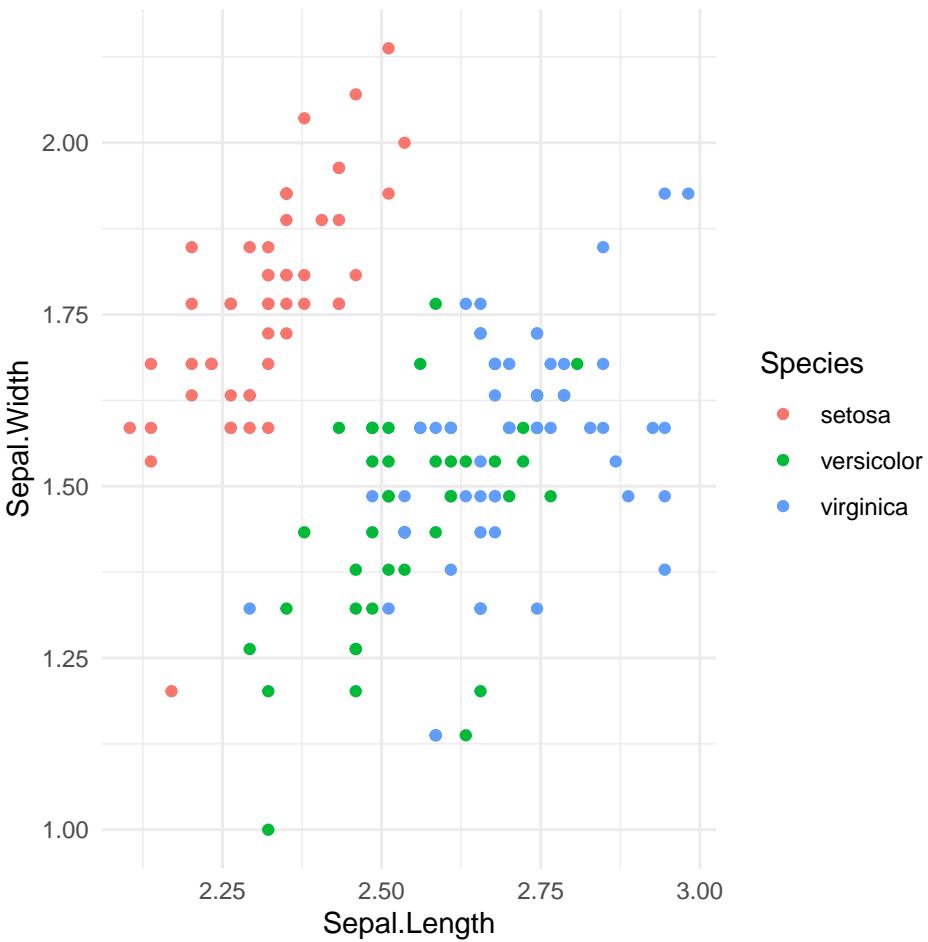
4.3 Mere om farver og punkt former

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

4.3.1 Automatisk farver

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

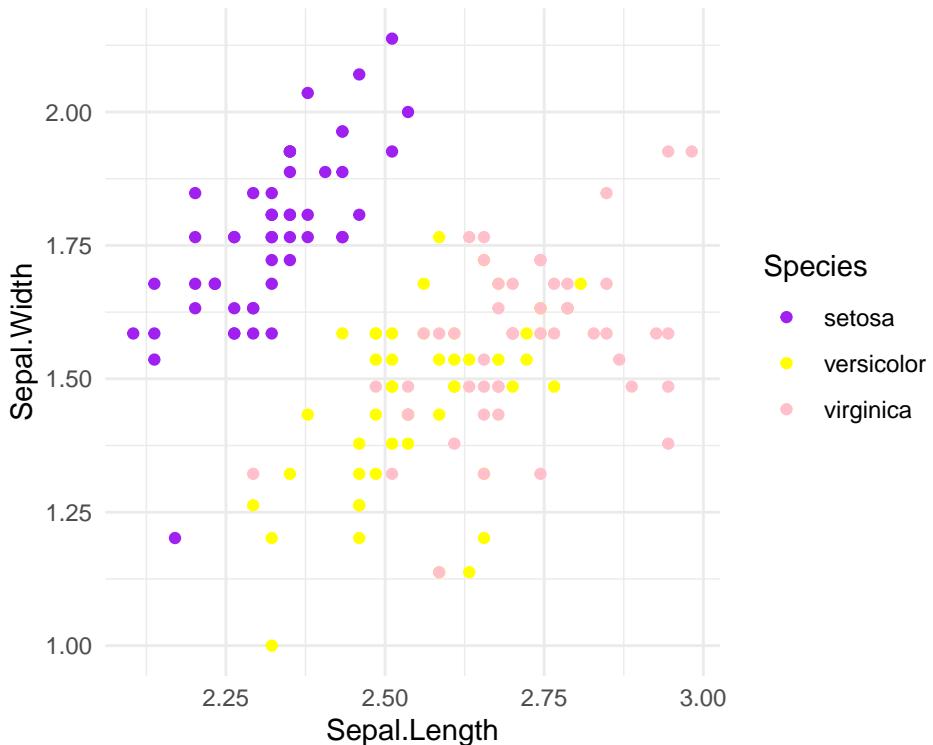
```
#automatisk løsning
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +
  geom_point() +
  theme_minimal()
```



4.3.2 Manuelle farver

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

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

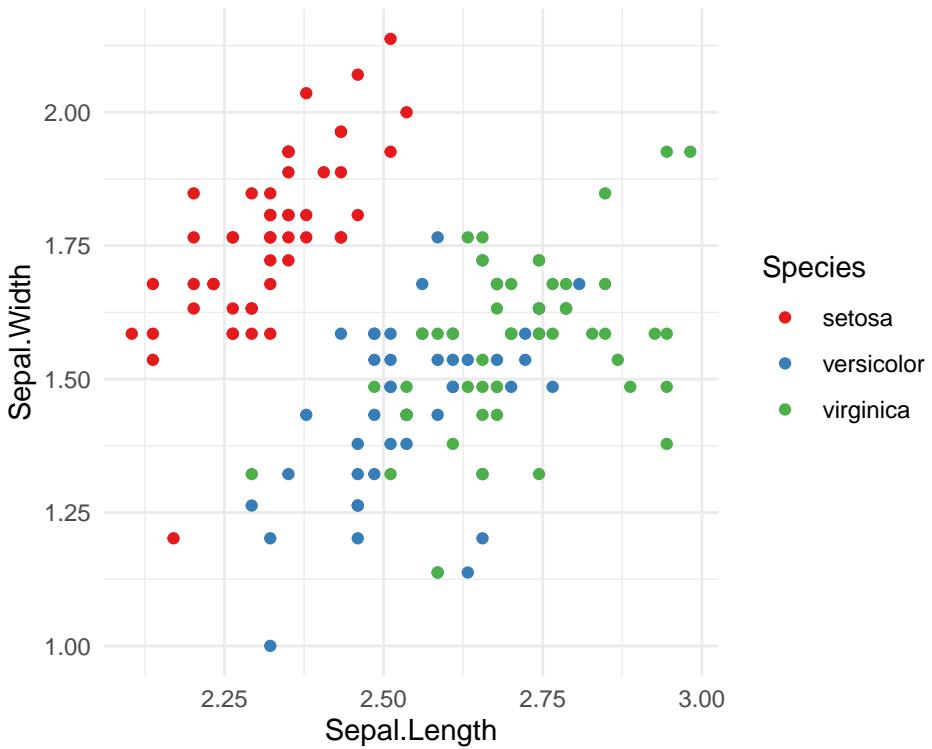


En faktastisk pakke er `RColorBrewer`. Pakken indeholder mange forskellige “colour palettes”, det vil sige grupper af farver, der passer godt med hinanden. Man kan således slippe for at selv samle et godt kombination der passer til plottet. Nogle af de colour palettes tager også i betragtning, hvis man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

