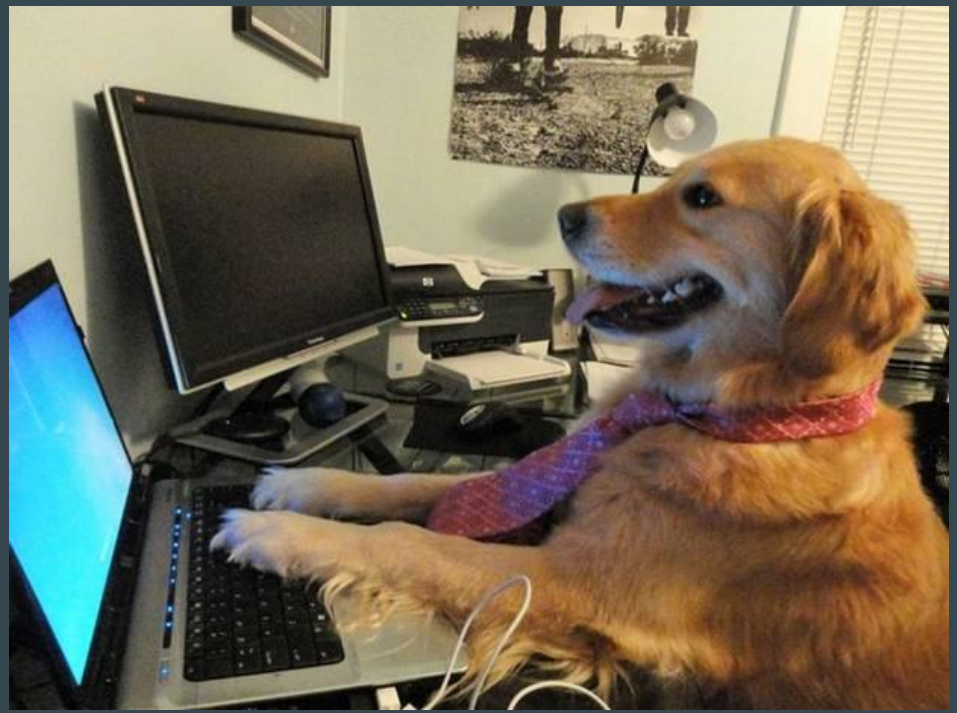
Introducing software development best practices for research in the behavioral and social sciences

...

Pablo Caceres
Nov - 2019
University of Wisconsin-Madison

Why should I care? I’m doing just fine!



Why should I care?

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

Introducing a set of **basic principles** in data management and software development may significantly improve research practices based 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

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, cloud computing, etc.). Roughly ~10 to 10 million of rows or TB of data

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....)
- **Python** and **R** for 2.5 years (definitely do this!)

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 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

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. Use this 8 simple principles to write better code
8. If doing ML: use a experiment tracking system
9. Test your code
10. Code reviews (when possible)
11. 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
|   |--\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
```

Use virtual environments

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

For Python **virtualenv** or **conda environment**

For R **conda environment**

If you're feeling adventurous, use **Docker containers** (probably overkill for small projects....)

Use version control systems

- Use **Git**, Mercurial, or SVN
- **Push frequently** after changes
- Use **branches** for experimentation
- 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
- Keep minimal **notes/logs** of major changes

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 file

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 with clean data

Use well-tested and properly supported software libraries

Why? Because popular libraries are:

- Thoroughly **tested** (less bugs)
- Optimized for **performance**
- Are **safer**
- Better **documentation**
- Better **support** (e.g., Stack Overflow, books, blogs)
- Better and more **tutorials**
- Are 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

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

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

Deep learning: Weights and Biases

ML in general: Mlflow or Sacred

Do code reviews (when possible)

Code reviews are critical for:

1. Finding **bugs**
2. Improve code **readability**
3. Improve **usability**
4. Improve code **performance**
5. Improve **documentation**
6. **Sharing progress** with your group/peers
7. Efficient **knowledge exchange**

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

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

Articles and References

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. Fehr, J., Heiland, J., Himpe, C., & Saak, J. (2016). Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software. arXiv preprint arXiv:1607.01191.
4. Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). Ten simple rules for reproducible computational research.
5. Hart, E. M., Barmby, P., LeBauer, D., Michonneau, F., Mount, S., Mulrooney, P., ... & Hollister, J. W. (2016). Ten simple rules for digital data storage.
6. 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).