STM4PSD – Workshop 2

R and RStudio

R is a free programming language that is used for statistical computing and for creating publication-quality graphics. It is well supported by the research community and is becoming very popular in industry. R is free to download via the R-project website at `https://www.r-project.org/`. The website also includes useful additional content such as manuals and additional packages. RStudio is a popular interface with which to use R. Like R, RStudio is freely available and is easy to install from the website `https://www.rstudio.com`.

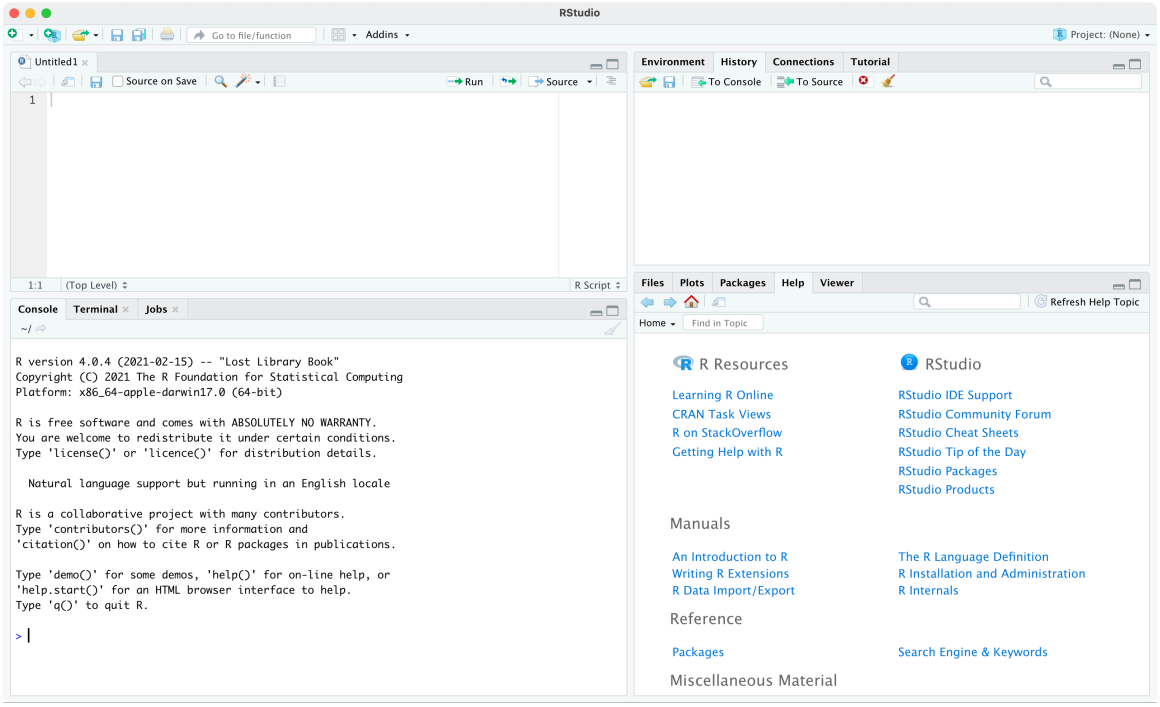

Figure 1: An example screen shot of the RStudio interface.

In Figure 1 an example screen shot of RStudio is provided.

- On the left-hand side are two windows. The window on the top left is the scripting window, which will be discussed shortly. It may not appear on your system immediately—if it doesn't, that's not a problem for now. The window on the bottom left is the console. This is where R commands are entered and resulting output displayed.
- On the right-hand side we have two other windows. The window in the top-right provides some information in regards to R objects/variables. The window in the bottom-right displays other sources of information such as help files and plots.

You will be become more familiar with RStudio functionality with practice.

Using R as a calculator

To familiarise yourself with R, we will start by performing some simple calculations. At the most basic level, you can think of R as just a calculator you can type into.

1. R commands can be entered directly into the R Console for immediate effect. To demonstrate, let us carry out some basic arithmetic. After the R prompt > in the R Console enter these commands:

```
3 + 2 * 4
2^2 + 3
```

   Note that the symbol ^ is used for powers and * for multiplication. Once you press Enter on the keyboard you should see the result displayed. The reason for the [1] preceding the output will become obvious later.

LA TROBE UNIVERSITY
All kinds of clever

2. An important feature of R is its built-in *functions*. A function is a named set of instructions that has a defined use. For example, the function `sqrt` will find the square-root of a number. For example, to calculate $\sqrt{16}$ type

   ```
   sqrt(16)
   ```

   Another important function is the exponential function. You can calculate the value of e^x using the `exp` function in R. For example, e^3 is calculated by writing

   ```
   exp(3)
   ```

3. The number 16 enclosed in brackets with the `sqrt` function is called an *argument* to the function. Functions may have *required* arguments or *optional* arguments. An important function that allows optional arguments is the logarithm function, written using `log`. Open the help file for this function by typing

   ```
   ?log
   ```

   Have a quick read of the help file to gain an appreciation of the function. Note that some examples can be found at the end. There are also some variations of the function; e.g. `log10`.

   Note that the `log` function includes an optional argument, `base`. In particular, it lists the function:

   ```
   log(x, base = exp(1))
   ```

   This shows both required arguments and optional arguments.

   - Required arguments (also known as compulsory arguments) are listed first. In this case, `x` is a required argument.
   - Optional arguments are listed after the required arguments, and you can identify them by the appearance of the = sign. The = sign indicates that, if you do not supply the optional argument yourself, then the value to the right of the = will be used. This value is called the *default value*. In the case of the `log` function, `base` is an optional argument with default value `exp(1)`.

   For example, to calculate $\log_2(40)$ you can write `log(40, base=2)`.

4. Now try calculating the following expression using R:

   $$e^5 + \sqrt{3 \times 15^2 + \log_5(9)}$$

   You should get 174.4202 as the answer.

Helpful hints:

- While your cursor is in the console window, you can press the up arrow on your keyboard to jump to previously evaluated lines.
- Sometimes you may write something and see the + symbol followed by a blinking cursor. This usually means there is a mismatched bracket. Type `sqrt(16` in the console to see an example. You can press the Esc key on your keyboard to escape this situation and try again.

Assigning variables

It is quite useful to be able to store calculations and other objects to a named reference for later usage. This named reference is called a variable.

5. We can assign values to a variable using the operator <- (the symbol < immediately followed by −). Sequentially enter the following commands in R and note the output. Don't just copy and paste all at once—type each one out, one-by-one, observing the output after each individual command.

   ```
   answer <- log(2, base = 10)
   answer
   round(answer)
   round(answer, digits = 1)
   floor(answer)
   ceiling(answer)
   ```

   After doing this, read the documentation for the `round`, `floor` and `ceiling` functions. Which of the functions have optional arguments, and what are they?

LA TROBE UNIVERSITY
All kinds of clever

Vectors are lists of elements

One of the most important features of R is its ability to work with *vectors*. A vector is simply a list of elements. R has been designed to be very efficient when working with vectors, so to be proficient in R, it is essential to be familiar with vectors.

Vectors can be created in many ways. Some cases are given below.

- **Specific elements**. You can create a vector containing specific elements using the `c` function. The letter `c` here stands for "combine", because its effect is to combine many individual elements into a single list. For example, to create a list containing the numbers 1, 3, 6, 10 and 15, you can write:

  ```
  x <- c(1, 3, 6, 10, 15).
  ```

  This stores the vector in the variable `x`. You won't see the output immediately. If you want to see the vector, simply write `x` in the console.

- **A range of numbers**. The symbol `:`, called the colon operator, can be used to specify a range of integers. Try the following examples yourself:

  ```
  y <- 1:100
  z <- -13:5
  ```

- **A sequence of numbers increasing by the same amount**. For example, the code below will create a vector containing the numbers $0, 3, 6, \ldots, 999$.

  ```
  threes <- seq(0, 999, by = 3)
  ```

  After doing this, display the contents of the vector by writing `threes` in the console. Note the meaning of `[1]` preceding output earlier. The number in the squared brackets tells us the index of the element displayed first in the associated line of output.

- **Join two vectors together**. The `c` function is extremely versatile. For example, it can also be used to add extra elements, or to join two vectors together. Using `y` and `z` as defined earlier, observe the output of the following:

  ```
  c(x,10)
  c(x,50)
  c(y,z)
  c(z,y)
  ```

  Note that this code segment is not storing the result in any variables. You will only see the output in the console.

- **A random ordering of numbers**. The `sample` function can be used to generate a random vector. There are a number of ways it can be applied. For the simplest way, type the following into the console yourself and see what you get:

  ```
  sample(15)
  ```

  Repeat the command again—you should get different results each time.

This is just a small number of ways vectors can be created, but they are the most common.

6. Create a vector called `numbers` containing the numbers $15, 20, 25, 30, \ldots, 105$.

7. The number of elements in a vector is called its *length*. The `length` function will tell you the length of a vector. Write `length(numbers)` in the console to find the length of the vector from the previous part.

8. The aforementioned power of R comes from the fact that arithmetic operations can be applied directly to vectors. If you have done any programming before, you are probably used to using loop structures like *for* and *while*. These exist in R, but can often be circumvented more efficiently. To see how to apply operations to vectors, observe the output from the following commands

   ```
   x <- 1:10
   y <- 2*x
   y
   x^2
   sqrt(x)
   sqrt(x^2)
   x + y
   x * y
   x+log(y + 2)
   y/c(1, 2)
   ```

   For the last command, R recognises that `c(1, 2)` is a shorter vector than `y` and therefore recycles values from `c(1, 2)`.

LA TROBE UNIVERSITY
All kinds of clever

Scripts

So far we have entered R commands directly into the R Console. However, it is much more convenient to use what are called script files. Using script files we can write and edit multiple commands, creating a record of what we have done, and electively choose which commands we would like to execute.

9. In the `File` menu select `New File` and then `R Script`. A new window is now open.

10. In the new window, enter the following (note that nothing will be executed in R yet until we tell it to).

```
x <- 1:10
y <- 2*x
x^2
x+y
x*y
y+2
y/c(1,2)
```

11. Now, we want to run (or execute) all of the commands in the R Console. On the top right of the script window, you should see a button that says "Source". Click on the small arrow to the right of that button, and select "Source with Echo" to run the code with all output shown.

    Another way is the highlight (select) the lines you want to run using the mouse and press the `Run` button to the left of the `Source` button.

12. The reason this is useful is because it allows us to easily modify our script, either when we want to make changes or if we notice an error. Change the first line of the above to "`x <- 1:50`" and run the script again.

Defining functions in R

13. Now suppose we wanted to repeat the same thing several times. Modifying the code is inconvenient: what if we had to do this hundreds of times?

    The solution is to write a function yourself. A function is a block of code used to accomplish a certain task. Most of the time, a function is built up from three key parts:

    - A set of inputs (called *arguments*).
    - A block of code, which does what we want it to do.
    - A *return value*, which is the output of the function.

    Here is a simple example of a function in R:

```
hypotenuse <- function(x, y) { sqrt(x^2+y^2) }
```

    You can write this in the console or in a script file. If you write it in a script file, you will need to Run/Source the script first.

    Note several important features:

    - The name of the function is `hypotenuse`, which is indicated by it being to the left of `<-`
    - The word "`function`" is a keyword telling R that you are defining a function.
    - After the word `function` is written `(x,y)`. This tells R that the function inputs will be variables `x` and `y`.
    - The code block is surrounded by `{` and `}`. The content between these braces is the code that will be run when the function is used.

    After defining the function above, we can then run the code in the console:

```
> hypotenuse(3,4)
[1] 5
> hypotenuse(4,5)
[1] 6.403124
> hypotenuse(5,6)
[1] 7.81025
```

    Before continuing on, make sure you can replicate what is shown above.

LA TROBE UNIVERSITY
All kinds of clever

14. Functions are not limited to just one small piece of code like what is shown above. They can also make use of any functions you have already defined. Consider the following example:

```
distance.between.points <- function(x1, y1, x2, y2) {
    x.new <- x1 - x2
    y.new <- y1 - y2
    hypotenuse(x.new, y.new)
}
```

This function calculates the distance between two points in the x-y plane. You can then use it in the console like so:

```
> distance.between.points(1,1,3,4)
[1] 3.605551
> distance.between.points(3,4,3,4)
[1] 0
```

Some important points to take note of:

- In R, the last line of code in the `{...}` block is the 'output' of the function.
- If you define a variable inside a function code, it will not be accessible outside of the function code. For example, typing "`x.new`" in the console will display `Error: object 'x.new' not found`.
- R allows use of the symbol `.` in variable and function names. This may be unusual if you are familiar with other programming languages, but it is common practice in R.

15. The formula for converting from degrees Celsius to degrees Fahrenheit is

$$F = C \times 9/5 + 32,$$

where F is the temperature in degrees Fahrenheit and C is the temperature in degrees Celsius. Write an R function named `to.fahrenheit` which takes a single argument, `temp`, which is the temperature in Celsius, and returns the result of converting that to Fahrenheit. For example, it should behave as follows:

```
> to.fahrenheit(0)
[1] 32
> to.fahrenheit(100)
[1] 212
```

16. The quadratic formula for finding roots to the quadratic $ax^2 + bx + c$ is

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Because of the $\pm$ symbol, this formula yields two different numbers. Write an R function named `quadratic` which takes arguments `a`, `b` and `c`, and returns a vector containing the two numbers calculated by the quadratic formula. For example, running `quadratic(1,-1,-2)` should return a vector containing -1 and 2.

If you have observed that there will be problems with taking square roots of negative numbers, ignore that for now.

LA TROBE UNIVERSITY
All kinds of clever