Solving the most common issues with R

Bernhard Angele

23/10/2014

Setting your working directory

Despite modern operating systems trying to hide this from you as best they can, the files on your computer are organised in a hierarchical structure of folders (also called directories) and subfolders (or subdirectories). You access a file by telling the system its file path (that is, its location in the directory structure) and its filename. In most programmes, you do this via an `Open File...` or `Save File...` window. In R, you have to tell the system the file path and name directly. To simplify matters (and so you don't always have to specify the full file path), R will allow you to set a **working directory** where it will look for files that you might need to open and where it will save files.

The format of the file paths varies by operating system: In Windows, paths begin by specifying a drive (denoted by a capital letter, e.g. A, B, or C, and a colon+backslash `:\`). The most common name for a hard drive is `C:\`, although on your computer the correct drive could also be `D:\` or `E:\` or `F:\`.

In Mac OS (and Unix/Linux), paths begin with `/`. This is called the *root* directory. From the root directory, you usually have to go through several subdirectories to get to your file location. For example, in Windows, you might have to go from `C:\` to `Users` to `JaneDoe` (or whatever your user name is) to `Documents` to `AdvancedStatistics` to find the file `Homework3.Rmd`. In file path format, this would be written as `C:\Users\JaneDoe\Documents\Advanced Statistics\Homework4_data.csv`.

On a Mac, you might have to go from `/` to `Users` to `JaneDoe` (or whatever your user name is) to `Documents` to `AdvancedStatistics` to find the file `Homework4_data.csv`. In file path format, this would be written as `/Users/JaneDoe/Documents/AdvancedStatistics/Homework4_data.csv`. On a Mac, you can also use `~/` as a shortcut for `/Users/JaneDoe/` and write your path as `~/Documents/AdvancedStatistics/Homework4_data.csv`.

It is highly recommended to set your working directory properly in R so that you don't have to write out the path every time you want to open or save a file. You do this by using the `setwd` command. You give `setwd` the desired path as a character string, that is, enclosed in quotation marks (`""`).

In Mac OS, this is straightforward:

```
setwd("/Users/JaneDoe/Documents/AdvancedStatistics/")
```

In Windows, things are a little less obvious since backslashes in file paths are not allowed in R. Because of this, you have to replace the backslashes in your Windows path with forward slashes:

```
setwd("C:/Users/JaneDoe/Documents/AdvancedStatistics/")
```

After you do this, you can open your data file without having to specify the path:

```
hw4data <- read.csv("Homework4_data.csv")
```

Getting RStudio to set the path for you

Open or create and save a file in the desired directory. In the menu bar, go to `Session` –> `Set Working Directory` –> `To Source File Location`. RStudio will generate a `setwd` command with the correct path and send it to the Console. You can simply copy it from there to your script or Rmd file.

Special directories

If you put your file on the desktop, it will be in `C:\Users\YourUsername\Desktop\` under Windows (replace `YourUsername` with your actual username and `C` with the correct drive letter if necessary) or `~/Desktop/` on a Mac. It is highly recommended to not keep your files on your desktop. Instead, keep them organised in a subfolder of your `Documents` folder (`C:\Users\YourUsername\Documents` on Windows or `~/Documents` on a Mac).

Access denied

If you get the file path from RStudio and it contains the word "Temp" or some variation of it, you have opened the file from inside a compressed file (e.g. a zip file) and Windows or Mac OS has extracted it to a temporary location so you can look at it. Usually, the system will not allow you to write to a temporary directory and you will get "Access denied" errors. Extract the file from the zip file to a normal location (preferably somewhere in your Documents folder) to be able to change it. In general, do not start working with a file unless you are absolutely sure where it is on your computer.

File not found/Cannot open the connection/cannot change working directory

This means that the directory you are trying to change into or the file you are trying to open was not found in the location that you indicated. *Very* carefully check the spelling of your file path and/or file name and check that the directory and file actually exist. If all fails, move the files into a different folder and change the path accordingly.

The + sign

If you are working with the console, and you press enter without finishing a command, R will print a `+` sign in the console to indicate that it is waiting for the end of your command. Press ESC to cancel entering the command and fix it.

Not finishing commands

Reasons why your command isn't finished may be that you forgot to close a parenthesis `)` or a curly brace `}`, or a quotation mark is missing (`"` or `'`).

Extraneous characters in your R code

There are characters that can only occur in certain locations in your code. For example, a line *cannot* start with `,`, `>`, `<`, `{`,`}`,`[`, or `]`. An exception is if the previous line isn't finished and would continue with that character, e.g. in a function definition:

```
hello_world <- function(){
  print("hello world")
  }
```

Using the editor as a scratchpad

Do *not* use the editor to paste parts of the R output from the console and save it for later. This is setting you up for disaster later when you want to run your script and end up getting dozens of errors due to extraneous characters and non-code being in your script. The ideal workflow is to strictly keep your script R code-only (plus comments) and use the code to have R print the data that you need to the console. You should not need a scratchpad because you can ask R to pull up whatever output you need at any time. The key to being able to do this is to include every important step in your analysis in the script.

When working with R Markdown, you should not need to copy and paste any R output either, since R should do that for you. It is fine to be lazy, though, and enter some numbers by hand. Just be aware that those won't be updated automatically if you change something about your data.

Missing commas when entering elements of a list or function parameters

If you are making a list or a vector of different objects, they must be separated by commas `,`. For example, `c(1,2,3,4)` works, but `c(1 2 3 4)` will produce an error. Same with `t.test(x = cond1, y = cond2)` (will work) vs. `t.test(x = cond1 y = cond2)` (won't work).

Misspelling variable names, function names, parameter names, and element names

To R, `subject` and `sujbect` are two completely different names. It can't guess what you mean when you misspell something, so it will simply throw an error and tell you that it couldn't find the object. Same goes for `participantdata` vs. `participant_data` vs `participantData`. A little exercise: why will the command below not work? Hint: there are at least 3 reasons, assuming that the variable names are all correct.

```
ezANOVA(data = hw4data
        dv = time,
        betwwen = alcohol')
```

R Markdown specific issues

Code outside of code chunks, text inside code chunks Code chunks are started using "`{r} and ended using "`. Having code outside a code chunk will result in it being printed, not executed. Having text inside a code chunk will of course result in an error as R thinks it's code and tries to parse it.

```
# The only exception are comments,
# that is, lines starting with #
```

Not issues per se, but good to know

- Some hints that are not immediately obvious:
  - When you're in the Editor, press `Ctrl` + `Enter` to send the current line to the Console.
  - When you're in the Console, press the `up arrow` key to go to the last command you typed.

    * You can go back using the `down arrow` key
    * Use `Home` to go to the beginning of the line.
    * Use `End` to go to the end of the line.
- When you're either in the Editor or in the Console, press `Tab` to autocomplete the current command
- This works with fragments of commands (e.g. pressing `Tab` after typing `ezAN` will cause it to be completed to `ezANOVA`).
- It also works with function parameters (e.g. pressing `Tab` after typing `ezANOVA(` will give you a list of all possible parameters).
- `Tab` can do even more. Try it in different situations and see what it does!
- Type `?` before a command in the console to get the help page for that command.