R Tips and Tricks

Alys Young

19/03/2021

This tutorial documents some of the tips and tricks that we Deakin HDR students have learnt over the years of coding. The idea and some topics were based on a QAEco coding club session at Uni Melb by Dr Saras Windecker.

Setting up your projects

RStudio layout

You dont have to keep the layout of R studio the same as the way it starts, find a layout, colour, font that works best for you. E.g. I love having my scripts panel and console the biggest, and the environment panel small (Tools > Global Options > Pane layout). Lots of people like using dark modes for the colour (Tools > Global Options > Appearance). You can also use rainbow parenthesis to help you get the right number of brackets (Tools > Global Options > Code > Editing). Play around with these in tools > global options.

R Projects

Using R projects rather than just scripts helps to organise all the projects that you work on. Projects create their own directory for that project and this is set as the working directory, keeps the environments of different projects seperate, allow you to open multiple projects at once and run multiple things at once, and are used for version control. To create a new project, go to File > New Project

Folders

Everyone sets up their folders different, so find a way that works for you Here are some examples:

- Data_raw - where all the raw data goes
- Data_process - where all the data you are processing and cleaning goes
- Data_clean - final data used in your next steps
- Outputs - your good outputs
- Scripts - the scripts you are using
- Archive_scripts - Old scripts
- Archive_outputs - old outputs

Scripts

It seems like personal preference if you like to have many small scripts or fewer big ones. Most people like to have functions that you create in seperate scripts which are sourced in (described below). If you like to split you scripts by what their function is, it can be good to name them using numbers to indicate the order they go in e.g. `1_DataCleaning.`, `2_DataManipulation.`, `3_Models.`, `4_Plots.` Other people also have a master scripts which contain all the essential code needed to reproduce the results. This master script is good if you want to publish your code.

R Markdown

These notes are written in RMarkdown. Its a method of writing doccuments and easily embedding code and code outputs. People use RMarkdown for writting papers, communicating with stakeholders and supervisors, creating resumes ect. My new favourite way to use it is like a tradition lab notebook which records what we tried and why. I have started doing the data cleaning and manipulation in markdown so that I have a record of what we tried, why decisions were made, and communicate that process to supervisors. It can also be good to keep notes of your project in a "README" markdown file; this also gets used by github to show what your project is about.

Setting up you code

Along with the heading at the top of the script, I also always include `rm(list=ls())` at the top too. This clears all the code in your consol. Sometimes you want to keep elements there if they take a long time to create or run, but ideally you would still clean it so that you know those elements can still be created from the code in your script. Otherwise you might be using elements which dont have the value that you expct which will change your results and make the code non-reproducible.

Code diagnostics

To help keep your code clean and identify errors for debugging, turn on R Diagnostics. To enable diagnostics, go to Tools > Global Options > Code > Diagnostics and turn on the types that you want to see. Thanks Simone Stevenson for this recommendation!

Load packages

If you're sharing your code, it can be difficult to know which packages are already installed on someone elses computer. use this structure `if (!require(ggplot2)) install.packages('ggplot2')`, changing `ggplot2` to be the package you want installed. You can then load packages as usual using `library(ggplot2)`. But you can also use packages without loading them by using `::` symbols after the package name and then after the `::` write the function you want to use. For example, if we wanted to manipulate data using the dplyr package and the function mutate, we could say `dplyr::mutate()`. This would be the same as loading the `dplyr` packages and writing the function mutate `mutate()`

Source a file

Write code in another file which makes elements in an environment and use the function `source` to load the R script and all its ojects into your environment. Try it by making a new blank scipt, code a simple vector, e.g. `x <- c(1,4,9)` and save the script. Go to a new script, `source()` the script with the vector code in it, e.g. `source('your_directory/your_script.R')` and `x` should appear in your environmnet.

Naming of elements

It can be easier to understand your code if elements are named in a logical manner. For example, shapefiles could have the suffix `_shp`, dataframes `_df`, rasters `_ras`. Remember, its important that code can be read by humans so while being succient is good, it can be better to write longer code that is easier to understand by yourself and other people.

