Using R for Introduction to Econometrics

Christoph Hanck, Martin Arnold, Alexander Gerber and Martin Schmelzer

2018-07-26

Contents

Chapter 1

Introduction

The interest in the freely available statistical programming language and software environment `R` (R Core Team, 2018) is soaring. By the time we wrote first drafts for this project, more than 11000 addons (many of them providing cutting-edge methods) were made available on the Comprehensive `R` Archive Network (CRAN), an extensive network of FTP servers around the world that store identical and up-to-date versions of `R` code and its documentation. `R` dominates other (commercial) software for statistical computing in most fields of research in applied statistics. The benefits of it being freely available, open source and having a large and constantly growing community of users that contribute to CRAN render `R` more and more appealing for empirical economists and econometricians alike.

A striking advantage of using `R` in econometrics is that it enables students to explicitly document their analysis step-by-step such that it is easy to update and to expand. This allows to re-use code for similar applications with different data. Furthermore, `R` programs are fully reproducible, which makes it straightforward for others to comprehend and validate results.

Over the recent years, `R` has thus become an integral part of the curricula of econometrics classes we teach at the University of Duisburg-Essen. In some sense, learning to code is comparable to learning a foreign language and continuous practice is essential for the learning success. Needless to say, presenting bare `R` code

on slides does not encourage the students to engage with hands-on experience on their own. This is why R is crucial. As for accompanying literature, there are some excellent books that deal with R and its applications to econometrics (e.g. Kleiber and Zeileis, 2008). However, such sources beyond the scope of undergraduate students in economics having little understanding of econometric methods and barely any experience in programming at all. Consequently, we started to compile a collection of reproducible reports for use in class. These reports provide guidance on how to implement selected applications from the textbook *Introduction to Econometrics* (Stock and Watson, 2015) which serves as a basis for the lecture and the accompanying tutorials. The process is facilitated considerably by `knitr` (Xie, 2018) and `rmarkdown` (Allaire et al., 2018). In conjunction, both R packages provide powerful tools for dynamic report generation which allow to seamlessly combine pure text, LaTeX, R code and its output in a variety of formats, including PDF and HTML. Being inspired by *Using R for Introductory Econometrics* (Heiss, 2016)[1] and with this powerful toolkit at hand we wrote up our own empirical companion to Stock and Watson (2015), the result which you to look at, is **U**sing **R** **f**or **I**ntroduction **t**o **E**conometrics (*URFITE*).

Similarly to the book by Heiss (2016) this project is neither a comprehensive econometrics textbook nor is it intended to be a general introduction R. We feel that Stock and Watson do a great job at explaining the intuition and theory of econometrics, and at any rate better than we could in yet another introductory textbook! *URFITE* is best described as an interactive script in the style of a reproducible research report which aims to provide students with a platform-independent e-learning arrangement by seamlessly intertwining theoretical core knowledge and empirical skills in undergraduate econometrics. Of course, the focus is on empirical applications with R. We leave out derivations and proofs wherever we can. Our goal is to enable students not only to learn how results of case studies can be replicated with R but we also intend to strengthen their ability in using the newly acquired skills in other empirical applications — immediately within *URFITE*.

To realize this, each chapter contains interactive R programming exercises. These exercises are used as supplements to code chunks that display how previously discussed techniques can be implemented within R. They are generated using the DataCamp light widget and are backed by an R session which is maintained on DataCamp's servers. You may play around with the example exercise presented below.

This interactive application is only available in the HTML version.

As you can see above, the widget consists of two tabs. `script.R` mimics an `.R`-file, a file format that is commonly used for storing R code. Lines starting with a # are commented out, that is, they are not recognized as code. Furthermore, `script.R` works like an exercise sheet where you may write down the solution you come up with. If you hit the button *Run*, the code will be executed, submission correctness tests are run and you will be notified whether your approach is correct. If it is not correct, you will receive feedback suggesting improvements or hints. The other tab, `R Console`, is a fully functional R console that can be used for trying out solutions to exercises before submitting them. Of course you may submit (almost any) R code and use the console to play around and explore. Simply type a command and hit the enter key on your keyboard.

As an example, consider the following line of code presented in chunk below. It tells R to compute the number of packages available on `CRAN`. The code chunk is followed by the output produced.

```
# compute the number of packages available on CRAN
nrow(available.packages(repos = "http://cran.us.r-project.org"))
```

```
## [1] 12773
```

Each code chunk is equipped with a button on the outer right hand side which copies the code to your clipboard. This makes it convenient to work with larger code segments. In the widget above, you may click on `R Console` and type `nrow(available.packages())` (the command from the code chunk above) and execute it by hitting *Enter* on your keyboard[2]

[1] Heiss (2016) builds on the popular *Introductory Econometrics* by Wooldridge (2016) and demonstrates how to replicate the applications discussed therein using R.

[2] The R session is initialized by clicking anywhere into the widget. This might take a few seconds. Just wait for the indicator next to the button *Run* to turn green.

Note that some lines in the widget are out-commented which ask you to assign a numeric value to a variable and then to print the variable's content to the console. You may enter your solution approach to `script.R` and hit the button *Run* in order to get the feedback described further above. In case you do not know how to solve this sample exercise (don't panic, that is probably why you are reading this), a click on *Hint* will provide you with some advice. If you still can't find a solution, a click on *Solution* will provide you with another tab, `Solution.R` which contains sample solution code. It will often be the case that exercises can be solved in many different ways and `Solution.R` presents what we consider as comprehensible and idiomatic.

Conventions Used in this Book

- *Italic* text indicates new terms, names, buttons and alike.
- `Constant width text`, is generally used in paragraphs to refer to `R` code. This includes commands, variables, functions, data types, databases and file names.
- Constant width text on gray background is used to indicate `R` code that can be typed literally by you. It may appear in paragraphs for better distinguishability among executable and non-executable code statements but it will mostly be encountered in shape of large blocks of `R` code. These blocks are referred to as code chunks (see above).

Acknowledgements

We thank Alexander Blasberg and Kim Hermann for proofreading and their constructive criticism. We also thank the *Stiferverband für die Deutsche Wissenschaft e.V.* and the Ministry of Science and Research North Rhine-Westphalia for their financial support. We are also indebted to all past students of our introductory econometrics courses at the University of Duisburg-Essen for their feedback.

1.1 A Very Short Introduction to `R` and *RStudio*

`R` Basics

As mentioned before, this book is not intended to be an introduction to `R` but as a guide on how to use its capabilities for applications commonly encountered in undergraduate econometrics. Those having basic knowledge in `R` programming will feel comfortable starting with Chapter **??**. This section, however, is meant for those who have not worked with `R` or *RStudio* before. If you at least know how to create objects and call functions, you can skip it. If you would like to refresh your skills or get a feeling for how to work with *RStudio*, keep reading.

First of all start *RStudio* and create a new `R` script by selecting *File*, *New File*, *R Script*. In the editor pane, type

```
1 + 1
```

and click on the button labeled *Run* in the top right corner of the editor. By doing so, your line of code is sent to the console and the result of this operation should be displayed right underneath it. As you can see, `R` works just like a calculator. You can do all arithmetic calculations by using the corresponding operator (`+`, `-`, `*`, `/` or `^`). If you are not sure what the last operator does, try it out and check the results.

Vectors

`R` is of course more sophisticated than that. We can work with variables or, more generally, objects. Objects are defined by using the assignment operator `<-`. To create a variable named `x` which contains the value `10` type `x <- 10` and click the button *Run* yet again. The new variable should have appeared in the environment

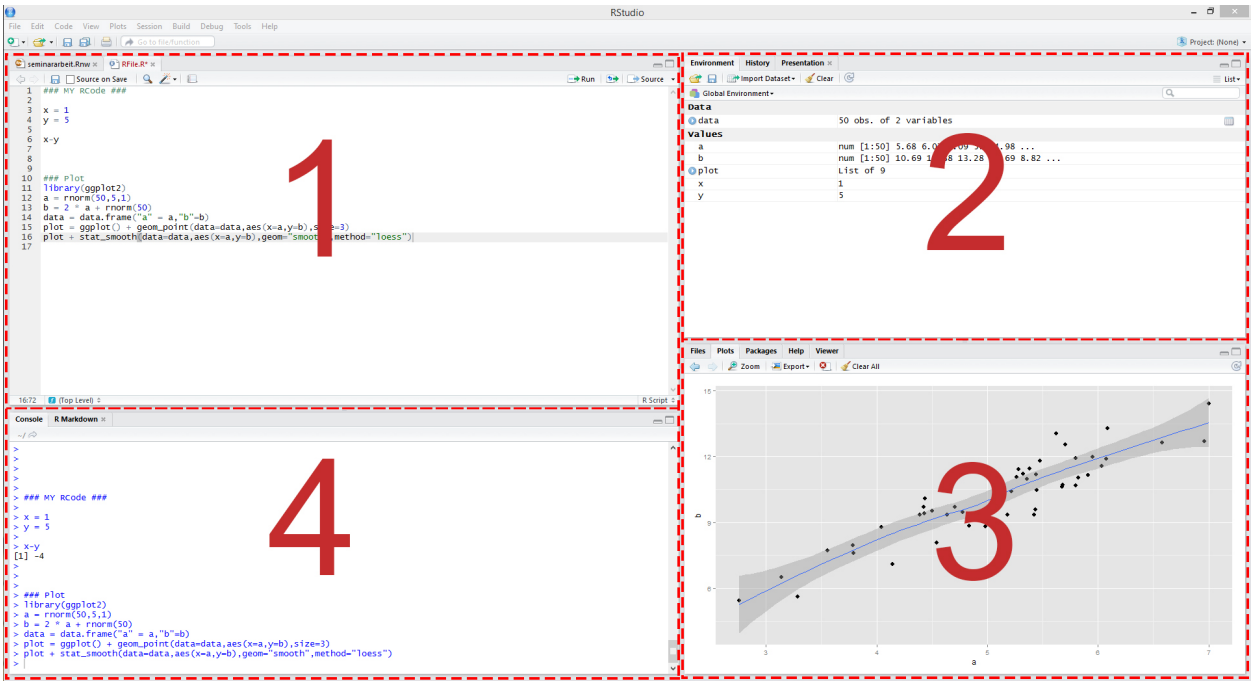


Figure 1.1: RStudio: the four panes

pane on the top right. The console however did not show any results, because our line of code did not contain any call that creates output. When you now type `x` in the console and hit return, you ask `R` to show you the value of `x` and the corresponding value should be printed in the console.

`x` is a scalar, a vector of length 1. You can easily create longer vectors by using the function `c()` (*c* for "concatenate" or "combine"). To create a vector `y` containing the numbers 1 to 5 and print it, do the following.

```
y <- c(1, 2, 3, 4, 5)
y
```

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

You can also create a vector of letters or words. For now just remember that characters have to be surrounded by quotes, else they will be parsed as object names.

```
hello <- c("Hello", "World")
```

Here we have created a vector of length 2 containing the words `Hello` and `World`.

Do not forget to save your script! To do so, select *File*, *Save*.

Functions

You have seen the function `c()` that can be used to combine objects. In general, all function calls look the same: a function name is always followed by round parentheses. Sometimes, the parentheses include arguments.

Here are two simple examples.

```
z <- seq(from = 1, to = 5, by = 1)
```

```
mean(x = z)
```

```
## [1] 3
```

In the first line we use a function called `seq()` to create the exact same vector as we did in the previous section, calling it `z`. The function takes on the arguments `from`, `to` and `by` which should be self-explanatory. The function `mean()` computes the arithmetic mean of its argument `x`. Since we pass the vector `z` as the argument `x`, the result is `3`!

If you are not sure which arguments a function expects, you may consult the function's documentation. Let's say we are not sure how the arguments required for `seq()` work. We then type `?seq` in the console. By hitting return the documentation page for that function pops up in the lower right pane of *RStudio*. In there, the section *Arguments* holds the information we seek. On the bottom of almost every help page you find examples on how to use the corresponding functions. This is very helpful for beginners and we recommend to look out for those.

Of course, all of the commands presented above also work in interactive widgets throughout the book. You may try them below.

This interactive application is only available in the HTML version.

Chapter 2

Linear Regression with One Regressor

This chapter introduces the basics in linear regression and shows how to perform regression analysis in `R`. In linear regression, the aim is to model the relationship between a dependent variable Y and one or more explanatory variables denoted by $X_1, X_2, \ldots, X_k$. Following the book we will focus on the concept of simple linear regression throughout the whole chapter. In simple linear regression, there is just one explanatory variable X_1. If, for example, a school cuts its class sizes by hiring new teachers, that is, the school lowers X_1, the student-teacher ratios of its classes, how would this affect Y, the performance of the students involved in a standardized test? With linear regression we can not only examine whether the student-teacher ratio *does have* an impact on the test results but we can also learn about the *direction* and the *strength* of this effect.

The following packages are needed for reproducing the code presented in this chapter:

- `AER` - accompanies the Book *Applied Econometrics with R* Kleiber and Zeileis (2008) and provides useful functions and data sets.
- `MASS` - a collection of functions for applied statistics.

Make sure these are installed before you go ahead and try to replicate the examples. The safest way to do so is by checking whether the following code chunk executes without any errors.

```
library(AER)
library(MASS)
```

2.1 Simple Linear Regression

To start with an easy example, consider the following combinations of average test score and the average student-teacher ratio in some fictional school districts.

	1	2	3	4	5	6	7
TestScore	680	640	670	660	630	660.0	635
STR	15	17	19	20	22	23.5	25

To work with these data in `R` we begin by generating two vectors: one for the student-teacher ratios (`STR`) and one for test scores (`TestScore`), both containing the data from the table above.

```
# Create sample data
STR <- c(15, 17, 19, 20, 22, 23.5, 25)
TestScore <- c(680, 640, 670, 660, 630, 660, 635)
```

```
# Print out sample data
STR
```

```
## [1] 15.0 17.0 19.0 20.0 22.0 23.5 25.0
```

```
TestScore
```

```
## [1] 680 640 670 660 630 660 635
```

In a simple linear regression model, we model the relationship between both variables by a straight line, formally

$$Y = b \cdot X + a.$$

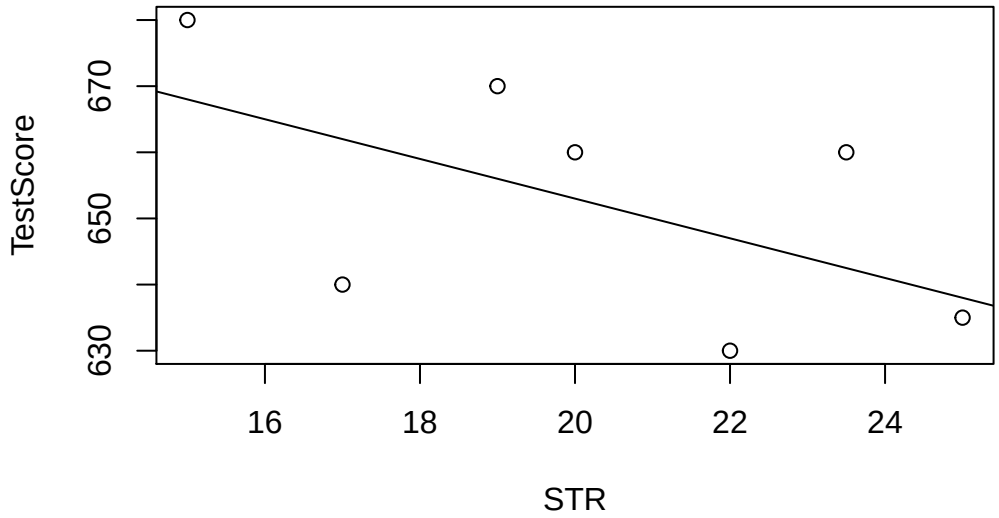
For now, let us suppose that the function which relates test score and student-teacher ratio to each other is

$$TestScore = 713 - 3 \times STR.$$

It is always a good idea to visualize the data you work with. Here, it is suitable to use `plot()` to produce a scatterplot with `STR` on the x-axis and `TestScore` on the y-axis. Just call `plot(y_variable ~ x_variable)` whereby `y_variable` and `x_variable` are placeholders for the vectors of observations we want to plot. Furthermore, we might want to add a systematic relationship to the plot. To draw a straight line, `R` provides the function `abline()`. We just have to call this function with arguments `a` (representing the intercept) and `b` (representing the slope) after executing `plot()` in order to add the line to our plot.

The following code reproduces Figure 4.1 from the textbook.

```
# create a scatterplot of the data
plot(TestScore ~ STR)

# add the systematic relationship to the plot
abline(a = 713, b = -3)
```



We find that the line does not touch any of the points although we claimed that it represents the systematic relationship. The reason for this is randomness. Most of the time there are additional influences which imply that there is no bivariate relationship between the two variables.

In order to account for these differences between observed data and the systematic relationship, we extend our model from above by an *error term* u which captures additional random effects. Put differently, u accounts for all the differences between the regression line and the actual observed data. Beside pure randomness, these deviations could also arise from measurement errors or, as will be discussed later, could be the consequence of leaving out other factors that are relevant in explaining the dependent variable.

Which other factors are plausible in our example? For one thing, the test scores might be driven by the teachers' quality and the background of the students. It is also possible that in some classes, the students were lucky on the test days and thus achieved higher scores. For now, we will summarize such influences by an additive component:

$$TestScore = \beta_0 + \beta_1 \times STR + \text{other factors}$$

Of course this idea is very general as it can be easily extended to other situations that can be described with a linear model. The basic linear regression function we will work with hence is

$$Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Key Concept 4.1 summarizes the terminology of the simple linear regression model.

Key Concept 4.1
Terminology for the Linear Regression Model with a Single Regressor

The linear regression model is

$$Y_i = \beta_0 + \beta_1 X_1 + u_i$$

where

- the index i runs over the observations, $i = 1, \dots, n$
- Y_i is the *dependent variable*, the *regressand*, or simply the *left-hand variable*
- X_i is the *independent variable*, the *regressor*, or simply the *right-hand variable*
- $Y = \beta_0 + \beta_1 X$ is the *population regression line* also called the *population regression function*
- β_0 is the *intercept* of the population regression line
- β_1 is the *slope* of the population regression line
- u_i is the *error term*.

2.2 Estimating the Coefficients of the Linear Regression Model

In practice, the intercept β_0 and slope β_1 of the population regression line are unknown. Therefore, we must employ data to estimate both unknown parameters. In the following, a real world example will be used to demonstrate how this is achieved. We want to relate test scores to student-teacher ratios measured in Californian schools. The test score is the district-wide average of reading and math scores for fifth graders. Again, the class size is measured as the number of students divided by the number of teachers (the student-teacher ratio). As for the data, the California School data set (`CASchools`) comes with an `R` package called `AER`, an acronym for Applied Econometrics with R (`?`). After installing the package with `install.packages("AER")` and attaching it with `library(AER)` the data set can be loaded using the function `data()`.

```
# install the AER package (once)
install.packages("AER")

# load the AER package
library(AER)

# load the the data set in the workspace
data(CASchools)
```

Once a package has been installed it is available for use at further occasions when invoked with `library()` — there is no need to run `install.packages()` again!

It is interesting to know what kind of object we are dealing with. `class()` returns the class of an object. Depending on the class of an object some functions (for example `plot()` and `summary()`) behave differently.

Let us check the class of the object `CASchools`.

```
class(CASchools)
```

```
## [1] "data.frame"
```

It turns out that `CASchools` is of class `data.frame` which is a convenient format to work with, especially for performing regression analysis.

With help of `head()` we get a first overview of our data. This function shows only the first 6 rows of the data set which prevents an overcrowded console output.

> Press ctrl + L to clear the console. This command deletes any code that has been typed in and executed by you or printed to the console by R functions. The good news is that anything else is left untouched. You neither loose defined variables etc. nor the code history. It is still possible to recall previously executed R commands using the up and down keys. If you are working in *RStudio*, press ctrl + Up on your keyboard (CMD + Up on a Mac) to review a list of previously entered commands.

```
head(CASchools)
```

```
##   district                          school  county grades students
## 1    75119              Sunol Glen Unified Alameda  KK-08      195
## 2    61499            Manzanita Elementary   Butte  KK-08      240
## 3    61549     Thermalito Union Elementary   Butte  KK-08     1550
## 4    61457 Golden Feather Union Elementary   Butte  KK-08      243
## 5    61523        Palermo Union Elementary   Butte  KK-08     1335
## 6    62042         Burrel Union Elementary  Fresno  KK-08      137
##   teachers calworks   lunch computer expenditure    income   english  read
## 1    10.90   0.5102  2.0408       67    6384.911 22.690001  0.000000 691.6
## 2    11.15  15.4167 47.9167      101    5099.381  9.824000  4.583333 660.5
## 3    82.90  55.0323 76.3226      169    5501.955  8.978000 30.000002 636.3
## 4    14.00  36.4754 77.0492       85    7101.831  8.978000  0.000000 651.9
## 5    71.50  33.1086 78.4270      171    5235.988  9.080333 13.857677 641.8
## 6     6.40  12.3188 86.9565       25    5580.147 10.415000 12.408759 605.7
##    math
## 1 690.0
## 2 661.9
## 3 650.9
## 4 643.5
## 5 639.9
## 6 605.4
```

We find that the data set consists of plenty of variables and that most of them are numeric.

By the way: an alternative to `class()` and `head()` is `str()` which is deduced from 'structure' and gives a comprehensive overview of the object. Try!

Turning back to `CASchools`, the two variables we are interested in (i.e., average test score and the student-teacher ratio) are *not* included. However, it is possible to calculate both from the provided data. To obtain the student-teacher ratios, we simply divide the number of students by the number of teachers. The average test score is the arithmetic mean of the test score for reading and the score of the math test. The next code chunk shows how the two variables can be constructed as vectors and how they are appended to `CASchools`.

```
# compute STR and append it to CASchools
CASchools$STR <- CASchools$students/CASchools$teachers

# compute TestScore and append it to CASchools
CASchools$score <- (CASchools$read + CASchools$math)/2
```

If we ran `head(CASchools)` again we would find the two variables of interest as additional columns named `STR` and `score` (check this!).

Table 4.1 from the textbook summarizes the distribution of test scores and student-teacher ratios. There are several functions which can be used to produce similar results, e.g.,

- `mean()` (computes the arithmetic mean of the provided numbers)
- `sd()` (computes the sample standard deviation)
- `quantile()` (returns a vector of the specified sample quantiles for the data).

The next code chunk shows how to achieve this. First, we compute summary statistics on the columns `STR` and `score` of `CASchools`. In order to get nice output we gather the measures in a `data.frame` named `DistributionSummary`.

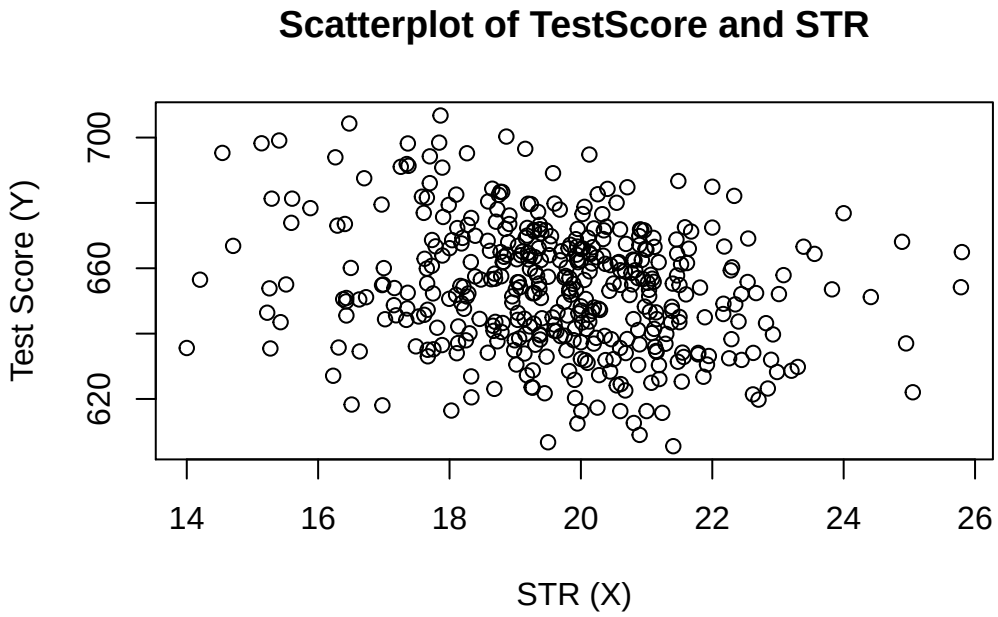
```
# compute sample averages of STR and score
avg_STR <- mean(CASchools$STR)
avg_score <- mean(CASchools$score)

# compute sample standard deviations of STR and score
sd_STR <- sd(CASchools$STR)
sd_score <- sd(CASchools$score)

# set up a vector of percentiles and compute the quantiles
quantiles <- c(0.10, 0.25, 0.4, 0.5, 0.6, 0.75, 0.9)
quant_STR <- quantile(CASchools$STR, quantiles)
quant_score <- quantile(CASchools$score, quantiles)

# gather everything in a data.frame
DistributionSummary <- data.frame(
                                  Average = c(avg_STR, avg_score),
                                  StandardDeviation = c(sd_STR, sd_score),
                                  quantile = rbind(quant_STR, quant_score)
                                  )

# print the summary to the console
DistributionSummary
```

```
##               Average StandardDeviation quantile.10. quantile.25.
## quant_STR    19.64043          1.891812      17.3486     18.58236
## quant_score 654.15655         19.053347     630.3950    640.05000
##             quantile.40. quantile.50. quantile.60. quantile.75.
## quant_STR       19.26618     19.72321      20.0783     20.87181
## quant_score    649.06999    654.45000     659.4000    666.66249
##             quantile.90.
## quant_STR       21.86741
## quant_score    678.85999
```

As for the sample data, we use `plot()`. This allows us to detect characteristics of our data, such as outliers which are harder to discover by looking at mere numbers. This time we add some additional arguments to the call of `plot()`.

The first argument in our call of `plot()`, `score ~ STR`, is again a formula that variables. However, this time the two variables are not saved in separate vectors but are columns of `CASchools`. Therefore, `R` would not find them without the argument `data` being correctly specified. `data` must be in accordance with the name of the `data.frame` to which the variables belong to, in this case `CASchools`. Further arguments are used to change the appearance of the plot: while `main` adds a title, `xlab` and `ylab` add custom labels to both axes.

```
plot(score ~ STR,
     data = CASchools,
     main = "Scatterplot of TestScore and STR",
     xlab = "STR (X)",
     ylab = "Test Score (Y)")
```



The plot (Figure 4.2 in the book) shows the scatterplot of all observations on the student-teacher ratio and test score. We see that the points are strongly scattered, and that the variables are negatively correlated. That is, we expect to observe lower test scores in bigger classes.

The function `cor()` (see `?cor` for further info) can be used to compute the correlation between two *numeric* vectors.

```
cor(CASchools$STR, CASchools$score)
```

```
## [1] -0.2263627
```

As the scatterplot already suggests, the correlation is negative but rather weak.

The task we are now faceing is to find a line which best fits the data. Of course we could simply stick with graphical inspection and correlation analysis and then select the best fitting line by eyeballing. However, this would be rather subjective: different observers would draw different regression lines. On this account, we are interested in techniques that are less arbitrary. Such a technique is given by ordinary least squares (OLS) estimation.

The Ordinary Least Squares Estimator

The OLS estimator chooses the regression coefficients such that the estimated regression line is as "close" as possible to the observed data points. Here, closeness is measured by the sum of the squared mistakes

made in predicting Y given X. Let b_0 and b_1 be some estimators of β_0 and β_1. Then the sum of squared estimation mistakes can be expressed as

$$\sum_{i=1}^{n}(Y_i - b_0 - b_1 X_i)^2.$$

The OLS estimator in the simple regression model is the pair of estimators for intercept and slope which minimizes the expression above. The derivation of the OLS estimators for both parameters are presented in Appendix 4.1 of the book. The results are summarized in Key Concept 4.2.

Key Concept 4.2
The OLS Estimator, Predicted Values, and Residuals

The OLS estimators of the slope β_1 and the intercept β_0 in the simple linear regression model are

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{n}(X_i - \overline{X})^2},$$
$$\hat{\beta}_0 = \overline{Y} - \hat{\beta}_1 \overline{X}.$$

The OLS predicted values $\widehat{Y}_i$ and residuals $\hat{u}_i$ are

$$\widehat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i,$$
$$\hat{u}_i = Y_i - \widehat{Y}_i.$$

The estimated intercept $\hat{\beta}_0$, the slope parameter $\hat{\beta}_1$ and the residuals ($\hat{u}_i$) are computed from a sample of n observations of X_i and Y_i, i, ..., n. These are *estimates* of the unknown true population intercept (β_0), slope (β_1), and error term (u_i).

The formulas presented above may not be very intuitive at first glance. The following interactive application aims to help you understand the mechanics of OLS. You can add observations by clicking into the coordinate system where the data are represented by points. Once two or more observations are available, the application computes a regression line using OLS and some statistics which are displayed in the right panel. The results are updated as you add further observations to the left panel. A double-click resets the application, i.e., all data are removed.

There are many possible ways to compute $\hat{\beta}_0$ and $\hat{\beta}_1$ in R. For example, we could implement the formulas presented in Key Concept 4.2 with two of R's most basic functions: `mean()` and `sum()`.

```
attach(CASchools) # allows to use the variables contained in CASchools directly

# compute beta_1_hat
beta_1 <- sum((STR - mean(STR))*(score - mean(score))) / sum((STR - mean(STR))^2)

# compute beta_0_hat
beta_0 <- mean(score) - beta_1 * mean(STR)

# print the results to the console
beta_1
```

```
## [1] -2.279808
```

```
beta_0
```

```
## [1] 698.9329
```

Of course, there are even more manual ways to perform these tasks. With OLS being one of the most widely-used estimation techniques, `R` of course already contains a built-in function named `lm()` (**l**inear **m**odel) which can be used to carry out regression analysis.

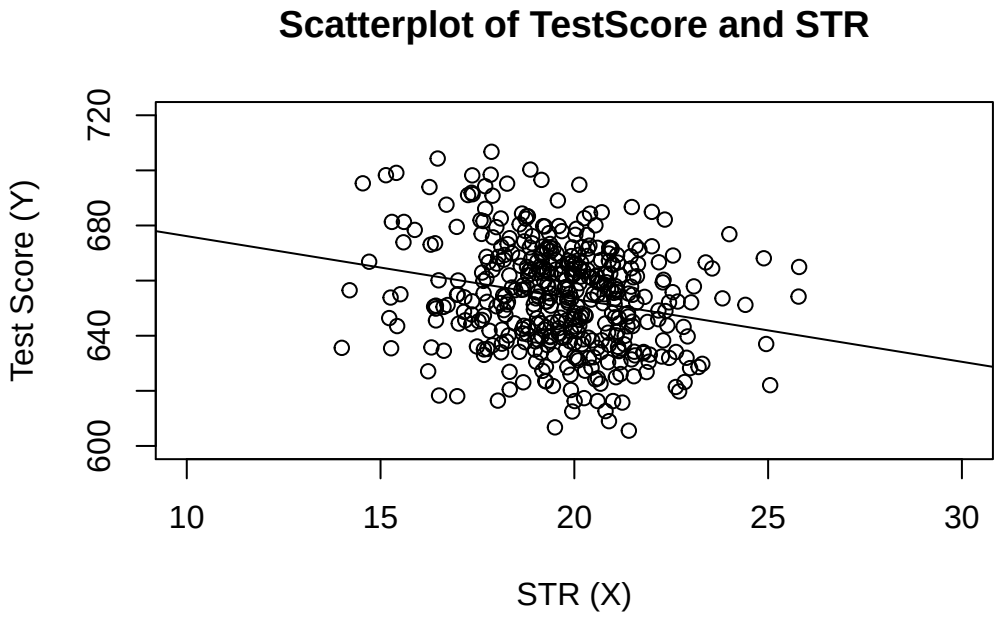
The first argument of the function to be specified is, similar to `plot()`, the regression formula with the basic syntax `y ~ x` where `y` is the dependent variable and `x` the explanatory variable. The argument `data` determines the data set to be used in the regression. We now revisit the example from the book where the relationship between the test scores and the class sizes is analyzed. The following code uses `lm()` to replicate the results presented in figure 4.3 of the book.

```
# estimate the model and assign the result to linear_model
linear_model <- lm(score ~ STR, data = CASchools)

# print the standard output of the estimated lm object to the console
linear_model
```

```
##
## Call:
## lm(formula = score ~ STR, data = CASchools)
##
## Coefficients:
## (Intercept)          STR
##      698.93        -2.28
```

Let us add the estimated regression line to the plot. This time we also enlarge the ranges of both axes by setting the arguments `xlim` and `ylim`.

```
# plot the data
plot(score ~ STR,
     data = CASchools,
     main = "Scatterplot of TestScore and STR",
     xlab = "STR (X)",
     ylab = "Test Score (Y)",
     xlim = c(10, 30),
     ylim = c(600, 720))

# add the regression line
abline(linear_model)
```



Did you notice that this time, we did not pass the intercept and slope parameters to `abline`? If you call `abline()` on an object of class `lm` that only contains a single regressor, `R` draws the regression line automatically!

2.3 Measures of Fit

After fitting a linear regression model, a natural question is how well the model describes the data. Visually, this amounts to assessing whether the observations are tightly clustered around the regression line. Both the *coefficient of determination* and the *standard error of the regression* measure how well the OLS Regression line fits the data.

The Coefficient of Determination

R^2, the *coefficient of determination*, is the fraction of the sample variance of Y_i that is explained by X_i. Mathematically, the R^2 can be written as the ratio of the explained sum of squares to the total sum of squares. The *explained sum of squares* (ESS) is the sum of squared deviations of the predicted values $\hat{Y}_i$, from the average of the Y_i. The *total sum of squares* (TSS) is the sum of squared deviations of the Y_i from their average. Thus we have

$$ESS = \sum_{i=1}^{n} \left(\hat{Y}_i - \overline{Y} \right)^2, \tag{2.1}$$

$$TSS = \sum_{i=1}^{n} \left(Y_i - \overline{Y} \right)^2, \tag{2.2}$$

$$R^2 = \frac{ESS}{TSS}. \tag{2.3}$$

Since $TSS = ESS + SSR$ we can also write

$$R^2 = 1 - \frac{SSR}{TSS}$$

where SSR is the sum of squared residuals, a measure for the errors made when predicting the Y by X. The SSR is defined as

$$SSR = \sum_{i=1}^{n} \hat{u}_i^2.$$

R^2 lies between 0 and 1. It is easy to see that a perfect fit, i.e., no errors made when fitting the regression line, implies $R^2 = 1$ since then we have $SSR = 0$. On the contrary, if our estimated regression line does not explain any variation in the Y_i, we have $ESS = 0$ and consequently $R^2 = 0$.

The Standard Error of the Regression

The *Standard Error of the Regression* (SER) is an estimator of the standard deviation of the residuals $\hat{u}_i$. As such it measures the magnitude of a typical deviation from the regression line, i.e. the magnitude of a typical residual.

$$SER = s_{\hat{u}} = \sqrt{s_{\hat{u}}^2} \quad \text{where} \quad s_{\hat{u}}^2 = \frac{1}{n-2} \sum_{i=1}^{n} \hat{u}_i^2 = \frac{SSR}{n-2}$$

Remember that the u_i are *unobserved*. This is why we use their estimated counterparts, the residuals $\hat{u}_i$, instead. See Chapter 4.3 of the book for a more detailed comment on the SER.

Application to the Test Score Data

Both measures of fit can be obtained by using the function `summary()` with an `lm` object provided as the only argument. While the function `lm()` only prints out the estimated coefficients to the console, `summary()` provides additional predefined information such as the regression's R^2 and the SER.

```
mod_summary <- summary(linear_model)
mod_summary
```

```
##
## Call:
## lm(formula = score ~ STR, data = CASchools)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -47.727 -14.251   0.483  12.822  48.540
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 698.9329     9.4675  73.825  < 2e-16 ***
## STR          -2.2798     0.4798  -4.751 2.78e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.58 on 418 degrees of freedom
## Multiple R-squared:  0.05124,    Adjusted R-squared:  0.04897
## F-statistic: 22.58 on 1 and 418 DF,  p-value: 2.783e-06
```

The R^2 in the output is called *Multiple R-squared* and has a value of 0.051. Hence, 5.1% of the variance of the dependent variable *score* is explained by the explanatory variable STR. That is, the regression explains

little of the variance in *score*, and much of the variation in test scores remains unexplained (cf. Figure 4.3 of the book).

The SER is called *Residual standard error* and equals 18.58. The unit of the SER is the same as the unit of the dependent variable. That is, on average the deviation of the actual achieved test score and the regression line is 18.58 points.

Now, let us check whether `summary()` uses the same definitions for R^2 and SER as we do when computing them manually.

```
# compute R^2 manually
SSR <- sum(mod_summary$residuals^2)
TSS <- sum((score - mean(score))^2)
R2 <- 1 - SSR/TSS

# print the value to the console
R2
```

```
## [1] 0.05124009
```

```
# compute SER manually
n <- nrow(CASchools)
SER <- sqrt(SSR / (n-2))

# print the value to the console
SER
```

```
## [1] 18.58097
```

We find that the results coincide. Note that the values provided by `summary()` are rounded to two decimal places. Can you do so using `R`?

2.4 The Least Squares Assumptions

OLS performs well under a quite broad variety of different circumstances. However, there are some assumptions which need to be satisfied in order to achieve reliable results.

Key Concept 4.3
The Least Squares Assumptions

$$Y_i = \beta_0 + \beta_1 X_i + u_i, \; i = 1, \dots, n$$

where

1. The error term u_i has conditional mean zero given X_i: $E(u_i|X_i) = 0$.
2. $(X_i, Y_i), i = 1, \dots, n$ are independent and identically distributed (i.i.d.) draws from their joint distribution.
3. Large outliers are unlikely: X_i and Y_i have nonzero finite fourth moments.

Assumption 1: The Error Term has Conditional Mean of Zero

This means that no matter which value we choose for X, the error term u must not show any systematic pattern and must have a mean of 0. Consider the case that, unconditionally, $E(u) = 0$, but for low and high values of X, the error term tends to be positive and for midrange values of X the error tends to be negative. We can use R to construct such an example. To do so we generate our own data using `R`'s built-in random number generators.

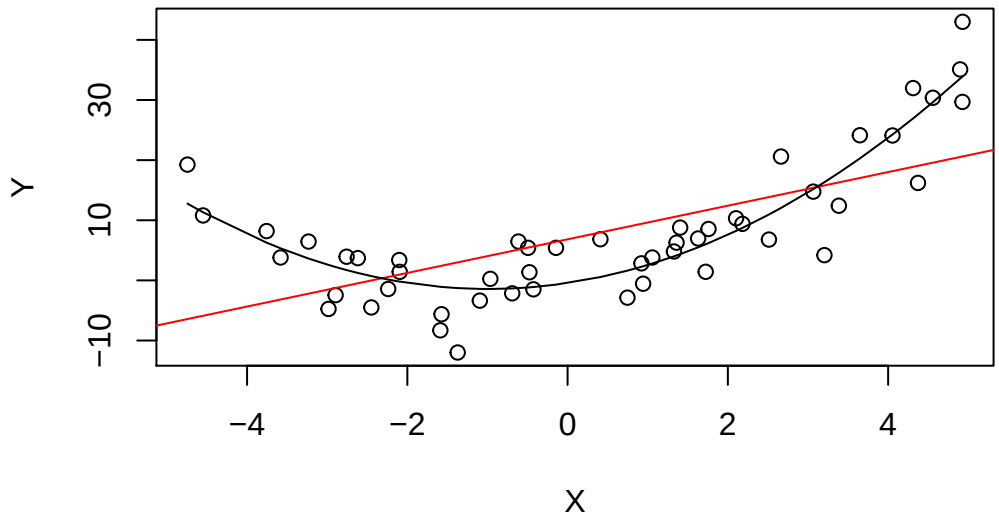
We will use the following functions:

- `runif()` - generates uniformly distributed random numbers
- `rnorm()` - generates normally distributed random numbers
- `predict()` - does predictions based on the results of model fitting functions like `lm()`
- `lines()` - adds line segments to an existing plot

We start by creating a vector containing values that are unifromly distributed on the interval $[-5, 5]$. This can be done with the function `runif()`. We also need to simulate the error term. For this we generate normally distributed random numbers with a mean equal to 0 and a variance of 1 using `rnorm()`. The Y values are obtained as a quadratic function of the X values and the error.

After generating the data we estimate both a simple regression model and a quadratic model that also includes the regressor X^2 (this is a multiple regression model, see Chapter **??**). Finally, we plot the simulated data and add a the estimated regression line of a simple regression model as well as the predictions made with a quadratic model to compare the fit graphically.

```
# set a random seed to make the results reproducible
set.seed(321)

# simulate the data
X <- runif(50, min = -5, max = 5)
u <- rnorm(50, sd = 5)

# the true relation
Y <- X^2 + 2*X + u

# estimate a simple regression model
mod_simple <- lm(Y ~ X)

# predict using a quadratic model
prediction <- predict(lm(Y ~ X +  I(X^2)), data.frame(X = sort(X)))

# plot the results
plot(Y ~ X)
abline(mod_simple, col = "red")
lines(sort(X), prediction)
```



The plot shows what is meant by $E(u_i|X_i) = 0$ and why it does not hold for the linear model:

Using the quadratic model (represented by the black curve) we see that there are no systematic deviations of the observation from the predicted relation. It is credible that the assumption is not violated when such a model is employed. However, using a simple linear regression model we see that the assumption is probably violated as $E(u_i|X_i)$ varies with the X_i.

Assumption 2: Independently and Identically Distributed Data

Most sampling schemes used when collecting data from populations produce i.i.d.-samples. For example, we could use `R`'s random number generator to randomly select student IDs from a university's enrollment list and record age X and earnings Y of the corresponding students. This is a typical example of simple random sampling and ensures that all the (X_i, Y_i) are drawn randomly from the same population.

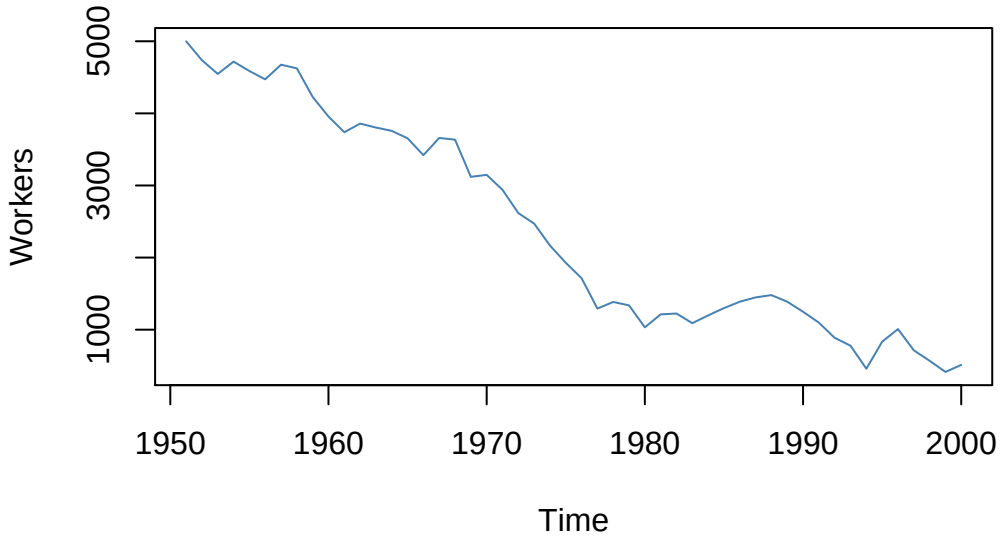
A prominent example where the i.i.d. assumption is not fulfilled is time series data where we have observations on the same unit over time. For example, take X as the number of workers in a production company over time. Due to business transformations, the company cuts job periodically by a specific share but there are also some non-deterministic influences that relate to economics, politics etc. Using `R` we can easily simulate such a process and plot it.

We start the series with a total of 5000 workers and simulate the reduction of employment with an autoregressive process that exhibits a downward movement in the long-rong and has normally distributed errors:[1]

$$employment_t = -5 + 0.98 \cdot employment_{t-1} + u_t$$

```
# set random seed
set.seed(123)

# generate a date vector
Date <- seq(as.Date("1951/1/1"), as.Date("2000/1/1"), "years")

# initialize the employment vector
X <- c(5000, rep(NA, length(Date)-1))

# generate time series observations with random influences
for (i in 2:length(Date)) {
    X[i] <- -50 + 0.98 * X[i-1] + rnorm(n = 1, sd = 200)
}

#plot the results
plot(x = Date,
     y = X,
     type = "l",
     col = "steelblue",
     ylab = "Workers",
     xlab = "Time")
```

[1] See Chapter **??** for more on autoregressive processes and time series analysis in general.



It is evident that the observations on the number of employees cannot be independent in this example: the level of today's employment is correlated with tomorrows employment level. Thus, the i.i.d. assumption is violated.

Assumption 3: Large Outliers are Unlikely

It is easy to come up with situations where extreme observations, i.e. observations that deviate considerably from the usual range of the data, may occur. Such observations are called outliers. Technically speaking, assumption 3 requires that X and Y have a finite kurtosis.[2]

Common cases where we want to exclude or (if possible) correct such outliers is when they are apparently typos, conversion errors or measurement errors. Even if it seems like extreme observations have been recorded correctly, it is advisable to exclude them before estimating a model since OLS suffers from *sensitivity to outliers*.

What does this mean? One can show that extreme observations receive heavy weighting in the estimation of the unknown regression coefficients when using OLS. Therefore, outliers can lead to strongly distorted estimates of regression coefficients. To get a better impression of this issue, consider the following application where we have placed some sample data on X and Y which are highly correlated. The relation between X and Y seems to be explained pretty good by the plotted regression line: all of the blue dots lie close to the red line and we have $R^2 = 0.92$.

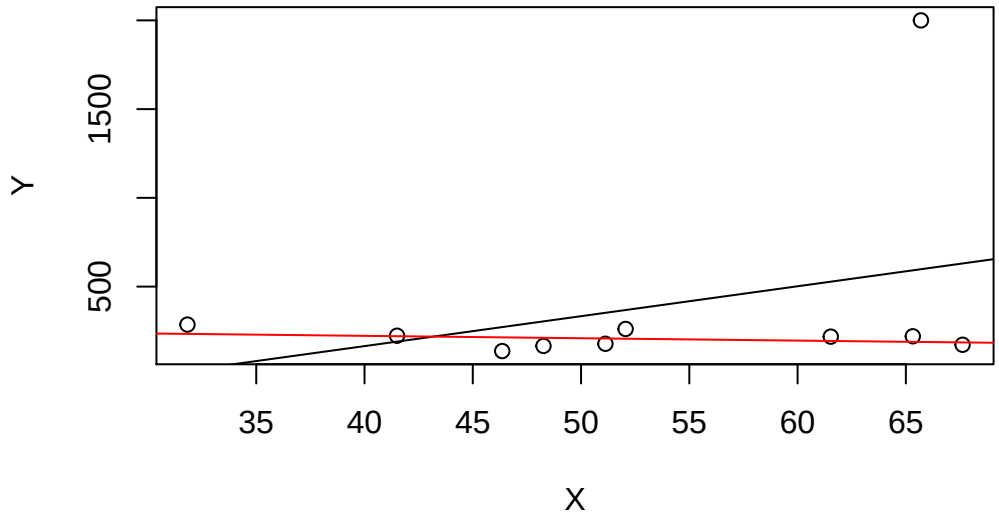
Now go ahead and add a further observation at, say, $(18, 2)$. This observations clearly is an outlier. The result is quite striking: the estimated regression line differs greatly from the one we adjudged to fit the data well. The slope is heavily downward biased and R^2 decreased to a mere 29%! Double-click inside the coordinate system to reset the app. Feel free to experiment. Choose different coordinates for the outlier or add additional ones.

The following code roughly reproduces what is shown in figure 4.5 in the book. As done above we use sample data generated using `R`'s random number functions `rnorm()` and `runif()`. We estimate two simple regression models, one based on the original data set and another using a modified set where one observation is change to be an outlier and then plot the results. In order to understand the complete code you should be familiar with the function `sort()` which sorts the entries of a numeric vector in ascending order.

```
# set random seed
set.seed(123)
```

[2] See Chapter 4.4 of the book.

```r
# generate the data
X <- sort(runif(10, min = 30, max = 70))
Y <- rnorm(10 , mean = 200, sd = 50)
Y[9] <- 2000

# fit model with outlier
fit <- lm(Y ~ X)

# fit model without outlier
fitWithoutOutlier <- lm(Y[-9] ~ X[-9])

# plot the results
plot(Y ~ X)
abline(fit)
abline(fitWithoutOutlier, col = "red")
```



2.5 The Sampling Distribution of the OLS Estimator

Because $\hat{\beta}_0$ and $\hat{\beta}_1$ are computed from a sample, the estimators themselves are random variables with a probability distribution — the so-called sampling distribution of the estimators — which describes the values they could take on over different samples. Although the sampling distribution of $\hat{\beta}_0$ and $\hat{\beta}_1$ can be complicated when the sample size is small and generally changes with the number of observations, n, it is possible, provided the assumptions discussed in the book are valid, to make certain statements about it that hold for all n. In particular

$$E(\hat{\beta}_0) = \beta_0 \quad \text{and} \quad E(\hat{\beta}_1) = \beta_1,$$

that is, $\hat{\beta}_0$ and $\hat{\beta}_1$ are unbiased estimators of β_0 and β_1, the true parameters. If the sample is sufficiently large, by the central limit theorem the *joint* sampling distribution of the estimators is well approximated by the bivariate normal distribution (2.1). This implies that the marginal distributions are also normal in large samples. Core facts on the large-sample distributions of $\hat{\beta}_0$ and $\hat{\beta}_1$ are presented in Key Concept 4.4.

Key Concept 4.4
Large Sample Distribution of $\hat{\beta}_0$ and $\hat{\beta}_1$

If the least squares assumptions in Key Concept 4.3 hold, then in large samples $\hat{\beta}_0$ and $\hat{\beta}_1$ have a joint normal sampling distribution. The large sample normal distribution of $\hat{\beta}_1$ is $N(\beta_1, \sigma^2_{\hat{\beta}_1})$, where the variance of the distribution, $\sigma^2_{\hat{\beta}_1}$, is

$$\sigma^2_{\hat{\beta}_1} = \frac{1}{n}\frac{Var\left[(X_i - \mu_X)\, u_i\right]}{\left[Var\left(X_i\right)\right]^2}. \tag{2.4}$$

The large sample normal distribution of $\hat{\beta}_0$ is $N(\beta_0, \sigma^2_{\hat{\beta}_0})$ with

$$\sigma^2_{\hat{\beta}_0} = \frac{1}{n}\frac{Var\left(H_i u_i\right)}{\left[E\left(H_i^2\right)\right]^2}, \text{ where } H_i = 1 - \left[\frac{\mu_X}{E\left(X_i^2\right)}\right] X_i. \tag{2.5}$$

The interactive simulation below continuously generates random samples (X_i, Y_i) of 200 observations where $E(Y|X) = 100 + 3X$, estimates a simple regression model, stores the estimate of the slope β_1 and visualizes the distribution of the $\widehat{\beta}_1$s observed so far using a histogram. The idea here is that for a large number of $\widehat{\beta}_1$s, the histogram gives a good approximation of the sampling distribution of the estimator. By decreasing the time between two sampling iterations, it becomes clear that the shape of the histogram approaches the characteristic bell shape of a normal distribution centered at the true slope of 3.

This interactive part of URFITE is only available in the HTML version.

Simulation Study 1

Whether the statements of Key Concept 4.4 really hold can also be verified using `R`. For this we first we build our own population of 100000 observations in total. To do this we need values for the independent variable X, for the error term u, and for the parameters β_0 and β_1. With these combined in a simple regression model, we compute the dependent variable Y. In our example we generate the numbers X_i, $i = 1, \dots, 100000$ by drawing a random sample from a uniform distribution on the interval $[0, 20]$. The realizations of the error terms u_i are drawn from a standard normal distribution with parameters $\mu = 0$ and $\sigma^2 = 100$ (note that `rnorm()` requires σ as input for the argument `sd`, see `?rnorm`). Furthermore we chose $\beta_0 = -2$ and $\beta_1 = 3.5$ so the true model is

$$Y_i = -2 + 3.5 \cdot X_i.$$

Finally, we store the results in a data.frame.

```
# simulate data
N <- 100000
X <- runif(N, min = 0, max = 20)
u <- rnorm(N, sd = 10)

# population regression
Y <- -2 + 3.5 * X + u
population <- data.frame(X, Y)
```

From now on we will consider the previously generated data as the true population (which of course would be *unknown* in a real world application, otherwise there would be no reason to draw a random sample in the first place). The knowledge about the true population and the true relationship between Y and X can be used to verify the statements made in Key Concept 4.4.

First, let us calculate the true variances $\sigma^2_{\hat{\beta}_0}$ and $\sigma^2_{\hat{\beta}_1}$ for a randomly drawn sample of size $n = 100$.

```
# set sample size
n <- 100

# compute the variance of beta_hat_0
H_i <- 1 - mean(X) / mean(X^2) * X
var_b0 <- var(H_i * u) / (n * mean(H_i^2)^2 )

# compute the variance of hat_beta_1
var_b1 <- var( ( X - mean(X) ) * u ) / (100 * var(X)^2)
```

```
# print variances to the console
var_b0
```

```
## [1] 4.045066
```

```
var_b1
```

```
## [1] 0.03018694
```

Now let us assume that we do not know the true values of β_0 and β_1 and that it is not possible to observe the whole population. However, we can observe a random sample of n observations. Then, it would not be possible to compute the true parameters but we could obtain estimates of β_0 and β_1 from the sample data using OLS. However, we know that these estimates are outcomes of random variables themselves since the observations are randomly sampled from the population. Key Concept 4.4 describes their distributions for large n. When drawing a single sample of size n it is not possible to make any statement about these distributions. Things change if we repeat the sampling scheme many times and compute the estimates for each sample: using this procedure we simulate outcomes of the respective distributions.

To achieve this in R, we employ the following approach:

- We assign the number of repetitions, say 10000, to `reps` and then initialize a matrix `fit` were the estimates obtained in each sampling iteration shall be stored row-wise. Thus `fit` has to be a matrix of dimensions `reps`$\times 2$.
- In the next step we draw `reps` random samples of size `n` from the population and obtain the OLS estimates for each sample. The results are stored as row entries in the outcome matrix `fit`. This is done using a `for()` loop.
- At last, we estimate variances of both estimators using the sampled outcomes and plot histograms of the latter. We also add a plot of the density functions belonging to the distributions that follow from Key Concept 4.4. The function `bquote()` is used to obtain math expressions in the titles and labels of both plots. See `?bquote`.

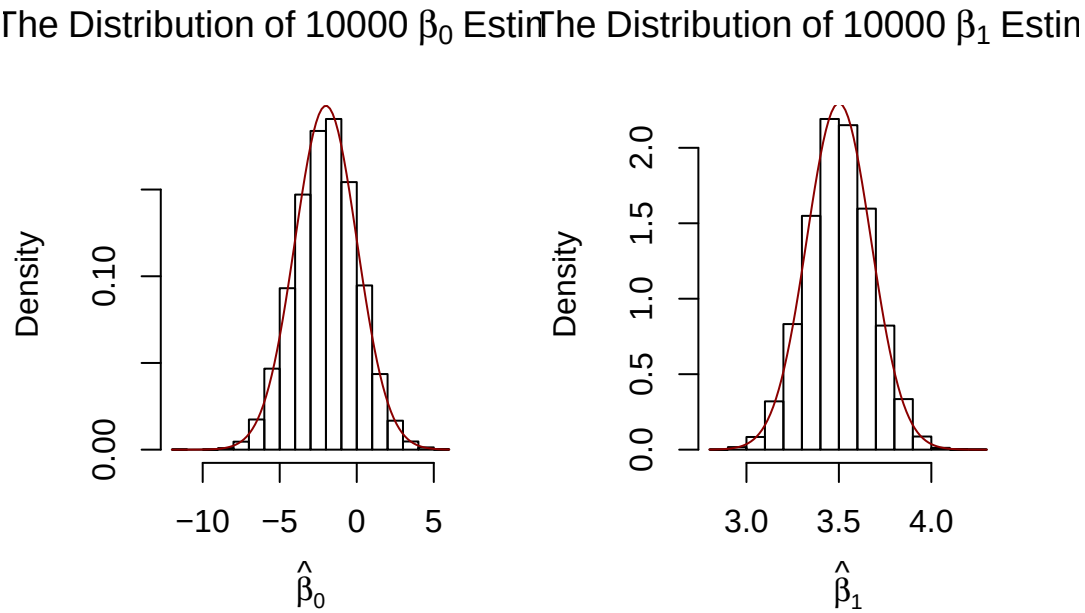
```
# set repetitions and sample size
n <- 100
reps <- 10000

# initialize the matrix of outcomes
fit <- matrix(ncol = 2, nrow = reps)

# loop sampling and estimation of the coefficients
for (i in 1:reps){
 sample <- population[sample(1:N, n), ]
 fit[i, ] <- lm(Y ~ X, data = sample)$coefficients
}

# compute variance estimates using outcomes
var(fit[, 1])
```

```
## [1] 4.057089
```

```
var(fit[, 2])
```

```
## [1] 0.03021784
```

```
# divide plotting area as 1-by-2 array
par(mfrow = c(1, 2))

# plot histograms of beta_0 estimates
hist(fit[ ,1],
     main = bquote(The ~ Distribution  ~ of ~ 10000 ~ beta[0] ~ Estimates),
     xlab = bquote(hat(beta)[0]),
     freq = F)

# add true distribution to plot
curve(dnorm(x, -2, sqrt(var_b0)), add = T, col = "darkred")

# plot histograms of beta_hat_1
hist(fit[ ,2],
     main = bquote(The ~ Distribution  ~ of ~ 10000 ~ beta[1] ~ Estimates),
     xlab = bquote(hat(beta)[1]),
     freq = F)

# add true distribution to plot
curve(dnorm(x, 3.5, sqrt(var_b1)), add = T, col = "darkred")
```



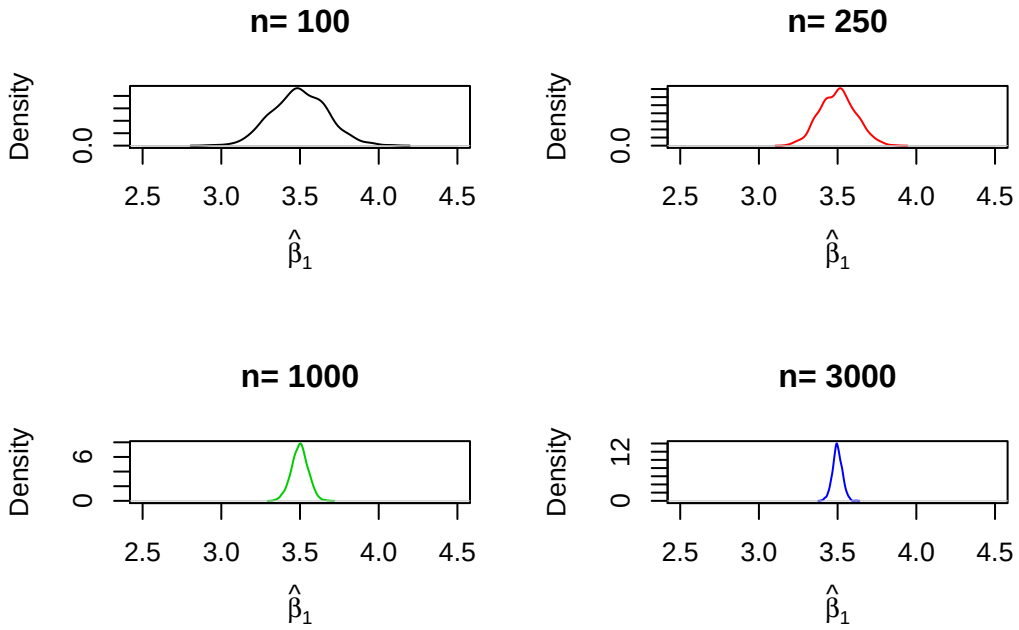
Our variance estimates support the statements made in Key Concept 4.4, coming close to the theoretical values. The histograms suggest that the distributions of the estimators can be well approximated by the respective theoretical normal distributions stated in Key Concept 4.4.

Simulation Study 2

A further result implied by Key Concept 4.4 is that both estimators are consistent, i.e., they converge in probability to the true parameters we are interested in. This is beacuase they are asymptotically unbiased

and their variances converge to 0 as n increases. We can check this by repeating the simulation above for a sequence of increasing sample sizes. This means we no longer assign the sample size but a *vector* of sample sizes: `n <- c(...)`. Let us look at the distributions of β_1. The idea here is to add an additional call of `for()` to the code. This is done in order to loop over the vector of sample sizes `n`. For each of the sample sizes we carry out the same simulation as before but plot a density estimate for the outcomes of each iteration over `n`. Notice that we have to change `n` to `n[j]` in the inner loop to ensure that the `j`th element of `n` is used. In the simulation, we use sample sizes of $100, 250, 1000$ and 3000. Consequently we have a total of four distinct simulations using different sample sizes.

```
# set random seed for reproducibility
set.seed(1)

# set repetitions and the vector of sample sizes
reps <- 1000
n <- c(100, 250, 1000, 3000)

# initialize the matrix of outcomes
fit <- matrix(ncol = 2, nrow = reps)

# divide the plot panel in a 2-by-2 array
par(mfrow = c(2, 2))

# loop sampling and plotting

# outer loop over n
for (j in 1:length(n)) {

  # inner loop: sampling and estimating of the coefficients
  for (i in 1:reps){
    sample <- population[sample(1:N, n[j]), ]
    fit[i, ] <- lm(Y ~ X, data = sample)$coefficients
  }

  # draw density estimates
  plot(density(fit[ ,2]), xlim=c(2.5, 4.5), col = j,
       main = paste("n=", n[j]), xlab = bquote(hat(beta)[1]))

}
```



We find that, as n increases, the distribution of $\hat{\beta}_1$ concentrates around its mean, i.e. its variance decreases. Put differently, the likelihood of observering estimates close to the true value of $\beta_1 = 3.5$ grows as we increase the sample size. The same behavior can be observed if we analyze the distribution of $\hat{\beta}_0$ instead.

Simulation Study 3

Furthermore, (4.1) reveals that the variance of the OLS estimator for β_1 decreases as the variance of the X_i increases. In other words, as we increase the amount of information provided by the regressor, that is, increasing $Var(X)$, which is used to estimate β_1, we become more confident that the estimate is close to the true value (i.e. $Var(\hat{\beta}_1)$ decreases). We can visualize this by reproducing Figure 4.6 from the book. To do this, we sample observations (X_i, Y_i), $i = 1, \dots, 100$ from a bivariate normal distribution with

$$E(X) = E(Y) = 5,$$
$$Var(X) = Var(Y) = 5$$

and

$$Cov(X, Y) = 4.$$

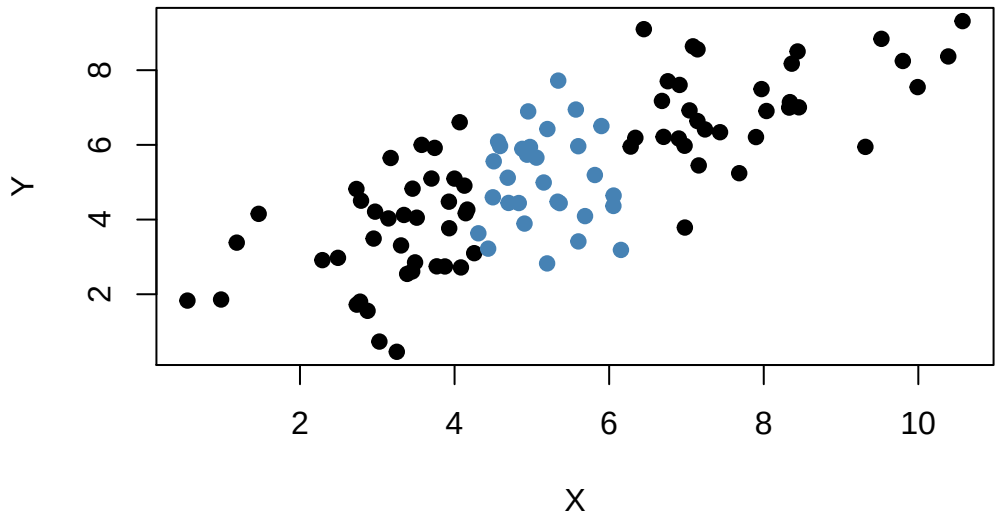
Formally, this is written down as

$$\begin{pmatrix} X \\ Y \end{pmatrix} \overset{i.i.d.}{\sim} \mathcal{N}\left[\begin{pmatrix} 5 \\ 5 \end{pmatrix}, \begin{pmatrix} 5 & 4 \\ 4 & 5 \end{pmatrix} \right]. \tag{4.3}$$

To carry out the random sampling, we make use of the function `mvrnorm()` from the package `MASS` (**?**) which allows to draw random samples from multivariate normal distributions, see `?mvtnorm`. Next, we use `subset()` to split the sample into two subsets such that the first set, `set1`, consists of observations that fulfill the condition $|X - \overline{X}| > 1$ and the second set, `set2`, includes the remainder of the sample. We then plot both sets and use different colors to make the observations distinguishable.

```
# load the MASS package
library(MASS)
```

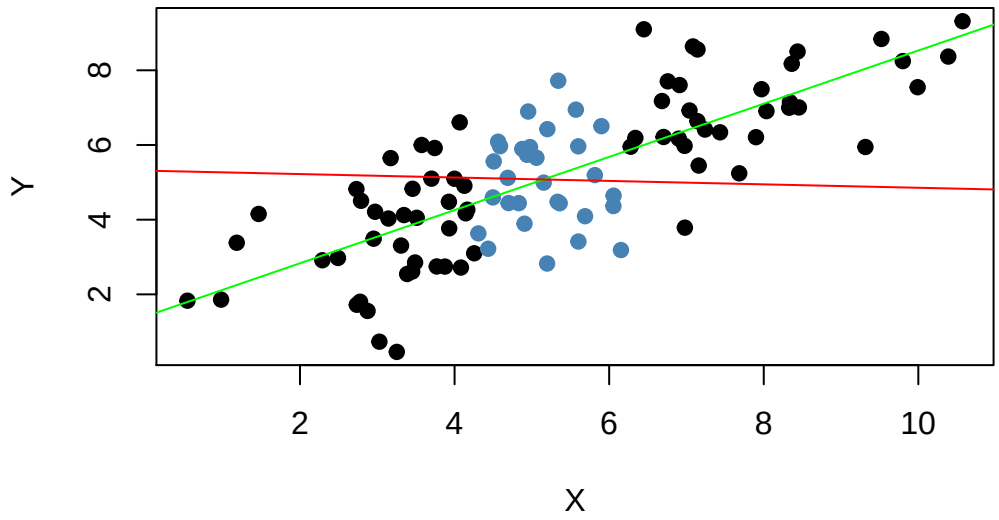
```
# set random seed for reproducibility
set.seed(4)

# simulate bivarite normal data
bvndata <- mvrnorm(100,
                   mu = c(5, 5),
                   Sigma = cbind(c(5, 4), c(4, 5))
                   )

# assign column names / convert to data.frame
colnames(bvndata) <- c("X", "Y")
bvndata <- as.data.frame(bvndata)

# subset the data
set1 <- subset(bvndata, abs(mean(X) - X) > 1)
set2 <- subset(bvndata, abs(mean(X) - X) <= 1)

# plot both data sets
plot(set1, xlab = "X", ylab = "Y", pch = 19)
points(set2, col = "steelblue", pch = 19)
```



It is clear that observations that are close to the sample average of the X_i have less variance than those that are farther away. Now, if we were to draw a line as accurately as possible through either of the two sets it is obvious that choosing the observations indicated by the black dots, i.e. using the set of observations which has larger variance than the blue ones, would result in a more precise line. Now, let us use OLS to estimate slope and intercept for both sets of observations. We then plot the observations along with both regression lines.

```
# estimate both regression lines
lm.set1 <- lm(Y ~ X, data = set1)
lm.set2 <- lm(Y ~ X, data = set2)

# plot observations
plot(set1, xlab = "X", ylab = "Y", pch = 19)
points(set2, col = "steelblue", pch = 19)

# add both lines to the plot
abline(lm.set1, col = "green")
```

```
abline(lm.set2, col = "red")
```



Evidently, the green regression line does far better in describing data sampled from the bivariate normal distribution stated in (4.3) than the red line. This is a nice example for demonstrating why we are interested in a high variance of the regressor X: more variance in the X_i means more information from which the precision of the estimation benefits.

2.6 Exercises

This interactive part of URFITE is only available in the HTML version.

Bibliography

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., and Chang, W. (2018). *rmarkdown: Dynamic Documents for R*. R package version 1.9.

Heiss, F. (2016). *Using R for Introductory Econometrics*. CreateSpace Independent Publishing Platform.

Kleiber, C. and Zeileis, A. (2008). *Applied econometrics with R*. Springer.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Stock, J. and Watson, M. (2015). *Introduction to Econometrics, Third Update, Global Edition*. Pearson Education Limited.

Wooldridge, J. (2016). *Introductory econometrics*. Cengage Learning, sixth edition.

Xie, Y. (2018). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.20.