In [ ]:
```
#installation
```

In [ ]:
```
!pip install mysql-connector-python
```

In [ ]:
```
!pip install --upgrade mysql-connector-python
```

In [ ]:
```
#connection
```

In [1]:
```
import mysql.connector

conn = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    password="@afreenswt26._.",
    auth_plugin="mysql_native_password"
)
```

In [3]:
```
import mysql.connector

conn = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    password="@afreenswt26._.",
    auth_plugin="mysql_native_password"
)

# Create a cursor object
cursor = conn.cursor()

# Execute a simple query
cursor.execute("SELECT VERSION()")

# Fetch and print the result
version = cursor.fetchone()
print("Connected to MySQL Server version:", version[0])

# Close the connection
cursor.close()
conn.close()
```

```
Connected to MySQL Server version: 8.0.41
```

In [ ]:
```
#creating data base
```

In [5]:
```
import mysql.connector

# Establish connection
conn = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    password="@afreenswt26._.",
    auth_plugin="mysql_native_password"
)

# Create a cursor object
cursor = conn.cursor()
```

```
# Create database
cursor.execute("CREATE DATABASE IF NOT EXISTS ShopEasy")

# Confirm creation
cursor.execute("SHOW DATABASES")
for db in cursor:
    print(db)

# Close connection
cursor.close()
conn.close()
```

```
('information_schema',)
('mysql',)
('performance_schema',)
('shopeasy',)
('sys',)
```

In [ ]:
```
#reconnecting and creating tables
```

In [7]:
```
conn = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    password="@afreenswt26._.",
    auth_plugin="mysql_native_password",
    database="ShopEasy"
)

cursor = conn.cursor()

# Create Customers table
cursor.execute("""
CREATE TABLE IF NOT EXISTS customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(255),
    Email VARCHAR(255),
    Gender VARCHAR(10),
    Age INT,
    GeographyID INT
);
""")

# Create Products table
cursor.execute("""
CREATE TABLE IF NOT EXISTS products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255),
    Category VARCHAR(255),
    Price DECIMAL(10,2)
);
""")

# Create Geography table
cursor.execute("""
CREATE TABLE IF NOT EXISTS geography (
    GeographyID INT PRIMARY KEY,
    Country VARCHAR(255),
    City VARCHAR(255)
);
""")
```

```
# Create Engagement Data table
cursor.execute("""
CREATE TABLE IF NOT EXISTS engagement_data (
    EngagementID INT PRIMARY KEY,
    ContentID INT,
    ContentType VARCHAR(255),
    Likes INT,
    EngagementDate DATE,
    CampaignID INT,
    ProductID INT,
    ViewsClicksCombined VARCHAR(255)
);
""")

# Create Customer Reviews table
cursor.execute("""
CREATE TABLE IF NOT EXISTS customer_reviews (
    ReviewID INT PRIMARY KEY,
    CustomerID INT,
    ProductID INT,
    ReviewDate DATE,
    Rating INT,
    ReviewText TEXT
);
""")

# Create Customer Journey table
cursor.execute("""
CREATE TABLE IF NOT EXISTS customer_journey (
    JourneyID INT PRIMARY KEY,
    CustomerID INT,
    ProductID INT,
    VisitDate DATE,
    Stage VARCHAR(255),
    Action VARCHAR(255),
    Duration INT
);
""")

# Commit and close
conn.commit()
cursor.close()
conn.close()

print("All tables created successfully!")
```

```
All tables created successfully!
```

In [ ]:
```
#installing pandas to load data
```

In [ ]:
```
!pip install pandas mysql-connector-python
```

In [ ]:
```
#data insertion
```

In [ ]:
```
#one way of insertion before cleaning
```

In [ ]:
```
import mysql.connector
import pandas as pd
```

```
# Database Connection
conn = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    password="@afreenswt26_",
    database="ShopEasy"
)
cursor = conn.cursor()

# Function to insert data into MySQL
def insert_data(df, table_name, columns):
    for _, row in df.iterrows():
        values = tuple(row)
        placeholders = ", ".join(["%s"] * len(row))
        query = f"INSERT INTO {table_name} ({columns}) VALUES ({placeholders})"
        try:
            cursor.execute(query, values)
        except mysql.connector.IntegrityError:
            pass  # Ignore duplicate primary keys

# Load and insert Customers data
df_customers = pd.read_csv("/mnt/data/customers.csv")
insert_data(df_customers, "customers", "CustomerID, CustomerName, Email, Gender,

# Load and insert Products data
df_products = pd.read_csv("/mnt/data/products.csv")
insert_data(df_products, "products", "ProductID, ProductName, Category, Price")

# Load and insert Geography data
df_geography = pd.read_csv("/mnt/data/geography.csv")
insert_data(df_geography, "geography", "GeographyID, Country, City")

# Load and insert Engagement data
df_engagement = pd.read_csv("/mnt/data/engagement_data.csv")
insert_data(df_engagement, "engagement_data", "EngagementID, ContentID, ContentT

# Load and insert Customer Reviews data
df_reviews = pd.read_csv("/mnt/data/customer_reviews.csv")
insert_data(df_reviews, "customer_reviews", "ReviewID, CustomerID, ProductID, Re

# Load and insert Customer Journey data
df_journey = pd.read_csv("/mnt/data/customer_journey.csv")
insert_data(df_journey, "customer_journey", "JourneyID, CustomerID, ProductID, V

# Commit and close connection
conn.commit()
cursor.close()
conn.close()

print("Data successfully inserted into all tables!")
```

In [ ]:
```
#cleaning importing each file into table
```

In [128…
```
print(customers_df.isnull().sum())
```

```
CustomerID      0
CustomerName    0
Email           0
Gender          0
Age             0
GeographyID     0
dtype: int64
```

In [136…
```
print("Duplicates records:",customers_df.duplicated().sum())
```

```
Duplicates records: 0
```

In [138…
```
customers_df.drop_duplicates(subset=['CustomerID'], keep='first', inplace=True)
print("✅ Duplicates removed! Unique customers:", customers_df.shape[0])
```

```
✅ Duplicates removed! Unique customers: 100
```

In [112…
```
customers_df.head()
```

Out[112…

	CustomerID	CustomerName	Email	Gender	Age	Geograph
0	1	Emma Anderson	emma.anderson@example.com	Male	50	
1	2	Sarah Brown	sarah.brown@example.com	Female	37	
2	3	Robert Hernandez	robert.hernandez@example.com	Female	26	
3	4	David Garcia	david.garcia@example.com	Male	25	
4	5	Emma Miller	emma.miller@example.com	Female	41	

In [116…
```
print(customers_df.shape)
```

```
(100, 6)
```

In [118…
```
print(customers_df.dtypes)
```

```
CustomerID       int32
CustomerName    object
Email           object
Gender          object
Age              int32
GeographyID      int32
dtype: object
```

In [120…
```
customers_df = customers_df.astype({'CustomerID': 'int32', 'GeographyID': 'int32
```

In [171…
```
import pymysql

# Database connection
conn = pymysql.connect( host="127.0.0.1",
        user="root",
        password="@afreenswt26._.",
        database="ShopEasy")
cursor = conn.cursor()

# Insert statement
insert_query = """
    INSERT INTO customers (CustomerID, CustomerName, Email, Gender, Age, Geograp
```

```
    VALUES (%s, %s, %s, %s, %s, %s)
    ON DUPLICATE KEY UPDATE
        CustomerName=VALUES(CustomerName),
        Email=VALUES(Email),
        Gender=VALUES(Gender),
        Age=VALUES(Age),
        GeographyID=VALUES(GeographyID);
"""

# Converting DataFrame to list of tuples
customer_records = [tuple(row) for row in customers_df.itertuples(index=False, n

# Execute batch insert
cursor.executemany(insert_query, customer_records)
conn.commit()

print("✅ Data inserted successfully!")

# Close connection
cursor.close()
conn.close()
```

```
✅ Data inserted successfully!
```

In [ ]:
```
#cleaning importing product file into table
```

In [192…
```
print(products_df.isnull().sum())
```

```
ProductID      0
ProductName    0
Category       0
Price          0
dtype: int64
```

In [194…
```
print(products_df.dtypes)
```

```
ProductID        int64
ProductName     object
Category        object
Price          float64
dtype: object
```

In [196…
```
import pandas as pd

# Load data from CSV
products = pd.read_csv("C:/Users/aadil/Downloads/products.csv")

# Display first few rows to verify
print(products.head())
```

```
   ProductID      ProductName Category   Price
0          1    Running Shoes   Sports  223.75
1          2  Fitness Tracker   Sports  196.68
2          3         Yoga Mat   Sports  485.32
3          4        Dumbbells   Sports   26.21
4          5      Soccer Ball   Sports   41.26
```

In [198…
```
print(type(products))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [202…
```
products.drop_duplicates(subset=['ProductID'], keep='first', inplace=True)
```

In [204…
```
insert_query = """
INSERT IGNORE INTO Products (ProductID, ProductName, Category, Price)
VALUES (%s, %s, %s, %s)
"""
```

In [206…
```
query = "SELECT * FROM Products LIMIT 15;"  # Fetch first 10 rows
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)
```

```
(1, 'Running Shoes', 'Sports', Decimal('223.75'))
(2, 'Fitness Tracker', 'Sports', Decimal('196.68'))
(3, 'Yoga Mat', 'Sports', Decimal('485.32'))
(4, 'Dumbbells', 'Sports', Decimal('26.21'))
(5, 'Soccer Ball', 'Sports', Decimal('41.26'))
(6, 'Tennis Racket', 'Sports', Decimal('36.07'))
(7, 'Basketball', 'Sports', Decimal('225.12'))
(8, 'Football Helmet', 'Sports', Decimal('44.75'))
(9, 'Baseball Glove', 'Sports', Decimal('327.36'))
(10, 'Golf Clubs', 'Sports', Decimal('81.59'))
(11, 'Ski Boots', 'Sports', Decimal('340.20'))
(12, 'Ice Skates', 'Sports', Decimal('37.56'))
(13, 'Swim Goggles', 'Sports', Decimal('145.97'))
(14, 'Cycling Helmet', 'Sports', Decimal('472.32'))
(15, 'Climbing Rope', 'Sports', Decimal('410.17'))
```

In [ ]:
```
#cleaning importing geography file into table
```

In [208…
```
print(geography_df.isnull().sum())
```

```
GeographyID    0
Country        0
City           0
dtype: int64
```

In [210…
```
print(geography_df.dtypes)
```

```
GeographyID     int64
Country        object
City           object
dtype: object
```

In [212…
```
print("Duplicates records:",geography_df.duplicated().sum())
```

```
Duplicates records: 0
```

In [214…
```
insert_query = """
INSERT INTO Geography (GeographyID, Country, City)
VALUES (%s, %s, %s)
"""

# Convert DataFrame rows into tuples for insertion
data_to_insert = [tuple(row) for row in geography_df.itertuples(index=False, nam

# Execute the query
try:
```

```
    cursor.executemany(insert_query, data_to_insert)
    conn.commit()  # Save changes
    print("✅ Data inserted successfully into Geography table")
except pymysql.MySQLError as e:
    print(f"❌ Error inserting data: {e}")
```

```
❌ Error inserting data: (1062, "Duplicate entry '1' for key 'geography.PRIMAR
Y'")
```

In [220…
```
print("Duplicate GeographyID count:", geography_df.duplicated(subset=['Geography
```

```
Duplicate GeographyID count: 0
```

In [222…
```
cursor.execute("SELECT COUNT(*) FROM Geography;")
print("Rows in Geography table:", cursor.fetchone()[0])
```

```
Rows in Geography table: 10
```

In [224…
```
query = "SELECT * FROM Geography LIMIT 10;"  # Fetch first 10 rows
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)
```

```
(1, 'UK', 'London')
(2, 'Germany', 'Berlin')
(3, 'France', 'Paris')
(4, 'Spain', 'Madrid')
(5, 'Italy', 'Rome')
(6, 'Netherlands', 'Amsterdam')
(7, 'Belgium', 'Brussels')
(8, 'Sweden', 'Stockholm')
(9, 'Switzerland', 'Zurich')
(10, 'Austria', 'Vienna')
```

In [ ]:
```
#cleaning importing engagement file into table
```

In [226…
```
print(engagement_df.isnull().sum())
```

```
EngagementID          0
ContentID             0
ContentType           0
Likes                 0
EngagementDate        0
CampaignID            0
ProductID             0
ViewsClicksCombined   0
dtype: int64
```

In [228…
```
print(engagement_df.dtypes)
```

```
EngagementID                  int64
ContentID                     int64
ContentType                  object
Likes                         int64
EngagementDate       datetime64[ns]
CampaignID                    int64
ProductID                     int64
ViewsClicksCombined          object
dtype: object
```

In [230…
```
print("Duplicates records:",engagement_df.duplicated().sum())
```

```
Duplicates records: 0
```

In [244…
```
engagement_df['EngagementDate'] = pd.to_datetime(engagement_df['EngagementDate']
```

In [246…
```
engagement_df["ContentType"] = engagement_df["ContentType"].astype(str)

# Convert EngagementDate to MySQL-compatible string format
engagement_df["EngagementDate"] = engagement_df["EngagementDate"].dt.strftime("%

# SQL Query for Insertion
insert_query = """
INSERT INTO engagement_data (EngagementID, ContentID, ContentType, Likes, Engage
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
ON DUPLICATE KEY UPDATE
    Likes = VALUES(Likes),
    ViewsClicksCombined = VALUES(ViewsClicksCombined);
"""

# Convert DataFrame rows into tuples
data_to_insert = [tuple(row) for row in engagement_df.itertuples(index=False, na

# Execute the query
try:
    cursor.executemany(insert_query, data_to_insert)
    conn.commit()  # Save changes
    print("✅ Engagement data inserted successfully!")
except pymysql.MySQLError as e:
    print(f"❌ Error inserting data: {e}")
```

```
✅ Engagement data inserted successfully!
```

In [250…
```
query = "SELECT * FROM engagement_data LIMIT 10;"  # Fetch first 10 rows
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)
```

```
(1, 39, 'Blog', 190, datetime.date(2023, 8, 30), 1, 9, '1883-671')
(2, 48, 'Blog', 114, datetime.date(2023, 3, 28), 18, 20, '5280-532')
(3, 16, 'video', 32, datetime.date(2023, 12, 8), 7, 14, '1905-204')
(4, 43, 'Video', 17, datetime.date(2025, 1, 21), 19, 20, '2766-257')
(5, 16, 'newsletter', 306, datetime.date(2024, 2, 21), 6, 15, '5116-1524')
(6, 32, 'Socialmedia', 648, datetime.date(2023, 6, 18), 18, 19, '8237-1641')
(7, 33, 'SOCIALMEDIA', 1, datetime.date(2025, 10, 1), 12, 2, '750-34')
(8, 47, 'Blog', 1, datetime.date(2025, 3, 31), 17, 6, '891-35')
(9, 48, 'blog', 123, datetime.date(2024, 3, 19), 13, 16, '5571-1527')
(10, 4, 'Blog', 25, datetime.date(2023, 12, 3), 15, 15, '4279-297')
```

print(customer_reviews_df.head())

In [ ]:
```
#cleaning and inserting customer_reviews data into table
```

In [262…
```
print(customer_reviews_df.isnull().sum())
```

```
ReviewID      0
CustomerID    0
ProductID     0
ReviewDate    0
Rating        0
ReviewText    0
dtype: int64
```

In [264…
```
print(customer_reviews_df.dtypes)
```

```
ReviewID       int64
CustomerID     int64
ProductID      int64
ReviewDate    object
Rating         int64
ReviewText    object
dtype: object
```

In [266…
```
print("Duplicates records:",customer_reviews_df.duplicated().sum())
```

```
Duplicates records: 0
```

In [167…
```
import pandas as pd

# Load your DataFrame
customer_reviews_df = pd.read_csv("C:/Users/aadil/Downloads/customer_reviews.csv

# Connect to SQLite database
conn = sqlite3.connect("shop_easy.db")  # Change to your actual database name
cursor = conn.cursor()

# Insert the DataFrame into the existing SQLite table
customer_reviews_df.to_sql("customer_reviews", conn, if_exists="append", index=F

# Verify the insertion
print(pd.read_sql("SELECT * FROM customer_reviews LIMIT 10;", conn))

# Close the connection
conn.close()
```

```
   ReviewID  CustomerID  ProductID  ReviewDate  Rating  \
0         1          77         18  2023-12-23       3
1         2          80         19  2024-12-25       5
2         3          50         13  2025-01-26       4
3         4          78         15  2025-04-21       3
4         5          64          2  2023-07-16       3
5         6          81          1  2025-12-21       4
6         7          16          1  2024-01-29       3
7         8          55          8  2024-08-15       5
8         9           3         13  2023-09-01       4
9        10          78          6  2024-06-17       5

                                ReviewText
0     Average  experience,  nothing  special.
1              The  quality  is    top-notch.
2     Five  stars  for  the  quick  delivery.
3    Good  quality,  but  could  be  cheaper.
4     Average  experience,  nothing  special.
5      Customer  support  was  very  helpful.
6     Average  experience,  nothing  special.
7              The  quality    is  top-notch.
8  I  love  this  product,  will  buy  again!
9     Excellent  product,  highly  recommend!
```

In [ ]:
```
#cleaning and inserting customer_journey datta into table
```

In [293…
```
print(customer_journey_df.isnull().sum())
```

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

In [165…
```
import pandas as pd
import sqlite3

# Connect to the database
conn = sqlite3.connect("shopeasy.db")

# Load the CSV file
customer_journey_df = pd.read_csv("C:/Users/aadil/Downloads/customer_journey.csv

# Handling Missing Values in 'Duration'
customer_journey_df["Duration"].fillna(customer_journey_df["Duration"].median(),

# Insert Data
customer_journey_df.to_sql("customer_journey", conn, if_exists="append", index=F

# Commit and close
conn.commit()
conn.close()

print("Data inserted successfully after handling missing values!")
```

```
Data inserted successfully after handling missing values!
```

```
C:\Users\aadil\AppData\Local\Temp\ipykernel_2180\1044891107.py:11: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  customer_journey_df["Duration"].fillna(customer_journey_df["Duration"].median
(), inplace=True)
```

In [297…

```python
query = "SELECT * FROM customer_journey LIMIT 10;"  # Fetch first 10 rows
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)
```

```
(1, 64, 18, '2024-06-10', 'Checkout', 'Drop-off', 182.0)
(2, 94, 11, '2025-07-09', 'Checkout', 'Drop-off', 182.0)
(3, 34, 8, '2024-06-14', 'ProductPage', 'View', 235.0)
(4, 33, 18, '2025-05-28', 'Checkout', 'Drop-off', 182.0)
(5, 91, 10, '2023-02-11', 'Homepage', 'Click', 156.0)
(6, 54, 11, '2025-12-19', 'Homepage', 'View', 264.0)
(7, 80, 4, '2023-08-25', 'Homepage', 'View', 298.0)
(8, 99, 10, '2025-07-03', 'ProductPage', 'View', 287.0)
(9, 31, 4, '2025-06-13', 'ProductPage', 'View', 278.0)
(10, 44, 16, '2025-04-23', 'ProductPage', 'View', 30.0)
```

In [309…