Headings and commenting

Using logical heading and commenting out your code are useful in organisation and help if you ever have to come back to the code later. This section shows you how Alys Young lays out her code to give you inpiration. I find boxes around heading easy to find in the script, but I use starts * (shift+8) instead of hashes # (shift+3) the whole way to keep my automatic table of contents cleaner. The table of contents can be accessed for each script by clicking the button on the top right of an active script which looks like horizontal lines (in line with the run button, next to the source button).

Top of the doccument

Write detailed notes with important information at the top of your script.

```
###################
### Script title ###
###################
# Project aim:
#
# Author:
#
# Collaborators:
#
# Date:
#
# Script aim:
```

In the code

For main sections of the code I use boxes again and number them. My first main section is usually "set up".

```
##########################
## 1. Write headings here ## ----------------------------------------
##########################
```

Sub headings can be more simple. By using the line after the title, it will appear in the table of contents. Otherwise you can use 4 or more hash, hyphen or lines.
Hashes ####
Hyphen —-
Equals ====

```
# 1.1 Write sub-headings like this ----------------------------------
```

Snippets

You can create pre-made pieces of code so that you don't have to type it out every time; these are called snippets. Snippets can be viewed and modified in tools > global options > code > edit snippets Here you can see what snippets are available by defult and create your own using the same format as the ones currently there.

For example, I use a snippet for the the heading for the top of the document as shown below.

```
snippet header
    ###################
    ### Project Title ###
    ###################
    # Project aim:
    #
    # Author:
    #
    # Collaborators:
    #
    # Date:
    #
    # Script aim:
```

Thank you to August Hao for showing me snippets.

Other notes

Key board shortcuts

Shortcuts are a great way to increase your speed and make coding easier. They can be found here https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts . Some of the ones I use most often:

- A new sub-heading
    - Windows: ctrl + shift + r
    - Mac: cmnd + shift + r
- Dplyr pipe `%>%`
    - Windows: ctrl + shift + m
    - Mac: cmnd + shift + m
- To jump to the end of your line of code
    - Windows: crt + arrow key
    - Mac: cmd + arrow

Time to run code

It can be good to check how long pieces of code will take to run, either for knowing parts that will take a long time and plan your time accordingly, or to make your code more efficient. To find the time taken, you can use `system.time()` or basic actions, `microbenchmark::microbenchmark` to compare coding options down to the microsecond, or `profvis::profvis` for large chunks of code.

```
# make a random dataframe
df <- data.frame(v = 1:4, name = letters[1:4])

# e.g. to find the value on the 3rd row, 2nd column
df[3, 2]
```

```
## [1] "c"
```

```
# but how long did that take?
system.time(df[3, 2]) # very fast
```

```
##    user  system elapsed
##       0       0       0
```

```
# compare multiple pieces of code to see which is most efficient
  # does the process many times so you can see which is faster
  # use the median value to compare
library("microbenchmark")

# e.g. 3 methods of finding the same value as we did above
microbenchmark(df[3, 2], df[3, "name"], df$name[3])
```

```
## Unit: microseconds
##          expr     min       lq     mean  median       uq      max neval
##       df[3, 2] 12.571 12.7765 13.44557 12.9165 13.1300  40.425   100
##  df[3, "name"] 12.422 12.7725 14.81431 12.9095 13.0975 180.276   100
##     df$name[3]  1.104  1.2695  1.47851  1.4930  1.5570   4.700   100
```

```
# 3rd methods was the fastest
```

More resources

Code styling

http://swcarpentry.github.io/r-novice-inflammation/06-best-practices-R/index.html
http://adv-r.had.co.nz/Style.html

Further learning R

http://adv-r.had.co.nz https://swirlstats.com https://rstudio.com/resources/cheatsheets/

GitHub

Thanks to Saras Windecker for these links Download git: https://git-scm.com/downloads
Create a github account: https://github.com
Jenny Bryan's Happy Git with R book https://happygitwithr.com/