# Datorlaboration 2

Måns Magnusson

29 januari 2014

---

**Instruktioner**

- Denna laboration ska göras **en och en**.
- Det är tillåtet att diskutera med andra, 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
- Innan du lämnar in laborationen:
  1. Starta om R-Studio eller rensa den globala miljön (Global environment) med `rm(list = ls())`.
  2. Ladda in funktionerna i R med `source`.
  3. Kontrollera att inget annat än funktionerna laddas in.
  4. Testa att funktionerna fungerar en sista gång.
- Deadline för labben framgår på **kurshemsidan**

---

# Innehåll

# Del I
# Datorlaboration

## 1 Logik i R

### 1.1 Logiska vektorer

1. Skapa följande logiska vektor i R: `TRUE FALSE FALSE`
2. Använd funktionen `seq()` för att skapa följande sekvenser:
   (a) 10 9 8 7 6 5 4 3
   (b) 3 5 7 9 11 13 15 17
3. Använd funktionen `rep()` för att skapa följande logiska vektorer:
   (a) `TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE`
   (b) `TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE`
4. Använd den logiska vektorn i (3a) för att plocka ut värden ur den numeriska vektor (2a) ovan. Vad får du då för vektor?
   Upprepa samma sak med 3b och 2b.

### 1.2 Logiska operatorer

1. Skapa nu variablerna `a`, `b`, `c` och `d` på följande sätt:

```
> a <- TRUE
> b <- FALSE
> c <- FALSE
> d <- TRUE
```

2. Utryck följande satser och undersök om de är sanna eller falska (försök fundera ut vad det borde bli innan du låter R beräkna svaret):
   (a) `a` och icke `b`
   (b) (`a` eller `c`) och (icke `d` eller `b`)
   (c) (`a` eller icke `b`) och (`a` eller (`b` och `c`) eller (`a` och `b` och `d`))
   (d) `a` exklusivt eller `b`
3. Skapa nu vektorerna `a`, `b`, `c` och `d` på följande sätt:

```
> a <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
> b <- c(FALSE, FALSE, TRUE, TRUE, TRUE)
> c <- c(TRUE, FALSE, FALSE, FALSE, TRUE)
> d <- c(FALSE, FALSE, TRUE, FALSE, FALSE)
```

4. Utryck följande satser och undersök om de är sanna eller falska på samma sätt som i 2a - 2c ovan.

### 1.3 Relationsoperatorer

1. Skapa nu vektorerna `minText`, `minaNummer` och `minBoolean`, där `minBoolean` är vektor `a,b,c` från 3.

```
> minText <- c(rep("John", 5), rep("Frida", 5), rep("Lo", 5))
> minaNummer <- seq(1, 11, length = 15)
> minBoolean <- c(a, b, c)
```

2. Skapa följande logiska vektorer som indikerar när:

   (a) `minaNummer` är större än 3

   (b) `minText` inte är John och `minaNummer` inte har värdet 8

3. Skapa nu en logisk vektor på följande vis:

   (a) När `minText` **inte** är `Frida` **och** `minaNummer` är **större** än medianen av `minaNummer` **och** `minBoolean` är sann.
   [**Tips!** När det är komplicerade logiska uttryck är det enklare att räkna ut dem del för del]
   Rätt svar att jämföra med ges nedan:

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[12] FALSE FALSE FALSE  TRUE
```

# 2 Vektorer

1. Givet vektorn `w <- seq(5,10,1.2)`. Använd `rep()` för att skapa en sekvens där varje element i `w` har upprepats tre gånger som du kallar `ny_w`, alltså:

```
 [1] 5.0 5.0 5.0 6.2 6.2 6.2 7.4 7.4 7.4 8.6 8.6 8.6 9.8 9.8 9.8
```

2. Beräkna `log(ny_w)` och avunda till 2 decimaler. Avrunda även nedåt till närmsta heltal.
   [**Tips!** `?round` och `?floor`]

3. Kontrollera om elementen i sekvensen `ny_w` uppfyller följande villkor:
   (Svaret blir alltså en logisk vektor.)

   (a) Elementen är mindre än 3

   (b) Elementen är inte lika med 5

   (c) Elementen är är större än 3 eller mindre än 2.

4. Använd funktionen `which()` för att ta reda på vilka element i `ny_w` som uppfyller villkoren i **3**.

   (a) Välj ut dessa element med en logisk vektor

   (b) Välj ut dessa element med hjälp av `which()`

5. Skapa följande vektor (**Obs!** du kommer får en varning av R):

```
saknade <- c(log(-1), 3, 0/0, 6, NA, 9)

Warning: NaNs produced
```

6. Använd `is.na()` och `is.nan()` för att undersöka om elementeten inte är tillgängliga (**N**ot **A**vailable) eller ej nummer/ej definierade (**N**ot **a** **N**umber).

7. Använd vektorn i uppgift 5. Om du vill ta bort `NaN`, hur gör du då? Om du vill ta bort `NA`, hur gör du då? [**Tips!** Använd den logiska operatorn för **icke**]

8. Ta bort `NA` och `NaN` i sekvensen, och beräkna medelvärde och varians för de element som är kvar. Nedan är det korrekta resultatet

```
[1] 6
[1] 9
```

## 2.1 Byte av en variabels typ (atomära klass)

1. Skapa sekvensen `x`, `y` och `z` och testa koden nedan. Hur sker omvandligarna mellan variabeltyper?

```
x <- seq(-10, 10)
x
as.logical(x)
y <- c(TRUE, FALSE, TRUE, FALSE)
y
as.character(y)
as.integer(y)
z <- seq(1, 5)
z
z <- as.character(z)
z
z <- c(z, "hejhopp", "TRUE")
z
as.numeric(z)
as.logical(z)
```

## 2.2 Ändra enskilda element i en vektor

1. Skapa vektorn `w <- seq(1, 10, 0.5)`. Detta bör ge följande vektor:

```
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
```

2. Ändra det sjätte elementet i vektorn med `[]` till 7 på följande sätt:

```
w[6] <- 7
```

3. Ändra nu de tre första elementen till 6, 1, 4 på följande sätt:

```
w[1:3] <- c(6, 1, 4)
# Eller w[c(1, 2, 3)] <- c(6, 1, 4)
```

4. Ändra nu det två sista elementet till 12 och 13. Du borde då få följande vektor:

```
 [1]  6.0  1.0  4.0  2.5  3.0  7.0  4.0  4.5  5.0  5.5  6.0 12.0 13.0
```

5. Det går också att använda en logisk vektor för att göra ändringar. Exempelvis kan alla element mindre än 5 ändras till 2.5 på detta sätt:

```
logisk <- w < 3
w[logisk] <- 2
w
```

```
 [1]  6.0  2.0  4.0  2.0  3.0  7.0  4.0  4.5  5.0  5.5  6.0 12.0 13.0
```

6. Vill du ta bort ett eller flera element från en vektor använder du (precis som i föregående laboration) minustecknet och skriver över vektorn.

```
w <- w[-(3:4)]
w
```

```
 [1]  6.0  2.0  3.0  7.0  4.0  4.5  5.0  5.5  6.0 12.0 13.0
```

# 3 Matriser

1. Skapa en matris enligt koden nedan. Studera matrisen, hur ser den ut?

```
x <- c(1, 2, 3, 4, 5, 6)
minMatris <- matrix(x, nrow = 3, ncol = 2)
```

2. Pröva följande operationer:

   (a) Testa att multiplicera alla element med 10. Addera sedan 3 till varje element.

   (b) Dividera elementen med 4

   (c) Beräkna resten (modulo) för elementen i matrisen om matrisen divideras med 2.

3. Nu ska ni skapa en **stor** matris. Skapa vektorerna `a`, `b`, `c` och `d` (se nedan). Sätt sedan samman dessa vektorer med `cbind()` till en matris.

