Computational Reproducibility Cookbook

Daniel H. Baker, University of York

2023-02-15

1 Intro

This document contains notes describing a process for making scientific papers computationally reproducible. It is written in R markdown, and intended to serve as a handbook for the 'ReproduceMe' pilot project at the University of York (in 2023). Most of the examples here involve R, as we anticipate using R for most projects, however many of the same things can be achieved in other languages.

1.1 Background on reproducibility

The goal of computational reproducibility is that all of the analyses in a paper can be reconstructed. For modelling and simulation papers, this requires sharing of code. For empirical papers it requires sharing of both code and data. Although data sharing has become commonplace in recent years, researchers appear to be much less willing to share analysis code. This could be for any number of reasons, such as fear that they have made an error, or lack of confidence in their own coding skills. It could also be simply be that current norms in most fields do not require code sharing. However making one's work reproducible has numerous benefits, including increasing the transparency of the analysis, and the confidence of readers, reviewers and editors. Other researchers can then use parts of an analysis pipeline in their own work, speeding up scientific progress. Finally, it is potentially the case that a reproducible workflow potentially benefits the authors themselves if they wish to revisit their analysis in the future.

2 Five levels of computational reproducibility

Computational reproducibility can be as simple as posting a script and data online. However there are additional steps that can make things easier for the end user, integrate the analysis code with the manuscript and figure generation, and preserve the computational environment used (e.g. package versions). A useful framework is the *reproducibility pyramid* illustrated in Figure 1. The relative widths of each layer of the pyramid indicate how common each level is, though note the proportions are not to scale - the vast majority of current research is not reproducible at all. Some further comments on each level follow.

Level 0 - the study is not computationally reproducible, usually because code and/or data are unavailable. This is the case for essentially all published research from the 20th century, as well as the vast majority of work published today. Note that non-reproducible work is not necessarily of a lower quality than reproducible work, it is just that this is harder to evaluate: we must trust that the authors' account of their analysis is full and accurate. Many high profile cases of research fraud might not have been possible, or would have been caught sooner, had reproducibility been the norm, or a requirement for publication.

Level 1 - a single script conducts the analysis using local raw or preprocessed data that is manually downloaded. This is the most basic form of computational reproducibility. However it usually requires quite a lot of effort on the part of the end user. For example there may be many files that need to be downloaded and stored in specific places on the computer, or the code might need modifying to locate the local copies of the data files. In some cases it could also be necessary for the end user to separately download and/or install additional packages or code repositories in order for the code to work. Finally, the output is likely to be

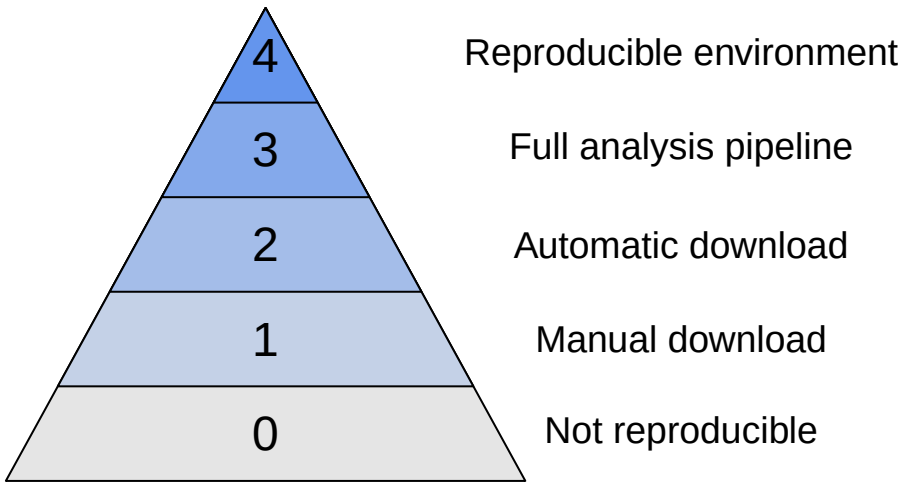

Figure 1: The reproducibility pyramid, indicating five levels of computational reproducibility.

the raw function output, e.g. for a statistical test, and it may require effort and expertise to find the values reported in the paper.

Level 2 - a single script automatically downloads and analyses raw or preprocessed data, and produces a formatted output containing values that can be incorporated into a paper. This is a more user-friendly solution, because the end user only needs to download a single script, and then all downloading of data is automated. There are several ways to do this, but the OSF provide an R package that makes it straightforward. At level 2, we anticipate that the output of any analysis is provided in a user-friendly format, such that the values included in a Results section are apparent from the output of the script.

Level 3 - a single script automatically downloads and analyses all raw data, generates all tables and figures, and produces a pdf of the entire manuscript (note that this script may execute other scripts, e.g. in different programming languages). This is a more impressive solution, as it is clear precisely where all of the values reported in the paper have come from, and how the figures were constructed. It can be substantially more work to get to this stage, but if the paper is already written then converting to an executable format is largely procedural. The journal eLife piloted something along these lines a few years ago, though it seems to have been forgotten about now.

Level 4 - all code is embedded in a Docker container (or similar) that includes the software required to run (e.g. specific package versions, and versions of the programming environment). This would be extremely technically challenging to do from scratch. However fortunately there are some useful tools already available. The Rocker project provides standard Docker containers for R studio, which can be downloaded and used as a wrapper for the entire computational environment. I have not tried doing this yet, but there is plenty of documentation, and also a useful paper by Peikert and Brandmaier (2021) that explains how to go about it. A less extreme version is to use the *renv* package to manage package versions in R.

Discussions about which level to aim for should be had with the study authors before work begins on making things reproducible. It is always easier to program something when the goal is clear, and there may be idiosyncracies specific to an individual study that means that Level 3 or 4 is not practical. Sometimes there are also restrictions on sharing of raw data (e.g. where this could potentially be used to identify a participant), and in such cases we might aim for reproducibility based on preprocessed or de-identified data. All of this is totally fine - for this pilot project our goal is simply to make things more reproducible than Level 0, so we need to be pragmatic rather than puritanical.

3 Implementation

3.1 *R Markdown* is really good

Markdown is a generic convention for document formatting. R Markdown takes this basic concept and integrates it into the R environment. This means you can produce a single script that contains both text and computer code. When the script is executed the code runs too, meaning that analyses can be conducted, and an output document is generated. It is possible to automate all parts of an analysis pipeline in this way, with different parts of the code importing and processing the data, generating figures, and formatting the output in the style of a paper. Actually you can 'knit' the markdown file to a variety of formats including pdf, Word documents, html and epub. Similar systems exist for other programming languages, including Jupyter notebooks for Python, and Matlab live scripts. We now have several examples of full papers written in R Markdown, available at the following repositories:

Baker (2021)

Baker et al., (2021)

Segala et al. (2023)

Baker et al. (2023)

In each case, the Rmd file contains the markdown script that will auto-generate everything in the paper.

3.2 Markdown file structure

one big chunk, all at start, interleaved throughout

3.3 Flags for level of analysis

3.4 Automating figure generation

3.5 Downloading data and other resources

3.6 Package management

renv and automatic activation

3.7 Python chunks

3.8 Command line and Matlab functions

3.9 Output formats

3.10 Typesetting equations using LaTeX

When Markdown creates pdf files as output, it actually uses LaTeX (pronounced 'laytech', or sometimes 'laatech'). This is a document typesetting system very popular in the mathematical sciences because it is good at typesetting equations, and it is possible to incorporate pieces of LaTeX code in a Markdown file (such as the fancy text in this paragraph for the word Latex). Some journals also accept LaTeX files for submission.

A full overview of how to use LaTeX is beyond the scope of this document. However it is straightfoward to incorporate a simple equation between pairs of dollar symbols. For example, the text `$y = \sqrt{x^2}$` generates the following equation: $y = \sqrt{x^2}$

3.11 Citations and automatic figure/equation referencing

3.12 Github syncing

4 An example:

Uploading data Downloading data Analysis and embedding code in text Autogenerating figures Github syncing Renv

5 Modality-specific issues

MRI, manual analyses

References

Peikert A, Brandmaier AM. 2021. A reproducible data analysis workflow with R markdown, git, make, and docker. *Quantitative and Computational Methods in Behavioral Sciences* **1**:e3763. doi:10.5964/qcmb.3763