Datorlaboration 7

Måns Magnusson

10 mars 2015

Instruktioner

- Denna laboration ska göras i grupper om **två och två**. Det är viktigt för gruppindelningen att inte ändre 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**.
- Utgå från laborationsfilen som går att ladda ned **här**
- Laborationen består av två delar:
  - Datorlaborationen
  - Inlämningsuppgifter
- I laborationen finns det extrauppgifter markerade med *. Dessa kan hoppas över.
- Deadline för labben framgår på **kurshemsidan**

Innehåll

Del I

Datorlaboration

Kapitel 1

Enklare statistisk analys och simulering

1.1 Enklare statistiska metoder

1.1.1 Korstabulering och χ^2-tester

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

```
data(iris)
```

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

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

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

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

4. Vill vi använda flera kategoriska variabler med `ftable()`:

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

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

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

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

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

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

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

1.1.2 t-test

1. Vill vi jämföra två grupper avseende en kontinuerlig variabel använder vi funktionen `t.test()`. Inledningvis kan vi testa om `Sepal.Length` för `iris` versicolor är mindre 6. Vi börjar med att plocka ut elementen för de olika arterna:

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

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

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

1.1.3 Sambandsmått

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

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

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

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

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

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

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

1.1.4 Beskrivande statistik

1. Hitta det hösta och minsta värdet i `iris` (för alla variabler). Ta reda på vilken rad dessa värden finns på med relationsoperatorn == och `which.max()`.
   (a) Beräkna medelvärden för alla kolumnerna med `colMeans()`.
   (b) Beräkna nu kvartiler för `Petal.Width` med funktionen `quantiles()`. Beräkna den 1 och 99 procentiga percentilen? [**Tips!** `?quantile`]

(c) Beräkna nu kvartiler för samtliga variabler.

(d) Testa nu att använda funktionen `summary()` på `Petal.Width` först och sedan på hela datasetet. Vad får du för resultat?

2. Skapa en logisk vektor som anger om en variabeln `Petal.Width` är större än 2. Spara denna variabel som `small` på följande sätt.

3. Använd `table()` för att se vilka arter (variabeln `Species`) som är `small`.

1.2 Slumptal och simulering

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.

1.2.1 `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]  9  8  9 10  9
```

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. ;)

1.3 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.

1.4 * Extraproblem

Skapa funktionen `expectedValue(x,p)`, som givet vektorn `x` (innehåller numeriska värden) och `p` (innehåller sannolikheter för värdena i `x`) ska beräkna det teoretiska väntevärdet:

$$E[X] = \sum_{i=1}^{K} x_i p_i(X = x)$$

där index i går över alla möjliga värden på x. Ex: Om X följer fördelningen i tabellen nedan, då blir väntevärdet: $0 \cdot 0.3 + 1 \cdot 0.1 + 2 \cdot 0.6 = 1.3$

x	p(x)
0	0.3
1	0.1
2	0.6

1. Skapa funktionen `variance(x, p)`, som givet vektorn `x` (innehåller numeriska värden) och `p` (innehåller sannolikheter för värdena i `x`) ska beräkna det teoretiska variansen. Läs [här] om du inte vet hur variansen beräknas.
2. Testa nu `expectedValue()` och `variance()` på fallen nedan.

```
x1<-1:50
p1<-x1/sum(x1)
x2<-1:30
p2<-(sin(x2)+1)/sum(sin(x2)+1)
```

3. Simulera ett urval från fördelningarna ovan av storlek 1000 och visualisera detta i ett histogram [**Tips!** `hist()`]
4. I vissa fall vill vi simulera data från en given sannolikhetsmodell. Det kan exempelvis vara en linjär modell som ser ut på följande sätt:

$$y_i = \alpha + \beta x_i + \epsilon_i$$

där $\epsilon_i \sim \mathcal{N}(0, \sigma)$. Detta görs enklast i flera steg. I detta fall simulerar jag även x_i samt sätter $\alpha = 2, \beta = 4, \sigma = 1$ och $n = 100$.

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

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

5. Kör koden ovan och visualisera det simulerade datamaterialet i ett spridningsdiagram [**Tips!** `plot()`]
6. Pröva lite olika värden för σ, β och α och visualisera de olika datamaterialen.

Kapitel 2

Introduktion till linjär regression

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

2.1 Anpassa en regressionsmodell

Den vanliga regressionmodellen bygger på modellen

$$y_i = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon_i$$

eller med matrisnotation

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

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

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

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

