

# Applied Missing data analysis with SPSS and R(Studio)

Martijn Heymans and Iris Eekhout

2018-10-02



# Contents



# Preface

The attention for missing data is growing and so will be the application of methods to solve the missing data problem. From our experience, researchers with missing data still find it difficult to reserve time to evaluate the missing data and from that to find a reasonable solution to handle their missing data for their main data analysis. This manual is developed for researchers that are looking for a solution of their missing data problem or want to learn more about missing data. The manual is developed as a result of a missing data course that we give. Further, we are also active in providing statistical advice in general and more specific about missing data. Because our time to give advice is mostly limited we wanted to give researchers a practical guide to help them get started with their missing data problem. Leading methodologists and statisticians and leading journals have published papers about the problems of missing data and warned researchers to take missing data seriously (Sterne et al., BMJ 2009, Little et al. NEJM 2012, Peng et al. 2015, JAMA). Hopefully this manual will help researchers to find the best solution for their missing data problem. We hope you will enjoy this manual and that you learn from it, at least to take missing data seriously and that you will use recommended methods to solve your missing data problem.

## 0.1 The goal of this Manual

In this manual the software packages SPSS and R play a central role. The combination of these two software packages may seem a coincidence, but it is not. For a long time, SPSS was the most popular software package worldwide to do statistical data analysis. Currently, R is growing in popularity fast and will probably become one of the most popular Software packages to do data analysis. Also for applied researchers. Both SPSS and R have their advantages and disadvantages. An advantage of SPSS is that it is a user-friendly software package compared to R and works with windows where you can for example drag your variables to. Subsequently, you can click the OK button and the statistical analysis procedure you prespecified gives you the output results. A disadvantage of SPSS may be that you are overloaded with statistical output

that may not all needed to answer your research question. Compared to SPSS you could say that R is a more user-unfriendly software package where you need to use R code to activate statistical procedures and to get statistical results. R output will show more specific results, without extra information. Furthermore, R works much faster when it comes to running statistical procedures by using 1 or 2 lines of R code, compared to visiting a couple of windows in SPSS to activate the same statistical test. There is one other advantage of R and that is, that it is open source. This makes it possible for applied researchers to follow the calculations of complex procedures as the estimation of missing values closely along the line. You could say that R brings you to the heart of the matter. With R it is possible to turn complex data analysis functions and formula's into computer code that can be used by everybody and vice versa. Because it is open source, you are able to read the code that is used for the analysis and to relate that code or pieces of code to the statistical output. This makes it possible to evaluate step by step the code and thus the statistical procedures and relate them to the subsequent results. You can copy specific parts of code from functions that others have written and evaluate what happens. This is one of the major advantages of R if you compare it to the closed source statistical package SPSS. R brings you a big learning environment when it comes to the understanding of all kind of statistical procedures as missing data analysis.

## 0.2 Multiple Imputation in SPSS and R

Multiple Imputation (MI) is a procedure that is developed in the 1970's by Donald Rubin. Later, around the 1990's Multiple imputation was further developed and became more popular. For a long time, MI was only available for S-Plus and R software (S-plus is the commercial alternative of R), where it was further developed by Stef van Buuren, a statistician from TNO, Leiden, The Netherlands. For a long time, it was not possible to do MI analysis in SPSS because it was not available in SPSS. So, it was far out of reach for applied researchers for a long time. It became available from SPSS version 17. From that time MI is now used more by applied researchers. In this manual the handling of missing data is the main topic. We will also show how to apply these methods in both software packages SPSS and R. To apply the imputation methods that are discussed both software packages make use of random starting procedures. SPSS and R use for that intern random number generators. Because these are different, result might slightly differ. Our intention is not to compare the software packages SPSS and R and their output resultys. Both are trustful packages, it is more the estimation procedures that might lead to the differences. The imputation methods, will be applied in SPSS version 24 and with R software version 3.4.3. The R examples will be presented by using the output from RStudio version (version 1.1.383 – © 2009-2017 RStudio, Inc.). RStudio is an integrated development environment (IDE) for R. RStudio includes a wide range of productivity enhancing features and runs on all major platforms. As

already stated, R allows you to program the statistical formula's yourself. We have therefore chosen to explain the formula's in more detail in combination with the application in R. The more applied researchers will be satisfied with the explanation and application of methods in SPSS.

### 0.3 Notation and annotation in this manual

The name of R packages, libraries and functions can be recognized by using Courier new lettertype, for example the package mice will be written as mice.

R code of the procedures used in the manual is marked grey and the explanation in these grey parts can be found in the grey parts itself annotated by the # symbol. The lines that start with the symbol > are R Code lines that have been running in the R Console in RStudio. Example:

**R code XX**

```
# Activate the foreign package and read in the SPSS dataset  
library(foreign)  
dataset <- read.spss(file="data/Backpain 50 missing.sav", to.data.frame=T)
```

```
## re-encoding from UTF-8
```





# Chapter 1

## Software applications

Statistical software programs can help us to analyze our data. SPSS and R are such programs. Although SPSS and R are among the most popular programs to do statistical data analyses nowadays, they do not have much in common. One of the greatest differences is that SPSS works with menu options that make windows appear and you can click buttons to select options, whereas R works with lines of code that you have to type in to run analyses. This makes SPSS more user-friendly than R for applied researchers. In SPSS you are overloaded with output tables, and in R you only get output on demand. In this Chapter we will explore the different possibilities of the SPSS (IBM 2016) and the R software language (Matloff, 2011, Dalgaard, 2008). We will run R via RStudio, the integrated development environment (IDE) for R. RStudio includes a wide range of productivity enhancing features, which makes it easier to work with than with the R console on its own.

### 1.1 SPSS, Data and Variable View windows

In this manual we work with SPSS version 24 (IBM, 2016). When you start SPSS Version 24 a start-up window appears. In this window, you can directly open the files that were active during your previous use of SPSS. These files can be found and easily opened in the “Recent files” window (Figure ??). If you do not want to see this window the next time that you open SPSS, select “Don’t show this dialog in the future”.

When you click on Close on the right side below, the window will close and you will see an empty Data View window. Now you are in the SPSS Data Editor window. This window is always open when you start SPSS. The name “SPSS Data Editor” is also visible at the top of the screen and is called “IBM SPSS Statistics Data Editor” (Figure ??).

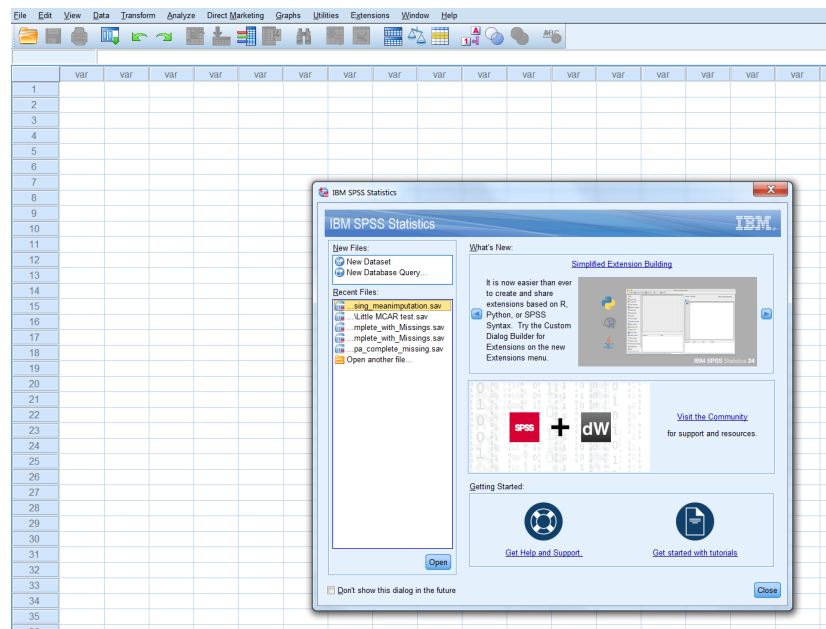


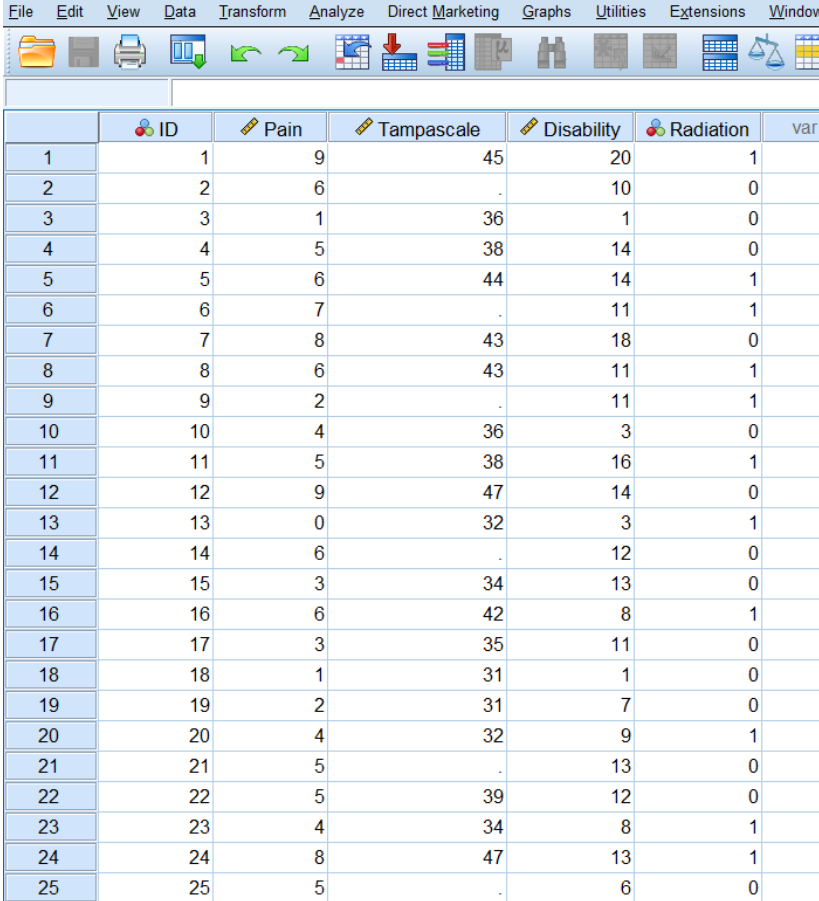
Figure 1.1: First window after you have started SPSS

In the SPSS Data Editor, you have the possibility to go to the Data View and Variable View windows. In the Data View window, you can enter data yourself or read in data by using the options in the file menu. In Figure 1.2 you see an example of a dataset in the Data View window. Each row in the Data View window represents a case and in the columns you will find the variable names. In the Data View window, you can do all kind of data manipulations by using the different menu's above in the window. From here you can click on the tab Variable View, in the lower left corner of the window. Then the Variable view window will appear (Figure ??).

In the Variable View window, you can add new variables, by entering the name in the name column. Further, you can change the columns by using the following options: Type: Here you can change the type of variables in your dataset. Mostly you work with numeric variables, i.e. a variable whose values are numbers. Other possibilities are Date variables which is a numeric variable whose values are displayed in one of several calendar-date or clock-time formats or String variables, a character (text) variable that can contain any characters up to the defined length. String values are not numeric and therefore are not used in calculations. Width: By default SPSS defines a numeric variable with 8 digits for each new variable.

Decimals: the number of decimal places displayed.

Label: The variable name.



	ID	Pain	Tampascale	Disability	Radiation	var
1	1	9	45	20	1	
2	2	6	.	10	0	
3	3	1	36	1	0	
4	4	5	38	14	0	
5	5	6	44	14	1	
6	6	7	.	11	1	
7	7	8	43	18	0	
8	8	6	43	11	1	
9	9	2	.	11	1	
10	10	4	36	3	0	
11	11	5	38	16	1	
12	12	9	47	14	0	
13	13	0	32	3	1	
14	14	6	.	12	0	
15	15	3	34	13	0	
16	16	6	42	8	1	
17	17	3	35	11	0	
18	18	1	31	1	0	
19	19	2	31	7	0	
20	20	4	32	9	1	
21	21	5	.	13	0	
22	22	5	39	12	0	
23	23	4	34	8	1	
24	24	8	47	13	1	
25	25	5	.	6	0	

Figure 1.2: Data View window in SPSS

	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
1	ID	Numeric	8	0		None	None	8	Right	Nominal	Input
2	Pain	Numeric	2	0		None	None	8	Right	Scale	Input
3	Tampascale	Numeric	2	0		None	None	13	Right	Scale	Input
4	Disability	Numeric	8	0		None	None	9	Right	Scale	Input
5	Radiation	Numeric	1	0		None	None	9	Right	Nominal	Input
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											

Figure 1.3: Variable View window in SPSS

**Values:** To assign numbers to the categories of a variable. To define Variable values do the following: 1. Click the button in the Values cell for the variable that you want to define. 2. For each value, enter the value and a label. 3. Click Add to enter the value label. 4. Click OK.

**Missing:** Here you can define specified data values as user-missing. You can enter up to three discrete (individual) missing values, a range of missing values, or a range plus one discrete value.

**Columns:** To change the number of characters displayed in the Data View window.

**Align:** Here you can specify the alignment of your data.

**Measure:** Here you can specify the level of each variable, scale (continuous), ordinal or nominal.

**Role:** Here you can define the role of the variable during your analysis. Examples are, Input for independent variable, Target for dependent or outcome variable, Both, independent and dependent variable. There are more possibilities, but most of the times you use the default Input setting.

## 1.2 Analyzing data in SPSS

All statistical procedures in SPSS can be found under the Analyze button (Figure ??). Here you also will find the option “Multiple Imputation” which plays an important role in this manual. We will use this menu later on in Chapter 4.

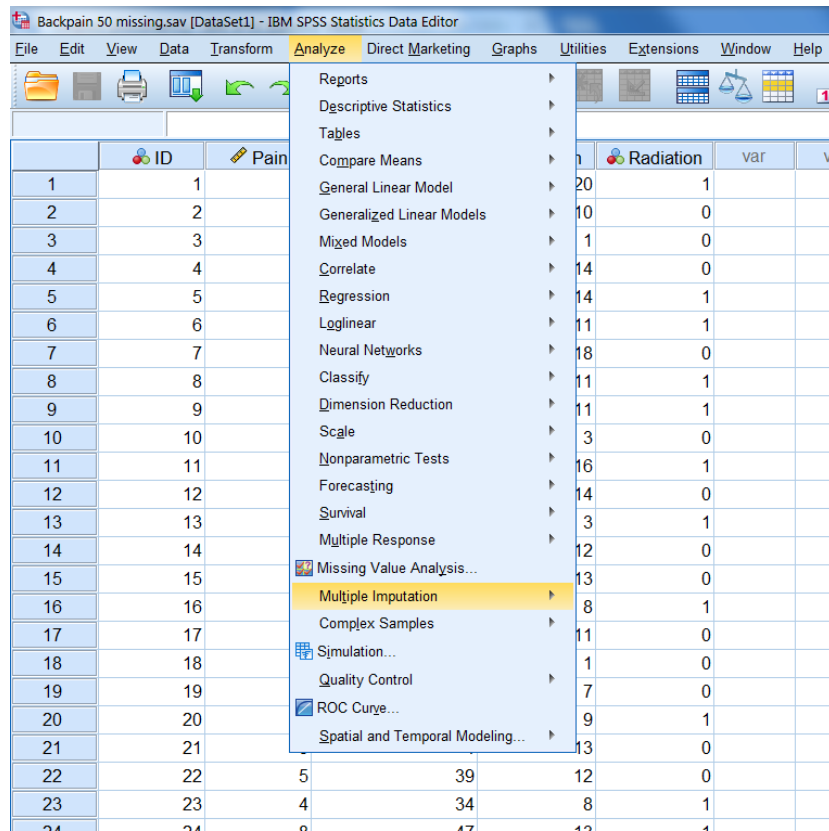


Figure 1.4: Statistical procedures that can be found under the Analyze menu in SPSS

### 1.3 Data Transformations in SPSS

Two other interesting buttons are Data and Transform. The Data menu allows you to make changes to the data editor. Here you can add new variables or cases. You can also use the Split File option, to get analyses results separately for categories of a variable. The Transform menu allows you to manipulate your variables by for example dichotomizing a numeric variable.

### 1.4 The Output window in SPSS

If you have run your analyses in SPSS, an SPSS Output (or viewer) Window will pop-up. The main body of the Output Window consists of two panes (left and right panes). In the left pane you will find an outline of the output. In the right pane you will find the actual output of your statistical procedure (Figure ??).

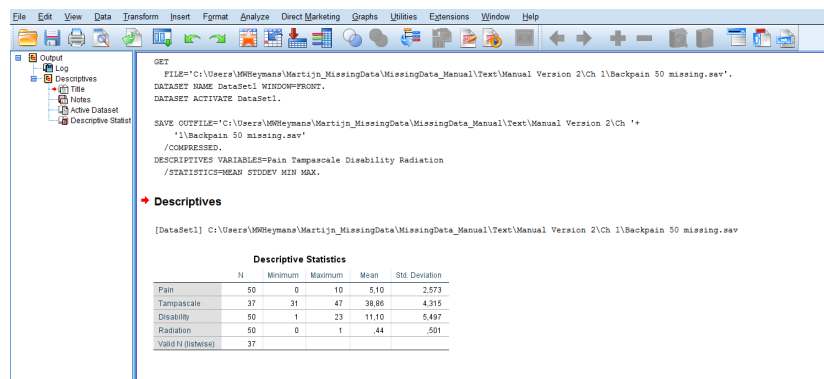


Figure 1.5: Part of the Output or Viewer window in SPSS after making use of Descriptive Statistics under the Analyze menu

### 1.5 The Syntax Editor in SPSS

In the syntax editor of SPSS, you use the SPSS syntax programming language. You can run all SPSS procedures by typing in commands in this syntax editor window, instead of using the graphical user interface, i.e. by using your mouse and clicking on the menu's. You can get access to the syntax window in two ways. The first is just by opening a new syntax file by navigating to File -> New -> Syntax. This will open a new syntax window (Figure ??).

