First, we need to import the necessary libraries and read the data from the CSV file. We will use pandas library to read the csv file.

Code:

**import pandas as pd**

**data = pd.read\_csv('orders.csv')**

Task 1: Compute the total revenue generated by the online store for each month in the dataset.

We can create a new column 'month' by extracting the month from the order\_date column. Then we can group the data by month and sum the revenue.

Code:

**# extract month from order\_date column**

**data['month'] = pd.to\_datetime(data['order\_date']).dt.month**

**# group by month and sum the revenue**

**monthly\_revenue = data.groupby('month')['product\_price'].sum()**

Task 2: Compute the total revenue generated by each product in the dataset.

We can group the data by product\_id and sum the revenue.

**product\_revenue = data.groupby('product\_id')['product\_price'].sum()**

**Task 3:** Compute the total revenue generated by each customer in the dataset.

We can group the data by customer\_id and sum the revenue.

**customer\_revenue = data.groupby('customer\_id')['product\_price'].sum()**

Task 4: Identify the top 10 customers by revenue generated.

We can sort the data by revenue in descending order and select the top 10 customers.

**top\_10\_customers = customer\_revenue.sort\_values(ascending=False)[:10]**

Now, we can write the tests for our code. We will use unittest library to write the tests.

Code:

**import unittest**

**class TestOrderAnalysis(unittest.TestCase):**

**def setUp(self):**

**self.data = pd.read\_csv('orders.csv')**

**def test\_monthly\_revenue(self):**

**self.assertEqual(list(monthly\_revenue), [3000.0, 2800.0, 4500.0, 4200.0])**

**def test\_product\_revenue(self):**

**self.assertEqual(product\_revenue.loc[1], 5200.0)**

**self.assertEqual(product\_revenue.loc[2], 3700.0)**

**self.assertEqual(product\_revenue.loc[3], 4000.0)**

**self.assertEqual(product\_revenue.loc[4], 4400.0)**

**def test\_customer\_revenue(self):**

**self.assertEqual(customer\_revenue.loc[1], 5600.0)**

**self.assertEqual(customer\_revenue.loc[2], 3700.0)**

**self.assertEqual(customer\_revenue.loc[3], 4700.0)**

**self.assertEqual(customer\_revenue.loc[4], 3400.0)**

**def test\_top\_10\_customers(self):**

**self.assertEqual(list(top\_10\_customers.index), [1, 3, 4, 2])**

**if \_\_name\_\_ == '\_\_main\_\_':**

**unittest.main()**

Finally, we can dockerize our application. We will create two separate services, one for the task and one for the test. We will use Dockerfile to build the images.

Dockerfile for task service:

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", "app.py" ]

Dockerfile for test service:

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", "test.py" ]

We also need to create a docker-compose file to run both the services.

docker-compose.yml:

yaml

version: "3.9"

services:

task: