A Minimal Book Example

Yihui Xie

2017-09-24

Contents

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

For now, you have to install the development versions of **bookdown** from Github:

```
devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need to install XeLaTeX.

Chapter 2

Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 2.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 2.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (**?**) in this sample book, which was built on top of R Markdown and **knitr** (**?**).

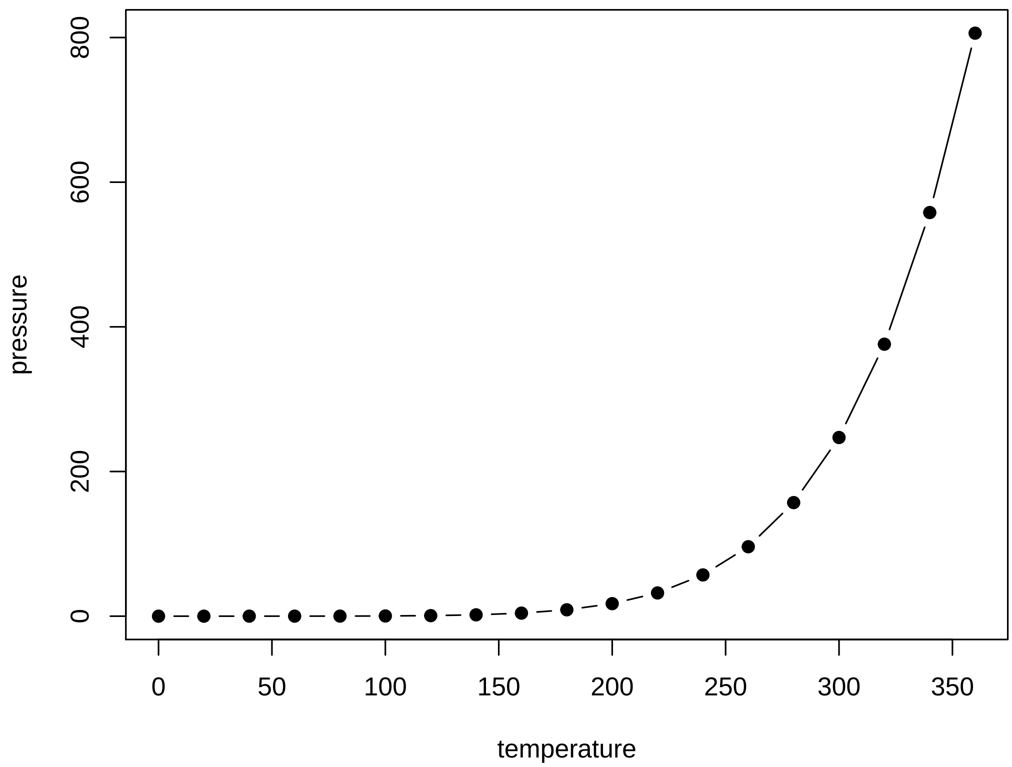


Figure 2.1: Here is a nice figure!

Table 2.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 3

Linear Regression with One Regressor

This chapter introduces the simple linear regression model which relates one variable, X, to another variable Y. If for example a school cuts the class sizes by hiring new teachers, that is the school lowers the student-teacher ratios of their classes, how would this effect the performance of the students? With linear regression we can not only examine whether the student-teacher ratio (X) does have an impact on the test results (Y). We can also learn about the direction and the strength of this effect.

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

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

To work with this data in R, we create 2 vectors, one for the student-teacher ratios and one for test scores which contain 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

$$y = bx + a.$$

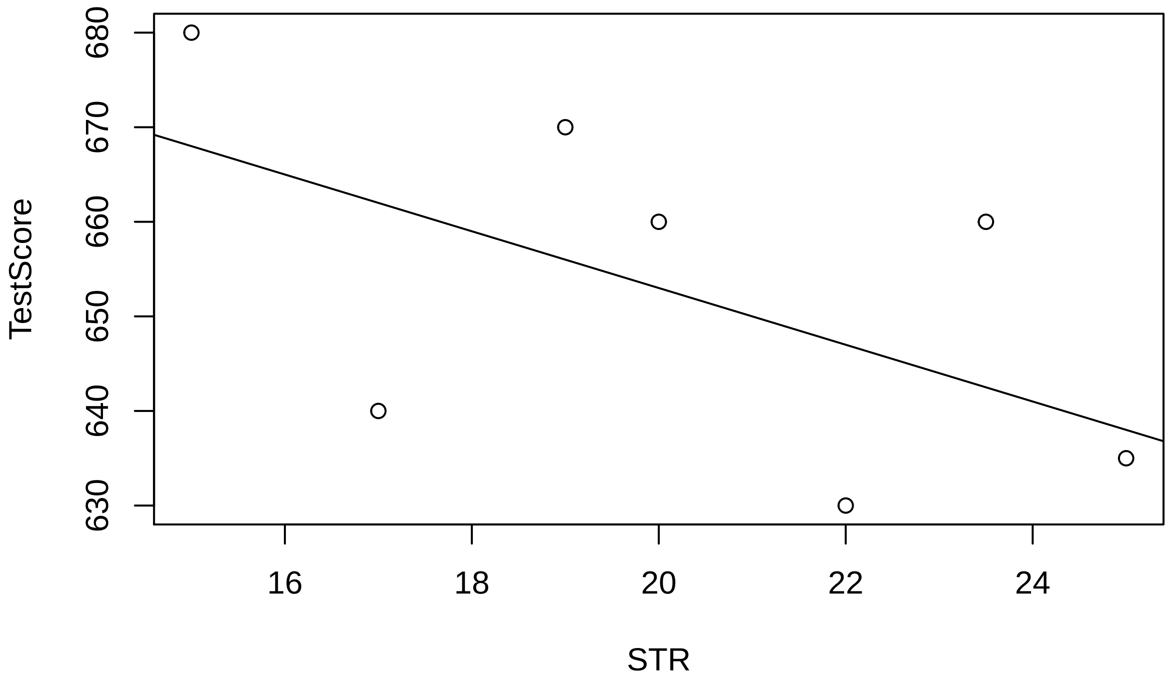
Let us suppose that the true function which relates test score and student-teacher ratio to each other is

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

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 scatterplot 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 scatterplot. 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 our 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 exacerbate finding of 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 measerment errors or, as will be discussed later, are the consequence of leaving out other factors that are relevant in explaining the dependent variable. Which other factor 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 extented 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 subscript i runs over the observations, $i = 1, ..., n$
- 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*

3.1 Estimating the Coefficients of the Linear Regression Model

In practice, the intercept β_0 and slope $\beta_1$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 in 420 California school destricts. 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). The California School dataset is available a 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 dataset can be loaded using the `data` function.

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

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

For several reasons it is interesting to know what kind of object we are dealing with. `class(object_name)` returns the type (class) of an object. Depending on the class of an object several functions (such as `plot` and `summary`) behave differently.

```
class(CASchools)
```

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

It turns out that `CASchools` is of class `data.frame` which is a convienient format to work with.

With help of the function `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 (hint: press ctrl + L to clear the console). An alternative to `class` and `head` is `str` which is deduced from 'structure' and gives a comprehensive overview of the object.

```
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 dataset consists of plenty of variables and most of them are numeric. The two variables we are intersted 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 divide the number of students by the number of teachers. The avarage 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 and how they are added to the dataframe `CASchools`

```
CASchools$STR <- CASchools$students/CASchools$teachers
CASchools$score <- (CASchools$read + CASchools$math)/2
```

If we run `head` again we would now find the two variables of interest `STR` and `score` (check this!).

Table 4.1 from the text book summarizes the distribution of test scores and student-teacher ratios. The functions `mean` (computes the arithmetic mean of the provided numbers), `sd` (computes the standard deviation), and `quantile` (returns a vector of the specified quantiles) can be used to obtain the same results.

In order to have a nice display format we gather all the data after the computation in a `data.frame` object named `DistributionSummary`.

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

# compute standard deviations
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)
                                  )
DistributionSummary
```

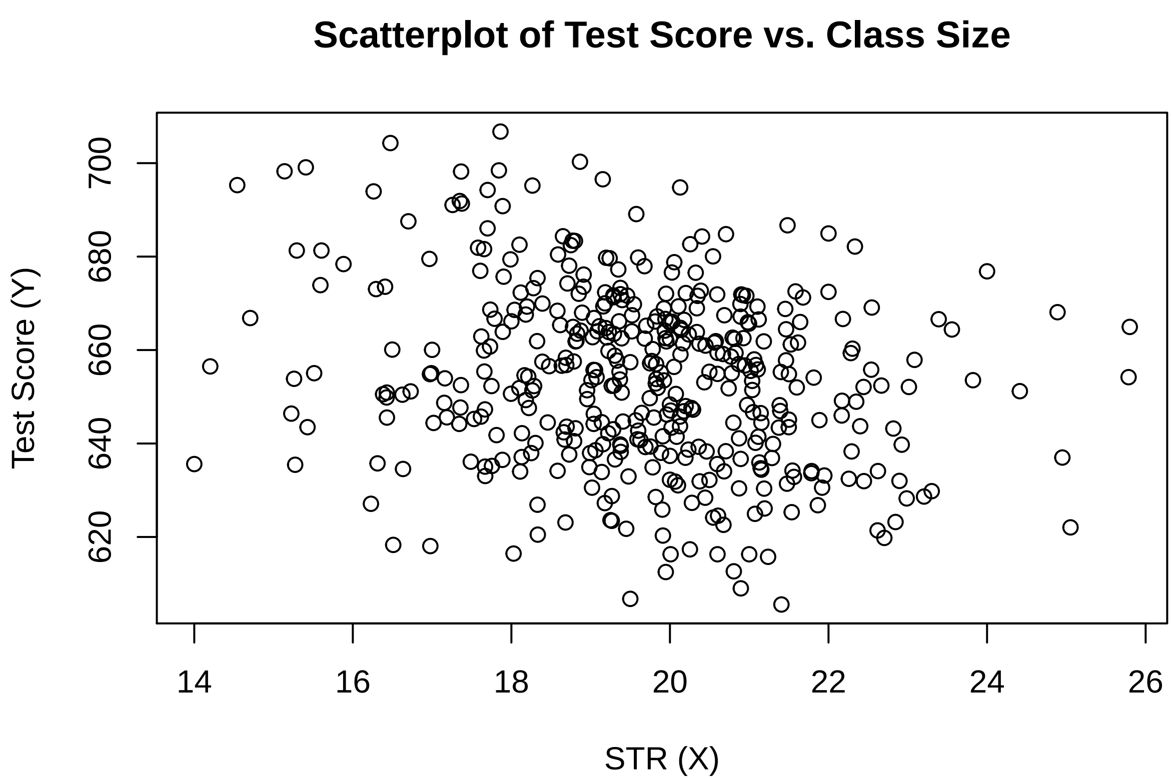
```
##                Average StandardDeviation quantile.10. quantile.25.
```

```
## 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
```

R already contains a `summary` function which can be applied to `data.frames`. Try it!

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 `plot` function. The first argument `score ~ STR` is again a formula representing the y and the x variables. However, this time the two variables are not saved in seperate vectores but are gathered in a `data.frame`. Therefore, R would not find the variables without the argument `data` beeing correctly specified. `data` must be in accordance with the name of the `data.frame` to which the variables belong, i.e. `CASchools`. The other arguments used change the appearance of the plot: `main` adds a title and `xlab` and `ylab` are adding custom labels to the axes.

```r
plot(score ~ STR,
     data = CASchools,
     main = "Scatterplot of Test Score vs. Class Size",
     xlab = "STR (X)",
     ylab = "Test Score (Y)"
)
```



The plot (figure 4.2 in the book) shows the scatterplot of all observations. We see that the points are strongly scatterd and an apparent relationship cannot be detected by looking at it. 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` can be used to compute the correlation between 2 numerical vectors.

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

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

As the scatterplot already suggested the correlation is negative but rather weak.

The task we are facing now is to find the regression line which fits best to the data. Of course we could simply do graphical inspection, some correlation analysis and then select the best fitting line by eyeballing. This is 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.

3.2 The Ordinary Least Squares (OLS) Estimator

The OLS estimator chooses the regression coefficients such that the estimated regression line is as close as possible to the observed data points. 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 dervivation 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

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2} \tag{3.1}$$

$$\tag{3.2}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X} \tag{3.3}$$

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

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i, \tag{3.4}$$

$$\tag{3.5}$$

$$\hat{u}_i = Y_i - \hat{Y}_i. \tag{3.6}$$

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 unkown true population intercept (β_0), slope (β_1), and error term (u_i).

There are many possible ways to compute $\left(\hat{\beta}_0\right)$ and $\left(\hat{\beta}_1\right)$ in R. We could implement the formulas with two of R's most basic functions: `mean` and `sum`. Of course there are also other and even more manual ways to do the same tasks.

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

```
## The following objects are masked from CASchools (pos = 3):
##
##     calworks, computer, county, district, english, expenditure,
##     grades, income, lunch, math, read, school, score, STR,
##     students, teachers
```

```
# 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
```

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 is the regression formula with the basic syntax `y ~ x` where `y` is the dependent variable and `x` the explanatory variable. The argument `data` specifies the data set to be used in the regression. We now revisit the example from the book where the relationship between the students' test scores and the class sizes is analysed. The following code uses `lm` to replicate the results presented in figure 4.3 in the book.

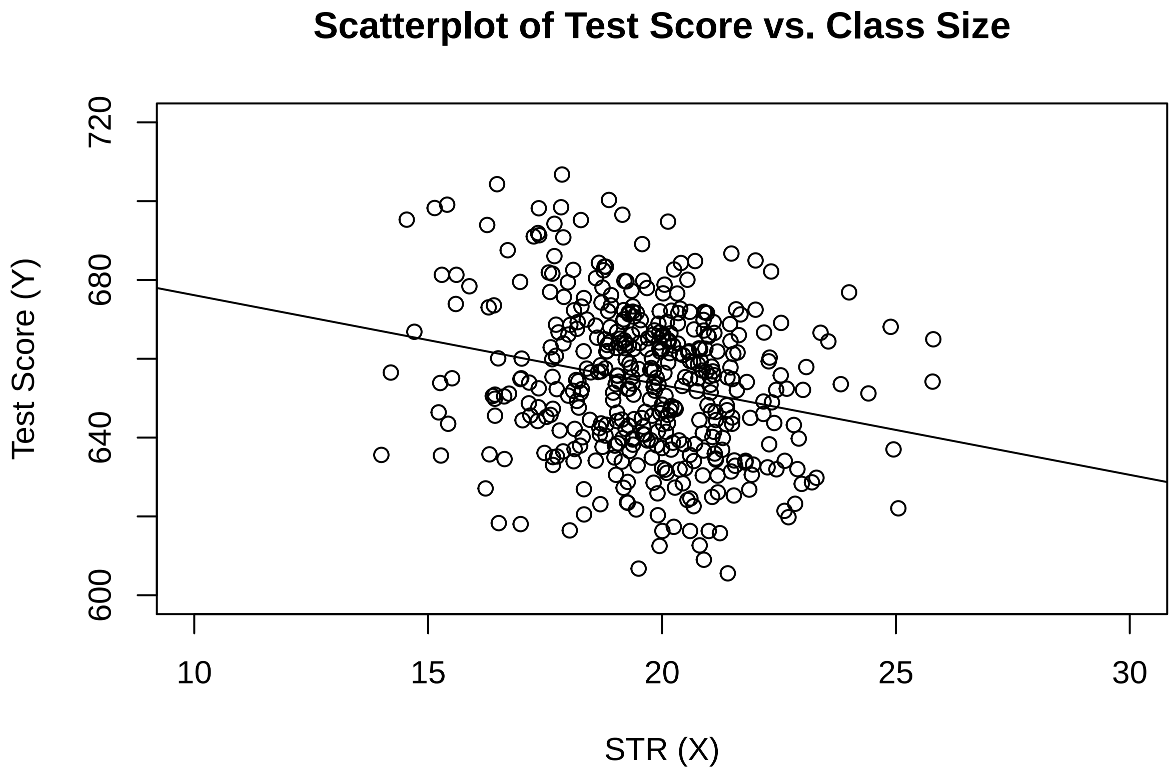
```
# estimate the model and assign the result to linear_model
linear_model <- lm(score ~ STR, data = CASchools)

# 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
```

```
# plot the data
plot(score ~ STR,
     data = CASchools,
     main = "Scatterplot of Test Score vs. Class Size",
     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, Rdraws the regression line automatically!

3.3 Measures of fit

After estimating a linear regression, the question occurs how well that regression line describes the data. Are the observations tightly clustered arround the regression line, or are they spread out? Both, the R^2 and the *standard error of the regression* (SER) measure how well the OLS Regression line fits the data.

3.3.1 The R^2

The R^2 is the fraction of sample variance of Y_i that is explained by X_i. Mathemethically, 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.

$$ESS = \sum_{i=1}^{n} \left(\hat{Y}_i - \bar{Y}\right)^2 \tag{3.7}$$

$$\tag{3.8}$$

$$TSS = \sum_{i=1}^{n} \left(Y_i - \bar{Y}\right)^2 \tag{3.9}$$

$$\tag{3.10}$$

$$R^2 = \frac{ESS}{TSS} \tag{3.11}$$

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 $R^2 = 0$.

3.3.2 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 measure the magnitude of a typical deviation from the regression, 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.

3.3.3 Application to the Test Score Data

Both measures of fit can be obtained by using the function `summary` with the `lm` object provided as the only argument. Whereas `lm` only prints out the coefficients, `summary` provides additional predefined information such as the 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 takes the value 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 some of the variance but much of the variation in test scores remains unexplained (compare figure 4.3 in 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 out 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 the `summary` function uses the same definition 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
R2
```

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

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

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

We find that the results coincide.

3.4 The Least Squares Assumptions

OLS performs well under a great 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

$$Y_i = \beta_0 + \beta_1 X_i + u_i, \; i = 1, ..., n, \text{ where}$$

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

3.4.1 Assumption #1: The Error Term has Conditional Mean of Zero

This means that no matter which X-value we choose, the error term should not show any systematic pattern and 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 can start by creating a vector of X-values. For our example we decide to generate uniformly distributed numbers which can be done with the function `runif`. We also need to simulate the error term. For this we generate normally distributed numbers with a mean equal to 0. Finally, the Y-value is obtained as a quadratic function of the X-values and the error term. Next, 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.

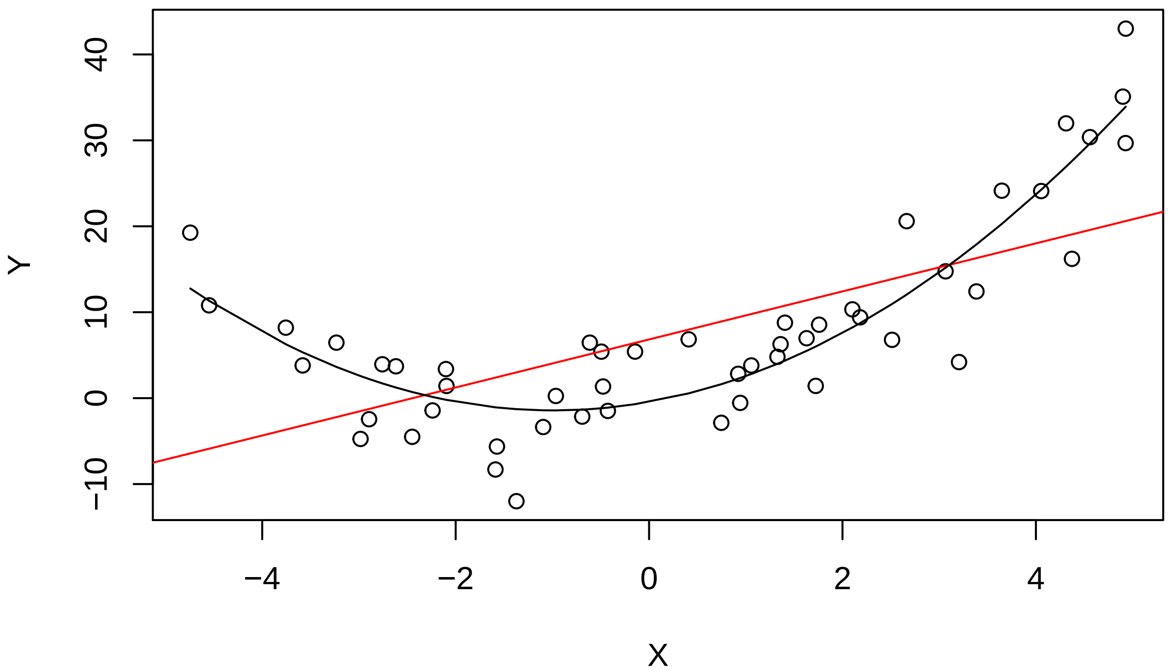
```
# set a random seed to make the results reproducible
set.seed(321)

# simulate the data
X <- runif(50, min = -5, max = 5)
u <- rnorm(50, sd = 5)
Y <- X^2 + 2*X + u # the true relation

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

# precit 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)
```



This shows what is meant by $E(u_i|X_i) = 0$: using the quadratic model 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.

3.4.2 Assumption #2: All (X_i, Y_i) are Independtly and Identically Distributed

Most common 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 time after time. Using R we can simulate such a process and plot it. We start the series with a total of 5000 workers and simulate the reduction of employment as a declining process with normal distributed random influences.

```
# 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")
```

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

3.4.3 Assumption #3: Sensitivity to Outliers

```
set.seed(123)
x     <- sort(runif(10, min = 30, max = 70 ))
y     <- rnorm(10 , mean = 200, sd = 50)
y[9] <- 2000

fit               <- lm(y ~ x)
fitWithoutOutlier <- lm(y[-9] ~ x[-9])
```