```python
import pandas as pd
customer_reviews_df = pd.read_csv("C:/Users/aadil/Downloads/customer_reviews.csv
```

In [311…

```python
import sqlite3
import pandas as pd

conn = sqlite3.connect('shopeasy.db')
query = "SELECT * FROM customer_reviews"
customer_reviews_df = pd.read_sql(query, conn)
print(customer_reviews_df.head())
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
File C:\ProgramData\anaconda3\Lib\site-packages\pandas\io\sql.py:2674, in SQLiteDatabase.execute(self, sql, params)
   2673 try:
-> 2674     cur.execute(sql, *args)
   2675     return cur

OperationalError: no such table: customer_reviews

The above exception was the direct cause of the following exception:

DatabaseError                             Traceback (most recent call last)
Cell In[311], line 6
      4 conn = sqlite3.connect('shopeasy.db')
      5 query = "SELECT * FROM customer_reviews"
----> 6 customer_reviews_df = pd.read_sql(query, conn)
      7 print(customer_reviews_df.head())

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\io\sql.py:706, in read_sql(sql, con, index_col, coerce_float, params, parse_dates, columns, chunksize, dtype_backend, dtype)
    704 with pandasSQL_builder(con) as pandas_sql:
    705     if isinstance(pandas_sql, SQLiteDatabase):
--> 706         return pandas_sql.read_query(
    707             sql,
    708             index_col=index_col,
    709             params=params,
    710             coerce_float=coerce_float,
    711             parse_dates=parse_dates,
    712             chunksize=chunksize,
    713             dtype_backend=dtype_backend,
    714             dtype=dtype,
    715         )
    717     try:
    718         _is_table_name = pandas_sql.has_table(sql)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\io\sql.py:2738, in SQLiteDatabase.read_query(self, sql, index_col, coerce_float, parse_dates, params, chunksize, dtype, dtype_backend)
   2727 def read_query(
   2728     self,
   2729     sql,
   (...)
   2736     dtype_backend: DtypeBackend | Literal["numpy"] = "numpy",
   2737 ) -> DataFrame | Iterator[DataFrame]:
-> 2738     cursor = self.execute(sql, params)
   2739     columns = [col_desc[0] for col_desc in cursor.description]
   2741     if chunksize is not None:

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\io\sql.py:2686, in SQLiteDatabase.execute(self, sql, params)
   2683     raise ex from inner_exc
   2685 ex = DatabaseError(f"Execution failed on sql '{sql}': {exc}")
-> 2686 raise ex from exc

DatabaseError: Execution failed on sql 'SELECT * FROM customer_reviews': no such table: customer_reviews
```

In [ ]:
```
#checking whether i have table
```

In [313…
```
import sqlite3

conn = sqlite3.connect("shopeasy.db")
cursor = conn.cursor()

cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

print(tables)  # Check if 'customer_reviews' is listed

conn.close()
```

```
[('customer_journey',)]
```

In [ ]:
```
#creating because it is missing
```

In [317…
```
import pandas as pd
import sqlite3

conn = sqlite3.connect("shopeasy.db")

# Load CSV again
customer_reviews_df = pd.read_csv("C:/Users/aadil/Downloads/customer_reviews.csv

# Insert into database
customer_reviews_df.to_sql("customer_reviews", conn, if_exists="replace", index=

conn.close()
```

In [ ]:
```
#reassuring whether it is created or not
```

In [319…
```
conn = sqlite3.connect("shopeasy.db")
query = "SELECT * FROM customer_reviews LIMIT 5"
customer_reviews_df = pd.read_sql(query, conn)

print(customer_reviews_df.head())
conn.close()
```

```
   ReviewID  CustomerID  ProductID  ReviewDate  Rating  \
0         1          77         18  2023-12-23       3
1         2          80         19  2024-12-25       5
2         3          50         13  2025-01-26       4
3         4          78         15  2025-04-21       3
4         5          64          2  2023-07-16       3

                              ReviewText
0   Average experience, nothing special.
1           The quality is top-notch.
2   Five stars for the quick delivery.
3  Good quality, but could be cheaper.
4   Average experience, nothing special.
```

In [ ]:
```
#loading data into data frame
```

In [321…
```
import sqlite3
import pandas as pd
```

```
conn = sqlite3.connect('shopeasy.db')
query = "SELECT * FROM customer_reviews"
customer_reviews_df = pd.read_sql(query, conn)
print(customer_reviews_df.head())
```

```
   ReviewID  CustomerID  ProductID  ReviewDate  Rating  \
0         1          77         18  2023-12-23       3
1         2          80         19  2024-12-25       5
2         3          50         13  2025-01-26       4
3         4          78         15  2025-04-21       3
4         5          64          2  2023-07-16       3

                              ReviewText
0   Average  experience,  nothing  special.
1           The  quality  is    top-notch.
2   Five  stars  for  the  quick  delivery.
3  Good  quality,  but  could  be  cheaper.
4   Average  experience,  nothing  special.
```

In [ ]:
```
#finding top 5 products with highest rating
```

In [325…
```
top_products = customer_reviews_df.groupby('ProductID')['Rating'].mean().sort_va
print(top_products)
```

```
ProductID
8     5.0
19    4.4
1     4.0
18    4.0
15    4.0
Name: Rating, dtype: float64
```

In [ ]:
```
#finding most frequent customer action
```

In [327…
```
journey_df = pd.read_sql("SELECT Action FROM customer_journey", conn)
print(journey_df['Action'].value_counts())
```

```
Action
View        58
Click       22
Drop-off    14
Purchase     6
Name: count, dtype: int64
```

In [ ]:
```
#customer dropoff analysis
```

In [335…
```
drop_off_analysis = customer_journey_df[customer_journey_df['Action'] == 'Drop-o
    .groupby('Stage')['Action'].count().reset_index(name='drop_off_count') \
    .sort_values(by='drop_off_count', ascending=False)
print(drop_off_analysis)
```

```
      Stage  drop_off_count
0  Checkout              12
1  checkout               2
```

In [333…
```
print(customer_journey_df['Action'].unique())  # Check all unique actions
```

```
['Drop-off' 'View' 'Click' 'Purchase']
```

In [ ]:
```
#due to case sensitive i changed to lowercase  so total 14
```

In [337…
```
customer_journey_df['Stage'] = customer_journey_df['Stage'].str.strip().str.lowe

drop_off_analysis = customer_journey_df[customer_journey_df['Action'] == 'Drop-o
    .groupby('Stage')['Action'].count().reset_index(name='drop_off_count') \
    .sort_values(by='drop_off_count', ascending=False)

print(drop_off_analysis)
```

```
     Stage  drop_off_count
0  checkout              14
```

In [ ]:
```
#analyzing time spent before dropoff
```

In [339…
```
avg_duration = customer_journey_df[customer_journey_df['Action'] == 'Drop-off']
    .groupby('Stage')['Duration'].mean().reset_index(name='avg_duration')
print(avg_duration)
```

```
     Stage  avg_duration
0  checkout         182.0
```

In [ ]:
```
#analyzing whether the new or returning customers are dropping off
```

In [341…
```
drop_off_customers = customer_journey_df[customer_journey_df['Action'] == 'Drop-
customer_drop_counts = drop_off_customers.groupby('CustomerID').size().reset_ind
print(customer_drop_counts)
```

```
    CustomerID  drop_off_count
0            1               1
1            9               1
2           15               1
3           23               1
4           30               1
5           33               1
6           38               1
7           40               1
8           43               1
9           58               1
10          64               1
11          67               1
12          77               1
13          94               1
```

In [ ]:
```
#since we have only id with the help of visit date we are finding whether is tha
#A new customer is someone whose first recorded visit in the dataset matches the
#A returning customer has multiple visits, meaning their drop-off is not their f
```

In [343…
```
#finding first visit for each customer

customer_journey_df['VisitDate'] = pd.to_datetime(customer_journey_df['VisitDate

# Get the first visit date for each customer
first_visit = customer_journey_df.groupby('CustomerID')['VisitDate'].min().reset
first_visit.rename(columns={'VisitDate': 'FirstVisitDate'}, inplace=True)

# Merge with the main dataframe
customer_journey_df = customer_journey_df.merge(first_visit, on='CustomerID', ho
```

In [ ]:
```
#identifying and checking whether they are visiting first time
```

In [345…
```
drop_offs = customer_journey_df[customer_journey_df['Action'] == 'Drop-off']

# Check if the drop-off happened on the first visit
drop_offs['is_new_customer'] = drop_offs['VisitDate'] == drop_offs['FirstVisitDa

# Count drop-offs by customer type
drop_off_summary = drop_offs['is_new_customer'].value_counts().reset_index()
drop_off_summary.columns = ['IsNewCustomer', 'DropOffCount']

print(drop_off_summary)
```

```
   IsNewCustomer  DropOffCount
0           True            10
1          False             4
```

```
C:\Users\aadil\AppData\Local\Temp\ipykernel_19612\863999664.py:4: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  drop_offs['is_new_customer'] = drop_offs['VisitDate'] == drop_offs['FirstVisitD
ate']
```

In [ ]:
```
 #Interpretation of Results

#If True has a high count → Many drop-offs are from first-time customers, meanin


#If False has a high count → Returning customers are dropping off, suggesting fr

#That means first-time customers are dropping off significantly more than return

1. Complicated onboarding or checkout process


2. Lack of trust in the platform (security, return policies, etc.)


3. Pricing issues or unexpected costs at checkout
```

In [ ]:
```
#common actions before conversion
```

In [5]:
```
query = """
SELECT stage, action, COUNT(*) AS action_count
FROM customer_journey
GROUP BY stage, action
ORDER BY action_count DESC;
"""
df = pd.read_sql_query(query, conn)
df
```

Out[5]:

	Stage	Action	action_count
0	Homepage	View	35
1	ProductPage	View	19
2	Homepage	Click	15
3	Checkout	Drop-off	12
4	Checkout	Purchase	6
5	ProductPage	Click	5
6	checkout	Drop-off	2
7	homepage	Click	2
8	homepage	View	2
9	productpage	View	2

In [ ]:
```
#Average time spent per stage
```

In [19]:
```
query = """
SELECT stage, ROUND(AVG(duration), 2) AS avg_duration_seconds
FROM customer_journey
WHERE duration IS NOT NULL
GROUP BY stage
ORDER BY avg_duration_seconds DESC;
"""
df = pd.read_sql_query(query, conn)
df
```

Out[19]:

	Stage	avg_duration_seconds
0	ProductPage	186.67
1	homepage	185.75
2	checkout	182.00
3	Checkout	171.39
4	Homepage	158.52
5	productpage	136.00

In [ ]:
```
#most viewed product pages
```

In [21]:
```
query = """
SELECT productid, COUNT(*) AS view_count
FROM customer_journey
WHERE stage = 'ProductPage' AND action = 'View'
GROUP BY productid
ORDER BY view_count DESC
LIMIT 10;
"""
df = pd.read_sql_query(query, conn)
df
```

Out[21]:

	ProductID	view_count
0	20	3
1	18	2
2	12	2
3	10	2
4	8	2
5	16	1
6	13	1
7	9	1
8	6	1
9	5	1

In [ ]:
```
# customer table loading again
```

In [37]:
```
customers_df = pd.read_csv("C:/Users/aadil/Downloads/customers.csv")  # Load CSV
customers_df.to_sql("customers", conn, if_exists="replace", index=False)  # Save
```

Out[37]: 100

In [ ]:
```
# geography table loading  again
```

In [43]:
```
geography_df = pd.read_csv("C:/Users/aadil/Downloads/geography.csv")  # Load CSV
geography_df.to_sql("geography", conn, if_exists="replace", index=False)  # Save
```

Out[43]: 10

In [ ]:
```
#customer who dropped off at check out
```

In [51]:
```
query = """
SELECT DISTINCT c.customerid, c.customername, c.email, g.country, g.city
FROM customer_journey cj
JOIN customers c ON cj.customerid = c.customerid
JOIN geography g ON c.geographyid = g.geographyid
WHERE cj.stage = 'Checkout' AND cj.action = 'Drop-off';
"""
dropoff_customers_df = pd.read_sql_query(query, conn)
df
```

Out[51]:

	CustomerID	CustomerName	Email	Country	City
0	64	Sarah Martinez	sarah.martinez@example.com	Belgium	Brussels
1	94	Jane Anderson	jane.anderson@example.com	Belgium	Brussels
2	33	David Thomas	david.thomas@example.com	Spain	Madrid
3	23	Isabella Garcia	isabella.garcia@example.com	Italy	Rome
4	58	Jane Williams	jane.williams@example.com	Austria	Vienna
5	77	David Lopez	david.lopez@example.com	Spain	Madrid
6	67	Alex Johnson	alex.johnson@example.com	Spain	Madrid
7	38	John Garcia	john.garcia@example.com	Spain	Madrid
8	15	Emma Martinez	emma.martinez@example.com	Switzerland	Zurich
9	1	Emma Anderson	emma.anderson@example.com	Germany	Berlin
10	40	Olivia Thomas	olivia.thomas@example.com	Germany	Berlin
11	43	Olivia Hernandez	olivia.hernandez@example.com	Sweden	Stockholm

In [ ]:
```
#Exporting the drop-off customer results(first five)
```

In [53]:
```
dropoff_customers_df.to_csv("dropoff_customers.csv", index=False)
```

In [55]:
```
dropoff_customers_df.head()
```

Out[55]:

	CustomerID	CustomerName	Email	Country	City
0	64	Sarah Martinez	sarah.martinez@example.com	Belgium	Brussels
1	94	Jane Anderson	jane.anderson@example.com	Belgium	Brussels
2	33	David Thomas	david.thomas@example.com	Spain	Madrid
3	23	Isabella Garcia	isabella.garcia@example.com	Italy	Rome
4	58	Jane Williams	jane.williams@example.com	Austria	Vienna

In [57]:
```
from IPython.display import FileLink
FileLink("dropoff_customers.csv")
```

Out[57]: dropoff_customers.csv

In [ ]:
```
#Customer Reviews Analysis

