1

The problem statement: describes a coding task where you need to analyze an online store's customer orders data from a CSV file

Objective:

Develop a Python program to process and analyze customer orders data from an online store. The tasks include:

1. Monthly Revenue Calculation :

- Compute the total revenue generated by the store for each month in the dataset.

2. Product Revenue Calculation :

- Compute the total revenue generated by each product in the dataset.

3. Customer Revenue Calculation :

- Compute the total revenue generated by each customer in the dataset.

4. Top 10 Customers :

- Identify the top 10 customers by the revenue they generated .

Suggested Approach:

1. Data Reading and Preprocessing :

- Read the orders.csv file using pandas or similar library.

- Parse the date column correctly for monthly aggregation.

2. Calculations :

- Aggregate the data to compute the required revenues (monthly, product-wise, customer-wise).

- Identify the top 10 customers by sorting the customer revenue data.

3. Code Structure :

- Organize the code into functions for each major task.

- Ensure the code handles potential errors (e.g., missing data, incorrect data types).

4. Testing :

- Write unit tests for each function using a testing framework like pytest.

- Ensure the tests cover normal cases, edge cases, and error handling.

5. Dockerization :

- Create a Dockerfile for the main application.

- Create a Dockerfile for the test suite.

- Write a Docker Compose file to manage the services.

[11:10 pm, 30/7/2024] Mom: - Aggregate the data to compute the required revenues (monthly, product-wise, customer-wise).

- Identify the top 10 customers by sorting the customer revenue data.

2

. Main Python Code

python

import pandas as pd

from datetime import datetime

# Function to read and preprocess data

def read\_data(file\_path):

try:

data = pd.read\_csv(file\_path)

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

return data

except Exception as e:

print(f"Error reading the data: {e}")

return None

# Function to compute monthly revenue

def compute\_monthly\_revenue(data):

data['month'] = data['order\_date'].dt.to\_period('M')

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

return monthly\_revenue

# Function to compute revenue by product

def compute\_product\_revenue(data):

product\_revenue = data.groupby('product\_name')['product\_price'].sum().reset\_index()

return product\_revenue

# Function to compute revenue by customer

def compute\_customer\_revenue(data):

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

return customer\_revenue

# Function to identify top 10 customers

def top\_customers(data, top\_n=10):

customer\_revenue = compute\_customer\_revenue(data)

top\_customers = customer\_revenue.sort\_values(by='product\_price', ascending=False).head(top\_n)

return top\_customers

# Main function

def main():

data = read\_data('orders.csv')

if data is not None:

monthly\_revenue = compute\_monthly\_revenue(data)

product\_revenue = compute\_product\_revenue(data)

customer\_revenue = compute\_customer\_revenue(data)

top\_10\_customers = top\_customers(data)

print("Monthly Revenue:")

print(monthly\_revenue)

print("\nRevenue by Product:")

print(product\_revenue)

print("\nRevenue by Customer:")

print(customer\_revenue)

print("\nTop 10 Customers by Revenue:")

print(top\_10\_customers)

if \_\_name\_\_ == "\_\_main\_\_":

main()

# 2. Tests

Create a file named test\_orders.py for testing.

python

import unittest

import pandas as pd

from main import read\_data, compute\_monthly\_revenue, compute\_product\_revenue, compute\_customer\_revenue, top\_customers

class TestOrders(unittest.TestCase):

def setUp(self):

self.data = pd.DataFrame({

'order\_id': [1, 2, 3, 4],

'customer\_id': [101, 102, 101, 103],

'order\_date': ['2023-01-01', '2023-01-15', '2023-02-01', '2023-02-15'],

'product\_id': [201, 202, 201, 203],

'product\_name': ['Product A', 'Product B', 'Product A', 'Product C'],

'product\_price': [100, 200, 150, 250],

'quantity': [1, 2, 1, 1]

})

self.data['order\_date'] = pd.to\_datetime(self.data['order\_date'])

def test\_monthly\_revenue(self):

monthly\_revenue = compute\_monthly\_revenue(self.data)

expected = pd.DataFrame({'month': ['2023-01', '2023-02'], 'product\_price': [300, 400]})

pd.testing.assert\_frame\_equal(monthly\_revenue, expected)

def test\_product\_revenue(self):

product\_revenue = compute\_product\_revenue(self.data)

expected = pd.DataFrame({'product\_name': ['Product A', 'Product B', 'Product C'], 'product\_price': [250, 200, 250]})

pd.testing.assert\_frame\_equal(product\_revenue, expected)

def test\_customer\_revenue(self):

customer\_revenue = compute\_customer\_revenue(self.data)

expected = pd.DataFrame({'customer\_id': [101, 102, 103], 'product\_price': [250, 200, 250]})

pd.testing.assert\_frame\_equal(customer\_revenue, expected)

def test\_top\_customers(self):

top\_10\_customers = top\_customers(self.data)

expected = pd.DataFrame({'customer\_id': [101, 103, 102], 'product\_price': [250, 250, 200]})

pd.testing.assert\_frame\_equal(top\_10\_customers, expected)

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

unittest.main()

# 3. Docker Setup

Dockerfile for the main application :

Dockerfile

# Dockerfile

FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

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

Dockerfile for the tests :

Dockerfile

# Dockerfile.test

FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

CMD ["python", "-m", "unittest", "discover", "-s", "tests"]

requirements.txt :

pandas

unittest

docker-compose.yml :

yaml

version: '3.8'

services:

app:

build:

context: .

dockerfile: Dockerfile

volumes:

- .:/app

test:

build:

context: .

dockerfile: Dockerfile.test

volumes:

- .:/app

command: python -m unittest discover -s tests

# 4. README.md

markdown

# Online Store Orders Analysis

Overview

This project analyzes customer orders data from an online store to compute various revenue metrics and identify top customers.

Setup

# Prerequisites

- Docker

- Docker Compose

# Running the Application

1. Build and run the application:

bash

docker-compose up --build app

2. View the output in the console.

# Running the Tests

1. Build and run the tests:

bash

docker-compose up --build test

2. View the test results in the console.

Code Structure

- `main.py`: Main script for data processing and analysis.

- `test\_orders.py`: Unit tests for the main script.

- `Dockerfile`: Dockerfile for the main application.

- `Dockerfile.test`: Dockerfile for running the tests.

- `docker-compose.yml`: Docker Compose file to manage the services.

- `requirements.txt`: Python dependencies.

Notes

- Ensure the `orders.csv` file is in the root directory of the project.

- The code includes error handling and comments to explain the logic.