Now you can start writing your syntax directly in this window. You can also generate syntax by accessing statistical procedures through the dropdown menus

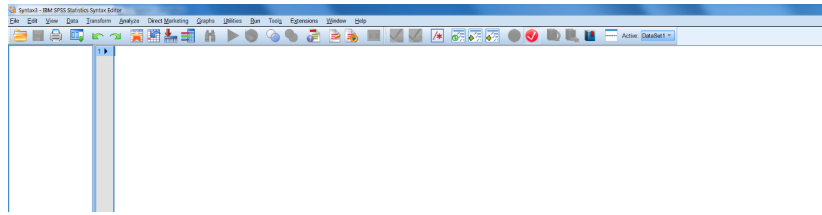


Figure 1.6: Screenshot of new syntax file

and clicking the Paste button instead of clicking the OK button after you have specified the options. When you have clicked the Paste button, a new Syntax Editor window will pop up or the new syntax will automatically be added to the open Syntax Editor window. This is a very useful way to keep track of the analysis that you have performed. An example can be found in Figure ??, where the syntax is shown for the Descriptive Statistics procedure of Figure ??.

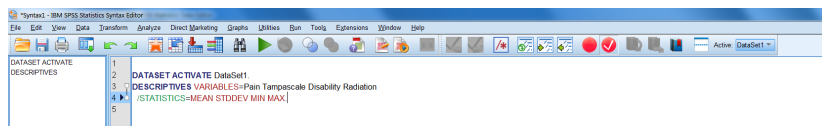


Figure 1.7: Screenshot of Syntax editor of SPSS including the Syntax code for descriptive statistics

By using the SPSS Syntax it is possible for users to perform the same analyses over and over again or to adapt the analysis via the syntax code for complex calculations in the data. In this manual we will not use SPSS syntax code to access statistical procedures, however we recommend to use the SPSS syntax to keep track of the analysis that you have performed. SPSS is most frequently used via the graphical user interface, and we will use that method also in this manual.

## 1.6 Reading and saving data in SPSS

Reading in data in SPSS is very easy; via the menu File choose for File -> Open -> Data. All kind of file types can be selected. Of course the SPSS .sav files, but also .por, .xlsx, .csv, SAS, Stata, etc. (Figure 1.8). After you have selected a specific file type you may have to go through several steps before you see the data in the Data View window. These steps are not necessary for SPSS files, they open directly in the data editor.

Saving files in SPSS is possible via the Save Data As option under the menu File. You can choose the same kind of file types (Figure ??).

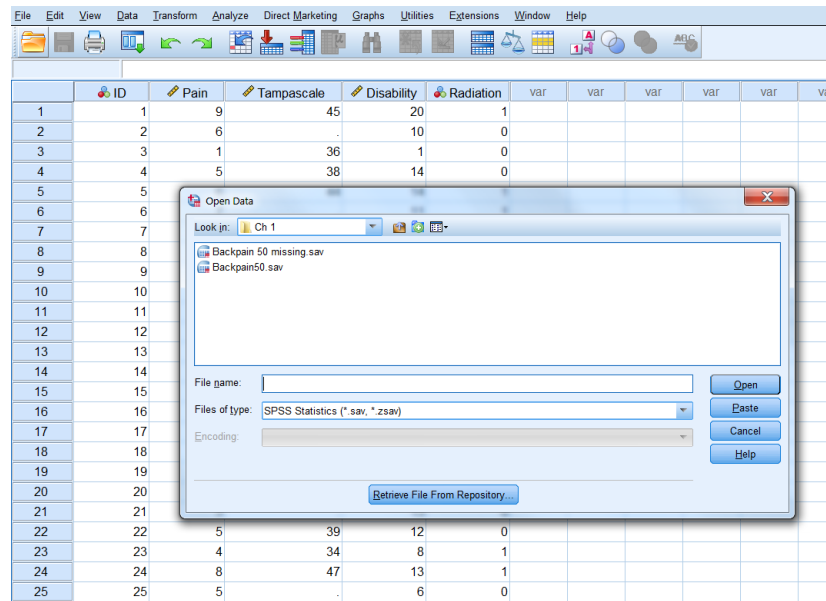


Figure 1.8: Window to read in different file types in SPSS

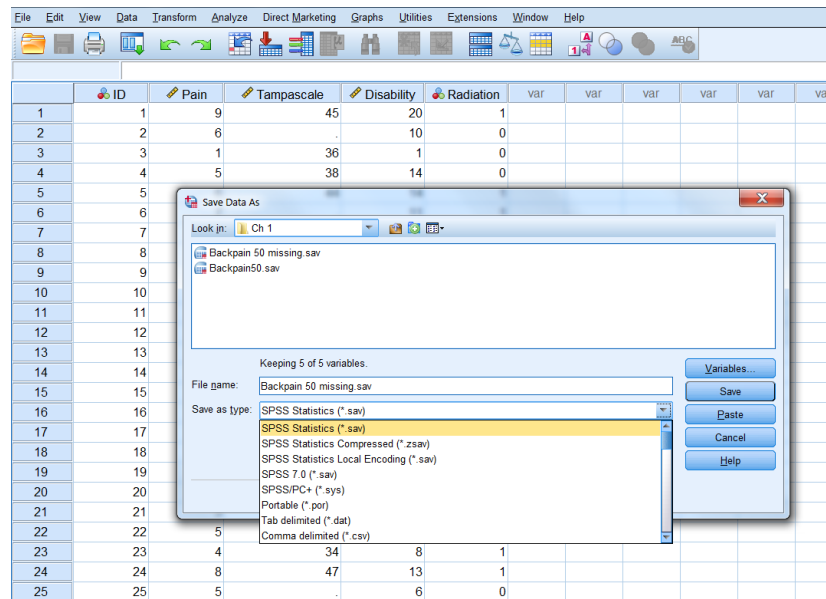


Figure 1.9: Option Save Data As under the menu File



## 1.7 R and RStudio

RStudio is an integrated environment to work with the software program R. Consequently, to work with RStudio, R has to be installed. RStudio uses the R language and is also freely available. In this manual we will only show some possibilities and options in RStudio that are needed to run the R code and the programs that are discussed in this manual. For more information about RStudio and its possibilities visit the RStudio website at [www.rstudio.com](http://www.rstudio.com). When you open RStudio the following screen will appear.

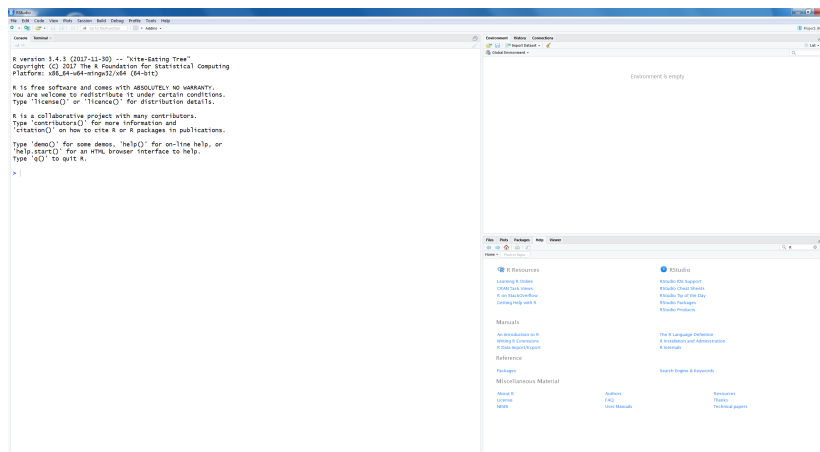


Figure 1.10: First screen that appears after you have started RStudio

There are three windows opened:

1. On the left is the Console window

This is the main window to run R code (see below for more information about the Console window).

2. Right above is the window where you can choose between the Environment and History tabs (e.g. history tracks the code you typed in the Console window).
3. At the right site below is the window where you can choose between Files, Plots, Packages, Help and Viewer tabs.

### 1.7.1 The role of the Console Window

When you enter code in the Console window you will directly receive a result. For example, you can type the following code and the result will appear directly in the Console window.

```
3 + 3
```

```
## [1] 6
```

As you can see, you can use R as a large calculator. Other multiplication procedures as divide, square, etc. can also be executed. However, the main use for R is its functions. For example, when you generate 20 random numbers you can use the following function code (we will discuss more about functions in R later):

```
rnorm(20)
```

```
## [1] 0.92131529 2.65701851 1.90415747 0.85969862 0.08844802
## [6] -1.90699760 0.79545041 -0.28056647 -0.33686286 -1.23675298
## [11] -0.37910063 1.49171080 0.67527605 -0.05150813 1.53159985
## [16] -2.95832118 -0.81409661 -0.60473972 -0.32302686 -0.09596981
```

The number [1] between brackets is the index of the first number or item in the vector. That can be useful when you have many numbers printed in the Console window.

## 1.7.2 R assignments and objects

In R it is possible to create objects and to assign values to these objects. In this way it is for example possible to store some intermediate results and recall or use them later on. Assigning values to objects is done by using the assignment operator `<-` (R code below). You can also use the `=` sign as an assignment operator. This is not recommended because this is also a symbol used for some mathematical operations. For example, when we want to assign the value 3 to the object `x`, we use the following code.

```
x <- 3
```

When we subsequently type in the letter `x` we get the following result:

```
x
```

```
## [1] 3
```

As result we see the value 3 again. Now the value 3 is assigned to the object `x`. In R all kind of information can be assigned to an object, i.e. one number, a vector of numbers, results from analysis or other R objects such as data frames, matrices or lists. Objects can have all kinds of different names, composed of different letters and numbers. Here are some examples where number 3 is assigned to different objects with different names:

```
test <- 3
test.1 <- 3
test.manual <- 3
test
```

```
## [1] 3
```

```
test.1
```

```
## [1] 3
```

```
test.manual
```

```
## [1] 3
```

Note that some letters and words are used by R itself. It is not recommended to use these letters as names for objects in R that you create yourself. For example, the letter T and F are used as TRUE and FALSE by R. Other letters that are already in use are c, q, t, C, D, I and diff, df, and pt.

### 1.7.3 Vectors, matrices, lists and data frames

*Vectors* In the previous paragraph we created the one-vector x, i.e. a vector that contained only one number, the number 3. Mostly you do your analysis on more numbers than just one. R has several possibilities to create objects with more data. We start with a simple dataset, called a data vector, which is an array of numbers (such as created in R code 1.2. A vector can be created by the following code:

```
y <- c(1, 2, 3, 4, 5)
y
```

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

Now the numbers 1, 2, 3, 4 and 5 are assigned to the data vector y. The “c” in the above code stand for concatenate which makes that all separate (one-vector) numbers are merged into one vector. It is also possible to create character vectors, which are vectors that contain strings (text). An example:

```
y <- c("a", "b", "test")
y
```

```
## [1] "a"      "b"      "test"
```

Vectors can also be made by using the “:” symbol. With that symbol it is easy to generate a sequence of numbers. An example:

```
y <- 1:10
y
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
y <- 2:6
y
```

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

Both lines of code produce a vector containing a sequence of numbers. The first example produces the numbers 1 to 10 and the second example the numbers 2 to 6.

*Matrix* A matrix in R contains rows and columns and can be created by using the matrix function.

```
matrix(c(1, 2, 3, 4, 5, 6), nrow=2, ncol=3)
```

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

Now we have created a matrix with 2 rows and 3 columns. In essence we converted the vector c(1, 2, 3, 4, 5, 6) into a matrix.

*List* Another popular R object class is a list. The advantage of a list is that it can contain components of different formats. Let’s look at an example using the following code:

```
x <- 1:5
x
```

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

```
y <- c("a", "b", "test")
z <- list(x=x, y=y)
z
```

```
## $x
## [1] 1 2 3 4 5
##
## $y
## [1] "a"      "b"      "test"
```

The code created the list object `z` consisting of the two components `x` and `y` which are the vectors that were created above. You can see that in a list two components of different data type can be combined, a numeric and a character factor. The names of the list components are indicated by the dollar sign, *The list component can be obtained separately by typing `zx` for component `x` or `xy` for component `y`.*

*Dataframe* Mostly we work with datasets that contain information of different variables and persons. In R such a dataset is called a dataframe. In essence, a dataframe in R is a list, where each component of the list is a vector of equal length. This example code first creates a list, consisting of 3 vectors of the same length. Then this list is converted into a dataframe, using the `data.frame` function.

```
k <- list(a=1:10, b=11:20, c=21:30)
k
```

```
## $a
## [1] 1 2 3 4 5 6 7 8 9 10
##
## $b
## [1] 11 12 13 14 15 16 17 18 19 20
##
## $c
## [1] 21 22 23 24 25 26 27 28 29 30
```

```
z <- data.frame(k)
z
```

```
##      a  b  c
## 1    1 11 21
## 2    2 12 22
## 3    3 13 23
## 4    4 14 24
## 5    5 15 25
## 6    6 16 26
## 7    7 17 27
## 8    8 18 28
## 9    9 19 29
## 10  10 20 30
```

Typically, a dataframe is created by reading in an existing dataset. How to create a dataframe by reading in a dataset will be further discussed in the paragraph “Reading in and saving data”.

### 1.7.4 Indexing Vectors, Matrices, Lists and Data frames

*Vectors* An important operation in R is to select a subset of a given vector. This is called indexing vectors. This subset of the vector elements can be assigned to another vector. An example how to do this:

```
y <- c(3, 5, 2, 8, 5, 4, 8, 1, 3, 6)
y[c(1, 4)]
```

```
## [1] 3 8
```

The R code `y[c(1, 4)]`, extracts the first and fourth element of the vector. Another example is by using the “:” symbol, to extract several subsequent elements:

```
y[2:5]
```

```
## [1] 5 2 8 5
```

The R code `y[2:5]`, extracts the second to the fifth element of the vector.

A minus sign excludes the specific element from the vector, like:

```
y[-3]
```

```
## [1] 3 5 8 5 4 8 1 3 6
```

The R code `y[-3]`, excludes the third element of the vector (i.e. 2). A new vector `z` can be created where the third and fourth element of the `y` vector are excluded, by using the following code:

```
z <- y[-c(3, 4)]
z
```

```
## [1] 3 5 5 4 8 1 3 6
```

*Matrices* When we index matrices we can choose to index rows, columns or both. Here are some examples:

First we construct the matrix

```
z <- matrix(1:9, nrow=3)
z
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

We extract from the first row the number in the second column

```
z[1, 2]
```

```
## [1] 4
```

We extract all numbers in each column in the first row

```
z[1, ]
```

```
## [1] 1 4 7
```

We extract all numbers in each row of the first column

```
z[, 1]
```

```
## [1] 1 2 3
```

A minus sign can also be used to delete specific elements or complete rows or columns.

Omit the first row from the matrix

```
z[-1, ]
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    3    6    9
```

*Lists* We first create a list with 3 components, each component consists of a vector of the same length and with 10 elements each.

```
k <- list(a=1:10, b=11:20, c=21:30)
k

## $a
## [1] 1 2 3 4 5 6 7 8 9 10
##
## $b
## [1] 11 12 13 14 15 16 17 18 19 20
##
## $c
## [1] 21 22 23 24 25 26 27 28 29 30
```

There are several options to index list components and elements of the components. If we want to index the individual component b we can use the following code:

```
k$b

## [1] 11 12 13 14 15 16 17 18 19 20
```

```
k[["b"]]

## [1] 11 12 13 14 15 16 17 18 19 20
```

```
k[[2]]

## [1] 11 12 13 14 15 16 17 18 19 20
```

As you have may noticed, to extract individual list components double square brackets are used, compared to single brackets for indexing vectors and matrices.

When we use single brackets, we get the following results:

```
k["b"]

## $b
## [1] 11 12 13 14 15 16 17 18 19 20
```

The difference between using single and double brackets is that single brackets return the component data type, which is in this case a vector (but could be any kind of data type) and single brackets always return a list.



*Data frames* Indexing data frames follows the same method as indexing matrices, but we can also use the method that is used to index lists, since data frames are essentially lists of vectors of the same length. Data frames consist of rows and columns which can be accessed separately or both to extract specific elements. We use the data frame that was constructed in R code 1.11.

```
z
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Examples of indexing the first column are:

```
z[, 1]
```

```
## [1] 1 2 3
```

```
k$a
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

The second row can be accessed by using:

```
z[2, ]
```

```
## [1] 2 5 8
```

### 1.7.5 Vectorized Calculation

An advantage of working with R is that it is possible to perform vectorized calculations. This means that you can do calculations elementwise, i.e. the same calculation is done on each element of an object. Let's look at an example.

We first create a vector `z` with numerical variables.

```
z <- c(1, 2, 3, 4, 5, 6, 7, 8)
z
```

```
## [1] 1 2 3 4 5 6 7 8
```

Now it is fairly easy to square each element of the vector by typing:

```
z^2
```

```
## [1] 1 4 9 16 25 36 49 64
```

The  $^2$  means take the power of 2. We see that each element is squared. These vectorized calculations can be done by using all kinds of mathematical functions, e.g. taking the square root, logarithms, adding constant values to each element, etc.

### 1.7.6 R Functions

Functions play an important role in R. Functions can be seen as lines of R code to run all kind of data procedures and to return a result. Let's write a small function that we name "sum.test" that generates a sequence of numbers from 1 to 5 and sums all values from 1 to the value we define beforehand, which is in our example the value 3. This means that the function will sum all values from 1 to 3, i.e.  $1 + 2 + 3$  when it is at the value of 3 and all values from 1 to 4 when we use as input value the value 4. The function can be found in R code 1.26.

```
print.sum.test <- function(x)
{
  for (i in 1:5)
  {
    if (i==x) y <- sum(1:i)
  }
  return(y)
}
print.sum.test(3)
```

```
## [1] 6
```

At the first line we define the function `print.sum.test` that includes one argument `x`. Then in the body of the function (in italic) which starts with a left brace and ends with another brace, the actual calculations take place. The return statement gives back the result. At the end we call the function with the statement `print.sum.test(3)`, where the value 3 is actually used in the function.

Once the function is defined, we can call the function and plug in other values. For example:

```
print.sum.test(4)
```

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

That will directly lead to the result 10 when the value 4 is used.

You can write functions yourself but in R many functions are available which means that many calculations are done by using function calls. A function name is followed by a set of parentheses which contain some arguments. In the above self-written function, we already made use of a function, which is the sum function. We can use it separately as follows:

```
sum(3,4)
```

```
## [1] 7
```

The arguments in this function are the numbers 3 and 4 and the result is their sum. This function uses as arguments numbers or complete vectors. If you want to see the formal arguments of each function you can use the args function. You have to type the name of the function as the argument in the args function. For example, we can use it for the matrix function:

```
args(matrix)
```

```
## function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
## NULL
```

As a result the arguments of the functions are listed with their default settings. In this case the arguments are:

data: an optional data vector (including a list or expression vector).

nrow: the desired number of rows.

ncol: the desired number of columns.

byrow: logical. If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.

dimnames: A dimnames attribute for the matrix: NULL or a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions.

### 1.7.7 The Help function

There are several possibilities to start the help facilities in R and to get more information about functions and their arguments in R.

You can just type help or use the question mark as follows: