LECTURE 2

Ege Rubak (based on Jakob G. Rasmussen's material)

PART 1: packages and programming

If R does not contain the functions/statistics you need, odds are that somebody has implemented it in a package. Installing a package - here a package for handling the multivariate normal distribution

```
install.packages("mvtnorm")  # only need to install it once
```

Loading a package

```
library(mvtnorm)  # need to do this everytime R is started
```

In Rstudio you can also use the Packages Window to the right.

Now the following functions work for calculating the density of and simulating multivariate normal distributions

```
dmvnorm(c(1,2,2), mean = rep(0,3), sigma = diag(3))  # evaluating the 3-dim standard normal density
```

```
## [1] 0.0007053506
```

```
rmvnorm(5, mean = rep(0,3), sigma = diag(3)) # every row is a simulation
```

```
##            [,1]       [,2]       [,3]
## [1,] -2.0188852 -0.4863255  1.6342043
## [2,]  0.8269136 -2.6533095  0.6419060
## [3,]  0.5657087 -0.5394127 -0.7620711
## [4,]  0.8248615 -0.8188777 -1.6376881
## [5,]  0.9128800  0.4471728  0.4731925
```

If you don't know what a package contains, you can try

```
library(help = "mvtnorm")
```

Another useful package - rmarkdown - used to make reports/slides with text/math/R-output fast

```
# install.packages("rmarkdown")  # takes a minute or so, so I won't run it now
```

It is possible to use a function from a package with out loading the package first via `library`. This is done using the :: notation like this:

```
rmarkdown::render("markdown_example.Rmd")
```

There are thousands of other packages for specific needs.

Google is a good way of finding out whether there is a package that suits your need. You can also make packages yourself, but we won't go into this. If there is nothing premade in R or any packages, you will need to program it yourself.

For-loop

Calculating 1+2+. . . +10 as an example

```r
s = 0
for (i in 1:10){
  s = s + i
}  # any vector can be used instead of 1:10
s
```

```
## [1] 55
```

Calculating the first ten Fibonacci numbers

```r
f = rep(0,10)
f[1] = f[2] = 1
for (i in 3:10){
  f[i] = f[i-2]+f[i-1]
}
f
```

```
##  [1]  1  1  2  3  5  8 13 21 34 55
```

Note that built-in functions are usually faster than for-loops created from scratch

If-then-else conditions

Determining the sign of a number

```r
x = -3
if (x<0) {
  signx = -1
} else{
  if (x==0){
    signx = 0
  } else{
    signx = 1
  }
}
signx
```

```
## [1] -1
```

Functions

A function for finding the sign of a number

```r
signfct = function(x){  # notation: output = function(input1,input2,...){blablabla}
  signx <- 0 # Assume 0 until found otherwise
  if (x<0) {
    signx <- -1
  }
  if (x>0){
    signx <- 1
  }
  return(signx)
}
signfct(-3);signfct(0);signfct(0.2)
```

```
## [1] -1
```

```
## [1] 0
```

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

There is a built-in function `sign`

```r
sign(-3);sign(0);sign(0.2)
```

```
## [1] -1
```

```
## [1] 0
```

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

```r
sign(-3:4)  # this will even take vectors or matrices
```

```
## [1] -1 -1 -1  0  1  1  1  1
```

```r
signfct(-3:4)  # our function is not that smart, due to the if-condition only accepting a single term
```

```
## Warning in if (x < 0) {: the condition has length > 1 and only the first
## element will be used
```

```
## Warning in if (x > 0) {: the condition has length > 1 and only the first
## element will be used
```

```
## [1] -1
```

Morale: always think about all the types of input you would like to have and try them out.

PART 1 exercises

I) Make a function with a for loop that can calculate the product of all the entries in an input vector. Compare with the built-in function `prod` (don't call your function prod, or you won't be able to use the built-in function easily).

II) Make a function that will calculate the Fibonacci number up to n (an input parameter).

- Does it handle n=1 or 2 correctly? (hint: an `if` statement may be useful here)
- Does it handle negative numbers correctly? (hint: the `stop` function can be used to give an error message)
- Does it handle decimal numbers correctly?

III) Install and load the package `ggplot2` for creating nice plots in R. Look at the package help and experiment a bit. To get an idea of the possibilities search for ggplot2 at https://images.google.com. This package is part of a collection of packages called the `tidyverse` which are very powerful for data manipulation and graphics.

PART 2: overview of statistical analysis, linear models, and regression

We will use the lecture slides (made with rmarkdown) for this part.

PART 2 exercises

I) Consider the built-in dataset `cars`

a) Make the design matrix X for a simple linear regression for `cars` (`dist` as a function of `speed`).
b) Estimate beta. Plot the data and the estimated line in the same figure. (hint: the function `abline` is useful for plotting the line)
c) Estimate sigma^2.

II) Maybe a second order polynomial is better at capturing the relation between speed and distance? Redo exercise I with a second order polynomial. (Hint: `curve` may be useful)

III) Consider the dataset `trees`

a) Make the design matrix for a multiple regression model for modelling the tree volume as a function of girth and height.
b) Estimate the vector of coefficients beta and the variance sigma^2.

PART 3: the lm-function and ANOVA kind of models

Obviously linear models are implemented in R - we use the `lm` function

```
mod1 <- lm(dist ~ speed, data = cars); mod1
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Coefficients:
## (Intercept)        speed
##     -17.579        3.932
```

`y~x` is R formula language for $y = \beta_0 + \beta_1 x + \epsilon$ I.e. ignore the constand term, the parameters, and the error term. If we don't want the constant term, we can write y~-1+x The `lm` function creates an lm-class object with lots of content

```
class(mod1)
```

```
## [1] "lm"
```

```
names(mod1)
```

```
##  [1] "coefficients" "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"       "qr"            "df.residual"
##  [9] "xlevels"       "call"         "terms"         "model"
```

```
summary(mod1)  # the estimate for beta and sigma can be found here
```

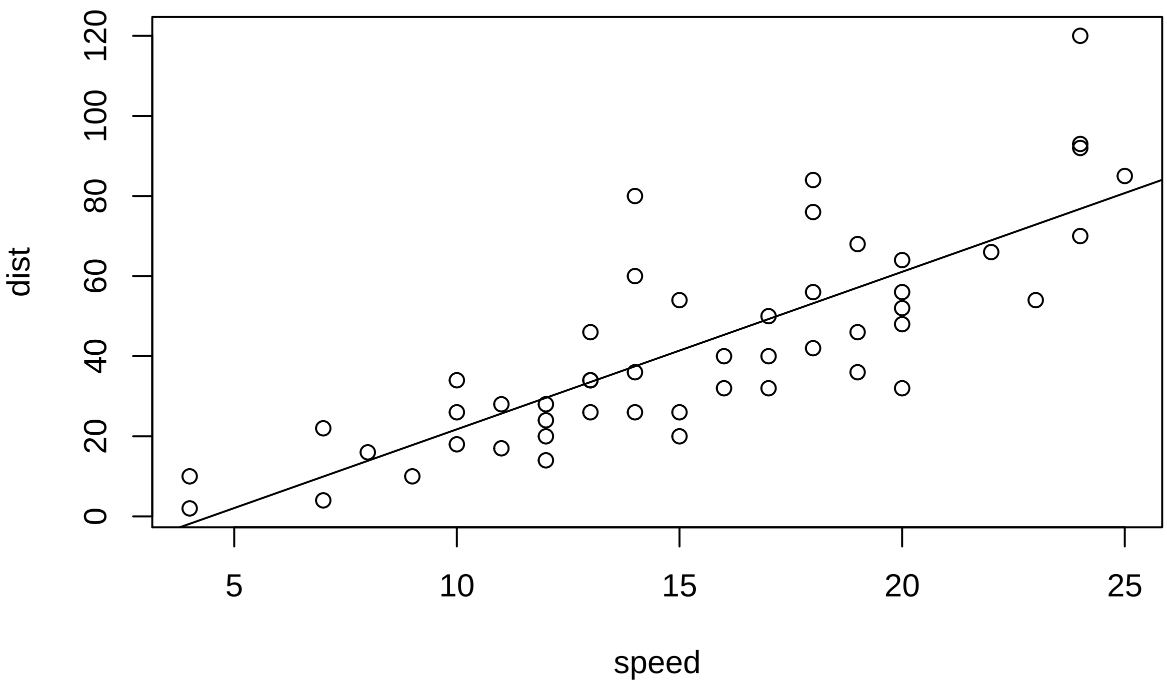
```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Plotting the estimated model

```
beta_hat1 = coef(mod1); beta_hat1 # we extract the estimates of beta into a vector (with names)
```

```
## (Intercept)       speed
##  -17.579095    3.932409
```

```
plot(dist ~ speed, data = cars)
abline(beta_hat1)
```



Functions of the x-variables

If we want the second order polynomial, we should be careful: `+`, `-`, `*`, `^` have special meanings in the R formula language. If we enclose terms involving these in `I()`, they have their usual meaning

```
lm(Volume ~ Girth + Height, data = trees) # y = beta0 + beta1*x1 + beta2*x2 + epsilon
```

```
##
## Call:
## lm(formula = Volume ~ Girth + Height, data = trees)
##
## Coefficients:
## (Intercept)        Girth       Height
##    -57.9877       4.7082       0.3393
```

```
lm(Volume ~ I(Girth + Height), data = trees) # y = beta0 + beta1*(x1+x2) + epsilon
```

```
##
## Call:
## lm(formula = Volume ~ I(Girth + Height), data = trees)
##
## Coefficients:
##       (Intercept)  I(Girth + Height)
```

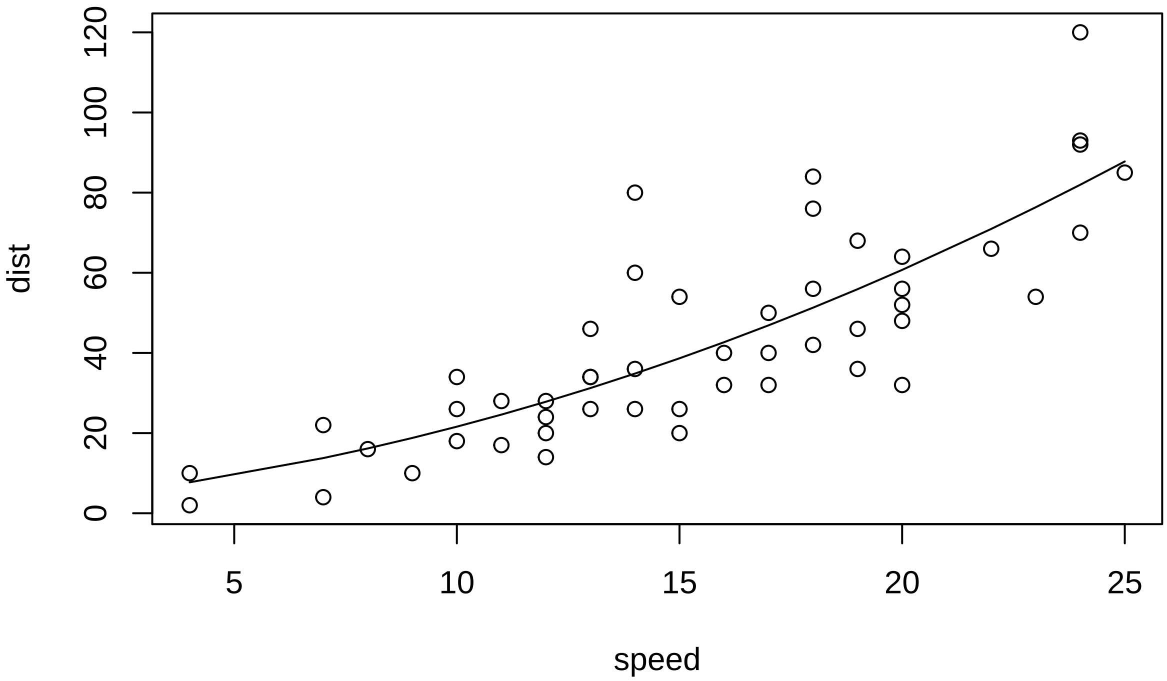
```
##          -110.88               1.58
```

Second order polynomial used to model `cars`

```r
mod2 = lm(dist ~ speed + I(speed^2), data = cars); mod2
```

```
##
## Call:
## lm(formula = dist ~ speed + I(speed^2), data = cars)
##
## Coefficients:
## (Intercept)        speed   I(speed^2)
##     2.47014      0.91329      0.09996
```

```r
beta_hat2 = coef(mod2)
plot(dist ~ speed, data = cars)
lines(fitted(mod2) ~ speed, data = cars)
```



Categorical variables

So far we have considered the x variables as continuous/numeric. What if they are categorical, i.e. represent groups? This is considered in the lecture slides and afterwards we continue with the material below. We make a one-way ANOVA to compare different kinds of insect sprays

```r
# ?InsectSprays
head(InsectSprays)
```

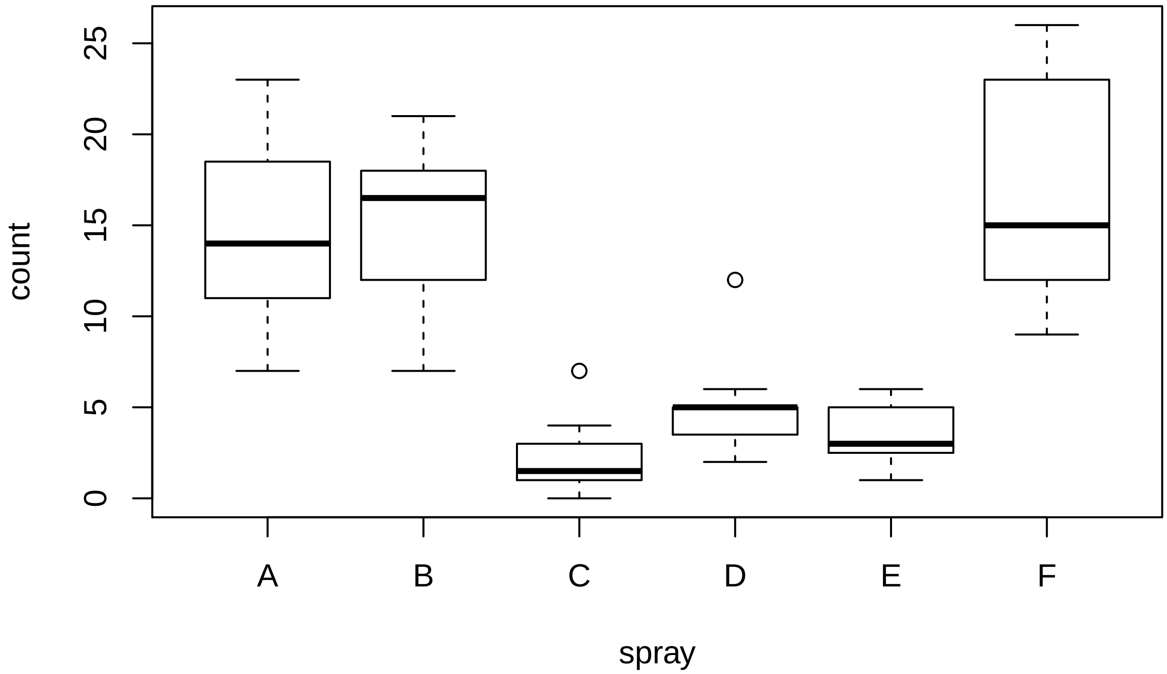
```
##   count spray
## 1    10     A
## 2     7     A
## 3    20     A
## 4    14     A
## 5    14     A
## 6    12     A
```

```r
class(InsectSprays$count); class(InsectSprays$spray)
```

```
## [1] "numeric"
```

```
## [1] "factor"
```

```r
plot(count ~ spray, dat = InsectSprays)
```




```r
mod3 <- lm(count ~ spray, data = InsectSprays); mod3  # type A is the reference group here
```

```
##
## Call:
## lm(formula = count ~ spray, data = InsectSprays)
##
## Coefficients:
## (Intercept)       sprayB       sprayC       sprayD       sprayE
##     14.5000       0.8333     -12.4167      -9.5833     -11.0000
##      sprayF
##      2.1667
```

```r
summary(mod3)
```

```
##
## Call:
## lm(formula = count ~ spray, data = InsectSprays)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.333 -1.958 -0.500  1.667  9.333
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    14.5000     1.1322  12.807  < 2e-16 ***
## sprayB          0.8333     1.6011   0.520    0.604
```

```
## sprayC      -12.4167     1.6011  -7.755 7.27e-11 ***
## sprayD       -9.5833     1.6011  -5.985 9.82e-08 ***
## sprayE      -11.0000     1.6011  -6.870 2.75e-09 ***
## sprayF        2.1667     1.6011   1.353    0.181
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.922 on 66 degrees of freedom
## Multiple R-squared:  0.7244, Adjusted R-squared:  0.7036
## F-statistic:  34.7 on 5 and 66 DF,  p-value: < 2.2e-16
```

A two-way ANOVA (i.e. two factors)

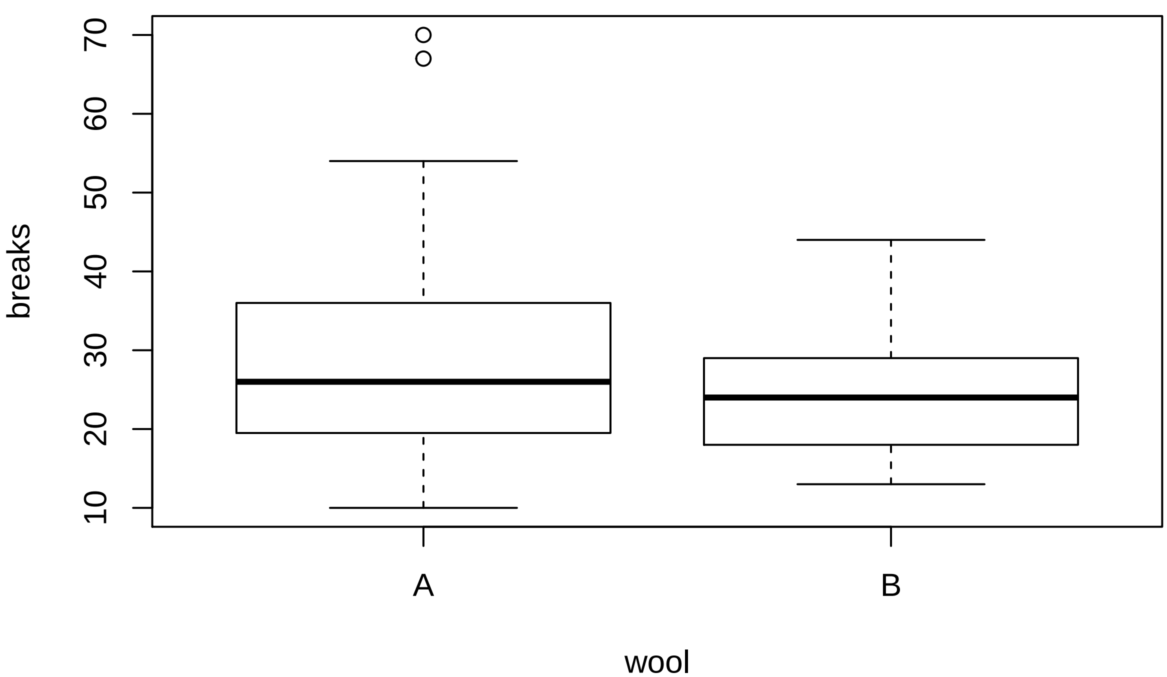
```r
# ?warpbreaks
head(warpbreaks)
```

```
##   breaks wool tension
## 1     26    A       L
## 2     30    A       L
## 3     54    A       L
## 4     25    A       L
## 5     70    A       L
## 6     52    A       L
```

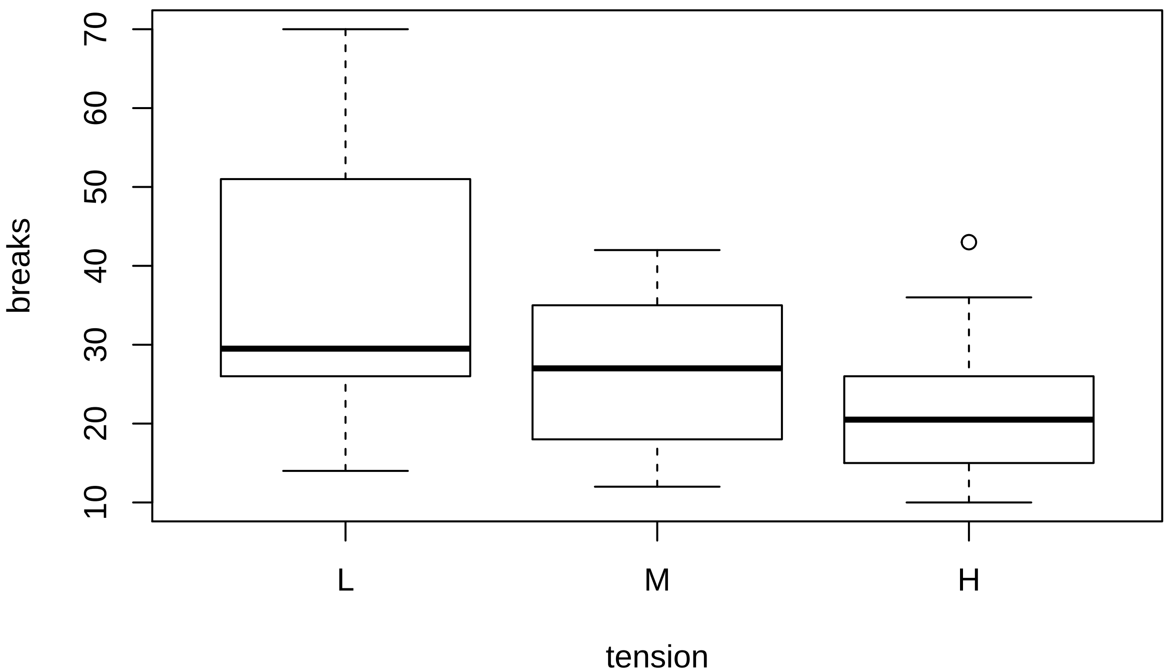
```r
table(warpbreaks[,c("wool", "tension")]) # a quick overview of the number of combinations
```

```
##     tension
## wool L M H
##    A 9 9 9
##    B 9 9 9
```

```r
plot(breaks ~ wool, data = warpbreaks)  # breaks vs wool type
```



```r
plot(breaks ~ tension, data = warpbreaks)  # breaks vs tension
```




```r
lm(breaks ~ wool + tension, data = warpbreaks)  # wool A and tension L are reference groups
```

```
##
## Call:
## lm(formula = breaks ~ wool + tension, data = warpbreaks)
##
## Coefficients:
## (Intercept)        woolB     tensionM     tensionH
##      39.278       -5.778      -10.000      -14.722
```

```r
lm(breaks ~ wool + tension + wool:tension, data = warpbreaks)  # wool*tension is interaction
```

```
##
## Call:
## lm(formula = breaks ~ wool + tension + wool:tension, data = warpbreaks)
##
## Coefficients:
##    (Intercept)            woolB         tensionM         tensionH
##          44.56           -16.33           -20.56           -20.00
## woolB:tensionM  woolB:tensionH
##          21.11            10.56
```

The model without interaction means that wool and tension have separate additive effects Interaction means that different types of wool have different behavior depending on tension

To sum up: there are a lot of different terms that can go into a linear model as x. All types can be combined.

PART 3 exercises

I) Consider the data `ToothGrowth`. The data contains two explanatory variables, a factor `supp` and a numeric variable `dose` (it only contains 3 different values so it could also be sensible to convert it to a factor of e.g. low, medium and high dose, but we will treat it as numeric here). We start by ignoring the factor `supp`.

   a) Plot the data (only `len` and `dose`).
   b) Model the relation between `len` and `dose` with a simple linear regression, and add the estimated line to the plot.
   c) Try a second order polynomium, and add the curve to the plot. Does it seem to fit better?
   D) And a third order polynomium. What happens here?

II) Now ignore `dose` in `ToothGrowth`.
   a) Plot the data (`len` vs `supp`).
   b) Model the relation with an ANOVA kind of model. Does it seem that `OJ` or `VC` yields the highest values for len?

III) Now all the data.
   a) Plot all the data. (hint: try to plot `len` vs `dose` with `col=supp` to get different colors for each group)
   b) Make a model with both explanantory variable (only use a first order polynomial for dose). The model has the interpretation that we have different lines depending on the type in `supp`. They have the same slope (beta_dose). But different intercepts (beta_intercept or beta_intercept+beta_suppVC). Add both lines to the plot.
   c) Include an interaction term. Interaction between a continuous and categorical variable is simple: The lines can now have different slopes (beta_intercept or beta_intercept+beta_dose:suppVC) Include the lines in the plot (you may want to make a new plot).