Session 1: Basics of working with `R`

My Name *123456789*

07 June, 2022

Highlights:

This is my mini-reflection. Paragraphs must be indented.
It can contain multiple paragraphs.

Threshold Concepts:

threshold concept 1
threshold concept 2
threshold concept 3
threshold concept 4

Introduction

NOTE: This is an R Markdown document. This type of document is a plain text file that can recognize chunks of code. When you execute code within the document, the results are displayed beneath. R Markdown files are *computational notebooks* which implement a coding philosophy called *literate programming*. Literate computing emphasizes the use of natural language to communicate with humans and chunks of code to communicate with the computer. By making the main audience other humans, this style of coding flips around the usual way in which code is written (computer is main audience, humans come second). This helps to make learning how to code more intuitive and accessible.

Let us first explain the anatomy of an R Markdown document.

At the top of the document is a *header* (called a YAML header) contained between two sets of three dashes `---`. The header includes metainformation about the document, including possibly the title of the document, the name(s) of the author(s), and how to process the document (e.g., what type of output to produce). After the header comes the *body* of the document, that is, the main contents of the document (text and code).

The header is quite an important compontent of the document, but we will for the moment take is as given and concentrate on the body of the document. This file, for example, introduces some basic instructions to work with `R` and R Markdown.

Chunks of code are key to writing computational notebooks. Whenever you see a chunk of code in an R Markdown document as follows, you can run it (by clicking the green 'play' icon on the top right corner of the code window) to see the results. Try it below!

```r
print("Hello, Workshop")
```

```
## [1] "Hello, Workshop"
```

You can see that the function `print()` displays the argument on the screen.

Notice the way the document speaks to you in natural language, and to the computer in `R`. The computer, instead of being the focus of the document, is an assistant to illustrate concepts (like, what is a chunk of code?).

Each of the documents in this workshop (the *Sessions*) is like a set of lecture notes. You can *knit* the document to produce a PDF file to study. The documents, on the other hand, are potentially much more than simple lecture notes, and they can in fact become the foundation for your own experiments and annotations. As you read and study, you can 'customize' the notes based on your developing understanding of the subject matter and/or to complement the document with other examples and information. To make the readings uniquely yours, you can type directly into the document (the original template is still available from the package, and if you need a fresh start, you can always create a new file with it). In addition, you can use the following style to create a *textbox*:

NOTICE:
This is an annotation. Here I can write my thoughts as I study, or I can add useful links or other information to help me learn.
To create a new paragraph, I need to type two blanks after the last one.

A textbox allows you to highlight important information. Try creating your own textbox next.
By now you will already have noticed a few conventions for writing in R Markdown:

- A chunk of R code begins with three backticks (“‘‘’) and the letter ‘r’ between curly brackets; a chunk of code concludes with three backticks.
- Hashtags (#) indicate headers: one hashtag is a primary header, two hashtags is a secondary header, three a tertiary header, etc.
- Asterisks are used for changing the font to *Italics*, **Bold**, and ***Italics+Bold***.
- Underscores do the same thing: *Italics*, **Bold**, ***Italics+Bold***.
- Dashes are used to create unnumbered lists.
- Hyperlinks can be introduced by using square brackets followed by the url in brackets.

There are some other typing conventions that you can find in this useful cheatsheet.

RStudio Window

If you are reading this, you must already have learned how to create a new project and documents using the templates in the workshop package. We can now proceed to discuss some basic concepts regarding data types and operations.

Preliminaries

It is good practice to clear the working memory in RStudio when you begin working on something new. This is to make sure that there is no extraneous info in memory that might confuse your work. The command in `R` to clear the workspace is `rm` (for ‘remove’), followed by a list of items to be removed. To clear the workspace from *all* objects, run the following chunk (by clicking on the ‘play’ icon):

```r
rm(list = ls())
```

Note that `ls()` lists all objects currently on the workspace.
An alternative way of doing this is to just click on the little broom icon in the ‘Environment’ tab (top right of your screen in the standard configuration of the RStudio app).
Next, we probably want to load all *packages* that we wish to use in our work session. A package is a unit of shareable code that augments the capabilities of base `R`. Packages can be installed from different sources; the most common one is CRAN (Comprehensive R Archive Network), but sometimes from GitHub repositories, among other places where developers put their packages. A big advantage of CRAN is that it keeps track of dependencies, that is, the ways in which packages depend on one another. This reduces the probability of packages not working.
When packages are in CRAN, they can be installed using `install.packages()`. A very useful set of packages is `tidyverse`, which includes code for data manipuation and visualization. If you have not done so before, you can install the package with `install.packages("tidyverse")`. As an alternative, you can install it from RStudio using the ‘Install Packages’ command in the ‘Tools’ menu, on the menu bar above. Simply click on ‘Tools - Install Packages…’ on the menu bar, and enter `tidyverse` in the window. Or you can use the ‘Packages’ pane in the app (in the bottom right).

Once the package is part of your *library*, it becomes available for use in your computer; to use the package afterwards, you must load it into memory. For this, we use the command `library()`, which is used to load a package, that is, to activate it for use, for example:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Data Frames in `R`

`R` can work with different data types, including scalars (essentially matrices with only one element), vectors (matrices of one dimension), and matrices where each dimension is greater than two. The following chunks of code illustrate the difference between these types of data.

This is a scalar:

```
1
```

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

Notice that a scalar is an object with only one row and one column. In contrast, this is a vector:

```
c(1, 2, 3, 4)
```

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

The vector above is a data object with one row and four columns. And this is a matrix:

```
matrix(c(1, 2, 3, 4, 0, 0, 0, 0, 1),
       nrow = 3,
       ncol=3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    0
## [2,]    2    0    0
## [3,]    3    0    1
```

This particular matrix has three rows and three columns, but matrices generally can be of size $n \times m$, meaning they can have n rows and m columns. The command `c()` is used to concatenate (i.e., to string together) the arguments (i.e., the inputs). The command `matrix()` creates a matrix using the data provided, and with the indicated number of rows `nrow` and columns `ncol`.

An important data *class* in `R` is a data frame. A data frame is a *rectangular* table that consists of rows and columns - commonly a set of vectors that have been put together for convenience. Data frames are used to store data in digital format. If you have used a spreadsheet software before, data frames will be familiar to you: they look a lot like a table in a spreadsheet.

A data frame can accommodate large amounts of information (several billion individual items, depending on the memory of a computer). The data can be numeric, alphanumeric (i.e., characters), logical (i.e., TRUE/FALSE), and so on. Each grid cell in a data frame has an address that can be identified based on the row and column it belongs to. R can use these addresses to perform mathematical operations. R labels columns alphabetically and rows numerically (or less commonly alphabetically).

To illustrate a data frame, we will create the following vectors for regions in Northern Italy. These data come from a survey of Ph.D. students expected to finish their studies between 2008 and 2014 (see Muscio and Ramaciotti, 2018). The first vector includes the names of the regions; the second vector contains the number of respondents to the survey; third is the number of active spinoffs founded by the students surveyed; next is the number of employees in those spinoffs; and lastly, there is a vector with the total number of active spinoffs in the regions in the period 2005-2006:

```r
nome_regione <- c("Emilia-Romagna",
                  "Friuli-Venezia Giulia",
                  "Liguria",
                  "Lombardia",
                  "Piemonte",
                  "Trentino-Alto Adige",
                  "Veneto")
phd_students <- c(52,
                  22,
                  17,
                  89,
                  36,
                  7,
                  46)
active_spinoff <- c(3,
                    4,
                    3,
                    5,
                    4,
                    0,
                    2)
employees <- c(13,
               7,
               8,
               3,
               16,
               0,
               1)
spinoff0506 <- c(393,
                 68,
                 21,
                 313,
                 322,
                 18,
                 306)
```

Note that `<-` is an assignment. In other words, it assigns the values on the right to the name on the left. By assigning some value to a name, we can easily retrieve the value later if needed.

After executing the chunk of code above, you will notice that five values appear in your Environment (upper right pane tab). These are five vectors of size 1:7 - one that is composed of alphanumeric information (or 'chr', for 'character'), and four that are numeric ('num').

These vectors can be assembled into a data frame. We do this for convenience, so we know that all these data belong together in some way. Please note: to create a data frame, the vectors must have the same length. In other words, you cannot create a table with vectors that have different numbers of rows. (Some other data types allow you to do this, but not data frames).

We will now create a data frame, which we will call `df` (not very imaginative!). You can, if you wish, choose a different name. There are some rules of thumb for naming objects (variable names must start with a letter and can be a combination of letters, digits, periods, and underscores). In most cases it helps if the names are intuitive, easy to remember, and not too long.

The function used to create a data frame is `data.frame()`, and its arguments are the vectors that we wish to collect in it. The next chunk of code assembles into a data frame the vectors that we created before:

```
df <- data.frame(nome_regione,
                 phd_students,
                 active_spinoff,
                 employees,
                 spinoff0506)
```

After running the chunk above, you will see a new object in your environment, a data frame called `df`. It is important to emphasize that data frames are *rectangular* tables: all vectors used to create a data frame must be the same length. We cannot have a vector with 6 rows and a vector with 7 rows in the same table. This is because each row represents an object of interest. In the case of our example, each row represents a *region*. Suppose that we had seven regions but only information for six of them. We could exclude the region that lacks information, or we could include it and indicate that other information about it is missing. The effect is the same, the vectors must be all the same size.

If you double-click on `df` in the Environment tab, you will see that this data frame has three columns (labeled `nome_regione`, `phd_students`, `active_spinoff`, `employees`, `spinoff0506`), and 7 rows. The row numbers and column names can be used to identify particular cells in the data frame.

You can enter data into a data frame and then use the many built-in functions of `R` to perform various types of analysis. You can also display the data frame by typing it in as an `R` command, like this:

```
df
```

```
##            nome_regione phd_students active_spinoff employees spinoff0506
## 1        Emilia-Romagna           52              3        13         393
## 2 Friuli-Venezia Giulia           22              4         7          68
## 3               Liguria           17              3         8          21
## 4             Lombardia           89              5         3         313
## 5              Piemonte           36              4        16         322
## 6   Trentino-Alto Adige            7              0         0          18
## 7                Veneto           46              2         1         306
```

The variable `nome_regione` at the moment is a character vector. If we summarize the data frame we can see this:

```
summary(df)
```

```
##  nome_regione        phd_students   active_spinoff   employees
##  Length:7           Min.   : 7.00   Min.   :0.0    Min.   : 0.000
##  Class :character   1st Qu.:19.50   1st Qu.:2.5    1st Qu.: 2.000
##  Mode  :character   Median :36.00   Median :3.0    Median : 7.000
##                     Mean   :38.43   Mean   :3.0    Mean   : 6.857
##                     3rd Qu.:49.00   3rd Qu.:4.0    3rd Qu.:10.500
##                     Max.   :89.00   Max.   :5.0    Max.   :16.000
##   spinoff0506
##  Min.   : 18.0
##  1st Qu.: 44.5
##  Median :306.0
##  Mean   :205.9
##  3rd Qu.:317.5
##  Max.   :393.0
```

We can turn the character variable into a *factor*.

What is a factor? Factors are a class of data used to store variables that are *categories* (e.g., labels, names, classes). Typically, this means that the variable has two or more *levels*. In the present case, the factor variable has seven levels, corresponding to one of seven unique regions in Northern Italy. If we had information for multiple periods of time, each region might appear more than once, for each year that information was available.

```
df$nome_regione <- factor(df$nome_regione)
```

Notice how we can reference columns in a data frame by referring to its name with `$` after the name of the table. We use the `<-` assignment again, to store the results of `factor(df$nome_regione)` into column `nome_regione` in data frame `df` (we could have stored it in a new data frame if we wished, but we do not really need to do that at the moment). The function `factor()` converts a variable into a factor. Compare the summary of the factor to the summary of the character variable:

```
summary(df)
```

```
##                 nome_regione  phd_students   active_spinoff   employees
##  Emilia-Romagna       :1     Min.   : 7.00   Min.   :0.0    Min.   : 0.000
##  Friuli-Venezia Giulia:1     1st Qu.:19.50   1st Qu.:2.5    1st Qu.: 2.000
##  Liguria              :1     Median :36.00   Median :3.0    Median : 7.000
##  Lombardia            :1     Mean   :38.43   Mean   :3.0    Mean   : 6.857
##  Piemonte             :1     3rd Qu.:49.00   3rd Qu.:4.0    3rd Qu.:10.500
##  Trentino-Alto Adige  :1     Max.   :89.00   Max.   :5.0    Max.   :16.000
##  Veneto               :1
##   spinoff0506
##  Min.   : 18.0
##  1st Qu.: 44.5
##  Median :306.0
##  Mean   :205.9
##  3rd Qu.:317.5
##  Max.   :393.0
##
```

How would you explain the difference between characters and a factor?

