Datorlaboration 7

Josef Wilzén, Johan Alenlöv och Måns Magnusson

22 april 2024

Instruktioner

- Denna laboration ska göras i grupper om **två och två**. Det är viktigt för gruppindelningen att inte ändra grupper.
- En av ska vara **navigatör** och den andra **programmerar**. Navigatörens ansvar är att ha ett helhetsperspektiv över koden. Byt position var 30:e minut. **Båda** ska vara engagerade i koden.
- Det är tillåtet att diskutera med andra grupper, men att plagiera eller skriva kod åt varandra är **inte tillåtet**. Det är alltså **inte** tillåtet att titta på andra gruppers lösningar på inlämningsuppgifterna.
- Använd gärna Teams för att ställa frågor. Det finns olika kanaler:
  - **Questions**: Skriv era frågor här. Svar kommer att ges öppet direkt i kanalen. Publicera inte kod till inlämningsuppgifter här (andra kan då se det). Det går bra att skriva frågor om inlämningsuppgifter här så länge ni inte inkluderar kod med lösningar till dessa uppgifter. Det går bra att publicera kod till övningsuppgifter här.
  - **Raise_your_hand**: Skriv här om ni vill ha hjälp men inte ställa er fråga öppet. Skriv något i stil med "Jag vill ha hjälp". Då kommer en lärare att kontaka er när de har tid (i chatten på Teams). Vill flera ha hjälp så bildar de olika kommentarerna en kö, och hjälp kommer att ges i ordning efter kön. En "tumme upp" på kommentaren innebär att läraren har börjat hjälpa den aktuella studenten. Ett "hjärta" på kommentaren innebär att läraren har hjälpt klart studenten.
- Använd inte `å`, `ä` eller `ö` i variabel- eller funktionsnamn.
- Utgå från laborationsmallen, som går att ladda ned **här** (obs ny mall jämfört med tidigare veckor), när du gör inlämningsuppgifterna. Spara denna som `labb[no]_grupp[no].R` , t.ex. labb5_grupp01.R om det är laboration 5 och ni är grupp 01. Ta inte med hakparenteser i filnamnet. Denna fil ska **inte** innehålla något annat än de aktuella funktionerna, namn- och ID-variabler och ev. kommentarer. Alltså **inga** andra variabler, funktionsanrop för att testa inlämningsuppgifterna eller anrop till markmyassignment-funktioner.
- Precis innan inlämning på Lisam, döp om er R-fil till en **`.txt`** fil, detta görs för att kunna skicka in filen till Ouriginal för plagieringskontroll. Exempel: `labb5_grupp01.R` blir då `labb5_grupp01.txt` Ladda upp den filen (som slutar på .txt) på Lisam under rätt inlämning innan deadline.
- Laborationen består av två delar:
  - Datorlaborationen (= övningsuppgifter)
  - Inlämningsuppgifter
- I laborationen finns det extrauppgifter markerade med *. Dessa kan hoppas över.
- Deadline för laboration framgår på LISAM
- **Tips!** Använd "fusklapparna" som finns här. Dessa kommer ni också få ha med på tentan.

Innehåll

Del I

Datorlaboration

Kapitel 1

Introduktion till ggplot2

Paketet `ggplot2` skiljer sig från den grundläggande grafikfunktionaliteten som finns implementerat i R. Paketet bygger på vad som brukar kallas "The grammar of graphics" (därav `gg` i `ggplot2`) och är ett försök till ett formellt språk för att uttrycka hur en visualisering ska se ut. Mer teori bakom denna gramatik går att finna i [3, 2, 1] och är grunden bakom exempelvis SPSS grafiksystem. Genom att ha en grundläggande förståelse för denna grammatik kan vi enkelt och snabbt skapa mycket komplicerade visualiseringar.

I R:s basgrafiksystem kunde man se grafikfunktionaliteten lite som ett papper vi ritar på. Vi ritar initialt upp vår graf och kan sedan lägga till/rita "ovanpå" det befintliga pappret. `ggplot` är annorlunda. Med `ggplot` skapar vi ett grafikobjekt och vi kan lägga till bit för bit av grafen för att när vi sedan är klar med vår graf visualisera den. Det gör det enklare att bygga upp komplicerade grafer utan att behöva använda särskilt mycket kod.

Här finns en bra katalog över de flesta graferna i `ggplot2`.

1.1 Grunden i ggplot2

Till skillnad från basgrafiken utgår ggplot **alltid** från en `data.frame`. Baserat på denna data.frame skapas sedan grafen med två huvudsakliga komponenter:

- `aes` (aestetic) som handlar om utseendet på grafen, färger, former m.m.
- `geom` (geometrics) som beskriver vilken typ av graf vi vill ha (bar, line, points)

Vi lägger sedan till dessa komponenter till vår graf och `data.frame`.

När det gäller de olika geometriska argumenten, d.v.s. de olika typer av grafer som går att skapa, finns det ett mycket stor antal vi kan använda oss av. Några exempel är:

geom	Beskrivning
geom_point	Scatterplot
geom_line	Line graph
geom_bar	Barplot
geom_boxplot	Boxplot
geom_histogram	Histogram

Exakt hur dessa geometriska figurer ska se ut styrs sedan med `aes`. Nedan finns några exempel:

aes	Beskrivning
x	x-axel
y	y-axel
size	storlek
col	färg
shape	form

De enskilda geometriska figurerna kan i sin tur ha ett antal olika aestetics. Nedan finns lite exempel.

geom	Specifika aestetics
geom_points	point shape, point size
geom_line	linetype, line size
geom_bar	y min, y max, fill color, outline color

Med dessa verktyg har vi en grund för att bygga upp ett mycket stort antal visualiseringar.

1.1.1 Skapa en ggplot (linje eller scatter)

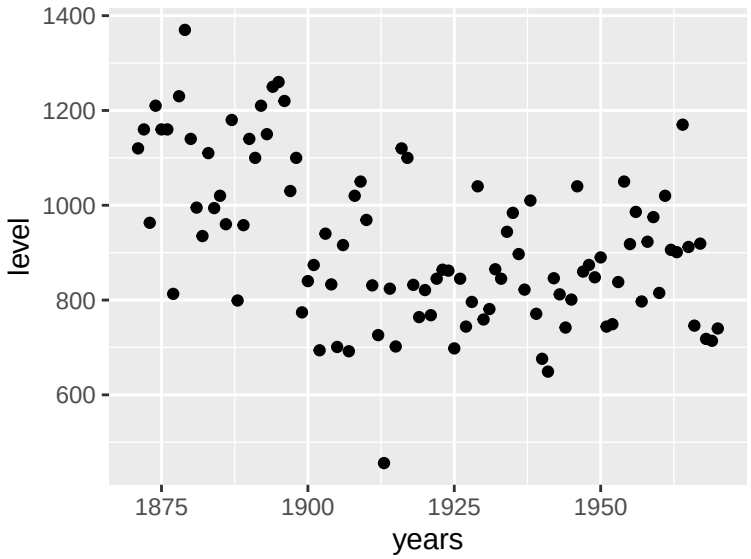
1. Vi börjar med att läsa in the datamaterialet `Nile`.

```
library(ggplot2)

data(Nile)
Nile <- data.frame(level=as.vector(Nile))
Nile$years <- 1871:1970
```

2. För att skapa en `ggplot` börjar vi med att skapa grunden för plotten med funktionen `ggplot()`. Nedan är ett exempel på att skapa en `ggplot` med `Nile`, sedan lägger vi till att `x` ska utgöras av variabeln `years` och `level`. Sedan lägger vi till att plotten ska utgöras av punkter. Vi sparar grafen som variabeln `p`. För att skapa grafen tittar vi bara på `p`:

```
p <- ggplot(data=Nile, aes(x=years, y=level)) + geom_point()
# går även att skriva:
# ggplot(data=Nile) + aes(x=years, y=level) + geom_point()
p
```




```
# alt: print(p)
```

3. Vill vi ändra till en linjegraf (vilket känns bättre) här byter vi bara ut geometrin:

```
p <- ggplot(data=Nile,aes(x=years, y=level))  + geom_line()
```

4. Vill vi lägga till både punkter och linjer i samma graf kan vi bara ta `p` och lägga till punkter. Här blir det tydligt hur vi i `ggplot` lägger till lager på lager och sedan producerar en visualisering:

```
p <- p + geom_point()
```

5. På samma sätt kan vi också lägga till rubriker och axelettiketter:

```
p <- p + xlab("Years") + ylab("Water level") + ggtitle("Nile series")
```

6. Eftersom vi kan spara ggplot-objekt i variabler så kan spara ett antal plottar i t.ex. en lista och sen plotta den senare med print().

```
p1 <- ggplot(data=Nile,aes(x=years, y=level)) + geom_line() + ggtitle("linje")
p2 <- ggplot(data=Nile,aes(x=years, y=level)) + geom_point() + ggtitle("punkter")

class(p1)

mina_plottar<-list(p1=p1,p2)

# rensa alla plottar i plot-fönstret och kör sedan koden nedan:
for(i in 1:2){
  print(mina_plottar[[i]])
}
```

1.1.2 Enklare modifikationer av ett `ggplot`-objekt

1. Vill vi ändra färg och form på olika delar i en graf behöver vi ange exakt var dessa förändringar ska ske.

```
p <- ggplot(data=Nile,aes(x=years, y=level)) + geom_line(color="red", linewidth=3)+
geom_point(color="blue", size=4)
```

2. Det finns många färger att välja på. Se här och här för listor över tillgängliga färger. Ändra plotten ovan med andra färger från dessa listor.

3. Om vi vill att färgen på någon del av grafen ska bero av en annan variabel så behöver vi ändra `aes` i den delen som berörs. T.ex. om vi vill att färgen på punkterna ska bero på en kategorisk variabel `cat_var`, så anger vi `geom_line(aes(color=cat_var))`. Notera skillnaden om vi vill ha en ”fix” färg, då vi anger `geom_line(color="red")` utan att anävnda `aes`. Nu ska färgen på vår graf ändras, skapa först en ny faktorvariabel.

```
Nile$period <- "- 1900"
Nile$period[Nile$years >= 1900] <- "1900 - 1945"
Nile$period[Nile$years > 1945] <- "1945 + "
Nile$period <- as.factor(Nile$period)
```

4. Nu låter vi färgen på linjen bero på variabeln `period` genom att ändra `aes` på lämpligt ställe.

```
p <- ggplot(data=Nile,aes(x=years, y=level)) + geom_line(aes(color=period)) + geom_point()
```

5. Vill vi istället modifiera punkterna lägger vi till det i `geom_point()`.

```
p <- ggplot(data=Nile,aes(x=years, y=level))  + geom_line() + geom_point(aes(color=period))
```

6. Vill vi lägga det i hela grafen kan vi lägga till färgen i den huvudsakliga styrningen av aestetics i grafen.

```
p <- ggplot(data=Nile,aes(x=years, y=level, color=period)) + geom_line() + geom_point()
```

7. Baserat på graferna ovan pröva att göra följande ”fixa” förändringar. Alltså om ni ändrar typ av linje så ska typen vara samma för all data i grafen.
   (a) Ändra typ av linje i grafen [**Tips!** `linetype`, kolla här]
   (b) Ändra typ av punkter i grafen [**Tips!** `shape`, kolla här]
   (c) Gör punkterna transparaenta [**Tips!** `alpha`]
8. Gå nu till dokumentationen för `geom_point()` och `geom_line()`. Läs under rubriken ”Aesthetics” och ta reda på vilka mer saker som ni kan styra genom att ändra aes. Testa nu att låta följande saker bero på variabeln `period`.
   (a) Typen av linje i grafen
   (b) Typen av punkter i grafen
   (c) Punkternas storlek
9. Lägga till räta linjer: Ibland vill vi lägga till räta linjer till plottar, t.ex. som referenslinjer. Kör koden nedan och se vad som händer.

```
x<-scale(1:100)
set.seed(4939)
y<-2+4*x+rnorm(n = 100)
A<-data.frame(x=x,y=y)
p<-ggplot(data = A,mapping = aes(x=x,y=y))+geom_point()

p+geom_hline(yintercept=0,linetype=3) p+geom_hline(yintercept=-5:5,linetype=3)

p+geom_vline(xintercept=c(-1,0,1),linetype=3)

p+geom_abline(slope = 4,intercept = 2,linetype=2,col="blue")

p+geom_abline(slope = 4,intercept =2,linetype=1,linewidth=1.5,col="red")
+ geom_vline(xintercept=0,linetype=3)
+ geom_hline(yintercept=0,linetype=3)
```

10. Vi kan även lägga till regressionslinjer med hjälp av `geom_smooth()`.
    (a) Linjär regression:

```r
x<-scale(1:100)
set.seed(4939)
y<-2+4*x+rnorm(n = 100,sd=1)
A<-data.frame(x=x,y=y)

# anpassade värden från linjär regression:
p+geom_smooth(method = lm, se = FALSE)

# med osäkerhetsband:
p+geom_smooth(method = lm, se = TRUE)
```

(b) Icke-linjär regression:

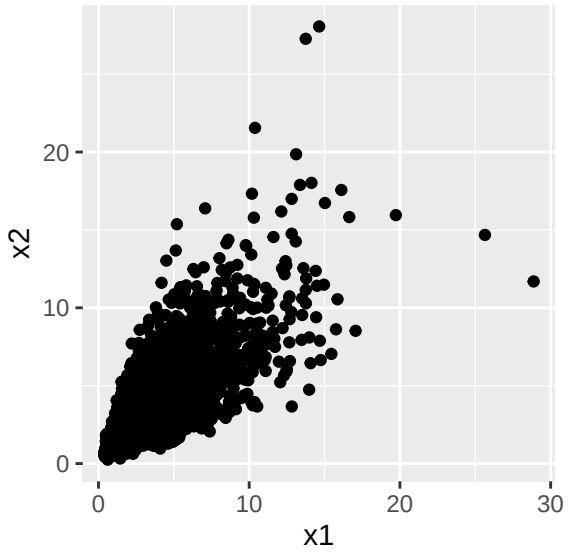
```r
# linjär regression:
ggplot(mpg, aes(displ, hwy)) +geom_point() +geom_smooth(method = lm,se=TRUE)

# icke-linjär regression:
ggplot(mpg, aes(displ, hwy)) +geom_point() +geom_smooth(method = loess,se=TRUE)

# span = styr hur linjen blir
ggplot(mpg, aes(displ, hwy)) +geom_point() +geom_smooth(span=0.20)
ggplot(mpg, aes(displ, hwy)) +geom_point() +geom_smooth(span=0.35)
ggplot(mpg, aes(displ, hwy)) +geom_point() +geom_smooth(span=0.8)
```

11. Ibland när vi arbeter med scatter plots så har vi så mycket observationer/data att det är svårt att tolka grafen då en massa punkter ligger på varandra, då har vi problemet som kallas *overplotting*.

```r
library(MASS)
no_obs<-5000
# skapa lite data:
set.seed(26)
X<-mvrnorm(n = no_obs,mu = c(1,1),Sigma = matrix(c(1,0.8,0.8,1),2,2)/2.5)
df<-data.frame(x1=exp(X[,1]),x2=exp(X[,2]))
ggplot(data = df,mapping = aes(x = x1,y = x2))+geom_point()
```



12. Det finns olika sätt att hantera detta. Ett sätt är att ändra punkternas transparens. Gör vi det att ”mörkare regioner” i plotten innebär att fler datapunkter ligger där. Vi kan också testa att minska

ner på punktstorleken.

```
ggplot(data = df,mapping = aes(x = x1,y = x2))+geom_point(alpha=0.5)
ggplot(data = df,mapping = aes(x = x1,y = x2))+geom_point(alpha=0.2)
ggplot(data = df,mapping = aes(x = x1,y = x2))+geom_point(alpha=0.1)

ggplot(data = df,mapping = aes(x = x1,y = x2))+geom_point(size=0.1)
ggplot(data = df,mapping = aes(x = x1,y = x2))+geom_point(size=0.3)
```

13. Ett annat sätt att hantera overplotting är att minska ner på mängden observationer/data. Detta kan t.ex. göras genom att dra ett slumpmässigt urval av alla observationer och plotta dessa istället för alla observationer. Tanken är då att urvalet ska ge en representativ bild av hela datamängden.

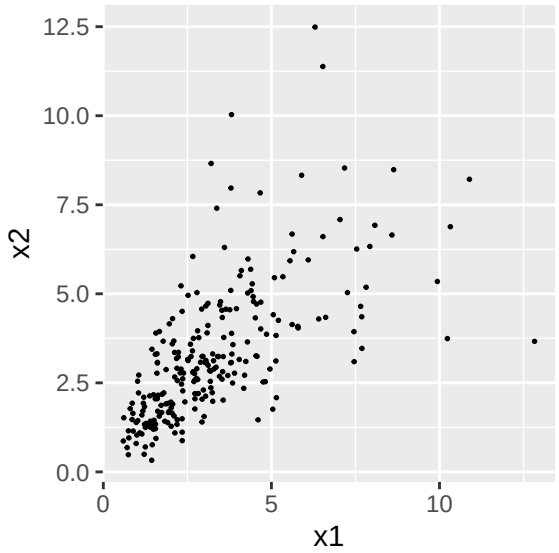
```
dim(df)

[1] 5000    2

# dra slumpmässigt urval med ett radindex:
set.seed(964)
rad_index<-sample(x = nrow(df),size = 250)
df_small<-df[rad_index,]
dim(df_small)

[1] 250   2

ggplot(data = df_small,mapping = aes(x = x1,y = x2))+geom_point(size=0.3)
```



1.1.3 Barplot

För att pröva dessa diagram använder vi oss av datamaterialet `mtcars`. Vi börjar med att läsa in datamaterialet `mtcars`. För att få mer information om detta datamaterial, använd `?mtcars`. Vi gör också om ett par variabler:

```
data(mtcars)
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)
```

Till skillnad från basgrafiken använder vi inte olika funktioner för olika plottar utan vi använder bara olika `geoms`.

1. Stapeldiagram utgår från kategoriska variabler. Vill vi exempelvis skapa ett stapeldiagram anger vi bara en axel och ett annat geom, men i övrigt är det inge större skillnad mot en linjegraf:

```
p <- ggplot(data=mtcars,aes(x=cyl)) + geom_bar()
```

2. Vi kan också enkelt lägga till funktionen `coord_flip()` för att skapa ett liggande stapelidagram istället för ett stående.

```
p + coord_flip()
```

3. Skillnaden ligger i att det finns lite andra aestetics för stapeldiagram än för övriga diagram som `fill`. Läs under "Aesthetics" i dokumentationen för `geom_bar()`.

```
p <- ggplot(data=mtcars,aes(x=cyl)) + geom_bar(fill="pink", colour="darkorchid4")
```

4. Det finns olika sätt att få en relativ skala på y-axeln, se nedan för exempel.

```
ggplot(data=mtcars, aes(x = cyl)) +
geom_bar(aes(y = after_stat(count)/sum(after_stat(count))))+ylab("Proportions")

ggplot(mtcars, aes(x = cyl)) +
geom_bar(aes(y = after_stat(count)/sum(after_stat(count)))) +
scale_y_continuous(labels=scales::percent) +   ylab("relative frequencies")
```

5. För att skapa stapeldiagram med flera grupper behöver vi dels lägga till en till variabel som indikerar att vi vill ha ex. olika färger för olika grupper samt ange hur dessa diagram ska se ut. Pröva exemplen nedan:

```
p1 <- ggplot(data=mtcars) + aes(x=cyl, fill=gear) + geom_bar(position="stack")
p2 <- ggplot(data=mtcars) + aes(x=cyl, fill=gear) + geom_bar(position="dodge")

p1 + scale_fill_discrete(name="Testa\nDetta")
p1 + scale_fill_manual(values=c("black", "blue", "red"))

p2 + scale_fill_discrete(name="Testa\nDetta")
p2 + scale_fill_manual(values=c("black", "blue", "red"))
```

6. För fler detaljer kring grupperade barplots, se här.

1.1.4 Histogram

1. När vi vill göra histogram så utgår vi ifrån en kontinuerlig variabel. Om vi vill göra ett histogram så använder vi oss av `geom_histogram()`.

```
p <- ggplot(data=mtcars) + aes(x=mpg)
p + geom_histogram(fill="yellow", colour="red",binwidth=7)
p + geom_histogram(fill="yellow", colour="red",binwidth=3)
```

2. Det går också att skapa histogram med `geom_bar()`, men det är bättre att använda `geom_histogram()`. Den egentliga skillnaden mellan ett stapeldiagram och ett histogram är bara huruvida variabeln är kontinuerlig eller inte.

3. Ibland vill vi ha en relativ skala på y-axeln när vi gör histogram. Detta kan göras med att lägga till `aes(y=..density..)` Testa koden nedan. Notera hur skalan på y-axeln ändras.

```
p <- ggplot(data=mtcars) + aes(x=mpg)
p + geom_histogram(aes(y=..density..),fill="yellow", colour="red",binwidth=3)

set.seed(324)
df<-data.frame(var=rnorm(5000))
ggplot(data=df,aes(x=var))+geom_histogram(aes(y=..density..),fill="darkred", colour="red",binwi
```

4. Density plots: Vi kan representera fördelningen över olika värden med histogram. Vi kan tänka att den övre kanten av histogrammet är approxiamtion av en underliggande täthetsfunktion. Det finns statistika metoder som kan skatta den underliggande tätheten (se t.ex. funktionen `density()`). Ibland vill vi plotta dessa täthetsfunktioner,och då kan vi använda `geom_density()`.

```
# skapa data:
set.seed(4390)

y<-c(rnorm(n = 1000,mean = 7,sd = 1),
rnorm(n = 200,mean = 4,sd = 0.1),
rgamma(n = 2000,shape = 1,rate = 1))

Y<-data.frame(var1=y)

# vanligt histogram:
ggplot(data = Y,aes(x=var1,y=..density..))+geom_histogram(binwidth = 0.3)

# density plots:
ggplot(data = Y,aes(x=var1))+geom_density()
ggplot(data = Y,aes(x=var1))+geom_density(adjust=1)
ggplot(data = Y,aes(x=var1))+geom_density(adjust=0.1)
ggplot(data = Y,aes(x=var1))+geom_density(adjust=1.5)

# fler ex:
ggplot(data = Y,aes(x=var1))+geom_density(adjust=0.1,alpha=0.5,fill="red")
ggplot(data = Y,aes(x=var1))+geom_density(adjust=0.1,alpha=0.1,fill="blue")

# kombinera:
ggplot(data = Y,aes(x=var1,y=..density..))+
        geom_histogram(binwidth = 0.3)+geom_density(adjust=0.3,col="blue",linewidth=1.5)
```

5. Utgå från iris-data. Gör ett histogram för varje numerisk variabel. Ändra `binwidth` så att figuerna ser bra ut.

1.1.5 Boxplot

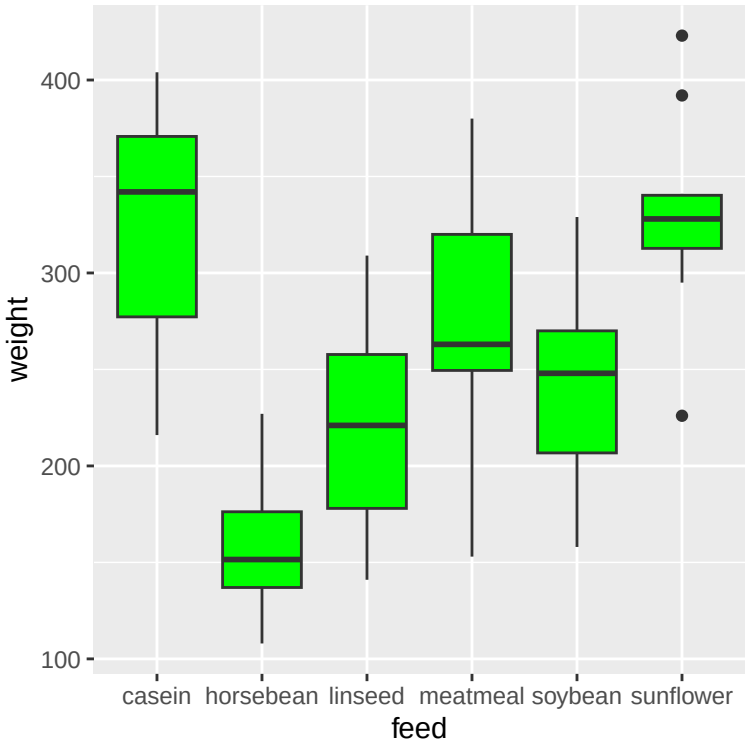
1. Boxplottar kan användas när vi vill plotta en kontinuerlig variabel eller när vi vill gruppera en kontinuerlig efter en kategorisk variabel.

   (a) En kontinuerlig variabel:

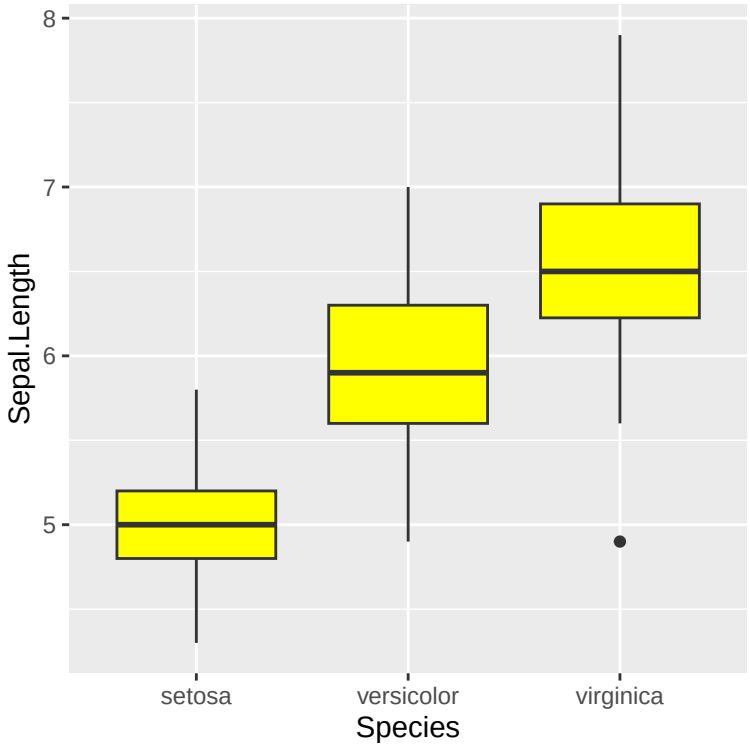
```
?chickwts
ggplot(data = chickwts,aes(y=weight))+geom_boxplot(fill="green")
ggplot(data = iris,aes(y=Sepal.Length))+geom_boxplot(fill="yellow")
```

(b) Gruppera:

```
ggplot(data = chickwts,aes(y=weight,x=feed))+geom_boxplot(fill="green")
```




```
ggplot(data = iris,aes(y=Sepal.Length,x=Species))+geom_boxplot(fill="yellow")
```



2. Vi kan sen lägga till fler lager till våra plottar.

```
p <- ggplot(data=mtcars,aes(x=cyl, y=mpg)) + geom_boxplot()
p
p + coord_flip() + xlab("X") + ggtitle("Hejsan")
```

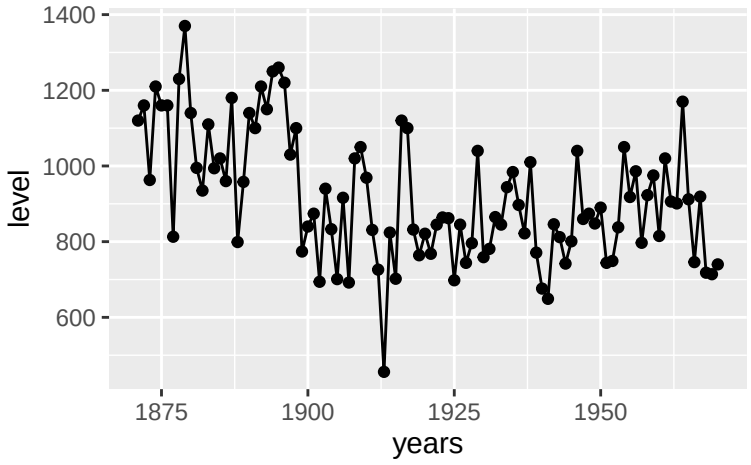
1.2 Grafiska teman/profiler

En av de stora fördelarna med att ggplot skiljer ut själva plotten från utseendet är att det är enkelt att skapa strukturer för olika delar av en graf som vi vill använda flera gånger. En av de bästa exemplen på detta är teman i `ggplot`. Ett tema är en uppsättning med inställningar för en grafisk profil som vi vill använda i `ggplot2`.

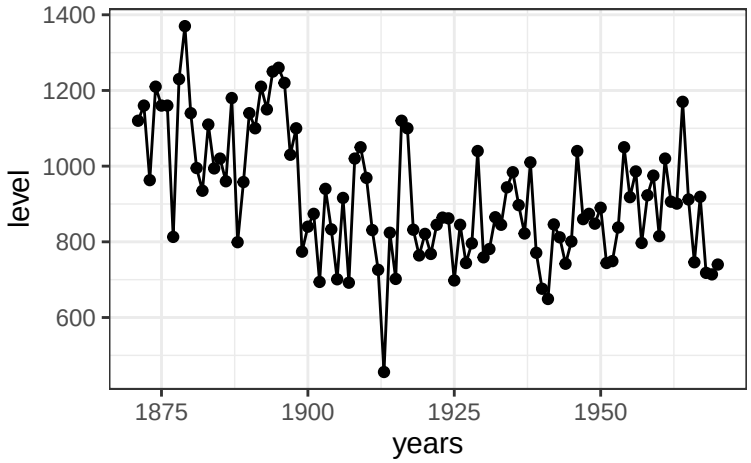
Den stora fördelen är att har vi väl skapat ett tema (vilket kan ta lite tid) kan temat läggas till mycket enkelt till samtliga grafer. Detta underlättar kopplingen mellan exempelvis grafiska profiler och de grafer som produceras, vilket gör att `ggplot2` är mycket populärt i företag och organisationer. Ett exempel på rapport som använder `ggplot2` genomgående är Pensionsmyndighetens Orange rapport.