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

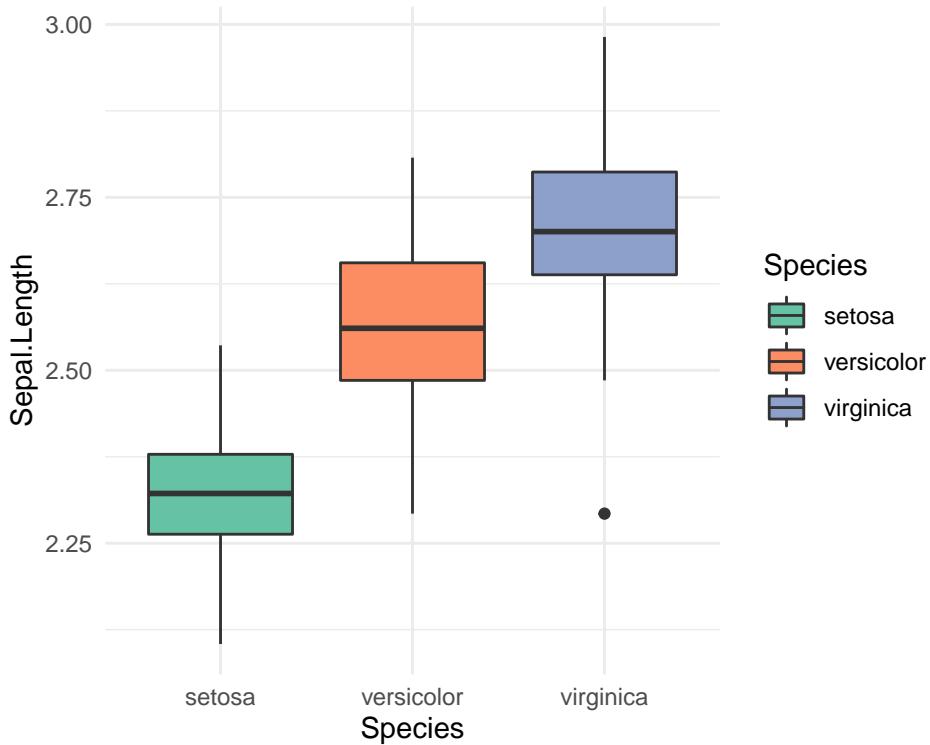
```
#install.packages("RColorBrewer")
library(RColorBrewer)

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



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

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



Her er en oversigt over de fire funktioner.

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

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

Farver i RColourBrewer

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

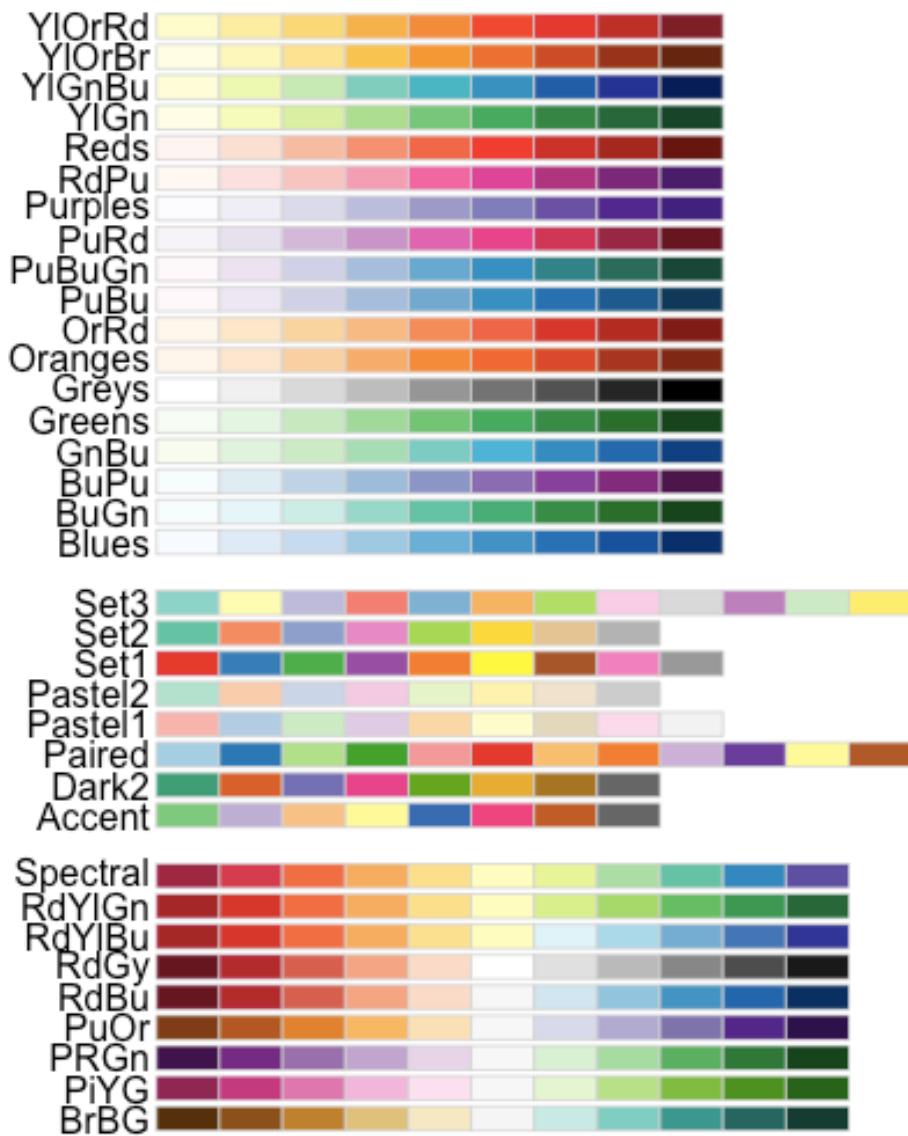
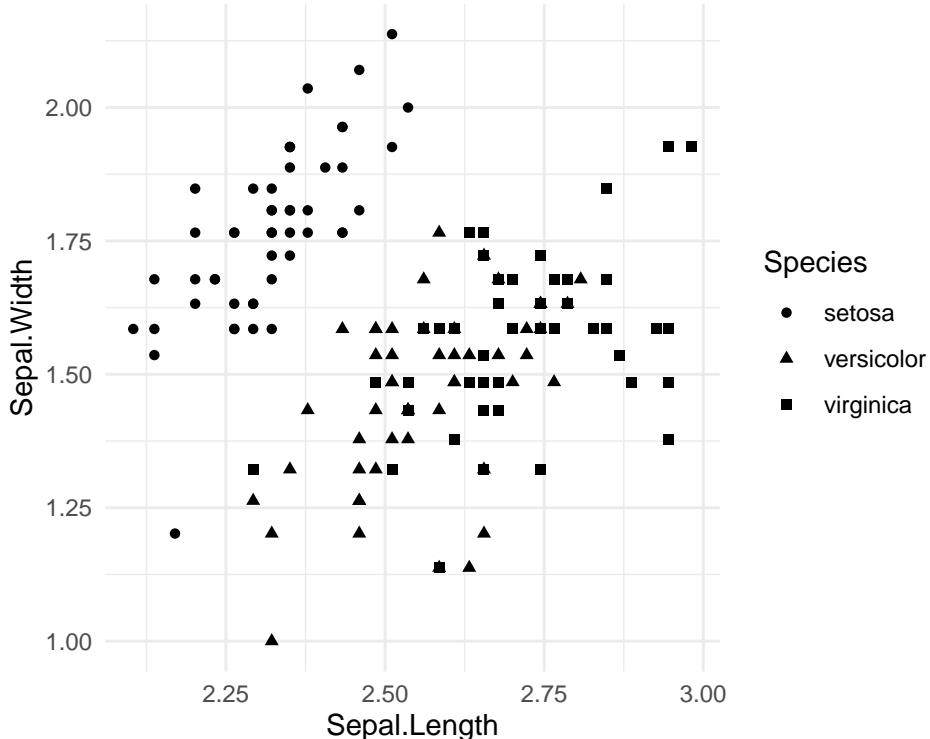


Figure 4.1: Mulige colour palettes tilgængelige i RColourBrewer

4.3.3 Punkt former

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

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

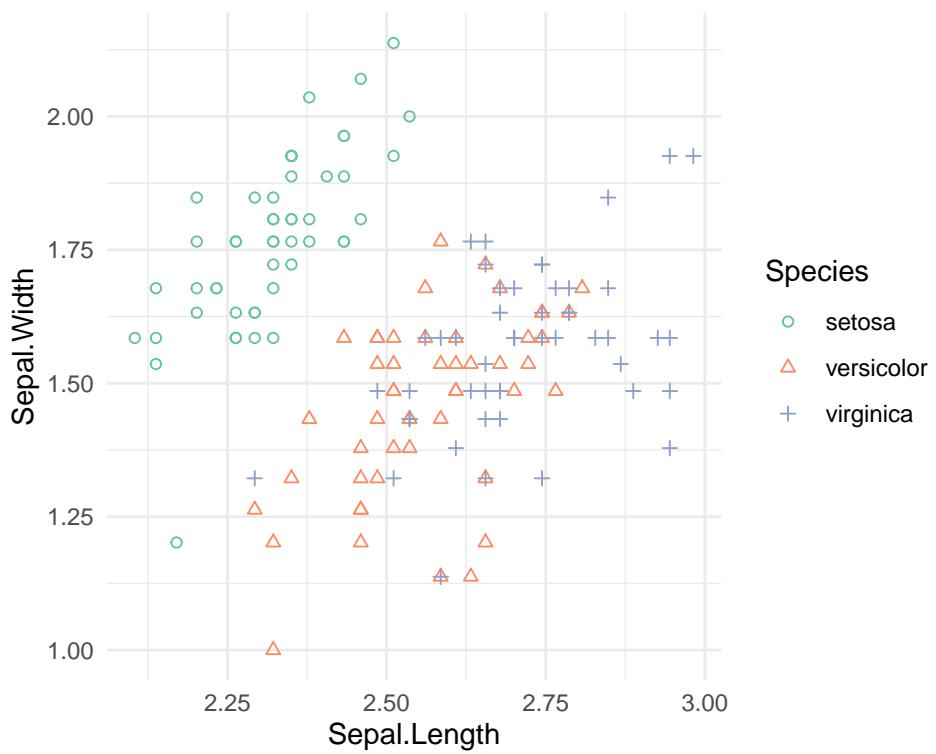


Så har jeg fået både en farve og en punkt form til hver art i variablen `Species`.

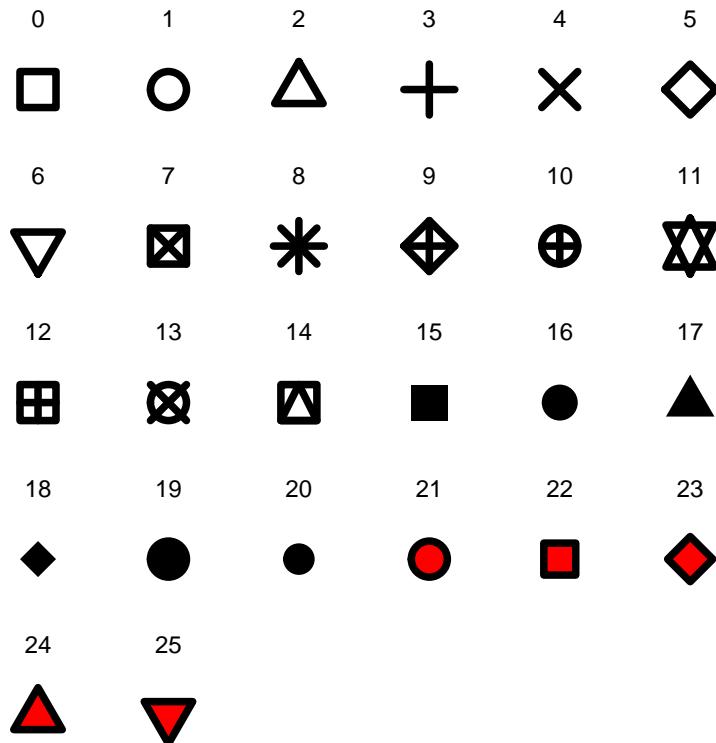
Sætte punkt form manuelt

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

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



Reference for punkt former



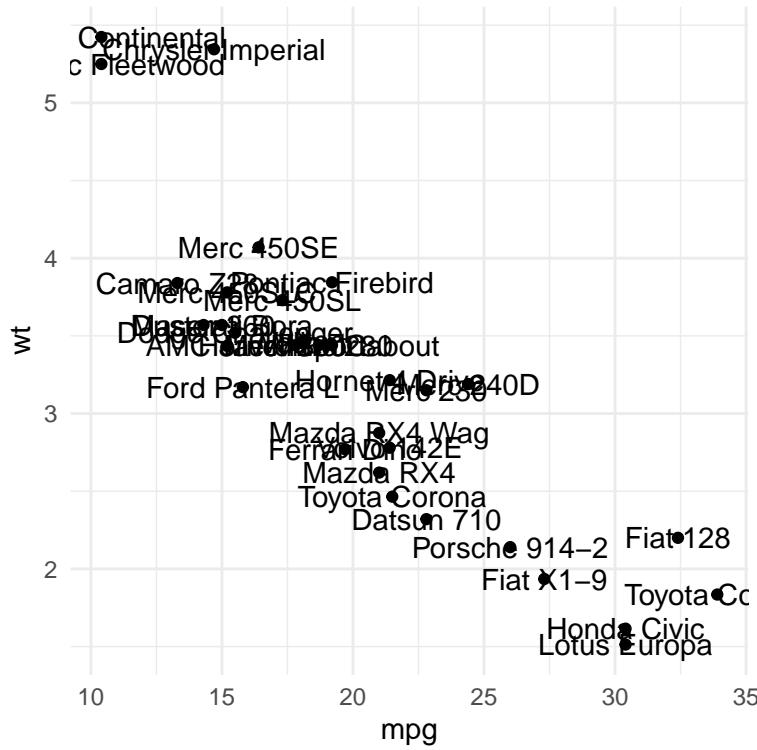
4.4 Annotations (geom_text)

4.4.1 Tilføje labeller direkte på plottet.

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

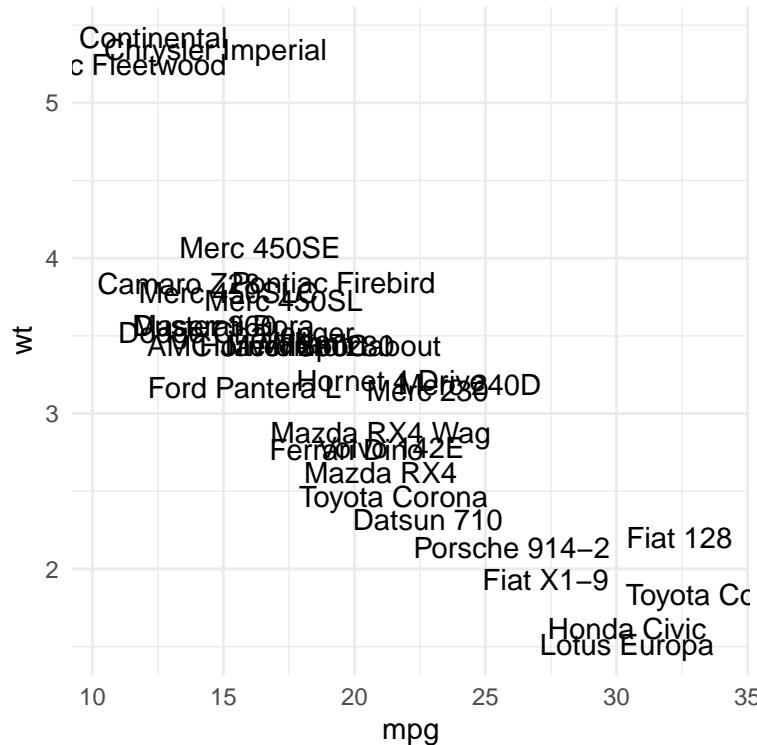
```
data(mtcars)

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 selve punkterne:

```
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`, som i følgende.

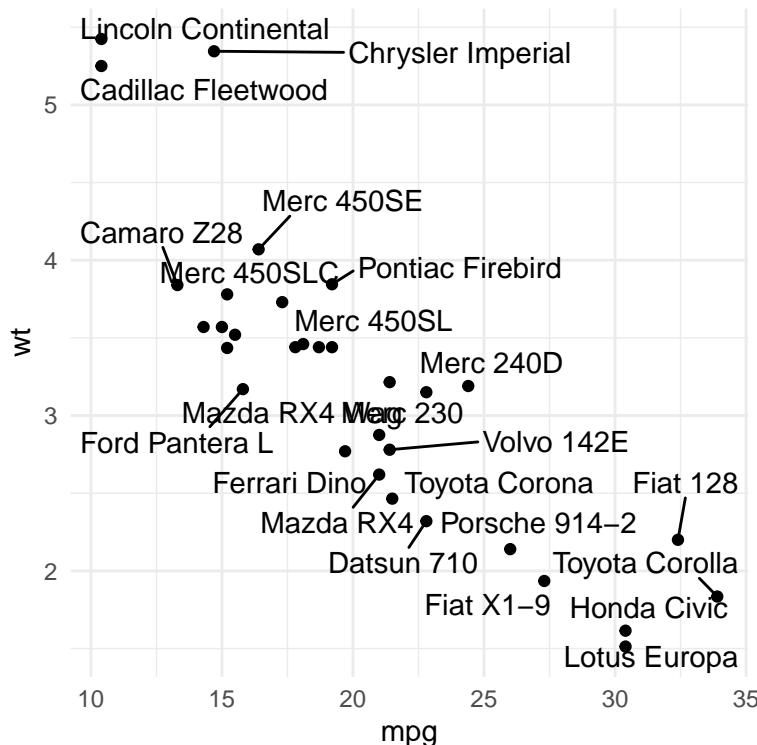
4.4.2 Pakken `ggrepel` for at tilføje tekst labeller

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

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

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

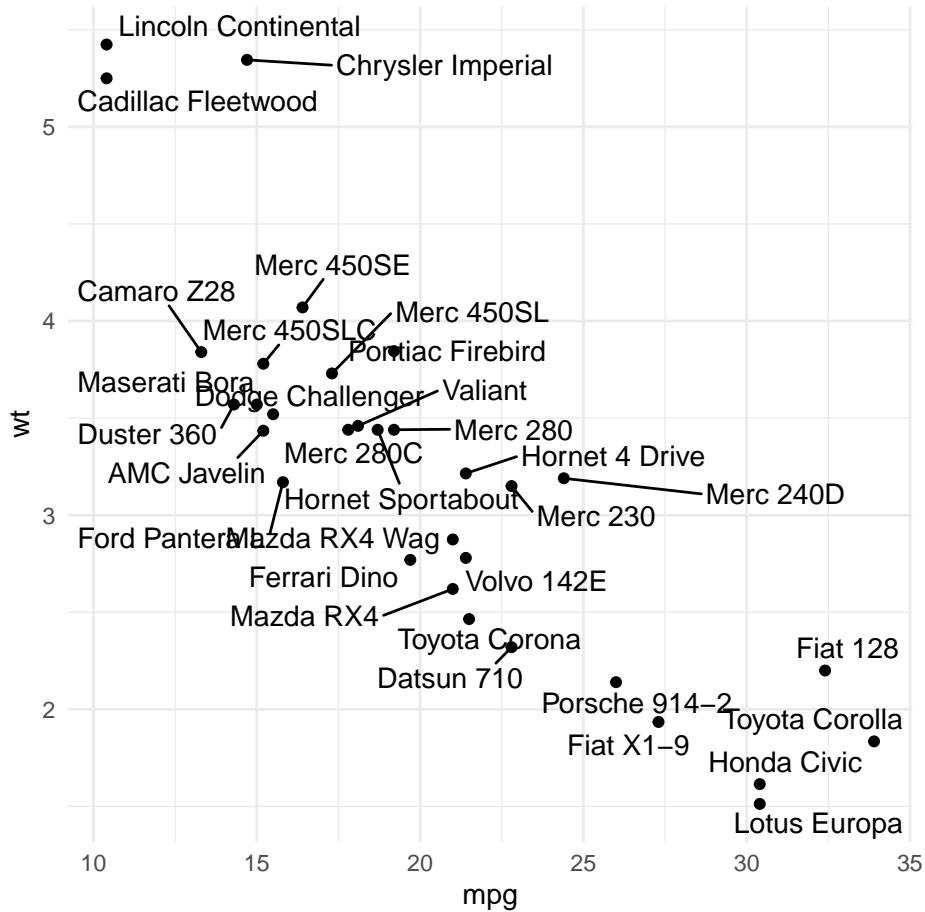
```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



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

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

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



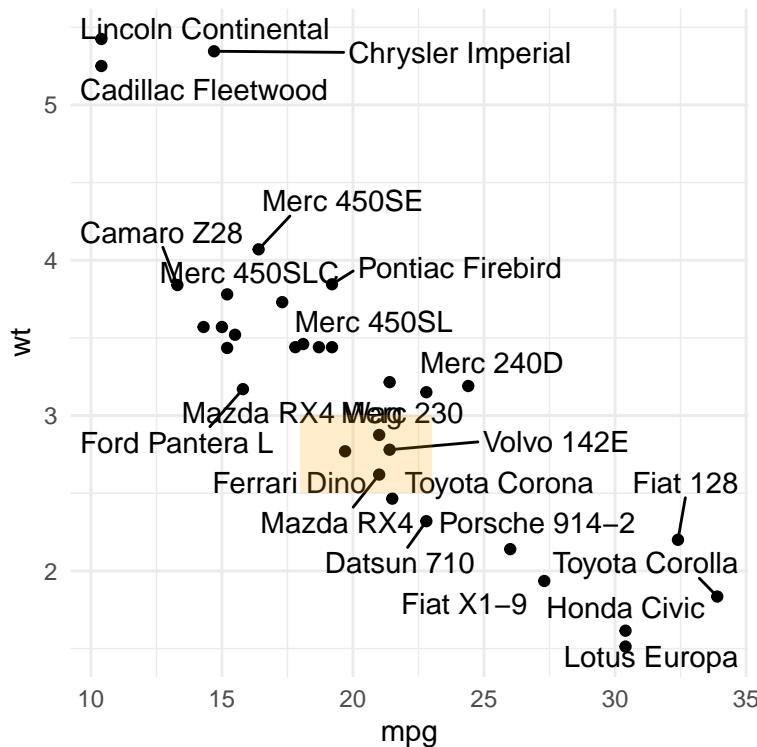
Så kan du se, at jeg ikke længere få en advarsel, og der tilhører tekst til alle punkterne nu.