#Identifying highest  rated products
```

In [61]:
```
products_df = pd.read_csv("C:/Users/aadil/Downloads/products.csv")  # Load CSV
products_df.to_sql("products", conn, if_exists="replace", index=False)  # Save t
```

Out[61]: 20

In [161…
```
query = """
SELECT p.ProductID, p.ProductName, ROUND(AVG(r.Rating), 2) AS AvgRating, COUNT(r
FROM customer_reviews r
JOIN products p ON r.ProductID = p.ProductID
GROUP BY p.ProductID, p.ProductName
ORDER BY AvgRating DESC, TotalReviews DESC
LIMIT 5;
"""
top_rated_products_df = pd.read_sql_query(query, conn)
top_rated_products_df
```

Out[161…

	ProductID	ProductName	AvgRating	TotalReviews
0	8	Football Helmet	5.0	3
1	19	Hockey Stick	4.4	5
2	15	Climbing Rope	4.0	6
3	11	Ski Boots	4.0	6
4	1	Running Shoes	4.0	4

In [65]:
```
#lowest rated product
```

In [159…
```
query = """
SELECT p.ProductID, p.ProductName, ROUND(AVG(r.Rating), 1) AS AvgRating, COUNT(r
FROM customer_reviews r
JOIN products p ON r.ProductID = p.ProductID
GROUP BY p.ProductID, p.ProductName
ORDER BY AvgRating ASC, TotalReviews DESC
LIMIT 5;
"""
low_rated_products_df = pd.read_sql_query(query, conn)
low_rated_products_df
```

Out[159…

	ProductID	ProductName	AvgRating	TotalReviews
0	7	Basketball	2.7	3
1	4	Dumbbells	3.0	5
2	12	Ice Skates	3.0	2
3	16	Kayak	3.4	10
4	9	Baseball Glove	3.4	5

In [ ]:
```
# customer  reviews table loading again
```

In [71]:
```
customer_reviews_df = pd.read_csv("C:/Users/aadil/Downloads/customer_reviews.csv
customer_reviews_df.to_sql("customer_reviews", conn, if_exists="replace", index=
```

Out[71]: 100

In [ ]:
```
#classification to find negative and positive reviews based on ratings
```

In [73]:
```
def classify_sentiment(rating):
    if rating >= 4:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    else:
        return "Negative"

# Apply sentiment classification
customer_reviews_df["Sentiment"] = customer_reviews_df["Rating"].apply(classify_

# Show results
customer_reviews_df.head()
```

Out[73]:

	ReviewID	CustomerID	ProductID	ReviewDate	Rating	ReviewText	Sentiment
0	1	77	18	2023-12-23	3	Average experience, nothing special.	Neutral
1	2	80	19	2024-12-25	5	The quality is top-notch.	Positive
2	3	50	13	2025-01-26	4	Five stars for the quick delivery.	Positive
3	4	78	15	2025-04-21	3	Good quality, but could be cheaper.	Neutral
4	5	64	2	2023-07-16	3	Average experience, nothing special.	Neutral

In [ ]:
```
#identifying repeated words frequently
```

In [75]:
```
from collections import Counter
import re

# Function to clean text
def clean_text(text):
    text = re.sub(r"[^a-zA-Z\s]", "", str(text))  # Remove special characters
    return text.lower()

# Apply text cleaning
customer_reviews_df["CleanedReview"] = customer_reviews_df["ReviewText"].apply(c

# Separate positive & negative reviews
positive_reviews = " ".join(customer_reviews_df[customer_reviews_df["Sentiment"]
negative_reviews = " ".join(customer_reviews_df[customer_reviews_df["Sentiment"]

# Find most common words
positive_words = Counter(positive_reviews.split()).most_common(10)
negative_words = Counter(negative_reviews.split()).most_common(10)
```

```
print("Top Positive Words:", positive_words)
print("Top Negative Words:", negative_words)
```

```
Top Positive Words: [('the', 35), ('was', 19), ('for', 17), ('five', 16), ('star
s', 16), ('quick', 16), ('delivery', 16), ('quality', 12), ('very', 11), ('produc
t', 11)]
Top Negative Words: [('the', 6), ('product', 5), ('experience', 3), ('not', 2),
('a', 2), ('with', 2), ('average', 2), ('nothing', 2), ('special', 2), ('did',
1)]
```

In [ ]:
```
#Sentiment analysis
```

In [77]:
```
sentiment_counts = customer_reviews_df["Sentiment"].value_counts()
print(sentiment_counts)
```