1. Med `ggplot2` kommer en del teman förinstallerade och precis som allt annat i R är det enkelt att bara lägga till den grafiska profilen efter att vi skapat en graf.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line() + geom_point()
p
```




```
p <- p + theme_bw()
p
```



2. Pröva på liknande sätt följande teman: `theme_grey()`, `theme_classic()`.
3. Ett tema i `ggplot2` är bara en funktion, så det är enkelt att titta på hur temat ser ut och sedan utgå från ett befintligt tema för att anpassa det till det utseende vi själva vill ha. Sedan kan detta tema enkelt spridas till alla som arbetar med visualisering med `ggplot`.

```
theme_bw
```

4. Att ändra den grafiska profilen innebär då bara att ändra denna temafunktion (även om det kan innebära en del jobb).
5. Det finns också ett separat R-paket med ett antal vanliga teman kallat `ggthemes`. De olika teman som är installerade i `ggthemes` framgår här.
6. Med dessa går det enkelt att skapa olika färgsättningar för samma graf. I `ggthemes`-paketet finns också färgpaletter som passar bra för personer som är färgblinda.

```
library(ggthemes)

Error in library(ggthemes): there is no package called 'ggthemes'

p + theme_excel()
```

```
Error in theme_excel(): could not find function "theme_excel"

p + theme_economist()

Error in theme_economist(): could not find function "theme_economist"

p + theme_tufte()

Error in theme_tufte(): could not find function "theme_tufte"
```

1.3 Kombinera plottar

1.3.1 facet_grid() och facet_wrap()

Det finns olika sätt att kombinera flera plottar eller skapa subplots med ggplot2. Ett sätt är att använda `facet_grid()`. `facet_grid()` är bra när vi vill skapa subplots baserat på olika kategoriska variabler i datasetet.

1. `facet_grid()` utgår från ett formelobjekt, som kommer att styra hur våra subplottar skapas. Låt `var1` och `var2` vara två kategoriska variabler i vår data.frame med minst två unika värden. Beroende på hur vi definerar formelobjektet så kommer `facet_grid()` att dela upp data i olika delar och skapa samma sorts plot men för olika delmängder av data i de olika delplottarna. Nedan följer exempel på hur formelobjeten kan skapas
   (a) `var1~.` skapar rader med plottar där data delas upp baserat på värdena i `var1`.
   (b) `.~var1` skapar kolumner med plottar där data delas upp baserat på värdena i `var1`.
   (c) `var1~var2` skapar rader och kolumner med plottar där data delas upp baserat på värdena i både `var1` och `var2`.
2. Utgå från Iris-data. Vi vill gör subplots baserat på variablen ”species”.
   (a) Vi börjar med att göra histogram över variabeln ”Sepal.Length”.

```
# all data i samma plot:
ggplot(data = iris,mapping = aes(x = Sepal.Length))
+ geom_histogram(binwidth = 0.2,fill="green",col="black")

# kolumner:
ggplot(data = iris,mapping = aes(x = iSepal.Length))
+ geom_histogram(binwidth = 0.2,fill="green",col="black")
+ facet_grid(.~Species)

# rader:
ggplot(data = iris,mapping = aes(x = Sepal.Length))
+ geom_histogram(binwidth = 0.2,fill="green",col="black")
+ facet_grid(Species~.)
```

   (b) Nu vill vi göra scatter plots mellan ”Sepal.Length” och ”Sepal.Width”

```
# all data i samma plot:
ggplot(data = iris,mapping = aes(x = Sepal.Length,y=Sepal.Width)
+ geom_point(size=3)

# rader:
ggplot(data = iris,mapping = aes(x = Sepal.Length,y=Sepal.Width))
+ geom_point(size=3)+facet_grid(Species~.)
```

```
# med regressionslinje:
ggplot(data = iris,mapping = aes(x = Sepal.Length,y=Sepal.Width))
+ geom_point(size=3)+facet_grid(Species~.)
+ geom_smooth(method = lm,se=FALSE)
```

3. Nedan följer några exempel med mpg-data.

```
mpg<-as.data.frame(mpg)
head(mpg)
ggplot(data = mpg,mapping = aes(x =cty ,y=hwy))
+ geom_point(size=1)
+ facet_grid(cyl~.)

ggplot(data = mpg,mapping = aes(x =cty ,y=hwy))
+ geom_point(size=1)
+ facet_grid(cyl~drv)

ggplot(data = mpg,mapping = aes(x =cty ,y=hwy))
+ geom_point(size=0.8)
+ facet_grid(trans~class)
```

4. Ett alternativ är `facet_wrap()`, som liknar `facet_grid()`. Skillnaden är att `facet_wrap()` själv bestämmer hur rader och kolumner ska ordnas. Testa koden nedan:

```
library(ggthemes)
# facet_wrap():
ggplot(data = mpg,mapping = aes(x =cty ,y=hwy))+ geom_point(size=0.8)+ facet_wrap(vars(trans))
# facet_grid():
ggplot(data = mpg,mapping = aes(x =cty ,y=hwy))+ geom_point(size=0.8)+ facet_grid(trans~.)
```

5. Se här för mer detaljer om `facet_grid()`och `facet_wrap()`.

1.3.2 cowplot

Ibland vill vi kombinera godtyckliga plottar till subplottar. Detta kan göras med paket cowplot.

1. Vill vi skapa subplots så använder vi funktionen `plot_grid()`. Kolla i dokumentationen för `plot_grid()`. Anta att vi har sparat tre ggplot2-objekt som vi kallar `p1`, `p2` och `p3`.
   (a) Alternativ 1: Skriv in alla plot-objekt direkt som objekt `plot_grid(p1,p2,p3)`. Detta är ofta smidigt om vill skapa enstaka plottar med subplots.
   (b) Alternativ 2: Lägg alla plottarna i en lista `plot_grid(plotlist=list(p1,p2,p3))`. Detta är ofta avnändbart om vi vill använda i `plot_grid()` en loop eller en funktion. Andra fall då `plotlist=` är bra är om mängden plottar eller deras namn varierar.
2. Skapa först några ggplot2-objekt och spara dessa.

```
library(cowplot)
mpg<-as.data.frame(mpg)
head(mpg)

p0<-ggplot(data = mpg,mapping = aes(x =  cty ))
+ geom_histogram(binwidth = 2)
```

```
p1<-ggplot(data = mpg,mapping = aes(x =  cty ,y=hwy))
+ geom_point(size=1)

p2<-ggplot(data = mpg,mapping = aes(x =  cty ,y=hwy))
+ geom_point(size=1)
+ facet_grid( cyl~.)

p3<-ggplot(data = mpg,mapping = aes(x =  cty ,y=hwy))
+ geom_point(size=0.8)
+ facet_grid(drv ~.)
```

3. Nu sätter vi samman dessa plottar:

```
plot_grid(p0,p1)
plot_grid(p1,p0)

g1<-plot_grid(p3,p1)
print(g1)

plot_grid(p0,p1,nrow = 1)

plot_grid(p0,p1,nrow = 2,labels = "AUTO")

plot_grid(p0,p1,nrow = 2,labels = c("ABC","123"))


p_list<-list(p0,p1,p2,p3)
plot_grid(plotlist = p_list,nrow = 2,ncol = 2,labels = "AUTO")
```

4. Skapa nu minst tre plottar baserat på iris-data som ni sätter ihop med `plot_grid()`.

1.4 Mer övningar

Det finns en stor mängd guider och introduktioner till ggplot2 på internet. Nedan följer några tips om ni vill öva mer:

- R Graphics Cookbook, 2nd edition: Finns kaptiel för olika typer av plottar t.ex. Bar Graph. Här finns mycket information om hur ni kan kontrollera detaljer i plottarna.
- An Introduction to ggplot2
- 3 Data visualisation
- 28 Graphics for communication

Kapitel 2

Enklare statistisk analys

2.1 Enklare statistiska metoder mm

2.1.1 Kombinatorik

1. Det finns inbyggda funktioner i R för kombinatorik. Testa att köra `?Special`. Leta reda på `factorial()` och `choose()` och läs om dem.
2. Faktulet: Beräkna nu 3!, 6!, och 12! med funktionen `factorial()`.
3. Binomialkoefficienter: Räkna nu ut följande $\begin{pmatrix} 10 \\ 2 \end{pmatrix}$, $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$och $\begin{pmatrix} 20 \\ 5 \end{pmatrix}$med funktionen `choose()`.
4. Skapa nu en egen funktion för binomialkoefficienter utan att använda `choose()`. Tips `?factorial()`
5. Nu ska ni implementera en funktion som ska beräkna täthetsfunktionen för för binomialfördelningen. Titta i `?dbinom` eller här och se hur tätheten ska beräknas. Ni kan sedan använda `dbinom()` för att se om funktionen räknar rätt. Testa med några olika värden för `n` och `p`.

2.1.2 Korstabulering och χ^2-tester

1. Vi börjar med att läsa in det interna datasetet `iris` med `data(iris)` i R. Vi ska nu använda detta dataset för att pröva att analysera data i R.

```
data(iris)
```

2. Som ett första steg vill vi pröva att producera korstabeller. I `iris` finns bara en kategorisk variabel, därför klassindelar vi två kontinuerliga variabler på följande sätt.

```
iris$Petal.Length.Cat <- cut(iris$Petal.Length, breaks=3)
iris$Sepal.Length.Cat <- cut(iris$Sepal.Length, breaks=3)
```

3. Vi börjar med att skapa en korstabell på följande sätt:

```
table(iris$Petal.Length.Cat, iris$Species)
```

4. När vi vill använda flera kategoriska variabler är `ftable()` lämpligt:

```
ftable(iris$Petal.Length.Cat, iris$Sepal.Length.Cat, iris$Species)
```

5. För att göra ett χ^2-test använder vi funktionen `chisq.test()`. Funktionen behöver en tabell att testa, så vi skapar och sparar tabellen ovan och testar den sedan..

```
tab <- table(iris$Petal.Length.Cat, iris$Species)
chisq.test(tab)
```

6. Om vi tittar på tabellen ovan är det många värden som är 0 (d.v.s. mindre än 5). Då behöver vi korrigera vårt test med Yates korrektion. Detta kan vi göra i R genom att lägga till argumentet `correct=TRUE`.

```
chisq.test(tab, correct=TRUE)
```

7. Ett annat sätt är att använda Fishers exakta test istället. Det görs på följande sätt:

```
fisher.test(tab)
```

2.1.3 t-test

1. Vill vi jämföra två grupper avseende en kontinuerlig variabel använder vi funktionen `t.test()`. Läs i dokumentationen för `t.test()` vilka argument som finns och hur de fungerar. Undersök vad funktionen returnerar.
2. Inledningvis kan vi testa om `Sepal.Length` för `iris` versicolor är mindre 6. Vi börjar med att plocka ut elementen för de olika arterna:

```
versLength <- iris$Sepal.Length[iris$Species=="versicolor"]
test_obj<-t.test(x=versLength, alternative="greater", mu=6)
print(test_obj)
str(test_obj)
test_obj$p.value
```

3. Om vi nu vill testa om skillnaden mellan två blomsterarter är olika anger vi två vektorer, en för varje art:

```
virginLength <- iris$Sepal.Length[iris$Species=="virginica"]
test_obj2<-t.test(x=versLength, y=virginLength)
print(test_obj)

test_obj2$statistic
test_obj2$p.value
test_obj2$conf.int
test_obj2$estimate
```

2.1.4 Sambandsmått

För att studera samband mellan två eller flera kontinuerliga variabler studerar vi ofta korrelation och kovarians.

1. För att beräkna kovarians och korrelation används `cov()` och `cor()`.

```r
cor(iris$Petal.Length, iris$Petal.Width)
cov(iris$Petal.Length, iris$Petal.Width)
```

2. Vi kan också enkelt skapa kovarians- och korrelationsmatriser på samma sätt:

```r
cor(iris[,1:4])
cov(iris[,1:4])
```

3. Vill vi göra ett enkelt hypotestest för korrelationen använder vi `cor.test()`:

```r
cor.test(iris$Petal.Length, iris$Petal.Width)
```

2.1.5 Beskrivande statistik

1. Hitta det hösta och minsta värdet i `iris` (för alla variabler). Ta reda på vilken rad dessa värden finns på med relationsoperatorn == och `which.max()`.
   (a) Beräkna medelvärden för alla kolumnerna med `colMeans()`.
   (b) Beräkna nu kvartiler för `Petal.Width` med funktionen `quantiles()`. Beräkna den 1 och 99 procentiga percentilen? [**Tips!** `?quantile`]
   (c) Beräkna nu kvartiler för samtliga variabler.
   (d) Testa nu att använda funktionen `summary()` på `Petal.Width` först och sedan på hela datasetet. Vad får du för resultat?
2. Skapa en logisk vektor som anger om en variabeln `Petal.Width` är större än 2. Spara denna variabel som `small` på följande sätt.
3. Använd `table()` för att se vilka arter (variabeln `Species`) som är `small`.

2.2 * Extraproblem

1. Skapa en funktion som beräknar värdet på en hypergeometrisk fördelning.
2. Skapa en funktion som beräknar värdet på en Geometrisk fördelning.
3. Pokerhänder: Räkna ut sannolikheten för ett par, triss och ”Royal flush” i en pokerhand på 5 kort. Se här för mer info.
4. I vissa fall vill vi simulera data från en given sannolikhetsmodell. Det kan exempelvis vara en linjär modell som ser ut på följande sätt:
$$y_i = \alpha + \beta x_i + \epsilon_i$$
där $\epsilon_i \sim \mathcal{N}(0, \sigma)$. Detta görs enklast i flera steg. I detta fall simulerar jag även x_i samt sätter $\alpha = 2, \beta = 4, \sigma = 1$ och $n = 100$.

```r
alpha <- 2
beta <- 4
sigma <- 1
n <- 100

x <- rnorm(n)
y <- alpha + beta*x + rnorm(n, sd=sigma)
```

5. Kör koden ovan och visualisera det simulerade datamaterialet i ett spridningsdiagram [**Tips!** `ggplot()`]
6. Pröva lite olika värden för σ, β och α och visualisera de olika datamaterialen.
7. Nu ska vi utöka modellen oven till: $y_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$ Kör koden nedan.Testa nu att ändra σ, β_1, β_2 och α. Testa att ändra värdet på σ på ett sådant sätt att den kvadratiska sambandet blir väl synligt och blir svårt att se.

