huber-intro-2

Welcome to the second module. Again, if you find any errors or if you have suggestions for improvement, please let me know via stephan.huber@hs-fresenius.de .

Before you start working, you should set your working directory to where all your data and script files are or should be stored. Within RStudio you can go to 'Session'> 'Set working directory', or you can type in setwd(YOURPATH). Please do this now.

```
setwd(getwd())
```

R is an interpreter that uses a command line based environment. This means that you have to type commands, rather than use the mouse and menus. This has many advantages. Foremost, it is easy to get a full transcript of everything you did and you can replicate your work easy.

As already mentioned, all commands in R are functions where arguments come (or do not come) in round brackets after the function name.

You can store your workflow in files, the so-called scripts. These scripts have typically file names with the extension, e.g., foo.R .

You can open an editor window to edit these files by clicking 'File' and 'New'. Try this. Under 'File' you also find the options 'Open file...', 'Save' and 'Save as'. Alternatively, just type CTRL+SHIFT+N.

You can run (send to the Console window) part of the code by selecting lines and pressing CTRL+ENTER or click 'Run' in the editor window. If you do not select anything, R will run the line your cursor is on.

You can always run the whole script with the console command source, so e.g. for the script in the file foo.R you type source('foo.R'). You can also click 'Run all' in the editor window or type CTRL+SHIFT+S to run the whole script at once.

Make a script called firstscript.R. Therefore, open the editor window with 'File' > 'New'. Type plot(rnorm(100)) in the script, save it as firstscript.R in the working directory. Then type source("firstscript.R") on the command line.

```
#source("firstscript.R")
```

Run your script again with source("firstscript.R"). The plot will change because new numbers are generated.

```
#source("firstscript.R")
```

Vectors were already introduced, but they can do more. Make a vector with numbers 1, 4, 6, 8, 10 and call it vec1.

```
vec1 <- c(1,4,6,8,10)
```

Elements in vectors can be addressed by standard [i] indexing. Select the 5th element of this vector by typing vec1[5].

```
vec1[5]
```

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

Replace the 3rd element with a new number by typing vec1[3]=12.

```
vec1[3] <- 12
```

Ask R what the new version is of vec1.

```
vec1
```

```
## [1]  1  4 12  8 10
```

You can also see the numbers of vec1 in the environment window. Make a new vector vec2 using the seq() (sequence) function by typing seq(from=0, to=1, by=0.25) and check its values in the environment window.

```
vec2 <-seq(from=0, to=1, by=0.25)
```

Type sum(vec1).

```
sum(vec1)
```

```
## [1] 35
```

The function sum sums up the elements within a vector, leading to one number (a scalar). Now use + to add the two vectors.

```
vec1+vec2
```

```
## [1]  1.00  4.25 12.50  8.75 11.00
```

If you add two vectors of the same length, the first elements of both vectors are summed, and the second elements, etc., leading to a new vector of length 5 (just like in regular vector calculus).

Matrices are nothing more than 2-dimensional vectors. To define a matrix, use the function matrix. Make a matrix with matrix(data=c(9,2,3,4,5,6),ncol=3) and call it mat.

```
mat<-matrix(data=c(9,2,3,4,5,6),ncol=3)
```

The third type of data structure treated here is the data frame. Time series are often ordered in data frames. A data frame is a matrix with names above the columns. This is nice, because you can call and use one of the columns without knowing in which position it is. Make a data frame with t = data.frame(x = c(11,12,14), y = c(19,20,21), z = c(10,9,7)).

```
t <- data.frame(x = c(11,12,14), y = c(19,20,21), z = c(10,9,7))
```

Ask R what t is.

```
t
```

```
##    x  y  z
## 1 11 19 10
## 2 12 20  9
## 3 14 21  7
```

The data frame is called t and the columns have the names x, y and z. You can select one column by typing t$z. Try this.

```
t$z
```

```
## [1] 10  9  7
```

Another option is to type t[[“z”]]. Try this as well.

```
t[["z"]]
```

```
## [1] 10  9  7
```

Compute the mean of column z in data frame t.

```
mean(t$z)
```

```
## [1] 8.666667
```

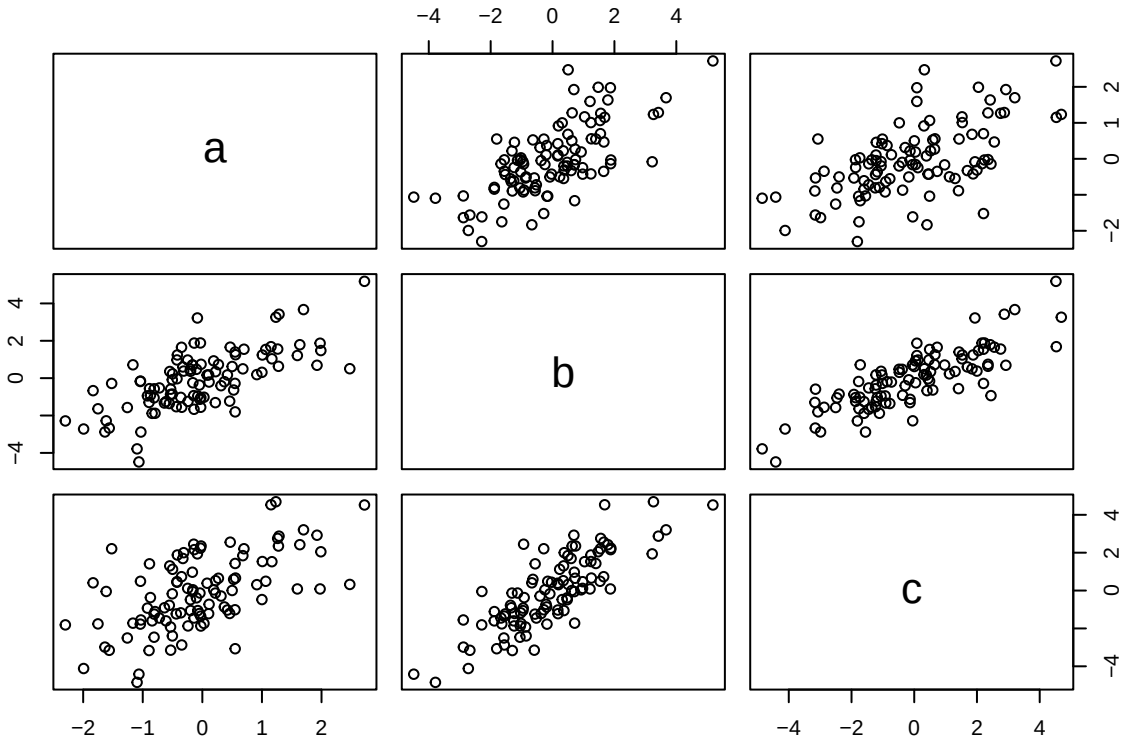
In the following question you will be asked to modify a script that will appear as soon as you move on from this question. When you have finished modifying the script, save your changes to the script and type submit() and the script will be evaluated. There will be some comments in the script that opens up. Be sure to read them!

Make a script file which constructs three random normal vectors of length 100. Call these vectors x1, x2 and x3. Make a data frame called t with three columns (called a, b and c) containing respectively x1, x1+x2 and x1+x2+x3. Call plot(t) for this data frame. Then, save it and type submit() on the command line.

```
# Text behind the #-sign is not evaluated as code by R.
# This is useful, because it allows you to add comments explaining what the script does.

# In this script, replace the ... with the appropriate commands.

x1 = rnorm(100)
x2 = rnorm(100)
x3 = rnorm(100)
t = data.frame(a=x1, b=x1+x2, c=x1+x2+x3)
plot(t)
```



Do you understand the results?

Another basic structure in R is a list. The main advantage of lists is that the `columns` (they are not really ordered in columns any more, but are more a collection of vectors) don't have to be of the same length, unlike matrices and data frames. Make this list L <- list(one=1, two=c(1,2), five=seq(0, 1, length=5)).

```
L <- list(one=1, two=c(1,2), five=seq(0, 1, length=5))
```

The list L has names and values. You can type L to see the contents.

```
L
```

```
## $one
## [1] 1
##
## $two
## [1] 1 2
##
## $five
## [1] 0.00 0.25 0.50 0.75 1.00
```

L also appeared in the environment window. To find out what's in the list, type names(L).

```
names(L)
```

```
## [1] "one"  "two"  "five"
```

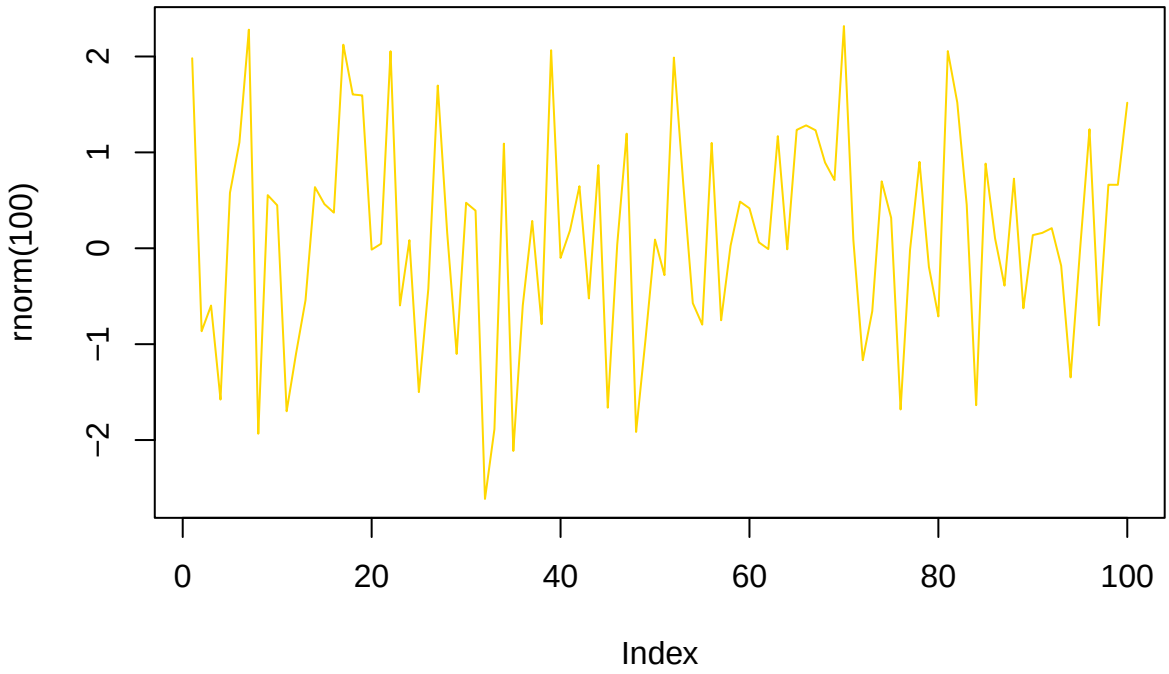
Add 10 to the column called five.

```
L$five + 10
```

```
## [1] 10.00 10.25 10.50 10.75 11.00
```

Plotting is an important statistical activity. So it should not come as a surprise that R has many plotting facilities. Type plot(rnorm(100), type="l", col="gold").

```
plot(rnorm(100), type="l", col="gold")
```



Hundred random numbers are plotted by connecting the points by lines in a gold color.

Another very simple example is the classical statistical histogram plot, generated by the simple command hist. Make a histogram of 100 random numbers.

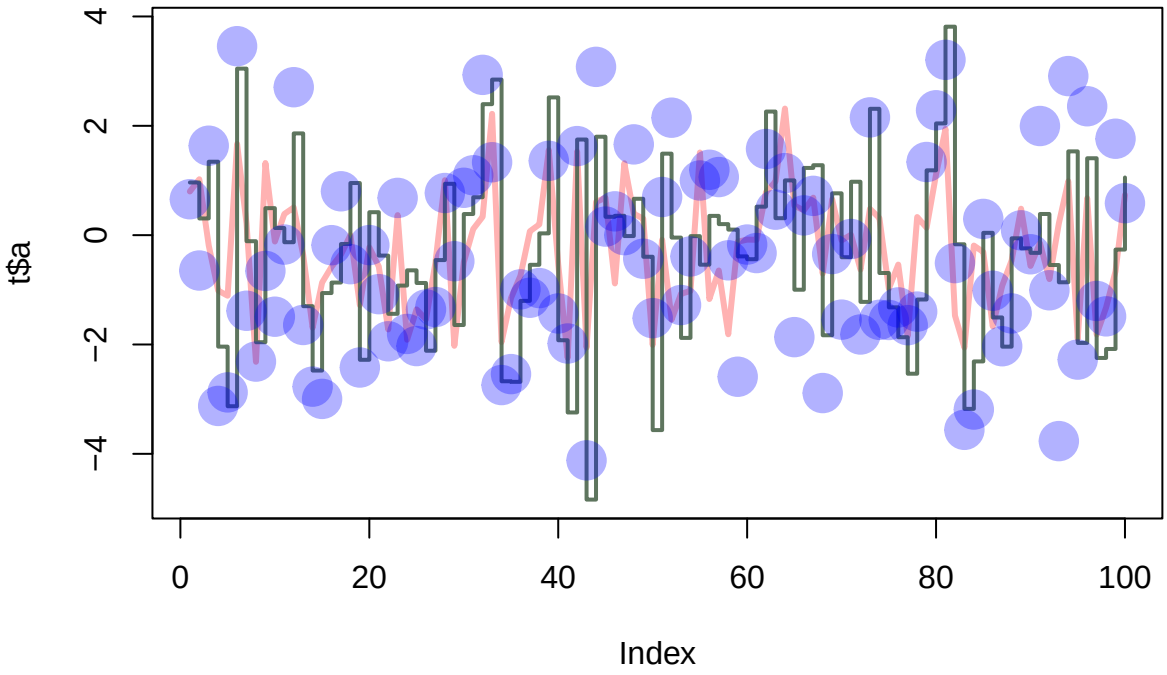
```
hist(rnorm(100))
```

The script that opens up is the same as the script you made before, but with more plotting commands. Type submit() on the command line to run it (you don't have to change anything yet).

```
# Text behind the #-sign is not evaluated as code by R.
# This is useful, because it allows you to add comments explaining what the script does.

# Make data frame
x1 = rnorm(100)
x2 = rnorm(100)
x3 = rnorm(100)
t = data.frame(a=x1, b=x1+x2, c=x1+x2+x3)

# Plot data frame
plot(t$a, type='l', ylim=range(t), lwd=3, col=rgb(1,0,0,0.3))
lines(t$b, type='s', lwd=2, col=rgb(0.3,0.4,0.3,0.9))
points(t$c, pch=20, cex=4, col=rgb(0,0,1,0.3))
```




```
# Note that with plot you get a new plot window while points and lines add to the previous plot.
```

Try to find out by experimenting what the meaning is of rgb, the last argument of rgb, lwd, pch, cex. Type play() on the command line to experiment. Modify lines 11, 12 and 13 of the script by putting your cursor there and pressing CTRL+ENTER. When you are finished, type nxt() and then ?par.

```
?par
```

You searched for par in the R help. This is a useful page to learn more about formatting plots. Google 'R color chart' for a pdf file with a wealth of color options.

To copy your plot to a document, go to the plots window, click the 'Export' button, choose the nicest width and height and click 'Copy' or 'Save'.

After having almost completed the second learning module, you are getting closer to become a nerd as you know...

...that everything in R is stored in objects (values, vectors, matrices, lists, or data frames),

...that you should always work in scripts and send code from scripts to the Console,

...that you can do it if you don't give up.

Please continue choosing another swril learning module.