**Project 1: Sales Data Insights Pipeline**

**1.Project Objective**

To build a **simple data pipeline** using **Python, SQLite, and CSV files** to:

* Load sales data into a database
* Execute analysis queries
* Display insights

This shows a basic data engineering workflow.

**2️. Project Folder Structure**

**Folder Organization:**

sales\_pipeline/

├── data/ # Contains raw data files

│ └── sales\_data.csv

├── db/ # Contains the SQLite database

│ └── sales.db

├── scripts/ # Python and SQL scripts

│ ├── load\_data.py

│ └── run\_queries.py

│ └── queries.sql

└── README.md

**Purpose:**

* Keeps everything organized.
* Makes the pipeline easy to maintain.

**3️.Data Preparation**

**File:** data/sales\_data.csv

**Description:**  
A CSV file containing sample sales records, e.g.:

OrderID,Product,Category,Sales,Region,OrderDate

1001,Laptop,Electronics,1200,North,2023-01-10

1002,Mobile,Electronics,800,South,2023-01-15

1003,Book,Stationery,20,North,2023-01-12

1004,Pen,Stationery,5,East,2023-01-11

1005,Tablet,Electronics,600,West,2023-01-18

**Purpose:**  
This is the **input dataset** to be loaded into the database.

**4️.Loading Data into SQLite**

**Script:** scripts/load\_data.py

**What it does:**

* Imports required libraries (pandas, sqlite3, os)
* Reads the CSV file into a pandas DataFrame
* Connects to an SQLite database (sales.db)
* Writes the data into a table called sales

**Code:**

**import pandas as pd** #This **imports the pandas library** and gives it the alias pd.

#We use pandas to **read the CSV file** and work with it like a table

**import sqlite3** #It **connects to and create the sales.db file**, and write data into it.

**import os** # It help **build file paths**

# Load the CSV- It builds the path to the CSV file: Like : data/sales\_data.csv

csv\_path = os.path.join("data", "sales\_data.csv")

df = pd.read\_csv(csv\_path) #df:DataFrame, It reads the CSV file

# Connect to SQLite DB- db:database, same as CVS\_path. Creates the path for db like: db/sales.db

db\_path = os.path.join("db", "sales.db")

conn = sqlite3.connect(db\_path) # conn is connection, here we are connecting sqlite db.

# Here we created a new table as sales, used replace if any table exists in db. Index as false: we do not want pandas to create row index

df.to\_sql("sales", conn, if\_exists="replace", index=False)

print("✅ Data loaded into SQLite database successfully.")

conn.close() # the connection is closed

**Output:**  
✅ Prints confirmation:

✅ Data loaded into SQLite database successfully.

**Effect:**

* Creates or updates sales.db with a fresh sales table.

**5️. Writing SQL Queries**

**File:** scripts/queries.sql

**Example Queries:**

-- Show all records

SELECT \* FROM sales;

-- Total sales by region

SELECT Region, SUM(Sales) AS TotalSales

FROM sales

GROUP BY Region;

-- Top 3 highest sales transactions

SELECT \* FROM sales

ORDER BY Sales DESC

LIMIT 3;

-- Total sales by product category

SELECT Category, SUM(Sales) AS CategorySales

FROM sales

GROUP BY Category;

**Purpose:**  
These queries analyze the sales data.

**6️. Running the Queries**

**Script:** scripts/run\_queries.py

**Key Code:**

import sqlite3 # it loads sqlite lib and connects to db

import os # used to creat file paths

“”” Get absolute path to db( why did I use absolute path here is there was an error at 1st used relative path (db\_path = os.path.join("..", "db", "sales.db")which popped an error saying as “unable to open db file”. Hence, googled and check for alternative, it gave the absolute path version to avoid this error”””

# I understood using absolute path we can run the code in any terminal : which creates the full path

# Get the absolute path to the project root folder

# \_\_file\_\_ = path to this Python file (run\_queries.py)

# os.path.abspath(\_\_file\_\_) = full path to this file

# os.path.dirname() = go up one folder (from scripts/)

# another os.path.dirname() = go up again to reach project root (sales\_pipeline/)

base\_dir = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# create full path to the database file: db/sales.db

db\_path = os.path.join(base\_dir, "db", "sales.db")

conn = sqlite3.connect(db\_path) # connecting to db

# create full path to the SQL file: scripts/queries.sql

queries\_file = os.path.join(base\_dir, "scripts", "queries.sql")

with open(queries\_file, "r") as f:

    sql\_script = f.read()

# Splits the entire SQL text into separate queries wherever a semicolon (;) is found

# **q.strip()**  
Removes any extra spaces

# **if q.strip()**  
Ensures that only non-empty queries are included

queries = [q.strip() for q in sql\_script.split(";") if q.strip()]

“”” Used Loop to execute each query one by one: queries is a list of SQL statements.

enumerate() is a function, it creates index(position). start=1 means the numbering will begin at 1”””

for i, query in enumerate(queries, start=1): # I is the query number as 1,2,3

    print(f"\n--- Query {i} ---")

    cursor = conn.execute(query)

# Get column names from the result

columns = [description[0] for description in cursor.description]

    rows = cursor.fetchall()

    print(columns)

    for row in rows:

        print(row)

# Close the database connection

conn.close()

print("\n✅ All queries executed.")

**Output:**  
✅ Prints each query’s results, e.g.:

--- Query 1 ---

['OrderID', 'Product', 'Category', 'Sales', 'Region', 'OrderDate']

(1001, 'Laptop', 'Electronics', 1200, 'North', '2023-01-10')

--- Query 2 ---

['Category', 'CategorySales']

('Electronics', 2600)

('Stationery', 25)

**Effect:**  
Shows all records and aggregates as designed.

**7️.End Result**

**Outcome:**

* Data pipeline **automatically loads CSV data** into SQLite.
* **All queries run in sequence** and produce readable outputs.

✅ **Project Completed Successfully**