```
alpha <- 2
beta <- c(2,4)
sigma <- 1
n <- 100

x <- rnorm(n)
y <- alpha + beta[1]*x+beta[2]*x^2 + rnorm(n, sd=sigma)
```

Kapitel 3

Housing data

1. Ni ska nu analysera datamaterialet HUS, som innehåller information om en mängd hus. Följande variabler finns i data:
   - Försäljningspris (dollar)
   - Bostadsyta (kvadratfot)
   - Antal sovrum
   - Antal badrum
   - Förekomst av luftkonditionering, 1 = luftkonditionering finns, 0 annars
   - Antal bilar som garaget är konstruerat för
   - Förekomst av pool, 1 = pool finns, 0 annars
   - Byggår
   - Tomtstorlek (kvadratfot)
2. Läs in datamaterialet "HUS.csv" och spara det som `HUS`. Det finns även ett dataset "HUS_eng.csv", som har engelska namn på variablerna men innehåller samma data. Materialet finns att tillgå här.
3. Gör följande med `HUS`. Använd ggplot2 för att skapa figurerna.
   (a) Plotta en boxplot över variabeln `Försäljningspris`. Ta fram besrkivande statistik med `summary()` för `Försäljningspris`. Det verkar finnas en del outliers (extremt stora värden), dessa vill vi ta bort från data, kör följande kod:

```
# ta bort alla hus som har pris större än 3:e kvartilen:
index<-HUS[,1]<quantile(HUS[,1])[4]
HUS<-HUS[index,]
```

   (b) Plotta ett histogram för `Försäljningspris`, (bonusfråga: ser fördelningen symmetrisk ut?)
   (c) Gör en scatter plot mellan `Försäljningspris` och `Bostadsyta`.
      i. Låt färgen på punkterna bero på värdet på `Antal.sovrum`
      ii. Låt färgen på punkterna bero på värdet på `Luftkonditionering`
      iii. Använd `facet_grid()` för att skapa subplots som beror på värdet på `Luftkonditionering`.
   (d) Beräkna ett tvåsidigt konfidensintervall (KI) med $\alpha = 0.05$ för `Försäljningspris` (tips `?t.test()`) spara i `testPris`.
   (e) Välj ut KI från `testPris`. tips: `?t.test()` och läs under rubriken "Value". Kör sedan `str(testPris)`. Testa `class(testPris)`
   (f) Beräkna ett tvåsidigt KI med $\alpha = 0.10$ för `Försäljningspris`
   (g) Beräkna ett enkelsidigt (undre) KI med $\alpha = 0.01$ för `Försäljningspris`

(h) Kör koden nedan, vad blir resutlatet? Hur ska medelvärdet för variabeln `Luftkonditionering` tolkas?

```
summary(HUS)
```

4. Skapa följande frekvenstabeller från HUS (tips: `?table()`)
   (a) För `Antal.sovrum`
   (b) För `Luftkonditionering`
   (c) Mellan variablerna `Antal.sovrum` och `Luftkonditionering`, spara som `sovLuftTab. Testa class(sovLuftTab)`
   (d) Mellan variablerna `Antal.sovrum` och `Antal.badrum`
5. Kör `prop.table(sovLuftTab)` och `prop.table(sovLuftTab,margin=1)`. Vad händer med tabellen?
6. Beräkna fishers exakta test för tabellen `sovLuftTab`, använd $\alpha = 0.05$**,** vad är noll-hypotesen? Kan vi förkasta den? (tips: `?fisher.test`())
7. Antag att variablen `Försäljningspris` representerar en hel population med hus. Dra ett slumpmässigt stickprov med 20 hus utan återläggning. Beräkna ett tvåsidigt KI ($\alpha = 0.01$) utifrån stickprovet för populationsmedelvärdet.
8. Antag nu att hela datasetet HUS är alla hus i en population. Dra ett stickprov slumpmässigt (dvs välj rader) på 40 hus utan återläggning. Gör följande beräkningar utifrån stickprovet:
   (a) Numeriska variaber: Beräkna ett tvåsidigt KI med $\alpha = 0.01$ för populationsmedelvärdet.
   (b) Binära(0/1) variaber: Beräkna ett tvåsidigt KI med $\alpha = 0.01$ för populationsandelen. Tips: `?prop.test(), ?table()`

Kapitel 4

Frivillig fördjupning: Introduktion till linjär regression

Denna laboration kommer inte gå in i teorin bakom (multipel) linjär regression utan kommer fokusera hur man anpassar regressionmodeller, analyserar resultatet och gör diagnostiska tester i R.

4.1 Anpassa en regressionsmodell

Den vanliga regressionmodellen bygger på modellen

$$y_i = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon_i \tag{4.1}$$

eller med matrisnotation

$$\mathbf{y} = \mathbf{Bx} + \epsilon$$

där $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$ där de okända parametrarna är β samt σ i modellen och baserat på våra data vill vi uppskatta dessa parametrar.

I R (och till skillnad mot de flesta statistikprogram) skiljer vi på att anpassa (eller skatta) en modell och studera resultatet från modellen eller använda den för ex. prediktion. För att anpassa en modell använder vi funktionen `lm()`. Det gör att vi kan anpassa en linjär modell och sedan spara den som ett vanligt R-objekt. Sedan kan vi studera detta objekt på ett stort antal sätt, skriva egna funktioner för specifik analys, eller för att grafiskt visualisera modellen.

1. Vi börjar med att läsa in datasetet `Prestige` som finns i paketet `car` . Vi behöver också paketet `MASS`. Installera dessa paket om du inte har dem installerade.

```
library(car)

Loading required package: carData

data(Prestige)
library(MASS)
```

2. Läs på kort om de variabler som finns i datasetet med `?Prestige`.
3. Börja med att visualisera sambandet mellan `prestige` och `income`. [**Tips!** `plot()`]
4. Vi ska nu anpassa vår första linjära regressionsmodell. För att anpassa en modell i R behöver vi ange två saker, dels en `formula` som beskriver hur modellen ser ut, och sedan vilket dataset som innehåller variablerna vi vill ha i vår modell. För att anpassa en linjär regression med inkomst som oberoende variabel gör vi på följande sätt:

```
minModell <- lm(prestige ~ income, data=Prestige)
minModell
```

```
Call:
lm(formula = prestige ~ income, data = Prestige)

Coefficients:
(Intercept)       income
    27.1412       0.0029
```

5. Om vi tittar på det resulterande objektet ser vi regressionskoefficienternas värden. Detta är bra för en snabb koll, men senare kommer vi gå in mer på hur vi kan få ut mer utförliga resultat.

6. `prestige ~ income` är ett objekt av klass `fomula`. `fomula`-objekt används i många funktioner i R för att specificera statistiska modeller på ett flexibelt sätt. Testa att köra `class(prestige ~ income)` och `str(prestige ~ income)`. Likt andra objekt kan `fomula`-objekt sparas och manipuleras. Kör koden nedan.

```
x<-prestige ~ income
minModell <- lm(x, data=Prestige)
minModell
```

7. Det går också att bara ange vektorer (de behöver inte heller ligga i samma `data.frame`):

```
minModell2 <- lm(Prestige$prestige ~ Prestige$income)
```

8. Vill vi lägga till flera variabler (andel kvinnor, utbildning) gör vi på följande sätt:

```
minModell3 <- lm(prestige ~ income + women + education, data=Prestige)
```

9. Vi kan på detta sätt enkelt lägga till ett stort antal variabler. Som standard så inkluderas alltid en intercept (β_0 i 4.1) i modellen, vill vi ta bort denna använder lägger vi till `-1`:

```
minModell4 <- lm(prestige ~ income + women - 1, data=Prestige)
```

10. Nu kommer också fördelarna med att ha definierat faktorvariabler. Lägger vi in en faktorvariabel förstår R detta automatiskt och "under the hood" skapas dummyvariabler för vilka koefficienter skattas. Dummyvariabler kommer att förklaras närmare i senare kurser om regression.

