.daddy_code v1.0

Rasmus Hammar

Table of contents

1 Allmänt

Lite allmänna funktioner som är bra att kunna.

1.1 Paket & inläsning av data

För att installera ett paket.

```r
# För att installera paketet som heter palmerpenguins
install.packages("palmerpenguins")

# Stats
install.packages("MKinfer")
install.packages("lmboot")
install.packages("boot.pval")

# Matte
install.packages("pracma")
install.packages("expm")
install.packages("deSolve")

# KvantBio
install.packages("plotrix")
install.packages("scales")
```

För att använda funktioner i ett paket måste man ladda in paketet i R genom `library()`

```r
# Exempel data
library(palmerpenguins)

# Stats
library(MKinfer)
library(lmboot)
library(boot.pval)

# Matte
library(pracma)
library(expm)
library(deSolve)
```

```r
# KvantBio
library(plotrix)
library(scales)
```

För att läsa in data från t.ex. en .csv eller .txt fil. Öppna filen och identifiera vilka symboler som **sep**arerar kolumner samt används som **dec**imaltecken.

```r
file.choose() # Returnerar vägen till filen man väljer, hitta fönstret
 ↪  med alt + tab.

my_data <- read.table(
  file = file.choose(), # Välj fil interaktivt, alt. filens sökväg som
   ↪  en text string
  header = TRUE, # Om det finns namn på kolumnerna
  sep = ",", # Här används ett kommatecken för att skilja på kolumner
  dec = "." # Här används en punkt som decimaltecken
)
```

För att titta på data kan man använda

```r
# View(penguins) # Öppnar som separat flik
head(penguins, # Printar de första n raderna
     n = 3)
```

```
# A tibble: 3 x 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
# i 2 more variables: sex <fct>, year <int>
```

```r
tail(penguins, # Printar de sista n raderna
     n = 3)
```

```
# A tibble: 3 x 8
  species   island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>     <fct>           <dbl>         <dbl>             <int>       <int>
1 Chinstrap Dream            49.6          18.2               193        3775
```

```
2 Chinstrap Dream            50.8          19                 210        4100
3 Chinstrap Dream            50.2          18.7               198        3775
# i 2 more variables: sex <fct>, year <int>
```

För att skapa olika datastrukturer

1.2 Variabel

```
var_1 <- "Text/character string" # Någonting inuti "" är text
var_2 <- 25 # Nummer, för decimaltal 25.99
var_3 <- TRUE # Boolean, kan bara vara TRUE (T) eller FALSE (F)
var_4 <- NA # Not Available (NA), inte samma som "NA" (text string)!
var_5 <- NULL # Ingenting, tenderar att radera saker!
```

1.3 Vektor

```
vec_1 <- c() # Skapar en tom vektor
vec_1 <- c(36, 56, 48) # Skapa en vektor med tre värden
vec_1[4] <- 25 # Sätta in ett värde i index plats 4
vec_1 # Printa innehållet av variablen (vektorn)
```

```
[1] 36 56 48 25
```

```
vec_1[2] # Printa innehållet i index 2
```

```
vec_1[c(1,3)] # Printa index 1 och 3
```

Vektorer kan innehålla namn och indexeras med namn

```
vec_2 <- c(element_1 = 99, element_2 = 100, element_3 = 101)
vec_2["element_4"] <- 102 # Sätt in ett värde med namn
vec_2
```

```
element_1 element_2 element_3 element_4
       99       100       101       102
```

```r
vec_2["element_2"]
```

```r
vec_2[c("element_1", "element_3")]
```

1.4 Data frame

```r
df <- data.frame( # Skapa en dataframe med kolumn_namn = vektor
  vec_1, # Kolumn 1, vec_1 från tidigare
  letters = LETTERS[1:4] # Kolumn 2, lite bokstäver
)
df
```

```
  vec_1 letters
1    36       A
2    56       B
3    48       C
4    25       D
```

Man kan lägga till kolumner med $

```r
df$col_3 <- vec_2 # Kolumn 3
df
```

```
  vec_1 letters col_3
1    36       A    99
2    56       B   100
3    48       C   101
4    25       D   102
```

Andra sätt att lägga till kolumner är med namn

```r
df["col4"] <- letters[1:4]
df
```

```
  vec_1 letters col_3 col4
1    36       A    99    a
```

```
2    56       B   100    b
3    48       C   101    c
4    25       D   102    d
```

En kolumn kan användas med $ vilket ger en vektor (viktig distinktion vid vissa andra tillfällen).

```
df$letters # Ger en vektor, samma som med df[["letters]]
```

Eller med indexering/namn, N.B. enkla `[ ]` ger ett subset, alltså en mindre del av samma objekt typ

```
df[3] # Ger en data frame med enbart kolumn 3
df["col4"] # Ger en data frame med kolumn "col4"
```

medan dubbla `[[ ]]` ger objektet inuti

```
df[[1]] # Ger det "mindre objektet inuti", alltså en vektor i detta fall
```

Enkla `[ ]` kan indexeras med rader och kolumner samtidigt

```
df[2] # Kolumn 2 (en data frame med en kolumn)
df[3,2] # Rad 3, kolumn 2 (en vektor av längd ett)
df[ ,2] # Alla rader, kolumn 2 (en vektor)
df[3, ] # Rad 3, alla kolumner (en data frame med en rad)
```

På tal om subsetting, för att ta ett subset av en data frame kan man filtrera rader baserat på ett kriterium

```
penguins_chin <- penguins[penguins$species == "Chinstrap", ]
# Utan komma läses det som att du vill filtrera kolumner! '==' är det
 ↪  matematiska 'lika med' (eftersom '=' är samma som '<-' )
head(penguins_chin, 3)
```

```
# A tibble: 3 x 8
  species   island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>     <fct>           <dbl>         <dbl>             <int>       <int>
1 Chinstrap Dream            46.5          17.9               192        3500
2 Chinstrap Dream            50            19.5               196        3900
3 Chinstrap Dream            51.3          19.2               193        3650
# i 2 more variables: sex <fct>, year <int>
```

```r
penguins_heavy <- penguins[penguins$body_mass_g > 3500, ]
head(penguins_heavy, 3)
```

```
# A tibble: 3 x 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 <NA>    <NA>                NA            NA                  NA          NA
# i 2 more variables: sex <fct>, year <int>
```

Man kan även göra detta för att få en kolumn filtrerad/subset baserad på en annan kolumn.

```r
penguins_chin_weight <- penguins$body_mass_g[penguins$species ==
 ↪  "Chinstrap", ]
```

```
Error in penguins$body_mass_g[penguins$species == "Chinstrap", ] :    incorrect
number of dimensions
```

Det blir en error därför att nu försöker man ta både rader och kolumner (2 dimensioner) på en kolumn (kolumn/vektor har bara 1 dimension) därför att $ ger en vektor. Om vi istället provar utan extra kommatecken för att ange rader & kolumner samtidigt fungerar det (ger en vektor). Alternativt ange kolumnen med namn istället för med $ (ger en mindre data frame).

```r
# Utan extra kommatecken
penguins_chin_weight <- penguins$body_mass_g[penguins$species ==
 ↪  "Chinstrap"]
head(penguins_chin_weight)
```

```
[1] 3500 3900 3650 3525 3725 3950
```

```r
# Ange kolumnen med namn
penguins_chin_weight <- penguins[penguins$species == "Chinstrap",
 ↪  "body_mass_g"]
head(penguins_chin_weight)
```

```
# A tibble: 6 x 1
  body_mass_g
```

```
        <int>
1        3500
2        3900
3        3650
4        3525
5        3725
6        3950
```

Vi kan även bli av med `NA` på ett kontrollerat sätt med den här metoden. Om man vill se till att vikt och näbblängd inte har `NA` kan man använda funtionen `in.na()` och operatorn `!`.

```
is.na(NA) # Kolla om NA är NA
```

```
[1] TRUE
```

```
!is.na(NA) # ! negerar ett TRUE/FALSE uttryck
```

```
[1] FALSE
```

```
vec_NA <- c(NA, 50)
is.na(vec_NA) # Kolla vilka element som är NA
```

```
[1]  TRUE FALSE
```

```
!is.na(vec_NA)
```

```
[1] FALSE  TRUE
```

```
# Alla rader där kolumnen body_mass_g inte är NA och alla kolumner
penguins_no_NA <- penguins[!is.na(penguins$body_mass_g), ]
# Alla rader där kolumnen bill_length_mm inte är NA och alla kolumner
penguins_no_NA <- penguins_no_NA[!is.na(penguins_no_NA$bill_length_mm),
 ↪  ]

# Tabell med antal rader & kolumner
```

```r
  data.frame(
    rows = c(penguins = nrow(penguins), no_NA = nrow(penguins_no_NA)),
    columns = c(penguins = ncol(penguins), no_NA = ncol(penguins_no_NA))
  )
```

```
         rows columns
penguins  344       8
no_NA     342       8
```

1.5 Matris

Vi kan skapa matriser med `rbind()` (går även att använda `cbind()` men då placeras vektorerna vertikalt från det övre vänstra hörnet, istället för horisontellt). Med `rbind()` blir matrisen som den ser ut när man skriver den.

```r
  rbind( # en "character matrix"
    c("a", "b"),
    c("c", "d")
  )
```

```
     [,1] [,2]
[1,] "a"  "b"
[2,] "c"  "d"
```

```r
  cbind( # skillnaden med cbind()
    c("a", "b"),
    c("c", "d")
  )
```

```
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

```r
  M <- rbind( # Skapar en matris radvis med namngivna rader och kolumner
    r1 = c(col1 = 0.90, col2 = 0, col3 = 0, col4 = 0, col5 = 0.45),
    r2 = c(          0,     0.45,        0,        0,           0),
    r3 = c(          0,     0.23,     0.45,        0,           0),
```

```
  r4 = c(          0,        0,     0.23,     0.45,           0),
  r5 = c(          0,        0,        0,     0.23,        0.20)
)
M
```

```
   col1 col2 col3 col4 col5
r1  0.9 0.00 0.00 0.00 0.45
r2  0.0 0.45 0.00 0.00 0.00
r3  0.0 0.23 0.45 0.00 0.00
r4  0.0 0.00 0.23 0.45 0.00
r5  0.0 0.00 0.00 0.23 0.20
```

För matris matte se Matte sektionen.

Matriser kan indexeras ungefär på samma sätt som data frames.

```
M[3, 2]
M[3:5, 2:3]
M[c(3,5), c(2,3)]
M[c("r3","r5"), c("col2","col3")]
```

1.6 Lista (den riktiga listan)

I en vektor **måste** alla element vara av samma typ (numerisk/text osv.) och i en data frame **måste** alla kolumner (som är vektorer) vara lika långa.

Listor kan ha olika typer av objekt i sig och brukar användas för att bunta ihop olika objekt man vill hålla tillsammans. Till exempel är output från `t.test()` en lista.

```
my_list <- list() # En tom lista
my_list$df_1 <- df # Lägg till en data frame med $
my_list[[2]] <- var_2 # Lägg till en variabel med index, N.B. [[]]
my_list[["plats_3"]] <- vec_2 # Lägg till en vektor med namn, N.B. [[]]
my_list
```

```
$df_1
  vec_1 letters col_3 col4
1    36       A    99    a
2    56       B   100    b
```

```
3    48       C   101    c
4    25       D   102    d

[[2]]
[1] 25

$plats_3
element_1 element_2 element_3 element_4
       99       100       101       102
```

Och man få ut objekten i en lista på samma sätt.

1.7 Lite andra funktioner

Ett objekts typ heter "klass" och kan kollas med funktionen `class()`

```
class(var_1) # var_1 är text
class(var_2) # var_2 är numerisk
class(var_3) # var_3 är logical/boolean (TRUE/FALSE)
class(vec_1) # vec_1 är numerisk vektor
class(df) # df är en data frame
class(M) # M är en matris
class(my_list) # my_list är en lista
```

För att se till att någonting skrivs i konsollfönstret används `print()`

```
print(var_1) # Tar enbart ett objekt! Se paste()
```

```
[1] "Text/character string"
```

För att skriva ihop text m.m. används `paste()`, den tar dock enbart enskilda element (inte vektor/data frame/list

```
message <- paste(
  "Hello", "world", var_1, 55, vec_2[3], # Alla objekt man vill skriva
   ↪ ihop
  sep = " " # Separator att infoga mellan varje objekt, här ett
   ↪ mellanslag
)
print(message)
```

```
[1] "Hello world Text/character string 55 101"
```

Notera att `print()` enbart accepterar ett objekt, därav den vanliga kombinationen `print(paste(arg1, arg2))`.

```r
print(paste("Nu","klarar","vi","tentan!", sep = "! "))
```

```
[1] "Nu! klarar! vi! tentan!"
```

Kolla längden på saker med

```r
length(vec_1) # Längden på en vektor
length(penguins) # Längden på en data frame är antalet kolumner
nrow(penguins) # För antalet rader i en data frame
ncol(penguins) # Antalet columner på en data frame
```

Få en vektor med namnen på kolumner

```r
colnames(penguins)
```

```
[1] "species"           "island"            "bill_length_mm"
[4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
[7] "sex"               "year"
```

Få alla unika element i en vektor

```r
unique(penguins$species)
```

```
[1] Adelie    Gentoo    Chinstrap
Levels: Adelie Chinstrap Gentoo
```

Konvertera/tvinga (eng. coerce) objekt till en viss typ.

```r
as.numeric()    # Bra ifall en kolumn är text men ska vara siffror
as.character()
as.factor()     # Kan vara viktig för ANOVA om grupperna är 1,2,3,4...
```

```r
as.data.frame() # Kan tvinga en matris till data frame
as.matrix()     # Kan tvinga en data frame till matris
```

1.8 Kontrollstrukturer

Kontrollstrukturer är några speciella funktioner som möjliggör mer kontroll över vilka rader med kod som ska köras.

1.8.1 Loop

För att köra några rader kod flera gånger används loopar. Detta är exempel på for-loop.

```r
vector <- c("Nu", "klarar", "vi", "tentan!")
for (variable in vector) {
  print(variable)
}
```

```
[1] "Nu"
[1] "klarar"
[1] "vi"
[1] "tentan!"
```

```r
for (i in 1:4) {
  print(paste(i, ":", vector[i]))
}
```

```
[1] "1 : Nu"
[1] "2 : klarar"
[1] "3 : vi"
[1] "4 : tentan!"
```

```r
n <- c()
n[1] <- 55
for (i in 1:5) {
  n[i+1] <- i
}
```

```
n
```

```
[1] 55  1  2  3  4  5
```

1.8.2 If-else

För att köra vissa rader enbart om ett visst kriterium uppfylls används if-else kombination. Resultatet av att "köra koden" i parantesen ska bli `TRUE` för att köra den följande koden.

```
if ("Tenta!" == "Tenta!") { # Blir TRUE
  print(TRUE)
} else {
  print(FALSE)
}
```

```
[1] TRUE
```

```
if (5^2 < 100) { # Blir TRUE
  print("Smaller")
} else {
  print("Bigger")
}
```

```
[1] "Smaller"
```

```
if (50^2 < 100) { # Blir FALSE
  print("Smaller")
} else {
  print("Bigger")
}
```

```
[1] "Bigger"
```

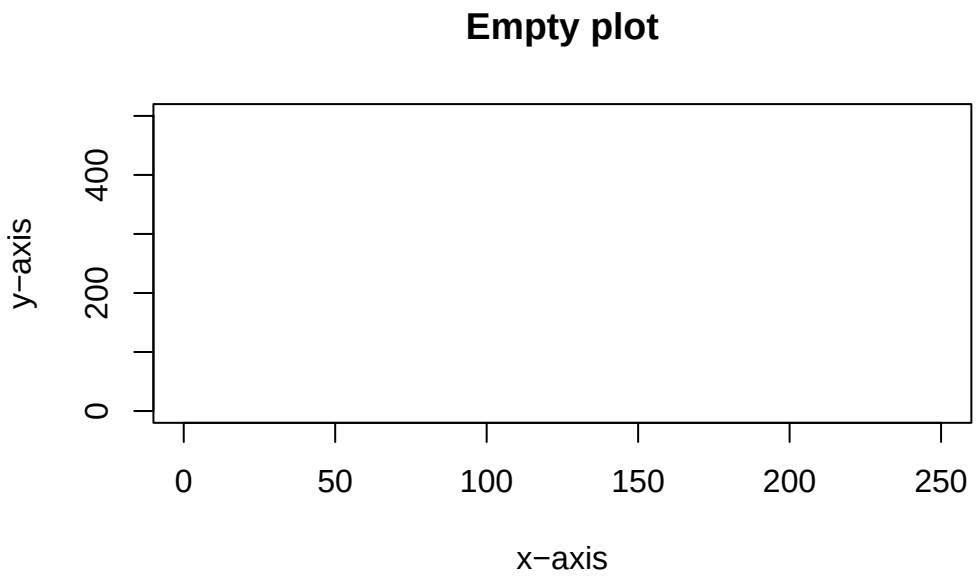
1.9 Plotting

Här följer plotfunktioner med många möjliga argument. Vissa argument visas flera gånger som alternativ.

För att kontrollera hur många plots att visa på samma gång, använd.

1.9.1 Tomt plotfönster

```
plot(
  x = NULL,
  y = NULL,

  xlim = c(0, 250),
  ylim = c(0, 500),

  xlab = "x-axis",
  ylab = "y-axis",
  main = "Empty plot"
)
```



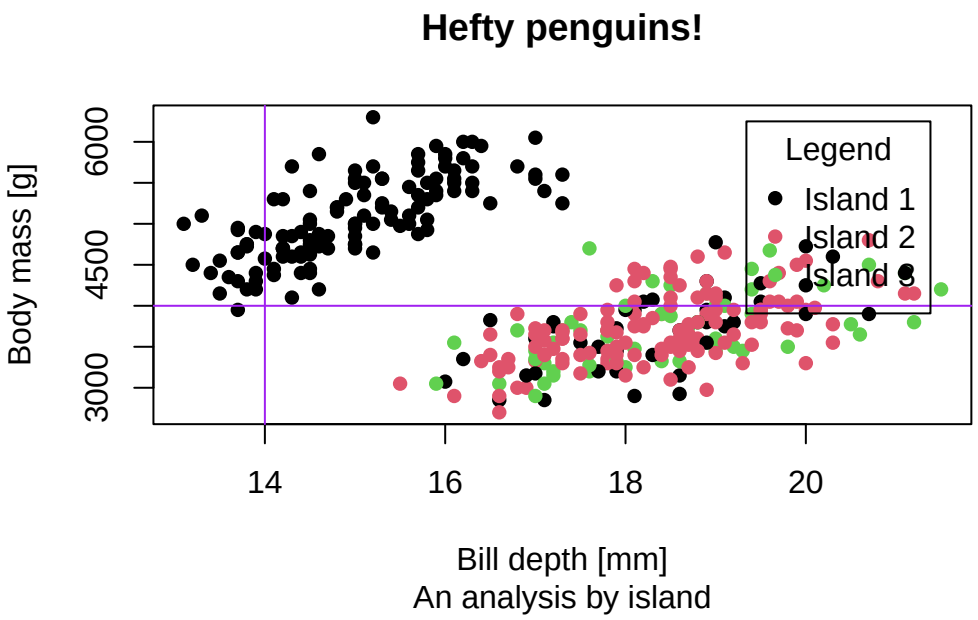
1.9.2 Scatterplot

Om x = kontinuerlig & y = kontinuerlig ger detta ett punktdiagram eller linjediagram.

De flesta argument fungerar likadant/liknande för andra typer av plots.

```
plot(
  # Värden för axlarna
  x = penguins$bill_depth_mm,
  y = penguins$body_mass_g,

  # Annotering
  xlab = "Bill depth [mm]",
  ylab = "Body mass [g]",
  main = "Hefty penguins!",
  sub = "An analysis by island",
#  ann = FALSE, # alt. ta bort alla vanliga annoteringar.

#  xaxt = "n",  # x-axis text = 'n' (none)
#  yaxt = "n",    # Tar bort axel markörerna

  # Linjer & punkter
  type = "p",    # 'l' = line, 's' = stairs, 'p' = point,
                   # 'b' = both (line & pont), 'o' = both (overplotted),
                   # 'h' = histogram liknande, 'n' = none.
  pch = 16,      # 15 = ifylld kvadrat, 16 = ifylld prick,
                   # 17 = ifylld triangel, 18 = ifylld diamant.
  cex = 1,       # Storlek på punkt.

#  lwd = 3,      # Storlek på linje.
#  lty = 1,      # "blank", "solid", "dashed", "dotted",
                   # "dotdash", "longdash", twodash".

# col = "blue",  # Färg
  col = penguins$island, # Färg baserat på faktor
# col = alpha("blue", 0.30) # Färg (blå) & transparens (30%)
)

# Lägg till en linje från 'a' till 'b'.
abline(
#  a = intercept,
#  b = slope,
```

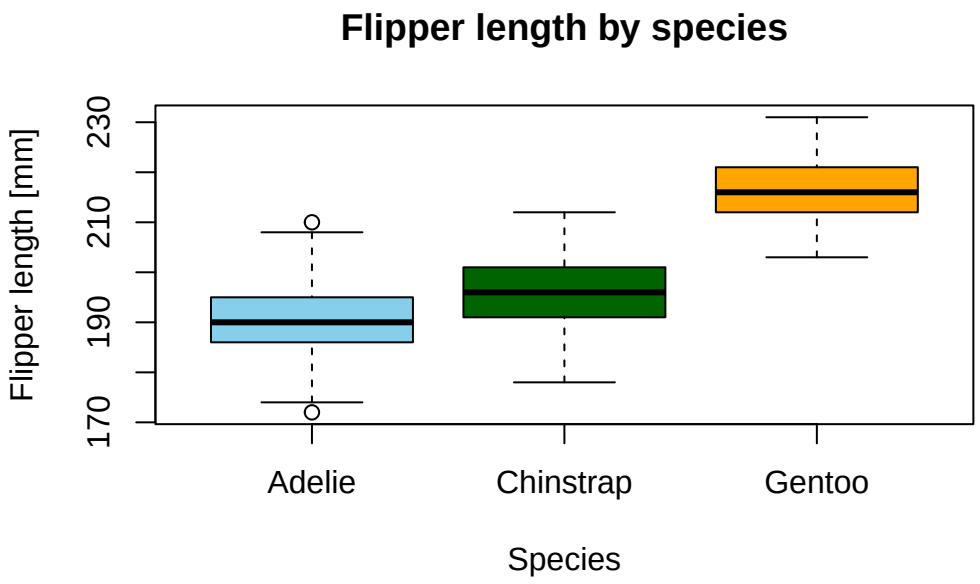
```r
  h = 4000, # y-värde för horizontella linjer
  v = 14,   # x-värde för vertikala linjer
#  reg = lm_objekt, # Regressionsmodell, se sektion 2.4.2.4 Predict
  col = "purple"
)

# För att lägga till en legend
legend(
#  x = 20, y = 6000, # x & y koordinater för legenden.

  x = "topright",     # Kan ges "top", "topleft", "topright",
                        # "bottom", "bottomleft", "bottomright".
  inset = 0.05,       # Avstånd till ytterkanterna som en % av plot
   ↪  storleken.

  legend = c("Island 1", "Island 2", "Island 3"), # Namn i legenden.
  pch = 16,  # Sätt olika punkter med c(15, 16, 17). (Krävs för färger!)
  col = 1:3, # Eftersom faktorn har tre grupper.
               # Kan även vara c("blue","green","orange").

#  lty = c(1, 2, 3), # Sätt olika linje typer,
#  fill = 1:3,       # Färg, men utan att ange form/linje.

  title = "Legend", # Legend title
  bty = "o",        # 'o' = box, 'n' = no box, eller '7', 'L', 'U', 'C'.
)
```



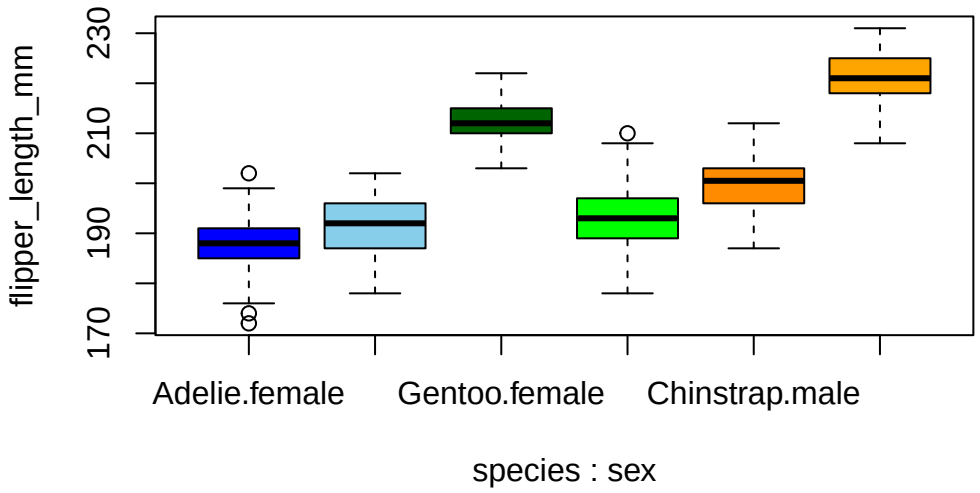
1.9.3 Boxplot

Om x = faktor & y = kontinuerlig ger detta en boxplot/låddiagram.

```
plot(
  x = penguins$species,
  y = penguins$flipper_length_mm,

  xlab = "Species",
  ylab = "Flipper length [mm]",
  main = "Flipper length by species",

  col = c("skyblue", "darkgreen", "orange")
)
```



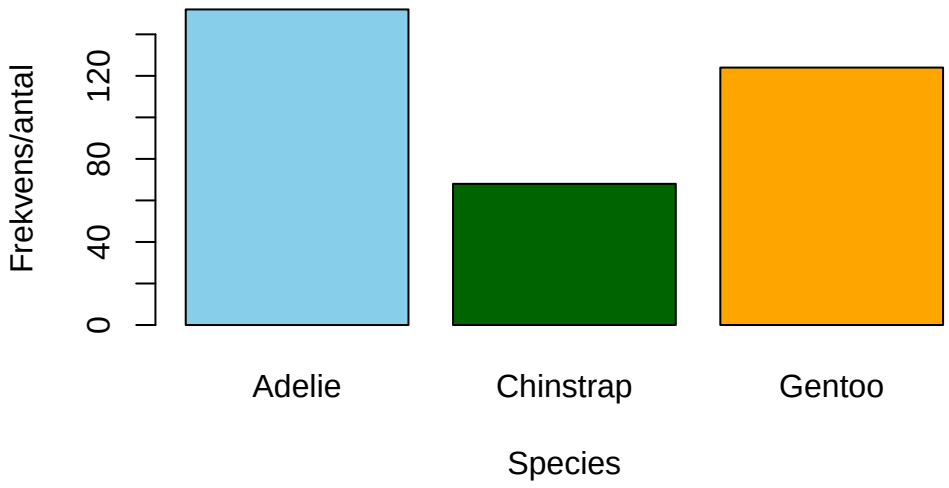

```
boxplot(
  flipper_length_mm ~ species * sex,
  data = penguins,
  col = c("blue","skyblue",
          "darkgreen","green",
          "darkorange","orange")
)
```



1.9.4 Barplot

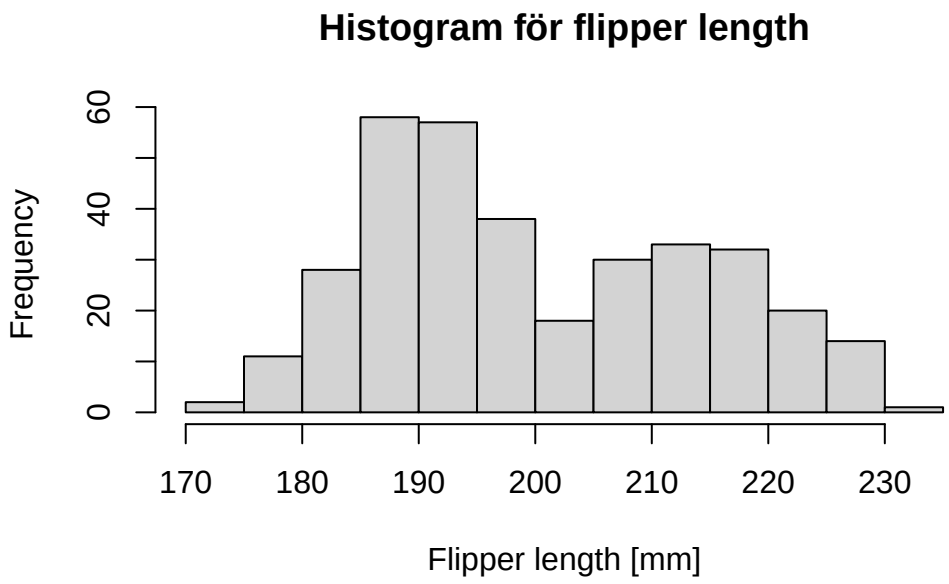
Om x = faktor & inget y ger detta en barplot/stapeldiagram.

```
plot(
  x = penguins$species,

  xlab = "Species",
  ylab = "Frekvens/antal",

  col = c("skyblue", "darkgreen", "orange")
)
```



1.9.5 Histogram

```
hist(
  penguins$flipper_length_mm,
  main = "Histogram för flipper length",
  xlab = "Flipper length [mm]",

#  breaks = 99,    # Hur många staplar
#  col = "blue",   # Färg
)
```

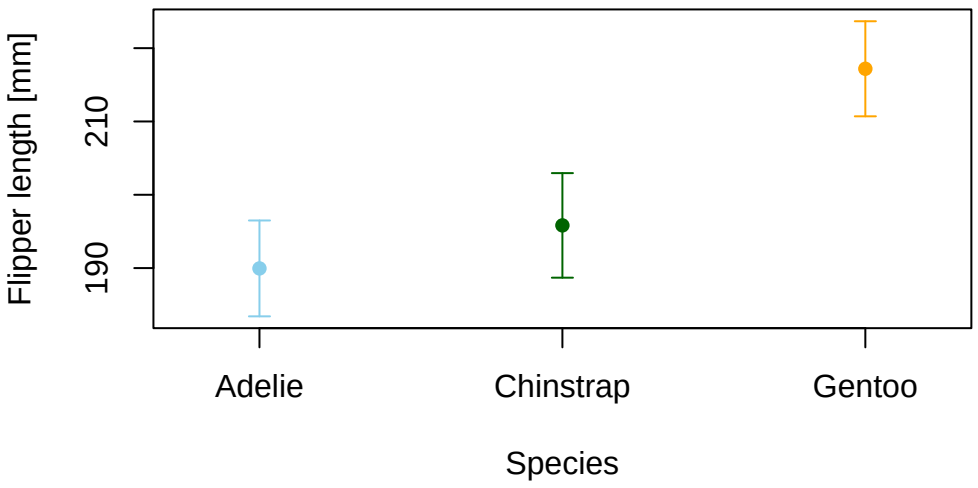


1.9.6 Error plot med `plotCI()`

Används för att illustrera medelvärdet (en punkt) ± standardavvikelse eller standard error of the mean (SEM, se 2.1 Enkel statistik).

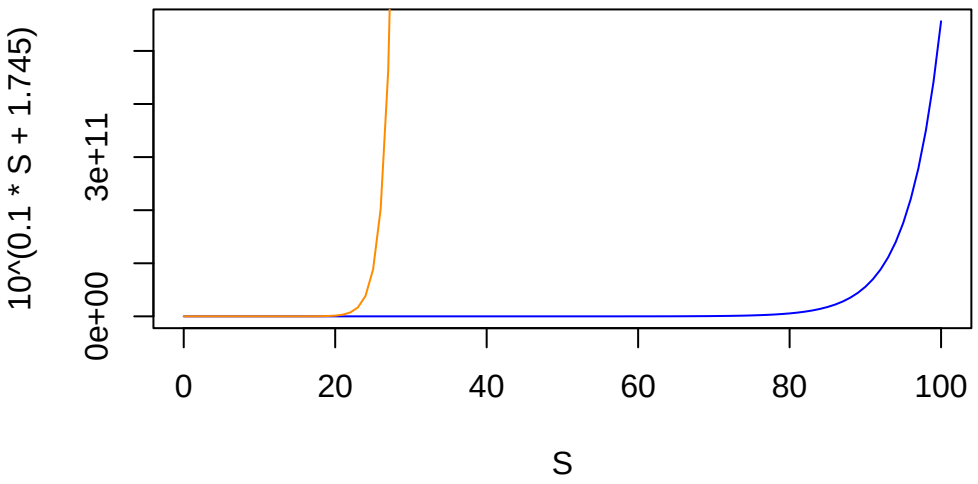
```
# Data för att plotta
peng_mean <- aggregate(
  flipper_length_mm ~ species,
  data = penguins,
  FUN = mean
)
peng_sd <- aggregate(
  flipper_length_mm ~ species,
  data = penguins,
  FUN = sd
)

# Plot
plotCI(
  x = c(2,3,4), # x-koordinater att placera punkter på
```

```
  y = peng_mean$flipper_length_mm, # y-koordinater att placera punkter
   ↪ på

  xlab = "Species",
  ylab = "Flipper length [mm]",

  # Upper interval, alltså övre gränsen för sträcken, är lika med
    # medelvärdet + standardavvikelsen
  ui = peng_mean$flipper_length_mm + peng_sd$flipper_length_mm,
  # Lower interval, alltså den nedre gränsen, är lika med
    # medelvärdet - standardavvikelsen
  li = peng_mean$flipper_length_mm - peng_sd$flipper_length_mm,

  xlim = c(1.75,4.25), # För att ge lite extra plats på sidorna

  xaxt = "n", # Ta bort siffrorna på x-axeln

  pch = 16,
  col = c("skyblue", "darkgreen", "orange")
)
axis(
  side = 1, # vilken sida/axel att lägga till (1 = botten, 2 = vänster,
   ↪ osv.)
  at = c(2,3,4), # Vid vilka x-koordinater att placera text
  labels = peng_mean$species # Lägg till arterna på x-axeln
)
```



1.9.7 Plot av funktion