```
library(car)
data(Prestige)
library(MASS)
```

2. Läs på kort om de variabler som finns i datasetet med `?Prestige`.

3. Börja med att visualisera sambandet mellan `prestige` och `income`. [**Tips!** `plot()`]

4. Vi ska nu anpassa vår första linjära regressionsmodell. För att anpassa en modell i R behöver vi ange två saker, dels en formula som beskriver hur modellen ser ut, och sedan vilket dataset som innehåller variablerna vi vill ha i vår modell. För att anpassa en linjär regression med inkomst som oberoende variabel gör vi på följande sätt:

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


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

```
Coefficients:
(Intercept)       income
    27.1412       0.0029
```

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

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

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

7. Vill vi lägga till en variabel (andel kvinnor) gör vi på följande sätt:

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

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

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

9. Nu kommer också fördelarna med att ha definierat faktorvariabler. Lägger vi in en faktorvariabel förstår R detta automatiskt och "under the hood" skapas dummyvariabler för vilka koefficienter skattas.

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

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

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

2.2 Analysera resultatet från en linjär regression

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

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

```
summary(minModell)


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

Residuals:
   Min     1Q Median     3Q    Max
```

```
-33.01  -8.38  -2.38   8.43  32.08

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

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

2. Med denna får vi en sammanfattning av de flesta resultat i modellanpassningen. Pröva denna funktion på `minModell3`. Hitta följande storheter i sammanfattningen av modellen:
   β, σ, R^2, F-statistikan samt p-värdena för de olika β-koefficienterna.

3. Pröva att analysera modell 5 och 6 på samma sätt. Hur ser en kategorisk variabel samt interaktionseffekter ut i R?

4. Funktionen `summary()` ger en bra bild av resultatet, men vi vill ofta också studera en ANOVA-tabell för vår modells ingående variabler. För detta används funktionen `anova()`. Pröva den på modellerna 3, 5 och 6.

5. De flesta (icke-bayesianska) statistiker och forskare är mycket förtjusta i p-värden. För att få ut p-värden till ANOVA-tabellen använder vi oss av argumentet `test='Chisq'`. Pröva att på detta sätt testa för variabler i modell 5.

6. Vi kan också inkludera flera modeller i en ANOVA-tabell. Vad ger det för resultat?

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

2.2.1 Använda parametrar och resultat för vidare analys

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

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

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

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

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

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

2.3 Tester och diagnostik

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

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

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

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

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

   (a) Identifiera uteliggare

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

   (b) Identifiera observationer med starkt inflytande på modellen

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

   (c) Utvärdera linjäritetsantagande för regressionkoefficienterna

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

   (d) Studera multikollinearitet mellan regressionskoefficienterna

   ```
   vif(minModell)
   ```

   (e) Oberoende feltermer (framförallt för tidsserieregression)

   ```
   durbinWatsonTest(minModell)
   ```

   (f) Residualernas fördelning

   ```
   qqPlot(minModell)
   ```

   (g) Konstant varians (homoscedasticitet)

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

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

2.3.1 Anscombes data

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

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

```
data(anscombe)
```

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

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

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

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

Kapitel 3

Housing data

1. Ni ska nu analysera datamaterialet HUS, som innehåller information om en mängd hus. Följande variabler finns i data:

   - Försäljningspris (dollar)
   - Bostadsyta (kvadratfot)
   - Antal sovrum
   - Antal badrum
   - Förekomst av luftkonditionering, 1 = luftkonditionering finns, 0 annars
   - Antal bilar som garaget är konstruerat för
   - Förekomst av pool, 1 = pool finns, 0 annars
   - Byggår
   - Tomtstorlek (kvadratfot)

2. Läs in datamaterialet "HUS.csv" och spara det som `HUS`. Det finns även ett dataset "HUS_eng.csv", som har engelska namn på variablerna men innehåller samma data. Materialet finns att tillgå **[här]**.

