

‘From Statistics to Data Mining’

Computer Lab Session n° 1:


Introduction to (1/2)

Master 1 COSI / CPS²
Saint-Étienne, France

Fabrice Muhlenbach

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet de Saint-Étienne
18 rue du Professeur Benoît Luras
42000 SAINT-ÉTIENNE, FRANCE
<http://perso.univ-st-etienne.fr/muhlfabr/>

Outcome

The primary objective of this lab is to become comfortable with the  free software programming language while using variables and functions, reading data from the terminal and importing data from files.



1 Working with Variables

1.1 Variable Assignment


We assign values to variables with the assignment operator ‘=’. Just typing the variable by itself at the prompt will print out the value. We should note that another form of assignment operator “<=” is also in use.

```
> x = 1  
> x
```

1.2 Basic Data Types

There are several basic  data types that are of frequent occurrence in routine  calculations.

1.2.1 Numeric

Decimal values are called **numerics** in . It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

```
> x = 10.5      # assign a decimal value
```

```
> x                # print the value of x
[1] 10.5
> class(x)         # print the class name of x
[1] "numeric"
```

Furthermore, even if we assign an integer to a variable k , it is still being saved as a numeric value.

```
> k = 1
> k                # print the value of k
[1] 1
> class(k)         # print the class name of k
[1] "numeric"
```

To know if k is an integer or not, we can use `is.integer` function.

```
> is.integer(k)    # is k an integer?
[1] FALSE
```

1.2.2 Integer

In order to create an **integer** variable in \mathbb{R} , we invoke the `as.integer` function. We can be assured that y is indeed an integer by applying the `is.integer` function.

```
> y = as.integer(3)
> y                # print the value of y
[1] 3
> class(y)         # print the class name of y
[1] "integer"
> is.integer(y)    # is y an integer?
[1] TRUE
```

1.2.3 Complex

A **complex** value in \mathbb{R} is defined via the pure imaginary value i .

```
> z = 1 + 2i       # create a complex number
> z                # print the value of z
[1] 1+2i
> class(z)         # print the class name of z
[1] "complex"
```


1.2.4 Logical

A **logical** value is often created via comparison between variables.

```
> x = 1; y = 2      # sample values
> z = x > y         # is x larger than y?
> z                # print the logical value
[1] FALSE
> class(z)          # print the class name of z
[1] "logical"
```

Standard logical operations are "&" (and), "|" (or), and "!" (negation).

1.2.5 Character

A **character** object is used to represent string values in . We convert objects into character values with the `as.character()` function:

```
> x = as.character(3.14)
> x                # print the character string
[1] "3.14"
> class(x)         # print the class name of x
[1] "character"
```

Two character values can be concatenated with the `paste` function.

```
> fname = "Joe"; lname = "Smith"
> paste(fname, lname)
[1] "Joe_Smith"
```

1.2.6 Vector Matrix

A **vector** is a sequence of data elements of the same basic type. Members in a vector are officially called **components**. Nevertheless, we will just call them **members** in this site.

Here is a vector containing three numeric values 2, 3 and 5.

```
> c(2, 3, 5)
[1] 2 3 5
```

Vectors can be combined via the function `c`. For examples, the following two vectors *n* and *s* are combined into a new vector containing elements from both vectors.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> c(n, s)
[1] "2" "3" "5" "aa" "bb" "cc" "dd" "ee"
```

We retrieve values in a vector by declaring an index inside a single square bracket '[']' operator.

To produce a vector slice between two indexes, we can use the colon operator ':'. This can be convenient for situations involving large vectors.

```
> s[2:4]
[1] "bb" "cc" "dd"
```

More information for the colon operator is available in the [R](#) documentation.

```
> help(" : ")
```

1.2.7 Matrix

A **matrix** is a collection of data elements arranged in a two-dimensional rectangular layout. The following is an example of a matrix with 2 rows and 3 columns.

$$A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

We reproduce a memory representation of the matrix in [R](#) with the matrix function. The data elements must be of the same basic type.

```
> A = matrix(
+   c(2, 4, 3, 1, 5, 7), # the data elements
+   nrow=2,              # number of rows
+   ncol=3,              # number of columns
+   byrow = TRUE)       # fill matrix by rows
> A                      # print the matrix
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

An element at the m^{th} row, n^{th} column of A can be accessed by the expression $A[m, n]$.

```
> A[2, 3]      # element at 2nd row, 3rd column
[1] 7
```

The entire m^{th} row A can be extracted as $A[m,]$.

```
> A[2, ]       # the 2nd row
[1] 1 5 7
```

Similarly, the entire n^{th} column A can be extracted as $A[,n]$.

```
> A[,3]           # the 3rd column
[1] 3 7
```

We can also extract more than one rows or columns at a time.

```
> A[,c(1,3)]      # the 1st and 3rd columns
      [,1] [,2]
[1,]     2     3
[2,]     1     7
```

If we assign names to the rows and columns of the matrix, than we can access the elements by names.

```
> dimnames(A) = list(
+   c("row1", "row2"),           # row names
+   c("col1", "col2", "col3")) # column names

> A                               # print A
      col1 col2 col3
row1     2    4    3
row2     1    5    7

> A["row2", "col3"] # element at 2nd row, 3rd column
[1] 7
```

We construct the transpose of a matrix by interchanging its columns and rows with the function `t`.

```
> t(B)           # transpose of B
      [,1] [,2] [,3]
[1,]     2    4    3
[2,]     1    5    7
```

We can combine the columns of two matrices with `cbind`, similarly, we can combine the rows of two matrices if they have the same number of columns with the `rbind` function.

1.2.8 List

A list is a generic vector containing other objects.

For example, the following variable x is a list containing copies of three vectors n , s , b , and a numeric value 3.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x = list(n, s, b, 3) # x contains copies of n, s, b
```

1.2.9 Data Frame


A **data frame** is used for storing data tables. It is a list of vectors of equal length. For example, the following variable *df* is a data frame containing three vectors *n*, *s*, *b*.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b) # df is a data frame
```



For labelled data frames, the top line of the table, called the **header**, contains the column names. Each horizontal line afterward denotes a **data row**, which begins with the name of the row, and then followed by the actual data. Each data member of a row is called a **cell**.

To retrieve data in a cell, we would enter its row and column coordinates in the *single square bracket* '[' operator. The two coordinates are separated by a comma. In other words, the coordinates begins with row position, then followed by a comma, and ends with the column position. The order is important.

1.3 Functions



An  function is invoked by its name, then followed by the parenthesis, and zero or more arguments. The following apply the function *c* to combine three numeric values into a vector (*c* stands for “combine”).

```
> c(1, 2, 3)
[1] 1 2 3
```

A function in  is just another object that is assigned to a symbol. You can define your own functions in , assign them a name, and then call them just like the built-in functions:

```
> f <- function(x, y) {c(x+1, y+1)}
> f(1, 2)
[1] 2 3
```

2 Basic Operations in R

When you enter an expression into the  console and press the Enter key,  will evaluate that expression and display the results (if there are any).

```
> 1 + 2 + 3
[1] 6
> 1 + 2 * 3
[1] 7
> (1 + 2) * 3
[1] 9
```

Not all functions are of the form `f(...)`. Some of them are in the form of operators (for example, we used the addition operator `+` above). Here are a few examples of operators:

```
> 17 + 2
[1] 19
> 2 ^ 10
[1] 1024
> 3 == 4
[1] FALSE
```

3 Working Directory

The code samples above assume the data files are located in the **R working directory**, which can be found with the `getwd()` function.


```
> getwd()           # get the current working directory
```

You can select a different working directory with the `setwd()` function, and avoid entering the full path of the data files.

```
> setwd("<new_path>") # set the working directory
                        # to "<new path>"
```


The forward slash should be used as the path separator even on Windows platform.

```
> setwd("U:/MyDoc")   # set the working directory to "U:\MyDoc"
```





In your case, you'll save the data files or the  scripts on a folder of your personal drive (called "U:").




4 Session Management







4.1 The Workspace

All variables created in  are stored in a common workspace. To see which variables are defined in the workspace, you can use the function `ls` (list).


4.2 An Interactive Session

 runs on the commonly used platforms for personal computing: Windows, Mac OS X, Linux, and some versions of UNIX. In the usual desktop environments for these platforms, users will typically start  as they would most applications, by clicking on the  icon or on the  file in a folder of applications. An application will then appear looking much like other applications on the platform: for example, a window and associated toolbar. In the standard version, at least on most platforms, the application is called the “R Console”.

The application has a number of drop-down menus; some are typical of most applications (“File”, “Edit”, and “Help”). Others such as “Packages” are special to . The real action in running , however, is not with the menus but in the console window itself. Here the user is expected to type input to  in the form of expressions; the program underlying the application responds by doing some computation and if appropriate by displaying a version of the results for the user to look at (printed results normally in the same console window, graphics typically in another window).

This interaction between user and system continues, and constitutes an  session. Everything that you do interactively with  happens in a session. A session starts when you start up . A session can also be started from other special interfaces or from a command shell (the original design), without changing the fundamental concept. During an  session, you (the user) provide expressions for evaluation by , for the purpose of doing any sort of computation, displaying results, and creating objects for further use. The session ends when you decide to quit from .

5 Extension Packages

Sometimes we need additional functionality beyond those offered by the core  library. In order to install an extension package, you should invoke the `install.packages()` function at the prompt and follow the instruction.

```
> install.packages()
```

Here is a small list of useful packages:

- **gdata**: Various R programming tools for data manipulation
- **foreign**: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase, ...
- **MASS**: Support Functions and Datasets for Venables and Ripley’s MASS
- **e1071**: Misc Functions of the Department of Statistics (e1071), TU Wien
- **TeachingDemos**: Demonstrations for teaching and learning
- **RcppEigen**: Rcpp integration for the Eigen templated linear algebra library
- **RODBC**: ODBC Database Access
- **amap**: Another Multidimensional Analysis Package
- **Rcmdr**: R Commander
- **ISwR**: Data used for the book “Introductory Statistics with R”(Dalgaard, 2008)
- **HSAUR**: Data used for the book “A Handbook of Statistical Analyses Using R” (Everitt and Hothorn, 2010)

- **IPSUR**: Data used for the book “Introduction to Probability and Statistics Using R” (Kerns, 2010)
- **nutshell**: Data used for the book “R in a Nutshell” (Adler, 2010)

You can install the packages you need to use in your personal drive (“U:”).

6 Input / Output

6.1 A Kind of ‘hello world’ Program

A simple way for reading values from the terminal is to use the `readline` function.

```
> h<-" Hello "
> yourname<-readline("What is your name? ")
> print(paste(h, yourname))
```

The function `readline` reads a line from the terminal (in interactive use), the function `print` prints its argument and returns it, and the function `paste` concatenate vectors after converting to character (note that if we don’t want a space character between the two strings, we can use `paste0` or change the parameters of the `paste` function).

6.2 Data Import / Export

You will be able to read your data in many kinds of formats with the **foreign** package (data stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase, ...).

The sample data can be read in **comma separated values** (CSV) format with the `read.csv` function. Each cell inside such data file is separated by a special character, which usually is a comma, although other characters can be used as well.

The first row of the data file can contain the column names instead of the actual data.

Warning: In various European locales, as the comma character serves as decimal point, the `read.csv2` function should be used instead.

If your sample data is in Excel format (in ‘xlsx’ format from Excel 2007 version) and needs to be imported into R prior to use, you can use the functions from the **XLConnect** package (for reading data from an Excel spreadsheet and returning the value into a data frame). The following shows how to load an Excel spreadsheet named “mydata.xlsx”. As the package is not in the core R library, it has to be installed and loaded into the R workspace.

```
> library(XLConnect)
> mydata <- readWorksheet(loadWorkbook("mydata.xlsx"), sheet=1)
```

Consider the sample of the dataset of housing prices seen during the lecture:

| | | | |
|------|---|-------|-----|
| 2104 | 3 | 10050 | 400 |
| 1416 | 2 | 7534 | 232 |
| 1534 | 3 | 4305 | 315 |
| 852 | 2 | 2152 | 178 |
| 1990 | 4 | 9850 | 240 |

This dataset can be found as a CSV file on *Claroline* platform. We can read this file and import the data into R as a data frame and add the name of the attributes (with the property `colnames` of the data frame).

```
> hp <- read.table("housing_prices.csv", sep="," , header=FALSE, na.strings=".")
> colnames(hp) <- c("LivingArea", "Bedrooms", "GardenSize", "Price")
```

You can see the result by asking the class of *hp*, printing the value of this data frame or printing the summary of it.

```
> class(hp)
[1] "data.frame"
> hp
> summary(hp)
```


For saving the data frame in a file on the working directory, we can use the write.table function.

```
> write.table(hp, file = "housing.complete.txt", append = FALSE,
+             quote = TRUE, sep = ",", col.names = TRUE)
```

For a future use, we can simple import the file in a data frame with the read.file function or more simply with the read.csv function.

```
> hp <- read.table("housing.complete.txt", sep=",")
> hp2 <- read.csv("housing.complete.txt")
```

Exercise

- Go to the UCI Machine Learning Repository, find the real Housing Data Set and save the data in your personal folder.
- Open your downloaded file in Excel (or another spreadsheet) with the appropriate columns, and save it as a CSV file.
- Open your file in  as a data frame.
- Add the name of the columns to your data frame for each attributes (*CRIM*, *ZN*, *INDUS*, *CHAS*, *NOX*, *RM*, *AGE*, *DIS*, *RAD*, *TAX*, *PTRATIO*, *B*, *LSTAT* and *MEDV*).
- Print (on the screen) the summary of your data frame.
- Save your data frame into a CSV file on your personal folder.

7 Recommended Readings

The main literature for this section is:

- Adler (2010), “R in a Nutshell – a Desktop Quick Reference,” Chapters 1 to 15.
- Allerhand (2011), “A Tiny Handbook of R.”

- Albert and Rizzo (2012), “R by Example.”
- Braun and Murdoch (2007), “A First Course in Statistical Programming with R,” Chapters 1 to 4.
- Crawley (2005), “Statistics: An Introduction using R,” Appendix 1: “Fundamentals of the R Language.”
- Crawley (2007), “The R Book.”
- Cohen and Cohen (2008), “Statistics and Data with R: An Applied Approach Through Examples,” Part I “Data in statistics and R.”
- Dalgaard (2008), “Introductory Statistics with R,” Chapter 1 “Basics” and Chapter 2 “The R environment.”
- Everitt and Hothorn (2010), “A Handbook of Statistical Analyses Using R,” Chapter 1 “An Introduction to R.”
- Kerns (2010), “Introduction to Probability and Statistics Using R,” Chapter 2 “An Introduction to R.”
- Meys and de Vries (2012), “R For Dummies.”
- Teetor (2011), “R Cookbook.”
- ?, “The R language definition.”
- Venables et al. (2013), “An Introduction to R,” Chapter 1 “Introduction and preliminaries,” Chapter 2 “Simple manipulations; numbers and vectors,” Chapter 3 “Objects, their modes and attributes,” Chapter 5 “Arrays and matrices,” Chapter 6 “Lists and data frames,” Chapter 7 “Reading data from files.”
- Zuur et al. (2009), “A Beginner’s Guide to R.”

References

- Adler, J. (2010). *R in a Nutshell – a Desktop Quick Reference*. O’Reilly.
- Albert, J. and M. Rizzo (2012). *R by Example*. Use R! Springer.
- Allerhand, M. (2011). *A Tiny Handbook of R*. Springer.
- Braun, W. J. and D. J. Murdoch (2007). *A First Course in Statistical Programming with R*. Cambridge University Press.
- Cohen, Y. and J. Y. Cohen (2008). *Statistics and Data with R: An Applied Approach Through Examples*. John Wiley & Sons, Ltd.
- Crawley, M. J. (2005). *Statistics: An Introduction using R*. John Wiley & Sons, Ltd.
- Crawley, M. J. (2007). *The R Book*. John Wiley & Sons, Ltd.
- Dalgaard, P. (2008). *Introductory Statistics with R* (2nd ed.). Springer.

- Everitt, B. S. and T. Hothorn (2010). *A Handbook of Statistical Analyses Using R* (2nd ed.). Chapman & Hall / CRC.
- Kerns, G. J. (2010). *Introduction to Probability and Statistics Using R* (1st ed.).
- Meys, J. and A. de Vries (2012). *R For Dummies*. John Wiley & Sons, Ltd.
- Teetor, P. (2011). *R Cookbook*. O'Reilly.
- Venables, W. N., D. M. Smith, and the R Core Team (2013). An introduction to R –notes on R: A programming environment for data analysis and graphics.
URL <http://cran.r-project.org/doc/manuals/R-intro.html>.
- Zuur, A. F., E. N. Ieno, and E. Meesters (2009). *A Beginner's Guide to R*. Use R! Springer.