Code Reviews

From: Successfully Delivering Data Science Projects 2019

For PRIVATE USE ONLY, not for open publication

January 2019

Ian Ozsvald

@IanOzsvald – ianozsvald.com

Table of Content

Table of Contents

High level – we're aiming for

- Trusted code that works correctly
    - Code that 'feels correct' and is 'easy to read'
    - Code that is 'well structured'
- Code that's easy to read by other colleagues
    - It doesn't 'feel complicated'
    - It doesn't feel as though it could be refactored into small components
- Documents that enable reproducibility
- No accidental mistakes
- Sharing of techniques between colleagues to spread new knowledge
- A consistent approach across the team that reduces surprise when a team member reads your code

Time investment

- Some time invested beats no time invested
    - We write better code if we know someone else will review it
- Regular code reviews pay off with higher quality across all projects
    - We can focus on more interesting problems that time-sinking debugging issues
- Iteratively deepen - bite off something you can tackle in e.g. 30 minutes or half a day and use this as a basis for subsequent reviews
- For the first review - scan the whole Notebook/module, try to get a feel for what is going on. If it isn't clear - this might flag fundamental structure and documentation problems
- Note that hard to read code takes a lot of time to review - this probably hides bugs and ultimately will slow down the team
- Sometimes identify a section of a the code and do a deep-dive, perhaps suggesting possible replacement code that could be tested by the author

Documentation

- Does sufficient documentation exist for someone else to run the program?
    - Does it include how to setup the project on a new machine?
    - Does it should sample output to help when running it?
- Are suitable diagrams included to explain e.g. complex UI event driven relationships, data and process dependencies

Tests

- Do we see “assert” statements that automatically check assumptions?
- Have “test_” functions (or classes) been created for use by py.test?
- Do the tests have sufficient documentation that if it failed, you’d know why and what you might want to fix?

Python code

- Have you read PEP8 for the Python coding standards? Check for naming standards
- All imports should be at the top of the file (not scattered throughout)
- Avoid shadowing - don’t overwrite a function name (e.g. making a local print function that shadows the Python print function)
- Functions should be less than 1 page long so they’re easily read on screen
- Functions should have a sensible name and one line of useful documentation
- Variables should also be reasonably self documenting and short
- Filenames shouldn’t be hardcoded but should be in a CONSTANT (in Python we run a constant value in all caps e.g. “INPUT_FILENAME=’data.csv’”, preferably at the top of the file for easy reference
- Passwords & credentials should never be stored within the code, there are various solutions and using environment variables is common
- Check for use of Boolean and/or/not operators (commonly) vs the rarely used bitwise equivalents &/|/~, beware anyone with a C background mistakenly using bitwise operations as you’ll get subtle hard-to-debug errors in your code
- Check for mistaken use of a default list (e.g. “def fn(mylist=[]):”) in a function’s argument list (check for the use of “df fn(mylist=None):” instead)
- Modules should probably be less than 150 lines long and if they’re over 300 lines - a very strong justification should be in place (as the knock-on cost in interpretability increases as module length increases)

Notebooks

- Does the top of the Notebook contain some documentation that sets the scene? This might include an executive summary to tell you what you’ll learn by following the Notebook
- Are data sources clearly described?
- Do you believe that this Notebook, if run again, will use the same data to produce the same results? If not – is that a problem?
- Was it run from the top to the bottom? If not, are you sure it is worth reviewing? What if it cannot actually generate the current result if it were run from top to bottom?

- Does it contain a watermark at the top for future reproducibility?
- Are diagrams usefully employed to visually communicate results?
- Has the analyst left interpretations in markdown that are easy to understand?

Refactoring

- Near-duplicated lines (or functions) are very hard to read - avoid duplicated lines if they can be simplified
    - Consider writing a function that simplifies the repeated code
    - Replace magic numbers (e.g. 1, 100, -1) with constants that enhance readability (e.g. True, MAX_SIZE, UNKNOWN)
- Check for overly complex lines, perhaps copied from StackOverflow answers and duplicated, and try to simplify them e.g. replacing a mix of older numpy/pandas code from older pandas answers to more modern pandas-only solutions
- Beware solutions that are clearly copied from StackOverflow for rapidly evolving libraries like pandas – those solutions might be obsolete. Flag these for a sanity check and possible replacement
- Identify functions which have an external effect (e.g. they modify a database or write a file) and try to isolate them so that the underlying logic can be tested separately from the external effect

More advanced Python – introduce where sensible

- Iterator protocol
- functools, collections and itertools
- Pandas pipes
- Pandas apply which may open the door to Dask's parallelised apply

Giving feedback

- Be supportive and give constructive criticism
- On BitBucket/GitHub use Pull Requests and leave feedback including Tasks that can be checked off
- Agree which tasks are minor and which are blockers

Checklist

1. Do you have instructions for running the code and building all necessary data?

2. Do Notebooks look like they were checked-in in a freshly-run state (i.e. In[1] to In[n] in sequence)

3. Does it look reproducible?

4. Which areas look like they need attention? Do you have enough time or do you need to focus on the worst offenders?

5. Does any code look suspicious? Can it be simplified? Are you confident that it "probably works" or is it *really suspicious*?

6. Is there a clear Introduction and Conclusion that give it structure?

7. Next – find a section to focus on and "go deeper", flagging any other areas for future review, and iterate