```r
# Definiera en funktion som en funktion med bara ett argument.
curve_fun <- function(x) {
  10^(0.36*x+1.945)
}

curve(
  10^(0.10*S+1.745), # Matematiska funktionen att rita
  xname="S",         # Vad 'x' heter om man ger den matematiska
   ↪  funktionen direkt
  xlim=c(0,100),     # Intervallet
  col = "blue"
)
curve(
  curve_fun,         # Namn på funktionen med funktionen
  xlim = c(0,100),
  col = "darkorange",
  add = TRUE         # Lägg till linjen i existerande graf
)
```


10^(0.1 * S + 1.745)
0e+00
3e+11
0
20
40
60
80
100
S

2 Stats

2.0.1 Tolkning av p-värde

Drivna över bristningsgränsen av kursen har en student blivit galen och vandrar, med slö blick, vilset i stadsskogen. Studenten betraktar ett träd. Hur högt är trädet? Vilsen betraktar studenten ett annat träd och stapplar vidare. Vad är sannolikheten att genomsnittshöjden på de träd studenten slumpmässigt observerar skiljer sig från genomsnittshöjden på alla träd i stadsskogen? Kanske de skiljer sig från träden i fjällen?

- $\mathbf{H_0}$ = Ingen skillnad mellan medelvärden
- $\mathbf{H_1}$ = Skillnad mellan medelvärden

p-värdet är sannolikheten att en stackars vilsen student slumpmässigt skulle betrakta dessa träd och konstatera att deras höjd skiljer sig från resten av skogens (alltså acceptera $\mathbf{H_1}$), när det i verkligheten inte finns någon skillnad mellan observerade och skogens medelvärden (alltså om $\mathbf{H_0}$ är sann).

- Om p-värde > alfa, alltså p > 0.05, säger vi att det inte finns en skillnad (acceptera $\mathbf{H_0}$).
- Om p-värde < alfa, alltså p < 0.05, säger vi att det finns en skillnad (acceptera $\mathbf{H_1}$).

2.0.2 Rapportera test

Rapportera alltid testets namn, statistiken (t.ex. t-/F-värdet), frihetsgrader och p-värde, samt en mening som ger slutsatsen/tolkningen av testet. Andra relevanta saker kan också inkluderas, såsom konfidens intervall.

Exempel: Förklarande mening ([testets namn], t([frihetsgrader]) = [värdet], p = [värdet]).

> Det fanns en skillnad mellan Grupp 1 och den förväntade vikten (mu = 3000 g) enligt internet (One sample t-test, t(150) = 18.776, p < 2.2e-16).

Tips från coachen, gör detta snyggare än jag…

2.0.3 Formel notation

Många av funktionerna för de statistiska testen kan, eller måste, ges en formel. Formler har följande komponenter:

- *Respons*, en kontinuerlig variabel vi vill förutspå, värdet på y-axeln
- *Faktor*, en kategorisk variabel vilken delar in *Respons* i grupper (för regression är detta en kontinuerlig variabel som direkt orsakar *Respons*)

- '~', skiljer *Respons* från *Faktor*, alltså ger Vänster- och Högerled (`Respons ~ Faktor`)
- '+', adderar en ytterligare *Faktor* (`Respons ~ Faktor_1 + Faktor_2`)
- '*', adderar en ytterligare *Faktor* **och** interaktionen (`Respons ~ Faktor_1 * Faktor_2`)
- ':', adderar enbart interaktionen (`Respons ~ Faktor_1:Faktor_2`)

2.0.4 Beslutsschema

Beslutsträd statistik

Blå box – gå vidare i trädet

Grön box – gå vidare till ANOVOR

Lila box – gå vidare till analys av samband

Röd box – du har nått ditt test

Svart box – testet tas inte upp på denna kurs

Figure 1: Stats flowchart 1

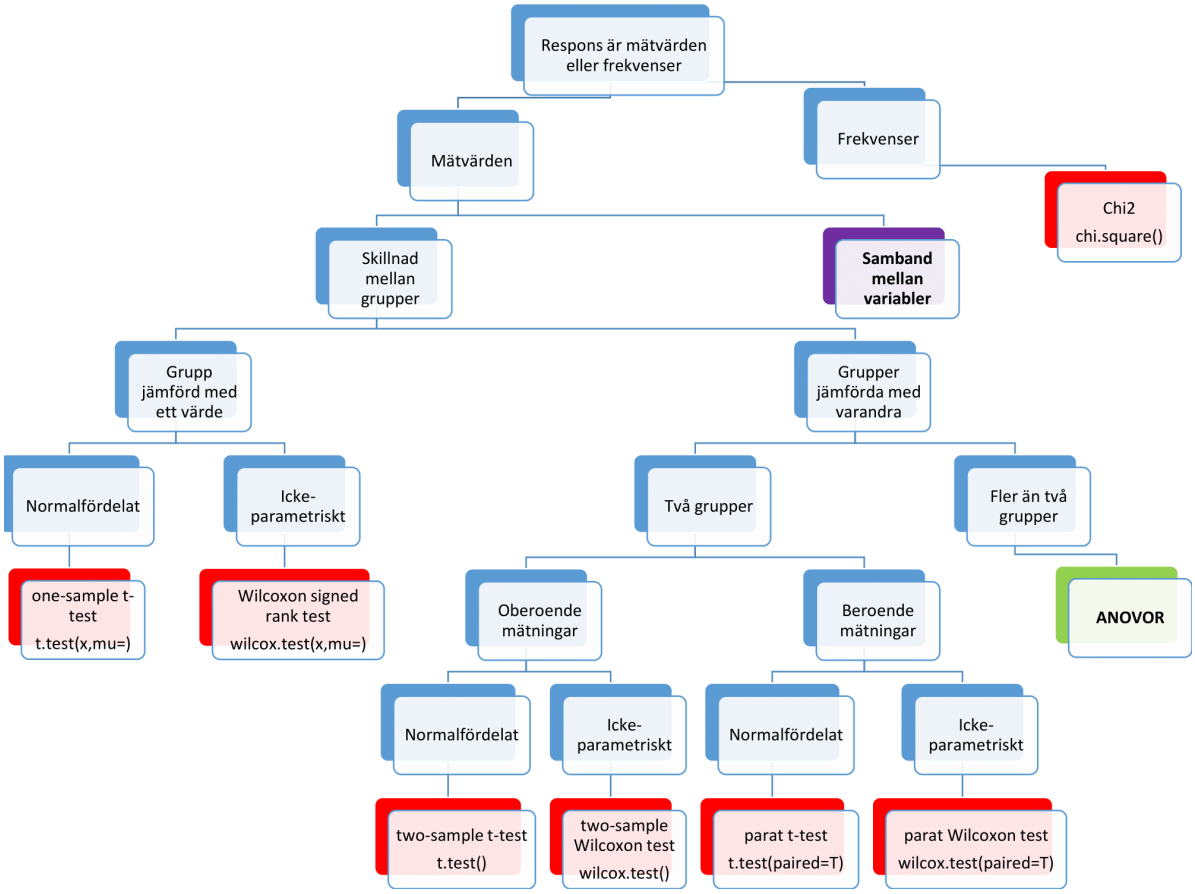


Figure 2: Stats flowchart 2

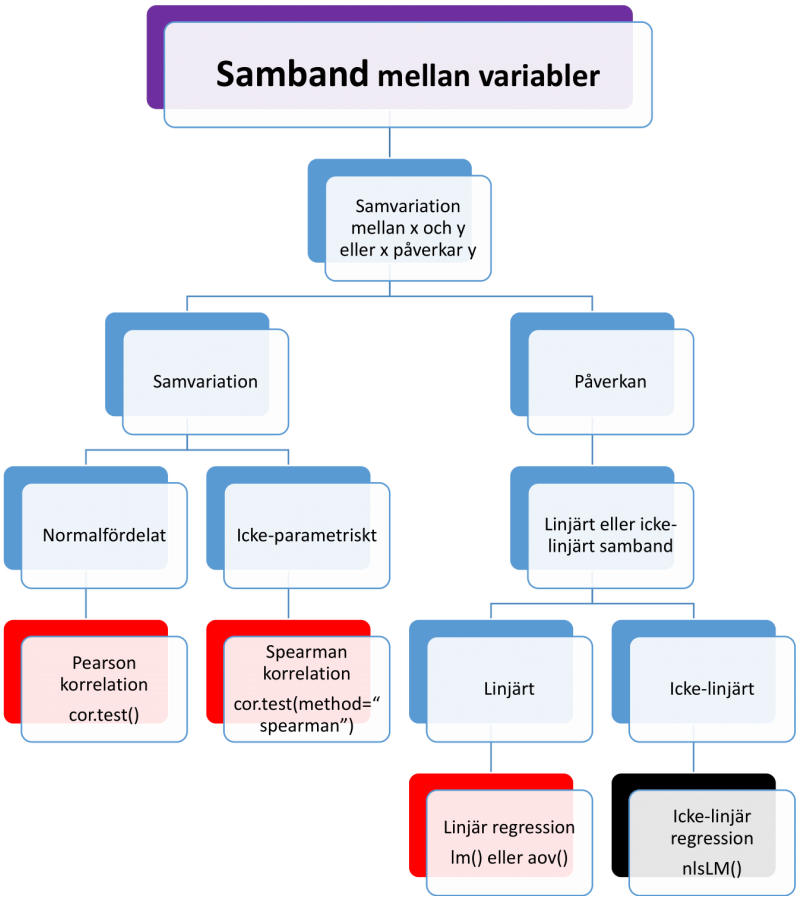


Figure 3: Stats flowchart 3

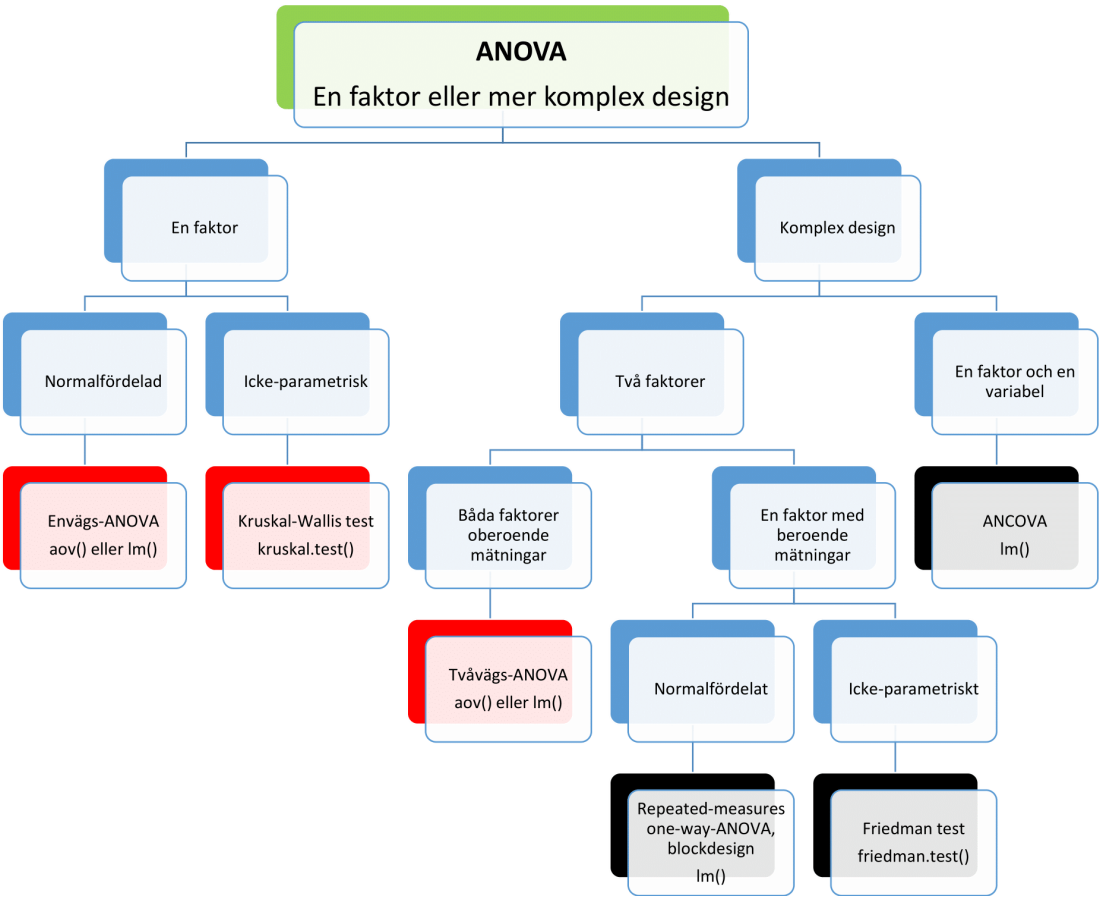


Figure 4: Stats flowchart 4

2.1 Enkel statistik

Några enkla/grundläggande statistik funktioner.

```r
mean(penguins$bill_length_mm, na.rm = TRUE)    # Medelvärde
```

```
[1] 43.92193
```

```r
median(penguins$bill_length_mm, na.rm = TRUE) # Median
```

```
[1] 44.45
```

```r
sum(penguins$bill_length_mm, na.rm = TRUE)     # Summa av vektor
```

```
[1] 15021.3
```

```r
sd(penguins$bill_length_mm, na.rm = TRUE)      # Standardavvikelse
```

```
[1] 5.459584
```

```r
# Funktioner som ger vektorer med svar. Fungerar på både matriser & data
 ↪  frames!
# N.B. stor bokstav i mitten! Finns andra funktioner med liknande namn.
rowSums(M)  # ger en (kolumn) vektor med summan för varje rad.
```

```
  r1   r2   r3   r4   r5
1.35 0.45 0.68 0.68 0.43
```

```r
colSums(M)  # ger en (rad) vektor med summan för varje kolumn.
```

```
col1 col2 col3 col4 col5
0.90 0.68 0.68 0.68 0.65
```

```r
rowMeans(M) # Samma, men för medelvärde.
```

```
   r1    r2    r3    r4    r5
0.270 0.090 0.136 0.136 0.086
```

```r
colMeans(M) # Samma, men för medelvärde.
```

```
 col1  col2  col3  col4  col5
0.180 0.136 0.136 0.136 0.130
```

```r
# Aggregate är lite speciell. Den grupperar rader efter en faktor och
 ↪  applicerar en annan funktion på varje grupp.
aggregate(
  bill_length_mm ~ island,
  data = penguins,
  FUN = mean, # Använd 'mean' eller 'median'
)
```

```
     island bill_length_mm
1    Biscoe       45.25749
2     Dream       44.16774
3 Torgersen       38.95098
```

Standard error of the mean (SEM) finns det ingen egen funktion för, men vi kan skapa en. Formeln är $SEM = \frac{Sd}{\sqrt{n}}$, där Sd = standardavvikelse och n = antal observationer. SEM beskriver hur exakt/pålitligt medelvärdet är om det används som det sanna medelvärdet för populationen.

```r
SEM <- function(x, na.rm = FALSE) {
  # Om argumentet na.rm = TRUE, ta bort NA från x
  if (na.rm) {
    x <- na.omit(x)
  }

  # Beräkna SEM
  SEM <- sd(x)/sqrt(length(x))
  return(SEM)
```

```r
}

SEM(penguins$bill_length_mm, na.rm = TRUE)
```

```
[1] 0.2952205
```

2.2 Transformation

Om data inte ser nice ut, prova att transformera data (för paired/beroende transformera differensen). För denna kurs prova logaritmering.

```r
log()   # Basen e
log10() # Basen 10
```

2.3 Sannolikhet

2.3.1 Slumpade tal

Slumpa ett tal från normalfördelning. Ungefär 96 % av alla observationer i en normalfördelad population ligger inom $\pm$ två standardavvikelser.

```r
rnorm(
  n = 5,    # Antal tal
  mean = 0, # Väntevärde för fördelningen
  sd = 1    # Standard avvikelse
)
```

```
[1]  0.91981974  0.43820388 -0.09624338  0.65853491 -1.69476787
```

Slumpade tal från binomialfördelning. Ger n tal där varje tal är hur många gynnsamma utfall det blev, baserat på totalt antal försök och sannolikheten.

```r
rbinom(
  n = 5,     # Antal tal
  size = 10, # Totalt antal försök
  prob = 0.5 # Sannolikhet för gynnsamt utfall
)
```

```
[1] 8 4 5 4 6
```

Slumpade tal från en uniform fördelning (samma sannolikhet för för varje tal).

```
runif(
  n = 5,   # Antal tal
  min = 0, # Nedre intervall
  max = 10 # Övre intervall
)
```

```
[1] 9.5694051 6.4465251 0.4009026 4.8515904 5.3912065
```

2.3.2 Binomial fördelning

Min binomiala studie:

- 20 försök
- varje försök är Ja/Nej
- 37 % sannolikhet för Ja för varje oberoende försök

För utfall med binomial fördelning används `dbinom()` för att beräkna sannolikheten för exakt x antal gynsamma utfall.

```
f5 <- dbinom(
  x = 5, # Exakt antal 'Ja'
  size = 20, # Totalt antal försök
  prob = 0.37 # Sannolikheten för försöket
)
print(paste("Sannolikhet att exakt fem av 20 försök blir 'Ja':", f5))
```

```
[1] "Sannolikhet att exakt fem av 20 försök blir 'Ja': 0.10508981363014"
```

Dessa sannolikheter kan adderas.

```
f6 <- dbinom(
  x = 6, # Exakt 6st Ja
  size = 20,
  prob = 0.37
```

```
)
print(paste("p att exakt 5-6st av 20 försök blir 'Ja':", (f5+f6)))
```

```
[1] "p att exakt 5-6st av 20 försök blir 'Ja': 0.259388349515663"
```

eller `1 - pbinom()` för att beräkna sannolikheten för alla utfall **över** det man anger.

```
f15_20 <- (1 - pbinom(
  q = 14, # utfall över det angivna
  size = 20,
  prob = 0.37
))
print(paste("p att få 15 eller fler 'Ja':",f15_20))
```

```
[1] "p att få 15 eller fler 'Ja': 0.0006215957873217 94"
```

Enbart `pbinom()` ger den motsatta sannolikheten (alltså 0-14 Ja) vilket är kompliment till 15-20 Ja. Använd `lower.tail = FALSE` för att få 15-20 Ja istället.

```
f15_20_igen <- pbinom(
  q = 14,
  size = 20,
  prob = 0.37,
  lower.tail = FALSE # Istället för 1 - pbinom()
)
print(paste("p att få 15 eller fler 'Ja':",f15_20_igen))
```

```
[1] "p att få 15 eller fler 'Ja': 0.0006215957873218 03"
```

2.4 Skillnad mellan grupper

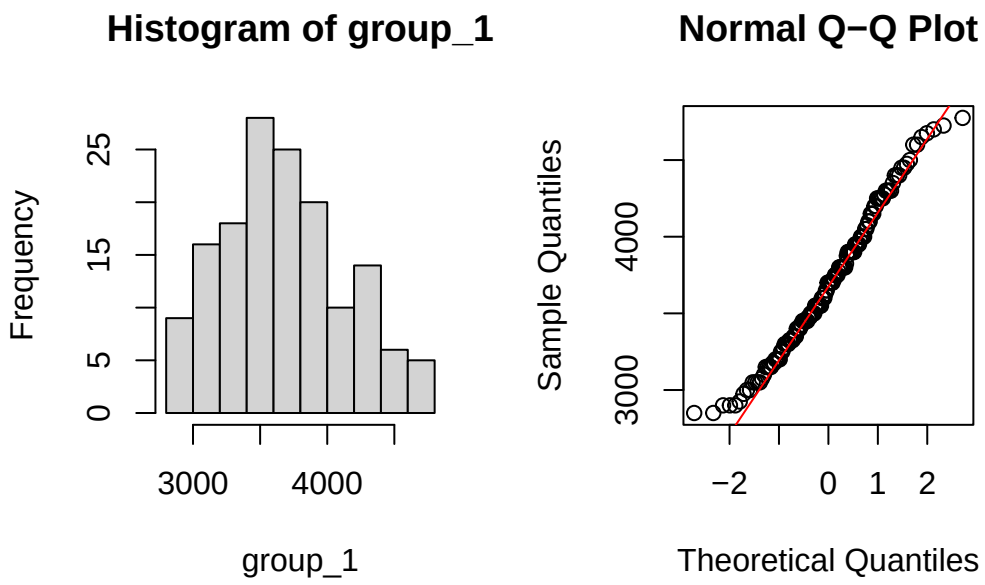
2.4.1 T-test

2.4.1.1 Krav/antaganden

- Data på intervall eller kvotskala (kontinuerlig)
- Normalfördelad (parametrisk) inom varje grupp

- Lika varians (homogen varians, ej för Welch t-test)
- Inga outliers

```r
group_1 <- penguins$body_mass_g[penguins$species == "Adelie"]

par(mfrow = c(1,2)) # Ändrar hur många "rutor/celler" för
# plots det finns enligt rad x column. Bra vana att alltid
# ändra tillbaka till standard c(1,1) efteråt.

hist(group_1)

qqnorm(group_1)
qqline(group_1, col = "red")
```




```r
par(mfrow = c(1,1)) # Ändra tillbaka inställningarna.
```

2.4.1.2 Ensidigt

För att testa om en grupp med kontinuerlig data skiljer sig från ett känt eller teoretiskt väntevärde.

```r
grupp_1 <- penguins$body_mass_g[penguins$species == "Adelie"]

t.test(
  x = grupp_1, # En vektor
  mu = 3000    # Väntevärdet vi hittade på internet
)
```

```
    One Sample t-test

data:  grupp_1
t = 18.776, df = 150, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 3000
95 percent confidence interval:
 3626.926 3774.398
sample estimates:
mean of x
 3700.662
```

Resultat:

Det fanns en skillnad mellan Grupp 1 och den förväntade vikten enligt internet (One sample t-test, $t(150) = 18.776$, $p < 2.2e\text{-}16$).

2.4.1.3 Tvåsidigt

För att testa om det finns en skillnad mellan två grupper av kontinuerlig data.

```r
grupp_1 <- penguins$body_mass_g[penguins$species == "Adelie"]
grupp_2 <- penguins$body_mass_g[penguins$species == "Gentoo"]

t.test(
  x = grupp_1, # En vektor
  y = grupp_2, # En annan vektor
)
```

```
    Welch Two Sample t-test

data:  grupp_1 and grupp_2
t = -23.386, df = 249.64, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
 -1491.183 -1259.525
sample estimates:
mean of x mean of y
 3700.662  5076.016
```

Alternativt sätt att skriva koden är med formel.

```r
peng_groups <- penguins[ which(penguins$species == "Adelie" |
 ↪  penguins$species == "Gentoo"), ]

t.test( # Respons = mätdata & Faktor = två olika arter (alltså två
 ↪  grupper)
  body_mass_g ~ species,
  data = peng_groups
)
```

```
    Welch Two Sample t-test

data:  body_mass_g by species
t = -23.386, df = 249.64, p-value < 2.2e-16
alternative hypothesis: true difference in means between group Adelie and group Gentoo is not
95 percent confidence interval:
 -1491.183 -1259.525
sample estimates:
mean in group Adelie mean in group Gentoo
            3700.662             5076.016
```

Resultat:

Det fanns en skillnad mellan Grupp 1 och Grupp 2 (Welch two sample t-test, t(249.64) = -23.386, p < 2.2e-16).

2.4.1.4 Paired/beroende

First rule of Crayfish Fight Club is you do not talk about Crayfish Fight Club.

Vi studerar kräftor och vill undersöka vilken effekt storleken på klorna har på vilken kräfta som vinner strider om resurser. För att göra detta snorklar vi någonstans tropiskt och letar efter kräftor som slåss. Efter några månader "hårt arbete" har vi samlat ihop följande data. Varje rad är storleken på klorna i cm med vinnare och förlorare i respektive kolumn.

```r
crayfish_fight_club <- data.frame(
  winner = c(86, 84, 75, 93, 102, 87, 88, 91, 87, 74),
  loser = c( 70, 79, 68, 85,  90, 89, 91, 82, 80, 69)
)
head(crayfish_fight_club)
```

```
  winner loser
1     86    70
2     84    79
3     75    68
4     93    85
5    102    90
6     87    89
```

Alltså består varje rad av mätningar på två individer där grupperingen är **beroende** på vem de jämförs med, och därför gör vi ett parat t-test. (Det känns inte rimligt att jämföra vinnaren på rad tre (75 cm) med förloraren på rad fyra (85 cm) eftersom de inte stred med varandra, vilket är vad vi hade gjort om vi behandlat data som oberoende, alltså `paired = FALSE`.) Notera att för parad/beroende data måste **differensen** vara normalfördelad.

```r
t.test(
  crayfish_fight_club$winner,
  crayfish_fight_club$loser,
  paired = TRUE
)
```

```
    Paired t-test

data:  crayfish_fight_club$winner and crayfish_fight_club$loser
t = 3.5266, df = 9, p-value = 0.006448
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
  2.29474 10.50526
sample estimates:
mean difference
            6.4
```

Resultat:

Paired t-test, $t(9) = 3.5266$, $p = 0.006448$, Signifikant.

2.4.1.5 Wilcox-test (icke parametriskt)

För när data inte uppfyller krav för t-test eller vi vill titta på skillnader i median.

```r
wilcox.test( # Motsvarar vanligt t-test för två grupper
  grupp_1,
  grupp_2
)

wilcox.test( # Med formel
  body_mass_g ~ species,
  data = peng_groups
)
```

```r
wilcox.test( # Motsvarar parat t-test för beroende data
  crayfish_fight_club$winner,
  crayfish_fight_club$loser,
  paired = TRUE
)
```

```
Warning in wilcox.test.default(crayfish_fight_club$winner,
crayfish_fight_club$loser, : cannot compute exact p-value with ties

    Wilcoxon signed rank test with continuity correction

data:  crayfish_fight_club$winner and crayfish_fight_club$loser
V = 52, p-value = 0.01431
alternative hypothesis: true location shift is not equal to 0
```

Resultat:

Ingen skillnad.

För denna kurs, ignorera `Warning: cannot compute exact p-value with ties`. Det betyder att det finns två av samma mätvärde vilket inte går att rangordna. (Om data är `c(25, 26, 26, 27)` är det `(1,2,3,4)` eller `(1,3,2,4)`?)

2.4.1.6 Bootstrap (icke parametriskt)

För när data inte uppfyller krav för t-test men vi behöver ett konfidensintervall. Resurskrävande, alltså om du har mycket data tar det längre tid att köra funktionen. Ekvivalent med Wilcoxon men ger konfidensintervall.

```r
boot.t.test( # Med två vektorer
  grupp_1,
  grupp_2
)
```

```
    Bootstrap Welch Two Sample t-test

data:  grupp_1 and grupp_2
bootstrap p-value < 2.2e-16
bootstrap difference of means (SE) = -1375.749 (58.54961)
95 percent bootstrap percentile confidence interval:
 -1490.979 -1260.480

Results without bootstrap:
t = -23.386, df = 249.64, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1491.183 -1259.525
sample estimates:
mean of x mean of y
 3700.662  5076.016
```

```r
boot.t.test( # Med formel
  body_mass_g ~ species,
  data = peng_groups
)
boot.t.test( # Paired/beroende data
  crayfish_fight_club$winner,
  crayfish_fight_club$loser,
  paired = TRUE
)
```

2.4.2 ANOVA

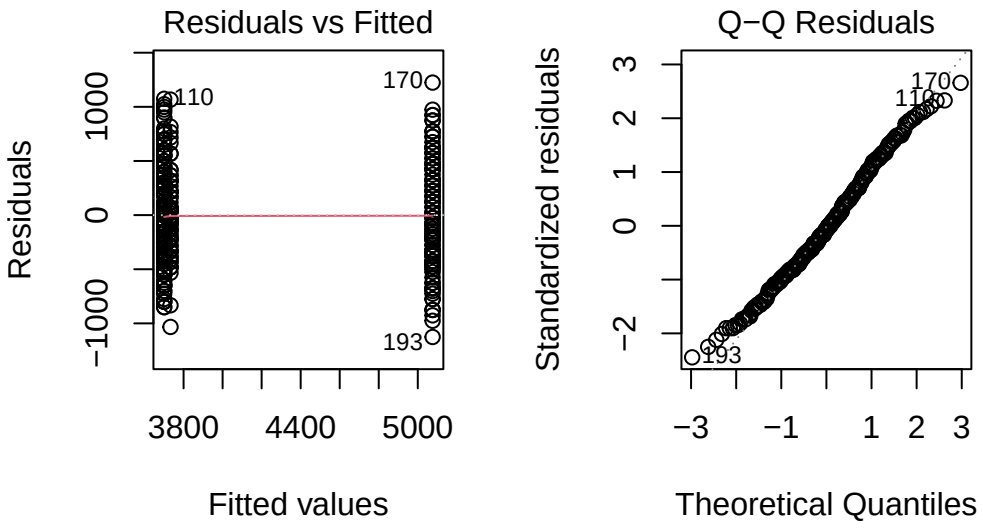
ANOVA undersöker skillnad mellan många grupper och faktorernas effekt på respons variabeln (liknande regression där kontinuerlig variabel kausalt påverkar respons).

2.4.2.1 Krav/antaganden

Anova är nästan samma som t-test,

- Data på intervall eller kvotskala (kontinuerlig)
- Normalfördelad (parametrisk) inom varje grupp (**Bootstrap**)
- Lika varians (homogen varians, ej för Welch t-test)
- Inga outliers
- **Oberoende data** (vi kör inte parat/beroende i denna kurs)
- **Förutsätter lika varians mellan grupper**
- **Kan hantera fler än två grupper samt fler än en faktor**

```r
# Skapa modellen, sedan diagnostik.
diagnostic_model <- aov(
  body_mass_g ~ species,
  data = penguins
)

par(mfrow = c(1,2))

plot(diagnostic_model, which = 1) # Lika varians om
# spridningen/"längden" på varje stapel med punkter är lika.

plot(diagnostic_model, which = 2) # QQ-plot för normalfördelning.
```




```
par(mfrow = c(1,1))
```

2.4.2.2 Envägs ANOVA

Envägs ANOVA innebär att vi använder en faktor (med flera grupper). Hur hittar man enkelt faktorer och grupper i ett dataset? En faktor är en kolumn med grupper. I `penguins` data frame är `penguins$species` en kolumn med grupper, alltså en faktor.

```
unique(penguins$species) # Unika element i kolumnen,
```

```
[1] Adelie    Gentoo    Chinstrap
Levels: Adelie Chinstrap Gentoo
```

```
# alltså, faktorn 'species' har grupperna 'Adelie', 'Gentoo',
 ↪  'Chinstrap'
```

ANOVA med funktionen `aov()`, `summary()` och `TukeyHSD()`.

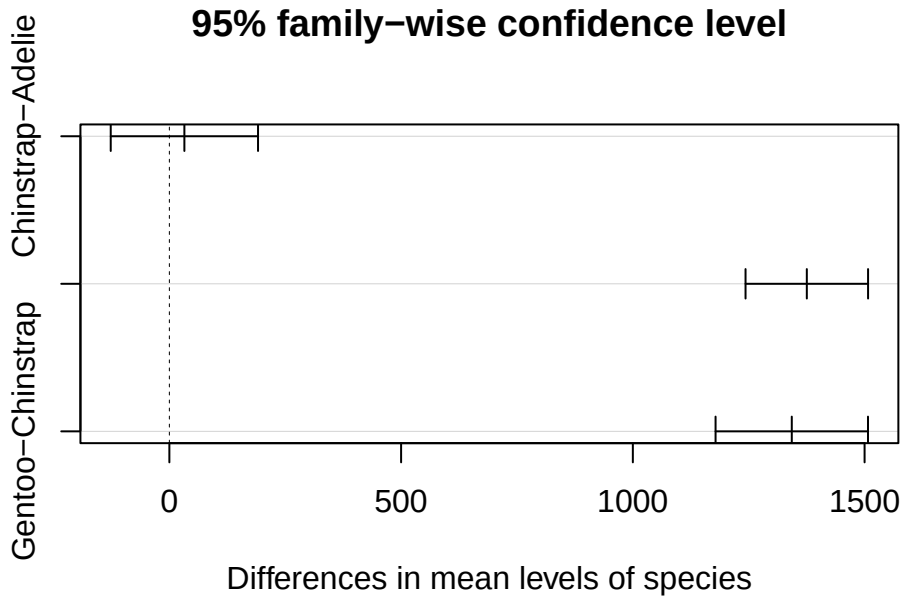
```r
anova_aov <- aov( # Skapa modell
  body_mass_g ~ species,
  data = penguins
)
summary(anova_aov) # Kolla om det fanns en skillnad mellan någon av
 ↪  grupperna
```

```
             Df    Sum Sq  Mean Sq F value Pr(>F)
species       2 146864214 73432107   343.6 <2e-16 ***
Residuals   339  72443483   213698
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
2 observations deleted due to missingness
```

```r
TukeyHSD(anova_aov) # Kolla mellan vilka grupper det fanns en skillnad
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = body_mass_g ~ species, data = penguins)

$species
                      diff       lwr       upr     p adj
Chinstrap-Adelie   32.42598 -126.5002  191.3522 0.8806666
Gentoo-Adelie    1375.35401 1243.1786 1507.5294 0.0000000
Gentoo-Chinstrap 1342.92802 1178.4810 1507.3750 0.0000000
```

```r
plot(TukeyHSD(anova_aov)) # Plot
```



ANOVA med funktionen `lm()`, `anova()` och `summary()`. Detta ger problem med att göra post-hoc test (`TukeyHSD()`) för att hitta mellan vilka grupper skillnaden finns. Fördelen skulle vara att få ett intercept och möjligen bygga en matematisk modell.

```r
anova_lm <- lm( # Skapa modell
  body_mass_g ~ species,
  data = penguins
)
anova(anova_lm) # Kolla om det fanns en skillnad mellan någon av
 ↪  grupperna
```

```
Analysis of Variance Table

Response: body_mass_g
           Df    Sum Sq  Mean Sq F value    Pr(>F)
species     2 146864214 73432107  343.63 < 2.2e-16 ***
Residuals 339  72443483   213698
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(anova_lm) # Kolla mellan vilka grupper det fanns en skillnad
```

```
Call:
lm(formula = body_mass_g ~ species, data = penguins)

Residuals:
     Min       1Q   Median       3Q      Max
-1126.02  -333.09   -33.09   316.91  1223.98

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       3700.66      37.62   98.37   <2e-16 ***
speciesChinstrap    32.43      67.51    0.48    0.631
speciesGentoo     1375.35      56.15   24.50   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 462.3 on 339 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared:  0.6697,    Adjusted R-squared:  0.6677
F-statistic: 343.6 on 2 and 339 DF,  p-value: < 2.2e-16
```

2.4.2.3 Tvåvägs ANOVA

Samma som envägs ANOVA men med två faktorer. Se Formel notation.