4.4.3 Tilføje rektangler i regioner af interesse (annotate)

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

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

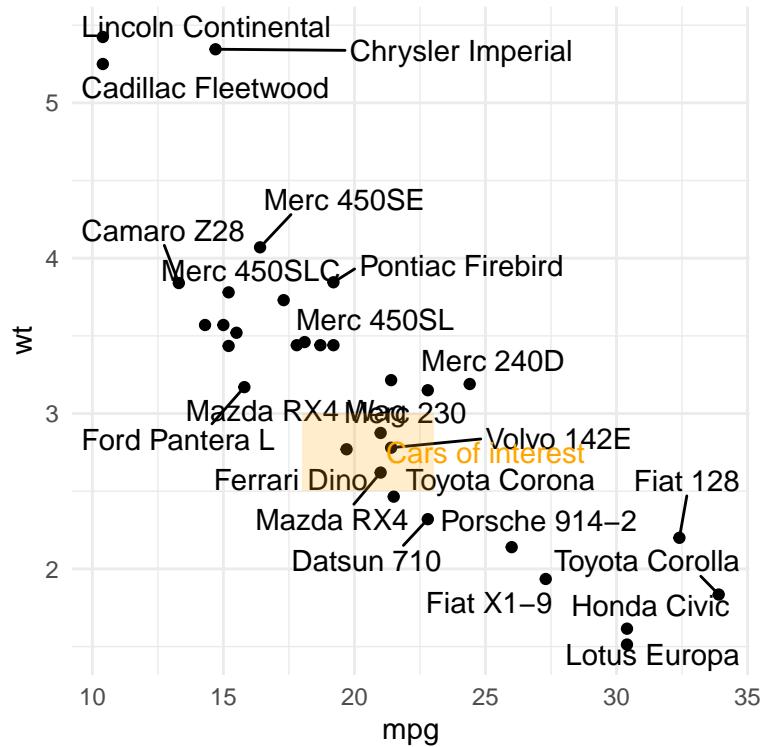
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



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

```
ggplot(mtcars,aes(x=mpg,y=wt)) +
  geom_point() +
  geom_text_repel(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()
```

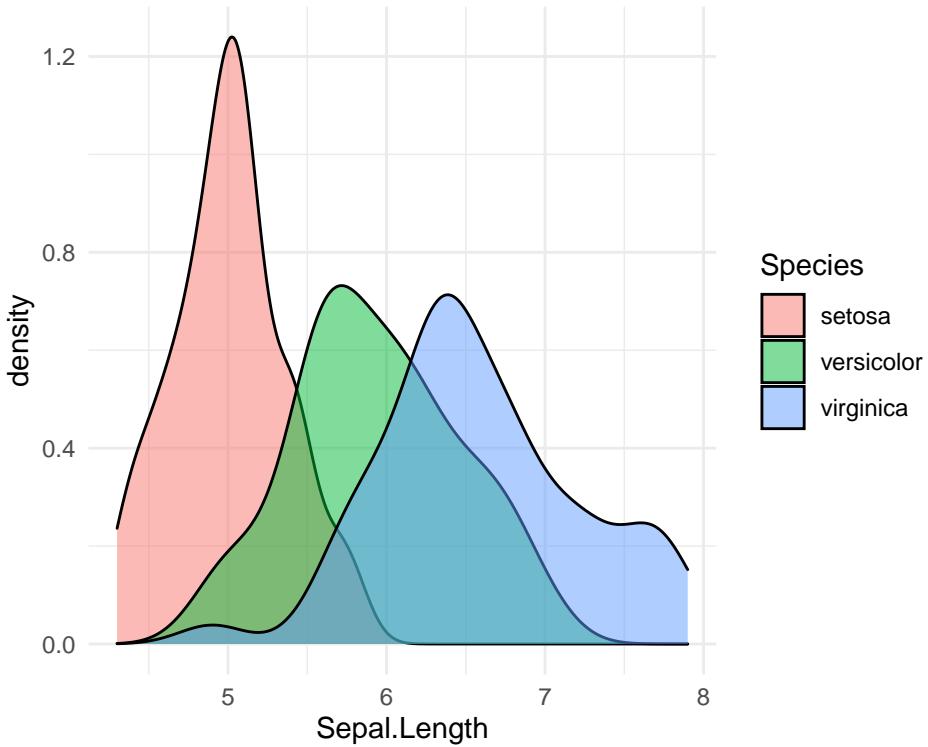
```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



4.5 Adskille plots med facets (facet_grid/facet_wrap)

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

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



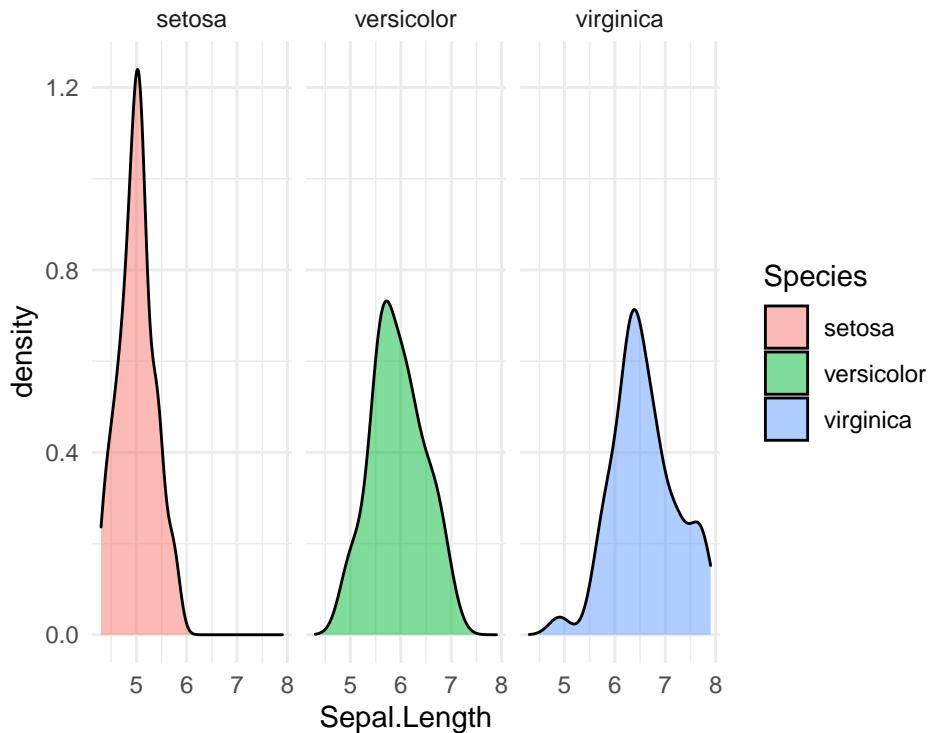
Med funktionen `facet_grid()` eller `facet_wrap()` bruger vi `~` (tilde) tegn til at angive hvordan vi gerne vil visualisere de forskellige plots - skal man opdele dem over rækker (variablerne venstre til `~`) eller over kolonner (variabler højre til `~`)?

#notrun

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

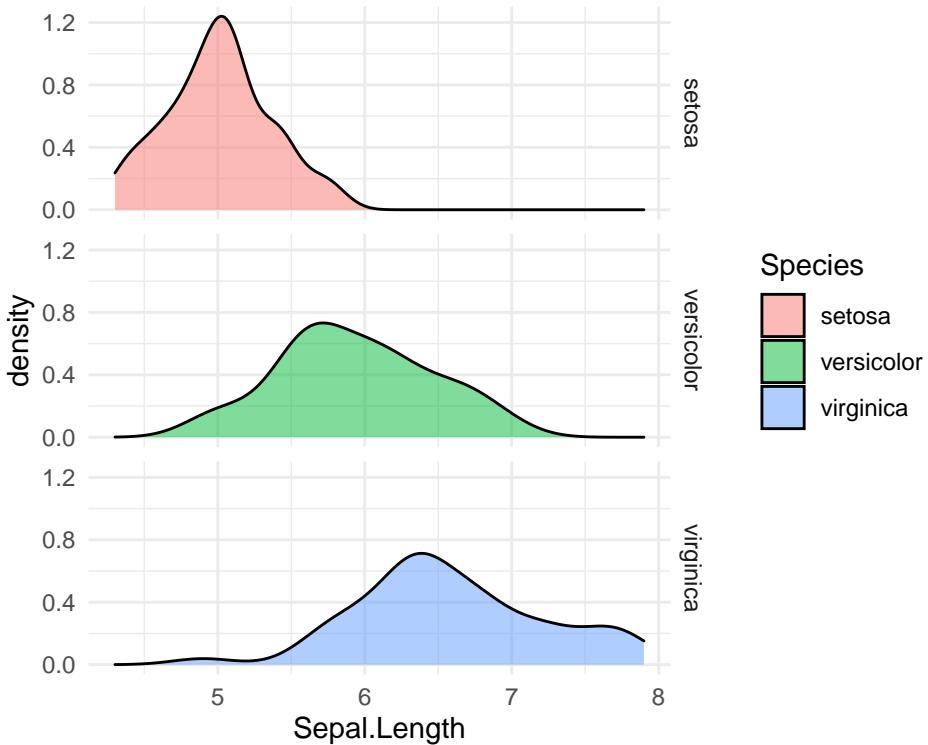
Ovenstående density plots af Sepal.Length kan adskilles på Species, således at man får tre plots med en kolon til hver af de tre arter:

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



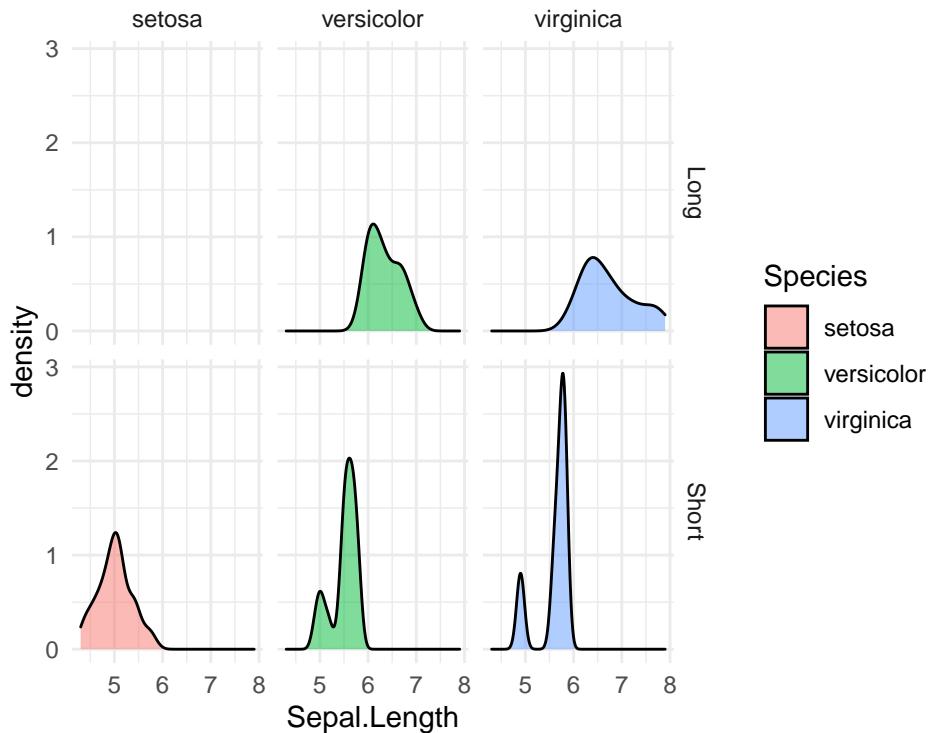
Man kan gøre det over rækkerne (man skal dog husk at bruge en “.” efter “~” for at betegne at man kun vil adskille plots over rækkerne, mens kan af en eller anden grund kan man dropper “.” hvis man kun vil adskiller over kolonner som i ovenstående).

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



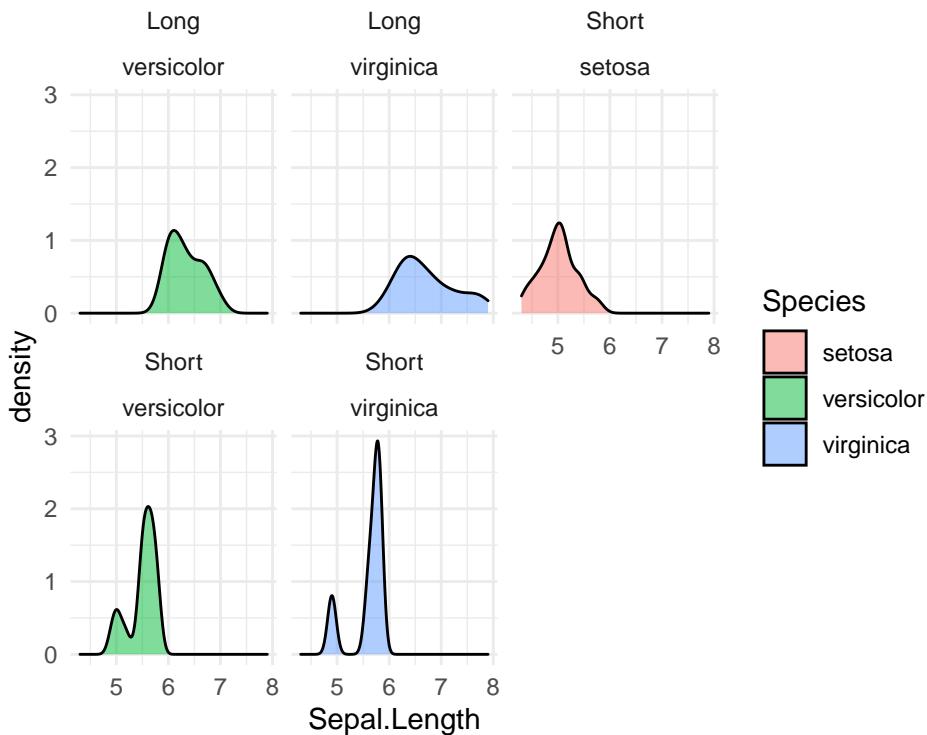
Her angives Sepal.Group~Species, der betyder, at plotterne bliver adskilt efter både Sepal.Group og Species - Sepal.Group over rækkerne, og Species over kolonnerne:

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



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

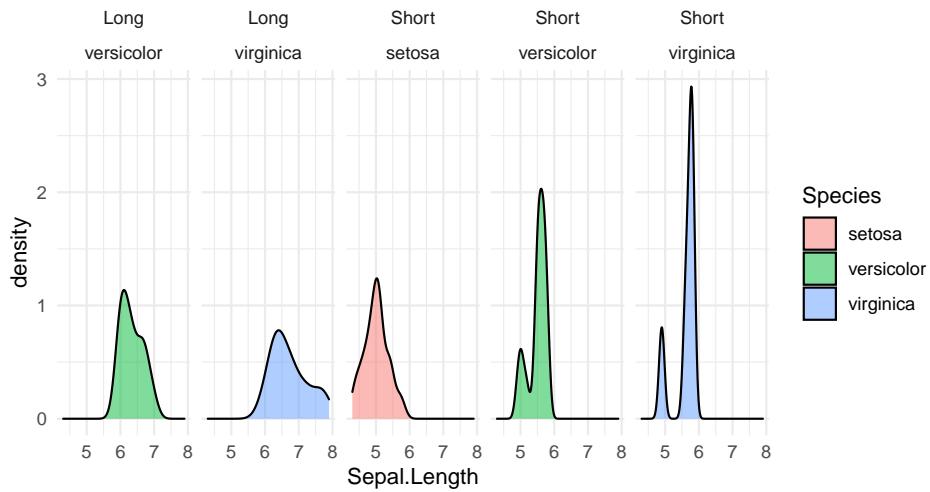
```
#same plot, replace facet_grid with facet_wrap
ggplot(iris,aes(x=Sepal.Length,fill=Species)) +
  geom_density(alpha=0.5) +
  facet_wrap(Sepal.Length~Species) +
  theme_minimal()
```



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

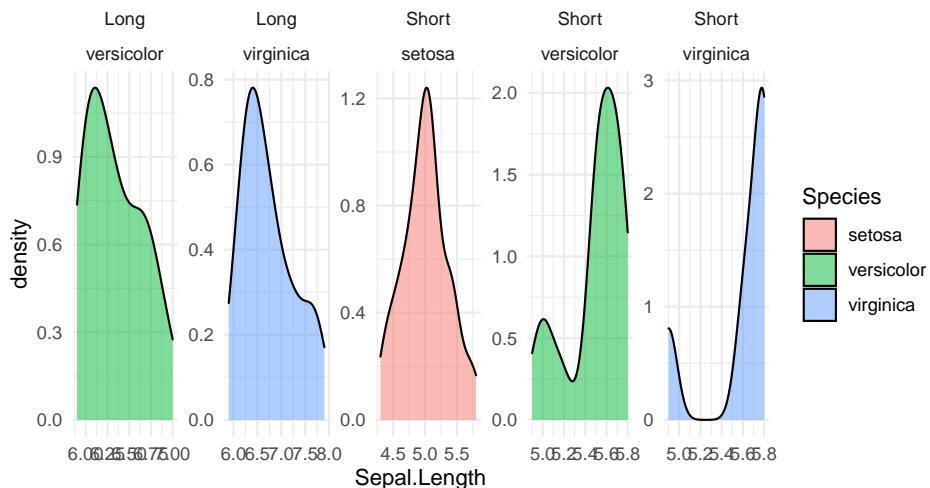
Med `facet_wrap()` kan man også fortælle at vi gerne vil have plotterne over 1 row (`nrow = 1` eller `ncol = 5`):

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



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

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



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

4.6 Gemme dit plot

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

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

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

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

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

4.7 Problemstillinger

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

Problem 2) (Factorer og plots)

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

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

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

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

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



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

```
library(ggplot2)
library(tidyverse)
head(penguins)
```

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

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

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

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

Problem 3) Manuelt farver og punkter

a) Lav en scatter plot med `ggplot`:

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

b) Lav følgende ændringer til det plot:

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

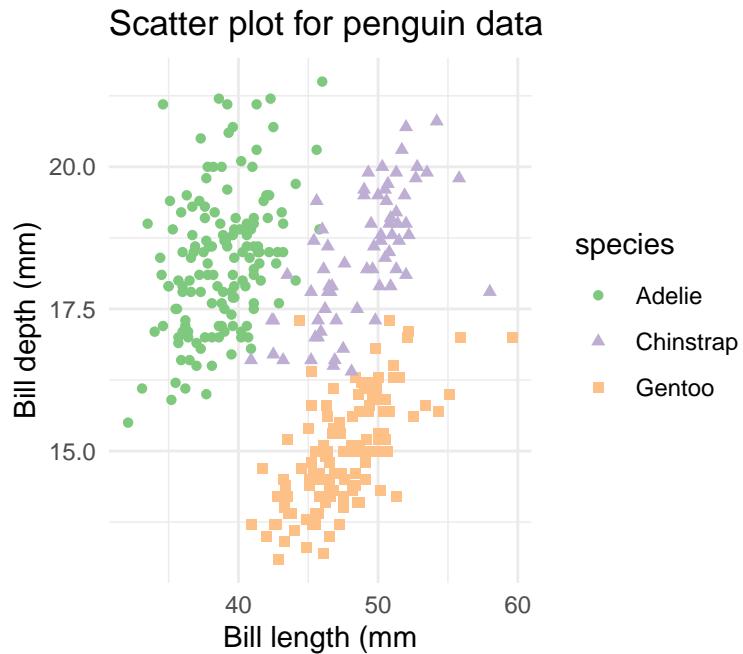


Figure 4.2: Min løsning

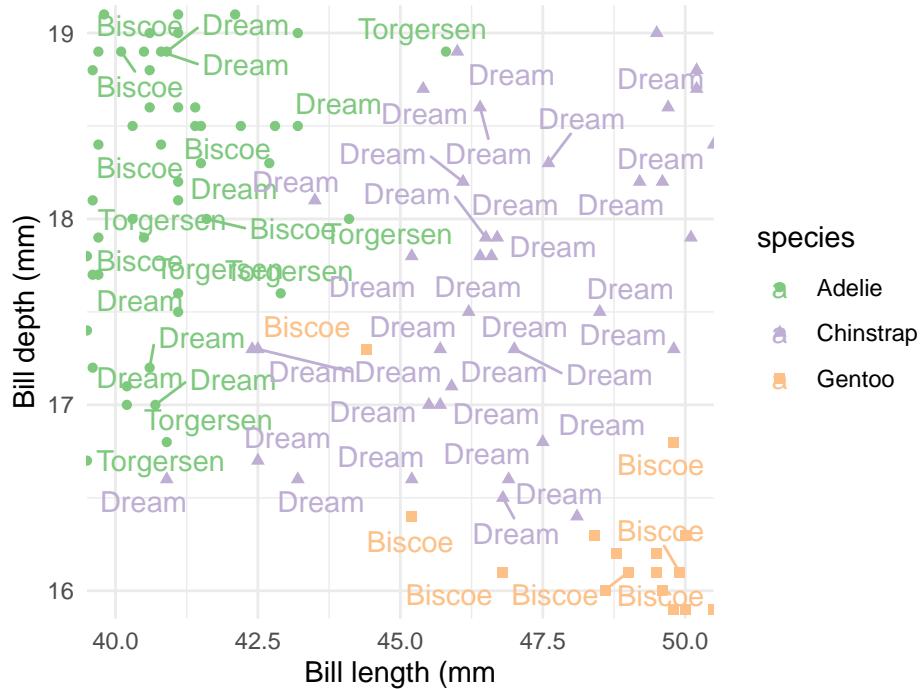
Problem 4) Coordinate systemer

Tag overstående scatter plot fra 3) og

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

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

Scatter plot for penguin data



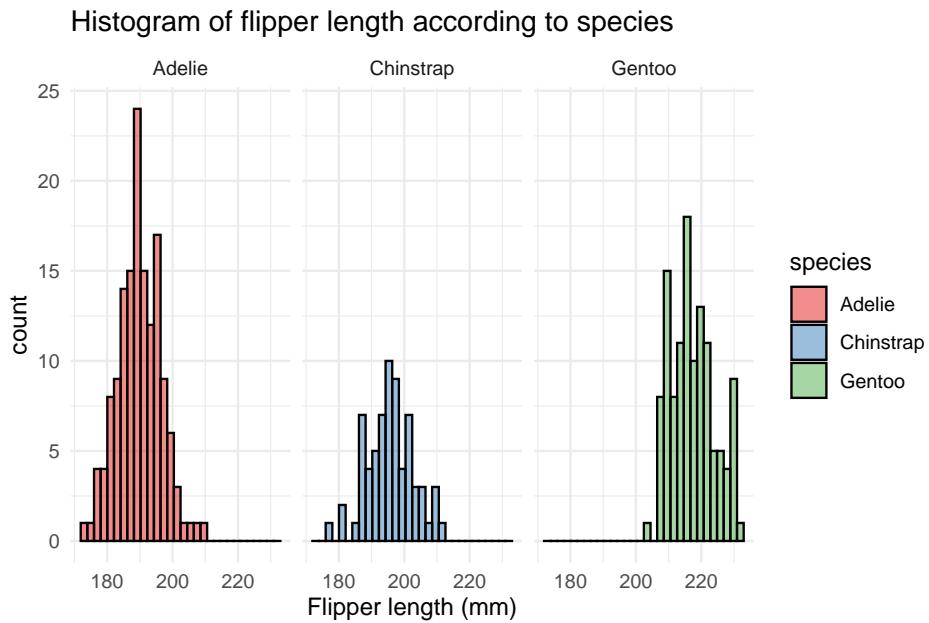
Problem 5) Histogram med facets

Lav en histogram:

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

Her er min løsning:

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



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

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

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

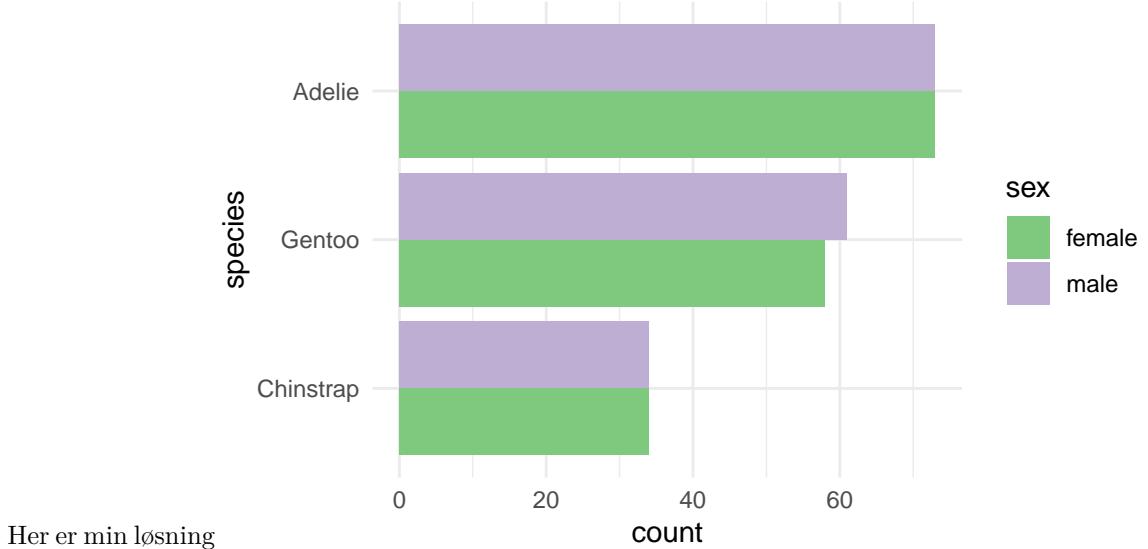
c) Kan du forklare hvorfor der er blank plotter i din grid? Eksperimenter med `facet_wrap` i stedet for `facet_grid`.

Problem 7) *Coordinate systemer*

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

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

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



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

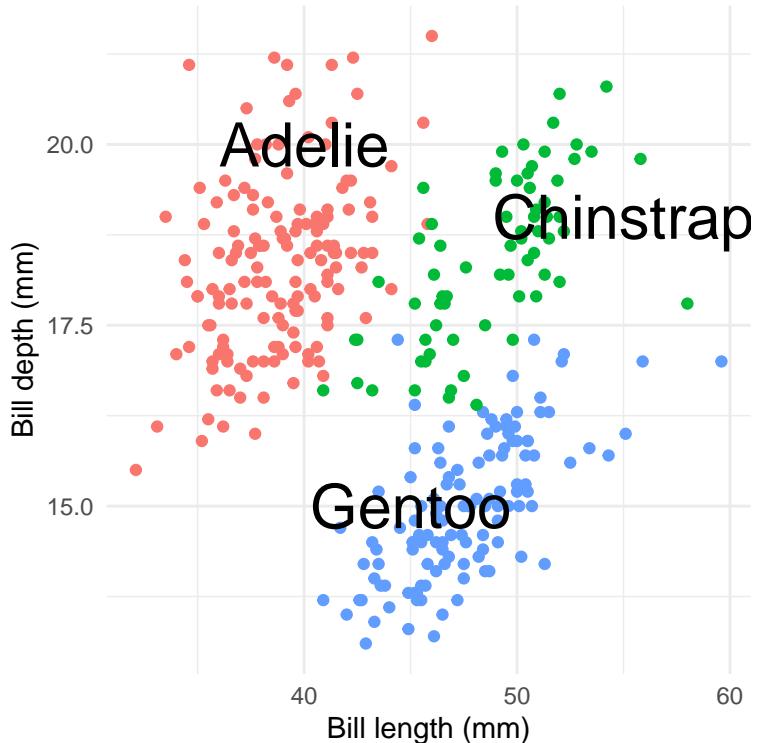
- Tilføj “jitter” punkter ovenpå boxplots.
- Specifier nogle farver manuelt for både boxes og punkterne (en farve til hver art)
- Giv det en hensigtsmæssig titel og nogle akser-labels
- Tilføj en ny variabel `island_binary` til `penguins`, som er “Biscoe” hvis `island` er ‘Biscoe’ og “not Biscoe” hvis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

Problem 9) *Annotations og linjer.*

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

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

Her er min løsning:



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

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

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

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

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

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

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

Her er min løsning:

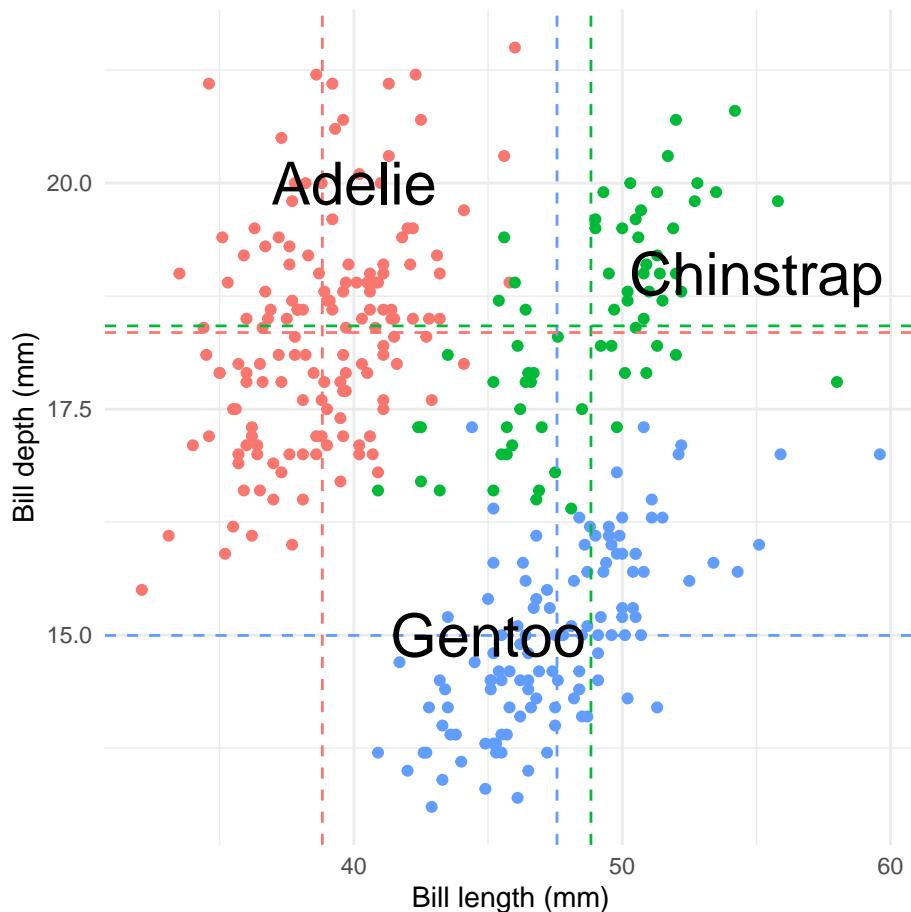


Figure 4.3: min løsning

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

4.8 Ekstra links

R Graphics cookbook

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