Data Handling: Import, Cleaning and Visualisation

Lecture 10: Understanding Basic Statistics with R

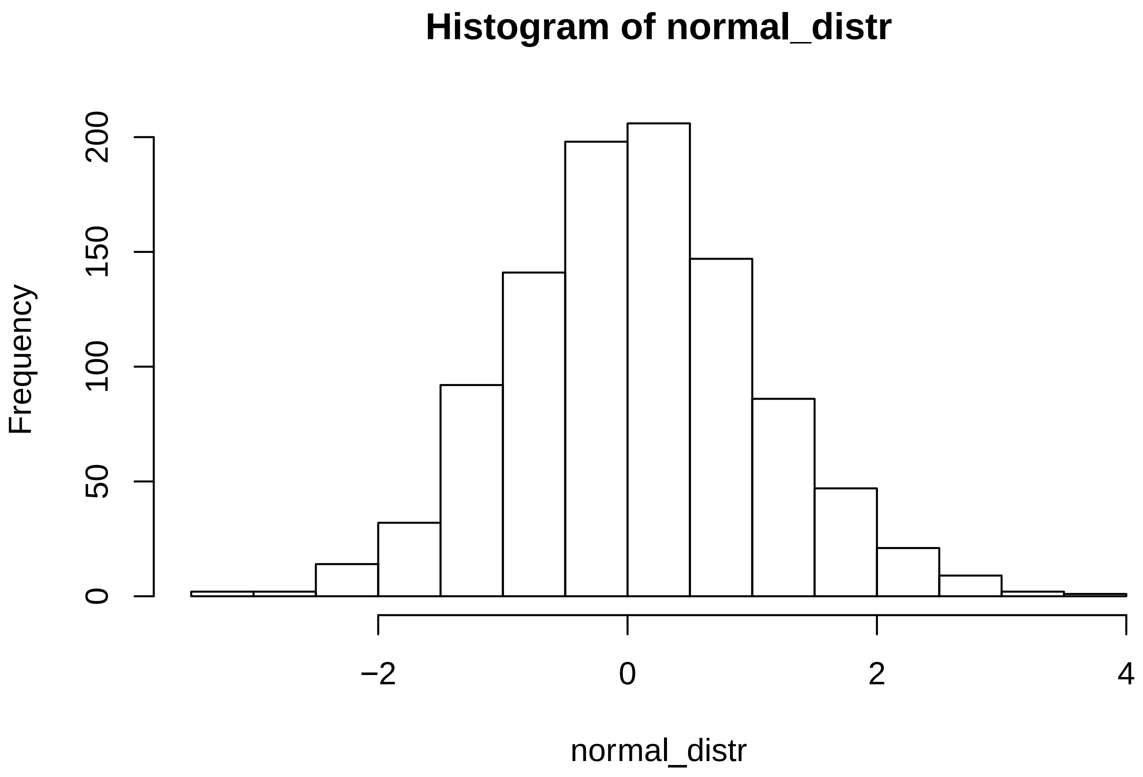
Prof. Dr. Ulrich Matter
(University of St. Gallen)

06/12/2018

1 Unterstanding Statistics and Probability with Code

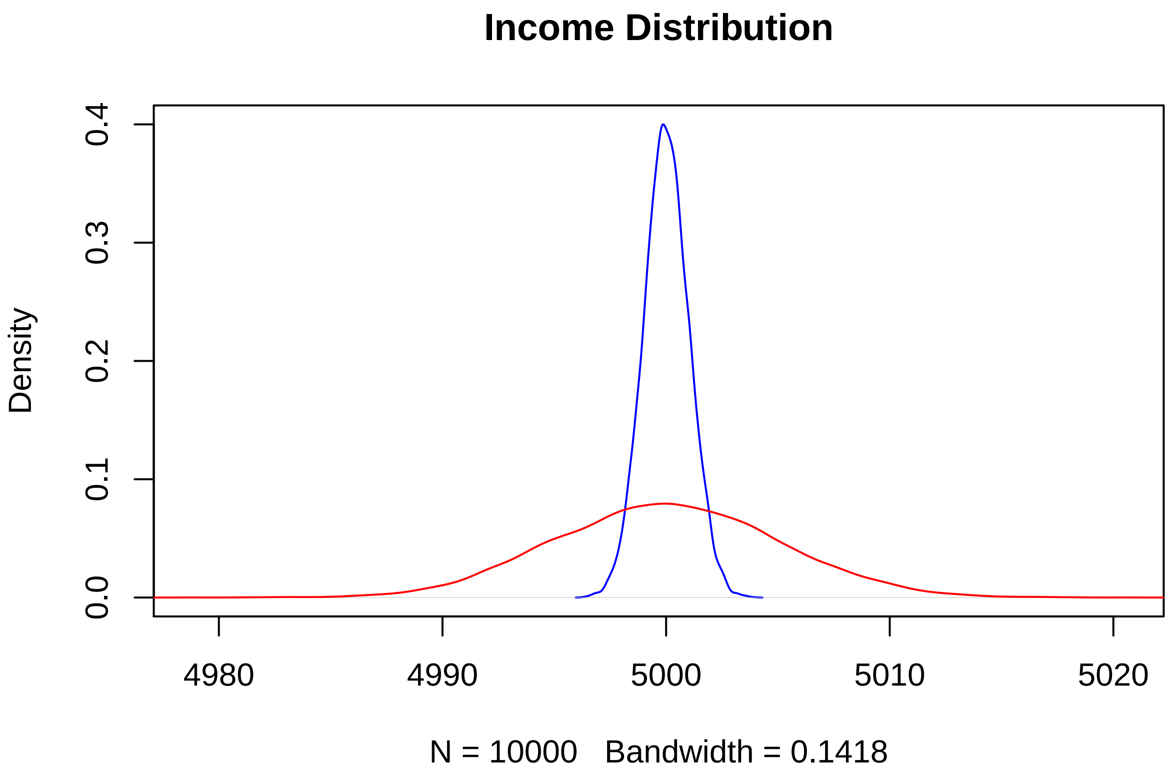
1.1 Random Draws and Distributions

```r
normal_distr <- rnorm(1000)
hist(normal_distr)
```



1.2 Illustration of Variability

```r
# draw a random sample from a normal distribution with a large standard deviation
largevar <- rnorm(10000, mean = 5000, sd = 5)
# draw a random sample from a normal distribution with a small standard deviation
littlevar <- rnorm(10000, mean = 5000, sd = 1)

# visualize the distributions of both samples with a density plot
plot(density(littlevar), col = "blue",
     xlim=c(min(largevar), max(largevar)), main="Income Distribution")
lines(density(largevar), col = "red")
```



Note: the red curve illustrates the distribution of the sample with a large standard deviation (a lot of variability) whereas the blue curve illustrates the one with a rather small standard deviation.

1.3 Skewness and Kurtosis

```
# Install the R-package called "moments" with the following command (if not installed yet):
# install.packages("moments")

# load the package
library(moments)
```

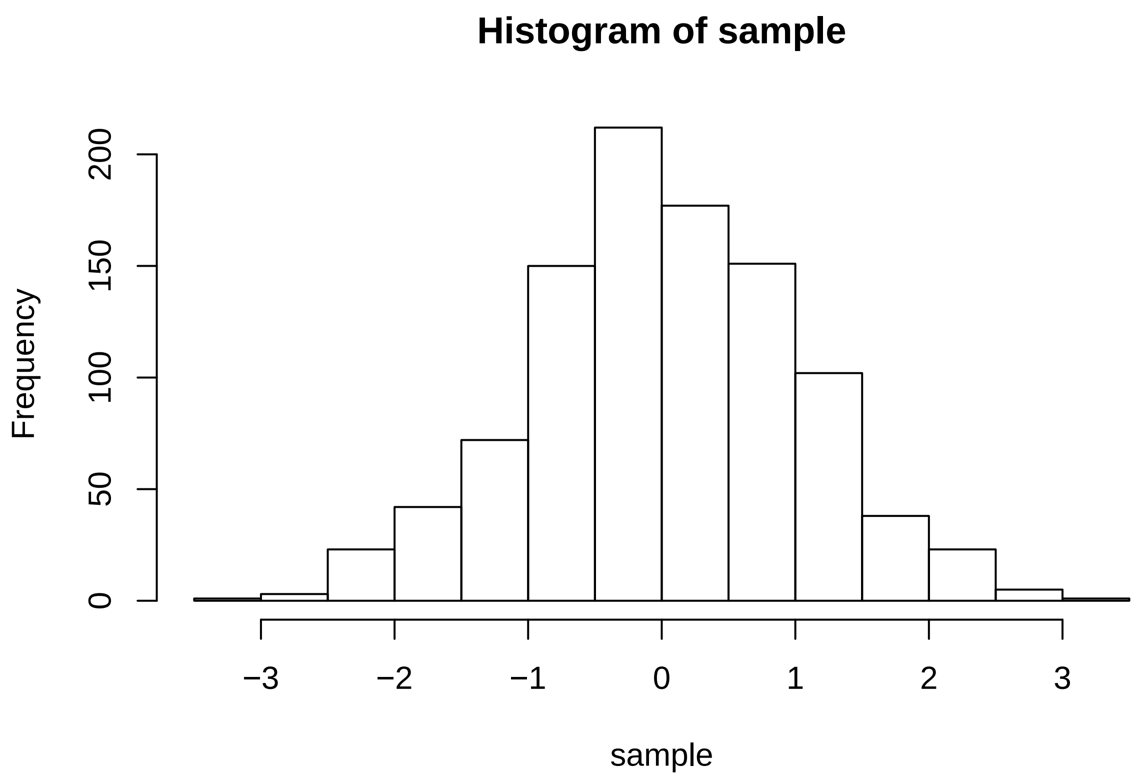
Recall Day 1's slides on Skewness and Kurtosis. A helpful way to memorize what a certain value of either of these two statistics means is to visualize the respective distribution (as shown in the slides).

1.3.1 Skewness

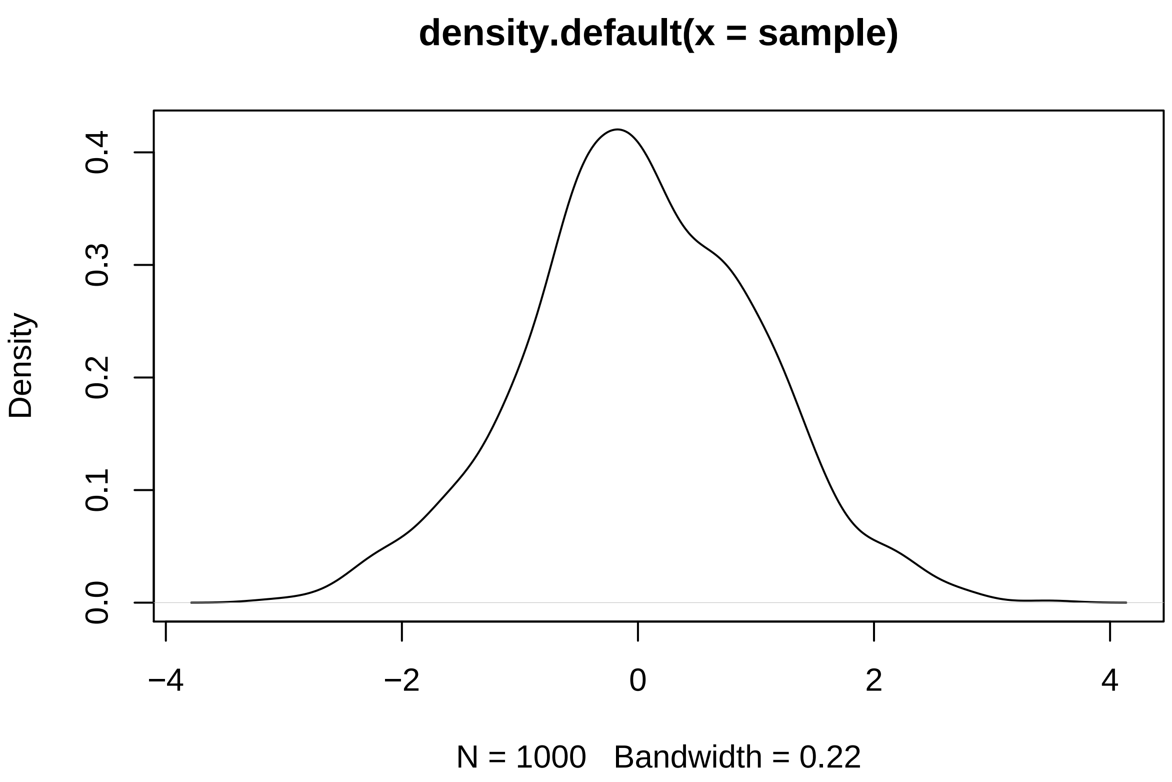
Skewness refers to how symetric the frequency distribution of a variable is. For example, a distribution can be 'positively skewed' meaning it has a long tail on the right and a lot of 'mass' (observations) on the left. We can see that when visualizing the distribution in a histogram or a density plot. Lets have a look at this in R (note the comments in the code explaing what each line does):

```
# draw a random sample of simulated data from a normal distribution
# the sample is of size 1000 (hence, n = 1000)
sample <- rnorm(n = 1000)
```

```r
# plot a histogram and a density plot of that sample
# note that the distribution is neither strongly positively nor negatively skewed
# (this is to be expected, as we have drawn a sample from a normal distribution!)
hist(sample)
```



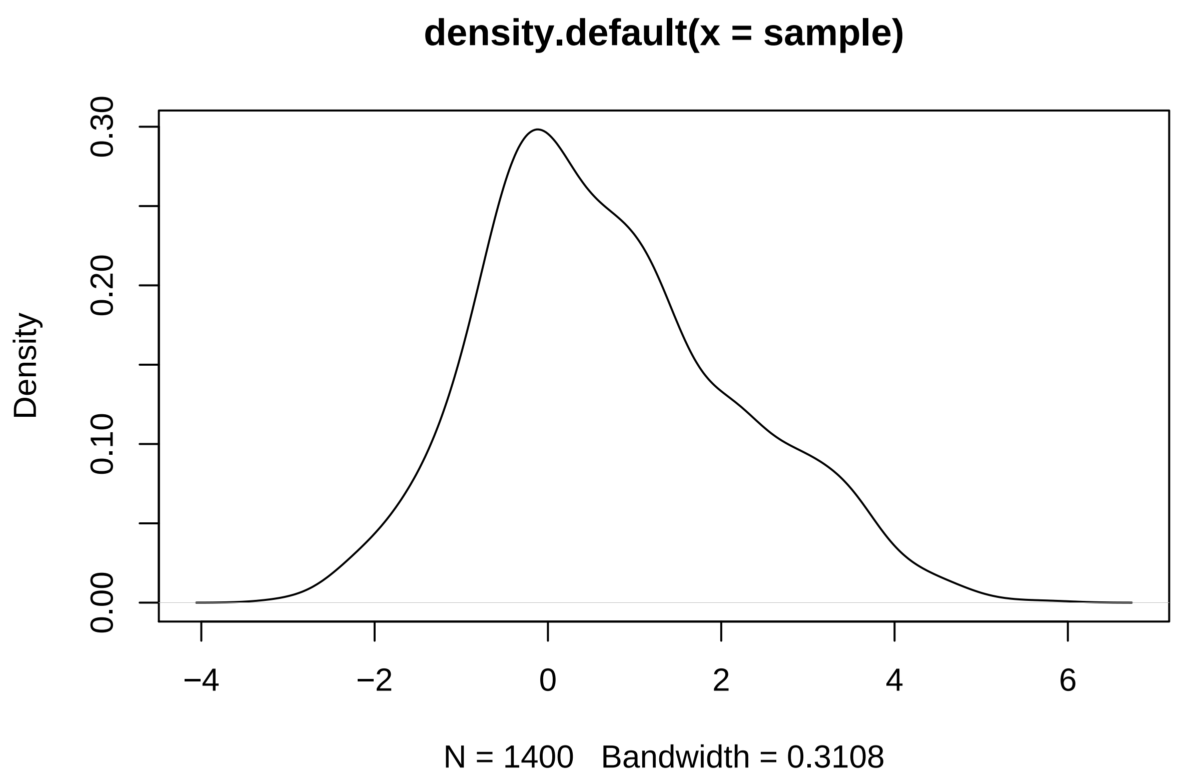
```r
plot(density(sample))
```



```
# now compute the skewness
skewness(sample)
```

```
## [1] 0.003031765
```

```
# Now we intentionally change our sample to be strongly positively skewed
# We do that by adding some outliers (observations with very high values) to the sample
sample <- c(sample, (rnorm(200) + 2), (rnorm(200) + 3))

# Have a look at the distribution and re-calculate the skewness
plot(density(sample))
```



```
skewness(sample)
```

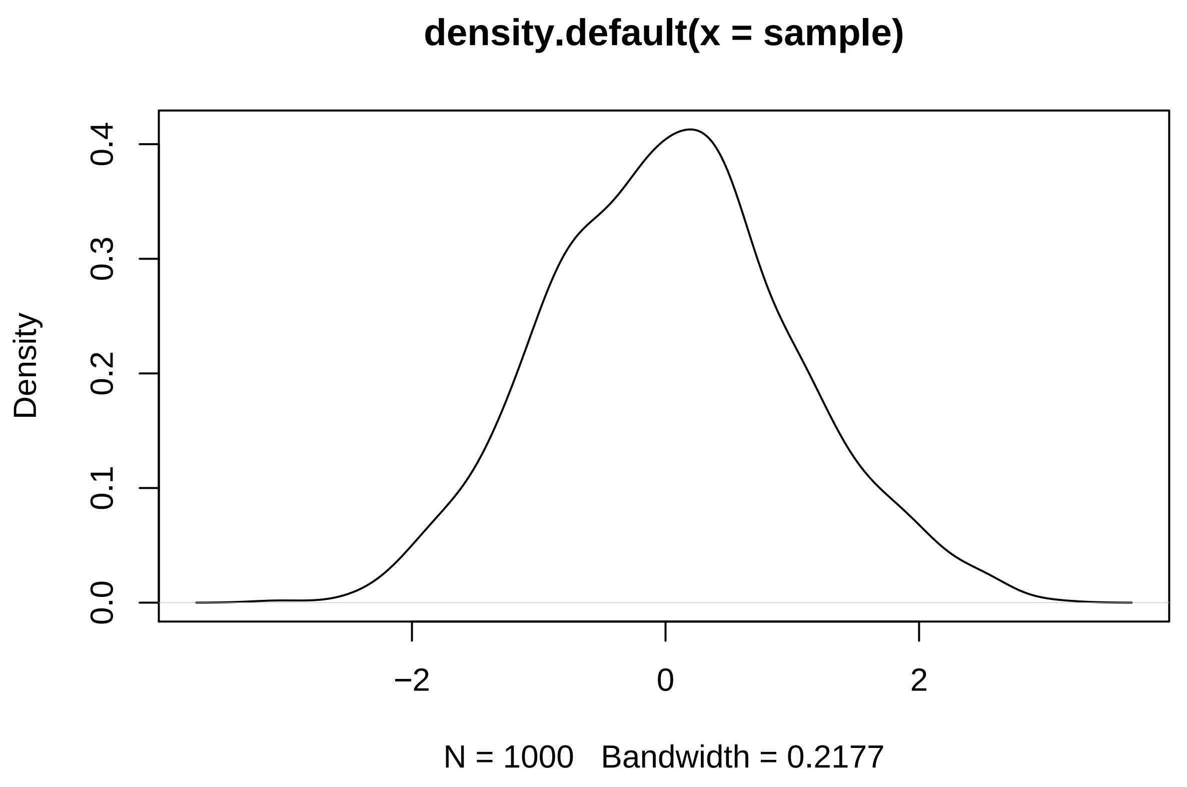
```
## [1] 0.4456965
```

```
#
```

1.3.2 Kurtosis

Kurtosis refers to how much 'mass' a distribution has in its 'tails'. It thus tells us something about whether a distribution tends to have a lot of outliers. Again, plotting the data can help us understand this concept of kurtosis. Lets have a look at this in R (note the comments in the code explaing what each line does):

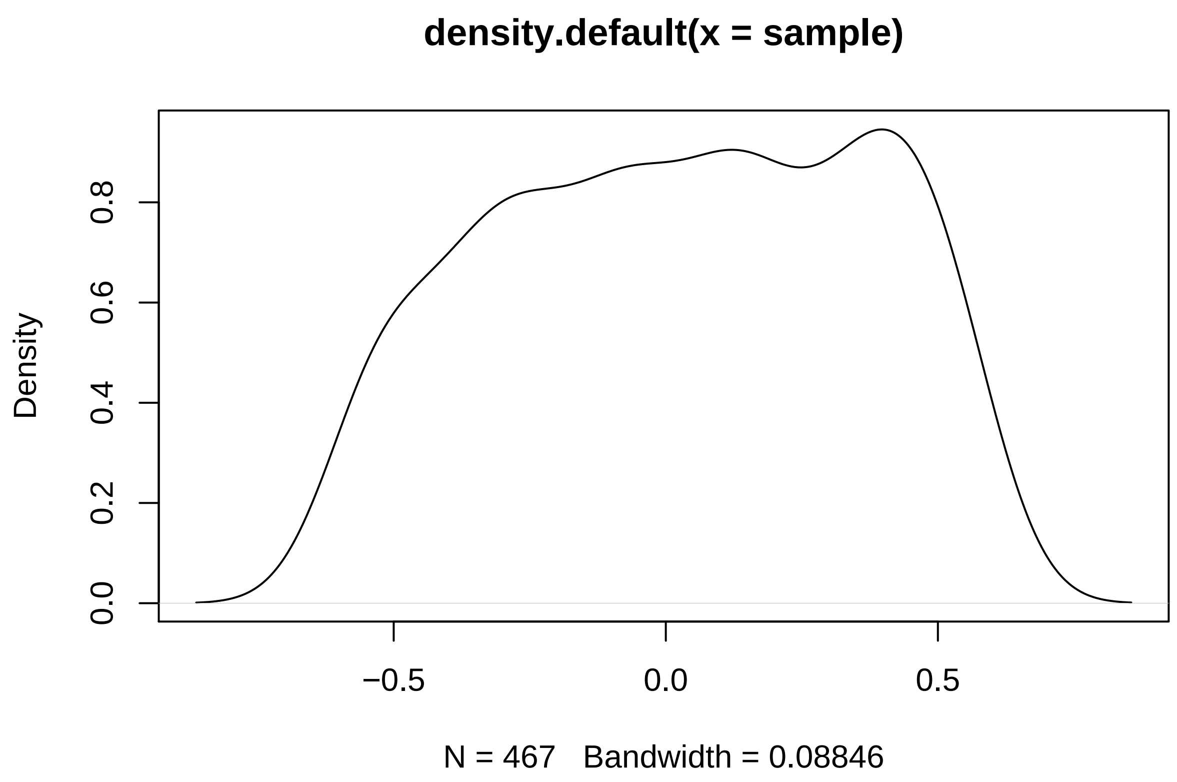
```
# draw a random sample of simulated data from a normal distribution
# the sample is of size 1000 (hence, n = 1000)
sample <- rnorm(n = 1000)

# plot the density & compute the kurtosis
plot(density(sample))
```



```r
kurtosis(sample)
```

```
## [1] 2.871878
```

```r
# now lets remove observations from the extremes in this distribution
# we thus intentionally alter the distribution to have less mass in its tails
sample <- sample[ sample > -0.6 & sample < 0.6]

# plot the distribution again and see how the tails of it (and thus the kurtosis has changed)
plot(density(sample))
```



```
# re-calculate the kurtosis
kurtosis(sample)
```

```
## [1] 1.864989
```

```
# as expected, the kurtosis has now a lower value
```

1.3.3 Implement the formulas for skewness and kurtosis in R

Skewness

```
# own implementation
sum((sample-mean(sample))^3) / ((length(sample)-1) * sd(sample)^3)
```

```
## [1] -0.1156478
```

```
# implementation in moments package
skewness(sample)
```

```
## [1] -0.1157718
```

Kurtosis

```
# own implementation
sum((sample-mean(sample))^4) / ((length(sample)-1) * sd(sample)^4)
```

```
## [1] 1.860995
```

```
# implementation in moments package
kurtosis(sample)
```

```
## [1] 1.864989
```

1.4 The Law of Large Numbers

The Law of Large Numbers (LLN) is an important statistical property which essentially describes how the behavior of sample averages is related to sample size. Particularly, it states that the sample mean can come as close as we like to the mean of the population from which it is drawn by simply increasing the sample size. That is, the larger our randomly selected sample from a population, the closer is that sample's mean to the mean of the population.

Think of playing dice. Each time we roll a fair die, the result is either 1, 2, 3, 4, 5, or 6, whereby each possible outcome can occur with the same probability (1/6). In other words we randomly draw die-values. Thus we can expect that the average of the resulting die values is $(1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5$.

We can investigate this empirically: We roll a fair die once and record the result. We roll it again, and again we record the result. We keep rolling the die and recording results until we get 100 recorded results. Intuitively, we would expect to observe each possible die-value about equally often (given that the die is fair) because each time we roll the die, each possible value (1,2,..,6) is equally likely to be the result. And we would thus expect the average of the resulting die values to be around 3.5. However, just by chance it can obviously be the case that one value (say 5) occurs slightly more often than another value (say 1), leading to a sample mean slightly larger than 3.5. In this context, the LLN states that by increasing the number of times we are rolling the die, we will get closer and closer to 3.5. Now, let's implement this experiment in R:

```r
# first we define the potential values a die can take
dvalues <- 1:6 # the : operater generates a regular sequence of numbers (from:to)
dvalues
```

```
## [1] 1 2 3 4 5 6
```

```r
# define the size of the sample n (how often do we roll the die...)
# for a start, we only roll the die ten times
n <- 10
# draw the random sample: 'roll the die n times and record each result'
results <- sample( x = dvalues, size = n, replace = TRUE)
# compute the mean
mean(results)
```

```
## [1] 3.3
```

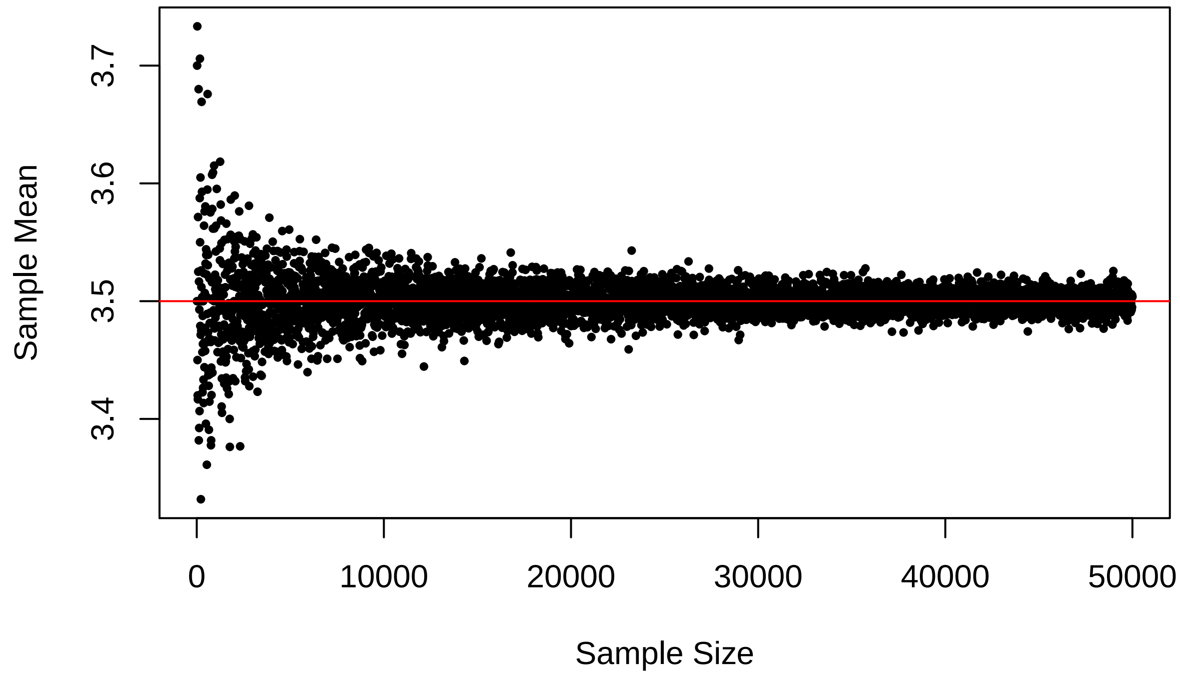
As you can see we are relatively close to 3.5, but not quite there. So let's roll the die more often and calculate the mean of the resulting values again:

```r
n <- 100
# draw the random sample: 'roll the die n times and record each result'
results <- sample( x = dvalues, size = n, replace = TRUE)
# compute the mean
mean(results)
```

```
## [1] 3.72
```

We are getting closer to 3.5! Now let's scale up these comparisons and show how the sample means are getting closer to 3.5 when increasing the number of times we roll the die up to 50'000.

```r
# Essentially, what we are doing here is repeating the experiment above many times, each time increasin
# define the set of sample sizes
ns <- seq(from = 10, to = 50000, by = 10)
# initiate an empty list to record the results
means <- list()
length(means) <- length(ns)
# iterate through each sample size: 'repeat the die experiment for each sample size'
for (i in 1:length(ns)) {
    means[[i]] <- mean(sample( x = dvalues, size = ns[i], replace = TRUE))
}

# visualize the result: plot sample means against sample size
plot(ns, unlist(means),
     ylab = "Sample Mean",
     xlab = "Sample Size",
     pch = 16,
     cex = .6)
abline(h = 3.5, col = "red")
```



We observe that with smaller sample sizes the sample means are broadly spread around the population mean of 3.5. However, the more we go to the right extreme of the x-axis (and thus the larger the sample size), the narrower the sample means are spread around the population mean.

1.5 The Central Limit Theorem

The Central Limit Theorem (CLT) is an almost miraculous statistical property enabling us to test the statistical significance of a statistic such as the mean. In essence, the CLT states that as long as we have a large enough sample, the t-statistic (applied, e.g., to test whether the mean is equal to a particular value) is approximately standard normal distributed. This holds independently of how the underlying data is distributed.

Consider the dice play example above. We might want to statistically test whether we are indeed playing with a fair die (which would imply an expected mean value of 3.5!). In order to test that we would roll the die 100 times and record each resulting value. We would then compute the sample mean and standard deviation in order to assess how likely it was to observe the mean we observe if the population mean actually is 3.5 (thus our H0 would be pop_mean = 3.5, or in plain English 'the die is fair'). However, the distribution of the resulting die values are not standard normal distributed. So how can we interpret the sample standard deviation and the sample mean in the context of our hypothesis?

The simplest way to interpret these measures is by means of a *t-statistic*. A t-statistic for the sample mean under our working hypothesis that pop_mean = 3.5 is constructed as `t(3.5) = (sample_mean - 3.5) / (sample_sd/sqrt(n))`. Let's illustrate this in R:

```r
# First we roll the die like above
n <- 100
# draw the random sample: 'roll the die n times and record each result'
results <- sample( x = dvalues, size = n, replace = TRUE)
# compute the mean
sample_mean <- mean(results)
# compute the sample sd
sample_sd <- sd(results)
# estimated standard error of the mean
mean_se <- sample_sd/sqrt(n)

# compute the t-statistic:
t <- (sample_mean - 3.5) / mean_se
t
```

```
## [1] 0.8193447
```

At this point you might wonder what the use of t is if the underlying data is not drawn from a normal distribution. In other words: what is the use of t if we cannot interpret it as a value that tells us how likely it is to observe this sample mean, given our null hypothesis? Well, actually we can. And here is where the magic of the CLT comes in: It turns out that there is a mathematical proof (i.e. the CLT) which states that the t-statistic itself can arbitrarily well be approximated by the standard normal distribution. This is true independent of the distribution of the underlying data in our sample! That is, if we have a large enough sample, we can simply compute the t-statistic and look up how likely it is to observe a value at least as large as t, given the null hypothesis is true (-> the p-value):

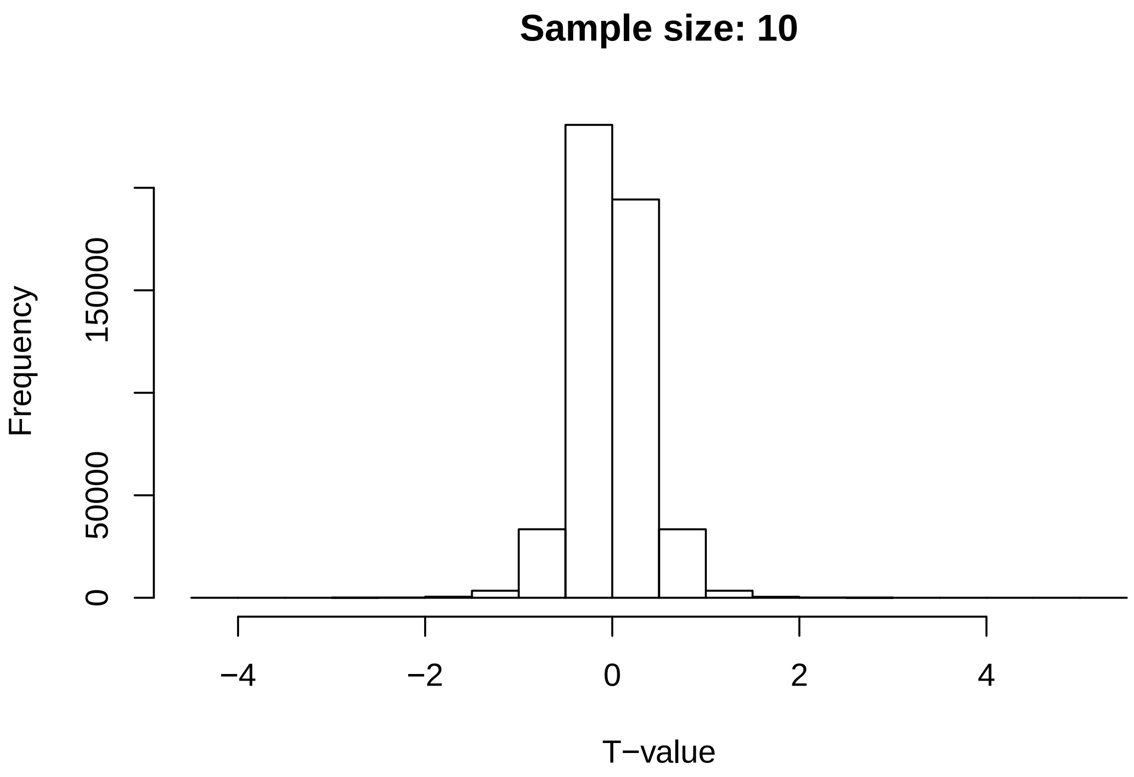
```r
# calculate the p-value associated with the t-value calculated above
2*pnorm(-abs(t))
```

```
## [1] 0.4125898
```

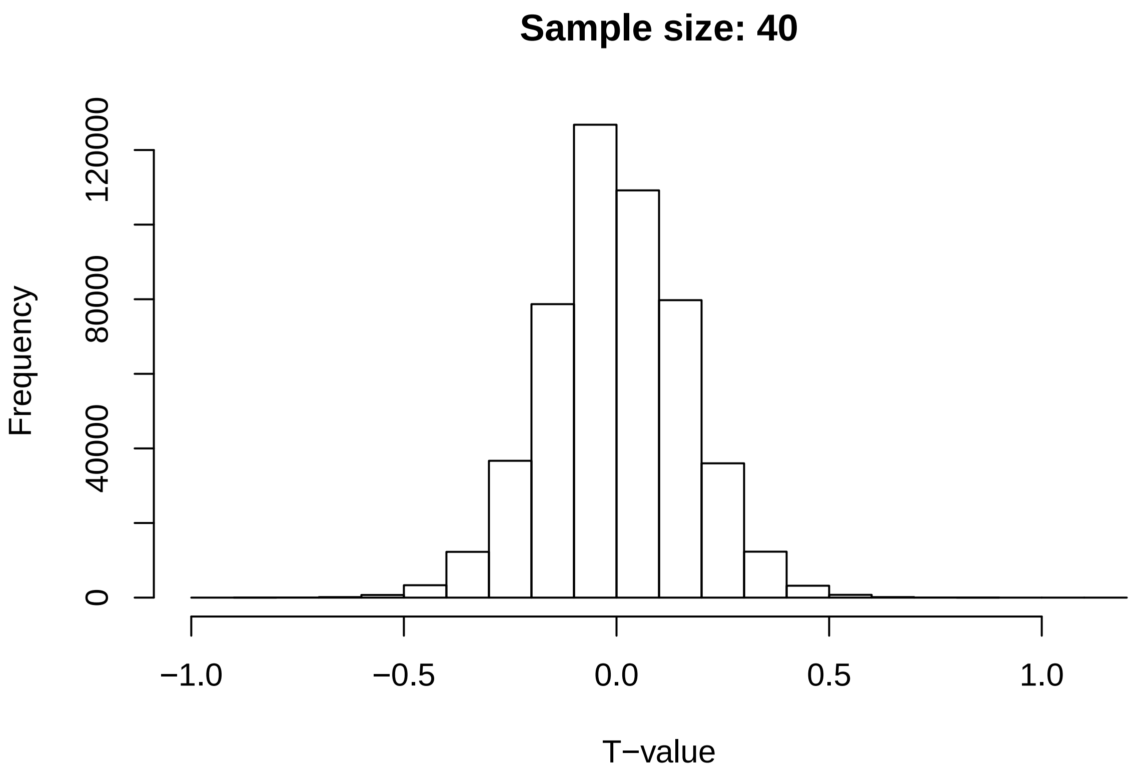
In that case we could not reject the null hypothesis of a fair die. However, as pointed out above, the t-statistic is only assymptotically (meaning with very large samples) normally distributed. We might not want to trust this hypothesis test too much because we were using a sample of only 100 observations.

Let's turn back to R in order to illustrate the CLT at work. Similar to the illustration of the LLN above, we will repeatedly compute the t-statistic of our dice play experiment and for each trial increase the number of observations.

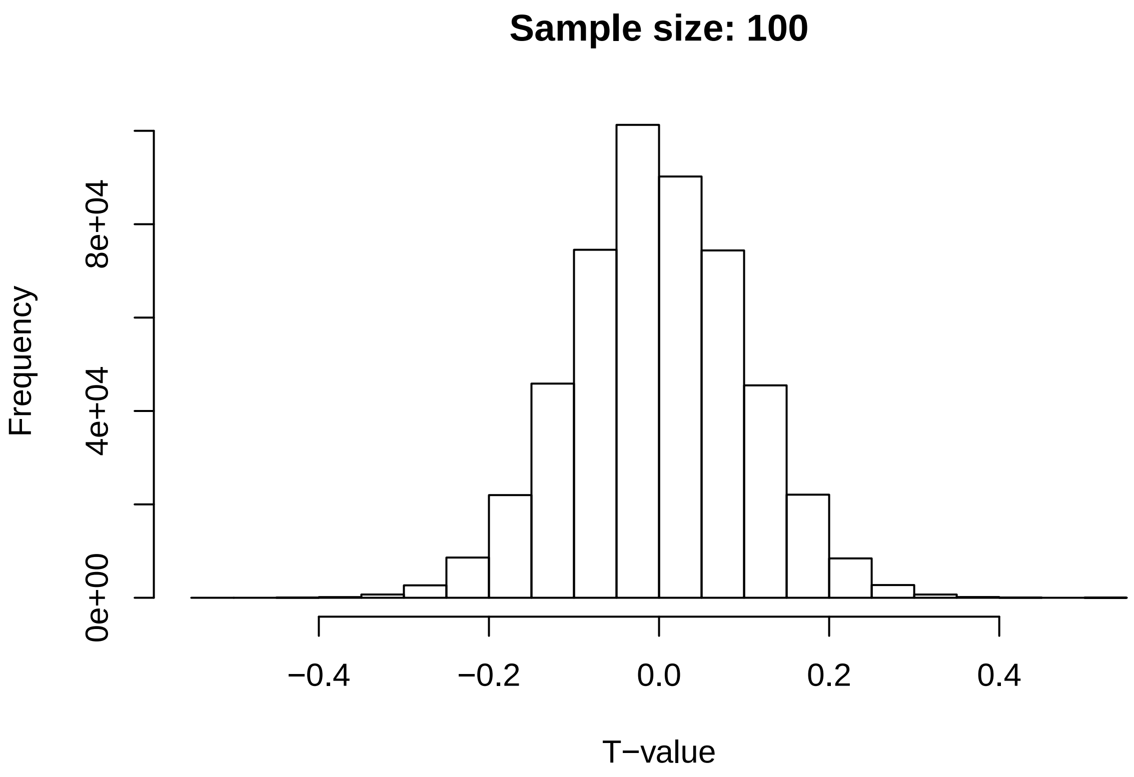
```r
# define the set of sample sizes
ns <- c(10, 40, 100)
# initiate an empty list to record the results
ts <- list()
length(ts) <- length(ns)
# iterate through each sample size: 'repeat the die experiment for each sample size'
for (i in 1:length(ns)) {

     samples.i <- sapply(1:500000, function(j) sample( x = dvalues, size = ns[i], replace = TRUE))
     ts[[i]] <- apply(samples.i, function(x) (mean(x) - 3.5) / sd(x), MARGIN = 2)
}

# visualize the result: plot the density for each sample size

# plot the density for each set of t values
hist(ts[[1]], main = "Sample size: 10", xlab = "T-value")
```



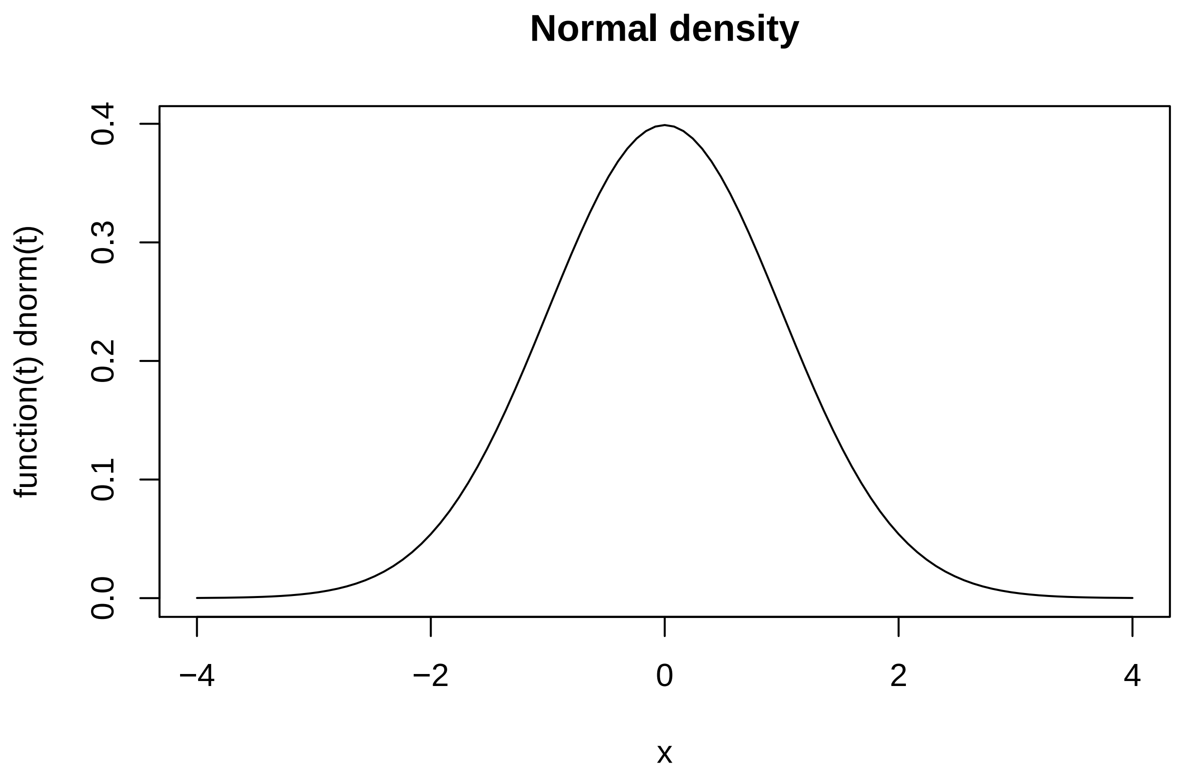
```r
hist(ts[[2]], main = "Sample size: 40", xlab = "T-value")
```



```r
hist(ts[[3]], main = "Sample size: 100", xlab = "T-value")
```



```r
# finally have a look at the actual standard normol distribution as a reference point
plot(function(t)dnorm(t), -4, 4, main = "Normal density")
```



Note how the histogram is getting closer to a normal distribution with increasing sample size.

2 Basic Statistics and Econometrics with R[1]

In the previous part we have learned how to implement basic statistics functions in R, as well as notices that for most of these statistics R already has a build-in function.

For example, mean and median:

```r
# initiate sample
a <- c(10,22,33, 22, 40)
names(a) <- c("Andy", "Betty", "Claire", "Daniel", "Eva")

# compute the mean
mean(a)
```

```
## [1] 25.4
```

```r
# compute the median
median(a)
```

```
## [1] 22
```

…as well as different measures of variability,

```r
range(a)
```

```
## [1] 10 40
```

```r
var(a)
```

```
## [1] 132.8
```

```r
sd(a)
```

```
## [1] 11.52389
```

and test statistics such as the t-test.

```r
# define size of sample
n <- 100
# draw the random sample from a normal distribution with mean 10 and sd 2
sample <- rnorm(n, mean = 10, sd = 2)

# Test H0: mean of population = 10
t.test(sample, mu = 10)
```

```
##
##  One Sample t-test
##
## data:  sample
## t = -0.66544, df = 99, p-value = 0.5073
## alternative hypothesis: true mean is not equal to 10
## 95 percent confidence interval:
##   9.470124 10.263728
## sample estimates:
## mean of x
##  9.866926
```

[1] From Matter (2018)

2.1 Regression analysis with R

Finally, R and additional R packages offer functions for almost all models and tests in modern applied econometrics. While the syntax and return values of these functions are note 100% consistent (due to the fact that many different authors contributed some of these functions), applying them is usually quite similar. Functions for econometric models typically have the two main parameters `data` (the dataset to use in the estimation, in `data.frame`-format) and `formula` (a symbolic description of the model to be fitted, following a defined convention). Typically, the following three steps are involved when running regressions in R (assuming the data is already loaded and prepared):

1. In `formula`, we define the model we want to fit as an object of class 'formula'. For example, we want to regress `Examination` on `Education` with the `swiss`-dataset used above.

```
model1 <- Examination~Education
```

2. Fit the model with the respective estimator. Here we use `lm()` the R workhorse for linear models to fit the model with OLS:

```
fit1 <- lm(formula = model1, data = swiss)
```

3. Get summary statistics with the generic `summary()`. In the case of an object of class `lm` (the output of `lm()`), this amounts to the typical coefficient t-statistics an p-values shown in regression tables:

```
summary(fit1)
```

```
##
## Call:
## lm(formula = model1, data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.9322  -4.7633  -0.1838   3.8907  12.4983
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.12748    1.28589   7.876 5.23e-10 ***
## Education    0.57947    0.08852   6.546 4.81e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.773 on 45 degrees of freedom
## Multiple R-squared:  0.4878, Adjusted R-squared:  0.4764
## F-statistic: 42.85 on 1 and 45 DF,  p-value: 4.811e-08
```

2.2 Linear model

Putting the pieces together, we can write a simple script to analyse the `swiss`-dataset (in spirit of the data visualization above):

```
# load data
data(swiss)

# linear regression with one variable
# estimate coefficients
model_fit <- lm(Examination~Education, data = swiss)
# t-tests of coefficients (and additional statistics)
summary(model_fit)
```

```
##
## Call:
## lm(formula = Examination ~ Education, data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.9322  -4.7633  -0.1838   3.8907  12.4983
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.12748    1.28589   7.876 5.23e-10 ***
## Education    0.57947    0.08852   6.546 4.81e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.773 on 45 degrees of freedom
## Multiple R-squared:  0.4878, Adjusted R-squared:  0.4764
## F-statistic: 42.85 on 1 and 45 DF,  p-value: 4.811e-08
```

Specifying the linear model differently can be done by simply changing the formula of the model. For example, we might want to control for the share of agricultural occupation.

```r
# multiple linear regression
# estimate coefficients
model_fit2 <- lm(Examination~Education + Catholic + Agriculture, data = swiss)
# t-tests of coefficients (and additional statistics)
summary(model_fit2)
```

```
##
## Call:
## lm(formula = Examination ~ Education + Catholic + Agriculture,
##     data = swiss)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.3146 -2.8308 -0.3486  3.3494  7.4172
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.53695    2.63707   7.029 1.17e-08 ***
## Education    0.42418    0.08678   4.888 1.46e-05 ***
## Catholic    -0.07976    0.01679  -4.750 2.29e-05 ***
## Agriculture -0.06757    0.03963  -1.705   0.0954 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.304 on 43 degrees of freedom
## Multiple R-squared:  0.7279, Adjusted R-squared:  0.7089
## F-statistic: 38.35 on 3 and 43 DF,  p-value: 3.212e-12
```

2.3 Other econometric models

Various additional packages provide all kind of functions to estimate diverse economietric models. Many generalized linear models can be fitted with the `glm()`-function (`formula` and `data` used as in `lm()`):

- Probit model: `glm(..., family = binomial(link = "probit"))`.
- Logit model: `glm(..., family = "binomial")`.
- Poisson regression (count data): `glm(..., family="poisson")`.

Other frequently used packages for econometric regression analysis: - Panel data econometrics: package `plm`. - Time-series econometrics: package `tseries`. - Basic IV regression: `ivreg()` in package `AER`. - Generalized methods of moments: package `gmm`. - Generalized additive models: package `gam`. - Lasso regression: package `glmnet`. - Survival analysis: package `survival`.

See also the CRAN Econometrics Task View

2.4 Regression tables

Economists typically present regression results in detailed regression tables, with regression coefficients in rows and model specifications in columns. The package `stargazer` provides some functions to generate such tables very easily for outputs of the most common regression functions (`lm()`, `glm()`, etc.):

```r
# load packages
library(stargazer)

# print regression results as text
stargazer(model_fit, model_fit2, type = "text")
```

```
##
## =================================================================
##                                  Dependent variable:
##                      --------------------------------------------
##                                      Examination
##                               (1)                     (2)
## -----------------------------------------------------------------
## Education                  0.579***                0.424***
##                            (0.089)                 (0.087)
##
## Catholic                                          -0.080***
##                                                    (0.017)
##
## Agriculture                                        -0.068*
##                                                    (0.040)
##
## Constant                  10.127***               18.537***
##                            (1.286)                 (2.637)
##
## -----------------------------------------------------------------
## Observations                  47                      47
## R2                          0.488                   0.728
## Adjusted R2                 0.476                   0.709
## Residual Std. Error    5.773 (df = 45)         4.304 (df = 43)
## F Statistic         42.854*** (df = 1; 45) 38.347*** (df = 3; 43)
## =================================================================
## Note:                                 *p<0.1; **p<0.05; ***p<0.01
```

Alternatively the regression table can be directly generated as LaTeX table:

```
# load packages
library(stargazer)

# print regression results as text
stargazer(model_fit, model_fit2, type = "latex")
```

```
##
## % Table created by stargazer v.5.2.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harva
## % Date and time: Thu, Sep 06, 2018 - 08:05:54
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lcc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{2}{c}{\textit{Dependent variable:}} \\
## \cline{2-3}
## \\[-1.8ex] & \multicolumn{2}{c}{Examination} \\
## \\[-1.8ex] & (1) & (2)\\
## \hline \\[-1.8ex]
##  Education & 0.579$^{***}$ & 0.424$^{***}$ \\
##   & (0.089) & (0.087) \\
##   & & \\
##  Catholic &  & $-$0.080$^{***}$ \\
##   &  & (0.017) \\
##   & & \\
##  Agriculture &  & $-$0.068$^{*}$ \\
##   &  & (0.040) \\
##   & & \\
##  Constant & 10.127$^{***}$ & 18.537$^{***}$ \\
##   & (1.286) & (2.637) \\
##   & & \\
## \hline \\[-1.8ex]
## Observations & 47 & 47 \\
## R$^{2}$ & 0.488 & 0.728 \\
## Adjusted R$^{2}$ & 0.476 & 0.709 \\
## Residual Std. Error & 5.773 (df = 45) & 4.304 (df = 43) \\
## F Statistic & 42.854$^{***}$ (df = 1; 45) & 38.347$^{***}$ (df = 3; 43) \\
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{2}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
## \end{tabular}
## \end{table}
```

References

Matter, Ulrich. 2018. "A Brief Introduction to Programming with R." Lecture notes. St. Gallen: University of St. Gallen.