```r
anova2_aov <- aov( # Skapa modell
  body_mass_g ~ species * sex,
  data = penguins
)
summary(anova2_aov) # Kolla om det fanns en skillnad mellan någon av
 ↪  grupperna
```

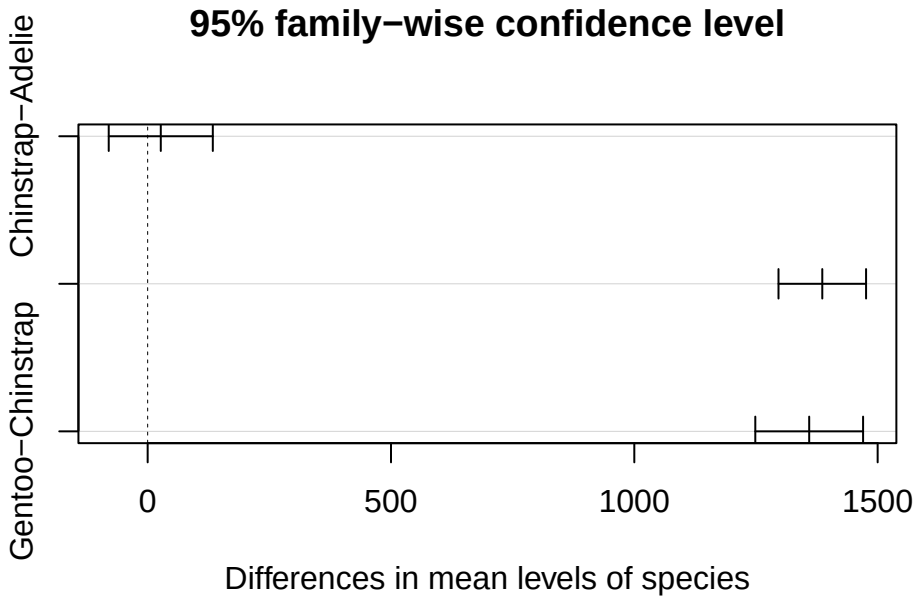
```
             Df    Sum Sq  Mean Sq F value   Pr(>F)
species       2 145190219 72595110 758.358  < 2e-16 ***
sex           1  37090262 37090262 387.460  < 2e-16 ***
species:sex   2   1676557   838278   8.757 0.000197 ***
Residuals   327  31302628    95727
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

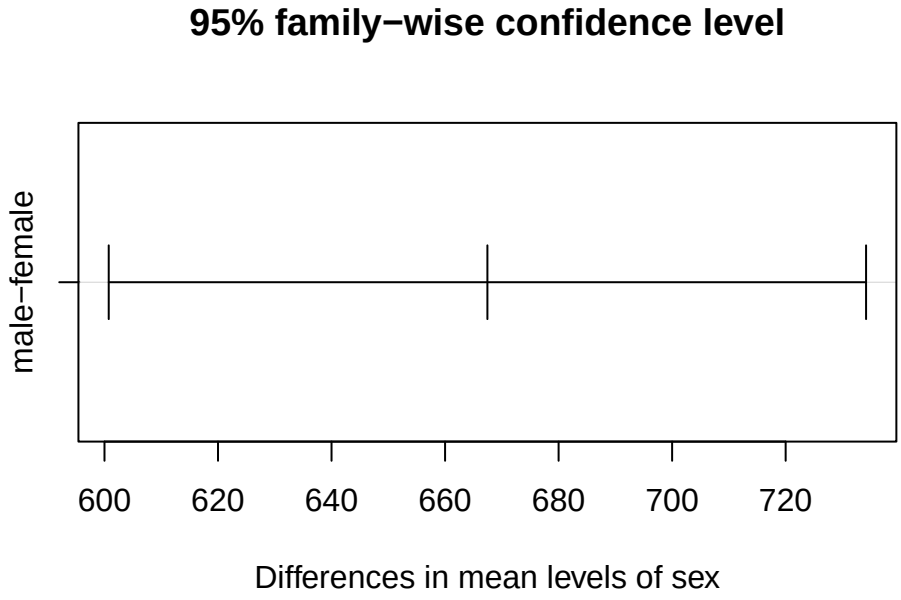
```
11 observations deleted due to missingness
```

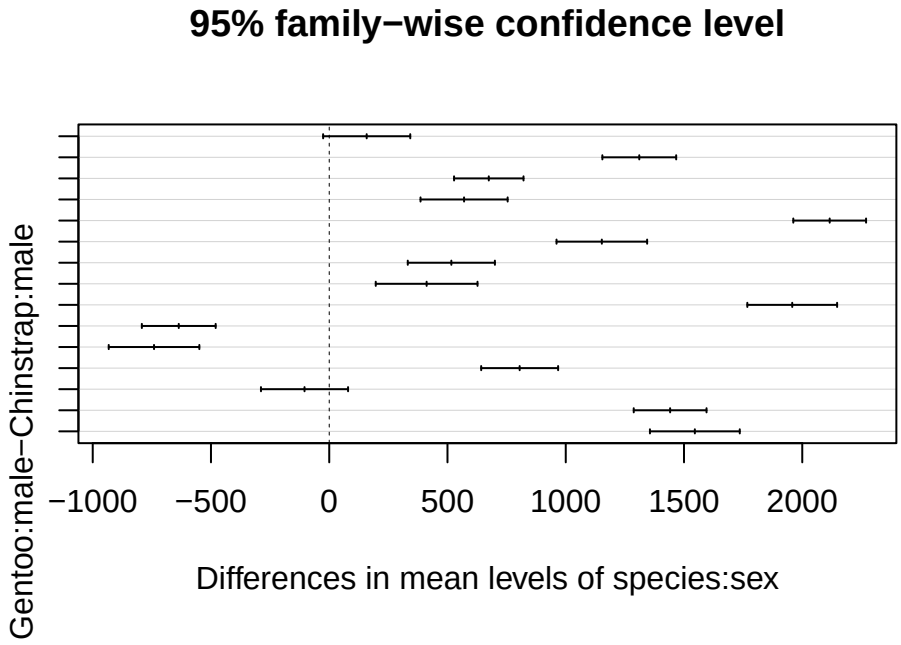
```
TukeyHSD(anova2_aov) # Kolla mellan vilka grupper det fanns en skillnad
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = body_mass_g ~ species * sex, data = penguins)

$species
                       diff       lwr       upr     p adj
Chinstrap-Adelie   26.92385  -80.0258  133.8735 0.8241288
Gentoo-Adelie    1386.27259 1296.3070 1476.2382 0.0000000
Gentoo-Chinstrap 1359.34874 1248.6108 1470.0866 0.0000000

$sex
                diff      lwr      upr p adj
male-female 667.4577 600.7462 734.1692     0

$`species:sex`
                                      diff       lwr       upr     p adj
Chinstrap:female-Adelie:female   158.3703  -25.7874  342.5279 0.1376213
Gentoo:female-Adelie:female     1310.9058 1154.8934 1466.9181 0.0000000
Adelie:male-Adelie:female        674.6575  527.8486  821.4664 0.0000000
Chinstrap:male-Adelie:female     570.1350  385.9773  754.2926 0.0000000
Gentoo:male-Adelie:female       2116.0004 1962.1408 2269.8601 0.0000000
Gentoo:female-Chinstrap:female  1152.5355  960.9603 1344.1107 0.0000000
Adelie:male-Chinstrap:female     516.2873  332.1296  700.4449 0.0000000
Chinstrap:male-Chinstrap:female  411.7647  196.6479  626.8815 0.0000012
Gentoo:male-Chinstrap:female    1957.6302 1767.8040 2147.4564 0.0000000
Adelie:male-Gentoo:female       -636.2482 -792.2606 -480.2359 0.0000000
Chinstrap:male-Gentoo:female    -740.7708 -932.3460 -549.1956 0.0000000
Gentoo:male-Gentoo:female        805.0947  642.4300  967.7594 0.0000000
Chinstrap:male-Adelie:male      -104.5226 -288.6802   79.6351 0.5812048
Gentoo:male-Adelie:male         1441.3429 1287.4832 1595.2026 0.0000000
Gentoo:male-Chinstrap:male      1545.8655 1356.0392 1735.6917 0.0000000
```

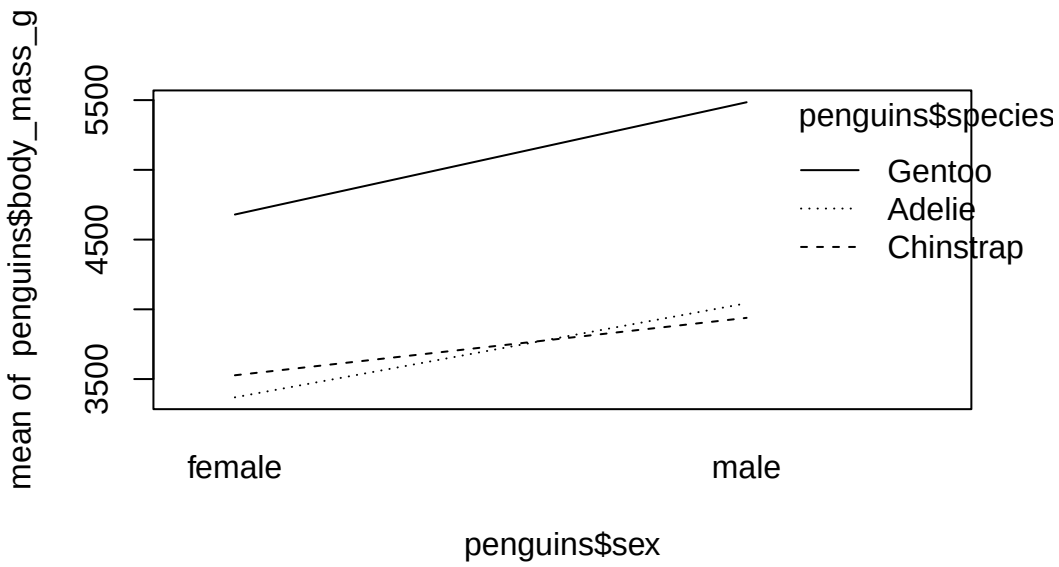
```
plot(TukeyHSD(anova2_aov)) # Plot
```


95% family−wise confidence level
Chinstrap−Adelie
Gentoo−Chinstrap
0
500
1000
1500
Differences in mean levels of species


95% family−wise confidence level
male−female
600
620
640
660
680
700
720
Differences in mean levels of sex




```r
# Interaktions plot
interaction.plot(
  x.factor = penguins$sex,          # x-axeln
  trace.factor = penguins$species, # linjerna
  response = penguins$body_mass_g, # y-axeln
)
```



2.4.2.4 Kruskal-Wallis (icke parametrisk)

Icke parametrisk variant av **envägs** ANOVA

```
kruskal.test(
  body_mass_g ~ species,
  data = penguins
)
```

```
    Kruskal-Wallis rank sum test

data:  body_mass_g by species
Kruskal-Wallis chi-squared = 217.6, df = 2, p-value < 2.2e-16
```

Detta är utanför kursen! För att hitta mellan vilka grupper skillnaden finns (post-hoc). Kan inte hantera NA och har krashat R…

```
pairwise.wilcox.test(
  x = penguins$body_mass_g, # Vektor med mätdata
  g = penguins$species,     #
  p.adjust.method = "holm"
```

```
)
```

```
    Pairwise comparisons using Wilcoxon rank sum test with continuity correction

data:  penguins$body_mass_g and penguins$species

          Adelie Chinstrap
Chinstrap 0.49   -
Gentoo    <2e-16 <2e-16

P value adjustment method: holm
```

2.4.2.5 Bootstrap (icke parametriskt)

Om någon av grupperna inte är normalfördelade kan man använda bootstrap. Alla grupper ska fortfarande ha samma varians.

ANOVA.boot() kan inte hantera NA, så se till att filtrera data frame.

```
# Subset to remove NA
peng_boot <- penguins[!is.na(penguins$body_mass_g), ]
peng_boot <- peng_boot[!is.na(peng_boot$species), ]
peng_boot <- peng_boot[!is.na(peng_boot$sex), ]

anova_boot <- ANOVA.boot(
  body_mass_g ~ species * sex,
  data = peng_boot
)
anova_boot[c("terms","df","p-values")]
```

```
$terms
[1] "species    " "sex        " "species:sex" "Residuals  "

$df
[1]   2   1   2 327

$`p-values`
[1] 0.000 0.000 0.001
```

Detta är inte särskillt nice output format så vi undviker helst att använda `ANOVA.boot()`.

2.5 Kontinuerliga samband

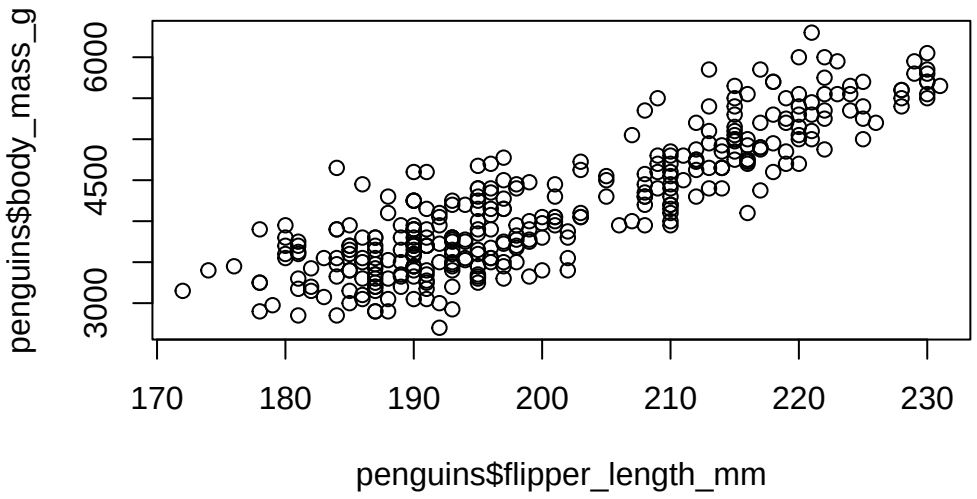
2.5.1 Samvariation/korrelation

När två kontinuerliga variabler ökar/minskar tillsammans (positiv korrelation) eller i motsatt riktning (negativ korrelation), möjligtvis till följd av en gemensam tredje (dold) variabel.

2.5.1.1 Krav/antaganden

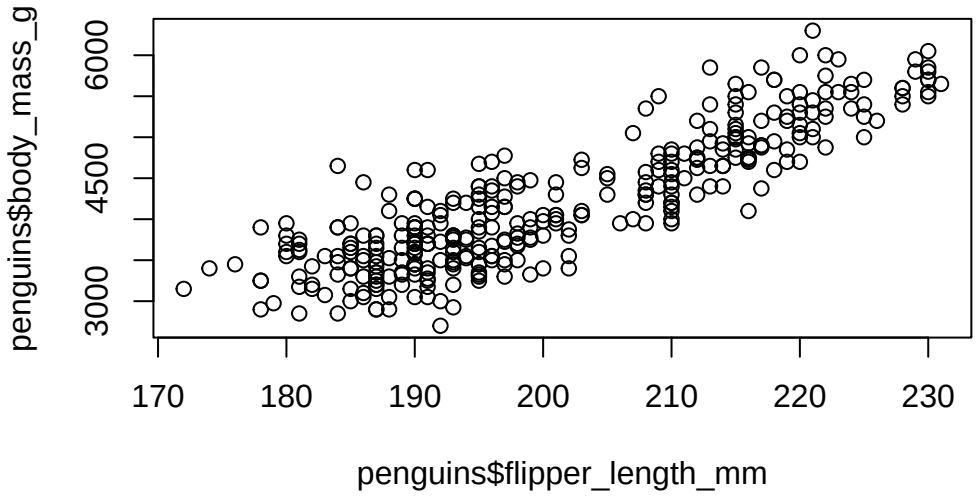
- Två kontinuerliga variabler/mätdata
- Linjärt samband (kan göras linjär genom transformation om nödvändigt)

```
plot( # Ögonmåtta att sambandet inte är uppenbarligen icke-linjärt
  penguins$flipper_length_mm,
  penguins$body_mass_g
)
```



2.5.1.2 Pearson korrelation

```r
plot( # Kolla alltid data
  penguins$flipper_length_mm,
  penguins$body_mass_g
)
```




```r
cor.test(
  penguins$flipper_length_mm,
  penguins$body_mass_g
)
```

```
    Pearson's product-moment correlation

data:  penguins$flipper_length_mm and penguins$body_mass_g
t = 32.722, df = 340, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.843041 0.894599
sample estimates:
      cor
0.8712018
```

p-värdet tolkas som vanligt, cor är r^2-värdet som beskriver styrka och riktning av korrelationen.

2.5.1.3 Spearman (icke parametrisk)

```r
cor.test(
  penguins$flipper_length_mm,
  penguins$body_mass_g,
  method = "spearman" # Specificera spearman
)
```

```
Warning in cor.test.default(penguins$flipper_length_mm, penguins$body_mass_g, :
Cannot compute exact p-value with ties
```

```
    Spearman's rank correlation rho

data:  penguins$flipper_length_mm and penguins$body_mass_g
S = 1066875, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.8399741
```

För denna kurs, ignorera `Warning: cannot compute exact p-value with ties`. Det betyder att det finns två av samma mätvärde vilket inte går att rangordna. (Om data är `c(25, 26, 26, 27)` är det `(1,2,3,4)` eller `(1,3,2,4)`?)

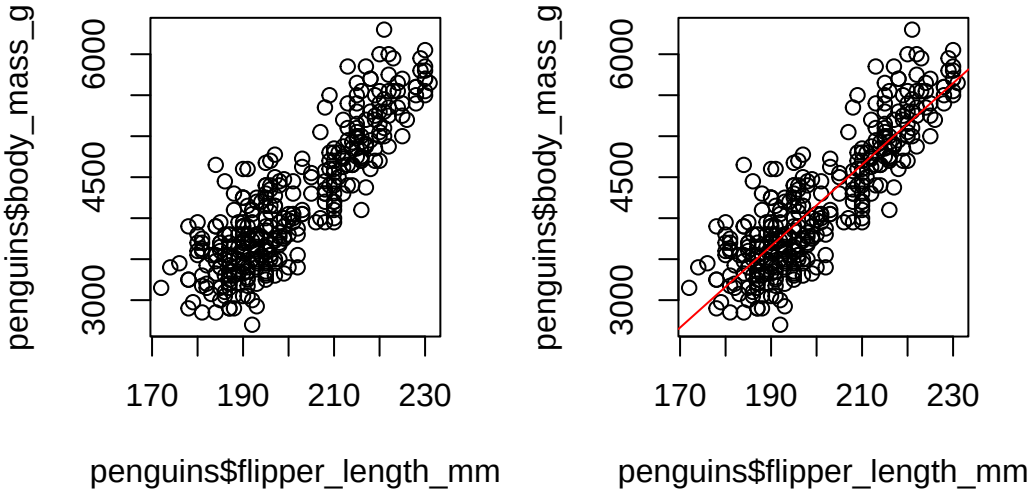
2.5.2 Regression

Regression används då en kontinuerlig variabel kausalt orsakar en annan kontinuerlig variabel.

2.5.2.1 Krav/antaganden

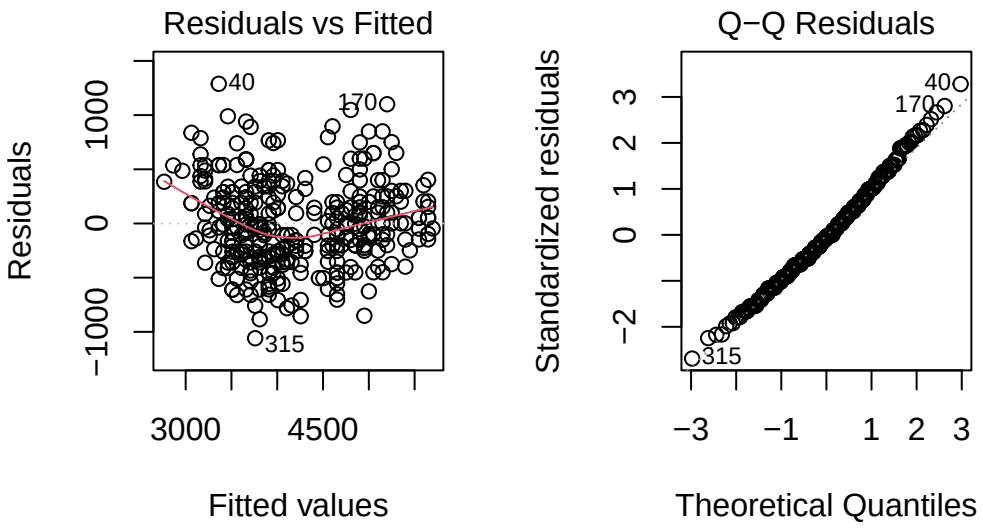
- **Linjärt samband**
- Normalfördelad (parametrisk)
- Lika varians

```r
par(mfrow = c(1,2))

plot( # Ögonmåtta om det finns en linjär trend.
  x = penguins$flipper_length_mm,
  y = penguins$body_mass_g
)

# Skapa modellen, sedan diagnostik.
diagnostic_model <- aov(
  body_mass_g ~ flipper_length_mm,
  data = penguins
)

plot( # Plot med linjen från modellen som hjälp
  x = penguins$flipper_length_mm,
  y = penguins$body_mass_g
)
abline(diagnostic_model,
       col = "red")
```



```r
par(mfrow = c(1,1))
```

```r
par(mfrow = c(1,2))

plot(diagnostic_model, which = 1) # Lika varians om
# spridningen av punkter är uniform & röda linjen är platt.

plot(diagnostic_model, which = 2) # QQ-plot för normalfördelning.
```



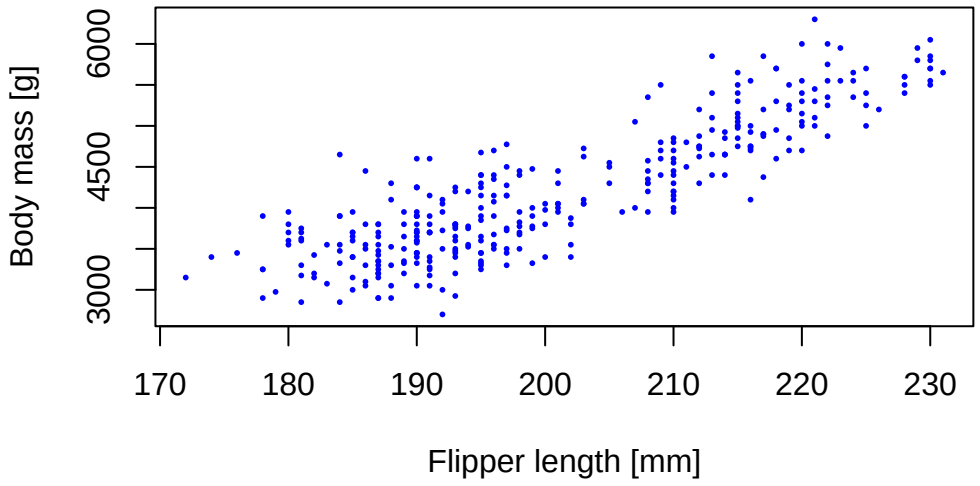

```r
par(mfrow = c(1,1))
```

2.5.2.2 Linjär regression

Skapas med en formel liknande för ANOVA.

Med det högst biologiska antagandet att längre fenor orsakar högre kroppsvikt (genom att pingvinerna simmar fortare och därmed kan äta mer, säger vi) enligt en linjär trend, väljer vi att göra linjär regression.

```r
plot( # Scatterplot för att kolla sambandet
  penguins$flipper_length_mm,
  penguins$body_mass_g,
  pch = 16,
  cex = 0.4,
  col = "blue",
  xlab = "Flipper length [mm]",
  ylab = "Body mass [g]"
)
```




```r
mass_by_flipper <- lm( # Skapa model
  body_mass_g ~ flipper_length_mm,
  data = penguins
)
summary(mass_by_flipper)
```

```
Call:
lm(formula = body_mass_g ~ flipper_length_mm, data = penguins)

Residuals:
```

```
     Min       1Q   Median       3Q      Max
-1058.80  -259.27   -26.88   247.33  1288.69

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       -5780.831    305.815  -18.90   <2e-16 ***
flipper_length_mm    49.686      1.518   32.72   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 394.3 on 340 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared:  0.759, Adjusted R-squared:  0.7583
F-statistic:  1071 on 1 and 340 DF,  p-value: < 2.2e-16
```

2.5.2.3 Bootstrap

Alternativt med `lm()` och sedan `boot_summary()`.

```r
boot.anova_lm <- lm( # Skapa model
  body_mass_g ~ species * sex,
  data = peng_boot )
boot_summary(boot.anova_lm)
```

```
                          Estimate Lower.bound Upper.bound p.value
(Intercept)              3368.8356  3296.45916  3435.65879   0.000
speciesChinstrap          158.3703    31.32257   293.85606   0.014
speciesGentoo            1310.9058  1205.61419  1425.13706   0.000
sexmale                   674.6575   578.76645   775.89744   0.000
speciesChinstrap:sexmale -262.8928  -449.13583   -82.31911   0.008
speciesGentoo:sexmale     130.4372   -29.37236   280.44679   0.115
```

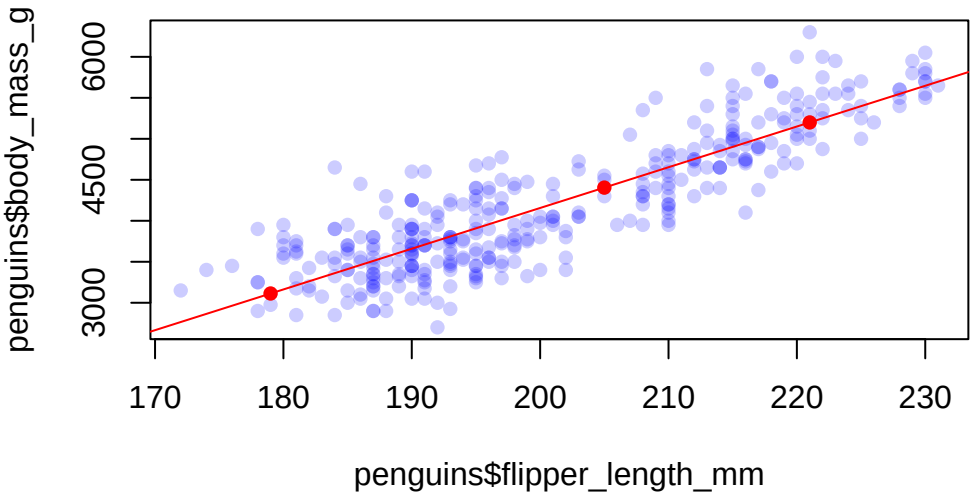
2.5.2.4 Predict

Om längden på fenorna orsakar kroppsmassa kan vi använda modellen för att förutse vikten på en pingvin baserat på fenornas längd.

Argumentet `newdata = data.frame()` måste innehålla samma kolumn namn som x-axeln och en vektor med de värden på x-axeln man vill förutspå y-värdet för.

```r
# Data frame med nya värden att förutspå.
# Måste ha en kolumn med samma namn som x-axeln i modellen!
measured_flipper_length <- data.frame(
  flipper_length_mm = c(205, 221, 179)
)

# De värden vi gissar med hjälp av modellen
estimated_weight <- predict(
  object = mass_by_flipper, # Modellen
  newdata = measured_flipper_length) # nya x-värden
```

```r
plot(
  penguins$flipper_length_mm,
  penguins$body_mass_g,

  pch = 16,
  col = alpha("blue", 0.2)
)
points(
  x = measured_flipper_length$flipper_length_mm, # newdata
  y = estimated_weight, # Gissade värden

  pch = 16,
  col = "red"
)
abline( # Lägg till linjen från modellen.
  reg = mass_by_flipper,
  col = "red",
  lwd = 1
)
```



2.6 Fördelning av frekvens data

2.6.1 Chi-square (X^2-test)

Det enda testet på kursen som hanterar frekvenser/räknedata. Principiellt behövs tre saker, en faktor med grupper som blir kolumner, en faktor med grupper som blir rader, individer/föremål att räkna.

Till ära av Tobbes doktorsavhandling används här data om Gemsbock och Red hartbeest och deras observerade fördelning på fyra områden.

```
bockar <- rbind(
  Gemsbock =      c(FR = 5, CGA = 7, WMA = 12, NP = 67),
  Red_hartbeest = c(     6,       8,       54,      13)
)

bockar
```

```
              FR CGA WMA NP
Gemsbock       5   7  12 67
Red_hartbeest  6   8  54 13
```

2.6.1.1 Krav/antaganden

- Frekvens data (nominal skala)
- Väntevärdet/expected i varje cell ska vara minst 5 (Fishers exact test)

```
low_expected <- rbind(
  c(2,3,4),
  c(4,5,6)
)
low_expected_test <- chisq.test(
  low_expected
)
```

```
Warning in chisq.test(low_expected): Chi-squared approximation may be incorrect
```

```
low_expected_test$expected
```

```
     [,1] [,2] [,3]
[1,] 2.25    3 3.75
[2,] 3.75    5 6.25
```

2.6.1.2 Anpassning

Kollar om en rad data "kan anpassas" efter en fördelning, vanligen lika många i varje kategori. Så om data ser ut såhär,

```
bockar["Gemsbock", , drop = FALSE] # Det extra ', drop = FALSE' gör att
 ↪  vi behåller namnet på raden (som annars försvinner mystiskt när
 ↪  matrisen har bara en rad).
```

```
         FR CGA WMA NP
Gemsbock  5   7  12 67
```

```
# Detta kan ignoreras och skulle då vara 'bockar["Gemsbock", ]'.

bockar_anpassning <- chisq.test(
  bockar["Gemsbock", ]
```

```
)
bockar_anpassning # Visa resultatet
```

```
    Chi-squared test for given probabilities

data:  bockar["Gemsbock", ]
X-squared = 115.9, df = 3, p-value < 2.2e-16
```

```
bockar_anpassning$expected # Visa väntevärden
```

```
   FR   CGA   WMA    NP
22.75 22.75 22.75 22.75
```

Alternativt, anpassning mot en specifik fördelning.

```
bockar_anpassning <- chisq.test(
  bockar["Gemsbock", ],
  p = c(0.10, 0.15, 0.35, 0.40) # Totalt ska dessa bli 1
)
bockar_anpassning # Visa resultatet
```

```
    Chi-squared test for given probabilities

data:  bockar["Gemsbock", ]
X-squared = 43.182, df = 3, p-value = 2.251e-09
```

```
bockar_anpassning$expected # Visa väntevärden
```

```
   FR   CGA   WMA    NP
 9.10 13.65 31.85 36.40
```

2.6.1.3 Oberoende

Testa slumpmässiga observationer från två olika faktorer samtidigt för att se om de kommer från samma fördelning. Innebär att alla kolumner och alla rader kan ha olika summor. Så om data ser ut såhär,

```r
# Skapa ny kolumn med summan av varje rad
bockar_sums <- cbind(
  bockar,
  Row_sum = rowSums(bockar)
)
# Skapa ny rad med summan av varje kolumn
bockar_sums <- rbind(
  bockar_sums,
  Col_sum = colSums(bockar_sums)
)
bockar_sums
```

```
              FR CGA WMA NP Row_sum
Gemsbock       5   7  12 67      91
Red_hartbeest  6   8  54 13      81
Col_sum       11  15  66 80     172
```

Varken alla rader eller alla kolumner har samma summor, alltså gör vi ett oberoende test.

```r
bockar_oberoende <- chisq.test(
  bockar
)
bockar_oberoende
```

```
    Pearson's Chi-squared test

data:  bockar
X-squared = 62.966, df = 3, p-value = 1.365e-13
```

```r
bockar_oberoende$expected
```

```
                   FR       CGA     WMA       NP
Gemsbock      5.819767 7.936047 34.9186 42.32558
Red_hartbeest 5.180233 7.063953 31.0814 37.67442
```

2.6.1.4 Homogenitet

Testa slumpmässiga observationer från **en** faktor och kontrollera den andra. Innebär att **antingen** alla rader **eller** kolumner har samma summa (medan den andra varierar).

