STM1001 Lecture: Introduction to R

Amanda Shaker

STM1001 - Making Sense of Data

Copyright

Introduction

- Welcome!
- In this lecture we will cover an Introduction to R and RStudio
- By the end of this lecture you will:
  - know how to get started with R and RStudio
  - understand fundamentals of R coding
  - know how to import a data set
- All of the material we cover in this lecture, including short videos, is available in the LMS in the tile called **"Introduction to R"**
- Once you have either attended this lecture, or gone through the material in the **"Introduction to R"** tile in your own time, you will be ready to attempt Computer Lab 1 this week

Why R?

- R involves coding, which for many of us will mean a learning curve
- So **why R?**
  - R is a very popular programming language, so it's **often required in job offers**. And, it is **built for statistics**.
  - Coding is a skill that will benefit you in years to come. Desirable on CV, or if you do more research in the future. And it will help to boost other skills such as problem-solving
  - **Much more powerful than spreadsheets**. And once you have the code written, it is reproducible - no need to repeat point-and-click work over and over again
  - R is **FREE**! This means you can install it on your own computer, and you don't need to be on campus, or use the virtual network, to do your coursework

Why R?

And perhaps the best reason of all: we make use of the computing power available to us in R.

Compare this...

$$T = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(s_2^2/n_2\right)^2}{n_2 - 1}} = \frac{\left(\frac{10.81^2}{8} + \frac{23.06^2}{8}\right)^2}{\frac{\left(10.81^2/8\right)^2}{7} + \frac{\left(23.06^2/8\right)^2}{7}} = 9.935 \approx 10.$$

$$df = \frac{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)^2}{\frac{\left(S_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(S_2^2/n_2\right)^2}{n_2 - 1}}$$

We are dealing with a two-tailed hypothesis test and $\alpha/2 = 0.05$. By looking at Table V column 0.05 and row 10, $t_{0.05} = 1.8125$. So $CV = \pm t_{0.05} = \pm 1.8125$. Therefore we reject H_0 if $t \leq -1.8125$ or $t \geq 1.8125$.

$$t = \frac{\bar{x}_1 - \bar{x}_2 - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} = \frac{46.5 - 52.375 - 0}{\sqrt{\frac{10.81^2}{8} + \frac{23.06^2}{8}}} = -0.652.$$

To this...

```r
t.test(wonions$Yield ~ wonions$Locality, var.equal = TRUE)
```

Where to find help

- There will be plenty of help available to you throughought the semester
- I cannot emphasise enough: **ask for help if you need it!**
- Check the **“Student Resources”** tile on LMS for various options
- You can also **refer back to this lecture**, the **“Introduction to R” tile**, or the **Computer Labs**.

1. Introduction and Rstudio Interface

- R is a freely available, popular and very powerful programming language for statistical computing and graphics.
- RStudio is a popular and user-friendly interface for using R.
- We will be using RStudio throughout this subject.
- If you are working on a computer in a Computer Lab on campus, RStudio has already been installed.
- If you are working on your own computer and have not yet installed R and RStudio, please do so before the first computer lab by following the instructions provided on the LMS

1. Introduction and Rstudio Interface

Editor

This is where we will write most of our code.

Console

This is where output will be displayed. Code can be written here as well if desired.

Environment and History

Where we can view data sets, variables, etc.

Files, Plots, Packages, Help and Viewer

Here we can view plots, help files, and more.

$\Rightarrow$ **RStudio**

2. Help & Support

- R is free, open source, and there is plenty of help available on the web via a simple web search.
- The R homepage `https://www.r-project.org/` also provides many links to help files.
- Or we can display the help file for a particular function in RStudio. For example:

```r
help(mean)
?mean
```

- Either of the above two commands will work.
- ⇒ **RStudio**

3. R Packages

- R has add-on packages which contain additional functions and data sets.
- Some are already included when R is installed on your computer and just need to be ‘loaded’ when we want to use them.
- Others can be downloaded (instructions can be found on LMS)
- For example, the `MASS` package is already included. We can load it by going to the ‘Packages’ tab in RStudio, or by running the command

```
library(MASS)
```

- $\Rightarrow$ **RStudio**

4. Basic mathematical operations

- R can be used to perform basic mathematical functions such as:

```
4 + 5
36 / 6
```

- We could put both of these commands on one line by separating them with a ; symbol:

```
4 + 5; 36 / 6
```

- ⇒ **RStudio**

5. The assignment operator

- The <- symbol, called the leftwards assignment operator, is an important one.
- It is used to store values or objects under a given name that we choose. For example:

```
x <- 5
```

- This means that `x` now has a value of 5.
- We could use any name we like in place of `x` .
- $\Rightarrow$ **RStudio**

6. Logical operators

- Logical operators: we make a statement, and R returns `TRUE` or `FALSE` :
  - == means 'equal to'
  - ! means 'not'
  - & means 'and'
  - | means 'or'
- For example:

```
x == 5 # x is equal to 5
x != 5 # x is not equal to 5
x == 5 & x == 6 # x is equal to 5 AND x is equal to 6
x == 5 | x == 6 # x is equal to 5 OR x is equal to 6
```

- ⇒ **RStudio**

7. Recommended best practices for R Coding

- Google's R Style guide is a pdf document available online that contains recommendations for writing R code.
- Many R users choose to adopt the style recommended and we will be following many of the recommendations in this subject.
- Some examples:
  - Use plenty of comments. This makes your code easier for you and others to follow and understand.
  - Use a space in between operators and after commas.

```r
# Below are some good and bad examples
x + 5 # GOOD: spaces between operators
x+5  # BAD: no space between operators
```

- More examples are in the Google style guide itself.

7. Recommended best practices for R Coding

- Comments are a good way of making our code easier to follow.
- Anything after the # symbol is a comment and will be ignored.
- In RStudio, comments are green.
- For example:

```r
# This is a comment and will be ignored
x + 3
```

- $\Rightarrow$ **RStudio**

8. Functions

- In R, a function is a block of code that has been written to perform a certain task.
- We can refer to functions by name.
- For example, the `round` function can be used to round a number.
- Suppose we wanted to round the number 5.6789 to two decimal places and we want to find out how to use the `round` function to do this.
- In R, we could run the command `help(round)` to view the help file for the `round` function.
- $\Rightarrow$ **RStudio**

8. Functions

- From the help file, we can see that the `round` function has two *arguments*: `x` and `digits` .
- These arguments can be specified when we *call* (or use) the function.
- `x` is the number we wish to round.
- `digits` is the number of digits we wish to round to.
- In the help file, the function is specified as `round(x, digits = 0)`
- This means that the `digits` argument is optional. We can specify it, but if we don't, the assumed value will be 0.
- The `x` argument does not have a default value and must be specified.

8. Functions

- So we can round 5.6789 to two decimal places as follows:

```r
round(5.6789, 2)
```

- Notice that we have specified the arguments in the same order as given in the help file.
- We could also call the function like this:

```r
round(x = 5.6789, digits = 2)
```

- Using argument names like this is not required, but it can make it easier for us to keep track of the arguments. It also means we can specify them in any order.
- This is especially useful when we are using functions that have many arguments.
- ⇒ **RStudio**

8. Functions

- Functions can be broadly categorised into three groups:
  1. Built-in functions: such as the `round` function
  2. Functions in add-on packages: for example, functions available from within the `MASS` package
  3. User-defined functions: We can even write our own functions, although this is not required in this subject unless you are doing the Data Science stream.
- We will come across many more functions throughout this subject.

9. Objects

Types of objects we will be considering:

1. Variables
2. Vectors
3. Matrices
4. Data frames
5. Lists

9. Objects > Variables

- We can think of a variable as an object that has a name which represents one value.
- This can either be a number, ‘character string’ or logical value:

```r
x <- 5 # Number
cs.var <- "abc and 123" # Character string
is.numeric(x) # Logical value: TRUE or FALSE
```

- We can also store the result of a mathematical function as a numeric variable:

```r
sqrt.of.4 <- sqrt(4) # Square root of 4
```

- ⇒ **RStudio**

9. Objects > Variables

Missing values

- Another type of variable in R is missing values.
- R uses `NA` to represent missing values.
- We will learn more about how to handle missing values later.

10. Objects > Vectors

- So far, we have looked at variables that contain one value, or *element*.
- A vector is a type of variable that contains one or more elements.
- The elements can be any type, but only one type (e.g. not a mix of numeric and character).
- One way to create a vector is with the c function. For example:

```
num.vector <- c(1, 3, 5, 7, 8) # Numeric vector
char.vector <- c("blue", "red", "green", "blue")
```

- ⇒ **RStudio**

10. Objects > Vectors

- We can refer to or change specific elements using *indices* within square brackets `[]` .
- For example:

```
char.vector[3] # 3rd element
char.vector[c(2, 4)] # 2nd & 4th elements
char.vector[1:3] # Elements 1 to 3
char.vector[-3] # All elements except the 3rd
```

- ⇒ **RStudio**

10. Objects > Vectors

- We can also use the `:` symbol, `seq` and `rep` functions to create vectors:

```
one.to.five <- c(1:5)
even.numbers.up <- seq(from = 2, to = 10, by = 2)
five.zeros <- rep(0, times = 5)
```

- ⇒ **RStudio**

10. Objects > Vectors

- If we want to know the number of elements in a vector, we can use the `length` function:

```
length(one.to.five)
```

- ⇒ **RStudio**

11. Objects > Matrices

- We can think of a matrix as a group of vectors combined together as rows or columns into a table.
- Like vectors, all of the elements must be of the same type.
- We can use the `cbind` and `rbind` functions to create matrices. For example, consider the two vectors

```r
names <- c("Peter", "John", "Mary", "Albert")
country <- c("Australia", "New Zealand", "UK", "Singapore")
```

- Then we could combine them in either of the following ways:

```r
matrix.1 <- cbind(names, country)
matrix.2 <- rbind(names, country)
```

- $\Rightarrow$ **RStudio**

11. Objects > Matrices

- If we wanted to know the dimensions of a matrix we could use the `dim` function.
- Or, if we wanted to know the number of rows or columns, we could use `nrow` or `ncol` :

```
dim(matrix.1) # Dimensions
nrow(matrix.1) # Number of rows
ncol(matrix.1) # Number of columns
```

- $\Rightarrow$ **RStudio**

11. Objects > Matrices

- Like vectors, we can refer to or change an element(s) in a matrix using square brackets.
- We refer to the row number first, followed by the column number. For example:

```
matrix.1[2, 1] # 2nd row, first column
matrix.1[ , 1] # All rows, first column
```

- $\Rightarrow$ **RStudio**

12. Objects > Data frames

Data frames are similar to matrices but with the following differences:

- They can contain columns that are of different variable types
- We can use the $ operator, as well as indices, to refer to a particular column
- Character string vectors within a data frame can be converted to *factors* using `as.factor`. We will learn more about factors later.

12. Objects > Data frames

- Suppose we create a vector of ages, and then want to combine this with `names` and `country` to create a data frame.
- We can do this with the `data.frame` function:

```
age <- c(60, 30, 40, 55)
people <- data.frame(names, country, age)
people[3, ] # Refer to the 3rd row
people$age # Refer to the column called age
```

- ⇒ **RStudio**

13. Objects > Lists

- Suppose we wanted to combine objects such as variables, vectors, matrices, data frames into one object.
- In R, this combined object would be called a `list`.
- We can create a list using the `list` function:

```
list.1 <- list(num.vector = num.vector,
               matrix.1 = matrix.1,
               people = people,
               new.vector = c(5, 10, 15, 20, 25))
```

- Notice that we must give each object a name in the list.

13. Objects > Lists

- Once a list has been created, we can see the objects within a list using the `names` function.
- We can refer to any of the objects using the $ operator.

```
names(list.1)
list.1$people
```

- ⇒ **RStudio**

14. Graphics with R

- R provides us with the capability to produce high-quality graphics with precision.
- We will now cover some of the basics:
    - Graphics device settings
    - Basic graphics options
- More on graphics will be covered as we encounter them throughout this course.

14. Graphics with R > Graphics device settings

- Let us begin with a very simple command:

```
curve(x^2)
```

- This will produce a graph of x^2 in the Plots window of RStudio.
- $\Rightarrow$ **RStudio**

14. Graphics with R > Graphics device settings

- Using the `windows` function, we could open up a separate window, called a 'device', in which to display the plot:

```
windows()
curve(x^2)
```

- Every time we run the `windows()` command, a new device is opened.
- The size of the graphics device can be changed with the `height` and `width` arguments:

```
windows(width = 7, height = 3)
curve(x^2)
```

- $\Rightarrow$ **RStudio**

14. Graphics with R > Graphics device settings

- We can display multiple plots on one device using the par function and `mfrow` argument as follows:

```
windows(width = 7, height = 3)
par(mfrow = c(1, 2))
curve(x^2)
curve(x^3)
```

- `mfrow = c(1, 2)` tells R to divide the graphics window into 1 row and 2 columns.
- In other words, we should see two plots side-by-side.
- $\Rightarrow$ **RStudio**