Lesson Description - Implementing PostgreSQL Interaction

We now have tests for our `pgdump` implementation, and we have a basic understanding of mocking. Let's start following the errors to completion.

Documentation For This Video

- The `pytest-mock` package
- The `subprocess` package
- The `subprocess.Popen` class
- The `mocker.patch` function
- The `pytest.raises` function
- The `sys.exit` function

Initial Implementation

Our first error is from not having a `src/pgbackup/pgdump.py` file, so let's be sure to create that. We can guess that we'll also have an error for the missing function, so let's skip ahead a little and implement that:

src/pgbackup/pgdump.py

```
import subprocess

def dump(url):
      return subprocess.Popen(['pg_dump', url],
stdout=subprocess.PIPE)
```

This will get our tests to passing, but what happens when the `pg_dump` utility isn't installed?

Adding Tests For Missing PostgreSQL Client

Let's add another test that tells our `subprocess.Popen` to raise an `OSError` instead of succeeding. This is the kind of error that we will receive if the end-user of our package doesn't have the `pg_dump` utility installed. To cause our stub to raise this error we need to set the `side_effect` attribute when we call `mocker.patch`. We'll pass in an `OSError` to this attribute. Finally, we'll want to exit with a status code of 1 if we catch this error and pass the error message through. That means we'll need to use pytest.raises again to ensure we receive a SystemExit error. Here’s what the final tests look like for our pgdump module:

tests/test_pgdump.py

```
import pytest
import subprocess

from pgbackup import pgdump

url = "postgres://bob:password@example.com:5432/db_one"

def test_dump_calls_pg_dump(mocker):
    """
    Utilize pg_dump with the database URL
    """
    mocker.patch('subprocess.Popen')
    assert pgdump.dump(url)
    subprocess.Popen.assert_called_with(['pg_dump', url],
stdout=subprocess.PIPE)

def test_dump_handles_oserror(mocker):
    """
    pgdump.dump returns a reasonable error if pg_dump isn't
installed.
    """
    mocker.patch('subprocess.Popen', side_effect=OSError("no such
file"))
    with pytest.raises(SystemExit):
        pgdump.dump(url)
```

Implementing Error Handling

Since we know that subprocess.Popen can raise an OSError, we’re going to wrap that call in a try block, print the error message, and use sys.exit to set the error code:

src/pgbackup/pgdump.py

```
import sys
import subprocess

def dump(url):
    try:
        return subprocess.Popen(['pg_dump', url],
stdout=subprocess.PIPE)
    except OSError as err:
        print(f"Error: {err}")
        sys.exit(1)
```

Manual Testing

We can have a certain amount of confidence in our code because we've written tests that cover our expected cases, but since we used patching, we don't know that it works. Let's manually load our code into the python REPL to test it out:

```
(pgbackup-E7nj_BsO) $ PYTHONPATH=./src python
>>> from pgbackup import pgdump
>>> dump = pgdump.dump('postgres://demo:password@54.245.63.9:80/
sample')
>>> f = open('dump.sql', 'w+b')
>>> f.write(dump.stdout.read())
>>> f.close()
```

Note: We needed to open our `dump.sql` file using the `w+b` flag because we know that the `.stdout` value from a subprocess will be a `bytes` object and not a `str`.

If we exit and take a look at the contents of the file using `cat`, we should see the SQL output. With the `pgdump` module implemented, it's now a great time to commit our code.