För detta behöver vi ny data. Ett stort företag vill undersöka om antalet vaccinerade anställda skiljer sig mellan avdelningarna. För att kontrollera för de olika avdelningarna slumpades 100 anställda från varje avdelning och frågade om de var vaccinerade eller inte.

```
uppsala_site <- rbind( # Data
  vaccinerad = c(RnD = 99, HR = 45, Lön = 63, IT = 76),
  ej_vaccinerade = c(   1,      55,       37,      24)
)

uppsala_site
```

```
               RnD HR Lön IT
vaccinerad      99 45  63 76
ej_vaccinerade   1 55  37 24
```

Vi kan utforska företagets data lite mer genom att lägga till rad och kolumn summor.

```
# Skapa ny kolumn med summan av varje rad
uppsala_sums <- cbind(
  uppsala_site,
  Row_sum = rowSums(uppsala_site)
)
# Skapa ny rad med summan av varje kolumn
uppsala_sums <- rbind(
  uppsala_sums,
  Col_sum = colSums(uppsala_sums)
)
uppsala_sums
```

```
               RnD  HR Lön  IT Row_sum
vaccinerad      99  45  63  76     283
ej_vaccinerade   1  55  37  24     117
Col_sum        100 100 100 100     400
```

Detta säger oss att den faktorn är kontrollerad, alltså gör vi ett Homogenitetstest.

```r
vaccin_homogen <- chisq.test(
  uppsala_site
)

vaccin_homogen
```

```
    Pearson's Chi-squared test

data:  uppsala_site
X-squared = 74.839, df = 3, p-value = 3.923e-16
```

```r
vaccin_homogen$expected
```

```
                RnD    HR   Lön    IT
vaccinerad     70.75 70.75 70.75 70.75
ej_vaccinerade 29.25 29.25 29.25 29.25
```

2.6.2 Fisher's exact (icke parametrisk)

Ifall en eller flera celler med väntevärden/expected är < 5 kan man använda Fisher's exact test.

```r
uppsala_fisher <- fisher.test(
  uppsala_site
)

uppsala_fisher # Inga expected values
```

```
    Fisher's Exact Test for Count Data

data:  uppsala_site
p-value < 2.2e-16
alternative hypothesis: two.sided
```

3 Matte

3.1 Matematiska beräkningar

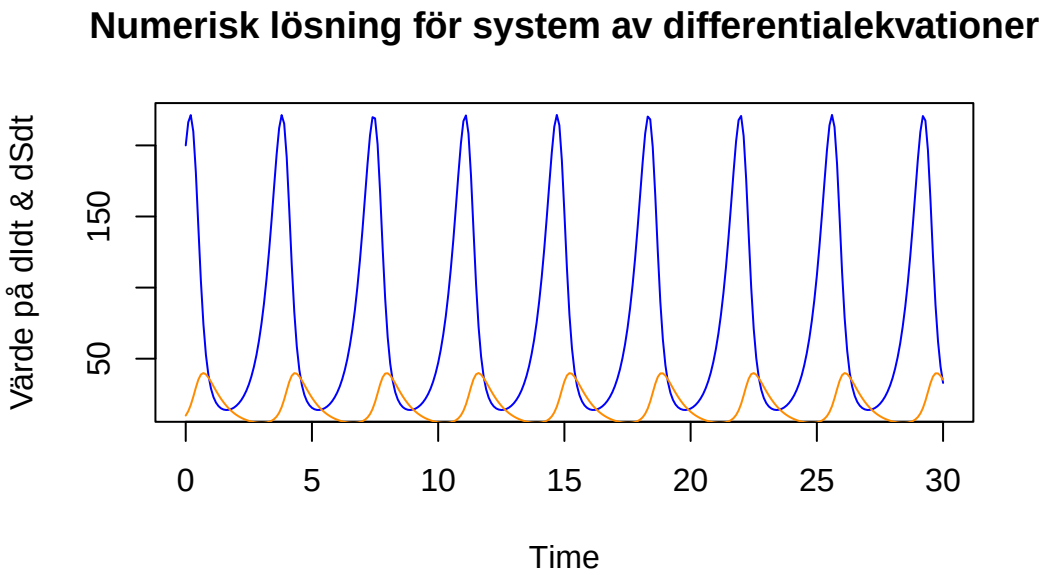
```
1 + 1   # addition
1 - 1   # subtraktion
5 * 4   # multiplikation
6 / 2   # division
5^2     # upphöjt med

sqrt(25)  # roten ur, (positiv)
exp(3)    # e^3
log(7)    # log(7) med basen e
log10(7)  # log(7) med basen 10
```

3.2 Differentialekvationer

```
# Funktion med ekvationssystemet att lösa
DE_fun <- function(times, # Intervall
                   y,     # Begynnelsevärden
                   parms  # Andra värden
) {
  # variabler att använda (från parms)
  alfa <- parms["alfa"]
  beta <- parms["beta"]
  delta <- parms["delta"]
  gamma <- parms["gamma"]

  # Begynnelsevärden (från y)
  B <- y["B"]
  P <- y["P"]

  # Differentialekvationen att lösa numeriskt
  #   (Minst 1 ekv. men sedan är det bara att lägga till fler ekv.
  #   på var sin rad. Varje ekv. måste ha ett begynnelsevärde &
  #   sedan returneras i result_vec)
  dB <- alfa * B - beta * B * P
  dP <- delta * B * P - gamma * P
```

```r
  # Spara & returnera resultatet (ode() är lite speciell...)
  result_vec <- c( # Först som vektor
    dB,
    dP
  )
  result_list <- list(result_vec) # Sedan som lista
  return(result_list) #Returnera
}

# Vektor med alla tidssteg (t) att sätta in
DE_time_span <- seq(
  from = 0, # Början på intervall
  to = 30,  # Slut på intervall
  by = 0.1  # Steglängd
)

# Vektor med begynnelsevärden
#   (enklast att namnge dem och sedan indexera med namnet i fun)
DE_init <- c(
  B = 200,
  P = 10
)

# Vektor med andra värden för formeln
#   (enklast att namnge dem och sedan indexera med namnet i fun)
DE_params <- c(
  alfa = 2.5,
  beta = 0.15,
  delta = 0.02,
  gamma = 1.5
)

# Lösa ekvationerna numeriskt
DE_sol <- ode(
  y = DE_init,           # Begynnelsevärden
  times = DE_time_span,  # Intervall
  func = DE_fun,         # Funktionen
  parms = DE_params,     # Andra värden
  method = "rk4"         # Runge-Kutta version 4
)
# Konvertera till data frame för att göra den lättare att hantera
```

```r
DE_sol <- as.data.frame(DE_sol)


# Plotta dB respektive dP mot tid
plot( # Plot fönster & dB mot tid
  DE_sol$time,
  DE_sol$B,
  type = "l",

  col = "blue",

  xlab = "Time",
  ylab = "Värde på dIdt & dSdt",
  main = "Numerisk lösning för system av differentialekvationer"
)
lines( # Lägg till dP mot tid
  DE_sol$time,
  DE_sol$P,

  col = "darkorange"
)
```



4 KvantBio

4.1 Fibonacci

```
n <- c(1,2)
for (i in 3:25) {
  n[i] <- n[i-1] + n[i-2]
}
n
```

```
 [1]     1     2     3     5     8    13    21    34    55    89
[11]   144   233   377   610   987  1597  2584  4181  6765 10946
[21] 17711 28657 46368 75025 121393
```

4.2 Hantera modeller

Kunna använda modeller och lösa för saknade parametrar. Översätt de matematiska funktionerna till funktioner i R. Funktionen `uniroot()` är användbar för att hitta nollställen.

4.2.1 Allometri & energiförbrukning

$$K = P + M + E$$

Bioenergetisk modell där K = konsumtion, P = produktion/ökning i massa, M = metabolism, E = exkretion.

$$log_{10}(O_2) = 0.36 \cdot \frac{S}{TL} + 1.945$$

Modell för citronhajars metabolism baserat på sambandet mellan simhastighet och syreförbrukning (för hajar som väger 1kg).

4.2.2 Individuell tillväxt (von Bertalanffys tillväxtekvation)

$$L(t) = L_\infty + (L_0 - L_\infty) \cdot e^{-kt}$$

Där L_inf är maxlängd vid oändlig tid, L_noll är längden vid t = 0 , k är en tillväxtfaktor och t är tid/ålder.

För att hitta längden vid givet t.

```r
# Parametrar
L_inf <- 85
k <- 0.00082
L_noll <- 30

# Funktion av t
growth_av_t <- function(t) {
  L_t <- L_inf + (L_noll - L_inf) * exp(-k * t)
  return(L_t)
}
growth_av_t(t = 10)
```

```
[1] 30.44916
```

För att hitta L_inf (eller annan okänd från funktionen) kan vi möblera om till

$$0 = L_\infty + (L_0 - L_\infty) \cdot e^{-kt} - L_t$$

och använda `uniroot()`.

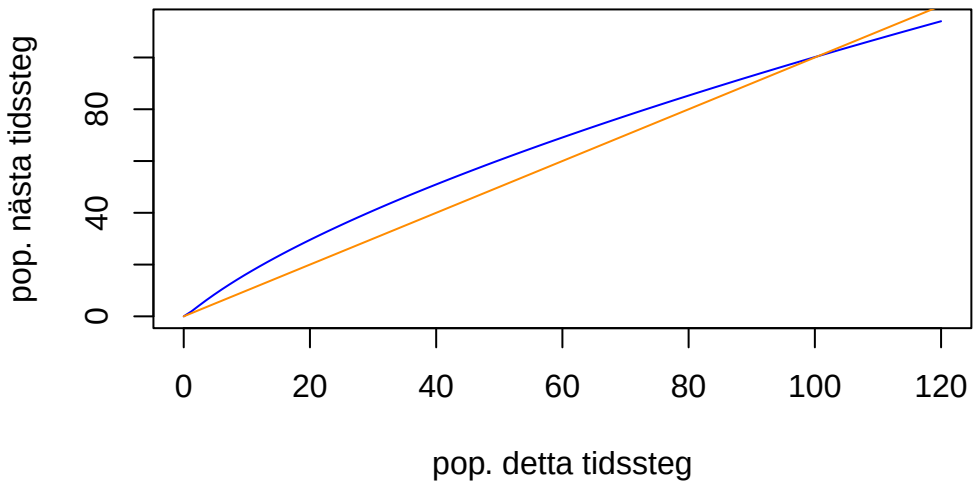
```r
# Parametrar
k <- 0.00082
L_noll <- 30
L_t <- 95
t <- 2000

# Funktion av L_inf
growth_av_L_inf <- function(L_inf) {
  noll_punkt <- L_inf + (L_noll - L_inf) * exp(-k * t) - L_t
  return(noll_punkt)
}

# Hitta nollpunkten med uniroot()
uniroot(
  f = growth_av_L_inf,
  interval = c(0,200)
)$root
```

```
[1] 110.6432
```

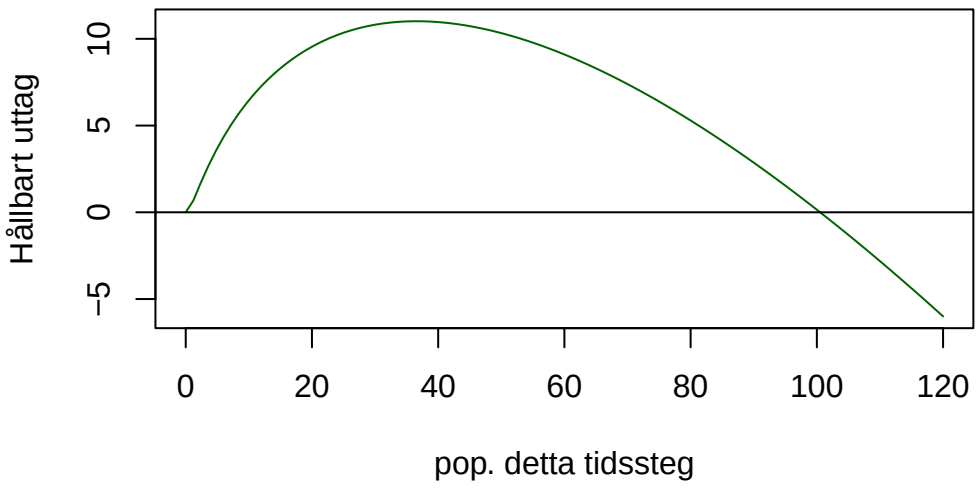
4.3 Hållbart uttag

Hur tillväxer en population och vid vilken storlek kan man plocka ut så många individer som möjligt och samtidigt få en stabil population?

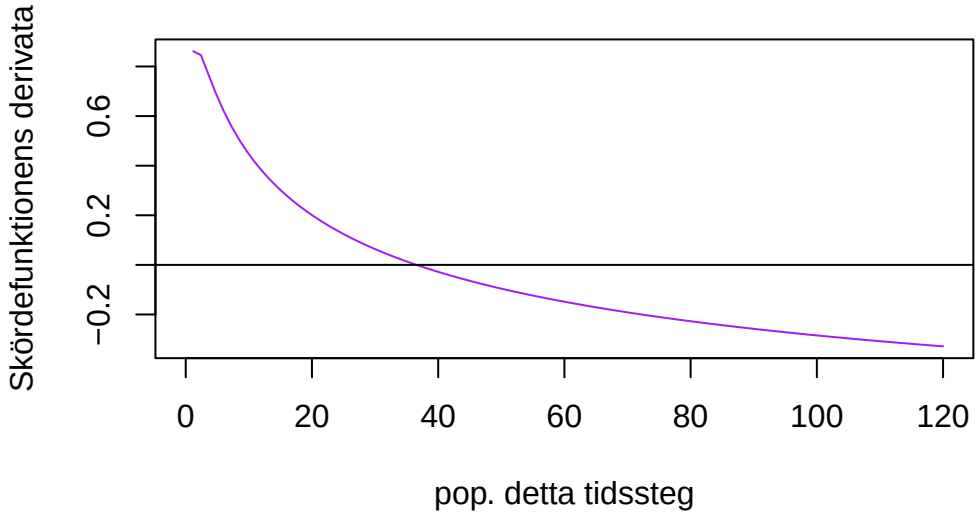
```r
# Rekryteringsfunktion, n individer detta år ger n+x individer nästa år
recruitment <- function(n) {
  recruit <- 2.17 * sqrt(n) * log(n+1)
  return(recruit)
}
# Plot
curve(
  recruitment,
  xlim = c(0,120),
  col = "blue",
  xlab = "pop. detta tidssteg",
  ylab = "pop. nästa tidssteg"
)
curve(
  1*x,
  add = TRUE,
  col = "darkorange"
)
```




```r
# Skördekurva
harvest <- function(n) {
  harvest <- recruitment(n) - n
  return(harvest)
}

# Plot
curve(
  harvest,
  xlim = c(0,120),
  xlab = "pop. detta tidssteg",
  ylab = "Hållbart uttag",
  col = "darkgreen"
)
abline(
  h = 0
)
```




```r
# Hitta x-värde för maxpunkt, alltså pop. storlek för max hållbart
↪  uttag.

# Derivata av skördefunktionen
harvest_deriv <- function(n) {
  harvest_deriv <- fderiv(harvest, n)
  return(harvest_deriv)
}

# Plot (indikerar intervallet för uniroot())
curve(
  harvest_deriv,
  xlim = c(0,120),
  xlab = "pop. detta tidssteg",
  ylab = "Skördefunktionens derivata",
  col = "purple"
)
abline(
  h = 0
)
```




```
# Hitta nollpunk numeriskt = pop. storlek för max hållbart uttag
harvest_deriv_root <- uniroot(
  harvest_deriv,
  interval = c(20,60)
)
```

```
# Insättning av nollpunkt i skördefunktion ger max hållbart uttag
harvest(harvest_deriv_root$root)
```

```
[1] 11.0155
```

4.4 Diskret logistisk tillväxt

$$x_{n+1} = x_n + r \cdot x_n \cdot (1 - \frac{x_n}{k})$$

Funktion för diskret logistisk tillväxt.

```
dlt <- function(n, r, k) {
  n_ny <- n + r*n*(1-n/k)
```

```
  return(n_ny)
}
```

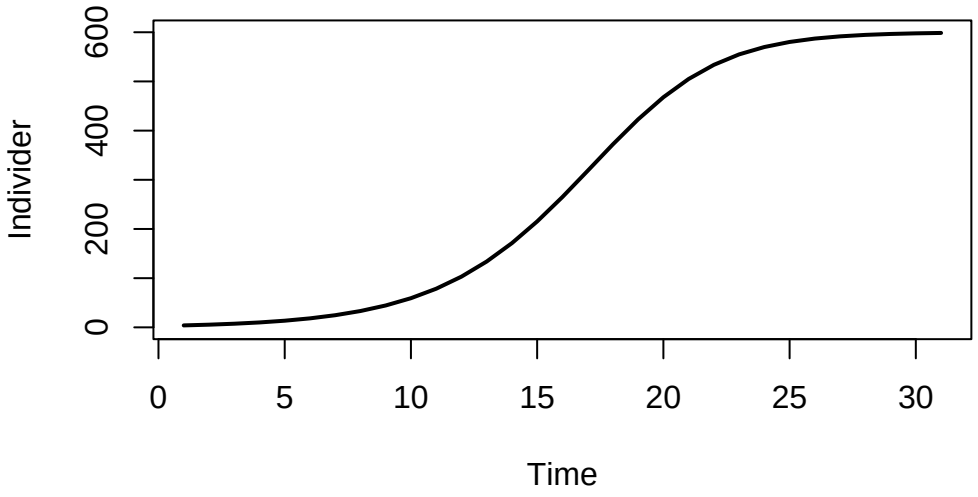
4.4.1 Modellera en population

Definiera värden på parametrar.

```
r <- 0.36 # perkapitareproduktion (egenvärde från Leslie-matris)
k <- 600  # Bärkraft

sluttid <- 30
t <- 1:(sluttid+1)
```

En populationsutveckling med konstant r-värde.

