# Introduction to R for reproducible science - Day 1

By David Souto

# Worksheet 1 – Need redoing: most of the info is in the .Rmd worksheets now, this can then be more of a summary of the schedule, could be just 1 doc

Files are available on the OSF page. Exercise files and solutions for this session on GitHub. We’ll download them to begin with. Later on, we will show how to clone, synch.

## Introduction to the Open and Reproducible Science (20’)

## Introduction to the RStudio interface (20’)

Follow along (see live-coding session in OSR & files, including solutions)

## Exercises: Organization and troubleshooting

### ****Organizing files****

Sharing data and materials starts with being able to organize information in a way that: 1/ doesn’t require much thinking 2/ is memorable so you can find the files later and 3/ is predictable by following a logical if not conventional structure for others.

Let’s start by creating a project folder, my preference goes for a folder that lives in the computer hard drive and not in the cloud. In recent version of windows this is not the default. It can be changed.

This is level 1 of folder / workflow management. We’ll get to level two on day 3 as we will talk about sharing data.

* + 1. Set up your computer so extensions are not hidden.

If you haven’t done so already. In windows … On a mac.

* + 1. Set up folders for different days

Self-explanatory. Try to keep the same folder and subfolder structure

1.1.3. Download the example from the workshop GitHub for day1.

Go to XXX.

1.1.4. Extract contents to the day 1 folder.

1.1.5. Note the way the files are named (<https://www.datanovia.com/en/blog/r-coding-style-best-practices/#google_vignette>). We use underscore to separate words, with brief descriptions of what it does, using verbs. We use numbers to specify the exercise number (or order of execution). E.g.:

**01\_clean.R**

**02\_analyze.R**

We don’t specify the version. That is the benefit of having a version control system, which we will implement later.

Try to apply similar conventions to all the files you will produce in the workshop.

1.1.6. Create a README.Rmd file. In Windows this can be done by creating a Text document type and changin the extension to .Rmd. Or by creating directly an Rmd file type in RStudio.

Every project folder should have a README.Rmd file. (Note how GitHub treats README files) that can explain the context of the project and the contents of the project folder.

### Finding errors in scripts and functions

Exercise 1.2.1: Adapt a piece of code with different levels of scaffolding, demonstrating loops / functions, in the context of a manual power calculation (can preview what is coming).

Code contains errors. Code produced by just one person is even more likely to contain errors. Visualizing your data fully is one way of spotting unexpected behaviour. Other ways are to implement safety checks. [See post on academia vs industry coding practice.]

You are going to spot different types of errors in the data. 1/ execution errors (wrong version, wrong package, etc.), 2/ data processing errors (e.g. zeros, variable types, …., criterion) See if you can solve them by a combination of the following tools. Look in the solution folder if you draw a blank.

**browser(): how to use**

**print(): display the state of a variable**

You will find in the folder a file that has been messed with coming from xxx. Finding errors is an overlooked topic. Ideally, we want to produce code without errors. We want to be able to spot them easily when things break. We also want to spot them when things don’t break, that is we can still run the code, but something unexpected happens.

There are many strategies to self-correct …

Lastly, how would you make the code more foolproof?

1.2.1. Source the file, using source(xxx) on the command line or press XX.

(to show strategies to find errors as we get acquainted with advanced R – we want science to self-correct) Help yourself v2. Use different ways of getting help around a function. Basic debugging (browser()). (example from e.g. a manual power calculation. Ex. From Baker’s book or something more complex code (Munafo’s code). Introduce a mistake, exercise could be to narrow down the error).

Conventions useful to make your code readable.

Level 2: Code standards / Codecheck initiatives / Code review practice within lab / GitHub session, how to invite people to collaborate on code, even ask people to review code

Level 3? Copilot or <https://codepal.ai/code-fixer> at can help you find errors in code and explain the problem. AI isn’t able to spot all errors and won’t replace a deep understanding of your data.

### Computational reproducibility

Now you have fixed the errors (or loaded the working version from the solutions folder).

Source XXX.R

Note how every run gives a slightly different number.

Understanding RNGs: Put XXX on top.

Will the code run in 10 years? Every package has a version number and can depend on other packages of a specific version number and specific versions of R. How can we show this in our code?

Level 1: You are sharing the tools (packages) you have used to perform the analysis.

sessionInfo()

or more explicity:

devtools::session\_info()

A computer screen shot of white text

AI-generated content may be incorrect.

What does “loaded via a namespace” mean? Base packages are automatically loaded and have access to those packages, but the user has no direct access to those. They need to load them by library(package). A namespace is simply a naming system to organize code.

Run library(devtools) and then session\_info()

Note you can use devtools to install specific version of a package too, with devtools::install\_version()

Level 2: You are creating a reproducible environment with packages packrat and docker (<https://rviews.rstudio.com/2018/01/18/package-management-for-reproducible-r-code/>)

We are not going to level 2. It is good to be aware of it depending on your needs.

### Documenting projects with README files

Open README.Rmd (a R Markdown file), simple formatting of text and code for documentation. We will see how we can directly code within an R markdown file later on.

1.4.1. Onen the file in RStudio

1.4.2. Onen the file in RStudio, press on Knit, then Knit to HTML



You see a formatted version of the document

1.4.3. Try out headings and “knit”

### Heading

## Heading

# Heading

1.4.4. Try a bullet point list and “knit”

\* Point 1

\* Point 2

1.4.5.

Typical in a project README

\* Project Title: The name of your project.

\* Description: A brief overview of what the project does.

\* Installation Instructions: How to set up the project.

\* Usage Examples: How to use the project effectively.

\* Contributing Guidelines: Information on how others can contribute to the project.

\* License Information: The licensing terms for the project.

This we move to 2.4

Level 2: Insert code snippets to comment code.

Pointers

Level 3: Produce reproducible and interactive documents with data.

Pointers