3. Gör följande med `HUS`

   (a) Plotta en boxplot över variabeln `Försäljningspris`. Ta fram besrkivande statistik med `summary()` för `Försäljningspris`. Det verkar finnas en del outliers (extremt stora värden), dessa vill vi ta bort från data, kör följande kod:

   ```
   # ta bort alla hus som har pris st<U+00F6>rre <U+00E4>n 3:e kvartilen:
   index<-HUS[,1]<quantile(HUS[,1])[4]
   HUS<-HUS[index,]
   ```

   (b) Plotta ett histogram för `Försäljningspris` med `breaks=40`, (bonusfråga: ser fördelningen symmetrisk ut?)

   (c) Beräkna ett tvåsidigt konfidensintervall (KI) med $\alpha = 0.05$ för `Försäljningspris` (tips `?t.test()`) spara i `testPris`.

   (d) Välj ut KI från `testPris`. tips: `?t.test()` och läs under rubriken "Value". Kör sedan `str(testPris)`. Testa `class(testPris)`

   (e) Beräkna ett tvåsidigt KI med $\alpha = 0.10$ för `Försäljningspris`

   (f) Beräkna ett enkelsidigt (undre) KI med $\alpha = 0.01$ för `Försäljningspris`

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

   ```
   summary(HUS)
   ```

4. Skapa följande frekvenstabeller från HUS (tips: `?table()`)

   (a) För `Antal.sovrum`

   (b) För `Luftkonditionering`

   (c) Mellan variablerna `Antal.sovrum` och `Luftkonditionering`, spara som `sovLuftTab. Testa class(sovLuftTab)`

   (d) Mellan variablerna `Antal.sovrum` och `Antal.badrum`

5. Kör `prop.table(sovLuftTab)` och `prop.table(sovLuftTab,margin=1)`. Vad händer med tabellen?

6. Beräkna fishers exakta test för tabellen `sovLuftTab`, använd $\alpha = 0.05$**,** vad är noll-hypotesen? Kan vi förkasta den? (tips: `?fisher.test`())

7. Antag att variablen `Försäljningspris` representerar en hel population med hus. Dra ett slumpmässigt stickprov med 20 hus utan återläggning. Beräkna ett tvåsidigt KI ($\alpha = 0.01$) utifrån stickprovet för populationsmedelvärdet.

8. Antag nu att hela datasetet HUS är alla hus i en population. Dra ett stickprov slumpmässigt (dvs välj rader) på 40 hus utan återläggning. Gör följande beräkningar utifrån stickprovet:

   (a) Numeriska variaber: Beräkna ett tvåsidigt KI med $\alpha = 0.01$för populationsmedelvärdet.

   (b) Binära(0/1) variaber: Beräkna ett tvåsidigt KI med $\alpha = 0.01$för populationsandelen. Tips: `?prop.test(), ?table()`

Kapitel 4

Texthantering och regular expression i R med `stringr`

Gör följande delar i *Handling and Processing Strings in R* av Gaston Sanchez.

Kap 2 Hela

Kap 3 3.1, 3.3

Kap 4 4.2.1 - 4.2.3

Kap 5 5 - 5.2.2, 5.2.6, 5.3.1-5.3.2

Kap 6 6-6.1.3, 6.2.2, 6.4 - 6.4.1, 6.4.3, 6.4.5, 6.4.7, 6.4.9 - 6.4.10

Boken finns fritt tillgänglig **[här]**.

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.

För att installera markmyassignment krävs paketet devtools (som därför först måste installeras):

```
> install.packages("devtools")
> devtools::install_github("MansMeg/markmyassignment")
```

För att automatiskt återkoppla en laboration behöver du först ange vilken laboration det rör sig om på följande sätt:

```
> library(markmyassignment)
> set_assignment("[assignment path]")
```

där `[assignment path]` är en adress du får av läraren till varje laboration.

För att se vilka uppgifter som finns i laborationen kan du använda funktionen `show_tasks()` på följande sätt:

```
> show_tasks()
```

För att få återkoppling på en uppgift använder du funktionenen `mark_my_assignment()`. För att rätta samtiga uppgifter i en laboration gör du på följande sätt:

```
> mark_my_assignment()
```

Tänk på att uppgifterna som ska kontrolleras måste finnas som funktioner i R:s globala miljö. Du kan också kontrollera en eller flera enskilda uppgifter på följande sätt:

```
> mark_my_assignment(tasks="foo")
> mark_my_assignment(tasks=c("foo", "bar"))
```

Det går också att rätta en hel R-fil med samtliga laborationer. Detta är bra att göra innan du lämnar in din laboration. För att rätta en hel fil gör du på följande sätt:

```
> mark_my_assignment(mark_file = "[my search path to file]")
```

där `[my search path to file]` är sökvägen till din fil.

Obs! När hela filer kontrolleras måste den globala miljön vara tom. Använd `rm(list=ls())` för att rensa den globala miljön.

Kapitel 5

Inlämningsuppgifter

För att använda `markmyassignment` i denna laboration ange:

```
library(markmyassignment)

Loading required package: methods
Loading required package: yaml
Loading required package: testthat
Loading required package: httr

lab_path <-
"https://raw.githubusercontent.com/MansMeg/KursRprgm/master/Labs/Tests/d7.yml"
set_assignment(lab_path)

Assignment set:
D7 : Statistisk programmering med R: Lab 7
```

5.1 `diagonalize_matrix()`

Vi ska skapa en funktion, `diagonalize_matrix()`, som kan diagonalisera en godtycklig matris. Argumentet ska vara `X`. Funktionen ska först testa om matrisen är diagonaliserbar, d.v.s. om matrisens egenvektorer är linjärt oberoende (d.v.s. determinanten av egenvektorerna är skild från 0). Av numeriska skäl kan detta vara svårt att uppfylla. I denna uppgift ska därför determinanten avrundas till 10 decimaler för att testa om matrisen är diagonaliserbar.

Hur man diagonaliserar en matris framgår **[här]**. Funktionen ska returnera en lista med de namngivna elementen `P`, `D` och `Pinv`.

Är matrisen inte diagonaliserbar ska funktionen avbrytas med meddelandet ”`Matrix not diagonizable.`”.

Tips! `eigen()`

```
A <- matrix(1:4,ncol=2)
diagonalize_matrix(X = A)

$P
         [,1]     [,2]
[1,] -0.56577 -0.90938
[2,] -0.82456  0.41597

$D
       [,1]     [,2]
[1,] 5.3723  0.00000
[2,] 0.0000 -0.37228

$Pinv
```

```
         [,1]     [,2]
[1,] -0.42223 -0.92305
[2,] -0.83697  0.57428

A <- matrix(c(0,1,0,0),ncol=2)
diagonalize_matrix(X = A)

Matrix not diagonizable.

A <- matrix(c(4,1,-2,1),ncol=2)
diagonalize_matrix(X = A)

$P
        [,1]    [,2]
[1,] 0.89443 0.70711
[2,] 0.44721 0.70711

$D
     [,1] [,2]
[1,]    3    0
[2,]    0    2

$Pinv
        [,1]    [,2]
[1,]  2.2361 -2.2361
[2,] -1.4142  2.8284
```

5.2 my_grouped_test()

Nu ska ni skapa en funktion som ska beräkna gruppvisa konfindesintervall (KI) för en variabel. Innan ni börjar se till att HUS-data är inläst och kör koden nedan för att ta bort de extremt stora värdena:

```
# Download
file_path <-
  "https://raw.githubusercontent.com/MansMeg/KursRprgm/master/Labs/DataFiles/HUS.csv"
HUS <- repmis::source_data(file_path)

Downloading data from: https://raw.githubusercontent.com/MansMeg/KursRprgm/master/Labs/DataFiles/HUS
   SHA-1 hash of the downloaded data file is:
777f771c0493bca3910d619136aed9f4265a2a1d

# Small corrections (removing outliers)
index<-HUS[,1] < quantile(HUS[,1])[4]
HUS<-HUS[index,]
```

Funktion ska heta `my_grouped_test()` och ha argumenten:

- `data_vector` - är en numerisk vektor
- `my_groups` - är en factor/character-vektor som grupperar dataVector
- `alpha` - är signifikansnivån för intervallet, `alpha=0.05` ska ge ett 95 % konfidensintervall.

Funktionen ska returnera en matris `result` där raderna motsvarar grupperna i `my_groups` och har fyra kolumner: Undre gräns för KI, medelvärdet, övre gräns för KI och antal observationer i varje grupp. Se testfallen för namen på kolumnerna. Raderna ska ha samma namn som grupperna `my_groups`.

Förslag till lösning:

1. Se till att my_groups är en faktor. Räkna ut hur många grupper som finns i my_groups. **Tips!** `?levels()` `?table()`

2. Skapa en tom matris av rätt storlek, kalla den `result`. Ge den lämpliga namn. **Tips!** `?colnames()`, `?rownames()`

3. Spara antalet observationer för varje grupp i den fjärde variabeln i `result`.

4. Använd `by()` kombinerat med `t.test()` för att beräkna gruppvisa KI, spara i `group_test`. Testa `?by()`, ni ser att det finns ett argument som heter "`...`" för funktionen `by()`. Dessa tre punkter kan ersättas med argument som behövs till funtionen "`FUN`". Mer tips: `str(''objekt från t.test'')` och `?by` läs under rubriken "value" för att kolla vad `by()` returnerar.

5. Loopa över antalet grupper och välj ut KI och medelvärde för varje grupp från `group_test`. Spara på rätt ställen i `result`.

6. Returnera `result`.

Testa om testfallen nedan fungerar:

```
my_grouped_test(HUS[,1],HUS$Luftkonditionering,0.01)

  Lower CI-limit   Mean Upper CI-limit No of obs.
0         161468 173473         185479         82
1         213182 220556         227930        308

my_grouped_test(HUS[,2],HUS$Pool,0.10)

  Lower CI-limit   Mean Upper CI-limit No of obs.
0         1918.7 1955.5         1992.3        372
1         1890.6 2041.5         2192.4         18

# Chickwts-data
data(chickwts)
my_grouped_test(chickwts[,1],chickwts[,2],0.05)

          Lower CI-limit   Mean Upper CI-limit No of obs.
casein            282.64 323.58         364.52         12
horsebean         132.57 160.20         187.83         10
linseed           185.56 218.75         251.94         12
meatmeal          233.31 276.91         320.51         11
soybean           215.18 246.43         277.68         14
sunflower         297.89 328.92         359.95         12
```

5.3 `sum_of_random_dices()`

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_dices()`. 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_dices()` 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 `dices`.
- 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 `dices` 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:

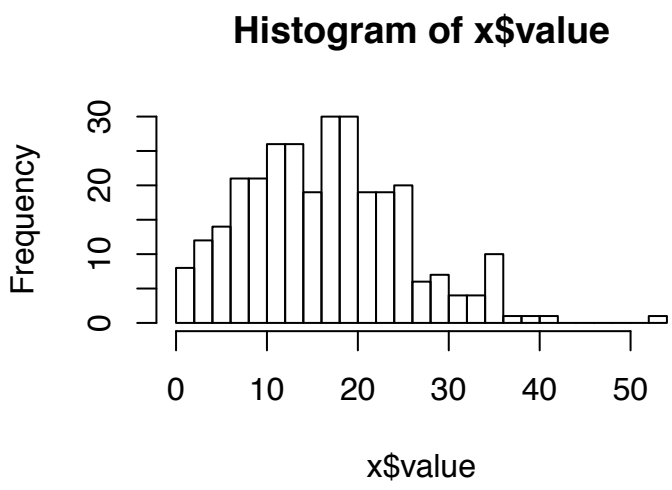
```
sum_of_random_dices(K=5,lambda=3,my_seed=42)

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

sum_of_random_dices(K=5,lambda=8,my_seed=4711)

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

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



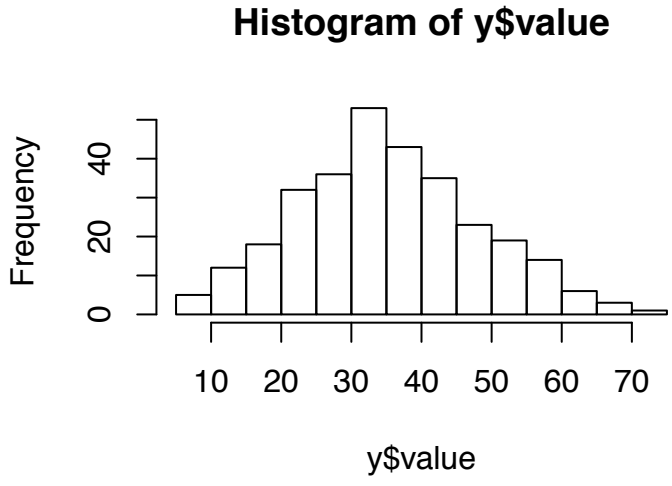

```
mean(x$value)
```

```
[1] 16.893
```

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

```
[1] 8.7547
```

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



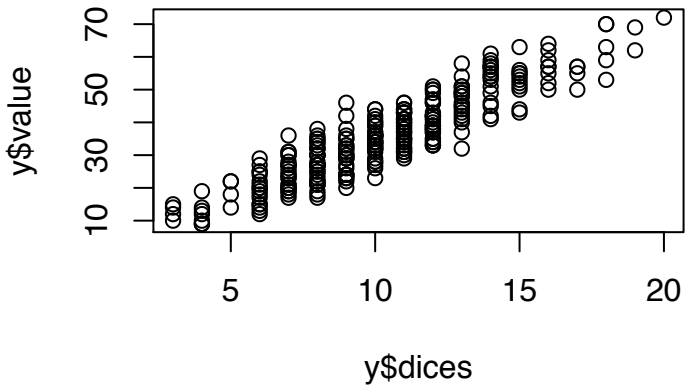

```
mean(y$value)
```

```
[1] 35.813
```

```
sd(y$value)
```

```
[1] 12.852
```

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



5.4 `wordcount()`

Nu är uppgiften att skapa en funktion som ska kunna räkna hur många gånger olika ord förekommer i texten. Funktionen ska heta `wordcount()` och ha argumentet `text` som ska vara en `character`-vektor. Funktionen ska ta en text (i form av en text vektor) och returnera en `data.frame` med två variabler `word` (textvariabel) och `freq` (integervariabel).

I variabeln `word` ska respektive ord ingå, men med små bokstäver, och i variabeln `freq` ska frekvensen av orden framgå. Den `data.frame` som returneras ska vara sorterad efter variabeln `word`. Funktionen ska också skriva ut meningen "`The most common word is '[ord]' and it occured [antal] times.`" med `message()`.

Tips! `table()`

Nedan är ett förslag på hur ni kan implementera funktionen.

1. Läs in paktet i `stringr` i den aktuella R-sessionen.
2. Börja med att sätta ihop de olika textelementen till en textsträng, men denna gång använd mellanslag som avskiljare istället för `\n`.
3. Ta bort punkter och kommatecken i textsträngen.
4. Gör om alla ord till endast gemener.
5. Dela upp teckensträngen med `str_split()` för att få ut respektive ord. [**Tips!** Tänk på att du får ut en lista med denna funktion, inte en vektor. `unlist()` kan då vara till hjälp.]
6. Räkna respektive ord och skapa en data.frame med respektive ord i kolumn 1 och antalet förekomster av detta ord i kolumn 2. Döp kolumn 1 till "`word`", och kolumn 2 till "`freq`".
7. Sortera datasetet efter `word`.
8. Använd `str_c()` och `message()` för att baserat på datsetet ovan skriva ut följande mening "`The most common word is '[ord]' and it occured [antal] times.`".
9. Returnera din `data.frame`.

Kolla om testfallen nedan fungerar:

```
# Laddar ned testdata
library(downloader)
transtrommer_remote <-
  "https://raw.githubusercontent.com/MansMeg/KursRprgm/master/Labs/DataFiles/transtrom.txt"
transtrommer_local <- paste0(getwd(), "/transtrom.txt")
```

```
download(url = transtrommer_remote, destfile = transtrommer_local)

# Test
text<-readLines("transtrom.txt")
worddata<-wordcount(text=text)

The most common word is 'the' and it occured 8 times.

head(worddata)

        word freq
1          a    6
2        and    4
3 approached    1
4         as    1
5     before    1
6      black    1

sorted_worddata <- worddata[order(worddata[,2], decreasing=TRUE),]
head(sorted_worddata)

   word freq
59  the    8
1     a    6
2   and    4
24 have    3
40   of    3
62 they    3

head(wordcount(text=c(rep("a",10), rep("b", 5))))

The most common word is 'a' and it occured 10 times.

  word freq
1    a   10
2    b    5
```

5.5 `henify()`

Med hjälp av funktionerna i `stringr` ska ni skriva en funktion som ska byta ut "han" och "hon" i en text mot "hen". Detta kan göras med hjälp av regular expressions. Det ska gå att göra denna operation med endast ett regular expression (om vi ignorerar versaler i hen). Det är dock inte ett måste att implementera detta med bara en rad kod.

Funktionen ska heta `henify(text)`

- argumentet `text` ska antingen ange sökvägen till den textfil som ska användas som då slutar på "`.txt`" eller vara en teckenvektor innehållande en text.

Arbetsordning:

1. Läs in paktet i stringr i den aktuella R-sessionen.
2. Iakttag följade fall:
   (a) Om `text` är en sökväg till en fil så ska filen med texten läsas in som och sparas i en character-vektor. Detta kan du kolla med om de fyra sista teckenen är "`.txt`".
   Tips! `str_sub()`, `readLines()`. Glöm inte att `.` är ett metatecken.
   (b) Om första elementet i text inte slutar på "`.txt`" så ska text användas som en textvektor i resten av funktionen.

3. Använd nu `str_replace_all()` för att ersätta Han/han och Hon/hon med Hen/hen. Här behöver ni använda regular expressions. Iakttag följande:

   - Tänk på att han/hon ska kunna börja med stor eller liten bokstav.
   - Han/hon kan komma först i strängen, någonstans i mitten, eller sist.
   - Vi ska inte välja ord där han/hon är en del av ordet, tex "storhandla" ska inte bli "storhendla" eller att "hona" ska blir "hena".

4. Returnera den ändrade texten.

Kolla om testfallen nedan fungerar:

```
library(downloader)
transtrommer_remote <-
  "https://raw.githubusercontent.com/MansMeg/KursRprgm/master/Labs/DataFiles/transtrom2.txt"
transtrommer_local <- paste0(getwd(), "/transtrom2.txt")
download(url = transtrommer_remote, destfile = transtrommer_local)

# test 1:
henify(text="transtrom2.txt")

 [1] "N<e4>r hen kom ner p<e5> gatan efter k<e4>rleksm<f6>tet"
 [2] "virvlade sn<f6> i luften."
 [3] "Vintern hade kommit"
 [4] "medan de l<e5>g hos varann."
 [5] "Natten lyste vit."
 [6] "Hen gick fort av gl<e4>dje..."
 [7] "Hela staden sluttade."
 [8] "F<f6>rbipasserande leenden -"
 [9] "alla log bakum uppg<e4>llda kragar."
[10] "Det var fritt!"
[11] "Och alla fr<e5>getecken b<f6>rjade sjunga om Guds tillvaro."
[12] "S<e5> tyckte hen!"

# test 2:
txt1<-c("Hon, hankar handskar, han", "handlar du honorarer?","honartad hona han.")
txt1

[1] "Hon, hankar handskar, han" "handlar du honorarer?"
[3] "honartad hona han."

henify(txt1)

[1] "Hen, hankar handskar, hen" "handlar du honorarer?"
[3] "honartad hona hen."
```

Nu är du klar!