

1.

PART 1: SQL (MySQL) Commands

◆ CREATE DATABASE supermarket;

► What it does:

Creates a new database named supermarket in your MySQL server.

? Common questions:

- What is a database?
A structured collection of data. Here, supermarket will hold sales-related data.
 - What happens if the database already exists?
MySQL will return an error unless you use CREATE DATABASE IF NOT EXISTS.
-

◆ SHOW TABLES;

► What it does:

Lists all tables in the currently selected database.

◆ USE supermarket;

► What it does:

Switches the context to the supermarket database so further operations affect this database.

◆ CREATE TABLE supermarket (...);

► What it does:

Defines a table with these columns:

Column	Data Type	Meaning
order_id	VARCHAR(20)	Unique order ID (primary key)
branch	VARCHAR(50)	Store branch name
city	VARCHAR(50)	City where transaction occurred
gender	VARCHAR(100)	Customer's gender
product	VARCHAR(100)	Product category

Column	Data Type	Meaning
Quantity	VARCHAR(100)	Quantity bought (ideally should be INT)
Payment	VARCHAR(50)	Payment method used
Sales	FLOAT	Total sales amount
unit_price	FLOAT	Price per unit of product

! Notes:

- Quantity is stored as VARCHAR, which should be INT or FLOAT for numeric operations.
- order_id is set as a PRIMARY KEY, ensuring uniqueness.

◆ INSERT INTO supermarket (...) VALUES (...);

► What it does:

Inserts one row into the supermarket table.

sql

CopyEdit

```
INSERT INTO supermarket (...) VALUES (
    1, 'Alex', 'Yangon', 'female', 'Health and beauty', '7', 'Ewallet', 548.9715, 74.69
);
```

! Note:

- '1' should ideally be a string to match VARCHAR.
- Be sure your inserted data types match column types.

◆ SELECT * FROM supermarket;

► What it does:

Displays all data in the supermarket table.

✅ PART 2: Python (Pandas + SQL)

◆ import pandas as pd, import numpy as np

► What it does:

- pandas is used for reading, writing, and manipulating data.

- numpy is used for numerical operations.

◆ **pip install sqlalchemy mysql-connector**

◆ **pip install pymysql**

► **What it does:**

Installs the necessary connectors:

Package	Purpose
sqlalchemy	ORM layer for interacting with SQL databases
mysql-connector	Official MySQL connector (not used in your code)
pymysql	Lightweight MySQL connector used with Pandas

◆ **data_excel = pd.read_excel("sales data.xlsx")**

► **What it does:**

Reads an Excel file (sales data.xlsx) into a DataFrame named data_excel.

? Questions:

- What if the file doesn't exist?
A FileNotFoundError will occur.
- What if it's not in Excel format?
Pandas will throw a format or parser error.

◆ **data_excel.head()**

► **What it does:**

Displays the **first 5 rows** of the DataFrame. Useful for a quick check.

◆ **SQLAlchemy Connection:**

python

CopyEdit

```
from sqlalchemy import create_engine
```

```
engine = create_engine("mysql+pymysql://root:W%402915djkq%23@localhost:3306/supermarket")
```

► **What it does:**

Creates a connection engine to the MySQL database using:

Part	Value
mysql+pymysql	Connection type
root	Username
W@2915djkg#	Password (URL-encoded)
localhost:3306	Server location and port
supermarket	Database name

◆ **pd.read_sql("SELECT * FROM supermarket", engine)**

► **What it does:**

Runs a SQL query and returns the result into a Pandas DataFrame data_sql.

◆ **Alternative Connection Using pymysql**

python

CopyEdit

```
conn = pymysql.connect(...)
```

```
data_sq = pd.read_sql('SELECT * FROM supermarket', conn)
```

► **What it does:**

- Uses direct pymysql.connect() to establish a connection.
- Reads SQL table into Pandas DataFrame.

❓ **Questions:**

- Why use this if SQLAlchemy is available?
Direct pymysql may offer lower-level control or compatibility.
-

✅ **PART 3: Export Pandas DataFrame to MySQL**

◆ **Export using to_sql()**

python

CopyEdit

```
engine = create_engine('mysql+pymysql://username:password@localhost/sales_db')
data.to_sql('sales', con=engine, if_exists='replace', index=False)
```

► **What it does:**

- Creates a connection to another database (sales_db)
- Saves the DataFrame data into MySQL as table sales

► **Parameters explained:**

Parameter	Meaning
'sales'	Name of the new table
con=engine	Connection to MySQL
if_exists='replace'	Replace table if it exists (use 'append' to add)
index=False	Don't write DataFrame's index as a new column

2.

1. Imports and Setup

python

CopyEdit

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

► **What it does:**

- **pandas:** For data loading and transformation.
 - **matplotlib.pyplot:** Basic plotting library.
 - **seaborn:** For more advanced, beautiful plots (though not used explicitly here).
-

✓ 2. Load Excel Data

python

CopyEdit

```
df = pd.read_excel("sales data.xlsx", sheet_name="sales data")
```

► **What it does:**

- Loads the "sales data" sheet from an Excel file into a DataFrame df.

❓ **Common issues:**

- File not found → FileNotFoundError
 - Sheet doesn't exist → ValueError
-

✅ **3. Data Transformation / Cleaning**

◆ **A. Convert date to datetime**

python

CopyEdit

```
df['date'] = pd.to_datetime(df['date'], errors='coerce')
```

► Converts string dates to datetime objects.

- errors='coerce': Invalid values become NaT (missing date).
-

◆ **B. Convert cost to numeric**

python

CopyEdit

```
df['cost'] = pd.to_numeric(df['cost'], errors='coerce')
```

► Cleans the 'cost' column (if it has extra symbols, text, or missing values).

◆ **C. Drop missing values**

python

CopyEdit

```
df.dropna(subset=['order_value_EUR', 'date'], inplace=True)
```

► Drops rows where either:

- order_value_EUR (the sales value) is missing
 - date is missing
-

◆ **D. Create a month column**

python

CopyEdit

```
df['month'] = df['date'].dt.to_period('M')
```

➤ Extracts **month and year** (e.g., '2024-07') to allow monthly aggregations.

4. Visualizations

◆ 1. Sales Over Time (Daily)

python

CopyEdit

```
plt.figure(figsize=(20,10))
```

```
df.groupby('date')['order_value_EUR'].sum().plot(title='Daily Sales Over Time', marker='o')
```

- Aggregates total daily sales.
 - marker='o': Adds dots to indicate daily values.
 - Large figsize: Makes the plot readable.
-

◆ 2. Sales by Country

python

CopyEdit

```
country_sales = df.groupby('country')['order_value_EUR'].sum().sort_values(ascending=False)
```

```
country_sales.plot(kind='bar', title='Sales by Country')
```

- Total sales grouped by each country.
 - Sorted in descending order.
 - Horizontal bars allow comparison.
-

◆ 3. Sales by Product Category (Pie Chart)

python

CopyEdit

```
category_sales = df.groupby('category')['order_value_EUR'].sum()
```

```
category_sales.plot(kind='pie', autopct='%1.1f%%', title='Sales by Product Category')
```

- Pie chart shows share of each category.
- autopct='%1.1f%%': Shows % values (e.g., 23.5%)

◆ 4. Monthly Sales Trend

python

CopyEdit

```
monthly_sales = df.groupby('month')['order_value_EUR'].sum()
```

```
monthly_sales.plot(marker='o', title='Monthly Sales Trend')
```

- Plots total sales **per month** to observe long-term trends.
- Useful for spotting seasonality or growth.

◆ 5. Top 5 Customers by Sales

python

CopyEdit

```
top_customers = df.groupby('customer_name')['order_value_EUR'].sum().nlargest(5)
```

```
top_customers.plot(kind='barh', title='Top 5 Customers by Sales')
```

- Identifies top 5 high-value customers.
 - Uses horizontal bars for clear labeling.
-

3. import pyodbc

Part 1: Connecting to MySQL using pyodbc

python

CopyEdit

```
import pyodbc
```

```
conn = pyodbc.connect(  
    'DRIVER={MySQL ODBC 8.0 Unicode Driver};'  
    'SERVER=localhost;'  
    'PORT=3306;'  
    'DATABASE=student;'
```



```
'UID=root;'
'PWD=root;'
)
```

✅ Explanation:

- **pyodbc** is a Python library to connect to databases using ODBC drivers.
- This code sets up a connection to a MySQL database named student on localhost:3306.
- It uses MySQL ODBC 8.0 Unicode Driver and login credentials:
 - Username: root
 - Password: root

⚠ Troubleshooting Tips:

- Make sure the **MySQL ODBC driver** is installed.
- If connection fails:
 - Check if MySQL service is running
 - Confirm database name is correct
 - Check username/password

◆ Part 2: Execute SQL and Fetch Data

python

CopyEdit

```
cursor = conn.cursor()
cursor.execute("SELECT * FROM stud")
rows = cursor.fetchall()
for row in rows:
    print(row)
conn.close()
```

✅ Explanation:

- **.cursor()**: Initializes a cursor to execute SQL commands.
- **.execute()**: Runs an SQL query (in this case, selects all data from the stud table).
- **.fetchall()**: Retrieves all rows from the query result.
- **print(row)**: Displays each row one by one.
- **conn.close()**: Closes the database connection.

◆ Part 3: List Available ODBC Drivers

```
python  
CopyEdit  
import pyodbc  
print(pyodbc.drivers())
```

✅ Explanation:

- Prints the list of installed ODBC drivers.
- Useful to confirm whether **MySQL ODBC 8.0 Unicode Driver** is installed.

◆ Part 4: SQL Queries for Setup (Assumed to run in MySQL Workbench/CLI)

```
sql  
CopyEdit  
CREATE DATABASE student;  
USE student;  
  
CREATE TABLE stud (  
    id INT,  
    name VARCHAR(40)  
);  
  
INSERT INTO stud VALUES (1, 'ved');  
  
SELECT * FROM stud;
```

✅ Explanation:

- CREATE DATABASE student;; Creates a new database.
 - USE student;; Sets the current working database.
 - CREATE TABLE stud (...): Creates a table with id and name.
 - INSERT INTO stud ...: Adds a record (1, 'ved').
 - SELECT * ...: Views the table contents.
-

◆ Part 5: Resetting MySQL User Password (optional system step)

sql

CopyEdit

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';
```

```
FLUSH PRIVILEGES;
```

✓ Explanation:

- **ALTER USER:** Changes the root user's password.
 - **FLUSH PRIVILEGES:** Applies the changes immediately.
 - Use this only if you face a login issue or want to reset your password.
-

◆ Part 6: Read Excel with Pandas

python

CopyEdit

```
!pip install openpyxl
```

```
import pandas as pd
```

```
df = pd.read_excel("./supermarket_sales.xlsx")
```

```
df
```

```
df.isnull().sum()
```

✓ Explanation:

- **!pip install openpyxl:** Installs the engine used to read .xlsx files.
 - **pd.read_excel(...):** Reads the Excel file into a pandas DataFrame.
 - **df.isnull().sum():** Checks for missing/null values in each column.
-

Summary Table

Section	Description
pyodbc.connect()	Connect to MySQL using ODBC
cursor.execute()	Run SQL queries
fetchall()	Fetch results from database

Section	Description
SQL commands (MySQL)	Create DB, tables, insert data, reset root password
pyodbc.drivers()	Check installed ODBC drivers
pandas.read_excel()	Load Excel data for analysis
isnull().sum()	Find missing data in DataFrame

5.

Step-by-Step Explanation

◆ 1. Importing Required Libraries

python

CopyEdit

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Purpose:

- pandas – to handle data (read CSV, manipulate columns)
 - train_test_split – to divide data into training and testing sets
 - RandomForestClassifier – the machine learning model
 - classification_report, confusion_matrix, accuracy_score – evaluation metrics
-

◆ 2. Loading the Dataset

python

CopyEdit

```
df = pd.read_csv("heart_disease.csv")
```

Assumes:

- The CSV file is named heart_disease.csv
- It includes features (e.g., age, sex, cholesterol) and a target column named target

◆ 3. Feature and Target Split

python

CopyEdit

```
X = df.drop("target", axis=1) # All input features
```

```
y = df["target"]           # The target variable
```

- **X**: independent variables (features)
- **y**: dependent variable (label – typically 0 = no disease, 1 = disease)

◆ 4. Train-Test Split

python

CopyEdit

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Splits the data into **80% training** and **20% testing**
- `random_state=42` ensures reproducibility
- Avoids overfitting and allows model validation

◆ 5. Training the Model


python

CopyEdit

```
model = RandomForestClassifier(random_state=42)
```

```
model.fit(X_train, y_train)
```

- Creates a **Random Forest** with default settings.
- Fits the model to the training data.

 **Random Forest** = multiple decision trees combined to improve prediction and reduce overfitting.

◆ 6. Making Predictions

python

CopyEdit

```
y_pred = model.predict(X_test)
```

- Uses the trained model to predict the labels of the test set.

◆ 7. Model Evaluation

python

CopyEdit

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

◆ **confusion_matrix:** Shows

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

◆ **classification_report:**

- **Precision:** $TP / (TP + FP)$
- **Recall:** $TP / (TP + FN)$
- **F1-score:** Harmonic mean of precision and recall
- **Support:** Number of true samples for each label

◆ **accuracy_score:**

- Overall ratio of correctly predicted samples to total samples
-

Example Output Interpretation (assumed):

lua

CopyEdit

Confusion Matrix:

```
[[26  4]
```

```
 [ 2 29]]
```

Classification Report:

```
precision  recall  f1-score  support
```

```
0         0.93    0.87    0.90     30
```

1	0.88	0.94	0.91	31
accuracy			0.90	61
macro avg	0.91	0.90	0.90	61
weighted avg	0.91	0.90	0.90	61

Accuracy Score: 0.9016

Common Questions and Tips

Question	Answer
Why Random Forest?	It's robust, reduces overfitting, handles nonlinear data well.
What's the target variable?	Whether the patient has heart disease (1) or not (0).
What if data is imbalanced?	Use metrics like F1-score or apply techniques like SMOTE.
How to improve accuracy?	Try tuning <code>n_estimators</code> , <code>max_depth</code> , or using cross-validation.
What preprocessing might be missing?	Encoding categorical variables, scaling (though not essential for trees), checking for nulls.