Session 1: Intro into reproducibility

Reproducibility, Open Software and Open Data, Reproducible Scripts

Kosmas Hench
2025-05-26

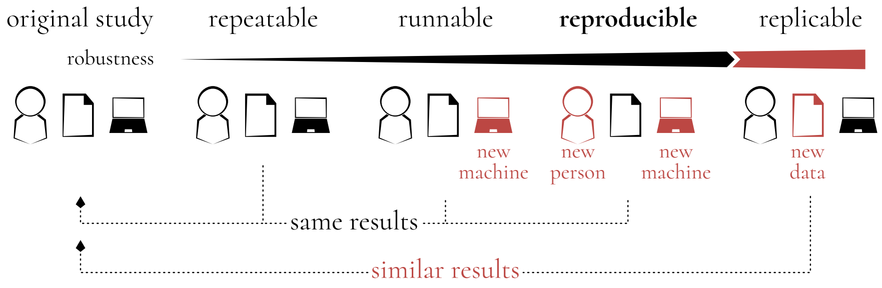
Imagine yourself reviewing a manuscript for a publication. You have inspected the results and something about the findings in Figure 2 strikes you as odd. As a consequence, you would like to confirm that the reported effect is in fact as strong as the authors claim it to be. Where do you start?
Now, imagine yourself submitting a manuscript. Using an elaborate new type of analysis, you have uncovered a very unexpected effect and show your exciting results in Figure 2…
How do you allow your readers to convince themselves that your analysis is sound?
In this workshop we are going to keep this scenario in mind. We are going to talk about tools that allow you to enable your reviewers to confirm your analysis by re-running it. A (massive) positive side effect of this is that these tools will also help you to re-run your analysis yourself if you need to.

created using dale3

1. Reproducibility as part of good scientific conduct

Results of an analysis are *repeatable* if it is possible to re-run the *same code* on the *same input data* using the *same machine* to create the *same results*. If the same results are also achievable on a different machine, the analysis is said to be *runnable*, and if this is also possible for other people the analysis is ***reproducible*** (Pellizzari *et al.*, 2017; Essawy *et al.*, 2020).

Reproducibility: an analysis is reproducible if somebody else can run the code on the original data and produce the exact same results.



One would assume that obtaining the same results from identical code this is a rather low bar to take, but unfortunately it is often much harder to guarantee than we would like. Differences in operating systems, software versions, random numbers and the usage of proprietary software or closed data formats can easily obstruct reproducibility – even if all used code and data is provided. Similar to ***replicability***, *preregistration*, or *open science*, reproducibility is thus an element of *good scientific conduct* that takes an effort achieve.

Replicability: an experiment is replicable if it can be re-done from scratch and still produces similar data. This is as more a feature of study design than of reproducibility.

Reproducibility is an ideal that one can (and should) strive for, but the sad news is that for complex studies, perfect reproducibility to the last bit might not be achievable. The good news is that there are many tools and techniques that can help us to get closer to that ideal. The aim of this workshop is to introduce some of them. Over the course of the next five sessions, we are going to introduce methods that can be applied to improve reproducibility, steadily increasing complexity as we go. The hope is that at the end, you will be able to address reproducibility of future projects with tools that are tailored to the level of complexity that you are facing.

2. Open Data, Open Software

The first hurdle for reproducibility is the question if all resources and tools required to run the analysis are freely available. This starts with the code that you have produced for the analysis, but extends to the raw data and the software used. We will get to the distribution of code in Session 2, so lets focus on data and software for now. Both should ideally to be *available* and *accessible*. This means that one should be able to find the data and the software online, but also that the data should be readable and that the software should be usable – the ideals to strive for are ***open data*** and ***open software***.

Open Data: data that is freely accessible online, ideally using non-proprietary formats.

For data to be *open*, the most important factor is that the data are in fact publicly shared. Typical all-purpose venues for sharing data repositories are platforms like PANGAEA, Dryad or figshare. Depending on the research field, there might be more specialized platforms available like genbank and the European Nucleotide Archive for genetic data or Movebank for movement data of animals. While generally all raw data and results should be shared, there are important exceptions when data should *not* be made public. Specifically personal data (e.g. names or email addresses) or other sensitive data (e.g. GPS position of endangered species subject to poaching) require special caution. In these cases data privacy might often trump reproducibility.

Besides the general availability of data, there is also the issue of data formats. Data that is available but that can not be read is of little use. A typical offender here are proprietary data formats - the most sustainable way of sharing (and archiving) data is to use open data formats that are often variants of *plain text files*. For text for example `txt` or `md` files are common, for tables `csv` or `tsv` files are preferred over `xlsx`. If your data does initially come in a proprietary format (e.g. because it is created like this by a machine that you are using), often it is possible to export the data to an open format. In that case it can be a good idea to share an open version of the data alongside the original raw data.

Open Software: software that is freely available, ideally running on windows, mac and linux.

Similar to the readability of the data, the software used for the analysis can also be a barrier to reproducibility. Proprietary software that comes with a price tag might present a paywall that excludes others from re-running your analysis. Further more software that only runs on a specific operating system excludes everyone who is dependent on a different system. Using open and free software that is cross-platform compatible is the most inclusive solution and typically also the most future-proof one. (All the tools used in this workshop should tick those boxes…)

3. Reproducible Scripts

This is a workshop about reproducible *coding*, so the focus is on analyses that rely on a scripting language (e.g. `R` or `python`) rather than on a graphical user interface (GUI). When such an analysis is done using a code script (rather than in an interactive coding session), this can already be seen as a first step towards reproducibility. This is because a script has the potential to be self-documenting: in it, every step of the analysis is written down and thus logged for posterity (Alston and Rick, 2021).

In practice this means that there is often a difference in the way that a script is written versus how it ideally should be run to produce its outputs: Integrated development environments (IDEs) like `RStudio` or `Visual Studio Code` (or `Positron`) are extremely convenient tools and I absolutely recommend using them for writing your analysis code. The initial composition of an analysis script is often a rather iterative process, and it is extremely helpful to be able to inspect the working environment (the state of assigned variable or the precise effect of specific function calls) while coding the script. However, the iterative history of an interactive coding session is usually not traceable. So, after the script has been drafted, it is best to make sure that it also runs correctly non-interactively. Bryan *et al.* call this *"saving the source code not the workspace"* (chapter 2). This means that ideally you should be able to run the script from the command line using `RScript --vanilla script.R` or `python script.py` and still get the correct output. For users of `RStudio`, this also means you want to disable the option "Restore .RData into workspace at startup" and set "Save workspace to .RData on exit" to "never" within the Global Options menu.

Still, there are a couple of aspects that can impair reproducibility of a single coding script. Aspects presenting hard barriers are *modularity* (or rather a lack thereof), the use of *random numbers* and information on *software* and *package versions*. Beyond those, reproducibility can be improved if some consideration is given to a *consistent coding style* and *clear documentation* of the code.

Modularity. The idea of modularity is sometimes also called a "*project based approach*" (Bryan *et al.* 2024: chapter 3, Wickham and Çetinkaya Rundel 2024: chapter 6). It refers to a setup in which all the data and code needed for the analysis are contained within a single folder. This folder represents a kind of stand-alone unit that can be shared without external dependencies of other custom code or information. The topic will be further discussed in SESSION 3 on project structure.

Random numbers. When a script relies on some element of randomness, this can also be an obstruction to reproducibility. The re-creation of a random number will surely differ from the original, and so will for example a random subset of samples. However, random numbers in computer code are not as random as their name implies. They are in fact created using a *random number generator* (RNG), which is an algorithm that approximates randomness. However, these RNGs are deterministic and usually allow the user to specify an explicit starting point for the random number generation process called the "seed". The function in `R` is `set.seed(<some number>)`, in `python` it is `random.seed(<some number>)` or `np.random.seed(<some number>)` (depending on the used packages). When this is added to the start of the script, every re-run of it will use the same "random" numbers.

Version numbers. A reoccurring theme in this workshop will be the fact that the exact results of an analysis often depend on the computing environment, that is: on the precise versions of programs and their packages being used. This is because over time, some functionality and syntax of the used third-pary code is bound to change. In the worst case this can result in analysis code that is simply not executable when using a later version of some used packages. A first step in enabling other people to recreate the correct working environment is to create and report a summary the environment that was used during the original analysis. In `R` this can be done with the function `sessionInfo()`, and in `python` with `session_info.show()` (from the `session_info` package).

Consistency and documentation. Finally, code that is easy to follow usually helps others (most of all yourself, later) to re-run it correctly. The clarity of code is improved if there is some consistency within it, that is when the naming of variables and functions, as well as the overall structure of the code follows a single style. Several style guides exist (e.g. from google or tidyverse). Yet, most people will end up using their own style – the guides can serve as a well-thought-out inspiration for developing a personal style. In addition to a clear code structure, energy invested in documenting code is a courtesy towards people encountering it later (again, most often this will be yourself). Unfortunately, there is no bullet-proof recipe for code documentation, both *under-* and *over-documentation* are a thing. So while documenting, one needs to strike a balance between providing necessary context and background and obfuscating the code with lengthy prose. Again, the aim should be to maximize the clarity of the overall code. One advice that I have heard in the past is to use the comments to note the aim of the code and the motivation behind it (the *why*). In contrast to these, the function (the *what*) of the code is to a large degree conveyed by the code itself.

Recap

Regardless of our project size or complexity, we want our analysis to be executable in a **non-interactive** way and **bundled** with all its dependencies in a single project directory.
We want to enable others to confirm our random processes by fixing the **seed** of the RNG and report the used **version numbers** of our software.
Finally, we try to write understandable code by using a **consistent coding style** and explaining the thought-process behind the code with appropriate **documentation**.
If all of these points are taken care of, a large step towards reproducibility has already been made.

References and Further Reading

Alston, J.M. and Rick, J.A. (2021). A Beginner's Guide to Conducting Reproducible Research. *The Bulletin of the Ecological Society of America*, 102(2):e01801. DOI: 10.1002/bes2.1801.

Bryan, J. *et al.* (2024). What They Forgot to Teach You About R. https://rstats.wtf, accessed 2024-03-12.

Essawy, B.T. *et al.* (2020). A taxonomy for reproducible and replicable research in environmental modelling. *Environmental Modelling & Software*, 134:104753. DOI: 10.1016/j.envsoft.2020.104753.

Pellizzari, E. *et al.* (2017). Reproducibility: A primer on semantics and implications for research. *RTI Press*. DOI: 10.3768/rtipress.2017.bk.0020.1708.

Sandve, G.K. *et al.* (2013). Ten Simple Rules for Reproducible Computational Research. *PLOS Computational Biology*, 9(10):e1003285. DOI: 10.1371/journal.pcbi.1003285.

Wickham, H. and Çetinkaya Rundel, M. (2024). R for Data Science (2e). https://r4ds.hadley.nz, accessed 2024-03-12.