Software Engineering for Data Scientists

For PRIVATE USE ONLY, not for open publication

Training 2022

Ian Ozsvald

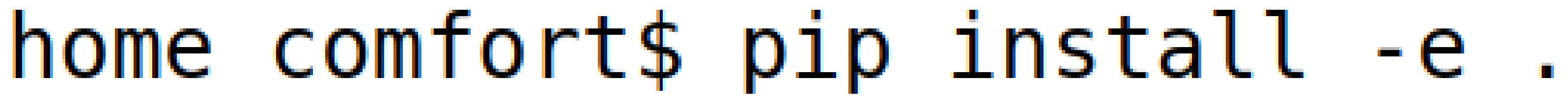
@IanOzsvald – ianozsvald.com

Review

- Pandera homework – did you find 4 errors? Tighter date?
- Give me some examples of how you could profitably use this? *Why* might it save you time and pain?
- Who can give me testing examples for any useful types of testing (real world examples please)

How does Python load modules?

- We’ll look at sys.path – this might be hard to follow
- Knowing even vaguely how it works means you can ask sensible questions if you hit problems
- Open “understanding_paths”



```
home comfort$ pip install -e .
```

Good folder layouts

- We had everything in 1 folder – what's bad about this?
- What's the worst you've seen?

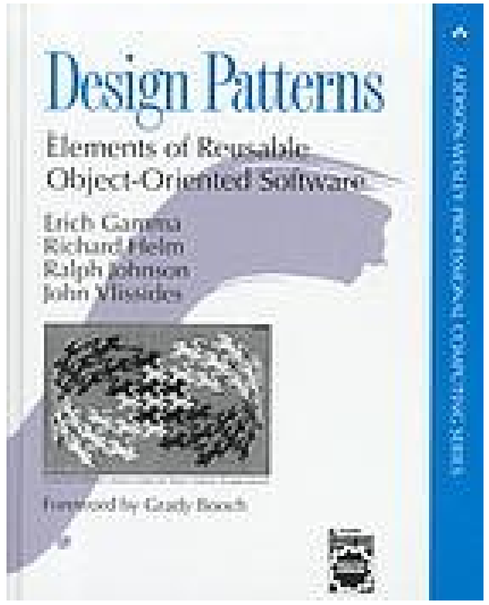
Standard code layouts

- I like “cookiecutter datascience”, I’ve trimmed my example
- Standard data folders (raw, processed), standard source & test folders
- Let’s look at “home_comfort”

Review

- We did “pip install -e .” inside “home_comfort”
- You should be able to run the data processor in src/
- Can you run pytest?
- We could run the Notebook too

Design Patterns (just briefly)



- "Gang of Four" classic Design Patterns
- "factory" to make new classes, "visitor" and MORE
- You can go pattern-happy (please don't, no need)

- "Program to an interface, not an implementation." (Gang of Four 1995:18)
- Composition over inheritance: "Favor 'object composition' over 'class inheritance'." (Gang of Four 1995:20)

Development patterns

- SOLID (mainly for OOP, I don’t generally recommend this for R&D)
- DRY – Don’t Repeat Yourself – gives easy wins
- YAGNI – You Ain’t Gonna Need It – avoid over-complication

Development Patterns

- DRY & YAGNI and sensible
- KISS – used to design jet aircraft...
- Dependency Inversion (SOLI**D**) is useful, see our example – make a subunit do a job regardless of implementation details (process_data & partial)

Classes or Functions?

- Classes are the gospel truth if you come from e.g. Java
- Not true in Python (or many languages) – just another tool
- Useful for *data hiding*, *state* and *encapsulation*

Prefer functions

- Prefer simple functions that do 1 clear “thing”
- Combine behaviour, pass around state – easy to test
- Refactor minimally to use classes where you need state or encapsulation (95% good advice)

Pandas weirdness

- Find “pandas_weirdness.ipynb”
- Think about where you’ve had Pandas problems

Your big wins

- Push data checks back to source of issues – catch early
- Use automation (e.g. flake8) to spot common mistakes
- Keep asking “what’s the utility in testing/review/dev/…”
- Unit tests & TDD build confidence – start with 1 test

How do you take this back to work?

- What did you “do wrong” before? Consider...
- black/nbqa/flake8, pytest and unittests, pandera, writing a library of shared code, standard folder layout, code reviews, minimal Pandas feature set
- Spend 5 mins – what are your top pragmatic wins?

How do you take this back to work?

- "I didn't do X and it cost me Y" – what's X & Y?
- "I'm going to try Z next week…" - what's Z?
- What holds you *back* from trying your best new wins?
- What other issues do you have that we've not addressed?

Almost there

- Newsletter – NotANumber (“buttondown notanumber”)
- PyDataLondon – please join meetup+conference
- PyDataUK slack – friendly for all UK meetups
- Higher Performance Python? Pandas class?

Thanks for having me

Appendix