EDDA: Assignment 0

This assignment consists of getting acquainted with the basics of `R` *and* `RStudio`*, and solving several exercises. You do not have to report on this assignment.*

1 Introduction to `R` and `RStudio`

You can download and install a version of `R` on your computer from web-page `http://www.r-project.org`. It is recommended to use `RStudio`, a useful IDE for `R`, downloadable from `www.rstudio.com`. Start up `RStudio`, and choose `File` → `New` → `R Script`. Now you should have 4 windows:

- top left *script window* for typing, editing, saving, executing R-commands;
- bottom left *console window* for direct typing and executing commands;
- top right *workspace window* for the introduced variables, imported datasets, history;
- bottom right *plot/help window* for graphics (view, save, etc) and help-function.

The sign '>' at the beginning of a line in the console window is the `R` *prompt*. If you type a command that is not syntactically complete and type *enter*, then `R` will show a + to ask for continuation of the command. You can type commands over several lines by grouping them within braces { and }. You can repeat previous commands by hitting the ↑ button several times in the `R` console window.

`R` can be used interactively in the console window. When you type your commands directly in the console window, the typed commands are not saved, although you can recover them in *History* window. You may prefer to work from the script window. This can be used as text editor in which a sequence of commands, or even a full program, can be written, and run. You can execute lines (or selected text) in the script window by pressing the keyboard keys `Ctrl+Enter` (on Mac: `Cmd+Enter`), or `Run` button in the script window. The contents of the script can be saved (choose `File -> Save`), and reopened later. The default file name extension for script files is `.R`, but script files are just plain text files and can be edited also in your favorite text editor. Choose a default directory in `Session -> Set Working Directory`. `R` will look in that directory for data files and save files to that directory.

To install an `R` package which is not yet in your `R` library, in `RStudio` choose `Tools` menu and select `Install Packages`, you must be connected to the Internet. From the next window that opens, choose a CRAN mirror, type the desired package and press `Install`, R will take care of the rest.

1.1 Simple object manipulation, vectors and matrices

```
> x=3
```

The object `x` is created and *assigned* the value 3 using the assignment symbol `=`. (`R` fanatics use `<-` instead.)

```
> x
> x==3; x==4
```

The double `==` asks whether left and right side are equal. Two commands separated by the semicolon ";" are executed in the batch mode.

```
> y=c(x,6)
```

The concatenation function `c` concatenates the objects `x` and 6, and the resulting object is a vector called `y`.

```
> y
> 1:10
> seq(0,10,length=40)
> rep(2,5)
> rep(1:3,5)
> y=c(1.4,1.6,7.5,3.1,9.5,1.0,3.8,2)
> length(y)
```

Vector `y` has a `length`, the number of its elements. The replicate function `rep` creates a vector consisting of repeated sub-vectors. The colon operator `1:10` makes a vector of consecutive integers. We can make the sequence run the other way `10:1`. Another version of the sequence function `sec` is to give a starting value, ending value, and an increment value, e.g., `seq(0,10,by=0.1)`.

```
> y[2]
> y[c(1,4,5)]
> y[-1]
> y<4
> y[y<4]
```

If we want to reference an element in the vector, we use square brackets as in `y[2]`, giving us the second element of vector `y`.. If we would like the locations within the vector `y` that satisfy some condition, such as those elements which are greater than 2, we would use the command `which(y>0)`.

The `matrix` function can be used to create a matrix in `R`. We need to provide a vector containing the elements of the matrix, and specify either the number of rows or the number of columns of the matrix. This number should divide evenly into the length of the vector, or we will get a warning.

```
> matrix(1:6,nrow=2); matrix(1:6,ncol=3)
```

Note that `R` places the row numbers on the left and the column numbers on top. Also note that `R` filled in the matrix column-by-column. If we prefer to fill in the matrix row-by-row, we must activate the byrow setting, e.g.,

```
> M=matrix(1:6,ncol=3,byrow=TRUE); M
```

In place of the vector `1:6`, any vector containing the desired elements of the matrix can be used. We can also create a matrix by binding vectors or matrices together either row-wise or column-wise using the `rbind` or the `cbind` functions, respectively. To find the dimensions of a matrix `M` (i.e., the number of rows and the number of columns):

```
> dim(M); nrow(M); ncol(M)
```

To refer to the element (2,3), column 3, row 2, columns 1 and 2 of row 1 of `M`,

```
> M[2,3]; M[,3]; M[2,]; M[1,1:2]
```

To obtain the transpose of a matrix, we use the transpose function `t(M)`. To add or subtract two matrices (or vectors) which have the same dimensions, use the plus and minus symbols. To multiply two matrices with compatible dimensions (i.e., the number of columns of the first matrix equals the number of rows of the second matrix), use the matrix multiplication operator %*%

```
> t(M); A+B; A-B; A%*%B
```

If we just use the multiplication operator *, `R` will multiply the corresponding elements of the two matrices, provided they have the same dimensions. Likewise, to multiply two vectors to get their scalar product, we use the same operator:

```
> a%*%b
```

Technically, we should use the transpose of vector `a`, but `R` will transpose `a` for us rather than giving us an error message.

To find the determinant of a square matrix `M`, use the determinant function `det(M)`. To obtain the inverse $\texttt{M}^{-1}$ of an invertible square matrix `M`, we use the `solve` function `solve(M)`. If the matrix is singular (not invertible), or almost singular, or is not square, we get an error message.

1.2 Plotting and getting help

```
> help(boxplot); boxplot(y)
> z=c(2.7,4.3,9.5,1.4,5.5,7.2); boxplot(y,z)
> x=seq(0,10,length=30); x
> y=sin(x); y
```

A function, such as the sine, applied to a vector creates a vector obtained by applying the function to every coordinate.

```
> plot(x,sin(x))
> plot(x,sin(x),type="l",xlab="x",ylab="sin(x)")
```

So `plot(x,y)` just plots the pairs of coordinates `(x[i],y[i])` as points in a coordinate plane. The `type="l"` command tells `R` to connect the points by line segments. The `type`-option can have other values as well (see help documentation on `plot`) to get different plots. For smooth curves you need a fine grid of points. To restrict the `x` and `y` coordinates ranges in plots, specify the parameters `xlim` and `ylim` in the `plot` command.

```
> x=seq(0,10,length=300)
> plot(x,sin(x),type="l")
```

Note that whenever you make a new plot the old one will disappear (this can be changed), so save it if you don?t want to lose it. To save a plot you can choose `File -> Save as ->` after selecting the plot window.

You can add some additional settings (title, informative axis labels, etc.) to the `plot` command:

```
> plot(x,y,main="Plot",xlab="x",ylab="y",pch=20,cex=1.5,col="blue")
```

Option `main=` sets the title for the plot, `xlab=` and `ylab=` specify the labels for the horizontal and vertical axes, respectively. The number after `pch=` is a code for the symbol to use for the points. You can try numbers from 1 to 25. You can also use any symbol on your keyboard for the points, using quotes. The setting `cex=1.5` makes the plotting characters 1.5 times their usual size. You can set the color of the plotting characters by using the setting `col=`. To see the many available options for colors, type the command `colors()`. To add additional points to the plot, use the `points` command:

```
> points(x,cos(x),pch=15,col="red")
```

If you want to add a vertical or horizontal line to the plot, use the `abline` command. For example,

```
> abline(v=1,lty=2,lwd=2)
```

will place a vertical line through the point 1 on the horizontal axis. The setting `lty=2` changes it from a solid line to a dashed line, and the setting `lwd=2` doubles the line width. We can also change its color using the `col=` setting. Use `h=` instead of `v=` to make it a horizontal line. You might need to change the plotting ranges to make your added lines visible. To add a line with, for example, an intercept of 50 and a slope of 5, use the command

```
> abline(a=50, b=5)
```

To add a plot of a function to the existing plot over a certain range, first create a vector consisting of many points (the more the better) in this range and compute the vector of their function values:

```
> z=seq(from=0,to=1.2,by=0.01); f=212.68*x/(0.06412+x)
```

Now we can add the curve to the plot, using the `lines` command:

```
> lines(z,f,col="blue",lwd=1.5)
```

1.3 Arithmetic and build-in R functions

Try some arithmetic operations.

```
> 2*3-7; 2^3
> y^3
```

Mathematical operations are applied coordinate-wise to vectors.

```
> 4*(3:9)
> mean(y)
```

The last command computes the sample mean of the elements of the object `y`.

```
> var(y)
> sum((y-mean(y))^2)/(length(y)-1)
```

To understand this command, argue 'from the inside'. `y-mean(y)` is the vector where from each component of `y` the mean of the whole vector is subtracted. Also the square is taken component-wise, so `(y-mean(y))^2` is a vector of the same length as `y`. The function `sum` sums the elements of this vector. Dividing by `length(y)-1` gives the sample variance of the elements of `y`.

```
> sort(y); sample(y)
```

The elements of `y` are sorted in ascending and random order, respectively.

```
> median(y)
```

The median of a sample is any number such that half of the data are to the left (or equal to) this value and half of the data are to the right (or equal) to this value.

1.4 Simulating data

A *random number generator* simulates samples from standard distributions (normal, uniform, chisquare, binomial, etc.).

```
> x=rnorm(100)
```

The object `x` contains a randomly generated sample of size 100 from the standard normal distribution. Every time you re-evaluate this command, `x` will have another value.

```
> hist(x,prob=TRUE)
```

A histogram of the data contained in `x`.

```
> x=rbinom(1,30,0.5)
```

A single draw from the binomial distribution with parameters 30 and 0.5, i.e. the number of heads when you (fairly) throw a (fair) coin 30 times.

1.5 Loops

As we have seen calculations in `R` are "vectorized": a procedure is automatically applied to every coordinate of a vector. If desired, an explicit loop can be programmed with a `for` statement.

```
> for (i in 1:10) print(i)
```

Not very exciting. A more typical use is as follows.

```
> m=numeric(500)
> for (i in 1:500) m[i]=mean(rexp(25))
> hist(m)
```

You have first set up an empty numerical vector of length 500, and next filled this with the averages of 500 independent random samples of size 25 from the exponential distribution (0.5 times a chisquare with 2 degrees of freedom). The histogram looks normal. (As it should, by the Central Limit Theorem.)

```
> hist(m,prob=TRUE)
> u=seq(min(m),max(m),length=100)
> lines(u,dnorm(u,mean(m),sqrt(var(m))))
```

These commands rescale the vertical axis of the histogram so that the area is 1, and add a best fitting normal density curve.

```
> hist(rexp(500))
```

This is equivalent to what you get if you replace 25 by 1 in the preceding. This histograms does not look normal!

1.6 Reading data from files

Usually you will want to import data from a file corresponding to data associated with a homework problem. Such a file will usually end with the extension `.txt`. Often data files for this course will consist of columns of numbers. If each column has a title on top describing what the data in the column represents (e.g., `age`, `weight`, `income`, etc.), we will say that the file has a `header`. For instance, the first 4 lines of the file `mortality.txt` are:

```
   id state teen mort
1   1    AL 17.4 13.3
2   2    AR 19.0 10.3
3   3    AZ 13.8  9.4
```

The easiest way to import the data into `R` and have it readily available for the current and future sessions is to first save the data file into the R working directory. The function `read.table` is used to read data from text files:

```
> data=read.table(file="mortality.txt",header=TRUE)
```

The `read.table` command creates the `data.frame` with the name `data`, which has 48 rows and 4 columns. If you want to load the data file from some other directory, you need to type the full path name in the `read.table` command. Note that, in the absence of a header, the columns will be named `V1`, `V2`, etc., and the rows will be numbered. Be careful: if you put `header=TRUE` when there is no header, then R will think your first row of data is the header.

To view the data set:

```
> data   # to view the data set
> head(data) # to view first rows (useful if data is too large)
> dim(data) # the numbers of rows and columns in your data
> data$teen
```

The dollar sign followed by a name is used to select the column with this name. An alternative is to use *subscripts*.

```
> data[,3]
> data[1:5,3]
```

You may want to rename the columns in your R data table so that each column has an appropriate descriptive title. To do so, use the R command

```
> names(data) <- c("name1","name2","name3","name4")
```

1.7 Data types

Basic data types in `R` include `numeric` for numbers, `character` for letters, and `factor` for categories of nominal variables. The latter type is used for variables that encode class membership of individuals.

```
> labels=1:10
> labels
> sum(labels)
```

The vector `1:10` consists of the numbers 1 to 10 and labels is therefore of type `numeric`. If we would like to use them as labels of a categorical variable, for instance for subjects 1 to 10, then it is better to coerce it to type `factor`.

```
> labels=as.factor(labels)
> labels
> sum(labels)
```

Although the difference is not visible, internally `labels` is now of type `factor` and numerical functions do not apply to it.

1.8 Histograms and QQ-plots

Histograms and QQ-plots are methods to gain insight in the "distribution" of data or of a population. We shall often be interested in histograms and QQ-plots of random samples taken from a given population. Many statistical procedures assume that data follows a normal distribution. Histograms and QQ-plots give an indication of the plausibility of this assumption for a given data set.

Histograms are excellent, but tend to be unstable if applied to small samples. For instance, a histogram of a small number of randomly chosen individuals from a normal population may well look asymmetric or have outliers. QQ-plots are a better method to verify whether data can be assumed to be sampled from a given distribution.

Recall that the *α-quantile* of a population distribution is the number ξ_α such that a fraction α of the population is smaller than ξ_α.

```
> qnorm(0.95); qnorm(0.975)
```

Two quantiles of the normal distribution often used in testing theory.

Given a random sample $X_1, \ldots, X_n$ of size n from a population, we may expect the ith smallest value in the sample to be close to the i/nth quantile of the population distribution, since i/n is the fraction of sample values smaller than the ith smallest. A *QQ-plot* is essentially a plot of the ordered sample values

$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$$

versus the population quantiles

$$\xi_{1/n} \leq \xi_{2/n} \leq \cdots \leq \xi_{n/n}.$$

The only difference is that one uses the quantiles at $i/(n+1)$, or another slight adaptation, rather than at i/n.

A QQ-plot of a sample against its own distribution will show a straight line through the origin with slope 1, apart from some random aberrations. If the sample has been shifted to a different mean value and scaled to a different variance (e.g. a sample that is normal with some mean and variance compared to the standard normal distribution), then the QQ-plot will still show a straight line, but with a different intercept and slope.

We can make a QQ-plot relative to any population distribution, but here we shall plot against the standard normal distribution only, with the function `qqnorm`. Thus a plot using `qqnorm` of a sample from a normal distribution will show approximately a straight line, and a deviation from a line indicates that the sample was not taken from a normal population.

```
> x=rnorm(30)
> par(mfrow=c(1,2))
> hist(x)
> qqnorm(x)
```

A histogram and QQ-plot of a sample of size 30 from the normal distribution. The QQ-plot should look linear. (The command `par(mfrow=c(1,2))` instructs `R` to place the two plots in 1 row of 2 columns. It remains in force until `mfrow` is changed, for instance back to normal with `par(mfrow=c(1,1))`.)

```
> hist(10*x+3)
> qqnorm(10*x+3)
```

What has changed?

```
> x=rnorm(10)
> hist(x)
> qqnorm(x)
```

A histogram and QQ-plot of a sample of size 10 from the normal distribution. The histogram looks bad; the QQ-plot better.

```
> x=runif(100)
> hist(x)
> qqnorm(x)
```

A plot of the ordered values of a random sample from the uniform distribution relative to the quantiles of the normal distribution. (Sampling from the uniform distribution is equivalent to choosing an "arbitrary" point from the numbers between 0 and 1. This is independently repeated 100 times.)

```
> x=rchisq(30,5)
> hist(x)
> qqnorm(x)
```

The function `rchisq` samples from yet another population distribution, the *chisquare distribution*, in this case with 5 *degrees of freedom*. It suffices to know that it is far from normal, as you can see.

1.9 Two-sample t-test

Now we simulate possible outcomes of the *two-sample t-test* for comparing, say, heights of men and women. We use simulated data, as follows.

```
> n=30; m=30; mu=180; nu=175; sd=10
> x=rnorm(n,mu,sd); y=rnorm(m,nu,sd)
> t.test(x,y,var.equal=TRUE)
> t.test(x,y,var.equal=TRUE)[[3]]
```

The first 5 lines set up the parameters of the problem: the numbers of men and women sampled from the populations of all men and women (`n` and `m`), the means of the populations of all men and women (`mu` and `nu`), and the standard deviations of these populations (`sd`; for simplicity we take these to be equal for men and women). After fixing these parameters we sample (independently) from the populations and assign the heights of the sampled men and women to the vectors `x` and `y`, respectively. Finally, we perform the t-test. The `t.test` command produces a small report on the test. This includes the p-value for testing the null hypothesis that the population means of the two populations are equal. It also gives a confidence interval (= interval estimate) for the difference of the population means. The report is actually an R-object of type `list`, with the p-value as its third component. The latter number is extracted in the last line.

We now study the p-value and the *power function* of the t-test. The power function gives, for every given set of parameters (n, m, μ, ν, sd), the probability that the t-test rejects the null hypothesis. We can approximate it by repeatedly simulating data and computing the fraction of times that the t-test rejects the null hypothesis.

```
> n=m=30; mu=180; nu=175; sd=10; B=1000; p=numeric(B)
> for (b in 1:B) {x=rnorm(n,mu,sd); y=rnorm(m,nu,sd)
+  p[b]=t.test(x,y,var.equal=TRUE)[[3]]}
> power=mean(p<0.05)
```

This script assumes that we reject the null hypothesis if the p-value is smaller than 0.05. Thus the test is performed at significance level 5%.

2 Practice R-session

We complete the introduction to R by presenting a practice R-session that summarizes some common basic R commands. The right column contains some explanation of the corresponding command in the left column. Type these commands directly in the console window and see what you get. At the end of the session, navigate through your created plots, using the arrow buttons in the top line of the plot window (bottom right), and use the Export button to save plots as separate image files.

`> x=1:20`	make a vector `x` with values 1, 2,. . . , 20,
`> x`	print value of `x` on the screen,
`> m=matrix(x,4,5,byrow=T)`	a matrix `m` with 4 rows and 5 columns with the values of x ordered row-wise,
`> m`	print matrix `m` on the screen,
`> m[2,3]`	print element (2,3) of `m` on the screen,
`> m[2,]`	print all elements of 2^{nd} row of `m`,
`> m[,3]`	print all elements of 3^{rd} column of `m`,
`> y=sample(1:100,20)`	generate random sample of size 20 from the numbers $1, 2, 3, \ldots, 100$,
`> z=x+y`	the sum of `x` and `y` coordinate-wise,
`> y=x+2*y`	transform `y` coordinate-wise,
`> cbind(x,y)`	form a matrix with columns `x` en `y`,
`> plot(x,y)`	plot `y` against `x`,
`> abline(100,2)`	add the line `y=100+2*x` to the last plot,
`> x=rnorm(50,0,sqrt(2))`	generate a sample of size 50 from normal distribution with $\mu = 0$, $\sigma = \sqrt{2}$,
`> y=rnorm(50)`	idem for standard normal distribution,
`> mean(x)`	sample mean of the values in `x`,
`> sd(x)`	sample standard deviation of the values in `x`,
`> var(x)`	sample variance of the values in `x`,
`> cor(x,y)`	sample correlation between `x` and `y`,
`> x[x<0]`	select negative elements in `x`,
`> sum(x<0)`	count number of negative elements in `x`,
`> hist(x,prob=T)`	plot a histogram of `x`,
`> help(hist)`	help info anout the function `hist`,
`> ?hist`	the same,
`> u=seq(-5,5,0.1)`	form sequence of points between -5 and 5 with step size 0.1,

`> v=dnorm(u,0,sqrt(2))`	compute normal density with $\mu = 0$ and $\sigma = \sqrt{2}$ in all points of vector `u`,
`> lines(u,v)`	add plot of computed normal density,
`> {hist(x,prob=T)`	plot of histogram (not executed yet),
`+ lines(u,v)}`	plus addition. command, execute both,
`> plot(ecdf(x))`	plot empirical distribut. function of `x`,
`> lines(u,pnorm(u,0,sqrt(2)))`	add true distribution function,
`> qqnorm(x)`	plot the normal QQ-plot of `x`,
`> data=rexp(25,rate=0.25)`	generate random sample of size 25 from exponential distribution with $\lambda = 0.25$.

3 Homework

Exercise 1. Normal distribution in R

In this exercise you will practice with the normal distribution in `R`.

a) Generate two samples of sizes 100 and 100000 from a standard normal distribution. Make histograms and compute the means and standard deviations of the samples.

b) For a standard normal distribution, compute the following 3 probabilities: that an arbitrary outcome is smaller than 2, that it is bigger then -0.5 and that it is between -1 and 2.

c) Can you verify the outcomes of b) using only the data from a)?

d) Repeat a and b for a normal distribution with `mean`=3 and `sd`=2. Find also the value such that 95% of the outcomes is smaller than that value.

e) Any normal variable $X \sim N(\mu, \sigma^2)$ can be generated from a standard normally distributed $Z \sim N(0,1)$ as $X = \mu + \sigma Z$. Generate in this way a sample of size 1000 from a normal distribution with $\mu = -10$ and $\sigma = 5$, and verify that the sample mean and standard deviation are close to the true values μ and σ.

Exercise 2. Various (not normal) distributions

Generate the following samples, and plot for each of them the histogram and the boxplot:

- sample of size 10000 from the lognormal distribution with $\mu = \sigma = 2$,
- sample of size 40 from the binomial distribution with $n = 50$ and $p = 0.25$,
- sample of size 60 from the uniform distribution on the interval [-2,3],
- sample of size 200 from the Poisson distribution with $\lambda = 9$.

Comment on the forms of the histograms.

Exercise 3. Summarizing data

It has been argued that many cases of infant mortality are caused by teenage mothers who, for various reasons, do not receive proper prenatal care. In the file `mortality.txt` data from the Statistical Abstract of the United States (1995) are listed on teenage birth rate per 1000 (`teen`) and the infant mortality rate per 1000 live births (`mort`) for the 48 contiguous states (`state`). You can load data sets using the `import Dataset` button in the workspace (top right) window, or using a command like `read.table("filename.txt",header=TRUE)`. The commands `getwd()` and `setwd()` are also useful in this context.

After having imported this data set, you can assess the different columns by their names. Type e.g. `mortality$teen` or `mortality$mort` to extract one of the last two columns. Alternatively, you can use `mortality[,3]` and `mortality[,4]` to extract these columns. Compute numerical summaries of the birth and mortality rates. You can use `summary`, `min`, `max`, `range`, `mean`, `median`, `sd`, `var`. Use the `help`-function to see what these commands produce.

Exercise 4. QQ-plots

Place the file `assign0.RData` in your working `R`-directory, run the R-command `load(file="assign0.RData")`.

The data vectors `x1`, `x2`, `x3` should be now available in your `R`-session. Make a histogram and a QQ-plot for each of them, and decide which ones could have been sampled from a normal distribution. Experiment by simulating some normal samples of similar sizes and looking at their QQ-plots, before you make up your mind!

Exercise 5. *P*-values of the *t*-test

We study the p-values of the two-sample t-test (see Section 1.9 of Assignment 0). Set `m=n=30`.

a) Set `mu=nu=180` and `sd=10`. As in Section 1.9, generate 1000 samples `x=rnorm(n,mu,sd)` and `y=rnorm(m,nu,sd)`, and record the 1000 p-values for testing H_0: `mu`=`nu`. Compute the fractions of p-values that are smaller than 5% and that are smaller than 10%. What is the distribution of the p-values (make a histogram)?

b) Set `mu=nu=180` and `sd=1`. Answer the same questions as in a).

c) Set `mu=180`, `nu=175` and `sd=6`. Answer the same questions as in a).

d) Explain your findings.