CSSS508, Lecture 10

Model Results and Reproducibility

Michael Pearce
(based on slides from Chuck Lanfear)

May 31, 2023

Topics

Last time, we learned about,

1. Basic mapping: `ggplot`, `ggmap`, and `ggrepel`
2. Advanced mapping: GIS with `sf` and `tidycensus`

Today, we will cover,

1. Reproducible research
2. Best practices
3. Wrapping up the course!

Reproducible Research

Why Reproducibility?

Reproducibility is not *replication*.

- **Replication** is running a new study to show if and how results of a prior study hold.
- **Reproducibility** is about rerunning *the same study* and getting the *same results*.

Reproducible studies can still be *wrong*... and in fact reproducibility makes proving a study wrong *much easier*.

Reproducibility means:

- Transparent research practices.
- Minimal barriers to verifying your results.

Any study that isn't reproducible can be trusted only on faith.

UW CS&SS

Reproducibility Definitions

Reproducibility comes in three forms (Stodden 2014):

1. **Empirical:** Repeatability in data collection.

2. **Statistical:** Verification with alternate methods of inference.

3. **Computational:** Reproducibility in cleaning, organizing, and presenting data and results.

R is particularly well suited to enabling **computational reproducibility**.[1]

They will not fix flawed research design, nor offer a remedy for improper application of statistical methods.

Those are the difficult, non-automatable things you want skills in.

[1] Python is equally well suited.

Computational Reproducibility

Elements of computational reproducibility:

- **Shared data**

  - Researchers need your original data to verify and replicate your work.

- **Shared code**

  - Your code must be shared to make decisions transparent.

- **Documentation**

  - The operation of code should be either self-documenting or have written descriptions to make its use clear.

- **Version Control**

  - Documents the research process.
  - Prevents losing work and facilitates sharing.

UW CS&SS

Levels of Reproducibility

For academic papers, degrees of reproducibility vary:

1. "Read the article"

2. Shared data with documentation

3. Shared data and all code

4. **Interactive document**

5. **Research compendium**

Interactive Documents

Interactive documents—like R Markdown docs—combine code and text together into a self-contained document.

- Load and process data
- Run models
- Generate tables and plots in-line with text
- In-text values automatically filled in

Interactive documents allow a reader to examine your computational methods within the document itself; in effect, they are self-documenting.

By re-running the code, they reproduce your results on demand.

Common Platforms:

- **R:** R Markdown
- **Python:** Jupyter Notebooks

Research Compendia

A **research compendium** is a portable, reproducible distribution of an article or other project.

Research compendia feature:

- An interactive document as the foundation

- Files organized in a recognizable structure (e.g. an R package)

- Clear separation of data, method, and output. *Data are read only*.

- A well-documented or even *preserved* computational environment (e.g. Docker)

`rrtools` by UW's [Ben Markwick](#) provides a simplified workflow to accomplish this in R.

Bookdown

`bookdown`—which is integrated into `rrtools`—can generate documents in the proper format for articles, theses, books, or dissertations.

`bookdown` provides an accessible alternative to writing $\LaTeX$ for typesetting and reference management.

You can integrate citations and automate reference page generation using bibtex files (such as produced by Zotero).

`bookdown` supports `.html` output for ease and speed and also renders `.pdf` files through $\LaTeX$ for publication-ready documents.

For University of Washington theses and dissertations, consider Ben Marwick's `huskydown` package which uses Markdown but renders via a UW approved $\LaTeX$ template.

Best Practices

Organization Systems

Organizing research projects is something you either do accidentally—and badly—or purposefully with some upfront labor.

Uniform organization makes switching between or revisiting projects easier.

I suggest something like the following:

```
project/
    readme.md
    data/
        derived/
            processed_data.RData
        raw/
            core_data.csv
    docs/
        paper.Rmd
    syntax/
        functions.R
        models.R
```

1. There is a clear hierarchy
    - Written content is in `docs`
    - Code is in `syntax`
    - Data is in `data`
2. Naming is uniform
    - All lower case
    - Words separated by underscores
3. Names are self-descriptive

Workflow versus Project

To summarize Jenny Bryan, <u>one should separate workflow from projects.</u>

Workflow

- The software you use to write your code (e.g. RStudio)

- The location you store a project

- The specific computer you use

- The code you ran earlier or typed into your console

Project

- The raw data

- The code that operates on your raw data

- The packages you use

- The output files or documents

Projects *should not modify anything outside of the project* nor need to be modified by someone else (or future you) to run.

Projects *should be independent of your workflow*.

Portability

For research to be reproducible, it must also be *portable*. Portable software operates *independently of workflow* such as fixed file locations.

Do Not:

- Use `setwd()` in scripts or .Rmd files.
- Use *absolute paths* except for *fixed, immovable sources* (secure data).
    - `read_csv("C:/my_project/data/my_data.csv")`
- Use `install.packages()` in script or .Rmd files.
- Use `rm(list=ls())` anywhere but your console.

Do:

- Use RStudio projects (or the here package) to set directories.
- Use *relative paths* to load and save files:
    - `read_csv("./data/my_data.csv")`
- Load all required packages using `library()`.
- Clear your workspace when closing RStudio.
    - Set *Tools > Global Options... > Save workspace...* to **Never**

Divide and Conquer

Often you do not want to include all code for a project in one `.Rmd` file:

- The code takes too long to knit.
- The file is so long it is difficult to read.

There are two ways to deal with this:

1. Use separate `.R` scripts or `.Rmd` files which save results from complicated parts of a project, then load these results in the main `.Rmd` file.

   - This is good for loading and cleaning large data.
   - Also for running slow models.

2. Use `source()` to run external `.R` scripts when the `.Rmd` knits.

   - This can be used to run large files that aren't impractically slow.
   - Also good for loading project-specific functions.

Tools

Some opinionated advice

On Formats

Avoid "closed" or commercial software and file formats except where absolutely necessary.

Use open source software and file formats.

- It is always better for *science*:

  - People should be able to explore your research without buying commercial software.
  - You do not want your research to be inaccessible when software is updated.

- It is often just *better*.

  - It is usually updated more quickly
  - It tends to be more secure
  - It is rarely abandoned

The ideal: Use software that reads and writes *raw text*.

On Text

Writing and formatting documents are two completely separate jobs.

- Write first
- Format later
- Markdown was made for this

Word processors—like Microsoft Word—try to do both at the same time, usually badly.

They waste time by leading you to format instead of writing.

Find a good modular text editor and learn to use it:

- [Overleaf] (https://www.overleaf.com)
- Atom
- Sublime (Commercial)

On Version Control

Version control originates in collaborative software development.

The Idea: All changes ever made to a piece of software are documented, saved automatically, and revertible.

Version control allows all decisions ever made in a research project to be documented automatically.

Version control can:

1. Protect your work from destructive changes
2. Simplify collaboration by merging changes
3. Document design decisions
4. Make your research process transparent

UW CS&SS

Git and GitHub

`git` is the dominant platform for version control, and [GitHub](#) is a free (and now Microsoft owned) platform for hosting **repositories**.

Repositories are folders on your computer where all changes are tracked by Git.

Once satisfied with changes, you "commit" them then "push" them to a remote repository that stores your project.

Others can copy your project ("pull"), and if you permit, make suggestions for changes.

Constantly committing and pulling changes automatically generates a running "history" that documents the evolution of a project.

`git` is integrated into RStudio under the *Tools* menu. [It requires some setup.](#)[1]

[1] You can also use the [GitHub desktop application](#).

GitHub as a CV

Beyond archiving projects and allowing sharing, GitHub also serves as a sort of curriculum vitae for the programmer.

By allowing others to view your projects, you can display competence in programming and research.

If you are planning on working in the private sector, an active GitHub profile will give you a leg up on the competition.

If you are aiming for academia, a GitHub account signals technical competence and an interest in research transparency.

Wrapping up the Course

What You've Learned

A lot!

- How to get data into R from a variety of formats
- How to do "data custodian" work to manipulate and clean data
- How to make pretty visualizations
- How to automate with loops and functions
- How to combine text, calculations, plots, and tables into dynamic R Markdown reports
- How to acquire and work with spatial data

You all are now **R**ockstars!!

What Comes Next?

- **Learn more statistics!! (e.g. take more CSSS courses)**

  - Learn foundations to statistical inference, create and evaluate models, consider survey design, make fancy visualizations, etc.
  - All of this is much easier to do if you already know R!

- **Practice, practice, practice!**

  - Replicate analyses you've done for practice (maybe in another language)
  - Think about data using `dplyr` verbs, tidy data principles
  - R Markdown for reproducibility

- **Do more advanced projects**

  - Use version control (git) in RStudio
  - Create interactive Shiny web apps
  - Write your own functions and put them in a package

Course Plugs

If you...

- would like to review math - **CSSS 505: Review of Math for Social Scientists**
- have no stats background yet - **SOC 504: Applied Social Statistics**
- want to learn some stat theory - **CSSS 510: Maximum Likelihood**
- want to master visualization - **CSSS 569: Visualizing Data**
- study events or durations - **CSSS 544: Event History Analysis**
- want to use network data - **CSSS 567: Social Network Analysis**
- want to work with spatial data - **CSSS 554: Spatial Statistics**
- want to work with time series - **CSSS 512: Time Series and Panel Data**

Thank you!

- Please submit your course evals! I *greatly appreciate* any feedback you may have.
- Remember to submit your final assignment (HW 8; due now!) and provide peer review feedback by Monday at 11:59pm!
- Hand in (optional) HW 9 if you are short of the 20 points necessary to pass.
- Feel free to reach out at any point in the future with questions or comments!