Walmart – sales analysis

Table of contents.

S.no.	Contents
1.	Introduction.
2.	Problem statement.
3.	Tools used.
4.	Overview of python.
5.	Overview of SQL.
6.	Setting up the environment for Walmart – sales analysis.
7.	Steps to load the data into python environment.
8.	Data exploration.
9.	Data manipulation in python.
10.	Connecting Python to MySQL Workbench and Loading the Dataset.
11.	Data Analysis Using SQL.
12.	Retrieving Basic Information About the Dataset Using SQL Queries.
13.	Business Problems.
14.	Conclusion.

1.Introduction:

I am Maria Monisha, a data analyst with a keen interest in uncovering valuable insights from data. This documentation highlights my Walmart Sales Analysis project, which I undertook to showcase and further hone my skills in data analysis. In this project, I utilized Python and SQL to analyze Walmart's sales data, exploring trends, patterns, and insights that can inform strategic decisions.

The primary aim of this project was to gain a deeper understanding of Walmart's sales performance by leveraging data analysis techniques. The process involved cleaning and preprocessing the data, conducting exploratory data analysis, visualizing key metrics, and extracting actionable insights. Python served as a robust tool for data manipulation and visualization, while SQL enabled efficient querying and data retrieval.

Through this project, I aimed to demonstrate my proficiency in handling real-world datasets and applying analytical tools to solve complex business problems. The analysis not only provided valuable insights but also reinforced my expertise in Python and SQL, making it a significant step in my journey as a data analyst.

1.1 Overview of Walmart:

Walmart Inc. is a leading American multinational retail corporation, founded in 1962 by Sam and James "Bud" Walton in Rogers, Arkansas, and headquartered in Bentonville, Arkansas. It operates 10,586 stores and clubs across 24 countries under 46 brand names as of October 2022.

Walmart is the world's largest company by revenue and the largest private employer, with 2.1 million employees globally. Its business includes hypermarkets, discount department stores, grocery stores, and Sam's Club retail warehouses.
While Walmart has seen success in countries like the United States, Canada, and China, its ventures in Germany, Japan, and South Korea were less successful. The company remains a leader in the retail industry, offering affordable products and services to customers worldwide.

2. Problem Statement:

Walmart generates extensive sales data across its global operations, but deriving actionable insights from this data can be challenging. The lack of efficient analysis may lead to missed opportunities in identifying sales trends, optimizing inventory, and understanding customer preferences. This project aims to resolve these challenges by using Python and SQL to analyse Walmart's sales data, uncover key patterns, and provide data-driven recommendations to improve operational efficiency, enhance customer satisfaction, and drive revenue growth.

3. Tools Used:

1.Dataset Source

- **Kaggle:**
  Used to obtain the Walmart sales dataset for analysis.

2.Programming Language

- **Python:**
  Employed for data exploration, manipulation, and visualization.

3.Libraries

- **Pandas:**
  For data manipulation and analysis.

- **Kaggle:**
  To access and manage the dataset directly from the Kaggle platform.

- **PyMySQL:**
  To connect Python with the SQL database.

- **SQLAlchemy:**
  For working with SQL databases using Python.

- **Create Engine:**
  To establish database connections and execute queries.

4.Database Querying

- **SQL:**
  Used for retrieving and analysing insights from the dataset efficiently.

4. Overview of Python:

Python is an interpreted, high-level, and object-oriented programming language with a focus on simplicity and readability. Its dynamic semantics and versatile design make it ideal for a variety of applications, including web development, automation, machine learning, and data analysis. Python's extensive standard library, combined with its ability to integrate with other tools and technologies, has earned it widespread popularity among developers and analysts.

Python is particularly well-suited for data analysis due to its robust ecosystem of libraries and tools, such as:

Pandas: For data manipulation and analysis.

NumPy: For numerical computations and handling large datasets efficiently.

Matplotlib and Seaborn: For data visualization and creating insightful charts and graphs.

Scikit-learn: For machine learning and predictive analytics.

Python's simplicity and flexibility allow analysts to clean, manipulate, and visualize data with ease. Its ability to handle structured and unstructured data, along with support for diverse file formats, makes it highly effective for data-driven tasks.

Python empowers data analysts in the following ways:

Efficiency: Automates repetitive tasks, enabling analysts to focus on deriving insights.

Visualization: Creates compelling visualizations to communicate findings effectively.

Integration: Connects seamlessly with databases, APIs, and other tools for data extraction and analysis.

Scalability: Handles large datasets and complex computations with ease.

Community Support: A large and active community ensures access to extensive resources, tutorials, and solutions to common challenges.
Overall, Python's ease of use, versatility, and powerful libraries make it an essential tool for modern data analysts to gain actionable insights from data.

5. Overview of SQL

SQL (Structured Query Language) is a powerful programming language designed to manage and interact with relational databases. It enables users to create, retrieve, update, and delete data efficiently. SQL is widely used for organizing and querying large datasets, making it an essential tool for data management across industries.

SQL is highly valued in data analysis due to its ability to:

Efficiently Query Data: Extract specific information from large datasets using clear and concise commands.

Data Manipulation: Filter, aggregate, and sort data to uncover patterns and trends.

Integration: Seamlessly connect with data visualization tools, programming languages like Python, and reporting software.

Scalability: Handle both small and enterprise-level datasets effectively.

Accessibility: Use standardized syntax, making it easy to learn and implement across various database systems like MySQL, PostgreSQL, and SQL Server.

How SQL is Used in Data Analysis ?

Data Extraction: Analysts use SQL queries to retrieve relevant data from large databases for analysis.

Data Cleaning: SQL helps clean and preprocess raw data by removing duplicates, handling missing values, and normalizing data.

Aggregations: It enables calculations such as totals, averages, and counts to summarize data meaningfully.

Joins and Relationships: SQL allows combining data from multiple tables, making it easier to analyse complex datasets.

Reporting and Insights: Generate reports and actionable insights by querying and visualizing data through SQL-compatible tools.

SQL's combination of simplicity, versatility, and performance makes it a cornerstone of data analysis, helping analysts derive valuable insights from structured datasets efficiently.

6. Setting Up the Environment for Walmart Sales Analysis:

To initiate the Walmart Sales Analysis project, I carefully executed the following steps:

Dataset Access:

- I created an API token on Kaggle, which generated a JSON file.
- This file was moved to my desktop and placed into a newly created .kaggle folder in the home directory.

Python Environment Configuration:

- The Python environment was set up in Visual Studio Code (VS Code).
- I installed the Kaggle library using the command pip install kaggle and verified the installation with pip list.

Dataset Download and Extraction:

- Utilizing the Kaggle API, I downloaded the dataset by executing Kaggle datasets download -d [dataset link] in the terminal.
- The downloaded file was then unzipped using appropriate terminal commands.

This structured setup ensured a robust foundation for the successful execution of the analysis.

7. Steps to Load the Dataset into Python Environment

Preparation:

Created a file named requirements.txt in Visual Studio Code to list and manage the required libraries for the project.

Installing and Verifying Pandas:

- Installed the Pandas library using the command:
- pip install pandas
- Verified the installation by importing the library with:
- import pandas as pd
- Checked the installed version using:
- print(pd.__version__)

Loading the Dataset:

Loaded the Walmart dataset into a Pandas Data Frame using the following command:

```
df=pd.read_csv(r'C:\Users\maria\OneDrive\Desktop\Project
Walmart\unzipped\Walmart.csv', encoding_errors='ignore')
```

Explanation:

- pd.read_csv() is a function to read CSV files and load them into a Data Frame.
- The r before the file path ensures the path is treated as a raw string,
- avoiding issues with backslashes.
- encoding_errors='ignore' prevents potential errors caused by encoding issues during file reading.

These steps reflect my systematic efforts to establish a reliable environment for data exploration and analysis.

```
[12]
# IMPORTING DEPENENCIES
import pandas as pd

[13]
print(pd.__version__)

...  2.2.3

[14]
#Loading the data

df = pd.read_csv(r'C:\Users\maria\OneDrive\Desktop\Project - Walmart\unzipped\Walmart.csv', encoding_errors='ignore')
```

8.Data Exploration:

Initial Exploration in MS Excel:

Opened the dataset in MS Excel for preliminary review and identified key issues to address for efficient analysis:

- **Dollar Sign Removal:** Found that the dollar sign in monetary values needed to be removed to enable arithmetic calculations.
- **Creating a New Column:** Realized the need to create a new column representing the total cost for better insights.
- **Dataset Size:** Noted that the dataset contains approximately 10,000 rows.

Exploration in Python:

Began detailed exploration using Python by executing the following commands:

df.shape: Displays the dimensions of the dataset (number of rows and columns).

```
[15]
df.shape

...  (10051, 11)
```

df.head(): Provides a preview of the first few rows of the dataset to understand its structure.

```
[16]  df.head()
```

Python

	invoice_id	Branch	City	category	unit_price	quantity	date	time	payment_method	rating	profit_margin
0	1	WALM003	San Antonio	Health and beauty	$74.69	7.0	05/01/19	13:08:00	Ewallet	9.1	0.48
1	2	WALM048	Harlingen	Electronic accessories	$15.28	5.0	08/03/19	10:29:00	Cash	9.6	0.48
2	3	WALM067	Haltom City	Home and lifestyle	$46.33	7.0	03/03/19	13:23:00	Credit card	7.4	0.33
3	4	WALM064	Bedford	Health and beauty	$58.22	8.0	27/01/19	20:33:00	Ewallet	8.4	0.33
4	5	WALM013	Irving	Sports and travel	$86.31	7.0	08/02/19	10:37:00	Ewallet	5.3	0.48

df.describe(): Generates summary statistics for numerical columns, including count, mean, and standard deviation.

```
[17]  df.describe()
```

Python

	invoice_id	quantity	rating	profit_margin
count	10051.000000	10020.000000	10051.000000	10051.000000
mean	5025.741220	2.353493	5.825659	0.393791
std	2901.174372	1.602658	1.763991	0.090669
min	1.000000	1.000000	3.000000	0.180000
25%	2513.500000	1.000000	4.000000	0.330000
50%	5026.000000	2.000000	6.000000	0.330000
75%	7538.500000	3.000000	7.000000	0.480000
max	10000.000000	10.000000	10.000000	0.570000

df.info(): Provides an overview of the dataset, including data types and memory usage.

```
[18]  df.info()
```

Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10051 entries, 0 to 10050
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   invoice_id      10051 non-null  int64  
 1   Branch          10051 non-null  object 
 2   City            10051 non-null  object 
 3   category        10051 non-null  object 
 4   unit_price      10020 non-null  object 
 5   quantity        10020 non-null  float64
 6   date            10051 non-null  object 
 7   time            10051 non-null  object 
 8   payment_method  10051 non-null  object 
 9   rating          10051 non-null  float64
 10  profit_margin   10051 non-null  float64
dtypes: float64(3), int64(1), object(7)
memory usage: 863.9+ KB
```

df.duplicated().sum(): Identifies the number of duplicate rows in the dataset.

```
[19]  df.duplicated().sum()                                                    Python
...   np.int64(51)
```

df.isnull().sum(): Detects the number of missing values in each column.

```
[20]  df.isnull().sum()                                                        Python
...   invoice_id         0
      Branch             0
      City               0
      category           0
      unit_price        31
      quantity          31
      date               0
      time               0
      payment_method     0
      rating             0
      profit_margin      0
      dtype: int64
```

These steps demonstrate my methodical approach to data exploration, leveraging both MS Excel for initial insights and Python for in-depth analysis to prepare the dataset for further processing.

9. Data Manipulation in Python

Removing Duplicate Records:

- Identified duplicate rows in the dataset and removed them using: df.drop_duplicates(inplace=True)
- Verified the dataset size after removal using df.shape, which showed a reduction in the number of rows.
- Ensured no duplicates remained by executing df.duplicated().sum(), which returned 0, confirming all duplicates were eliminated.

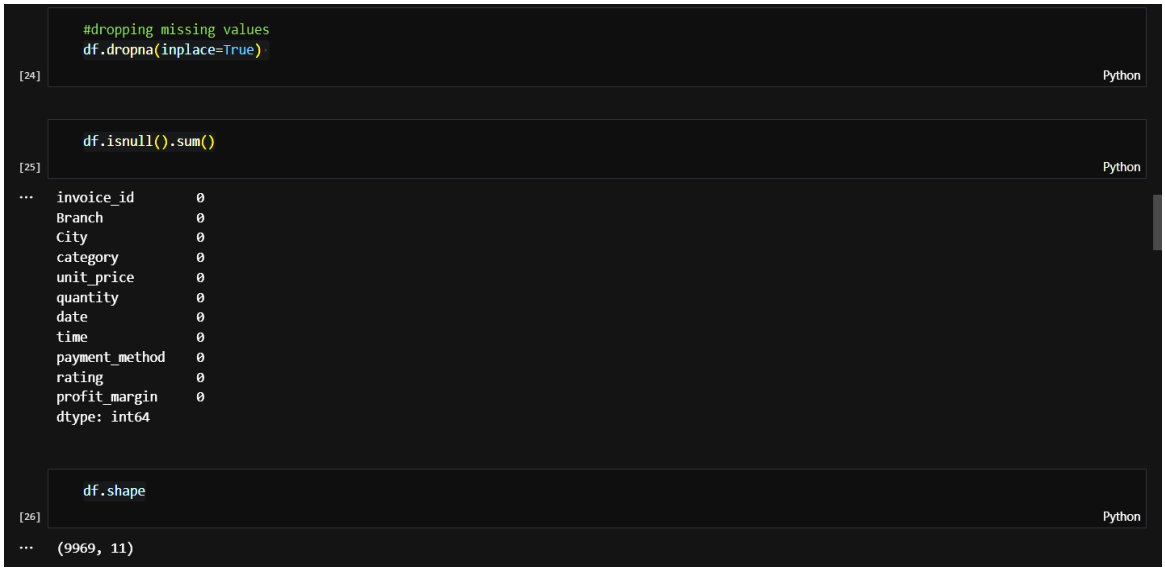
```
[21]  df.drop_duplicates(inplace=True)                                         Python

[22]  df.duplicated().sum()                                                    Python
...   np.int64(0)

[23]  df.shape                                                                 Python
...   (10000, 11)
```

Handling Missing Values:

- Analysed missing values in each column using df.isnull().sum() and identified that the "quantity" and "unit_price" columns contained null values.
- Since accurate imputation was not feasible, null values were removed with: df.dropna(inplace=True)
- Verified the absence of missing values again using df.isnull().sum() (all columns returned 0 null values).
- Checked the dataset size reduction after removal using df.shape, which showed
- the dataset shrinking from (10,000, 11) to (9,969, 11).



Correcting Data Types:

- Explored the data types of all columns using df.dtypes and found that the "unit_price" column was incorrectly stored as object instead of float.
- Attempted to convert the column to float using df['unit_price'].astype(float), but encountered an error due to dollar signs in the values.

- Resolved this issue by removing the dollar signs and converting the column to float with the following command:
  df['unit_price'] = df['unit_price'].str.replace('$', '').astype(float)

- Verified the change by checking the first few rows using df.head() (dollar signs were removed) and confirmed the data type change using df.info() (the "unit_price" column now displayed as float).

```
[27]  df.dtypes                                                    Python

      invoice_id          int64
      Branch             object
      City               object
      category           object
      unit_price         object
      quantity          float64
      date               object
      time               object
      payment_method     object
      rating            float64
      profit_margin     float64
      dtype: object

[28]  df['unit_price'].astype(float)                               Python

      ---------------------------------------------------------------------------
      ValueError                                Traceback (most recent call last)
      Cell In[28], line 1
      ----> 1 df['unit_price'].astype(float)
```

```
[29]  df['unit_price'].str.replace('$','').astype(float)           Python

      0       74.69
      1       15.28
      2       46.33
      3       58.22
      4       86.31
              ...
      9995    37.00
      9996    58.00
      9997    52.00
      9998    79.00
      9999    62.00
      Name: unit_price, Length: 9969, dtype: float64

      + Code   + Markdown

[30]  df.head()                                                    Python
```

	invoice_id	Branch	City	category	unit_price	quantity	date	time	payment_method	rating	profit_margin
0	1	WALM003	San Antonio	Health and beauty	$74.69	7.0	05/01/19	13:08:00	Ewallet	9.1	0.48
1	2	WALM048	Harlingen	Electronic accessories	$15.28	5.0	08/03/19	10:29:00	Cash	9.6	0.48
2	3	WALM067	Haltom City	Home and lifestyle	$46.33	7.0	03/03/19	13:23:00	Credit card	7.4	0.33
3	4	WALM064	Bedford	Health and beauty	$58.22	8.0	27/01/19	20:33:00	Ewallet	8.4	0.33
4	5	WALM013	Irving	Sports and travel	$86.31	7.0	08/02/19	10:37:00	Ewallet	5.3	0.48

```
[31]  df['unit_price']=df['unit_price'].str.replace('$','').astype(float)
                                                                          Python

[32]  df.head()
                                                                          Python
```

	invoice_id	Branch	City	category	unit_price	quantity	date	time	payment_method	rating	profit_margin
0	1	WALM003	San Antonio	Health and beauty	74.69	7.0	05/01/19	13:08:00	Ewallet	9.1	0.48
1	2	WALM048	Harlingen	Electronic accessories	15.28	5.0	08/03/19	10:29:00	Cash	9.6	0.48
2	3	WALM067	Haltom City	Home and lifestyle	46.33	7.0	03/03/19	13:23:00	Credit card	7.4	0.33
3	4	WALM064	Bedford	Health and beauty	58.22	8.0	27/01/19	20:33:00	Ewallet	8.4	0.33
4	5	WALM013	Irving	Sports and travel	86.31	7.0	08/02/19	10:37:00	Ewallet	5.3	0.48

```
[33]  df.info()
                                                                          Python

<class 'pandas.core.frame.DataFrame'>
Index: 9969 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   invoice_id      9969 non-null   int64
 1   Branch          9969 non-null   object
 2   City            9969 non-null   object
 3   category        9969 non-null   object
 4   unit_price      9969 non-null   float64
 5   quantity        9969 non-null   float64
 6   date            9969 non-null   object
 7   time            9969 non-null   object
```

Exploring Column Structure:

- Retrieved and reviewed the names of all existing columns using df.columns to ensure clarity about the dataset structure.

```
[34]  df.columns
                                                                          Python

Index(['invoice_id', 'Branch', 'City', 'category', 'unit_price', 'quantity',
       'date', 'time', 'payment_method', 'rating', 'profit_margin'],
      dtype='object')
```

Creating a Derived Column:

- Added a new column, "Total_cost," to calculate the total cost for each row by multiplying the "quantity" and "unit_price" columns using:

- df['Total_cost'] = df['quantity'] * df['unit_price']

- Verified the successful creation of the new column by viewing the dataset with df.head().

```
[35]  df['Total_cost']=df['quantity']*df['unit_price']                    Python

[36]  df.head()                                                           Python
```

	invoice_id	Branch	City	category	unit_price	quantity	date	time	payment_method	rating	profit_margin	Total_cost
0	1	WALM003	San Antonio	Health and beauty	74.69	7.0	05/01/19	13:08:00	Ewallet	9.1	0.48	522.83
1	2	WALM048	Harlingen	Electronic accessories	15.28	5.0	08/03/19	10:29:00	Cash	9.6	0.48	76.40
2	3	WALM067	Haltom City	Home and lifestyle	46.33	7.0	03/03/19	13:23:00	Credit card	7.4	0.33	324.31
3	4	WALM064	Bedford	Health and beauty	58.22	8.0	27/01/19	20:33:00	Ewallet	8.4	0.33	465.76
4	5	WALM013	Irving	Sports and travel	86.31	7.0	08/02/19	10:37:00	Ewallet	5.3	0.48	604.17

Final Verification:

Performed a final check on the dataset using df.info() to confirm:

- ❖ All columns had the correct data types.
- ❖ There were no null values remaining in the dataset.

Ensured the dataset was clean, consistent, and ready for further analysis or storage.

These steps highlight my methodical approach to data cleaning and preparation using Python. By ensuring the dataset was free of duplicates, null values, and inconsistencies, I have effectively prepared it for further analysis. The next phase involves uploading this manipulated data into MySQL Workbench for advanced querying and visualization.

10. Connecting Python to MySQL Workbench and Loading the Dataset:

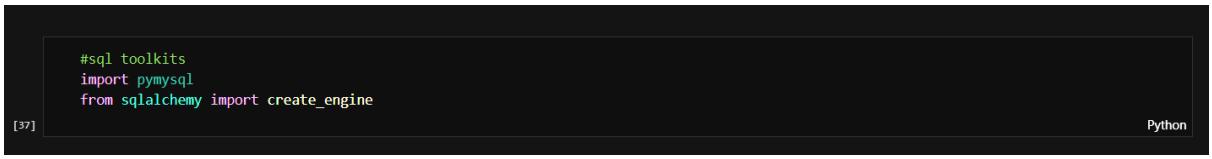
Launching MySQL Workbench:

- ❖ Started MySQL Workbench to prepare for the database connection and ensured it was running correctly.

Installing Necessary SQL Toolkits:

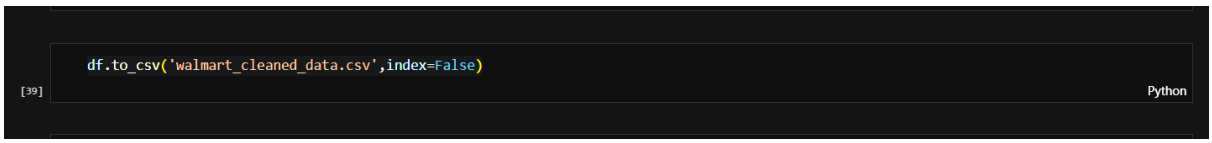
Installed essential Python libraries to facilitate the connection between Python and MySQL using the following commands:

- ❖ pip install pymysql
- ❖ pip install sqlalchemy
- ❖ Imported the required modules:
    - import pymysql
    - from sqlalchemy import create_engine
- ❖ Verified successful installation using pip list.



Exporting Cleaned Data to CSV Format:

- ❖ Saved the cleaned dataset as a CSV file to enable easy transfer to MySQL using the following code:
- ❖ df.to_csv('walmart_cleaned_data.csv', index=False)



Understanding MySQL Credentials:

Familiarized myself with essential MySQL credentials, including:

- Host
- Port
- User
- Password

These credentials were required for establishing a connection between Python and MySQL.

Creating a Connection Engine:

- Created a connection engine using SQLAlchemy to link Python with the MySQL database. The code used was:

- engine_mysql=create_engine("mysql+pymysql://user:password@localhost:3306/dbname")

- Replaced user, password, and db name with appropriate MySQL credentials.

```
#mysql_connection

engine_mysql = create_engine("mysql+pymysql://root:Monisha%401709@localhost:3306/walmart")
```

Testing the Connection:

- Verified the connection to MySQL using a try-except block:

- try:
  engine_mysql
  print("connection successfully done to mysql")
  except:
  print("unable to connect")

- This ensured that the connection was established and errors were handled gracefully.

```
#mysql_connection

engine_mysql = create_engine("mysql+pymysql://root:Monisha%401709@localhost:3306/walmart")


try:
    engine_mysql
    print("connection succefully done to mysql")

except:
    print("unable to connect")
```

[41] Python

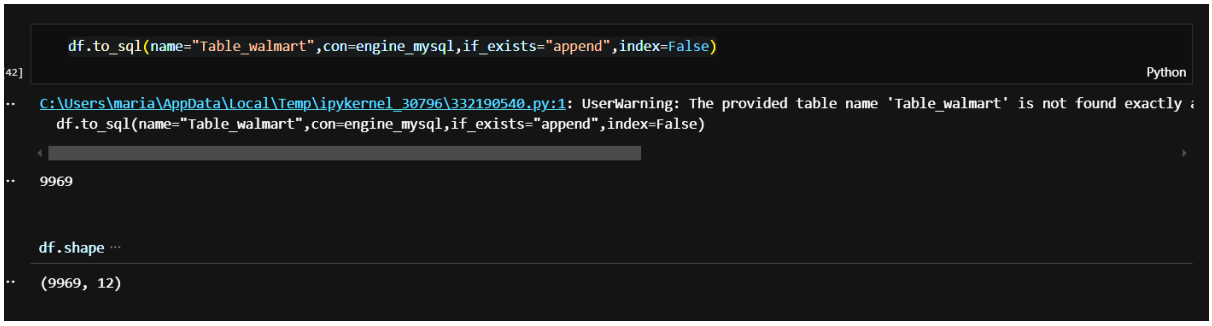
```
connection succefully done to mysql
```

Loading the Dataset into MySQL:

- Transferred the cleaned dataset to MySQL and created a new table using:

  df.to_sql(name="Table_walmart",con=engine_mysql,if_exists="append", index=False)

- ❖ Specified the table name as "Table_walmart" and set if_exists="append" to add the data without overwriting any existing tables.



This workflow demonstrates my systematic approach to integrating Python with MySQL. By ensuring all prerequisites were met and using professional methods for data transfer, I successfully loaded the cleaned dataset into MySQL Workbench for further analysis.

11. Data Analysis Using SQL:

Exploratory Analysis:

Understanding the structure and contents of the dataset through queries like SELECT *, LIMIT, or DESCRIBE.

Statistical Insights:

Using SQL functions like COUNT, AVG, SUM, etc., to derive meaningful metrics.

Data Validation:

Ensuring the dataset is clean and ready for more advanced analysis.
By performing these tasks, you're actively analyzing the dataset using SQL.

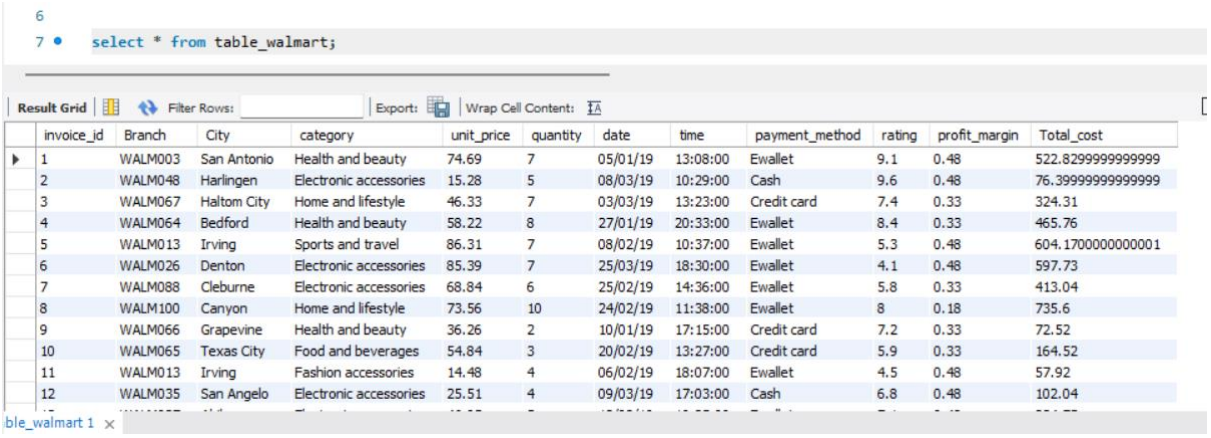
12. Retrieving Basic Information About the Dataset Using SQL Queries:

1.To view all the records in the dataset:

Query:

```
SELECT  * FROM table_walmart ;
```

Explanation: This retrieves all the rows and columns from the table, giving an overview of the dataset's structure and content.



```
6
7 • select * from table_walmart;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

invoice_id	Branch	City	category	unit_price	quantity	date	time	payment_method	rating	profit_margin	Total_cost
1	WALM003	San Antonio	Health and beauty	74.69	7	05/01/19	13:08:00	Ewallet	9.1	0.48	522.8299999999999
2	WALM048	Harlingen	Electronic accessories	15.28	5	08/03/19	10:29:00	Cash	9.6	0.48	76.39999999999999
3	WALM067	Haltom City	Home and lifestyle	46.33	7	03/03/19	13:23:00	Credit card	7.4	0.33	324.31
4	WALM064	Bedford	Health and beauty	58.22	8	27/01/19	20:33:00	Ewallet	8.4	0.33	465.76
5	WALM013	Irving	Sports and travel	86.31	7	08/02/19	10:37:00	Ewallet	5.3	0.48	604.1700000000001
6	WALM026	Denton	Electronic accessories	85.39	7	25/03/19	18:30:00	Ewallet	4.1	0.48	597.73
7	WALM088	Cleburne	Electronic accessories	68.84	6	25/02/19	14:36:00	Ewallet	5.8	0.33	413.04
8	WALM100	Canyon	Home and lifestyle	73.56	10	24/02/19	11:38:00	Ewallet	8	0.18	735.6
9	WALM066	Grapevine	Health and beauty	36.26	2	10/01/19	17:15:00	Credit card	7.2	0.33	72.52
10	WALM065	Texas City	Food and beverages	54.84	3	20/02/19	13:27:00	Credit card	5.9	0.33	164.52
11	WALM013	Irving	Fashion accessories	14.48	4	06/02/19	18:07:00	Ewallet	4.5	0.48	57.92
12	WALM035	San Angelo	Electronic accessories	25.51	4	09/03/19	17:03:00	Cash	6.8	0.48	102.04

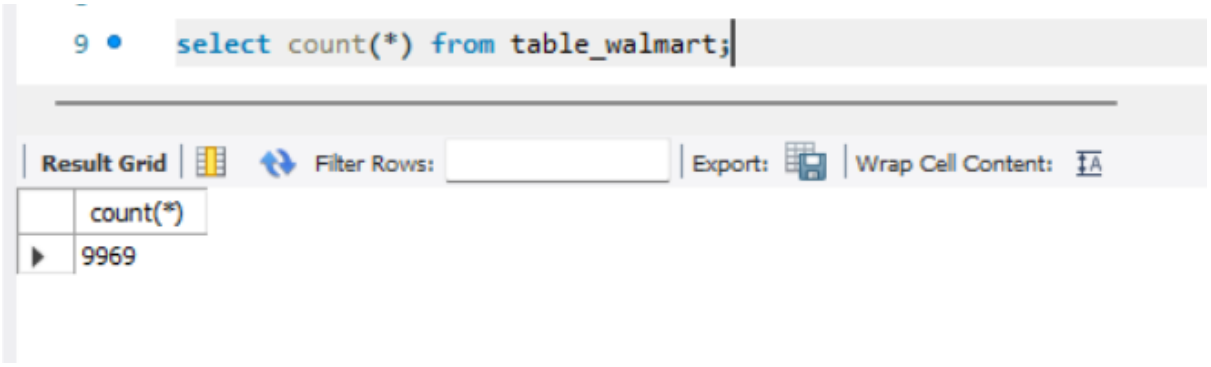
ble_walmart 1 ×

2.To count the total number of records in the dataset:

Query:

```
SELECT COUNT(*) FROM table_walmart;
```

Explanation: This returns the total number of rows (records) present in the table.



```
9 • select count(*) from table_walmart;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

count(*)
9969

3.To find all the unique payment methods used:

Query:

```
SELECT DISTINCT payment_method FROM table_walmart;
```

Explanation: This fetches a list of all distinct payment methods without duplicates.

```
10
11 •    select distinct payment_method from table_walmart;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

payment_method
Ewallet
Cash
Credit card

4.To count the number of transactions for each payment method:

Query:

```
SELECT payment_method, COUNT(*)
FROM table_walmart
GROUP BY payment_method;
```

Explanation: This groups the transactions by payment method and counts how many transactions were made for each.

```
12
13 •    select payment_method , count(*)
14      from table_walmart
15      group by payment_method;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

payment_method	count(*)
Ewallet	3881
Cash	1832
Credit card	4256

5.To count the number of unique branches in the dataset:

Query:

```
SELECT COUNT(DISTINCT branch) FROM table_walmart;
```

Explanation: This returns the count of unique branches recorded in the dataset.

```
16
17 •  select count(distinct branch) from table_walmart;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	count(distinct branch)
▶	100

6.To find the maximum quantity sold in a single transaction:

Query:

SELECT MAX(quantity) FROM table_walmart;

Explanation: This retrieves the highest quantity sold in a single transaction.

```
18
19 •  select max(quantity) from table_walmart;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	max(quantity)
▶	10

7.To find the minimum quantity sold in a single transaction:

Query:

SELECT MIN(quantity) FROM table_walmart;

Explanation: This retrieves the lowest quantity sold in a single transaction.

```
20
21 •  select min(quantity) from table_walmart;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	min(quantity)
▶	1

13. Business Problems :

1.Question:

What are the total number of transactions and quantity sold for each payment method, ordered by the highest number of transactions?

```
select payment_method,count(*) as no_of_transactions ,sum(quantity) as no_of_qty_sold from table_walmart
group by payment_method
order by no_of_transactions desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

payment_method	no_of_transactions	no_of_qty_sold
Credit card	4256	9567
Ewallet	3881	8932
Cash	1832	4984

Explanation:

This query groups the data by payment_method and calculates the total number of transactions and the total quantity sold for each method. It uses COUNT(*) to count transactions and SUM(quantity) to calculate the total quantity. The results are sorted in descending order by the number of transactions.

2.Question:

Which category in each branch has the highest average rating?

```
select * from
(
select branch , category , avg(rating) as avg_rating,
rank() over(partition by branch order by avg(rating) desc) as rank_
from table_walmart
group by 1,2
) as sub_query
where rank_ = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

branch	category	avg_rating	rank_
WALM001	Electronic accessories	7.45	1
WALM002	Food and beverages	8.25	1
WALM003	Sports and travel	7.5	1
WALM004	Food and beverages	9.3	1
WALM005	Health and beauty	8.366666666666667	1
WALM006	Fashion accessories	6.797058823529412	1
WALM007	Food and beverages	7.55	1
WALM008	Food and beverages	7.4	1
WALM009	Sports and travel	9.6	1
WALM010	Electronic accessories	9	1

Result 9

Explanation:

The subquery calculates the average rating (AVG(rating)) for each category within each branch and assigns a rank (RANK() OVER) based on the highest average rating. The outer query filters the top-ranked (rank_ = 1) categories for each branch.

3.Question:

Which day of the week has the most transactions for each branch, and what are the highest transaction counts across all branches?

```
92
93 •    select * from
94      (
95      select
96      branch,dayname(date_format(date,'%d/%m/%y')) as day_name,
97      count(*) as no_of_transactions,
98      rank() over(partition by branch order by branch,count(*) desc)
99      as rank_
100     from table_walmart
101     group by branch,day_name
102     ) as sub_query
103     where rank_ = 1
104     order by no_of_transactions desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

branch	day_name	no_of_transactions	rank_
WALM058	Wednesday	45	1
WALM009	Sunday	42	1
WALM069	Thursday	42	1
WALM074	Wednesday	41	1
WALM030	Wednesday	40	1
WALM082	Thursday	40	1
WALM038	Sunday	37	1
WALM029	Thursday	36	1
WALM046	Wednesday	35	1
WALM075	Friday	35	1
WALM084	Tuesday	35	1
WALM087	Saturday	35	1
WALM003	Tuesday	33	1
WALM035	Saturday	32	1
WALM050	Sunday	32	1

Result 10

Explanation:

The query calculates the total number of transactions per day of the week (DAYNAME) for each branch. Using RANK(), it identifies the day with the most transactions (rank_ = 1) for each branch. The results are ordered by the number of transactions in descending order.

4.Question:

What is the total quantity of items sold for each payment method?

```
108
109 •   select payment_method , sum(quantity) as total_quantity
110     from table_walmart
111     group by payment_method;
112
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

payment_method	total_quantity
Ewallet	8932
Cash	4984
Credit card	9567

Explanation:

This query groups data by payment_method and calculates the total quantity of items sold (SUM(quantity)) for each method.

5.Question:

What is the minimum, maximum, and average ratings for each category in each city?

```
117
118 •    select city,category,
119      min(rating) as min_rating,
120      max(rating) as max_rating,
121      avg(rating) as avg_rating
122      from table_walmart
123      group by city,category;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

city	category	min_rating	max_rating	avg_rating
San Antonio	Health and beauty	5	9.1	7.05
Harlingen	Electronic accessories	9.6	9.6	9.6
Haltom City	Home and lifestyle	3	9.5	6.227777777777778
Bedford	Health and beauty	6.1	9.3	8.15
Irving	Sports and travel	5.3	5.3	5.3
Denton	Electronic accessories	4.1	9	6.7
Cleburne	Electronic accessories	5.8	7.8	7.25
Canyon	Home and lifestyle	3	9	6.25
Grapevine	Health and beauty	7.2	7.2	7.2
Texas City	Food and beverages	5.5	5.9	5.7
Irving	Fashion accessories	3	9.8	6.206896551724138
San Angelo	Electronic accessories	3	7	5.8307692307692305
Abilene	Electronic accessories	7.1	8.8	7.966666666666666
San Angelo	Food and beverages	8.2	8.5	8.35
Lewisville	Health and beauty	5.5	5.7	5.6
Corpus Christi	Sports and travel	4.5	4.5	4.5

Result 12

Explanation:

The query calculates the minimum, maximum, and average ratings (MIN(rating), MAX(rating), AVG(rating)) for each city and category combination. It groups the data by both columns.

6.Question:

What is the total revenue and profit for each category, and which category generates the highest profit?

```
127
128 •   select category,
129     sum(total_cost) as revenue,
130     sum(total_cost*profit_margin) as total_profit
131     from table_walmart
132     group by category
133     order by total_profit desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category	revenue	total_profit
Fashion accessories	489480.89999999997	192314.89320000037
Home and lifestyle	489250.06	192213.6380999999
Electronic accessories	78175.02999999998	30772.489499999978
Food and beverages	53471.28000000006	21552.862200000003
Sports and travel	52497.93000000002	20613.808199999996
Health and beauty	46851.17999999998	18671.7345

Explanation:

The query calculates the total revenue (SUM(total_cost)) and total profit (SUM(total_cost * profit_margin)) for each category. The results are grouped by category and sorted in descending order by total profit.

7.Question:

What is the most frequently used payment method in each branch?

```
138 •  with cte
139    as
140    (
141    select branch,
142    payment_method,
143    count(payment_method) as total_trans,
144    rank() over(partition by branch order by count(payment_method) desc) as rank_
145    from table_walmart
146    group by branch,payment_method
147    )
148    select * from cte where rank_ = 1;
149
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

branch	payment_method	total_trans	rank_
WALM001	Ewallet	45	1
WALM002	Ewallet	37	1
WALM003	Credit card	115	1
WALM004	Ewallet	44	1
WALM005	Ewallet	56	1
WALM006	Ewallet	50	1
WALM007	Ewallet	52	1
WALM008	Ewallet	39	1
WALM009	Credit card	139	1
WALM010	Ewallet	47	1
WALM011	Ewallet	39	1
WALM012	Ewallet	52	1
WALM013	Ewallet	44	1
WALM014	Ewallet	28	1
WALM015	Ewallet	57	1
WALM016	Ewallet	46	1

Result 14

Explanation:

A Common Table Expression (CTE) is used to calculate the number of transactions (COUNT(payment_method)) for each payment method in each branch. RANK() assigns a rank based on the highest transaction count. The main query filters the most frequently used payment method (rank_ = 1) for each branch.

8.Question:

How many invoices were generated in each branch during different shifts of the day?

```
154
155 •   select branch,count(*) as no_of_invoice,
156     case
157      when hour(time_format(time,'%T')) < 12 then 'Morning'
158      when hour(time_format(time,'%T')) between 12 and 17 then 'Afternoon'
159      else 'Evening'
160     end as shift
161     from table_walmart
162     group by branch,shift
163     order by branch,no_of_invoice desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

branch	no_of_invoice	shift
WALM001	36	Afternoon
WALM001	30	Evening
WALM001	8	Morning
WALM002	29	Afternoon
WALM002	21	Evening
WALM002	15	Morning
WALM003	95	Afternoon
WALM003	50	Morning
WALM003	41	Evening
WALM004	27	Afternoon
WALM004	24	Evening
WALM004	9	Morning
WALM005	35	Evening
WALM005	34	Afternoon
WALM005	15	Morning
WALM006	33	Afternoon

Result 15

Explanation:

The query categorizes transactions into shifts (Morning, Afternoon, Evening) based on the time of the transaction (TIME_FORMAT). It groups data by branch and shift and counts the number of invoices (COUNT(*)). Results are ordered by branch and invoice count.

9.Question:

Which branches experienced the highest revenue decrease from 2022 to 2023, and what are the top five branches with the steepest decline?

```
169 •  with revenue_2022_
170    as(select branch,sum(total_cost) as revenue from table_walmart
171    where year(date_format(date,'%d/%m/%y'))=2022
172    group by branch
173    order by branch asc
174    ),revenue_2023_
175    as(select branch,sum(total_cost) as revenue from table_walmart
176    where year(date_format(date,'%d/%m/%y'))=2023
177    group by branch
178    order by branch asc
179    )select ls.branch,ls.revenue as last_year_revenue,cs.revenue as current_year_revenue,
180    round(((ls.revenue-cs.revenue )/ls.revenue )*100,2) as revenue_decrease_ratio
181    from revenue_2022_ as ls
182    join revenue_2023_ as cs
183    on ls.branch = cs.branch
184    where ls.revenue > cs.revenue
185    order by revenue_decrease_ratio desc
186    limit 5 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

branch	last_year_revenue	current_year_revenue	revenue_decrease_ratio
WALM045	1731	647	62.62
WALM047	2581	1069	58.58
WALM098	2446	1030	57.89
WALM033	2099	931	55.65
WALM081	1723	850	50.67

Explanation:

Two CTEs (revenue_2022_ and revenue_2023_) calculate total revenue for each branch for the years 2022 and 2023, respectively. The main query calculates the revenue decrease ratio and filters branches where 2023 revenue is less than 2022 revenue. The results are ordered by the highest revenue decrease percentage, limited to the top five branches.

14. Conclusion for the Walmart SQL Project:

This project provided a comprehensive analysis of Walmart's business operations, covering key metrics such as transaction trends, product ratings, revenue, profitability, and customer behaviours. By leveraging advanced SQL queries, we extracted valuable insights that can help Walmart optimize its business strategies and improve overall performance.

Key findings include:

1. **Payment Methods**: The distribution of transactions and quantities sold across payment methods highlighted the most preferred payment option by customers, enabling targeted strategies to improve the customer experience.
2. **Top-Performing Categories**: Identifying the highest-rated categories for each branch allows Walmart to focus on enhancing its offerings and replicating success in other branches.
3. **Day-wise Trends**: Analysing the busiest days for each branch helped determine peak shopping days, providing insights for staffing and inventory planning.
4. **Branch and Shift Analysis**: Understanding the volume of invoices generated during different shifts in each branch ensures resource allocation during high-traffic periods.
5. **Profitability by Category**: Revealing revenue and profit contributions from each category guides Walmart in maximizing profitability by focusing on high-margin products.
6. **Year-over-Year Revenue Comparison**: Pinpointing branches with significant revenue decreases between 2022 and 2023 empowers Walmart to investigate and address underlying issues in these locations.

Through this project, SQL techniques such as aggregation, ranking, subqueries, Common Table Expressions (CTEs), and window functions were effectively utilized to extract actionable insights. These analyses empower Walmart to enhance customer satisfaction, boost profitability, and sustain long-term growth by aligning its strategies with data-driven decision-making.

Thank you.

Contact me through,

Email : hiremaria17@gmail.com

LinkedIn : https://www.linkedin.com/in/maria1709