R-anvisningar till *Grundläggande statistik*

2025-10-01

Innehåll

Introduktion **5**

Installation av R och RStudio **7**
0.1 Inledning . . . 7
0.2 Installation av R . . . 7
0.3 Installation av RStudio . . . 7
0.4 Gränssnittet i RStudio . . . 8
0.5 Paket i R . . . 8

1 Datahantering och grafer **9**
1.1 Uppstart och orientering . . . 9
1.2 *Packages* från CRAN . . . 10
1.3 Objekt och funktioner . . . 11
1.4 Sekvenser av funktioner . . . 13
1.5 Datainskrivning . . . 14
1.6 Urval ur en tabell med `select` och `filter` . . . 16
1.7 Grafer med `ggplot2` . . . 17
1.8 AI-uppgift I . . . 19

2 Beskrivande statistik **21**
2.1 Repetition från datorövning 1 . . . 21
2.2 Import av data från en Excelfil . . . 22
2.3 Ändra och skapa nya kolumner med `mutate` . . . 23
2.4 Sammanfattande lägesmått . . . 24
2.5 Sammanfattande spridningsmått . . . 25
2.6 Ordna upp beskrivande statistik och exportera . . . 27
2.7 Kumulativt medelvärde . . . 28
2.8 Tredimensionella grafer med `plotly` . . . 29
2.9 AI-uppgift II . . . 30

3 Tester vid ett eller flera stickprov **33**
3.1 Repetition från datorövning 2 . . . 33
3.2 Allmänt om hypotestester . . . 34

3.3 Ett stickprov 36
3.4 Två stickprov 42
3.5 AI-uppgift III 47

4 Variansanalys och regression **49**
4.1 Allmänt om variansanalys 49
4.2 Variansanalys. En faktor 49
4.3 Variansanalys. En faktor med block 52
4.4 Bonus. Två faktorer med block 54
4.5 Regression 55
4.6 Korrelation 58

Introduktion

Detta dokument ger en introduktion till R för en kurs i grundläggande statistik.

Installation av R och RStudio

0.1 Inledning

För att köra R-kod på sin dator krävs en installation av programspråket R. För att effektivt arbeta i R används ofta en utvecklingsmiljö (ett tilläggsprogram som på flera sätt förenklar arbetet) och här ges anvisningar till RStudio - den vanligaste utvecklingsmiljön för R. För att komma ingång måste man alltså installera R och RStudio.

0.2 Installation av R

Programspråket R kan laddas ner från https://www.r-project.org/ med följande steg:

1. Klicka på *CRAN* längst upp till vänster.
2. Klicka på den översta länken under 0-Cloud.
3. Välj en nedladdning beroende på operativsystem.
4. För Windows, välj *base*. För macOS, välj den senaste tillgängliga versionen.
5. Installera R från den nedladdade filen. Installation sker som för andra nedladdade program.

0.3 Installation av RStudio

RStudio kan laddas ner från https://www.posit.co/ med följande steg:

1. Klicka på *Download RStudio* uppe till höger.
2. Scrolla nedåt och välj *Download* under *Install RStudio*.
3. Installera RStudio från den nedladdade filen. Installation sker som för andra nedladdade program.

0.4 Gränssnittet i RStudio

När man nu öppnar RStudio ser man att fönstret är uppdelat i fyra delar och att varje del består av en eller flera flikar. De viktigaste är i nuläget

- *Console* där kod körs och resultat skrivs ut,
- *Environment* där man ser skapade objekt,
- *History* där man ser tidigare körd kod,
- *Plots* där man ser skapade grafer, och
- *Help* där man ser hjälpsidor för funktioner.

Ofta skriver man inte sin kod direkt i konsollen, utan i ett separat *skript* - en vanlig textfil som innehåller den kod man vill köra. Genom att organisera sin kod i ett skript kan man lätt strukturera och dokumentera sitt arbete. I RStudio kan man öppna ett nytt skript genom att gå till *File > New File > R Script* eller genom att klicka *Ctrl + Shift + N*. Ett tomt skript öppnar sig då i det övre vänstra delfönstret. Om man skriver

```
a <- 5
```

i skriptet och trycker *Ctrl + Enter* bör man se att koden i skriptet körs i konsollen. Om man tittar i fliken *Environment* ska man också se att det nu skapats ett objekt *a*.

0.5 Paket i R

En av de stora styrkorna med R är att språket kan byggas ut av dess användare. De här tilläggen kan sedan samlas i paket (*packages*) och delas med andra. Rs officiella bibliotek för paket kallas för *CRAN* (*Comprehensive R Archive Network*) och består av drygt 20 000 uppladdade paket som innehåller allt från fritt tillgänglig data till avancerade statistiska modeller.

För att använda ett specifikt paket måste det först installeras. Om man vet namnet på paketet man vill installera kan man köra

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

I det här fallet installeras paketet `tidyverse`, vilket innehåller funktioner för hantering av data.

I RStudio kan man också installera paket från *Packages*-fliken.

Paket måste också laddas för varje ny session. Innan man kan använda innehållet i ett paket måste man därför köra

```
library(tidyverse)
```

Kapitel 1

Datahantering och grafer

Datorövning 1 handlar om grunderna till R. Efter övningen ska vi kunna

- Starta RStudio och orientera oss i gränssnittet,
- Installera och ladda tilläggspaket (*Packages*),
- Definera objekt och tillämpa funktioner i R,
- Transformera en tabell med data genom att välja kolumner och filtrera rader,
- Skapa grafer med `ggplot2`.

1.1 Uppstart och orientering

För att arbeta i R måste vi installera språket R och ett gränssnitt för att arbeta i R, vanligen *RStudio*. Både R och RStudio kan laddas ner gratis, från https://www.r-project.org/ respektive https://posit.co/.

Starta RStudio, till exempel genom att gå till Startmenyn och söka på RStudio eller genom att dubbelklicka på en fil som öppnas i RStudio. Gränssnittet i RStudio är uppdelat i fyra delar och varje del består av en eller flera flikar. De viktigaste är i nuläget

- *Console* där kod körs och resultat skrivs ut,
- *Environment* där man ser skapade objekt,
- *History* där man ser tidigare körd kod,
- *Plots* där man ser skapade grafer, och
- *Help* där man ser hjälpsidor för funktioner.

Uppgift 1.1 (Help-fliken)**.** Hitta fliken *Help*, klicka på husikonen under fliken. Finns det en länk med *RStudio Cheat Sheets*? Följ den länken för att hitta guider till R som kan bli nyttiga längre fram. För nu, gå tillbaka till RStudio.

Ofta skriver man inte sin kod direkt i konsollen, utan i ett separat *skript* - en

vanlig textfil som innehåller den kod man vill köra. I RStudio kan man öppna ett nytt skript genom att gå till *File* > *New File* > *R Script* eller genom att klicka *Ctrl* + *Shift* + *N*.

Uppgift 1.2 (Ett första skript)**.** Öppna ett nytt skript genom File-menyn eller genom *Ctrl* + *Shift* + *N*. Skriv

```
a <- 5
```

i skriptet och tryck *Ctrl* + *Enter*. Titta i flikarna *Console* och *Environment*. Har något hänt? Du bör se att koden i skriptet körts i konsollen och att ett nytt objekt `a` ligger i *Environment*.

1.2 *Packages* från CRAN

En av de stora styrkorna med R är att språket kan byggas ut av dess användare. De här tilläggen kan sedan samlas i paket (*packages*) och delas med andra. Rs officiella bibliotek för paket kallas för *CRAN* (*Comprehensive R Archive Network*).

För att använda ett specifikt paket måste det först installeras. Om man vet namnet på paketet man vill installera kan man köra

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

I det här fallet installeras paketet `tidyverse`, vilket innehåller funktioner för hantering av data.

I RStudio kan man också installera paket från *Packages*-fliken.

Uppgift 1.3 (Installera tidyverse-paketet)**.** Kör raden ovan för att installera `tidyverse`. Du kan antingen köra raden genom att skriva den i *Console* eller genom att skriva i ett skript och köra därifrån genom *Ctrl* + *Enter*.

Uppgift 1.4 (Installera gapminder-paketet)**.** Paketet `gapminder` innehåller lite intressant data vi kommer använda senare. Installera paketet `gapminder` genom att fylla i den saknade biten och köra raden nedan.

```
install.packages("___")
```

Paket måste också laddas för varje ny session. Innan man kan använda innehållet i ett paket måste man därför köra

```
library(tidyverse)
```

Uppgift 1.5 (Ladda gapminder-paketet)**.** Ladda paketet `gapminder` genom att fylla i och köra raden nedan.

```
library(___)
```

Uppgift 1.6 (Paket som inte finns)**.** Vad händer om man försöker installera ett paket som inte finns på *CRAN*? Testa till exempel

```
install.packages("ThisIsNotTheNameOfAnyPackage")
```

och

```
library(ThisIsNotTheNameOfAnyPackage)
```

1.3 Objekt och funktioner

Ett *objekt* i R är en namngiven informationsmängd. Objekt kan se ut på många olika sätt - under kursens gång används objekt som består av insamlad data (konstruerade som vektorer eller tabeller), objekt som är statistiska modeller, och flera andra former. I R skapar man objekt med *assign*-pilen `<-` (mindre än och bindestreck).

I ett tidigare exempel fanns raden

```
a <- 5
```

Här skapas ett objekt med namnet `a` som innehåller informationen `5`. *Assign*-pilen pekar alltså på det namn man vill ge objektet och pekar från objektets innehåll.

Ett lite mer komplicerat exempel på ett objekt ges av

```
b <- c(3, 1, 4, 1, 5, 9)
```

Här skapas ett objekt `b` som innehåller en *serie* numeriska värden (en *vektor*). Värdena i en vektor är ordnade och man kan plocka ut ett specifikt värde med hakparenteser.

```
b[3]                # Det tredje värdet i vektorn b
```

```
## [1] 4
```

```
b[c(3,5)]           # Det tredje och femte värdet i b
```

```
## [1] 4 5
```

Uppgift 1.7 (Skapa en vektor)**.** Skapa ett objekt med namnet `new_vector` som innehåller värden 5, 7 och 10 genom att fylla i följande rad.

```
new_vector <- c(_, _, _)
```

Uppgift 1.8 (Ta ut andra värdet)**.** Använd hakparenteser för att plocka ut det andra värdet ur vektorn `new_vector`.

Objekt kan manipuleras genom att tillämpa *funktioner*. En funktion tar någon ingående data och ger något utgående resultat. Funktioner anges genom

att skriva funktionens namn följt av ingående data inom parenteser, och resultatet kan antingen skrivas ut i konsollen eller sparas som ett nytt objekt. En grundinstallation av R innehåller en mängd färdiga funktioner, t.ex.

```
sum(b)
```

```
## [1] 23
```

vilket ger summan av värdena i vektorn `b`,

```
plot(b)
```

som ger en simpel graf, och

```
sqrt(b)
```

```
## [1] 1.732051 1.000000 2.000000 1.000000 2.236068 3.000000
```

som beräknar kvadratroten för varje element i vektorn.

Uppgift 1.9 (Summera vektorn)**.** Fyll i och kör följande rad för att beräkna summan av vektorn `new_vector`

```
sum(___)
```

Vid konstruktionen av vektorn användes också en grundläggande funktion - funktionen `c()`, som tar en serie värden och skapar en sammanhängande vektor av värden.

Alla R-funktioner har en tillhörande hjälpfil som kan plockas fram genom att skriva frågetecken följt av funktionsnamnet, t.ex. `?sum`. Från hjälpfilen får man att `sum()` tar numeriska vektorer som ingående värde och beräknar summan. Man kan styra funktionens beteende genom att sätta ett argument `na.rm` (vilket här styr hur funktionen hanterar saknade värden). Som illustration kan man titta på

```
b <- c(3, 1, 4, 1, 5, 9, NA)  # Lägger till ett saknat värde
sum(b)                        # na.rm = FALSE är grundinställning
```

```
## [1] NA
```

```
sum(b, na.rm = TRUE)          # na.rm sätts till TRUE
```

```
## [1] 23
```

Det första försöket `sum(b)` ger utfallet `NA`, men om man sätter `na.rm = TRUE` beräknas summan efter att det saknade värdet plockats bort. Notera också att skript kan kommenteras med `#`.

1.4 Sekvenser av funktioner

Ofta vill man genomföra flera operationer på ett objekt. Man behöver då genomföra en sekvens av funktioner. Säg till exempel att man har värdena

$$(-4, -2, -1, 1, 2, 4)$$

och vill ta absolutvärde (vilket gör negativa tal till motsvarande positiva tal) och sedan summera. Den typen av sekvenser kan genomföras på ett par olika sätt. Ett första sätt är att spara resultatet i varje steg och sedan använda utfallet i nästa steg:

```
c <- c(-4, -2, -1, 1, 2, 4)    # Skapa en vektor av värden
c_absolute <- abs(c)           # Ta absolutvärden. Skapa c_absolut
sum(c_absolute)                # Summera värden i c_absolut
```

```
## [1] 14
```

Här skapas ett objekt `c` som innehåller en vektor där några tal är negativa. I nästa rad används `abs` för att skapa absolutvärden. Slutligen summeras absolutvärdena med `sum`. Notera att det är möjligt att skapa ett objekt med namnet `c` trots att det redan är namnet på en funktion - R förstår ur sammanhanget om objektet eller funktionen ska användas. Det kan dock bli lite oklart för en läsare, så försök som regel att undvika att skapa objekt med vanliga funktionsnamn som `sum` och `mean`.

Uppgift 1.10 (Kvadrat, summa och roten ur)**.** Fyll i och kör följande rader för att ta varje värde i `new_vector` i kvadrat, *sedan* summera, och sedan ta roten ur.

```
new_vector_squared <- new_vector^2 # Ta kvadraten av varje värde
new_vector_squared_sum <- sum(___) # Summera kvadraterna
sqrt(___)                          # Ta kvadratroten ur summan
```

Ett alternativ till att spara utfallet i varje steg är att skriva en senare funktion *runt* en tidigare funktion. Det fungerar för att R utvärderar funktioner inifrån-ut. Med samma exempel som tidigare får man

```
sum(abs(c(-4, -2, -1, 1, 2, 4)))
# Ta summan av absolutvärden av vektorn
```

medan beräkningen i övningen blir

```
sqrt(sum(new_vector^2)) # Ta roten ur summan av vektorn i kvadrat
```

Den här typen av skrivning kan spara plats men blir snabbt svårläst.

Ett sista alternativ är att använda en så kallad *pipe* (namnet kommer från att en sekvens funktioner kallas en *pipeline*). En pipe skrivs `%>%` och kan i RStudio tas fram med kortkommandon *Ctrl + Shift + M*. Pipen tar utfallet av en funktion till vänster och sänder till en funktion till höger. Den kan utläsas i dagligt tal som *och sen*. Med samma exempel som tidigare kan vi skriva

```
library(tidyverse)

c(-4, -2, -1, 1, 2, 4) %>%        # Skapa en datamängd och sen
  abs() %>%                       # ta absolutvärden, och sen
  sum()                           # beräkna summan.
```

```
## [1] 14
```

Uppgift 1.11 (Kvadrat, summa och rot med pipe)**.** Fyll i de saknade funktionerna och kör följande rader för att ta varje värde i `new_vector` i kvadrat, *sedan* summera, och sedan ta roten ur, denna gång genom att länka funktionerna med en pipe `%>%`.

```
new_vector^2 %>% # Ta kvadraterna av new_vector, och sen
  ___() %>%      # beräkna summan, och sen
  ____()         # Ta kvadratroten med sqrt()
```

1.5 Datainskrivning

Det första praktiska steget i en statistisk analys är att importera data. I R kan det göras genom att direkt skriva in sin data och spara som ett nytt objekt, men ett bättre och vanligare sätt är att importera sin data från en extern fil eller databas.

I ett tidigare exempel användes funktionen `c()` för att skapa en vektor av data. Ofta ordnas flera vektorer i en tabell där varje kolumn är en vektor och varje rad en observation av någon enhet. En datatabell (en `data.frame` i R) skapas genom funktionen `data.frame()` följt av namngivna vektorer. Exempeldata kan skrivas in genom föjande.

```
dat <- data.frame(Vecka = c(7,7,7,7,7,7,
                            11,11,11,11,11,11),
                  Behandling = c("A","A","A","B","B","B",
                                 "A","A","A","B","B","B"),
                  Vikt = c(232,161,148,368,218,257,
                           1633,2213,972,2560,2430,855),
                  N = c(2.63,2.90,2.99,3.54,3.30,2.85,
                        1.53,1.90,NA,2.58,NA,NA))
dat
```

Radbrytningar och blanksteg är oviktiga i R, och används bara för läsbarhet här. Saknade värden skrivs in som `NA` för *not available*. Notera att alla kolumner inte behöver vara av samma datatyp men att värden inom en kolumn måste vara det. Här är *Behandling* text medan övriga kolumner är numeriska.

Uppgift 1.12 (Alea iacta est)**.** Kasta din tärning tio gånger och skriv in resultatet i en datatabell i R med hjälp av grundkoden nedan. Om du saknar en

tärning, fråga lämplig person om du kan få en. Behåll tärningen, den behövs till nästa datorövning (och närhelst man står inför ett avgörande livsbeslut).

```
dat_dice <- data.frame(Kast = c(1,2,3,4,5,6,7,8,9,10),
                       Utfall = c(_,_,_,_,_,_,_,_,_,_))
dat_dice
```

Objektet `dat` är av typen `data.frame` - en tabell med rader och kolumner. Man kan ange en specifik kolumn i en data.frame med dollartecken följt av kolumnens namn.

```
dat$Vikt
```

```
##  [1]  232  161  148  368  218  257 1633 2213  972 2560 2430  855
```

Man kan också plocka ut rader och kolumner med hakparenteser och ordningstal.

```
dat[2,3]          # Andra raden och tredje kolumnen
dat[2, ]          # Tomt värde ger alla värden.
dat[ ,3]          # Alla värden i kolumn 3
```

Uppgift 1.13 (Plocka ut en specifik kolumn)**.** I den tidigare övningen skapade du ett objekt `dat_dice`. Använd dollartecken för att plocka ut kolumnen *Utfall* från det objektet.

```
dat_dice$___
```

Genom att plocka ut en kolumn från en data.frame kan man beräkna vanlig beskrivande statistik med funktioner som `mean()` (medelvärde) och `sd()` (standardavvikelsen).

```
mean(dat$Vikt)
```

```
## [1] 1003.917
```

```
sd(dat$Vikt)
```

```
## [1] 951.3067
```

Funktionen `plot()` ger en enkel graf.

```
plot(dat$Vecka, dat$Vikt)
```

Uppgift 1.14 (Tärningsgraf)**.** Använd datan i objektet `dat_dice` och skapa ett diagram med kolumnen kast på x-axeln och kolumnen Utfall på y-axeln.

```
plot(dat_dice$___, dat_dice$___)
```

1.6 Urval ur en tabell med `select` och `filter`

En vanlig operation på en tabell är att göra ett urval - antingen ett urval av rader (t.ex. ett visst land), vilket kallas *filtrering* eller ett urval av variabler (t.ex. land och befolkning), vilket kallas *selektion*. Här tittar vi på hur det kan göras med funktionerna `filter()` och `select()` från paketet `tidyverse`. Vi använder gapminder-datan som kan laddas med `library(gapminder)`.

För att filtrera på ett givet land kan man använda pipe-funktionen från datan till en filter-funktion, t.ex.

```
gapminder %>%                       # Ta gapminder-datan och sen
  filter(country == "Sweden")       # filtrera för ett land
```

Inom filterfunktionen anges ett logisk villkor `country == Sweden"` och utfallet är de rader där villkoret är sant. Notera de dubbla likhetstecknen - de måste användas för ett logisk villkor eftersom enkelt likhetstecken används för att skapa objekt och sätta funktionsargument.

Uppgift 1.15 (Filtrera för land)**.** Vad måste ändras i koden för att istället plocka ut rader där landet är Finland? Hur många rader har det urvalet i datan?

```
gapminder %>%                       # Ta gapminder-datan och sen
  filter(country == "Sweden")       # filtrera för ett land
```

Om man vill välja flera länder kan man använda funktionen `%in%` på ett liknande sätt.

```
gapminder %>%
  filter(country %in% c("Sweden", "Finland"))
```

och om man vill ha mer än ett villkor kan man rada dem i filter-funktionen:

```
gapminder %>%                                   # Ta datan, och sen
  filter(country %in% c("Sweden", "Finland"), # filtrera för land
         year == 2002)                          # och för år
```

För att se fler eller färre rader kan man använda en pipe `%>%` till funktionen `print()`. Följande skriver ut fem rader

```
gapminder %>%
  filter(country %in% c("Sweden", "Finland")) %>%
  print(n = 5)
```

Om man istället vill göra ett urval av kolumner kan man använda `select()`. Som argument anges de kolumner man vill välja, t.ex.

```
gapminder %>%                   # Ta datan, och sen
  select(country, lifeExp)      # välj kolumner
```

Uppgift 1.16 (Befolkade länder)**.** Funktionen `arrange()` sorterar data efter

en angiven kolumn. Följande stycke ger oss Europas länder efter befolkning.

```
gapminder %>%
  filter(continent == "Europe", year == 1962) %>%
  select(country, lifeExp, pop) %>%
  arrange(-pop)
```

Gör lämpliga ändringar för att få Asiens länder från 2002 ordnade efter förväntad medellivslängd.

1.7 Grafer med ggplot2

Vi kan nu börja titta på grafer. R har en mängd grundläggande funktioner för grafer. Vi såg tidigare ett exempel på funktionen `plot()`.

```
gm_2002 <- gapminder %>% filter(year == 2002)
```

```
plot(gm_2002$lifeExp, gm_2002$gdpPercap)
```

Ett object `gm_2002`skapas efter att ha filtrerat för året 2002. Tecknet `$` används för att välja kolumnerna lifeExp och gdpPercap ur objektet `gm_2002`.

För mer avancerade grafer används dock ofta funktioner ur Rs paketbibliotek. Här illustreras det mest populära - `ggplot2`. I `ggplot2` byggs grafer upp med tre grundläggande byggstenar:

- *data*, informationen man vill visualisera,
- *aestethics*, en koppling mellan data och visuella element såsom grafens axlar, objekts storlek och färg,
- *geometries*, de geometriska former som visas i grafen.

En graf skrivs med en startfunktion `ggplot` som anger namnet på datan och grafens *aestethics*, och därefter sätts geometriska element genom funktioner som börjar med `geom_`. Ett spridningsdiagram kan t.ex. skapas med `geom_point`.

```
ggplot(gm_2002, aes(x = lifeExp, y = gdpPercap)) +
  geom_point()
```

Grafen kan byggas ut genom att sätta *aestethics* för färg och storlek. Man kan också dela en graf i småfönster med `facet_wrap` och styra grafens utseende genom att sätta ett tema såsom `theme_bw`.

```
ggplot(gm_2002, aes(x = lifeExp, y = gdpPercap,
                    color = continent, size = pop)) +
  geom_point() +
  facet_wrap(~ continent)
```

Uppgift 1.17 (Livslängd över tid)**.** Vad ska ändras i stycket nedan för att skapa en graf med år på x-axeln, medellivslängd på y-axeln och skilda småfönster för olika kontinenter?

```
ggplot(gm_2002, aes(x = ____, y = ____)) +
  geom_point() +
  facet_wrap(~ continent)
```

Andra graftyper kan skapas med andra `geom_`-funktioner. Stapeldiagram ges av `geom_col` (`col` för *column*). Man kan också använda `geom_bar` om man bara vill räkna antal rader per någon kategori. Följande väljer ut en kontinent och år och plottar ländernas befolkning stapeldiagram.

```
dat_small <- gapminder %>%
  filter(continent == "Europe", year == "2002")

ggplot(dat_small, aes(pop, country, fill = gdpPercap)) +
  geom_col(color = "black")
```

Argumentet `fill` styr färgen för ytor (här staplarnas ytor) medan `color` i `geom_col()` styr kanten runt varje stapel.

Man kan styra grafiken i en `ggplot` genom funktionen `theme()`. Det är ett ganska komplicerat ämne, men låt oss titta på några grunder. Vi börjar med att skapa en enkel graf: en boxplot över medellivslängd per kontinent.

```
dat_small <- gapminder %>%
  filter(year == 2002)

ggplot(dat_small, aes(x = lifeExp, y = continent)) +
  geom_boxplot()
```

Vi kan ändra utseendet på grafen genom argument inom geometrier och med funktionen `theme()`. I `theme()` sätter man de specifika egenskaper man vill ändra genom att tillskriva dem ett *element*. Valet av element beror på typen av grafiskt objekt - text sätts t.ex. med `element_text()` och ytor med `element_rect()` (för *rectangle*). Vi ger ett exempel med ändrad bakgrund, rutmönster, och teckenstorlek.