```
minModell5 <- lm(prestige ~ income + type, data=Prestige)
```

11. Till sist kan det vara så att vi vill lägga till interaktionseffekter i modellen. Detta görs enkelt med `:` på följande sätt (detta inkluderar både additiva och multiplikativa effekter):

```
minModell6 <- lm(prestige ~ income:women, data=Prestige)
```

4.2 Analysera resultatet från en linjär regression

Nu har vi anpassat (och smygtittat på) lite olika modeller baserat på datasetet `Prestige`. Vi ska nu studera de resultat vi fått lite mer. Nu har vi stor nytta av R:s objektorienterade uppbyggnad och generiska funktioner.

1. Vi börjar med att använda funktionen `summary()`. Se exemplet nedan:

```
summary(minModell)


Call:
lm(formula = prestige ~ income, data = Prestige)

Residuals:
   Min     1Q Median     3Q    Max
-33.01  -8.38  -2.38   8.43  32.08

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.71e+01   2.27e+00    12.0   <2e-16 ***
income      2.90e-03   2.83e-04    10.2   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.1 on 100 degrees of freedom
Multiple R-squared:  0.511, Adjusted R-squared:  0.506
F-statistic:  105 on 1 and 100 DF,  p-value: <2e-16
```

2. Med denna får vi en sammanfattning av de flesta resultat i modellanpassningen. Pröva denna funktion på `minModell3`. Hitta följande storheter i sammanfattningen av modellen:
   β, σ, R^2, F-statistikan samt p-värdena för de olika β-koefficienterna.
3. Pröva att analysera modell 5 och 6 på samma sätt. Hur ser en kategorisk variabel samt interaktionseffekter ut i R?
4. Funktionen `summary()` ger en bra bild av resultatet, men vi vill ofta också studera en ANOVA-tabell för vår modells ingående variabler. För detta används funktionen `anova()`. Pröva den på modellerna 3, 5 och 6.
5. De flesta (icke-bayesianska) statistiker och forskare är mycket förtjusta i p-värden. För att få ut p-värden till ANOVA-tabellen använder vi oss av argumentet `test='Chisq'`. Pröva att på detta sätt testa för variabler i modell 5.
6. Vi kan också inkludera flera modeller i en ANOVA-tabell. Vad ger det för resultat?

```
anova(minModell, minModell3, test="Chisq")
```

4.2.1 Använda parametrar och resultat för vidare analys

Vill vi använda vissa speciella delar av en modell kan vi plocka ut delar från modellen med olika funktioner. Exempelvis kanske vi är intresserad av att använda β-koefficenterna i modellen till något annat. Eller använda modellen för att göra prediktioner på nya data.

1. Pröva att använda funktionen `coef()` på modellobjekt 3 ovan. Pröva att spara ned resultatet som ett nytt objekt. Vad får du för resultat? Jämför med `print()`.

2. På ett liknande sätt kan vi snabbt få ut alla residualer (om vi vill studera dessa) med funktionen `resid()`. Pröva denna funktion på modell 3 ovan och plotta resiudalerna i ett histogram. Är residualerna normalfördelade? [**Tips!** `hist()`,`geom_histogram()`]

3. Pröva att med hjälp av residualvektorn och relationsoperatorer ta fram vilket yrke har den största negativa residualen (d.v.s. lägst prestige kontrollerat för utbildning, inkomst och könsfördelning). [**Tips!** `which.min()`]

4. Vi kan också få ut de predicerade värdena för varje observation med `predict()`. Pröva att på detta sätt se vilket yrke som har den högsta respektive lägsta förvänatade prestigen. [**Tips!** `which.max()`]

5. Vi kan också använda `predict()` för att skapa prediktioner på nya data. Då behöver vi ta ett nytt dataset (men med samma variabelnamn och variabeltyper) och ange detta data med argumentet `newdata`. Pröva att ta de fem första raderna i datasetet `Prestige` och spara det som `newPrestige`. Pröva att predicera variabeln `prestige` för detta dataset med modell 6.

4.3 Tester och diagnostik

I linjär regression görs ett stort antal antagande om i modellen som ligger till grund för att kunna dra slutsatser från materialet. Nedan följer ett antal tester och visualiseringar för att diagnostisera ett antal centrala antaganden i linjär regression.

Observera att syftet med denna del är att testa lite olika funktionalitet i R, för mer fördjupad genomgång av de olika antagandena och hur dessa problem avhjälpes se dokumentationen till funktionerna eller i litteratur på området (en bra sammanfattning är här).

Välj en godtycklig modell ovan (eller skapa en egen ny modell) och utför följande diagnostik:

1. Det första och enklaste sättet att diagnostisera vår modell är att vi använder funktionen `plot()` på vårt modellobjekt. Pröva `plot()` på modell 3, 5 och 6.

2. Nedan finns lite olika diagnostiska verktyg och kodexempel. Pröva på detta sätt att...

   (a) Identifiera uteliggare

   ```
   outlierTest(minModell)
   qqPlot(minModell)
   leveragePlots(minModell)
   ```

   (b) Identifiera observationer med starkt inflytande på modellen

   ```
   avPlot(minModell)
   influencePlot(minModell)
   ```

   (c) Utvärdera linjäritetsantagande för regressionkoefficienterna

   ```
   crPlots(minModell)
   ceresPlots(minModell)
   ```

   (d) Studera multikollinearitet mellan regressionskoefficienterna

   ```
   vif(minModell)
   ```

   (e) Oberoende feltermer (framförallt för tidsserieregression)

   ```
   durbinWatsonTest(minModell)
   ```

(f) Residualernas fördelning

```
qqPlot(minModell)
```

(g) Konstant varians (homoscedasticitet)

```
ncvTest(minModell)
spreadLevelPlot(minModell)
```

3. Pröva att baserat på dessa tester att anpassa den modell för variabeln `prestige` som du själv tror mest på. Vad är dina slutsatser?

4.3.1 Anscombes data

Vi ska nu pröva ett klassiskt exempel när det gäller linjär regression, `anscombe`s data. Materialet består av fyra x-variabler och fyra y-variabler där materialet ser mycket olika ut, även om de enskilda regressionsmodellerna har exakt samma resultat.

1. Börja med att läsa in materialet i R med `data()`.

```
data(anscombe)
```

2. Anpassa nu följande fyra regressionsmodeller i R utan att plotta materialet.

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \epsilon \\ y_2 &= \beta_0 + \beta_1 x_2 + \epsilon \\ y_3 &= \beta_0 + \beta_1 x_3 + \epsilon \\ y_4 &= \beta_0 + \beta_1 x_4 + \epsilon \end{aligned}$$

3. Studera koefficienterna i modellerna. De ska vara nästan identiska.

4. Pröva nu att plotta datamaterialet och använd de diagnostiska verktygen ovan. Vilka antaganden är problematiska i respektive modell, och vilka diagnostiska verktyg fångar upp detta?

Nu är du klar!

Litteraturförteckning

[1] Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

[2] Leland Wilkinson. *The grammar of graphics*. Springer, 2012.

[3] Leland Wilkinson, D Wills, D Rope, A Norton, and R Dubbs. *The grammar of graphics*. Springer, 2006.