Intro

Good research code

Patrick Mineault

Intro

Who is this lecture for?

Kyle 🚀🦈🛸🦕

@KyleMorgenstein

is my code fast? no. but is it well documented? no. but does it work? also no.

1:29 PM · Dec 14, 2020 · Twitter for iPhone

5.9K Retweets **1K** Quote Tweets **58.6K** Likes

Who is this lecture for?

- Most people who do coding-heavy research are not trained in CS or software engineering
- You're probably in this bucket
- Bad consequences:
  - You feel like you don't know what you're doing
  - Imposter syndrome
  - Low productivity
  - Bugs
  - You hate your code and you don't want to work on it
  - You never graduate
  - You have great sadness in your heart
- It doesn't have to be all bad!

My weird perspective

- Patrick Mineault, PhD in neuroscience
- (wildly underqualified) software engineer at Google
- Research scientist at Facebook on brain-computer interfaces
- Technical chair of Neuromatch Academy
- Independent researcher and technologist
- Occasionally taught CS

Regrets, I've had a few

- Mostly self-taught in programming
- Didn't study CS until very late
- Wasted months working with bad code of my own making
- Not a great coder, but better than in grad school
- I think you might be curious

Organization

- Assume that you know a little bit about Python, git and the command line
  - If you don't, that's ok! This is vertically integrated advice. Get inspired, follow more detailed tutorials after, and come back to this.
- 5 practical tips to better code
  - Concrete examples
  - 5-minute action items
  - Everybody leaves having learned an actionable thing
- Interrupt me and chat!
- But first, I will indulge in theory...

Open question

Q: What does coding look like in the brain?

Coding is very working-memory intensive

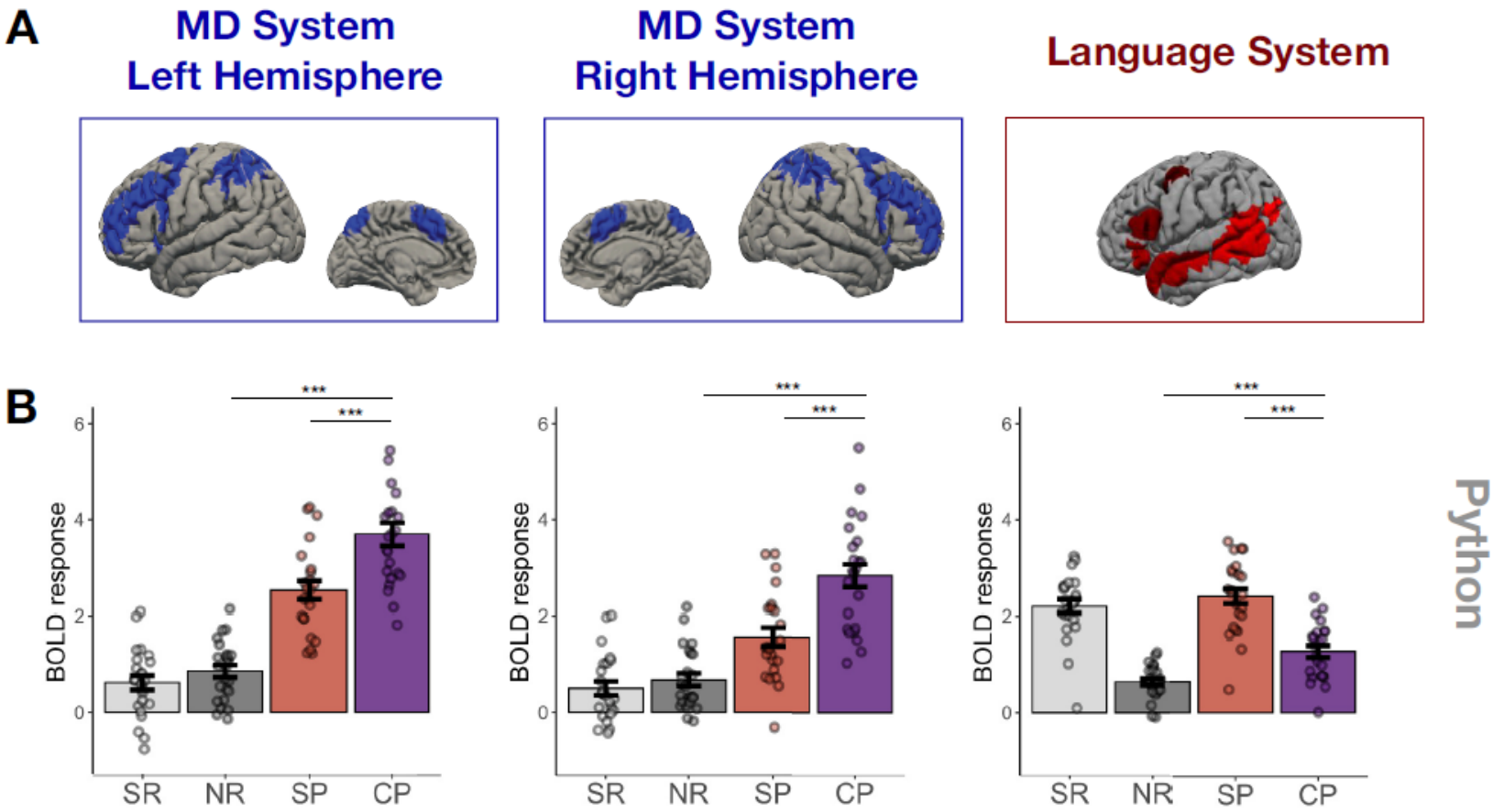


Figure 1: Code and working memory in the brain, Ivanova et al. (2020)

Coding is very working-memory intensive

- MD: Multiple-demand system
- CP: code programming
- SP: sentence programming
- SR: sentence reading
- NR: non-word reading

Consequence

You will get overloaded.

Principle 1: conserve your WM

- Reduce the cognitive load of understanding your code
- Simple is better than complex. Complex is better than complicated.

Research code is very LTM-intensive

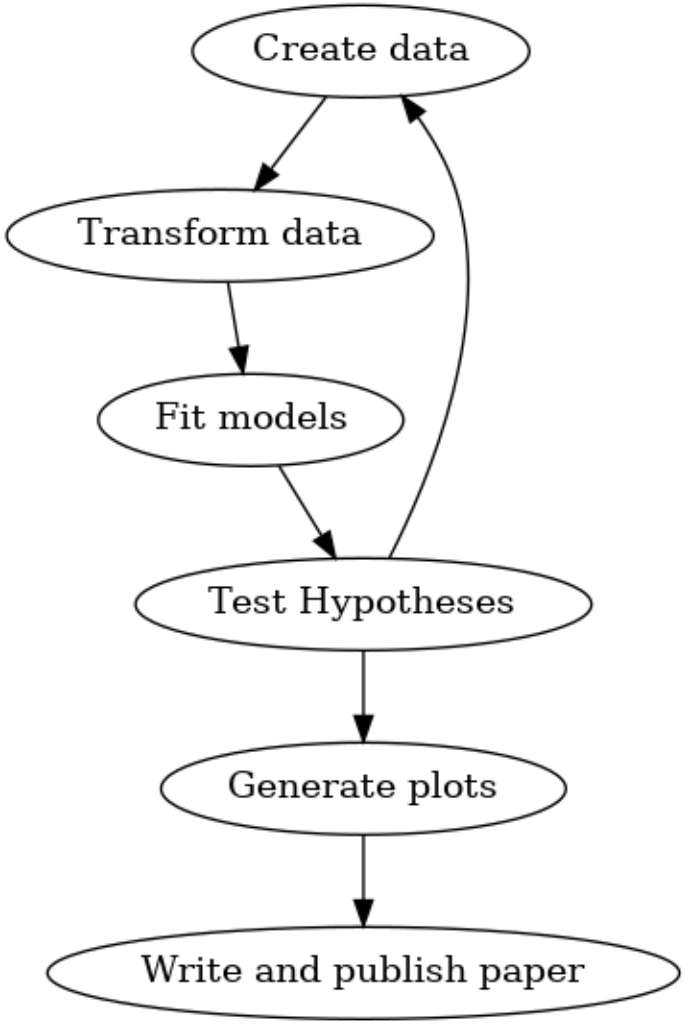


Figure 2: Theory

Research code is very LTM-intensive

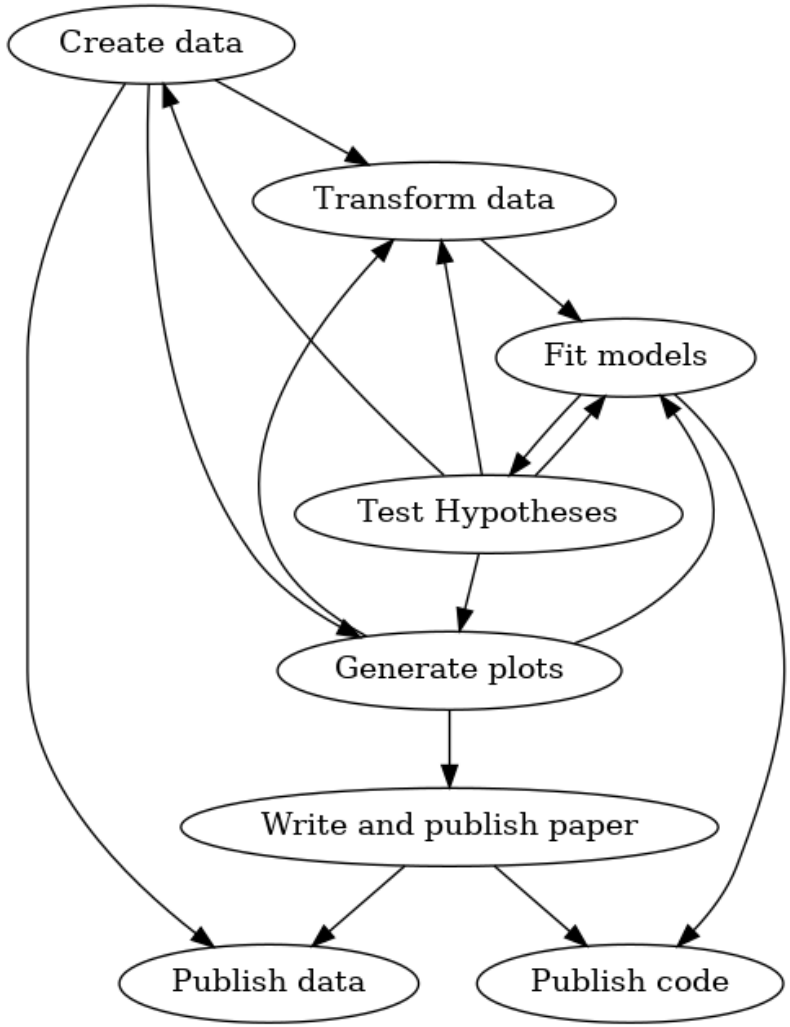


Figure 3: Practice

Research code

- Endpoint is unclear
- Correct can be hard to define
- Lots of exploration and dead ends
- Sometimes, there are manual steps involving human judgement
- You have to remember all the dead ends for the code to even make sense

Principle 2: write for your future self in mind

- Future you will have forgotten 90% of what you wrote
- Kernighan's Law - Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

Thesis

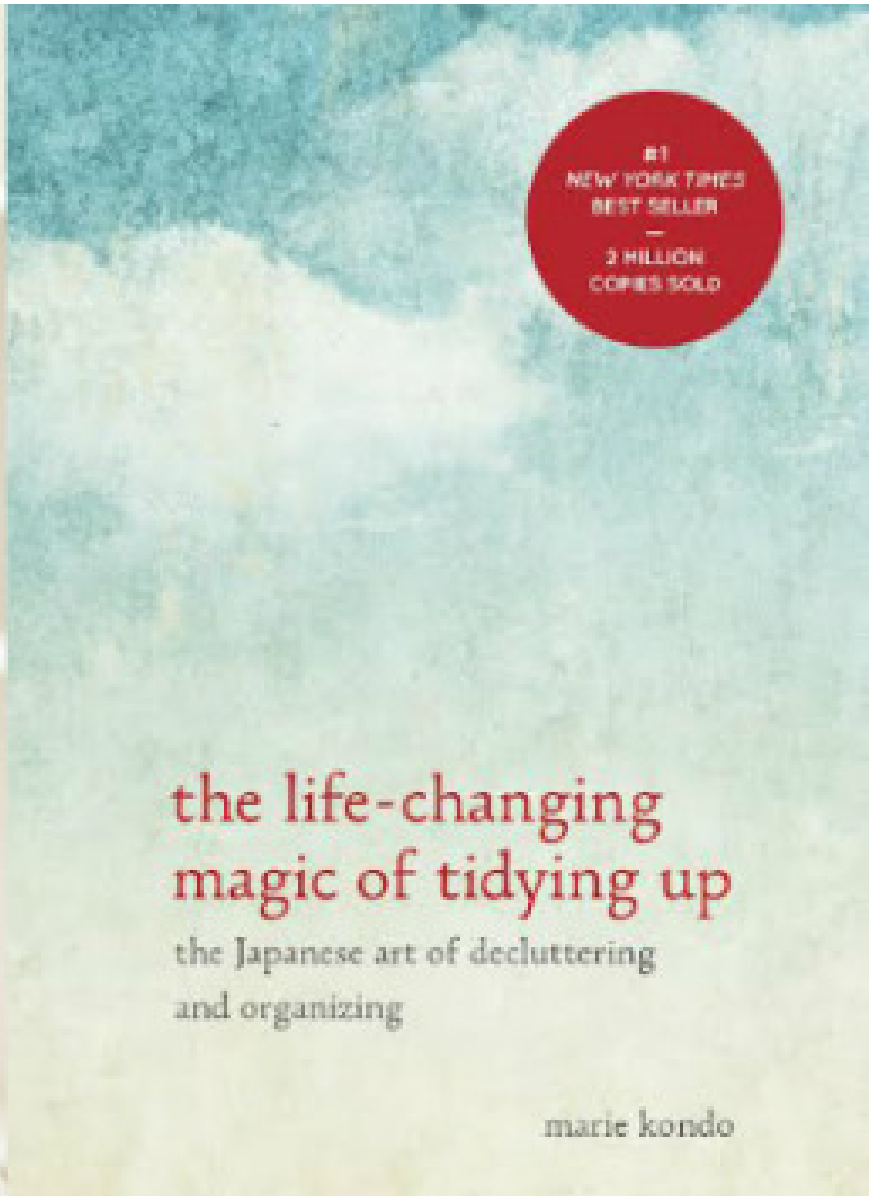
Writing good research code boils down to saving your memory - both working and long-term.

Practical Lessons

Disclaimer: this is a low-judgement zone

- It's ok to write garbage code when you're in a rush
- It's not ok to keep building more and more on top of garbage code
  - sure, there's the moral imperative to create replicable code to bring forward the shining light of science and truth...
  - and yes, people have messed up the world real bad by doing things fast and loose
  - but also, do you want to scrap 6 months of research because you forgot to transpose a matrix?
  - you will get bitten back
- Guidelines not rules

Lesson 1: keep things tidy



What needs to be tidy

- Project folder structure
- Code style
- Notebooks
- Scripts
- Prereq: Git & Github: if you're going to keep things clean, you will mess up and need a time machine.

Project folder structure

- Consensus: one repo = one project $\approx$ one paper
- Lots of templates around:
  - Turing Way
  - Research Software Engineering with Python
  - Data science cookiecutter
  - Shablona

Shablona

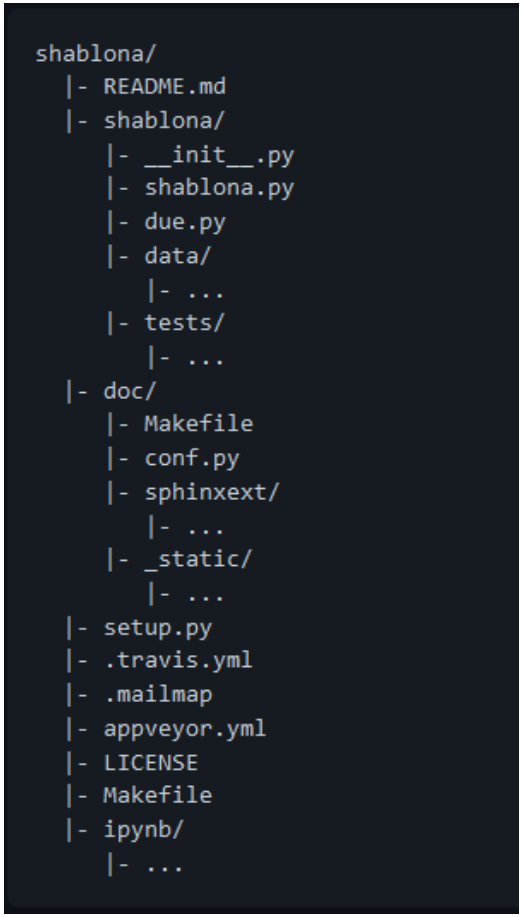


Figure 4: Shablona

Shablona

- Lightweight, good starter template
- Keeps docs, data, scripts and code tidy and in their own little box
- You can `import shablona` to access the code in the packages
- Use as a template to start a new project via big green button
- Or build it from scratch to understand the moving pieces
- **Important**: Is compatible with Python packaging. That means you can install locally with `pip install -e .`, and the code inside the special folder (placeholder: `shablona`) becomes a package `shablona`

Live demo

Packages, how do they work?

Whatever template you use, make sure it makes a local package for your code that you can `pip install`. That will make it easier to re-use your code in other places.

If you're curious, I wrote a long-form note on how packages really work.

Other conventions

- Notebooks → `Start with a Capital Letter.ipynb`
- Reusable functions and packages, etc. → `snake_case.py`
- Tests under `tests` folder

Organizing scripts

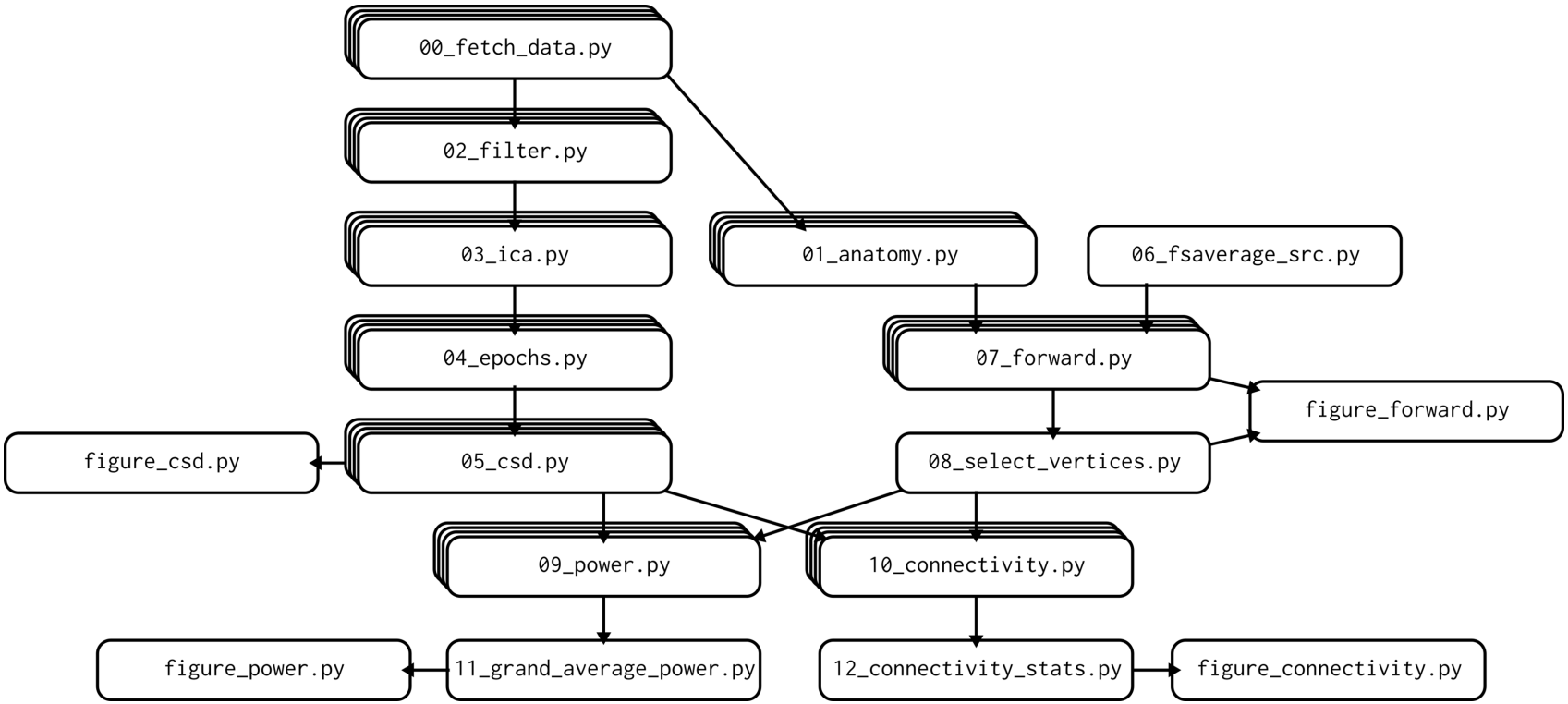


Figure 5: From Van Vliet (2020)

Organizing scripts

- Use filenames that indicate hierarchy, e.g. `00_fetch_data.py`
  - One issue: you can't `import` these scripts because you can't start a module name with a digit.
  - Start with an underscore, `_00_fetch_data.py`, or with a prefix, `step_00_fetch_data.py`, those are valid module names
- Figure code separate from processing steps code, e.g. `figure_csd.py`
- Use a master script to bind everything together
  - Plain Python
  - Bash files
  - Build tools: `doit`, `make`
  - Specialized tools like `nipype`

Code style

- Use a consistent style
- Python has a style guide - PEP8.
  - Indentation
  - Line length
  - Spaces
  - Variable names
  - imports
- Orgs like Google have their even more pedantic style guides.
- There are linters which will catch style issues
  - flake8
  - pylint
- Install them in VSCode

Docstrings

Numpy style or Google style.

```
def my_doubler(x):
    """Doubles x.

    Args:
        x: the number to double

    Returns:
        Twice x
    """
    return x * 2
```

IPython notebooks

If you use notebooks to develop software, you are probably using the wrong tool. – Yihui Xie

- Notebooks are hard to keep tidy because of nonlinear execution
- Restart and Run All is your friend
- If your notebook doesn't run top to bottom - it's not reproducible
- It's ok to write plotting code in a notebook, but don't write real functions.
- Import the code from your installable package (see `shablona` above)
- You can auto-reload your package code when it changes, makes development easier. In a cell:

```
%load_ext autoreload
%autoreload 2
```

Why does this matter?

You don't have to constantly ask yourself where stuff is, how you should do thing X, etc. and that allows you to focus on the stuff that matters.

Aside: day 3

Everything at Google is one giant monorepo with billions of lines of code. By ~day 3, it was time to go do a code. Everything is organized according to strict conventions, so it's not *that bad* to jump in.

Lesson 1

- Keep things tidy
- Free your W&LTM from having to remember where stuff is
- Your 5-minute exercise: use the `shablona` template for a project