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?

Tips for Conducting a Code Review

Now that we know what we are looking for, let's go over some tips on how to actually write your code review. When your coworker 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 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 a standard of quality for the whole team.

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.

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.

```
BAD: Make model evaluation code its own module - too repetitive.

BETTER: Make the model evaluation code its own module. This will simplify models.py t
o be less repetitive and focus primarily on building models.

GOOD: How about we consider making the model evaluation code its own module? This wou
ld simplify models.py to only include code for building models. Organizing these eval
uations methods into separate functions would also allow us to reuse them with differ
ent models without repeating code.
```

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.

```
BAD: I wouldn't groupby genre twice like you did here... Just compute it once and use
that for your aggregations.

BAD: You create this groupby dataframe twice here. Just compute it once, save it as g
roupby_genre and then use that to get your average prices and views.
```

```
GOOD: Can we group by genre at the beginning of the function and then save that as a
groupby object? We could then reference that object to get the average prices and vie
ws without computing groupby twice.
```

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:

```
first_names = []
last_names = []

for name in enumerate(df.name):
    first, last = name.split(' ')
    first_names.append(first)
    last_names.append(last)

df['first_name'] = first_names
df['last_names'] = last_names
BAD: You can do this all in one step by using the pandas str.split method.
GOOD: We can actually simplify this step to the line below using the pandas str.split
method. Found this on this stack overflow post: https://stackoverflow.com/questions/1
4745022/how-to-split-a-column-into-two-columns
df['first_name'], df['last_name'] = df['name'].str.split(' ', 1).str
```