## **Overview**

This document outlines the steps taken to clean datasets, load them into a MySQL database, and extract insights using SQL queries. The datasets include information about customers, products, sales, and stores.

### **Purpose**

The code automates the following processes:

1. **Data Loading**: Importing CSV files into Pandas DataFrames.
2. **Data Cleaning**: Handling missing values and ensuring correct data types.
3. **Data Merging**: Combining datasets for comprehensive analysis.
4. **Data Storage**: Loading cleaned data into a MySQL database.
5. **Data Analysis**: Executing SQL queries to derive insights.

## **Prerequisites**

**Python Libraries**: Ensure you have the following libraries installed:  
  
pip install pandas sqlalchemy mysql-connector-python

**MySQL Database**: Create a MySQL database named datacleaning.

CREATE DATABASE data cleaning;

## **Step-by-Step Process**

### **1. Import Libraries**

**Code:**

python

import pandas as pd

from sqlalchemy import create\_engine

from sqlalchemy.exc import SQLAlchemyError

**Purpose:** Import necessary libraries for data manipulation (pandas) and database interaction (sqlalchemy).

### **2. Define MySQL Connection Parameters**

**Code:**

python

username = 'abcd'

password = 'abcd'

host = 'localhost'

database = 'datacleaning'

**Purpose:** Set up connection parameters to access the MySQL database.

### **3. Load Datasets**

**Code:**

python

try:

customers = pd.read\_csv('Customers.csv', encoding='ISO-8859-1')

products = pd.read\_csv('Products.csv', encoding='ISO-8859-1')

sales = pd.read\_csv('Sales.csv', encoding='ISO-8859-1', header=None)

stores = pd.read\_csv('Stores.csv', encoding='ISO-8859-1')

sales.columns = ['Order Number', 'Line Item', 'Order Date', 'Delivery Date',

'CustomerKey', 'StoreKey', 'ProductKey', 'Quantity', 'Currency Code']

sales['Order Date'] = pd.to\_datetime(sales['Order Date'], format='%d-%m-%Y', errors='coerce')

except Exception as e:

print(f"Error loading CSV files: {e}")

exit(1)

**Purpose:** Load CSV files into Pandas DataFrames. This step includes setting the correct column names for the sales dataset and converting the order date into a proper datetime format.

### **4. Data Preparation**

#### **Ensure Proper Data Types**

**Code:**

python

sales['CustomerKey'] = sales['CustomerKey'].astype(str)

customers['CustomerKey'] = customers['CustomerKey'].astype(str)

sales['ProductKey'] = sales['ProductKey'].astype(str)

products['ProductKey'] = products['ProductKey'].astype(str)

**Purpose:** Convert key columns to string type to ensure compatibility when merging datasets.

#### **Handle Missing Values**

**Code:**

python

if 'age' in customers.columns:

customers['age'].fillna(customers['age'].median(), inplace=True)

**Purpose:** Fill missing values in the age column of the customers DataFrame with the median age, ensuring data integrity.

### **5. Merge Datasets**

**Code:**

python

merged\_data = sales.merge(customers, on='CustomerKey', how='inner').merge(products, on='ProductKey', how='inner')

**Purpose:** Merge the sales, customers, and products datasets on their respective keys to create a comprehensive dataset for analysis.

#### **Check Merge Success**

**Code:**

python

if merged\_data.empty:

print("Merged data is empty. Check for missing customer or product IDs.")

else:

print(merged\_data.head())

**Purpose:** Verify the success of the merging process by checking if the resulting DataFrame is empty.

### **6. Load Data into MySQL**

**Code:**

python

try:

engine = create\_engine(f'mysql+mysqlconnector://{username}:{password}@{host}/{database}?connect\_timeout=300')

with engine.connect() as connection:

merged\_data.to\_sql('sales\_data', con=connection, if\_exists='replace', index=False, chunksize=500)

customers.to\_sql('customers', con=connection, if\_exists='replace', index=False, chunksize=500)

products.to\_sql('products', con=connection, if\_exists='replace', index=False, chunksize=500)

stores.to\_sql('stores', con=connection, if\_exists='replace', index=False, chunksize=500)

print("Data loaded successfully into MySQL database.")

except SQLAlchemyError as e:

print(f"Error loading data into MySQL: {e}")

**Purpose:** Create an SQLAlchemy engine to connect to the MySQL database and load the cleaned datasets into their respective tables.

## **7. SQL Queries**

Below are SQL queries that can be executed to extract key insights from the loaded data. Each query addresses specific business questions:

**Total Sales per Product:**  
SELECT P.ProductName, SUM(S.Quantity \* P.Price) AS TotalSales

FROM sales\_data S

JOIN products P ON S.ProductKey = P.ProductKey

GROUP BY P.ProductName;

**Top Customers by Sales Amount:**  
SELECT C.Name, SUM(S.Quantity \* P.Price) AS TotalSpent

FROM sales\_data S

JOIN customers C ON S.CustomerKey = C.CustomerKey

JOIN products P ON S.ProductKey = P.ProductKey

GROUP BY C.Name

ORDER BY TotalSpent DESC

LIMIT 10;

**Sales Trend Over Time:**  
SELECT DATE\_TRUNC('month', S.Order\_Date) AS SaleMonth, SUM(S.Quantity \* P.Price) AS MonthlySales

FROM sales\_data S

JOIN products P ON S.ProductKey = P.ProductKey

GROUP BY SaleMonth

ORDER BY SaleMonth;

**Number of Sales per Customer:**  
SELECT C.Name, COUNT(S.Order\_Number) AS NumberOfSales

FROM sales\_data S

JOIN customers C ON S.CustomerKey = C.CustomerKey

GROUP BY C.Name;

**Average Sale Amount:**  
SELECT AVG(S.Quantity \* P.Price) AS AverageSale

FROM sales\_data S

JOIN products P ON S.ProductKey = P.ProductKey;

**Sales by Region:**  
SELECT C.Region, SUM(S.Quantity \* P.Price) AS TotalSales

FROM sales\_data S

JOIN customers C ON S.CustomerKey = C.CustomerKey

JOIN products P ON S.ProductKey = P.ProductKey

GROUP BY C.Region;

**Products Not Sold:**  
SELECT P.ProductName

FROM products P

LEFT JOIN sales\_data S ON P.ProductKey = S.ProductKey

WHERE S.Order\_Number IS NULL;

**Customer Retention Rate:**  
SELECT COUNT(DISTINCT C.CustomerKey) AS ActiveCustomers,

COUNT(DISTINCT CASE WHEN S.Order\_Date >= NOW() - INTERVAL '1 year' THEN C.CustomerKey END) AS RetainedCustomers

FROM customers C

LEFT JOIN sales\_data S ON C.CustomerKey = S.CustomerKey;

**Sales Distribution by Product:**  
SELECT P.ProductName, COUNT(S.Order\_Number) AS SalesCount

FROM sales\_data S

JOIN products P ON S.ProductKey = P.ProductKey

GROUP BY P.ProductName;

**Sales in a Specific Time Frame:**  
SELECT SUM(S.Quantity \* P.Price) AS TotalSales

FROM sales\_data S

JOIN products P ON S.ProductKey = P.ProductKey

WHERE S.Order\_Date BETWEEN '2023-01-01' AND '2023-12-31';

### **10 . Power BI Visualization**

#### **Connect to MySQL Database**

1. **Open Power BI Desktop.**
2. **Get Data**:
   * Click on “Get Data” from the Home ribbon.
   * Select “MySQL database” from the list of data sources.
3. **Enter Connection Details**:
   * **Server**: Enter your server name (e.g., localhost).
   * **Database**: Enter data cleaning.
   * Click “OK.”
4. **Import Tables**:
   * Select the tables you want to visualise (e.g., sales\_data, customers, products, stores).
   * Click “Load” to import the selected tables.

#### **Create Visualisations**

1. **Create Dashboard**:
   * Use the available visualisations in Power BI (e.g., bar charts, line charts, pie charts) to represent insights from your data.
2. **Build Reports**:
   * Drag and drop fields from the data model onto the report canvas.
   * Use slicers for filtering data dynamically (e.g., by date, product, customer).
3. **Save and Publish**:
   * Save your Power BI report locally or publish it to Power BI Service for sharing with stakeholders.

## **Conclusion**

This documentation provides a comprehensive guide for:

* Cleaning datasets.
* Loading them into a MySQL database.
* Executing SQL queries to derive insights.
* Connecting to Power BI for interactive visualisation.