Software Engineering Practices – Part 2

In part 2 of software engineering practices, you'll learn about the following practices of software engineering and how they apply in data science.

* Testing
* Logging
* Code reviews

# 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.

## 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, or 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 (TDD): A development process in which 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.

## 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, or 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 (TDD): A development process in which 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.

# Unit testing tools

To install pytest, run pip install -U pytest in your terminal. You can see more information on getting started [here](https://docs.pytest.org/en/latest/getting-started.html).

* 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 detects these tests for you.

test\_ is the default; if you wish to change this, you can learn how in this [pytest configuration](https://docs.pytest.org/en/latest/customize.html).

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

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

## Test-driven development and data science

* Test-driven development: Writing tests before you write the code that’s being tested. Your test fails at first, and you know you’ve finished implementing a task when the test passes.
* Tests can check for different scenarios and edge cases before you even start to write your function. When start implementing your function, you can run the test to get immediate feedback on whether it works or not 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 is experiencing a lot of experimentation and breakthroughs.

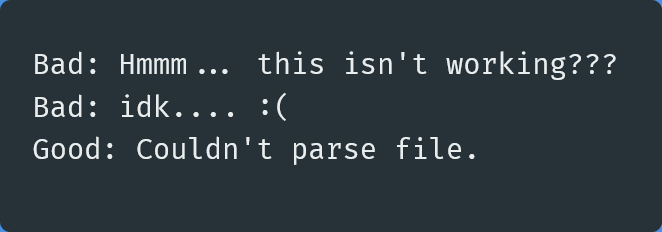
# Logging

Logging is valuable for understanding the events that occur while running your program. For example, if you run your model overnight and the results the following morning are not what you expect, log messages can help you understand more about the context in those results occurred. Let's learn about the qualities that make a log message effective.

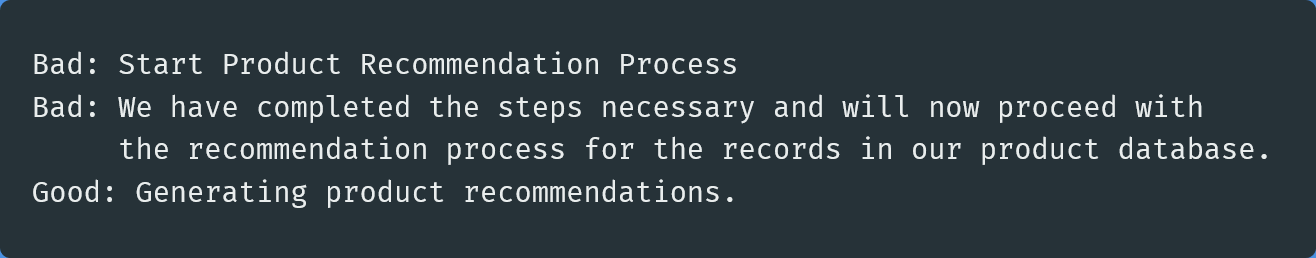
## Log messages

Logging is the process of recording messages to describe events that have occurred while running your software. Let's take a look at a few examples, and learn tips for writing good log messages.

### Tip: Be professional and clear



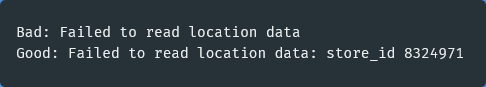
### Tip: Be concise and use normal capitalization



### Tip: Choose the appropriate level for logging

Debug: Use this level for anything that happens in the program. Error: Use this level to record any error that occurs. Info: Use this level to record all actions that are user driven or system specific, such as regularly scheduled operations.

### Tip: Provide any useful information



# 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.

## Questions to ask yourself when conducting a code review

First, let's look over some of the questions we might ask ourselves while reviewing code. These are drawn 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 I 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 I can vectorize?
* Can I use better data structures to optimize any steps?
* Can I shorten the number of calculations needed for any steps?
* Can I use generators or multiprocessing to optimize any steps?

