FIT2086 Studio 1
Introduction to R and Descriptive Statistics

Daniel F. Schmidt, with material from Enes Makalic

July 21, 2018

Contents

1 Introduction

R is a popular, free software environment and programming language for statistical computing and computer graphics. The R language is widely used in the machine learning and statistics communities for analysing data and developing novel machine learning algorithms and statistical software. The R software environment is open source and runs on all modern operating systems, including MS Windows, Mac OSX and Linux. In this workshop, we will be learning how to use R[W] with a program called RStudio[W], a powerful user interface to R that is designed to make programming with R more intuitive and easier for the users.

2 Introduction to R

Before we can begin using R, we need to start the program RStudio. After RStudio is started, you will see several windows, the most important of which is the Console window. You will be typing most R commands in this window as well as using the window to examine the command results. R is powerful statistical programming language that can also be used as a calculator.

1. Lets try a few simple calculations:

```
1 + 1
(log(10) - 1/5) * 3
sin(pi^2)
```

2. R understands all the standard mathematical operations (e.g., addition, subtraction, multiplication, etc.), mathematical functions (e.g., trigonometric, logarithmic, etc.) and mathematical constants (e.g., π). Evaluate the following expression using R:

$$\frac{1}{2}\sqrt{1+\cos(\pi/2)}\log 10$$

3. We can create new R objects by using the "assign" operator which is an arrow with a minus sign `->` or, in reverse, `<-`, or is the regular `=` operator. For example, the code below creates two objects `x` and `y`:

```
x <- 1
2 -> y
x = x + y
```

   To display the current value stored in an object, type the name of the object in the Console window. For example:

```
z = y * x
z
```

4. There are two ways to list all objects that exist in memory. First, we may use the function:

```
ls()
```

   which displays the names of the three objects (`x`, `y` and `z`) that currently exist in memory. Alternatively, you can click on the "Environment" tab in RStudio which displays all existing objects as well as the current value that is assigned to each object in a neatly formatted table. To delete an object from memory, we use the `rm()` function:

```
rm(x, y)
```

   where the names listed inside the brackets tell R which objects should be deleted from memory. Use the `ls()` command now to list all objects in memory. To delete all objects from memory, you can use the R command:

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

5. *Getting help*: R comes with extensive documentation that covers all R commands in detail with many examples of how to use the commands. To display the documentation for the command `ls()` type:

   ```
   help("ls")
   ```

   The help for the function `ls()` will be displayed in the "Help" tab of RStudio. A tip on using help: examples of the command in action are generally listed at the end of the help file.

   If you are not sure about the name of an R command, you can do a keyword search by using the apropos command. For example, to search for all R commands containing the keyword "med", try:

   ```
   apropos("med")
   ```

   Other ways to get help include the internet search engine http://www.rseek.org which only searches over R help documents, or you could try the RStudio "Help" tab.

3 Data management in R

R uses one folder, known as the "working directory", to read or write data files. In particular, R assumes that all data files can be found in this working directory. To see the location of the current working directory, try:

```
getwd()
```

which displays the full path to current working directory. For example, on this laptop, the current working directory is set to the folder "C:/Users/dschmidt/Documents".

There are at least two ways to change the current working directory. The first is to use the `setwd()` command:

```
setwd("C:/Rcode/"
```

where `C:/Rcode/` is the full path to the new directory. We will now see how to load data from a file into R. We will assume that the data is stored in the CSV ("comma separated values") file format. Here, each line of the file represents a row of data values (i.e., numbers, letters), and every data value is separated by a comma. Furthermore, we will assume that the first line of the data file contains a list of variable labels which we will use to refer to the different variables in the data set.

1. Download the file `heart.csv`. Now let's load this file into R by typing:

   ```
   heart <- read.csv("heart.csv", header = TRUE)
   ```

2. The above command `read.csv()` tells R to load the data file `heart.csv` and store the contents of the file in an R object called `heart`. In this example, the command takes two options: (1) the filename (`heart.csv`), and (2) header (`HEADER = TRUE`) which tells R that the first line of the file contains variable labels. If the first line in `heart.csv` does not contain any labels, we can load the same file by using `HEADER = FALSE` in the above command. To get more information about the `read.csv()` command, type `help("read.csv")`.

   The object `heart` is an example of a data frame, which is an R data type used for all data sets. In the next section, we will see how to display the contents of a data frame.

3. We can see how many variables and samples are contained in a data frame by using the `ncol()` and `nrow()` commands respectively. For example,

   ```
   nrow(heart)
   ncol(heart)
   ```

4. We have seen before that we can view the content of an object by typing the name of the object in the Console window. This is also true when it comes to viewing data frames. To view the content of the data frame `heart`, type:

   ```
   heart
   ```

Unfortunately, this approach prints out the content of the entire data frame which can be quite large and difficult to read. Instead, we can preview the data frame using the `head()` command:

```
head(heart)
```

which, by default, only displays the first six rows of the data frame `heart`. Now, read the R documentation for the command `head()` and work out how to print the first ten lines of the `heart` data frame (instead of six) using an option for `head()`.

5. Note that we can also use RStudio to view data frames graphically by typing:

   ```
   View(heart)
   ```

6. We will now examine how to access and manipulate individual variables stored within a data frame. First, lets list all the variable names in the data frame `heart` using the `names()` or the `str()` command:

   ```
   names(heart)
   str(heart)
   ```

   Although both commands list all the variable names in the data frame, the `str()` command displays more information about each variable (e.g., the variable type and the first few data values).

7. Next, lets examine how we can access data for a single variable. There are three equivalent ways of doing this in R:

   ```
   heart$AGE
   heart[,"AGE"]
   heart[,1]
   ```

   The above commands display the content of the variable `AGE` stored within the data frame `heart`. In the last command, we use the number `1` to tell R we wish to display the first column (i.e., `AGE`) of the data frame.

   In order to access more than one variable in the data frame, we use the combine function `c()` which combines multiple objects into a list. For example, to access the `AGE` and `HD` variables in our data frame, we can type:

   ```
   heart[,c("AGE","HD")]
   ```

   or, equivalently, we could use the column numbers of the variables:

   ```
   heart[,c(1,14)]
   ```

8. We have learnt so far how to tell R to display all the rows for a set of named variables. But, what if we are not interested in viewing the entire sample for the named variables and only wish to display a particular subgroup of people? As an example, suppose we only wish to view rows 10 to 15 of the data frame `heart`. There are two ways we can do this in R:

   ```
   heart[10:15,]
   heart[c(10,11,12,13,14,15),]
   ```

9. It is of course possible to combine row and column referencing in the same R command. For example, to display rows 3 and 88 for variables `SEX` and `CP`, we can use:

   ```
   heart[c(3,88), c("SEX","CP")]
   ```

   As another example, we can display rows 10 to 50 for the variables `RESTECG` and `HD` with the command:

   ```
   heart[10:50, c("RESTECG","HD")]
   ```

10. The last type of variable referencing that we will cover is called logical referencing. Logical referencing is used when, for example, we are interested in displaying the `AGE` variable, but only for those individuals who are male. Alternatively, suppose we want to view the `SEX` and `CP` variables for individuals whose `CHOL` variable is less than 150. We can accomplish both tasks with the following two commands:

```
heart[heart$SEX == 0, "AGE"]
heart[heart$CHOL < 150, c("SEX","CP")]
```

11. Furthermore, we can combine logical expressions, such as `heart$SEX == 0` and `heart$CHOL < 150`, with the logical AND operator `&` and the logical OR operator `|`. For example, to display data for individuals who are male AND whose `CHOL` is less than 150, we use:

```
heart[heart$SEX == 0 & heart$CHOL < 150, ]
```

12. R is very powerful and allows the user a lot of flexibility when it comes to manipulating data frames, including adding and removing of new variables. Suppose we want to add a new variable to our data frame `heart` whose values are equal to the `AGE` of the person divided by their `CHOL` entry. We can do this in R as follows:

```
heart$AC <- heart$AGE / heart$CHOL
```

    The name of the new variable is `AC` and its value is equal to `heart$AGE` divided by `heart$CHOL`.

13. To delete a variable from a data frame, we assign it the special value `NULL` as follows:

```
heart$AC <- NULL
```

4 Descriptive Statistics in R

We will now cover several important approaches to summarising data in R.

1. Begin by reloading the dataset into R

```
heart <- read.csv("heart.csv", header = TRUE)
```

2. R provides many important descriptive statistics functions including functions to compute the sample mean, the sample quantiles, and the sample variance, among others. Below are some examples:

```
   mean(heart$AGE)     # sample mean
  median(heart$AGE)    # sample median
quantile(heart$AGE)    # sample quantiles
    min(heart$AGE)     # minimum value
    max(heart$AGE)     # maximum value
   range(heart$AGE)    # sample range
    var(heart$AGE)     # sample variance
     sd(heart$AGE)     # sample standard deviation
```

3. A convenient way to display the most useful summary statistics for a variable or a set of variables is to use the `summary()` command:

```
summary(heart$AGE)
summary(heart)
```

4. Basic tabulation and cross-tabulation can be accomplished with the `table()` command in R. As an example, to compute the frequency distribution of `SEX`, we run:

```
table(heart$SEX)
```

5. Unfortunately, this frequency table is somewhat difficult to read as the values `0` and `1` of the variable `SEX` are not labelled. In fact, by default, R treats `SEX` as an numeric variable and not as a factor variable. To convert `SEX` into a factor variable, with two levels `0` (corresponding to males) and `1` (corresponding to females), we can use the `factor()` command:

   ```
   heart$SEX <- factor(heart$SEX, labels=c("MALE","FEMALE"), levels=c(0,1))
   ```

   The `factor()` function takes an option `labels` which tells R how to label the values of the factor. In this case, the label `MALE` (`FEMALE`) is used when `SEX` is `0` (`1`). The levels of the factor (`0` and `1`) are specified with the option `levels`. Note how the combine operator `c()` is used to create a list of labels and a list of levels.

6. The `table()` function can also be used to create contingency tables. For example,

   ```
   table(heart$SEX, heart$HD)
   ```

   creates a contingency table for the variables `SEX` (rows) and `HD` (columns). See if you can work out how to recode `HD` as a factor variable, with the labels `NO` and `YES` for values 0 and 1, respectively, and then cross-tabulate again.

5 Script Files

Using script-files will ensure reproducible data management and analysis. Log files are used to make a full record of your RStudio session, or of the output of your script files. R script files are text files which contain a series of commands. You can tell R to execute (run) these script files, and they will essentially issue these commands to the R console as if you were typing them, one after another.

To create an R script, go to File, New File, R Script. This will create an empty script file that you can begin editing using the R editor. Save this script file as `studio1.R`. Its a good idea to open your script file with some header information about what it does. This can be done using comments. Comments are pieces of text that begin with a `#` symbol, and are ignored by R when executing a script. For example, a good header to use might be:

```
#########################################################
# R script:  studio1.R
# Project:  Studio 1
#
# Date:  19/07/2017
# Author:  D. Schmidt
#
# Purpose:  Introductory script for Studio 1
#########################################################
```

Opening headers can be as detailed as required, and in general more information is better. Comments should be used throughout a script file to annotate the script and make it clear, both to you, and to others what the script is doing. Whitespace (lines, extra spaces) is also ignored by R when running a script, so you can, and should, use this liberally to improve readability.

To execute your script file, from the command console type

```
source("studio1.R")
```

This runs your script by telling R to treat the contents of your script file as if they were literally being typed into the command console. You can also issue this command by clicking the “source” button in the top-right of your editor. For the rest of your studios you should use the script file editor to record the commands that you use.

6 Flow Control and Functions

It is important that you familiarise yourself with the flow control commands used in R. The important flow control commands implemented in R are `for` loops, `while` loops and `if` statements. When executing a loop or a function, R suppresses the output from the functions you are calling. The `cat()` function can be used to display text and numbers to the console. The special symbol `\n` can be used in a string to force a newline when printing using `cat()`, i.e.,

```
i = 45
cat("The value of i is:", i, "\n")
```

1. Implement a loop that prints out all the odd numbers from 1 to 15.

2. The other key programming tool required to use R effectively are user-defined functions. The `function()` function is used to define a function; the `return()` function is used to return values from the function. For example

   ```
   mydiv <- function(a, b)
   {
     return(a/b)
   }
   mydiv(10, 3)
   ```

   The important thing to note with R is that the function will not be defined (and usable) until you have executed the code that defines the function. If this is in a script file, you can simply run it using source. Redefining a function will overwrite the previous function, so there is no need to remove it before re-running your script if you are updating your function.

   Write a function that takes an argument n, calculates the factorial of n, and returns it.

3. Returning multiple return values can be done in R using R lists, or R “structures”. These objects are placeholders for multiple fields of data. To create a list use the `list()` function, i.e.,

   ```
   retval = list()
   ```

   Then, once the list has been created you can access fields in the list (or add new fields) using the `$` operator; for example:

   ```
   retval$a = 1
   retval$b = "you can store strings too"
   retval$c = list()
   retval$c$g = "You can store lists inside lists"

   retval
   ```

   Write a function that takes a vector of numbers and returns the minimum and maximum number, and the difference between the maximum and minimum (i.e., the range) of the vector.
   (hint: the `length()` *function lets you find the length of a vector).*
   (hint: remember to return multiple return values using a list as described above).

7 Basic Examination of a Dataset in R

We will now calculate some basic descriptive statistics in R on a simple example dataset.

1. Load the in-built tooth growth dataset using `data("ToothGrowth")`. If a dataset is built-in, there is usually a help file associated with it that describes the data. Use `help("ToothGrowth")` to find out about the variables in this dataset.

2. Identify and calculate the appropriate statistics for central tendency and for dispersion for the three variables.

3. Compute a box plot and histogram for the `len` variable. The functions `boxplot()` and `hist()` can be used to produce these plots. Use the `help()` function to learn how to use `boxplot()` and `hist()`.

4. Calculate the 5th, 25th ,50th ,75th and 95th quantiles of the `len` variable using the `quantile()` function. *(hint: the combine function* `c()` *can be used to build vectors of numbers, for example, for the probs parameter of the* `quantile()` *function).*

5. Scatterplot and calculate the correlation coefficient between the `dose` and `len` variables:

```
plot(ToothGrowth$len, ToothGrowth$dose)
cor(ToothGrowth$len, ToothGrowth$dose)
```

   See if you can add appropriate labels to the x and y axis.

8 Categorical Data

In this section we will explore some descriptive statistics when variables are categorical.

1. Copy the mushroom dataset `mushroom.csv` from the Moodle page and load it into R.

2. Create a pie chart for each variable to show the relative frequencies of the values, e.g.,

```
tab = table(mush$cap.shape)
pie(tab)
```

3. Cross-tabulate the `mush$class` variable against each of the other variables. Which variable appears most strong associated with `class`?

   The following code fragment shows how to cross-tabulate `class` with `cap.shape`:

```
table(mush$class, mush$cap.shape)
```

9 More descriptive statistics for numeric variables

Load the `wine.csv` dataset. This dataset contains measurements of various quantities related to wines, along with a score (quality) determining the overall quality of the wine. Clearly, it is of interest to winemakers to know which attributes of a wine contribute to quality, and this dataset is an excellent example of the wide range of applicability of data science concepts.

We will learn more advanced procedures to model the relationship between the wine attributes and wine quality in the coming weeks, but for now we can measure the strength of association using correlation coefficient.

1. Write a loop that computes the correlation coefficient between each of the wine attributes and the wine quality variable. Your loop should display the information to the console in the form

```
Correlation between fixed.acidity and quality = -0.1136628
```

   for each of the variables. *(hint: you can use the* `names(wine)[i]` *to access the name of the i-th variable in a dataset).*

2. Which of the attributes is most strongly associated? Which ones appear to have no association?

10 Further reading

This concludes the first introduction to using the R language. For further reading, you should download and read the documents `R0001.pdf` through to `R0007.pdf` from Moodle before Studio 2. These offer more introductory information on how to use R and will serve as a useful reference material. The file `Basic commands in R.R` script is available on Moodle and also summarises a number of useful R commands.