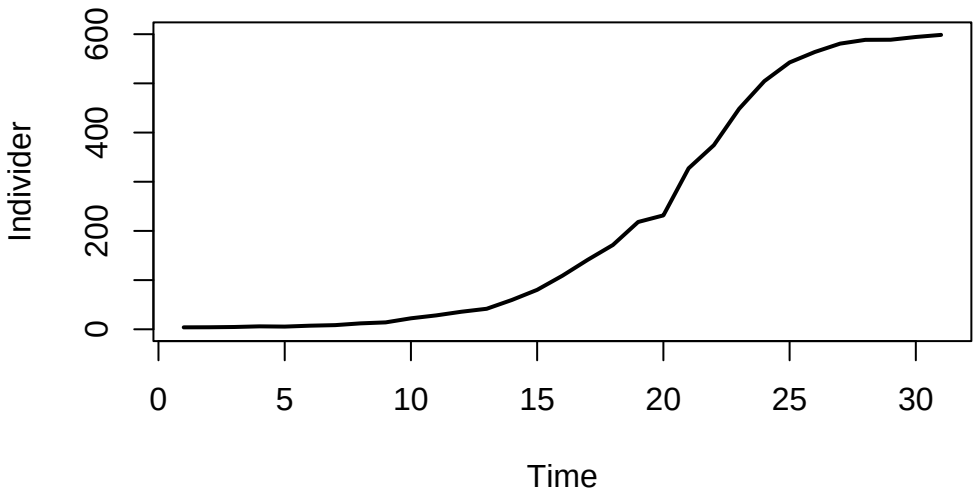
```
n <- c(4) # Startvärde & vektor att lagra resultat i

for (i in 1:sluttid) { # Loop för pop. utveckling
  n[i+1] <- dlt(n = n[i],
                r = r,
                k = k)
}

plot(
  x = t,
  y = n,
  ylim = c(0,k),

  type = "l",
  lwd = 2,

  xlab = "Time",
  ylab = "Individer"
)
```



Samma, men med varierande r-värde.

```
# r <- 0.36 # perkapitareproduktion (egenvärde från Leslie-matris)
k <- 600  # Bärkraft

sluttid <- 30
t <- 1:(sluttid+1)

n <- c(4) # Startvärde & vektor att lagra resultat i

for (i in 1:sluttid) { # Loop för pop. utveckling
  n[i+1] <- dlt(n = n[i],
                r = rnorm(n = 1,
                          mean = 0.36,
                          sd = 0.2),
                k = k)
}

plot(
  x = t,
  y = n,
```

```
  ylim = c(0,k),

  type = "l",
  lwd = 2,

  xlab = "Time",
  ylab = "Individer"
)
```



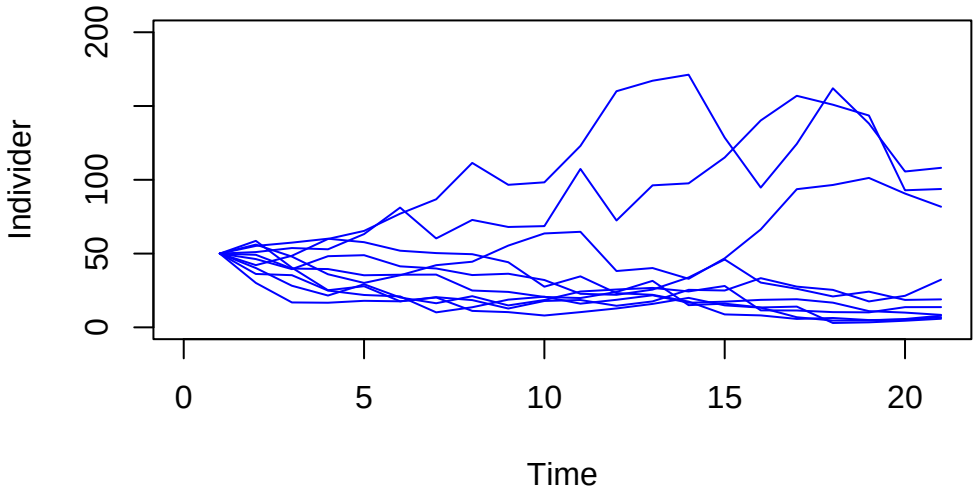
4.4.2 Modellera många populationer & beräkna utdöende risk

Definiera värden på parametrar.

```
# r defineras i for-loop
k <- 1000 # Bärkraft

sluttid <- 20
t <- 1:(sluttid+1)

collapse <- 10 # Färre individer = pop. dör ut
```

Skapa många linjer för hypotetiska populationers öden.

```
plot( # Skapa tom plot
  NULL,
  xlim = c(0, sluttid+1),
  ylim = c(0, 200),
  type = "l",
  xlab = "Time",
  ylab = "Individer"
)

n <- c(50) # Startvärde & vektor att lagra resultat i
pop_collapse <- 0 # Räknare för hur många populationer har dött ut
attempts <- 10

for (i in 1:attempts) { # Loop för upprepade försök/linjer

  for (j in 1:sluttid) { # Loop för en linje/population
    n[j+1] <- dlt(n = n[j],
                  r = rnorm(n = 1, # slumpad reproduktion
                            mean = 0,
                            sd = 0.25),
                  k = k)
  }

  if (min(n) < collapse) { # Kolla om pop. collapsade
    pop_collapse <- pop_collapse +1 # Addera 1 till räknare
    print(paste("Försök",i,"dog ut. Totalt",pop_collapse,"collapsade
     ↪  populationer."))
  }

  lines( # Addera linje för ett försök
    x = t,
    y = n,
    lwd = 1,
    col = "blue"
  )

}
```




```
[1] "Försök 1 dog ut. Totalt 1 collapsade populationer."
[1] "Försök 3 dog ut. Totalt 2 collapsade populationer."
[1] "Försök 5 dog ut. Totalt 3 collapsade populationer."
[1] "Försök 8 dog ut. Totalt 4 collapsade populationer."
```

För att beräkna utdöenderisk i procent.

```
extiction_rate <- (pop_collapse / attempts) *100
print(paste("Utdöenderisken är",extiction_rate,"%"))
```

```
[1] "Utdöenderisken är 40 %"
```

4.5 Lesliematriser

För användning av en livstabell översätts den till en Leslie/Lefkovitch matris.

Livstabell för Caretta caretta

Stage #	Class	Size (cm)	Approx. Age	P(Survival)	P(Growth)	Fecundity
1	eggs, hatchlings	<10,0	<1	0	0,6747	0
2	small juveniles	10,1-58,0	1-7	0,737	0,0486	0
3	large juveniles	58,1-80,0	8-15	0,661	0,0147	0
4	subadults	80,0-87,0	16-21	0,6907	0,0518	0
5	novice breeders	>87,0	22	0	0,8091	127
6	1st-yr remigrants	>87,0	23	0	0,8091	4
7	mature breeders	>87,0	24-54	0,8091	0	80

En Leslie-matris använder åldersklasser där ingen kan "stanna" i klassen, alltså är diagonalen 0.

En Lefkovitch-matris använder ontogenetiska stadier där en individ kan antingen "stanna" i samma klass eller "gå vidare" till nästa klass, alltså kan diagonalen vara >0.

Leslie/Lefkovich-matrisen

Nästa år (t+1) Händer under året I år (t)

$$\begin{bmatrix} N_0(t+1) \\ N_1(t+1) \\ N_2(t+1) \\ N_3(t+1) \end{bmatrix} = \begin{bmatrix} s_0 & f_1 & f_2 & f_3 \\ g_0 & s_1 & 0 & 0 \\ 0 & g_1 & s_2 & 0 \\ 0 & 0 & g_2 & s_3 \end{bmatrix} \begin{bmatrix} N_0(t) \\ N_1(t) \\ N_2(t) \\ N_3(t) \end{bmatrix}$$

$$\begin{bmatrix} s_0 & f_1 & f_2 & f_3 \\ g_0 & s_1 & 0 & 0 \\ 0 & g_1 & s_2 & 0 \\ 0 & 0 & g_2 & s_3 \end{bmatrix}$$

s_i = P(överleva och stanna i samma klass)

g_i = P(överleva och tillväxa till nästa klass/ålder)

f_i = fekunditet I klassen/åldern

```
caretta_matris <- rbind( # En Lefkovitch matris
  r1 = c(c1 = 0, c2 = 0, c3 = 0, c4 = 0, c5 = 127, c6 = 4, c7 = 80),
  r2 = c(0.6747,  0.737,      0,      0,        0,      0,       0),
  r3 = c(     0, 0.0486,  0.661,      0,        0,      0,       0),
  r4 = c(     0,      0, 0.0147, 0.6907,        0,      0,       0),
  r5 = c(     0,      0,      0, 0.0518,        0,      0,       0),
  r6 = c(     0,      0,      0,      0,   0.8091,      0,       0),
  r7 = c(     0,      0,      0,      0,        0, 0.8091,  0.8091)
)
caretta_matris
```

```
       c1     c2     c3     c4       c5     c6      c7
r1 0.0000 0.0000 0.0000 0.0000 127.0000 4.0000 80.0000
r2 0.6747 0.7370 0.0000 0.0000   0.0000 0.0000  0.0000
r3 0.0000 0.0486 0.6610 0.0000   0.0000 0.0000  0.0000
r4 0.0000 0.0000 0.0147 0.6907   0.0000 0.0000  0.0000
r5 0.0000 0.0000 0.0000 0.0518   0.0000 0.0000  0.0000
r6 0.0000 0.0000 0.0000 0.0000   0.8091 0.0000  0.0000
r7 0.0000 0.0000 0.0000 0.0000   0.0000 0.8091  0.8091
```

För att få ut egenvärdet (r-värdet) och egenvektorn (stabil stadiefördelning). Om λ är 0.90 innebär det att populationen minskar med 10% (förändrings faktor för populationen). Den relativa fördelningen ($\frac{egenvektor}{sum(egenvektor)} \cdot 100$) indikerar hur många procent av den totala populationen bör finnas i varje klass. Kan användas som `p = relativ_dist` i `chisq.test()` för att göra ett anpassningstest med en observerad population.

```
caretta_eigen <- eigen(caretta_matris)

# Egentligen det största egenvärdet, men det bör vara första.
Re(caretta_eigen$values[1])
```

```
[1] 0.9450846
```

```
# Samma index för egenvektor som egenvärdet.
Re(caretta_eigen$vectors[,1])
```

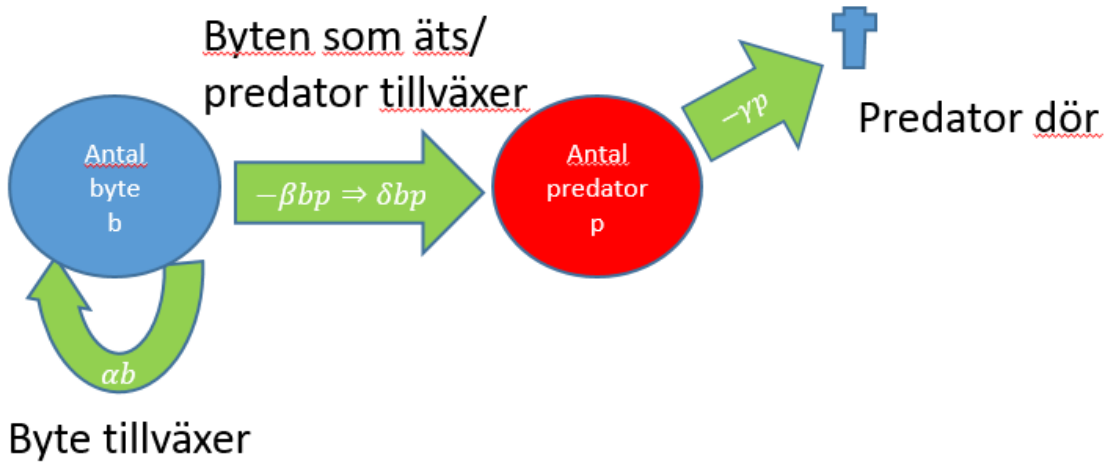
```
[1] 0.2908382596 0.9430228664 0.1613283678 0.0093226030 0.0005109710
[6] 0.0004374493 0.0026027957
```

```
# Relativ fördelning i procent
caretta_dist <- Re(caretta_eigen$vectors[,1])
caretta_dist_procent <- (caretta_dist/sum(caretta_dist))*100
caretta_dist_procent
```

```
[1] 20.65519760 66.97304431 11.45746547  0.66208692  0.03628892  0.03106745
[7]  0.18484934
```

4.6 Lotka-Volterra byte-predator modell

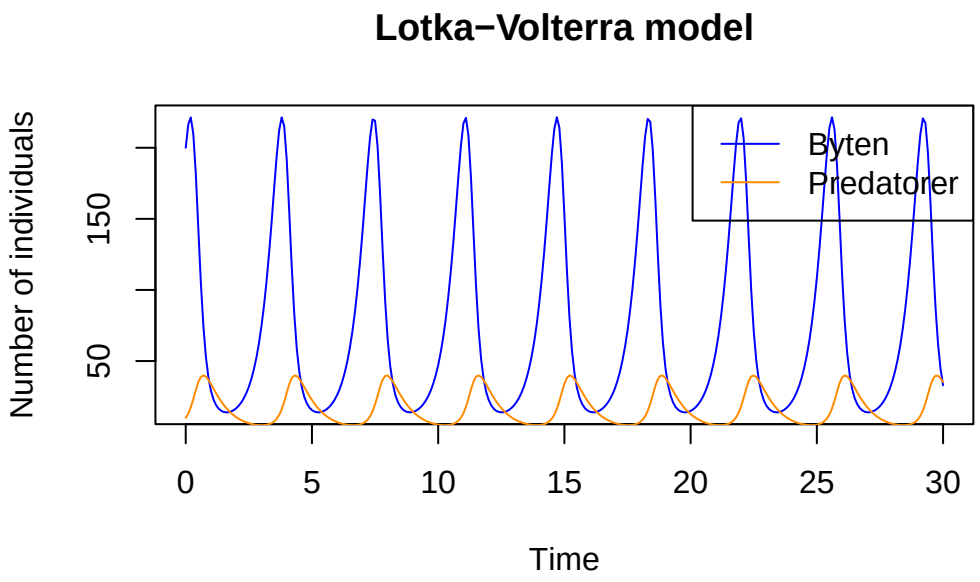
En specifik model som bygger på att lösa ett ekvationssystem av differentialekvationer.



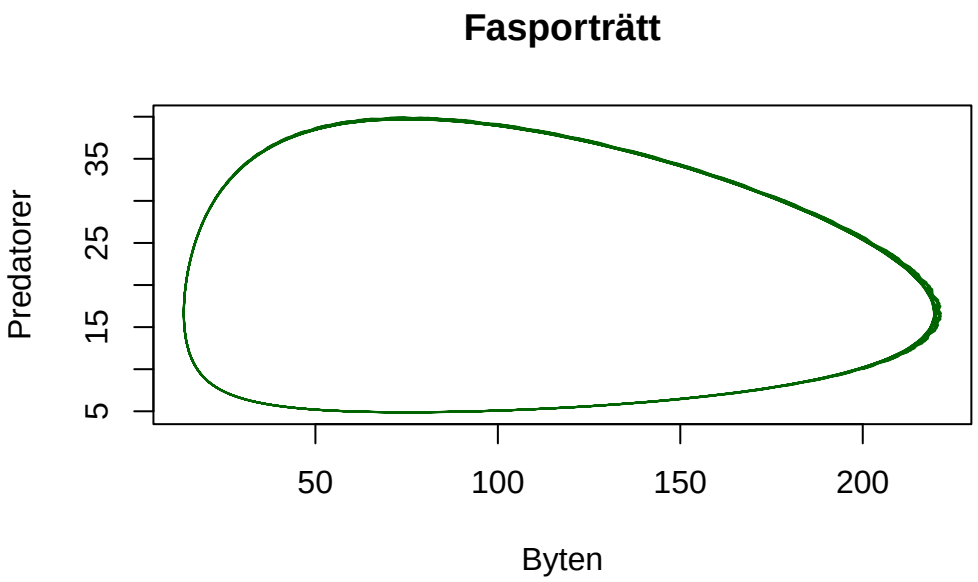

```r
# Funktion med ekvationssystemet att lösa
LV_fun <- function(times, # Intervall
                   y,     # Begynnelsevärden
                   parms  # Andra värden
) {
  # variabler att använda (från parms)
  alfa <- parms["alfa"]
  beta <- parms["beta"]
  delta <- parms["delta"]
  gamma <- parms["gamma"]
#  K <- parms["K"] # alt. där K = bärkraft

  # Begynnelsevärden (från y)
  B <- y["B"]
  P <- y["P"]

  # Differentialekvationen att lösa numeriskt
  dBdt <- (alfa * B - beta * B * P)
#  dBdt <- (alpha * B(1 - B/K) - beta * B * P) # alt. där K = bärkraft

  dPdt <- (delta * B * P - gamma * P)

  # Spara & returnera resultatet
  result_vec <- c(
    dBdt,
    dPdt
```

```r
  )
  result_list <- list(result_vec)
  return(result_list)
}

# Vektor med alla tidssteg (t) att sätta in
time_span_LV <- seq(
  0,
  30,
  by = 0.1
)

# Vektor med begynnelsevärden
init_LV <- c(
  B = 200,
  P = 10
)

# Vektor med andra värden för formeln
params_LV <- c(
  alfa = 2.5,    # Tillväxt byten
  beta = 0.15,   # Byten som äts
  delta = 0.02,  # Tillväxt predator
  gamma = 1.5    # Död predator
#  K = 500 # alt. där K = bärkraft
)

# Lösa ekvationerna numeriskt
sol_LV <- ode(
  y = init_LV,           # Begynnelsevärden
  times = time_span_LV,  # Intervall
  func = LV_fun,         # Funktionen
  parms = params_LV,     # Andra värden
  method = "rk4"         # Runge-Kutta version 4
)
sol_LV <- as.data.frame(sol_LV)


# Plotta antal insekter [miljoner] respektive spindlar [tusen] mot tid
plot(
  sol_LV$time,
```

```
  sol_LV$B,
  type = "l",

  col = "blue",

  xlab = "Time",
  ylab = "Number of individuals",
  main = "Lotka-Volterra model"
)
lines(
  sol_LV$time,
  sol_LV$P,

  col = "darkorange"
)
legend(
  "topright",
  legend = c("Byten",
             "Predatorer"),

  lty = c(1, 1),
  col = c("blue",
          "darkorange")
)
```




```r
# "Fasporträtt" (två pop. mot varandra där tiden blir att följa linjen)
plot(
  sol_LV$B,
  sol_LV$P,
  type = "l",

  col = "darkgreen",

  xlab = "Byten",
  ylab = "Predatorer",
  main = "Fasporträtt"
)
```


Fasporträtt
Predatorer
5
15
25
35
50
100
150
200
Byten