```
Sentiment
Positive    62
Neutral     29
Negative     9
Name: count, dtype: int64
```

In [ ]:
```
#correlate review trends with product performance
```

In [81]:
```
engagement_data_df = pd.read_csv("C:/Users/aadil/Downloads/engagement_data.csv")
engagement_data_df.to_sql("engagement_data", conn, if_exists="replace", index=Fa
```

Out[81]: 100

In [83]:
```
query = """
SELECT
    p.ProductID,
    p.ProductName,
    ROUND(AVG(r.Rating), 2) AS AvgRating,
    COUNT(r.ReviewID) AS TotalReviews,
    COALESCE(SUM(CAST(substr(ed.ViewsClicksCombined, 1, instr(ed.ViewsClicksComb
    COALESCE(SUM(CAST(substr(ed.ViewsClicksCombined, instr(ed.ViewsClicksCombine
FROM products p
LEFT JOIN customer_reviews r ON p.ProductID = r.ProductID
LEFT JOIN engagement_data ed ON p.ProductID = ed.ProductID
GROUP BY p.ProductID, p.ProductName
ORDER BY AvgRating DESC;
"""
product_performance_df = pd.read_sql_query(query, conn)
product_performance_df.head()
```

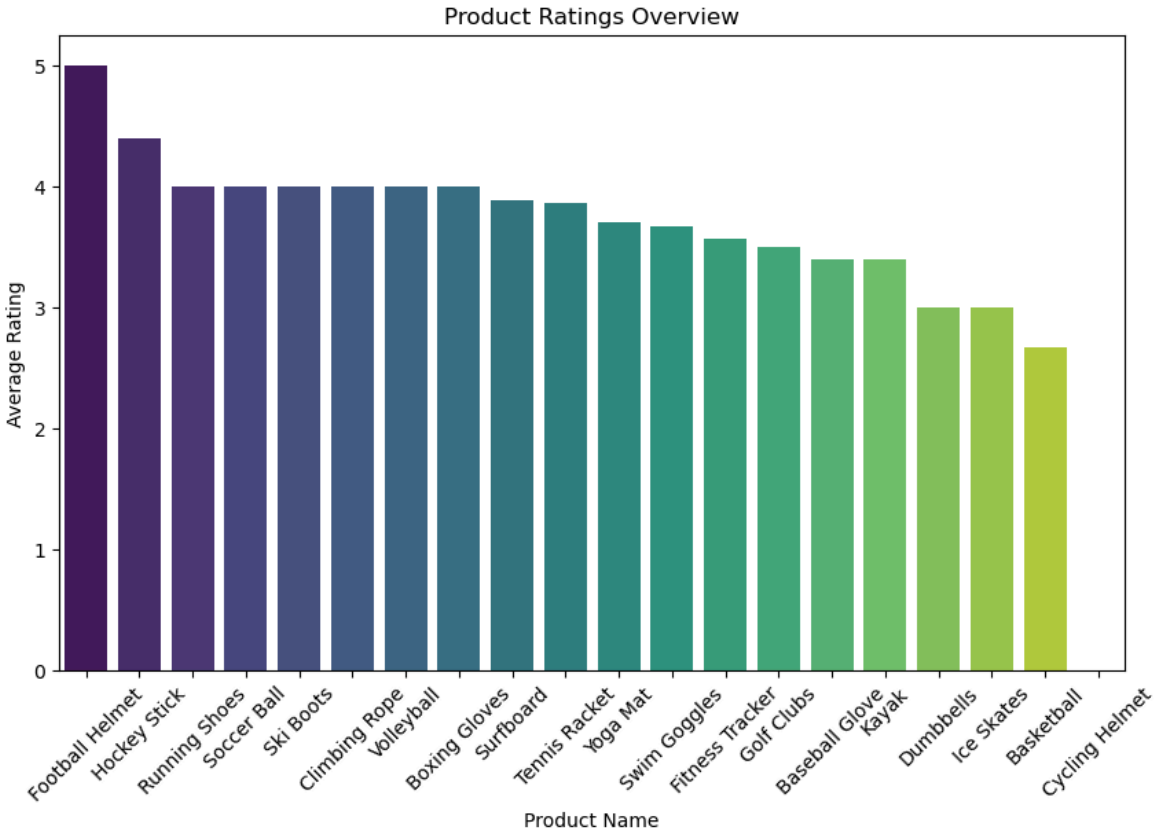
Out[83]:

	ProductID	ProductName	AvgRating	TotalReviews	TotalViews	TotalClicks
0	8	Football Helmet	5.0	21	41289	9669
1	19	Hockey Stick	4.4	25	77235	12740
2	1	Running Shoes	4.0	20	92632	24796
3	5	Soccer Ball	4.0	6	7245	540
4	11	Ski Boots	4.0	12	30654	7518

In [87]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Set figure size
plt.figure(figsize=(10, 6))

# Plot bar chart for Avg Rating
sns.barplot(x='ProductName', y='AvgRating', data=product_performance_df, palette

# Labels and title
plt.xlabel("Product Name")
plt.ylabel("Average Rating")
plt.title("Product Ratings Overview")
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility

# Show plot
plt.show()
```

```
C:\Users\aadil\AppData\Local\Temp\ipykernel_2180\1250237080.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x='ProductName', y='AvgRating', data=product_performance_df, palett
e='viridis')
```



In [ ]:
```
#market effectiveness
#calculating customer retention rate

#Customer Retention Rate

#Retention Rate = (Repeat Customers / Total Customers) × 100
#This helps understand how many customers return to shop again.
```

In [103…
```
customer_journey_df = pd.read_csv("C:/Users/aadil/Downloads/customer_journey.csv
customer_journey_df.to_sql("customer_journey", conn, if_exists="replace", index=
```

Out[103… 100

In [111…
```
query = """
SELECT
    COUNT(DISTINCT CASE WHEN OrderCount > 1 THEN CustomerID END) AS RepeatCustom
    COUNT(DISTINCT CustomerID) AS TotalCustomers,
    ROUND((COUNT(DISTINCT CASE WHEN OrderCount > 1 THEN CustomerID END) * 100.0)
FROM (
    SELECT CustomerID, COUNT(CustomerID) AS OrderCount
    FROM customer_journey
    GROUP BY CustomerID
) subquery;
"""

retention_rate_df = pd.read_sql_query(query, conn)
retention_rate_df
```

Out[111…

	RepeatCustomers	TotalCustomers	RetentionRate
0	26	65	40.0

In [ ]:
```
# First time buyer and repeated buyer
```

In [123…
```
query_repeat_vs_first_time = """
SELECT
    CASE
        WHEN purchase_count > 1 THEN 'Repeat Buyer'
        ELSE 'First-Time Buyer'
    END AS BuyerType,
    COUNT(CustomerID) AS CustomerCount
FROM (
    SELECT CustomerID, COUNT(CustomerID) AS purchase_count
    FROM customer_journey
    GROUP BY CustomerID
) subquery
GROUP BY BuyerType;
"""

repeat_vs_first_time_df = pd.read_sql_query(query_repeat_vs_first_time, conn)
repeat_vs_first_time_df
```

Out[123…

	BuyerType	CustomerCount
0	First-Time Buyer	39
1	Repeat Buyer	26

In [ ]:
```
#best performing products per region
```

In [131…
```
query_best_products_region = """
SELECT c.GeographyID, cj.ProductID, COUNT(cj.CustomerID) AS TotalSales
FROM customer_journey cj
JOIN customers c ON cj.CustomerID = c.CustomerID
GROUP BY c.GeographyID, cj.ProductID
ORDER BY c.GeographyID, TotalSales DESC;
"""
```

```
best_products_region_df = pd.read_sql_query(query_best_products_region, conn)
best_products_region_df
```

Out[131…

	GeographyID	ProductID	TotalSales
0	1	19	2
1	1	18	1
2	1	17	1
3	1	16	1
4	1	15	1
...	...	...	...
75	10	1	3
76	10	16	1
77	10	15	1
78	10	11	1
79	10	4	1

80 rows × 3 columns

In [135…

```
query_best_products_region = """
SELECT c.GeographyID, cj.ProductID, p.ProductName, COUNT(cj.CustomerID) AS Total
FROM customer_journey cj
JOIN customers c ON cj.CustomerID = c.CustomerID
JOIN products p ON cj.ProductID = p.ProductID
GROUP BY c.GeographyID, cj.ProductID
ORDER BY c.GeographyID, TotalSales DESC;
"""

best_products_region_df = pd.read_sql_query(query_best_products_region, conn)
best_products_region_df
```

Out[135…

	GeographyID	ProductID	ProductName	TotalSales
0	1	19	Hockey Stick	2
1	1	18	Volleyball	1
2	1	17	Surfboard	1
3	1	16	Kayak	1
4	1	15	Climbing Rope	1
...	...	...	...	...
75	10	1	Running Shoes	3
76	10	16	Kayak	1
77	10	15	Climbing Rope	1
78	10	11	Ski Boots	1
79	10	4	Dumbbells	1

80 rows × 4 columns

In [145…

```
query_best_performing_products = """
WITH ranked_products AS (
    SELECT
        c.GeographyID,
        cj.ProductID,
        p.ProductName,
        COUNT(cj.CustomerID) AS TotalSales,
        RANK() OVER (PARTITION BY c.GeographyID ORDER BY COUNT(cj.CustomerID) DE
    FROM customer_journey cj
    JOIN customers c ON cj.CustomerID = c.CustomerID
    JOIN products p ON cj.ProductID = p.ProductID
    GROUP BY c.GeographyID, cj.ProductID
)
SELECT GeographyID, ProductID, ProductName, TotalSales
FROM ranked_products
WHERE rnk = 1;
"""

best_performing_products_df = pd.read_sql_query(query_best_performing_products,
best_performing_products_df
```

Out[145…

	GeographyID	ProductID	ProductName	TotalSales
0	1	19	Hockey Stick	2
1	2	20	Boxing Gloves	2
2	3	14	Cycling Helmet	1
3	3	12	Ice Skates	1
4	3	11	Ski Boots	1
5	3	9	Baseball Glove	1
6	3	8	Football Helmet	1
7	3	4	Dumbbells	1
8	4	8	Football Helmet	3
9	5	10	Golf Clubs	3
10	6	15	Climbing Rope	2
11	7	6	Tennis Racket	2
12	7	2	Fitness Tracker	2
13	8	20	Boxing Gloves	1
14	8	15	Climbing Rope	1
15	8	12	Ice Skates	1
16	8	8	Football Helmet	1
17	9	18	Volleyball	3
18	10	20	Boxing Gloves	3
19	10	1	Running Shoes	3

In [ ]:
```
#best 5 selling products per region
```

In [149…
```
query_top_5_products = """
WITH ranked_products AS (
    SELECT
        c.GeographyID,
        cj.ProductID,
        p.ProductName,
        COUNT(cj.CustomerID) AS TotalSales,
        RANK() OVER (PARTITION BY c.GeographyID ORDER BY COUNT(cj.CustomerID) DE
    FROM customer_journey cj
    JOIN customers c ON cj.CustomerID = c.CustomerID
    JOIN products p ON cj.ProductID = p.ProductID
    GROUP BY c.GeographyID, cj.ProductID
)
SELECT GeographyID, ProductID, ProductName, TotalSales
FROM ranked_products
WHERE rnk <= 5;
"""
```

```
top_5_products_df = pd.read_sql_query(query_top_5_products, conn)
top_5_products_df
```

Out[149…

	GeographyID	ProductID	ProductName	TotalSales
0	1	19	Hockey Stick	2
1	1	18	Volleyball	1
2	1	17	Surfboard	1
3	1	16	Kayak	1
4	1	15	Climbing Rope	1
…	…	…	…	…
75	10	1	Running Shoes	3
76	10	16	Kayak	1
77	10	15	Climbing Rope	1
78	10	11	Ski Boots	1
79	10	4	Dumbbells	1

80 rows × 4 columns

In [ ]:
```
#
```

In [151…
```
query_avg_rating = """
SELECT
    cj.ProductID,
    p.ProductName,
    COUNT(cr.ReviewID) AS TotalReviews,
    AVG(cr.Rating) AS AvgRating
FROM customer_journey cj
JOIN products p ON cj.ProductID = p.ProductID
LEFT JOIN customer_reviews cr ON cj.ProductID = cr.ProductID
GROUP BY cj.ProductID;
"""

avg_rating_df = pd.read_sql_query(query_avg_rating, conn)
avg_rating_df
```

Out[151…

	ProductID	ProductName	TotalReviews	AvgRating
0	1	Running Shoes	28	4.000000
1	2	Fitness Tracker	42	3.571429
2	3	Yoga Mat	14	3.714286
3	4	Dumbbells	25	3.000000
4	5	Soccer Ball	9	4.000000
5	6	Tennis Racket	28	3.857143
6	7	Basketball	3	2.666667
7	8	Football Helmet	27	5.000000
8	9	Baseball Glove	25	3.400000
9	10	Golf Clubs	28	3.500000
10	11	Ski Boots	30	4.000000
11	12	Ice Skates	8	3.000000
12	13	Swim Goggles	27	3.666667
13	14	Cycling Helmet	0	NaN
14	15	Climbing Rope	36	4.000000
15	16	Kayak	50	3.400000
16	17	Surfboard	45	3.888889
17	18	Volleyball	18	4.000000
18	19	Hockey Stick	25	4.400000
19	20	Boxing Gloves	18	4.000000

In [ ]: