pip install pandas nltk pyodbc sqlalchemy

In [1]:
```
import pandas as pd
import pyodbc
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

Download the VADER lexicon for sentiment analysis if not already present.

In [2]:
```
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data...
```

Out[2]: True

In [7]:
```
# Define a function to fetch data from a SQL database using a SQL query
def fetch_data_from_sql():
    # Define the connection string with parameters for the database connection
    conn_str = (
        "Driver={SQL Server};"  # Specify the driver for SQL Server
        "Server=localhost\SQLEXPRESS01;"  # Specify your SQL Server instance
        "Database=PortfolioProject_MarketingAnalytics;"  # Specify the database
        "Trusted_Connection=yes;"  # Use Windows Authentication for the connecti
    )
    # Establish the connection to the database
    conn = pyodbc.connect(conn_str)

    # Define the SQL query to fetch customer reviews data
    query = "SELECT ReviewID, CustomerID, ProductID, ReviewDate, Rating, ReviewT

    df = pd.read_sql(query, conn)

    # Close the connection to free up resources
    conn.close()
    return df
```

In [8]:
```
# Fetch the customer reviews data from the SQL database
customer_reviews_df = fetch_data_from_sql()
```

```
C:\Users\ayush\AppData\Local\Temp\ipykernel_18676\2394557839.py:16: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database strin
g URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please c
onsider using SQLAlchemy.
  df = pd.read_sql(query, conn)
```

In [9]:
```
# Initialize the VADER sentiment intensity analyzer for analyzing the sentiment
sia = SentimentIntensityAnalyzer()
```

In [10]:
```
# Define a function to calculate sentiment scores using VADER
def calculate_sentiment(review):
    # Get the sentiment scores for the review text
```

```
    sentiment = sia.polarity_scores(review)
    # Return the compound score, which is a normalized score between -1 (most ne
    return sentiment['compound']
```

In [11]:
```
# Define a function to categorize sentiment using both the sentiment score and t
def categorize_sentiment(score, rating):
    # Use both the text sentiment score and the numerical rating to determine se
    if score > 0.05:  # Positive sentiment score
        if rating >= 4:
            return 'Positive'  # High rating and positive sentiment
        elif rating == 3:
            return 'Mixed Positive'  # Neutral rating but positive sentiment
        else:
            return 'Mixed Negative'  # Low rating but positive sentiment
    elif score < -0.05:  # Negative sentiment score
        if rating <= 2:
            return 'Negative'  # Low rating and negative sentiment
        elif rating == 3:
            return 'Mixed Negative'  # Neutral rating but negative sentiment
        else:
            return 'Mixed Positive'  # High rating but negative sentiment
    else:  # Neutral sentiment score
        if rating >= 4:
            return 'Positive'  # High rating with neutral sentiment
        elif rating <= 2:
            return 'Negative'  # Low rating with neutral sentiment
        else:
            return 'Neutral'  # Neutral rating and neutral sentiment
```

In [12]:
```
# Define a function to bucket sentiment scores into text ranges
def sentiment_bucket(score):
    if score >= 0.5:
        return '0.5 to 1.0'  # Strongly positive sentiment
    elif 0.0 <= score < 0.5:
        return '0.0 to 0.49'  # Mildly positive sentiment
    elif -0.5 <= score < 0.0:
        return '-0.49 to 0.0'  # Mildly negative sentiment
    else:
        return '-1.0 to -0.5'  # Strongly negative sentiment
```

In [13]:
```
# Apply sentiment analysis to calculate sentiment scores for each review
customer_reviews_df['SentimentScore'] = customer_reviews_df['ReviewText'].apply(
```

In [14]:
```
# Apply sentiment categorization using both text and rating
customer_reviews_df['SentimentCategory'] = customer_reviews_df.apply(
    lambda row: categorize_sentiment(row['SentimentScore'], row['Rating']), axis
```

In [15]:
```
# Apply sentiment bucketing to categorize scores into defined ranges
customer_reviews_df['SentimentBucket'] = customer_reviews_df['SentimentScore'].a
```

In [16]:
```
# Display the first few rows of the DataFrame with sentiment scores, categories,
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  SentimentScore SentimentCategory  \
0   Average experience, nothing special.         -0.3089    Mixed Negative
1           The quality is top-notch.          0.0000          Positive
2   Five stars for the quick delivery.         0.0000          Positive
3  Good quality, but could be cheaper.         0.2382    Mixed Positive
4   Average experience, nothing special.         -0.3089    Mixed Negative

  SentimentBucket
0    -0.49 to 0.0
1     0.0 to 0.49
2     0.0 to 0.49
3     0.0 to 0.49
4    -0.49 to 0.0
```

In [17]:
```
# Save the DataFrame with sentiment scores, categories, and buckets to a new CSV
customer_reviews_df.to_csv('fact_customer_reviews_with_sentiment.csv', index=Fal
```