Chapter 2

Introduction to R

This chapter introduces R as a statistical computing environment and familiarizes the reader with RStudio as an integrated development interface for working efficiently with data. It covers the fundamentals of data management, including how to import, manipulate, and organize data sets for analysis. The chapter also introduces plotting and graphical tools in R for visualizing data, followed by an overview of basic statistical concepts and methods that form the foundation for applied data analysis.

2.1 R Resources and Help

Very large user community for R. Google search for "Some topic R" usually leads quickly to the desired help. Here are the links to a few online tutorials

- UCLA Institute for Digital Research and Education
- StatMethods

Two online resources will provide you the solution to the vast majority of your R questions. Getting on those websites is usually the result of a Google search.

- Statistical Data Analysis R: This resource contains the function manual for R/RStudio including all packages. Example for a function boxplot. The most helpful part are the examples at the bottom of the page.
- Stack Overflow: Resources for developers. For example, a Google search for "r ggplot two y axis" may give you the following result. Note that all questions on Stack Overflow have to be accompanied by a re-creatable dataset.

There are also many R books on GitHub:

- Principles of Econometrics with R
- Introduction to Econometrics with R

- Geocomputation with R
- Introduction to Data Science
- Forecasting: Principles and Practice

Besides many online resources, there are also three useful textbooks:

- Applied Econometrics with R by Christian Kleiber and Achim Zeileis
- Introductory Statistics with R by Peter Dalgaard
- An Introduction to Statistical Learning with Applications in R by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani

An additional online tutorial is Using R for Introductory Statistics by John Verzani. If you prefer a video, the following Introduction to R and RStudio has been proven useful for people learning R/RStudio.

2.2 Opening RStudio

Work in RStudio is done in four windows:

1. Script Window
   - This is were you type your R Script (.R) and where you execute commands.
   - Comparable to do-file/editor in Stata.
   - This window needs to be opened by File ⇒ New File ⇒ R Script.
2. Console window
   - Use of R interactively. Should only be used for quick calculations and not part of an analysis.
3. Environment
   - Lists all the variables, data frames, and user-created functions.
   - It is tempting to use the "Import Dataset" function …Don't.
4. Plots/Packages/Help

There is a base version of R that allows doing many calculations but the power of R comes through its packages. To use functions associated with a particular package, click "Install'' in the packages window of RStudio and type in the name of the package. Or alternatively, use

```
install.packages("ggplot2")
```

To use a package, you have to activate it by either checking the box in the window "Packages" or by including `library(packagename)`. Those packages are updated on a regular basis by users.

The `#` allows you to include comments in your script file that are not read by R. It is good practice to start any new script with clearing the memory using the command `rm(list=ls())`. Use the command `get()` to determine the current working directory or set the new working directory with the command `setwd()`, e.g., `setwd("E:/")`. For file paths, replace \ with /. Next, you want to load all libraries necessary for your entire script file with the command `library()`. It

is also good practice to save your R-script on a regular basis. The frontmatter, i.e., the top of a R-script file, could look as follows

```
rm(list=ls())
load("DataAnalysisPAData.RData")
library(openxlsx)
```

2.2.1 In-class Exercise 1

Create a R-script file with the following components:

1. Two lines for the title and the date (use #)
2. Clearing all current contents
3. Setting the correct working directory
    - This should be a folder to which you have downloaded all materials.
4. Installing and loading the package `openxlsx`.

2.3 Functions

At the core of R are functions that "do things" based on your input. The basic structure is

- `object = functionname(argument1=value,argument2=value,...)`

The structure has the following components

- `object`: Output of the function will be assigned to object.
- `functionname`: Name of the system function. You can also create and use your own functions. More about this later.
- `argument`: Arguments are function specific.
- `value`: The value you want a particular argument to take.

If a function is executed without an specific assignment, the output will be displayed in the console window. Before using a function, read the documentation. Many functions have default settings. Be aware of default values. In most cases, those defaults are set to values that satisfy most uses. For example, consider the help file for the function t.test.

- `t.test(x,y=NULL,...,mu=0,conf.level=0.95,...)`

For this function we have the following default values

- `y=NULL`
- `mu=0`
- `conf.level=0.95`

2.4 Data in R

The main data types which can appear in the Environment window of R are vectors, matrices, data frames, and lists. Vectors are one-dimensional data structures that store elements of the same type (e.g., all numeric or all character) and form the basic building blocks for most objects in R.

```
preselection   = seq(1788,2016,4)
midterm        = seq(by=4,to=2018,from=1790)
```

Matrices are two-dimensional structures with rows and columns, where all elements must be of the same type, making them useful for mathematical and linear algebra operations. Note that only numerical values are allowed.

```
somematrix     = matrix(8,10,4)
```

Data frames are tabular data structures where each column can have a different data type, but all columns have the same length, making them ideal for representing data sets with multiple variables. Data frames are by far the most common data type in R and are comparable to Excel spreadsheets. Lists are flexible containers that can hold elements of different types and lengths, including other lists, making them useful for organizing complex or hierarchical data.

```
myfirstlist = list(preselection,midterm,somematrix)
```

2.4.1 Using R as a Calculator

Entering heights of people and storing it in a vector named `height`:

```
height = c(71,77,70,73,66,69,73,73,75,76)
```

Calculating the sum, product, natural log, mean, and (element-wise) squaring is done with the following commands:

- `sum(height)`
- `prod(height)`
- `log(height)` # Default is the natural log
- `meanheight = mean(height)`
- `heightsq = height^2`

Removing (i.e., deleting) unused elements: `rm(heightsq,meanheight)`

2.4.2 Creating a Data Frame from Scratch

Data frames are the most commonly used tables in R/RStudio. They are similar to an Excel sheet.

- Column names represent the variables and rows represent observations.
- Column names must be unique and without spaces.

Suggestion: Use only lower-case variable names and objects.

```
studentid       = 1:10
studentnames    = c("Andrew","Linda","William","Daniel","Gina",
                    "Mick","Sonny","Wilbur","Elisabeth","James")
students        = data.frame(studentid,studentnames,height)
rm(studentid,height,studentnames)
```

2.4.3 In-class Exercise 2

Create a data frame called `students` containing the following information:

Name	Economics	English
Mindy	80.0	52.5
Gregory	60.0	60.0
Shubra	95.0	77.5
Keith	77.5	30.0
Louisa	97.5	95.0

Notes:

- Use *name* as the column header for the students' names.
- Once you have created the data frame, remove the unused vectors.

2.4.4 Indexing

Indexing refers to identifying elements in your data:

- For most objects: `students[row number,coloumn number]`
  - `students[3,2]` returns 95. What does `students[3,]` return?
- If you want to select certain columns: `students[c("name")]`
  - Other example: `students[c("name","english")]`
- Selecting results based on certain conditions: `students[which(students$economics>80),]`

Referring to a particular column in a data frame is done through the dollar symbol:

- `students$english`
- You will use this functionality very often.

Creating a new column: `students$average = rowMeans(students[c("economics","english")])`

2.4.5 Importing Data into R

In almost all cases, the data is imported into R from an external data set. The data has to be "machine-readable" which means that the first row must contain the variable names and the actual data starts in the second row. Machine-readable data can be imported as follows:

- `read.csv("filename.csv")`: If you have a comma separated value (.csv) file then this is the easiest and preferred way to import data.

- `readWorkbook(file="filename.xlsx",sheet="sheet name")`: Requires the package openxlsx. Note that there are many packages reading Excel and this is one of the most reliable and user-friendly.
- Importing data from other software packages (e.g., SAS, Stata, Minitab, SPSS) or .dbf (database) files can be achieved using the package foreign. The package works also for Stata data up to version 12. To important data from Stata version 13 and above, the package readstata13.

2.4.6 Sub-setting a Data Frame

To extract variables or observations based on certain criteria, the command `subset()` must be used. Consider the data `vehicles`. Extracting vehicle information only for the year 2015 is done as follows.

```
cars2015 = subset(vehicles,year==2015)
```

Note that the double equal sign conducts a logical test. Using a single equal sign does not extract any data and simply returns the original data without (!) an error message. To list all the distinct values in a column, the command `unique()` can be used. This command only makes sense in the case of categorical data in a particular column. For example, listing all EPA vehicle size classes ($VClass$) can be accomplished as follows.

```
unique(cars2015$vclass)
```

Suppose you are only interested in the variables $co2tailpipegpm$ and $vclass$ for the model year 2015.

```
cars2015 = subset(vehicles,year==2015,select=c("co2tailpipegpm","vclass"))
```

Suppose you are only interested in "Compact Cars" and "Large Cars" in the column $VClass$ for the year 2015. There the notation is a bit odd (note that the many line breaks are not necessary to include in R):

```
cars2015 = subset(vehicles,
                  year==2015 & vclass %in% c("Compact Cars", "Large Cars"),
                  select=c("make","co2tailpipegpm","vclass"))
```

2.4.7 In-class Exercises 3

From the vehicles data set, extract the GHG Score and the vehicle class from the 2014 model year for the following manufacturers: Toyota, Ford, and Audi. Your new data set should contain the following columns: $ghgScore$, $make$, and $VClass$. Is the resulting data frame sensible or do you see a problem?

2.4.8 Aggregating Data

To aggregate data based on a function, e.g., sum or mean:

```
cars2015 = aggregate(co2tailpipegpm~make+vclass,
                     FUN=mean,data=cars2015)
```

2.4.9 Writing Data Frame to .csv-File

To write data to the current working directory

```
write.csv(cars2015,"cars2015.csv",row.names=FALSE)
```

Using the option `row.names=FALSE` avoids an index column in the output file.

2.4.10 Reshaping Data from Long to Wide and Viceversa

R has the ability to reshape data from long to wide format and back. For this demonstration, we use the data in `compactcars` and the command `reshape()` from the package reshape2:

```
cars = melt(compactcars,id=c("year","make","model","displ","drive"))
```

Reshaping the data is generally very useful but also tricky. For detailed information see the section How can I reshape my data in R.

2.4.11 Extending the Basic `table()` Function

The required package to extend the basic `table()` function is called gmodels. Compare the outputs of the functions `table()` and `CrossTable()`. The commands below do the following:

- Standard `table()` function.
- The function `CrossTable()` includes the proportions along the two dimensions of gun ownership and gender. Note that a description of the cell content is at the top of the results page.

```
df        = subset(gss,year==2022,
                   select=c("owngun","sex"))
table(df$owngun,df$sex)
```

```
##
##       1   2
##   1 424 331
##   2 623 879
##   3  27  17
```

```
CrossTable(df$owngun,df$sex,prop.chisq=FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
```

```
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  2301
##
##
##                | df$sex
##    df$owngun |         1 |         2 | Row Total |
## -------------|-----------|-----------|-----------|
##             1 |       424 |       331 |       755 |
##               |     0.562 |     0.438 |     0.328 |
##               |     0.395 |     0.270 |           |
##               |     0.184 |     0.144 |           |
## -------------|-----------|-----------|-----------|
##             2 |       623 |       879 |      1502 |
##               |     0.415 |     0.585 |     0.653 |
##               |     0.580 |     0.716 |           |
##               |     0.271 |     0.382 |           |
## -------------|-----------|-----------|-----------|
##             3 |        27 |        17 |        44 |
##               |     0.614 |     0.386 |     0.019 |
##               |     0.025 |     0.014 |           |
##               |     0.012 |     0.007 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1074 |      1227 |      2301 |
##               |     0.467 |     0.533 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Note that for almost any R command, you can store the output by assigning it to an object:

- `somename = CrossTable(gssgun$owngun,gssgun$sex,prop.chisq=FALSE)`

2.4.12 Merging Datasets

Consider two data sets from school districts in Ohio:

- `ohioscore` contains an identifier column IRN and a score that indicates the quality of the school.
- `ohioincome` contains the same identifier than the previous sheet in addition to median household income and enrollment.

To merge the two data frames based on the column IRN, the function `merge()` must be used:

```
ohioschool = merge(ohioscore,ohioincome,by=c("irn"))
```