

Introducing software development best practices for research in the behavioral and social sciences

...

Pablo Caceres

Dec - 2019

University of Wisconsin-Madison

Why should I care? I'm a scientist, not a coder 🙄

Graham Lea @evolvable · May 2, 2018

science code doesn't need to follow the rules of good software engineering, because science is not about creating software but about experimenting with building prototypes of models. 🙄 Great tip from [@jeremyphoward](#)



François Chollet ✓
@fchollet

Buggy code is bad science. Poorly tuned benchmarks are bad science. Poorly factored code is bad science (hinders reproducibility, increases chances of a mistake). If your field is all about empirical validation, then your code *is* a large part of your scientific output.

This is why

Research in the behavioral and social sciences (B&SS) is **increasingly relying on complex computational procedures**

In general, B&SS have **little formal training** in data management and software development best practices for scientific computing

As result, we have a high **“technical debt”** in scientific computing that is **impairing** the pace, quality, and trustworthiness of scientific results

My proposal : Introducing a set of **basic principles** in data management and software development may significantly improve research practices that rely on heavy computation

Keep in mind...

I'm not a software engineer and I don't have a CS degree 🙋

This is **not about** the best way of writing code (kinda...)

This is **not about** the best programming language for the B&SS (kinda too...)

This is **not about** the best statistical procedures or machine learning approaches

This is **not about** deploying data pipelines in production systems

Keep in mind...

This is about what I believe are **good software engineering practices for scientific computing in the B&SS**

This is more useful for projects that **do not require large scale computing** (HTC, HPC, Hadoop, Spark, etc.)

Based on **my experience** writing code for scientific research in sociology, public policy/economics, cognitive neuroscience and psychology in the past ~8 years:

- **SPSS** and **Excel** for 4 years (don't do this...) 🙅
- **STATA** and **MATLAB** for 2 years (don't do this either if you can...) 🙅
- **Python** and **R** for 2.5 years (definitely do this if you can!) 🙋

This is very opinionated talk, yet, informed by recommendations of experts in scientific computing and software development

What is this about then?

This is about **simple principles** that allow creating data projects which are:

REPRODUCIBLE: Others can produce my exact same results given the same data

REUSABLE: I can reuse part of my code base for future projects

RELIABLE: I can trust my results

MAINTAINABLE: I can safely modify my pipeline in the future or easily fix my code

EXTENSIBLE: I can add more analysis, plots, etc, without creating a mess

SHAREABLE: I can share my code base and results with my group and the wider scientific community (I can put my code online without feeling embarrassed 🙄)

Outline

1. Use free and open-source software (FOSS)
2. Create simple and well-organized data file system ("code base")
3. Use virtual environments
4. Use version control systems
5. Add and maintain documentation
6. Maintain your raw-data (single authoritative version of your data)
6. Use well-tested and supported libraries
7. Text editors
8. Use this 8 simple principles to write better code
9. If doing ML: use a experiment tracking system
10. Test your code
11. Code reviews (when possible)
12. More resources

Use free and open-source software (FOSS)

Python, R, Scala, SQL, Julia, Bash, etc. instead of SPSS, STATA, MATLAB, SAS, etc.

Otherwise, reproducibility, reusability, and shareability, are not possible

Far more people use FOSS than proprietary software

Proprietary software is not accessible in lower income countries

FOSS is the present AND the future of data science ecosystems

Most new statistical and machine learning procedures are created in FOSS

Create simple and well-organized data file system

Separate code from data and results

Split your code into modules

Add a **README.md** file

Add a **LICENSE.txt**

Add a **requirements.txt** file

```
|--\my_awesome_project
| |README.md
| |LICENSE.txt
| |requirements.txt
| |--\code
| | |--cool-script.py
| | |--helper-script.py
| | |--run-experiment.sh
| |--\data
| | |--fantastic-data.csv
| | |--fantastic-data_schema.csv
| |--\docs
| | |--brilliant-manuscript.pdf
| |--\results
| | |--fig-1.tiff
| | |--fig-2.tiff
| | |--table-1.tiff
| | |--table-2.tiff
| |--\test
| | |--test-my-code.py
```

Use virtual environments

Virtual environments create an **isolated environment** for your project

For Python **venv** or **conda environment**

For R **conda environment**

If you're feeling adventurous, use **Docker containers**

Docker containers are an **ideal solution** for reproducibility, you may have to put more effort in managing some extra complexity

CODING TIME



Use version control systems

- Use **Git**, Mercurial, or SVN
- **Push frequently** after changes
- Use **branches** for experimentation and collaboration
- Use a third **backup system** with continuous sync (Box, Dropbox, Drive, etc.)
- Optimized for **plain text** (txt, md). Not that good for tabular data, docx, or PDFs.
- **Bad for "large files"** (limited to 100 megabytes per file in GitHub).
- When files are too large: **zip** the files (OK). **Read data from the web** (Better)
- **Raw data should not change**, and therefore, should not require version tracking
- BUT, if your data is changing, consider using a **data version control system** (DVC)
- Keep minimal **notes/logs** of major changes

CODING TIME



Add and maintain documentation

README.txt: about, installing instructions, usage instructions, etc.

