No	Name	What it does?
1	**attach**	Attach your data frame to your working environment.
2	**boxplot**	Creates a boxplot.
3	**confint**	A metafor package function that gives you the confidence intervals of effect sizes.
4	**cumul**	A metafor package function that does cumulative meta analysis.
5	**data**	Attach a data frame to your working environment.
6	**detach**	Remove an attached data frame from your working environment.
7	**dir**	Gives the names of files and folders in your workind directory.
8	**escalc**	A metafor package function. Calculates effect size from various statistics.
9	**forest**	A metafor package that creates forest plots.
10	**funnel**	A metafor package that creates funnel plots.
11	**getwd**	Gives your location on your computer directory (folder) tree.
12	**hist**	Creates a histogram.
13	**install.packages**	Install packages from cran repositories.
14	**leave1out**	A metafor package that conducts a leave-one-out meta analysis.
15	**library**	Attaches a package to your workin environment. In order to use packages, you have to use this function.
16	**ls**	Lists all the objects that is in your working environment.
17	**order**	Orders a data frame
18	**predict**	
19	**radial**	A metafor function that creates a radial plot.
20	**read.table**	Read data from a file that is in csv kind of text format.
21	**read.xlsx**	An xlsx package function that reads data from Excel files.
22	**regtest**	A metafor package function
23	**rma**	A metafor package function
24	**rstudent**	A metafor package function
25	**sd**	Calculates the standard deviation of a vector.

No	Name	What it does?
26	**setwd**	With this you can determine your working directory.
27	**summary**	Gives a summary of most of the data formats within R.
28	**trimfill**	A metafor package function
29	**$ (dollar sign)**	Used to select a specific column or columns, or in other words variables from a data frame.

Command: `install.packages`

Function: This functions is used to install packages. A package is a collection of functions that you use, help files that you can learn about the functions, and also possible data set that can be used to apply functions.

Example:

```
> install.packages("metafor")
trying URL 'http://cran.mirror.garr.it/mirrors/CRAN/bin/windows/contrib/3.1/metafor_1.9-5.zip'
Content type 'application/zip' length 2027094 bytes (1.9 MB)
opened URL
downloaded 1.9 MB
package 'metafor' successfully unpacked and MD5 sums checked
The downloaded binary packages are in
        C:\Users\mehmet-odtu\AppData\Local\Temp\RtmpGWJ9yg\downloaded_packages
> install.packages('xlsx')
Installing package into 'C:/Users/mehmet-odtu/Documents/R/win-library/3.1'
(as 'lib' is unspecified)
trying URL 'http://cran.mirror.garr.it/mirrors/CRAN/bin/windows/contrib/3.1/xlsx_0.5.7.zip'
Content type 'application/zip' length 400944 bytes (391 KB)
opened URL
```

```
downloaded 391 KB


package 'xlsx' successfully unpacked and MD5 sums checked


The downloaded binary packages are in
        C:\Users\mehmet-
odtu\AppData\Local\Temp\RtmpmYyCVz\downloaded_packages
```

As you can see, the command given above installs the package metafor so that you can use it to conduct meta-analyses. Any package you can find in http://cran.r-project.org/ can be installed this way.

For more information about this command, type `?install.packages` and press `Enter`.

Command: `library`

Function: Library load installed packages, so that you can use them in your current R session.

Example:

```
> library(metafor)
Loading 'metafor' package (version 1.9-5). For an overview
and introduction to the package please type: help(metafor).
> library(xlsx)
Loading required package: rJava
Loading required package: xlsxjars
```

This commands loads the metafor package. After giving this command you can be able to use metafor package. Remember, installing a package is not the same thing as loading a package.

For more information about this command, type `?library` and press `Enter`.

Command: `read.xlsx`

Function: Imports content of an Excel file for you to use within R. As you can see from the previous command there is an Excel file named as `Devine.xlsx`. Within read.xlsx function you have to specify

the position of the tab your data resides in. `sheetIndex=1` argument of `read.xlsx` function. Or else, you can use `sheetName` argument, like `sheetName='data'`, or `sheetName='whatisyoursheetname'`.

You have to assign the result of the `read.xlsx` function's output to a variable in order to use, as shown below. read.xlsx function creates a data frame.

Example:

```
> newData <- read.xlsx('Devine.xlsx', sheetIndex=1)
> newData
   study     d         w          v pub
1      1  0.57 19.226568 0.05201136   D
2      2  0.42  8.320691 0.12018233   J
3      3  0.35  8.867126 0.11277611   J
4      4  0.08  4.996160 0.20015371   D
5      5 -0.15  4.986527 0.20054037   D
```

Here you can see the first five lines of newly created data set that is named as `newData`.

The name that you define is up to you. You can name it as new_data or new.data or NewData or New_Data. R is case sensitive so just remember the exact name that you give to your objects, and use it.

For more information about this command, type `?read.xlsx` and press `Enter`.

Command: `str`

Function: Using this function you can see the variables of the data frame that was created with the use of `read.xlsx()` function.

Example:

```
> newData <- read.xlsx('Devine.xlsx', sheetIndex=1)
> str(newData)
'data.frame':       54 obs. of  5 variables:
 $ study: num  1 2 3 4 5 6 7 8 9 10 ...
 $ d    : num  0.57 0.42 0.35 0.08 -0.15 -0.23 0.88 0.56 1.27 0.69 ...
 $ w    : num  19.23 8.32 8.87 5 4.99 ...
 $ v    : num  0.052 0.12 0.113 0.2 0.201 ...
 $ pub  : Factor w/ 2 levels "D","J": 1 2 2 1 1 1 2 2 2 2 ...
```

Command: `summary`

Function: This function give you minimum, maximimum, first, and third quantiles, and mean, median of the variables in your data set.

Example:

```
> summary(newData)
     study             d                  w                 v           pub
 Min.   : 1.00   Min.   :-0.2300   Min.   : 3.207   Min.   :0.01294   D:21
 1st Qu.:14.25   1st Qu.: 0.2550   1st Qu.: 4.989   1st Qu.:0.06816   J:33
 Median :27.50   Median : 0.4600   Median : 8.303   Median :0.12044
 Mean   :27.50   Mean   : 0.4963   Mean   :12.123   Mean   :0.14087
 3rd Qu.:40.75   3rd Qu.: 0.6975   3rd Qu.:14.672   3rd Qu.:0.20044
 Max.   :54.00   Max.   : 1.3800   Max.   :77.300   Max.   :0.31178
```

If you want to select only some of the variables from a data frame to summarize, you can simply use column operator. You can see an instance of this if you inspect the code listing below.

```
> summary(newData[, 2:5])
       d                 w                 v           pub
 Min.   :-0.2300   Min.   : 3.207   Min.   :0.01294   D:21
 1st Qu.: 0.2550   1st Qu.: 4.989   1st Qu.:0.06816   J:33
 Median : 0.4600   Median : 8.303   Median :0.12044
 Mean   : 0.4963   Mean   :12.123   Mean   :0.14087
 3rd Qu.: 0.6975   3rd Qu.:14.672   3rd Qu.:0.20044
 Max.   : 1.3800   Max.   :77.300   Max.   :0.31178
```

Command: `names`

Function: Similar to `str()` function, but gives a shorter output. Using this function you can see the variables of the data frame that was created with the use of `read.xlsx()` function.

Example:

```
> newData <- read.xlsx('Devine.xlsx', sheetIndex=1)
> names(newData)
[1] "study" "d"     "w"     "v"     "pub"
```

Command: `$`

Function: Using dollar sign operator, you can select a specific variable from a data frame.

Example:

```
> newData$pub
 [1] D J J D D D J J J J J J J J J J J J J J J J J D D D D D D J D J J J
D J D D D D D D D D J J J J J J J J J
Levels: D J
> newData$d
 [1]  0.57  0.42  0.35  0.08 -0.15 -0.23  0.88  0.56  1.27  0.69  0.62
0.46  0.19  1.36  1.36  1.26  1.38  1.00  0.59
[20]  0.32  0.59  0.16  0.70 -0.20 -0.08  0.39  0.29  0.58  0.80  0.74
0.04  0.40  0.30 -0.02  0.48  1.10  0.42  0.77
[39]  0.52  1.08  0.27  0.12  0.34  0.21 -0.21  0.63  0.28  0.25  0.20
0.46  0.37  0.52  0.78  0.54
```

In the example above, using the dollar sign operator (`$`) variables of d and pub were selected.

Command: `attach` and `detach`

Function: Using `attach()` you can reach your data frame variables without using the name of the data frame. Just the opposite of `attach()` function, `detach()` function removes your data frame names from search path.

Example:

```
> pub
Error: object 'pub' not found
> newData$pub
 [1] D J J D D D J J J J J J J J J J J J J J J J D D D D D D D J D J J J D J D
[48] J J J J J J J
Levels: D J
> attach(newData)
> pub
 [1] D J J D D D J J J J J J J J J J J J J J J J D D D D D D D J D J J J D J D
[48] J J J J J J J
Levels: D J
> detach(newData)
> pub
Error: object 'pub' not found
```

As you can see above, pub is a variable of `newData` data frame and you can only reach this variable using `$` (dollar sign notation) before running `attach()` function. But after you attach a data frame to the working environment you can reach a variable just by using its name alone. After `attach()` you do not have to use data frame name and `$` sign.

After your work is done, do not forget to detach your data frame from the working environment.

For more information about this command, type `?attach` and press `Enter`.

Command: `1:10`

Function: This command creates a sequence of numbers. This operator is known as colon operator (i.e., :). It can be very useful at times for selecting specific variables from a data frame.

Example:

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 1:43
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28
[29] 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
> 1:15
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

As you can see `1:10` creates a vector of numbers from 1 to 10. But you have to assign this data to use it later. So next commands are much more useful.

```
> dataSet <- 1:15
> dataSet
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

A sequence of numbers from 1 to 15 assigned to dataSet variable.

```
> str(newData)
'data.frame':       54 obs. of  5 variables:
 $ study: num  1 2 3 4 5 6 7 8 9 10 ...
 $ d    : num  0.57 0.42 0.35 0.08 -0.15 -0.23 0.88 0.56 1.27 0.69 ...
 $ w    : num  19.23 8.32 8.87 5 4.99 ...
 $ v    : num  0.052 0.12 0.113 0.2 0.201 ...
 $ pub  : Factor w/ 2 levels "D","J": 1 2 2 1 1 1 2 2 2 2 ...
> newData[, 1:3]
   study     d          w
1      1  0.57 19.226568
2      2  0.42  8.320691
3      3  0.35  8.867126
4      4  0.08  4.996160
5      5 -0.15  4.986527
6      6 -0.23  4.968439
7      7  0.88  3.207351
8      8  0.56  3.375284
9      9  1.27  5.436694
10    10  0.69  5.673344
```

With the use of `1:3`, you can be able to select first three variables from `newData` data frame. Beware of the comma before `1:3` and also the use of square brackets. You have to use this comma and square bracket around `1:3` to select columns. This is a special notation to select columns (variables) from a data frame.

Command: `ls`

Function: Used to list all objects in the current working environment.

Example:

```
> ls()
[1] "newData"
```

You can see the variables that you have created this way, as shown above code listing.

Command: `getwd`

Function: Returns an absolute filepath representing the current working directory of the R. It means 'get working directory'. Directory is synonym of folder.

Example:

```
> getwd()
[1] "C:/Users/mehmet-odtu/Desktop"
```

Useful to see in which folder you are.

Command: `dir`

Function: Shows you the files that are in the working directory. As previously stated you can see your working directory with `getwd` function.

Example:

```
> dir()
 [1] "1_Basics.ppt"
 [2] "1b_UsingExcel.pptx"
 [3] "1c_LoadingBorensteinEtAlDataIntoR.pptx"
 [4] "desktop.ini"
 [5] "Devine.xlsx"
 [6] "hsMethodScripts"
 [7] "michelle krummel youtube latex tutorials"
```

```
 [8] "psy559-research-proposal-makaleleri"
 [9] "psy614week9classNotes"
[10] "R_CommandsFunctions.docx"
[11] "R_CommandsFunctions.pdf"
[12] "slr"
```

Here are all of the files in my desktop. Using this, you can see if you have pasted an excel data file in the same working directory as R.

Command: `read.table`

Function: This function can be used to read csv (comma separated values) type of data files. Excel and SPSS can be used to create csv type of data files. Files of the type csv are ordinary text files and data within this files are just separated by commas, as the name implies.

Basic use is like that: `read.table(file, header = FALSE, sep = "")`. First you have to write the name of the file. `header` argument determines if the file has column names on the top line. If your file has variable names on the top line, you should write `header = TRUE` while using the function. `sep` argument determines the type of separator. `sep=","` means that values within the file are separated by commas. But besides commas, csv files may use semicolons (i.e., ;), or colons (i.e., :) to separate data within the file.

Example:

```
> read.table('Devine.csv', header=TRUE, sep=';')
   study     d          w           v pub
1      1  0.57 19.226568 0.05201136   D
2      2  0.42  8.320691 0.12018233   J
3      3  0.35  8.867126 0.11277611   J
4      4  0.08  4.996160 0.20015371   D
5      5 -0.15  4.986527 0.20054037   D
6      6 -0.23  4.968439 0.20127047   D
```

As always, you have assign the output of the function to a variable in order to use it later.

```
> anotherDataSet <- read.table('Devine.csv', header=TRUE, sep=';')
> anotherDataSet
```

```
  study     d         w          v pub
1     1  0.57 19.226568 0.05201136   D
2     2  0.42  8.320691 0.12018233   J
3     3  0.35  8.867126 0.11277611   J
4     4  0.08  4.996160 0.20015371   D
5     5 -0.15  4.986527 0.20054037   D
```

You can find more information on the use of this function by `?read.table` command.

Command: `sd`

Function: This function computes the standard deviation of the values.

Example:

```
> sd(newData$d)
[1] 0.407761
```

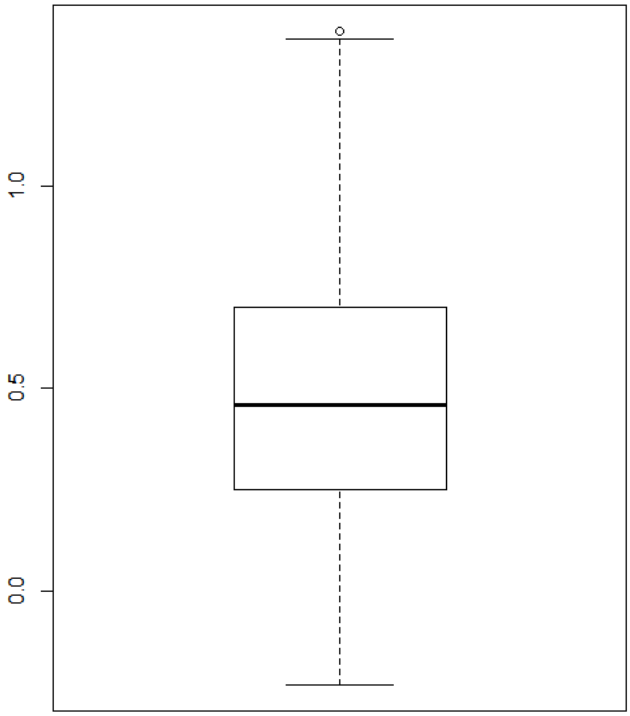
You can find more information on the use of this function by `?sd` command.

Command: `boxplot`

Function: This function computes the standard deviation of the values. In the example below a boxplot of `d` variable of `newData` data frame is created with `boxplot()`.

Example:

```
> boxplot(newData$d)
```



You can find more information on the use of this function by `?boxplot` command.

Command: `order`

Function: Helps you to order your variable from smaller values to larger values.

Example:

```
> newData$d
 [1]  0.57  0.42  0.35  0.08 -0.15 -0.23  0.88  0.56  1.27  0.69  0.62
[12]  0.46  0.19  1.36  1.36  1.26  1.38  1.00  0.59  0.32  0.59  0.16
[23]  0.70 -0.20 -0.08  0.39  0.29  0.58  0.80  0.74  0.04  0.40  0.30
[34] -0.02  0.48  1.10  0.42  0.77  0.52  1.08  0.27  0.12  0.34  0.21
[45] -0.21  0.63  0.28  0.25  0.20  0.46  0.37  0.52  0.78  0.54
> order(newData$d)
 [1]  6 45 24  5 25 34 31  4 42 22 13 49 44 48 41 47 27 33 20 43  3 51
[23] 26 32  2 37 12 50 35 39 52 54  8  1 28 19 21 11 46 10 23 30 38 53
[45] 29  7 18 40 36 16  9 14 15 17
```

Previous code gives an instance of `order()` function behavior. This tehavior of the `order()` function is kind of counterintuitive at first, but it is quite logical. It gives the place of the value that should be in the first place, and the second value that should be inserted to the second place in an ascending ordering, and so on. In the example above, `order()` function returns `6` as the first value, and `45` as the second value. If you look at the `newData$d` variable you can see that in the 6th place you have a value of `-0.23` and in the 45th place you have a value of `-0.21`. So, in an ascending ordering these values must be selected as the first and second value. This way, `order()` function gives you the place of the values that should be used in an ascending order.