Contents

[Q1. Refactoring 2](#_Toc171682478)

[Code 2](#_Toc171682479)

[Screenshot 2](#_Toc171682480)

[Q2. Installing pytest 3](#_Toc171682481)

[Screenshot 3](#_Toc171682482)

[Q3. Writing first unit test 4](#_Toc171682483)

[Code 4](#_Toc171682484)

[Screenshot 4](#_Toc171682485)

[Q4. Mocking S3 with Localstack 5](#_Toc171682486)

[Screenshot 5](#_Toc171682487)

[Q5. Creating test data 6](#_Toc171682488)

[Code 6](#_Toc171682489)

[Screenshot 6](#_Toc171682490)

[Q6. Finish the integration test 6](#_Toc171682491)

[Code 7](#_Toc171682492)

[Screenshot 7](#_Toc171682493)

# Q1. Refactoring

Before we can start covering our code with tests, we need to refactor it. We'll start by getting rid of all the global variables.

* Let's create a function main with two parameters: year and month.
* Move all the code (except read\_data) inside main
* Make categorical a parameter for read\_data and pass it inside main

Now we need to create the "main" block from which we'll invoke the main function. How does the if statement that we use for this looks like?

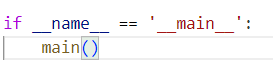
Hint: after refactoring, check that the code still works. Just run it e.g. for March 2023 and see if it finishes successfully.

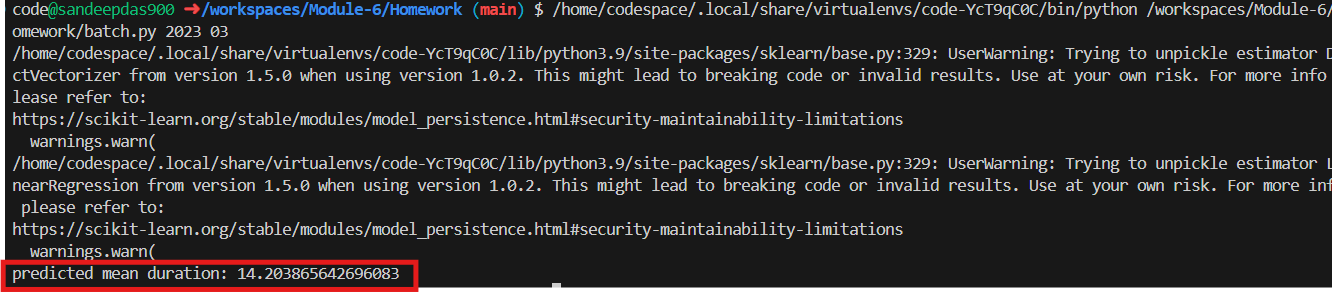
To make it easier to run it, you can write results to your local filesystem. E.g. here:

output\_file = f'taxi\_type=yellow\_year={year:04d}\_month={month:02d}.parquet'

Code - [Module-6/Homework/batch.py at main · sandeepdas900/Module-6 (github.com)](https://github.com/sandeepdas900/Module-6/blob/main/Homework/batch.py)

### Screenshot





# Q2. Installing pytest

Now we need to install pytest:

pipenv install --dev pytest

Next, create a folder tests and create two files. One will be the file with tests. We can name it test\_batch.py.

What should be the other file?

Hint: to be able to test batch.py, we need to be able to import it. Without this other file, we won't be able to do it.

### Screenshot



# Q3. Writing first unit test

Now let's cover our code with unit tests.

We'll start with the pre-processing logic inside read\_data.

It's difficult to test right now because first reads the file and then performs some transformations. We need to split this code into two parts: reading (I/O) and transformation.

So let's create a function prepare\_data that takes in a dataframe (and some other parameters too) and applies some transformation to it.

(That's basically the entire read\_data function after reading the parquet file)

Now create a test and use this as input:

data = [

(None, None, dt(1, 1), dt(1, 10)),

(1, 1, dt(1, 2), dt(1, 10)),

(1, None, dt(1, 2, 0), dt(1, 2, 59)),

(3, 4, dt(1, 2, 0), dt(2, 2, 1)),

]

columns = ['PULocationID', 'DOLocationID', 'tpep\_pickup\_datetime', 'tpep\_dropoff\_datetime']

df = pd.DataFrame(data, columns=columns)

Where dt is a helper function:

from datetime import datetime

def dt(hour, minute, second=0):

return datetime(2023, 1, 1, hour, minute, second)

Define the expected output and use the assert to make sure that the actual dataframe matches the expected one.

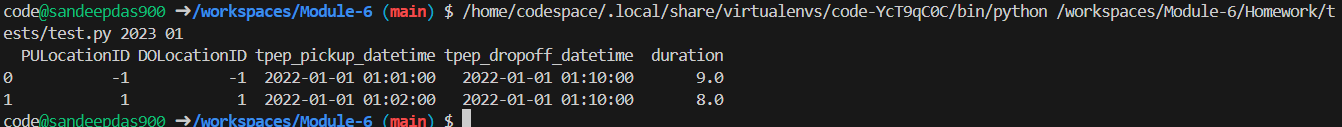
Tip: When you compare two Pandas DataFrames, the result is also a DataFrame. The same is true for Pandas Series. Also, a DataFrame could be turned into a list of dictionaries.

How many rows should be there in the expected dataframe?

**Answer – 2**

Code - [Module-6/Homework/tests/test.py at main · sandeepdas900/Module-6 (github.com)](https://github.com/sandeepdas900/Module-6/blob/main/Homework/tests/test.py)

### Screenshot



# Q4. Mocking S3 with Localstack

Now let's prepare for an integration test. In our script, we write data to S3. So we'll use Localstack to mimic S3.

First, let's run Localstack with Docker compose. Let's create a docker-compose.yaml file with just one service: localstack. Inside localstack, we're only interested in running S3.

Start the service and test it by creating a bucket where we'll keep the output. Let's call it "nyc-duration".

With AWS CLI, this is how we create a bucket:

aws s3 mb s3://nyc-duration

Then we need to check that the bucket was successfully created. With AWS, this is how we typically do it:

aws s3 ls

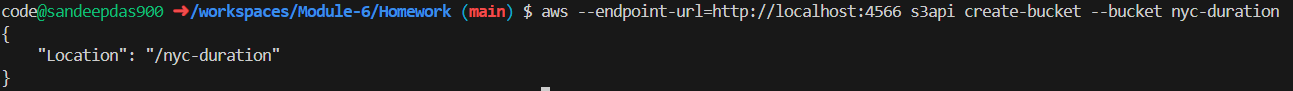
In both cases we should adjust commands for localstack. Which option do we need to use for such purposes?

**- --endpoint-url**

* --profile
* --region
* --version

**Answer --endpoint-url**

### Screenshot





# Q5. Creating test data

Now let's create integration\_test.py

We'll use the dataframe we created in Q3 (the dataframe for the unit test) and save it to S3. You don't need to do anything else: just create a dataframe and save it.

We will pretend that this is data for January 2023.

Run the integration\_test.py script. After that, use AWS CLI to verify that the file was created.

Use this snipped for saving the file:

df\_input.to\_parquet(

input\_file,

engine='pyarrow',

compression=None,

index=False,

storage\_options=options

)

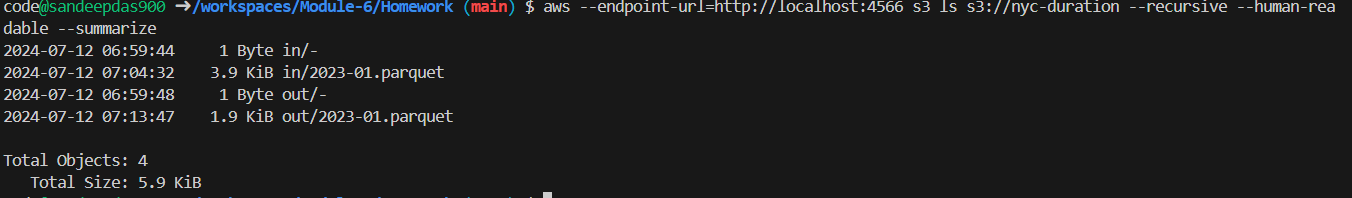
What's the size of the file?

* 3620
* 23620
* 43620
* 63620

**Answer – 3620**

Code - [Module-6/Homework/integration\_test.py at main · sandeepdas900/Module-6 (github.com)](https://github.com/sandeepdas900/Module-6/blob/main/Homework/integration_test.py)

### Screenshot



# Q6. Finish the integration test

We can read from our localstack s3, but we also need to write to it.

Create a function save\_data which works similarly to read\_data, but we use it for saving a dataframe.

Let's run the batch.py script for January 2023 (the fake data we created in Q5).

We can do that from our integration test in Python: we can use os.system for doing that (there are other options too).

Now it saves the result to localstack.

The only thing we need to do now is to read this data and verify the result is correct.

What's the sum of predicted durations for the test dataframe?

* 13.08
* 36.28
* 69.28
* 81.08

**Answer – 36.28**

Code - [Module-6/Homework/integration\_test.py at main · sandeepdas900/Module-6 (github.com)](https://github.com/sandeepdas900/Module-6/blob/main/Homework/integration_test.py)

### Screenshot

