REAL-TIME DATA ENGINEERING USE CASES, ALONG WITH SAMPLE DATA AND PROBLEM STATEMENTS

Use Case 1: Real-Time Sales Data Analysis and Insights

Problem Statement:

You are working for a retail company that wants to gain insights into their sales performance across different stores and products. The company has been collecting real-time sales data from multiple stores, and your task is to clean, process, and analyze the data to derive meaningful insights and create visual representations.

Sample Data Structure:
Sales Data File (sales_data.csv)
Date: (e.g., "2024-11-01", "2024-11-02")
Store ID: (e.g., "S001", "S002")
Product ID: (e.g., "P001", "P002")
Units Sold: (e.g., 50, 75)
Sales Amount: (e.g., 500.75, 1200.50)
Discount Applied: (e.g., 10.5, 5.0)
Customer Segment: (e.g., "Regular", "Premium", "New")

Tasks for Students:
1. **Data Cleaning: Handle missing values, incorrect formats, and outliers.**
2. **Data Aggregation: Aggregate sales data at different levels, such as by date, store, and product.**
3. **Analysis:**
   - **Calculate total sales and average sales per product.**
   - **Identify the store with the highest sales performance.**
   - **Analyze the impact of discounts on sales amounts.**
4. **Visualization:**
   Use tools like Pandas and Matplotlib to visualize:
   - **Sales trends over time.**
   - **Sales distribution across different stores.**
   - **Performance comparison of products.**

Expected Outcome:

Students should present a cleaned and well-structured dataset along with meaningful visualizations and insights that can help the company make data-driven decisions.

Solution:

Step 1: Import the necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import numpy as np
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
sns.set_style("whitegrid")
```

Step 2: Sample dataset creation

```
np.random.seed(42)
sample_data = pd.DataFrame({
   'Date': pd.date_range(start='2024-11-01', end='2024-11-10', freq='D').repeat(5),
   'Store_ID': np.random.choice(['S001', 'S002', 'S003', 'S004', 'S005'], size=50),
   'Product_ID': np.random.choice(['P001', 'P002', 'P003', 'P004'], size=50),
   'Units_Sold': np.random.randint(10, 100, size=50),
   'Sales_Amount': np.random.uniform(100, 2000, size=50),
   'Discount_Applied': np.random.uniform(0, 20, size=50),
   'Customer_Segment': np.random.choice(['Regular', 'Premium', 'New'], size=50)
})
sample_data.to_csv('sales_data.csv', index=False)
sample_data.head()
```

	Date	Store_ID	Product_ID	Units_Sold	Sales_Amount	Discount_Applied	Customer_Segment
0	2024-11-01	S004	P004	98	310.692560	8.220740	Premium
1	2024-11-01	S005	P003	69	934.739354	0.661015	Regular
2	2024-11-01	S003	P004	50	483.266484	6.901425	New
3	2024-11-01	S005	P003	38	1801.950832	12.687027	Regular
4	2024-11-01	S005	P004	24	1003.203424	13.614109	Premium

Step 3: Load the dataset

```
# Load the data
data_path = 'sales_data.csv'
raw_data = pd.read_csv(data_path)

raw_data.head()
```

	Date	Store_ID	Product_ID	Units_Sold	Sales_Amount	Discount_Applied	Customer_Segment
0	2024-11-01	S004	P004	98	310.692560	8.220740	Premium
1	2024-11-01	S005	P003	69	934.739354	0.661015	Regular
2	2024-11-01	S003	P004	50	483.266484	6.901425	New
3	2024-11-01	S005	P003	38	1801.950832	12.687027	Regular
4	2024-11-01	S005	P004	24	1003.203424	13.614109	Premium

Step 4: Data Cleaning

- Handle missing values, incorrect formats, and outliers.

```
# Data Cleaning and Handling Missing Values
df = raw_data.copy()
df['Date'] = pd.to_datetime(df['Date'])
df['Units_Sold'].fillna(df['Units_Sold'].median(), inplace=True)
df['Sales_Amount'].fillna(df['Sales_Amount'].median(), inplace=True)
df['Discount_Applied'].fillna(0, inplace=True)

# Handle Outliers Using IQR Method
for col in ['Units_Sold', 'Sales_Amount', 'Discount_Applied']:
   Q1 = df[col].quantile(0.25)
   Q3 = df[col].quantile(0.75)
   IQR = Q3 - Q1
   lower_bound = Q1 - 1.5 * IQR
   upper_bound = Q3 + 1.5 * IQR
   df[col] = df[col].clip(lower_bound, upper_bound)
clean_data = df
clean_data.head()
```

	Date	Store_ID	Product_ID	Units_Sold	Sales_Amount	Discount_Applied	Customer_Segment
0	2024-11-01	S004	P004	98	310.692560	8.220740	Premium
1	2024-11-01	S005	P003	69	934.739354	0.661015	Regular
2	2024-11-01	S003	P004	50	483.266484	6.901425	New
3	2024-11-01	S005	P003	38	1801.950832	12.687027	Regular
4	2024-11-01	S005	P004	24	1003.203424	13.614109	Premium

Step 5: Data Aggregation

- Aggregate sales data at different levels, such as by date, store, and product

```
# Aggregate Sales Data
aggregations = {
   'daily': clean_data.groupby('Date').agg({
       'Sales_Amount': 'sum',
```

```
        'Units_Sold': 'sum'
    }).reset_index(),

    'store': clean_data.groupby('Store_ID').agg({
        'Sales_Amount': 'sum',
        'Units_Sold': 'sum',
        'Discount_Applied': 'mean'
    }).reset_index(),

    'product': clean_data.groupby('Product_ID').agg({
        'Sales_Amount': ['sum', 'mean'],
        'Units_Sold': ['sum', 'mean'],
        'Discount_Applied': 'mean'
    }).reset_index()
}

aggregations['daily'].head(), aggregations['store'].head(),
aggregations['product'].head()
```

```
(         Date  Sales_Amount  Units_Sold
 0 2024-11-01   4533.852653         279
 1 2024-11-02   5330.525008         324
 2 2024-11-03   6808.062764         293
 3 2024-11-04   3730.868303         215
 4 2024-11-05   4494.789939         159,
   Store_ID  Sales_Amount  Units_Sold  Discount_Applied
 0     S001   5289.787043         303          8.959271
 1     S002  10239.253317         527         11.173990
 2     S003  10507.868275         673          9.522757
 3     S004  10266.176143         760         13.270385
 4     S005  11209.450991         312          9.875348,
   Product_ID  Sales_Amount              Units_Sold             Discount_Applied
                        sum         mean        sum       mean             mean
 0       P001   7533.813584  1076.259083        384  54.857143        11.548957
 1       P002  17926.919833   995.939991        729  40.500000        12.388022
 2       P003   8013.296590   890.366288        519  57.666667         7.411671
 3       P004  14038.505761   877.406610        943  58.937500        10.651170)
```

Step 6: Data Analysis

- Calculate total sales and average sales per product.
- Identify the store with the highest sales performance.
- Analyze the impact of discounts on sales amounts.

```
# Sales Analysis Insights
analysis = {
   'total_sales': clean_data['Sales_Amount'].sum(),
   'avg_sales_per_product': clean_data.groupby('Product_ID')['Sales_Amount'].mean(),
   'top_store': clean_data.groupby('Store_ID')['Sales_Amount'].sum().idxmax(),
   'discount_impact': clean_data.groupby(
       pd.qcut(clean_data['Discount_Applied'], 4)
   )['Sales_Amount'].mean()
}

# Display Analysis Results
print(f"Total Sales: ${analysis['total_sales']:,.2f}")
print(f"Top Performing Store: {analysis['top_store']}")
print("\nAverage Sales per Product:")
for product, avg_sales in analysis['avg_sales_per_product'].items():
   print(f"{product}: ${avg_sales:,.2f}")
```

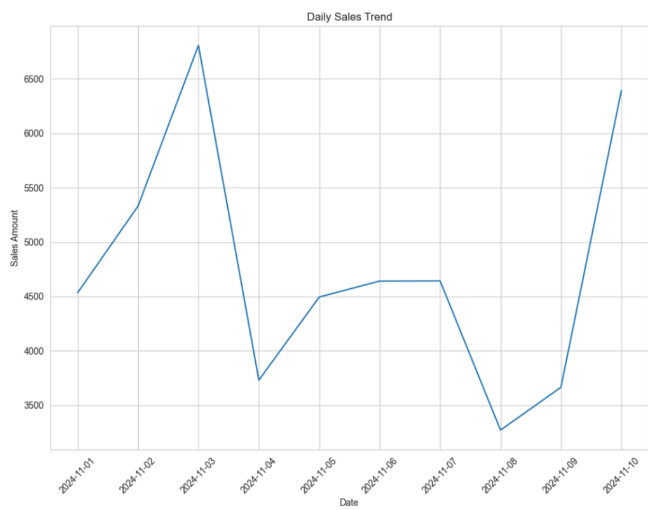
```
Total Sales: $47,512.54
Top Performing Store: S005

Average Sales per Product:
P001: $1,076.26
P002: $995.94
P003: $890.37
P004: $877.41
```

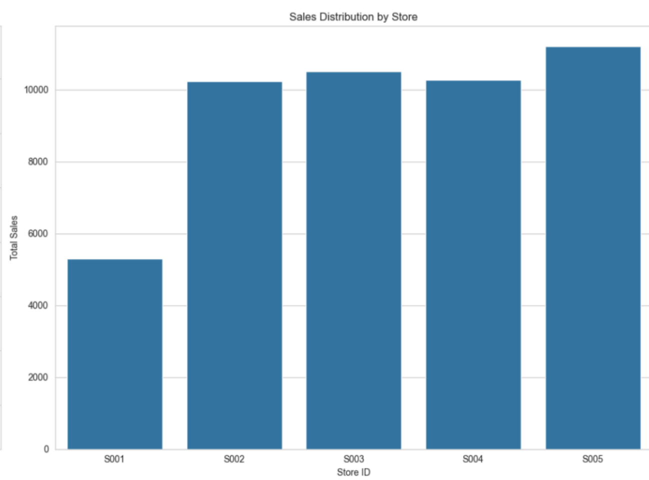
Step 7: Data Visualization (using pandas, matplotlib and seaborn)

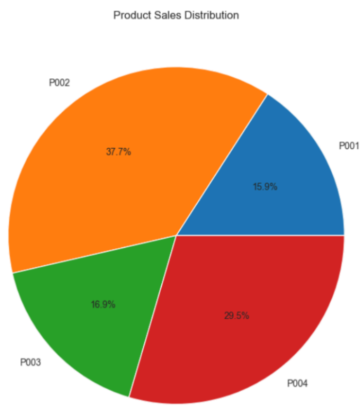
- Sales trends over time.
- Sales distribution across different stores.
- Performance comparison of products.

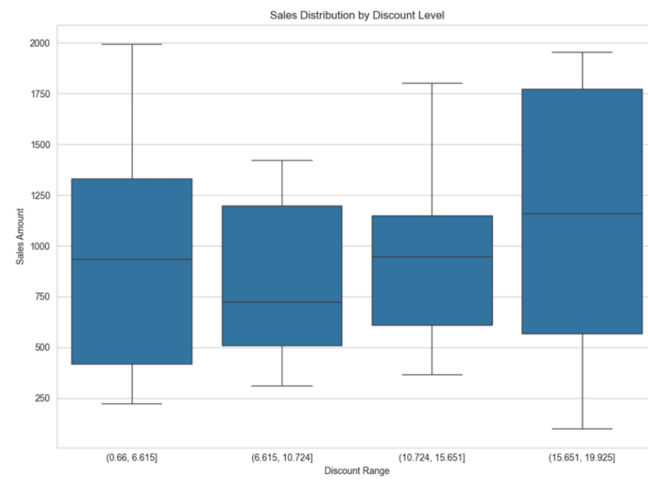
```
fig = plt.figure(figsize=(20, 15))

# Sales Trends Over Time
plt.subplot(2, 2, 1)
daily_sales = clean_data.groupby('Date')['Sales_Amount'].sum()
sns.lineplot(data=daily_sales)
plt.title('Daily Sales Trend')
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Sales Amount')
```

```
# Sales Distribution Across Stores
plt.subplot(2, 2, 2)
store_sales = clean_data.groupby('Store_ID')['Sales_Amount'].sum()
sns.barplot(x=store_sales.index, y=store_sales.values)
plt.title('Sales Distribution by Store')
plt.xlabel('Store ID')
plt.ylabel('Total Sales')

# Product Performance Comparison
plt.subplot(2, 2, 3)
product_sales = clean_data.groupby('Product_ID')['Sales_Amount'].sum()
plt.pie(product_sales.values, labels=product_sales.index, autopct='%1.1f%%')
plt.title('Product Sales Distribution')

# Discount Impact Analysis
plt.subplot(2, 2, 4)
sns.boxplot(x=pd.qcut(clean_data['Discount_Applied'], 4),
y=clean_data['Sales_Amount'])
plt.title('Sales Distribution by Discount Level')
plt.xlabel('Discount Range')
plt.ylabel('Sales Amount')

plt.tight_layout()
plt.show()
```






Product Sales Distribution
P001
15.9%
P002
37.7%
P003
16.9%
P004
29.5%


Sales Distribution by Discount Level
Sales Amount
Discount Range
(0.66, 6.615]
(6.615, 10.724]
(10.724, 15.651]
(15.651, 19.925]

Use Case 2: Data Pipeline for Customer Feedback Analysis

Problem Statement:

A tech company collects customer feedback data from different channels such as emails, social media, and surveys. The feedback contains structured (ratings) and unstructured (comments) data. Your job is to build a data pipeline to ingest, clean, and analyze this feedback to identify key trends and sentiment.

Sample Data Structure:
Feedback Data File (feedback_data.csv)
Customer ID: (e.g., "C12345", "C67890")
Feedback Channel: (e.g., "Email", "Social Media", "Survey")
Rating (1-5): (e.g., 3, 4, 5)
Comment:
(e.g., "The service was great!", "Could improve product quality.")
Date:
(e.g., "2024-11-01", "2024-11-02")

Tasks for Students:

1. Data Ingestion: Use tools like Apache Kafka or Apache Spark to ingest data in real time.
2. Data Cleaning: Process and clean the comments by removing unnecessary characters, correcting misspellings, and handling missing values.
3. Sentiment Analysis: Perform sentiment analysis on the unstructured text data to classify feedback as positive, neutral, or negative.
4. Trend Analysis:

- **Analyze feedback trends over time.**
- **Determine which feedback channels generate the most negative or positive comments.**
- **Calculate average ratings per channel and identify areas for improvement.**

5. Data Visualization: Create visualizations to show sentiment distribution, feedback trends, and channel performance.

Expected Outcome:

Students should create an end-to-end data pipeline that processes feedback data and provides insights through sentiment analysis and trend visualizations. The final output should be a report with actionable insights for improving customer satisfaction.

<u>Solution:</u>

Step 1: Install and import the necessary libraries

```
%pip install pyspark textblob
```

```
from pyspark.sql import SparkSession
import pandas as pd
from pyspark.sql.types import (
   StructType, StructField, StringType,
   IntegerType, TimestampType
)
from pyspark.sql.functions import col, to_timestamp, udf, count, avg
from pyspark.sql import functions as F
import datetime
import random
from textblob import TextBlob
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Initialize the Spark Session

```
# Initialize Spark session
spark = SparkSession.builder \
   .appName("CustomerFeedbackAnalysis") \
   .config("spark.driver.memory", "2g") \
   .getOrCreate()

# Define schema for feedback data
feedback_schema = StructType([
   StructField("Customer_ID", StringType(), False),
   StructField("Feedback_Channel", StringType(), False),
   StructField("Rating", IntegerType(), False),
   StructField("Comment", StringType(), True),
   StructField("Date", TimestampType(), False)
])
```

Step 3: Data Ingestion:

- Using Apache Spark to ingest data in real time.

Generate Sample Data in Spark

```
def generate_spark_data(num_records=1000):
   """Generate sample data in Spark"""
   channels = ["Email", "Social Media", "Survey"]
   comments = ["Great service!", "Need improvement", "Average experience",
"Fantastic!", "Horrible service!", "Will recommend!"]

   data = []
   current_date = datetime.datetime.now()

   for _ in range(num_records):
       data.append((
           f"C{random.randint(10000, 99999)}",
           random.choice(channels),
           random.randint(1, 5),
           random.choice(comments),
           (current_date - datetime.timedelta(days=random.randint(0, 30)))
       ))

   # Create Spark DataFrame
   return spark.createDataFrame(data, ["Customer_ID", "Feedback_Channel", "Rating",
"Comment", "Date"])

# Generate the Spark DataFrame
spark_df = generate_spark_data(10000)
spark_df.show(5)
```

```
+-----------+----------------+------+------------------+--------------------+
|Customer_ID|Feedback_Channel|Rating|           Comment|                Date|
+-----------+----------------+------+------------------+--------------------+
|     C10750|           Email|     4|Average experience|2024-10-12 13:13:...|
|     C50931|    Social Media|     3|    Great service!|2024-10-19 13:13:...|
|     C36204|           Email|     5|    Great service!|2024-10-25 13:13:...|
|     C56077|           Email|     2|        Fantastic!|2024-10-10 13:13:...|
|     C99219|           Email|     3|Average experience|2024-10-21 13:13:...|
+-----------+----------------+------+------------------+--------------------+
only showing top 5 rows
```

Step 4: Convert Spark DataFrame to Pandas DataFrame

```
def spark_to_pandas(spark_df):
   """Convert Spark DataFrame to Pandas DataFrame"""
   try:
       pandas_df = spark_df.toPandas()
       print(f"Successfully converted {len(pandas_df)} records to pandas DataFrame")
       return pandas_df
   except Exception as e:
       print(f"Error converting to pandas: {e}")
       return None

# Convert Spark DataFrame to Pandas DataFrame
pandas_df = spark_to_pandas(spark_df.limit(1000))
print(pandas_df.head())
```

```
Successfully converted 1000 records to pandas DataFrame
  Customer_ID Feedback_Channel  Rating             Comment  \
0      C10750            Email       4  Average experience
1      C50931     Social Media       3       Great service!
2      C36204            Email       5       Great service!
3      C56077            Email       2           Fantastic!
4      C99219            Email       3  Average experience

                        Date
0 2024-10-12 13:13:45.299460
1 2024-10-19 13:13:45.299460
2 2024-10-25 13:13:45.299460
3 2024-10-10 13:13:45.299460
4 2024-10-21 13:13:45.299460
```

Step 5: Data Cleaning

- Process and clean the comments by removing unnecessary characters, correcting misspellings, and handling missing values.

```
def clean_data(df):
   """Clean the feedback comments"""
   # Remove unnecessary characters and handle missing values
   df['Comment'] = df['Comment'].str.replace(r'[^a-zA-Z0-9\s]', '',
regex=True).str.strip()
   df['Comment'] = df['Comment'].fillna('No Comment')
   return df
```

```
# Clean the data
pandas_df = clean_data(pandas_df)
```

Step 6: Sentiment Analysis:

- Perform sentiment analysis on the unstructured text data to classify feedback as positive, neutral, or negative.

```
def analyze_sentiment(comment):
   """Analyze sentiment of the comment"""
   analysis = TextBlob(comment)
   if analysis.sentiment.polarity > 0:
       return 'Positive'
   elif analysis.sentiment.polarity == 0:
       return 'Neutral'
   else:
       return 'Negative'

# Perform sentiment analysis
sentiment_udf = udf(analyze_sentiment, StringType())
spark_df = spark_df.withColumn("Sentiment", sentiment_udf(col("Comment")))
```

```
def perform_analysis(pandas_df):
   """Perform basic analysis on the pandas DataFrame"""
   if pandas_df is None:
       print("No data available for analysis")
       return None

   analysis = {
       'total_records': len(pandas_df),
       'channel_distribution': pandas_df['Feedback_Channel'].value_counts().to_dict(),
       'average_rating': pandas_df['Rating'].mean(),
       'rating_distribution':
pandas_df['Rating'].value_counts().sort_index().to_dict()
   }

   return analysis

# Perform analysis on the processed Pandas DataFrame
analysis = perform_analysis(pandas_df)
if analysis:
   print("\nAnalysis Results:")
```

```
    for key, value in analysis.items():
        print(f"{key}: {value}")
```

```
Analysis Results:
total_records: 1000
channel_distribution: {'Survey': 346, 'Email': 337, 'Social Media': 317}
average_rating: 2.967
rating_distribution: {1: 203, 2: 208, 3: 203, 4: 191, 5: 195}
```

Step 8: Trend Analysis:

- Analyze feedback trends over time.
- Determine which feedback channels generate the most negative or positive comments.
- Calculate average ratings per channel and identify areas for improvement.

```
# Trend analysis
def trend_analysis(spark_df):
    """Analyze trends over time and by channel"""
    # Convert date to date format
    spark_df = spark_df.withColumn("Date", to_timestamp(col("Date")))

    # Average ratings per channel
    avg_rating_per_channel =
spark_df.groupBy("Feedback_Channel").agg(avg("Rating").alias("Average_Rating"))
    avg_rating_per_channel.show()

    # Count of sentiments by channel
    sentiment_distribution = spark_df.groupBy("Feedback_Channel",
"Sentiment").agg(count("Sentiment").alias("Count"))
    sentiment_distribution.show()

    # Generate plots
    return avg_rating_per_channel, sentiment_distribution

avg_rating_per_channel, sentiment_distribution = trend_analysis(spark_df)
```

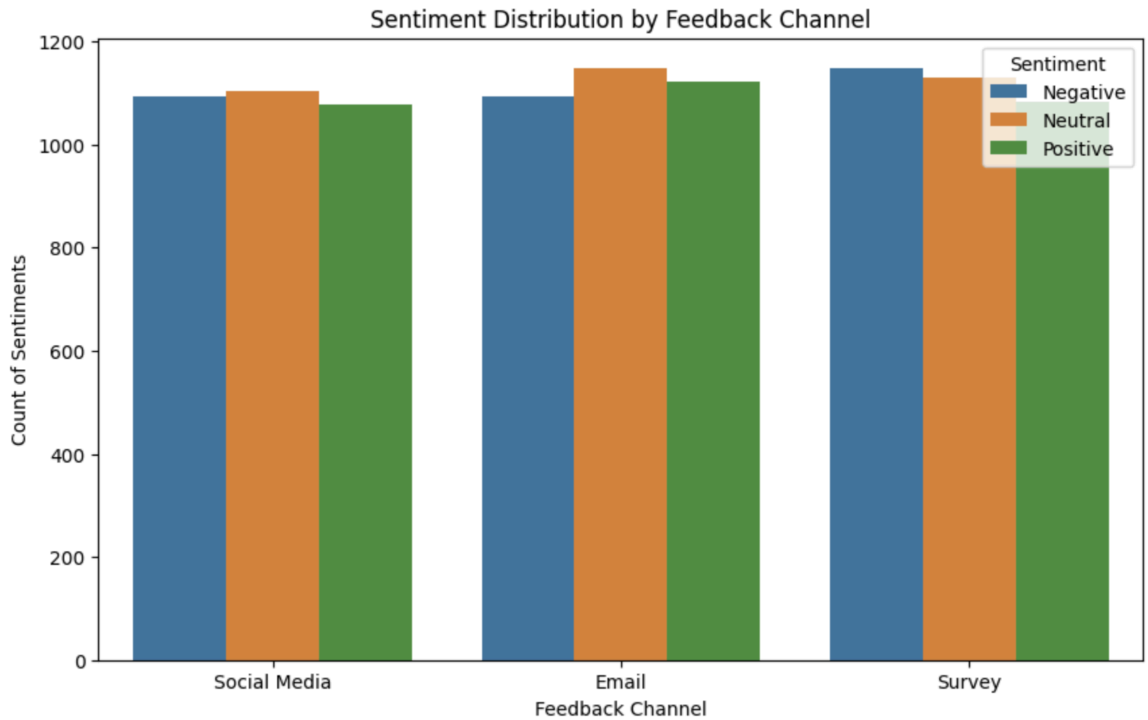
```
+---------------+------------------+
|Feedback_Channel|    Average_Rating|
+---------------+------------------+
|          Email| 2.982461355529132|
|   Social Media|2.9984728161270615|
|         Survey|3.0288518738845926|
+---------------+------------------+

[Stage 12:>                                                         (0 + 10) / 10]
+---------------+---------+-----+
|Feedback_Channel|Sentiment|Count|
+---------------+---------+-----+
|   Social Media| Negative| 1094|
|          Email|  Neutral| 1149|
|   Social Media|  Neutral| 1103|
|          Email| Positive| 1121|
|         Survey| Positive| 1084|
|   Social Media| Positive| 1077|
|         Survey| Negative| 1148|
|          Email| Negative| 1094|
|         Survey|  Neutral| 1130|
+---------------+---------+-----+
```

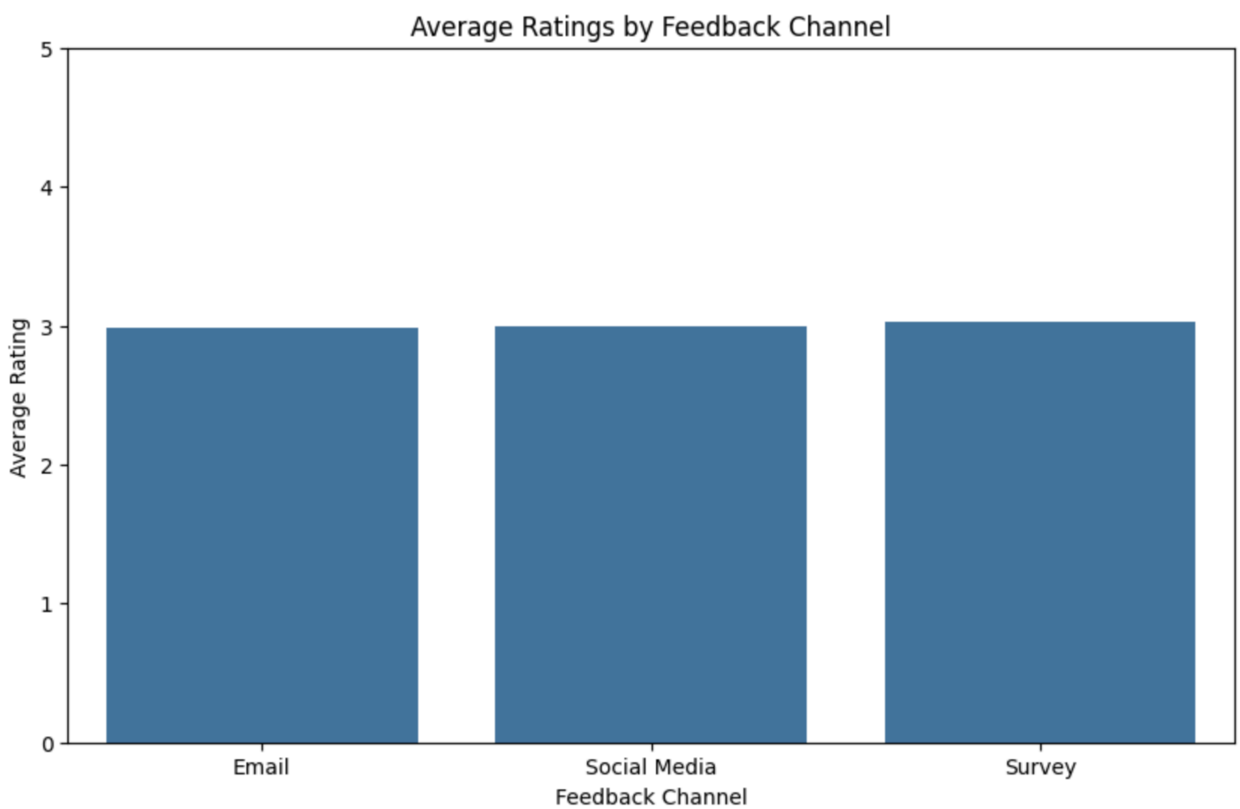
Step 9: Data Visualization:

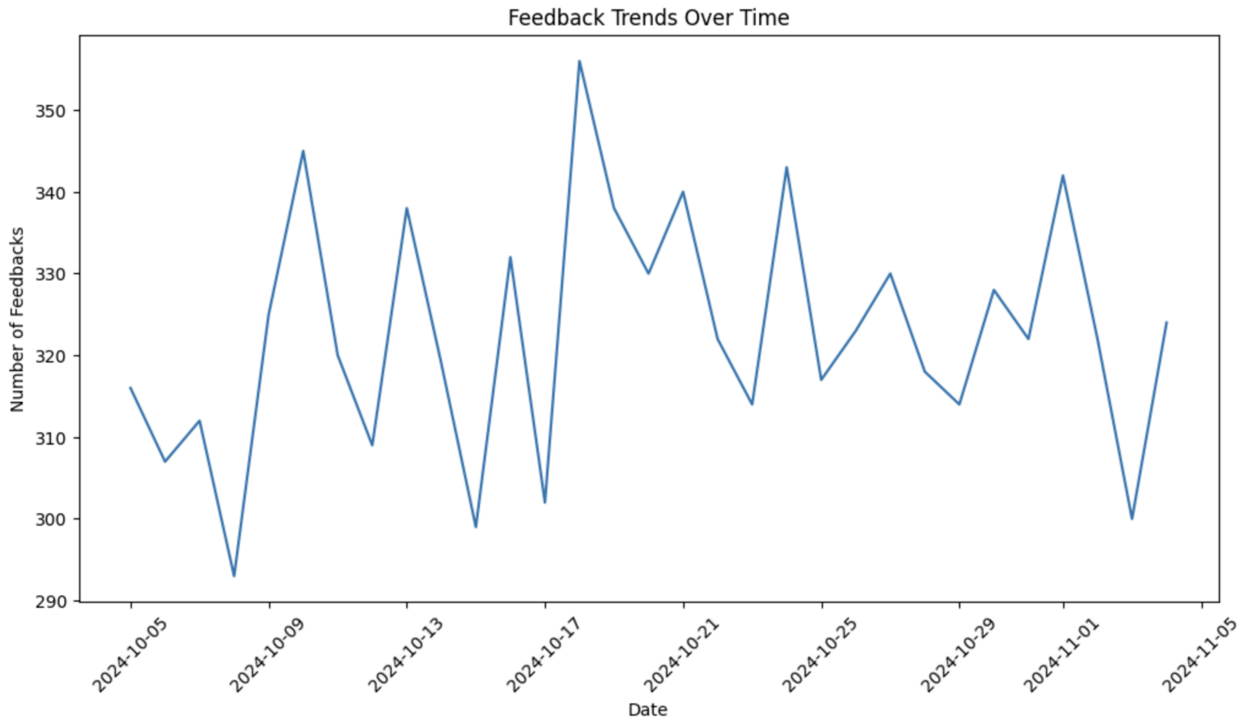
- Create visualizations to show sentiment distribution, feedback trends, and channel performance.

```
# Visualization Functions
def visualize_sentiment_distribution(sentiment_distribution):
   """Visualize sentiment distribution"""
   sentiment_pd = sentiment_distribution.toPandas()
   plt.figure(figsize=(10, 6))
   sns.barplot(x='Feedback_Channel', y='Count', hue='Sentiment', data=sentiment_pd)
   plt.title('Sentiment Distribution by Feedback Channel')
   plt.ylabel('Count of Sentiments')
   plt.xlabel('Feedback Channel')
   plt.legend(title='Sentiment')
   plt.show()

def visualize_average_ratings(avg_rating_per_channel):
   """Visualize average ratings by feedback channel"""
   avg_rating_pd = avg_rating_per_channel.toPandas()
   plt.figure(figsize=(10, 6))
```

```
    sns.barplot(x='Feedback_Channel', y='Average_Rating', data=avg_rating_pd)
    plt.title('Average Ratings by Feedback Channel')
    plt.ylabel('Average Rating')
    plt.xlabel('Feedback Channel')
    plt.ylim(0, 5)  # Set y-axis limit
    plt.show()

def visualize_trends_over_time(spark_df):
    """Visualize feedback trends over time"""
    trend_data =
spark_df.groupBy(F.to_date(col("Date")).alias("Date")).agg(count("Customer_ID").alias(
"Feedback_Count"))
    trend_data_pd = trend_data.toPandas()

    plt.figure(figsize=(12, 6))
    sns.lineplot(data=trend_data_pd, x='Date', y='Feedback_Count')
    plt.title('Feedback Trends Over Time')
    plt.ylabel('Number of Feedbacks')
    plt.xlabel('Date')
    plt.xticks(rotation=45)
    plt.show()

# Visualizations
visualize_sentiment_distribution(sentiment_distribution)
visualize_average_ratings(avg_rating_per_channel)
visualize_trends_over_time(spark_df)
```


Sentiment Distribution by Feedback Channel
Sentiment
Negative
Neutral
Positive
Count of Sentiments
1200
1000
800
600
400
200
0
Social Media
Email
Survey
Feedback Channel


Average Ratings by Feedback Channel
Average Rating
5
4
3
2
1
0
Email
Social Media
Survey
Feedback Channel


Feedback Trends Over Time
Number of Feedbacks
Date
290
300
310
320
330
340
350
2024-10-05
2024-10-09
2024-10-13
2024-10-17
2024-10-21
2024-10-25
2024-10-29
2024-11-01
2024-11-05