```
a <- rep(c(1, 2, 3, 4, 5), 10)
b <- 1:50
c <- (1:50)^2
d <- log(1:50)
storMat <- cbind(a, b, c, d)
```

4. Pröva att välja ut följande delar ur matrisen med `[ , ]`:

   (a) Elementen (1, 4) och (4, 3)

   (b) Rad 2

   (c) Kolumn 3

5. Pröva att skapa en logisk matris som indikerar alla element som är större än 20. [**Tips!** Använd relationsoperatorer]

6. Pröva att välja ut följande delar ur matrisen på samma sätt som 4 ovan:

   (a) Elementen (4, 4) och (2, 1)

   (b) Rad 1

   (c) Kolumn 4

7. Ändra nu följande enskilda värden, rader och kolumner i `storMat` till 0.
   [**Tips!** Använd slicing och ändra på ett liknande sätt som du gjorde med vektorer i 2.2 på sidan 5.]

   (a) Elementen (4, 4) och (2, 1)

   (b) Rad 1

   (c) Kolumn 4

8. Nu ska vissa värden i matrisen `storMat` ändras. Alla värden som är mindre än 3 ska sättas till 0. Alla värden som är strörre än 45 ska sättas till `NA`.
   [**Tips!** Gör detta i flera steg och använd relationsoperatorer på ett liknande sätt som i 2.2 på sidan 5.]

9. Skapa vektorerna `y` och `z` och matriserna `radMat` och `kolMat` på följande sätt:

```
y <- seq(4, 11)
z <- c(rep(2, 4), rep(9, 4))
radMat <- rbind(y, z)
kolMat <- cbind(y, z)
```

10. Studera dimensionerna på matriserna med funktionen `dim()`. Undersök även längden med `length()`.

11. För att göra om en matris till en vektor används `as.vector()`. Pröva funktionen på `radMat` ovan.

12. För att ta bort rader eller kolumner från en matris används minustecknet.

```
kolMat[-5, ]

      y z
[1,]  4 2
[2,]  5 2
[3,]  6 2
[4,]  7 2
[5,]  9 9
[6,] 10 9
[7,] 11 9
```

13. Om vi tar bort allt så bara en rad- eller kolumnmatris kvarstår reduceras detta till en vektor. Detta kan undvikas med argumentet `drop = FALSE`:

```
kolMat[, -1]

[1] 2 2 2 2 9 9 9 9

kolMat[, -1, drop = FALSE]

     z
[1,] 2
[2,] 2
[3,] 2
[4,] 2
[5,] 9
[6,] 9
[7,] 9
[8,] 9
```

# 4 data.frame

1. Skapa en `data.frame` som du kallar `minVecka` med vektorerna `myWeekdays`, `hours` och `tasks` på följande sätt:

```
study <- c("Monday", "Tuesday", "Wednesday", "Thursday")
job <- "Friday"
fun <- c("Saturday", "Sunday")
myWeekdays <- c(study, job, fun)
hours <- c(rep(4, 4), 6, 0, 0)
tasks <- c(rep("study", 4), "job", rep("fun", 2))
tasks <- factor(tasks)
minVecka <- data.frame(myWeekdays = myWeekdays, hours = hours, tasks = tasks)
minVecka

  myWeekdays hours tasks
1     Monday     4 study
2    Tuesday     4 study
3  Wednesday     4 study
4   Thursday     4 study
5     Friday     6   job
6   Saturday     0   fun
7     Sunday     0   fun
```

2. Studera din data.frame i "Enviroment"-fönstret i R-Studio samt undersök den med:
   (a) Funktionerna `head()` och `tail()`.
   (b) Funktionerna `summary()` och `str()`.
   (c) Funktionerna `dim()`, `ncol()` och `nrow()`.
   (d) Funktionerna `names()`, `colnames()` och `rownames()`.

3. Skapa en ny vektor med kostnaden för veckan. Lägg till `costs` till `minVecka` (se nedan).

```
costs <- c(70, 75, 58, 62, 140, 90, 70)
minVecka$costs <- costs
```

4. Undersök följande om `minVecka`:
   (a) Välj ut de tre sista dagarna i veckan.
   (b) Summera kostnaden för helgen.
   (c) Skapa en logisk vektor för de dagar du studerar eller jobbar mindre än 5 timmar.
   (d) Skapa en logisk vektor för de dagar du spenderar mer än 100 kr.
   (e) Du vill ändra schemat. På tisdag ska du jobba istället för plugga. Gör denna ändring.

5. Du tänker ha samma planering nästa vecka, så kopiera denna vecka och lägg till den efter sista raden `minVecka`. [**Tips!** `rbind()`].

6. Skapa en ny variabel som heter `veckonummer`. Den första veckan ska veckonummer vara 1, och för den andra veckan 2.

7. Du tänker vara lite mer sparsam den **andra** veckan. Ändra värdet på `cost` till det minsta värdet av `cost` från den **första** veckan för hela den **andra** veckan.

8. Att ta bort rader från en data.frame görs på samma sätt som med en matris. Dock kan även enskilda variabler tas bort på följande sätt:

```
minVecka

  myWeekdays hours tasks costs
1     Monday     4 study    70
2    Tuesday     4 study    75
3  Wednesday     4 study    58
4   Thursday     4 study    62
5     Friday     6   job   140
6   Saturday     0   fun    90
7     Sunday     0   fun    70

minVecka <- minVecka[-2, ]
minVecka$costs <- NULL
minVecka

  myWeekdays hours tasks
1     Monday     4 study
3  Wednesday     4 study
4   Thursday     4 study
5     Friday     6   job
6   Saturday     0   fun
7     Sunday     0   fun
```

# 5 Listor

1. Antag att du har vektorer med de veckodagar som du behöver studera, jobba och har fri tid. Samt hur många timmar du ska studera Programmering i R.

```
study <- c("Monday", "Tuesday", "Thursday")
job <- "Friday"
free <- c("Saturday", "Sunday")
study_hours <- c(2, rep(4, 3), 6, 0, 0)
```

2. Skapa en lista med vektorerna `study`, `hours`, `job` och `free` och döp den till `weekPlan`.

```
weekPlan <- list(study, hours, job, free)
```

3. Undersök den lista du skapat med funktionerna `summary()`, `length()` och `str()`.
4. Undersök hur mycket minne listan använder med `object.size()`.
5. Använd funktionen `names()` för att undersöka om elementen i listan har namn. Om inte namge varje element i listan med mostsvarande vektors namn. Spara listan med namn som `weekPlanNamed`.
6. Lägg till textelementet `note` sist i listan `weekPlanNamed` och döp element där note ligger till `myNote`. Här är ett exempel på hur man kan lägga till ett element i en lista.

```
# exempel:
minLista <- list(1:5, "hej")
minLista
```

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

[[2]]
[1] "hej"

minLista[3] <- "mer info"
minLista

[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "hej"

[[3]]
[1] "mer info"
```

7. Välj `job` från `weekPlan` med `[[]]` och `$`
8. Pröva att välja det första och andra listelementet från `weekPlan` med `[]`.
9. Radera note från listan `weekPlan` (Detta kan göras på samma sätt som vi tar bort element från en vektor eller som vi tar bort variabler från en `data.frame`).

# 6 Filhantering, input och output (I/O)

## 6.1 Filhantering

1. Använd `getwd()` för att se vilket som är ditt nuvarande "working directory" på datorn. Spara resultatet (textelement) i variabeln `myOldDir`.
2. Välj en katalog du vill arbeta i (ex. där du lagt labbfilerna), skriv ned sökvägen som ett textelement och spara som `mittWorkingDirectory`.
   [**Tips:** I R (och andra programmeringsspråk som använder regular expressions) har tecknet \ en särskild betydelse, vill du skapa en sökväg behöver du antingen använda `/` eller \\ för att skapa ett vanligt \ i en sökväg - det gäller bara er som har oturen att sitta med en dator med Windows]
3. Använd `setwd()` och `mittWorkingDirectory` för att ändra ditt working directory i R.
4. Använd `getwd()` för att se att sökvägen har ändrats.
5. Studera vilka filer som finns i ditt working directory på hårddisken med `dir()`. Stämmer det?

## 6.2 csv - filer och txt - filer

1. Använd följande kod i R för att läsa in och studera filen "`Apple.txt`" som är inkluderad. Observera att koden nedan kräver att `Apple.txt` ligger i din working directory. Vad betyder `sep=';'` och `header=TRUE` ? [**Tips:** `?read.table`]

```
> # Read data
> apple <- read.table(file = "Apple.txt", sep = ";", header = TRUE)
```

   (a) Studera det data du laddat in på samma sätt som du gjorde i uppgift 2 på s. 8.
   (b) Pröva att ta fram hela sökvägen till `Apple.txt` med funktionen `file.choose()`.

2. Välj ut variabeln `Open` och `Close` (öppnings och stängningspris för Apples aktie) och genomför följande analyser.

(a) Vad är det genomsnittliga öppningspriset?

(b) Vad är det lägsta stängningspriset?

(c) Vad har variabeln `Open` för variabeltyp / atomär klass?

3. Upprepa uppgift 10 till 11 för "`google.csv`" men använd `read.csv()` eller `read.csv2()` istället för `read.table()`. Behövs argumenten `sep=` och `header=`?

4. För att exportera dataset gör man på ett liknande sätt med funktionerna `write.csv()` och `write.table()`. Pröva att spara ned datasetet `apple` som en `.csv`-fil.

```
> write.csv(apple, file = "Apple.csv")
```

## 6.3 Rdata - filer

1. Pröva att spara datasetet `apple` i **R**-format som `Apple.RData` i din working directory.

```
> save(apple, file = "Apple.Rdata")
```

2. Pröva att spara både `apple` och `google` som `storebror.RData`.

3. Använd `save.image()` för att spara ned allt det du har i ditt "Global enviroment" som `alltJagHar.RData`.

4. Använd `dir()` för att se att filerna har sparats korrekt i ditt workspace.

5. Använd `ls()` to för att se vilka variabler som finns i R:s "workspace".

6. Rensa det du har i ditt workspace med rm().
[**Tips!** `rm()` har ett argument, `list=` för att ta bort flera variabler samtidigt. Med detta argument och funktionen `ls()` borde du kunna rensa ditt "Global enviroment" med en eller två rader kod.]

7. Ladda filen `apple.RData` med funktionen `load()`. Vilka objekt har du laddat in i R?

8. Rensa ditt Global enviroment igen med `rm()`. Ladda filen `storebror.RData` med funktionen `load()`. Vilka objekt har du laddat in i R?

9. Rensa ditt Global enviroment igen med `rm()`. Ladda filen `alltJagHar.RData` Vilka objekt har du laddat in i R?

# 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.

Ett förslag på hur du kan angripa problemet är att:

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.

Varje uppgift kan ge flera poäng så även om du inte lyckas med alla delar i en funktion kan du få poäng.

## Uppgift 1: Skottår 5p

Den 29 februari är en skottdag i kalendern och inträffar var fjärde år som exempelvis 2004, 2008, 2012 and 2016. Det vill säga de år som är jämt delbara med fyra. År som är jämt delbara med 100 innehåller inte skottdagar om de inte samtidgt är jämt delbara med 400. Exempelvis innehöll året 1900 inte en skottdag medan år 2000 innehöll en skottdag.

Vi ska skapa en funktion som testar om en vektor av år är skottår eller ej och returnerar detta som en data.frame. Skapa en funktion som heter `leapYear()`. Funktionen ska ha argumentet `years` som ska vara en textvektor.

Exempel på hur du kan implementera funktionen:

1. Konvertera `years` till en numerisk vektor.
2. Använd den numeriska vektorn för att testa om varje givet år är ett skottår. Generera en logisk vektor som är `TRUE` om året är ett skottår.
3. Skapa en data.frame med två variabler, `years` och `leapYear`. Variabeln years ska innehålla de den konverterade numeriska vektorn `years` och `leapYear` ska innehålla en logisk indikator, `TRUE` om året är ett skottår och annars `FALSE`.

Här är testexempel på hur funktionen ska fungera:

```
> my_test_years <- c("1900", "1984", "1997", "2000", "2001")
> myResult <- leapYear(years = my_test_years)
> str(myResult)

'data.frame': 5 obs. of  2 variables:
 $ years   : num  1900 1984 1997 2000 2001
 $ leapYear: logi  FALSE TRUE FALSE TRUE FALSE

> myResult

  years leapYear
1  1900    FALSE
2  1984     TRUE
3  1997    FALSE
4  2000     TRUE
5  2001    FALSE
```

## Uppgift 2: Spåret av en matris 4p

Diagonalmatriser och en matris spår (även kallad **trace**) är viktigt inom den linjära algebran. Mer information finns **här**. Vi ska nu skapa en funktion för att beräkna en matris spår. Funktionen ska kallas `matrixTrace()` och ha argumentet `X`, som är en godtyckligt stor kvadratisk matris. En matris spår är summan av dess diagonalelement:

$$\text{tr}(A) = a_{11} + a_{22} + \ldots + a_{nn} = \sum_{i=1}^{n} a_{ii}$$

Funktionen `matrixTrace()` ska returnera matrisens spår som ett numeriskt värde.
Exempel på hur du kan implementera funktionen:

1. Sätt värdena som inte ligger på diagonalen till 0. [**Tips!** `upper.tri()`, `lower.tri()`]
2. Gör om matrisen till en vektor och summera värdena.

Ett annat sätt är att använda funktionen `diag()`.
Här är testexempel på hur funktionen ska fungera:

```
> A <- matrix(2:5, nrow = 2)
> matrixTrace(X = A)

[1] 7

>
> B <- matrix(1:9, nrow = 3)
> matrixTrace(X = B)

[1] 15
```

## Uppgift 3: Snabbanalys av aktier 6p

Vi ska nu skapa en funktion för att göra en snabb analys av ett dataset som finns sparat som `.csv`. Detta är ett exempel på användningsområde för funktioner i R. Mycket data kommer in löpande och då kan det vara så att vissa standardanalyser vill vi göra snabbt med en förprogrammerad funktion. Vi ska skapa en funktion `fastAnalysis()` med argumentet `file` och `period_length`. Syftet är att snabbt analysera de senaste dagarnas aktiekurs.

Funktionen ska läsa in en `.csv` - fil som anges av argumentet `file` och returnera en lista med de namngivna listelementen `total_spridning`, `medel_slutpris`, `slutpris_upp` och `datum`.

Exempel på hur du kan implementera funktionen:

1. Läs in data (csv) med argumentet file (file innehåller både filnamn och sökväg). Använd funktionen `read.csv()`. [**Tips!** tänk på `stringsAsFactors` innebär att datumvariablerna blir en faktorvariabel].
2. Plocka ut de senaste `period_length` antal dagar från datasetet (anta att de senaste aktiekurserna är högst upp).
3. Räkna ut de värden som ska returneras av funktionen:
   (a) `total_spridning` (ett numeriskt element) är skillnaden mellan det högsta värdet på `High` och det lägsta värdet på `Low` under perioden.
   (b) `medel_slutpris` (ett numeriskt element) är det genomsnittliga slutpriset under perioden.
   (c) `slutpris_upp` (ett logiskt element) är ett logiskt värde som anger `TRUE` om slutpriset första dagen under perioden är lägre än slutpriset den sista dagen under perioden.
   (d) `datum` (vektor med två textelement) ska innehålla det första och det sista datumet för perioden. **OBS!** Detta ska vara en textvektor, inte en faktor.
4. Sätt ihop dessa värden till en namngiven lista med namnen ovan.

Här är testexempel på hur funktionen ska fungera:

```
> appleFil <- "AppleTest.csv"
```

```
> myList1 <- fastAnalysis(file = appleFil, period_length = 5)
> str(myList1)

List of 4
 $ total_spridning: num 11.8
 $ medel_slutpris : num 425
 $ slutpris_upp   : logi FALSE
 $ datum          : chr [1:2] "2012-01-24" "2012-01-18"

> myList2 <- fastAnalysis(file = appleFil, period_length = 10)
> str(myList2)

List of 4
 $ total_spridning: num 12.7
 $ medel_slutpris : num 424
 $ slutpris_upp   : logi FALSE
 $ datum          : chr [1:2] "2012-01-24" "2012-01-10"
```

*Nu är du klar!*