MTHM501: Working with Data:

Practical 1: Introducing RStudio

Getting Started in R

R

Before getting started you should download RStudio (if you haven't already done so). Follow the instructions here https://rstudio-education.github.io/hopr/starting.html.

R is an **object-orientated** programming language. This means that you create objects, and give them names. You can then **do things** to those objects like:

- perform calculations
- statistical tests
- make tables
- draw plots

Objects can be single numbers, characters, vectors of numbers, matrices, multi-dimensional arrays, lists containing different objects and so on.

We will use a fantastic IDE[1] provided by RStudio. This is free to download, provides some neat features, and crucially, looks the same on all operating systems!

RStudio

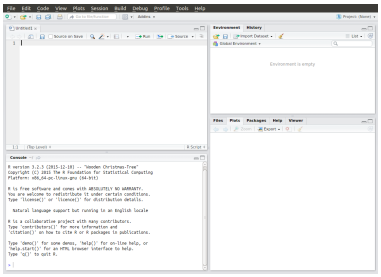

Figure 1: RStudio window

- **Script pane** (top-left): text editor
- **Console pane** (bottom-left): This is where you run R commands and view outputs.
- **Workspace/history pane** (top-right): lists objects that you have created or loaded
- **Plot/help pane** (bottom-right): this shows any plots that you create or any help files that you access.

Alter arrangement using *Tools > Global Options*, then *Pane Layout*

[1] Integrated Desktop Environment

Cheat Sheets

- **General repository**
  https://www.rstudio.com/resources/cheatsheets/.
- **RStudio cheatsheet**
  https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf.

Set-up

Setting up an R session

Guidelines

- Use a different folder for each new project.
- Set the **Working Directory** for R at the outset of each session to be the folder you've specified for the particular assignment you're working on. This can be done in RStudio by going to *Session > Set Working Directory > Choose Directory*. This sets the default search path to this folder.
- Always use **script files** to keep a record of your work, for reproducibility.
- **Additional**: Github can be used in addition

We will explore the **console** and **script** panes below, dealing with the other panes as and when they arise.

Console Pane

The console pane provides a direct interface with R, and looks similar to command line R (in Linux and Macs), and the console pane in R for Windows. You enter commands via the standard prompt `>`.

```
10 + 5 * 3
```

```
## [1] 25
```

Operators

Symbol	Meaning
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^`	to the power
`%%`	the remainder of a division (modulo)
`%/%`	the integer part

More operators

Function	Meaning
`log(x)`	$\log_e(x)$ (or $\ln(x)$)
`exp(x)`	e^x
`log(x, n)`	$\log_n(x)$
`log10(x)`	$\log_{10}(x)$
`sqrt(x)`	$\sqrt{x}$
`factorial(x)`	$x!$
`choose(n, x)`	binomial coefficients: $\frac{n!}{x!(n-x)!}$
`gamma(x)`	$\Gamma(x)$ for continuous x or$(x-1)!$ for integer x
`lgamma(x)`	natural log of $\Gamma(x)$
`floor(x)`	greatest integer $< x$
`ceiling(x)`	smallest integer $> x$

Even more operators

Function	Meaning
`trunc(x)`	closest integer to x between x and 0
	e.g `trunc(1.5) = 1`, `trunc(-1.5) = -1`
	`trunc` is like `floor` for positive values and like `ceiling` for negative values
`round(x, digits = 0)`	round the value of x to an integer
`signif(x, digits = 6)`	round x to 6 significant figures
`abs(x)`	the absolute value of x, ignoring the minus sign if there is one

History

R retains a **history** of all the commands you have used in a particular session. You can scroll back through these using the up and down arrows whilst in the console pane. (You can even save the history)

Note

One important thing to note is that unlike a language like C, R does not require the semicolon (`;`) symbol to denote the end of each command. A carriage return is sufficient. A semicolon can be used to allow multiple commands to be written on the same line if required. For example,

```
10 + 5 * 3; sin(10)
```

```
## [1] 25
## [1] -0.5440211
```

is equivalent to

```
10 + 5 * 3
sin(10)
```

```
## [1] 25
## [1] -0.5440211
```

Note II

If a command is incomplete, then R will change the `>` prompt for a `+` prompt. For example, typing `10 + 5 *` into the console pane will result in the `+` prompt appearing, telling you that the previous line is incomplete i.e.

```
> 10 + 5 *
+ 3
```

```
## [1] 25
```

You must either complete the line or hit the `Esc` key to cancel the command.

Scripts

Script pane and R scripts

You should use the **script pane** to write an R script that contains all the commands necessary for a particular project. In fact, one might argue that this is probably one of the most important features of R relative to a point-and-click statistical package such as SPSS.

Put simply, ***R scripts are just text files that contain commands to run in R***. They are **vitally important** for the following reasons:

Importance of scripts I

- They keep a systematic record of your analysis, which enables you to **reproduce** your work at a later date, or can be passed to collaborators or other users to enable them to replicate your work.
- This record means that you do not have to rely on your memory to figure out **what** you did.
- R scripts allow you to **comment** your code, which means that you also won't forget **why** you did it.

Importance of scripts II

- In more advanced settings, R scripts can also be run in **batch** mode, which means that you can ping a script off to run remotely on a server remotely
- Although programs like SPSS allow **outputs** to be saved, R scripts contain **inputs**, which are much more useful, since it is easier to generate the outputs from the inputs than it is to reconstruct the likely inputs from the outputs.
- In fact, R scripts can be combined with a markup language called 'markdown' to generate fully reproducible documents, containing both inputs and outputs. It does this using the fantastic `knitr` and `rmarkdown` packages. (In fact this workshop was written using `rmarkdown` and a package called `bookdown`.)

Comments

- RStudio comes with its own text editor, but if you are not using RStudio, others are available.
- R is case-sensitive. If something doesn't work, it's often because you have failed to capitalise, or capitalised where you shouldn't have. **NEVER, EVER, EVER** use Word to edit your R scripts! Word often tries to correct your grammar and is an absolute nightmare to work with when writing code. ***NEVER*** use it for writing code.

Example

In RStudio, you can open a new script in R using: *File > New File > R Script.*

Type the following into the **script file**:

```
## calculate the hypotenuse
## from a right-angled triangle
## with the two other sides
## equal to 3 and 4
sqrt(3^2 + 4^2)
```

Example II

Notice that nothing has happened. All you've done is write some commands into a text window. However, if you highlight these lines and then hit the `Run` button in the top right-hand corner of the script pane (or, in Windows certainly, press `Ctrl-Enter`), then RStudio runs these lines in the console pane. (Alternatively, you can manually copy-and-paste these lines into the console window.) This should return:

```
## [1] 5
```

Saving scripts

It is conventional to save R script files using the suffix '.R', though remember that they are simply **text files**, and can be viewed in any text editor. Make sure you have set up a folder to store the code for this practical in, and have changed the **working directory** to this folder, and then save this script file as something like "IntroToR.R".

> Make sure you save your script file regularly to prevent data loss!

Notes on legibility

NOTE: Use spaces within the code to make it clearer. R does not require this, but it is good practice to think about how to make you code legible. Different coders have different preferences, but
`plot(Worm.density ~ Vegetation, data = worms)`
is more readable than `plot(Worm.density~Vegetation,data=worms)`.
As Hadley Wickham says:

"*Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread.*"

Style

Note that different coders prefer different styles—there is no universal agreement. However, it's worth getting into the habit of writing your code neatly and with a thought towards legibility. One guide is the `tidyverse` guide here. (Note that unlike Python, R does not require specific indentation. Instead it uses curly brackets to group lines of code together. However, indentation is still key to good legibility.

Packages

R packages

R has hundreds of add-on **packages** that provide functionality for a wide range of techniques. These repositories are growing all of the time; some packages become redundant and are removed, others are updated, some are superceded or incorporated into others, and completely new ones appear regularly. A key part of becoming proficient with R is learning how to install and update packages.

Note: R packages can be thought of in a similar way to Matlab toolboxes or Python libraries.

CRAN

- The principal R package repository can be found on CRAN (the Comprehensive R Archive Network).
- Another popular repository, predominantly aimed at bioinformatics packages, is Bioconductor, though installation of packages through Bioconductor is more difficult than through CRAN, so we will focus on the latter only here.

Loading packages I

- Some packages are included as part of R's base package.
- To load a package, you can use the `library()` function, passing the name of the required package (quotes are not necessary for this function).
  - For example, to load the `tidyr` package, type:

```
library(tidyr)
```

Loading packages II

- If this doesn't return any error, then the package is loaded and you are now able to use *any function in `tidyr` in your R code.
- R packages must contain help files and documentation in order to be included on CRAN.
  - For example, the documentation for the `tidyr` package can be found here, through the *Reference Manual* link, plus some vignettes through the *Vignettes* link.

Installing packages I

If it's not installed, then you will need to install it:

```
install.packages("tidyr")
```

This will ask that you select a repository—choosing one close-to-home is a good idea. It might also ask you to set up a local R library in your user directory. This is a good idea, so just accept the default if it asks.

If it installs without any errors, then you can load the library using `library(tidyr)` as above.

Installing packages II

> **Note**: You only have to install a package **once** (unless you update R). You have to **load** the library once during each session. I prefer to enter all my calls to `library()` at the top of my script file, so I can quickly see which packages are required for my script to run.

Installing from zip files

> **Note**: you can also install a package from a local ZIP file, simply download the 'Package Source' from CRAN, and then type:

```
install.packages("PATH/TO/PACKAGENAME",
                 repos = NULL)
```

where `PATH/TO/PACKAGENAME` is the path to the package source file (be careful to get the path in the correct format—Windows users might have to use `\` instead of `/`). The `repos = NULL` argument tells R to look for the file locally, and not online.

Acknowledgements

I would like to acknowledge Dr TJ McKinley on whose work this is based