Day 4 - Guidelines

Today is mostly about Software Engineering concepts. You will learn how to refactor existing code to make it simpler, cleaner, thus easier to test.

1. Refactor AWS credentials checking

`Suggested time: 20-30 min`

In `src.aws`, both functions `upload_data_to_s3` and `download_data_from_s3` guard against environment variables not being set for AWS configuration.

A few things to note:

- This doesn't follow the DRY principle
- Is checking AWS credentials these functions' responsibilities? Most probably not.

By reducing coupling, repetitions, etc... not only do we ease the readability of our code base, but we also get the benefits of simplifying the testing process. All of this make our code base easier to maintain on the long run.

We will solve both of these issues by:

1. Factorizing the duplicated code in a function (technically, it's more of a procedure but Python doesn't make a difference between those)
2. Completely extract this logic out of the function using a decorator, to respect the Separation of Concerns principle.

1.1 Refactor with a function

This one is really straightforward:

- Create a function `check_aws_config` that will execute the duplicated code
- Call this function from `upload_data_to_s3` and `download_data_from_s3`

```
# src/aws.py

def check_aws_config():
    # TODO:
    # 1. Check for any missing AWS variable, and raise an error in case
    #    something is missing.
    # 2. Call this function from `upload_...` and `download_...`
```

1.2 Convert the function into a decorator

Knowing out to refactor code using the decorator pattern is an important Python skill. The concept can feel difficult to grasp... Take your time if you're not yet familiar with decorators, and **don't forget to look at the slides to get some hints.**

- Transform the `check_aws_config` to a decorator
- Then decorate `upload_data_to_s3` and `download_data_from_s3`

> 🌟 **Bonus**
>
> Can you **bypass the decorator when testing the functions**, and test the decorator in isolation?
>
> If you have time, think about it:
> 1. Is it doable without complication?
> 2. Is it something we want?

```
# src/aws.py

def check_aws_config(???):
    # TODO: refactor this function so it can be used as a decorator


@check_aws_config
def upload_data_to_s3(...):
    ...

@check_aws_config
def download_data_from_s3(...):
    ...
```

1.3 Run the test suite

Check for any regression, by running:

```
pytest -v tests/
```

Are all the tests still green? **They should be!**

2. Refactoring the training system

`Suggested time: 60-90 min`

⚠️ **WARNING**

This is the ultimate exercise of the workshop, as such it is probably the most demanding and certainly the most self-guided. **We strongly encourage you to work in groups of 2 to 4 people**, to discuss, criticize and build a solution incrementally.

Beside end-to-end tests that we wrote the first day, the core function of the training system hasn't been tested extensively, in particular in term of integration and unit tests. Since it is an important part of the system, **it should be tested**. Take a look at the module and try to think about how you would test such a module...

> The ultimate goal of this exercise is to write tests for the `train.py` module. However, writing unit tests *e.g.* for the `train` function would be a chore! This function is doing a lot of different things, and is dealing with different level of abstractions.

Before writing integration and/or unit tests dedicated to the training module, we need to improve the implementation, and most importantly implement a better Separation of Concerns.

Guidelines

1. Refactor the original code to make the testing process simpler
2. Write some tests to check your training system for correctness. Some suggestions:
   - Running an evaluation process on a fitted model should not raise any error
   - The `cross_validate` function must return an estimator, *i.e.* an object that has a `fit` and a `predict` methods

- The `hyperopt` function must be called only when we the train system is run under `optimization` mode
- The logger (used as an experiment tracker) must be used only when required, and can be turned off on demand
- ...

Some tips

Try to write tests with the current implementation
You will most likely run into road blocks: these can be insightful to find out where you should decouple the program.

Code splitting / separation of concern
Most functions are too big and do too many things. You should split those into simpler units that are easier to test independently. Take a moment to think what are the core responsibilities of the training system, then split those responsibilities into individual functions and/or classes.

Re-run tests frequently
Don't forget you wrote an end-to-end test for the training system on the first day of the workshop. This test will be very valuable now that you're refactoring this part. Also, as you split code into simpler units, you should start writing tests for these parts. You don't have to wait the end of the refactor to start writing tests.

> Just FYI: some people even write tests before the actual application code 😉

Final words

You've made it. Congrats 🚀

Don't forget, these things take time and experience. You will learn much more in the field, so go out and experiment!