Datorlaboration 6

Josef Wilzén och Måns Magnusson

23 februari 2016

Instruktioner

- Denna laboration ska göras i grupper om **två och två**. Det är viktigt för gruppindelningen att inte ändra grupper.
- En av ska vara **navigatör** och den andra **programmerar**. Navigatörens ansvar är att ha ett helhetsperspektiv över koden. Byt position var 30:e minut.
- Det är tillåtet att diskutera med andra grupper, men att plagiera eller skriva kod åt varandra är **inte tillåtet**.
- Använd inte `å`, `ä` eller `ö` i variabel- eller funktionsnamn.
- Utgå från laborationsfilen, som går att ladda ned **här**, när du gör inlämningsuppgifterna.
- Spara denna som labb[no]_[liuID1]_[liuID2].R , t.ex. labb5_josad732_manma684.R om det är labb 5. Ordna liuID för gruppmedlemmerna efter bokstavsordning i filnamnet. Ta inte med hakparenteser i filnamnet. Denna fil ska laddas upp på LISAM och ska **inte** innehålla något annat än de aktuella funktionerna, namn- och ID-variabler och ev. kommentarer. Alltså **inga** andra variabler, funktionsanrop för att testa inlämningsuppgifterna eller anrop till markmyassignment-funktioner.
- Laborationen består av två delar:
  - Datorlaborationen
  - Inlämningsuppgifter
- I laborationen finns det extrauppgifter markerade med *. Dessa kan hoppas över.
- Deadline för labben framgår på **LISAM**

Innehåll

Del I

Datorlaboration

Kapitel 1

Introduktion till grafik

I R finns en hel del funktionalitet för att arbeta med grafik. I det grundläggande R finns det som brukar kallas base graphs som är den grundläggande grafikfunktionaliteten. Utöver detta är paketet `ggplot2` mycket populär för visualisering. För en introduktion till grafisk visualisering av data är [1] ett standardverk.

I base-paketet fungerar grafiken på så sätt att vi lägger till lager för lager i en visualisering av data. Tänk dig att du ritar med en penna. Vi kommer i denna del använda oss av en del av de dataset som installeras tillsammans med R. Läs in dessa dataset på följande sätt. Undrar du vad materialen innehåller kan du använda `?iris`, `?mtcars`, `?Nile` för att få information om de olika variablerna.

```
data(iris)
data(mtcars)
data(Nile)
Nile <- as.data.frame(Nile)
Nile$years <- 1871:1970
```

Det finns många möjliga inställningar som går att göra. Dessa sammanfattas i dokumentationen för `par`. Sök efter `?par` i hjälpen för att få detaljerad information.

1.1 Visualisera en variabel

Vi inleder med att försöka visualisera data i en variabel.

1.1.1 Cirkeldiagram

Cirkeldiagram är ofta populärt, men bör generellt sett undvikas (se ex. [1]). Det är en typ av diagram som inte lämpas sig bra för människors tänkande och många har svårt att jämföra tårtbitar visuellt, **[här]** finns de vanligaste argumenten. För att skapa cirkeldiagram använder vi funktionen `pie()` på följande sätt.

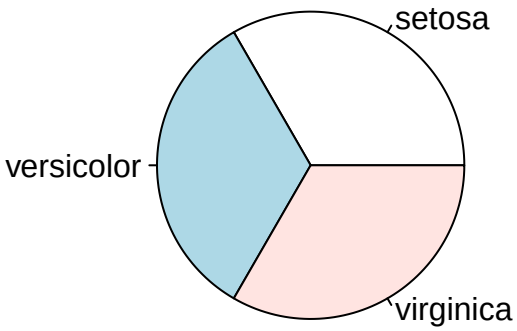
1. Som ett första steg måste vi beräkna frekvenser för den kategoriska variabel vi vill visualisera med `table()`. Testa `?table()`, `class(freqs)`, `str(freqs)`

```
freqs <- table(iris$Species)
freqs
```

```
    setosa versicolor  virginica
        50         50         50
```

2. Baserat på denna frekvensfördelning är det därefter möjligt att skapa ett cirkeldiagram med `pie()`:

```
pie(freqs)
```



3. Vill vi ändra på labels gör vi det genom att ange en ny textvektor som labels, en rubrik anger vi med `main`:

```
pie(freqs, labels=c("Del 1", "Del 2", "Del 3"))
pie(freqs, labels=c("Hej", "Hejsan", "Hej hej"),     main="Cirkeldiagram")
```

4. På ett liknande sätt kan vi sedan ange vilka färger vi vill ange med argumentet `col`. Samtliga färger som går att använda i R finns **[här]**.

```
pie(freqs, col=c("rosybrown1","yellowgreen", "khaki2"))
```

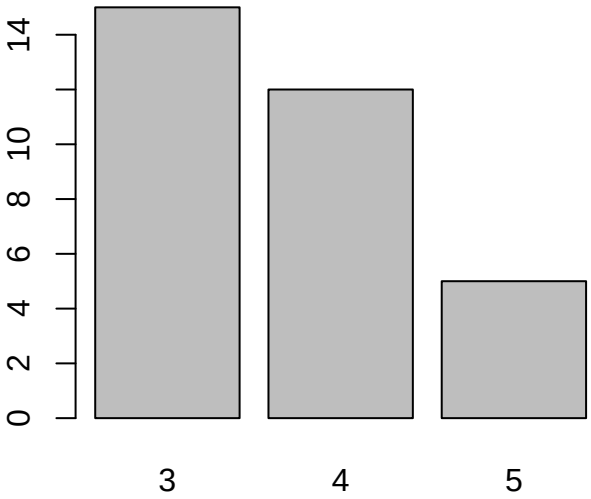
5. Vi kan självklart styra ännu mer i utformningen av cirkeldiagrammen. För mer hjälp för cirkeldiagram använd `?pie`. Men som sagt undvik cirkeldiagram i största möjliga mån.

1.1.2 Barcharts

Barcharts är enklare att tolka på ett korrekt sätt och är en av de mest grundläggande graftyperna.

1. För att skapa en barchart behöver vi (på samma sätt som för cirkeldiagrammen) utgå från frekvenser när vi skapar vårt diagram. Vi börjar med det allra enklaste diagrammet:

```
freqCars <- table(mtcars$gear)
barplot(freqCars)
```



2. Som framgår ovan får vi ett mycket enkelt diagram. Diagrammet använder sig av radnamn (`rownames()`) för `freqCars`. Pröva att kolla hur `freqCars` se ur och dess radnamn.

3. Att lägga till rubriker och titel gör vi på samma sätt som för cirkeldiagramen ovan, med argumenten `main`, `xlab` och `ylab`.

```
barplot(freqCars, main="Cars", xlab="Gears", ylab="Counts")
```

4. Vi kan också välja att ha horisontella staplar med argumentet `horiz=TRUE`.

```
barplot(freqCars, main="Cars", horiz=TRUE)
```

5. Som för cirkeldiagrammet kan vi också byta färg om vi vill med `col`.

```
barplot(freqCars, main="Cars", col="red")
```

6. Vi kan också ha flera variabler i samma stapeldiagram, så kallade grupperade eller "stackade" stapeldiagram. Vi börjar med grupperade stapeldiagram.

```
carTable <- table(mtcars$vs, mtcars$gear)
barplot(carTable, main="Car Distribution by Gears and VS",
xlab="Number of Gears",
col=c("darkblue","red"),legend= rownames(carTable))
```

7. På ett liknande sätt kan vi skapa "stackade" stapeldiagram:

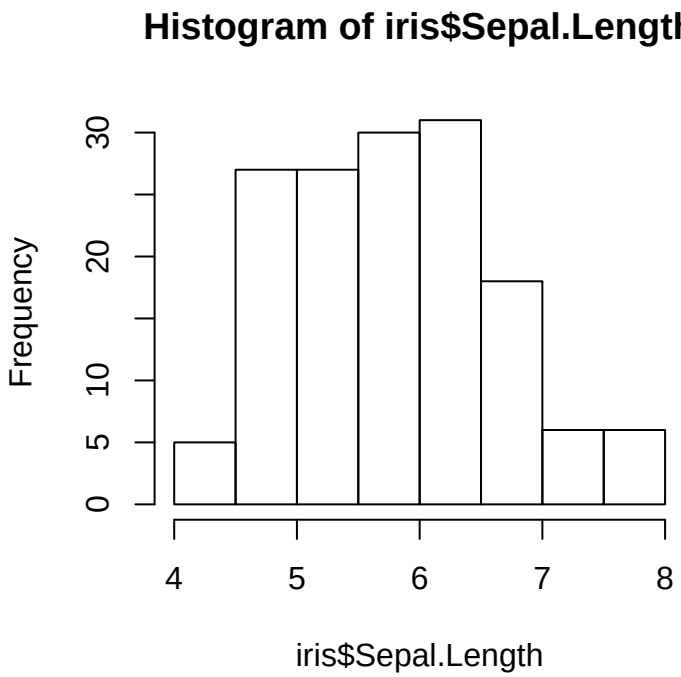
```
barplot(counts, main="Car Distribution by Gears and VS",
xlab="Number of Gears", col=c("darkblue","red"), beside=TRUE)
```

1.1.3 Histogram

Histogram kan vi använda för att visualisera en kontinuerlig variabel.

1. För att skapa ett enkelt histogram använder vi funktionen `hist()`.

```
hist(iris$Sepal.Length)
```



2. Att ändra rubriker och färger görs på samma sätt som i övriga diagram.

```
hist(iris$Sepal.Length, col="blue", main="Min titel", xlab="X-titel", ylab="Y-titel")
```

3. När det gäller histogram kan det vara så att i vissa fall vill vi ha fler eller färre staplar. Detta styrs med breaks. Pröva följande sätt att skapa ett histogram:

```
hist(iris$Sepal.Length, breaks=40, col="red")
```

4. Histogram är diskreta, men vi kan enkelt i R skapa en uppskattning av den underliggande fördelningen visuellt.

```
dens <- density(iris$Sepal.Length)
plot(dens)
```

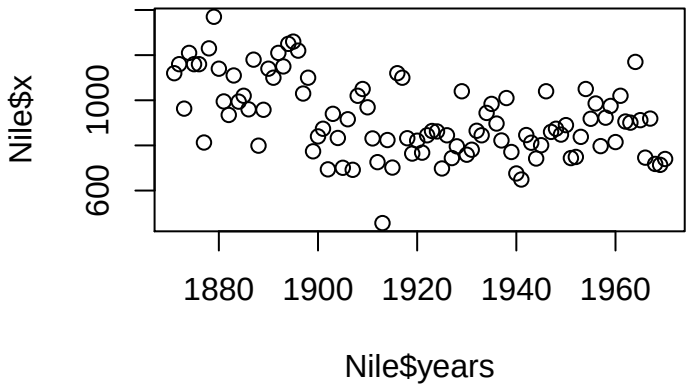
1.2 Visualisering i flera variabler

1.2.1 Sambandsdiagram

En av de vanligaste sätten att visualisera tvådimensionella data är med scatter plots. Vi ska nu pröva att visualiser den historiska utvecklingen av Nilens vattennivåer.

1. Att skapa en vanlig scatterplots görs med funktionen `plot()`. Det är en generisk funktion och i många fall använder vi plot för att visualisera olika typer av objekt. Testa `methods(plot)` för att se vilka metoder som finns för `plot()`. Grundutförandet ger dock en scatterplot på följande sätt:

```
plot(Nile$years, Nile$x)
```



2. Som tidigare kan vi också lägga till/förändra rubriker enkelt om vi vill.

```
plot(Nile$years, Nile$x, main="Water in the Nile", xlab="Years", ylab="Level")
```

3. Vill vi ändra färgen på våra punkter använder vi som vanligt parametern `col`.

```
plot(Nile$years, Nile$x, col="blue")
```

4. Vill vi att olika punkter ska ha olika färger anger vi bara en vektor med färgnamn.

```
colVector<- rep("blue",length(Nile$x))
colVector[Nile$years>1900] <- "red"
plot(Nile$years, Nile$x, col=colVector)
```

5. Studera hur `colVector` ser ut ovan så du förstår hur vektorn används för att styra färgen på punkterna. Pröva att använd ytterligare än färg till punkterna efter 1945.

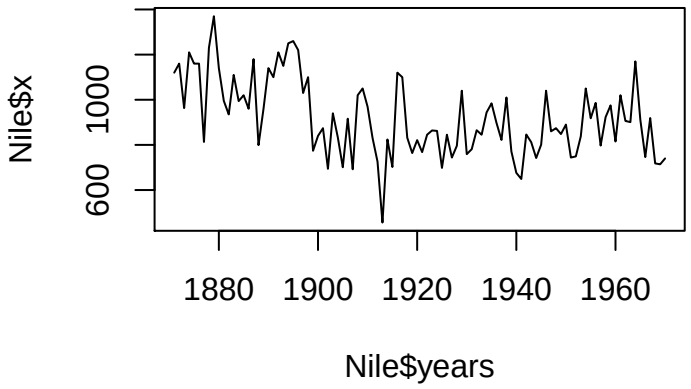
6. Vi kan också ändra hur punkterna ser ut och använda andra symboler. Det finns totalt 25 olika symboler i baspaketet. För att använda en punkttyp används argumentet pch. Med koden nedan kan vi snabbt se alla olika typer av punkter som finns i baspaketet.

```
plot(1:25, 1:25, type="p", pch = 1:25)
```

1.2.2 Linjediagram

1. Nilens vattennivåer är en tidsserie snarare än ett vanligt samband. För tidsserier vill vi ofta ha linjegrafer istället för enskilda punkter. För att ändra till en linjegraf anger vi bara `type=''l''`.

```
plot(Nile$years, Nile$x, type="l")
```



2. Precis som när det gäller punkter kan vi använda olika linjetyper med argumentet `lty`. Prova linjetyp 2 t.o.m. 10. Tips: använd en for-loop!

```
plot(Nile$years, Nile$x, type="o", lty=2)
```

3. Vill vi både ha linjer och punkter kan vi använda `type=''o''`.

```
plot(Nile$years, Nile$x, type="o", lty=2, pch=3)
```

4. En sista typ av linjegraf är en trappstegsgraf:

```
plot(Nile$years, Nile$x, type="s")
```

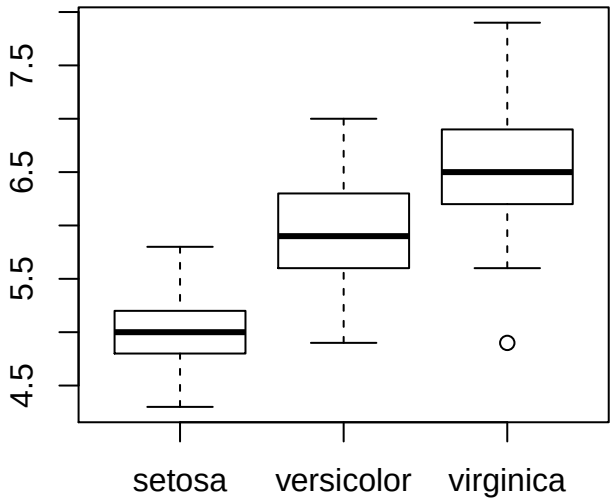
5. Pröva att lägg till argumentet `lwd=3` i plotten ovan. Vad innebär detta? Testa att köra `?par`

1.2.3 Boxplot

Vill vi jämföra olika fördelningar efter en kategorisk variabel gör vi det med fördel med en boxplot

1. Nedan finns kod för att producera en boxplot.

```
boxplot(Sepal.Length~Species, data=iris)
```



2. `Sepal.Length~Species` är ett exempel på formel-objekt. Formel-objekt används i olika sammanhang i R, men ett vanligt exempel är när linjär regerssion ska skattas med funktionen `lm()`. Testa att köra `?formula`. Kör sedan koden nedan:

```
Sepal.Length~Species
y<-Sepal.Length~Species
class(y)
str(y)
```

3. Precis som i tidigare diagram är det enkelt att lägga till färger.

```
boxplot(Sepal.Length~Species, data=iris, col=c("blue", "green", "red"))
```

4. Eller att lägga till rubriker.

```
boxplot(Sepal.Length~Species, data=iris, main="Blommor!")
```

1.3 Grafiska inställningar och tillägg

Ovan har vi sett en hel del av de figurer som går att producera. Nedan kommer lite mer inställningar och tillägg vi kan göra när vi arbetar med grafik i R.

1. För att kontrollera vilka värden som ska vara med på x- och y-axeln i en plot används argumenten `xlim=` och `ylim=`. Kör koden nedan. Ändra värdena på `xlim=` och `ylim=` och se vad som händer med plotten.

```
plot(Nile$years, Nile$x, type="s")
plot(Nile$years, Nile$x, type="s", xlim=c(1900,1945), ylim=c(600, 1200))
```

2. Vill vi lägga på punkter i grafen använder vi `points()`. Pröva att lägga till punkterna i plotten:

```
points(Nile$years, Nile$x, pch=20)
```

3. Vi kan också lägga till godtyckliga linjer (t.ex. som referenser) med `abline()`.

```
abline(v=1920, lty=4)
abline(h=900, lty=10)
```

4. Vad händer om du ändrar värdet i "`v=`" och "`h=`" till numeriska vektorer?

5. `abline()` kan också användas för att rita ut räta linjer med räta linjens ekvation (om vi ex. anpassat en regressionsmodell):

$$f(x) = a + bx$$

```
abline(a=6130,b=-2.7)
```

6. Vill vi ha flera grafer i en använder vi `par(mfrow=c(3,2))`. Men det vi säger till R med detta kommando är att vi vill ha tre rader och två kolumner med figurer. Dessa struktur kommer att fyllas radvis. Pröva att köra denna kod och skapa därefter 6 figurer (vilka som hellst). Ändra till en 2×2 figur och skapa på samma sätt fyra figurer. `par(mfrow=c(1,1))` återställer till en plot-struktur med en figur. Argumentet `mfcol=c(3,2)` fyller kolumnvis.

1.4 Spara figurer

I många fall vill spara specifika grafer i olika format. I R finns ett antal olika format som kan användas. De vanligaste är TIFF, BNP, JPEG, PNG och PDF. I alla fall används den aktuella funktionen för formatet `tiff()` för TIFF, `pdf()` för PDF o.s.v.

1. De olika grafikfunktionerna använder olika argument, men gemensamt är att vi först anger vilket format vi vill använda, sedan skapar vi vår figur och därefter stänger vi av "utskriften" till denna fil. Kör koden nedan för ett exempel:

```
jpeg(filename = "minJPEG.jpeg", width = 480, height = 480)
plot(Nile$years, Nile$x, type="l")
dev.off()
```

2. Figurer som skrivs ut på detta sätt hamnar i "working directory".

3. För pdf:er finns det ett snabbare sätt att snabbt skriva ut en figur som pdf:

```
plot(Nile$years, Nile$x, type="l")
dev.copy2pdf(file="MinNilenPlot.pdf")
```

4. Pröva nu att själv skriva ut en av dina figurer ovan i PNG- och TIFF-format.

1.5 * Extraproblem: Skapa en egen graf

1. Ladda in datasetet geyser med `data(faithful)`.
2. Testa att göra två histogram, dels över `waiting` och `eruptions`.
   (a) Ändra antalet `breaks` i histogrammen. Hur ser det ut om du har väldigt många eller vädigt få?
   (b) Pröva att ändra färg med `col=''blue''`. Testa att ändra färgen till `col=c(1,2,3)`
   (c) Ändra nu rubriken och axeltexterna med `xlab=` och `ylab=`.
   (d) Spara ned de två histogrammen i en figur (förslagsvis över och under varandra) som en pdf.
3. Gör en scatterplot av `waiting` mot `eruptions`. Huvudrubrik ska vara "Old faithful".
4. Det verkar finnas två tydliga kluster.
   (a) Ge de olika klustren olika färger.
   (b) Ge de olika klustren olika punktsymboler.
5. Lägg till en bildtext (eng: legend) till figuren (kolla på `?legend`) enligt nedan. Testa att lägga "legend" på någon annan plats i plotten.

```
legend("topleft", pch = c(1, 2), col = c("red", "blue"), legend = c("clu1", "clu2"))
```

6. Spara ned även denna graf i pdf-format.

Kapitel 2

Slumptal och simulering

2.1 Täthetsfunktioner

En central del inom statistik och analys handlar är simulering. Det ett stortantal fördelningar vi kan simulera och för vilka vi har täthetsfunktion (pdf), den kulmulativa fördelningsfunktionen (cdf) och den inversa kulmulativa fördelningsfunktionen. De flesta fördelningar har fyra varianter:

Prefix	Beskrivning	Exempel
`r`	simulera från fördelningen	`rnorm()`
`d`	täthetsfunktionen (pdf)	`dnorm()`
`p`	kulmulativ fördelninsgfunktion (cdf)	`pnorm()`
`q`	inversa kulmulativa fördelningsfunktionen	`qnorm()`

För att se vilka fördelningar som finns förinstallerade med R se `?distribution`.
Vill vi exempelvis simulera 100 tal från $\mathcal{N}(10, 1^2)$ gör vi på följande sätt:

```
minNormal <- rnorm(n=100, mean=10, sd=1)
```

1. Skapa en vektor med 30 slumptal från den uniforma fördelningen $\mathcal{U}(min = 1, max = 10)$. [**Tips!** `runif()`]
2. Skapa en vektor med 200 slumptal från normalfördelningen med medelvärde 1 och varians 2, döp den till `minNorm`.
3. Skapa ett histogram över `minNorm` [**Tips!** `hist()`]
4. Skapa en vektor med slumptal av längd 50 från Poissonfördelningen med medelvärde 8, döp den till `minPoission`
5. X är poissonfördelad med medelvärde 3. Uppgiften är nu att skapa ett stapeldiagram (se koden nedan) över sannolikheterna att X antar värdena $0, 1, ..., 11, 12$.
6. Låt X vara samma som ovan. Vad är sannolikheten att X antar följande värden [**Tips!** `ppois`()]:
   (a) Sitt medelvärde?
   (b) Mindre än eller lika med sitt medelvärde?
   (c) Större än sitt medelvärde?
   (d) Ett udda tal? Ni behöver inte ta hänsyn till tal som är större än 12.
   (e) Talen 4 till 6?
7. Skapa en vektor med 50 slumptal från t-fördelningen, med medelvärde 1 och 5 frihetsgrader (degrees of freedom), döp den till `minT`
8. Gör ett histogram över `minPoission` och över `minT`. Testa att ändra argumentet `breaks` i `hist()` till några olika värden.

2.2 sample() och set.seed()

Vill vi dra slumptal från ett antal element använder vi funktionen sample(). Med denna funktion kan vi dra ett stickprov (med eller utan återläggning) från en given vektor. Om vi exempelvis vill dra 5 slumpmässiga värden mellan 1 och 10 **med** återläggning gör vi det på följande sätt i R:

```
sample(x=1:10, size=5, replace=TRUE)

[1] 8 8 6 4 8
```

1. Dra ett stickprov ($n = 5$) **utan** återläggning från sekvensen `10:20`.
2. Upprepa uppgiften ovan men **med** återläggning.
3. Vi kan på samma sätt dra slumpmässiga element från andra vektorer (exempelvis Textvektorer). Tänk dig att du och några vänner ska organisera en fest. Skapa vektorn `namn`, som ska innehålla namnen för minst tre personer som strängar. Två personer behövs för att laga mat och två för att diska. Välj slumpmässigt vilka som ska göra de olika uppgifterna. Samma person ska kunna bli vald för båda uppgifterna.
4. Inte sällan vill vi att våra simuleringar och analyser ska vara reproducerbara, d.v.s. att vi ska få samma resultat varje gång vi gör en simulering. För detta används funktionen `set.seed()` och funktionen tar ett godtyckligt heltal för att initiera slumptalsgeneratorn i R.
   Upprepa uppgiften ovan två gånger, blir resultatet det samma? Upprepa två gånger till men kör först `set.seed(1234)` innan varje gång.Får du nu samma resultat?
5. Upprepa nu uppgiften ovan igen, men ändra argumentet `prob` så att de olika personer har olika sannolikhet att bli vald samt att du har sannolikheten 0 att bli vald. ;)

2.3 Exempel: `sum_of_dice()`

1. Nu ska ni testa att simulera olika tärningskast. I uppgiften betyder D6 en vanlig 6-sidig tärning (med sidorna 1,2,3,4,5,6) där alla utfall har samma sannolikhet (1/6).
   (a) Skriv en funktion `my_dice()`, som generar n stycken kast från en D6 och returnerar en vektor med kasten. [**Tips!** sample()]
   (b) Skriv en funktion `sum_of_dice()` som generar följande slumptal: $Y_k = \sum_{n=1}^{N} X_n$, där X är ett tärningskast från funktionen `myDice()`, N är ett heltal, k går från $1, 2, 3, ..., K$. Funktionen ska ha N och K som argument, `sum_of_dice(N, K)`. Funktionen ska alltså kasta N stycken D6, summera dessa, sen upprepa det K stycken gånger.

```
sum_of_dice <- function(N,K){
res <- integer(K)
for (k in 1:K){
res[k] <- sum(sample(1:6,size=N, replace=TRUE))
}
return(res)}
```

   (c) Testa nu `sum_of_dice(N, K)` med $N = 3, 5$ och $K = 100, 1000, 3000$. Plotta resultaten i histogram och beräkna också medelvärde, standardavvikelse, min och max. Nedan ser ni några exempel på tester.

```
set.seed(3827)
sum_of_dice(N=3,K=10)

 [1]  8  7 11  8 16 14 11 12 14 10

sum_of_dice(N=5,K=10)

 [1] 16 19 17  8 14 20 21 22 18 12
```

(d) Detta är ett klassiskt exempel på centrala gränsvärdessatsen.

Kapitel 3

Introduktion till R markdown och knitr

R markdown och knitr är ett system för att skapa dynamiska rapporter med R, vilket innebär att vi väver ihop R-kod, data och text i ett och samma dokument. Med knitr och markdown kan vi skapa reproducerbara analyser med full spårbarhet hur beräkning, databearbetningar och grafik skapats. Vår slutprodukt kan bli rapporter i Word, PDF eller HTML.

R markdown består av två delar. Dels **markup-språket** markdown som används för att skapa text, ekvationer, bilder m.m. och dels knitr för att integrera dokumentet med R-kod. knitr kommer från ordet knit (sticka) och är en ordlek med innebörden att vi stickar ihop R med, i detta fall, markdown. knitr kan dock användas med andra ordbehandlare som LaTeX och LyX.

Det som händer när vi **rendrerar** en Rmd fil är att knitr går igenom dokumentet, kör all R-kod och stoppar tillbaka svaret från R i markdownformat. När detta är gjort skapas ett word-, HTML- eller pdf-dokument från markdownfilen.

Det finns ett bra referensdokument **[här]** och en bra "fusklapp" **[här]**.

Under senare tid har reproducerbarhet när det gäller statistiska analyser kommit att bli allt mer centralt. Reproducerbarhet innebär att ett experiment eller forskningsresultat ska kunna återupprepas - reproduceras - av andra forskare. Idag innebär ofta reproducerbarheten att det finns krav på att hela analyser, med både text, data och kod ska kunna reproduceras av andra forskare. Mer information om reproducerbar forskning finns **[här]**.

3.1 Grunderna i markdown

1. För att skapa ett Rmd-dokument i R-Studio väljer vi New file → R markdown. Ange titel på dokumentet och HTML. Vi ska nu ha fått upp ett exempeldokument.

2. Vi borde fått upp ett dokument som borde se ut på följande sätt (första delen).

```
---
title: "My document"
author: "My name"
date: "1 januari 2015"
output: html_document
---

This is an R Markdown document. Markdown is a simple formatting
syntax for authoring HTML, PDF, and MS Word documents.
For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
```

3. Pröva att "knitta" dokumentet med knappen "knit HTML" i R-Studio. Pröva att "knitta" till både PDF, HTML och word-dokument.

4. I R-Studio finns hjälp om R-markdown med knappen "?". Pröva att få fram hjälp på detta sätt.

3.1.1 Grundläggande markdown

Lägg till det som framgår i kodblocken i ditt dokument och "knitta" ett HTML-dokument.

1. Rubriker skapas med `#`.

```
# Rubrik 1
## Rubrik 2
### Rubrik 3
```

2. Vill vi ha fet stil använder vi `**` och vill vi ha kursiv stil använder vi `*`.

```
I like both **bold** text and *italic* text.
```

3. Vill vi lägga till en länk använder vi hakparantes som anger länkens namn och därefter vanliga paranteser .

```
[Comics](http://xkcd.com/1239/) for the win!
```

4. Att lägga till en bild görs på ett liknande sätt som en länk, dock med `!` innan hakparantesen.

```
![My pic](http://imgs.xkcd.com/comics/social_media.png)
```

5. Vill vi lägga till listor använder vi `*` eller numrerar vår lista.

```
* lista (onumrerad)
* knitr
 + R
 + text
 + bilder

1. lista (numrerad)
2. Rmd
 + knitr
 + md
```

6. Det går också enkelt att skapa tabeller.

```
Kolumnrubrik 1 | Kolumnrubrik 2
-------------- | --------------
Cell 1         | Cell 2
Cell 3         | Cell 4
```

7. Mer exempel på formatering finns i referensbladet för R markdown [här].

3.1.2 Ekvationer

Ofta vill vi beskriva våra beräkningar med matematiska ekvationer. I markdown finns det möjlighet att skriva ekvationer med LaTeX ekvationsystem. För exempel på hur det går att skriva LaTeX-ekvationer, se följande **[minihandbok]**.

För att skapa ekvationer använder vi `$` och `$$`.

1. Vi börjar med att skapa följande ekvation i markdown.

$$a + b = 10$$

```
$$a+b=10$$
```

2. Vi kan skapa "inline"-ekvationer med `$`. Denna ekvation: $E = mc^2$, skrivs i markdown som:

```
Denna ekvation: $E=mc^2$
```

3. Vi kan skapa vilka ekvationer vi vill med LaTeX ekvationssystem. Fundera på koden nedan hur denna ekvation har byggts upp i LaTeX.

$$\frac{\int^{\infty}_{0}x^{2}_{a}dx}{10}$$

```
$$\frac{\int^{\infty}_{0}x^{2}_{a}dx}{10}$$
```

3.2 Integrera R-kod med knitr

Vi har nu gått igenom grunderna för att skapa ett markdowndokument. Den stora fördelen med markdown framgår dock först när vi kan integrera våra dokument med R-kod, grafik och tabeller.

Pröva att kopiera in koden nedan i ditt R-markdowndokument och "knitta" dokumenten till HTML eller pdf.

3.2.1 Inline-kod

1. Ibland kan vi vilja göra mindre, enklare beräkningar, direkt i ett dokument. För detta använder vi inlinekod. För att lägga till en enkel beräkning direkt i dokumentet används `` `r [R-kod här]` ``.

```
I know that 1 + 1 = `r 1 + 1`.
```

3.2.2 R-block - "chunks"

Med inline-kod kommer vi en liten bit mot ett dynamiskt dokument. Men vill vi ha mer komplicerade beräkningar, grafik eller tabeller måste vi skapa dessa i R-block, eller "chunks". Dessa styrs med så knitr-alternativ. Samtliga knitr-alternativ för kodblock finns listade **[här]**.

1. För att skapa ett R-block används följande kod.

````
```{r}
a <- 1 + 1
a
```
````

2. Om koden ovan körs kommer både R-koden att synas och resultatet. Vill vi inte vis R-koden anger vi bara `echo=FALSE` som knitr-alternativ.

````
```{r, echo=FALSE}
a <- 1 + 1
a
```
````

3. Vill vi istället dölja resultatet anger vi `results='hide'`.

````
```{r, results='hide'}
a <- 1 + 1
a
```
````

4. Vi kan också låta bli att köra koden med `eval=FALSE`.

````
```{r, eval=FALSE}
a <- 1 + 1
a
```
````

5. Det går självklart också att kombinera alternativ.

````
```{r, eval=FALSE, echo=FALSE}
a <- 1 + 1
a
```
````

3.2.3 Grafik

En av de stora fördelarna med knitr och Rmd är att vi i R kan skapa grafik som direkt skapas och sätts in i dokumentet.

1. För att lägga in ett diagram skapar vi diagrammet som vanligt i R. Under motorhuven skapas en grafikfil som sätts in i dokumentet automatiskt.

````
```{r, echo=FALSE}
data(faithful)
hist(faithful$waiting)
```
````

2. Även här finns flera knitralternativt för att styra hur grafiken ska placeras i dokumentet. `fig.height` och `fig.width` styr storleken och `fig.align` styr om diagrammet ska vara höger-, vänster- eller centrerat till mitten.

````
```{r, echo=FALSE,fig.width=3, fig.height=3, fig.align='center'}
data(faithful)
hist(faithful$waiting)
```
````

3.2.4 Tabeller

Utöver grafik och text är tabeller vanligt i statistiska analyser och rapporter. Med funktionen `kable()` i knitr-paketet kan vi skapa tabeller direkt från R-objekt.

1. För att skapa tabeller direkt behöver vi dels använda funktionen `kable()`, men vi behöver också ange att funktionen direkt skapar en markdowntabell som ska ses som markdownkod. Därför behöver vi ange att resultatet ska vara `'asis'`.

````
```{r, echo=FALSE, results='asis'}
knitr::kable(head(faithful))
```
````

2. Vi kan, precis som med grafiken, styra tabellernas grundutseende en del.

````
```{r, echo=FALSE, results='asis'}
knitr::kable(head(faithful), digits = 2, align = c("l", "c"))
```
````

3. Det finns mer avancerade funktioner för att skapa tabeller automatiskt från ex. linjära regressionsmodeller med paketen `xtable` och `tables`.

Kapitel 4

Input och output (I/O): Data på webben

Allt mer data lagras på webben med olika former av molntjänster. Anledningen till att data lagras på webben kan vara flera:

- Reproducerbarhet/öppna data för andra forskare
- Samarbete kring datainsamling
- Stabilitet
- Versionshantera förändringar i data

Nedan finns exempel från vanliga molntjänster och hur man kan ladda ned data från dessa direkt till R. Ett av paketen för att hantera denna typ av datainläsning är `repmis`, som kan läsa in de flesta csv, xlsx och Rdata-filer.

4.1 `downloader`

Ibland kan det vara så att vi vill ladda ned filer från R, men inte läsa in dem. Vi kanske vill ladda ned ett antal filer och sedan läsa in dem en och en. För att ladda ned filer i R finns funktionen `download.file()`. Dock kan det ibland vara lite klurigt att få den att fungera för så kallade secure http (https) adresser. Av bekvämlighet har därför paketet `downloader` skapats som gör nedladdning av filer mycket enkelt och bekvämt oberoende av operativsystem.

1. För att ladda ned data anger vi dels sökvägen till den aktuella filen och sedan sökvägen dit filen ska laddas ned. I detta fallet laddas filen ned till min working directory.

```
library(downloader)
apple_remote <- "https://github.com/STIMALiU/KursRprgm/blob/master/Labs/DataFiles/Apple.RData"
apple_local <- paste0(getwd(), "/Apple.RData")
download(url = apple_remote, destfile = apple_local)
```

4.2 Github

github är framförallt en tjänst för att versionshantera programkod i molnet. Dock används det också mycket för att lagra enklare datamaterial. Särskilt material som förändras mycket där vi behöver kunna följa vilka förändringar som gjorts.

Kör nedanstående exempel för att läsa in filen polls från github. Filen innehåller (nästan) samtliga opinionsmätningar i Sverige sedan 1998. Mer information finns **[här]**.

```
library(repmis)
data_url <- "https://github.com/MansMeg/SwedishPolls/raw/master/Data/Polls.csv"
polls <- repmis::source_data(data_url, sep = ",", dec = ".", header = TRUE)

Downloading data from: https://github.com/MansMeg/SwedishPolls/raw/master/Data/Polls.csv

SHA-1 hash of the downloaded data file is:
679dde5aac3b4fb6d011df4f36a2989dc3d9476d
```

```
head(polls[,1:12])

  PublYearMonth  Company    M  FP   C  KD    S   V  MP   SD  FI Uncertain
1      2016-feb Demoskop 27.2 5.8 6.6 3.0 23.0 7.1 5.5 18.8 1.9       7.7
2      2016-jan    Ipsos 25.8 6.6 7.2 2.7 24.7 6.6 6.5 16.7 1.3      16.7
3      2016-jan    Novus 23.9 6.1 6.4 3.6 24.4 6.6 5.7 20.3 2.1       7.4
4      2016-jan   YouGov 21.4 4.0 6.8 3.1 21.6 7.5 4.2 28.8 2.0        NA
5      2016-jan   Inizio 21.3 5.1 8.8 4.4 23.2 8.0 4.6 22.2 1.6       8.8
6      2016-jan   Sentio 20.1 3.9 7.9 2.8 23.0 6.9 5.6 25.5 2.9        NA
```

4.3 Google docs

Google docs eller google drive är en molntjänst för kalkylblad i molnet. Vi ska nu pröva att läsa in ett publicerat kalkylblad. **Här** finns dokumentet vi ska läsa in (det är samma data som faithful-datasetet i R). I sökvägen kan vi se textsträngen `16uE3bb_7rHt_g4zOBXAVHM40N3YqRIXLPQwsE1yenAQ`. Detta är det unika id:t för detta google-kalkylblad.

Vi kommer använda paketet RGoogleDoc som klarar att göra mycket med google docs (som att spara dataset, läsa textfiler m.m.). Vi kommer dock endast använda det för att läsa in data i R.

För att läsa in vårt material gör vi det i tre steg.

1. Först läser vi in paketet och skapar vi en koppling till det unika google doc id:t.

```
# för att installera:
devtools::install_github("duncantl/RGoogleDocs")

Skipping install for github remote, the SHA1 (75c995d0) has not changed since last
install.
Use 'force = TRUE' to force installation

library(RGoogleDocs)

Loading required package: methods

Attaching package: 'RGoogleDocs'
The following object is masked from 'package:methods':
    getAccess

google_ws_object <- publicWorksheet("16uE3bb_7rHt_g4zOBXAVHM40N3YqRIXLPQwsE1yenAQ")
```

2. I nästa steg laddar vi ned information om det aktuella kalkylbladet (hur många blad m.m.). Under con anger vi en koppling till google doc. Är ett dokument öppet kan vi ange NULL. Annars behöver vi ange vårt google acount om vi exempelvis vill komma åt dokument som inte är publika (ex. våra egna data från ett google forms).

```
google_worksheets <- getWorksheets(google_ws_object, con = NULL)

Found more than one class "URI" in cache; using the first, from namespace 'XML'
```

3. Det sista steget är att läsa in dokumentet i R. Det kan vi göra genom att ange vilket blad vi vill läsa in.

```
my_faithful <- sheetAsMatrix(google_worksheets[[1]], con = NULL, header = TRUE)
head(my_faithful)

  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
```

4.4 Dropbox

Vi kan också vilja läsa in data från dropbox på ett enkelt sätt. För att kunna komma åt data på dropbox behöver vi först dela den fil vi vill läsa in i R. Mer information om hur man delar filer med dropbox finns **[här]**.

När vi delar en fil på dropbox för vi en länk. Nedan är ett exempel på en länk:
https://www.dropbox.com/s/pbxmllhmoax5zn6/google.csv?dl=0

För att läsa in denna fil i R använder vi repmispaketet. Vill vi ha full kontroll på dropbox (skapa och läsa dolda filer m.m.) direkt från R finns paketet `rDrop`.

Länken från dropbox består av två delar, ett öppet dokument-id (`pbxmllhmoax5zn6` i exemplet ovan) och ett filnamn (`google.csv`). Med dessa två kan vi enkelt läsa in en fil i R på följande sätt:

```
library(repmis)
my_google <-
  repmis::source_DropboxData(file = "google.csv",
                             key = "pbxmllhmoax5zn6",
                             sep=";", dec = ",")

Error in repmis::source_DropboxData(file = "google.csv", key = "pbxmllhmoax5zn6", : unused
arguments (file = "google.csv", key = "pbxmllhmoax5zn6", sep = ";", dec = ",")

head(my_google)

Error in head(my_google): object 'my_google' not found
```

Kapitel 5

pxweb

Statistiska centralbyrån har utvecklat ett API för att ladda ned data direkt utan att först gå via deras webbplats. En koppling till deras API finns installerat som paketet `pxweb` i R. Detta paket fungerar för samtliga pxweb-apier. Men det största api:et är utan tvekan SCB:s.

1. Börja med att installera paketet `pxweb` och läs in det i R.

```
install.packages("pxweb")
library(pxweb)
```

2. Det går att få en snabb introduktion till detta paket med funktionen `vignette()`. Dock är denna labb mer utförlig än vignetten.

```
vignette(topic="pxweb")
```

5.1 Navigera i SCB:s API

Ofta vet vi inte exakt vad för data vi vill ha utan vill navigera igenom SCB:s databaser på ett effektivt sätt. Detta gör vi med funktionen `interactive_pxweb()`. Vill vi leta upp ett givet datamaterial som vi vill spara måste vi tillskriva datamaterialet till ett objekt.

Statistiska centralbyråns API består av två delar:

- Navigera mellan de olika datamaterialen.
- Välja ut delar av datamaterialet vi vill ladda ned.

1. För att navigera i Statistiska centralbyråns API kan vi använda funktionen `interactive_pxweb()` som listar förinstallerade api:er och ger möjlighet att välja vilket api vi vill hämta data från.

```
mitt_data <- interactive_pxweb()
```

2. För att anknyta direkt till Statistiska centralbyråns api kan då följande kod användas:

```
mitt_data <- interactive_pxweb(api = 'api.scb.se', version = 'v1', lang = 'sv')
```

3. Vi ska nu leta upp andelen arbetslösa från den senaste arbetskraftsundersökningen och spara ned detta som ett datamaterial i R. Vi befinner oss nu i den översta menyn. Ange `1` för att gå vidare till menyn för statistik om arbetsmarknad:

```
 ...
21. [UF] Utbildning och forskning
Enter the data (number) you want to explore: ('esc' = Quit, 'b' = Back)
1: 1
```

4. Pröva att "gå tillbaka" till huvudmenyn med `b`.
5. Navigera dig fram till följande datamaterial som vi ska läsa in:
   `[NAKUArblosaTK] Arbetslösa 15-74 år (AKU) efter arbetslöshetstidens längd, kön och ålder. Kvartal 2005K2 - 2015K4`
6. Vi ska nu ladda ned detta datamaterial till vår R-session. R frågan nu om du vill ladda ned datamaterialet (eller om vi bara vill ha kod för att ladda ned materialet längre fram). Ange `y` (yes).
7. Nästa steg efterfrågas om du vill ha materialet i orginalformat eller som en färdigformaterad data.frame i R. Ange `y` (yes).
8. Som ett sista steg efterfrågas nu om du vill ha koden för att ladda ned denna data direkt i R. Ange `y` (yes).
9. Nu är vi inne i den del av API:et som anger vilka delar av materialet du vill ladda ned. Varje variabel kommer nu dyka upp och vi får ange vilka data vi vill ha.
   Vill vi bara ha en kategori anger vi den siffran, vill vi ha allt material anger vi `*` och vill vi ha delar anger vi det antingen som en sekvens med `:` eller avgränsat med `,`.
   (a) För variabel `ARBETSLÖSHETSTID` ange **allt** (med `*`) som den del av materialet du vill ha.
   (b) För variabel `KÖN` ange `totalt` som den del du vill ha
   (c) För variabel `ÅLDER` ange grupperna `15-24`, `25-54` och `55-74` som den del du vill ha.
   (d) För variabel `TABELLINNEHÅLL` ang `Arbetslösa, 1000-tal` som den variabel du vill ha.
   (e) Variabeln `KVARTAL` innehåller många kategorier. Om det är fler än 10 kategorier så döljs de mittersta kategorierna. För att se alla kategorierer, ange `a`.
   (f) Välj nu ut tidsperioden 2010K1 till senaste kvartalet hos SCB.
10. Nu ska materialet laddas ned. Du borde också få ut följande kod:

```
myDataSetName <-
get_pxweb_data(
url = "http://api.scb.se/OV0104/v1/doris/sv/ssd/AM/AM0401/AM0401L/NAKUArblosaTK",
dims = list(Arbetsloshetstid = c('*'),Kon = c('1+2'),
Alder = c('15-24', '25-54', '55-74'),
ContentsCode = c('AM0401F0'),
Tid = c('2005K2', '2005K3', '2005K4', '2006K1', '2006K2', '2006K3', '2006K4',
'2007K1', '2007K2', '2007K3', '2007K4', '2008K1', '2008K2', '2008K3',
'2008K4', '2009K1', '2009K2', '2009K3', '2009K4', '2010K1', '2010K2',
'2010K3', '2010K4', '2011K1', '2011K2', '2011K3', '2011K4', '2012K1',
'2012K2', '2012K3', '2012K4', '2013K1', '2013K2', '2013K3', '2013K4',
'2014K1', '2014K2', '2014K3', '2014K4', '2015K1', '2015K2', '2015K3',
'2015K4')),clean = TRUE)
```

11. Titta på det material du laddat ned.

5.2 Ladda ned data från SCB direkt med kod

Ofta vill vi ladda ned/komma åt data från SCB som en löpande del i en analysprocess där vi använder de senaste data från SCB. Då vill vi använda pxweb, inte interaktivt, utan som R-kod.

1. Spara ned koden du fick ovan och kör den för att ladda ned datamaterialet direkt. (Nu sparas den som `myDataSetName`).

2. Pröva att ändra argumentet clean till `FALSE`. Studera hur materialet ser ut. Detta är den "råa" data som laddas ned från SCB.

3. Vi vill nu ladda ned data för alla tidpunkter. Ändra variabeln tid till `'*'` och ladda ned datat på nytt.

5.3 * Extraproblem: Andra api:er

Vi har nu prövat Statistiska centralbyråns API, men allt fler offentliga myndigheter (och företag) lägger ut sina resultat i form av ett pxweb-api. Med api_catalogue() är det möjligt att se vilka andra api:er som nu finns inkluderade.

1. Pröva att navigera i något annat API än Statistiska centralbyråns och ladda ned data därifrån.

Litteraturförteckning

[1] Edward R Tufte and PR Graves-Morris. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.

Del II

Inlämningsuppgifter

Tips!

Inlämningsuppgifterna innebär att konstruera funktioner. Ofta är det bra att bryta ned programmeringsuppgifter i färre små steg och testa att det fungerar i varje steg.

1. Lös uppgiften med vanlig kod direkt i R-Studio (precis som i datorlaborationen ovan) utan att skapa en funktion.
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i 1. ovan som en funktion.
4. Testa att du får samma resultat som i testexemplen, nu med funktionen.

Automatisk återkoppling med `markmyassignment`

Som ett komplement för att snabbt kunna få återkoppling på de olika arbetsuppgifterna finns paketet `markmyassignment`. Med detta är det möjligt att direkt få återkoppling på uppgifterna i laborationen, oavsett dator. Dock krävs internetanalsutning.

Information om hur du installerar och använder `markmyassignment` för att få direkt återkoppling på dina laborationer finns att tillgå **här**.

Samma information finns också i R och går att läsa genom att först installera `markmyassignment`.

```
install.packages("markmyassignment")
```

Om du ska installera ett paket i PC-pularana så behöver du ange följande:

```
install.packages("markmyassignment",lib="sökväg till en mapp i din hemkatalog")
```

Tänk på att i sökvägar till mappar/filer i R i Windowssystem så används ‘‘\\’’, tex ‘‘C:\\Users\\Josef’’.

Därefter går det att läsa information om hur du använder `markmyassignment` med följande kommando i R:

```
vignette("markmyassignment")
```

Det går även att komma åt vignetten **här**. Till sist går det att komma åt hjälpfilerna och dokumentationen i `markmyassignment` på följande sätt:

```
help(package="markmyassignment")
```

Lycka till!

Kapitel 6

Inlämningsuppgifter

För att använda `markmyassignment` i denna laboration ange:

```
library(markmyassignment)
```

```
Loading required package: yaml
Loading required package: testthat
Loading required package: httr
```

```
lab_path <-
"https://raw.githubusercontent.com/STIMALiU/KursRprgm/master/Labs/Tests/d6.yml"
suppressWarnings(set_assignment(lab_path))
```

```
Assignment set:
D6 : Statistisk programmering med R: Lab 6
```

6.1 `fast_swe_pop()`

Du ska nu skapa en funktion, utan argument, som kopplar upp sig mot SCB:s api med `pxweb`, laddar ned befolkningsstatistik och summerar upp befolkningen i Sverige för varje år.

Tips! `aggregate()`

```
library(pxweb)
pop <- fast_swe_pop()
```

```
head(pop)
```

```
   year population
 1 1968    7931193
 2 1969    8004270
 3 1970    8081142
 4 1971    8115165
 5 1972    8129129
 6 1973    8144428
```

```
tail(pop)
```

```
   year population
43 2010    9415570
44 2011    9482855
45 2012    9555893
46 2013    9644864
```

```
47 2014    9747355
48 2015    9851017
```

6.2 sum_of_random_dice()

Funktionen `sum_of_dice()` i sektionen Slumptal och Statistik summerar värdet från ett fixt antal tärnigar, nu ska ni skriva en funktion som kan summera ett slumpmässigt antal tärningar. Antalet tärningar ska vara Poissonfördelat med en parameter λ på följande sätt.

$$\begin{aligned} Y &\sim \text{Po}(\lambda) \\ X &= \sum_{i}^{Y} Z_i \end{aligned}$$

där Z_i är utfallet från en sexsidig tärning. Skapa en funkton `sum_of_random_dice()`. Argumenten ska vara:

- `K`: antal dragningar från slumpfördelningen
- `lambda`: ett positivt kontinuerligt tal (parameter i poissonfördelningen)
- `my_seed`: ett slumpfrö som styr slumptalsgenereringen, default ska vara `NULL`.

Funktionen `sum_of_random_dice()` ska returnera en `data.frame` med summan av ögonen på de slumpmässigt antalet tärningar samt antalet kastade tärningar.

Ett förslag på hur detta kan implementeras finns här:

- Ändra seeden till: `set.seed(mySeed)`.
- Sätt upp en tom `data.frame` som namnges `result`, som ska ha `K` rader och 2 kolumner. Första komunen ska ha namnet `value` och den andra `dice`.
- Gör följande for-loop över vektorn `1:K`
  - Dra ett slumptal från en poissionfördelning med parameter `lambda`. Spara slumptalet i `current_number`, vilket är antalet tärningar. Spara `current_number` i kolumnen `dice` på aktuell rad i `result`.
  - Anropa `sum_of_dice()` med argumenten `K=1` och `N=current_number`, spara resultatet i kolumnen `value` på aktuell rad i `result`. Observera att om `curret_number=0` så ska summan bli `0`.
- Returnera `result`.

Obs! Om du inte implementerar funktionen på detta sätt kan den fortfarande fungera korrekt, men eftersom slumptalen används i olika ordning kan det vara så att du inte får samma resultat som i exemplen nedan.

Testa om testfallen nedan fungerar:

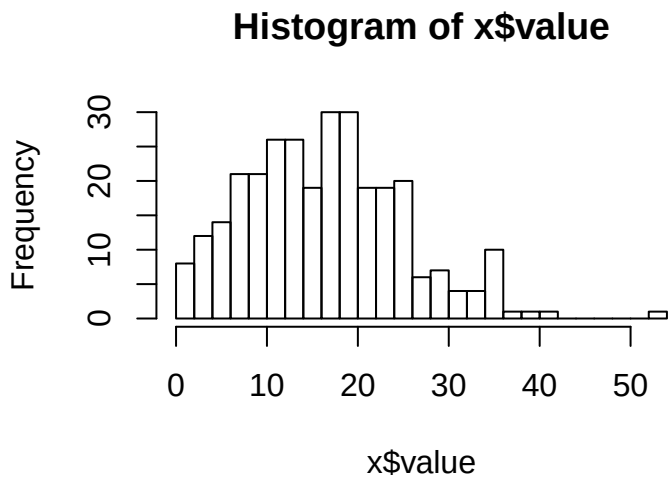
```
sum_of_random_dice(K=5,lambda=3,my_seed=42)

  value dice
1    21    5
2    13    4
3    17    4
4    27    7
5     4    1

sum_of_random_dice(K=5,lambda=8,my_seed=4711)
```

```
  value dice
1    41   13
2    20    5
3    22    5
4    25    9
5    24    6
```

```
# denna körnining ska ge olika resutlat vid olika anrop, för att testa my_seed=NULL
sum_of_random_dice(K=5,lambda=3)
```

```
  value dice
1    24    7
2     5    1
3     3    1
4    11    3
5     2    1
```

```
x <- sum_of_random_dice(K=300,lambda=5,my_seed=42)
hist(x$value, 20)
```



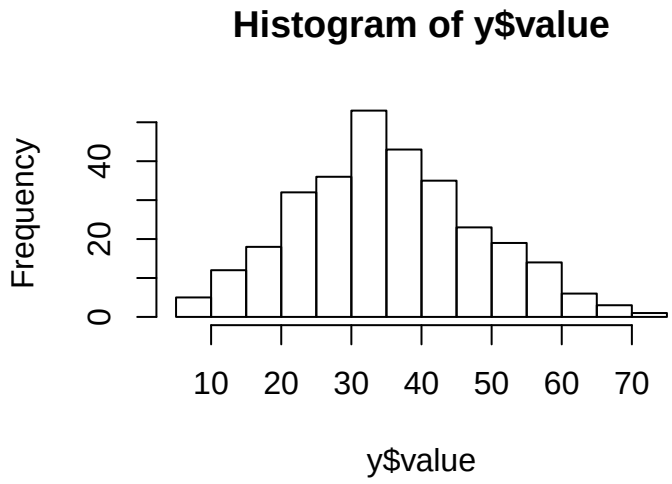

```
mean(x$value)
```

```
[1] 16.893
```

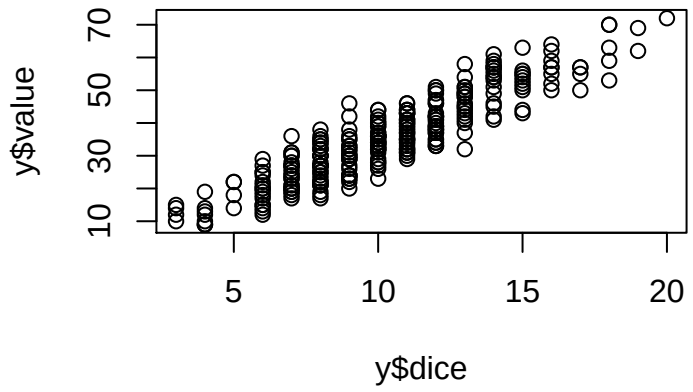
```
sd(x$value)
```

```
[1] 8.7547
```

```
y <- sum_of_random_dice(K=300,lambda=10,my_seed=4711)
hist(y$value, 20)
```




```
mean(y$value)

[1] 35.813

sd(y$value)

[1] 12.852

plot(y$dice, y$value)
```



6.3 Miniprojektet del I

En de av denna laboration är att genomföra miniprojektet del I. Se kurshemsidan för detaljer.