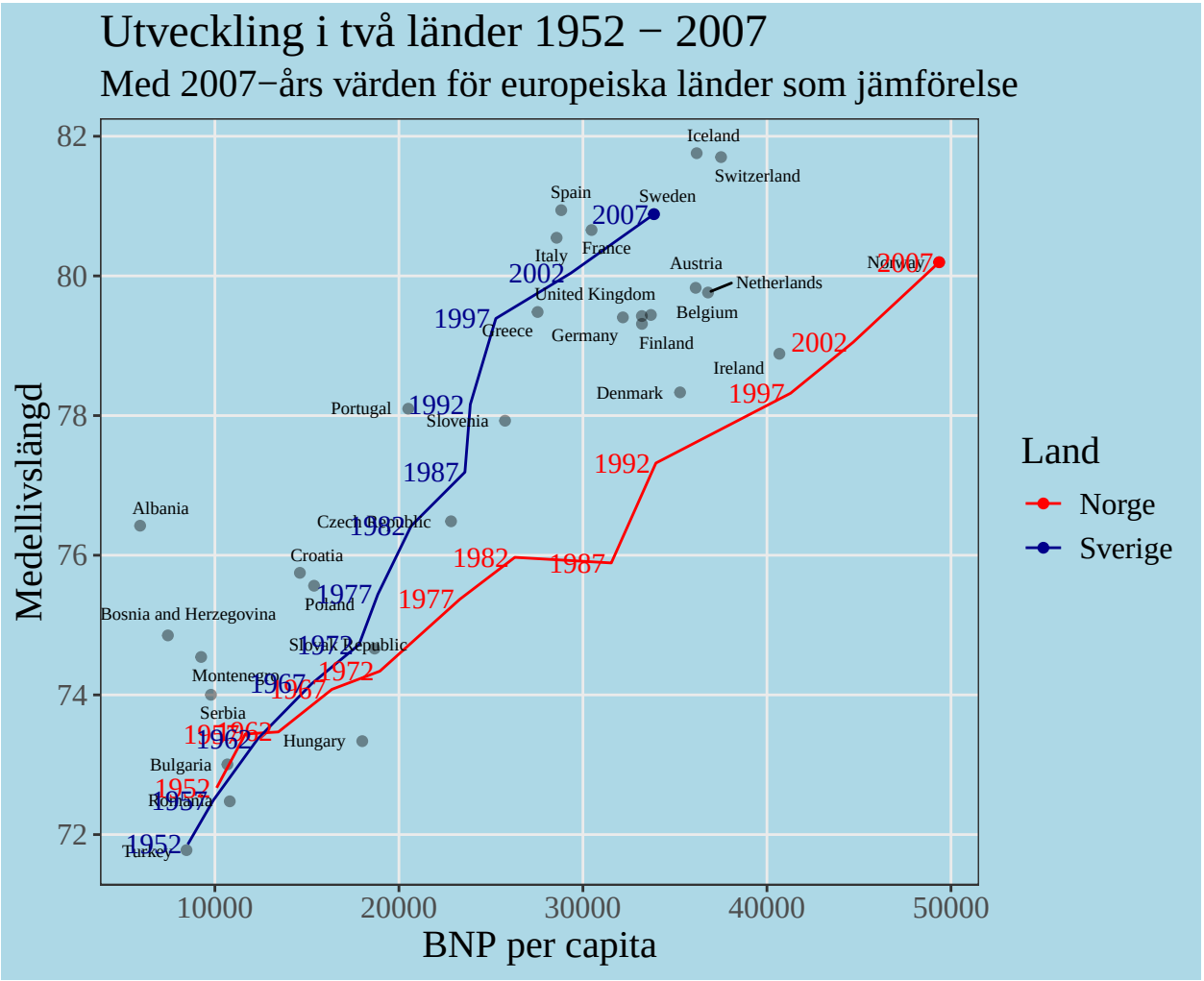
```
ggplot(dat_small, aes(lifeExp, continent)) +
  geom_boxplot(fill = "lightblue") +
  theme(panel.background = element_rect(fill = "red3"),
        text = element_text(size = 15,
                            color = "white", family = "serif"),
        axis.text = element_text(color = "white"),
        plot.background = element_rect(fill = "grey30",
                                       color = "black"),
        panel.grid.major.y = element_blank())
```

Uppgift 1.18 (Temaval 1)**.** Ändra färgvalen i grafen ovan för att skapa snyggast möjliga graf. Funktionen `colors()` ger de färger som finns tillängliga i R. Man kan också använda hex-koden för färger, t.ex. `fill = "#ffdd00"`.

Uppgift 1.19 (Temaval 2)**.** Ändra färgvalen i grafen ovan för att skapa fulast möjliga graf. Visa de två graferna för någon annan och se om de kan säga vilken som är vilken.

1.8 AI-uppgift I

Nedan har vi en graf gjord med `ggplot2` med flera olika *geom* och ändringar från standardtemat.



Uppgift 1.20 (Dekonstruera grafen)**.** Vilka geografiska former (punkter, linjer ytor) visas i grafen och hur är de kopplade till datans variabler? Vilka ändringar har gjorts från standardtemat (bakgrundsfärg, typsnitt, teckenstorlek)?

Uppgift 1.21 (Rekonstrera grafen)**.** Använd en chatbot (såsom Microsofts copilot eller OpenAIs ChatGPT) för att rekonstrera grafen. Börja med att beskriva grafen i ord och be om R-kod. Klipp ut koden och kör i R. Läs koden du fått. Finns det några delar som är svåra att förstå. Med vissa chatverktyg kan man också klippa in bilden och be om den bakomliggande koden.

Uppgift 1.22 (Steg i Excel)**.** Hur skulle man gå tillväga för att skapa samma graf i Excel? Be en chatbot om en stegvis instruktion.

Kapitel 2

Beskrivande statistik

Datorövning 2 handlar om beräkning av beskrivande statistik i R. Efter övningen ska vi kunna

- Importera data från en excelfil,
- Beräkna lämpliga lägesmått för en variabel,
- Beräkna lämpliga spridningsmått för en variabel,
- Konstruera grafer som jämför två eller flera gruppers läge och spridning.

2.1 Repetition från datorövning 1

När man arbetar i R är det klokt att använda ett avancerat gränssnitt som RStudio och att skriva sin kod i ett separat skript. I RStudio kan man starta ett nytt skript genom *Ctrl + Shift + N*.

Mycket funktionalitet i R ligger i tilläggspaket (*Packages*). Paket måste installeras första gången de används och laddas varje session de används, t.ex.

```
# install.packages("tidyverse") # Installera tidyverse
library(tidyverse)              # Ladda ett paket
```

Objekt skapas med *assign*-pilen `<-`. Det är ett sätt att importera data till R. Det är dock vanligare importera från en extern fil. Mer om det senare.

Vi arbetar nästan alltid med data på tabellformat där variablerna är kolumner och observationerna rader. I gapminder-datan ges raderna av ett land för ett visst år.

```
library(gapminder)
gapminder                       # Skriv ut objektet gapminder
```

Funktioner agerar på objekt och ger något utfall. Här nedan beräknas medelvärdet av livslängd med funktionen `mean()`. Dollartecknet används för att ange

en specifik kolumn i dataobjektet. Funktioner styrs av möjliga argument - här används `na.rm` för att ange att saknade värden inte ska tas med i beräkningen

```
mean(gapminder$lifeExp, na.rm = T) # Beräkna medel av lifeExp
```

```
## [1] 59.47444
```

Funktionerna `filter()` och `select()` kan användas för att välja kolumner och rader. Funktioner kan länkas samman med en pipe `%>%` för att skapa sekvenser av funktioner. Man kan tänka på pipen som *och sen*.

```
gapminder %>%
  filter(country == "Norway", year > 1972) %>%
  select(country, lifeExp)
```

Slutligen tittade vi på grafer med ggplot2-paketet. En ggplot byggs upp med tre grundelar: data, geometrier (grafens objekt och former), och *aesthetics* (utseende och placering av geometrierna). I ett enkelt spridningsdiagram är data två numeriska variabler, geometrierna är punkter, och punkternas placering ges av en x-koordinat och en y-koordinat. Ytterligare *aesthetics* kan vara punkternas färger (`color`) och storlek (`size`).

```
dat_small <- gapminder %>% filter(year == 2007)
ggplot(dat_small, aes(x = gdpPercap, y = lifeExp,
                      size = pop, color = continent)) +
  geom_point()
```

2.2 Import av data från en Excelfil

Inom vetenskapen är Excel det vanligaste filformatet för mindre datamängder. Till den här delen ska vi återigen arbeta med data från *Gapminder*.

Uppgift 2.1 (Excelfil från Canvas)**.** Hitta excelfilen *Gapminder.xlsx* på Canvas och ladda ner den. Hitta mappen som filen laddats ned till.

I R kan man läsa in data från en Excel-fil med funktionen `read_excel()` från paketet `readxl`. Som argument till funktionen sätts filens sökväg - dess placering på hårddisken. Stycket nedan importerar från en excelfil som ligger på hårddisken *C:* i mappen *Downloads*, under *User_name*, under *Users*.

```
library(readxl)  # Ladda readxl
```

```
gapminder <- read_excel("C:/Users/Name/Downloads/Gapminder.xlsx")
# Läs in från en lokal excelfil
gapminder
```

Uppgift 2.2 (Importera från excelfil)**.** Var ligger den nedladdade filen *Gapminder.xlsx*? Gör lämplig ändring i koden ovan för att läsa in data från den filen. Notera att R använder högerlutande snedstreck `/`, så om en kopierad sökväg har

vänster-snedstreck måste de ändras. Kontrollera att datan blivit korrekt inläst genom att köra objektnamnet `gapminder`.

En R-session har alltid en grundmapp, ett *Working directory*. Man kan se vilken mapp det är genom att köra

```
getwd() # Ange working directory
```

En filsökväg kan anges antingen som en fullständig sökväg, som ovan, eller relativt *working directory*. Om man till exempel har en fil *Gapminder.xlsx* som ligger i en mapp *Data* som i sin tur ligger i *working directory*, kan man importera data från filen med

```
gapminder <- read_excel("Data/Gapminder.xlsx")
# Läs in från en lokal excelfil (relativt wd)
gapminder
```

Uppgift 2.3 (Working directory)**.** Identifiera *working directory* för din nuvarande Rs-session genom att köra `getwd()`.

RStudio har också en inbyggd funktionalitet för att importera data. Man kan hitta den genom att gå till *Environment*-fliken och sedan *Import Dataset*. Det kan vara en bra hjälp, i synnerhet om man vill sätta datatyp för någon specifik kolumn.

Om du inte har tillgång till Canvas kan Gapminder-datan alternativt hämtas från paketet Gapminder.

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

2.3 Ändra och skapa nya kolumner med `mutate`

Variabler kan omräknas och nya variabler kan skapas med `mutate`-funktionen. I gapminder-datan finns bnp per capita. Om man vill ha nationell BNP kan man skapa en ny kolumn och beräkna den som bnp per capita gånger populationen.

```
gapminder <- gapminder %>%              # Ta datan, och sen
  mutate(gdp = gdpPercap * pop)         # Beräkna bnp
```

Den inledande delen med `gapminder <-` gör så att utfallet av beräkningen sparas i objektet `gapminder`. Vi kan skriva ut objektet och se resultatet av beräkningen:

```
gapminder
```

Om man vill skapa en kolumn med mellanrum i namnet måste man skriva namnet inom *backticks* ‘ för att ange att namnet ska tolkas som en enhet. Jag

rekommenderar att undvika mellanrum i kolumnnamn och istället använda stora bokstäver eller understreck för ett nytt ord (`NationalGDP` eller `National_GDP`).

```
gapminder <- gapminder %>%
  mutate(`National GDP` = gdpPercap * pop)
gapminder
```

2.4 Sammanfattande lägesmått

Den importerade datan anger medellivslängd, populationsstorlek och bnp per capita per land och år. Vi kan börja med att producera en bubbelgraf över datan - en av de presentationer Gapminder ofta använder. En bubbelgraf är ett spridningsdiagram där punktens storlek beror på en tredje variabel.

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp,
                      size = pop, color = continent)) +
  geom_point() +
  facet_wrap(~ year)
```

En interaktiv version kan vara bra om man vill identifiera någon specifik punkt.

```
g <- ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, size = pop,
                           color = continent, text = country)) +
  geom_point() +
  facet_wrap(~ year)

# install.packages("plotly")  # Kör om ej installerat tidigare
library(plotly)               # Ladda paketet plotly
ggplotly(g)                   # Interaktiv graf g
```

Under föreläsningen såg vi exempel på två lägesmått: medelvärdet (egentligen det *aritmetiska* medelvärdet) och medianen. De har bägge enkla funktioner i R: `mean()` respektive `median()`. Vi plockar ut en variabel ur datan och beräknar bägge.

```
gapminder$gdpPercap          # Vektorn med data
mean(gapminder$gdpPercap)    # Beräkna medelvärdet
median(gapminder$gdpPercap) # Beräkna medianen
```

Samma sak kan göras med en pipe `%>%` och `summarise()`.

```
gapminder %>%
  summarise(Mean = mean(gdpPercap),
            Median = median(gdpPercap))
```

Uppgift 2.4 (Lägesmått av livslängd)**.** Gör lämpliga ändringar i exemplet ovan för att beräkna lägesmått för medellivslängd (`lifeExp`).

Den andra lösningen, med en pipe och `summarise()`, kan enkelt utvecklas med

ett `group_by()`-steg för att beräkna medel och median per någon grupp, t.ex. per år.

```
gapminder %>%                           # Ta datan, och sen
  group_by(year) %>%                    # gruppera efter år, och
  summarise(Mean = mean(gdpPercap),     # beräkna medelvärde och
            Median = median(gdpPercap)) # medianen av gdpPercap
```

Uppgift 2.5 (Lägesmått per kontinent)**.** Gör lämpliga ändringar i exemplet ovan för att beräkna lägesmått per kontinent. Vad måste läggas till för att också beräkna maximum och minimum per kontinent (funktionerna `max()` och `min()`)?

Uppgift 2.6 (Upprepade mätningar)**.** Finns det några problem med att beräkna medelvärde per kontinent på den här datan?

I vetenskapliga publikationer redovisas medelvärden ofta med ett stapeldiagram. Som exempel ges staplar för medelvärdet av bnp per kontinent för 2007.

```
dat_gdp_2007 <- gapminder %>%       # Ta datan, och sen
  filter(year == 2007) %>%          # filtrera för 2007, och sen
  group_by(continent) %>%           # gruppera kontinenter, och
  summarise(Mean = mean(gdpPercap)) # beräkna medelvärdet

ggplot(dat_gdp_2007, aes(continent, Mean)) +
  geom_col()                  # Illustrera med kolumner (columns)
```

Uppgift 2.7 (Graf för livslängd)**.** Gör om stapeldiagrammet. Denna gång med livslängd (`lifeExp`) istället för bnp per capita (`gdpPercap`).

2.5 Sammanfattande spridningsmått

Under föreläsningarna såg vi några mått på spridning: varians, standardavvikelse och kvartilavstånd (IQR, *inter-quartile range*). De har alla motsvarande funktioner i R (`var()`, `sd()`, och `IQR()`) som kan användas på samma sätt som funktionerna för lägesmått.

```
gapminder$gdpPercap      # Vektorn med data

var(gapminder$gdpPercap) # Beräkna varians
sd(gapminder$gdpPercap)  # Beräkna standardavvikelse
IQR(gapminder$gdpPercap) # Beräkna kvartilavstånd
```

Alternativt med en pipe och `summarise()`.

```
gapminder %>%                       # Ta datan, och
  summarise(Var = var(gdpPercap),   # beräkna varians,
            Sd = sd(gdpPercap),     # standardavvikelse,
            IQR = IQR(gdpPercap))   # och kvartilavstånd
```

Lösningen med pipe och `summarise()` kan som tidigare utvecklas med `group_by()`.

```
gapminder %>%
  group_by(year) %>%
  summarise(Varians = var(gdpPercap),
            Standardavvikelse = sd(gdpPercap),
            Kvartilavstånd = IQR(gdpPercap))
```

Uppgift 2.8 (Graf för livslängd)**.** Gör lämpliga ändringar i det sista exempel för att istället beräkna spridningsmått för livslängd.

Vi avslutar med tre vanliga illustrationer av vetenskaplig data - ett linjediagram med felstaplar, ett stapeldiagram med felstaplar, och ett lådagram. För linjediagrammet beräknar vi medelvärdet och spridningsmått för bnp över år och kontinent. Som spridningsmått använder vi standard error (sv. *medelfel*), vilket beräknas som standardavvikelse delat på roten ur antalet observationer. Funktionen `n()` ger antalet observationer per kontinent och år.

```
dat_year_continent <- gapminder %>%
  group_by(year, continent) %>%
  summarise(Mean = mean(gdpPercap),
            SE = sd(gdpPercap) / sqrt(n()))
dat_year_continent
```

Med `ggplot2` kan vi bygga ett linjediagram med `geom_line()` och lägga till felstaplar med `geom_errorbar()`. Den senare behöver `aes()`-argument för `ymin` och `ymax` - nedre och övre del av felstapeln. Vi sätter dem till medelvärdet minus respektive plus ett medelfel.

```
ggplot(dat_year_continent, aes(x = year, y = Mean,
                               color = continent)) +
  geom_line() +
  geom_errorbar(aes(ymin = Mean - SE, ymax = Mean + SE))
```

Uppgift 2.9 (Bredd)**.** Felstaplarna från `geom_errorbar()` har väldigt breda ändar. Använd hjälpsidan för geomet `?geom_errorbar`, i synnerhet exemplen längst ned, och se om det går att ändra bredden.

En graf med staplar och felstaplar kan konstrueras på ett liknande sätt. Följande exempel visar staplar över livslängd per kontinent. Felstapeln ges av standardavvikelsen.

```
dat_2007 <- gapminder %>%          # Ta datan, och sen
  filter(year == 2007) %>%        # filtrera på år, och sen
  group_by(continent) %>%         # gruppera efter kontinent,
  summarise(Mean = mean(lifeExp), # summera med medelvärde,
```

```
            SD = sd(lifeExp))     # och standardavvikelse
dat_2007
```

Vi bygger en ggplot med `geom_col()` och `geom_errorbar()`. Felstapels konstruktion kan anges i en notis med funktionen `labs()`.

```
ggplot(dat_2007, aes(continent, Mean, fill = continent)) +
  geom_col()+
  geom_errorbar(aes(ymin = Mean - SD, ymax = Mean + SD)) +
  labs(title = "Average life expectancy by continent, 2007",
       caption = "Errorbars given by mean +/- standard deviation.
       Source: Gapminder")
```

Uppgift 2.10 (Staplar för 1982)**.** Gör lämpliga ändringar i exempel ovan för att konstruera ett stapeldiagram med felstaplar för året 1982 och variabeln medellivslängd.

Ett lådagram anger fördelningen av en variabel genom att illustrera minimum, maximum och kvartiler. Kvartiler är mått som delar en datamängd i fyra lika stora delar (så att en fjärdedel ligger under första kvartilen, en fjärdedel mellan första och andra kvartil, och så vidare). Med `ggplot2` kan vi bygga ett lådagram med `geom_boxplot()`. Exempel ger en låda per år och kontinent.

```
ggplot(gapminder, aes(year, lifeExp, fill = continent,
                      group = year)) +
  geom_boxplot() +
  facet_wrap(~ continent)
```

Uppgift 2.11 (Group-argumentet)**.** I lådagrammet används argumentet `group`. Vad gör det? Vad händer om man tar bort det?

2.6 Ordna upp beskrivande statistik och exportera

Efter att ha beräknat någon beskrivande statistik kan det vara bra att titta på hur resultaten kan snyggas upp och exporteras i något lämpligt format. Ta den tabell med medelvärden vi producerade i ett tidigare exempel.

```
dat_2007 <- gapminder %>%          # Ta datan, och sen
  filter(year == 2007) %>%         # filtrera på år, och sen
  group_by(continent) %>%          # gruppera efter kontinent,
  summarise(Mean = mean(lifeExp), # summera med medelvärde,
            SD = sd(lifeExp))      # och standardavvikelse
dat_2007
```

Objekt kan exporteras från R på liknande som det importeras - med särskilda exportfunktioner (*write*-funktioner) beroende på filtyp. För att exportera till en

csv-fil man man använda `write_csv()`. Ingående argument är det objekt man vill exportera och den fil man vill exportera till. R ger ingen varning om man skriver över en existerande fil, så var lite försiktiga här.

Precis som vid import använder R *working directory* om inget annat anges. Följande exporterar objektet `dat_2007` till en csv-fil *Exporterad data från R.csv* i *working directory*.

```
getwd()                              # Se nuvarande working directory
write_csv(dat_2007, "Exporterad data från R.csv") # Exportera data
```

Därifrån skulle man kunna öppna filen i något kalkylprogram, snygga till layouten, och sedan klippa in i ett textdokument.

2.7 Kumulativt medelvärde

Om man har data som av någon anledning samlas in i sekvens kan det vara intressant att beräkna och illustrera den med ett *kumulativt medelvärde*. En serie med kumulativa medelvärden beräknas genom att för varje nytt värde ta medelvärden av de värden man hittills samlat in - vid tio värden tar man medelvärdet av de tio, vid elva värden medelvärdet av de elva, och så vidare.

Med de tärningsvärden vi hade innan kan vi beräkna ett kumulativt medelvärde genom att först beräkna summan med `cumsum()` och sedan dela på antalet kast. För att förenkla beräkningen av antalen tar vi fram sekvensen med antal kast i ett `mutate()`-steg.

```
dat_dice <- data.frame(Utfall = c(6,3,2,3,5)) %>%
  mutate(Kast = 1:n())
dat_dice

dat_dice <- dat_dice %>%
  mutate(kumulativ_summa = cumsum(Utfall),
         kumulativt_medelvärde = kumulativ_summa / Kast)
dat_dice
```

Uppgift 2.12 (Kumulativt medelvärde)**.** Vad ska läggas till för att stycket nedan ska ge en linjegraf över medelvärdet?

```
ggplot(dat_dice, aes(x = Kast, y = ___)) +
  ___()
```

Uppgift 2.13 (Fler tärningskast)**.** Kasta din tärning ytterligare några gånger, gärna på en mjuk yta. Fyll i dina utfall och gör grafen från föregående uppgift. Kan man se en tendens för medelvärdet att minska i varians vid fler kast? Svänger kurvan in mot något specifikt värde?

```
dat_dice <- data.frame(Utfall = c(___)) %>%
  mutate(Kast = 1:n(),
```

```
         kumulativ_summa = cumsum(Utfall),
         kumulativt_medelvärde = kumulativ_summa / Kast)
dat_dice
```

Uppgift 2.14 (Kumulativ frekvens)**.** Om man vill titta på andelen gånger ett visst utfall inträffat talar man om *kumulativ frekvens* snarare än *kumulativt medelvärde*. Använd stycket nedan för att titta på andelen gånger utfallet varit en etta (ett *positivt* utfall, i begreppets kliniska mening). Om ett inte är ett möjligt utfall på din tärning, ändra ettan till något mer lämpligt.

```
dat_dice <- data.frame(Utfall = c(___)) %>%
  mutate(Kast = 1:n(),
         positivt_utfall = Utfall == 1,
         kumulativt_antal = cumsum(positivt_utfall),
         kumulativ_frekvens = kumulativt_antal / Kast)
dat_dice

ggplot(dat_dice, aes(x = Kast, y = kumulativ_frekvens)) +
  geom_line()
```

2.8 Tredimensionella grafer med `plotly`

De två avslutande avsnitten är mindre viktiga och kan läsas i mån av tid.

Paketet `plotly` kan användas för att göra interaktiva graf och grafer med tre dimensioner. Börja med att ladda paketet.

```
library(plotly)
```

Vi börjar med ett enkelt exempel på en 3d-graf med lite skapad data.

```
dat_ex <- data.frame(Var1 = c(1,2,3), Var2 = c(3,1,2),
                     Var3 = c(2,3,1), Type = c("A", "B", "C"))
dat_ex

plot_ly(dat_ex, x = ~Var1, y = ~Var2,
        z = ~Var3, color = ~Type) %>%
  add_markers()
```

Om grafen inte kommer upp direkt kan det fungera att trycka på den lilla ikonen med ett fönster och en pil i *Viewer*-fliken. Grafen ska då öppnas i en webbläsare.

Syntaxen till `plot_ly()` är inte helt olik `ggplot()`. Först anges datan, därefter argument för x- y-, och z-koordinater. Notera tilde-tecknet `~` före variabelnamnen. Eventuell färg sätts med `color`. Efter det lägger man till punkter (här *markers*) med en pipe in i `add_markers()`. Vi vill göra en liknande graf med gapminder-datan, men får börja med att filtrera på ett visst år.

Uppgift 2.15 (Filtrera för år)**.** Vad måste läggas till i funktionen nedan för att filtrera för data där året är 2007?

```
dat_2007 <- gapminder %>%
  filter(year == ___)
```

Vi kan nu konstruera en 3d-graf med datan.

Uppgift 2.16 (Gapminder i 3d)**.** Vad måste läggas till i funktionen nedan för en 3d-graf med befolkningsmängd (`pop`) på x-axeln, livslängd (`lifeExp`) på y-axeln, bnp per capita (`gdpPercap`) på z-axeln, och färg efter kontinent (`continent`)? För att kunna identifiera specifika länder kan man också sätta argumentet `text`.

```
plot_ly(data_2007, x = ~pop, y = ~___, z = ~___,
        color = ~continent, text = ~country) %>%
  add_markers()
```

Uppgift 2.17 (Log-transformationer)**.** Inom statistiken är det vanligt att transformera variabler för att ta bort extremeffekter och visa på specifika dataegenskaper. En vanlig transform är att *logaritmera* ett värde, vilket innebär att man istället för att använda det ursprungliga värdet använder exponenten i någon bas (ofta basen tio). Ta till exempel värdet 10000, dess tio-logaritm är 4, eftersom 10 upphöjt i 4 är 10000. Logaritmer är vanliga vid data med extremvärden.

Grafen i uppgiften ovan präglas mycket av skillnader i bnp och befolkningsstorlek. Testa att tio-logaritmera variablerna och se om det blir en mer eller mindre överskådlig graf. Logaritmen kan göras genom att byta den ursprungliga variabeln mot en variabel transformerad med `log10()`. Fyll i stycket nedan.

```
plot_ly(dat_2007, x = ~log10(pop), y = ~log10(___),
        z = ~___, color = ~___, text = ~country) %>%
  add_markers()
```

Uppgift 2.18 (Följa ett land)**.** Likt en ggplot kan man lägga till graf-element. Här använder man dock en pipe för lägga till ett nytt element. Fyll i kodstycket nedan. Vad, om något, har lagts till i grafen?

```
plot_ly(dat_2007, x = ~log10(pop), y = ~log10(___),
        z = ~___, color = ~continent, text = ~country) %>%
  add_markers() %>%
  add_lines(data = gapminder %>% filter(country == "Costa Rica"))
```

2.9 AI-uppgift II

Uppgift 2.19 (Standardavvikelse från gapminder-datan)**.** Använd en chatbot (såsom Microsofts co-pilot eller OpenAIs ChatGPT) för att få kod som beräknar standardavvikelsen av medellivslängd från gapminder-datan. Fråga också om tolkning av standardavvikelsen och om det finns några specifika tolkningsproblem för just den här gapminder-datan.

Uppgift 2.20 (3d-graf återigen)**.** Använd en chatbot för att få kod som skapar en 3d-graf lik den vi skapade ovan. Se om det går att utveckla grafen ytterligare, t.ex. med en linje som går ner till grafens *botten*, eller linjer som kopplar samma länder till kontinentens medelvärde.

Kapitel 3

Tester vid ett eller flera stickprov

Datorövning 3 handlar om hypotestest och konfidensintervall för ett eller två stickprov. Efter övningen ska vi kunna

- identifiera passande test för ett stickprov numerisk eller binär data,
- identifiera passande test för två stickprov numerisk eller binär data,
- analysera fördelningen av frekvenser i kvalitativ data.

3.1 Repetition från datorövning 2

När man startar en ny R-session bör man ladda de paket man vet kommer behövas med `library()`. Om paket inte finns installerade måste man först köra `install.packages()`.

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

I datorövning 2 tittade vi på hur insamlade variabler kan sammanfattas med lägesmått och spridningsmått. Ett enkelt sätt att ta fram dem är att använda `summarise()` och ange de mått och variabler man vill använda. Vi hade uppe ett exempel på data från Gapminder som vi importerade från en excel-fil. Idag kommer vi istället arbeta med *Palmers pingvindata*, en vanlig vetenskaplig exempeldata. Datan finns tillgänglig i paketet `palmerpenguins`. Vi laddar paketet och skriver ut de första raderna med data.

```
install.packages("palmerpenguins")
library(palmerpenguins)
penguins
```

```
# install.packages("palmerpenguins")
library(palmerpenguins)
penguins
```

Vi kan länka samman steg med en *pipe* `%>%`. Här ser vi ett filter, en gruppering, och en summering. Inom `summarise()` beräknas beskrivande statistik med funktionerna `mean()`, `median()` och `sd()`.

```
penguins %>%
  drop_na() %>%
  filter(year == 2007) %>%
  group_by(species, sex) %>%
  summarise(`Vikt, medel` = mean(body_mass_g),
            `Flipperlängd, median` = median(flipper_length_mm),
            `Näbblängd, standardavvikelse` = sd(bill_length_mm))
```

Beskrivande mått sammanfattas ofta i någon enkel vetenskaplig graf. Två vanliga val är lådagrammet, som illustrerar kvartiler och möjliga extremvärden, och stapeldiagrammet med felstaplar. Vi ger först ett exempel på ett lådagram över vikt.

```
ggplot(penguins, aes(body_mass_g, species, fill = sex)) +
  geom_boxplot() +
  facet_wrap(~ year)
```

Därefter ett exempel på ett stapeldiagram med felstaplar för samma data. Felstapeln ges av standardavvikelsen.

```
dat_sum <- penguins %>%
  drop_na() %>%
  group_by(species, sex) %>%
  summarise(Mean = mean(body_mass_g),
            SD = sd(body_mass_g))
dat_sum

ggplot(dat_sum, aes(sex, Mean)) +
  geom_col() +
  geom_errorbar(aes(ymin = Mean - SD, ymax = Mean + SD),
                width = 0.3) +
  facet_wrap(~ species)
```

3.2 Allmänt om hypotestester

Ett hypotestest används för att värdera en hypotes i situationer med slumpmässig spridning. Testets centrala utfall är ett p-värde, som ger sannolikheten för datan om nollhypotesen är sann. Om datan inte är i linje med nollhypotesen

bör p-värdet bli lågt. Ofta används en förbestämd signifikansnivå på 0.05 och vid p-värden under det talar man om ett *signifikant* utfall.

Hypotestestet är en allmän konstruktion och beroende på datatyp landar man i något specifikt test. I dagens datorövning tittar vi på följande test:

- För ett stickprov (en grupp)
  - t-test vid normalfördelad data
  - z-test vid binär data
  - χ^2-test vid nominal data
- För två stickprov (jämföra två grupper)
  - t-test vid normalfördelad data
  - z-test vid binär data
  - χ^2-test vid nominal data

För att illustrera de här testerna på en lite mer hanterbar datamängd drar vi slumpmässigt trettio observationer från pingvindatan.

```
set.seed(437)
dat <- slice_sample(penguins, n = 30)
dat %>% print(n = 30)
```

```
## # A tibble: 30 x 8
##    species   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>     <fct>             <dbl>         <dbl>             <int>       <int>
##  1 Adelie    Biscoe             39.6          20.7               191        3900
##  2 Adelie    Dream              38.3          19.2               189        3950
##  3 Gentoo    Biscoe             44.5          14.7               214        4850
##  4 Gentoo    Biscoe             44.9          13.8               212        4750
##  5 Gentoo    Biscoe             49.1          14.5               212        4625
##  6 Adelie    Biscoe             38.2          20                 190        3900
##  7 Adelie    Dream              39.2          18.6               190        4250
##  8 Adelie    Biscoe             39.7          18.9               184        3550
##  9 Adelie    Biscoe             38.1          16.5               198        3825
## 10 Adelie    Torgers~           36.6          17.8               185        3700
## 11 Chinstrap Dream              45.2          16.6               191        3250
## 12 Adelie    Dream              37.3          17.8               191        3350
## 13 Gentoo    Biscoe             46.8          14.3               215        4850
## 14 Gentoo    Biscoe             48.7          15.1               222        5350
## 15 Gentoo    Biscoe             51.3          14.2               218        5300
## 16 Adelie    Torgers~           38.9          17.8               181        3625
## 17 Chinstrap Dream              50.9          19.1               196        3550
## 18 Chinstrap Dream              45.5          17                 196        3500
## 19 Adelie    Biscoe             37.7          16                 183        3075
## 20 Gentoo    Biscoe             46.2          14.5               209        4800
## 21 Gentoo    Biscoe             49.1          15                 228        5500
## 22 Gentoo    Biscoe             43.4          14.4               218        4600
## 23 Chinstrap Dream              51            18.8               203        4100
```

```
## 24 Chinstrap Dream              50.8          19                 210        4100
## 25 Gentoo    Biscoe             50.4          15.7               222        5750
## 26 Chinstrap Dream              54.2          20.8               201        4300
## 27 Gentoo    Biscoe             45.4          14.6               211        4800
## 28 Chinstrap Dream              46            18.9               195        4150
## 29 Gentoo    Biscoe             51.5          16.3               230        5500
## 30 Chinstrap Dream              51.3          19.9               198        3700
## # i 2 more variables: sex <fct>, year <int>
```

3.3 Ett stickprov

3.3.1 Test av medelvärde för normalfördelad data

Om man har en normalfördelad variabel och vill testa om populationens medelvärde är skilt från något hypotetiskt värde μ_0 kan man använda ett *t-test för ett stickprov*. För vårt urval av pingvindata vill vi se om vikten är skild från 4500 gram.

```
ggplot(dat, aes(body_mass_g, 0)) +
  geom_point() +
  geom_vline(xintercept = 4500, color = "red", linewidth = 2)
```

I grafen ser vi att värdena ligger jämt spridda kring 4500, så 4500 är nog ganska rimligt som medelvärde, men låt oss göra ett formellt test. I R kan ett t-test genomföras med `t.test()`.

```
t.test(dat$body_mass_g, mu = 4500)   # Ett t-test på variabeln body_mass_g
```

```
##
##  One Sample t-test
##
## data:  dat$body_mass_g
## t = -1.6203, df = 29, p-value = 0.116
## alternative hypothesis: true mean is not equal to 4500
## 95 percent confidence interval:
##  4006.069 4557.264
## sample estimates:
## mean of x
##  4281.667
```

Utskriften ger ett p-värde från vilket vi kan dra en slutsats. I det här fallet är p-värdet högt (över fem procent) så vi kan inte förkasta nollhypotesen (vilken är att populationsmedelvärdet är lika med 4500).

Vi tittar nu på stegen bakom t-testet. Ett t-test bygger, som alla hypotestest, på en serie steg:

1. sätt upp en *nollhypotes* och en *alternativhypotes*,

2. beräkna ett *testvärde* från en testfunktion,
3. identifiera en *testfördelning*,
4. beräkna ett *p-värde*, eller uppskatta ett genom att ställa testvärde mot ett kritiskt värde,
5. dra en klar *slutsats* om statistisk signifikans.

Vi vill testa om medelskörden är skild från 4500, så hypoteser ges av

- H_0: μ lika med 4500
- H_1: μ ej lika med 4500

Alternativhypotesen är tvåsidig - vi tittar både på möjligheten att populationsmedelvärdet är större och på möjligheten att det är mindre.

Uppgift 3.1 (Ensidig mothypotes)**.** Hur hade hypoteserna sett ut om vi ville testa om medelvärdet är *större* än 4500?

Vi kan ta fram stickprovsmedelvärde och standardavvikelse i R.

```
mean(dat$body_mass_g)
```

```
## [1] 4281.667
```

```
sd(dat$body_mass_g)
```

```
## [1] 738.064
```

Vårt mål är att testa ett medelvärde och det är rimligt att anta normalfördelning för den undersökta variabeln. Det lämpliga testet är då ett t-test för ett stickprov och testvärdet kan beräknas av en testfunktion som ges av det observerade stickprovet minus nollhypotesens värde, delat på standardavvikelsen delat på roten ur antalet observationer.

$$t = \frac{\bar{y} - \mu_0}{s/\sqrt{n}}$$

Uppgift 3.2 (Handräkning)**.** Beräkna t-värdet på miniräknare eller telefon. Finns det beräknade t-värdet i utskriften från `t.test()`?

Nästa steg är att identifiera testfördelning, det vill säga den slumpfördelning testvärdet följet om nollhypotesen är sann. I det här fallet är testfördelningen en t-fördelning med n - 1 frihetsgrader. Vi har trettio observationer, så antalet frihetsgrader blir 29. I R kan man ta fram täthetsfunktionen för en t-fördelning med `dt()` och fördelningsfunktionen med `pt()`. Kurvorna kan plottas med `geom_function()` i en ggplot.

```
ggplot() +
  geom_function(fun = dt, args = list(df = 29)) +
  xlim(-5,5)
```

```
ggplot() +
  geom_function(fun = pt, args = list(df = 29)) +
  xlim(-5,5)
```

Från testfördelningen kan vi nu ta fram ett p-värde. P-värdet kan illustreras som ytan under t-fördelning *bortom* test-värdet. I ett tvåsidigt test tar vi med bägge svansarna.

```
ggplot() +
  geom_function(fun = dt, args = list(df = 29)) +
  geom_vline(xintercept = c(-1.6203, 1.6203),
             color = "red") +
  xlim(-5,5)
```

Ytorna i *svansarna* motsvarar p-värdet. De anger sannolikheten att få ett större t-värde än det vi fått, under antagandet att nollhypotesen stämmer. Det exakta p-värdet gavs av `t.test()`-funktionen som 0.116. En tolkning av det är om försöket upprepas ett stort antal gånger och nollhypotesen är sann, kommer vi 11.6 procent av gångerna få ett större testvärde än det observerade. Det vi observerar är ganska sannolikt om nollhypotesen stämmer, vilket tyder på att nollhypotesen är rimlig.

Det avslutande steget är att dra en formell slutsats och ge ett tydligt svar. Det beräknade p-värdet ställs mot en förbestämd signifikansnivå, oftast fem procent. Här är p-värdet över den nivån, så vi kan inte förkasta nollhyptesen. Slutsatsen är att det *inte* finns någon signifikant skillnad från 4500 i vikt.

Uppgift 3.3 (Kritiskt värde)**.** Om man gör ett t-test för hand kan man inte enkelt ta fram ett p-värde, men kan se om p-värdet är större eller mindre än fem procent genom att ställa testvärdet mot ett kritiskt värde. Använd en tabell för t-fördelning för att hitta det kritiska värdet.

I R kan man ta fram kritiska värden med `qt()`. För fem procent i svansarna har man 0.025 i respektive svans och det kritiska värdet ges av

```
qt(0.975, 29)
```

Uppgift 3.4 (Ensidigt test)**.** Använd `?t.test` för att ta fram funktionens hjälpsida. Försök att utifrån hjälpsidan beräkna ett *ensidigt* test för att se om medelvikten är *större* än 4500.

3.3.2 Konfidensintervall för normalfördelad data

Funktionen `t.test()` ger automatiskt ett konfidensintervall, direkt under utfallet av testet. Notera att konfidensintervallet inte beror på nollhypotesen. Konfidensintervall kan beräknas med skilda konfidensnivåer, oftast 95 procent, vilket sätts med argumentet `conf.level`.

Uppgift 3.5 (Konfidensnivå)**.** Gör lämplig ändring i koden nedan för att beräkna ett 99-procentigt konfidensintervall, istället för ett 95-procentigt.

```
t.test(dat$body_mass_g, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  dat$body_mass_g
## t = 31.775, df = 29, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  4006.069 4557.264
## sample estimates:
## mean of x
##  4281.667
```

Är ett 99-procentigt konfidensintervall bredare eller smalare än ett 95-procentigt?

3.3.3 Ett stickprov av binär data

Binär data är data där en observation har ett av två utfall, vilka kan kodas som noll och ett. Man talar ibland om utfallet ett som ett *positivt* utfall. Binär data kan sammanfattas med en proportion - antalet positiva utfall delat på det totala antalet upprepningar. En proportion kan testas med ett z-test för proportioner (eller *relativ frekvens*).

Vi tittar på könsfördelningen bland pingvinerna.

```
dat %>% count(sex)
```

```
## # A tibble: 2 x 2
##   sex        n
##   <fct>  <int>
## 1 female    16
## 2 male      14
```

Stickprovet har 16 honor av 30.

För att genomföra ett z-test sätter vi upp hypoteser om proportionen. I det här fallet vill vi testa om proportionen är 0.5.

- Nollhypotes H_0: p lika med 0.5
- Alternativhypotes H_1: p ej lika med 0.5

Ett test kan köras i R med `prop.test()`.

```
prop.test(x = 16, n = 30, p = 0.5, correct = F)
```

```
##
```

```
##  1-sample proportions test without continuity correction
##
## data:  16 out of 30, null probability 0.5
## X-squared = 0.13333, df = 1, p-value = 0.715
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.3614230 0.6976761
## sample estimates:
##         p
## 0.5333333
```

P-värdet ställs mot en förbestämd signifikansnivå (vanligen 5 procent). I det här fallet leder det höga p-värdet till att nollhypotesen accepteras.

Om man beräknat för hand hade vi fått z-värdet. Det värdet i kvadrat är utskriftens `X-squared`.

