Customer Behavior Analysis

Code Presentation

MARCH 31, 2025

SHOPEASY

SUNIL KUMAR G T

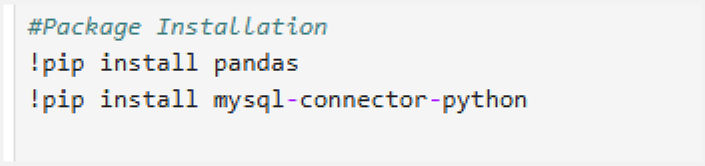
1. **Download CSV Files**

Download all the CSV files required for the project and keep them in same project folder

2. **Clean Datasets**

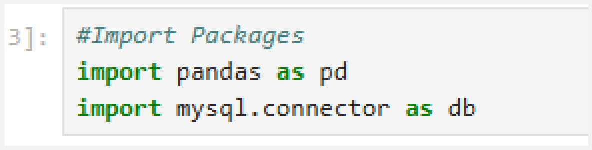
Clean all the six datasets as per the requirements:

- Install packages required for packages



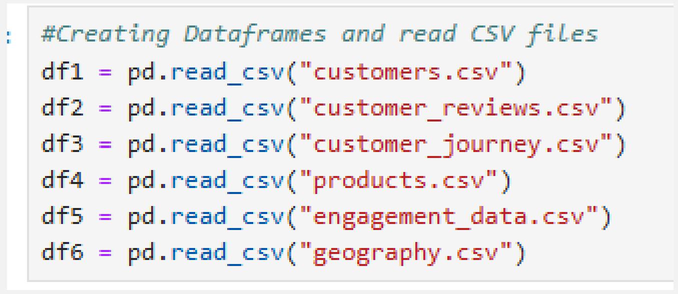
```
#Package Installation
!pip install pandas
!pip install mysql-connector-python
```

- Import Packages



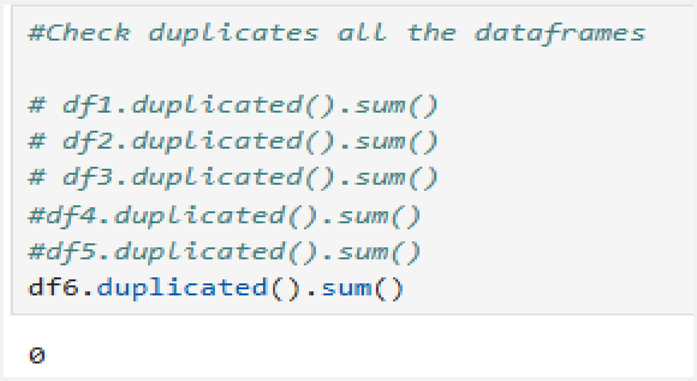
```
#Import Packages
import pandas as pd
import mysql.connector as db
```

- Create DataFrame to read all the CSV files



```
#Creating Dataframes and read CSV files
df1 = pd.read_csv("customers.csv")
df2 = pd.read_csv("customer_reviews.csv")
df3 = pd.read_csv("customer_journey.csv")
df4 = pd.read_csv("products.csv")
df5 = pd.read_csv("engagement_data.csv")
df6 = pd.read_csv("geography.csv")
```

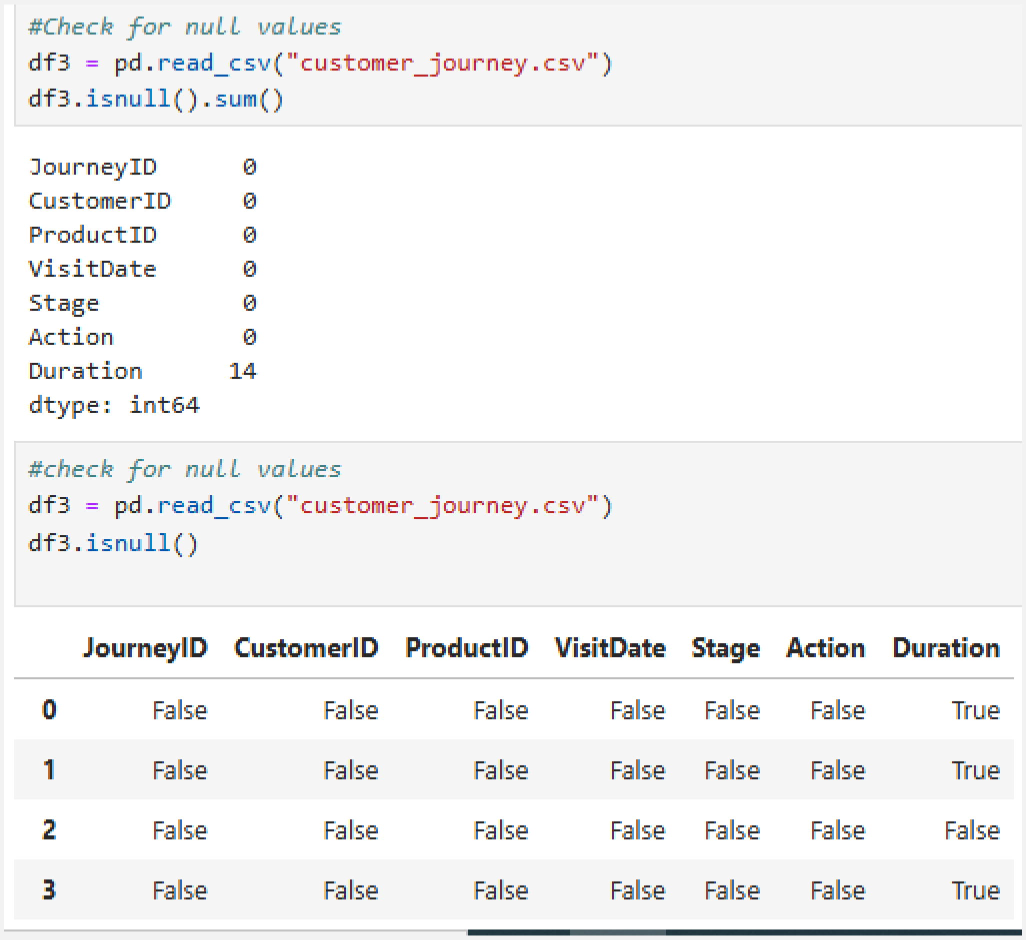
- Check for duplicates in all the dataframes, we can make sure there were no duplicates in all the dataframes



```
#Check duplicates all the dataframes

# df1.duplicated().sum()
# df2.duplicated().sum()
# df3.duplicated().sum()
#df4.duplicated().sum()
#df5.duplicated().sum()
df6.duplicated().sum()
```

```
0
```

- Check for null values listed in all the six dataframes



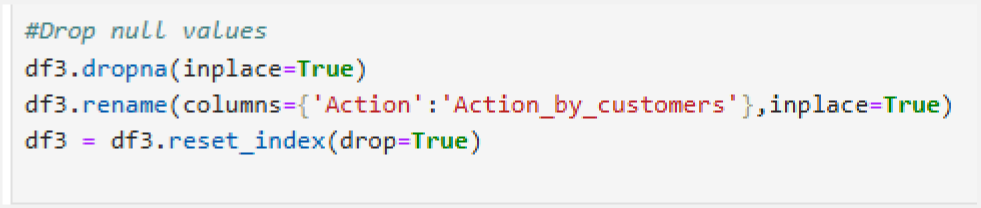
```
#Check for null values
df3 = pd.read_csv("customer_journey.csv")
df3.isnull().sum()
```

```
JourneyID      0
CustomerID     0
ProductID      0
VisitDate      0
Stage          0
Action         0
Duration      14
dtype: int64
```

```
#check for null values
df3 = pd.read_csv("customer_journey.csv")
df3.isnull()
```

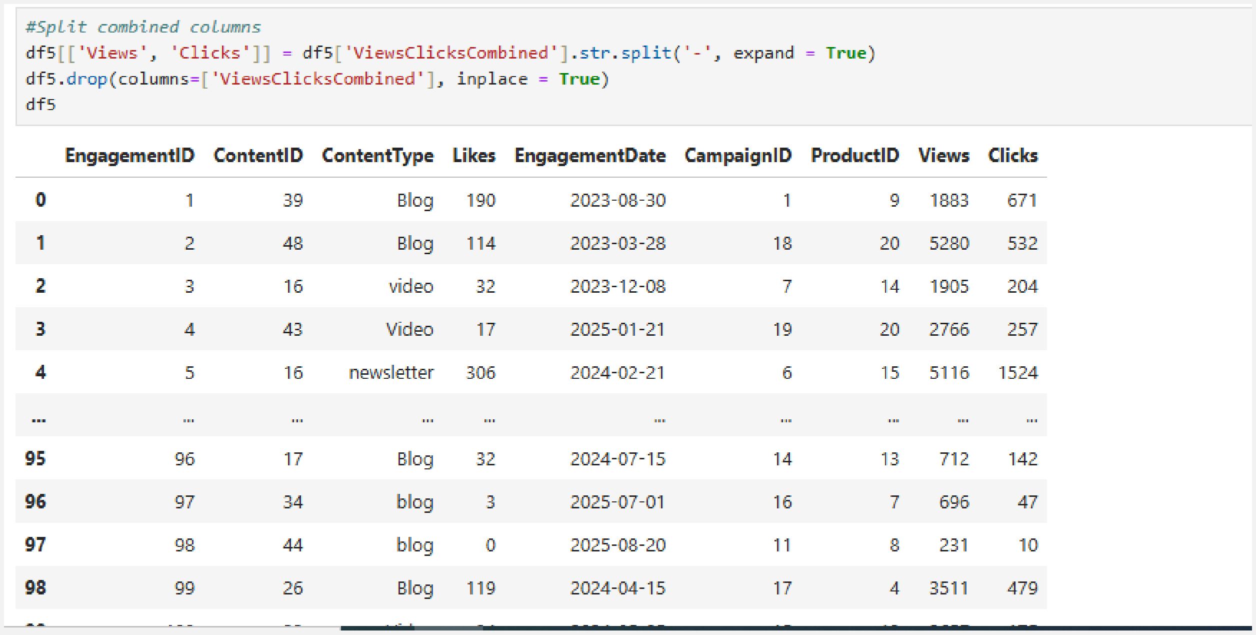
	JourneyID	CustomerID	ProductID	VisitDate	Stage	Action	Duration
0	False	False	False	False	False	False	True
1	False	False	False	False	False	False	True
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	True

- After identifying the null values, drop them in all the dataframes



```
#Drop null values
df3.dropna(inplace=True)
df3.rename(columns={'Action':'Action_by_customers'},inplace=True)
df3 = df3.reset_index(drop=True)
```

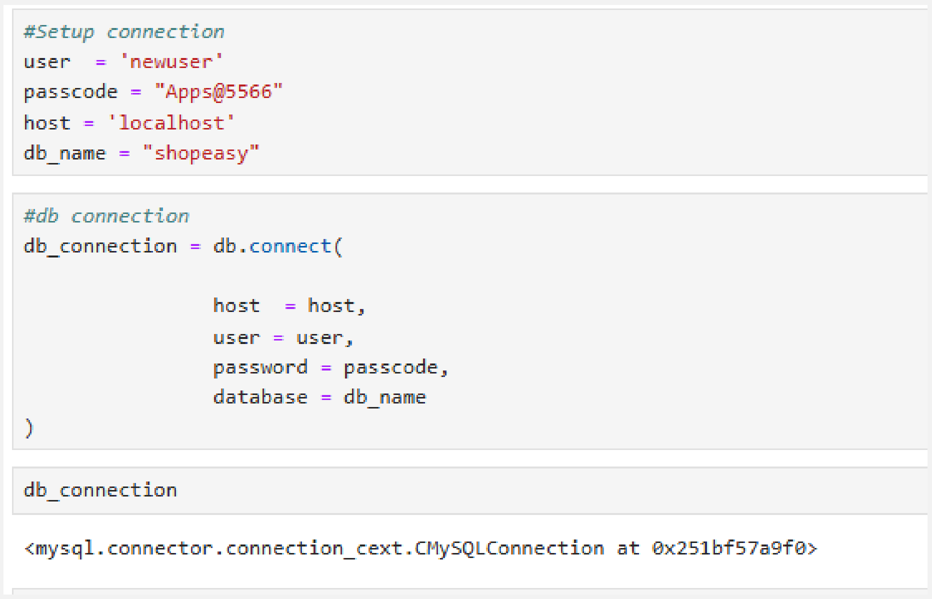
- Split combined header values if required for all the dataframes



```
#Split combined columns
df5[['Views', 'Clicks']] = df5['ViewsClicksCombined'].str.split('-', expand = True)
df5.drop(columns=['ViewsClicksCombined'], inplace = True)
df5
```

	EngagementID	ContentID	ContentType	Likes	EngagementDate	CampaignID	ProductID	Views	Clicks
0	1	39	Blog	190	2023-08-30	1	9	1883	671
1	2	48	Blog	114	2023-03-28	18	20	5280	532
2	3	16	video	32	2023-12-08	7	14	1905	204
3	4	43	Video	17	2025-01-21	19	20	2766	257
4	5	16	newsletter	306	2024-02-21	6	15	5116	1524
...	...	...	...	...	...	...	...	...	...
95	96	17	Blog	32	2024-07-15	14	13	712	142
96	97	34	blog	3	2025-07-01	16	7	696	47
97	98	44	blog	0	2025-08-20	11	8	231	10
98	99	26	Blog	119	2024-04-15	17	4	3511	479

- Once after cleaning up all the dataframes, establish connection with mysql datsource



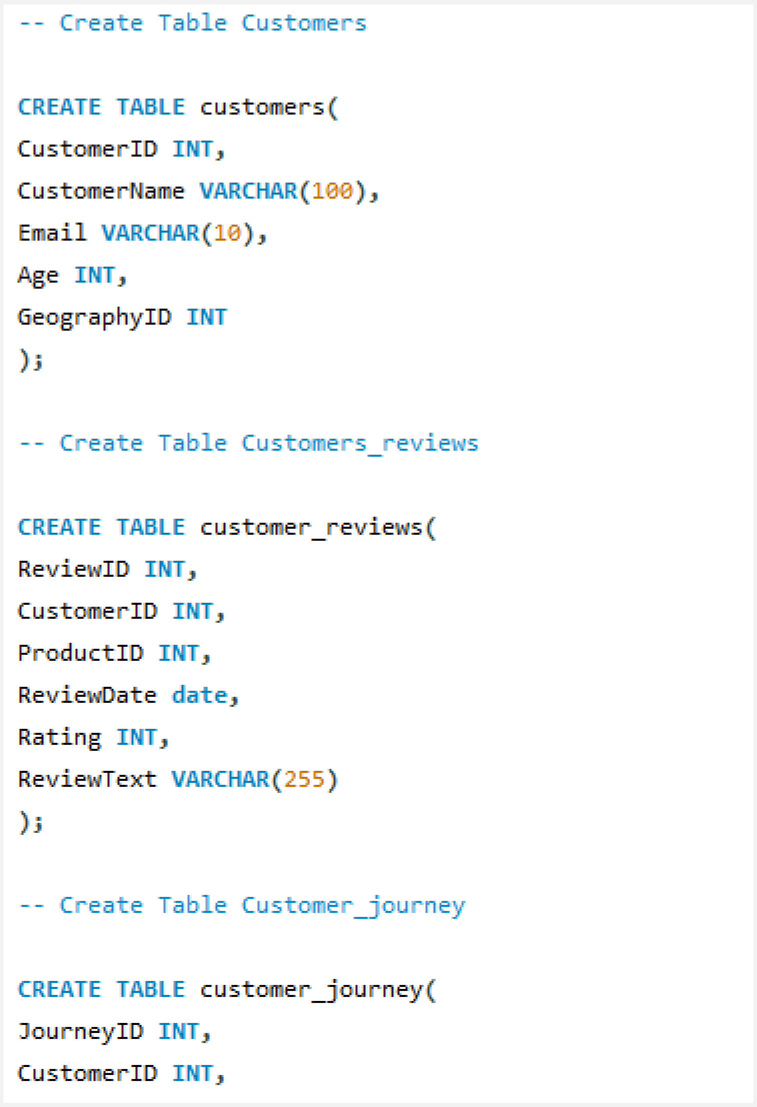
```
#Setup connection
user  = 'newuser'
passcode = "Apps@5566"
host = 'localhost'
db_name = "shopeasy"
```

```
#db connection
db_connection = db.connect(

                host  = host,
                user = user,
                password = passcode,
                database = db_name
)
```

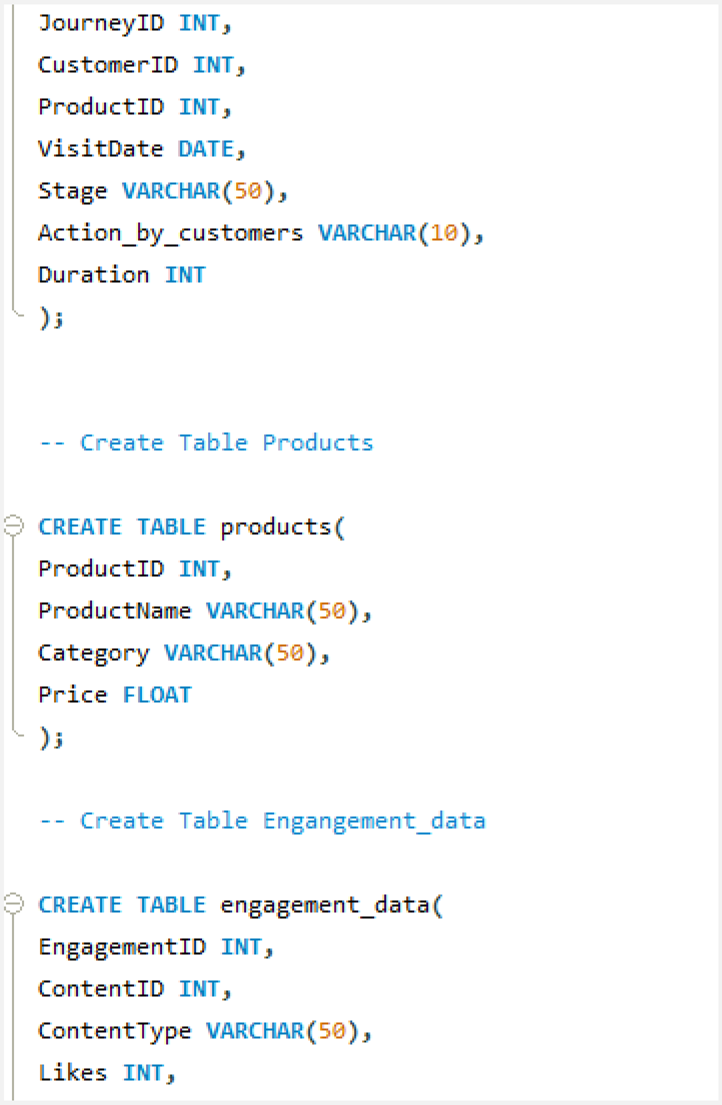
```
db_connection
```

```
<mysql.connector.connection_cext.CMySQLConnection at 0x251bf57a9f0>
```

- After setting up connection with shopeasy datasource, create six tables to load values for analysis

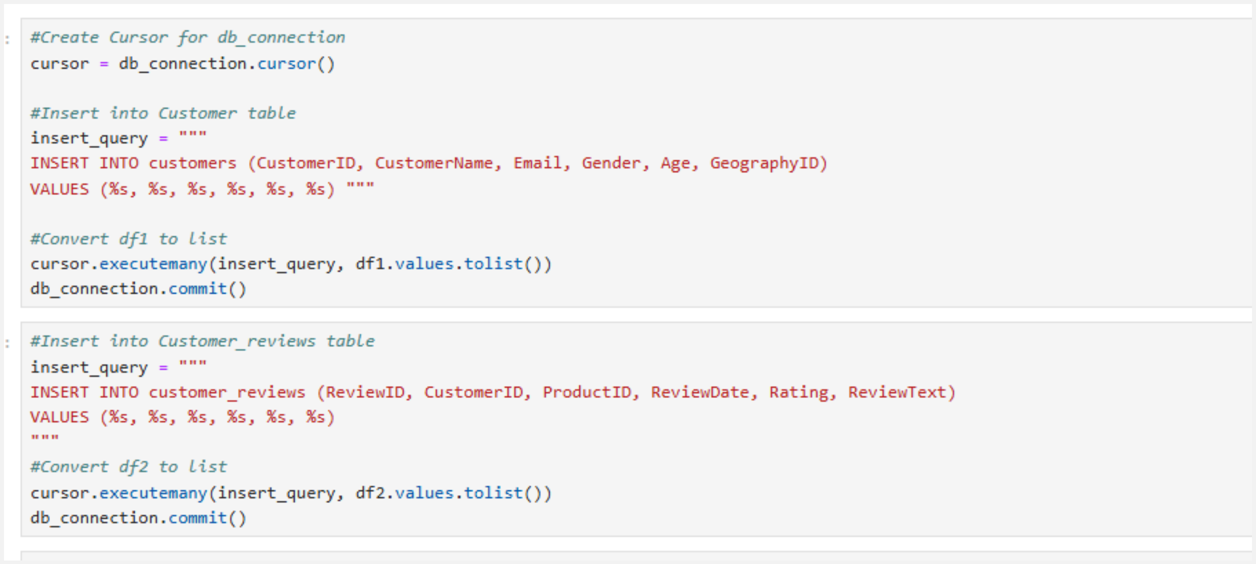


```sql
-- Create Table Customers

CREATE TABLE customers(
CustomerID INT,
CustomerName VARCHAR(100),
Email VARCHAR(10),
Age INT,
GeographyID INT
);

-- Create Table Customers_reviews

CREATE TABLE customer_reviews(
ReviewID INT,
CustomerID INT,
ProductID INT,
ReviewDate date,
Rating INT,
ReviewText VARCHAR(255)
);

-- Create Table Customer_journey

CREATE TABLE customer_journey(
JourneyID INT,
CustomerID INT,
```



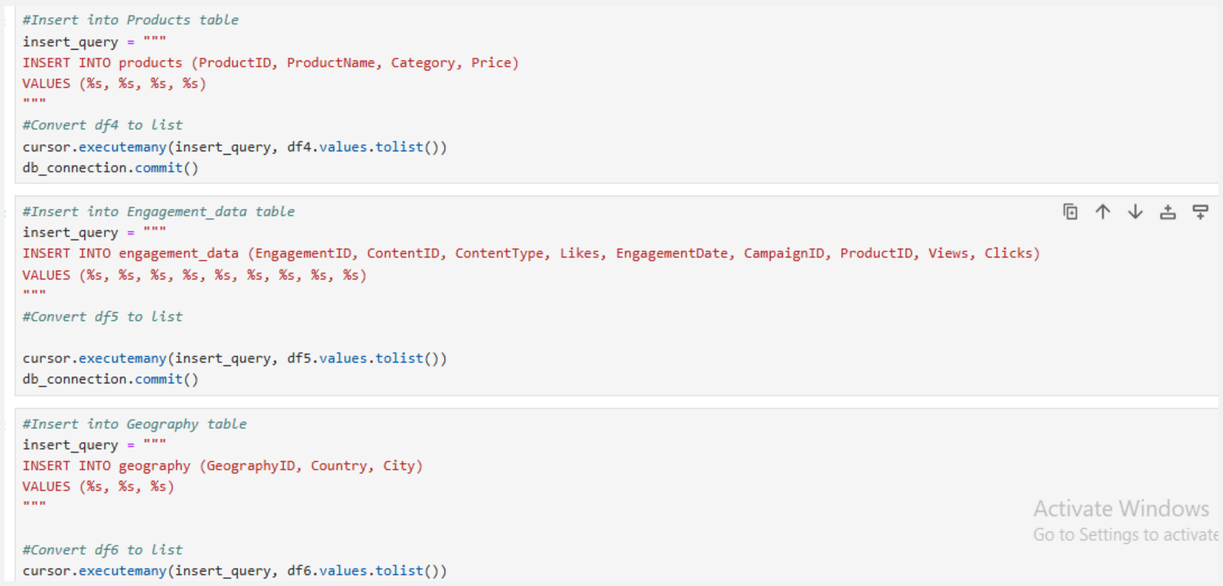
```sql
JourneyID INT,
CustomerID INT,
ProductID INT,
VisitDate DATE,
Stage VARCHAR(50),
Action_by_customers VARCHAR(10),
Duration INT
);


-- Create Table Products

CREATE TABLE products(
ProductID INT,
ProductName VARCHAR(50),
Category VARCHAR(50),
Price FLOAT
);

-- Create Table Engangement_data

CREATE TABLE engagement_data(
EngagementID INT,
ContentID INT,
ContentType VARCHAR(50),
Likes INT,
```

- After creating tables, get back to jupyter notebook to load values to all the created tables in mysql . Create *cursor()* for data connection , which acts as intermediate between mysql datasource and jupyter notebook python scripts. Which helps in execute all the SQL queries from Jupyter notebook



```python
#Create Cursor for db_connection
cursor = db_connection.cursor()

#Insert into Customer table
insert_query = """
INSERT INTO customers (CustomerID, CustomerName, Email, Gender, Age, GeographyID)
VALUES (%s, %s, %s, %s, %s, %s) """

#Convert df1 to list
cursor.executemany(insert_query, df1.values.tolist())
db_connection.commit()
```

```python
#Insert into Customer_reviews table
insert_query = """
INSERT INTO customer_reviews (ReviewID, CustomerID, ProductID, ReviewDate, Rating, ReviewText)
VALUES (%s, %s, %s, %s, %s, %s)
"""
#Convert df2 to list
cursor.executemany(insert_query, df2.values.tolist())
db_connection.commit()
```

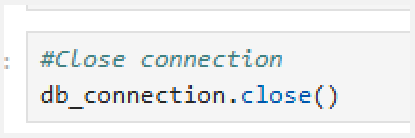


```python
#Insert into Products table
insert_query = """
INSERT INTO products (ProductID, ProductName, Category, Price)
VALUES (%s, %s, %s, %s)
"""
#Convert df4 to list
cursor.executemany(insert_query, df4.values.tolist())
db_connection.commit()
```

```python
#Insert into Engagement_data table
insert_query = """
INSERT INTO engagement_data (EngagementID, ContentID, ContentType, Likes, EngagementDate, CampaignID, ProductID, Views, Clicks)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
"""
#Convert df5 to list

cursor.executemany(insert_query, df5.values.tolist())
db_connection.commit()
```

```python
#Insert into Geography table
insert_query = """
INSERT INTO geography (GeographyID, Country, City)
VALUES (%s, %s, %s)
"""

#Convert df6 to list
cursor.executemany(insert_query, df6.values.tolist())
```

- After loading all the values to SQL tables, close the cursor which helps to prevent excessive memory usage also avoids connection issues



```
#Close connection
db_connection.close()
```

Once after updating values to the tables, work on executing the queries to provide insights based on the data and propose strategy for ShopEasy company.

Factors Influencing Customer Engagement

SELECT ed.ContentType,COUNT(*) AS TotalEngagements,SUM(ed.Views) AS TotalViews,SUM(ed.Clicks) AS TotalClicks

FROM engagement_data ed

GROUP BY ed.ContentType

ORDER BY TotalClicks DESC;

Customer Drop-off Stages

SELECT Stage, COUNT(*) AS DropoffCount

FROM customer_journey

WHERE Action_by_customers = 'View'

GROUP BY Stage

ORDER BY DropoffCount DESC;

Impact of Customer Reviews on Purchases

```
SELECT p.ProductName,ROUND(IFNULL(AVG(cr.Rating), 2),0) AS AvgRating,COUNT(cr.ReviewID)
AS TotalReviews,SUM(ed.Clicks) AS TotalClicks

FROM products p

LEFT JOIN customer_reviews cr ON p.ProductID = cr.ProductID

LEFT JOIN engagement_data ed ON p.ProductID = ed.ProductID

GROUP BY p.ProductID, p.ProductName

ORDER BY AvgRating ASC;
```

To identify negative reviews mentioning “price” or “quality concerns”:

```
SELECT * FROM customer_reviews

WHERE ReviewText LIKE '%price%' OR ReviewText LIKE '%quality%';
```

Best Performing Customer Segments:

```
SELECT c.Gender, c.Age, COUNT(cj.CustomerID) AS TotalEngagements

FROM customers c

JOIN customer_journey cj ON c.CustomerID = cj.CustomerID

GROUP BY c.Gender, c.Age

ORDER BY TotalEngagements DESC;
```

Best Performing locations:

```
SELECT g.Country, COUNT(c.CustomerID) AS CustomerCount

FROM customers c

JOIN geography g ON c.GeographyID = g.GeographyID

GROUP BY g.Country

ORDER BY CustomerCount DESC;
```

Best Performing Products:

```
SELECT cj.ProductID, COUNT(*) AS TotalViews

FROM customer_journey cj

WHERE cj.Stage = 'ProductPage'

GROUP BY cj.ProductID

ORDER BY TotalViews DESC;
```

Sentiment Analysis from Customer Reviews

```
SELECT ReviewID,ProductID,ReviewText,
  CASE
    WHEN ReviewText LIKE '%excellent%' OR ReviewText LIKE '%top-notch%' OR ReviewText LIKE
'%five stars%' THEN 'Positive'
    WHEN ReviewText LIKE '%nothing special%' OR ReviewText LIKE '%average%' THEN 'Neutral'
    WHEN ReviewText LIKE '%poor%' OR ReviewText LIKE '%bad%' OR ReviewText LIKE '%could be
better%' OR ReviewText LIKE '%disappointed%' THEN 'Negative'
    ELSE 'Unknown'
  END AS Sentiment
FROM customer_reviews;
```

Identifying Key Complaints from Low Ratings

```
SELECT ProductID,COUNT(*) AS LowRatingCount,ROUND(AVG(Rating),1) AS AvgLowRating

FROM customer_reviews

WHERE Rating <= 3

GROUP BY ProductID

ORDER BY LowRatingCount DESC;
```

Patterns between negative reviews and Product Performance

```
SELECT p.ProductID,p.ProductName,ROUND(IFNULL(AVG(cr.Rating),0),0) AS
AvgRating,COUNT(cr.ReviewID) AS TotalReviews,Round(AVG(ed.Views),0) AS AvgViews,

ROUND(AVG(ed.Clicks),0) AS AvgClicks

FROM products p

LEFT JOIN customer_reviews cr ON p.ProductID = cr.ProductID

LEFT JOIN engagement_data ed ON p.ProductID = ed.ProductID

GROUP BY p.ProductID, p.ProductName

ORDER BY AvgRating ASC;
```

Recommended strategies to enhance Customer Satisfaction

```
    SELECT p.ProductName,ROUND(AVG(cr.Rating), 1) AS AvgRating,SUM(ed.Views) AS
    TotalViews,SUM(ed.Clicks) AS TotalClicks

    FROM products p

    JOIN customer_reviews cr ON p.ProductID = cr.ProductID

    JOIN engagement_data ed ON p.ProductID = ed.ProductID

    GROUP BY p.ProductName

    HAVING AvgRating < 3.5 AND TotalViews > 1000

    ORDER BY AvgRating ASC;
```