### Is the documentation effective?

* Are inline comments concise and meaningful?
* Is there complex code that's missing documentation?
* Do functions 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?

## Tips for conducting a code review

Now that we know what we're looking for, let's go over some tips on how to actually write your code review. When your co-worker finishes up some code that they want to merge to the team's code base, they might send it to you for review. You provide feedback and suggestions, and then they may make changes and send it back to you. When you are happy with the code, you approve it and it gets merged to the team's code base.

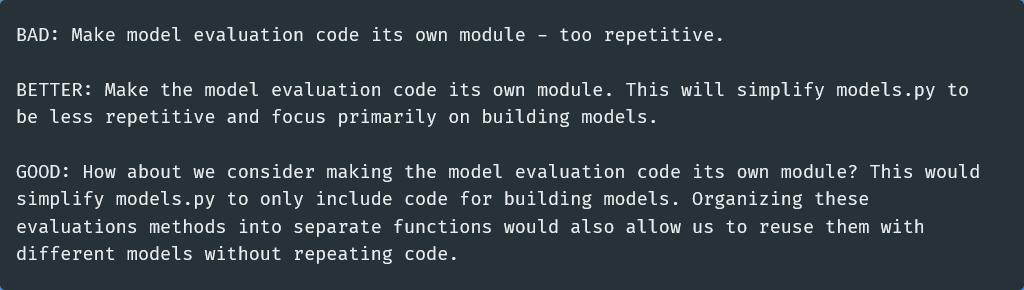
As you may have noticed, with code reviews you are now dealing with people, not just computers. So it's important to be thoughtful of their ideas and efforts. You are in a team and there will be differences in preferences. The goal of code review isn't to make all code follow your personal preferences, but to ensure it meets a standard of quality for the whole team.

### Tip: Use a code linter

This isn't really a tip for code review, but it can save you lots of time in a code review. Using a Python code linter like [pylint](https://www.pylint.org/) 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.

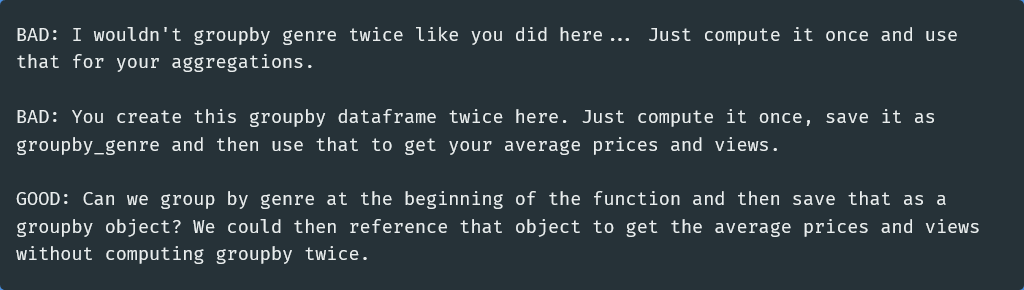
### Tip: Explain issues and make suggestions

Rather than commanding people to change their code a specific way because it's better, it will go a long way to explain to them the consequences of the current code and suggest changes to improve it. They will be much more receptive to your feedback if they understand your thought process and are accepting recommendations, rather than following commands. They also may have done it a certain way intentionally, and framing it as a suggestion promotes a constructive discussion, rather than opposition.



### Tip: Keep your comments objective

Try to avoid using the words "I" and "you" in your comments. You want to avoid comments that sound personal to bring the attention of the review to the code and not to themselves.



### Tip: Provide code examples

When providing a code review, you can save the author time and make it easy for them to act on your feedback by writing out your code suggestions. This shows you are willing to spend some extra time to review their code and help them out. It can also just be much quicker for you to demonstrate concepts through code rather than explanations.

Let's say you were reviewing code that included the following lines:

