Guidelines for data analysis

Cedric Arisdakessian

2022-07-28

This document provide some guidelines when analyzing data your own projects.

Coding principles

1. Clarity

When analyzing data, many people dive right in to the code and carry on until it produces the outputs they are expecting. Once it works, they save the file, export the figures, share it and move on. Unless you are a very experienced coder and can write excellent code on the first try, this is usually not a good idea for several reasons:

- As your project advances, you will need to adjust some parts of your code, maybe add a few steps, change some parameters. If your code is not clear and badly structured, it will take you much more time and energy to do those changes.
- Other people might not be be able to help you: If the code is congested/complicated, it will take them much more concentration and motivation to understand what you're doing. Even if they do, they might just be burnt out by the time they actually get to the core of the issue.
- If you have to get back into it days or weeks after you wrote it, you might not be able to understand your own code.

You will find here a few guiding principles to keep your code clear along with a few examples to illustrate the main points.

Before you start

2. Reproducibility

When you are analyzing data, it is pretty common to encounter functions that incorporate randomness (e.g. random subsampling when normalizing 16S data). Each time you use these functions, you get a different result, which makes your results change each time. Although this randomness is desired to avoid any sort of biases in the function behaviour, it complicates your analyses for multiple reasons:

- First, it complicates bug fixing. Depending on the output of the random function, some errors might occur later in your code or not. If the conditions to produce this error are very rare, it makes it hard to identify what causes it and therefore to fix it.
- Second, it's less ideal when you share your code with the community, or if you want to publish it. If people cannot reproduce your results, they might not trust your work.

For these reasons, most random functions include a "seed" argument, which forces the behaviour or this function to be deterministic. A seed is a fixed integer, that you can set to any value, and everytime you use this function with the same seed, you will get the same results. Naturally, different seeds should also lead to different outcomes. It is also possible that your function does not provide a seed argument (it is the case for most basic random number generators in R base), in which case you

File management

File management

In general, you want to avoid mixing all of the documents for your project in the same folder next to each other. What I propose in this section does not necessarily fit all projects, but should work well in most cases. A good rule of thumb is that your foldera should not be too congested (e.g. less than 10-15 files), but shouldn't contain a single file either.

Keep all of your work in the same folder, named after your project name and make a different subfolder for each of those categories:

a. Project code (name suggestions: "src", "code", . . .) with your .R or .Rmd files.
b. Project data (name suggestions: "data"): The data files generated in the context of your project. Don't hesitate to add subfolders if there are different types of data. For example, you could have raw sequencing reads in the "reads" folder, and then the reads processed by the pipeline in another one called "cmaiki-pipeline-outputs". If there are different types of metadata files (for example temporal/spatial), you can also consider making a folder for that.
c. Project outputs (name suggestions: "outputs", "results") for any file that your code produces. If you are generating a lot of data, you can divide this folder by output type (e.g. figures / stats / tables / . . .) or by type of analysis (ordination / permanova / deseq2 / . . .) or a

Absolute vs Relative path

Whenever you want to load a data file into R (or any tool really), you need be able to specify its location (called its *path*) on your computer. Your computer filesystem is organized in a tree-like structure and starts at the root (called "/" in Linux and MacOS, and C:/ in Windows). A branching occurs whenever you have a folder/subfolder. See below an example of a macOS file structure: