Using Projects

mjiwa

15 August, 2022

Goals

Learn how to structure code for use in R Markdown to put together easily-shared documents for reporting results

Code Transferability

During the previous section, I alluded to the fact that projects make it easier to transfer code between computers.

One of the key reasons for this is the `here()` function from the `here` package. When you have a project open, this function will return the filepath of the project.

Additionally, you can give the function a list of sub-directories as individual arguments to return the filepath of that sub-directory (e.g., `here::here("analysis", "myFile.R")` will return `"../myProject/analysis/myFile.R"`).

This means that the same functions will work across computers (which will almost certainly have the project files in different directories) and across operating softwares (which use different slashes as file separators).

Side-note

The function `here()` is not a particularly unique one (several other packages use the same function name), so you should specify which package you want to use each time you call this function (i.e., always use `here::here()` when using this function).

Using a Project – Modularising Code

Now we have a project set up, let's figure out how best to use it.

For data processing and analysis, one goal we should have is to ***modularise our code***.

This means we separate our code out into small scripts that do individual functions.

Lets start with a simple example: a script to store the IDs of datafiles that we want to work with in each of our analyses.

You can find this under: ***analysis*** **>** ***myProj_IDlist.R***.

Using a Project – Testing Modularisation

Great! Now, we can add `source(here::here("analysis", "myProj_IDlist.R"))` to the start of our analysis scripts, instead of having to copy and paste IDs between scripts (and then change every script, if we ever realise we need to exclude an ID, etc.).

Using a Project – Testing Modularisation

Lets try this out with a new script. In this one, we'll be reading in our raw data and turning it into a usable dataframe.

Find the file at ***analysis > myProj_readData.R***. You can note the use of our previous IDlist script on line 12, and another use of the `here::here()` function to read our data files on line 22.

You can run the script by running `source(here::here("analysis", "myProj_readData.R"))`. The result of this should be four variables in your R environment, including all of the trial data and demographic information for the project. All in a single line of code, using modular scripts that can run on any computer.

Using a Project – Practical Use

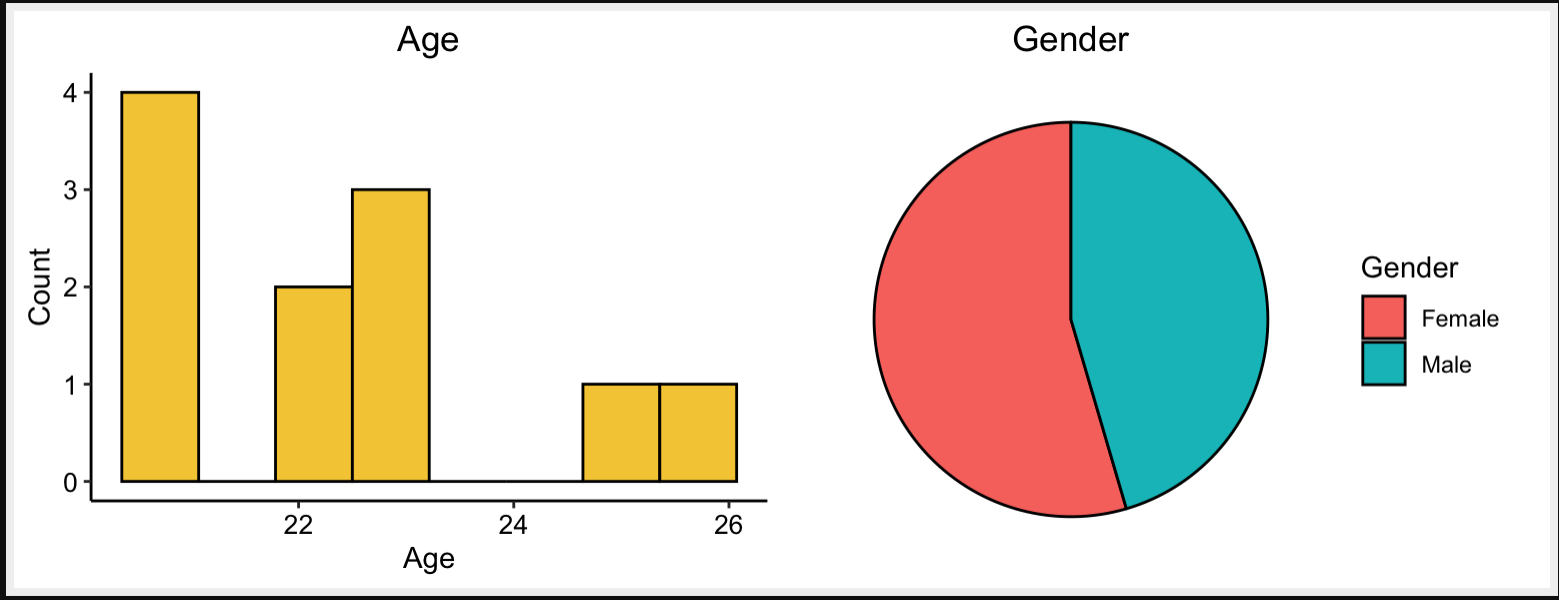
In many instances, we'll want to visualise our data to better understand it. We could make a script that:

- Loads the data
- Cleans the data
- Plots the data
- Runs analyses on the data

A better option would be to modularise these steps (as we've already started doing). We can use the modularised forms of the first two steps to use in the next step. As a toy example, I've put together a script that plots demographic information from our sample, using the `readData` script we previously put together.

You can run this script to generate the relevant output, as below:

```
source(here::here("analysis", "myProj_plotDemographics.R"))
```



Using a Project – Generating Shareable Output

We may also want to share this output with supervisors/collaborators. R Markdown offers an easy way to achieve this in a dynamic and low-memory manner.

For those unfamiliar, R Markdown documents can contain a mixture of text and R code. These are separated into "chunks", which we define using three backticks ```.

You can read more about R Markdown and code chunks here.

Using a Project – Generating Shareable Output

Our example R Markdown document is located at ***analysis*** **>** ***myProj_plotAggregated.Rmd***.

To generate shareable output, you'll need to *knit* the document using the 'knit' button in the RStudio toolbar. Knitting the document will run the code chunks and generate a combined document of the text and code output in the format specified in the header of your R Markdown file (can be html, pdf, word).

You can even use R Markdown to make slides or websites.

Example Exercise

Let’s pretend that we collected a few more datasets. We now need to add IDs 40, 43, and 44 to our sample.

If our workflow is messy, this might be annoying to implement and re-run our plotting/analyses.

Your job:

- Add IDs 40, 43, and 44 to the analysis (the data files are already in the correct folder).
- Re-run the plotting/analysis script to generate the updated output.

Take-away points

- If the prospect of adding or excluding participants from your sample (for example) is daunting – it doesn't need to be.
- By modularising our workflow, we can save time in writing and modifying code.
- If all of your R scripts start with `setwd(...)`, you could benefit from the use of projects.