Contents

1. Guidelines

You can find guidelines in the folder `./guidelines/` to help you to write final papers, final research projects, and presentations. Below you find some suggestions of tools that are commonly used in academia.

2. Latex and Beamer

From *LaTeX* webiste:

> LaTeX is a high-quality typesetting system; it includes features designed for technical and scientific writing. LaTeX is the de facto standard for the communication and publication of scientific documents.

If you are new to LaTeX, read this document to get started. There are many tutorials on that website to help you to learn LaTeX. Another good source is this tutorial by Mayya Komisarchik. You can also find some examples and templates in the folder `./latex-and-beamer/`.

Beamer is a latex document class that you can use to create professional-looking presentations. Here is a good tutorial to learn how to use it. You can find some examples and templates in the folder `./latex-and-beamer/`.

It is a good idea to invest on learning reference management to organize your bibliography. All those softwares for reference management either store or can export references in .bib format. That format is nicely integrated with LaTeX. Organizing references in .bib files make it much easier to cite previous work and include references.

3. Statistical Programming: R and Python

Here are some useful resources to learn R:

R style guide (Tideverse) (free) some tips about how to structure your code to make it more readable and reproducible.

R for Data Science (free) introductory-level book written by Hadley Wickham, Chief Scientist at RStudio, and an Adjunct Professor of Statistics at the University of Auckland, Stanford University, and Rice University. The website of the book contains examples and includes instructions to work with pipe and tydeverse. The latter is a popular package ecosystem for data analysis and dataviz.

Here are some useful resources to learn Python:

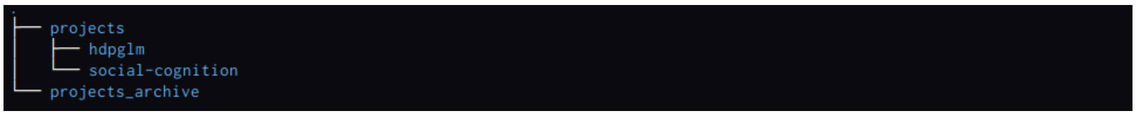
Python Style Guide (free) some tips about how to structure your code to make it more readable and reproducible.

Downey (2015) (free) good introductory-level book with the most important concepts and code examples in Python.

4. Project Management

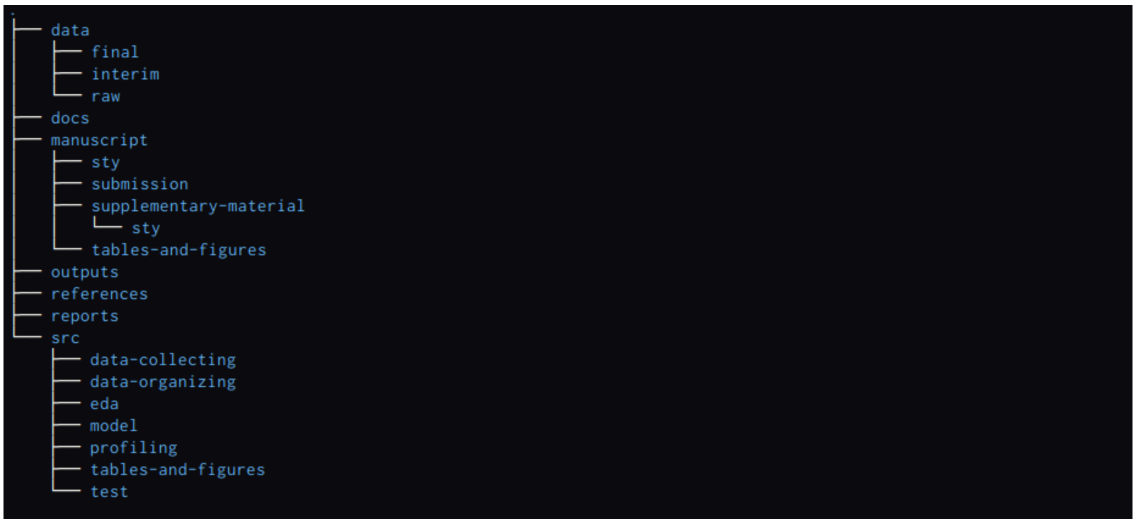
4.1. Project Folder Structure

It is a good idea to use a consistent folder structure to organize your research and software projects. That will make it easier to (1) navigate the folder structure to find files, (2) re-run and replicate the analysis later, (3) distribute your code and files for replication. I keep all past projects in a folder called `projects_archive` and all current projects in a folder named `projects`. Each project gets its own subfolder. For instance, if I am currently working on a projects `social-cognition` and another one called `hdpglm`, I will have these two subfolders in the `projects` folder:



```
.
├── projects
│   ├── hdpglm
│   └── social-cognition
└── projects_archive
```

Each project, by its turn, contains the same set of subfolders, although the file names may differ in each project. For instance, the folders and files inside the directory for the `social-cognition` project, or equivalently inside the directory `hdpglm`, will have the following basic structure:



```
.
├── data
│   ├── final
│   ├── interim
│   └── raw
├── docs
├── manuscript
│   ├── sty
│   ├── submission
│   ├── supplementary-material
│   │   └── sty
│   └── tables-and-figures
├── outputs
├── references
├── reports
└── src
    ├── data-collecting
    ├── data-organizing
    ├── eda
    ├── model
    ├── profiling
    ├── tables-and-figures
    └── test
```

The content of each folder is the following:

Folder	Subfolder	Content
data	./final	contains the final version of the data set used for the analysis
	./interim	contains intermediate data sets created to explore the data, merge ,etc. The raw data after some recoding and manipulation by scripts in the folder `.src/` go here
	./raw	contains the raw data sets as collected from external sources
docs		contains documents related to the project, such as grant proposal, codebooks of the data sets, etc.
manuscript	./	file with papers written for the project
	./sty	latex sty file I used to compile the papers
	./submission	contains versions of the papers submitted to journals. I create subfolders with the name of the journals to store the version submitted to that journal
	./supplementary-material	all supplementary material, including figures and latex files
	./tables-and-figures	all .tex tables and figures I used in the papers
outputs	./	files with the outputs of analyses. Usually, .Rout files and .log, but also files with simulated data sets
references	./	I keep all my .pdf files with papers and books in a single folder shared by all projects. In this folder, I keep files that are related to the project and that I use as reference, such as handouts, reports written by other scholars on the topic, etc.
reports	./	Reports I create for the project, such as exploratory data analysis
src		scripts will go in subfolders of this folder
	./data-collecting	scripts I used to collect the raw data
	./data-organizing	scripts I used for data wrangling and to create interim and final data sets from raw data
	./eda	scripts I used for exploratory data analysis, prior to final analysis
	./model	scripts that produce the final results and analysis.
	./tables-and-figures	I use this folder in case I create temporary tables and figures
	./test	A folder for running tests in the code

4.2. Git and Github

Version control softwares are very useful for production. Bryan (2018) contains some points about using version control. One of the most common version control software is Git. It can be used to keep track of changes you make to files and scripts. GitHub is a could platform that uses git. You can find a very nice introductory material here.

5. Integrated Development Environment (IDE)

You can write scripts as plain text for LaTeX, Python, R, and Stata in any text editor. You will be better off, however, if you used an Integrated Development Environment (IDE). IDE are softwares written to facilite code writing. The choce is personal, but here are some good options:

Here are the main classical options:

- Emacs: this is my personal choice. I use emacs for everything, including writing code in Latex, Python, R, academic papers, books, create presentations, organize and manage citation and references (with bibtex), communicate with HPC servers, organize my schedule using org-mode (a "package" to use with emacs), and so on. It is highly configurable, but dull at its out-of-the-box raw state, without configuration. It takes quite some time to learn and unleash all its potential, but it is worth the time invested.
- Vim: main emacs competitor. Good, but not as resourceful.

Some good out-of-the-box and readyand-easy-to-use IDEs are:

- RStudio: very easy to use and the barrier of entry is low. Good to write scripts in R.
- Sublime: good for many programming languages, including R and Python. Also easy to use.
- Atom: like Sublime
- Juputer Notebook: it runs on web browsers, and can be used with many programming languages. There is anice tutorial here.
- TexStudio: good for writing code in Latex.

6. References

- Bryan, J. (2018). Excuse Me, Do You Have A Moment To Talk About Version Control?, *The American Statistician*, 72(1), 20–27. (`https://peerj.com/preprints/3159v2/`)
- Downey, A. (2015). Think Python, : O'reilly Media, Inc.