Introduction to Quantitative Methods

Exercise 1.2: Further practice with the R software

Introduction

In this exercise we introduce the practical steps of the exercises, and in particular take first steps in learning how to use the R software. We will use one dataset and a few simple statistical analyses as examples of these steps, but otherwise the focus of this exercise is not yet on the statistics, as it will be in the rest of the exercises.

After you have gone through the exercise, please also answer the multiple-choice questions for this week on Moodle, as discussed at the end of these instructions.

Before you can start this exercise, you need to have downloaded and installed R and RStudio, as explained in exercise 1.1.

Initial steps

We suggest that you create a dedicated file folder for these exercises on your computer, where you will place the exercise files which you will download from Moodle. This will also be your "working directory" in R, as explained below.

At the start of this and all the remaining course exercises, you should carry out the following steps:

A. Download the files for that week's exercise from Moodle to the folder you use for them. Today there are two files:
    - A data file *MediaTimeUse2.csv*. It contains the data from the time use survey which was also used in the first lecture. It is a spreadsheet (.csv) file, so you can also take a look at it in Excel or other spreadsheet software.
    - An R script file *Exercise 1.2.R*.

B. Start RStudio.

C. Set the *working directory* for R to be the folder where you downloaded the files. This can be done, for example, by selecting the folder under the drop-down menu selection *Session/Set Working Directory/Choose Directory* at the top in RStudio.

D. Optionally, load a workspace file, from menu *Session/Load Workspace*. This restores a "workspace" which contains objects which have been created in a previous R session, so that they are again available in this session (more explanation of this is given below). If you skip this step, you will start with an empty workspace.

E. Open the script file that you downloaded, using the menu *File/Open file*. It opens in the top-left window in RStudio.
   - You can execute commands in the script by moving the cursor to the required line and pressing Ctrl + Return (in Windows) or Mac key + Return (on a Mac). If you want to run multiple commands at once, you first highlight the corresponding lines and then press these keys.
   - In later exercises you will often be asked to edit the script file to add new commands to it. Then remember to save the file again to keep these changes.
   - Individual commands can also be given by entering them in the Console window (bottom left in RStudio).

You are now ready to start this exercise, following the instructions on the next page.

More explanation of workspace files: The *workspace* of an R session contains all the objects (data, functions, etc.) which have been created in that session or loaded from a previous one. If you want to keep them available even after you finish the session, you should save the workspace. It will be saved in a file with an extension *.RData* in its name. When you then start a new session, you can re-instate the workspace by loading this file, as explained under D above. There are several alternative ways of saving a workspace file:

- When you finish a session and ask to close R, you are asked (if anything in the workspace has changed) if you want to save the current workspace. If you answer Yes to this question, the workspace will be saved as a file called *.RData* in the current working directory (*note*: this unusual-looking file name, which starts with "." with nothing before it, is the name that R uses by default for this file).
- You can also save the workspace at any point during the session:
  - If you give the command `save.image()`, the file will again be saved as the file *.RData* in the working directory.
  - You can also save the file under a different name, by using the menu *Session/Save Workspace As*, or by giving a command like

```
save.image(file="week1.RData")
```

A shortcut: There are several equivalent ways of starting R and RStudio. The steps above make it clear what is happening, but some other ways are quicker. If I already have a workspace file (such as *.RData*) from a previous session, what I normally do is the following:

- Go to the folder which contains that file.
- Double-click on the name of the file. If Rstudio has been associated with files of type .RData (which normally happens when RStudio is installed), this will open RStudio, set the working directory to be the folder where that file is, and load the workspace in the file to the R session. In other words, this one step combines B, C and D above.

Instructions for Exercise 1.2

In each exercise on this course, you will follow the step-by-step instructions and explanations like the ones below, alongside the script file in RStudio (the same numbering of the exercises is used in both). For some exercises the necessary R commands are included in the script, so all you need to do to be able to follow the explanations is to run the commands in order (this is the case for all the steps today). For other exercises, the instructions will ask you first to change the script by adding or editing commands in it, and then run those commands. The idea is that you first learn how to do something in R from the examples shown in the script file provided, and then edit it to perform a new analysis of your own.

The lines in the script which begin with #, and which are shown in green in RStudio, are comments rather than commands. Please read them as well, because they often provide additional information which is not included in the instructions. It is good practice to add your own commands where you think it useful for remembering what the script does.

Because the focus of today's exercise is on learning some basic ideas of the R language, the text of today's instructions is longer and more detailed than it will be for later exercises. Below is a description and explanation of the commands in the script file for this exercise.

1. Reading in a data set to R. Here this is done with the command

```
MediaTimeUse <- read.csv("MediaTimeUse2.csv")
```

   - Here *MediaTimeUse2.csv* is the file, in the working directory, which contains the data. Here it is a .csv file but R can read other file types too.
   - read.csv is an R *function* which does the job of loading files of this type into R.
   - The *"MediaTimeUse <-"* is an *assignment*, meaning that the file that is loaded by the call to the function *read.csv* is then saved in the current R workspace as an object called *MediaTimeUse* (in technical R terms, this is an object of the type "Data frame").
   - The *View(MediaTimeUse)* command in the script opens another window where you can see what this loaded dataset now looks like.

2. First example of statistical analysis of data. Here we consider one variable in the time use dataset that we have now read in to R. It is called *Work*, and it indicates the labour market status of each respondent. We first create a simple table of frequencies (counts of how many respondents in the sample have each labour market status) and a corresponding table of proportions (the frequencies divided by the total number of respondents). These methods will be discussed further in the next exercise. Here they are used just as illustrations of using R.
   - The first command in the script is

```
table(MediaTimeUse$Work)
```

     - Here *MediaTimeUse$Work* identifies the single variable *Work* in the data frame *MediaTimeUse*. Note that we pick the particular variable (*Work*) from the dataset (*MediaTimeUse*) with the '$' sign.
     - *table* is a function which produces a table of frequencies for a variable given as input (or *argument*) for the function, in this case for

MediaTimeUse$Work. (Technically, the result is a "vector" object, which is then printed out.)
    - When you run this command, its result is shown in the Console window in RStudio.
- The next command is

```
prop.table(table(MediaTimeUse$Work))
```

    - Here *prop.table* is a function which produces a table of proportions from a table of frequencies. Here that table of frequencies is produced by first calling the *table* function on the variable *MediaTimeUse$Work*. In other words, here the two functions work in a chain: first *table* does the frequencies and passes them on to *prop.table*, which then calculates the proportions.
- These two steps are shown more explicitly in the next set of commands:

```
empl.table <- table(MediaTimeUse$Work)
prop.table(empl.table)
```

    - This produces the same table of proportions as the previous command.
    - The difference here is that the table of frequencies that is produced by *table* is first assigned to a new object named *empl.table*, and that object is then given (by name) as an argument to *prop.table*.
    - This way, *empl.table* remains in the workspace, in case you may want to use it for something else later (but if you know you won't, the first version of the command is simpler). *empl.table* is the name we have chosen but we could have used practically any name, it is just a name for the table.

3. Example of plots in R. Here we draw a bar chart of the frequencies of the variable *Work*.
    - The first command in the script is

```
barplot(table(MediaTimeUse$Work))
```

    - This calls the function *barplot* to draw the plot. Its argument is a table of frequencies, which is in turn produced here by a call to the *table* function.
    - The plot appears in the Plots window in RStudio.
    - R has many powerful functions for creating different kinds of plots. All of them have many additional arguments for modifying the plots in various ways. An example of the use of such additional arguments is the next command:

```
barplot(table(MediaTimeUse$Work),
        col="lightblue",
        xlab="Respondent's labour market status")
```

    - Here *col* is an argument which changes the colour of the bars in the chart, and "lightblue" is the value we choose to give to it here. There are many different colours that we could choose.
    - Similarly, *xlab* is an argument which puts a title of text under the plot, here we have chosen the title "Respondent's labour market status".

4. How then can we learn about such different functions and their arguments in R?
    - For functions, the usual starting point is seeing examples of them, like the ones you will see in these exercises and in the materials mentioned on the course Moodle page. That way, you will gradually build up your knowledge of what exists in R. When you want to find out how to do particular statistical methods in R, internet searches can be useful (for example, try googling "R bar chart").
    - Such examples can also be helpful for finding out about the arguments of a function. However, when you know which function you are using, a more thorough source of information is its *help file*. It can be found by using the *help* function, which can also be shortened to *?*. For example, the commands

      ```
      ?barplot
      ```

      and

      ```
      help(barplot)
      ```

      both show the help text for the *barplot* function; it appears in the Help window in RStudio.
    - Here we have put these commands in the script file, and you can run them from there. Usually, however, we would ask for help files by giving the same commands in the Console window (enter the text at the cursor, and press Return). The same is true of any one-line commands that you may want to give during an R session, if you don't need to save them in a script file.
    - The text of a help file is very highly structured, because it is meant to be a comprehensive and standardised description of a function. It is not easy to read at first, and we do not expect it to make very much sense to you today. However, it is well worth getting into the habit of looking at the help files, because as you gain more experience they can become a good way of expanding your understanding of what R can do.
    - Note that the end of the help text there are examples of how a function and its arguments may be used. These can be very helpful.
    - As first practice here, please read what the help text for *barplot* says about the arguments *col* and *xlab* which we used above.

5. R packages. The functions we have used so far are part of "Base R", which is what you got when you first installed R. In addition, many add-on *packages* have been written for R (16,273 of them, at the time of writing; see https://cran.r-project.org/index.html, under "Packages"). Some of these packages were written by the core team of developers of R (and many were included when you installed R), but many are written by other users of R. Some of them implement advanced statistical methods which are not part of Base R. Others combine existing features of R in various ways, for example to achieve alternative forms of input or output for things that could also be done in Base R. Most of the packages we will use for these exercises are of this latter kind. In this exercise we illustrate this with the *descr* package, using it to obtain more attractive output for the tables and plots in Exercises 2 and 3 above.

   - First, you need to download and install the package, using the *install.packages* function. For the *descr* package, the command is
     ```
     install.packages("descr")
     ```
     You only need to do this once for each package on each computer you use. After that, you can comment this out from the script file, by adding # in front of the line [i.e. # install.packages("descr")]
   - You can see a list of the packages which have so far been installed on your computer under the *Packages* tab in RStudio.
   - General information on any installed package, including the functions which are included in it, can be found with the *help* function, as in
     ```
     help(package="descr")
     ```
   - After a package has been installed, you need to load it in any R session that you want to use it for, using the *library* function. Here it is
     ```
     library(descr)
     ```
     If a library has not been installed, this will give an error message. You should then install package first as explained above, and then load.
   - Now that the *descr* package has been loaded, the functions in it are available to you. The next command uses the *freq* function from *descr*:
     ```
     freq(MediaTimeUse$Work)
     ```
     This produces both a neatly formatted version of the table, including the frequencies and proportions (as percentages), and the bar chart.
   - This function again has its own additional arguments. In particular, any arguments for *barplot* can also be used here (internally, *freq* simply calls *barplot* and passes these arguments on to it). You can see the results of this if you run the last command
     ```
     freq(MediaTimeUse$Work,
          col="lightblue",
          xlab="Respondent's labour market status")
     ```

Questions on the exercise

For each week's topic, there is a set of multiple-choice questions on Moodle for you to answer to check your learning. Some are general questions on the methods, and some are about the exercise exercises. To find out the answers to some of the latter, you are often required to carry out some further analysis, using and adapting the methods that you have practiced in that exercise.

The questions on the exercise exercises (but not the multiple choices) are also given at the end of the exercise instructions, so that you can determine the answers to them before you attempt them on Moodle.

This week, the quiz includes also a set of questions on measurement levels of variables, and other concepts discussed in the lectures. They are not listed here. The following four questions are about the topics of this exercise exercise:

1. *What does the argument* **main** *of the function* barplot *do?*

2. *Suppose you wanted to draw a bar chart where the bars were horizontal rather than vertical. Which argument setting for* barplot *would achieve this?*

3. *Suppose you changed the first command in Exercise 5 to say*
   ```
   freq(MediaTimeUse$Work,plot=FALSE)
   ```
   What would this do?

4. *How many of the respondents in this sample are unemployed?*

In the other exercises after this, all or most of the exercises on the exercises will be about the statistics rather than R, i.e. they will be like question 4 (except usually a bit more difficult) than like 1-3.