More Basic Operations

`R` can perform many types of operations. Some operations are simply arithmetic (sums, multiplications, etc.) Others are logical, and return as a result values of TRUE/FALSE. And so on.

To perform operations effectively, it is useful to understand the way `R` locates information, for instance in a data frame. As noted above, each grid cell has an address, otherwise known as an *index*, that can be referenced in several convenient ways. For instance, assume that you wish to reference the first value of the data frame, say, row 1 of column `nome_regione`. To do this, you would call the data frame that you are interested in, and use square brackets with two values, the row and the column that you wish to refer to, separated by a comma:

```
df[1, 1]
```

```
## [1] Emilia-Romagna
## 7 Levels: Emilia-Romagna Friuli-Venezia Giulia Liguria Lombardia ... Veneto
```

This will recall the element in the first row and first column of `df`.

As an alternative, you could type:

```
df$nome_regione[1]
```

```
## [1] Emilia-Romagna
## 7 Levels: Emilia-Romagna Friuli-Venezia Giulia Liguria Lombardia ... Veneto
```

As you see, this has the same effect. Again, the string sign `$` is used to reference columns in a data frame. Therefore, `R` will call the first element of `nome_regione` in data frame `df`.

Each region is referenced by the number inside the brackets. So, the second region in the table is:

```
df$nome_regione[2]
```

```
## [1] Friuli-Venezia Giulia
## 7 Levels: Emilia-Romagna Friuli-Venezia Giulia Liguria Lombardia ... Veneto
```

Asking for `df[1,1]` is identical to asking for `df$nome_regione[1]`. Create a new chunk of code to recall the full column with the region names (i.e., by typing `CO$Country`).

Indexing is useful to conduct operations. Suppose for instance, that you wished to calculate the total number of active spinoffs in the period 2005-2006 of two regions, say Lombardia and Veneto. You can execute the following instructions:

```
df$spinoff0506[4] + df$spinoff0506[7]
```

```
## [1] 619
```

An issue with with indexing cells this way is that, if other regions were added to the table, the row numbers of each region might change, and as a consequence you might not be referencing the same regions with those numbers. Another possibility is if the table is very long, you might not even know which region is in which row to begin with.

So a better way to index the cells in a data frame is by using logical operators, like in the following chunk of code. Here, we are essentially asking for "population of (country which is France)" + "population of (country which is Germany)":

```
df$spinoff0506[df$nome_regione == "Lombardia"] + df$spinoff0506[df$nome_regione == "Veneto"]
```

```
## [1] 619
```

The text inside the square bracket tells `R` to look at the row with that region's names (we index by the name of the region, instead of the row number), and the text before the square brackets asks to look the number in the `spinoff0506` column for the row indexed.

Suppose that you wanted to calculate the total number of spinoffs in 2005-2006 in this data frame. To do this, you would use the instruction `sum`, and use the `$` to identify the column that you want to sum:

```
sum(df$spinoff0506)
```

```
## [1] 1441
```

As you can see, `R` can be used as a calculator, but it is much more powerful than that.

You have already seen how `R` allows you to store in memory the results of some instruction, by means of an assignment `<-`. You can also perform many other useful operations. For instance, you can find the maximum for a set of values:

```
max(df$spinoff0506)
```

```
## [1] 393
```

This does not have to be just the maximum of a column. You can ask for the max of any set of values:

```r
max(df$spinoff0506[df$nome_regione == "Lombardia"], df$spinoff0506[df$nome_regione == "Veneto"])
```

```
## [1] 313
```

And, if you wanted to find the name of the region with the largest number of spinoff companies, you can do this:

```r
df$nome_regione[df$spinoff0506 == max(df$spinoff0506)]
```

```
## [1] Emilia-Romagna
## 7 Levels: Emilia-Romagna Friuli-Venezia Giulia Liguria Lombardia ... Veneto
```

As you see, Emilia-Romagna is the region with the most spinoff companies in Northern Italy in 2005-2006. Likewise, the function for finding the minimum value for a set of values is `min`:

```r
min(df$spinoff0506)
```

```
## [1] 18
```

So which of the regions in the data frame had the smallest number of spinoffs in 2005-2006?

```r
df$nome_regione[df$spinoff0506 == min(df$spinoff0506)]
```

```
## [1] Trentino-Alto Adige
## 7 Levels: Emilia-Romagna Friuli-Venezia Giulia Liguria Lombardia ... Veneto
```

The data can be explored in fairly sophisticated ways by using indexing in imaginative ways.

Try calculating the mean number of spinoffs using the command `mean()`. To do this, create a new chunk of code and type your code. The keyboard shortcut to insert code chunks into R Markdown files is CTRL-ALT-I.

Data Analysis

A powerful feature of `R` is the flexibility to use calculations to implement data analysis. Your sample data frame contains information on population and number of cars per Country. Suppose that you would like to discover which country has the highest car ownership rate. Sure, big countries will have more cars. But in relative terms, is this still true?

We will define the mean spinoff size by region as the number of employees per spinoff in each region. This is caclculated as:

$$MS_i = \frac{employees_i}{spinoffs_i}$$

Where MS_i is the mean spinoff size in region i.

Here we introduced another feature of R Markdown documents: the text above between the `$$` signs is a piece of LaTex code. It allows you to type mathematical formulas in an R Markdown document; it does not execute any commands, and is not a piece of `R` code at all. Mathematical expressions can be written inline by using the notation x. Do not worry too much about how to write mathematical expressions at the moment, this is something that you can learn when and if needed, and there are many resources online to browse.

We can ask `R` to calculate the mean spinoff size as follows:

```r
MS <- df$employees/df$active_spinoff
```

This has created a new vector called `MS`. If you wanted to add this vector to the data frame as a new column, you could do the next:

```
df$MS <- df$employees/df$active_spinoff
```

We can create new columns on the fly and assign stuff to them, as long as the size of what we are assigning is compatible with the data frame (i.e., same number of rows). You can check that the new column was added to your existing `CO` data frame:

```
df
```

```
##            nome_regione phd_students active_spinoff employees spinoff0506
## 1        Emilia-Romagna           52              3        13         393
## 2 Friuli-Venezia Giulia           22              4         7          68
## 3               Liguria           17              3         8          21
## 4             Lombardia           89              5         3         313
## 5              Piemonte           36              4        16         322
## 6   Trentino-Alto Adige            7              0         0          18
## 7                Veneto           46              2         1         306
##         MS
## 1 4.333333
## 2 1.750000
## 3 2.666667
## 4 0.600000
## 5 4.000000
## 6      NaN
## 7 0.500000
```

As you can see, one of the values in column `MS` is `NaN`: this is the result of dividing by zero.

By the way, if you want to round off data, you can use the `round()` command. Here, we round to two decimals:

```
df$MS <- round(df$employees/df$active_spinoff, 2)
```

Knitting

Knitting is the process of converting the R Markdown document (source) into some other type of document (output). The output could be HTML, a Word file, or a PDF file. The

Practice

1. What was the approximate population of Earth circa 2016? How many barrels of oil per day did this population consume? (5)

2. What proportion of the world's oil is consumed by Canada and the US? Do these two countries have the same per capita levels of demand for energy use? Why do you think this is so? (10)

3. Calculate the consumption of oil in barrels per day per 1,000 people for the countries listed in the data frame. What type of pattern do you observe? Please explain the pattern. Hint: rank countries according to their oil consumption per 1,000 people. You can use the `order()` function to do this. (15)

4. Use simple regression analysis to assess the relationship in the dataset between a country's per-capita CO2 emissions (for 2015) and per-capita oil consumption. Start by creating a scatterplot of these two variables; then add labels to your plot (try changing the `size` of the text for clarity). What does a linear regression say is the relationship between per-capita CO2 emissions and per-capita oil consumption? How significant is the relationship, according to the regression model? Which countries are cleaner and which countries are less clean in their use of energy? (25)

5. How much CO2 would be produced, if every person in the world produced CO2 equal to that produced typically by one Canadian? What if every person in the world produced CO2 equal to that produced by a Chinese citizen in 2015? Compare both results to the actual CO2 production of the world in 2015. Hint: refer to your linear regression model to discuss this. (15)

6. Using the data in the data frame, find the six largest CO2 emitters in 1995, and then calculate the increase or decrease in the amount of generated CO2 emissions for each of them in the time period 1995 to 2015. Compare the size of the numbers. Which of these countries saw the largest increase in emissions since 1995? What could explain the changes? (10)

7. Which country in the world had the highest CO2 emissions per square kilometer of land area in 2015? Which country has the lowest? (5)