Using R for Introduction to Econometrics

Christoph Hanck, Martin Arnold, Alexander Gerber and Martin Schmelzer

2018-05-29

Contents

Chapter 1

Introduction

The interest in the freely available statistical programming language and software environment `R` 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 but 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.

A striking advantage of using `R` in econometrics courses 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 the results.

Over the recent years, `R` has thus become an integral part of the curricula of econometrics classes we teach at 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. Of course, presenting bare `R` code chunks on

slides has mostly a deterring effect for the students to engage with programming on their own. This is why we offer tutorials where both econometric theory and its applications using R are introduced, for some time now. As for accompanying literature, there are some excellent books that deal with R and its applications to econometrics like Kleiber & Zeilis (2008) and Hetekar (2010). However, we have found that these works are somewhat difficult to access, especially for undergraduate students in economics having little understanding of econometric methods and predominantly no experience in programming at all. Consequently, we have 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* by Stock & Watson (2014) which serves as a basis for the lecture and the accompanying tutorials. The process has been facilitated considerably with the release of `knitr` (2018) in 2012. `knitr` is an R package for dynamic report generation which allows 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 decided to write up our own empirical companion to Stock & Watson (2014) which resulted in **U**sing **R** f**o**r **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. *URFITE* is best described as an interactive script in the style of a reproducible research report which aims to provide students of economic sciences with a platform-independent e-learning arrangement by seamlessly intertwining theoretical core knowledge and empirical skills in undergraduate econometrics. Of course the focus is set on empirical applications with R; we leave out tedious derivations and formal proofs wherever we can. *URFITE* is closely aligned on Stock & Watson (2014) which does very well in motivating theory by real-world applications. However, we take it a step further and 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. To support 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) arbitrary 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] 12564
```

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 (2015) 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

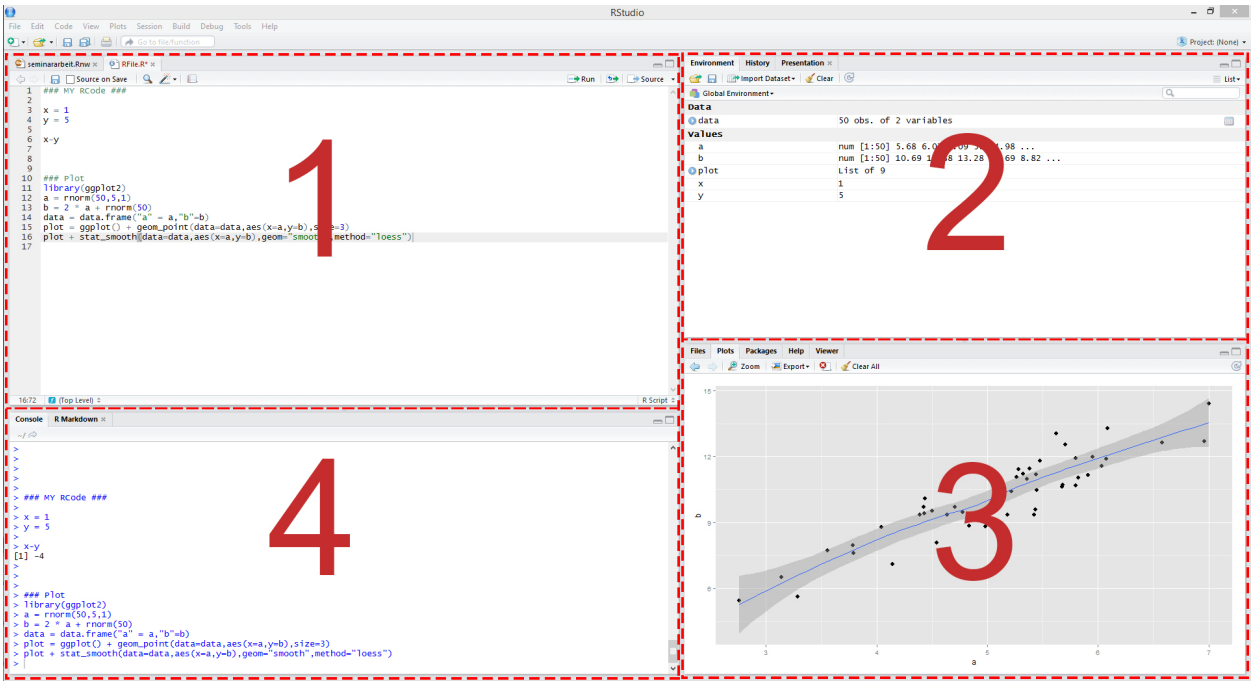


Figure 1.1: *RStudio*: the four panes

As you might have noticed, there are some out-commented lines in the widget that 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 prompt 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.

1.1 A Very Short Introduction to R and *RStudio*

R Basics

This section is meant for those who have never worked with R or *RStudio*. If you at least know how to create objects and call functions, you can skip it. If you would like to refresh your memories 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 send 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 the 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 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 wise 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, function calls look all 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 but naming it `z`. The function takes on the arguments `from`, `to` and `by` which should be self-explaining. The function `mean()` computes the arithmetic mean of its argument `x`. Since we pass the vector `z` as the argument `x` to `mean()`, the result is `3`!

If you are not sure what argument a function expects you may consult the function's documentation. Let's say we are not sure how the arguments required for `seq()` work. Then we can type `?seq` in the console and 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.

Chapter 2

Probability Theory

This chapter reviews some basic concepts of probability theory and demonstrates how they can be applied in R.

Most of the statistical functionalities in R's standard version are collected in the `stats` package. It provides simple functions which compute descriptive measures and facilitate calculus involving a variety of probability distributions. However, it also holds more sophisticated routines that e.g. enable the user to estimate a large number of models based on the same data or help to conduct extensive simulation studies. `stats` is part of the base distribution of R, meaning that it is installed by default so there is no need to run `install.packages("stats")` or `library("stats")`. Simply execute `library(help = "stats")` in the console to view the documentation and a complete list of all functions gathered in `stats`.

In what follows, we lay our focus on (some of) the probability distributions that are handled by R and show how to use the relevant functions to solve simple problems. Thereby we will repeat some core concepts of probability theory. Among other things, you will learn how to draw random numbers, how to compute densities, probabilities, quantiles and alike. As we shall see, it is very convenient to rely on these routines, especially when writing your own functions.

2.1 Random Variables and Probability Distributions

For a start, let us briefly review some basic concepts of probability theory.

- The mutually exclusive results of a random process are called the *outcomes*. 'Mutually exclusive' means that only one of the possible outcomes is observed.
- We refer to the *probability* of an outcome as the proportion of the time that the outcome occurs in the long run, that is if the experiment is repeated very often.
- The set of all possible outcomes of a random variable is called the *sample space*.
- An *event* is a subset of the sample space and consists of one or more outcomes.

These ideas are unified in the concept of a *random variable* which is a numerical summary of random outcomes. Random variables can be *discrete* or *continuous*.

- Discrete random variables have discrete outcomes, e.g. 0 and 1.
- A continuous random variable takes on a continuum of possible values.

Probability Distributions of Discrete Random Variables

A typical example for a discrete random variable D is the result of a die roll: in terms of a random experiment this is nothing but randomly selecting a sample of size 1 from a set of numbers which are mutually exclusive

outcomes. Here, the sample space is $\{1, 2, 3, 4, 5, 6\}$ and we can think of many different events, e.g. 'the observed outcome lies between 2 and 5'.

A basic function to draw random samples from a specified set of elements is the the function `sample()`, see `?sample`. We can use it to simulate the random outcome of a die roll. Let's role the die!

```
sample(1:6, 1)
```

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

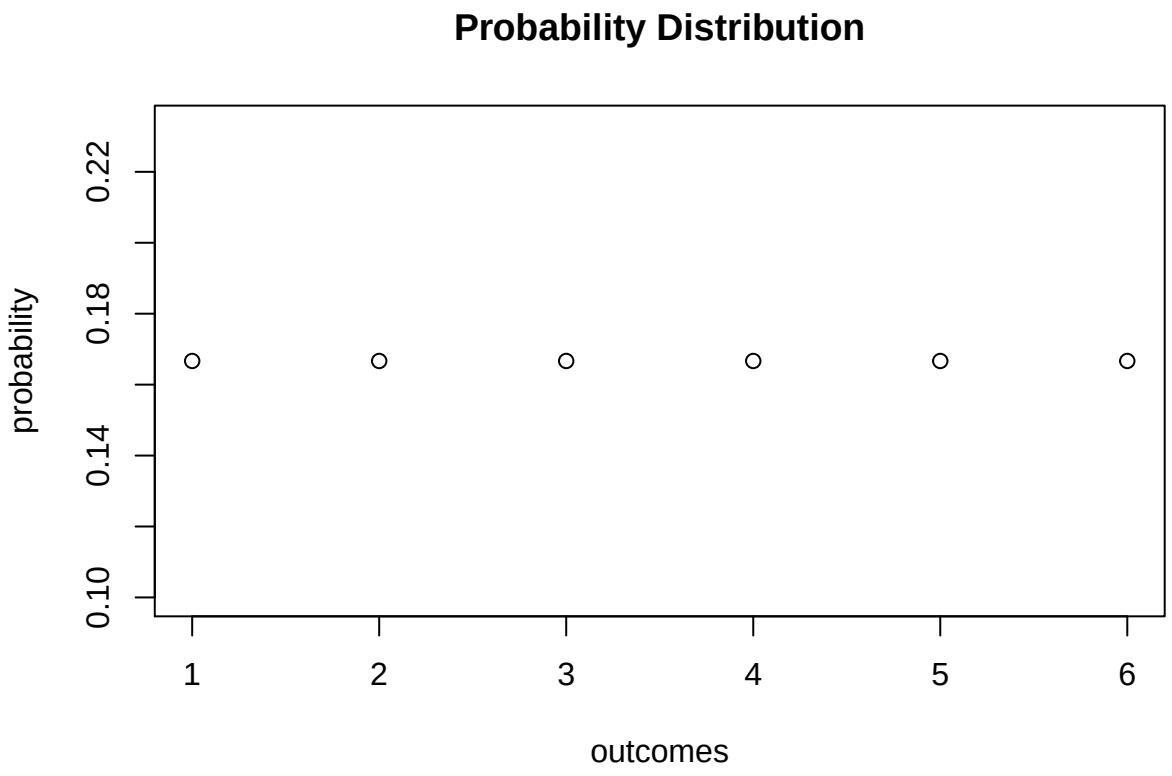
The probability distribution of a discrete random variable is the list of all possible values of the variable and their probabilities which sum to 1. The cumulative probability distribution function states the probability that the random variable is less than or equal to a particular value.

For the die roll, this is straightforward to set up

Outcome	1	2	3	4	5	6
Probability distribution	1/6	1/6	1/6	1/6	1/6	1/6
Cumulative probability distribution	1/6	2/6	3/6	4/6	5/6	1

We can easily plot both functions using R. Since the probability equals 1/6 for each outcome, we set up the vector `probability` by using the `rep()` function which replicates a given value a specified number of times.

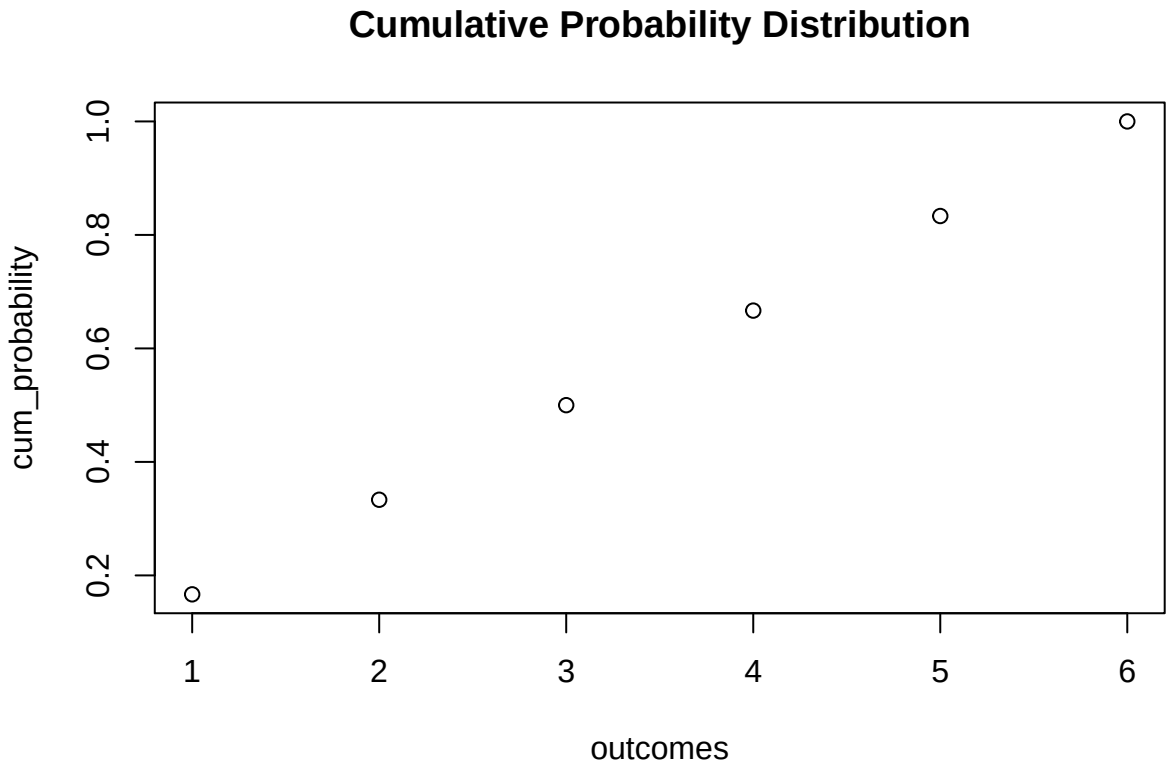
```
# generate the vector of probabilities
probability <- rep(1/6,6)

# plot the probabilites
plot(probability, xlab = "outcomes",
     main = "Probability Distribution"
     )
```



For the cumulative probability distribution we need the cumulative probabilities i.e. we need the cumulative

sums of the vector `probability`. These sums can be computed using `cumsum()`.

```
#generate the vector of cumulative probabilities
cum_probability <- cumsum(probability)

# plot the probabilites
plot(cum_probability,
     xlab = "outcomes",
     main = "Cumulative Probability Distribution"
     )
```



Bernoulli Trials

The set of elements from which `sample()` draws outcomes does not have to consist of numbers only. We might as well simulate coin tossing with outcomes H (heads) and T (tails).

```
sample(c("H","T"),1)
```

```
## [1] "T"
```

The result of a single coin toss is a *Bernoulli* distributed random variable i.e. a variable with two possible distinct outcomes.

Imagine you are about to toss a coin 10 times in a row and wonder how likely it is to end up with a sequence of outcomes like

$$H\ H\ T\ T\ T\ H\ T\ T\ H\ H.$$

This is a typical example of what we call a *Bernoulli experiment* as it consists of $n = 10$ Bernoulli trials that are independent of each other and we are interested in the likelihood of observing $k = 5$ successes H that

occur with probability $p = 0.5$ (assuming a fair coin) in each trial. Note that the order of the outcomes does not matter here.

It is a well known result that the number of successes k in a Bernoulli experiment follows a binomial distribution. We denote this as

$$k \sim B(n, p).$$

The probability of observing k successes in the experiment $B(n, p)$ is given by

$$f(k) = P(k) = \binom{n}{k} \cdot p^k \cdot q^{n-k} = \frac{n!}{k!(n-k)!} \cdot p^k \cdot q^{n-k}$$

where $\binom{n}{k}$ means the binomial coefficient.

In `R`, we can solve the problem stated above by means of the function `dbinom()` which calculates the probability of the binomial distribution given the parameters `x`, `size`, and `prob`, see `?binom`.

```
dbinom(x = 5,
       size = 10,
       prob = 0.5
       )
```

```
## [1] 0.2460938
```

We conclude that the probability of observing Head $k = 5$ times when tossing the coin $n = 10$ times is about 24.6%.

Now assume we are interested in $P(4 \leq k \leq 7)$ i.e. the probability of observing 4, 5, 6 or 7 successes for $B(10, 0.5)$. This is easily computed by providing a vector as the argument `x` in our call of `dbinom()` and summing up using `sum()`.

```
sum(
  dbinom(x = 4:7,
         size = 10,
         prob = 0.5
         )
  )
```

```
## [1] 0.7734375
```

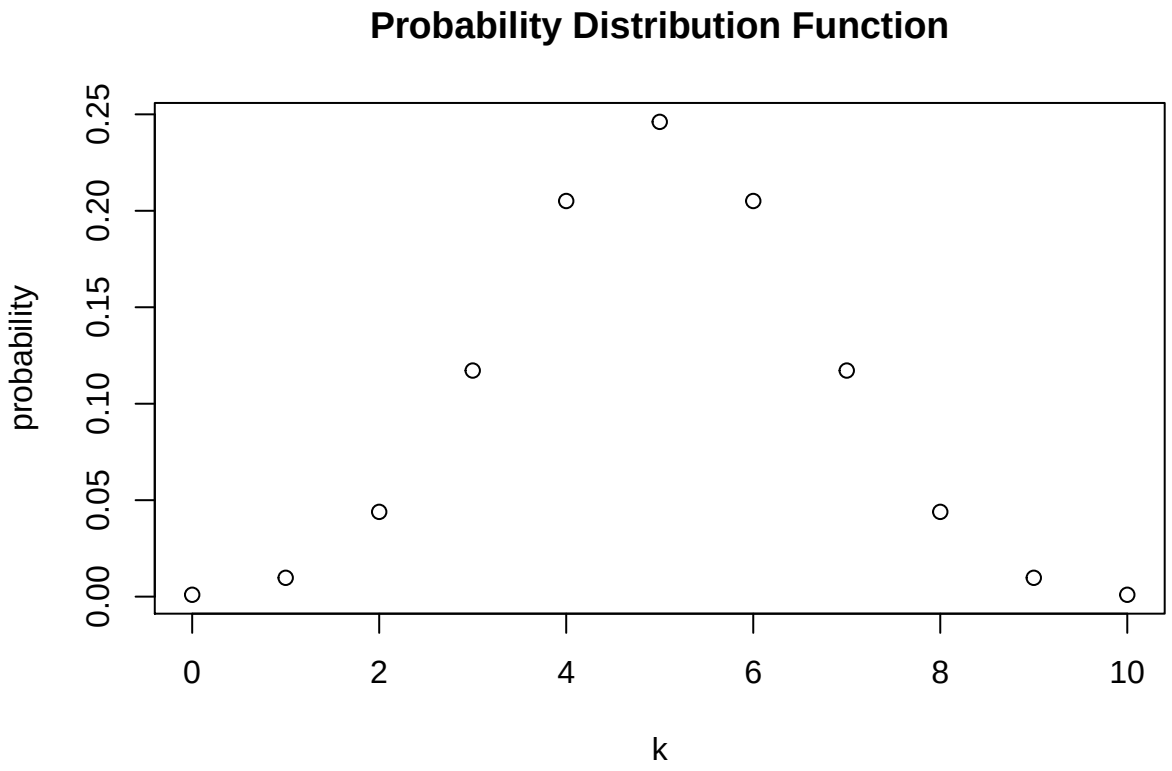
The probability distribution of a discrete random variable is nothing but a list of all possible outcomes that can occur and their respective probabilities. In the coin tossing example we face 11 possible outcomes for k.

```
# set up vector of possible outcomes
k <- 0:10
```

To visualize the probability distribution function of k we may therefore do the following:

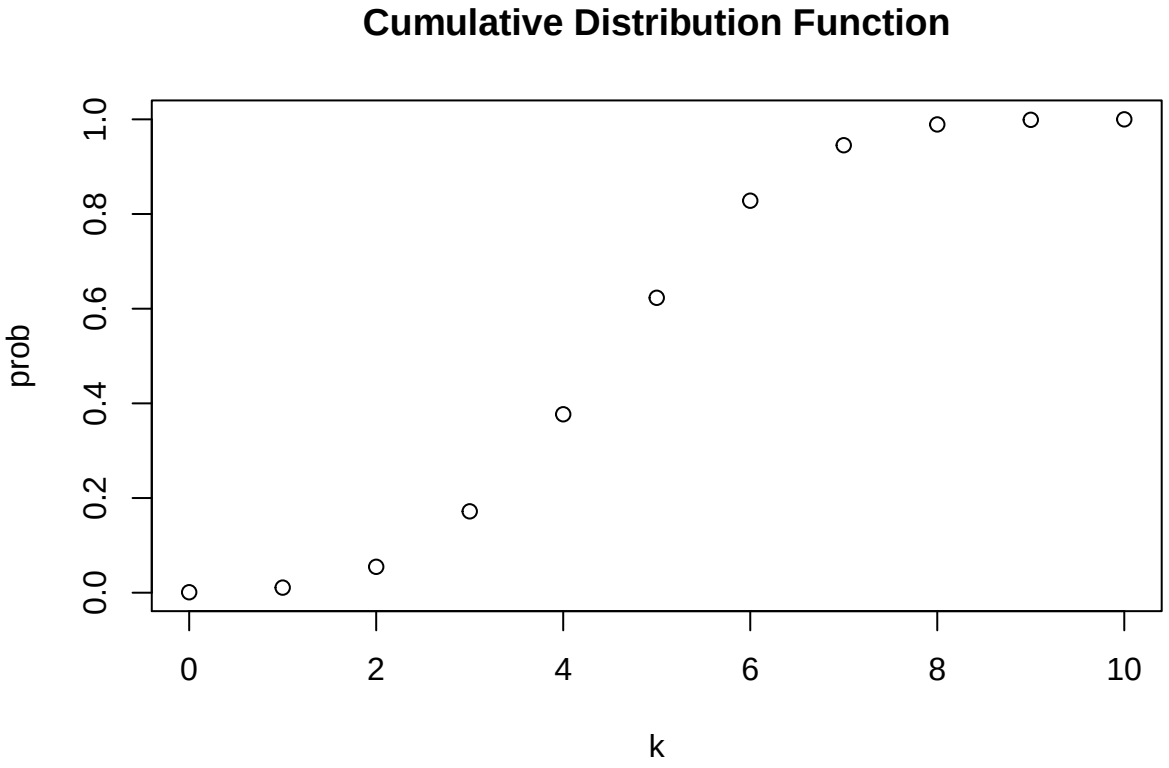
```
# assign probabilities
probability <- dbinom(x = k,
                      size = 10,
                      prob = 0.5
                      )

# plot outcomes against probabilities
plot(x = k,
     y = probability,
```

```
     main = "Probability Distribution Function"
     )
```



In a similar fashion we may plot the cumulative distribution function of k by executing the following code chunk:

```
# compute cumulative probabilities
prob <- cumsum(
              dbinom(x = 0:10,
                     size = 10,
                     prob = 0.5
                     )
              )

# plot the cumulative probabilities
plot(x = k,
     y = prob,
     main = "Cumulative Distribution Function"
     )
```



Expected Value, Mean and Variance

The expected value of a random variable is the long-run average value of its outcomes when the number of repeated trials is large. For a discrete random variable, the expected value is computed as a weighted average of its possible outcomes whereby the weights are the related probabilities. This is formalized in Key Concept 2.1.

Key Concept 2.1

Expected Value and the Mean

Suppose the random variable Y takes on k possible values, $y_1, \ldots, y_k$, where y_1 denotes the first value, y_2 denotes the second value, and so forth, and that the probability that Y takes on y_1 is p_1, the probability that Y takes on y_2 is p_2 and so forth. The expected value of Y, $E(Y)$ is defined as

$$E(Y) = y_1 p_1 + y_2 p_2 + \cdots + y_k p_k = \sum_{i=1}^{k} y_i p_i$$

where the notation $\sum_{i=1}^{k} y_i p_i$ means "the sum of y_i p_i for i running from 1 to k". The expected value of Y is also called the mean of Y or the expectation of Y and is denoted by μ_y.

In the die example, the random variable, D say, takes on 6 possible values $d_1 = 1, d_2 = 2, \ldots, d_6 = 6$. Assuming a fair die, each of the 6 outcomes occurs with a probability of 1/6. It is therefore easy to calculate the exact value of $E(D)$ by hand:

$$E(D) = 1/6 \sum_{i=1}^{6} d_i = 3.5$$

$E(D)$ is simply the average of the natural numbers from 1 to 6 since all wights p_i are 1/6. Convince yourself that this can be easily calculated using the function `mean()` which computes the arithmetic mean of a numeric vector.

```
mean(1:6)
```

```
## [1] 3.5
```

An example of sampling with replacement is rolling a die three times in a row.

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

# rolling a die three times in a row
sample(1:6, 3, replace = T)
```

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

Of course we could also consider a much bigger number of trials, 10000 say. Doing so, it would be pointless to simply print the results to the console: by default `R` displays up to 1000 entries of large vectors and omits the remainder (give it a go). Eyeballing the numbers does not reveal too much. Instead let us calculate the sample average of the outcomes using `mean()` and see if the result comes close to the expected value $E(D) = 3.5$.

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

# compute the sample mean of 10000 die rolls
mean(
    sample(1:6,
           10000,
           replace = T
           )
    )
```

```
## [1] 3.5039
```

We find the sample mean to be fairly close to the expected value. This result will be discussed in Chapter 2.3 in more detail.

Other frequently encountered measures are the variance and the standard deviation. Both are measures of the *dispersion* of a random variable.

Key Concept 2.2

Variance and Standard Deviation

The Variance of the discrete *random variable* Y, denoted σ_Y^2, is

$$\sigma_Y^2 = \text{Var}(Y) = E\left[(Y - \mu_y)^2\right] = \sum_{i=1}^{k}(y_i - \mu_y)^2 p_i$$

The standard deviation of Y is σ_Y, the square root of the variance. The units of the standard deviation are the same as the units of Y.

The variance as defined in Key Concept 2.2 *is not* implemented as a function in R. Instead we have the function `var()` which computes the *sample variance*

$$s_Y^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2.$$

Remember that s_Y^2 is different from the so called *population variance* of Y,

$$\text{Var}(Y) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu_Y)^2,$$

since it measures how the data is dispersed around the sample average $\overline{y}$ instead of the population mean μ_Y. This becomes clear when we look at our die rolling example. For D we have

$$\text{Var}(D) = 1/6\sum_{i=1}^{6}(d_i - 3.5)^2 = 2.92$$

which is obviously different from the result of s^2 as computed by `var()`.

```
var(1:6)
```

```
## [1] 3.5
```

The sample variance as computed by `var()` is an *estimator* of the population variance.

You may use this widget to play around with the functions presented above

2.2 Probability Distributions of Continuous Random Variables

Since a continuous random variable takes on a continuum of possible values, we cannot use the concept of a probability distribution as used for discrete random variables. Instead, the probability distribution of a continuous random variable is summarized by its *probability density function* (PDF).

The cumulative probability distribution function (CDF) for a continuous random variable is defined just as in the discrete case. Hence, the cumulative probability distribution of a continuous random variables states the probability that the random variable is less than or equal to a particular value.

For completeness, we present revisions of Key Concepts 2.1 and 2.2 for the continuous case.

Key Concept 2.3

Probabilities, Expected Value and Variance of a Continuous Random Variable

Let $f_Y(y)$ denote the probability density function of Y. Because probabilities cannot be negative, we have $f_Y \geq 0$ for all y. The Probability that Y falls between a and b where $a < b$ is

$$P(a \leq Y \leq b) = \int_a^b f_Y(y)\text{d}y.$$

We further have that $P(-\infty \leq Y \leq \infty) = 1$ and therefore $\int_{-\infty}^{\infty} f_Y(y)\text{d}y = 1$.

As for the discrete case, the expected value of Y is the probability weighted average of its values. Due to continuity, we use integrals instead of sums.

The expected value of Y is defined as

$$E(Y) = \mu_Y = \int y f_Y(y)\text{d}y.$$

The variance is the expected value of $(Y - \mu_Y)^2$. We thus have

$$\text{Var}(Y) = \sigma_Y^2 = \int (y - \mu_Y)^2 f_Y(y)\text{d}y.$$

Let us discuss an example:

Consider the continuous random variable X with probability density function

$$f_X(x) = \frac{3}{x^4}, x > 1.$$

- We can show analytically that the integral of $f_X(x)$ over the real line equals 1.

$$\begin{aligned}
\int f_X(x)\mathrm{d}x &= \int_1^\infty \frac{3}{x^4}\mathrm{d}x && (2.1)\\
&= \lim_{t\to\infty} \int_1^t \frac{3}{x^4}\mathrm{d}x && (2.2)\\
&= \lim_{t\to\infty} -x^{-3}|_{x=1}^t && (2.3)\\
&= -\left(\lim_{t\to\infty} \frac{1}{t^3} - 1\right) && (2.4)\\
&= 1 && (2.5)
\end{aligned}$$

- The expectation of X can be computed as follows:

$$\begin{aligned}
E(X) = \int x \cdot f_X(x)\mathrm{d}x &= \int_1^\infty x \cdot \frac{3}{x^4}\mathrm{d}x && (2.6)\\
&= -\frac{3}{2}x^{-2}|_{x=1}^\infty && (2.7)\\
&= -\frac{3}{2}\left(\lim_{t\to\infty} \frac{1}{t^2} - 1\right) && (2.8)\\
&= \frac{3}{2} && (2.9)
\end{aligned}$$

- Note that the variance of X can be expressed as $\mathrm{Var}(X) = E(X^2) - E(X)^2$. Since $E(X)$ has been computed in the previous step, we seek $E(X^2)$:

$$\begin{aligned}
E(X^2) = \int x^2 \cdot f_X(x)\mathrm{d}x &= \int_1^\infty x^2 \cdot \frac{3}{x^4}\mathrm{d}x && (2.10)\\
&= -3x^{-1}|_{x=1}^\infty && (2.11)\\
&= -3\left(\lim_{t\to\infty} \frac{1}{t} - 1\right) && (2.12)\\
&= 3 && (2.13)
\end{aligned}$$

So we have shown that the area under the curve equals one, that the expectation is $E(X) = \frac{3}{2}$ and we found the variance to be $\mathrm{Var}(X) = \frac{3}{4}$. However, this was quite tedious and, as we shall see soon, an analytic approach is not applicable for some probability density functions e.g. if integrals have no closed form solutions.

Luckily, `R` enables us to find the results derived above in an instant. The tool we use for this is the function `integrate()`. First, we have to define the functions we want to calculate integrals for as `R` functions, i.e. the PDF $f_X(x)$ as well as the expressions $x \cdot f_X(x)$ and $x^2 \cdot f_X(x)$.

```
# define functions
f <- function(x) 3/x^4
g <- function(x) x*f(x)
h <- function(x) x^2*f(x)
```

Next, we use `integrate()` and set lower and upper limits of integration to 1 and ∞ using arguments `lower` and `upper`. By default, `integrate()` prints the result along with an estimate of the approximation error to the console. However, the outcome is not a numeric value one can do further calculation with readily. In order to get only a numeric value of the integral, we need to use the $ operator in conjunction with `value`.

```
# calculate area under curve
AUC <- integrate(f,
                 lower = 1,
                 upper = Inf
                 )
AUC
```

```
## 1 with absolute error < 1.1e-14
```

```
# calculate E(X)
EX <- integrate(g,
                lower = 1,
                upper = Inf
                )
EX
```

```
## 1.5 with absolute error < 1.7e-14
```

```
# calculate Var(X)
VarX <- integrate(h,
                  lower = 1,
                  upper = Inf
                  )$value - EX$value^2
VarX
```

```
## [1] 0.75
```

Although there is a wide variety of distributions, the ones most often encountered in econometrics are the normal, chi-squared, Student t and F distributions. Therefore we will discuss some core `R` functions that allow to do calculations involving densities, probabilities and quantiles of these distributions.

Every probability distribution that `R` handles has four basic functions whose names consist of a prefix followed by a root name. As an example, take the normal distribution. The root name of all four functions associated with the normal distribution is `norm`. The four prefixes are

- `d` for "density" - probability function / probability density function
- `p` for "probability" - cumulative distribution function
- `q` for "quantile" - quantile function (inverse cumulative distribution function)
- `r` for "random" - random number generator

Thus, for the normal distribution we have the `R` functions `dnorm()`, `pnorm()`, `qnorm()` and `rnorm()`.

The Normal Distribution

The probably most important probability distribution considered here is the normal distribution. This is not least due to the special role of the standard normal distribution and the Central Limit Theorem which is treated shortly during the course of this section. Distributions of the normal family have a familiar symmetric, bell-shaped probability density. A normal distribution is characterized by its mean μ and its standard deviation σ what is concisely expressed by $N(\mu, \sigma^2)$. The normal distribution has the PDF

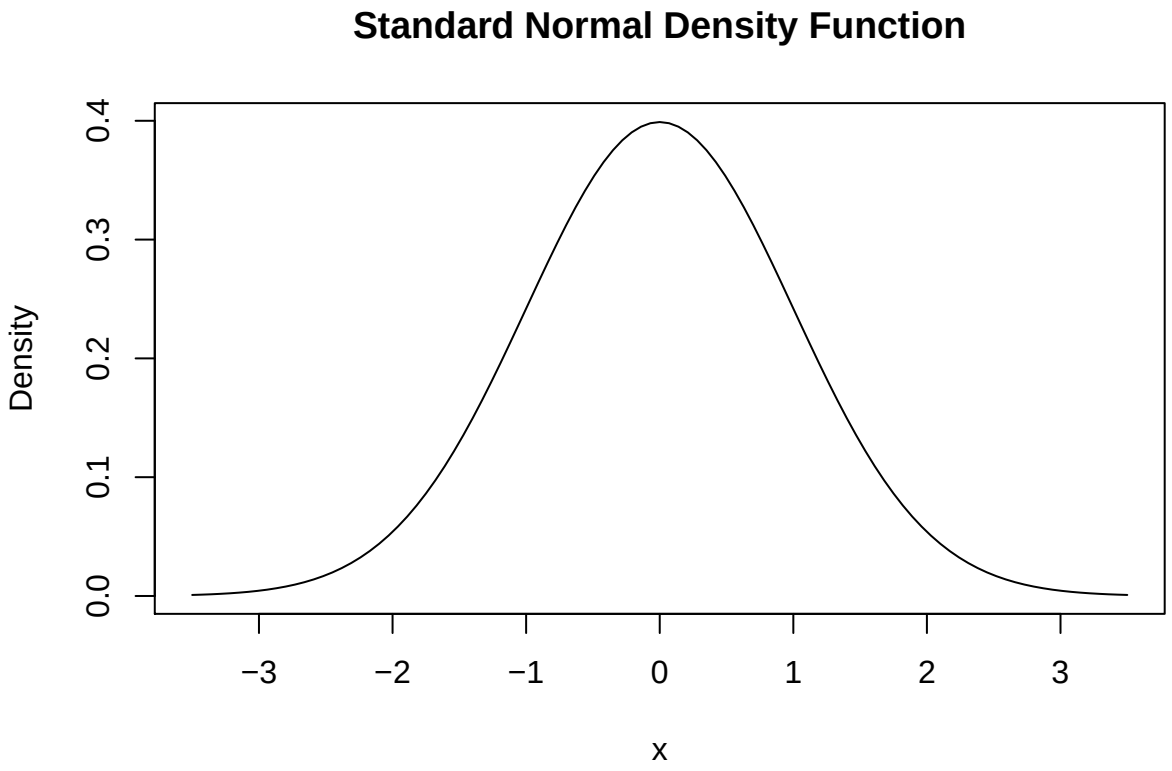
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-)^2/(2^2)}.$$

For the standard normal distribution we have $\mu = 0$ and $\sigma = 1$. Standard normal variates are often denoted by Z. Usually, the standard normal PDF is denoted by ϕ and the standard normal CDF is denoted by Φ. Hence,

$$\phi(c) = \Phi'(c) \quad , \quad \Phi(c) = P(Z \leq c) \quad , \quad Z \sim N(0,1).$$

In R, we can conveniently obtain density values of normal distributions using the function `dnorm()`. Let us draw a plot of the standard normal density function using `curve()` in conjunction with `dnorm()`.

```
# draw a plot of the N(0,1) PDF
curve(dnorm(x),
      xlim=c(-3.5, 3.5),
      ylab = "Density",
      main = "Standard Normal Density Function"
      )
```



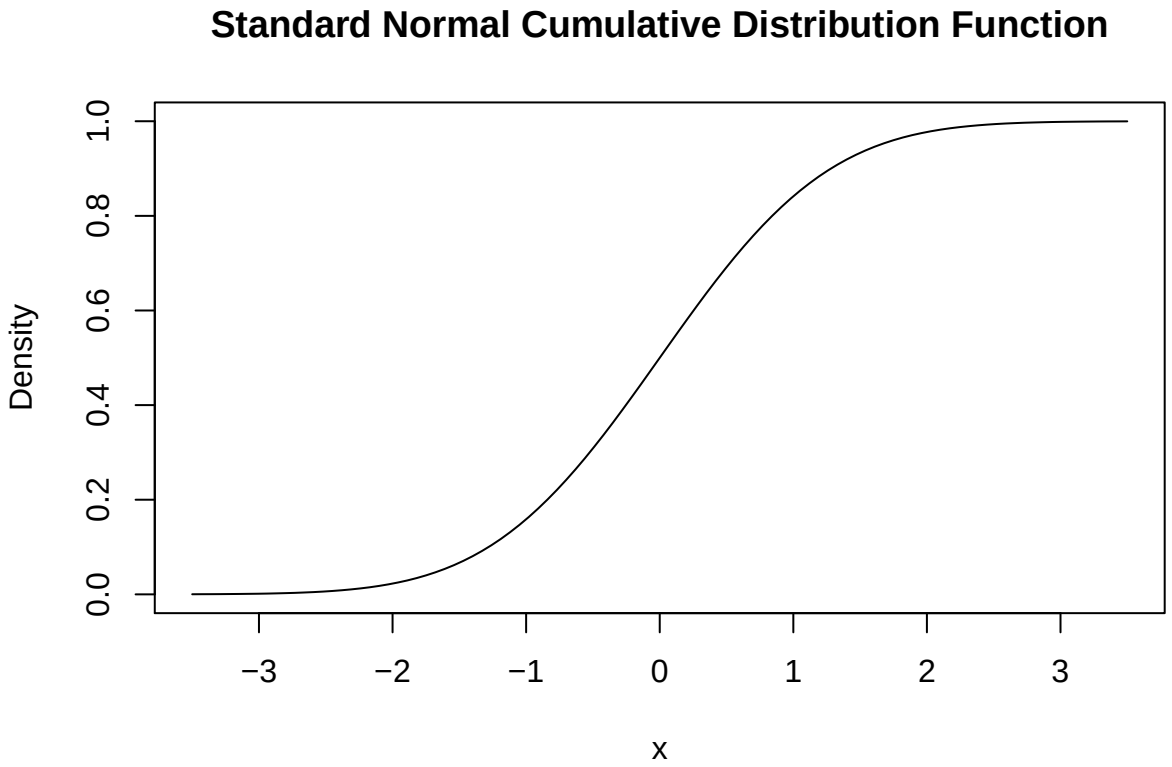
We can obtain the density at different positions by passing a vector of quantiles to `dnorm()`.

```
# compute denstiy at x=-1.96, x=0 and x=1.96
dnorm(x = c(-1.96, 0, 1.96))
```

```
## [1] 0.05844094 0.39894228 0.05844094
```

Similar to the PDF, we can plot the standard normal CDF using `curve()` and `pnorm()`.

```
# plot the standard normal CDF
curve(pnorm(x),
      xlim=c(-3.5, 3.5),
      ylab = "Density",
      main = "Standard Normal Cumulative Distribution Function"
      )
```



We can also use `R` to calculate the probability of events associated with a standard normal variate.

Let us say we are interested in $P(Z \leq 1.337)$. For some continuous random variable Z on $[-\infty, \infty]$ with density function $g(x)$ we would have to determine $G(x)$ which is the anti derivative of $g(x)$ so that

$$P(Z \leq 1,337) = G(1,337) = \int_{-\infty}^{1,337} g(x)\mathrm{d}x.$$

If $Z \sim N(0,1)$, we have $g(x) = \phi(x)$. There is no analytic solution to the integral above and it is cumbersome to come up with an approximation. However, we may circumvent this using `R` in different ways. The first approach makes use of the function `integrate()` which allows to solve one-dimensional integration problems using a numerical method. For this, we first define the function we want to compute the integral of as a `R` function `f`. In our example, `f` needs to be the standard normal density function and hence takes a single argument `x`. Following the definition of $\phi(x)$ we define `f` as

```
# define the standard normal PDF as a R function
f <- function(x) {
  1/(sqrt(2 * pi)) * exp(-0.5 * x^2)
}
```

Let us check if this function enables us to compute standard normal density values by passing it a vector of quantiles.

```
# define vector of quantiles
quants <- c(-1.96, 0, 1.96)

# compute density values
f(quants)
```

```
## [1] 0.05844094 0.39894228 0.05844094
```

```
# compare to results produced by dnorm()
f(quants) == dnorm(quants)
```

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

Notice that the results produced by `f()` are indeed equivalent to those given by `dnorm()`.

Next, we call `integrate()` on `f()` and specify the arguments `lower` and `upper`, the lower and upper limits of integration.

```
# integrate f()
integrate(f,
          lower = -Inf,
          upper = 1.337
          )
```

```
## 0.9093887 with absolute error < 1.7e-07
```

We find that the probability of observing $Z \leq 1.337$ is about 0.9094%.

A second and much more convenient way is to use the function `pnorm()` which also allows calculus involving the standard normal cumulative distribution function.

```
# compute the probability using pnorm()
pnorm(1.337)
```

```
## [1] 0.9093887
```

The result matches the outcome of the approach using `integrate()`.

Let us discuss some further examples:

A commonly known result is that 95% probability mass of a standard normal lies in the interval $[-1.96, 1.96]$, that is in a distance of about 2 standard deviations to the mean. We can easily confirm this by calculating

$$P(-1.96 \leq Z \leq 1.96) = 1 - 2 \times P(Z \leq -1.96)$$

due to symmetry of the standard normal PDF. Thanks to `R`, we can abandon the table of the standard normal CDF again and instead solve this by using the function `pnorm()`.

```
# compute the probability
1 - 2 * (pnorm(-1.96))
```

```
## [1] 0.9500042
```

Now consider a random variable Y with $Y \sim N(5, 25)$. As you should already know from your statistics courses it is not possible to make any statement of probability without prior standardizing as shown in Key Concept 2.4.

Key Concept 2.4

Computing Probabilities Involving Normal Random Variables

Suppose Y is normally distributed with mean μ and variance σ^2:

$$Y \sim N(\mu, \sigma^2)$$

Then Y is standardized by subtracting its mean and dividing by its standard deviation:

$$Z = \frac{Y - \mu}{\sigma}$$

Let c_1 and c_2 denote two numbers whereby $c_1 < c_2$ and further $d_1 = (c_1 - \mu)/\sigma$ and $d_2 = (c_2 - \mu)/\sigma$. Then

$$P(Y \leq c_2) = P(Z \leq d_2) = \Phi(d_2) \tag{2.14}$$
$$P(Y \geq c_1) = P(Z \geq d_1) = 1 - \Phi(d_1) \tag{2.15}$$
$$P(c_1 \leq Y \leq c_2) = P(d_1 \leq Z \leq d_2) = \Phi(d_2) - \Phi(d_1) \tag{2.16}$$

`R` functions that handle the normal distribution can perform this standardization. If we are interested in $P(3 \leq Y \leq 4)$ we can use `pnorm()` and adjust for a mean and/or a standard deviation that deviate from $\mu = 0$ and $\sigma = 1$ by specifying the arguments `mean` and `sd` accordingly. **Attention**: the argument `sd` requires the standard deviation, not the variance!

```
pnorm(4, mean = 5, sd = 5) - pnorm(3, mean = 5, sd = 5)
```

```
## [1] 0.07616203
```

An extension of the normal distribution in a univariate setting is the multivariate normal distribution. The PDF of two random normal variables X and Y is given by

$$g_{X,Y}(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho_{XY}^2}} \tag{2.17}$$
$$\cdot \exp\left\{\frac{1}{-2(1-\rho_{XY}^2)}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - 2\rho_{XY}\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]\right\}. \tag{2.18}$$

Equation (2.18) contains the bivariate normal PDF. Admittedly, it is hard to gain insights from this complicated expression. Instead, let us consider the special case where X and Y are uncorrelated standard normal random variables with density functions $f_X(x)$ and $f_Y(y)$ and we assume that they have a joint normal distribution. We then have the parameters $\sigma_X = \sigma_Y = 1$, $\mu_X = \mu_Y = 0$ (due to marginal standard normality) and $\rho_{XY} = 0$ (due to independence). The joint probability density function of X and Y then becomes

$$g_{X,Y}(x,y) = f_X(x)f_Y(y) = \frac{1}{2\pi} \cdot \exp\left\{-\frac{1}{2}\left[x^2 + y^2\right]\right\}, \tag{2.2}$$

the PDF of the bivariate standard normal distribution. The next plot provides an interactive three dimensional plot of (2.2). By moving the cursor over the plot you can see that the density is rotationally invariant.

The Chi-Squared Distribution

Another distribution relevant in econometric day-to-day work is the chi-squared distribution. It is often needed when testing special types of hypotheses frequently encountered when dealing with regression models.

The sum of M squared independent standard normal distributed random variables follows a chi-squared distribution with M degrees of freedom.

$$Z_1^2 + \cdots + Z_M^2 = \sum_{m=1}^{M} Z_m^2 \sim \chi_M^2 \quad \text{with} \quad Z_m \overset{i.i.d.}{\sim} N(0,1)$$

A χ^2 distributed random variable with M degrees of freedom has expectation M, mode at $M-2$ for $n \geq 2$ and variance $2 \cdot M$.

For example, if we have

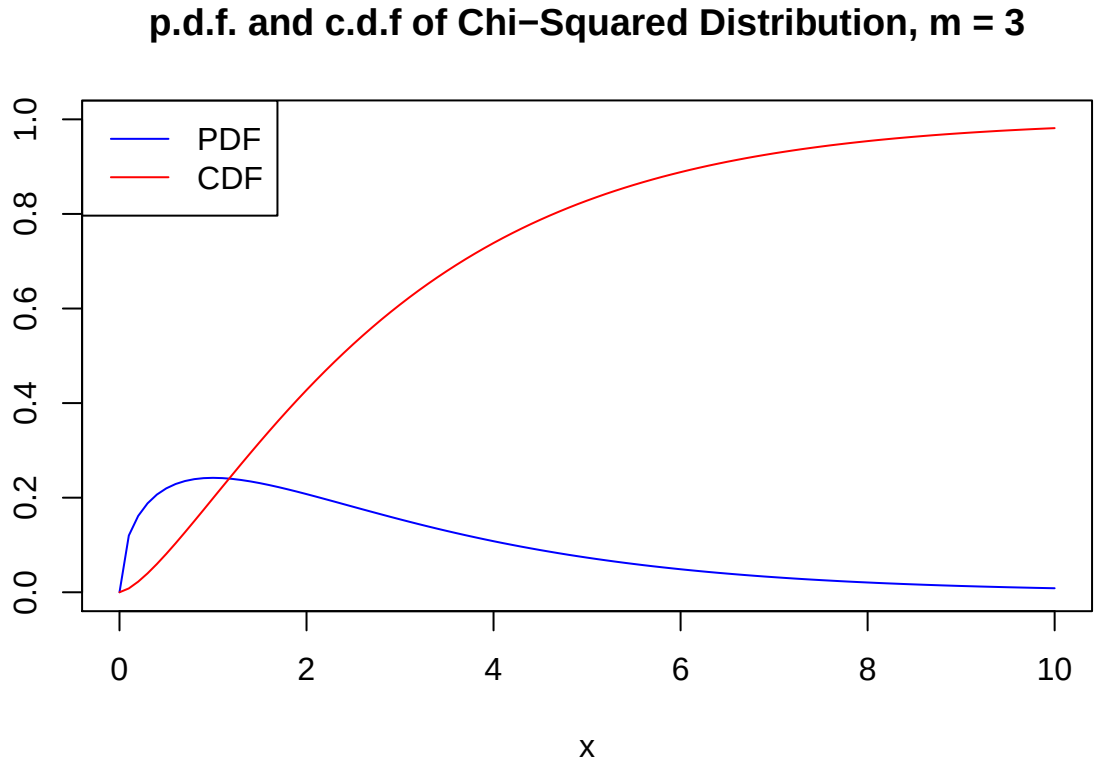
$$Z_1, Z_2, Z_3 \overset{i.i.d.}{\sim} N(0,1)$$

it holds that

$$Z_1^2 + Z_2^2 + Z_3^3 \sim \chi_3^2. \tag{2.3}$$

Using the code below, we can display the PDF and the CDF of a χ_3^2 random variable in a single plot. This is achieved by setting the argument `add = TRUE` in the second call of `curve()`. Further we adjust limits of both axes using `xlim` and `ylim` and choose different colors to make both functions better distinguishable. The plot is completed by adding a legend with help of the function `legend()`.

```
# plot the PDF
curve(dchisq(x, df=3),
      xlim = c(0, 10),
      ylim = c(0, 1),
      col = "blue",
      ylab = "",
      main = "p.d.f. and c.d.f of Chi-Squared Distribution, m = 3"
      )

# add the CDF to the plot
curve(pchisq(x, df = 3),
      xlim=c(0, 10),
      add = TRUE,
      col = "red"
      )

# add a legend to the plot
legend("topleft",
       c("PDF", "CDF"),
       col = c("blue", "red"),
       lty = c(1, 1)
       )
```



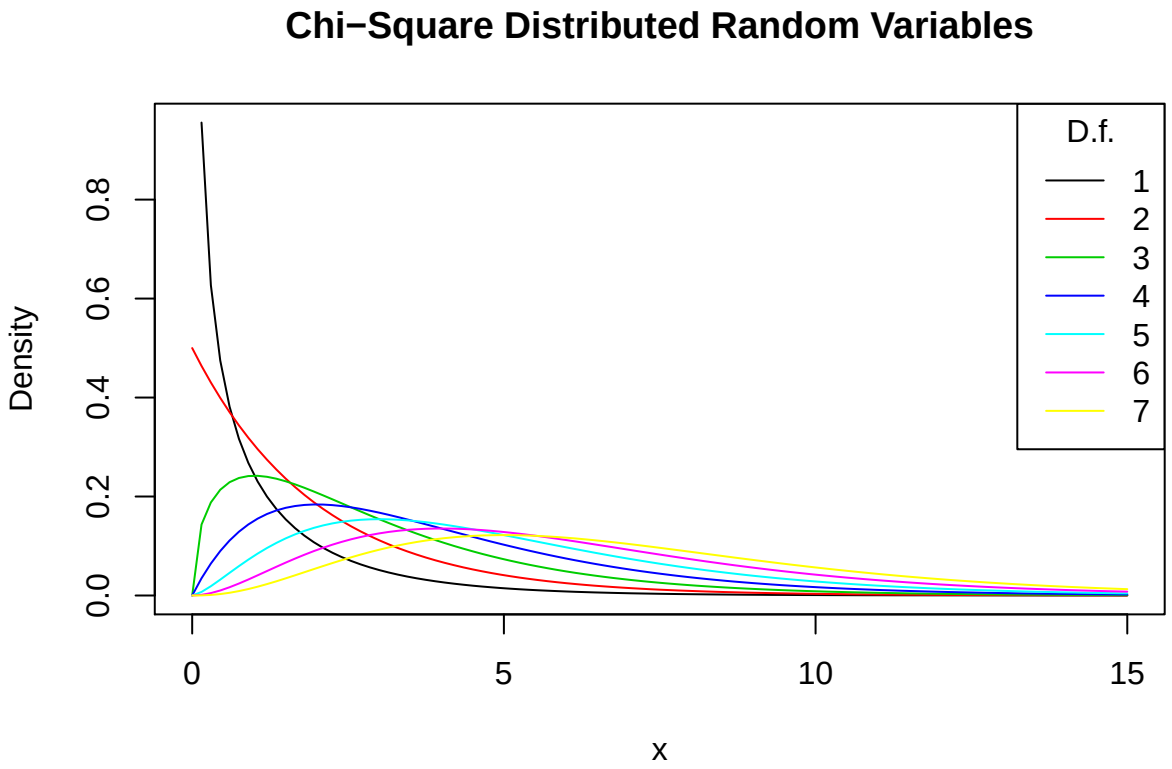
Since the outcomes of a χ^2_M distributed random variable are always positive, the domain of the related PDF and CDF is $\mathbb{R}_{\geq 0}$.

As expectation and variance depend (solely!) on the degrees of freedom, the distribution's shape changes drastically if we vary the number of squared standard normals that are summed up. This relation is often depicted by overlaying densities for different M, see e.g. the Wikipedia Article.

Of course, one can easily reproduce such a plot using `R`. Again we start by plotting the density of the χ^2_1 distribution on the interval $[0, 15]$ with `curve()`. In the next step, we loop over degrees of freedom $m = 2, ..., 7$ and add a density curve for each m to the plot. We also adjust the line color for each iteration of the loop by setting `col = m`. At last, we add a legend that displays degrees of freedom and the associated colors.

```
# plot the density for m=1
curve(dchisq(x, df = 1),
      xlim=c(0, 15),
      xlab = "x",
      ylab = "Density",
      main = "Chi-Square Distributed Random Variables"
      )

# add densities for m=2,...,7 to the plot using a for loop
for (m in 2:7) {
  curve(dchisq(x, df = m),
        xlim = c(0, 15),
        add = T,
        col = m
        )
}

# add a legend
legend("topright",
```

```
       as.character(1:7),
       col = 1:7 ,
       lty = 1,
       title = "D.f."
       )
```



It is evident that increasing the degrees of freedom shifts the distribution to the right (the mode becomes larger) and increases the dispersion (the distribution's variance grows).

The Student t Distribution

Let Z be a standard normal variate, W a random variable that follows a χ^2_M distribution with M degrees of freedom and further assume that Z and W are independently distributed. Then it holds that

$$\frac{Z}{\sqrt{W/M}} =: X \sim t_M$$

and we say that X follows a *Student t distribution* (or simply t distribution) with M degrees of freedom.

As for the χ^2_M distribution, the shape of a t_M distribution depends on M. t distributions are symmetric, bell-shaped and look very similar to a normal distribution, especially when M is large. This is not a coincidence: for a sufficient large M, the t_M distribution can be approximated by the standard normal distribution. This approximation works reasonably well for $M \geq 30$. As we will show later by means of a small simulation study, the t_∞ distribution *is* the standard normal distribution.

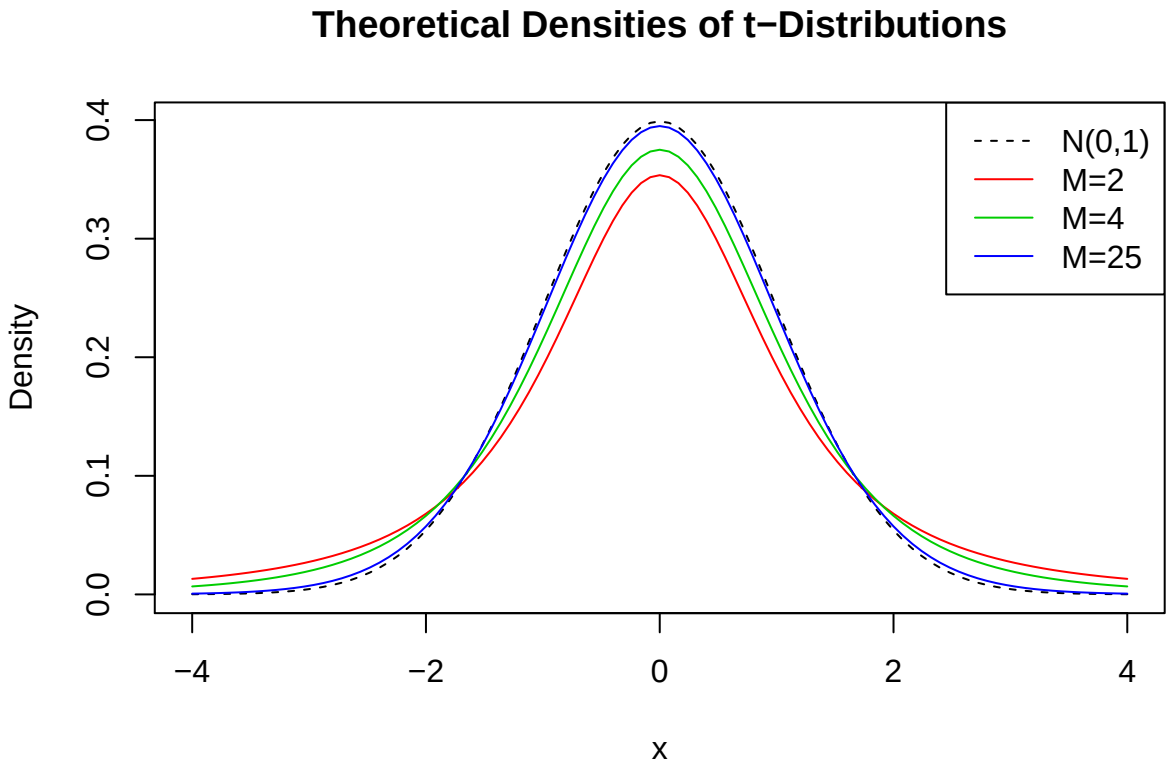
A t_M distributed random variable has an expectation if $M > 1$ and it has a variance if $n > 2$.

$$E(X) = 0 \ , \ M > 1 \tag{2.19}$$

$$\text{Var}(X) = \frac{M}{M-2} \ , \ M > 2 \tag{2.20}$$

Let us graph some t distributions with different M and compare them with the standard normal distribution.

```
# plot the standard normal density
curve(dnorm(x),
      xlim = c(-4,4),
      xlab = "x",
      lty = 2,
      ylab = "Density",
      main = "Theoretical Densities of t-Distributions"
      )

# plot the t density for m=2
curve(dt(x, df = 2),
      xlim = c(-4, 4),
      col = 2,
      add = T
      )

# plot the t density for m=4
curve(dt(x, df = 4),
      xlim = c(-4, 4),
      col = 3,
      add = T
      )

# plot the t density for m=25
curve(dt(x, df = 25),
      xlim = c(-4, 4),
      col = 4,
      add = T
      )

# add a legend
legend("topright",
       c("N(0,1)","M=2","M=4","M=25"),
       col = 1:4,
       lty = c(2, 1, 1, 1)
       )
```



The plot indicates what has been claimed in the previous paragraph: as the degrees of freedom increase, the shape of the t distribution comes closer to that of a standard normal bell. Already for $M = 25$ we find little difference to the dashed line which is the standard normal density curve. If M is small, we find the distribution to have slightly heavier tails than a standard normal, i.e. it has a "fatter" bell shape.

The F Distribution

Another ratio of random variables important to econometricians is the ratio of two independently χ^2 distributed random variables that are divided by their degrees of freedom M and n. The quantity

$$\frac{W/M}{V/n} \sim F_{M,n} \quad \text{with} \quad W \sim \chi^2_M \quad , \quad V \sim \chi^2_n$$

follows an F distribution with numerator degrees of freedom M and denominator degrees of freedom n, denoted $F_{M,n}$. The distribution was first derived by George Snedecor but was named in honor of Sir Ronald Fisher.

By definition, the domain of both PDF and CDF of an $F_{M,n}$ distributed random variable is $\mathbb{R}_{\geq 0}$.

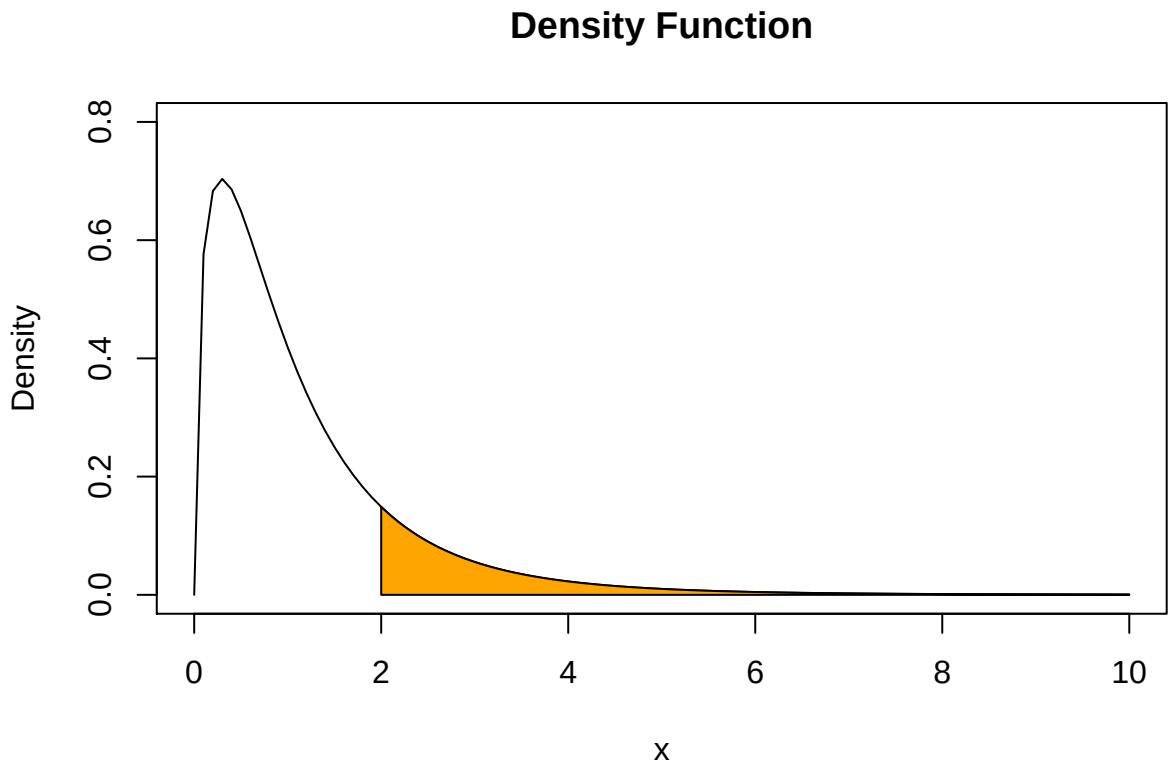
Say we have an F distributed random variable Y with numerator degrees of freedom 3 and denominator degrees of freedom 14 and are interested in $P(Y \geq 2)$. This can be computed with help of the function `pf()`. By setting the argument `lower.tail` to `TRUE` we ensure that `R` computes $1 - P(Y \leq 2)$, i.e. the probability mass in the tail right of 2.

```
pf(2, 3, 13, lower.tail = F)
```

```
## [1] 0.1638271
```

We can visualize this probability by drawing a line plot of the related density function and adding a color shading with `polygon()`.

```
# define coordinate vectors for vertices of the polygon
x <- c(2, seq(2, 10, 0.01), 10)
y <- c(0, df(seq(2, 10, 0.01), 3, 14), 0)

# draw density of F_{3, 14}
curve(df(x ,3 ,14),
      ylim = c(0, 0.8),
      xlim = c(0, 10),
      ylab = "Density",
      main = "Density Function"
      )

# draw the polygon
polygon(x, y, col="orange")
```



The F distribution is related to many other distributions. An important special case encountered in econometrics arises if the denominator degrees of freedom are large such that the $F_{M,n}$ distribution can be approximated by the $F_{M,\infty}$ distribution which turns out to be simply the distribution of a χ^2_M random variable divided by its degrees of freedom M,

$$W/M \sim F_{M,\infty} \quad , \quad W \sim \chi^2_M.$$

You may use this widget to play around with the functions presented above

2.3 Random Sampling and the Distribution of Sample Averages

To clarify the basic idea of random sampling, let us jump back to the die rolling example:

Suppose we are rolling the die n times. This means we are interested in the outcomes of n random processes $Y_i,\ i = 1, ..., n$ which are characterized by the same distribution. Since these outcomes are selected randomly, they are *random variables* themselves and their realizations will differ each time we draw a sample, i.e. each time we roll the die n times. Furthermore, each observation is randomly drawn from the same population, that is the numbers from 1 to 6, and their individual distribution is the same. Hence we say that $Y_1, \dots, Y_n$ are identically distributed. Moreover, we know that the value of any of the Y_i does not provide any information on the remainder of the sample. In our example, rolling a six as the first observation in our sample does not alter the distributions of $Y_2, \dots, Y_n$: all numbers are equally likely to occur. This means that all Y_i are also independently distributed. Thus, we say that $Y_1, \dots, Y_n$ are independently and identically distributed (*i.i.d*). The die example uses the most simple sampling scheme. That is why it is called *simple random sampling*. This concept is condensed in Key Concept 2.5.

Key Concept 2.5

Simple Random Sampling and i.i.d. Random Variables

In simple random sampling, n objects are drawn at random from a population. Each object is equally likely to end up in the sample. We denote the value of the random variable Y for the i^{th} randomly drawn object as Y_i. Since all objects are equally likely to be drawn and the distribution of Y_i is the same for all i, the $Y_i, \dots, Y_n$ are independently and identically distributed (i.i.d.). This means the distribution of Y_i is the same for all $i = 1, \dots, n$ and Y_1 is distributed independently of $Y_2, \dots, Y_n$ and Y_2 is distributed independently of $Y_1, Y_3, \dots, Y_n$ and so forth.

What happens if we consider functions of the sample data? Consider the example of rolling a die two times in a row once again. A sample now consists of two independent random draws from the set $\{1, 2, 3, 4, 5, 6\}$. In view of the aforementioned, it is apparent that any function of these two random variables is also random, e.g. their sum. Convince yourself by executing the code below several times.

```
sum(sample(1:6, 2, replace = T))
```

```
## [1] 6
```

Clearly this sum, let us call it S, is a random variable as it depends on randomly drawn summands. For this example, we can completely enumerate all outcomes and hence write down the theoretical probability distribution of our function of the sample data, S:

We face $6^2 = 36$ possible pairs. Those pairs are

$$(1,1)(1,2)(1,3)(1,4)(1,5)(1,6) \tag{2.21}$$
$$(2,1)(2,2)(2,3)(2,4)(2,5)(2,6) \tag{2.22}$$
$$(3,1)(3,2)(3,3)(3,4)(3,5)(3,6) \tag{2.23}$$
$$(4,1)(4,2)(4,3)(4,4)(4,5)(4,6) \tag{2.24}$$
$$(5,1)(5,2)(5,3)(5,4)(5,5)(5,6) \tag{2.25}$$
$$(6,1)(6,2)(6,3)(6,4)(6,5)(6,6) \tag{2.26}$$

Thus, possible outcomes for S are

$$\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.$$

Enumeration of outcomes yields

$$P(S) = \begin{cases} 1/36, & S = 2 \\ 2/36, & S = 3 \\ 3/36, & S = 4 \\ 4/36, & S = 5 \\ 5/36, & S = 6 \\ 6/36, & S = 7 \\ 5/36, & S = 8 \\ 4/36, & S = 9 \\ 3/36, & S = 10 \\ 2/36, & S = 11 \\ 1/36, & S = 12 \end{cases} \tag{2.27}$$

We can also compute $E(S)$ and $\text{Var}(S)$ as stated in Key Concept 2.1 and Key Concept 2.2.

```
# Vector of outcomes
S <- 2:12

# Vector of probabilities
PS <- c(1:6, 5:1)/36

# Expectation of S
ES <- S %*% PS
ES
```

```
##      [,1]
## [1,]    7
```

```
# Variance of S
VarS <- (S - c(ES))^2 %*% PS
VarS
```

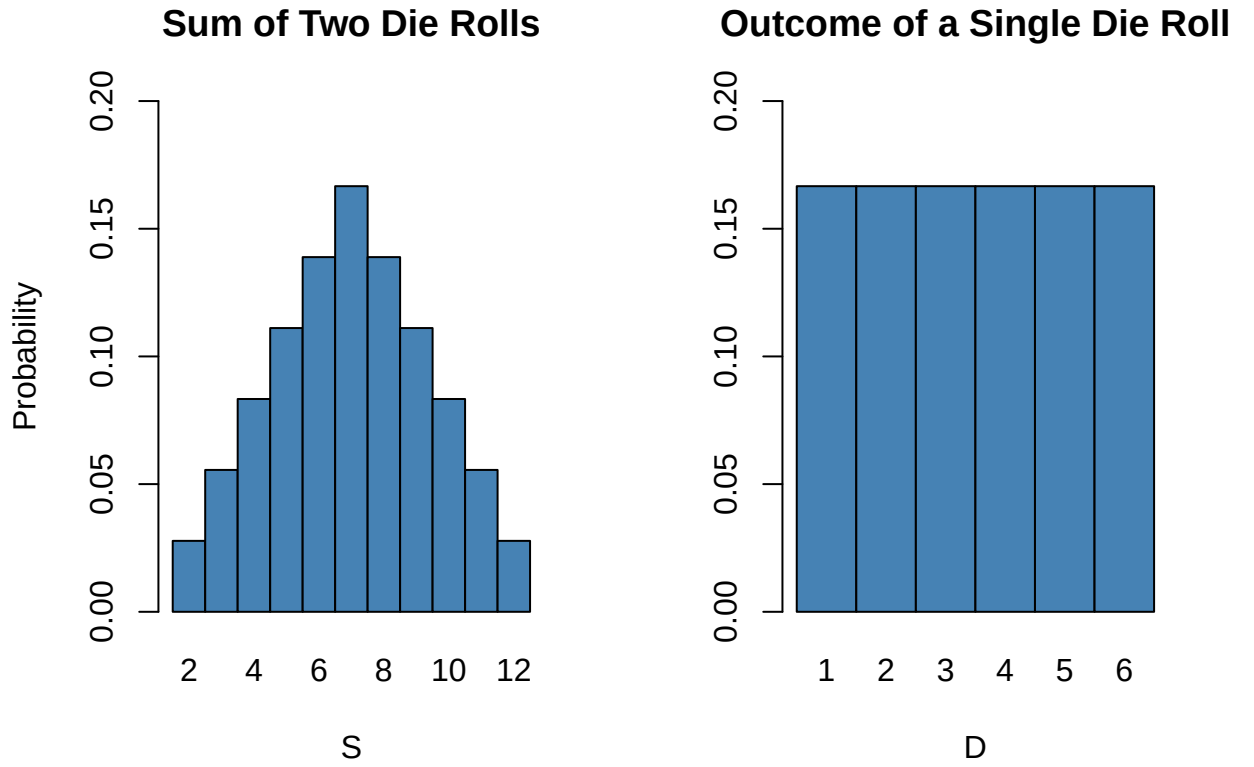
```
##          [,1]
## [1,] 5.833333
```

(The %*% operator is used to compute the scalar product of two vectors.)

So the distribution of S is known. It is also evident that its distribution differs considerably from the marginal distribution, i.e. the distribution of a single die roll's outcome, D . Let us visualize this using bar plots.

```
# divide the plotting area in one row with two columns
par(mfrow = c(1, 2))

# plot the distribution of S
names(PS) <- 2:12

barplot(PS, ylim = c(0, 0.2),
        xlab = "S",
        ylab = "Probability",
        col = "steelblue",
        space = 0,
        main = "Sum of Two Die Rolls"
        )

# plot the distribution of D
```

```
probability <- rep(1/6, 6)
names(probability) <- 1:6

barplot(probability,
        ylim = c(0, 0.2),
        xlab = "D",
        col = "steelblue",
        space = 0,
        main = "Outcome of a Single Die Roll"
        )
```



Many econometric procedures deal with averages of sampled data. It is almost always assumed that observations are drawn randomly from a larger, unknown population. As demonstrated for the sample function S, computing an average of a random sample also has the effect to make the average a random variable itself. This random variable in turn has a probability distribution which is called the sampling distribution. Knowledge about the sampling distribution of the average is therefore crucial for understanding the performance of econometric procedures.

The *sample average* of a sample of n observations $Y_1, \ldots, Y_n$ is

$$\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i = \frac{1}{n}(Y_1 + Y_2 + \cdots + Y_n).$$

$\overline{Y}$ is also called the sample mean.

Mean and Variance of the Sample Mean

Denote μ_Y and σ_Y^2 the mean and the variance of the Y_i and suppose that all observations $Y_1, \ldots, Y_n$ are i.i.d. such that in particular mean and variance are the same for all $i = 1, \ldots, n$. Then we have that

$$E(\overline{Y}) = E\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \frac{1}{n}E\left(\sum_{i=1}^{n} Y_i\right) = \frac{1}{n}\sum_{i=1}^{n} E\left(Y_i\right) = \frac{1}{n}\cdot n \cdot \mu_Y = \mu_Y$$

and

$$\begin{aligned}
\text{Var}(\overline{Y}) =& \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) && (2.28)\\
=& \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}(Y_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\text{cov}(Y_i, Y_j) && (2.29)\\
=& \frac{\sigma_Y^2}{n} && (2.30)\\
=& \sigma_{\overline{Y}}^2. && (2.31)
\end{aligned}$$

Note that the second summand vanishes since $\text{cov}(Y_i, Y_j) = 0$ for $i \neq j$ due to independence of the observations.

Consequently, the standard deviation of the sample mean is given by

$$\sigma_{\overline{Y}} = \frac{\sigma_Y}{\sqrt{n}}.$$

It is worthwhile to mention that these results hold irrespective of the underlying distribution of the Y_i.

The Sampling Distribution of $\overline{Y}$ when Y Is Normally Distributed

If the $Y_1, \ldots, Y_n$ are i.i.d. draws from a normal distribution with mean μ_Y and variance σ_Y^2, the following holds for their sample average $\overline{Y}$:

$$\overline{Y} \sim N(\mu_y, \sigma_Y^2/n) \tag{2.4}$$

For example, if a sample Y_i with $i = 1, \ldots, 10$ is drawn from a standard normal distribution with mean $\mu_Y = 0$ and variance $\sigma_Y^2 = 1$ we have

$$\overline{Y} \sim N(0, 0.1).$$

We can use `R`'s random number generation facilities to verify this result. The basic idea is to simulate outcomes of the true distribution of $\overline{Y}$ by repeatedly drawing random samples of 10 observation from the $N(0,1)$ distribution and computing their respective averages. If we do this for a large number of repetitions, the simulated data set of averages should quite accurately reflect the theoretical distribution of $\overline{Y}$ if the theoretical result holds.

The approach sketched above is an example of what is commonly known as *Monte Carlo Simulation* or *Monte Carlo Experiment*. To perform this simulation in `R`, we proceed as follows:

1. Choose a sample size `n` and the number of samples to be drawn `reps`.
2. Use the function `replicate()` in conjunction with `rnorm()` to draw `n` observations from the standard normal distribution `rep` times. **Note**: the outcome of `replicate()` is a matrix with dimensions `n` $\times$ `rep`. It contains the drawn samples as *columns*.
3. Compute sample means using `colMeans()`. This function computes the mean of each column i.e. of each sample and returns a vector.

```
# Set sample size and number of samples
n <- 10
reps <- 10000

# Perform random sampling
samples <- replicate(reps, rnorm(n)) # 10 x 10000 sample matrix

# Compute sample means
sample.avgs <- colMeans(samples)
```

After performing these steps we end up with a vector of sample averages. You can check the vector property of `sample.avgs`:

```
# Check that sample.avgs is a vector
is.vector(sample.avgs)
```

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

```
# print the first few entries to the console
head(sample.avgs)
```

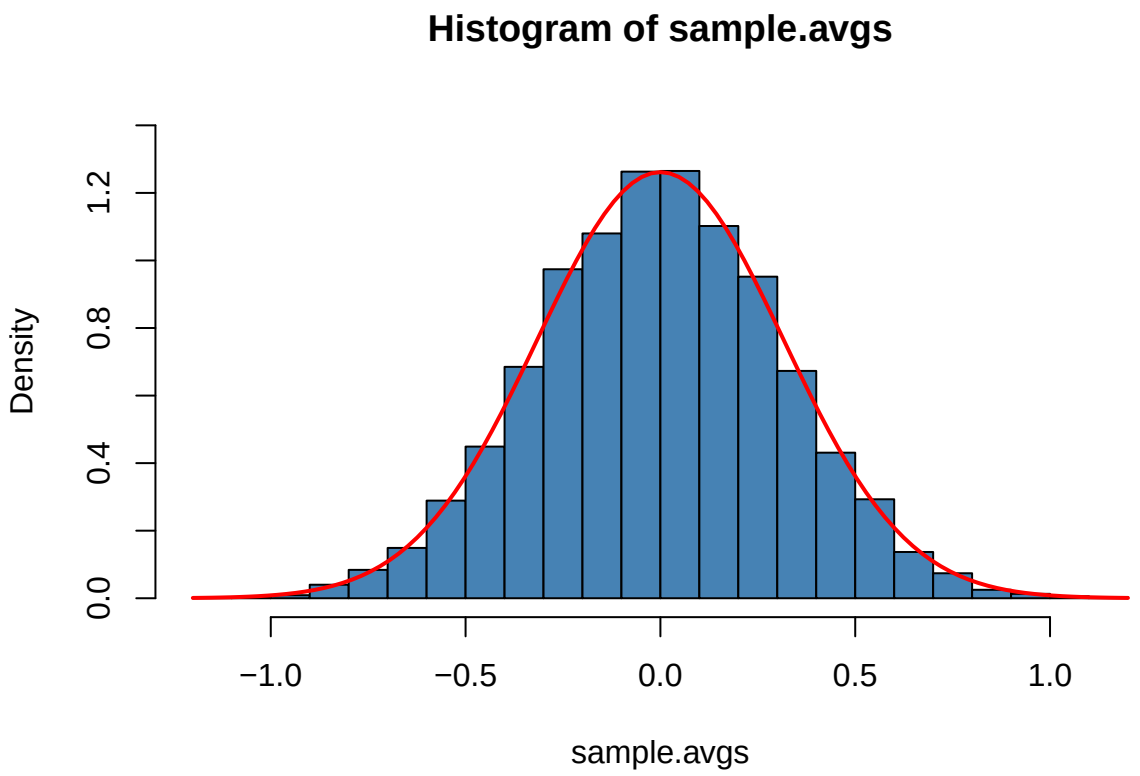
```
## [1] -0.12406767 -0.10649421 -0.01033423 -0.39905236 -0.41897968 -0.90883537
```

A straightforward approach to examine the distribution of univariate numerical data is to plot it as a histogram and compare it to some known or assumed distribution. This comparison can be done with help of a suitable statistical test or by simply eyeballing some graphical representations of these distributions. For our simulated sample averages, we will do the latter by means of the functions `hist()` and `curve()`.

By default, `hist()` will give us a frequency histogram i.e. a bar chart where observations are grouped into ranges, also called bins. The ordinate reports the number of observations falling into each of the bins. Instead, we want it to report density estimates for comparison purposes. This is achieved by setting the argument `freq = FALSE`. The number of bins is adjusted by the argument `breaks`.

Using `curve()`, we overlay the histogram with a red line which represents the theoretical density of a $N(0, 0.1)$ distributed random variable. Remember to use the argument `add = TRUE` to add the curve to the current plot. Otherwise `R` will open a new graphic device and discard the previous plot!

```
# Plot the density histogram
hist(sample.avgs,
     ylim = c(0, 1.4),
     col = "steelblue" ,
     freq = F,
     breaks = 20
     )

# overlay the theoretical distribution of sample averages on top of the histogram
curve(dnorm(x, sd = 1/sqrt(n)),
      col = "red",
      lwd = "2",
      add = T
      )
```



From inspection of the plot we can tell that the distribution of $\overline{Y}$ is indeed very close to that of a $N(0, 0.1)$ distributed random variable so that evidence obtained from the Monte Carlo Simulation supports the theoretical claim.

Let us discuss another example where using simple random sampling in a simulation setup helps to verify a well known result. As discussed before, the Chi-squared distribution with m degrees of freedom arises as the distribution of the sum of m independent squared standard normal distributed random variables.

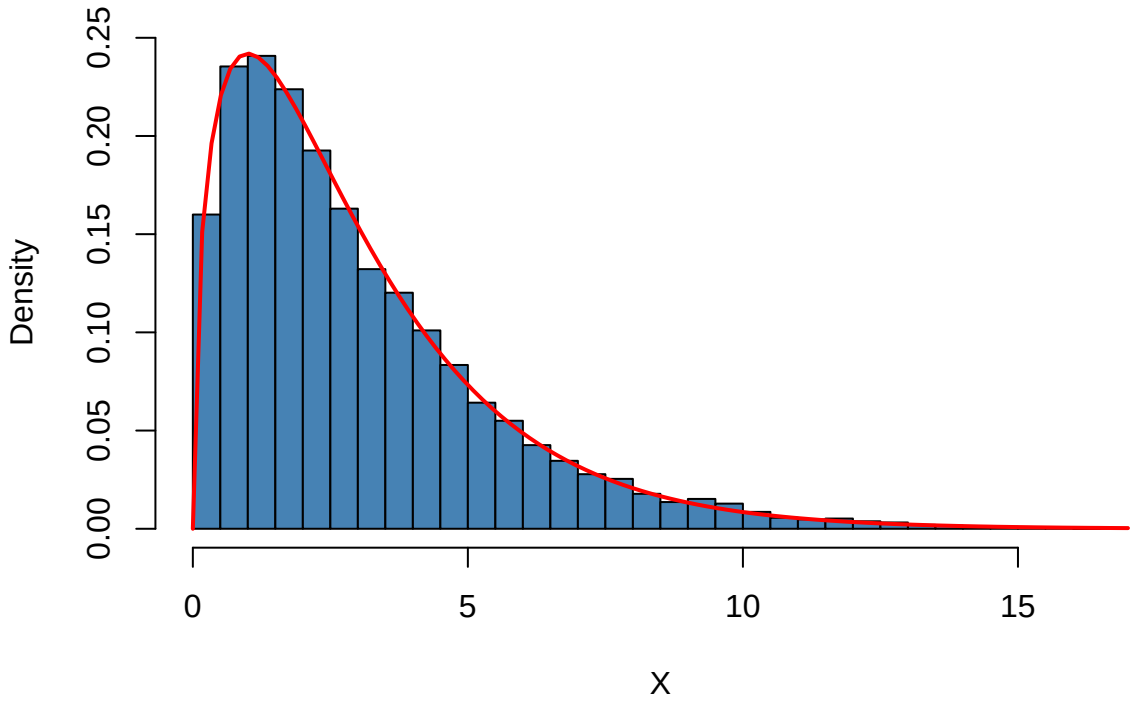
To visualize the claim stated in equation (2.3), we proceed similarly as in the example before:

1. Choose the degrees of freedom `DF` and the number of samples to be drawn `reps`.
2. Draw `reps` random samples of size `DF` from the standard normal distribution using `replicate()`.
3. For each sample, by squaring the outcomes and summing them up column wise. Store the results

Again, we produce a density estimate for the distribution underlying our simulated data using a density histogram and overlay it with a line graph of the theoretical density function of the χ^2_3 distribution.

```
# Number of repititions
reps <- 10000

# Set degrees of freedom of a chi-Square Distribution
DF <- 3

# Sample 10000 column vectors à 3 N(0,1) R.V.S
Z <- replicate(reps, rnorm(DF))

# Column sums of squares
X <- colSums(Z^2)

# Histogram of column sums of squares
hist(X,
     freq = F,
```

```
     col = "steelblue",
     breaks = 40,
     ylab = "Density",
     main = ""
     )

# Add theoretical density
curve(dchisq(x, df = DF),
      type = 'l',
      lwd = 2,
      col = "red",
      add = T
      )
```



Large Sample Approximations to Sampling Distributions

Sampling distributions as considered in the last section play an important role in the development of econometric methods. In general, there are two different approaches in characterizing sampling distributions: an "exact" approach and an "approximate" approach.

The exact approach aims to find a general formula for the sampling distribution that holds for any sample size n. We call this the *exact distribution* or *finite sample distribution*. In the previous examples of die rolling and normal variates, we have dealt with functions of random variables whose sample distributions are *excactly known* in the sense that we can write them down as analytic expressions and do calculations. However, this is not always possible. For $\overline{Y}$, result (2.4) tells us that normality of the Y_i implies normality of $\overline{Y}$ (we demonstrated this for the special case of $Y_i \overset{i.i.d.}{\sim} N(0,1)$ with $n = 10$ using a simulation study that involves simple random sampling). Unfortunately, the *exact* distribution of $\overline{Y}$ is generally unknown and often hard to derive (or even untraceable) if we drop the assumption that the Y_i have a normal distribution.

Therefore, as can be guessed from its name, the "approximate" approach aims to find an approximation to the sampling distribution whereby it is required that the sample size n is large. A distribution that is used as

a large-sample approximation to the sampling distribution is also called the *asymptotic distribution*. This is due to the fact that the asymptotic distribution *is* the sampling distribution for $n \to \infty$ i.e. the approximation becomes exact if the sample size goes to infinity. However, there are cases where the difference between the sampling distribution and the asymptotic distribution is negligible for moderate or even small samples sizes so that approximations using the asymptotic distribution are reasonably good.

In this section we will discuss two well known results that are used to approximate sampling distributions and thus constitute key tools in econometric theory: the *law of large numbers* and the *central limit theorem*. The law of large numbers states that in large samples, $\overline{Y}$ is close to μ_Y with high probability. The central limit theorem says that the sampling distribution of the standardized sample average, that is $(\overline{Y} - \mu_Y)/\sigma_{\overline{Y}}$ is asymptotically normally distributed. It is particularly interesting that both results do not depend on the distribution of Y. In other words, being unable to describe the complicated sampling distribution of $\overline{Y}$ if Y is not normal, approximations of the latter using the central limit theorem simplify the development and applicability of econometric procedures enormously. This is a key component underlying the theory of statistical inference for regression models. Both results are summarized in Key Concept 2.6 and Key Concept 2.7.

Key Concept 2.6

Convergence in Probability, Consistency and the Law of Large Numbers

The sample average $\overline{Y}$ converges in probability to μ_Y — we say that $\overline{Y}$ is *consistent* for μ_Y — if the probability that $\overline{Y}$ is in the range $(\mu_Y - \epsilon)$ to $(\mu_Y + \epsilon)$ becomes arbitrary close to 1 as n increases for any constant $\epsilon > 0$. We write this short as

$$\overline{Y} \xrightarrow{p} \mu_Y.$$

Consider the independently and identically distributed random variables $Y_i, i = 1, \dots, n$ with expectation $E(Y_i) = \mu_Y$ and variance $\text{Var}(Y_i) = \sigma_Y^2$. Under the condition that $\sigma_Y^2 < \infty$, that is large outliers are unlikely, the law of large numbers states

$$\overline{Y} \xrightarrow{p} \mu_Y.$$

The following application simulates a large number of coin tosses (you may set the number of trials using the slider) with a fair coin and computes the fraction of heads observed for each additional toss. The result is a random path that, as stated by the law of large numbers, shows a tendency to approach the value of 0.5 as n grows.

The core statement of the law of large numbers is that under quite general conditions, the probability of obtaining a sample average $\overline{Y}$ that is close to μ_Y is high if we have a large sample size.

Consider the example of repeatedly tossing a coin where Y_i is the result of the i^{th} coin toss. Y_i is a Bernoulli distributed random variable with

$$P(Y_i) = \begin{cases} p, & Y_i = 1 \\ 1 - p, & Y_i = 0 \end{cases}$$

where $p = 0.5$ as we assume a fair coin. It is straightforward to show that

$$\mu_Y = p = 0.5.$$

Say p is the probability of observing head and denote R_n the proportion of heads in the first n tosses,

$$R_n = \frac{1}{n} \sum_{i=1}^{n} Y_i. \tag{2.5}$$

According to the law of large numbers, the observed proportion of heads converges in probability to $\mu_Y = 0.5$, the probability of tossing head in a *single* coin toss,

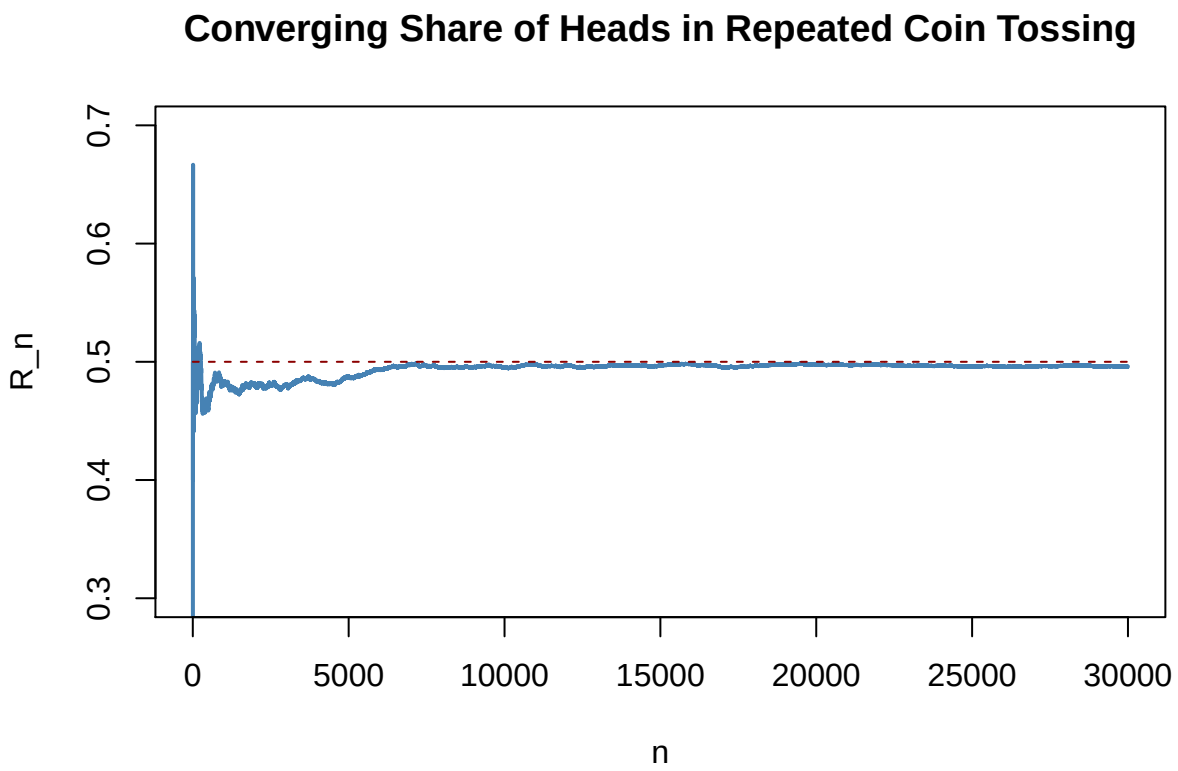
$$R_n \xrightarrow{p} \mu_Y = 0.5 \quad \text{as} \quad n \to \infty.$$

We can use `R` to compute and illustrate such paths by simulation. The procedure is as follows:

1. Sample `N` observations from the Bernoulli distribution e.g. using `sample()`.
2. Calculate the proportion of heads R_n as in (2.5). A way to achieve this is to call `cumsum()` on the vector of observations `Y` to obtain its cumulative sum and then divide by the respective number of observations.

We continue by plotting the path and also add a dashed line for the benchmark probability $R_n = p = 0.5$.

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

# Set number of coin tosses and simulate
N <- 30000
Y <- sample(0:1, N, replace =T)

# Calculate R_n for 1:N
S <- cumsum(Y)
R <- S/(1:N)

# Plot the path.
plot(R,
     ylim = c(0.3, 0.7),
     type = "l",
     col = "steelblue",
     lwd = 2,
     xlab = "n",
     ylab = "R_n",
     main = "Converging Share of Heads in Repeated Coin Tossing"
     )

# Add a dashed line for R_n = 0.5
lines(c(0, N),
      c(0.5, 0.5),
      col = "darkred",
      lty = 2,
      lwd = 1
      )
```



There are several things to be said about this plot.

- The blue graph shows the observed proportion of heads when tossing a coin n times.
- Since the Y_i are random variables, R_n is a random variate, too. The path depicted is only one of many possible realizations of R_n as it is determined by the 30000 observations sampled from the Bernoulli distribution. Thus, the code chunk above produces a different path every time you execute it (try this below!).
- If the number of coin tosses n is small, we observe the proportion of heads to be anything but close to its theoretical value, $\mu_Y = 0.5$. However, as more and more observation are included in the sample we find that the path stabilizes in neighborhood of 0.5. This is the message to take away: the average of multiple trials shows a clear tendency to converge to its expected value as the sample size increases, just as claimed by the law of large numbers.

Key Concept 2.6

The Central Limit Theorem

Suppose that $Y_1, \dots, Y_n$ are independently and identically distributed random variables with expectation $E(Y_i) = \mu_Y$ and variance $\text{Var}(Y_i) = \sigma_Y^2$ where $0 < \sigma_Y^2 < \infty$. The central Limit Theorem (CLT) states that, if the sample size n goes to infinity, the distribution of the standardized sample average

$$\frac{\overline{Y} - \mu_Y}{\sigma_{\overline{Y}}} = \frac{\overline{Y} - \mu_Y}{\sigma_Y / \sqrt{n}}$$

becomes arbitrarily well approximated by the standard normal distribution.

The application below demonstrates the CLT for the sample average of normally distributed random variables with mean 5 and variance 25^2. You may check the following properties:

- The distribution of the sample average is normal.
- As the sample size increases, the distribution of $\overline{Y}$ tightens around the true mean of 5.
- The distribution of the standardized sample average is close to the standard normal distribution for large n.

According to the central limit theorem, the distribution of the sample mean $\overline{Y}$ of the Bernoulli distributed random variables Y_i, $i = 1, ..., n$ is well approximated by the normal distribution with parameters $\mu_Y = p = 0.5$ and $\sigma^2_{\overline{Y}} = p(1-p) = 0.25$ for large n. Consequently, for the standardized sample mean we conclude that the ratio

$$\frac{\overline{Y} - 0.5}{0.5/\sqrt{n}} \tag{2.6}$$

should be well approximated by the standard normal distribution $N(0, 1)$. We employ another simulation study to demonstrate this graphically. The idea is as follows.

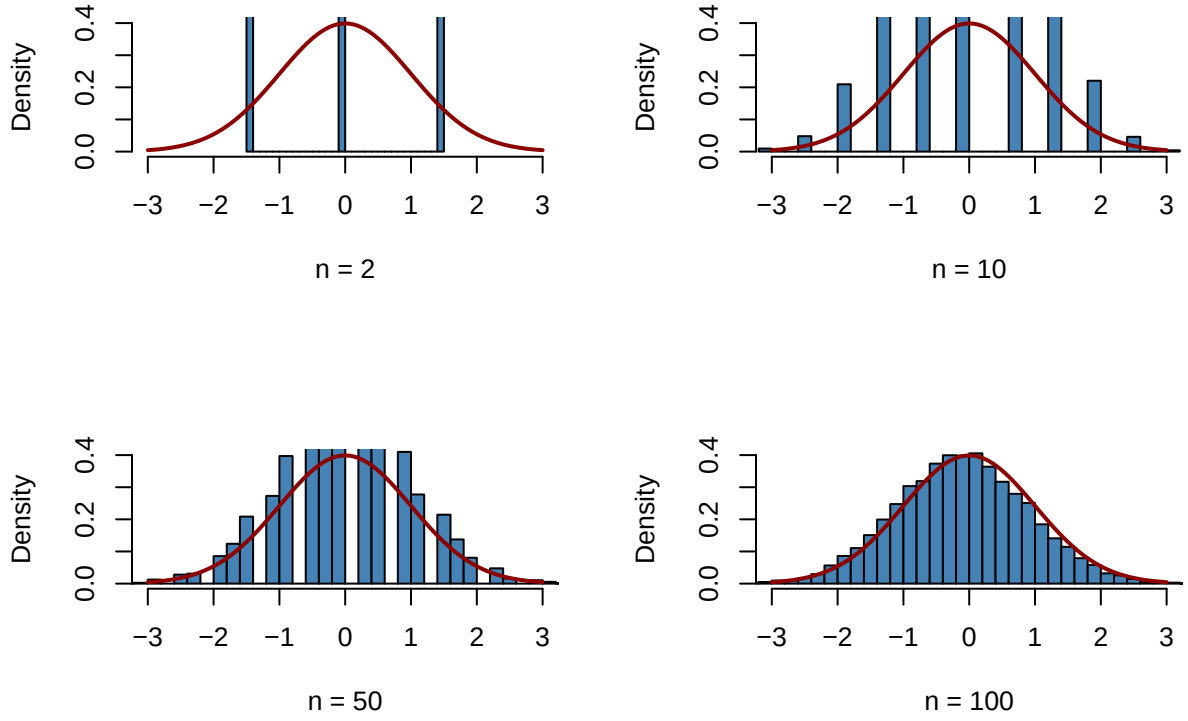
Draw a large number of random samples, 10000 say, of size n from the Bernoulli distribution and compute the sample averages. Standardize the averages as shown in (2.6). Next, visualize the distribution of the generated standardized sample averages by means of a density histogram and compare to the standard normal distribution. Repeat this for different sample sizes n to see how increasing the sample size n impacts the simulated distribution of the averages.

In R, we realized this as follows:

1. We start by defining that the next four subsequently generated figures shall be drawn in a 2×2 array such that they can be easily compared. This is done by calling `par(mfrow = c(2, 2))` before the figures are generated.
2. We define the number of repetitions `reps` as 10000 and create a vector of sample sizes named `sample.sizes`. We consider samples of sizes 2, 10, 50 and 100.
3. Next, we combine two `for()` loops to simulate the data and plot the distributions. The inner loop generates 10000 random samples, each consisting of `n` observations that are drawn from the Bernoulli distribution, and computes the standardized averages. The outer loop executes the inner loop for the different sample sizes `n` and produces a plot for each iteration.

```
# Subdivide the plot panel into a 2-by-2 array
par(mfrow = c(2, 2))

# Set number of repetitions and the sample sizes
reps <- 10000
sample.sizes <- c(2, 10, 50, 100)

# outer loop (loop over the sample sizes)
  for (n in sample.sizes) {

    samplemean <- rep(0, reps) #initialize the vector of sample menas
    stdsamplemean <- rep(0, reps) #initialize the vector of standardized sample menas

# inner loop (loop over repetitions)
    for (i in 1:reps) {
      x <- rbinom(n, 1, 0.5)
      samplemean[i] <- mean(x)
      stdsamplemean[i] <- sqrt(n)*(mean(x) - 0.5)/0.5
    }

# plot the histogram and overlay it with the N(0,1) density for every iteration
    hist(stdsamplemean,
         col = "steelblue",
         breaks = 40,
         freq = FALSE,
         xlim = c(-3, 3),
```

```
        ylim = c(0, 0.4),
        xlab = paste("n =", n),
        main = ""
        )

  curve(dnorm(x),
        lwd = 2,
        col = "darkred",
        add = TRUE
        )
}
```



We see that the simulated sampling distribution of the standardized average tends to deviate strongly from the standard normal distribution if the sample size is small, e.g. for $n = 5$ and $n = 10$. However as n grows, the histograms are approaching the bell shape of a standard normal and we can be confident that the approximation works quite well as seen for $n = 100$.

2.4 Exercises

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

Chapter 3

A Review of Statistics using R

This section reviews important statistical concepts:

- Estimation
- Hypothesis Testing
- Confidence Intervals

Since these types of statistical methods are heavily used in econometrics, we will discuss them in the context of inference about an unknown population mean and discuss several applications in `R`.

The `R` applications presented in this chapter rely on the following packages which are not part of base `R`:

- `readxl` allows to import data from Excel to `R`.
- `dplyr` provides a flexible grammar for data manipulation.
- `MASS` is a collection of functions for applied statistics.

Make sure these are installed before you go ahead and replicate the examples.

3.1 Estimation of the Population Mean

Key Concept 3.1

Estimators and Estimates

Estimators are functions of sample data that are drawn randomly from an unknown population. *Estimates* are numeric values computed by estimators based on the sample data. Estimators are random variables because they are functions of *random* data. Estimates are nonrandom numbers.

Think of some economic variable, for example hourly earnings of college graduates, denoted by Y. Suppose we are interested in μ_Y the mean of Y. In order to exactly calculate μ_Y we would have to interview every graduated member of the working population in the economy. We simply cannot do this for time and cost reasons. However, we could draw a random sample of n i.i.d. observations $Y_1, \dots, Y_n$ and estimate μ_Y using one of the simplest estimators in the sense of Key Concept 3.1 one can think of, that is

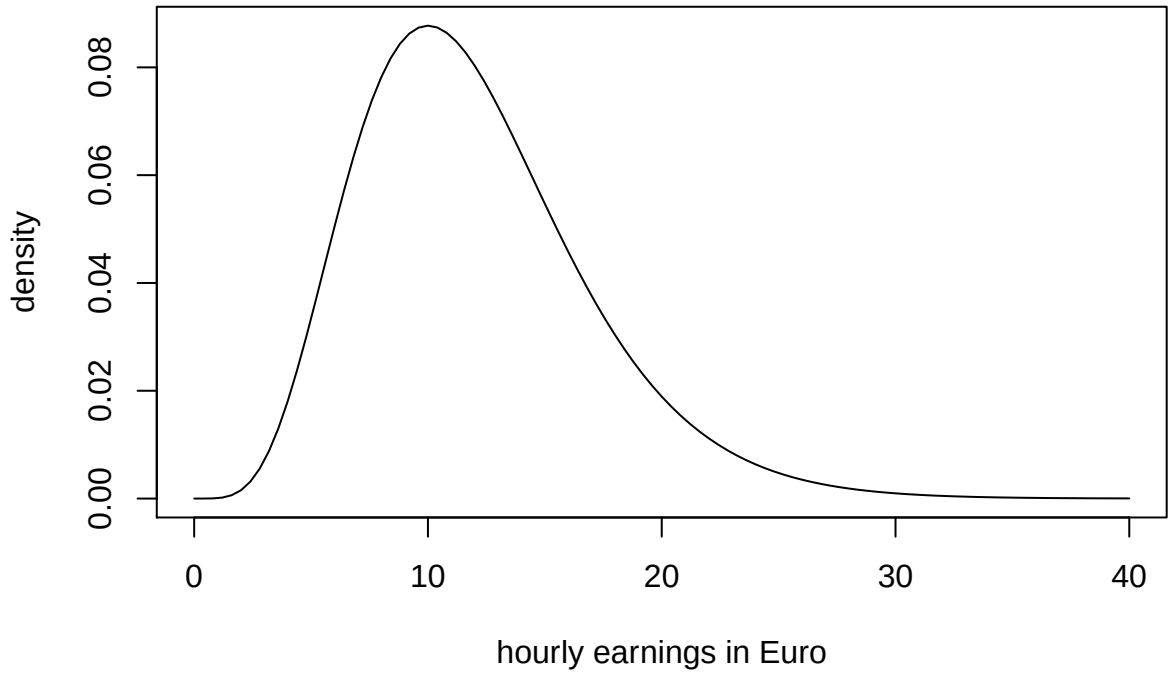
$$\overline{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i,$$

the sample mean of Y. Then again, we could use an even simpler estimator for μ_Y: the very first observation in the sample, Y_1. Is Y_1 a good estimator? For now, assume that

$$Y \sim \chi^2_{12}$$

which is not too unreasonable as hourly income is non-negative and we expect many hourly earnings to be in a range of 5 to 15. Moreover, it is common for income distributions to be skewed to the right — a property of the χ^2_{12} distribution.

```
# plot the chi_12^2 distribution
curve(dchisq(x, df=12),
      from = 0,
      to = 40,
      ylab = "density",
      xlab = "hourly earnings in Euro"
      )
```



We now draw a sample of $n = 100$ observations and take the first observation Y_1 as an estimate for μ_Y

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

# sample from the chi_12^2 distribution, keep only the first observation
rchisq(n = 100, df = 12)[1]
```

```
## [1] 8.257893
```

The estimate 8.26 is not too far away from $\mu_Y = 12$ but it is somewhat intuitive that we could do better: the estimator Y_1 discards a lot of information and its variance is the population variance:

$$\text{Var}(Y_1) = \text{Var}(Y) = 2 \cdot 12 = 24$$

This brings us to the following question: What is a *good* estimator of an unknown parameter in the first place? This question is tackled in Key Concepts 3.2 and 3.3

Key Concept 3.2

Bias, Consistency and Efficiency

Desirable characteristics of an estimator are unbiasedness, consistency and efficiency.

Unbiasedness:
If the mean of the sampling distribution of some estimator $\hat{\mu}_Y$ for the population mean μ_Y equals μ_Y

$$E(\hat{\mu}_Y) = \mu_Y$$

we say that the estimator is unbiased for μ_Y. The *bias* of $\hat{\mu}_Y$ is 0:

$$E(\hat{\mu}_Y) - \mu_Y = 0$$

Consistency:

We want the uncertainty of the estimator μ_Y to decrease as the number of observations in the sample grows. More precisely, we want the probability that the estimate $\hat{\mu}_Y$ falls within a small interval of the true value μ_Y to get increasingly closer to 1 as n grows. We write this as

$$\hat{\mu}_Y \xrightarrow{p} \mu_Y.$$

Variance and efficiency:

We want the estimator to be efficient. Suppose we have two estimators, $\hat{\mu}_Y$ and $\widetilde{\mu}_Y$ and for some given sample size n it holds that

$$E(\hat{\mu}_Y) = E(\widetilde{\mu}_Y) = \mu_Y$$

but

$$\text{Var}(\hat{\mu}_Y) < \text{Var}(\widetilde{\mu}_Y).$$

We then would prefer to use $\hat{\mu}_Y$ as it has a lower variance than $\widetilde{\mu}_Y$, meaning that $\hat{\mu}_Y$ is more *efficient* in using the information provided by the observations in the sample.

Key Concept 3.3

Efficiency of $\overline{Y}$: The BLUE Property

Let $\hat{\mu}_Y$ be a linear and unbiased estimator of μ_Y in the fashion of

$$\hat{\mu}_Y = \frac{1}{n}\sum_{i=1}^{n} a_i Y_i$$

with nonrandom constants a_i. We see that $\hat{\mu}_Y$ is a weighted average of the Y_i and the a_i are weights. For these type of estimators, $\overline{Y}$ with $a_i = 1$ for all $i = 1, \dots, n$ is the most efficient estimator. We say that $\overline{Y}$ is the **B**est **L**inear **U**nbiased **E**stimator (BLUE).

3.2 Properties of the Sample Mean

To examine properties of the sample mean as an estimator for the corresponding population mean, consider the following Rexample.

We generate a population `pop` which consists observations Y_i , $i = 1, \dots, 10000$ that origin from a normal distribution with mean $\mu = 10$ and variance $\sigma^2 = 1$. To investigate how the estimator $\hat{\mu} = \bar{Y}$ behaves we can draw random samples from this population and calculate $\bar{Y}$ for each of them. This is easily done by making

use of the function `replicate()`. The argument `expr` is evaluated `n` times. In this case we draw samples of sizes $n = 5$ and $n = 25$, compute the sample means and repeat this exactly $n = 25000$ times.

For comparison purposes we store results for the estimator Y_1, the first observation in a sample for a sample of size 5, separately.

```
# generate a fictive population
pop <- rnorm(10000, 10, 1)

# sample form pop and estimate the mean
est1 <- replicate(expr = mean(sample(x = pop, size = 5)), n = 25000)

est2 <- replicate(expr = mean(sample(x = pop, size = 25)), n = 25000)

fo <- replicate(expr = sample(x = pop, size = 5)[1], n = 25000)
```

Check that `est1` and `est2` are vectors of length 25000:

```
# check if object type is vector
is.vector(est1)
```

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

```
is.vector(est2)
```

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

```
# check length
length(est1)
```

```
## [1] 25000
```

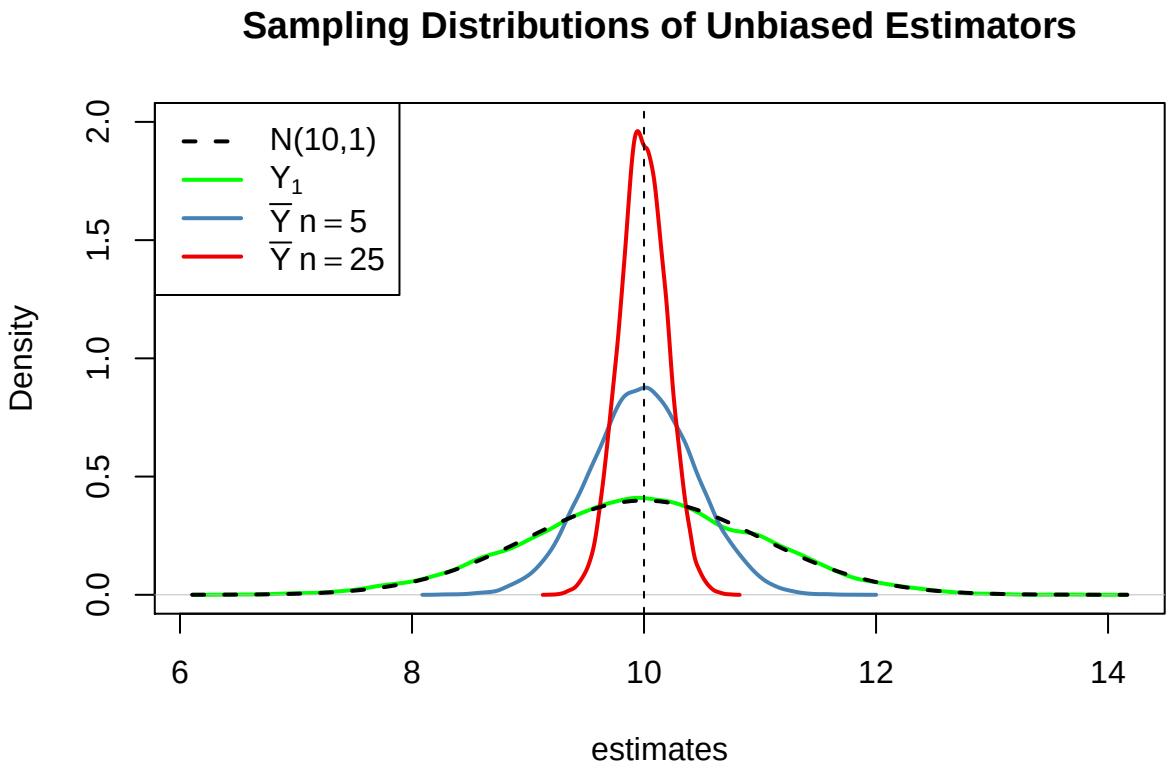
```
length(est2)
```

```
## [1] 25000
```

The code chunk below produces a plot of the sampling distributions of the estimators $\bar{Y}$ and Y_1 on the basis of the 25000 samples in each case. We also plot the density function of the $N(10, 1)$ distribution.

```
# plot density estimate Y_1
plot(density(fo),
      col = 'green',
      lwd = 2,
      ylim = c(0, 2),
      xlab = 'estimates',
      main = 'Sampling Distributions of Unbiased Estimators'
      )

# add density estimate for the distribution of the sample mean with n=5 to the plot
lines(density(est1),
     col = 'steelblue',
     lwd = 2,
     bty = 'l'
     )

# add density estimate for the distribution of the sample mean with n=25 to the plot
lines(density(est2),
      col = 'red2',
      lwd = 2
```

```
      )

# add a vertical line marking the true parameter
abline(v = 10, lty = 2)

# add N(10,1) density to the plot
curve(dnorm(x, mean = 10),
      lwd = 2,
      lty = 2,
      add = T
      )

# add a legend
legend("topleft",
       legend = c("N(10,1)",
                  expression(Y[1]),
                  expression(bar(Y) ~ n == 5),
                  expression(bar(Y) ~ n == 25)
                  ),
       lty = c(2, 1, 1, 1),
       col = c('black','green', 'steelblue', 'red2'),
       lwd = 2
       )
```



At first, notice that *all* sampling distributions (represented by the solid lines) are centered around $\mu = 10$. This is evidence for the *unbiasedness* of Y_1, $\overline{Y}_{n=5}$ and $\overline{Y}_{n=25}$. Of course, the theoretical density the $N(10, 1)$ distribution is centered at 10, too.

Next, have a look at the spread of the sampling distributions. Several things are remarkable:

- the sampling distribution of Y_1 (green curve) tracks the density of the $N(10, 1)$ distribution (black

dashed line) pretty closely In fact, the sampling distribution of Y_1 is the $N(10, 1)$ distribution. This is less surprising if You keep in mind that the Y_1 estimator does nothing but reporting an observation that is randomly selected from a population with $N(10, 1)$ distribution. Hence, $Y_1 \sim N(10, 1)$. Note that this result does not depend on the sample size n: the sampling distribution of Y_1 *is always* the population distribution, no matter how large the sample is.

- Both sampling distributions of $\overline{Y}$ show less dispersion than the sampling distribution of Y_1. This means that $\overline{Y}$ has a lower variance than Y_1. In view of Key Concepts 3.2 and 3.3, we find that $\overline{Y}$ is a more efficient estimator than Y_1. In fact, one can show that this holds for all $n > 1$.
- $\overline{Y}$ shows a behavior that is termed consistency (see Key Concept 3.2). Notice that the blue and the red density curves are much more concentrated around $\mu = 10$ then the green one. As the number of observations is increased from 1 to 5, the sampling distribution tightens around the true parameter. Increasing the sample size to 25 this effect becomes more dominant. This implies that the probability of obtaining estimates that are close to the true value increases with n.

A more precise way to express consitency of an estimator $\hat{\mu}$ for a parameter μ is

$$P(|\hat{\mu} - \mu| < \epsilon) \xrightarrow[n \to \infty]{p} 1 \quad \text{for any} \quad \epsilon > 0.$$

This expression says that the probability of observing a deviation from the true value μ that is smaller than some arbitrary $\epsilon > 0$ converges to 1 as n grows. Note that consistency does not require unbiasedness.

We encourage you to go ahead and modify the code. Try out different values for the sample size and see how the sampling distribution of $\overline{Y}$ changes!

$\overline{Y}$ is the Least Squares Estimator of μ_Y

Assume you have some observations $Y_1, \dots, Y_n$ on $Y \sim N(10, 1)$ (which is unknown) and would like to find an estimator m that predicts the observations as good as possible. Good means to choose m such that the total deviation between the predicted value and the observed values is small. Mathematically this means we want to find an m that minimizes

$$\sum_{i=1}^{n} (Y_i - m)^2. \tag{3.1}$$

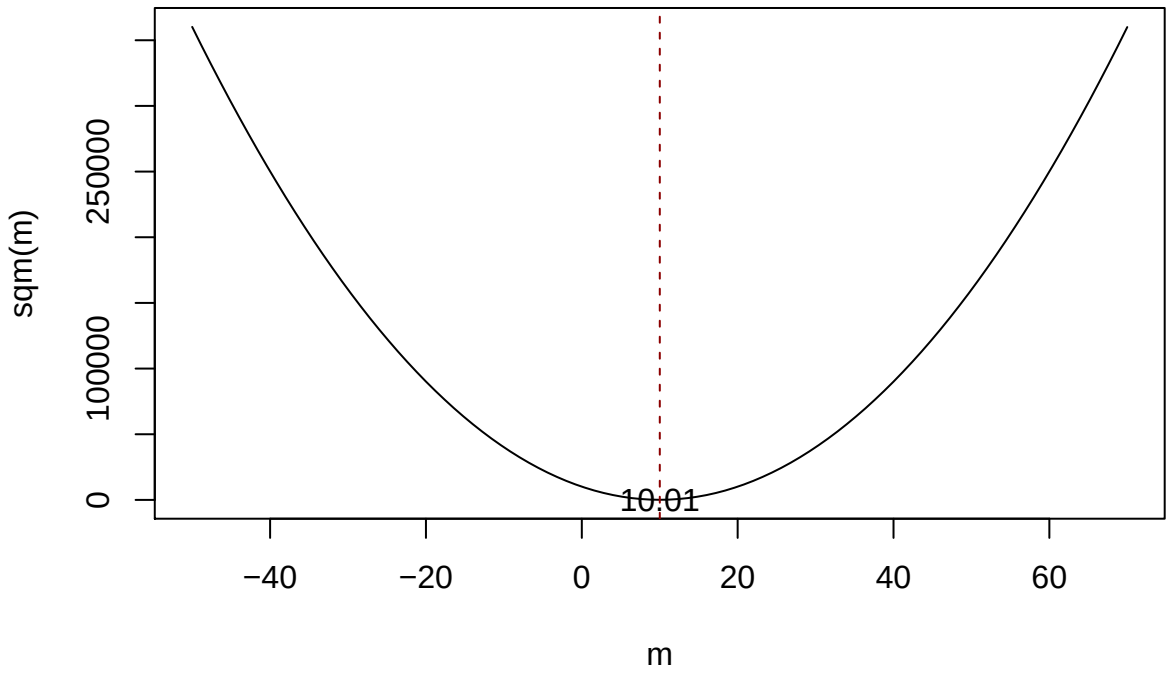
Think of $Y_i - m$ as the committed mistake when predicting Y_i by m. We could just as well minimize the sum of absolute deviations from m but minimizing the sum of squared deviations is mathematically more convenient (and may lead to a different result). That is why the estimator we are looking for is called the *least squares estimator*. As It turns out $m = \overline{Y}$, the estimator of $\mu_Y = 10$ is this wanted estimator.

We can show this by generating a random sample of fair size and plotting (3.1) as a function of m.

```
# define the function and vectorize it
sqm <- function(m) {
 sum((y-m)^2)
}
sqm <- Vectorize(sqm)

# draw random sample and compute the mean
y <- rnorm(100, 10, 1)
mean(y)
```

```
## [1] 10.00543
```

```r
# plot the objective function
curve(sqm(x),
      from = -50,
      to = 70,
      xlab = "m",
      ylab = "sqm(m)"
      )

# add vertical line at mean(y)
abline(v = mean(y),
       lty = 2,
       col = "darkred"
       )

# add annotation at mean(y)
text(x = mean(y),
     y = 0,
     labels = paste(round(mean(y), 2))
     )
```



Notice that (3.1) is a quadratic function so there is only one minimum. The plot shows that this minimum lies exactly at the sample mean of the sample data.

Some R functions can only interact with functions that take a vector as an input and evaluate the function body on every values of the vector, for example curve(). We call such functions vectorized functions and it is often a good idea to write vectorized functions yourself although this is cumbersome in some cases. Having a vectorized function in R is never a drawback since these functions work on both single values and vectors.
Let us look at the function sqm() which is nonvectorized
sqm <- function(m) {
    sum((y-m)^2) #body of the function
}
Providing e.g. c(1,2,3) as the argument m would cause an error since then the operation y-m is invalid: the vecors y and m are of incompatible dimensions. This is why we cannot use sqm() in conjunction with curve().
Here comes Vectorize() into play. It generates a vectorized version of a non-vectorized function.

Why Random Sampling is Important

So far, we assumed (sometimes implicitly) that observed data $Y_1, \dots, Y_n$ are the result of a sampling process that satisfies the assumption of i.i.d. random sampling. It is very important that this assumption is fulfilled when estimating a population mean using $\overline{Y}$. If this is not the case, estimates are biased.

Let us fall back to `pop`, the fictive population of 10000 observations and compute the population mean μ_{pop}:

```
# compute the population mean of pop
mean(pop)
```

```
## [1] 9.992604
```

Next we sample 10 observations from `pop` with `sample()` and estimate μ_{pop} using $\overline{Y}$ repeatedly. However this time we use a sampling scheme that deviates from simple random sampling: instead of ensuring that each member of the population has the same chance to end up in a sample, we assign a higher probability of being sampled to the 2500 smallest observations of the population by setting the argument `prob` to a suitable vector of probability weights:

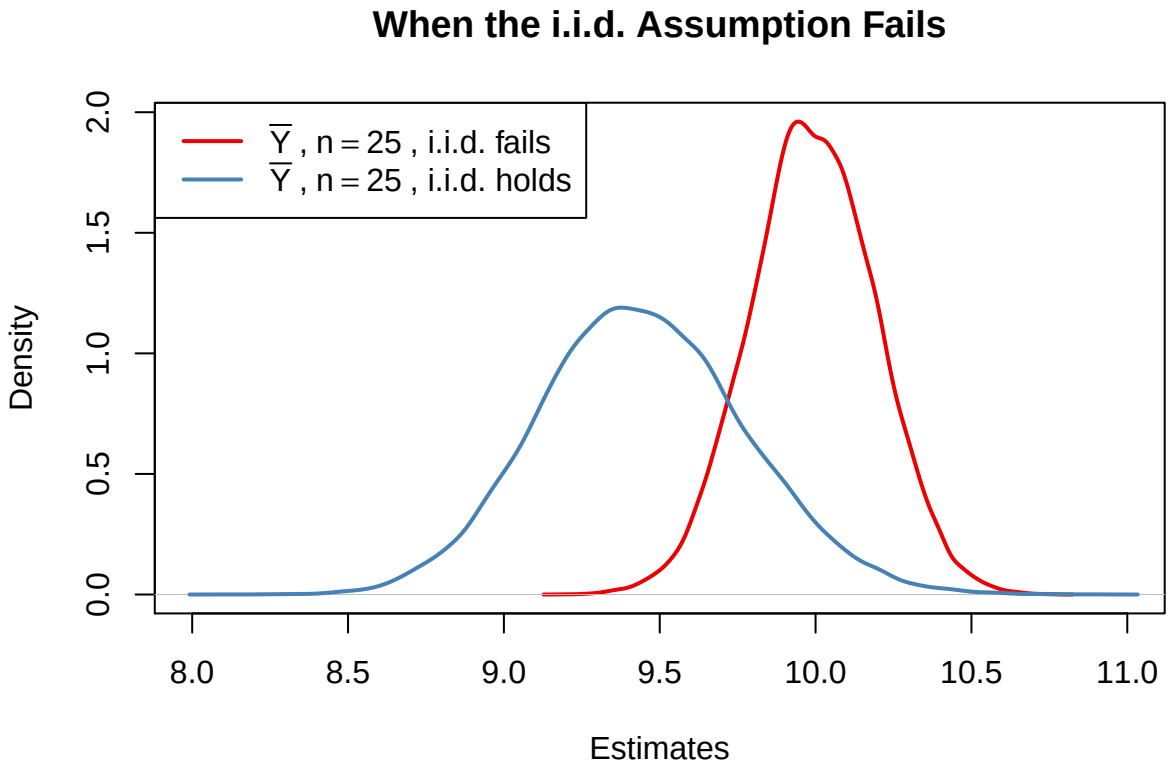
```
# simulate outcome for the sample mean when the i.i.d. assumption fails
est3 <-  replicate(n = 25000,
                   expr = mean(sample(x = sort(pop),
                                      size = 10,
                                      prob = c(rep(4, 2500), rep(1, 7500))
                                      )
                               )
                   )

# compute the sample mean of the outcomes
mean(est3)
```

```
## [1] 9.443454
```

Next we plot the sampling distribution of $\overline{Y}$ for this non-i.i.d. case and compare it to the sampling distribution when the i.i.d. assumption holds.

```
# sampling distribution of sample mean, i.i.d. holds, n=25
plot(density(est2),
      col = 'red2',
      lwd = 2,
      xlim = c(8, 11),
```

```
      xlab = 'Estimates',
      main = 'When the i.i.d. Assumption Fails'
     )

# sampling distribution of sample mean, i.i.d. fails, n=25
lines(density(est3),
      col = 'steelblue',
      lwd = 2
      )

# add a legend
legend("topleft",
       legend = c(expression(bar(Y) ~ "," ~ n == 25 ~ ", i.i.d. fails"),
                  expression(bar(Y) ~ "," ~ n == 25 ~ ", i.i.d. holds")
                  ),
       lty = c(1, 1),
       col = c('red2', 'steelblue'),
       lwd = 2
       )
```



We find that in this case failure of the i.i.d. assumption implies that, on average, we *underestimate* μ_Y using $\overline{Y}$: the corresponding distribution of $\overline{Y}$ is shifted to the left. In other words, $\overline{Y}$ is a *biased* estimator for μ_Y if the i.i.d. assumption does not hold.

3.3 Hypothesis Tests Concerning the Population Mean

In this section we briefly review concepts in hypothesis testing and discuss how to conduct hypothesis tests in R. We focus on drawing inferences about an unknown population mean.

About Hypotheses and Hypothesis Testing

In a significance test we want to exploit the information contained in a random sample as evidence in favor or against a hypothesis. Essentially, hypotheses are simple questions that can be answered by 'yes' or 'no'. When conducting a hypothesis test we always deal with two different hypotheses:

- The **null hypothesis**, denoted H_0 is the hypothesis we are interested in testing.
- The **alternative hypothesis**, denoted H_1, is the hypothesis that holds if the null hypothesis is false.

The null hypothesis that the population mean of Y equals the value $\mu_{Y,0}$ is written down as

$$H_0 : E(Y) = \mu_{Y,0}.$$

The alternative hypothesis states what holds if the null hypothesis is false. Often the alternative hypothesis chosen is the most general one,

$$H_1 : E(Y) \neq \mu_{Y,0},$$

meaning that $E(Y)$ may be anything but the value under the null hypothesis. This is called a *two-sided* alternative.

For the sake of brevity, we will only consider the case of a two-sided alternative in the subsequent sections of this chapter.

The p-Value

Assume that the null hypothesis is *true*. The p-value is the probability of drawing data and observing a corresponding test statistic that is at least as adverse to what is stated under the null hypothesis as the test statistic actually computed using the sample data.

In the context of the population mean and the sample mean, this definition can be stated mathematically in the following way:

$$p\text{-value} = P_{H_0}\left[|\overline{Y} - \mu_{Y,0}| > |\overline{Y}^{act} - \mu_{Y,0}|\right] \tag{3.2}$$

In (3.2), $\overline{Y}^{act}$ is the actually computed mean of the random sample. Visualized, the p-value is the area in the part of tails of the distribution of $\overline{Y}$ that lies beyond

$$\mu_{Y,0} \pm |\overline{Y}^{act} - \mu_{Y,0}|.$$

Consequently, in order to compute the p-value as in (3.2), knowledge about the sampling distribution of $\overline{Y}$ when the null hypothesis is true is required. However in most cases the sampling distribution of $\overline{Y}$ is unknown. Fortunately, due to the large-sample normal approximation we know that under the null hypothesis

$$\overline{Y} \sim N(\mu_{Y,0},\, \sigma^2_{\overline{Y}}) \quad , \quad \sigma^2_{\overline{Y}} = \frac{\sigma^2_Y}{n}$$

and thus

$$\frac{\overline{Y} - \mu_{Y,0}}{\sigma_Y/\sqrt{n}} \sim N(0,1).$$

So in large samples, the p-value can be computed *without* knowledge of the exact sampling distribution of $\overline{Y}$.

Calculating the p-Value When σ_Y Is Known

For now, let us assume that $\sigma_{\overline{Y}}$ is known. Then we can rewrite (3.2) as

$$p\text{-value} = P_{H_0}\left[\left|\frac{\overline{Y} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right| > \left|\frac{\overline{Y}^{act} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right|\right] \tag{3.3}$$

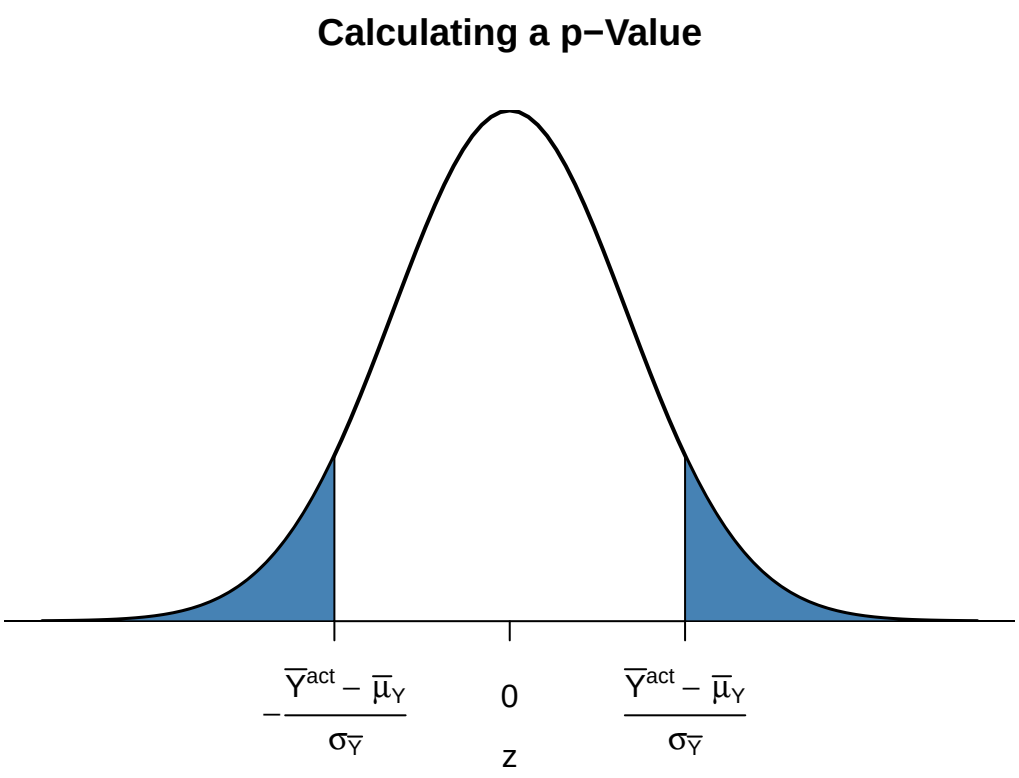
$$= 2 \cdot \Phi\left[-\left|\frac{\overline{Y}^{act} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right|\right]. \tag{3.4}$$

The p-value can be seen as the area in the tails of the $N(0,1)$ distribution that lies beyond

$$\pm\left|\frac{\overline{Y}^{act} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right| \tag{3.5}$$

We now use `R` to visualize what is stated in (3.4) and (3.5). The next code chunk replicates figure 3.1 of the book.

```
# plot the standard normal density on the interval [-4,4]
curve(dnorm(x),
      xlim = c(-4, 4),
      main = 'Calculating a p-Value',
      yaxs = 'i',
      xlab = 'z',
      ylab = '',
      lwd = 2,
      axes = 'F'
      )

# add x-axis
axis(1,
     at = c(-1.5, 0, 1.5),
     padj = 0.75,
     labels = c(expression(-frac(bar(Y)^"act"~-~bar(mu)[Y,0], sigma[bar(Y)])),
                0,
                expression(frac(bar(Y)^"act"~-~bar(mu)[Y,0], sigma[bar(Y)])))
     )

# shade p-value/2 region in left tail
polygon(x = c(-6, seq(-6, -1.5, 0.01), -1.5),
        y = c(0, dnorm(seq(-6, -1.5, 0.01)),0),
        col = 'steelblue'
        )

# shade p-value/2 region in right tail
polygon(x = c(1.5, seq(1.5, 6, 0.01), 6),
        y = c(0, dnorm(seq(1.5, 6, 0.01)), 0),
        col = 'steelblue'
        )
```



Sample Variance, Sample Standard Deviation and Standard Error

If σ_Y^2 is unknown, it must be estimated. This can be done efficiently using the sample variance

$$s_y^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2. \tag{3.6}$$

Furthermore

$$s_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2} \tag{3.7}$$

is a suitable estimator for the standard deviation of Y. In R, s_y is implemented in the function `sd()`, use `?sd`.

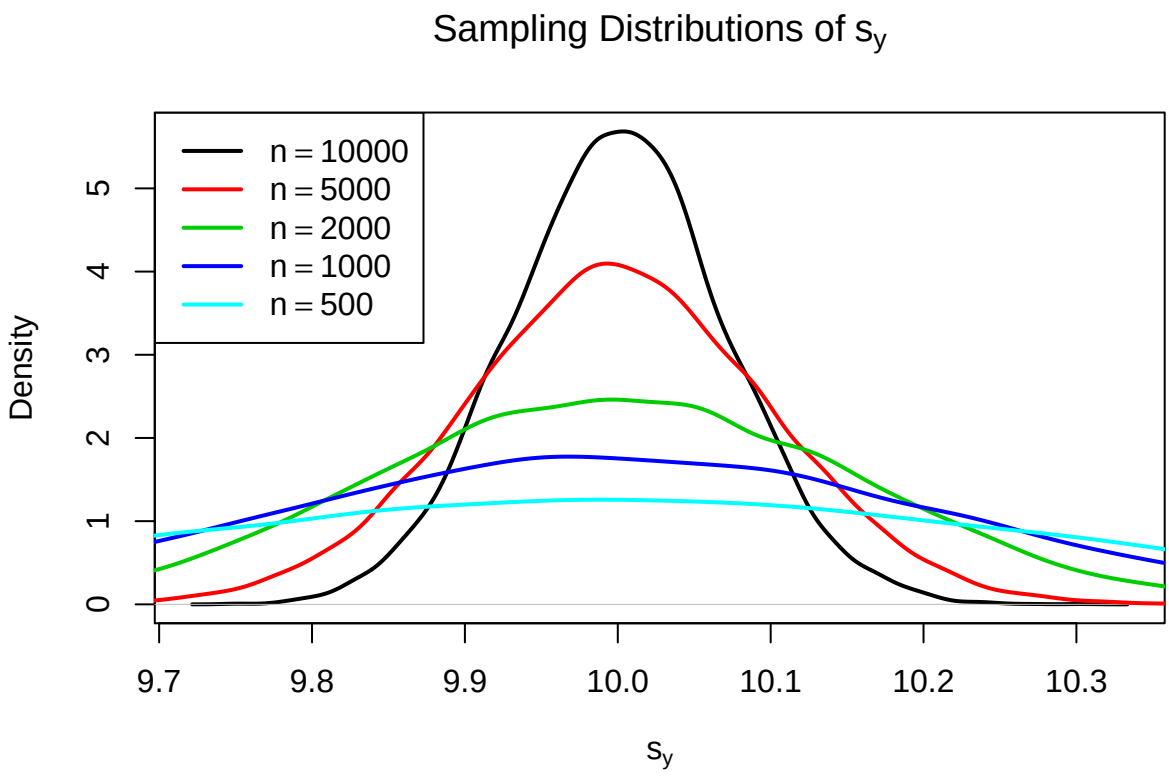
Using `R` we can get a notion that s_y is a consistent estimator for σ_Y, that is

$$s_Y \xrightarrow{p} \sigma_Y.$$

The idea here is to generate a large number of samples $Y_1, \dots, Y_n$ where, $Y \sim N(10, 10)$ say, estimate σ_Y using s_y and investigate how the distribution of s_Y changes as n gets larger.

```
# vector of sample sizes
n <- c(10000, 5000, 2000, 1000, 500)

# sample observations, estimate using sd() and plot estimated distributions
s2_y <- replicate(n = 10000, expr = sd(rnorm(n[1], 10, 10)))
```

```
plot(density(s2_y),
     main = expression('Sampling Distributions of' ~ s[y]),
     xlab = expression(s[y]),
     lwd = 2
     )

for (i in 2:length(n)) {
  s2_y <- replicate(n = 10000, expr = sd(rnorm(n[i], 10, 10)))
  lines(density(s2_y),
        col = i,
        lwd = 2
        )
}

# add a legend
legend("topleft",
       legend = c(expression(n == 10000),
                  expression(n == 5000),
                  expression(n == 2000),
                  expression(n == 1000),
                  expression(n == 500)
       ),
       col = 1:5,
       lwd = 2
)
```



The plot shows that the distribution of s_Y tightens around the true value $\sigma_Y = 10$ as n increases.

The function that estimates the standard deviation of an estimator is called the *standard error of the estimator*. Key Concept 3.4 summarizes the terminology in the context of the sample mean.

Key Concept 3.4

The Standard Error of $\overline{Y}$

Take an i.i.d. sample $Y_1, \dots, Y_n$. The mean of Y can be consistently estimated using $\overline{Y}$, the sample mean of the Y_i. Since $\overline{Y}$ is a random variable, it has a sampling distribution with variance $\frac{\sigma_Y^2}{n}$.

The standard error of $\overline{Y}$, denoted $SE(\overline{Y})$ is an estimator of the standard deviation of $\overline{Y}$:

$$SE(\overline{Y}) = \hat{\sigma}_{\overline{Y}} = \frac{s_Y}{\sqrt{n}}$$

The caret ($\widehat{}$) over σ indicates that $\hat{\sigma}_{\overline{Y}}$ is an estimator for $\sigma_{\overline{Y}}$.

As an example to underpin Key Concept 3.4, consider a sample of $n = 100$ i.i.d. observations of the Bernoulli distributed variable Y with success probability $p = 0.1$ and thus $E(Y) = p = 0.1$ and $\text{Var}(Y) = p(1-p)$. $E(Y)$ can be estimated by $\overline{Y}$ which then has variance

$$\sigma_{\overline{Y}}^2 = p(1-p)/n = 0.0009$$

and standard deviation

$$\sigma_{\overline{Y}} = \sqrt{p(1-p)/n} = 0.03.$$

In this case the standard error of $\overline{Y}$ is given by

$$SE(\overline{Y}) = \sqrt{\overline{Y}(1-\overline{Y})/n}.$$

Let us verify whether $\overline{Y}$ and $SE(\overline{Y})$ estimate the respective true values, on average.

```
# draw 10000 samples of size 100 and estimate the mean of Y and
# estimate the standard error of the sample mean

mean_estimates <- numeric(10000)
se_estimates <- numeric(10000)

for (i in 1:10000) {
  s <- sample(0:1,
              size = 100,
              prob = c(0.9, 0.1),
              replace = T
              )
  mean_estimates[i] <- mean(s)
  se_estimates[i] <- sqrt(mean(s)*(1-mean(s))/100)
}

mean(mean_estimates)
```

```
## [1] 0.099693
```

```
mean(se_estimates)
```

```
## [1] 0.02953467
```

Both estimators seem to be unbiased for the true parameters.

Calculating the p-value When the Standard Deviation is Unknown

When σ_Y is unknown, the p-value for a hypothesis test concerning μ_Y using $\overline{Y}$ can be computed by replacing $\sigma_{\overline{Y}}$ in (3.4) by the standard error $SE(\overline{Y}) = \hat{\sigma}_Y$. Then,

$$p\text{-value} = 2 \cdot \Phi \left(- \left| \frac{\overline{Y}^{act} - \mu_{Y,0}}{SE(\overline{Y})} \right| \right).$$

This is easily done in R:

```
# sample and estimate, compute standard error and make a hypothesis
samplemean_act <- mean(
  sample(0:1,
         prob = c(0.9, 0.1),
         replace = T,
         size = 100
         )
  )

SE_samplemean <- sqrt(samplemean_act * (1-samplemean_act)/100)

mean_h0 <- 0.1 #true null hypothesis

# compute the p-value
pvalue <- 2 * pnorm(-abs(samplemean_act - mean_h0)/SE_samplemean)
pvalue
```

```
## [1] 0.5382527
```

The t-statistic

In hypothesis testing, the standardized sample average

$$t = \frac{\overline{Y} - \mu_{Y,0}}{SE(\overline{Y})} \tag{3.8}$$

is called t-statistic. This t-statistic plays an important role in testing hypotheses about μ_Y. It is a prominent example of a test statistic.

Implicitly, we already have computed a t-statistic for $\overline{Y}$ in the previous code chunk.

```
# compute a t-statistic for the sample mean
tstatistic <- (samplemean_act - mean_h0) / SE_samplemean
tstatistic
```

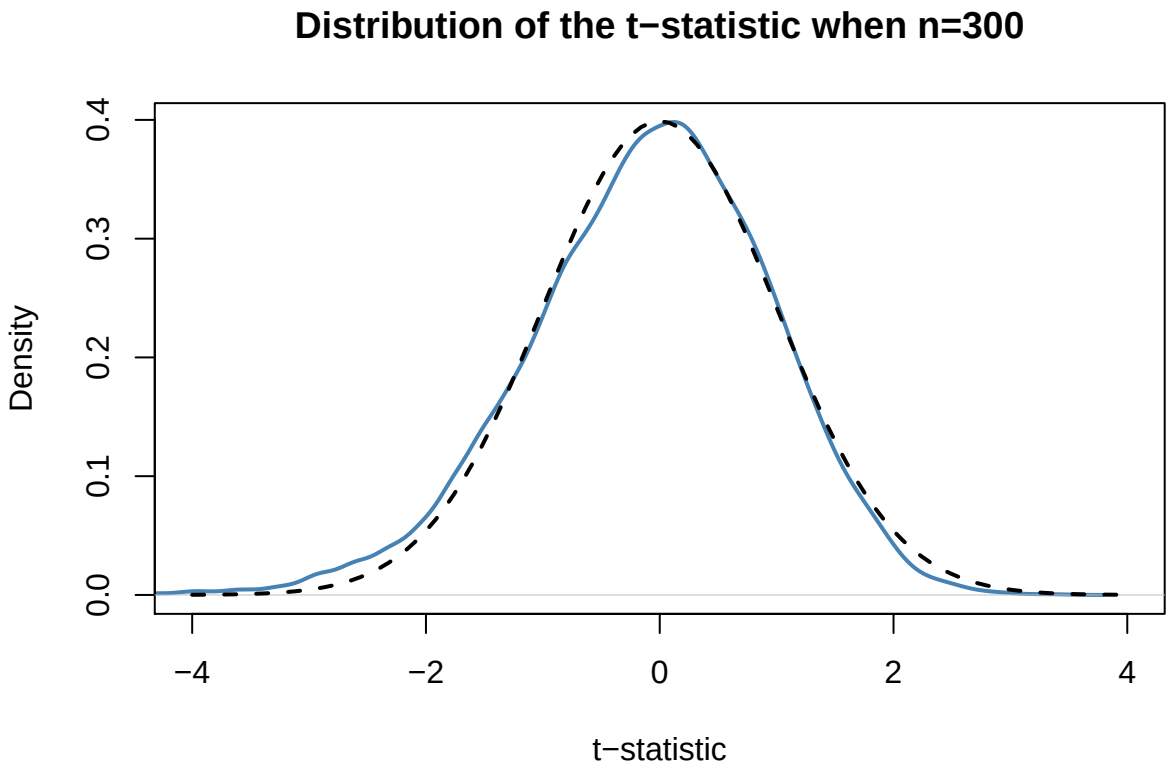
```
## [1] 0.6154575
```

Using Rwe can show that if $\mu_{Y,0}$ equals the true value, that is the null hypothesis is true, (3.8) is approximately distributed $N(0,1)$ when n is large.

```
# prepare empty vector for t-statistics
tstatistics <- numeric(10000)

# set sample size
n <- 300
```

```
# simulate 10000 t-statistics
for (i in 1:10000) {
  s <- sample(0:1,
              size = n,
              prob = c(0.9, 0.1),
              replace = T
              )
  tstatistics[i] <- (mean(s)-0.1)/(sqrt(mean(s)*(1-mean(s))/n))
}
```

```
# plot density and compare to N(0,1) density
plot(density(tstatistics),
     xlab = 't-statistic',
     main = 'Distribution of the t-statistic when n=300',
     lwd = 2,
     xlim = c(-4, 4),
     col = 'steelblue'
     )

# N(0,1) density (dashed)
curve(dnorm(x),
      add = T,
      lty = 2,
      lwd = 2
      )
```



Judging from the plot, the normal approximation works reasonably well for the chosen sample size. This normal approximation has already been used in the definition of the p-value, see (3.8).

Hypothesis Testing with a Prespecified Significance Level

Key Concept 3.5

The Terminology of Hypothesis Testing

In hypothesis testing, two types of mistakes are possible:

1. The null hypothesis *is* rejected although it is true (α-error / type-I-error)
2. The null hypothesis *is not* rejected although it is false (β-error / type-II-error)

The **significance level** of the test is the probability to commit a type-I-error we are willing to accept in advance. E.g. using a prespecified significance level of 0.05, we reject the null hypothesis if and only if the p-value is less than 0.05. The significance level is chosen before the test is conducted.

An equivalent procedure is to reject the null hypothesis if the test statistic observed is, in absolute value terms, larger than the **critical value** of the test statistic. The critical value is determined by the significance level chosen and defines two disjoint sets of values which are called **acceptance region** and **rejection region**. The acceptance region contains all values of the test statistic for which the test does not reject while the rejection region contains all the values for which the test does reject.

The **p-value** is the probability that, in repeated sampling under the same conditions, meaning i.i.d. sampling, the same null hypothesis and the same sample size, a test statistic is observed that provides just as much evidence against the null hypothesis as the test statistic actually observed.

The actual probability that the test rejects the true null hypothesis is called the **size of the test**. In an ideal setting, the size does not exceed the significance level.

The probability that the test correctly rejects a false null hypothesis is called **power**.

Reconsider `pvalue` computed further above:

```
# check whether p-value < 0.05
pvalue < 0.05
```

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

The condition is not fulfilled so we do not reject the null hypothesis (remember that the null hypothesis is true in this example).

When working with a t-statistic instead, it is equivalent to apply the following rule:

$$\text{Reject } H_0 \text{ if } |t^{act}| > 1.96$$

We reject the null hypothesis at the significance level of 5% if the computed t-statistic lies beyond the critical value of 1.96 in absolute value terms. 1.96 is the 0.05-quantile of the standard normal distribution.

```
# check the critical value
qnorm(p = 0.05)
```

```
## [1] -1.644854
```

```
# check whether the null is rejected using the t-statistic computed further above
abs(tstatistic) > 1.96
```

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

Just like using the p-value, we cannot reject the null hypothesis using the corresponding t-statistic. Key Concept 3.6 summarizes the procedure of performing a two-sided hypothesis about the population mean $E(Y)$.

Key Concept 3.6

Testing the Hypothesis $E(Y) = \mu_{Y,0}$ Against the Alternative $E(Y) \neq \mu_{Y,0}$

1. Estimate μ_Y using $\overline{Y}$ and compute the standard error of $\overline{Y}$, $SE(\overline{Y})$.
2. Compute the t-statistic.
3. Compute the p-value and reject the null hypothesis at the 5% level of significance if the p-value is smaller than 0.05 or equivalently, if

$$\left|t^{act}\right| > 1.96.$$

One-sided Alternatives

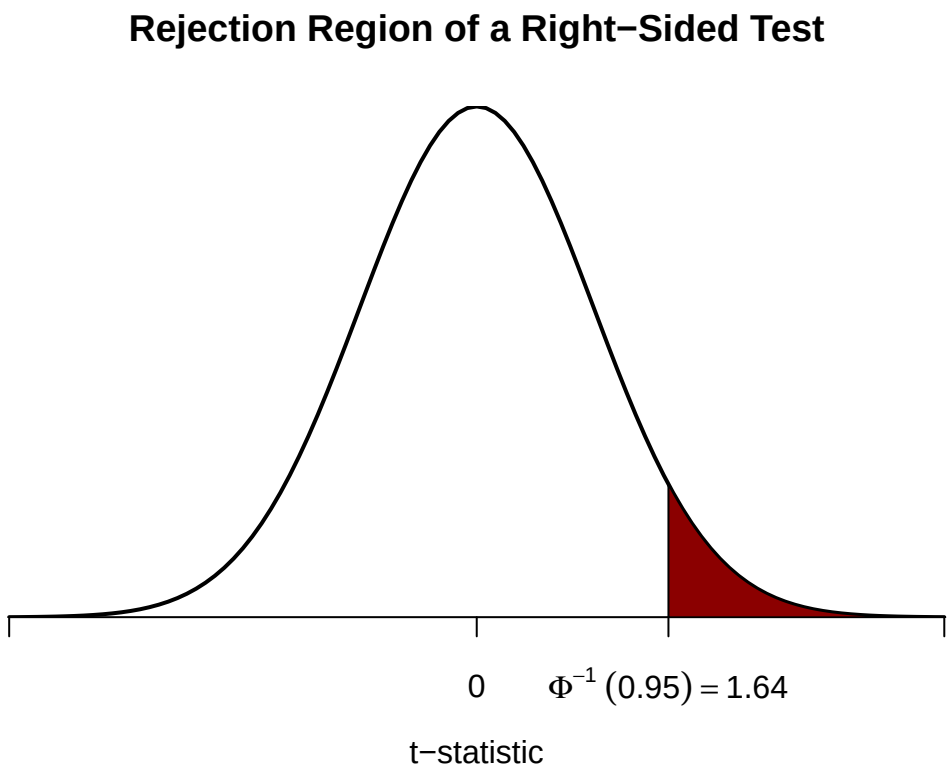
Sometimes we are interested in finding evidence that the mean is bigger or smaller than some value hypothesized under the null. One can come up with many examples here but, to stick to the book, take the presumed wage gap between well and less educated working individuals. Since we hope that this differential exists, a relevant alternative (to the null hypothesis that there is no wage differential) is that good educated individuals earn more, i.e. that the average hourly wage for this group, μ_Y is *bigger* than $\mu_{Y,0}$ the know average wage of less educated workers.

This is an example of a *right-sided test* and the hypotheses pair is chosen to be

$$H_0 : \mu_Y = \mu_{Y,0} \quad \text{vs} \quad H_1 : \mu_Y > \mu_{Y,0}.$$

We reject the null hypothesis if the computed test-statistic is larger than the critical value 1.64, the 0.95-quantile of the $N(0,1)$ distribution. This ensures that $1 - 0.95 = 5\%$ probability mass remains in the area to the right of the critical value. Similar as before we can visualize this in `R` using the function `polygon()`.

```
# plot the standard normal density on the domain [-4,4]
curve(dnorm(x),
      xlim = c(-4, 4),
      main = 'Rejection Region of a Right-Sided Test',
      yaxs = 'i',
      xlab = 't-statistic',
      ylab = '',
      lwd = 2,
      axes = 'F'
)

# add x-axis
axis(1,
     at = c(-4, 0, 1.64, 4),
     padj = 0.5,
     labels = c('', 0, expression(Phi^-1~(.95)==1.64), '')
)

# shade rejection region in the left tail
polygon(x = c(1.64, seq(1.64, 4, 0.01), 4),
        y = c(0, dnorm(seq(1.64, 4, 0.01)), 0),
        col = 'darkred'
)
```



Analogously for the left-sided test we have

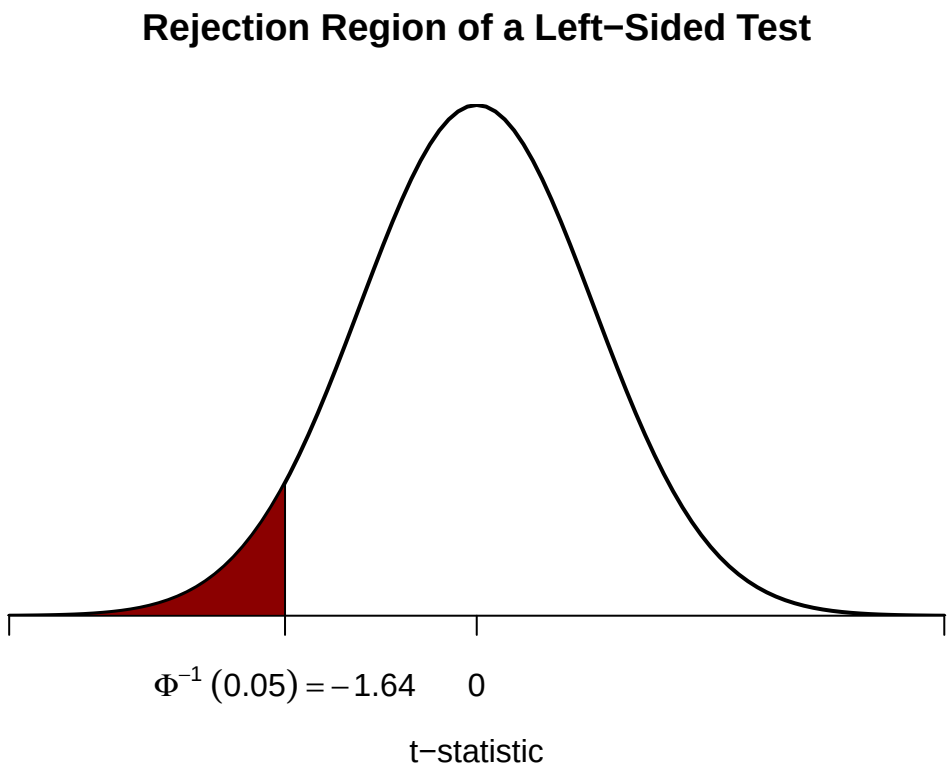
$$H_0 : \mu_Y = \mu_{Y,0} \quad \text{vs.} \quad H_1 : \mu_Y < \mu_{Y,0}.$$

The null is rejected if the observed test statistic falls short of the critical value which, for a test at the 0.05 level of significance, is given by -1.64, the 0.05-quantile of the $N(0,1)$ distribution. 5% probability mass lies to the left of the critical value.

It is straightforward to adapt the code chunk above to the case of a left-sided test. We only have to adjust the color shading and the tick marks.

```
# plot the standard normal density on the domain [-4,4]
curve(dnorm(x),
      xlim = c(-4, 4),
      main = 'Rejection Region of a Left-Sided Test',
      yaxs = 'i',
      xlab = 't-statistic',
      ylab = '',
      lwd = 2,
      axes = 'F'
)

# add x-axis
axis(1,
     at = c(-4, 0, -1.64, 4),
     padj = 0.5,
     labels = c('', 0, expression(Phi^-1~(.05)==-1.64), '')
)

# shade rejection region in right tail
polygon(x = c(-4, seq(-4, -1.64, 0.01), -1.64),
```

```
        y = c(0, dnorm(seq(-4, -1.64, 0.01)), 0),
        col = 'darkred'
)
```



3.4 Confidence Intervals for the Population Mean

As stressed before, we will never estimate the *exact* value of the population mean of Y using a random sample. However, we can compute confidence intervals for the population mean. In general, a confidence interval for a unknown parameter is a set of values that contains the true parameter with a prespecified probability, the *confidence level*. Confidence intervals are computed using the information available in the sample. Since this information is the result of a random process, confidence intervals are random variables themselves.

Key Concept 3.7 shows how to compute confidence intervals for the unknown population mean $E(Y)$.

Key Concept 3.7

Confidence Intervals for the Population Mean

A 95% confidence interval for μ_Y is a `random variable` that contains the true μ_Y in 95% of all possible random samples. When n is large we can use the normal approximation. Then, 99%, 95%, 90% confidence intervals are

$$99\% \text{ confidence interval for } \mu_Y = \left\{\overline{Y} \pm 2.58 \times SE(\overline{Y})\right\}, \tag{3.9}$$

$$95\% \text{ confidence interval for } \mu_Y = \left\{\overline{Y} \pm 1.96 \times SE(\overline{Y})\right\}, \tag{3.10}$$

$$90\% \text{ confidence interval for } \mu_Y = \left\{\overline{Y} \pm 1.64 \times SE(\overline{Y})\right\}. \tag{3.11}$$

These confidence intervals are sets of null hypotheses we cannot reject in a two-sided hypothesis test at the given level of confidence.

Now consider the following statements.

1. The interval
$$\left\{\overline{Y} \pm 1.96 \times SE(\overline{Y})\right\}$$
covers the true value of μ_Y with a probability of 95%.

2. We have computed $\overline{Y} = 5.1$ and $SE(\overline{Y} = 2.5$ so the interval
$$\{5.1 \pm 1.96 \times 2.5\} = [0.2, 10]$$
covers the true value of μ_Y with a probability of 95%.

While 1. is right (this is exactly in line with the definition above), 2. is completely wrong and none of your lecturers wants to read such a sentence in a term paper, written exam or similar, believe us. The difference is that, while 1. is the definition of a random variable, 2. is one possible *outcome* of this random variable so there is no meaning in making any probabilistic statement about it. Either the computed interval *does cover* μ_Y or it *does not*!

In R, testing of hypotheses about the mean of a population on the basis of a random sample is very easy due to functions like `t.test()` from the `stats` package. It produces an object of type `list`. Luckily, one of the most simple ways to use `t.test()` is when you want to obtain a 95% confidence interval for some population mean. We start by generating some random data and calling `t.test()` in conjunction with `ls()` to obtain a breakdown of the output components.

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

# generate some sample data
sampledata <- rnorm(100, 10, 10)

# checke type
typeof(t.test(sampledata))
```

```
## [1] "list"
```

```
# display list elements produced by t.test
ls(
  t.test(sampledata)
)
```

```
## [1] "alternative" "conf.int"    "data.name"   "estimate"    "method"
## [6] "null.value"  "p.value"     "parameter"   "statistic"
```

Though we find that many items are reported, at the moment we are only interested in computing a 95% confidence set for the mean.

```
t.test(sampledata)$"conf.int"
```

```
## [1]  9.306651 12.871096
## attr(,"conf.level")
## [1] 0.95
```

This tells us that the 95% confidence interval is

$$[9.31, 12.87]\,.$$

In this example, the computed interval obviously does cover the true μ_Y which we know to be 10.

Let us have a look at the whole standard output produced by `t.test()`.

```
t.test(sampledata)
```

```
##
##  One Sample t-test
##
## data:  sampledata
## t = 12.346, df = 99, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##   9.306651 12.871096
## sample estimates:
## mean of x
##  11.08887
```

We see that `t.test()` does not only compute a 95% confidence interval but automatically conducts a two-sided significance test of the hypothesis $H_0 : \mu_Y = 0$ at the level of 5% and reports relevant parameters thereof: the alternative hypothesis, the estimated mean, the resulting t-statistic, the degrees of freedom of the underlying t distribution (`t.test()` does not perform the normal approximation) and the corresponding p-value. This is very convenient!

In this example, we come to the conclusion that the population mean *is not* significantly different from 0 (which is correct) at the level of 5%, since $\mu_Y = 0$ is element of the 95% confidence interval

$$0 \in [-0.27, 0.12]\,.$$

We come to an equivalent result when using the p-value rejection rule since

$$p = 0.456 > 0.05.$$

3.5 Comparing Means from Different Populations

Suppose you are interested in the means of two different populations, denote them μ_1 and μ_2. More specifically you are interested whether these population means are different from each other and plan using a hypothesis test to verify this on the basis of independent sample data from both populations. A suitable pair of hypotheses is

$$H_0 : \mu_1 - \mu_2 = d_0 \quad \text{vs.} \quad H_1 : \mu_1 - \mu_2 \neq d_0 \tag{3.12}$$

where d_0 denotes the hypothesized difference in means. The book teaches us that H_0 can be tested with the t-statistic

$$t = \frac{(\overline{Y}_1 - \overline{Y}_2) - d_0}{SE(\overline{Y}_1 - \overline{Y}_2)} \tag{3.13}$$

where

$$SE(\overline{Y}_1 - \overline{Y}_2) = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}. \tag{3.14}$$

This is called a two sample t-test. For large n_1 and n_2, (3.13) is standard normal distributed under the null hypothesis. Analogously to the simple t-test we can compute confidence intervals for the true difference in population means:

$$(\overline{Y}_1 - \overline{Y}_2) \pm 1.96 \times SE(\overline{Y}_1 - \overline{Y}_2)$$

is a 95% confidence interval for d. In `R`, hypotheses as in (3.12) can be tested with `t.test()`, too. Note that `t.test()` chooses $d_0 = 0$ by default. This can be changed by setting the argument `mu` accordingly.

The subsequent code chunk demonstrates how to perform a two sample t-test in `R` using simulated data.

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

# draw data from two different populations with equal mean
sample_pop1 <- rnorm(100, 10, 10)
sample_pop2 <- rnorm(100, 10, 20)

# perform a two sample t-test
t.test(sample_pop1, sample_pop2)
```

```
##
##  Welch Two Sample t-test
##
## data:  sample_pop1 and sample_pop2
## t = 0.872, df = 140.52, p-value = 0.3847
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.338012  6.028083
## sample estimates:
## mean of x mean of y
## 11.088874  9.243838
```

We find that the two sample t-test does not reject the (true) null hypothesis that $d_0 = 0$.

3.6 An Application to the Gender Gap of Earnings

In this section discusses how to reproduce the results presented in the box *The Gender Gap of Earnings of College Graduates in the United States* of the book.

In order to reproduce table 3.1 you need to download the replication data which are hosted by Pearson and can be found and downloaded here. Download the data for chapter three as an excel spreadsheet (`cps_ch3.xlsx`). This data set contains data that range from 1992 to 2008 and earnings are reported in prices of 2008. There are several ways to import the `.xlsx`-files into `R`. Our suggestion is the function `read_excel()` from the `readxl` package. The package is not part of `R`'s base version and has to be installed manually.

```
# load the 'readxl' package
library(readxl)
```

You are now ready to import the data set. Make sure you use the correct path to import the downloaded file! In our example, the file is saved in a sub folder (`data`) of the working directory. If you are not sure what your current working directory is, use `getwd()`, see also `?getwd()`. This will give you the path that points to the place `R`is currently looking for files.

```
# import the data into R
cps <- read_excel(path = 'data/cps_ch3.xlsx')
```

Next, install and load the package `dyplr`. This package provides some handy functions that simplify data wrangling a lot. It makes use of the `%>%` operator.

```
# load the 'dplyr' package
library('dplyr')
```

First, get an overview over the data set. Next, use `%>%` and some functions from the `dplyr` package to group the observations by gender and year and compute descriptive statistics for both groups.

```
# Get an overview of the data structure
head(cps)
```

```
## # A tibble: 6 x 3
##   a_sex  year ahe08
##   <dbl> <dbl> <dbl>
## 1    1. 1992.  17.2
## 2    1. 1992.  15.3
## 3    1. 1992.  22.9
## 4    2. 1992.  13.3
## 5    1. 1992.  22.1
## 6    2. 1992.  12.2
```

```
# group data by gender and year and compute the mean, standard deviation
# and number of observations for each group
avgs <- cps %>%
        group_by(a_sex, year) %>%
        summarise(mean(ahe08),
                  sd(ahe08),
                  n()
                  )

# print results to the console
print(avgs)
```

```
## # A tibble: 10 x 5
## # Groups:   a_sex [?]
##    a_sex  year `mean(ahe08)` `sd(ahe08)` `n()`
##    <dbl> <dbl>         <dbl>       <dbl> <int>
##  1    1. 1992.          23.3       10.2   1594
##  2    1. 1996.          22.5       10.1   1379
##  3    1. 2000.          24.9       11.6   1303
##  4    1. 2004.          25.1       12.0   1894
##  5    1. 2008.          25.0       11.8   1838
##  6    2. 1992.          20.0        7.87  1368
##  7    2. 1996.          19.0        7.95  1230
##  8    2. 2000.          20.7        9.36  1181
##  9    2. 2004.          21.0        9.36  1735
## 10    2. 2008.          20.9        9.66  1871
```

With the pipe operator `%>%` we simply chain different `R` functions that produce compatible input and output. In the code above, we take the data set `cps` and use it as an input for the function `group_by()`. The output of `group_by` is subsequently used as an input for `summarise()` and so forth.

Now that we have computed the statistics of interest for both genders, we can investigate how the gap in earnings between both groups evolves over time.

```
# split the data set by gender
male <- avgs %>% filter(a_sex == 1)
```

```
female <- avgs %>% filter(a_sex == 2)

# Rename columns of both splits
colnames(male)   <- c("Sex", "Year", "Y_bar_m", "s_m", "n_m")
colnames(female) <- c("Sex", "Year", "Y_bar_f", "s_f", "n_f")

# Estimate gender gaps, compute standard errors and confidence intervals for all dates
gap <- male$Y_bar_m - female$Y_bar_f

gap_se <- sqrt(male$s_m^2 / male$n_m + female$s_f^2 / female$n_f)

gap_ci_l <- gap - 1.96 * gap_se

gap_ci_u <- gap + 1.96 * gap_se

result <- cbind(male[,-1], female[,-(1:2)], gap, gap_se, gap_ci_l, gap_ci_u)

# print results to the console
print(result, digits = 3)
```

```
##   Year Y_bar_m  s_m  n_m Y_bar_f  s_f  n_f  gap gap_se gap_ci_l gap_ci_u
## 1 1992    23.3 10.2 1594    20.0 7.87 1368 3.23  0.332     2.58     3.88
## 2 1996    22.5 10.1 1379    19.0 7.95 1230 3.49  0.354     2.80     4.19
## 3 2000    24.9 11.6 1303    20.7 9.36 1181 4.14  0.421     3.32     4.97
## 4 2004    25.1 12.0 1894    21.0 9.36 1735 4.10  0.356     3.40     4.80
## 5 2008    25.0 11.8 1838    20.9 9.66 1871 4.10  0.354     3.41     4.80
```

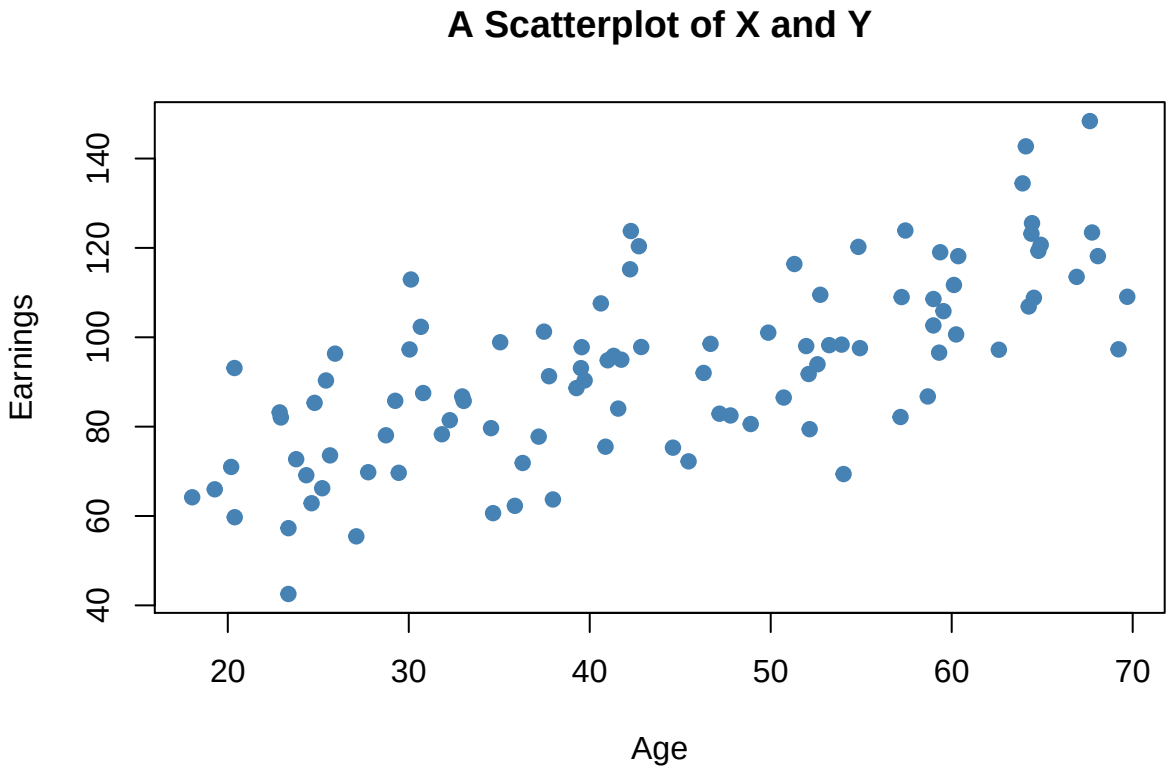
We observe virtually the same results as the ones presented in the book. The computed statistics suggest that there *is* a gender gap in earnings. Note that we can reject the null hypothesis that the gap is zero for all periods. Further, estimates of the gap and limits of the 95% confidence intervals indicate that the gap has been quite stable over the recent past.

3.7 Scatterplots, Sample Covariance and Sample Correlation

A scatter plot represents two dimensional data, for example n observation on X_i and Y_i, by points in a Cartesian coordinate system. It is very easy to generate scatter plots using the `plot()` function in `R`. Let us generate some fictional data on age and earnings of workers and plot it.

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

# generate data set
X <- runif(n = 100,
           min = 18,
           max = 70
           )
Y <- X + rnorm(n=100, 50, 15)

# plot observations
plot(X,
     Y,
     type = "p",
     main = "A Scatterplot of X and Y",
```

```
     xlab = "Age",
     ylab = "Earnings",
     col = "steelblue",
     pch = 19
     )
```



The plot shows positive correlation between age and earnings. This is in line with the assumption that older workers earn more than those who joined the working population recently.

Sample Covariance and Correlation

By now you should be familiar with the concepts of variance and covariance. If not, we recommend you to work your way through chapter 2 of the book (again).

Just like the variance, covariance and correlation of two variables are properties that relate to the (unknown) joint probability distribution of these variables. We can estimate covariance and correlation by means of suitable estimators using a random sample (X_i, Y_i), $i = 1, \dots, n$.

The sample covariance

$$s_{XY} = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})$$

is an estimator for the population variance of X and Y whereas the sample correlation

$$r_{XY} = \frac{s_{XY}}{s_X s_Y}$$

can be used to estimate the population correlation, a standardized measure for the strength of the linear relationship between X and Y. See chapter 3.7 in the book for a more detailed treatment of these estimators.

As for variance and standard deviation, these estimators are implemented as `R` functions in the `stats` package. We can use them to estimate population covariance and population correlation of the fictional data on age and earnings.

```
# compute sample covariance of X and Y
cov(X, Y)
```

```
## [1] 213.934
```

```
# compute sample correlation between X and Y
cor(X, Y)
```

```
## [1] 0.706372
```

```
# equivalent way to compute the sample correlation
cov(X, Y)/(sd(X) * sd(Y))
```

```
## [1] 0.706372
```

The estimates indicate that X and Y are moderately correlated.

The next code chunk uses the function `mvnorm()` from package `MASS` to generate bivariate sample data with different degree of correlation.

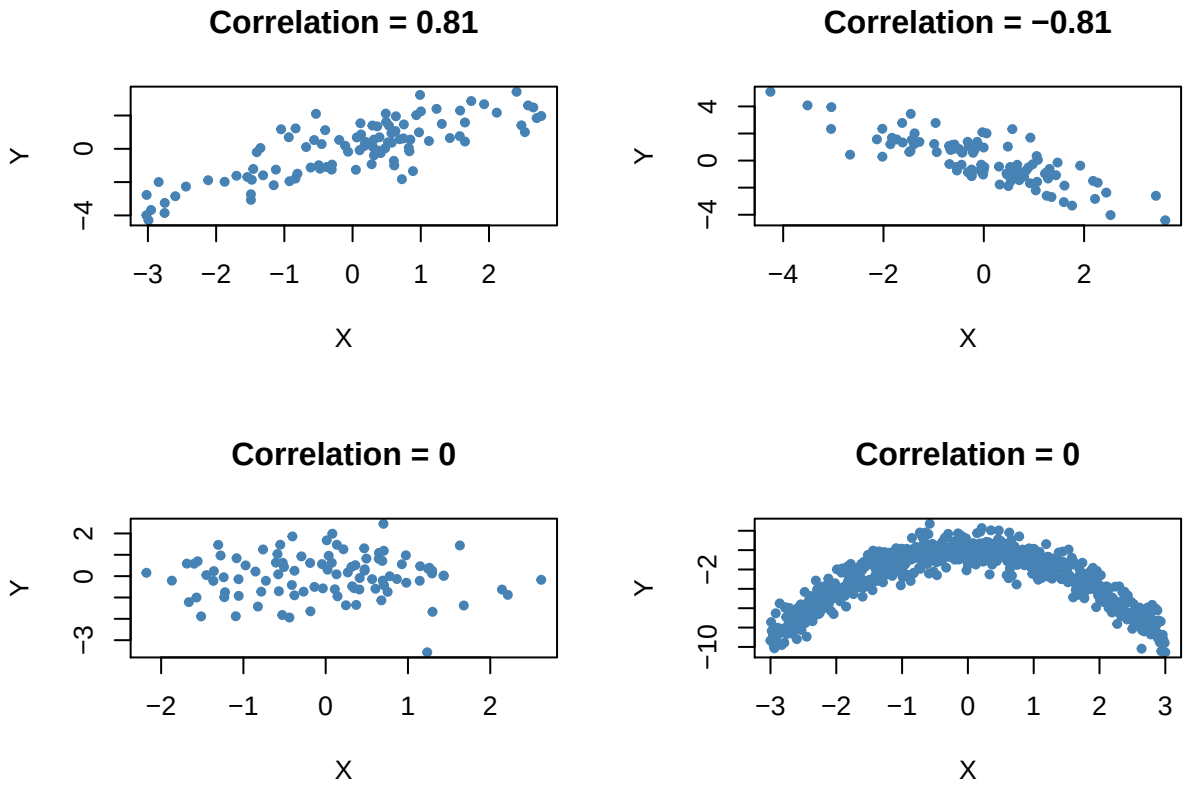
```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

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

# positive correlation (0.81)
example1 <- mvrnorm(100,
                    mu = c(0, 0),
                    Sigma = matrix(c(2, 2, 2, 3), ncol = 2),
                    empirical = TRUE
                    )

# negative correlation (-0.81)
example2 <- mvrnorm(100,
                    mu = c(0, 0),
                    Sigma = matrix(c(2, -2, -2, 3), ncol = 2),
                    empirical = TRUE
                    )

# no correlation
example3 <- mvrnorm(100,
                    mu = c(0, 0),
                    Sigma = matrix(c(1, 0, 0, 1), ncol = 2),
                    empirical = TRUE
                    )

# no correlation (quadratic relationship)
X <- seq(-3, 3, 0.01)
```

```
Y <- -X^2 + rnorm(length(X))

example4 <- cbind(X, Y)

# divide plot area as 2-by-2 array
par(mfrow = c(2, 2))

# plot data sets
plot(example1, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = 0.81")

plot(example2, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = -0.81")

plot(example3, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = 0")

plot(example4, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = 0")
```



3.8 Exercises

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

Chapter 4

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 the class sizes by hiring new teachers, that is the school lowers X_1, the student-teacher ratios of their 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* and provides useful functions and data sets.
- `MASS` is a collection of functions for applied statistics.

4.1 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
```

If we use a simple linear regression model, we assume that the true relationship between both variables can be represented by a straight line, formally

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

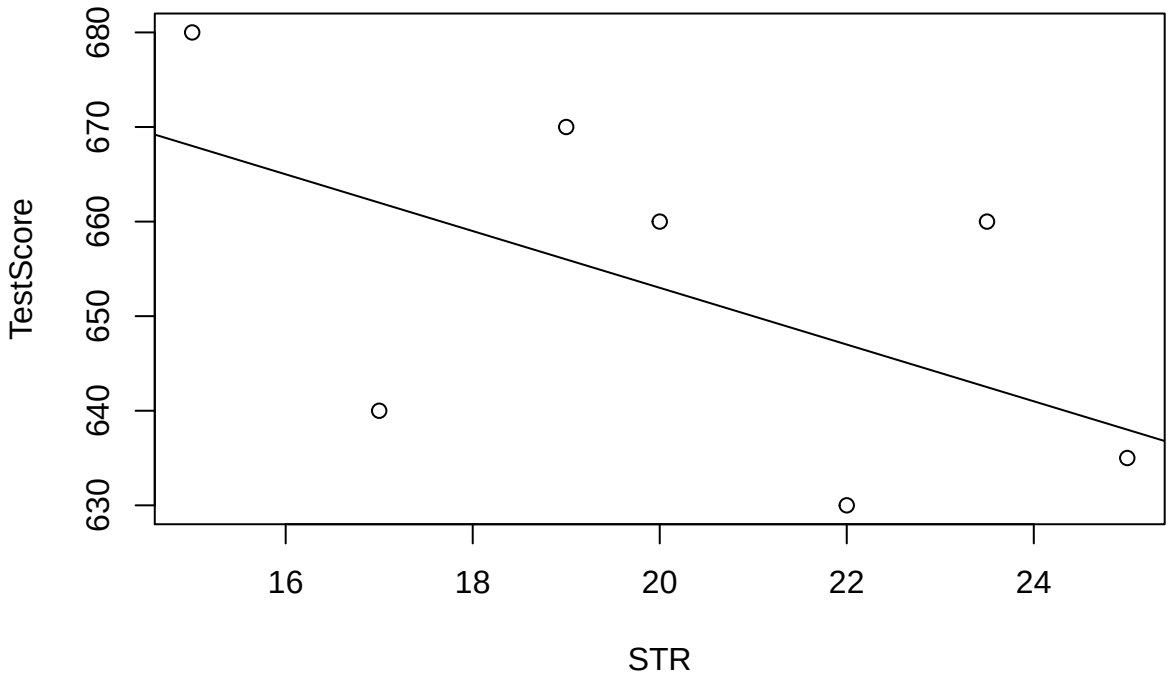
For now, let us suppose that the true function which relates test score and student-teacher ratio to each other is

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

If possible, it is always a good idea to visualize the data you work with in an appropriate way. For our purpose it is suitable to use the function `plot()` to produce a scatter plot with `STR` on the X-axis and `TestScore` on the Y axis. An easy way to do so is to 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 the true 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 scatter plot.

The following code reproduces figure 4.1 from the textbook.

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

# add the true 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 true relationship. The reason for this is the core problem of statistics, *randomness*. Most of the time there are influences which cannot be explained in a purely deterministic fashion and thus complicate finding the true relationship.

In order to account for these differences between observed data and the true relationship, we extend our model from above by an *error term* u which covers these random effects. Put differently, u accounts for all the differences between the true 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 imaginable 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*.

4.2 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 `data()` function.

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

# load the AER package
library(AER)

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

Note that 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!

For several reasons 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 and alike 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 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 this!

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 text book summarizes the distribution of test scores and student-teacher ratios. Remember that 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 have a nice display format we gather the computed measures in a `data.frame` named `DistributionSummary`.

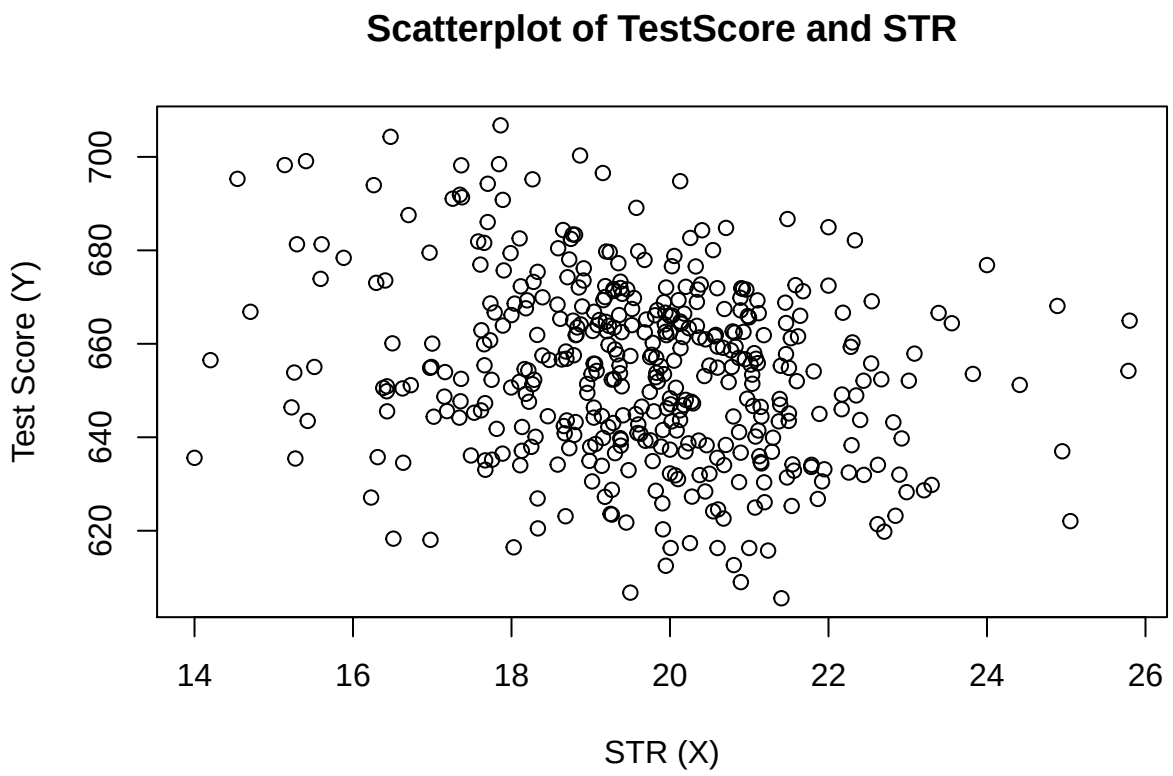
```
# compute sample averages of STR and score
avg_STR <- mean(CASchools$STR)
avg_score <- mean(CASchools$score)

# 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 done for the sample data, we use `plot()` for a visual survey. This allows us to detect specific characteristics of our data, such as outliers which are hard 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 states the dependent variable and the regressor. 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` are adding 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 scatter plot of all observations on student-teacher ratio and Test score. We see that the points are strongly scattered and an apparent relationship cannot be detected by only looking at them. Yet it can be assumed that both variables are negatively correlated, that is we expect to observe lower test scores in bigger classes.

The function `cor()` (type and execute `?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 scatter plot already suggests, the correlation is negative but rather weak.

The task we are facing now is to find a line which fits best to 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 is pretty unscientific and prone to subjective perception: different students would draw different regression lines. On this account, we are interested in techniques that are more sophisticated. Such a technique is 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. Therefore 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

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{n}(X_i - \overline{X})^2} \tag{4.1}$$

$$\tag{4.2}$$

$$\hat{\beta}_0 = \overline{Y} - \hat{\beta}_1 \overline{X} \tag{4.3}$$

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

$$\widehat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i, \tag{4.4}$$

$$\tag{4.5}$$

$$\hat{u}_i = Y_i - \widehat{Y}_i. \tag{4.6}$$

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. If 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
beta_1 <- sum((STR - mean(STR))*(score - mean(score))) / sum((STR - mean(STR))^2)

# compute beta_0
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 also other and even more manual ways to do the same tasks. Luckily, OLS is one of the most widely-used estimation techniques. Being a statistical programming language, R already contains a built-in function named `lm()` (**l**inear **m**odel) which can be used to carry out regression analysis.

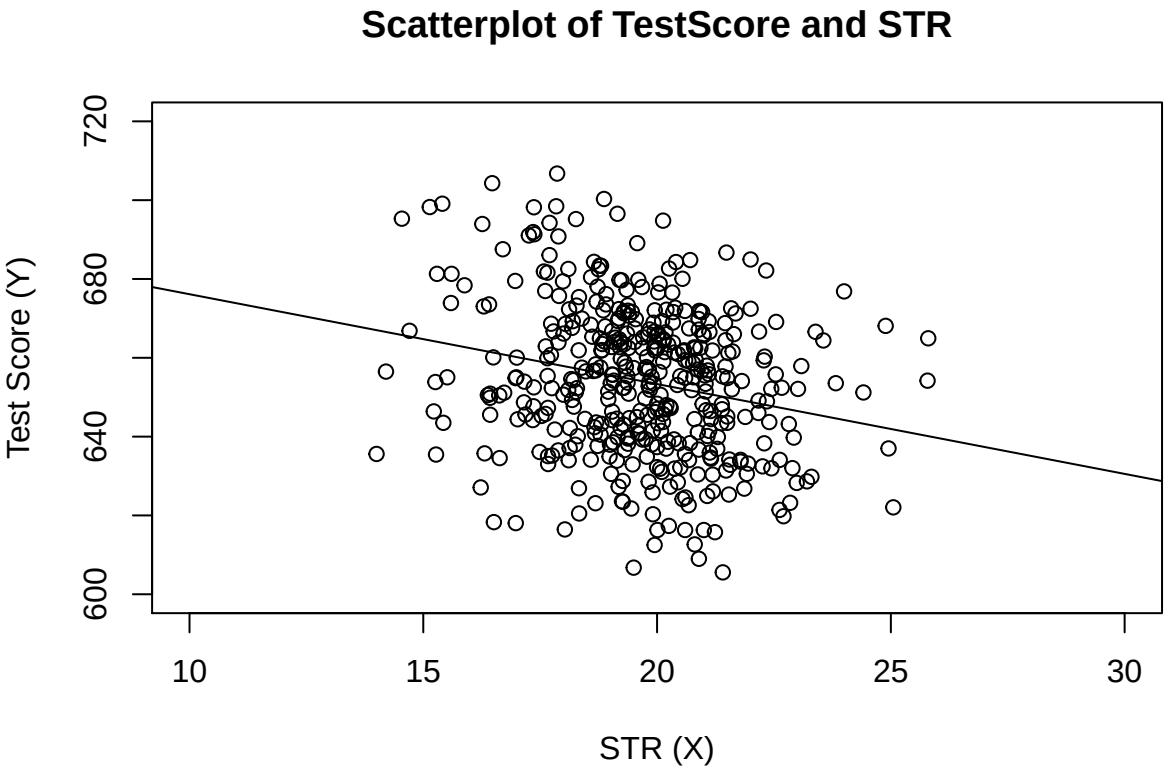
The first argument of the function to be specified is, similar to `plot()`, the regression formula with the basic syntax `y ~ x` where "`)y")` is the dependent variable and `x` the explanatory variable. The argument `data` determines the data set to be used in the regression. We now revisit the example from the book where the relationship between the test scores and the class sizes is analysed. 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 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 variable, `R` draws the regression line automatically!

4.3 Measures of Fit

After estimating a linear regression model, the question occurs how well the model describes the data. Visually this amounts to assess whether the observations are tightly clustered around the regression line, or if they are spread out. 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 thr 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

$$ESS = \sum_{i=1}^{n} \left(\hat{Y}_i - \overline{Y} \right)^2, \tag{4.7}$$

$$\tag{4.8}$$

$$TSS = \sum_{i=1}^{n} \left(Y_i - \overline{Y} \right)^2, \tag{4.9}$$

$$\tag{4.10}$$

$$R^2 = \frac{ESS}{TSS}. \tag{4.11}$$

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 regression error $\hat{u}_i$. As such it measures the magnitude of a typical deviation from the regression line, i.e. the magnitude of a typical regression error.

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

Remember that the u_i are *unobserved*. That 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* much of the variation in test scores remains unexplained (cf. Figure 4.3 of the book).

The SER is called *Residual standard error* and takes the value 18.58. The unit of the SER is the same as the unit of the dependent variable. In our context we can interpret the value as follows: 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 by 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`?

4.4 The Least Squares Assumptions

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

Key Concept 4.3

The Least Squares Assumptions

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

where

1. The error term u_i has conditional mean zero given X_i: $E(u_i|X_i) = 0$.
2. $(X_i, Y_i), i = 1, ..., n$ 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 $E(u) = 0$ 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 build in random number generators.

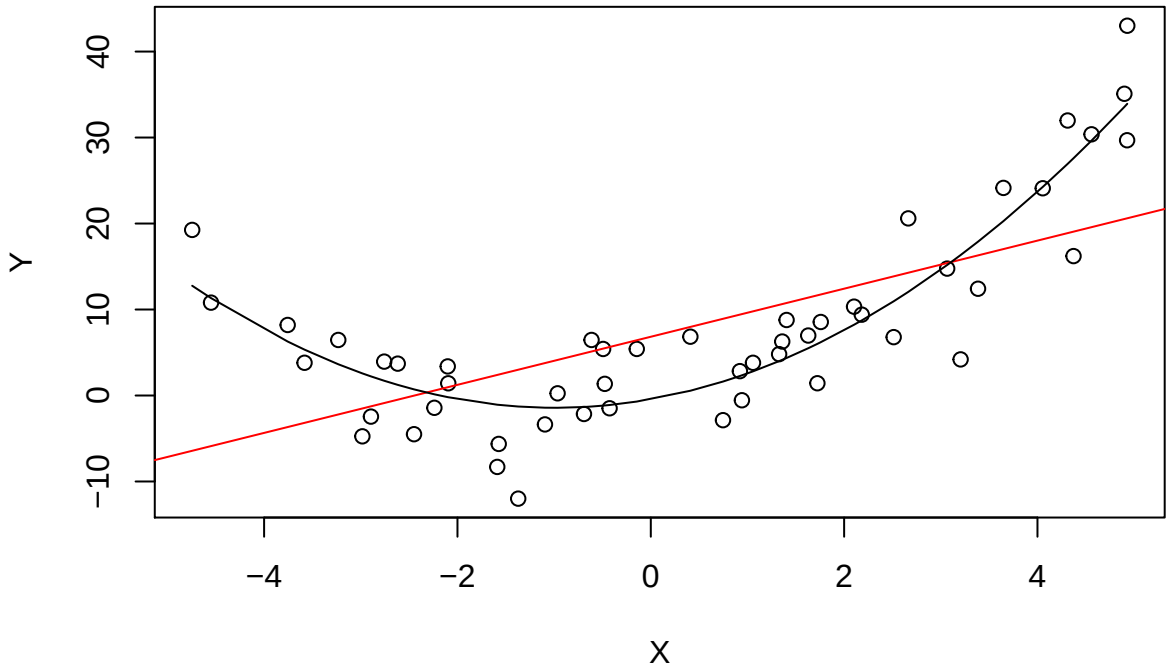
We will use the following functions you should be familiar with:

- `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 randomly scattered on the interval $[-5, 5]$. For our example we decide to generate uniformly distributed random numbers. 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. 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.

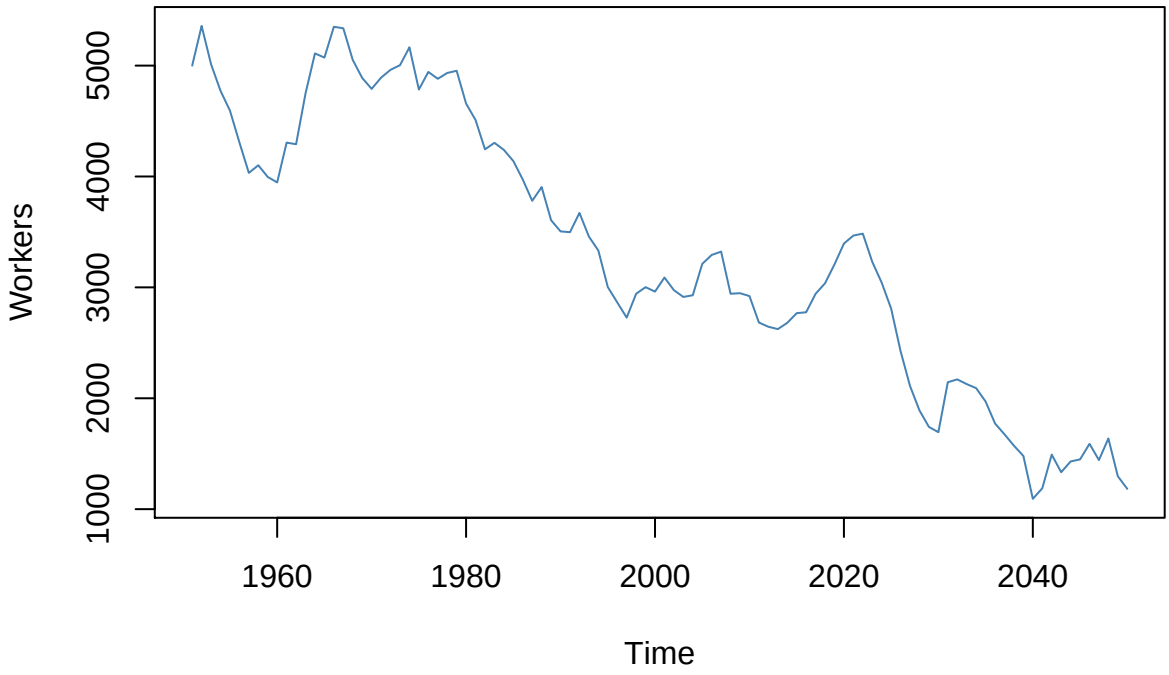
A prominent example where the i.i.d. assumption is not fulfilled is time series data where we have observations on the same unit over time. For example, take X as the number of workers employed by a production company over the course of time. Due to technological change, the company makes job cuts periodically but there are also some non-deterministic influences that relate to economics, politics and alike. 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 a simple autoregressive process that exhibits a downward trend and has normally distributed errors:[1]

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

[1] See Chapter 14 of the book for more on autoregressive processes and time series analysis in general.

```
# set random seed
set.seed(7)

# initialize the employment vector
X <- c(5000, rep(NA,99))

# generate a date vector
Date <- seq(as.Date("1951/1/1"), as.Date("2050/1/1"), "years")

# generate time series observations with random influences
for (i in 2:100) {
    X[i] <- 0.98 * X[i-1] + rnorm(1, sd = 200)
}

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



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*.

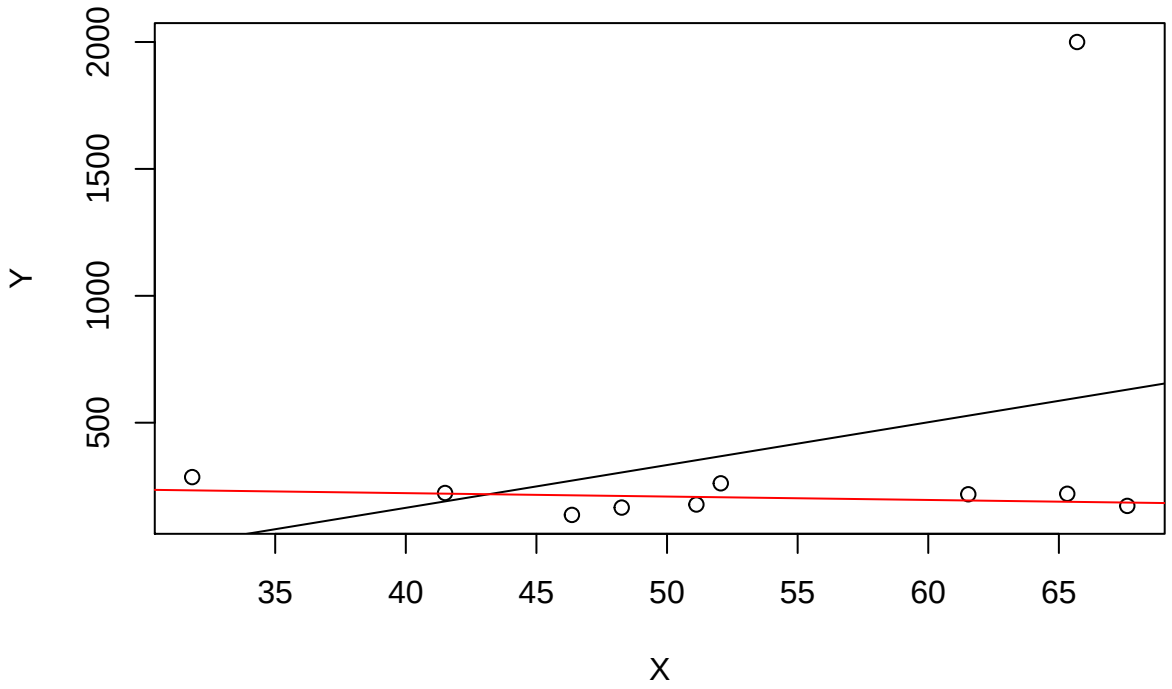
[2]See chapter 4.4 in the book.

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)

# 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")
```



4.5 The Sampling Distribution of the OLS Estimator

Because $\hat{\beta}_0$ and $\hat{\beta}_1$ are computed from a randomly drawn 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 random samples. Although the sampling distribution of $\hat{\beta}_0$ and $\hat{\beta}_1$ can be complicated when the sample size is small and generally differs with the number of observation, n, it is possible 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 β_0 and β_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

$$\sigma^2_{\hat{\beta}_1} = \frac{1}{n}\frac{Var\left[(X_i - \mu_X)\, u_i\right]}{\left[Var\left(X_i\right)\right]^2}. \tag{4.1}$$

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

$$\sigma^2_{\hat{\beta}_0} = \frac{1}{n}\frac{Var\left(H_i u_i\right)}{\left[E\left(H_i^2\right)\right]^2}, \text{ where } H_i = 1 - \left[\frac{\mu_X}{E\left(X_i^2\right)}\right] X_i. \tag{4.2}$$

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 to 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.

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`. Then we 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`.

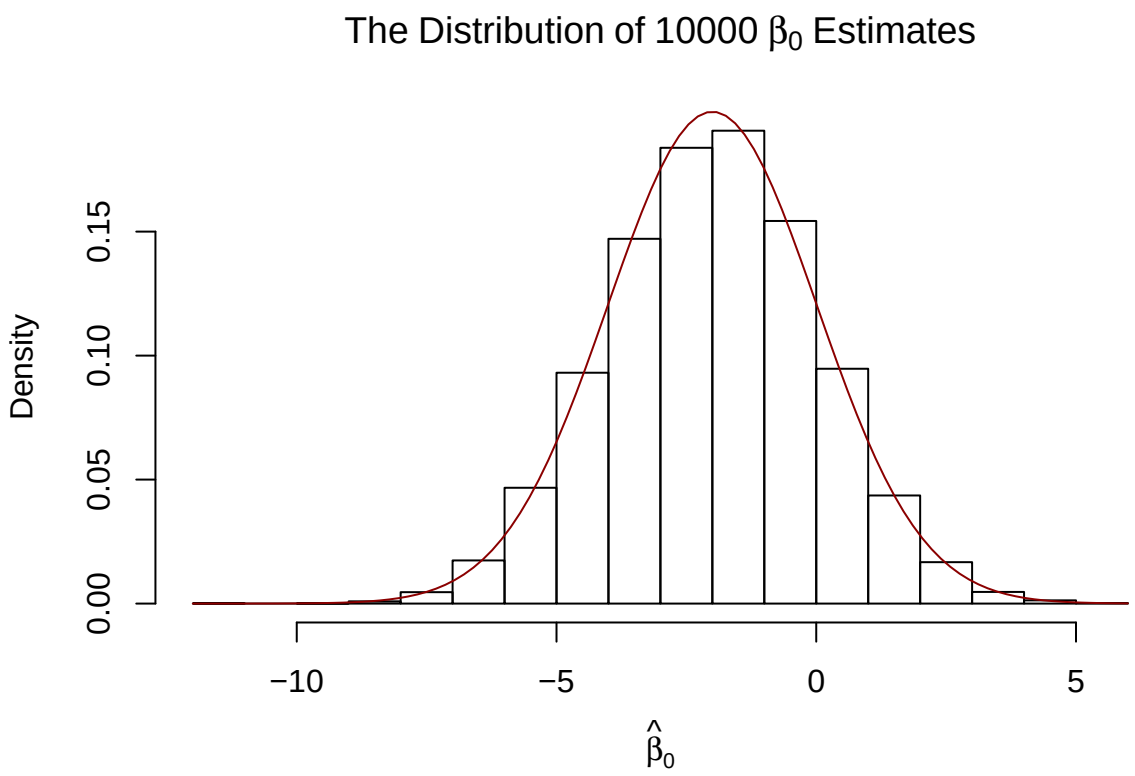
```
# set repetitions and sample size
n <- 100
reps <- 10000

# 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
```

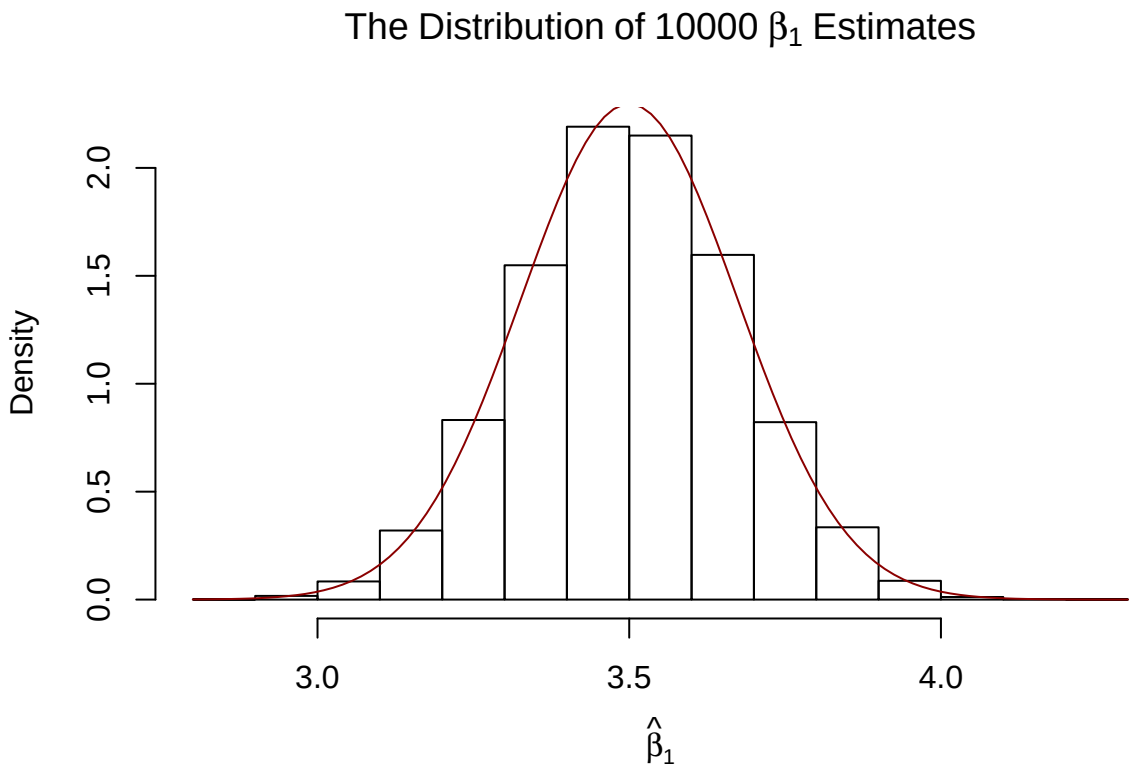
```
var(fit[, 2])
```

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

```
# 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")
```



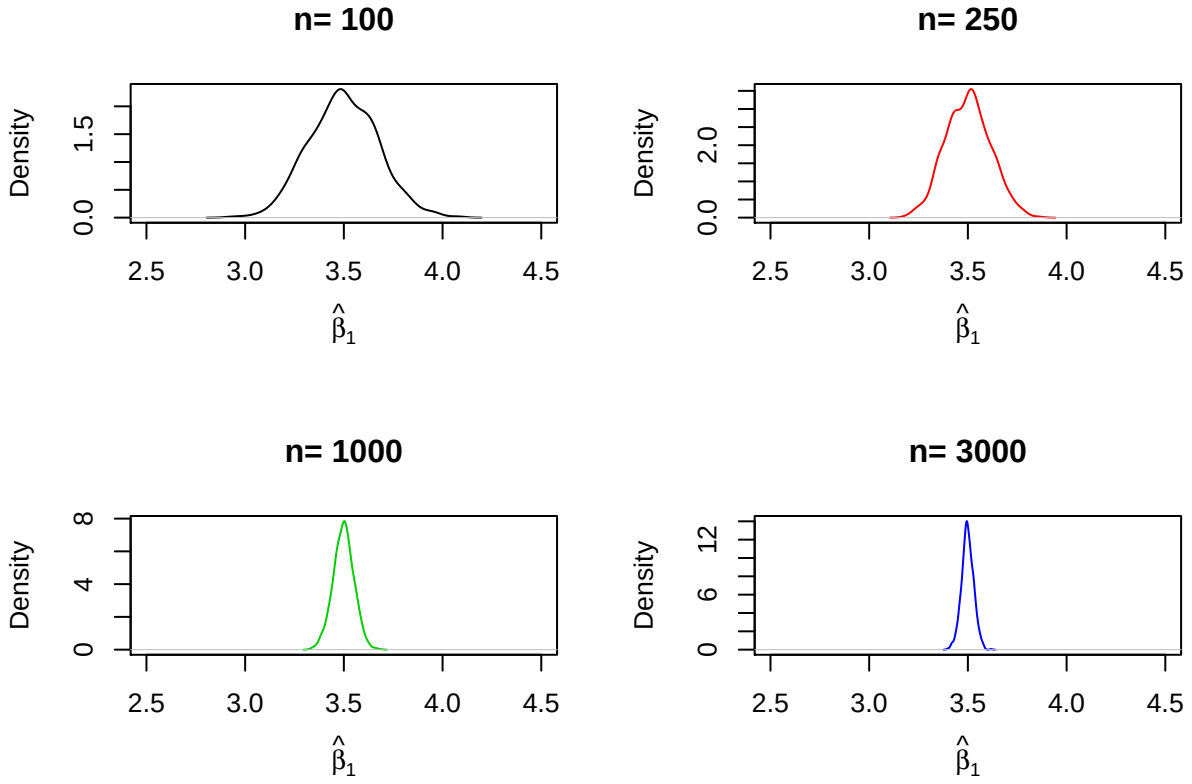
We are now able to say the following: first, our variance estimates are in favor of the claims made in Key Concept 4.4 since they come close to the computed theoretical values. Second, the histograms suggest that the distributions of the estimators can be well approximated by the respective theoretical 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 since their variances converge to 0 as n increases. We can check this by repeating the simulation above for an increasing sequence of 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)

# devide 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 are 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

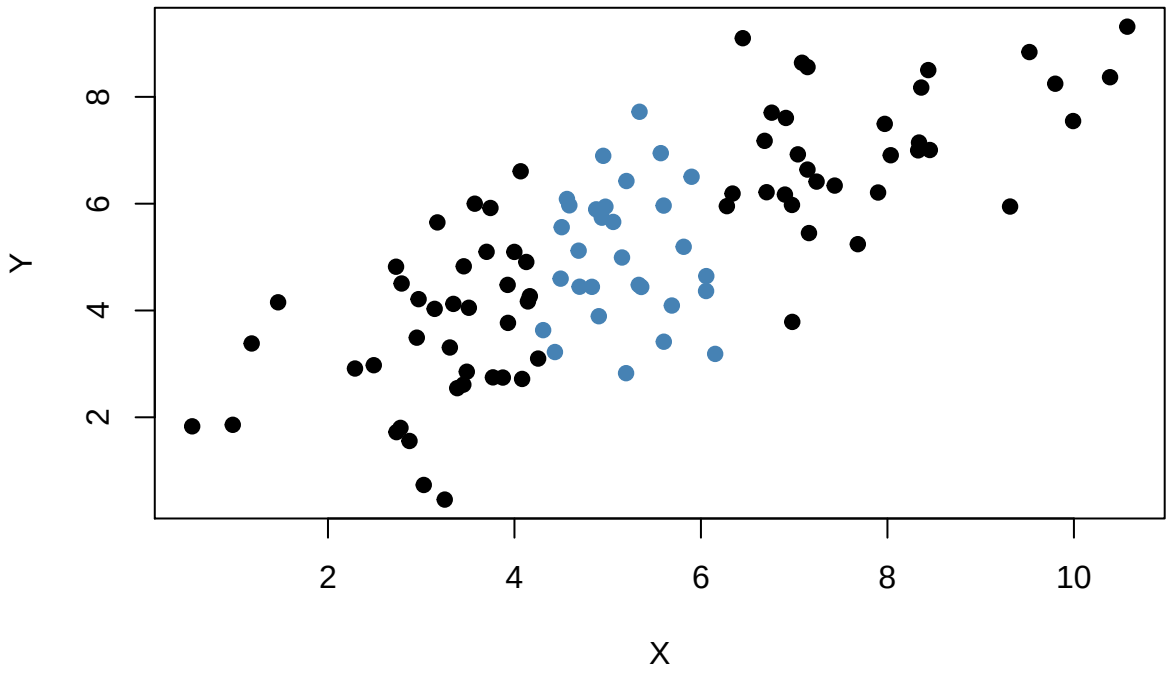
$$Cov(X, Y) = 4.$$

Formally, this is written down as

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

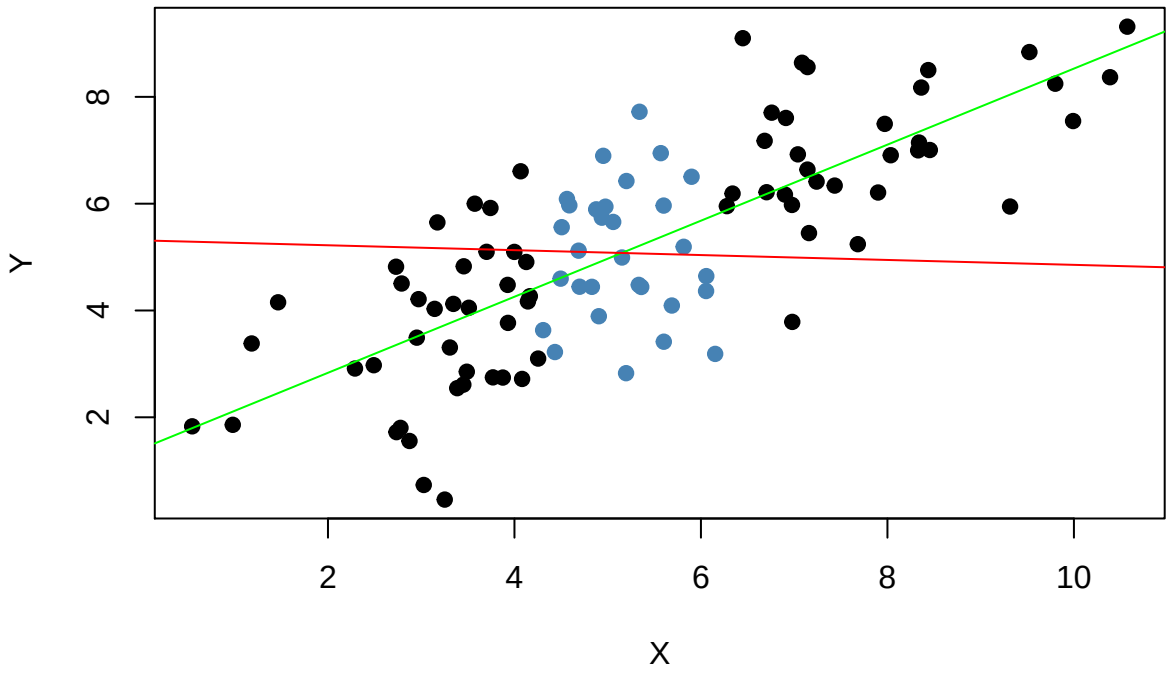
To carry out the random sampling, we make use of the function `mvrnorm()` from the package `MASS` which allows to draw random samples from multivariate normal distributions, see `?mvtnorm`. Next, we use `subset()` to split the sample into two subsets such that the first set, `set1`, consists of observations 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.

4.6 Exercises

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

Chapter 5

Hypothesis Tests and Confidence Intervals in the Simple Linear Regression Model

In this chapter, we continue our treatment of the simple linear regression model. The following subsections discuss how we may use our knowledge about the sampling distribution of the OLS estimator in order to make statements regarding its uncertainty.

These subsections cover the following topics:

- Testing Hypotheses regarding regression coefficients
- Confidence intervals for regression coefficients
- Regression when X is a dummy variable
- Heteroskedasticity and Homoskedasticity

The packages `AER` and `scales` are required for reproduction of the code chunks presented throughout this chapter.

5.1 Testing Two-Sided Hypotheses Concerning the Slope Coefficient

Using the fact that $\hat{\beta}_1$ is approximately normally distributed in large samples (see Key Concept 4.4), testing hypotheses about the true value β_1 can be done with the same approach as discussed in chapter 3.2.

Key Concept 5.1

General Form of the t-Statistic

Remember from chapter 3 that a general t-statistic has the form

$$t = \frac{\text{estimated value} - \text{hypothesized value}}{\text{standard error of the estimator}}.$$

Key Concept 5.2

Testing Hypotheses regarding β_1

For testing the hypothesis $H_0 : \beta_1 = \beta_{1,0}$, we need to perform the following steps:

1. Compute the standard error of $\hat{\beta}_1$, $SE(\hat{\beta}_1)$

$$SE(\hat{\beta}_1) = \sqrt{\hat{\sigma}^2_{\hat{\beta}_1}} \quad , \quad \hat{\sigma}^2_{\hat{\beta}_1} = \frac{1}{n} \times \frac{\frac{1}{n-2}\sum_{i=1}^{n}(X_i - \overline{X})^2 \hat{u}_i^2}{\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2\right]^2}.$$

2. Compute the t-statistic

$$t = \frac{\hat{\beta}_1 - \beta_{1,0}}{SE(\hat{\beta}_1)}.$$

3. Given a two sided alternative ($H_1 : \beta_1 \neq \beta_{1,0}$) we reject at the 5% level if $|t^{act}| > 1.96$ or, equivalently, if the p-value is less than 0.05.
   Recall the definition of the p-value:

$$\begin{aligned} p\text{-value} &= \Pr_{H_0}\left[\left|\frac{\hat{\beta}_1 - \beta_{1,0}}{SE(\hat{\beta}_1)}\right| > \left|\frac{\hat{\beta}_1^{act} - \beta_{1,0}}{SE(\hat{\beta}_1)}\right|\right] && (5.1)\\ &= \Pr_{H_0}(|t| > |t^{act}|) && (5.2)\\ &= 2 \cdot \Phi(-|t^{act}|) && (5.3) \end{aligned}$$

   The last equality holds due to the normal approximation for large samples.

Consider again the OLS regression stored in `linear_model` from Chapter 4 that gave us the regression line

$$\widehat{TestScore} = \underset{(9.47)}{698.9} - \underset{(0.49)}{2.28} \times STR \ , \ R^2 = 0.051 \ , \ SER = 18.6.$$

For testing a hypothesis concerning the slope parameter (the coefficient on STR), we need $SE(\hat{\beta}_1)$, the standard error of the respective point estimator. As common in the literature, standard errors are presented in parentheses below the point estimates.

Key Concept 5.1 states that it is rather cumbersome to compute the standard error and thereby the t-statistic by hand. The question you should be asking yourself right now is: can we obtain these values with minimum effort using `R`? Yes, we can. Let us first use `summary()` to get a summary on the estimated coefficients in `linear_model`.

```
# print the summary of the coefficients to the console
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 698.932949  9.4674911 73.824516 6.569846e-242
## STR          -2.279808  0.4798255 -4.751327  2.783308e-06
```

Looking at the second column of the coefficients' summary, we discover values for $SE(\hat{\beta}_0)$ and $SE(\hat{\beta}_1)$. Also, in the third column `t value`, we find t-statistics t^{act} suitable for tests of the separate hypotheses $H_0 : \beta_0 = 0$ and $H_0 : \beta_1 = 0$. Furthermore, the output provides us with p-values corresponding to both tests against the two-sided alternatives $H_1 : \beta_0 \neq 0$ respectively $H_1 : \beta_1 \neq 0$ in the fourth column of the table.

Let us have a closer look at the test of

$$H_0 : \beta_1 = 0 \quad vs. \quad H_1 : \beta_1 \neq 0.$$

Using our revisited knowledge about t-statistics we find that

$$t^{act} = \frac{-2.279808 - 0}{0.4798255} \approx -4.75.$$

What does this tell us about the significance of the estimated coefficient? We reject the null hypothesis at the 5% level of significance since $|t^{act}| > 1.96$ that is the observed test statistic falls into the region of rejection. Or, alternatively as leading to the same result, we have p-value $= 2.78 * 10^{-6} < 0.05$. We conclude that the coefficient is significantly different from zero. With other words, our analysis provides evidence that the class size *has an influence* on the students test scores. We say that β_1 is significantly different from 0 at the level of 5%.

Note that, although the difference is negligible in the present case as we will see later, `summary()` does not perform the normal approximation but calculates p-values using the t-distribution instead. Generally, the degrees of freedom of the assumed t-distribution are determined in the following manner:

$$\text{DF} = n - k - 1$$

where n is the number of observations used to estimate the model and k is the number of regressors, excluding the intercept. In our case, we have $n = 420$ observations and the only regressor is STR so $k = 1$. The simplest way to determine the model degrees of freedom is

```
# determine residual degrees of freedom
linear_model$df.residual
```

```
## [1] 418
```

Hence, for the assumed sampling distribution of $\hat{\beta}_1$ we have

$$\hat{\beta}_1 \sim t_{418}$$

such that the p-value for a two-sided significance test can be obtained by executing the following code:

```
2 * pt(-4.751327, df = 418)
```

```
## [1] 2.78331e-06
```

The result is very close to the value provided by `summary()`. However since n is sufficiently large one could just as well use the standard normal density to compute the p-value:

```
2 * pnorm(-4.751327)
```

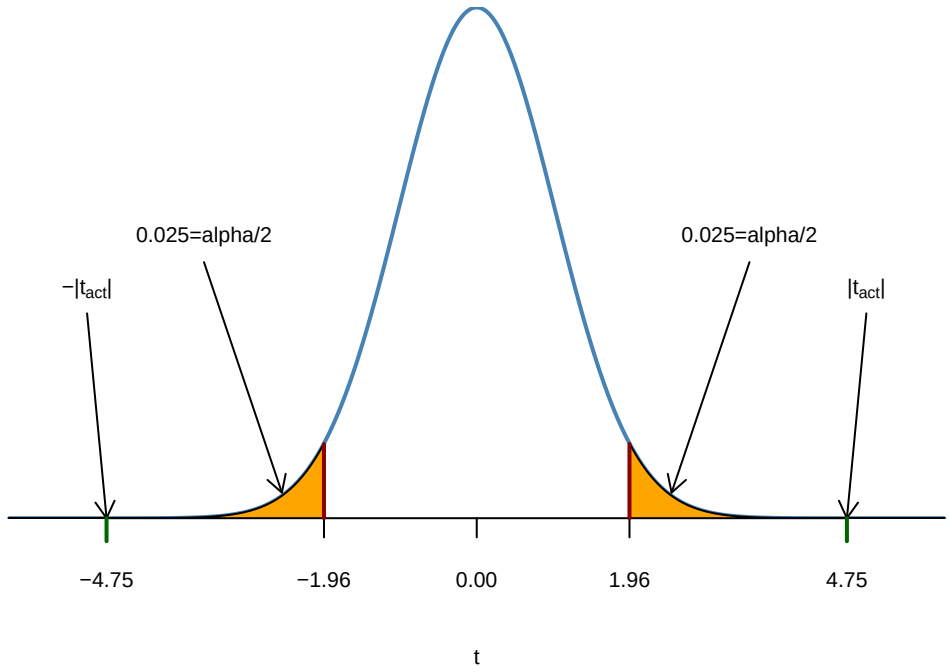
```
## [1] 2.02086e-06
```

The difference is indeed negligible. These findings tell us that, if $H_0 : \beta_1 = 0$ is true and we were to repeat the whole process of gathering observations and estimating the model, chances of observing a $\hat{\beta}_1 \geq |-4.75|$ are roughly $1 : 359285$ — so higher chances than winning the lottery next Saturday but still very unlikely!

Using `R` we may visualize how such a statement is made when using the normal approximation. This reflects the principles depicted in figure 5.1 in the book. Do not let the following code chunk deter you: the code is somewhat longer than the usual examples and looks unappealing but there is **a lot** of repetition since color shadings and annotations are added on both tails of the normal distribution. We recommend to execute the code step by step in order to see how the graph is augmented with the annotations.

```
# Plot the standard normal on the domain [-6,6]
t <- seq(-6, 6, 0.01)

plot(x = t,
     y = dnorm(t, 0, 1),
     type = "l",
```

```
     col = "steelblue",
     lwd = 2,
     yaxs = "i",
     axes = F,
     ylab = "",
     main = "Calculating the p-Value of a Two-Sided Test When t^act = -4.75",
     cex.lab = 0.7
     )

tact <- -4.75

axis(1, at = c(0, -1.96, 1.96, -tact, tact), cex.axis = 0.7)

# Shade the critical regions using polygon():

# critical region in left tail
polygon(x = c(-6, seq(-6, -1.96, 0.01), -1.96),
        y = c(0, dnorm(seq(-6, -1.96, 0.01)), 0),
        col = 'orange'
        )

# critical region in right tail

polygon(x = c(1.96, seq(1.96, 6, 0.01), 6),
        y = c(0, dnorm(seq(1.96, 6, 0.01)), 0),
        col = 'orange'
        )

# Add arrows and texts indicating critical regions and the p-value
arrows(-3.5, 0.2, -2.5, 0.02, length = 0.1)
arrows(3.5, 0.2, 2.5, 0.02, length = 0.1)

arrows(-5, 0.16, -4.75, 0, length = 0.1)
arrows(5, 0.16, 4.75, 0, length = 0.1)

text(-3.5, 0.22, labels = paste("0.025=",expression(alpha),"/2",sep = ""), cex = 0.7)
text(3.5, 0.22, labels = paste("0.025=",expression(alpha),"/2",sep = ""), cex = 0.7)

text(-5, 0.18, labels = expression(paste("-|",t[act],"|")), cex = 0.7)
text(5, 0.18, labels = expression(paste("|",t[act],"|")), cex = 0.7)

# Add ticks indicating critical values at the 0.05-level, t^act and -t^act
rug(c(-1.96, 1.96), ticksize  = 0.145, lwd = 2, col = "darkred")
rug(c(-tact, tact), ticksize  = -0.0451, lwd = 2, col = "darkgreen")
```



The p-Value is the area under the curve to left of -4.75 plus the area under the curve to the right of 4.75. As we already know from the calculations above, this value is very small.

5.2 Confidence Intervals for Regression Coefficients

As we already know, estimates of the regression coefficients β_0 and β_1 are afflicted with sampling uncertainty, see Chapter 4. Therefore, we will *never* estimate the exact true value of these parameters from sample data in an empirical application. However, we may construct confidence intervals for the intercept and the slope parameter.

A 95% confidence interval for β_i has two equivalent definitions:

- The interval is the set of values for which a hypothesis test to the level of 5% cannot be rejected.
- The interval has a probability of 95% to contain the true value of β_i. So in 95% of all samples that could be drawn, the confidence interval will cover the true value of β_i.

We also say that the interval has a confidence level of 95%. The idea of the confidence interval is summarized in Key Concept 5.3.

Key Concept 5.3

A Confidence Interval for β_i

Imagine you could draw all possible random samples of given size. The interval that contains the true value β_i in 95% of all samples is given by the expression

$$\text{KI}_{0.95}^{\beta_i} = \left[\hat{\beta}_i - 1.96 \times SE(\hat{\beta}_i)\,,\, \hat{\beta}_i + 1.96 \times SE(\hat{\beta}_i)\right].$$

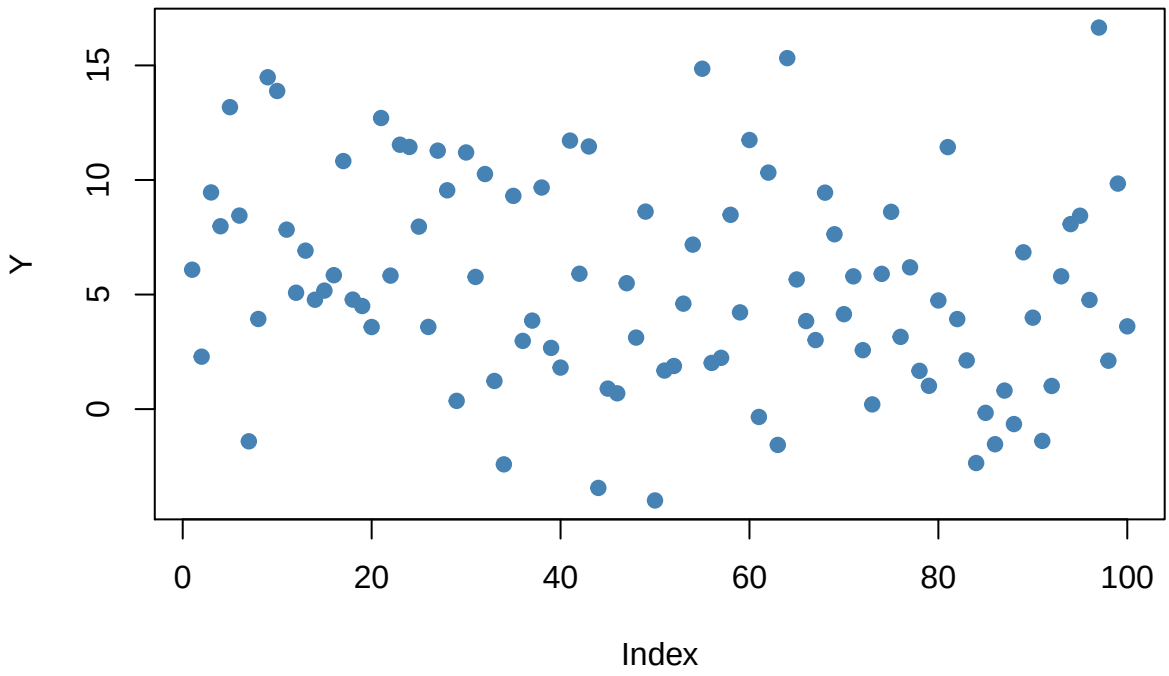
Equivalently, this interval can be seen as the set of null hypotheses for which a 5% two-sided hypothesis test does not reject.

Simulation Study: Confidence Intervals

To get a better understanding of confidence intervals we will conduct another simulation study. For now, assume that we are confronted with the following sample of $n = 100$ observations on a single variable Y where

$$Y_i \overset{i.i.d}{\sim} N(5, 25) \quad \forall \; i = 1, \ldots, 100.$$

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

# generate and plot the sample data
Y <- rnorm(n = 100,
           mean = 5,
           sd =5
           )

plot(Y,
     pch = 19,
     col = "steelblue"
     )
```



We assume that the data is generated by the model

$$Y_i = \mu + \epsilon_i$$

where μ is an unknown constant and we know that $\epsilon_i \overset{i.i.d.}{\sim} N(0, 25)$. In this model, the OLS estimator for μ is given by

$$\hat{\mu} = \overline{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i$$

(can you verify this?) i.e. the sample average of the Y_i. It further holds that

$$SE(\hat{\mu}) = \frac{\sigma_\epsilon}{\sqrt{n}} = \frac{5}{\sqrt{100}}$$

(see Chapter 3.3) A large sample 95% confidence interval for μ is then given by

$$KI^{\mu}_{0.95} = \left[\hat{\mu} - 1.96 \times \frac{5}{\sqrt{100}} \ , \ \hat{\mu} + 1.96 \times \frac{5}{\sqrt{100}}\right]. \tag{5.4}$$

It is fairly easy to compute this interval in R by hand. The following code chunk generates a named vector containing the interval bounds:

```
cbind(
  CIlower = mean(Y) - 1.96 * 5/10,
  CIupper = mean(Y) + 1.96 * 5/10
)
```

```
##       CIlower  CIupper
## [1,] 4.502625 6.462625
```

Knowing that $\mu = 5$ we see that our example covers the true value for the present sample. As opposed to real world examples, we can use R to get a better understanding of confidence intervals by repeatedly sampling data, estimating μ and computing the confidence interval for μ as in (5.4).

The procedure is as follows:

- We initialize the vectors `lower` and `upper` in which the simulated interval limits are to be saved. We want to simulate 10000 intervals so both vectors are set to have this length.
- We use a `for()` loop to sample 100 observations from the $N(5, 25)$ distribution and compute $\hat{\mu}$ as well as the boundaries of the confidence interval in every iteration of the loop.
- At last we join `lower` and `upper` in a matrix.

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

# initialize vectors of lower and upper interval boundaries
lower <- numeric(10000)
upper <- numeric(10000)

# loop sampling / estimation / CI
for(i in 1:10000) {
  Y <- rnorm(100, mean = 5, sd =5)
  lower[i] <- mean(Y) - 1.96 * 5/10
  upper[i] <- mean(Y) + 1.96 * 5/10
}

# join vectors of interval bounds in a matrix
CIs <- cbind(lower, upper)
```

According to Key Concept 5.3 we expect that the fraction of the 10000 simulated intervals saved in the matrix `CIs` that contain the true value $\mu = 5$ should be roughly 95%. We can easily check this using logical operators.

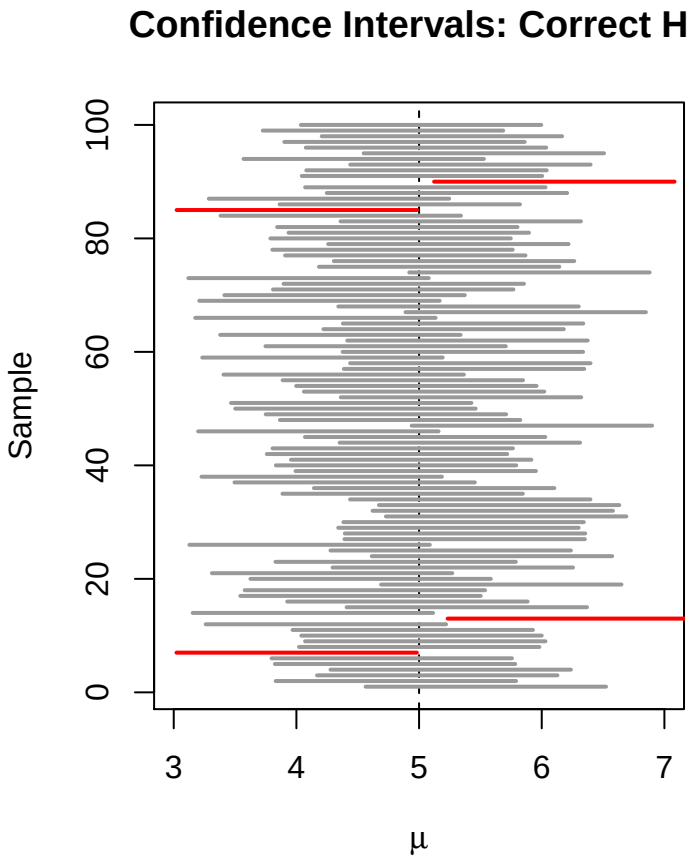
```
sum(CIs[, 1] <= 5 & 5 <= CIs[, 2])/10000
```

```
## [1] 0.9487
```

The simulation shows that the fraction of intervals covering $\mu = 5$, i.e. those intervals for which $H_0 : \mu = 5$ cannot be rejected is close to the theoretical value of 95%.

Let us draw a plot of the first 100 simulated confidence intervals and indicate those which *do not* cover the true value of μ. We do this by adding horizontal lines representing the confidence intervals on top of each other.

```
# identify intervals not covering mu
# (4 intervals out of 100)
ID <- which(!(CIs[1:100, 1] <= 5 & 5 <= CIs[1:100, 2]))

# initialize the plot
plot(0,
     xlim = c(3, 7),
     ylim = c(1, 100),
     ylab = "Sample",
     xlab = expression(mu),
     main = "Confidence Intervals: Correct H0")

# set up color vector
colors <- rep(gray(0.6), 100)
colors[ID] <- "red"

# draw reference line at mu=5
abline(v = 5, lty = 2)

# add horizontal bars representing the CIs
for(j in 1:100) {
  lines(c(CIs[j, 1], CIs[j, 2]),
        c(j, j),
        col = colors[j],
        lwd = 2)
}
```



We find that for the first 100 samples, the true null hypothesis is rejected in four cases so these intervals do not cover $\mu = 5$. We have indicated the intervals which lead to a rejection of the true null hypothesis by red color.

Let us now turn back to the example of test scores and class sizes. The regression model from Chapter 4 is stored in `linear_model`. An easy way to get 95% confidence intervals for β_0 and β_1, the coefficients on `(intercept)` and `STR`, is to use the function `confint()`. We only have to provide a fitted model object as an to this function. The confidence level is set to 95% by default but can be modified by setting the argument `level`, see `?confint`.

```
confint(linear_model)
```

```
##                 2.5 %     97.5 %
## (Intercept) 680.32312 717.542775
## STR          -3.22298  -1.336636
```

Let us check if the calculation is done as we expect it to be. For β_1, that is the coefficient on `STR`, according to the formula presented above the interval borders are computed as

$$-2.279808 \pm 1.96 \times 0.4798255 \;\Rightarrow\; \text{KI}_{0.95}^{\beta_1} = [-3.22, -1.34]$$

so this actually leads to the same interval. Obviously, this interval *does not* contain the value zero which, as we have already seen in the previous section, leads to the rejection of the null hypothesis $\beta_{1,0} = 0$.

5.3 Regression when X is a Binary Variable

Instead of using a continuous regressor X, we might be interested in running the regression

$$Y_i = \beta_0 + \beta_1 D_i + u_i \tag{5.2}$$

where D_i is a binary variable, a so-called *dummy variable*. For example, we may define D_i in the following way:

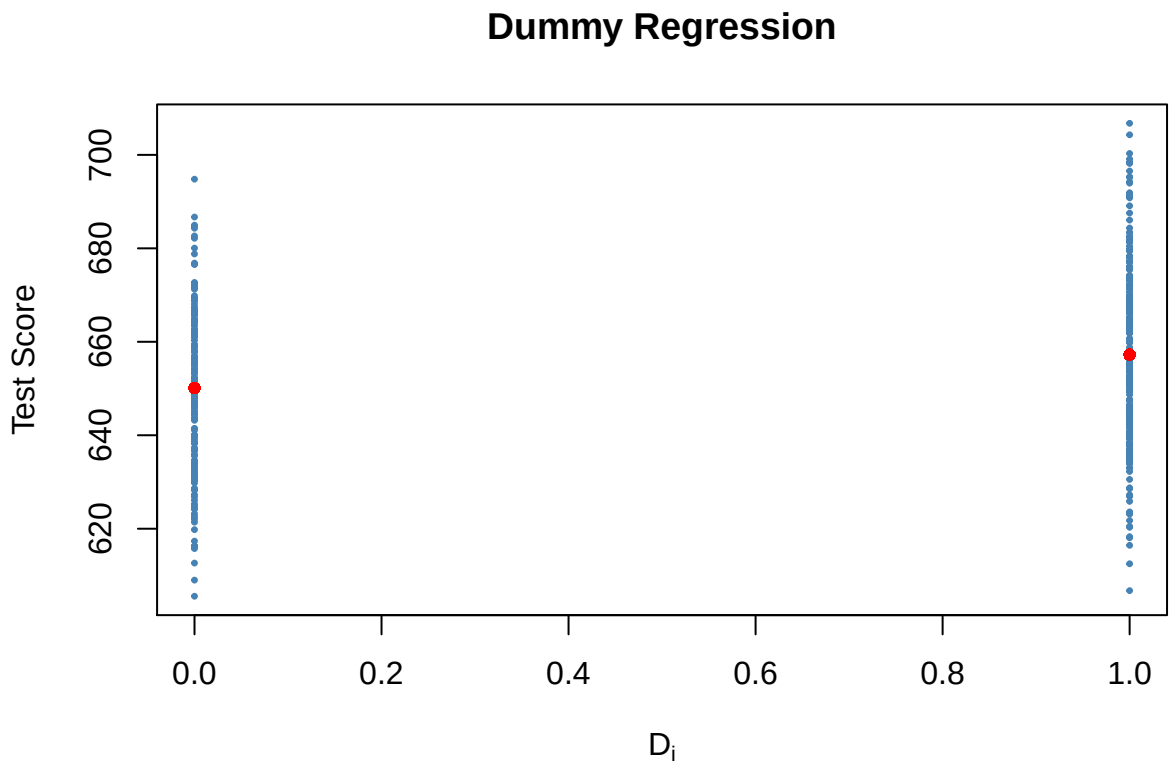
$$D_i = \begin{cases} 1 & \text{if } STR \text{ in } i^{th} \text{ school district} < 20 \\ 0 & \text{if } STR \text{ in } i^{th} \text{ school district} \geq 20 \end{cases} \tag{5.3}$$

The regression model now is

$$TestScore_i = \beta_0 + \beta_1 D_i + u_i. \tag{5.4}$$

Let us see how these data look like in a scatter plot:

```
# Create the dummy variable as defined above using a for loop
D <- CASchools$STR < 20
CASchools$D <- D

# Plot the data
plot(CASchools$D, CASchools$score,              # provide the data to be ploted
     pch = 20,                                  # use filled circles as plot symbols
     cex = 0.5,                                 # set size of plot symbols to 0.5
     col = "Steelblue",                         # set the symbols' color to "Steelblue"
     xlab = expression(D[i]),                   # Set title and axis names
     ylab = "Test Score",
     main = "Dummy Regression"
     )
```



We see that with D as the regressor, it is not useful to think of β_1 as a slope parameter since $D_i \in \{0, 1\}$, i.e. we only observe two discrete values instead of a continuum of regressor values lying in some range on the real line. Simply put, there is no continuous line depicting the conditional expectation function $E(TestScore_i|D_i)$ since this function is solely defined for X-positions 0 and 1.

Therefore, the interpretation of the coefficients in our regression model is as follows:

- $E(Y_i|D_i = 0) = \beta_0$ so β_0 is the expected test score in districts where $D_i = 0$ i.e. where STR is below 20.
- $E(Y_i|D_i = 1) = \beta_0 + \beta_1$ or, using the result above, $\beta_1 = E(Y_i|D_i = 1) - E(Y_i|D_i = 0)$. Thus, β_1 is *the difference in group specific expectations*, i.e. the difference in expected test score between districts with $STR < 20$ and those with $STR \geq 20$.

We will now use `R` to estimate the dummy regression model as defined by the equations (5.2) and (5.3) .

```
# estimate the dummy regression model
dummy_model <- lm(score ~ D, data = CASchools)
summary(dummy_model)
```

```
##
## Call:
## lm(formula = score ~ D, data = CASchools)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -50.496 -14.029  -0.346  12.884  49.504
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  650.077      1.393 466.666  < 2e-16 ***
## DTRUE          7.169      1.847   3.882  0.00012 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.74 on 418 degrees of freedom
## Multiple R-squared:  0.0348, Adjusted R-squared:  0.0325
## F-statistic: 15.07 on 1 and 418 DF,  p-value: 0.0001202
```

One can see that the expected test score in districts with $STR < 20$ ($D_i = 1$) is predicted to be $650.1+7.17 = 657.27$ while districts with $STR \geq 20$ ($D_i = 0$) are expected to have an average test score of only 650.1.

Group specific predictions can be added to the plot by execution of the following code chunk.

```
# add group specific predictions to the plot
points(x = CASchools$D,
       y = predict(dummy_model),
       col = "red",
       pch = 20
       )
```

Here we use the function `predict()` to obtain estimates of the group specific means. The red dots represent these sample group averages. Accordingly, $\hat{\beta}_1 = 7.17$ can be seen as the difference in group averages.

By inspection of the output generated with `summary(dummy_model)` we may also find an answer to the question whether there is a statistically significant difference in group means. This in turn would support the hypothesis that students perform differently when they are taught in small classes compared to those taught in large classes. We can assess this by a two-tailed test of the hypothesis $H_0 : \beta_1 = 0$. Conveniently the t-statistic and the corresponding p-value for this test are computed by `summary()`.

Since `t value` $= 3.88 > 1.96$ we reject the null hypothesis at the 5% level of significance. The same conclusion can be made when using the p-value which reports significance to the 0.00012% level.

As done with ')`linear_model')`, we may alternatively use the `confint()` function to compute a 95% confidence interval for the true difference in means and see if the hypothesized value is an element of this confidence set.

```
# confidence intervals for coefficients in the dummy regression model
confint(dummy_model)
```

```
##                  2.5 %    97.5 %
## (Intercept) 647.338594 652.81500
## DTRUE         3.539562  10.79931
```

We reject the hypothesis that there is no difference between group means at the 5% significance level since $\beta_{1,0} = 0$ lies outside of $[3.54, 10.8]$, the 95% confidence interval for the coefficient on D.

5.4 Heteroskedasticity and Homoskedasticity

All inference made in the previous chapters relies on the assumption that the error variance does not vary as regressor values change. But this will not necessarily be the case in empirical applications.

Key Concept 5.4

Heteroskedasticity and Homoskedasticity

- We say that the error term of our regression model is homoskedastic if the variance of the conditional distribution of u_i given X_i, $Var(u_i|X_i = x)$, is constant *for all* observations in our sample

$$\text{Var}(u_i|X_i = x) = \sigma^2 \ \forall \ i = 1, \dots, n.$$

- If instead there is dependence of the conditional variance of u_i on X_i, the error term is said to be heteroskedastic. We then write

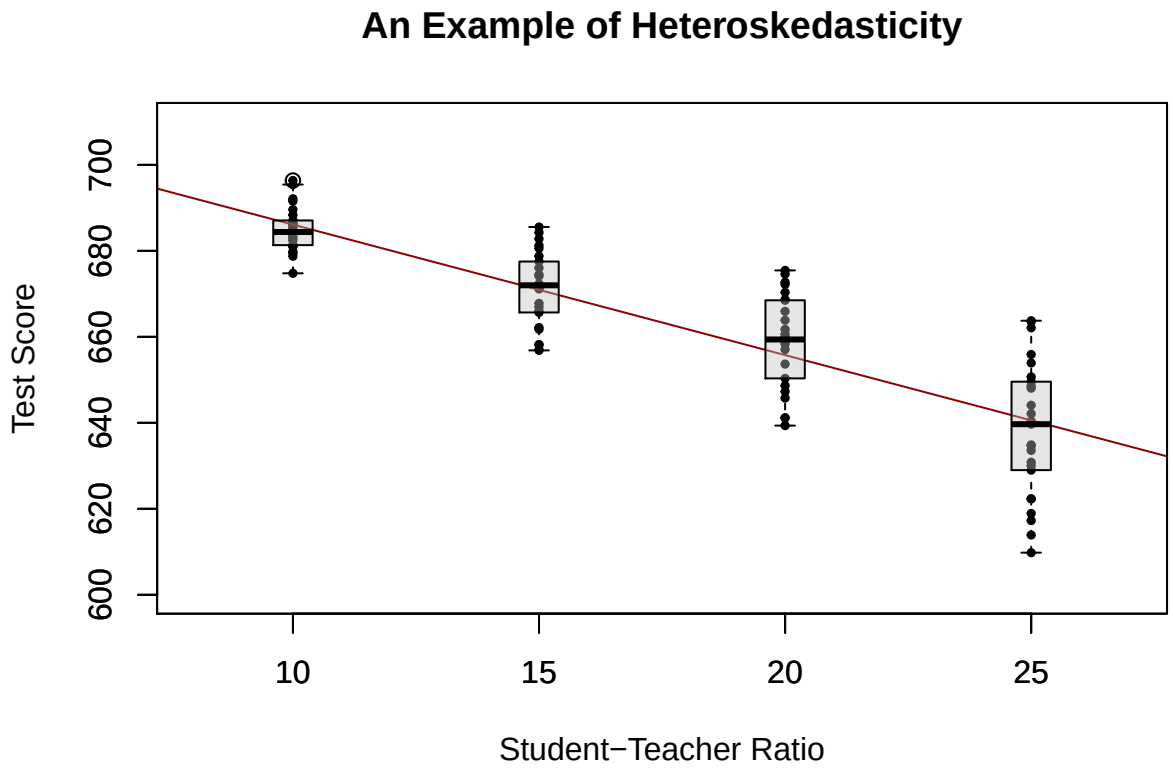
$$\text{Var}(u_i|X_i = x) = \sigma_i^2 \ \forall \ i = 1, \dots, n.$$

- Homoskedasticity is a *special case* of heteroskedasticity.

For a better understanding of heteroskedasticity, we generate some bivariate heteroskedastic data, estimate a linear regression model and then use box plots to depict the conditional distributions of the residuals.

```
# load scales package for adjusting color opacities
library(scales)

# Genrate some heteroskedastic data
set.seed(123)
x <- rep(c(10, 15, 20,25), each = 25)
e <- rnorm(100, sd = 12)
i <- order(runif(100, max = dnorm(e, sd = 12)))
y <- 720 - 3.3 * x + e[rev(i)]

# Estimate the model
mod <- lm(y ~ x)

# Plot the data
plot(x = x,
     y = y,
```

```
     main = "An Example of Heteroskedasticity",
     xlab = "Student-Teacher Ratio",
     ylab = "Test Score",
     cex = 0.5,
     pch = 19,
     xlim = c(8, 27),
     ylim = c(600, 710)
     )

# Add the regression line to the plot
abline(mod, col = "darkred")

# Add boxplots to the plot
boxplot(y ~ x,
        add = TRUE,
        at = c(10, 15, 20, 25),
        col = alpha("gray", 0.4),
        border = "black"
        )
```



For this artificial data it is clear to see that we face unequal conditional error variances. Specifically, we observe that the variance in test scores (and therefore the variance of the errors committed) *increases* with the student teacher ratio.

A Real-World Example for Heteroskedasticity

Think about the economic value of education: if there would not be an expected economic value-added to receiving education at university, you probably would not be reading this script right now. A starting point

to empirical verification of such a relation is to have data on graduates. More precisely, we need data on wages and education of graduates in order to estimate a model like

$$wage_i = \beta_0 + \beta_1 \cdot education_i + u_i.$$

What can be presumed about this relation? It is likely that, on average, higher educated workers earn more money than workers with less education, so we expect to estimate an upward sloping regression line. Also it seems plausible that better educated workers are more likely to meet the requirements for the well-paid jobs. However, workers with low education will have no shot at those well-paid jobs. Therefore it seems plausible that the distribution of earnings spreads out as education increases. In other words: we expect that there is heteroskedasticity.

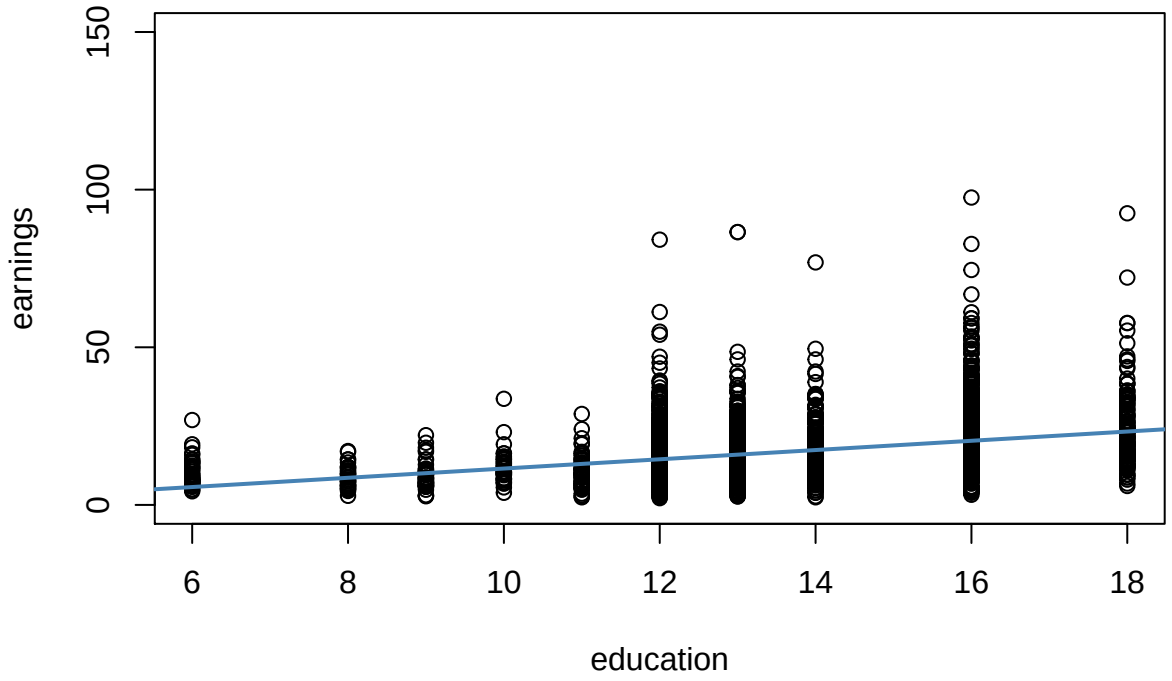
To verify this empirically we may use real data on hourly earnings and the number of years of education of employees. Such data can be found in `CPSSWEducation`. This data set is part of the package `AER` and stems from the Current Population Survey (CPS) which is conducted periodically by the Bureau of Labor Statistics in the United States.

The subsequent code chunks demonstrate how to import the data into `R` and how to produce a plot in the fashion of Figure 5.3 in the book.

```
# load package and attach data
library(AER)
data("CPSSWEducation")
attach(CPSSWEducation)

# get an overview
summary(CPSSWEducation)
```

```
##       age          gender        earnings        education
##  Min.   :29.0   female:1202   Min.   : 2.137   Min.   : 6.00
##  1st Qu.:29.0   male  :1748   1st Qu.:10.577   1st Qu.:12.00
##  Median :29.0                 Median :14.615   Median :13.00
##  Mean   :29.5                 Mean   :16.743   Mean   :13.55
##  3rd Qu.:30.0                 3rd Qu.:20.192   3rd Qu.:16.00
##  Max.   :30.0                 Max.   :97.500   Max.   :18.00
```

```
# estimate a simple regression model
labor_model <- lm(earnings ~ education)

# plot observations and add the regression line
plot(education,
     earnings,
     ylim = c(0, 150)
     )

abline(labor_model,
       col = "steelblue",
       lwd = 2
       )
```



From inspecting the plot we can tell that the mean of the distribution of earnings increases with the level of education. This is also suggested by formal analysis: the estimated regression model stored in `labor_mod` shows that there is a positive relation between years of education and earnings.

```
labor_model
```

```
##
## Call:
## lm(formula = earnings ~ education)
##
## Coefficients:
## (Intercept)    education
##      -3.134        1.467
```

The estimated regression equation states that, on average, an additional year of education increases a workers hourly earnings by about \$1.47. Once more we use `confint()` to obtain a 95% confidence interval for both regression coefficients.

```
confint(labor_model)
```

```
##                 2.5 %    97.5 %
## (Intercept) -5.015248 -1.253495
## education    1.330098  1.603753
```

Since the interval is $[1.33, 1.60]$ we can reject the hypothesis that the coefficient on `education` is zero at the 5% level.

Furthermore, the plot indicates that there is heteroskedasticity: if we assume the regression line to be a reasonably good representation of the conditional mean function $E(earnings_i|education_i)$, the dispersion of hourly earnings around that function clearly increases with the level of education, i.e. the variance of the distribution of earnings increases. In other words: the variance of the residuals (the errors made in explaining earnings by education) increases with education so that the regression errors are heteroskedastic. This example makes a case that the assumption of homoskedasticity is doubtful in economic applications.

Should We Care About Heteroskedasticity?

To answer the question whether we should worry about heteroskedasticity being present, let us see how the variance of $\hat{\beta}_1$ is computed under the assumption of homoskedasticity. In this case we have

$$\sigma^2_{\hat{\beta}_1} = \frac{\sigma^2_u}{n \cdot \sigma^2_X} \tag{5.5}$$

which is a simplified version of the general equation (4.1) presented in Key Concept 4.4. See Appendix 5.1 of the book for details on the derivation. `summary()` estimates (5.5) by

$$\overset{\sim}{\sigma}^2_{\hat{\beta}_1} = \frac{SER^2}{\sum_{i=1}^n (X_i - \overline{X})^2} \quad \text{where} \quad SER = \frac{1}{n-2} \sum_{i=1}^n \hat{u}_i^2.$$

Thus `summary()` estimates the *homoskedasticity-only* standard error

$$\sqrt{\overset{\sim}{\sigma}^2_{\hat{\beta}_1}} = \sqrt{\frac{SER^2}{\sum_{i=1}^n (X_i - \overline{X})^2}}.$$

This is in fact an estimator for the standard deviation of the estimator $\hat{\beta}_1$ that is *inconsistent* for the true value $\sigma^2_{\hat{\beta}_1}$ when there is heteroskedasticity. The implication is that t-statistics computed in the manner of Key Concept 5.1 do not have a standard normal distribution, even in large samples. This issue may invalidate inference drawn when using the previously treated tools for hypothesis testing: we should be cautious when making statements about the significance of regression coefficients on the basis of t-statistics as computed by `summary()` or confidence intervals produced by `confint()` if it is doubtful for the assumption of homoskedasticity to hold!

We will now use `R` to compute the homoskedasticity-only standard error estimate for $\hat{\beta}_1$ in the test score regression model `linear_model` by hand and see if it matches the value produced by `summary()`.

```
# Store model summary in 'model'
model <- summary(linear_model)

# Extract the standard error of the regression from model summary
SER <- model$sigma

# Compute the variation in 'size'
V <- (nrow(CASchools)-1) * var(CASchools$STR)

# Compute the standard error of the slope parameter's estimator and print it
SE.beta_1.hat <- sqrt(SER^2/V)
SE.beta_1.hat
```

```
## [1] 0.4798255
```

```
# Use logical operators to see if the value computed by hand matches the one provided
# in mod$coefficients. Round estimates to four decimal places
round(model$coefficients[2, 2], 4) == round(SE.beta_1.hat, 4)
```

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

Indeed, the estimated values are equal.

Computation of Heteroskedasticity-Robust Standard Errors

Consistent estimation of $\sigma_{\hat{\beta}_1}$ under heteroskedasticity is granted when the following *robust* estimator is used.

$$SE(\hat{\beta}_1) = \sqrt{\frac{\frac{1}{n-2}\sum_{i=1}^{n}(X_i - \overline{X})^2 \hat{u}_i^2}{\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2\right]^2}} \tag{5.6}$$

Standard error estimates computed this way are also referred to as Eicker-Huber-White standard errors. It can be quite cumbersome to do this calculation by hand. Luckily, there are `R` function for that purpose. A convenient one, named `vcovHC()` is part of the package ')`sandwich`'.[1] This function can compute a variety of standard error estimators. The one brought forward in (5.6) is computed when the argument `type` is set to `"HC0"`.

Let us now compute robust standard error estimates for the coefficients in `linear_model`.

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

# compute robust standard error estimates
vcov <- vcovHC(linear_model, type = "HC0")
vcov
```

```
##             (Intercept)        STR
## (Intercept)  106.908469 -5.3383689
## STR           -5.338369  0.2685841
```

The output of `vcovHC()` is the variance-covariance matrix of coefficient estimates. We are interested in the square root of the diagonal elements of this matrix since these values are the standard error estimates we seek.

> When we have k > 1 regressors, writing down the equations for a regression model becomes very messy. A more convinient way to denote and estimate so-called multiple regression models is by using matrix algebra. This is why functions like vcovHC() produce matrices. In the simple linear regression model, the variances and covariances of the coefficient estimators can be gathered in the variance-covariance matrix
>
> $$\text{Var}\begin{pmatrix}\hat{\beta}_0 \\ \hat{\beta}_1\end{pmatrix} = \begin{pmatrix}\text{Var}(\hat{\beta}_0) & \text{Cov}(\hat{\beta}_0, \hat{\beta}_1) \\ \text{Cov}(\hat{\beta}_0, \hat{\beta}_1) & \text{Var}(\hat{\beta}_1)\end{pmatrix} \tag{5.5}$$
>
> which is a symmetric matrix. So vcovHC() gives us $\widehat{\text{Var}}(\hat{\beta}_0)$, $\widehat{\text{Var}}(\hat{\beta}_1)$ and $\widehat{\text{Cov}}(\hat{\beta}_0, \hat{\beta}_1)$ but most of the time we are interested in the diagonal elements of the estimated matrix.

```
# compute the square root of the diagonal elements in vcov
robust_se <- sqrt(diag(vcov))
robust_se
```

```
## (Intercept)         STR
##   10.339655    0.518251
```

Now assume we want to generate a coefficient summary as provided by `summary()` but with *robust* standard error estimates for the coefficient estimators, robust t-statistics and corresponding p-values for the regression model ')`linear_model`'). This can be done using `coeftest()` from the package `lmtest`, see `?coeftest`. Further we specify in the argument `vcov.` that `vcov`, the Eicker-Huber-White estimate of the variance matrix we have computed before should be used.

[1] The package `sandwich` is a dependency of the package `AER` meaning that it is attached automatically if you load `AER`

```
# We invoke the function `coeftest()` on our model
coeftest(linear_model, vcov. = vcov)
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 698.93295   10.33966  67.597 < 2.2e-16 ***
## STR          -2.27981    0.51825  -4.399 1.382e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that values reported in the column `Std. Error` are equal to the ones received using `sqrt(diag(vcov))`.

How severe are the implications of using homoskedasticity-only standard errors in the presence of heteroskedasticity? The answer is: it depends. As mentioned above we may face the risk of drawing wrong conclusions when conducting significance tests. Let us illustrate this by generating another example of a heteroskedastic data set and use it to estimate a simple regression model. We take

$$Y_i = \beta_1 \cdot X_i + u_i \quad , \quad u_i \overset{i.i.d.}{\sim} N(0, 0.36 \cdot X_i^2)$$

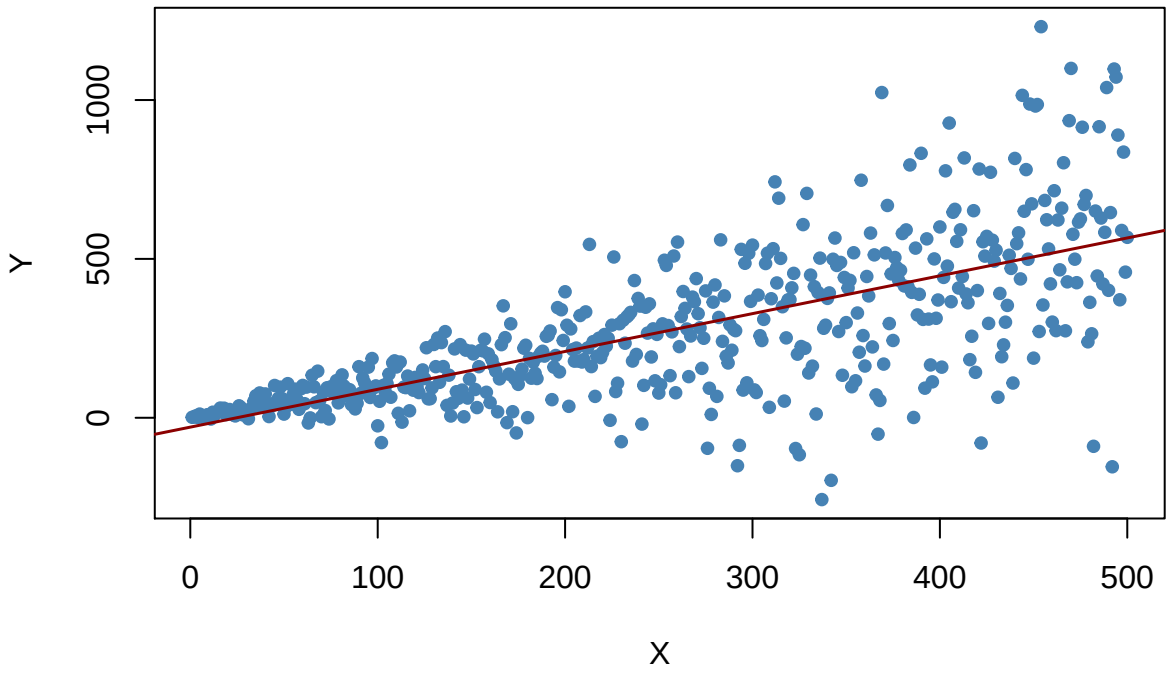
with $\beta_1 = 1$ as the data generating process. Clearly the assumption of homoskedasticity is violated here since the variance of the errors is a non-linear, increasing function of X_i but the errors have zero mean and are i.i.d. such that the assumptions made in Key Concept 4.3 are not violated. As before, the true conditional mean function we are interested in estimating is

$$E(Y_i|X_i) = \beta_1 X_i.$$

```
# set random seed
set.seed(21)

# generate heteroskedastic data
X <- 1:500
Y <- rnorm(n = 500, mean = X, sd = 0.6*X)

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

We plot the data and add the regression line.

```
# plot the data
plot(X, Y,
     pch = 19,
     col = "steelblue",
     cex = 0.8
     )

# add the regression line to the plot
abline(reg,
       col = "darkred",
       lwd = 1.5)
```



The plot shows that the data are heteroskedastic as the variance of Y grows with X. We continue by conducting a significance test of the (true) null hypothesis $H_0 : \beta_1 = 1$ twice, once using the homoskedasticity-only standard error formula and once with the robust version (5.6). An easy way to do this in `R` is the function `linearHypothesis()` from the package `car`, see `?linearHypothesis`. It allows to test linear hypotheses about parameters in linear models in a similar way as done with a t-statistic and offers various robust covariance matrix estimators. We test by comparing the tests' p-values to the significance level of 5%.

> linearHypothesis() computes a test statistic that follows an F distribution under the null hypothesis. We will not loose too much words on the theory behind it at this time. In general, the idea of the F test is to compare the fit of different models. When testing a hypothesis about a *single* coefficient using an F test, one can show that the test statistic is simply the square of the corresponding t-statistic:
>
> $$F = t^2 = \frac{\hat{\beta}_i - \beta_{i,0}}{SE(\hat{\beta}_i)} \sim F_{1,n-k-1}$$
>
> In linearHypothesis(), the hypothesis must be provided as a *string*. The function returns an object of class anova which contains further information on the test that can be accessed using the $ operator.

```
# test hypthesis using default standard error
linearHypothesis(reg, hypothesis.matrix = "X = 1")$'Pr(>F)'[2] < 0.05
```

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

```
# test hypothesis using robust standard error
linearHypothesis(reg, hypothesis.matrix = "X = 1", white.adjust = "hc0")$'Pr(>F)'[2] < 0.05
```

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

This is a good example of what can go wrong if we ignore heteroskedasticity: for the data set at hand the default method rejects the null hypothesis $\beta_1 = 1$ although it is true. When using the robust standard error formula the test does not reject the null. Of course we could argue that this is just a coincidence and both tests are equally well in maintaining the type I error rate of 5%. This can be further investigated by

computing *Monte Carlo* estimates of the rejection frequencies of both tests on the basis of a large number of random samples. We proceed as follows:

- initialize vectors `t` and `t.rob`.
- Using a `for()` loop, we generate 10000 heteroskedastic random samples of size 1000, estimate the regression model and check whether the tests falsly reject the null at the level of 5% using comparison operators. The results are stored in the respective vectors `t` and `t.rob`.
- After the simulation, we compute the fraction of false rejections for both tests.

```
# initialize vectors t and t.rob
t <- c()
t.rob <- c()

# loop sampling and estimation
for (i in 1:10000) {

  # sample data
  X <- 1:1000
  Y <- rnorm(n = 1000, mean = X, sd = 0.6*X)

  # estimate regression model
  reg <- lm(Y ~ X)

  # homoskedasdicity-only significance test
  t[i] <- linearHypothesis(reg, "X = 1")$'Pr(>F)'[2] < 0.05

  # robust significance test
  t.rob[i] <- linearHypothesis(reg, "X = 1", white.adjust = "hc0")$'Pr(>F)'[2] < 0.05

}

# compute the fraction of false rejections
cbind(t = sum(t), t.rob = sum(t.rob)) / 10000
```

```
##           t  t.rob
## [1,] 0.0781 0.0516
```

The results show that we face an increased risk of falsely rejecting the null using the homoskedasticity-only standard error for the testing problem at hand: with the common standard error estimator, 7.62% of all tests reject the null hypothesis falsely. In contrast, with the robust test statistic we are very close to the nominal level of 5%.

5.5 The Gauss-Markov Theorem

When estimating regression models, we know that the results of the estimation procedure are outcomes of a random process. However, when using unbiased estimators, at least on average, we estimate the true parameter. When comparing different unbiased estimators, it is therefore interesting to know which one has the highest precision: being aware that the likelihood of estimating the *exact* value of the parameter of interest is 0 in an empirical application, we want to make sure that the likelihood of obtaining an estimate very close to the true value is as high as possible. This means we want to use the estimator with the lowest variance of all unbiased estimators. The Gauss-Markov theorem states that, in the class of conditionally unbiased linear estimators, the OLS estimator has this property under certain conditions.

Key Concept 5.5

The Gauss-Markov Theorem for $\hat{\beta}_1$

Suppose that the assumptions made in Key Concept 4.3 hold *and* that the errors are *homoskedastic*. The OLS estimator is the best (in the sense of smallest variance) linear conditionally unbiased estimator (BLUE) in this setting.

Let us have a closer look at what this means:

- Estimators of β_1 that are linear functions of the $Y_1, \dots, Y_n$ and that are unbiased conditionally on the regressor $X_1, \dots, X_n$ can be written as
$$\widetilde{\beta}_1 = \sum_{i=1}^{n} a_i Y_i$$
where the a_i are weights that are allowed to depend on the X_i but *not* on the Y_i.
- We already know that $\widetilde{\beta}_1$ has a sampling distribution: $\widetilde{\beta}_1$ is a linear function of the Y_i which are random variables. If now
$$E(\widetilde{\beta}_1 | X_1, \dots, X_n) = \beta_1$$
we say that $\widetilde{\beta}_1$ is a linear unbiased estimator of β_1, conditionally on the $X_1, \dots, X_n$.
- We may ask if $\widetilde{\beta}_1$ is also the *best* estimator in this class, i.e. the most efficient one of all linear conditionally unbiased estimators where "most efficient" means smallest variance. The weights a_i play an important role here and it turns out that OLS uses just the right weights to have the BLUE property.

Simulation Study: BLUE Estimator

Consider the case of a regression of $Y_i, \dots, Y_n$ only on a constant. Here, the Y_i are assumed to be a random sample from a population with mean μ and variance σ^2. We know that the OLS estimator in this model is simply the sample mean:

$$\hat{\beta}_1 = \overline{\beta}_1 = \sum_{i=1}^{n} \underbrace{\frac{1}{n}}_{=a_i} Y_i \tag{5.6}$$

Clearly, each observation is weighted by

$$a_i = \frac{1}{n}.$$

and we also know that $\text{Var}(\hat{\beta}_1) = \text{Var}(\hat{\beta}_1) = \frac{\sigma^2}{n}$.

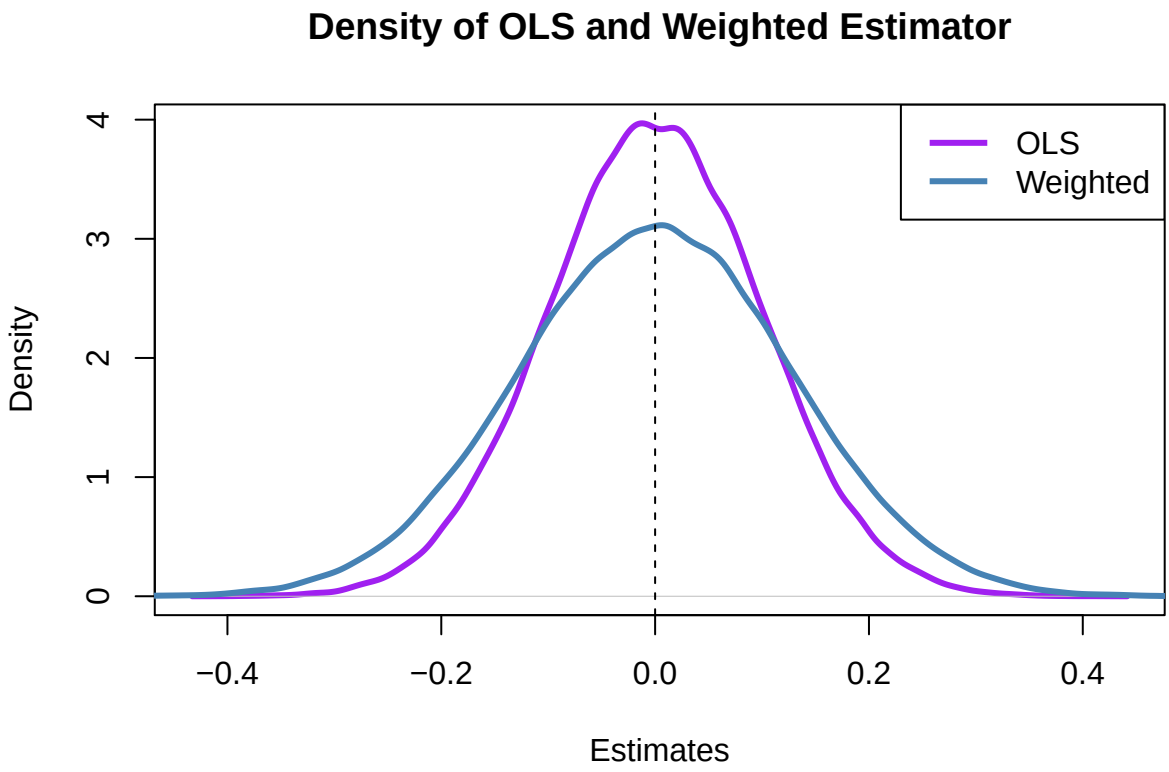
We will now use `R` to conduct a simulation study that demonstrates what happens to the variance of (5.6) if different weights

$$w_i = \frac{1 \pm \epsilon}{n}$$

are assigned to either half of the sample $Y_1, \dots, Y_n$ instead of using $\frac{1}{n}$, the weights implied by OLS.

```
# Set sample size and number of repitititions
n <- 100
reps <- 1e5

# Choose epsilon and create a vector of weights as defined above
epsilon <- 0.8
w <- c(rep((1+epsilon)/n, n/2),
```

```
       rep((1-epsilon)/n, n/2)
      )

# Draw a random sample y_1,...,y_n from the standard normal distribution,
# use both estimators 1e5 times and store the result in the vectors 'ols' and
# 'weightedestimator'

ols <- rep(NA, reps)
weightedestimator <- rep(NA, reps)

for (i in 1:reps) {
  y <- rnorm(n)
  ols[i] <- mean(y)
  weightedestimator[i] <- crossprod(w, y)
}

# Plot kernel density estimates of the estimators' distributions

# OLS
plot(density(ols),
     col = "purple",
     lwd = 3,
     main = "Density of OLS and Weighted Estimator",
     xlab = "Estimates"
     )

# Weighted
lines(density(weightedestimator),
      col = "steelblue",
      lwd = 3
      )

# Add a dashed line at 0 and add a legend to the plot
abline(v = 0, lty = 2)
legend('topright',
       c("OLS", "Weighted"),
       col = c("purple", "steelblue"),
       lwd = 3
       )
```



What conclusion can we draw from the result?

- Both estimators seem to be unbiased: the means of their estimated distributions are zero.
- The estimator using weights that deviate from those implied by OLS is less efficient than the OLS estimator: there is higher dispersion when weights are $w_i = \frac{1 \pm 0.8}{100}$ instead of $w_i = \frac{1}{100}$ as required by the OLS solution.

Hence, our simulation results confirm what is stated by the Gauss-Markov Theorem.

5.6 Using the t-Statistic in Regression When the Sample Size Is Small

The three OLS assumptions discussed in Chapter 4 (see Key Concept 4.3) are the foundation for the results on the large sample distribution of the OLS estimators in the simple regression model. What can be said about the distribution of the estimators and their t-statistics when the sample size is small and the population distribution of the data is unknown? Provided that the three least squares assumptions hold and the errors are normally distributed and homoskedastic (we refer to these conditions as the homoskedastic normal regression assumptions), we have normally distributed estimators and t-distributed test statistics in small samples.

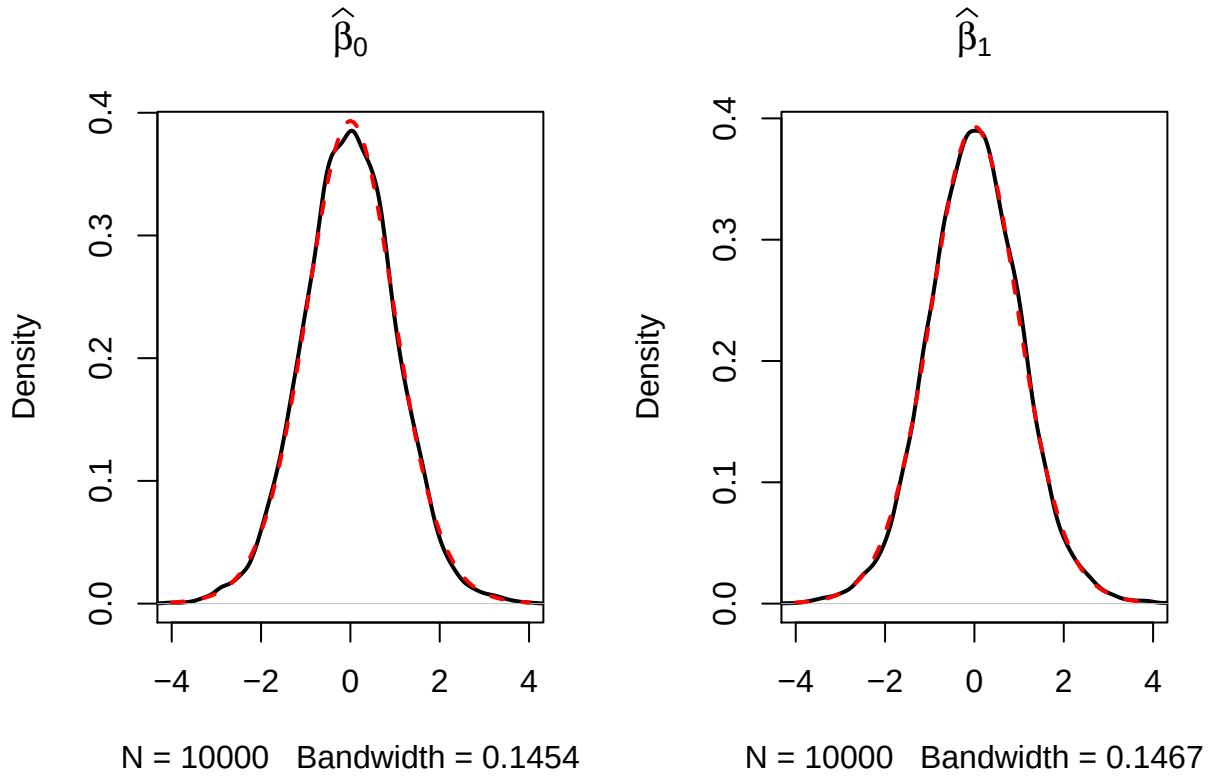
Recall the definition on a t-distributed variable

$$\frac{Z}{\sqrt{W/m}} \sim t_m$$

where Z is a standard normal random variable, W is χ^2 distributed with m degrees of freedom and Z and W are independent. See section 5.6 in the book for a more detailed discussion of the small sample distribution of t-statistics in regression.

Let us simulate the distribution of regression t-statistics based on a large number of small random samples when the sample size is small, say $n = 20$, and compare their simulated distributions to their theoretical distribution which should be t_{18}, the t-distribution with 18 degrees of freedom (recall that DF $= n - k - 1$).

```
# initialize vectors
beta_0 <- c()
beta_1 <- c()

# loop sampling / estimation / t statistics
for (i in 1:10000) {

  X <- runif(20, 0, 20)
  Y <- rnorm(n = 20, mean = X)
  reg <- summary(lm(Y ~ X))
  beta_0[i] <- (reg$coefficients[1, 1] - 0)/(reg$coefficients[1, 2])
  beta_1[i] <- (reg$coefficients[2, 1] - 1)/(reg$coefficients[2, 2])

}

# plot the distributions and compare with t_18 density function:

# divide plotting area
par(mfrow = c(1, 2))

# plot the simulated density of beta_0
plot(density(beta_0),
     lwd = 2 ,
     main = expression(widehat(beta)[0]),
     xlim = c(-4, 4)
    )

# add the t_18 density to the plot
curve(dt(x, df = 18),
      add = T,
      col = "red",
      lwd = 2,
      lty = 2
      )

# plot the simulated density of beta_1
plot(density(beta_1),
     lwd = 2,
     main = expression(widehat(beta)[1]), xlim = c(-4 ,4)
     )

# add the t_18 density to the plot
curve(dt(x, df = 18),
      add = T,
      col = "red",
      lwd = 2,
      lty = 2
      )
```



The outcomes are consistent with our expectations: the empirical distributions of both estimators seem to track the theoretical t_{18} distribution quite closely.

5.7 Exercises

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

Chapter 6

Regression with Panel Data

Regression using panel data may mititgate omitted variable bias when there is no information on variables available that correlate with both the regressors of interest and the independent variable and if these variables are constant in the time dimension or across entities. Provided that panel data is availabe, panel regression methods may improve upon multiple regression models which, as discussed in Chapter 9, produce results that are not internally valid in such a setting.

This chapter covers the following topics:

- Notation for panel data
- Fixed effects regression using time and/or entity fixed effects
- Computation of standard errors in fixed effects regression models

Following the book, for applications we make use of the data set `Fatalities` from the `AER` package which is a panel data set reporting annually state level observations on U.S. traffic fatalities for the period 1982 through 1988. The applications are concered with the question if there are effects of alcohol taxes and drunk driving laws on road fatalities and, if present, *how strong* these effects are.

For this purpose we will introduce a convenient `R` function that enables us to estimate linear panel regression models, the function `plm()` which comes with the package `plm`. Usage of `plm()` is very similar as for the `lm()` function which we have used throughout the previous chapters for estimation of simple and multiple regression models.

6.1 Panel Data

Key Concept 10.1

Notation for Panel Data

In contrast to cross-section data where we have observations on n subjects (entities), panel data has observations on n entities at $T \geq 2$ time periods. This is denoted

$$(X_{it}, Y_{it}),\ i = 1, \dots, n \quad \text{and} \quad t = 1, \dots, T$$

where the index i refers to the entity beeing observed while t refers to the time period.

Sometimes panel data is also called longitudinal data as it adds a temporal dimension to cross-sectional data. Let us have a glimpse at the data set `Fatalities` by checking its structure and listing of the first few observations.

```
# load package and data
library(AER)
data(Fatalities)

# obtain dimension and inspect structure
is.data.frame(Fatalities)
```

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

```
dim(Fatalities)
```

```
## [1] 336  34
```

```
str(Fatalities)
```

```
## 'data.frame':    336 obs. of  34 variables:
##  $ state       : Factor w/ 48 levels "al","az","ar",..: 1 1 1 1 1 1 1 2 2 2 ...
##  $ year        : Factor w/ 7 levels "1982","1983",..: 1 2 3 4 5 6 7 1 2 3 ...
##  $ spirits     : num  1.37 1.36 1.32 1.28 1.23 ...
##  $ unemp       : num  14.4 13.7 11.1 8.9 9.8 ...
##  $ income      : num  10544 10733 11109 11333 11662 ...
##  $ emppop      : num  50.7 52.1 54.2 55.3 56.5 ...
##  $ beertax     : num  1.54 1.79 1.71 1.65 1.61 ...
##  $ baptist     : num  30.4 30.3 30.3 30.3 30.3 ...
##  $ mormon      : num  0.328 0.343 0.359 0.376 0.393 ...
##  $ drinkage    : num  19 19 19 19.7 21 ...
##  $ dry         : num  25 23 24 23.6 23.5 ...
##  $ youngdrivers: num  0.212 0.211 0.211 0.211 0.213 ...
##  $ miles       : num  7234 7836 8263 8727 8953 ...
##  $ breath      : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ jail        : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 2 2 ...
##  $ service     : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 2 2 ...
##  $ fatal       : int  839 930 932 882 1081 1110 1023 724 675 869 ...
##  $ nfatal      : int  146 154 165 146 172 181 139 131 112 149 ...
##  $ sfatal      : int  99 98 94 98 119 114 89 76 60 81 ...
##  $ fatal1517   : int  53 71 49 66 82 94 66 40 40 51 ...
##  $ nfatal1517  : int  9 8 7 9 10 11 8 7 7 8 ...
##  $ fatal1820   : int  99 108 103 100 120 127 105 81 83 118 ...
##  $ nfatal1820  : int  34 26 25 23 23 31 24 16 19 34 ...
##  $ fatal2124   : int  120 124 118 114 119 138 123 96 80 123 ...
##  $ nfatal2124  : int  32 35 34 45 29 30 25 36 17 33 ...
##  $ afatal      : num  309 342 305 277 361 ...
##  $ pop         : num  3942002 3960008 3988992 4021008 4049994 ...
##  $ pop1517     : num  209000 202000 197000 195000 204000 ...
##  $ pop1820     : num  221553 219125 216724 214349 212000 ...
##  $ pop2124     : num  290000 290000 288000 284000 263000 ...
##  $ milestot    : num  28516 31032 32961 35091 36259 ...
##  $ unempus     : num  9.7 9.6 7.5 7.2 7 ...
##  $ emppopus    : num  57.8 57.9 59.5 60.1 60.7 ...
##  $ gsp         : num  -0.0221 0.0466 0.0628 0.0275 0.0321 ...
```

```
# list first few observations
head(Fatalities)
```

```
##   state year spirits unemp   income   emppop  beertax baptist  mormon
## 1    al 1982    1.37  14.4 10544.15 50.69204 1.539379 30.3557 0.32829
```

```
## 2    al 1983    1.36  13.7 10732.80 52.14703 1.788991 30.3336 0.34341
## 3    al 1984    1.32  11.1 11108.79 54.16809 1.714286 30.3115 0.35924
## 4    al 1985    1.28   8.9 11332.63 55.27114 1.652542 30.2895 0.37579
## 5    al 1986    1.23   9.8 11661.51 56.51450 1.609907 30.2674 0.39311
## 6    al 1987    1.18   7.8 11944.00 57.50988 1.560000 30.2453 0.41123
##   drinkage     dry youngdrivers    miles breath jail service fatal nfatal
## 1    19.00 25.0063     0.211572 7233.887     no   no      no   839    146
## 2    19.00 22.9942     0.210768 7836.348     no   no      no   930    154
## 3    19.00 24.0426     0.211484 8262.990     no   no      no   932    165
## 4    19.67 23.6339     0.211140 8726.917     no   no      no   882    146
## 5    21.00 23.4647     0.213400 8952.854     no   no      no  1081    172
## 6    21.00 23.7924     0.215527 9166.302     no   no      no  1110    181
##   sfatal fatal1517 nfatal1517 fatal1820 nfatal1820 fatal2124 nfatal2124
## 1     99        53          9        99         34       120         32
## 2     98        71          8       108         26       124         35
## 3     94        49          7       103         25       118         34
## 4     98        66          9       100         23       114         45
## 5    119        82         10       120         23       119         29
## 6    114        94         11       127         31       138         30
##    afatal     pop  pop1517  pop1820  pop2124 milestot unempus emppopus
## 1 309.438 3942002 208999.6 221553.4 290000.1    28516     9.7     57.8
## 2 341.834 3960008 202000.1 219125.5 290000.2    31032     9.6     57.9
## 3 304.872 3988992 197000.0 216724.1 288000.2    32961     7.5     59.5
## 4 276.742 4021008 194999.7 214349.0 284000.3    35091     7.2     60.1
## 5 360.716 4049994 203999.9 212000.0 263000.3    36259     7.0     60.7
## 6 368.421 4082999 204999.8 208998.5 258999.8    37426     6.2     61.5
##           gsp
## 1 -0.02212476
## 2  0.04655825
## 3  0.06279784
## 4  0.02748997
## 5  0.03214295
## 6  0.04897637
```

```
# summarize variables 'state' and 'year'
summary(Fatalities[, c(1, 2)])
```

```
##      state       year
##  al     :  7   1982:48
##  az     :  7   1983:48
##  ar     :  7   1984:48
##  ca     :  7   1985:48
##  co     :  7   1986:48
##  ct     :  7   1987:48
##  (Other):294   1988:48
```

We find that the data set consists of 336 observations on 34 variables. Notice that the variable `state` is a factor variable with 48 levels (one for each of the 48 contigous US states). The varaible `year` is also a factor variable that has 7 levels identifying the time period when the observation was made which gives us $7 \times 48 = 336$ observations in total. Since all variables are observed for all entities and over all time periods we say the the panel is **balanced**. If there were missing data for at least one entities in at least one time period we would call the panel **unbalanced**.

Example: Traffic Deaths and Alcohol Taxes

Coming to the question of how the alcohol taxes and traffic fatalities are related, we start by reproducing Figure 10.1 of the book. For this we estimate simple regressions using data for years 1982 and 1988 that model the relationship between beer tax (adjusted for 1988 dollars) and the traffic fatility rate. Beforehand we define the latter as the number of fatalitites per 10000 inhabitants and choose subsets of the observations made for years 1982 and 1988. Afterwards, we plot both subsets and add the corresponding estimated regression functions.

```
# define fatality rate
Fatalities$fatal_rate <- Fatalities$fatal / Fatalities$pop * 10000

# subset data
Fatalities1982 <- subset(Fatalities, year == "1982")
Fatalities1988 <- subset(Fatalities, year == "1988")
```

```
# estimate simple regression models using 1982 and 1988 data
library(lmtest)
library(sandwich)

fatal1982_mod <- lm(fatal_rate ~ beertax, data = Fatalities1982)
fatal1988_mod <- lm(fatal_rate ~ beertax, data = Fatalities1988)

coeftest(fatal1982_mod, vcov. = vcovHC(fatal1982_mod, type = "HC1"))
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.01038    0.14957 13.4408   <2e-16 ***
## beertax      0.14846    0.13261  1.1196   0.2687
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coeftest(fatal1988_mod, vcov. = vcovHC(fatal1988_mod, type = "HC1"))
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  1.85907    0.11461 16.2205 < 2.2e-16 ***
## beertax      0.43875    0.12786  3.4314  0.001279 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

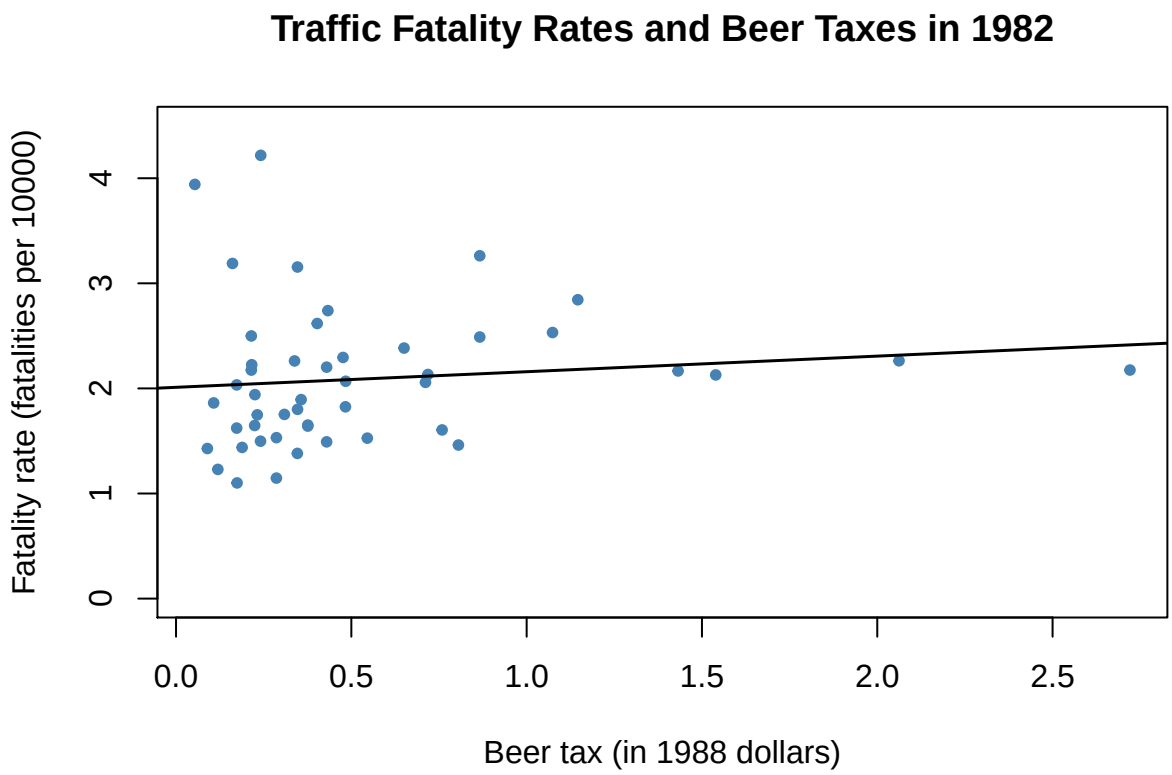
The estimated regression functions are:

$$\widehat{FatalityRate} = \underset{(0.15)}{2.01} + \underset{(0.13)}{0.15} \times BeerTax \quad \text{(1982 data)},$$

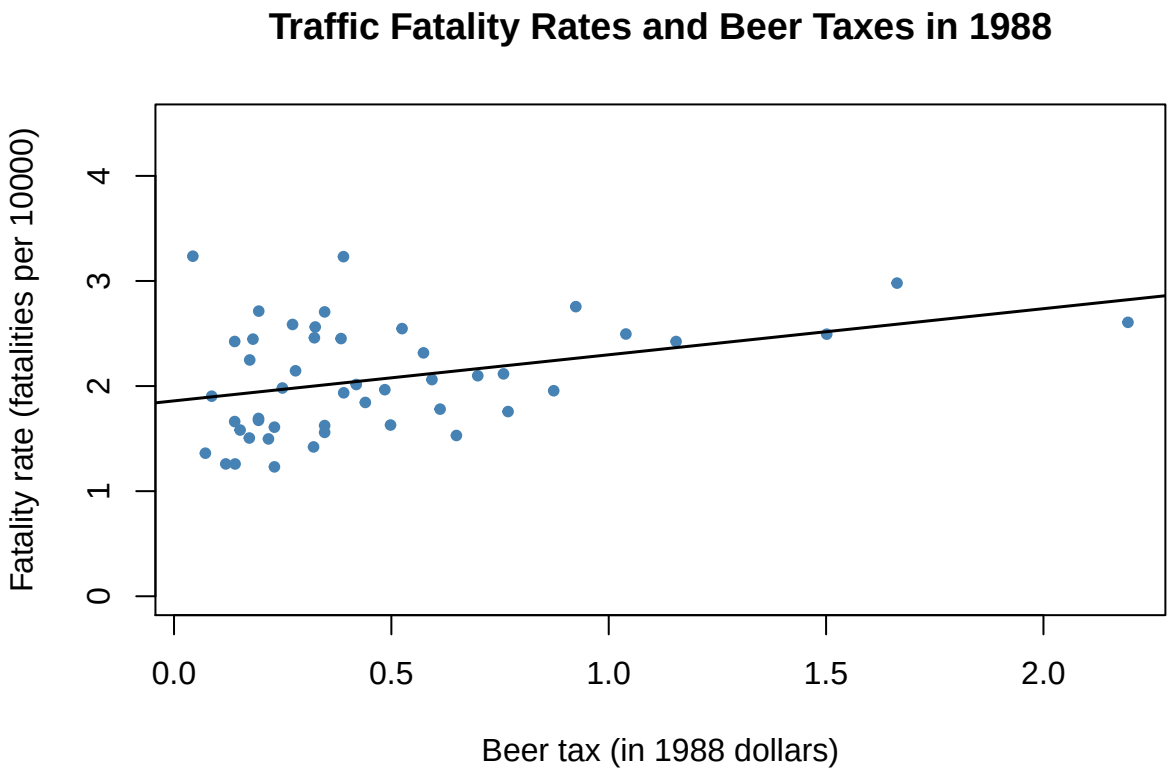
$$\widehat{FatalityRate} = \underset{(0.11)}{1.86} + \underset{(0.13)}{0.44} \times BeerTax \quad \text{(1988 data)}$$

```
# plot observations of interest and add estimated regression line for 1982 data
plot(x = Fatalities1982$beertax,
     y = Fatalities1982$fatal_rate,
```

```
     xlab = "Beer tax (in 1988 dollars)",
     ylab = "Fatality rate (fatalities per 10000)",
     main = "Traffic Fatality Rates and Beer Taxes in 1982",
     ylim = c(0, 4.5),
     pch = 20,
     col = "steelblue")

abline(fatal1982_mod, lwd = 1.5)
```




```
# plot observations of interest and add estimated regression line for 1988 data
plot(x = Fatalities1988$beertax,
     y = Fatalities1988$fatal_rate,
     xlab = "Beer tax (in 1988 dollars)",
     ylab = "Fatality rate (fatalities per 10000)",
     main = "Traffic Fatality Rates and Beer Taxes in 1988",
     ylim = c(0, 4.5),
     pch = 20,
     col = "steelblue")

abline(fatal1988_mod, lwd = 1.5)
```



In both plots, each point represents observations of beer tax and fatality rate for a given state in the respective year. The regression results indicate a postive relationship between the beer tax and the fatality rate for both years, wherby the estimated coefficient on beer tax for the 1988 data is almost three times as large as for the 1988 data set. This is contrary to our expectations: alcohol taxes are supposed to *lower* the rate of traffic fatalities. As known from Chapter 6, this is possibly due to omitted variable bias, since both models do not include any covariates, e.g. economic conditions. This could be corrected for using a multiple regression approach. However, both models cannot account for omitted *unobservable* factors that differ from state to state but can be assumed to be constant over the observation span, e.g. the populations attitude towas drunk driving. As shown in the next section, panel data allow us to hold such factors constant.

6.2 Panel Data with Two Time Periods: "Before and After" Comparisons

Suppose there are only $T = 2$ time periods $t = 1982, 1988$. This allows us to analyze differences in changes of the the fatality rate from year 1982 to 1988. We start by considering the population regression model

$$FatalityRate_{it} = \beta_0 + \beta_1 BeerTax_{it} + \beta_2 Z_i + u_{it}$$

where the Z_i are state specific characteristics that differ between states but are *constant over time*. For $t = 1982$ and $t = 1988$ we have

$$\begin{aligned} FatalityRate_{i1982} &= \beta_0 + \beta_1 BeerTax_{i1982} + \beta_2 Z_i + u_{i1982}, \\ FatalityRate_{i1988} &= \beta_0 + \beta_1 BeerTax_{i1988} + \beta_2 Z_i + u_{i1988}. \end{aligned}$$

We can eliminate the Z_i by regressing the difference in the fatality rate between 1988 and 1982 on the difference in beer tax between those years:

$$FatalityRate_{i1988} - FatalityRate_{i1982} = \beta_1 (BeerTax_{i1988} - BeerTax_{i1982}) + u_{i1988} - u_{i1982}$$

Using this regression model we can obtain an estimate for β_1 without worrying about a possible bias due to omission of the Z_i since these influences are eliminated from the model. Next we use use `R` to estimate a regression based on the differenced data and plot the estimated regression function.

```
# differences
diff_fatal_rate <- Fatalities1988$fatal_rate - Fatalities1982$fatal_rate
diff_beertax <- Fatalities1988$beertax - Fatalities1982$beertax

# estimate regression on differenced data
fatal_diff_mod <- lm(diff_fatal_rate ~ diff_beertax)

coeftest(fatal_diff_mod, vcov = vcovHC(fatal_diff_mod, type = "HC1"))
```

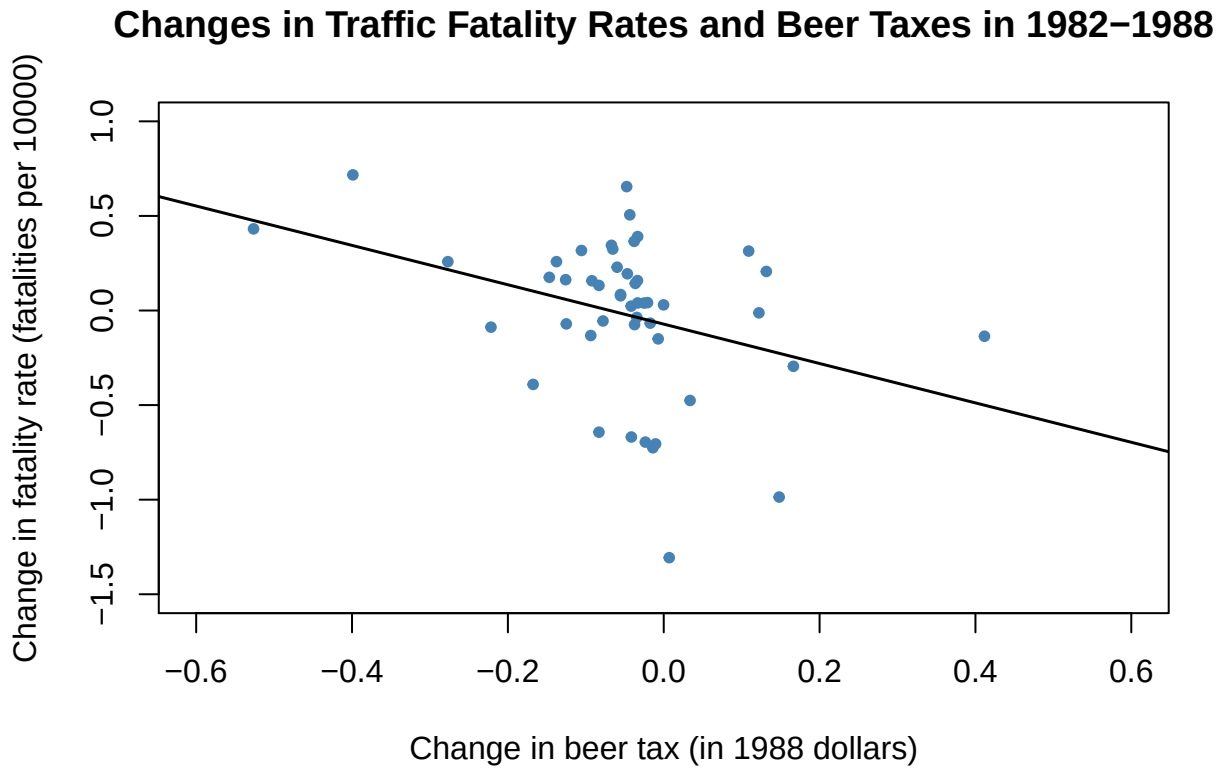
```
##
## t test of coefficients:
##
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.072037   0.065355 -1.1022 0.276091
## diff_beertax -1.040973   0.355006 -2.9323 0.005229 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that including the intercept allows for a change in the mean fatality rate in the time between 1982 and 1988 in the absence of a change in the beer tax.

We obtain the OLS estimated regression function

$$\widehat{FatalityRate_{i1988} - FatalityRate_{i1982}} = \underset{(0.065)}{-0.072} \underset{(0.36)}{-1.04} \times (BeerTax_{i1988} - BeerTax_{i1982}).$$

```
# plot differenced data
plot(x = diff_beertax,
     y = diff_fatal_rate,
     xlab = "Change in beer tax (in 1988 dollars)",
     ylab = "Change in fatality rate (fatalities per 10000)",
     main = "Changes in Traffic Fatality Rates and Beer Taxes in 1982-1988",
     xlim = c(-0.6, 0.6),
     ylim = c(-1.5, 1),
     pch = 20,
     col = "steelblue")

# add regression line to plot
abline(fatal_diff_mod, lwd = 1.5)
```



We observe that the estimated coefficient on beer tax is now negative and significantly different from zero at the level of 5%. Its interpretation is that raising the beer tax by \$1 causes trafic fatalities to decrease by 1.04 per 10000 people. This is rather large as the average fatality rate is approximately 2 persons per 10000 people.

```
# compute mean fatality rate over all states for all time periods
mean(Fatalities$fatal_rate)
```

```
## [1] 2.040444
```

Once more this outcome is likely to be a consequence of omitting factors that influence the fatality rate and are correlated with the beer tax *and* change over time. The message is that we need to be more careful and control for such factors before drawing conclusion about the effect of a raise in beer taxes.

Furthermore, note that the approach presented in this section discards information for years 1983 to 1987. A method that allows to use data for more than $T = 2$ time periods and allows us to add control variables is the fixed effects regression approach.

6.3 Fixed Effects Regression

Consider the panel regression model

$$Y_{it} = \beta_0 + \beta_1 X_{it} + \beta_2 Z_i + u_{it}$$

where the Z_i are unobserved time-invariant heterogeneities across the entities $i = 1, \dots, n$. We are interested in estimating β_1, the effect on Y_i of a change in X_i holding constant Z_i. Letting $\alpha_i = \beta_0 + \beta_2 Z_i$ we obtain the model

$$Y_{it} = \alpha_i + \beta_1 X_{it} + u_{it}. \tag{6.1}$$

Having invidual specific intercepts α_i, $i = 1, \ldots, n$, where each of these can be understood as the fixed effect of entity i, this model is called the *fixed effects regression model*. The variation in the α_i, $i = 1, \ldots, n$ comes from the Z_i. (6.1) can be rewritten as a regression model containing $n-1$ dummy regressors and a constant:

$$Y_{it} = \beta_i + \beta_1 X_{it} + \gamma_2 D2_i + \gamma_3 D3_i + \cdots + \gamma_n Dn_i + u_{it}. \tag{6.2}$$

Model (6.2) has n different intercepts — one for every entity. Both (6.1) and (6.2) are equivalnt representations of the fixed effects regression model.

The fixed effects regression model can be generalized to contain more than just one determinant of Y that is correlated with X and change over time. Key Concept 10.2 presents the generalized fixed effects regression model.

Key Concept 10.2

The Fixed Effects Regression Model

The fixed effects regression model is

$$Y_{it} = \beta_1 X_{1,it} + \cdots + \beta_k X_{k,it} + \alpha_i + u_{it} \tag{6.3}$$

with $i = 1, \ldots, n$ and $t = 1, \ldots, T$. The α_i are entity-specific intercepts that capture heterogeneities across entities. An equivalent representation of this model is given by

$$Y_{it} = \beta_0 + \beta_1 X_{1,it} + \cdots + \beta_k X_{k,it} + \gamma_2 D2_i + \gamma_3 D3_i + \cdots + \gamma_n Dn_i + u_{it} \tag{6.4}$$

where the $D2_i, D3_i, \ldots, Dn_i$ are dummy variables.

Estimation and Inference

Software packages use a so-called "entity-demeaned" OLS algorithm that is computationally more efficient than estimating regression models with $k + n$ regressors as needed for models (6.3) and (6.4).

Taking averages on both sides of (6.1) we obtain

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n} Y_{it} &= \beta_1 \frac{1}{n}\sum_{i=1}^{n} X_{it} + \frac{1}{n}\sum_{i=1}^{n} a_i + \frac{1}{n}\sum_{i=1}^{n} u_{it} \\ \overline{Y} &= \beta_1 \overline{X}_i + \alpha_i + \overline{u}_i. \end{aligned}$$

Substraction from (6.1) gives

$$Y_{it} - \overline{Y}_i = \beta_1 (X_{it} - \overline{X}_i) + (u_{it} - \overline{u}_i) \tag{6.5}$$

$$\widetilde{Y}_{it} = \beta_1 \widetilde{X}_{it} + \widetilde{u}_{it}. \tag{6.6}$$

In this model, the OLS estimate of the parameter of interest β_1 is equal to the estimate obtained using (6.2) — without the need to estimate $n-1$ dummies and an intercept.

We conclude that there are two ways of estimating β_1 in the fixed effects regression:

1. OLS regression of the dummy regression model (6.2)
2. OLS regression using the entity demeaned data (6.6)

Provided the fixed effects regression assumptions stated in Key Concept 10.3 hold, the sampling distribution of the OLS estimator in the fixed effects regression model has a normal distribution in large samples. The variance of the estimates can be estimated and we can compute standard errors, t-statistics and confidence intervals for model coefficients. In the next section, we will see how to estimate a fixed effects model using `R` and how to obtain a model summary that makes use of heteroskedasticity robust standard errors. Thereby we leave aside complicated formulas of the estimators. See Chapter 10.5 and Appendix 10.2 of the book for a discussion of the theoretical aspects.

Application to Traffic Deaths

Following Key Concept 10.2, the simple fixed effects model for estimation of the relation between traffic fatality rates and the beer taxes is

$$\widehat{FatalityRate}_{it} = \beta_1 BeerTax_{it} + StateFixedEffects + u_{it}, \tag{6.7}$$

the regression of the traffic fatalaty rate on beer tax and 48 binary regressor — one for each state.

We can simply use the function `lm()` to obtain an estimate of β_1.

```
fatal_fe_lm_mod <- lm(fatal_rate ~ beertax + state - 1, data = Fatalities)
fatal_fe_lm_mod
```

```
##
## Call:
## lm(formula = fatal_rate ~ beertax + state - 1, data = Fatalities)
##
## Coefficients:
## beertax  stateal  stateaz  statear  stateca  stateco  statect  statede
## -0.6559   3.4776   2.9099   2.8227   1.9682   1.9933   1.6154   2.1700
## statefl  statega  stateid  stateil  statein  stateia  stateks  stateky
##  3.2095   4.0022   2.8086   1.5160   2.0161   1.9337   2.2544   2.2601
## statela  stateme  statemd  statema  statemi  statemn  statems  statemo
##  2.6305   2.3697   1.7712   1.3679   1.9931   1.5804   3.4486   2.1814
## statemt  statene  statenv  statenh  statenj  statenm  stateny  statenc
##  3.1172   1.9555   2.8769   2.2232   1.3719   3.9040   1.2910   3.1872
## statend  stateoh  stateok  stateor  statepa  stateri  statesc  statesd
##  1.8542   1.8032   2.9326   2.3096   1.7102   1.2126   4.0348   2.4739
## statetn  statetx  stateut  statevt  stateva  statewa  statewv  statewi
##  2.6020   2.5602   2.3137   2.5116   2.1874   1.8181   2.5809   1.7184
## statewy
##  3.2491
```

As discussed in the previous section, it is also possible to estimate β_1 by applying OLS to the demeaned data,

$$\widetilde{FatalityRate} = \beta_1 \widetilde{BeerTax}_{it} + u_{it}.$$

```
# demeaned data
Fatalities_demeaned <- with(Fatalities,
            data.frame(fatal_rate = fatal_rate - ave(fatal_rate, state),
            beertaxs = beertax - ave(beertax, state)))

# estimate regression
summary(lm(fatal_rate ~ beertax - 1, data = Fatalities_demeaned))
```

Equivalently it is possible to use `plm()` from the package with the same name.

```
# install and load the 'plm' package
## install.packages("plm")
library(plm)
```

As for `lm()` we have to specify the regression formula and the data to be used in our call of `plm()`. Additionally, it is required to pass a vector of names of entity and time id variables to the `index` argument. For `Fatalities`, the id variable for entities is named `state` and the time id variable is `year`. Since the fixed effects estimator is also called the *within* estimator, we set `model = "within"`. Finally, the `coeftest()` function allows to obtain significance based on robust standard errors.

```
# estimate the fixed effects regression with plm()
fatal_fe_mod <- plm(fatal_rate ~ beertax,
                    data = Fatalities,
                    index = c("state", "year"),
                    model = "within")

# summary using robust standard errors
coeftest(fatal_fe_mod, vcov. = vcovHC(fatal_fe_mod, type = "HC1"))
```

```
##
## t test of coefficients:
##
##         Estimate Std. Error t value Pr(>|t|)
## beertax -0.65587    0.28880  -2.271  0.02388 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Agian, we find that the estimated coefficient is -0.6559. Notice that `plm()` uses the entity-demeaned OLS algorithm and does not report dummy coefficients. The estimated regression function is

$$\widehat{FatalityRate} = \underset{(0.29)}{-0.66} \times BeerTax + StateFixedEffects. \tag{6.8}$$

We conclude that the coefficient on $BeerTax$ is nagative and highly significant. The interpretation is that the estimated reduction in traffic fatalities due to a \$1 increase in the real beer tax is 0.66 per 10000 people which is still pretty high. Though including state fixed effects eliminates the risk of a bias due to omitted factors that vary across states but not over time, we suspect that there are other omitted varibales that vary over time and thus cause a bias.

6.4 Regression with Time Fixed Effects

Controlling for variables that are constant across entities but vary over time can be done by including time fixed effects. If there are *only* time fixed effects, the fixed effects regression model becomes

$$Y_{it} = \beta_0 + \beta_1 X_{it} + \delta_2 B2_t + \cdots + \delta_T BT_t + u_{it},$$

where only $T - 1$ dummies are included ($B1$ is omitted) since the model includes an intercept. This model eliminates omitted variables bias caused by exluding unobserved varaibles that evolve over time but are constant across entities only.

In some applications it is meaningful to include entity and time fixed effects. The **entity and time fixed effects** regression model is

$$Y_{it} = \beta_0 + \beta_1 X_{it} + \gamma_2 D2_i + \cdots + \gamma_n DT_i + \delta_2 B2_t + \cdots + \delta_T BT_t + u_{it}.$$

The combined model allows to eliminate bias from unobservables that change over time but are constant over entities and it controls for factors that differ across entities but are constant over time. Such models can be estimated using the OLS algorithm that is implemented in `R`.

The following code chunk shows how to estimate the combined entity and time fixed effects model of the relation between fatalities and beer tax,

$$FatalityRate_{it} = \beta_1 BeerTax_{it} + StateEffects + TimeFixedEffects + u_{it}$$

using both, `lm()` and `plm()`. It is straightforward to estimate this regression with `lm()` since it is just an extension of (6.7) so we only have to adjust the `formula` argument by adding the additional regressor `year` for time fixed effects. In our call of `plm()` we set another argument `effect = "twoways"` for inclusion of entity *and* time dummies.

```
# Estimate combined time and entity fixed effects regression model

# via lm()
fatal_tefe_lm_mod <- lm(fatal_rate ~ beertax + state + year - 1, data = Fatalities)
fatal_tefe_lm_mod
```

```
##
## Call:
## lm(formula = fatal_rate ~ beertax + state + year - 1, data = Fatalities)
##
## Coefficients:
##  beertax   stateal   stateaz   statear   stateca   stateco   statect
## -0.63998   3.51137   2.96451   2.87284   2.02618   2.04984   1.67125
##  statede   statefl   stategа   stateid   stateil   statein   stateia
##  2.22711   3.25132   4.02300   2.86242   1.57287   2.07123   1.98709
##  stateks   stateky   statela   stateme   statemd   statema   statemi
##  2.30707   2.31659   2.67772   2.41713   1.82731   1.42335   2.04488
##  statemn   statems   statemo   statemt   statene   statenv   statenh
##  1.63488   3.49146   2.23598   3.17160   2.00846   2.93322   2.27245
##  statenj   statenm   stateny   statenc   statend   stateoh   stateok
##  1.43016   3.95748   1.34849   3.22630   1.90762   1.85664   2.97776
##  stateor   statepa   stateri   statesc   statesd   statetn   statetx
##  2.36597   1.76563   1.26964   4.06496   2.52317   2.65670   2.61282
##  stateut   statevt   stateva   statewa   statewv   statewi   statewy
##  2.36165   2.56100   2.23618   1.87424   2.63364   1.77545   3.30791
## year1983  year1984  year1985  year1986  year1987  year1988
## -0.07990  -0.07242  -0.12398  -0.03786  -0.05090  -0.05180
```

```
# via plm()
fatal_tefe_mod <- plm(fatal_rate ~ beertax,
                      data = Fatalities,
                      index = c("state", "year"),
                      model = "within",
                      effect = "twoways")

coeftest(fatal_tefe_mod, vcov = vcovHC(fatal_tefe_mod, type = "HC1"))
```

```
##
## t test of coefficients:
##
##         Estimate Std. Error t value Pr(>|t|)
## beertax -0.63998    0.35015 -1.8277  0.06865 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that since we exclude the intercept, `lm()` does estimate coefficients for $(n-1)+(T-1) = 47+6 = 53$ binary variables! Again, `plm()` does only report the estimated coefficient on $BeerTax$.

The estimated regression function is

$$\widehat{FatalityRate} = \underset{(0.35)}{-0.64} \times BeerTax + StateEffects + TimeFixedEffects. \tag{6.9}$$

The value of -0.66, the result is close to the estimated coefficient for the regression model including only entity fixed effects. Unsurprisingly, the coefficient is less precisely estimated as before but significantly different from zero at the level of 10%.

In view of (6.8) and (6.9) we conclude that the estimated relationship between traffic fatalities and the real beer tax is not affected by omitted variable bias due to factors that are constant over time.

6.5 The Fixed Effects Regression Assumptions and Standard Errors for Fixed Effects Regression

This section focusses on the entity fixed effects regression model and presents model assumptions that need to hold in order for OLS to produce unbiased estimates that are normally distributed for large n. These assumptions are an extension of the assumptions made for the multiple regression model (see Key Concept 6.4) and are given in Key Concept 10.3. We will also briefly discuss standard errors in fixed effects regression models which differ from standard errors in multiple regression as the regression error can exhibit serial correlation in panel models.

Key Concept 10.3

The Fixed Effects Regression Assumptions

In the fixed effects regression model

$$Y_{it} = \beta_1 X_{it} + \alpha_i + u_{it} \quad , \quad i = 1, \dots, n, \; t = 1, \dots, T,$$

we assume the following:

1. The error term u_{it} has conditional mean zero, that is $E(u_{it}|u_{i1}, u_{i2}, \dots, u_{iT})$.
2. $(X_{i1}, X_{i2}, \dots, X_{i3}, u_{i1}, \dots, u_{iT})$, $i = 1, \dots, n$ are i.i.d. draws from their joint distribution.

3. Large outliers are unlikely i.e. (X_{it}, u_{it}) have nonzero finite fourth moments.
4. There is no perfect multicollinearity.

When there are multiple regressors, X_{it} is replaced by $X_{1,it}, X_{2,it}, \dots, X_{k,it}$.

The first assumption is that the error is uncorrelated with *all* observations of the variable X for the entity i over time. If this assumption is violated, we face omitted variables bias. The second assumption ensures that variables are i.i.d. *across* entities $i = 1, \dots, n$. Notice that this does not require the observations to be uncorrelated *within* an entity. We say that the X_{it} are allowed to be **autocorrelated** or **serially correlated** within entities. This is a common property of time series data. The same is allowed for errors u_{it}. Consult Chapter 10.5 of the book for a detailed explanation for why autocorrelation is a plausible characteristic in panel applications. The second assumption is granted if the entities are selected by simple random sampling. The third and fourth assumptions are analagous to the multiple regression assumptions made in Key Concept 6.4.

Standard Errors for Fixed Effects Regression

Similar as for heteroskedasticity, autocorrelation invalidates the usual standard error fomulas and also alters the way heteroskedasticity-robust standard errors must be computed since these are derived under the assumption that there is no autocorrelation. In the context of heteroskedasticity *and* autocorrelation so-called *heteroskedasticity and autocorrelation-consistent (HAC) standard errors* need to be used. *Clustered standard errors* belong to these type of standard errors. They allow for heteroskedasticity and autocorrelated errors within an entity but *not* for any kind of correlation across entities.

As shown in the examples throughout this chapter, it is fairly easy to specify usage of clustered standard errors in regression summaries procuced by function like `coeftest()` in conjunction with `vcovHC()` from the `sandwich` package. Conveniently, `vcovHC()` recognizes panel model objects (objects of class `plm`) and computes clusterd standard errors by default.

The regressions conducted during this chapter are a good examples for why usage of clustered standard errors is crucial in fixed effects models. For example, consider the entity and time fixed effects regression model for fatalities. Since `fatal_tefe_lm_mod` is an object of class `lm`, `coeftest()` does not compute clustered standard errros but uses robust standard errors that are only valid in the absence of autocorrelated errors.

```
# check class of the model object
class(fatal_tefe_lm_mod)
```

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

```
# Summary based on heteroskedasticity-robust standard errors (no adjustment for autocorrelation)
coeftest(fatal_tefe_lm_mod, vcov = vcovHC(fatal_tefe_lm_mod, type = "HC1"))[1, ]
```

```
##   Estimate Std. Error    t value   Pr(>|t|)
## -0.6399800  0.2547149 -2.5125346  0.0125470
```

```
# check class of the (plm) model object
class(fatal_tefe_mod)
```

```
## [1] "plm"        "panelmodel"
```

```
# Summary based on clusterd standard errors (adjustment for autocorrelation + heteroskedasticity)
coeftest(fatal_tefe_mod, vcov = vcovHC(fatal_tefe_mod, type = "HC1"))
```

```
##
## t test of coefficients:
##
##         Estimate Std. Error t value Pr(>|t|)
## beertax -0.63998    0.35015 -1.8277  0.06865 .
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The outcomes deviate rather strongly: imposing no autocorrelation we obtain a standard error of 0.25 which implyies significance of β_1, the coefficient on $BeerTax$ at the level of 5%. On the contrary, using the clusterd standard error (0.35) leads to acceptance of the hypothesis $H_0 : \beta_1 = 0$ at the same level of significance, see equation (6.9). Consult Appendix 10.2 of the book for insights on the computation of clustered standard errors.

6.6 Drunk Driving Laws and Traffic Deaths

There are two major sources of omitted variable bias that are not accounted for by all of the models of the relation between trafic fatalitites and beer taxes that we have considered so far: economic conditions and driving laws. Fortunately, `Fatalities` has recordings on state specific legal drinking age (`drinkage`), punishment (`jail`, `service`) and various economic indicators like unemployment rate (`unemp`) and per capita income (`income`). We may use this data to extend the preceeding analysis.

At first, we define the variables according to the regression results presented in Table 10.1 of the book.

```
# Discretize the minimum legal drinking age
Fatalities$drinkagec <- cut(Fatalities$drinkage,
                            breaks = 18:22,
                            include.lowest = TRUE,
                            right = FALSE
                            )

# Set minimum drinking age [21,22] to be the baseline level
Fatalities$drinkagec <- relevel(Fatalities$drinkagec, "[21,22]")

# Mandadory jail or community service?
Fatalities$punish <- with(Fatalities, factor(jail == "yes" | service == "yes",
                                             labels = c("no", "yes")
                                             )
                          )

# Observations on all variables for 1982 and 1988 only
Fatalities_1982_1988 <- Fatalities[with(Fatalities, year == 1982 | year == 1988), ]
```

Next, we estimate all seven models using `plm()`.

```
fatalities_mod1 <- lm(fatal_rate ~ beertax, data = Fatalities)

fatalities_mod2 <- plm(fatal_rate ~ beertax + state, data = Fatalities)

fatalities_mod3 <- plm(fatal_rate ~ beertax + state + year,
                       index = c("state","year"),
                       model = "within",
                       effect = "twoways",
                       data = Fatalities)

fatalities_mod4 <- plm(fatal_rate ~ beertax + state + year + drinkagec
                       + punish + miles + unemp + log(income),
                       index = c("state", "year"),
                       model = "within",
                       effect = "twoways",
```

```
                        data = Fatalities)

fatalities_mod5 <- plm(fatal_rate ~ beertax + state + year + drinkagec
                        + punish + miles,
                        index = c("state", "year"),
                        model = "within",
                        effect = "twoways",
                        data = Fatalities)

fatalities_mod6 <- plm(fatal_rate ~ beertax + year + drinkage
                        + punish + miles + unemp + log(income),
                        index = c("state", "year"),
                        model = "within",
                        effect = "twoways",
                        data = Fatalities)

fatalities_mod7 <- plm(fatal_rate ~ beertax + state + year + drinkagec
                        + punish + miles + unemp + log(income),
                        index = c("state", "year"),
                        model = "within",
                        effect = "twoways",
                        data = Fatalities_1982_1988)
```

Yet again we may use `stargazer()` to generate a comprehensive tabular presentation of the results.

```
library(stargazer)

rob_se <- list(
  sqrt(diag(vcovHC(fatalities_mod1, type = "HC1"))),
  sqrt(diag(vcovHC(fatalities_mod2, type = "HC1"))),
  sqrt(diag(vcovHC(fatalities_mod3, type = "HC1"))),
  sqrt(diag(vcovHC(fatalities_mod4, type = "HC1"))),
  sqrt(diag(vcovHC(fatalities_mod5, type = "HC1"))),
  sqrt(diag(vcovHC(fatalities_mod6, type = "HC1"))),
  sqrt(diag(vcovHC(fatalities_mod7, type = "HC1")))
)

stargazer(fatalities_mod1, fatalities_mod2, fatalities_mod3,
          fatalities_mod4, fatalities_mod5, fatalities_mod6, fatalities_mod7,
          digits = 3,
          type = "latex",
          se = rob_se,
          model.numbers = FALSE,
          column.labels = c("(1)", "(2)", "(3)", "(4)", "(5)", "(6)", "(7)")
          )
```

% Table created by stargazer v.5.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvard.edu
% Date and time: Tue, May 29, 2018 - 14:12:29 % Requires LaTeX packages: rotating

While columns (2) and (3) recap the results (6.8) and (6.9), column (1) presents an estimate of the coefficient of interest in the naive OLS regression of the fatality rate on beer tax without any fixed effects. We obtain a *positive* estimate for the coefficient on beer tax that is likely to be upward biased. Notice that the model fit is rather bad, too ($\overline{R^2} = 0.091$). The sign of the estimate changes as we extend the model by both entity and time fixed effects in models (2) and (3). Furthermore $\overline{R^2}$ increases substantially as fixed effects are included in the model equation. Nonetheless, as discussed before, the magnitudes of both estimates are by far too

Table 6.1:

	Dependent variable:						
	fatal_rate						
	OLS	Linear Panel Regression					
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
beertax	0.365***	−0.656**	−0.640*	−0.445	−0.690**	−0.456	−0.926***
	(0.053)	(0.289)	(0.350)	(0.291)	(0.345)	(0.301)	(0.337)
drinkagec[18,19)				0.028	−0.010		0.037
				(0.068)	(0.081)		(0.101)
drinkagec[19,20)				−0.018	−0.076		−0.065
				(0.049)	(0.066)		(0.097)
drinkagec[20,21)				0.032	−0.100*		−0.113
				(0.050)	(0.055)		(0.123)
drinkage						−0.002	
						(0.021)	
punishyes				0.038	0.085	0.039	0.089
				(0.101)	(0.109)	(0.101)	(0.161)
miles				0.00001	0.00002*	0.00001	0.0001***
				(0.00001)	(0.00001)	(0.00001)	(0.00005)
unemp				−0.063***		−0.063***	−0.091***
				(0.013)		(0.013)	(0.021)
log(income)				1.816***		1.786***	0.996
				(0.624)		(0.631)	(0.666)
Constant	1.853***						
	(0.047)						
Observations	336	336	336	335	335	335	95
R^2	0.093	0.041	0.036	0.360	0.066	0.357	0.659
Adjusted R^2	0.091	−0.120	−0.149	0.217	−0.134	0.219	0.157
Residual Std. Error	0.544 (df = 334)						
Note:							*p<0.1; **p<0.05; ***p<0.01

large.

Thus, model specifications (4) to (7) include covariates that shall capture the effect of overall state economic conditions as well as the legal framework. These covariates are as follows:

- `unemp`: a numeric variable stating the state specific unemployment rate.
- `log(income)`: the logarithm of real per capita income (in prices of 1988).
- `miles`: average miles per driver.
- `drinkage`: state specifiy minimum legal drinking age.
- `drinkagc`: a discretized version of `drinkage` that classifies states into four categories of minimal drinking age; 18, 19, 20, 21 and older. R denotes this as `[18,19)`, `[19,20)`, `[20,21)` and `[21,22]`. These categories are included as dummy regressors whereby `[21,22]` is chosen to be the reference category.
- `punish`: a dummy variable with levels `yes` and `no` that measures if drunk driving is severely punished by mandatory jail time or mandatory community service (first conviction).

Considering (4) as the base specification, we observe four interesting results:

1. Including the covariates does not lead to a major reduction of the estimated effect of the beer tax. Plus, the coefficient is not significantly different from zero at the level of 5% as the estimate is rather imprecise.

2. The minimum legal drinking age *does not* have an effect on traffic fatalities: none of the three dummy variables are significantly different from zero at any level of significance common in practice. Moreover, an F-Test of the joint hypothesis that all three coefficients are zero does not reject. The next code chunk shows how to test this hypothesis.

```
# Test if legal drinking age has no explanatory power
linearHypothesis(fatalities_mod4,
                 test = "F",
                 c("drinkagec[18,19)=0", "drinkagec[19,20)=0", "drinkagec[20,21)"),
                 vcov. = vcovHC(fatalities_mod4, type = "HC1"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## drinkagec[18,19) = 0
## drinkagec[19,20) = 0
## drinkagec[20,21) = 0
##
## Model 1: restricted model
## Model 2: fatal_rate ~ beertax + state + year + drinkagec + punish + miles +
##     unemp + log(income)
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df      F Pr(>F)
## 1    276
## 2    273  3 0.3691 0.7753
```

3. There is no evidence that punishment for first offenders has a deterring effects on drunk driving: the corresponding coefficient is not significant at the 10% level.

4. The economic variables have power in explaning traffic fatalities. We can check that the employment rate and per capita income are jointly significant at the level of 0.1%.

```
# Test if economic indicators have no explanatory power
linearHypothesis(fatalities_mod4,
                 test = "F",
```

```
                c("log(income)", "unemp"),
                vcov. = vcovHC(fatalities_mod4, type = "HC2"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## log(income) = 0
## unemp = 0
##
## Model 1: restricted model
## Model 2: fatal_rate ~ beertax + state + year + drinkagec + punish + miles +
##     unemp + log(income)
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df      F    Pr(>F)
## 1    275
## 2    273  2 30.482 1.125e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Model (5) omitts the economic factors. The result supports the presumption that economic indicators should remain part of the model as the coefficient on beer tax is sensitive to the inclusion of the latter.

Results for model (6) are in favour of the presumption that legal drinking age has little explanatory power and that the coefficient of interest is not senstive to changes in the functional form of the relation between drinking age and traffic fatalities.

Estimation outcomes for spefcification (7) reveals that reducing the amount of available information inflates standard errors but does not lead to drastic changes in coefficient estimates.

Summary

We have not found evidence that menacing severe punishments and increasing the minimum drinking ages are appropriate measures for reducing traffic fatalities due to drunk driving. Nonetheless we concluded that there is some effect of alcohol taxes on traffic fatalities which, however, is estimated imprecisely and cannot be interpreted as the causal effect of interest as there seems to be omitted variable bias. This bias persists even though we used a panel approach that controls for entity specific and time invariant unobservables. A possible source for this bias are omitted variables that change across entities *and* over time.

A powerful method that can be used if common panel regression approaches fail is instrumental variables regression. We will return to this concept in Chapter 12.