Practical examples to the course of Statistical Analysis

Anna Melnykova

Abstract

This file serves as a complement and an illustration to the theoretical course on Statistical Analysis. It is by no mean complete and may contain errors, misprints and minor inaccuracies. You are encouraged to ask your questions and send you coments to my mail anna.melnykova@univ-grenoble-alpes.fr, so that you will help to improve this document for the future generations and leave your trace in history.

Session 1. Simple Linear regression

Let us first recall the setting: we observe two random variables X and Y, and assume that they are linked by a linear relation, defined as follows:

$$Y = \beta_1 X + \beta_0 + \varepsilon, \tag{1}$$

where ε is an error of the measurements (of unknown distribution), Y is a dependent variable, X is an explanatory variable. β_1 and β_0 are unknown parameters of the model, which we will try to estimate.

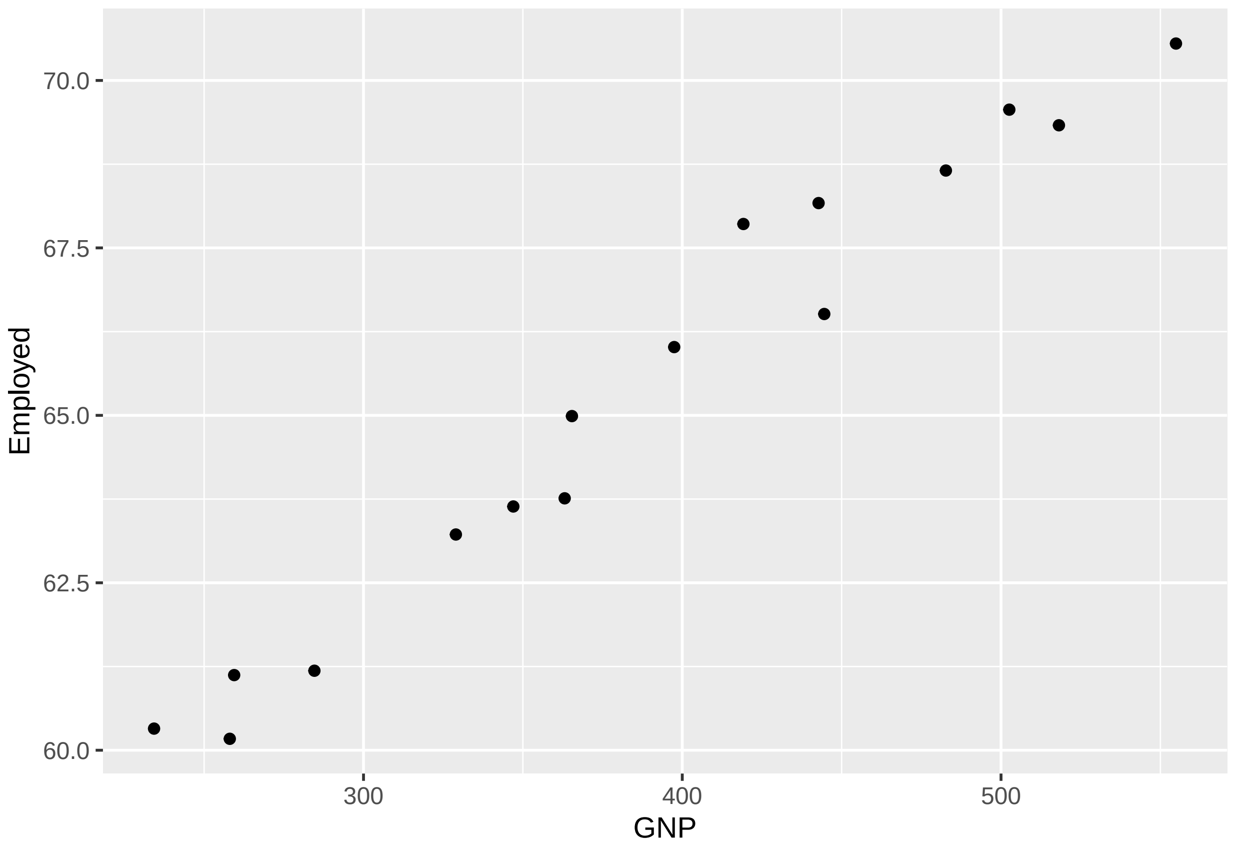
We start our experiment with setting up the environment and load the data. First, let us load some libraries which we will use. The first one is the set of standard datasets, the second one allows to create nice plots (but not only!). If the last library are not installed on your machine, you can use the command `install.packages("ggplot2")` in your R console prior to executing the next chunk of code.

In this example we will use the data from `longley` dataset, which contains economic data about the employment, unemployment, population, GNP and so on of the US population in years 1947-1962.

```r
library(datasets)
library(ggplot2)
data("longley")
gnp <- longley$GNP # That will be our X
emp <- longley$Employed # That will be our Y
```

Let us visualize the variables of interest. Without going into details of the `ggplot` function, note that as a first parameter we give the dataset loaded in our workspace by a command `data("longley")`, and then we use the names of variables (not the exctracted vectors!) to build the scatter plot (that's what the command `geom_point()` is doing).

```r
scat_plot <- ggplot(longley, aes(GNP, Employed))+geom_point()
scat_plot
```



We see a clear linear dependency between the GNP and the employed population. That is, we assume that we can express each observation of Y as a linear function of X, written as follows:

$$y_i = \beta_1 x_i + \beta_0 + \varepsilon_i,$$

where $i = 1, \ldots, n$, ε_i are independent indentically distributed error of measurements. Our aim is now to fit the regression line which explains the link between the variables. Recall that the natural estimators of the unknown parameters are such that they minimize the mean square error (i.e., distance between y_i and its "predicted" value $\beta_1 x_i + \beta_0$). Thus, the estimators are defined as follows:

$$\hat{\beta}_0, \hat{\beta}_1 = \underset{\beta_0, \beta_1}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_1 x_i - \beta_0)^2 .$$

Then, following the derivation which we have seen in the course, the point estimate of parameters β_1 (slope) and β_0 (intercept), are given by the following formula:

$$\hat{\beta}_1 = \frac{c_{xy}}{s_x^2}, \qquad \hat{\beta}_0 = \bar{y} + \frac{c_{xy}}{s_x^2}\bar{x}.$$

where c_{xy} denotes the empirical covariance between the vectors x and y, and s_x^2 is the empirical variance, $\bar{x}$ and $\bar{y}$ are empirical means of x and y respectively.

Important note: don't forget that the estimators $\hat{\beta}_1$ and $\hat{\beta}_0$ are, in fact, random variables, and for another realization of X and Y we will have slightly different values! What is important to know that those estimators are consistent (proof will be given in the course):

$$\mathbb{E}[\hat{\beta}_1] = \beta_1 \qquad\qquad \mathbb{E}[\hat{\beta}_0] = \beta_0.$$

For computing the value of the estimators in our specific case, you can use the following pre-computed values:

```
mean(gnp); var(gnp)
```

```
## [1] 387.6984

## [1] 9879.354
```

```
mean(emp); var(emp)
```

```
## [1] 65.317

## [1] 12.33392
```

```
cov(gnp, emp)
```

```
## [1] 343.3302
```

```
cor(gnp, emp)
```

```
## [1] 0.9835516
```

Note that the correclation coefficient is close to 1, which clearly indicates the existing linear dependency between the variables. Please note, however, that the linear correlation in data does not imply that there is an evidence of two events being dependent on the other, or that one thing causes another! In our example, if we compute the correlation coefficients between each pair of variables in a dataset, we could come to conclusion that the increase of GNP can be also explained by a total population grows, or that a population growth can be traced back to the increase of employed people, or that all those factors depend linearly on the year we are in (which is not true, of course!). More funny plots and spurious correlations can be found on this website: https://www.tylervigen.com/spurious-correlations

But let's go back to the point and check if your computations were correct!

```
b1 <- cov(gnp, emp)/var(gnp); b1
```

```
## [1] 0.03475229
```

```
b0 <- mean(emp) - b1*mean(gnp); b0
```

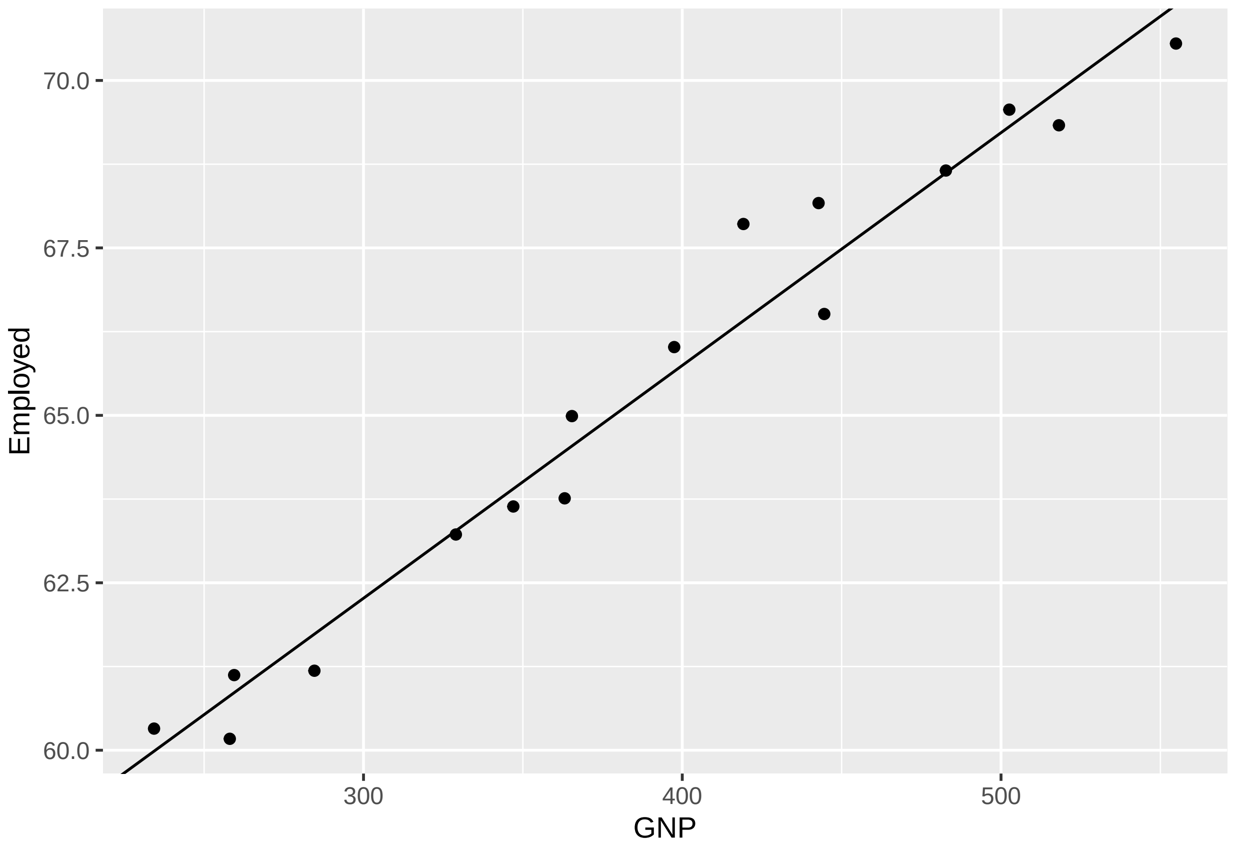
```
## [1] 51.84359
```

Now, we will check if it corresponds to the result of the built-in function in R (function `lm`, which stands for *linear model*), and plot the obtained line!

```
model_fit <- lm(Employed~GNP, data = longley)
model_fit
```

```
##
## Call:
## lm(formula = Employed ~ GNP, data = longley)
##
## Coefficients:
## (Intercept)          GNP
##    51.84359      0.03475
```

```
scat_plot + geom_abline(intercept = b0, slope = b1)
```

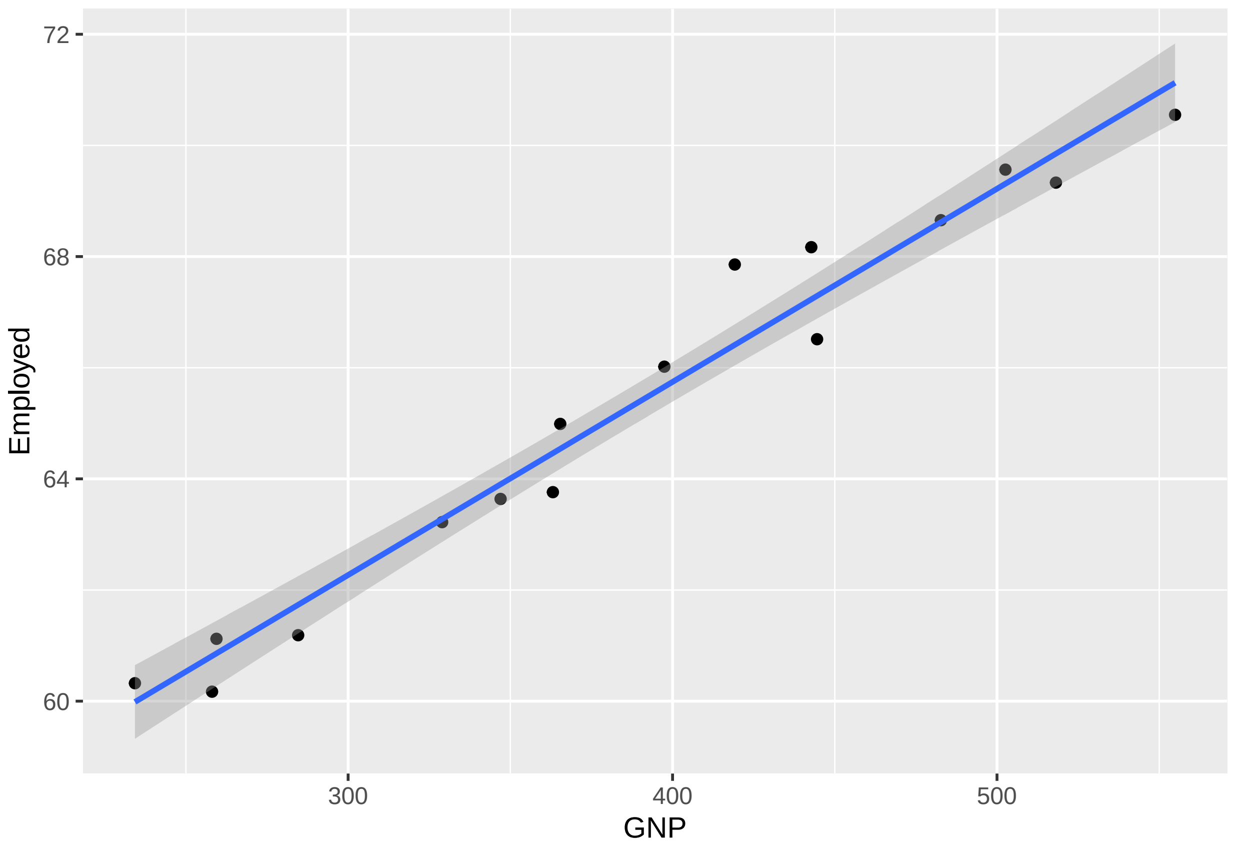


Feel free to execute the commands on your machine and have a look on all the information encoded in the variable `model_fit` now. In particular, you will find there a vector of residuals (`model_fit$residuals`), i.e. error of "approximation" of each observed value by the line, the respective fitted values (`model_fit$fitted_values`) and other interesting things. More generally, if you type `?lm` in your console, you will find all the info about the parameters which this function take, the output and so on.

To finish with, let me show how the same result can be obtained directly with the built-in methods of `ggplot`. Note that on the plot below, the built-in command is "smarter" and not only gives the point estimates of β_0 and β_1, but a confidence interval as well (of 95%, if I am not mistaken), which you see as a grey region on the plot. Finally, in order to obtain the confidence intervals (that's something we are not going to do right now), one should have more information about the distribution of the estimators $\hat{\beta}_0, \hat{\beta}_1$. These properties are linked to the properties of ε_i (for example, if we assume the error to be gaussian, the estimators would be normal). That will be a subject of further lectures.

```
# scat_plot + stat_smooth() # The first function is commented out as we don't really want to talk about
scat_plot + stat_smooth(method = lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```


72
68
64
60
Employed
300
400
500
GNP

Session 2: Properties of the estimators. Gaussian linear regression.

Now, our aim will be to test the properties of the least squares estimator. For the moment, we stick to the same model (1), but we put an additional assumption on the error term:

$$\varepsilon \sim N(0, \sigma^2).$$

With this additional assumption at hand, we can conclude that $\forall i$

$$y_i - \beta_1 x_i - \beta_0 \sim N(0, \sigma^2). \tag{2}$$

Then, we can propose an analogous way to estimate the parameters of the system, by using a maximum likelihood estimator. We can write the likelihood function as follows:

$$L(y; \beta_0, \beta_1, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0)^2\right)$$

If we take a logarithm of it, we will obtain the following:

$$\log L(y; \beta_0, \beta_1, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0)^2$$

Then, the natural estimators of the parameters can be written as follows:

$$\hat\beta_0, \hat\beta_1 = \arg\min_{\beta_0,\beta_1} \sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0)^2, \qquad \hat\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0)^2$$

Note that the MLE for Gaussian noise coincides with LSE!

Now, we will simulate several samples of X and Y with a chosen set of parameters. On each sample, we will compute $\hat\beta_1$ and $\hat\beta_1$ and see how they are distributed.

```r
n = 25 # number of observations in each sample
N = 1000 # number of samples
beta0 = 1 # intercept
beta1 = 2 # slope
sigma = sqrt(1/3) # standard deviation of the noise term

XY_list <- list() # list to store the simulated couples X, Y of n observations each
for (i in 1:N){
  X <- runif(n)
  Y <- beta1*X + beta0+rnorm(n, mean = 0, sd = sigma)
  XY_list[[i]] <- as.data.frame(cbind(X,Y))
}

par_set <- matrix(nrow = N, ncol = 3) # matrix to store the estimated parameters
for (i in 1:N){
  X <- XY_list[[i]][,1]
  Y <- XY_list[[i]][,2]
  b1 <- cov(X, Y)/var(X)
  b0 <- mean(Y) - b1*mean(X)
  sigma2 <- var(Y)
  par_set[i,] <- c(b0, b1, sigma2)
}
```

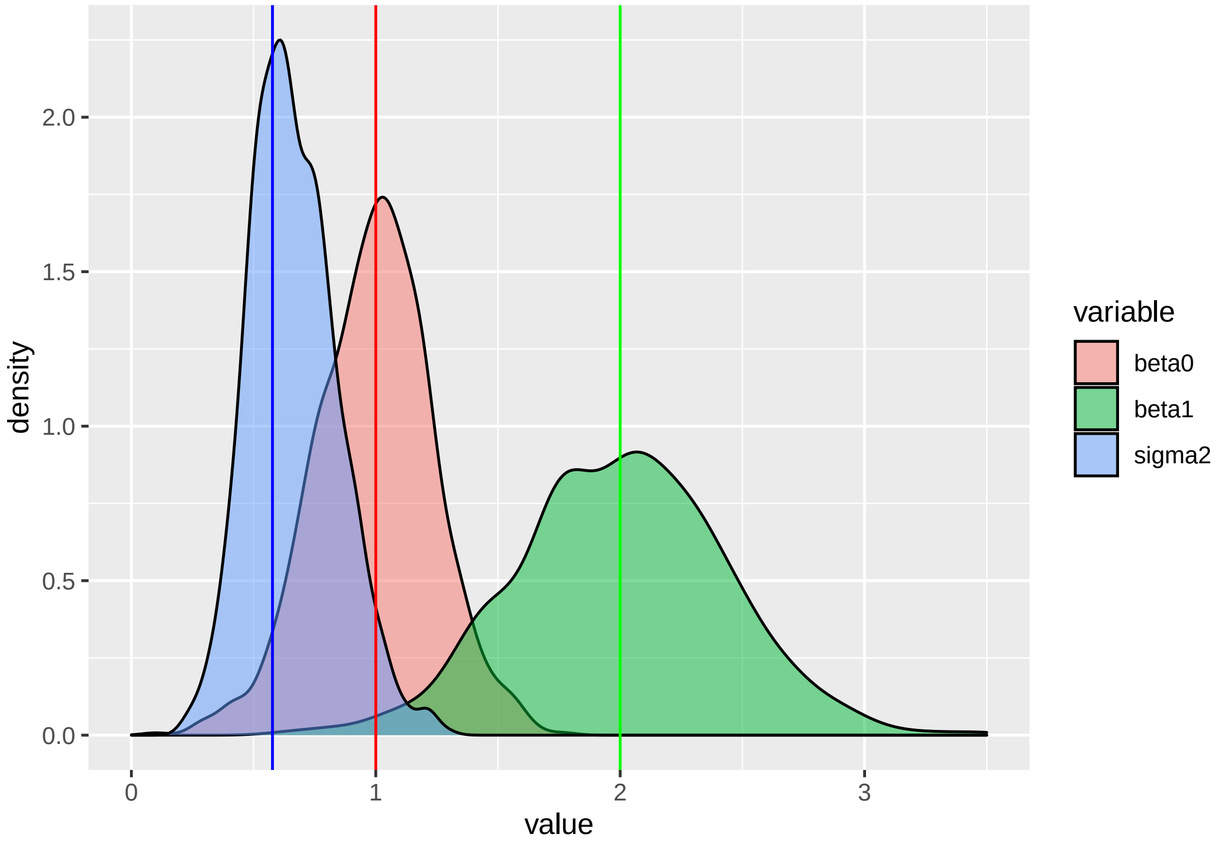
Now, let us try to build the plots of each parameter and see how they are distributed. Here, we do a small trick to convert a data frame with 3 columns into a data frame with 2 columns, which lets us build 3 densities on the same plot with only one command:

```
library(reshape2)
ds_pars <- as.data.frame(par_set)
colnames(ds_pars) <- c("beta0", "beta1", "sigma2")
plot_pars <- ggplot(melt(ds_pars), aes(x=value, fill = variable))+geom_density(alpha = 0.5)+xlim(0,3.5)
```

```
## No id variables; using all as measure variables
```

```
plot_pars
```

```
## Warning: Removed 1 rows containing non-finite values (stat_density).
```



It is easy to see that the estimator is more-or-less normally distributed around its mean value (vertical line of the respective color). You can execute the same code on your machine and try to see what changes if you change the number of observations in each sample `n` and the number of samples `N`.

Now, let us think about the following: we have demonstrated that the Maximum Likelihood Estimator in a Gaussian case coincides with the least squares estimator. So, what happens if we drop the Gaussian assumption and study the distribution of the estimators? Will they still be normal? Let's check it empirically. You can repeat the same experiment with the uniformly distributed error terms, i.e. now we assume that $\forall i, \quad \varepsilon_i \sim U([-1,1])$. Note that the variance of ε_i is $1/3$, so that we should have more or less the same picture. Relaunch this code and comment the result:

```
XY_list <- list() # list to store the simulated couples X, Y of n observations each
for (i in 1:N){
  X <- runif(n)
  Y <- beta1*X + beta0+runif(n, min = -1, max = 1)
  XY_list[[i]] <- as.data.frame(cbind(X,Y))
}
```

```r
par_set <- matrix(nrow = N, ncol = 3) # matrix to store the estimated parameters
for (i in 1:N){
  X <- XY_list[[i]][,1]
  Y <- XY_list[[i]][,2]
  b1 <- cov(X, Y)/var(X)
  b0 <- mean(Y) - b1*mean(X)
  sigma2 <- var(Y)
  par_set[i,] <- c(b0, b1, sigma2)
}
ds_pars <- as.data.frame(par_set)
colnames(ds_pars) <- c("beta0", "beta1", "sigma2")
plot_pars <- ggplot(melt(ds_pars), aes(x=value, fill = variable))+geom_density(alpha = 0.5)+xlim(0,3.5)
plot_pars
```

Pss… You are probably wondering how did I manage to insert the code, but not show the result? When writing your Rmd reports you can do the same trick by setting a parameter `eval = FALSE` in the chunk. If you want to show only the result, but to hide the code, you can write `echo = FALSE` (see the source code for more details).

Statistical test for the regression coefficients

Why is it pleasant to work under the assumption of the normal distribution? Short answer: because it helps us to quantify, if it's even worth the effort to consider a given regression coefficient. It may not be so obvious for a simple model we were working on (one explanatory variable), so let us move on somewhat more complicated example.

But first, let us recall the following: under the Gaussianity assumption, all residuals are normally distributed (see (2)). Thanks to that (not immediately, but you have probably seen it in the course), the following holds:

$$\hat{\beta}_0 \sim N\left(\beta_0, \frac{\sigma^2}{n}\left(1+\frac{\bar{x}_n^2}{s_x^2}\right)\right), \qquad \hat{\beta}_1 \sim N\left(\beta_1, \frac{\sigma^2}{ns_x^2}\right), \qquad \frac{(n-2)\hat{\sigma}^2}{\sigma^2} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0)^2 \sim \chi^2_{n-2}.$$

Now, imagine that you have computed your estimator for $\hat{\beta}_0$ and has found out that it is close to 0. How should you know if this coefficient is negligible or not? What you can do is to do the following statistical test:

$$\begin{aligned} H_0 :& \quad \beta_0 = 0 \\ H_1 :& \quad \beta_0 \neq 0. \end{aligned}$$

Your statistics for β_0 is, naturally, the obtained *value* of $\hat{\beta}_0$, obtained from your data sample. You have already guessed that under null hypothesis, the following holds:

$$\frac{n\hat{\beta}_0}{\sigma^2\left(1+\frac{\bar{x}_n^2}{s_x^2}\right)} \sim N(0,1).$$

But, there is a tiny thing: you don't know the value of σ^2! No worries, you can always replace its value by the estimator defined above. As a result, your statistics will transform from a normal law to a Student law (because that's exactly what happens if you divide a normal variable by a chi-squared variable). In other words,

$$\frac{\sqrt{n}\hat{\beta}_0}{\hat{\sigma}\left(1+\frac{\bar{x}_n^2}{s_x^2}\right)} \sim St(n-2).$$

Then, what remains to do is to compute the p-value of the test (reminder: for a two-sided test in our case it is defined as $P(\xi > |t|)$, if $\xi \sim St(n-2)$) and decide: if the p-value is very small, that means that it is highly unlikely that $\beta_0 = 0$, so we have to reject the null hypothesis. Spoiler: R is doing all these manipulations for you without any additional hassle, so you don't have to re-do the computations by hand each time.

Now, let us put our knowledge in practice and consider the following example. I have taken it from http://www.sthda.com/english/articles/40-regression-analysis/162-nonlinear-regression-essentials-in-r-polynomial-and-spline-regression-models/. Here we will consider the data about the mean household value (`medv`) in Boston households and will try to predict its value using the variable `lstat` (lower status of the population).

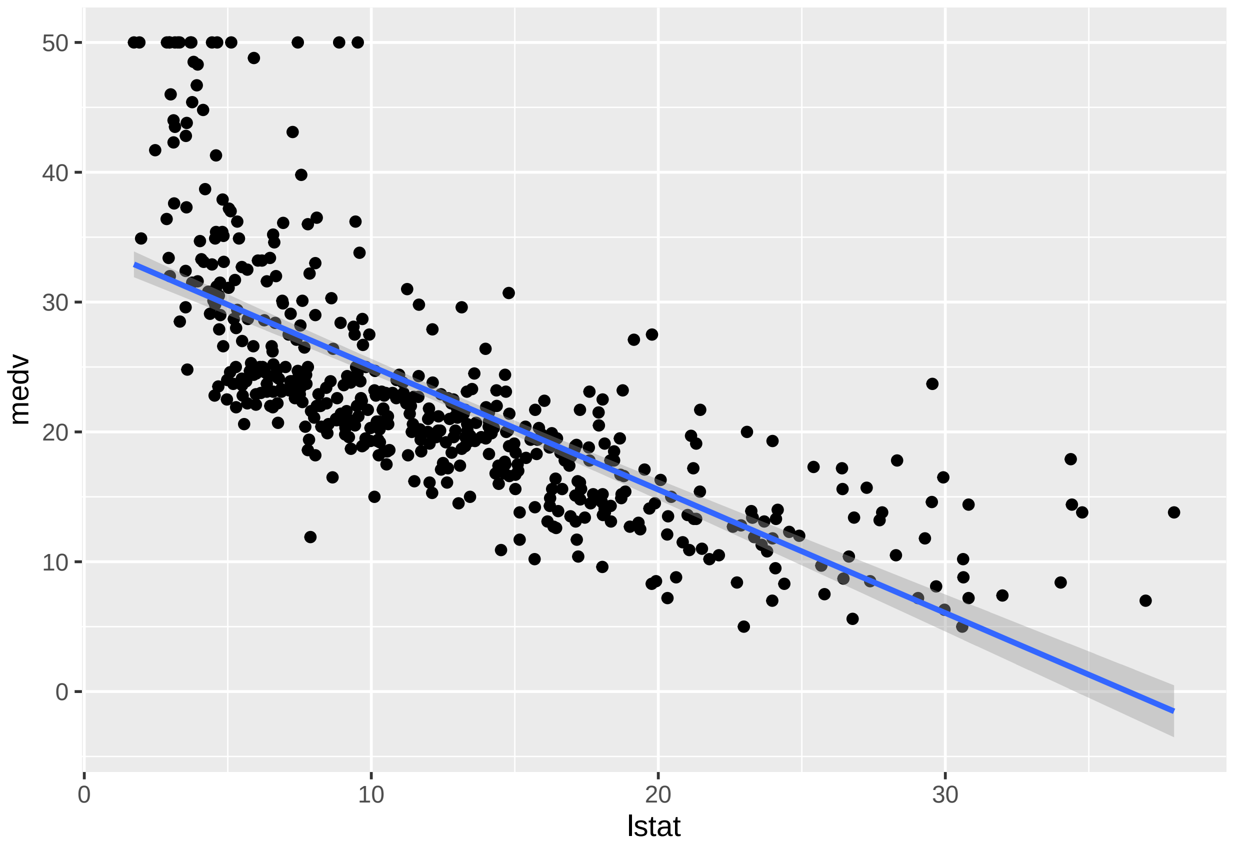
```r
library(MASS)
data("Boston", package = "MASS")
Boston_plot <- ggplot(Boston, aes(lstat, medv)) + geom_point()
model_fit <- lm(lstat ~ medv, Boston)
summary(model_fit)
```

```
##
## Call:
## lm(formula = lstat ~ medv, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.8631  -3.5959  -0.8133   2.4069  20.3152
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.55886    0.56823   44.98   <2e-16 ***
## medv        -0.57276    0.02335  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.826 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```r
Boston_plot + stat_smooth(method = lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



You may say that the plot does not really look like there is a linear relation between the variables. But nobody can prohibit us to do a linear regression anyway! We have very small p-value, so that means that our β_0 and β_1 are almost surely not zero. But it doesn't mean that it is the best possible model for our data. We can try to improve the situation by trying to express our observations as follows:

$$y_i = \beta_2 x_i^2 + \beta_1 x_1^2 + \beta_0 + \varepsilon_i.$$

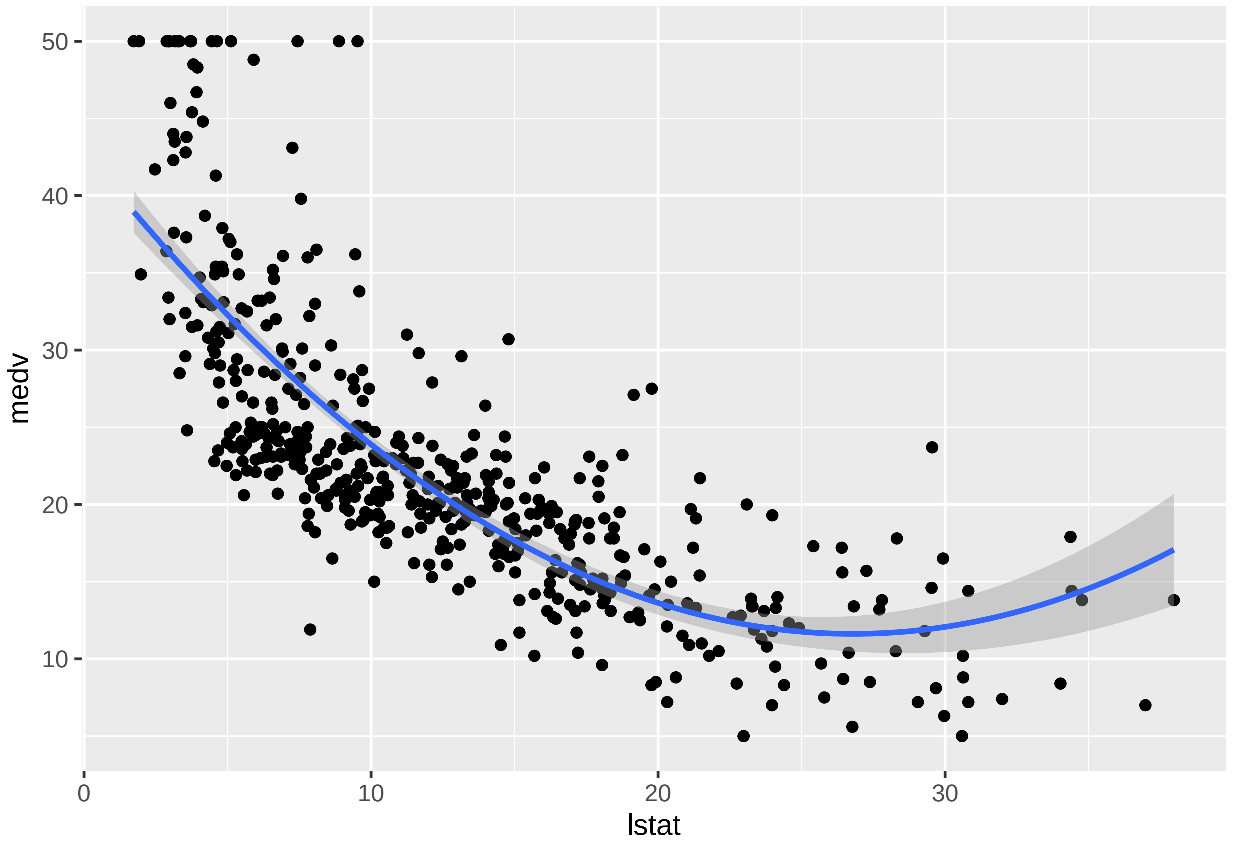
We can redo the computations with exactly the same formula:

```r
model_fit <- lm(medv ~ poly(lstat, 2, raw = TRUE), data = Boston)
summary(model_fit)
```

```
##
## Call:
```

```
## lm(formula = medv ~ poly(lstat, 2, raw = TRUE), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  42.862007   0.872084   49.15   <2e-16 ***
## poly(lstat, 2, raw = TRUE)1 -2.332821   0.123803  -18.84   <2e-16 ***
## poly(lstat, 2, raw = TRUE)2  0.043547   0.003745   11.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
```

```
Boston_plot + stat_smooth(method = lm, formula = y ~ poly(x, 2, raw = TRUE))
```



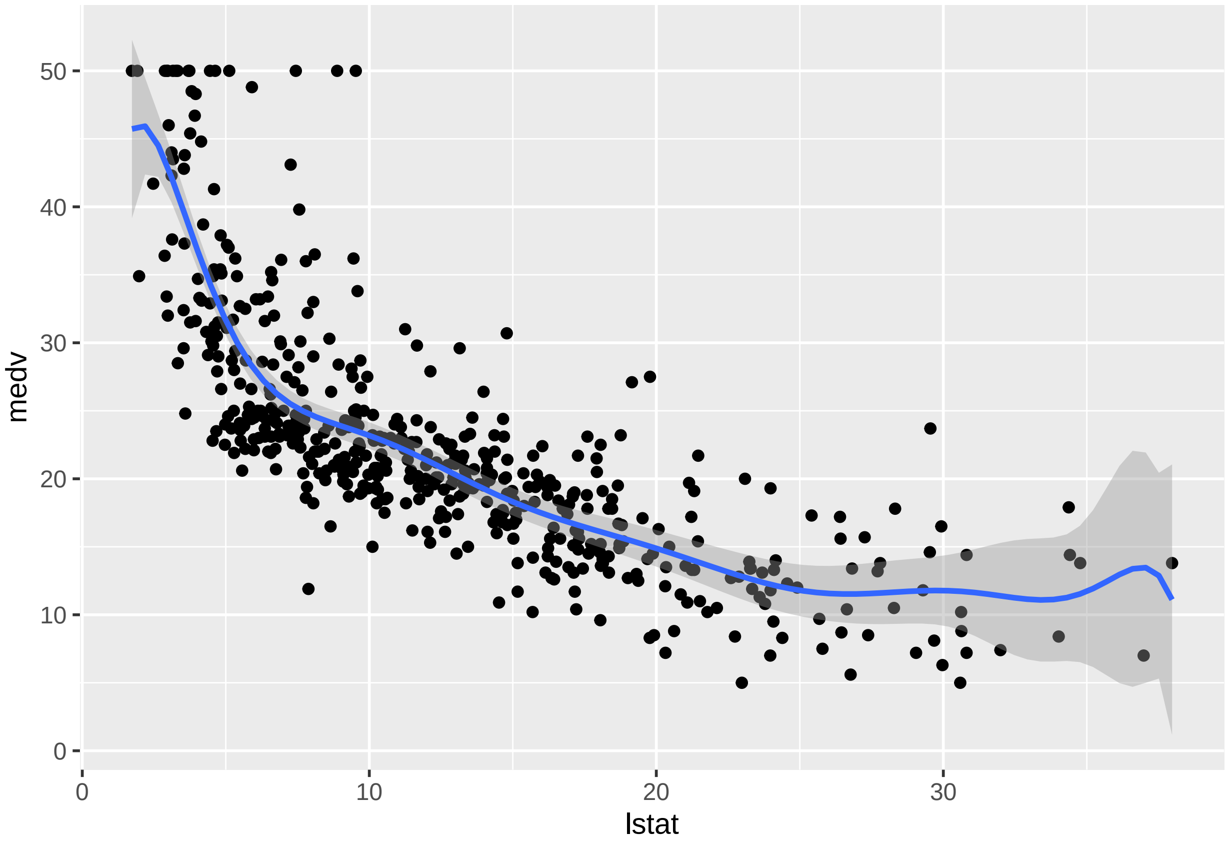
The plot starts to look better, but the residual error has even increased. What if we try to fit a really complicated polynomial and then just get rid of the excessive coefficients by looking at their p-values?

```
model_fit <- lm(medv ~ poly(lstat, 10, raw = TRUE), data = Boston)
summary(model_fit)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 10, raw = TRUE), data = Boston)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5340  -3.0286  -0.7507   2.0437  26.4738
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   8.712e+00  2.694e+01   0.323   0.7466
## poly(lstat, 10, raw = TRUE)1   5.004e+01  2.757e+01   1.815   0.0701 .
## poly(lstat, 10, raw = TRUE)2  -2.391e+01  1.134e+01  -2.108   0.0355 *
## poly(lstat, 10, raw = TRUE)3   5.255e+00  2.493e+00   2.108   0.0355 *
## poly(lstat, 10, raw = TRUE)4  -6.614e-01  3.277e-01  -2.018   0.0441 *
## poly(lstat, 10, raw = TRUE)5   5.184e-02  2.724e-02   1.903   0.0576 .
## poly(lstat, 10, raw = TRUE)6  -2.616e-03  1.464e-03  -1.787   0.0746 .
## poly(lstat, 10, raw = TRUE)7   8.507e-05  5.070e-05   1.678   0.0940 .
## poly(lstat, 10, raw = TRUE)8  -1.722e-06  1.090e-06  -1.579   0.1150
## poly(lstat, 10, raw = TRUE)9   1.973e-08  1.324e-08   1.490   0.1368
## poly(lstat, 10, raw = TRUE)10 -9.766e-11  6.922e-11  -1.411   0.1589
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.199 on 495 degrees of freedom
## Multiple R-squared:  0.6867, Adjusted R-squared:  0.6804
## F-statistic: 108.5 on 10 and 495 DF,  p-value: < 2.2e-16
```

```r
Boston_plot + stat_smooth(method = lm, formula = y ~ poly(x, 10, raw = TRUE))
```



Now, we have fitted a 10 order polynom. But, if we have a look at the p-values, we see that it is rather doubtful that most of those coefficents are bringing some information to our system (if we take a nominative level of 5%). So, we can as well limit ourselves to a polynomial of order 3 and conclude that it is a good model:

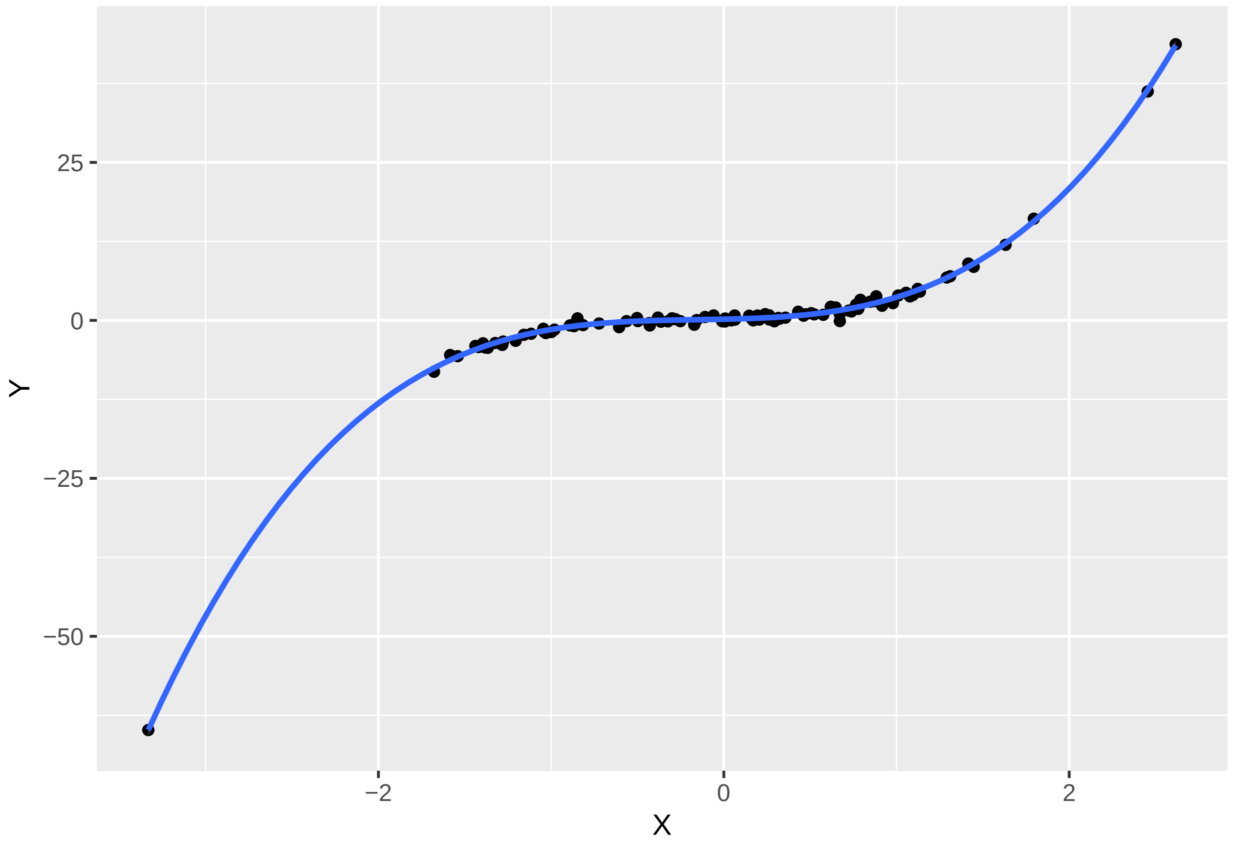
```
model_fit <- lm(medv ~ poly(lstat, 3, raw = TRUE), data = Boston)
summary(model_fit)
Boston_plot + stat_smooth(method = lm, formula = y ~ poly(x, 3, raw = TRUE))
```

Of course, it is not completely correct to say it without further study. For this data, it is not even obvious which model is correct. To better see what kind of outcome we are expecting to see in the "ideal case", let's simulate the date by ourselves and then run the regression on the obtaineed observations. What are we doing next, we are simulating a vector X as a standard normally distributed variable, then we fix a vector of parameters and, then, finally, we simulate our Y by the following formula:

$$y_i = \beta_3 x_i^3 + \beta_2 x_i^2 + \beta_1 x_i + \beta_0 + \varepsilon_i,$$

where $\varepsilon_i \sim N(0, \sigma^2)$. If we launch a polynomial regression on the couple (X, Y) (choosing order 4), we will obtain quite plausible results, indicating that the last coefficient can be omitted. But try to repeat the same experiment by trying a different X (for example, taking an absolute value of a normal variable), or a different order of a polynomial, and you will see that the result is very different from the one we expect.

```
n = 100
X <- rnorm(n, 0, 1)
poly_par <- c(0.1,0.5,1,2) # we create a randomly initialized vector of parameters
Y <- poly_par[1]+poly_par[2]*X+poly_par[3]*X^2+poly_par[4]*X^3+rnorm(n,0,0.5)
XY <- as.data.frame(cbind(X,Y))
colnames(XY) <- c("X","Y")
model_fit <- lm(Y ~ poly(X, 4, raw = TRUE))
ggplot(XY, aes(X,Y))+geom_point()+ stat_smooth(method = lm, formula = y ~ poly(x, 4, raw = TRUE))
```



```
summary(model_fit)
```

```
##
## Call:
## lm(formula = Y ~ poly(X, 4, raw = TRUE))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.70141 -0.30282 -0.01394  0.37865  1.15907
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              0.1862206  0.0721641   2.581   0.0114 *
## poly(X, 4, raw = TRUE)1  0.5719595  0.0763090   7.495 3.43e-11 ***
## poly(X, 4, raw = TRUE)2  0.9280163  0.0867001  10.704  < 2e-16 ***
## poly(X, 4, raw = TRUE)3  1.9819296  0.0205450  96.468  < 2e-16 ***
## poly(X, 4, raw = TRUE)4 -0.0005175  0.0110957  -0.047   0.9629
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4777 on 95 degrees of freedom
## Multiple R-squared:  0.9975, Adjusted R-squared:  0.9974
## F-statistic:  9318 on 4 and 95 DF,  p-value: < 2.2e-16
```

Homework: run this test multiple times, try to change the parameters, the range and the law of X, the number of simulation, the degree of the polynom etc, try to see what happens.

The conclusion: one shouldn't blindly rely on the outcome of the linear regression model to conclude what is the structure of the data. In fact, there are a lot of tools which can help us to choose a good model. Also, what is actually important for us is the prediction power of the model: not only we want to find a dependency between the observed data, but also be able to predict some data which is unobserved! More on this: in next sessions.

Session 3: Ridge and Lasso regression

Now, let us talk about the limitations of the linear regression. First, let us give the definition of the model in consideration in its vector form:

$$Y = X\beta + \varepsilon, \quad \varepsilon \sim N_p(0, \sigma I_p), \quad X, \beta \in R^p.$$

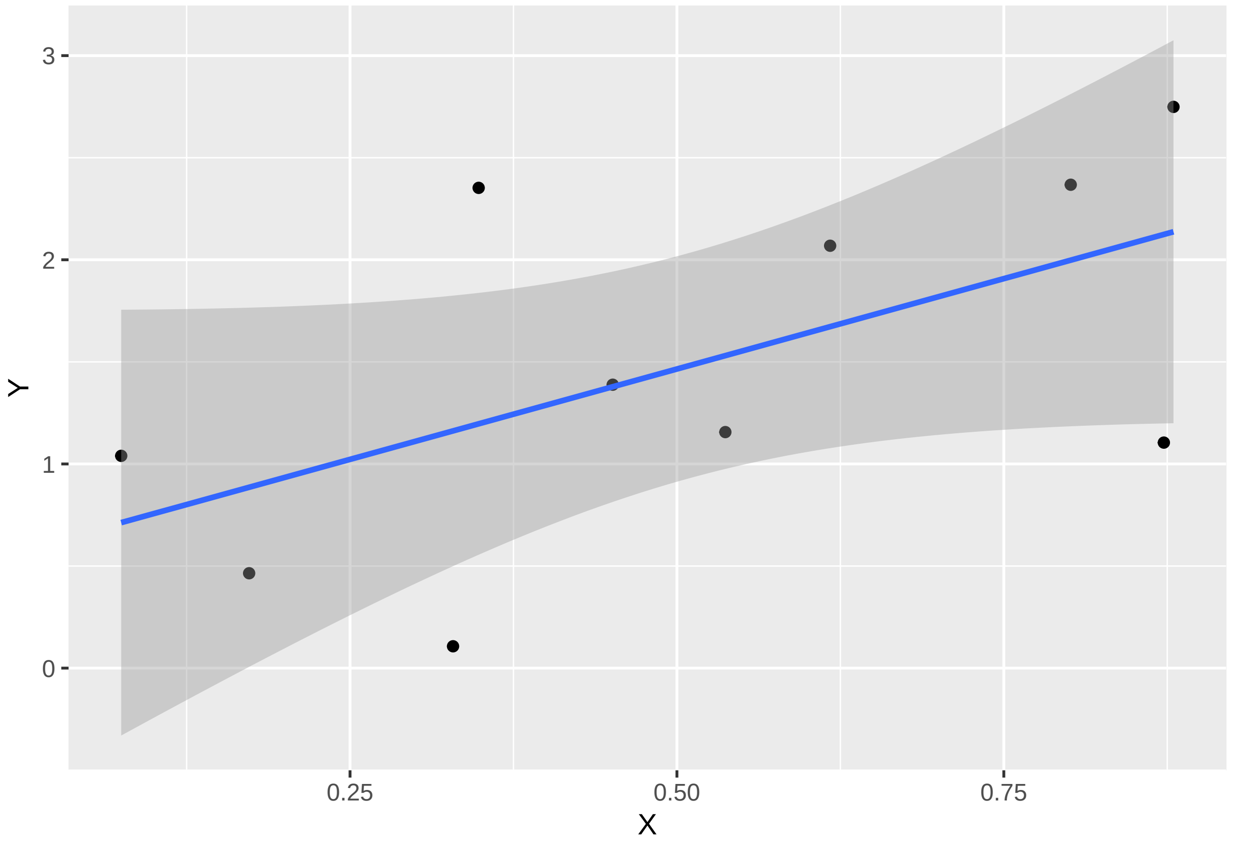
The codes provided below treat the case of two-dimensional problem (i.e., $p = 2$), but you can easily adapt the code for the other dimensions as well. To begin with, we simulate 10 points by fixing the parameters to some specific value, and try to fit the regression line with Ordinary Least Squares Estimator (OLSE), which is given as follows:

$$\hat{\beta}_{OLSE} = \arg\min_{\beta} \|Y - X\beta\|_2^2 = (X^T X)^{-1} X^T Y$$

Now, let us see how it performs in practice one more time. Try to execute the following code 10-15 times:

```
n = 10
beta0 = 1 # intercept
beta1 = 2 # slope
sigma = 1
X <- runif(n)
Y <- beta1*X + beta0+rnorm(n, 0, sigma)
XY <- as.data.frame(cbind(X,Y))
colnames(XY) <- c("X","Y")
ggplot(XY, aes(X, Y))+geom_point() + stat_smooth(method = lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



What you have probably observed is that each time the fitted line is positioned differently. It is because of the noise: even if we simulate the data with the same formula, each time we have a slightly different realization of noise, which influences greatly the accuracy of the prediction. What is the practical consequence of that? If you find estimate the parameters of the model on a small training dataset, they will not necessarily fit well the "unknown" data. Remember that the goal of fitting the model is not only in trying to explain the relations in the data you observe, but rather predict the relation in the data which is not (yet) observed!

Now, let us try to do the same procedure with Ridge regression. Recall that the penalized estimator is defined as:

$$\hat{\beta}_{Ridge} = \arg\min_{\beta} \left\{ \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\} = (X^T X + \lambda I)^{-1} X^T Y$$

Note that when $\lambda = 0$, Ridge transforms into an OLSE! In R, we can implement the estimator as follows:

```
ridge_estimator <- function(x,y, lambda){
  n <- length(x)
  X_mat <- matrix(data = c(rep(1,n),x), ncol = 2)
  lambda_mat <- matrix(c(lambda,0,0,lambda), ncol = 2)
  step1 <- t(X_mat)%*%X_mat + lambda_mat
  step2 <- t(X_mat)%*%y
  beta_estim <- solve(step1)%*%step2
  return(beta_estim)
}
lm(Y~X)$coefficients
```

```
## (Intercept)           X
##   0.5798811   1.7703387
```

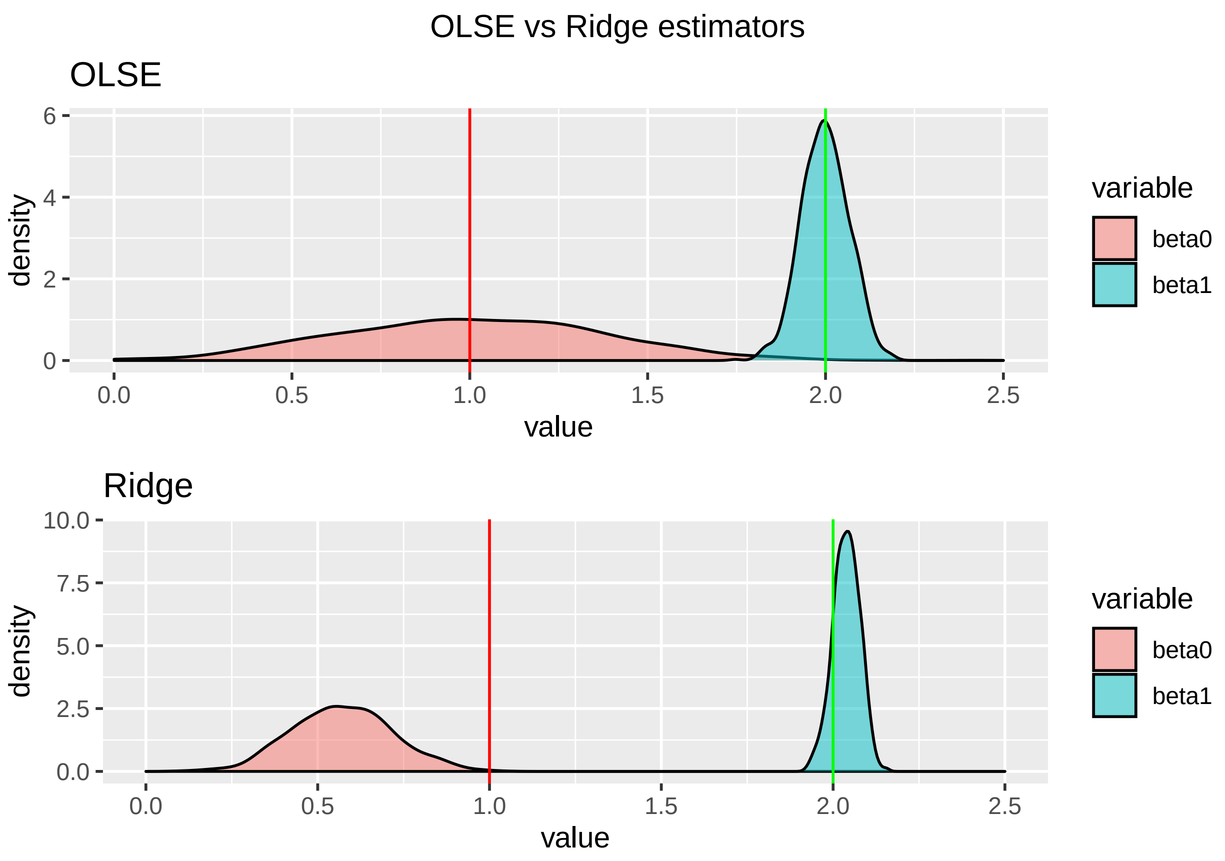
```
ridge_estimator(X,Y, lambda = 1)
```

```
##              [,1]
## [1,] 0.8804298
## [2,] 1.0058980
```

Remember that our true values are $\beta_0 = 1$ and $\beta_1 = 2$. Note that the ridge estimator tends to underestimate slightly the parameters in comparison to the OLSE estimator. Now, let us do a more profound study, simulating 1000 trajectories, as in the previous section.

```
## No id variables; using all as measure variables
## No id variables; using all as measure variables
```

```
## Warning: Removed 5 rows containing non-finite values (stat_density).
```



Then, by seeing the plots you can ask me "OK, but what is the interest, if we get a worse estimation?". Let us now do another experiment and try to generate a new sequence of points and try to evaluate the prediction error on this set of points with all the estimates we have computed on the previous step! We proceed as follows: first, we generate $n \times 10$ new observations of X and Y (meaning: we didn't use them to construct the estimators). And then we evaluate the following quantity:

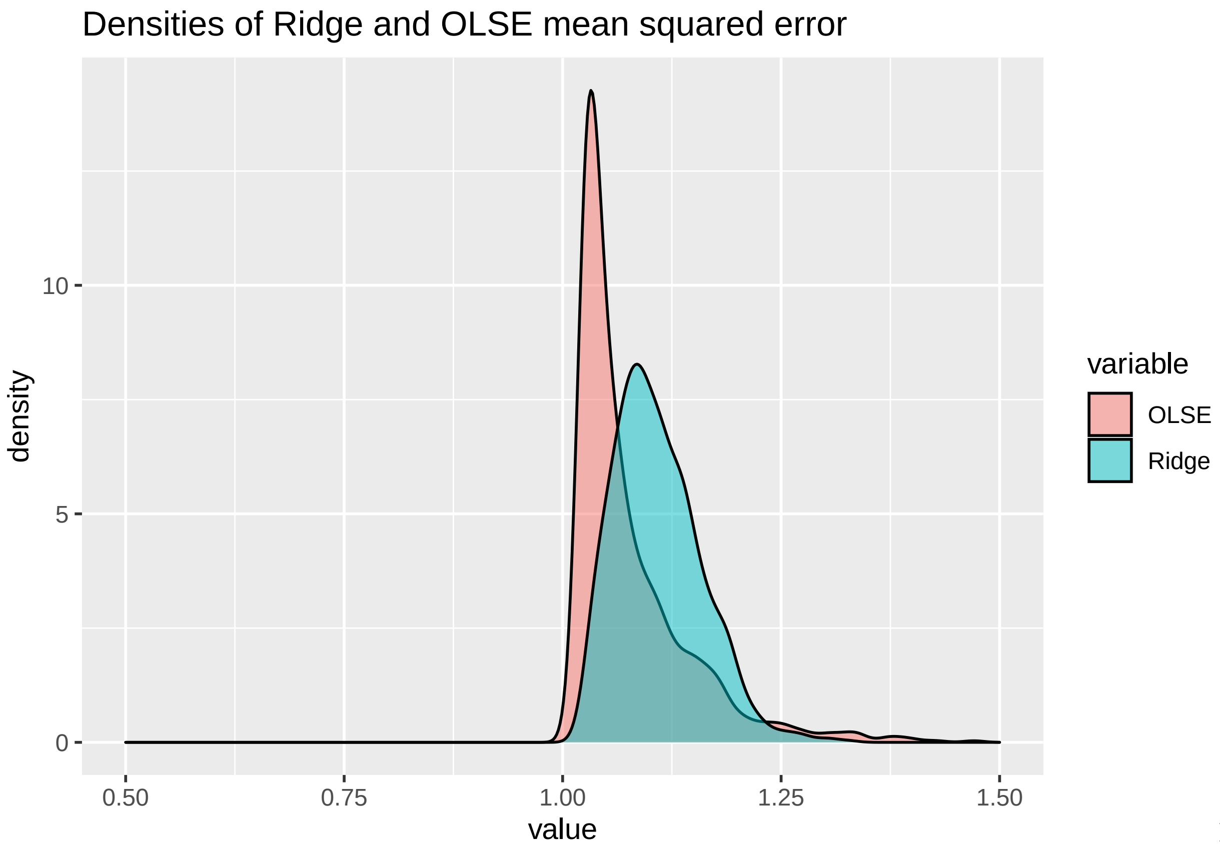
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y - X\hat{\beta}\right)^2}.$$

The code for conducting this experiment is provided below:

```r
# we simulate only a 1 set of new observations
set.seed(638)
n_new <- n*10
x <- runif(n_new)
y <- beta1*x + beta0+rnorm(n_new, 0, sigma)

mean_error_OLSE <- numeric() # vector for storing the errors
mean_error_Ridge <- numeric()
for (i in 1:N){
  X_mat <- matrix(data = c(rep(1,n_new),x), ncol = 2)
  mean_error_Ridge[i] <- sqrt(mean((y-X_mat%*%par_set_Ridge[i,])^2))
  mean_error_OLSE[i] <- sqrt(mean((y-X_mat%*%par_set_OLSE[i,])^2))
}
errors <- as.data.frame(cbind(mean_error_OLSE,mean_error_Ridge))
colnames(errors) <- c("OLSE", "Ridge")
ggplot(melt(errors), aes(x=value, fill = variable))+geom_density(alpha = 0.5)+xlim(0.5,1.5)+ggtitle("De
## No id variables; using all as measure variables

## Warning: Removed 3 rows containing non-finite values (stat_density).
```



Depending on the genrated data, the advantage of using Ridge may be more or less obvious, but what should draw your attention is the following: note that the right tail of the distribution for the RMSE for OLSE has a much more heavier tail than that of Ridge regression! In practice, it means that even though for a given dataset OLSE will be smaller in average, the risk to do a big prediction error (let's say, more than 1) is higher than for Ridge regression.

Homework: Try to launch the same experiments several times and try to pick different λ. What happens with the estimators of the parameters? With the Root Mean Square Error? Why?

Session 4: Variable selection

Recall the exercise session, where we have considered the data from the Swiss cantons, where the fertility level was evaluated as a function of agriculture, edicutation levels, examination scores, percent of catholic population and infant mortality. Here is this dataset and the model:

```r
data("swiss")
reg<- lm(Fertility ~ Agriculture+Examination+Education+Catholic+Infant.Mortality, data = swiss)
summary(reg)
```

```
##
## Call:
## lm(formula = Fertility ~ Agriculture + Examination + Education +
##     Catholic + Infant.Mortality, data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2743  -5.2617   0.5032   4.1198  15.3213
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      66.91518   10.70604   6.250 1.91e-07 ***
## Agriculture      -0.17211    0.07030  -2.448  0.01873 *
## Examination      -0.25801    0.25388  -1.016  0.31546
## Education        -0.87094    0.18303  -4.758 2.43e-05 ***
## Catholic          0.10412    0.03526   2.953  0.00519 **
## Infant.Mortality  1.07705    0.38172   2.822  0.00734 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.165 on 41 degrees of freedom
## Multiple R-squared:  0.7067, Adjusted R-squared:  0.671
## F-statistic: 19.76 on 5 and 41 DF,  p-value: 5.594e-10
```

We see that this is rather big model with 5 predictors and `RSE` given by 7.165. But what if we do not want to keep all these predictors and want to restrain ourselves to the most important ones? There are several reasons why we could be interested to do that:

1. *We want to explain the data in the simplest way.* Imagine, you are doing a presentation for some marketing directors, trying to explain what kind of advertisement is the most relevant for increasing the sales (and you did investment in ads in several social networks: TikToc, Instagram, YouTube, Facebook, LinkedIn, Telegram etc etc, on several TV channels, in public transport, in journals, and so on und so weiter…). You will not show to the marketing guy a huge list of p-values, and let him analyze, you will want to choose the most suitable model and say where to put the money.

2. From mathematical point of view, *the more parameters you have, the more difficult it is to obtain a good estimation*: estimating 2 parameters from a set of 100 observations is not the same as estimating 50 parameters from the same set.

3. To avoid *collinearity*: there can be highly correlated predictors which contain the same information. For example, imagine you want to predict a risk of developing a diabetis, basing on a dataset, where you have data about Body Mass Index, weight, height, age etc. It is clear that BMI already contains information from weight and height, so that it can be interesting to leave BMI and discard the rest for the sake of simplicity.

The general idea behind simplifying the model and choosing the most relevant variables is the following: we want to choose the variables to have the most drastical reduction in RSS error, but not more. Note also that decrease in RSS does not always mean an equivalent decrease in the prediction error (which is more important!).

Then, how can we achieve that?

Step-wise methods

As the name suggests, step-wise methods propose to add (*forward* selection) or eliminate (*backward*) selection of the variables. In the case of backward selection the algorithm is the following:

- Perform the linear regression including all the predictors
- Eliminate the predictor with the biggest non-significant p-value
- Repeat until all predictors are significant.

For example, we see that the p-value for the "Examination" obtained in the beginning of the section is insignificant. Let us try to perform the analysis again, by excluding it:

```
reg<- lm(Fertility ~ Agriculture+Education+Catholic+Infant.Mortality, data = swiss)
summary(reg)
```

```
##
## Call:
## lm(formula = Fertility ~ Agriculture + Education + Catholic +
##     Infant.Mortality, data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.6765  -6.0522   0.7514   3.1664  16.1422
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      62.10131    9.60489   6.466 8.49e-08 ***
## Agriculture      -0.15462    0.06819  -2.267  0.02857 *
## Education        -0.98026    0.14814  -6.617 5.14e-08 ***
## Catholic          0.12467    0.02889   4.315 9.50e-05 ***
## Infant.Mortality  1.07844    0.38187   2.824  0.00722 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.168 on 42 degrees of freedom
## Multiple R-squared:  0.6993, Adjusted R-squared:  0.6707
## F-statistic: 24.42 on 4 and 42 DF,  p-value: 1.717e-10
```

Note that the RSE has increased very insignificantly when we have excluded the "examination"! Then, strictly speaking, we can try to repeat the analysis by excluding "Agriculture": its p-value is significant at level 0.05, but not at 0.01, and see what happens:

```
reg<- lm(Fertility ~ Education+Catholic+Infant.Mortality, data = swiss)
summary(reg)
```

```
##
## Call:
## lm(formula = Fertility ~ Education + Catholic + Infant.Mortality,
##     data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.4781  -5.4403  -0.5143   4.1568  15.1187
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       48.67707    7.91908   6.147 2.24e-07 ***
## Education         -0.75925    0.11680  -6.501 6.83e-08 ***
## Catholic           0.09607    0.02722   3.530  0.00101 **
## Infant.Mortality  1.29615    0.38699   3.349  0.00169 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.505 on 43 degrees of freedom
## Multiple R-squared:  0.6625, Adjusted R-squared:  0.639
## F-statistic: 28.14 on 3 and 43 DF,  p-value: 3.15e-10
```

Now we see that RSE has increased even more (it is logical: if we try to base our choice on RSE, it will tend to choose the biggest model), but essentially we still have a plausible model. If we would like to conclude here, we could say that there is a linear dependency between the fertility rate in swiss cantones and the respectives level of education, the percentage of catholic population and the infant mortality (careful here, high child mortality is probably linked to the higher fertility rate, often combined with a poor healthcare!).

For the forward selection, the procedure is almost the same, but in the other direction: you start with no predictors at all, and then you take the one with the smallest less-than-critical p-value, keep it, then repeat the procedure until there is no more “important” predictors to add. **Homework:** do the forward selection on `swiss` dataset and see what it gives!

The advantage of forward and backward selection is obvious: the procedure is intuitively clear and simple to implement. The disadvantage is also easy to notice: it is already a bit time consuming to do the analysis on the set with 5 predictors. But imagine what happens if you have 60 of them or even more! **Question:** if you have p predictors, how many possible models do you have? Anyway, it becomes more convenient to do a criteria-based variable selection.

Criteria-based variable selection

The idea behind the criterias we will present below is the following: not only we want a model which fits data good, but also we want to be it as simple as possible. To get this effect, we need to introduce a penalization term, depending on the number of parameters (just like we have seen in Ridge regression before!). The most commonly used criterias are the following:

$$AIC = -2\text{log-likelihood} + 2p$$

and

$$AIC = -2\text{log-likelihood} + p\log n$$

Question: what is the likelihood for the linear regression model?

Here, AIC stays for the Akaike Information Criteria, and BIC — for Bayesian Information Criteria. Then we choose a “good” model as the one which has the smallest information criteria (either of them, depending on our preferences).

Question: Which criteria, in your opinion, penalizes more the dimension of the parameter vector?

In **R** we can do the variable selection automatically (both step-wise and criteria-based) using the command `step`, `AIC` and `BIC`. Like this:

```r
reg<- lm(Fertility ~ ., data = swiss)
step(reg)
```

```
## Start:  AIC=190.69
## Fertility ~ Agriculture + Examination + Education + Catholic +
##     Infant.Mortality
##
##                    Df Sum of Sq    RSS    AIC
## - Examination       1     53.03 2158.1 189.86
## <none>                          2105.0 190.69
## - Agriculture       1    307.72 2412.8 195.10
## - Infant.Mortality  1    408.75 2513.8 197.03
## - Catholic          1    447.71 2552.8 197.75
## - Education         1   1162.56 3267.6 209.36
##
## Step:  AIC=189.86
## Fertility ~ Agriculture + Education + Catholic + Infant.Mortality
##
##                    Df Sum of Sq    RSS    AIC
## <none>                          2158.1 189.86
## - Agriculture       1    264.18 2422.2 193.29
## - Infant.Mortality  1    409.81 2567.9 196.03
## - Catholic          1    956.57 3114.6 205.10
## - Education         1   2249.97 4408.0 221.43
```

```
##
## Call:
## lm(formula = Fertility ~ Agriculture + Education + Catholic +
##     Infant.Mortality, data = swiss)
##
## Coefficients:
##      (Intercept)       Agriculture         Education          Catholic
##          62.1013           -0.1546           -0.9803            0.1247
## Infant.Mortality
##           1.0784
```

```
AIC(reg)
```

```
## [1] 326.0716
```

```
BIC(reg)
```

```
## [1] 339.0226
```

Let us now try to compute the same criterias, but excluding the variable "Examination" (we have seen that it is not bringing much to the accuracy of the model):

```
reg<- lm(Fertility ~ .-Examination, data = swiss)
AIC(reg)
```

```
## [1] 325.2408
```

```
BIC(reg)
```

```
## [1] 336.3417
```

What do you observe? How would you interpret the results?

Cross-validation

No matter the approach we use, what we ultimately want is to have a model which does a good job in *predicting* the data we don't know. So, it is logical to test each model on some unknown data prior to selecting a good one. That's where cross-validation comes in handy. The idea of the method is the following:

- You split your data in a training and validating (or test) datasets. It is common take around 80% of values for training and 20% for validating, but in general you can take whatever proportion you find suitable.
- You build (train) the model on the training dataset. By "training" I mean estimating the parameters of the regression
- Test your model on the validating dataset.
- Repeat with the other "candidate" models and choose the one which fits the validating data best.

Here is the minimal working example on Swiss dataset (courtesy of https://support.rstudio.com/hc/en-us/articles/200486468)

```r
library(tidyverse)
library(caret)
training.samples <- swiss$Fertility %>% createDataPartition(p = 0.8, list = FALSE)
train.data  <- swiss[training.samples, ]
test.data <- swiss[-training.samples, ]
# Build the model
model <- lm(Fertility ~., data = train.data)
# Make predictions and compute the R2, RMSE and MAE
predictions <- model %>% predict(test.data)
RMSE(predictions, test.data$Fertility)
```

Try to launch this code several times and you will see that each time your RMSE will be slightly different (it varies between 6 and 10). It is because you split data randomly. Now let us try to do the same trick and see what happens if we remove the variable "Examination":

```r
training.samples <- swiss$Fertility %>% createDataPartition(p = 0.8, list = FALSE)
train.data  <- swiss[training.samples, ]
test.data <- swiss[-training.samples, ]
# Build the model
model <- lm(Fertility ~.-Examination, data = train.data)
# Make predictions and compute the R2, RMSE and MAE
predictions <- model %>% predict(test.data)
RMSE(predictions, test.data$Fertility)
```

If you relaunch the code you will see that the RMSE tends to be smaller if one variable is excluded. But, of course, it is difficult to do the conclusion basing on the evidence from only one experiment. To do the method more robust, it is common to use a k-fold validation. It is done as follows:

1. The dataset is splitted in k subsets of equal size
2. One of the subsets is reserved as a validating one, the model is trained on the rest
3. The RMSE is evaluated on the validating dataset
4. Now, another subset is taken as a validating dataset, the operation repeats
5. The final RMSE is evaluated as a mean RMSE over k procedures

Homework: try to do that on the Swiss data!