```
p0 <- 0.5
p <- 16/30
n <- 30
z_value <- (p - p0) / sqrt(p0 * (1 - p0) / n)
z_value^2
```

```
## [1] 0.1333333
```

Testets p-värde är ungefär 71 procent. Vår observation är alltså inte ett orimligt utfall om den faktiska sannolikheten är 0.5 och vi kan inte förkasta nollhypotesen på femprocentsnivån.

Om man löste uppgiften för hand skulle man istället för att beräkna p-värdet jämföra z-värdet med ett kritisk värde ur en tabell. Det kritiska värdet för femprocentig signifikans är 1.96. Vi kan också ta fram det genom `qnorm(0.975)`.

Hypotestestet för proportioner som används här, *z-testet*, bygger på en normalapproximation av en binomialfördelning. Approximation blir bättre när antalet observationer är stort och nollhypotesens värde p0 ligger nära 0.5. En vanlig tumregel för när approximationen är giltig är att n gånger p_0 gånger $(1 - p_0)$ ska vara större än 10.

3.3.4 Konfidensintervall för proportioner

Konstruktionen av ett konfidensintervall för en proportion är ganska lik konstruktionen för ett medelvärde. För en skattad proportion p och antal observationer n kan man beräkna p plus/minus ett z-värde från tabell gånger medelfelet, där medelfelet ges av roten ur $p \cdot (1 - p)/n$.

Om man tittar på utskriften från `prop.test()` kan man se ett konfidensintervall. Det intervallet är dock inte beräknat på samma sätt den formel som förekommer på föreläsningarna. För att få matchande utskrift kan vi använda paketet `binom` och funktionen `binom.asymp()`.

```
#install.packages("binom")
library(binom)
binom.asymp(x = 16, n = 30)
```

Uppgift 3.6 (99-procentigt konfidensintervall)**.** Gör lämplig ändring i koden nedan för att beräkna ett 99-procentigt konfidensintervall för andelen bortasegrar.

```
binom.asymp(x = 16, n = 30, conf.level = 0.95)
```

3.3.5 Chi-två-test för goodness-of-fit

Ett proportionstest kan ses som ett test av en variabel med två möjliga klasser som utfall. Ett *goodness-of-fit*-test utvecklar det till valfritt antal klasser. Testet utförs som ett chi-två-test genom att beräkna ett observerat antal O och ett förväntat antal E för varje klass. Testvärdet ges av att man beräknar $(O-E)^2/E$ för varje klass och sedan summerar. Testfunktionen är en chi-två-fördelning där antalet frihetsgrader beror på antalet klasser.

Vi illustrerar med pingvindatans fördelnings av arter. Vårt mål är att testa om arterna är lika vanliga.

```
dat %>% count(species)
```

```
## # A tibble: 3 x 2
##   species       n
##   <fct>     <int>
## 1 Adelie       10
## 2 Chinstrap     8
## 3 Gentoo       12
```

H_0: sannolikheterna för de olika utfallet ges av 33, 33 respektive 33 procent (lika vanliga arter)

H_1: minst något utfall har en annan sannolikhet än 33.

R har en inbyggd funktion för chi-två-test. Dess argument ges av observerade antal och sannolikheter.

```
chisq.test(x = c(10, 8, 12), p = c(1/3, 1/3, 1/3))
```

Situationen med flera klasser kan som sagt ses som en generalisering av fallet med två klasser. Det är alltså logiskt att chi-två-test kan användas även när man har två klasser. Följande exempel ger samma test som vi sett tidigare av andelen honor.

```
chisq.test(x = c(16, 14), p = c(0.5, 0.5), correct = F)
```

Likt `prop.test()` sätter vi `correct` till `FALSE` för att inte göra en korrektion. Notera att `x` här anges som positiva och negativa utfall istället för positiva utfall

och totalt antal utfall, vilket var fallet i `prop.test()`.

Chi-två-testet bygger på en underliggande normal-liknande approximation. En vanlig tumregel är att alla förväntade värden ska vara större än 5. R ger en varning om så inte är fallet.

```
chisq.test(c(6,4), p = c(0.51, 0.49))
```

Uppgift 3.7 (Chi-två med lika sannolikheter)**.** En vanlig tillämpning av goodness-of-fit-testet är för att testa om alla klasser är lika sannolika. En jämn fördelning är grundinställning i `chisq.test()` så i det fallet behöver man bara ange de observerade värdena. En datainsamling om M&M-godis gav följande antal.

```
dat_mnm <- data.frame(Color = c("blue", "brown", "green",
                                "orange", "red", "yellow"),
                      Count = c(180, 80, 88, 160, 134, 166))

ggplot(dat_mnm, aes(Color, Count, fill = Color)) +
  geom_col() +
  scale_fill_manual(values = dat_mnm$Color)
```

Använd de observerade värdena i kolumnen `Count` för att testa om alla godisfärger är lika vanliga.

3.4 Två stickprov

3.4.1 Test av normalfördelad data

Vid normalfördelad data från två stickprov eller grupper vill vi nästan alltid testa om populationerna har samma medelvärde. Det kan också ses som att vi testar om differensen mellan medelvärdena är noll. Vi skiljer mellan två fall:

- *matchade stickprov* - där varje observation i den ena gruppen är *kopplad* till en observation i den andra gruppen; och
- *oberoende stickprov* - där det inte finns någon sådan koppling mellan stickproven.

Typiska exempel på matchade stickprov är när man mäter samma individ för och efter en behandling och syskonstudier där ett syskon får en behandling och den andra en annan behandling.

3.4.1.1 t-test för två matchade stickprov

Vid matchade stickprov kan varje observation i en behandlingsgrupp paras med en observation i den andra gruppen. Själva testet är ett t-test för *ett* stickprov på differensserien beräknat från varje par. I R kan man antingen beräkna den differensserien eller använda `t.test()` med två dataserier och argumentet för

parvisa observationer satt till sant, `paired = T`. Som exempel ges följande data från en studie på äpple, där trädhöjd mätts före och efter en näringsbehandling.

```
dat_apple <- tibble(Tree = 1:4,
                    Before = c(48, 43, 30, 47),
                    After = c(51, 44, 42, 54))
dat_apple
```

Datan kan illustreras med ett punktdiagram där en linje binder samman paret. För att enkelt skapa grafen i `ggplot2` kan man först omstrukturera datan till lång form genom `pivot_longer`.

```
dat_long <- dat_apple %>%
  pivot_longer(-Tree, names_to = "Time", values_to = "Height")
dat_long
```

Uppgift 3.8 (Äppelgraf)**.** Fyll i kodstycket nedan för en graf av äppeldatan. Axlarna ges av `Time` och `Height`. Två observationer kan kopplas genom att sätta `Tree` som grupp.

```
ggplot(dat_long, aes(___, ___, group = ___)) +
  geom_point() +
  geom_line()
```

För att testa för skillnad före och efter behandling sätter vi upp hypoteser

- H_0: μ före behandling är lika med μ efter behandling
- H_1: μ före behandling är skild från μ efter behandling

Testet kan antingen utföras som ett enkelt t-test på differensserien

```
t.test(dat_apple$Before - dat_apple$After)
```

eller som ett t-test för två stickprov där man särskilt anger att datan är parad

```
t.test(dat_apple$Before, dat_apple$After, paired = T)
```

För bägge alternativen måste datan vara ordnad så att de två vektorerna matchar varandra parvis. Ett p-värde på 0.0987 ger att man inte förkastar vid en signifikansnivå på fem procent. Vi drar därmed slutsatsen att det inte finns någon signifikant skillnad före och efter behandling.

Uppgift 3.9 (Ensidigt test)**.** Gör ett tillägg till ett av kodstyckena med `t.test()` för att beräkna ett ensidigt test med mothypotesen att träden ökar i höjd efter behandling. Hjälpsidan för `t.test()` kan tas fram genom att köra `?t.test()`.

Konfidensintervallet beräknas från differenserna på samma sätt som vid ett stickprov med normalfördelad data. Tolkningen liknar den för ett stickprov: med 95 procents konfidens ligger den sanna skillnaden i medelvärden i intervallet.

3.4.1.2 t-test för två oberoende stickprov

Ett t-test för två oberoende stickprov testar om två populationsmedelvärden är lika. Vi återvänder till pingvindatan och testar om honor och hanar har skild kroppsvikt.

```
ggplot(dat, aes(sex, body_mass_g)) +
  geom_point()
```

```
ggplot(dat, aes(sex, body_mass_g)) +
  geom_boxplot(width = 0.3)
```

Ett t-test för två oberoende stickprov har nollhypotesen att grupperna har samma populationsmedelvärde och alternativhypotesen att populationsmedelvärdena är skilda (för det tvåsidiga fallet):

- H_0: μ_A är lika med μ_B
- H_1: μ_A är ej lika med μ_B

Testet kan utföras i R genom funktionen `t.test()`. Data kan antingen anges som en formel med dess data `y ~ x, data = dat` (vilket man kan läsa som *y uppdelat efter grupper i x*) eller som två skilda vektorer. Det förra alternativet är oftast enklare om man har datan på lång form - med en kolumn som anger grupp.

För formen med formel ger det

```
t.test(body_mass_g ~ sex, data = dat, var.equal = T)
```

och för formen med vektorer

```
vikt_hon <- dat$body_mass_g[dat$sex == "female"]

vikt_han <- dat$body_mass_g[dat$sex == "male"]

t.test(vikt_hon, vikt_han, var.equal = T)
```

Argumentet `var.equal = T` används för att beräkna testet där gruppernas varianser antas vara lika. Grundinställningen är testet där varianser inte antas vara lika, så `t.test(body_mass_g ~ sex, data = dat)` ger ett lite annat resultat.

Uppgift 3.10 (Ej lika varianser)**.** Vilka resultatvärden ändras i utskriften om man sätter `var.equal = F`?

Testet ger ett p-värde på 0.1602, vilket leder till att nollhypotesen inte förkastas på femprocentsnivån. Detta tyder på att det inte finns en viktskillnad mellan behandlingarna. Utskriften ger också ett 95-procentigt konfidensintervall. Tolkningen är att *skillnaden* mellan populationsmedelvärden ligger i intervallet med 95 procents konfidens. Notera att värdet noll ligger i intervallet.

Om man har fler än två grupper kan man vilja göra parvisa t-test - alltså ett

t-test för varje par av grupper. Ett exempel på funktionen `pairwise.t.test()` ges nedan. Funktionen bygger på att datan är i *lång* form, med en kolumn som anger det numeriska utfallet och en kolumn som anger behandlingen. För att jämföra arter skulle man till exempel skriva följande.

```
pairwise.t.test(dat$body_mass_g, dat$species,
                p.adjust.method = "none", pool.sd = F)
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  dat$body_mass_g and dat$species
##
##           Adelie  Chinstrap
## Chinstrap 0.5     -
## Gentoo    3.5e-08 3.8e-06
##
## P value adjustment method: none
```

Matchade observationer kan också kallas *parade* (eng. paired) så se upp med terminologin. Funktionen `pairwise.t.test()` för *parvisa jämförelse* mellan behandlingar, men testerna är t-test för oberoende stickprov.

3.4.2 z-test och konfidensintervall för två proportioner

Om man vill jämföra två proportioner kan man använda z-testet för två stickprov. Bland pingvinerna kan vi ta fram antalet per kön per art med `count()`.

```
dat %>% count(sex, species)
```

```
## # A tibble: 6 x 3
##   sex    species       n
##   <fct>  <fct>     <int>
## 1 female Adelie        6
## 2 female Chinstrap     3
## 3 female Gentoo        7
## 4 male   Adelie        4
## 5 male   Chinstrap     5
## 6 male   Gentoo        5
```

Säg att vi vill ställa Adelie (6 av 10 honor) mot övriga (10 av 20 honor).

- H_0: proportion A är lika med proportion B
- H_1: proportion A är skild från proportion B

I R kan testet genomföras med `prop.test()`-funktionen. Funktionens första argument är antalen honor, som en vektor med två värden, och dess andra argument är totalerna. Likt testet med ett stickprov finns en möjlighet att göra

en kontinuitetskorrektion med `correct`-argumentet. För att få samma resultat som räkning för hand anger vi att korrektion inte ska göras med `correct = F`.

```
prop.test(x = c(6, 10), n = c(10, 20), correct = F)
```

Notera att funktionen inte ger ett z-värde utan ett χ^2-värde (utskrivet `X-squared`). Det beror på att funktionen beräknar z-testet som ett likvärdigt χ^2-test. Det z-värde man får om man genomför testet som ett z-test är detsamma som roten ur utskriftens χ^2-värde. Testet ger ett högt p-värde på 0.60 vilket innebär att nollhypotesen inte förkastas: det finns ingen signifikant skillnad i proportionen honor.

Funktionen `prop.test()` ger också en utskrift av konfidensintervallet. Tolkning är att skillnaden i proportioner mellan populationerna ligger i intervallet med 95 procents konfidens. Notera att nollan ingår i intervallet.

Uppgift 3.11 (Burfågel)**.** Det finns en förvånansvärt stor mängd studier på kopplingen mellan innehav av burfågel och lungcancer. En sådan studie (Kohlmeier et al 1992) ger följande antal för burfågelägande och lungcancer.

```
dat_bird <- data.frame(Burfågel = c("Burfågel", "Ej_burfågel"),
                       Lungcancer = c(98, 141),
                       Ej_lungcancer = c(101, 328))
dat_bird
```

Datan tyder på att människor med burfågel har en förhöjd risk att drabbas av lungcancer. Genomför ett z-test för att se om andelen burfågelägare än densamma i de två patientgrupperna.

```
prop.test(x = c(___, ___), n = c(___, ___), correct = F)
```

3.4.3 Chi-två-test för korstabeller

Data med två kategoriska variabler kan presenteras med en korstabell. Vi tar uppdelningen av pingviner i kön och art. En korstabell kan konstrueras med `pivot_wider`.

```
dat_table <- dat %>% count(sex, species)
dat_table
dat_table_wide <- dat_table %>%
  pivot_wider(values_from = n, names_from = sex)
dat_table_wide
```

Datan tyder inte på några större skillnader mellan arter. Vi kan illustrera med uppdelade staplar.

```
ggplot(dat_table, aes(species, n, fill = sex)) +
  geom_col(position = position_fill(), color = "black") +
  geom_text(aes(label = n), position = position_fill(0.5), size = 7) +
  scale_fill_manual(values = c("hotpink", "green"))
```

Argumentet `position` i `geom_bar()` används för att skapa proportionella staplar.

Ett chi-två-test på en korstabell har nollhypotesen att det inte finns något samband mellan variabeln för rader och variabeln för kolumner. Chi-två-testet kan tillämpas på korstabeller med godtyckligt antal rader och kolumner - det är alltså inte begränsat till fallet med jämförelse mellan två grupper. Antal frihetsgrader ges av antal rader minus ett gånger antal kolumner minus ett. Testet kan enkelt göras med `chisq.test()`. Som ingångsvärde kan man plocka ut kolumnerna med numeriska värden genom hakparenteser.

```
dat_table_wide[, 2:3] # De två numeriska kolumnerna

chisq.test(dat_table_wide[, 2:3])
```

Utskriften ger teststorheten, antal frihetsgrader, och p-värdet. I det här fallet är p-värdet högt och slutsatsen blir att nollhypotesen inte förkastas - det finns inget signifikant samband mellan kön och art. Antalet frihetsgrader ges av antalet rader minus ett gånger antalet kolumner minus ett (här $(2-1)\cdot(3-1)=2$).

Chi-två-testet är ett asymptotiskt test - dess egenskaper är beroende av *stora* stickprov. Som gräns för storleken används ofta att samtliga förväntade antal ska vara större än 5. Funktionen ger en varning om förväntade värden är små. En möjlig lösning i sådana fall är att slå ihop klasser.

Exemplet på pingvindatan har några små klasser och funktionen gav därför en varning.

Uppgift 3.12 (Burfågeln återvänder)**.** En svensk studie på koppling mellan burfågel och lungcancer (Modigh et al, 1996) ger följande antal (för män).

```
dat_bird_swe <- data.frame(Burfågel = c("Burfågel", "Ej_burfågel"),
                Lungcancer = c(108, 144),
                Ej_lungcancer = c(171, 256))
dat_bird_swe
```

Genomför ett chi-två-test för att se om andelen cancerdrabbade än densamma i de två burfågelsgrupperna. Formulera tydliga hypoteser. För att få utfall som stämmer med en handräkning kan man sätta `correct = F`.

```
dat_bird_swe[, c(2,3)]
chisq.test(___, correct = F)
```

3.5 AI-uppgift III

Det finns många studier som pekar på att p-värdet är svårtolkat. Det är särskilt vanligt att p-värdet tolkas som sannolikheten att nollhypotesen är sann.

Den korrekta tolkningen är dock snarare tvärtom: p-värdet är sannolikheten för datautfallet givet att nollhypotesen är sann.

Uppgift 3.13 (Tolkning av p-värdet)**.** Be en AI-chatbot besvara en t-test-fråga t.ex. från en tidigare tenta eller uppgiften i början av den här övningen. Fråga särskilt om tolkningen av p-värdet och om det finns några vanliga feltolkningar.

Kapitel 4

Variansanalys och regression

Datorövning 4 handlar om variansanalys och regression. Efter övningen ska vi kunna

- skatta och tolka en anova-modell i R,
- göra parvisa jämförelser mellan behandlingar,
- skatta och tolka en regressionsmodell i R,
- beräkna och tolka korrelationen mellan två variabler.

4.1 Allmänt om variansanalys

Variansanalys (eller *anova-modellen*) är en statistisk modell där medelvärdet varierar beroende på en behandling och ett normalfördelat slumpfel. Från en anova-modell kan man beräkna ett F-test, som testar om det finns någon övergripande gruppskillnad, och post-hoc-test, som jämför specifika grupper med varandra.

Den specifika modellen beror på försöksupplägget. Här ges exempel på variansanalys med en faktor och en faktor med block.

4.2 Variansanalys. En faktor

Vid variansanalys med en faktor har man observationer av en kontinuerlig utfallsvariabel från två eller flera behandlingsgrupper. Som exempel används en datamängd på ett odlingsförsök med tre behandlingar (varav en kontroll). Exemplet finns tillgängligt i R som `PlantGrowth`.

```
PlantGrowth
```

Datan har 30 observationer av vikt `weight` och varje observation tillhör någon specifik behandling `group`. Datan kan illustreras med ett spridningsdiagram.

```
ggplot(PlantGrowth, aes(group, weight)) +
  geom_point()
```

Behandling 1 verkar vara något lägre än kontrollen medan behandling 2 verkar vara något högre.

En anova-modell kan i R skattas med funktionen `lm()` (för *linjär modell*). Från modellobjektet kan man sedan plocka fram en anova-tabell (som bland annat anger utfallet av F-testet) och genomföra parvisa jämförelser genom `emmeans`.

```
mod <- lm(weight ~ group, data = PlantGrowth)
```

Modellen anges som en formel `weight ~ group`, vilket kan utläsas *vikt beroende på behandlingsgrupp*. Därefter anges data med argumentet `data`.

För anova-tabellen finns flera alternativ. Här används funktionen `Anova()` från paketet `car`.

```
library(car)
Anova(mod)
```

Anova-tabellen ger kvadratsummor (`Sum Sq`), frihetsgrader (`Df`) och utfallet av ett F-test. Testets hypoteser ges av

H0: alla behandlingsgrupper har samma medelvärde H1: alla behandlingsgrupper har inte samma medelvärde

Det låga p-värdet tyder på att nollhypotesen bör förkastas, vilket alltså pekar på att det finns någon eller några skillnader i medelvärde.

Uppgift 4.1 (Anova för hand)**.** Anovatabell från `Anova()` ger kvadratsummor och frihetsgrader. Använd den informationen för att, för hand, beräkna medelkvadratsummor och F-värdet.

Uppgift 4.2 (Tabellvärde för F-fördelningen)**.** Anova-tabellen ger ett p-värde från vilket vi kan dra en direkt slutsats. Om man istället löser uppgiften för hand ställer man det beräknade F-värdet mot ett kritiskt värde från en tabell över F-fördelningen. Se efter om man kan hitta ett lämpligt tabellvärde för det aktuella testet (med 2 och 27 frihetsgrader). Det är möjligt att det inte finns en rad för 27 i en vanlig F-fördelningstabell, använd isåfall värdet på närmast övre rad (t.ex. 26 eller 25). I R kan kvantiler för F-fördelningen tas fram med `qf()`, t.ex.

```
qf(0.95, 2, 27)
```

En naturlig följdfråga är vilka behandlingsgrupper som skiljer sig åt. För att besvara det krävs *parvisa jämförelser* där behandlingarna jämförs två och två.

Parvisa jämförelse kan göras med paketet `emmeans` och funktionen med samma namn. Funktionen tar modellobjektet som första argument och en formel för jämförelsetyp som andra argument (här `pairwise ~ group`, en parvis jämförelse mellan nivåer i `group`).

```
# install.packages("emmeans")
library(emmeans)
emmeans(mod, pairwise ~ group)
```

I den nedre tabellen med jämförelser ges alla parvisa jämförelser. Nollhypotesen är att de två grupper som jämförs har samma medelvärde - ett lågt p-värde tyder alltså på att de två grupperna är signifikant skilda. Notera också att p-värden justeras med tukey-metoden, även känt som Tukeys HSD.

Om man istället vill använda Fishers LSD kan man styra justeringen med argumentet `adjust`.

```
emmeans(mod, pairwise ~ group, adjust = "none")
```

Parvisa jämförelser presenteras ofta med signifikansbokstäver (en. *compact letter display, cld*). Dessa kan plockas fram med `multcomp`-paketet och funktionen `cld()`.

```
em <- emmeans(mod, pairwise ~ group)

# install.packages("multcomp")
# install.packages("multcompView")
library(multcomp)
cld(em, Letters = letters)
```

Tolkning av grupperingen till höger är att grupper som delar en bokstav inte är signifikant skilda. I det här fallet är den lägsta nivån skild från de två högsta. I övrigt finns inga signifikanta skillnader. Jämför gärna med p-värdena från tabellen med parvisa jämförelser. Man bör se att parvisa jämförelser med ett p-värde under fem procent motsvaras av att de behandlingarna inte delar någon bokstav i bokstavstabellen.

Uppgift 4.3 (Anova med två behandlingar)**.** Följande kod skapar en datamängd med två behandlingar.

```
dat_two <- PlantGrowth %>% filter(group %in% c("trt1", "trt2"))
```

Använd den datan för att göra ett t-test för två oberoende stickprov med lika varians, ett t-test för två oberoende stickprov utan antagande om lika varians, och ett F-test (ofullständig exempelkod nedan). Vad kan sägas om p-värdena från de tre testen?

```
t.test(___ ~ group, data = dat_two, var.equal = T)
t.test(weight ~ ___, data = dat_two, var.equal = F)
```

```
mod <- lm(weight ~ group, data = dat_two)
Anova(mod)
```

Uppgift 4.4 (Mass-signifikans)**.** Anledning till att vi justerar p-värden är att man vid varje test har en sannolikhet att förkasta. Om man gör ett stort antal tester är man nästan garanterad att få något (falskt) signifikant resultat. Justering höjer p-värdena för att minska den risken. Följande kod simulerar data med 5 grupper och producerar de parvisa jämförelserna.

```
n_groups <- 5
dat_sim <- expand_grid(obs = 1:10, group = letters[1:n_groups]) %>%
  mutate(y = rnorm(n()))
mod <- lm(y ~ group, dat_sim)
emmeans(mod, pairwise ~ group, adjust = "none")
```

Kör koden tio gånger. Hur många gånger av de tio ger de parvisa jämförelserna *någon* signifikant skillnad (det vill säga något p-värde under 0.05)?

Uppgift 4.5 (Äppelinfektionsimport)**.** En studie har givit ett mått på infektion hos äppelträd. Fyra sorter jämförs med tre replikat per sort. Data finns i fliken *Äppelangrepp* i excelfilen *Uppgiftsdata.xslx* på canvassidan. Fyll i kodstycket nedan för att importera datan.

Uppgift 4.6 (Äppelinfektionsgraf)**.** Fyll i kodstycket nedan för att skapa en graf av äppeldata.

```
ggplot(___, aes(x = ___, y = ___)) +
  geom_point()
```

Uppgift 4.7 (Äppelinfektionsmodell)**.** Fyll i kodstycket nedan för att skatta en anovamodell och ta fram anovatabellen. Vad är F-testets noll- och alternativhypotes? Vilken slutsats kan man dra från testet?

```
mod <- lm(___ ~ ___, data = dat_apple)
Anova(mod)
```

4.3 Variansanalys. En faktor med block

I en blockdesign delas försöksobjekten (de enheter man ger en behandling och sedan mäter, t.ex. en försöksruta eller en planta) in i grupper av lika objekt (ett *block*). Sedan ger man enheterna inom blocket varsin behandling. Blockförsök är ofta balanserade, så att varje behandling förekommer en gång i varje block.

Som exempel på ett blockförsök kan vi titta på datan `oats` från paketet `MASS`. Datan kommer från ett agrikulturellt försök och blockdesignen sker genom att man delar in ett fält i flera delar (blocken) och sätter varje behandling i varje block. Datan har två faktorer (kväve `N` och sort `V`), men låt oss i den här första delen titta på en specifik sort.

```
library(MASS)
oats_marvel <- oats %>% filter(V == "Marvellous")
oats_marvel
```

En vanlig illustration av ett blockförsök är ett punktdiagram kombinerat med ett linjediagram.

```
ggplot(oats_marvel, aes(N, Y, color = B, group = B)) +
  geom_point(size = 4) +
  geom_line()
```

Färg och linje sammanbinder observationer från samma block. Det finns tecken på en blockeffekt: block I är nästan alltid högst och block V är nästan alltid lägst. Det finns också en tydlig behandlingseffekt i att högre kväve ger högre skörd.

Blockeffekten kan enkelt föras in i modellen genom att lägga till variabeln `B` i `lm`-funktionen. Anova-tabellen och parvisa jämförelser kan göras på samma sätt som tidigare. Resultaten påverkas av att modellen har en blockfaktor; man behöver vanligen inte ange det explicit.

```
mod_bl <- lm(Y ~ N + B, data = oats_marvel)
Anova(mod_bl)
```

P-värdet från F-testet på variabeln N är nu klart mindre än tidigare. Detta beror på att en stor del av variationen kan förklaras med blockeffekten, vilket är tydligt i att blockeffekten också har ett litet p-värde i F-testet.

Det kan vara intressant att jämföra med modellen utan block.

```
mod_wo_block <- lm(Y ~ N, data = oats_marvel)
Anova(mod_wo_block)
```

Det som är residualens kvadratsumma i modellen utan block är i blockmodellen uppdelat i en blockeffekt och en residualterm. Eftersom F-testet bygger på en jämförelse mellan behandlingseffekten och residualtermen leder blockdesignen till starkare signifikans i blockmodellen. Å andra sidan kostar blockfaktorn frihetsgrader vilket ger oss ett svagare test. Effekten av att ta med ett block beror alltså på om det finns en verklig skillnad mellan blocken eller ej.

Vi kan gå vidare med att titta på parvisa jämförelser mellan kvävenivåer. Funktionen `emmeans()` och `cld()` fungerar som tidigare.

```
cld(emmeans(mod_bl, ~ N), Letters = letters)
```

Signifikansbokstäver anger att den lägsta nivån är skild från övriga och att den näst lägsta är skild från den högsta. Även här kan det vara intressant att jämföra med modellen utan block.

```
cld(emmeans(mod_wo_block, ~ N), Letters = letters)
```

Modellen utan block ger samma medelvärden `emmean` men större medelfel `SE` och färre signifikanta skillnader.

Uppgift 4.8 (Block med två behandlingar. Graf)**.** Det minsta möjliga blocket är det med två behandlingar. Vi filtrerar havredatan för att den situationen.

```
dat_small_block <- oats %>%
  filter(V == "Marvellous", N %in% c("0.6cwt", "0.0cwt"))
dat_small_block
```

Fyll i stycket nedan för att skapa en graf med `N` på x-axeln, `Y` på y-axeln och en gruppering som länkar observationer från samma block.

```
ggplot(dat_small_block, aes(x = ___, y = ___, group = ___)) +
  geom_point() +
  geom_line()
```

Uppgift 4.9 (Block med två behandlingar. Test)**.** Eftersom det är ett försök med en förklarande faktor och block kan man modellera det med den tidigare blockmodellen. Men eftersom man bara har två observationer per block kan man också se det som matchade stickprov, vilket kan lösas med ett t-test. Fyll i stycket nedan för att göra de två testen - utfallsvariabeln är skörd `Y` och den förklarande faktorn är kvävenivån `N`. Jämför resultaten.

```
mod <- lm(___ ~ ___ + B, data = dat_small_block)
Anova(mod)

t.test(___ ~ ___, data = dat_small_block, paired = ___)
```

4.4 Bonus. Två faktorer med block

Exempeldata på havre tar med två förklarande faktorer och ett block. Datan kan illustreras med ett punktdiagram där `facet_wrap` delar grafen efter sort.

```
ggplot(oats, aes(N, Y, color = B)) +
  geom_point(size = 4) +
  facet_wrap(~ V)
```

Grafen visar samma kvävesamband som tidigare. Det finns inga tydliga skillnader mellan sorter, möjligen har sorten Victory givit något lägre skörd än övriga. Det finns fortfarande en tydlig blockeffekt, till exempel har block I höga värden och block V låga värden.

Modellen skattas genom att lägga till variabeln för sort (V för variety) i `lm`-formeln. En modell med två faktorer kan antingen vara med eller utan en *interaktion*. Interaktionstermen fångar påverkan mellan faktorerna. Ett exempel

hade varit om någon sort svarat starkare på ökad kväve än någon annan. Standardmodellen är att ta med interaktionen, vilket vi anger genom att sätta `N * V` istället för `N + V`. Blocket tas fortfarande med som en adderad faktor

```
mod_two_fact <- lm(Y ~ N * V + B, data = oats)
```

Anovatabellen kan plockas fram på samma sätt som tidigare.

```
Anova(mod_two_fact)
```

Raden `N:V` gäller interaktionseffekten mellan kväve och sort. I det här fallet är det ingen signifikant interaktion - vilket tyder på att sorterna svarar på kvävebehandling på liknande sätt. Samtliga huvudeffekter (raderna för N, V och B) är signifikanta. Kvadratsummorna och p-värdena tyder på att kväve förklarar mer av variationen än sort, vilket också är i linje med grafen ovan.

Vid flerfaktoriella försök kan man presentera parvisa jämförelser på flera olika sätt. Man kan ange huvudeffekter för en faktor utan att ange den andra faktorn, man kan ange medelvärden för samtliga kombinationer av två faktorer, och man kan ange medelvärden uppdelat efter nivåer i en annan faktor.

```
emmeans(mod_two_fact, ~ N)
emmeans(mod_two_fact, ~ N + V)
emmeans(mod_two_fact, ~ N | V)
```

Även här kan man göra jämförelser mellan nivåer genom att sätta `pairwise ~ N + V` eller beräkna signifikansbokstäver med `cld`. Följande kod jämför kvävenivåer *inom* sort.

Uppgift 4.10 (Sort uppdelat efter kvävenivå)**.** Gör lämplig ändring i koden ovan för att jämföra sorter *inom* kvävenivå. Finns det några signifikanta skillnader?

Uppgift 4.11 (Interaktion med ett block)**.** I modellen ovan är block en *additiv* faktor - den ingår inte i någon interaktionseffekt. Vad händer med testerna om man skattar modellen där samtliga interaktioner tas med? Varför?

```
mod_two_fact <- lm(Y ~ N * V * B, data = oats)
```

4.5 Regression

I en regression modelleras en numerisk variabel som en funktion av en annan numerisk variabel. Vid enkel linjär regression finns *en* sådan *förklarande variabel* och förhållandet mellan variablerna antas vara linjärt.

Ta som exempel data på förväntad medellivslängd och bnp per capita. Datan hämtas från `gapminder`-paketet. Paketet `ggrepel` och funktionen `geom_text_repel()` kan användas för att sätta punktetiketter som inte överlappar. För enklare tolkning av modellen transformeras bnp per capita till att vara i tusen dollar, snarare än dollar.

```
library(gapminder)
dat_eu07 <- gapminder %>%
  filter(year == 2007, continent == "Europe") %>%
  mutate(gdpPercap = gdpPercap / 1000)

library(ggrepel)
ggplot(dat_eu07, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_text_repel(aes(label = country), size = 3)
```

Datan visar ett positivt samband mellan variablerna - högre bnp per capita är kopplat till högre medellivslängd.

Uppgift 4.12 (Data för 1957)**.** Vad måste ändras i stycket nedan för att plocka ut data och göra en graf för Europa 1957?

```
dat_eu57 <- gapminder %>%
  filter(year == 2007, continent == "Europe") %>%
  mutate(gdpPercap = gdpPercap / 1000)

ggplot(dat_eu57, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_text_repel(aes(label = country), size = 3)
```

En regressionmodell kan i R skattas med `lm()`-funktionen. Syntaxen är väldigt lik den för anovamodellen, men istället för en faktor som förklarande variabel används nu en kontinuerlig variabel.

```
mod <- lm(lifeExp ~ gdpPercap, data = dat_eu07)
summary(mod)
```

Funktionen `summary` ger en sammanfattning av modellen. Skattningen av modellens konstanta parameter ges som raden `(Intercept)` och dess tolkning är som förväntat värde i medellivslängd om bnp per capita är noll. Det är ofta lutningsparametern som är mer intressant. Skattningen av lutningsparametern ges på den rad som har samma namn som den förklarande variabeln, här `gdpPercap`. Den skattade parametern är 0.2146. Lutningsparametern har den generella tolkning som ökningen i y-variabeln när x-variabeln ökar med 1. I det här fallet ger 0.2146 att ett lands medellivslängd ökar med ungefär 0.2146 år (eller 78 dagar) när bnp per capita ökar med 1000 dollar.

Uppgift 4.13 (Modell för 1957)**.** Skatta samma modell som ovan, denna gång med data från 1957. Tolka lutningsparametern i ord. Är effekten av ökad bnp större 2007 än den var 1957?

Man kan enkelt rita ut regressionlinjen i en graf med `geom_smooth()` och argumentet `method` satt till `lm`.

```
ggplot(dat_eu07, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_text_repel(aes(label = country), size = 3) +
  geom_smooth(method = lm)
```

Den blå linjen illustrerar regressionlinjen $72.27 + 0.2146$x. Det grå bandet kring linjen är ett konfidensintervall för skattningen av y-variabeln.

Uppgift 4.14 (Graf för 1957)**.** Använd `geom_smooth(method = lm)` för att lägga till en regressionslinje för data för 1957. Hur mycket påverkar de två avvikande länderna?

Utskriften från `summary` ger också tester av parametrarna (den högra kolumnen `Pr(>|t|)` ger p-värdet för ett test där nollhypotesen är att populationsparametern är noll). I det här fallet är både intercept och lutning skilda från noll. Motsvarande F-test för lutningen kan tas fram med en anova-tabell.

```
library(car)
Anova(mod)
```

Testerna av en regressionsmodell bygger på ett normalfördelningsantagande och ett antagande om *homoskedasticitet* (lika varians i y oavsett position på x-axeln). Antagandena kan undersökas genom att titta på skattningens *residualer* - skillnaden mellan det faktiska y-värdet och modellens värde. Residualerna kan undersökas med ett histogram eller en QQ-plot. En annan vanlig diagnosplot är ett spridningsdiagram med skattade värden på x-axeln och residualerna på y-axeln.

```
dat_eu07 <- dat_eu07 %>%
  mutate(Residualer = residuals(mod),
         Skattade = fitted(mod))

ggplot(dat_eu07, aes(sample = Residualer)) +
  geom_qq() + geom_qq_line()
```

```
ggplot(dat_eu07, aes(Skattade, Residualer)) + geom_point()
```

Om data följer en normalfördelning bör histogrammet visa en ungefärlig normalkurva, QQ-plotten bör visa punkter på den diagonala linjen och spridningsdiagrammet bör visa en slumpmässig spridning av punkter. Graferna pekar i det här fallet inte på några tydliga avvikelser från normalfördelningsantagandet, möjligen pekar QQ-plotten på mindre spridning i svansarna än en teoretisk normalfördelning.

Uppgift 4.15 (Icke-linjära samband)**.** Låt oss titta på hela gapminder-datan för 2007.

```
dat_2007 <- gapminder %>% filter(year == 2007)
ggplot(dat_2007, aes(gdpPercap, lifeExp)) + geom_point()
```

Hur ser sambandet mellan bnp och medellivslängd ut? Vad skulle vara problematiskt med simpel linjär regression i det här fallet? När vi tittade på normalfördelningen sa vi att man ofta kan logaritmera en variabeln och få *bättre* egenskaper. Vad ska ändras i koden ovan för att använda logaritmerad `gdpPercap` istället för den ursprungliga variabeln? Är det sambandet mer linjärt?

Uppgift 4.16 (Log-transformerad data)**.** Vad ska ändras i koden nedan för att använda logaritmerad `gdpPercap` istället för den ursprungliga variabeln? Är det sambandet mer linjärt?

```
dat_2007 <- gapminder %>% filter(year == 2007)
ggplot(dat_2007, aes(gdpPercap, lifeExp)) + geom_point()
```

4.6 Korrelation

Korrelation ger ett mått mellan -1 och 1 på hur väl två variabler samvarierar. En korrelation över noll tyder på ett positivt samband mellan variablerna - en observation med ett högt värde i den ena variabeln har också ett högt värde på den andra - medan en korrelation under noll tyder på ett negativt samband. I R kan korrelation beräknas med `cor()` och två variabler som första och andra argument. Funktionen `cor.test()` ger ett test där nollhypotesen är att korrelationen är noll.

```
cor(dat_eu07$lifeExp, dat_eu07$gdpPercap)
cor.test(dat_eu07$lifeExp, dat_eu07$gdpPercap)
```

Medellivslängd och bnp per capita har en stark positiv korrelation på 0.85 och den korrelation är signifikant skild från noll ($p < 0.001$). Notera att p-värdet är detsamma som för lutningsparametern i regressionen.

Uppgift 4.17 (Korrelationsmatris)**.** Om man har fler än två variabler sammanfattas korrelationer ofta med en korrelationsmatris.

```
dat_eu07[, 4:6]
cor(dat_eu07[, 4:6])
```

Vad är korrelationen mellan befolkningsstorlek och bnp per capita?

Uppgift 4.18 (Anscombes data)**.** Den raka regressionslinjen eller det enkla korrelationsmåttet säger lite om hur data egentligen ser ut. En vanlig illustration av detta är *Anscombes kvartett*, fyra exempel konstruerade av den brittiske statistikern Francis Anscombe 1973. Datan finns tillgänglig i R som datasetet `anscombe`.

```
anscombe
```

Plotta de fyra graferna (`x1` paras med `y1` och så vidare) i spridningsdiagram och beräkna korrelation för varje par. Ett exempel ges för den första mängden nedan. Kommentera utfallet.

```
ggplot(anscombe, aes(x1, y1)) + geom_point()
cor(anscombe$x1, anscombe$y1)
```

För en modern variant av Anscombes data kan man titta på *The Datasaurus Dozen*. Det är också en serie konstruerade variabler som har liknande egenskaper sett till medelvärden, standardavvikelser och korrelationer, men skiljer sig åt om du ritas i en graf.