Data Engineer Intern Practical Exam

Name:P.H.H.Chamika Deemantha

Email:chamikade01@gmail.com

Table of Contents

1 Part 1: Video Demonstration & Connection of GitHub

Below is a video demonstration link of the completed task.

url: https://www.youtube.com/watch?v=VSg1cmsiJ5Y

Below is the GitHub link for codebase.

url: https://github.com/chamikadeemantha/Delivergate_Practicle_Test

2 Part 2: Data Preparation

2.1 Data Import:

C: > Users > User > Desktop > Delivergate Pvt Ltd > xampp.py > ...

```python
import pandas as pd    Import "pandas" could not be resolved from source
from sqlalchemy import create_engine, Table, Column, Integer, String, Float, MetaData, DateTime    Import "sqlalchemy" could not be resolved
import pymysql    Import "pymysql" could not be resolved from source

# MySQL Database Connection Parameters
MYSQL_USER = 'root'
MYSQL_PASSWORD = ''  # Empty password as per XAMPP config
MYSQL_HOST = 'localhost'
MYSQL_PORT = 3306
MYSQL_DB = 'delivergatedb'

# Create a connection string
db_connection_str = f'mysql+pymysql://{MYSQL_USER}:{MYSQL_PASSWORD}@{MYSQL_HOST}:{MYSQL_PORT}/{MYSQL_DB}'
engine = create_engine(db_connection_str)

# File paths for the CSVs
customers_file_path = r'C:\Users\User\Desktop\Delivergate Pvt Ltd\customers.csv'
orders_file_path = r'C:\Users\User\Desktop\Delivergate Pvt Ltd\order.csv'

# Load CSV files into DataFrames
customers_df = pd.read_csv(customers_file_path, usecols=['customer_id', 'name'])  # Only keep relevant columns
orders_df = pd.read_csv(orders_file_path, usecols=['id', 'customer_id', 'total_amount', 'created_at'])  # Only keep relevant columns

# Rename columns to match the database schema
customers_df = customers_df.rename(columns={'name': 'customer_name'})
orders_df = orders_df.rename(columns={'id': 'order_id', 'created_at': 'order_date'})

# Prepare the metadata for creating tables
metadata = MetaData()

# Define the Customers table
customers_table = Table(
    'customers', metadata,
    Column('customer_id', Integer, primary_key=True),
    Column('customer_name', String(255))
)

# Define the Orders table
orders_table = Table(
    'orders', metadata,
    Column('order_id', Integer, primary_key=True),
    Column('customer_id', Integer),
    Column('total_amount', Float),
    Column('order_date', DateTime)
)

# Create tables in the MySQL database
metadata.create_all(engine)

# Insert data into customers table
customers_df.to_sql('customers', con=engine, if_exists='append', index=False)

# Insert data into orders table
orders_df.to_sql('orders', con=engine, if_exists='append', index=False)

print("Data has been imported into the MySQL database successfully!")
```

2.2 Database Connection:

C: > Users > User > Desktop > Flask App > app.py > ...

```python
import matplotlib    Import "matplotlib" could not be resolved from source
matplotlib.use('Agg')  # Use non-interactive backend
from flask import Flask, render_template, request, redirect, url_for    Import "flask" could not be resolved
from sqlalchemy import create_engine    Import "sqlalchemy" could not be resolved
import pandas as pd    Import "pandas" could not be resolved from source
import matplotlib.pyplot as plt    Import "matplotlib.pyplot" could not be resolved from source
from io import BytesIO
import base64

app = Flask(__name__)

# Database connection
MYSQL_USER = 'root'
MYSQL_PASSWORD = ''  # Empty password as per XAMPP config
MYSQL_HOST = 'localhost'
MYSQL_PORT = 3306
MYSQL_DB = 'delivergatedb'

db_connection_str = f'mysql+pymysql://{MYSQL_USER}:{MYSQL_PASSWORD}@{MYSQL_HOST}:{MYSQL_PORT}/{MYSQL_DB}'
engine = create_engine(db_connection_str)

@app.route("/", methods=["GET", "POST"])
def index():
    toast_message = None
    try:
        start_date = request.form.get("start_date")
        end_date = request.form.get("end_date")
        min_total_amount = request.form.get("min_total_amount", type=float, default=0)
        min_order_count = request.form.get("min_order_count", type=int, default=1)
    except ValueError:
        toast_message = "Please enter valid values for all filters."
        return render_template("index.html", toast_message=toast_message)

    # Query to get customers with enough orders
    customer_query = """
        SELECT o.customer_id, c.customer_name, COUNT(o.order_id) AS order_count
        FROM orders o
        JOIN customers c ON o.customer_id = c.customer_id
        WHERE o.total_amount >= %s
    """
    filters = [min_total_amount]
    if start_date and end_date:
        customer_query += " AND o.order_date BETWEEN %s AND %s"
        filters.extend([start_date, end_date])
    customer_query += " GROUP BY o.customer_id HAVING COUNT(o.order_id) >= %s"
    filters.append(min_order_count)

    connection = engine.raw_connection()
    try:
        customers_df = pd.read_sql(customer_query, connection, params=filters)
        if customers_df.empty:
            toast_message = "No customers found with the specified order count and total amount filters."
            return render_template("index.html", toast_message=toast_message)
        customer_ids = customers_df['customer_id'].tolist()
    finally:
        connection.close()
```

```python
    # Query to get orders of the filtered customers
    if customer_ids:
        orders_query = """
            SELECT o.*, c.customer_name
            FROM orders o
            JOIN customers c ON o.customer_id = c.customer_id
            WHERE o.customer_id IN (%s)
        """ % ','.join(map(str, customer_ids))

        connection = engine.raw_connection()
        try:
            orders_df = pd.read_sql(orders_query, connection)
        finally:
            connection.close()
    else:
        orders_df = pd.DataFrame()

    # Ensure total_amount is numeric, setting non-numeric values to NaN and then dropping them
    orders_df['total_amount'] = pd.to_numeric(orders_df['total_amount'], errors='coerce')
    orders_df = orders_df.dropna(subset=['total_amount'])
```

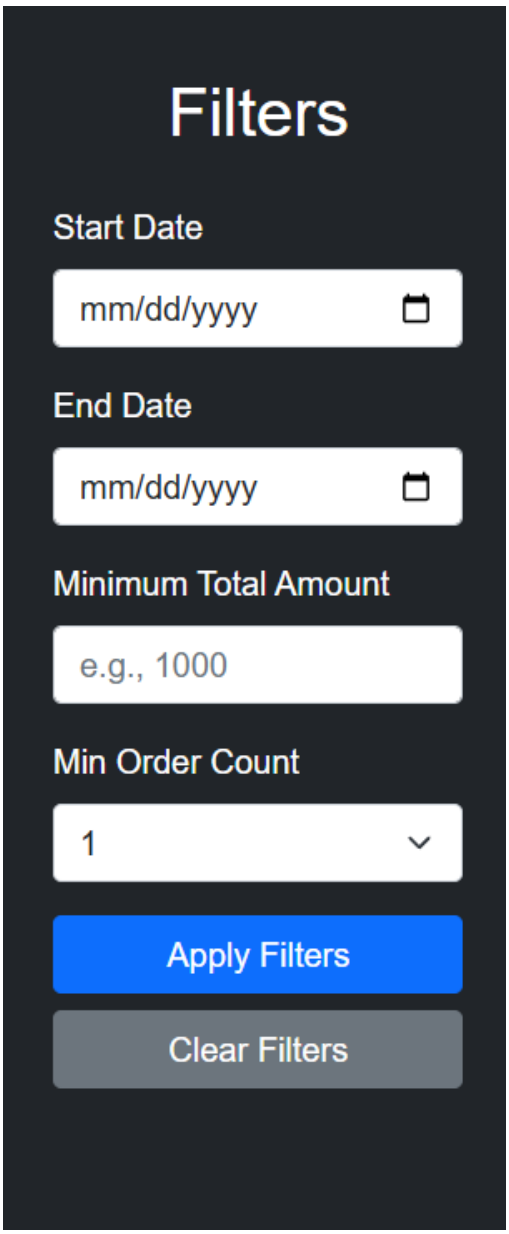
```python
    # Convert order_date to datetime and handle errors
    try:
        orders_df['order_date'] = pd.to_datetime(orders_df['order_date'], errors='coerce')
        orders_df = orders_df.dropna(subset=['order_date'])
        orders_df = orders_df.set_index('order_date')
    except Exception as e:
        toast_message = "Error processing dates. Please check the date format."
        return render_template("index.html", toast_message=toast_message)

    # Calculate Total Spent by each customer
    orders_df['Total Spent'] = orders_df.groupby('customer_id')['total_amount'].transform('sum')

    # Apply search filter for customer name if provided
    customer_name_filter = request.form.get("customer_name")
    if customer_name_filter:
        orders_df = orders_df[orders_df['customer_name'].str.contains(customer_name_filter, case=False, na=False)]

    # Key metrics
    total_revenue = orders_df['total_amount'].sum()
    unique_customers = orders_df['customer_id'].nunique()
    total_orders = len(orders_df)
```

```python
    # Visualizations
    top_customers_img = None
    revenue_over_time_img = None

    def plot_to_img(plot_func):
        fig, ax = plt.subplots(figsize=(8, 6))
        plot_func(ax)
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        img = BytesIO()
        fig.savefig(img, format='png')
        img.seek(0)
        plt.close(fig)
        return base64.b64encode(img.getvalue()).decode()

    if not orders_df.empty:
        # Group by customer and get top 10 customers by revenue
        top_customers = orders_df.groupby(['customer_name', 'customer_id']).total_amount.sum().nlargest(10)

        # Generate top customers plot if data exists
        if not top_customers.empty:
            top_customers_img = plot_to_img(lambda ax: top_customers.plot(kind='bar', title="Top 10 Customers by Revenue", ax=ax, xlabel="Customer Name"))

        # Generate revenue over time plot if there is data
        revenue_over_time = orders_df.resample('M').total_amount.sum()
        if not revenue_over_time.empty:
            revenue_over_time_img = plot_to_img(lambda ax: revenue_over_time.plot(kind='line', title="Revenue Over Time", ax=ax, xlabel="Order Date"))

    # Rename columns for a more readable table display
    orders_df = orders_df.rename(columns={
        'order_id': 'Order ID',
        'customer_id': 'Customer ID',
        'total_amount': 'Order Amount',
        'customer_name': 'Customer Name',
        'order_count': 'Order Count'
    })

    # Show only the first 10 rows of the filtered data
    orders_table_html = orders_df.head(10).to_html(classes='table table-striped', index=False)

    return render_template("index.html", total_revenue=total_revenue,
                           unique_customers=unique_customers,
                           total_orders=total_orders,
                           top_customers_img=top_customers_img,
                           revenue_over_time_img=revenue_over_time_img,
                           orders_table_html=orders_table_html,
                           toast_message=toast_message)

@app.route("/reset_filters", methods=["GET"])
def reset_filters():
    # Redirect to the main index page without any filters
    return redirect(url_for('index'))

if __name__ == "__main__":
    app.run(debug=True)
```

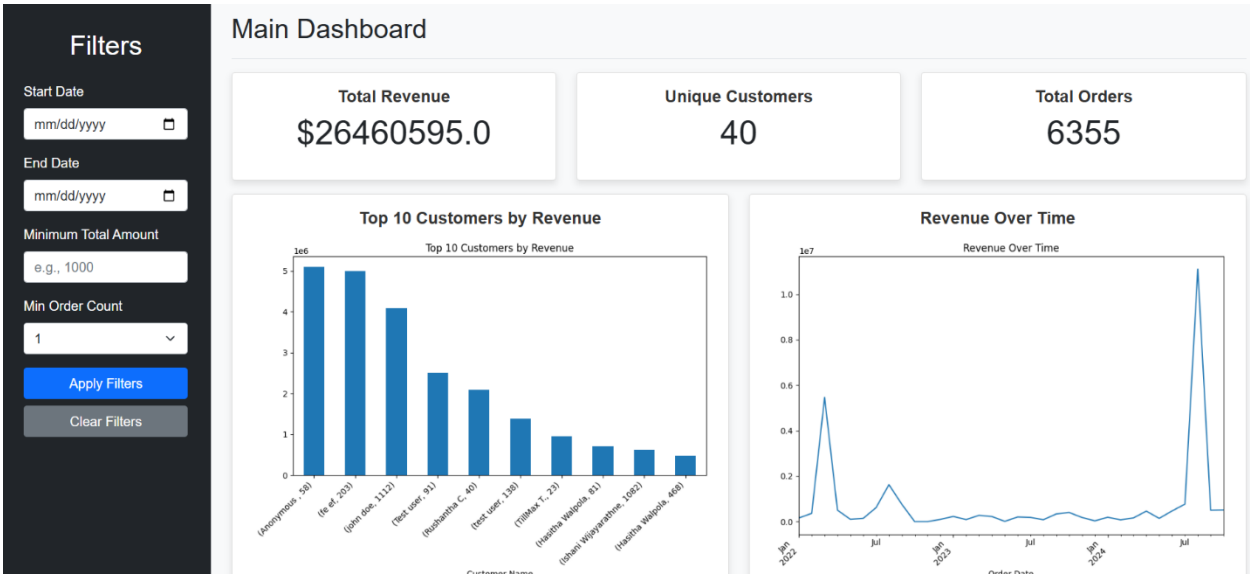
3 Part 3: Flask App Setup

After experiencing persistent MySQL connection issues with the Streamlit application and multiple unsuccessful attempts to resolve them, I've decided to transition to using Flask. Flask provides a more stable connection handling mechanism for MySQL, which will improve application reliability and user experience.

3.1 Sidebar Filters:



3.2 Main Dashboard:



Filtered Orders Table

Search by Customer Name | Search | Clear Search

Order ID	Customer ID	Order Amount	Customer Name	Total Spent
222	8	2500.0	customer three	27500.0
224	8	2500.0	customer three	27500.0
227	8	2500.0	customer three	27500.0
228	8	2500.0	customer three	27500.0
229	8	2500.0	customer three	27500.0
230	8	2500.0	customer three	27500.0
231	8	2500.0	customer three	27500.0
232	8	2500.0	customer three	27500.0
233	8	2500.0	customer three	27500.0
236	8	2500.0	customer three	27500.0

4 Part 4: Data Analysis

4.1 Machine Learning Model

A simple logistic regression model was successfully implemented to predict repeat purchasers based on total orders and revenue, with validation checks for data sufficiency and accuracy metrics included. Relevant images illustrating the model and results are provided.

Inspect the the datasets

```
#First few rows of orders_df
orders_df.head()
```

	id	display_order_id	total_amount	created_at	customer_id
0	13392	YTFA	425	2024-10-14 15:12:43	1251.0
1	13393	N1U7	1650	2024-10-14 15:17:25	1251.0
2	13394	PADV	1365	2024-10-14 17:02:16	468.0
3	13395	OKVW	525	2024-10-14 17:03:36	1251.0
4	13396	2G3Q	525	2024-10-14 17:04:49	468.0

Data Preprocessing

Checking Missing Values

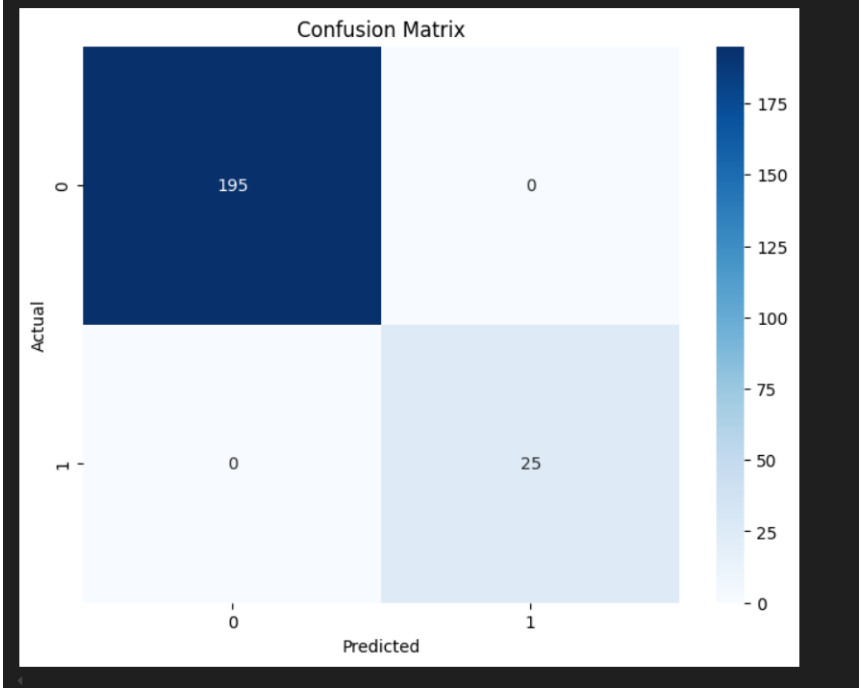
```
print(customers_df.isnull().sum())
```

```
customer_id     0
name            6
email          18
dtype: int64
```

```
print(orders_df.isnull().sum())
```

```
id                   0
display_order_id    36
total_amount         0
created_at           0
customer_id        125
dtype: int64
```

```
Model Accuracy: 100.00%
```




```
Cross-Validation Accuracy Scores: [1. 1. 1. 1. 1.]
Mean CV Accuracy: 100.00%
```