LICENSE.txt: MIT is the standard

Requirements.txt: list all dependencies required to run and reproduce your work

To-do.txt: to keep track of things to be fixed, added, etc.

Push frequently to maintain snapshots of your projects at different timepoints. This works as **automated documentation**.

Maintain your raw-data (a single authoritative version)

Readable and consistent **naming**

Use unique and consistent **identifiers**

Add **dates**

Keep a data **schema**

Keep multiple **copies** of the raw data

Use open non-proprietary **formats** (CSV, JSON, XML, etc.)

For relatively **small data files**, create clean data “on the fly”

For relatively **large data files**, create (repeatable) intermediate files

Use well-tested and properly supported software libraries

Why? Because popular libraries are:

- Thoroughly **tested** (less bugs)
- Optimized for **performance**
- Are **safer**
- Better **documented**
- Better **supported** (e.g., Stack Overflow, books, blogs)
- Have more **tutorials**
- More likely to be **maintained** long-term

Review what are the **most popular** libraries in your field

Python: pandas, numpy, scikit-learn, matplotlib, seaborn, Tensorflow, Keras, Pythorch, XGBoost, statsmodels, etc.

R: ggplot2, dplyr, data.table, glm, prophet, lme4, glmnet, etc.

Check **GitHub activity** and downloads as a guide

IDEs: Are Jupyter Notebooks & RStudio the solution?

What are good for?

- Plots
- Tutorials
- One time tasks
- Data exploration
- Creating narratives
- Instant feedback

* RStudio > Jupyter

** RStudio < Jupyter

What are bad (or suboptimal) for?

- Collaboration
- Poor or suboptimal text editor support*
- Testing
- Dependency management
- Version control*
- Modularization* (can't import notebooks)
- Maintenance (hard to edit and change)
- **Reproducibility** (out of order execution)** 🧙
- Encourages bad coding habits
- Require extra discipline
- Add cognitive load
- Bad for industrial applications

Jupyter Notebooks

*“Yet, as with other computing environments, using notebooks for research requires **special care**. Interactively running and editing code in notebooks can **delete key steps** or introduce “**hidden state**” that **confounds analyses and confuses readers** [12]. Analyses documented in notebooks **cannot be easily rerun if users do not first freeze their dependencies, share their data, and adequately describe their computing environment** [13]. And many notebooks **lack sufficient descriptive text to guide readers in using them** [11,14].”*

**If you REALLY want
to use it, use it well
and carefully**

EDITORIAL

Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks

Adam Rule¹, Amanda Birmingham², Cristal Zuniga³, Ilkay Altintas⁴, Shih-Cheng Huang^{4a}, Rob Knight^{3,5}, Niema Moshiri⁶, Mai H. Nguyen⁴, Sara Brin Rosenthal², Fernando Pérez⁷, Peter W. Rose^{4*}



Good for camping, suboptimal as a vehicle and as a house

What to do then?

Only text editors: hard and slow for exploration phase, but really good for reproducibility and long term maintainability of the project. **Best for ML/Heavy computation projects.**

Both: the “right tool for the right job”, at the **cost** of increasing complexity and maintainability of the code base. **Best for hybrid projects.**

Only Jupyter/RStudio: nice for exploration phase, but really bad for reproducibility and long term maintainability of the project. **Best for one-time data exploration, tutorials, blog post.**



Use this 8 simple principles to write better code 🐙

1. Write **modular** code
2. **Explicit** is better than implicit
3. Write **DRY** (Don't repeat yourself) code
4. Use **consistent** and **transparent** naming
5. **Iterate and re-run:** particularly if using IDE that allows for **out of order execution** like Jupyter Notebooks and R-Studio
6. Avoid **premature optimization**
7. **Refactor** code as needed
8. **Test code** for critical issues

🐛🐛🐛🐛🐛🐛🐛 CODING TIME 🐛🐛🐛🐛🐛🐛🐛



If doing ML: use a experiment tracking system

Machine learning is **iterative** and **experimental**

Tracking progress and changes is **hard and messy**

Reproducibility in ML is a problem

Tracking systems **facilitate** training, evaluation, reporting, and reproducibility

For Python: [Weights and Biases](#) or [Sacred](#)


For R and Python: [Mlflow](#) (more complete, but more complex to use)

🐱🐱🐱🐱🐱🐱🐱 CODING TIME 🐱🐱🐱🐱🐱🐱🐱



Do code reviews (when possible)

Code reviews are critical for:

- Finding **bugs** 
- Improve code **readability**
- Improve **usability**
- Improve code **performance**
- Improve **documentation**
- **Sharing progress** with your group/peers
- Efficient **knowledge exchange**

Note for R users

This are **language agnostic recommendations** (FOOS, dynamic languages)

There are tons of **R-specific** recommended best-practices online you can look at:

- R Studio projects
- RMarkdown
- R Packages for dependency management, control workflows, paths, etc

These are perfectly good tools for development, but be mindful of some limitations...

Don't transfer well to other programming languages

They add complexity by trying to hide software development complexity

Path of least resistance problem



A snippet of psychological wisdom from Skinner:

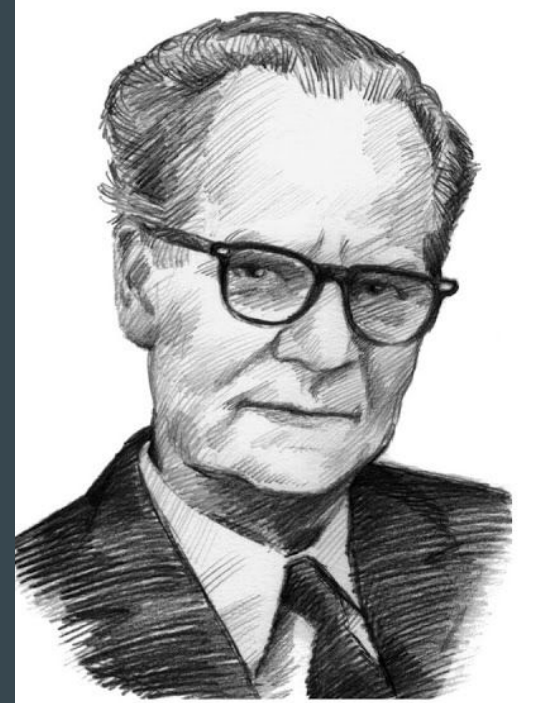
Controlling response: a behavior that makes future behavior more or less likely

E.g.: Prepare lunch the night before

Controlled response: the target behavior to be controlled in the future

E.g.: Eating healthier

Design an environment and pick tools that will ‘force’ you to make better science



Resources to learn

Version control:

- Tutorials: [Git/GitHub](#)
- Book: [Git-Book](#)

Virtual environments:

- [Virtualenv package](#)
- [Conda environments management](#)
- Tutorial: [virtual environments in Python](#)

Refactoring:

- Book: [Refactoring](#)
- Tutorial: [Production Data Science](#)

ML tracking systems:

- Tutorial: [Weights & Biases](#)
- Tutorial: [Mlflow](#)

Testing code:

- Talk: [Testing for Data Scientist](#)
- Tutorial: [Unit testing in Python](#)

Modular and Maintainable code for ML/Data Science:

- Talk: [Reproducible Data Science in Python](#)
- Talk: [Maintainable Code in Data Science](#)

References (General scientific computing)

1. Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., ... & Waugh, B. (2014). [Best practices for scientific computing](#). PLoS biology, 12(1), e1001745.
2. 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.
3. Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). [Ten simple rules for reproducible computational research](#).
4. Hart, E. M., Barmby, P., LeBauer, D., Michonneau, F., Mount, S., Mulrooney, P., ... & Hollister, J. W. (2016). [Ten simple rules for digital data storage](#).
5. Rule, A., Birmingham, A., Zuniga, C., Altintas, I., Huang, S. C., Knight, R., ... & Rose, P. W. (2019). [Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks](#). PLoS computational biology, 15(7).
6. Perez-Riverol, Y., Gatto, L., Wang, R., Sachsenberg, T., Uszkoreit, J., Leprevost, F. da V., ... Vizcaíno, J. A. (2016). [Ten Simple Rules for Taking Advantage of Git and GitHub](#). PLOS Computational Biology, 12(7), e1004947.
<https://doi.org/10.1371/journal.pcbi.1004947>
7. Taschuk, M., & Wilson, G. (2017). [Ten simple rules for making research software more robust](#). PLOS Computational Biology, 13(4), e1005412. <https://doi.org/10.1371/journal.pcbi.1005412>
8. Hinsien, K. (2015). [Technical Debt in Computational Science](#). Computing in Science & Engineering, 17(6), 103–107.
<https://doi.org/10.1109/MCSE.2015.113>

References (R language)

1. Gandrud, C. (2016). Reproducible research with R and R studio. Chapman and Hall/CRC.
2. Marwick, B., Boettiger, C., & Mullen, L. (2018). Packaging data analytical work reproducibly using R (and friends) [Preprint]. <https://doi.org/10.7287/peerj.preprints.3192v2>
3. Lowndes, J. S. S., Best, B. D., Scarborough, C., Afflerbach, J. C., Frazier, M. R., O'Hara, C. C., ... Halpern, B. S. (2017). Our path to better science in less time using open data science tools. Nature Ecology & Evolution, 1(6), 0160. <https://doi.org/10.1038/s41559-017-0160>