Software_Engineering_Practices_2

July 1, 2020

W'll learn about the following practices of software engineering and how they apply in data science.

- Testing
- Logging
- Code reviews

1 Testing

Testing your code is essential before deployment. It helps you catch errors and faulty conclusions before they make any major impact. Today, employers are looking for data scientists with the skills to properly prepare their code for an industry setting, which includes testing their code.

2 Testing And Data Science

- Problems that could occur in data science aren't always easily detectable; you might have values being encoded incorrectly, features being used inappropriately, unexpected data breaking assumptions
- To catch these errors, you have to check for the quality and accuracy of your analysis in addition to the quality of your code. Proper testing is necessary to avoid unexpected surprises and have confidence in your results.
- **TEST DRIVEN DEVELOPMENT**: a development process where you write tests for tasks before you even write the code to implement those tasks.
- **UNIT TEST**: a type of test that covers a "unit" of code, usually a single function, independently from the rest of the program.

Resources:

- Four Ways Data Science Goes Wrong and How Test Driven Data Analysis Can Help: Blog Post
- Ned Batchelder: Getting Started Testing: Slide Deck and Presentation Video

3 Unit Tests

We want to test our functions in a way that is repeatable and automated. Ideally, we'd run a test program that runs all our unit tests and cleanly lets us know which ones failed and which ones succeeded. Fortunately, there are great tools available in Python that we can use to create effective unit tests!

Unit Test Advantages and Disadvantages The advantage of unit tests is that they are isolated from the rest of your program, and thus, no dependencies are involved. They don't require access to databases, APIs, or other external sources of information. However, passing unit tests isn't always enough to prove that our program is working successfully. To show that all the parts of our program work with each other properly, communicating and transferring data between them correctly, we use integration tests. In this lesson, we'll focus on unit tests; however, when you start building larger programs, you will want to use integration tests as well.

You can read about integration testing and how integration tests relate to unit tests here. That article contains other very useful links as well.

4 Unit Testing Tools

To install `pytest`, run `pip install -U pytest` in your terminal. You can see more information on getting started here.

- Create a test file starting with `test_`
- Define unit test functions that start with `test_` inside the test file
- Enter `pytest` into your terminal in the directory of your test file and it will detect these tests for you!

`test_` is the default - if you wish to change this, you can learn how to in this `pytest` configuration

In the test output, periods represent successful unit tests and F's represent failed unit tests. Since all you see is what test functions failed, it's wise to have only one `assert` statement per test. Otherwise, you wouldn't know exactly how many tests failed, and which tests failed.

Your tests won't be stopped by failed `assert` statements, but it will stop if you have syntax errors.

5 Test Driven Development and Data Science

- **TEST DRIVEN DEVELOPMENT**: writing tests before you write the code that's being tested. Your test would fail at first, and you'll know you've finished implementing a task when this test passes.
- Tests can check for all the different scenarios and edge cases you can think of, before even starting to write your function. This way, when you do start implementing your function, you can run this test to get immediate feedback on whether it works or not in all the ways you can think of, as you tweak your function.

- When refactoring or adding to your code, tests help you rest assured that the rest of your code didn't break while you were making those changes. Tests also helps ensure that your function behavior is repeatable, regardless of external parameters, such as hardware and time.

Test driven development for data science is relatively new and has a lot of experimentation and breakthroughs appearing, which you can learn more about in the resources below.

- Data Science TDD
- TDD for Data Science
- TDD is Essential for Good Data Science Here's Why
- Testing Your Code (general python TDD)

6 Logging

Logging is valuable for understanding the events that occur while running your program. For example, if you run your model over night and see that it's producing ridiculous results the next day, log messages can really help you understand more about the context in which this occurred.

7 Code Reviews

Code reviews benefit everyone in a team to promote best programming practices and prepare code for production. Let's go over what to look for in a code review and some tips on how to conduct one.

- Code Review
- Code Review Best Practices

7.1 Questions to Ask Yourself When Conducting a Code Review

First, let's look over some of the questions we may ask ourselves while reviewing code. These are simply from the concepts we've covered in these last two lessons!

Is the code clean and modular?

- Can I understand the code easily?
- Does it use meaningful names and whitespace?
- Is there duplicated code?
- Can you provide another layer of abstraction?
- Is each function and module necessary?
- Is each function or module too long?

Is the code efficient?

- Are there loops or other steps we can vectorize?
- Can we use better data structures to optimize any steps?
- Can we shorten the number of calculations needed for any steps?
- Can we use generators or multiprocessing to optimize any steps?

Is documentation effective?

- Are in-line comments concise and meaningful?
- Is there complex code that's missing documentation?
- Do function use effective docstrings?
- Is the necessary project documentation provided?

Is the code well tested?

- Does the code high test coverage?
- Do tests check for interesting cases?
- Are the tests readable?
- Can the tests be made more efficient?

Is the logging effective?

- Are log messages clear, concise, and professional?
- Do they include all relevant and useful information?
- Do they use the appropriate logging level?

Tip: Use a code linter

This isn't really a tip for code review, but can save you lots of time from code review! Using a Python code linter like pylint can automatically check for coding standards and PEP 8 guidelines for you! It's also a good idea to agree on a style guide as a team to handle disagreements on code style, whether that's an existing style guide or one you create together incrementally as a team.