

Building a reproducible workflow in R

Project-oriented workflow

Daniela Palleschi

Wed Aug 21, 2024

Table of contents

Building a reproducible workflow in R	1
Broadening the reproducibility spectrum	2
Project management	2
Naming conventions	2
Literate programming	3
Documentation	4
Version control (not covered in this workshop)	4
Persistent (public) storage	4
Writing (not covered in this workshop)	5
Setting up a project	5

Learning Objectives

Today we will learn...

- about reproducibility practices beyond sharing code and data
- about project-oriented workflows
- what we will cover in this workshop

Building a reproducible workflow in R

- we now know some important principles of a reproducible workflow
 - and that ‘reproducibility’ is not black-and-white

- but even the reproducibility spectrum is an oversimplification (Peng, 2011)
- some additional resources that provide a list of tips include:
 - Bowers & Voors (2016); Nagler (1995); Wilson et al. (2017); Corker (2022)

Broadening the reproducibility spectrum

- there are different levels of reproducibility
 - the *bare minimum* is sharing the code and data
 - *and* including session information:
 - * which operating system was used
 - * which software/package versions were used
- going bigger:
 - project-oriented workflow
 - project-specific filepaths
 - contained in a single project folder
- we will be using RProjects to achieve this

Project management

- folder structure
- project-relative file paths
- appropriate documentation
 - e.g., README
- it's great to map out your project structure early on
 - but it will grow as you go along
 - reproducible principles facilitate adapting as it grows

Naming conventions

- there are some “rules” for naming files and folders
 - [The Turing Way: Naming files, folders, and other things](#)
 - [Jenny Bryan: naming things \(Reproducible Science Workshop 2015\)](#)
- 1. Avoid special characters
 - ensures machine readability

2. Make names concise but meaningful
 - ensures human-readability
3. Avoid spaces
 - try `CamelCase`, snake case (`snake_case`), or skewer case (`skewer-case`)
 - or use hyphens (-) to separate chunks, and underscores (_) to connect words of the same chunk
4. Consider default ordering
 - e.g., with dates: `YYYY-MM-DD`
 - with folders or files: numerical prefixes (e.g., `01-data_cleaning.R`, `02-data_visualisation.R`)
5. Be *consistent*

Literate programming

Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

— Knuth (1984), p. 97

- originally used to refer to writing programs
- but also applies to analysis code
 - especially if we’re aiming for reproducibility
- main concepts:
 - code is linear (this pre-dates Knuth, 1984)
 - informative but concise commenting
- main benefits:
 - facilitates maintenance
 - helpful for future-you, collaborators, etc.

Documentation

- metadata
 - project README
 - codebook/data dictionary
- README should contain
 - a project description
 - relevant links
 - description of folder structure
- can be updated as the project develops
- README.md files in GitHub/Lab are automatically used as a project description
 - .md is a plain text document
 - uses markdown syntax

Version control (not covered in this workshop)

- git: local tracking
- useful for the analysis and writing phases
 - but can be tricky for collaboration
- GitHub/GitLab: remote tracking
 - store your changes to your local git repository
 - then push them to your remote repository
- safe guards against local hardware/software issues
 - lost or damaged computer or local files
- and allows for collaboration or sharing

Persistent (public) storage

- GitHub/Lab are sub-optimal
 - developer-focused
 - typically lack thorough documentation/metadata
 - not very user-friendly for non-users
- OSF, Zenodo

- Open Science-focused
- can be linked to a GitHub/Lab repository
- facilitate thorough documentation
- user-friendly

Writing (not covered in this workshop)

- dynamic reports with Markdown syntax
 - e.g., Rmarkdown, Quarto
 - integration of data, code, and prose
 - * facilitates cross-referencing within document
 - * integration of citation management tools
 - * supports LaTeX syntax for example sentences and tables
- `papaja` package for APA-formatted Rmarkdown documents
- challenge: collaboration
 - not all collaborators know these tools
 - track changes not currently possible

Setting up a project

- tomorrow: hands-on
- required installations/recent versions of:
 - R
 - * preferably version 4.4.0, “Puppy Cup”
 - * check current version with `R.version`
 - * download/update: <https://cran.r-project.org/bin/macosx/>
 - RStudio
 - * preferably version 2023.12.1.402, “Ocean Storm”
 - * Help > Check for updates
 - * new install: <https://posit.co/download/rstudio-desktop/>

Learning objectives

Today we learned...

- about reproducibility practices beyond sharing code and data
- about project-oriented workflows
- what we will cover in this workshop

References

- Bowers, J., & Voors, M. (2016). How to improve your relationship with your future self. *Revista de Ciencia Política (Santiago)*, 36(3), 829–848. <https://doi.org/10.4067/S0718-090X2016000300011>
- Corker, K. S. (2022). An Open Science Workflow for More Credible, Rigorous Research. In M. J. Prinstein (Ed.), *The Portable Mentor* (3rd ed., pp. 197–216). Cambridge University Press. <https://doi.org/10.1017/9781108903264.012>
- Knuth, D. (1984). Literate programming. *The Computer Journal*, 27(2), 97–111.
- Nagler, J. (1995). Coding Style and Good Computing Practices. *PS: Political Science & Politics*, 28(3), 488–492. <https://doi.org/10.2307/420315>
- Peng, R. D. (2011). Reproducible Research in Computational Science. *Science*, 334(6060), 1226–1227. <https://doi.org/10.1126/science.1213847>
- Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6), e1005510. <https://doi.org/10.1371/journal.pcbi.1005510>