```
pip install openpyxl
```

```
/opt/anaconda3/lib/python3.12/pty.py:95: DeprecationWarning: This
process (pid=16869) is multi-threaded, use of forkpty() may lead to
deadlocks in the child.
  pid, fd = os.forkpty()
```

```
Requirement already satisfied: openpyxl in
/opt/anaconda3/lib/python3.12/site-packages (3.1.2)
Requirement already satisfied: et-xmlfile in
/opt/anaconda3/lib/python3.12/site-packages (from openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

```python
import pandas as pd
```

```python
import pandas as pd
# Load the dataset
file_path = "online_retail_II 2.xlsx" # Replace
data = pd.read_excel(file_path, sheet_name="Year 2010-2011")
# Display the first few rows
data.head()
```

```
/opt/anaconda3/lib/python3.12/site-packages/openpyxl/packaging/
core.py:99: DeprecationWarning: datetime.datetime.utcnow() is
deprecated and scheduled for removal in a future version. Use
timezone-aware objects to represent datetimes in UTC:
datetime.datetime.now(datetime.UTC).
  now = datetime.datetime.utcnow()
/opt/anaconda3/lib/python3.12/site-packages/openpyxl/packaging/core.py
:99: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
scheduled for removal in a future version. Use timezone-aware objects
to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  now = datetime.datetime.utcnow()
```

```
  Invoice StockCode                          Description  Quantity  \
0  536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1  536365     71053                  WHITE METAL LANTERN         6
2  536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3  536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4  536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  Price  Customer ID         Country
0 2010-12-01 08:26:00   2.55      17850.0  United Kingdom
1 2010-12-01 08:26:00   3.39      17850.0  United Kingdom
2 2010-12-01 08:26:00   2.75      17850.0  United Kingdom
3 2010-12-01 08:26:00   3.39      17850.0  United Kingdom
4 2010-12-01 08:26:00   3.39      17850.0  United Kingdom
```

```python
# Check the shape and data types
print(f"Dataset shape: {data.shape}")
print(data.info())
```

```python
# Check for missing values
print(data.isnull().sum())
# Summary statistics
data.describe()
```

```
Dataset shape: (541910, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541910 entries, 0 to 541909
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      541910 non-null  object
 1   StockCode    541910 non-null  object
 2   Description  540456 non-null  object
 3   Quantity     541910 non-null  int64
 4   InvoiceDate  541910 non-null  datetime64[ns]
 5   Price        541910 non-null  float64
 6   Customer ID  406830 non-null  float64
 7   Country      541910 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
None
Invoice             0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
Price               0
Customer ID    135080
Country             0
dtype: int64
```

```
            Quantity                    InvoiceDate          Price  \
count  541910.000000                         541910  541910.000000
mean        9.552234  2011-07-04 13:35:22.342307584       4.611138
min    -80995.000000            2010-12-01 08:26:00  -11062.060000
25%         1.000000            2011-03-28 11:34:00       1.250000
50%         3.000000            2011-07-19 17:17:00       2.080000
75%        10.000000            2011-10-19 11:27:00       4.130000
max     80995.000000            2011-12-09 12:50:00   38970.000000
std       218.080957                            NaN      96.759765

         Customer ID
count  406830.000000
mean    15287.684160
min     12346.000000
25%     13953.000000
50%     15152.000000
75%     16791.000000
```

```
max     18287.000000
std      1713.603074
```

```python
# Drop rows with missing CustomerID
data = data.dropna(subset=['Customer ID'])
# Fill missing Description with a placeholder
data['Description'] = data['Description'].fillna('Unknown')
# Verify missing values are handled
print(data.isnull().sum())
```

```
Invoice        0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
Price          0
Customer ID    0
Country        0
dtype: int64
```

```python
# Remove rows with negative Quantity or Price
#data = data[(data['Quantity'] > 0) & (data['Price'] > 0)]
# Verify the changes
#print(data[['Quantity', 'Price']].describe())

# Remove duplicate rows
data = data.drop_duplicates()
# Confirm there are no duplicates left
duplicates_count = data.duplicated().sum()
print(f"Number of duplicate rows remaining: {duplicates_count}")
# Display the dataset shape after removing duplicates
print(f"Dataset shape after removing duplicates: {data.shape}")
```

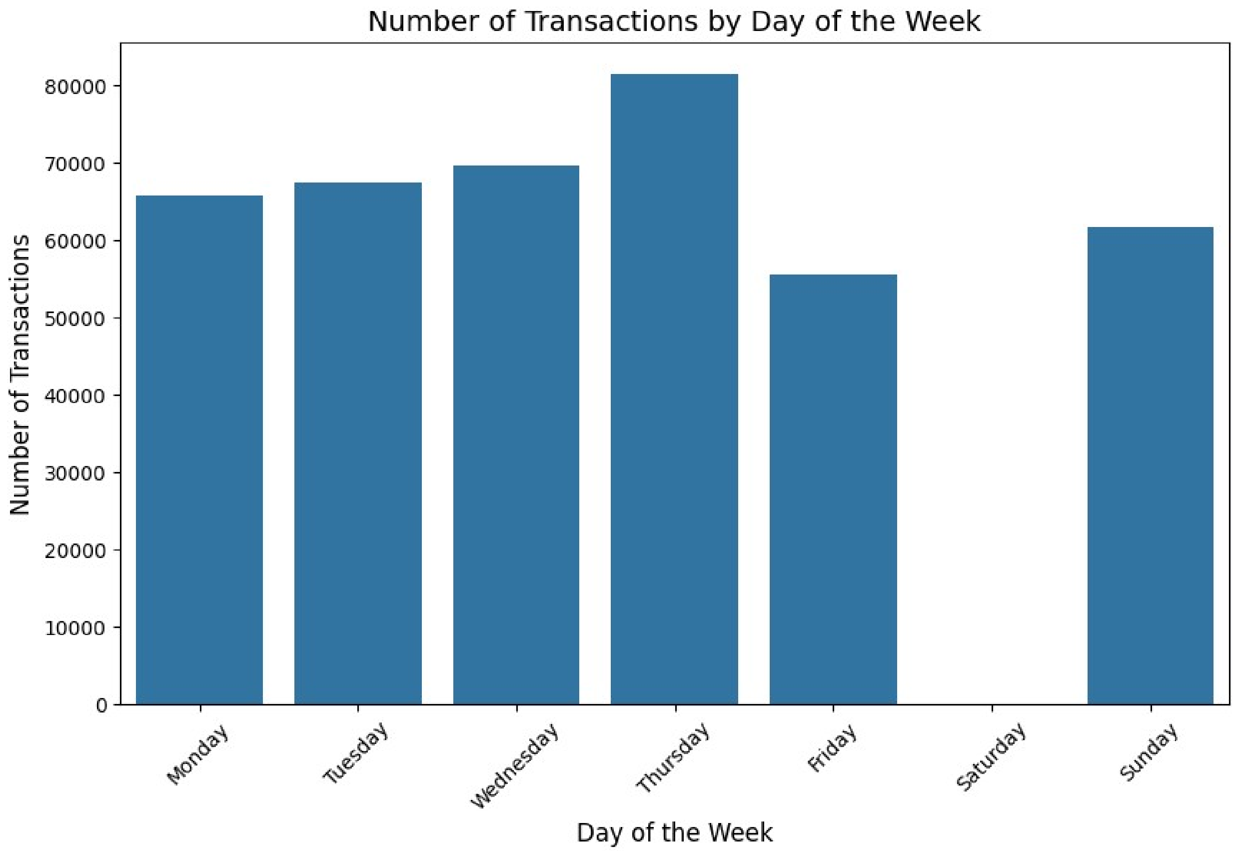
```
Number of duplicate rows remaining: 0
Dataset shape after removing duplicates: (401605, 8)
```

```python
# Ensure 'InvoiceDate' is in datetime format
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
# Add 'Day of Week' feature
data['DayOfWeek'] = data['InvoiceDate'].dt.day_name()
# Add 'Hour' feature to help categorize time of day
data['Hour'] = data['InvoiceDate'].dt.hour
# Categorize 'Hour' into 'Time of Day'
def categorize_time(hour):
    if 5 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 17:
        return 'Afternoon'
    elif 17 <= hour < 21:
        return 'Evening'
    else:
```

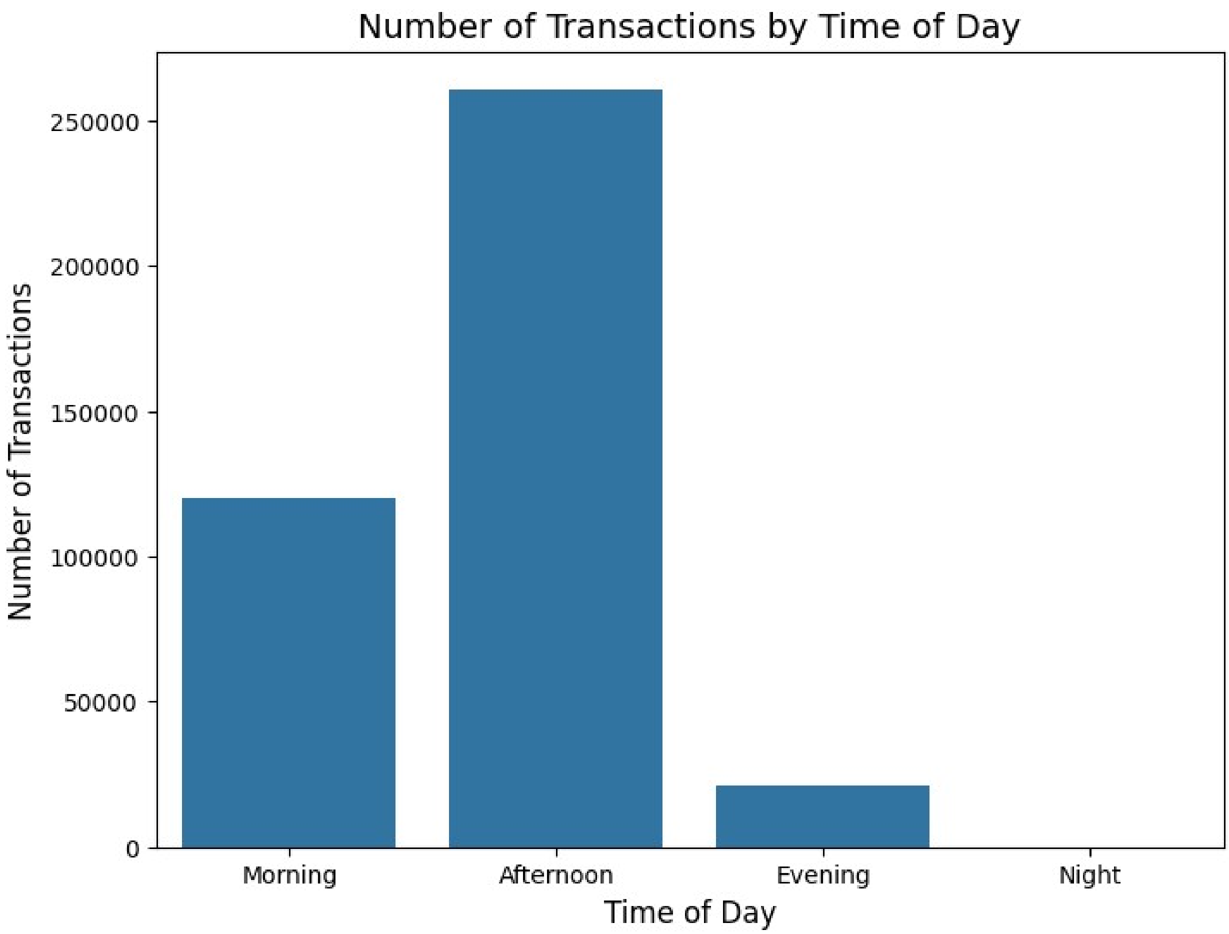
```python
        return 'Night'
data['TimeOfDay'] = data['Hour'].apply(categorize_time)
# Drop the temporary 'Hour' column
data.drop(columns=['Hour'], inplace=True)
# Verify the new features
print(data[['InvoiceDate', 'DayOfWeek', 'TimeOfDay']].head())
```

```
          InvoiceDate  DayOfWeek TimeOfDay
0 2010-12-01 08:26:00  Wednesday   Morning
1 2010-12-01 08:26:00  Wednesday   Morning
2 2010-12-01 08:26:00  Wednesday   Morning
3 2010-12-01 08:26:00  Wednesday   Morning
4 2010-12-01 08:26:00  Wednesday   Morning
```

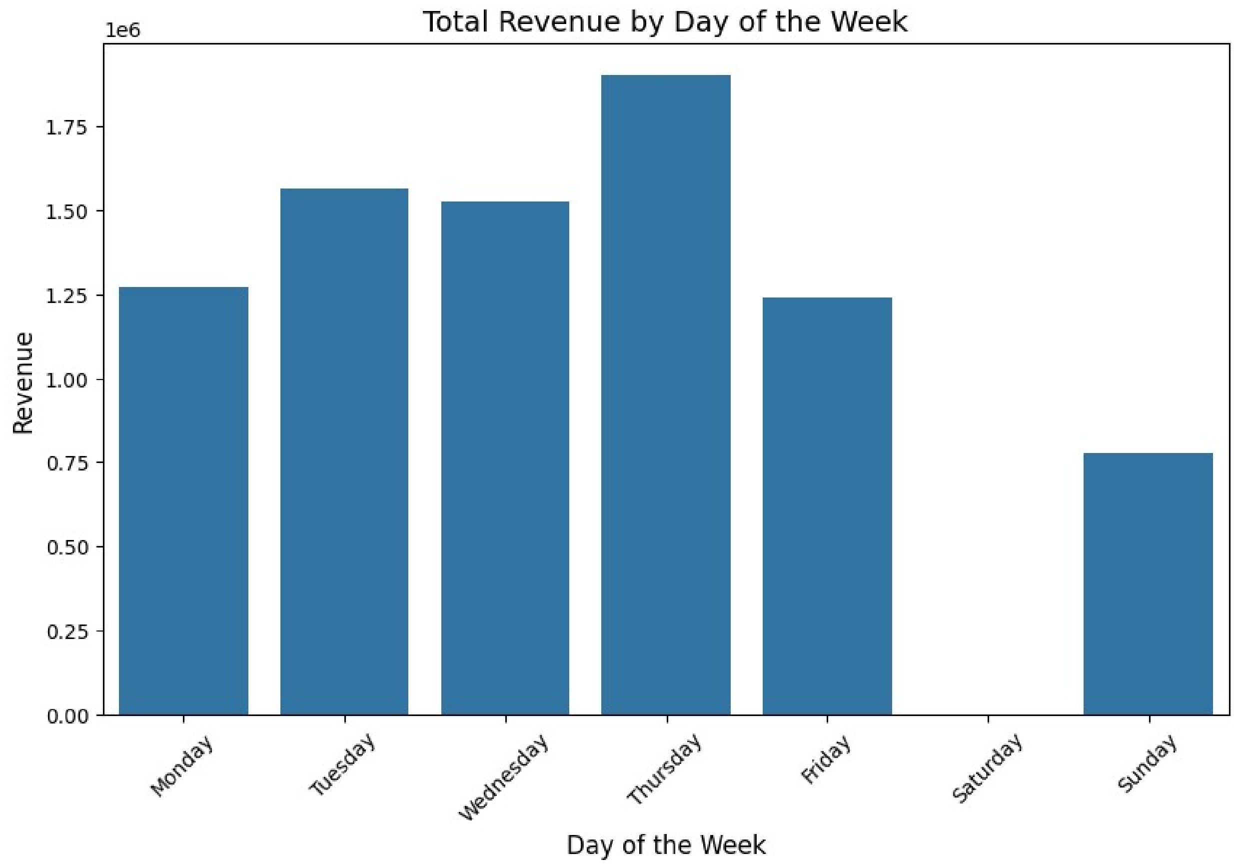
```python
import seaborn as sns
import matplotlib.pyplot as plt
# Transactions by Day of Week
plt.figure(figsize=(10, 6))
sns.countplot(x='DayOfWeek', data=data, order=['Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Number of Transactions by Day of the Week', fontsize=14)
plt.xlabel('Day of the Week', fontsize=12)
plt.ylabel('Number of Transactions', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```



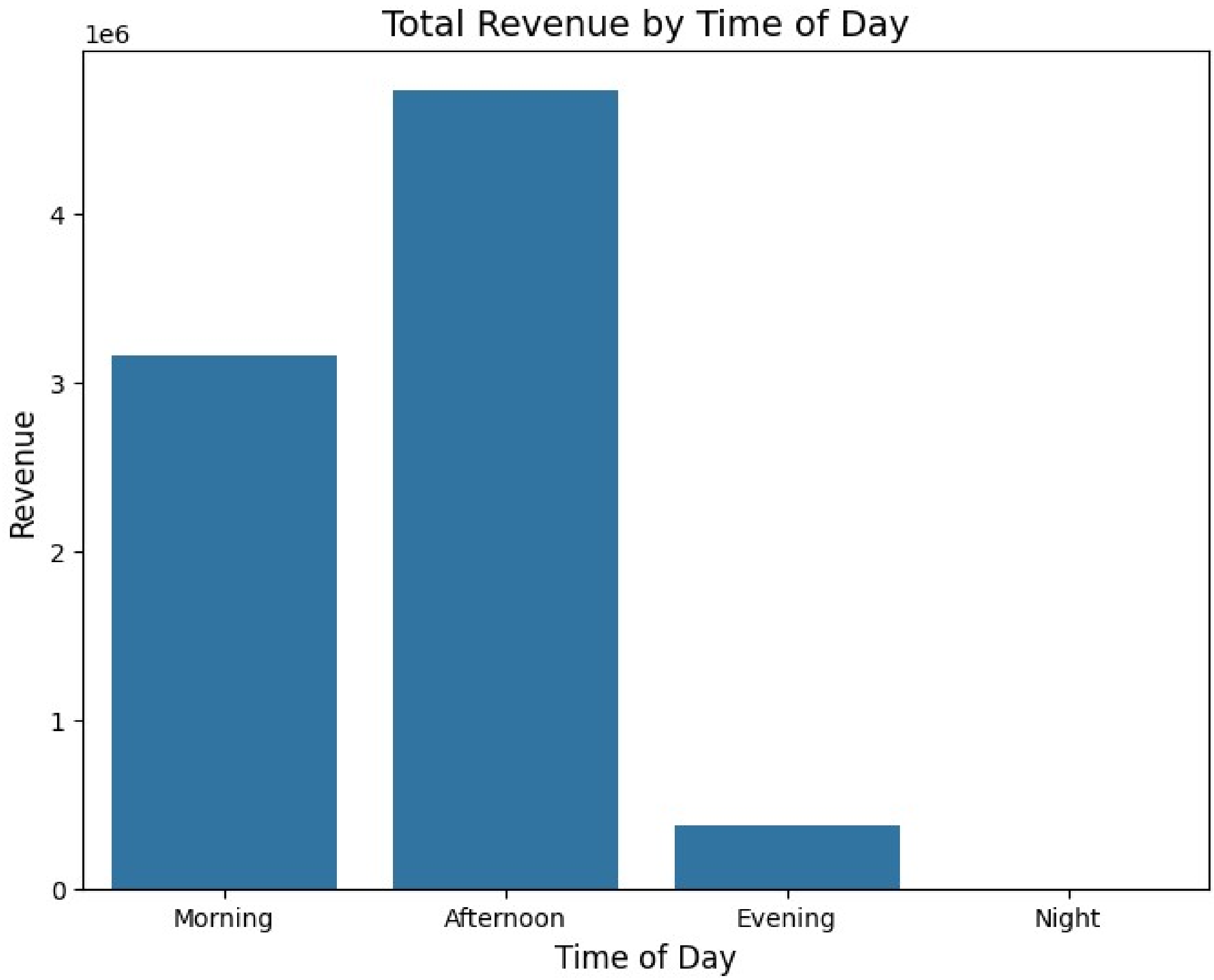

```python
# Transactions by Time of Day
plt.figure(figsize=(8, 6))
sns.countplot(x='TimeOfDay', data=data, order=['Morning', 'Afternoon', 'Evening', 'Night'])
plt.title('Number of Transactions by Time of Day', fontsize=14)
plt.xlabel('Time of Day', fontsize=12)
plt.ylabel('Number of Transactions', fontsize=12)
plt.show()
```



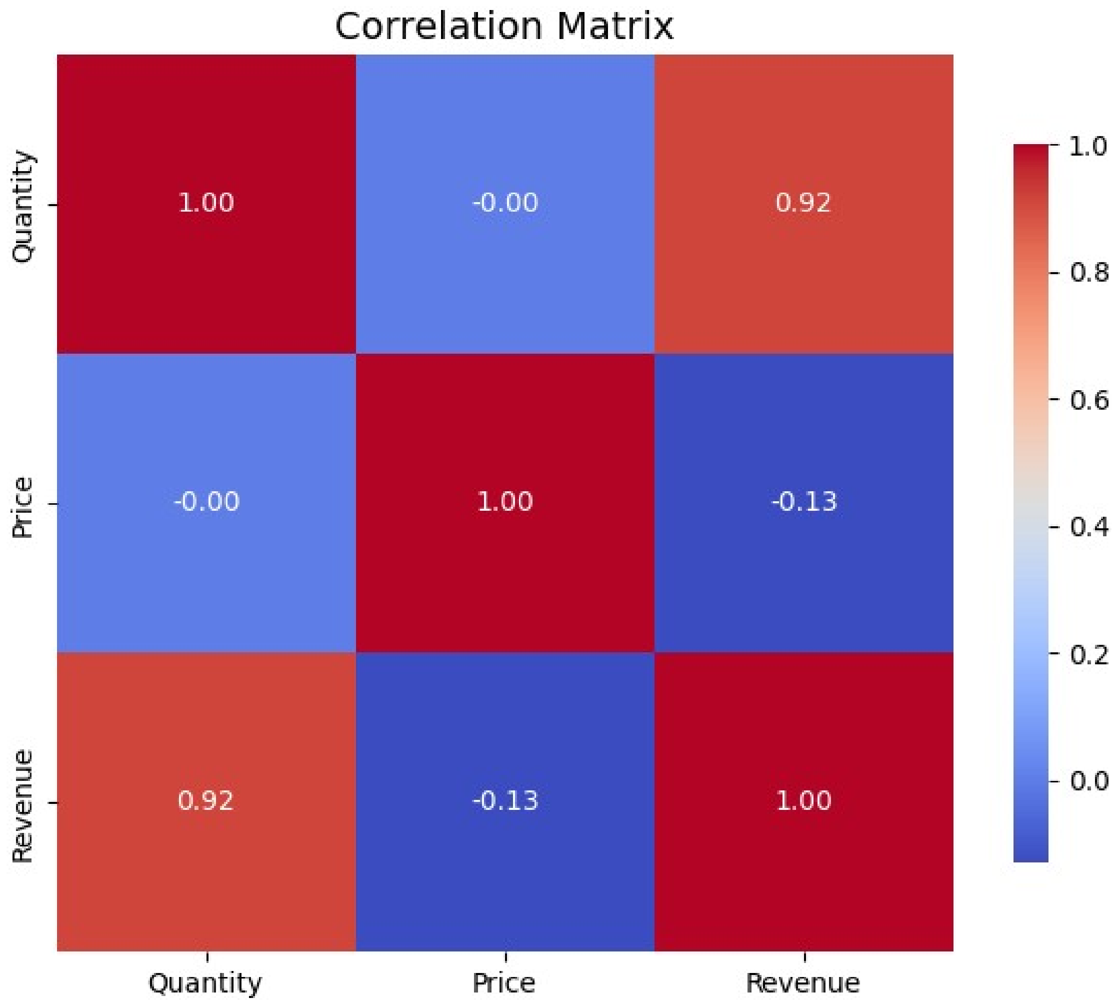

```
# Create a Revenue column
data['Revenue'] = data['Quantity'] * data['Price']
# Revenue by Day of Week
revenue_by_day = data.groupby('DayOfWeek')
['Revenue'].sum().reindex(['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.figure(figsize=(10, 6))
sns.barplot(x=revenue_by_day.index, y=revenue_by_day.values)
plt.title('Total Revenue by Day of the Week', fontsize=14)
plt.xlabel('Day of the Week', fontsize=12)
plt.ylabel('Revenue', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```




```
# Revenue by Time of Day
revenue_by_time = data.groupby('TimeOfDay')
['Revenue'].sum().reindex(['Morning', 'Afternoon', 'Evening',
'Night'])
plt.figure(figsize=(8, 6))
sns.barplot(x=revenue_by_time.index, y=revenue_by_time.values)
plt.title('Total Revenue by Time of Day', fontsize=14)
plt.xlabel('Time of Day', fontsize=12)
plt.ylabel('Revenue', fontsize=12)
plt.show()
```




```
# Calculate the correlation matrix
correlation_matrix = data[['Quantity', 'Price', 'Revenue']].corr()

# Display the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt='.2f', square=True, cbar_kws={"shrink": .8})
plt.title('Correlation Matrix', fontsize=14)
plt.show()
```




```
pip install scikit-learn

/opt/anaconda3/lib/python3.12/pty.py:95: DeprecationWarning: This
process (pid=16869) is multi-threaded, use of forkpty() may lead to
deadlocks in the child.
  pid, fd = os.forkpty()

Requirement already satisfied: scikit-learn in
/opt/anaconda3/lib/python3.12/site-packages (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn)
(1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn)
(1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn)
(1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
```

```
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn)
(2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import numpy as np

# Aggregate data at the customer level
customer_data = data.groupby('Customer ID').agg({
    'Revenue': 'sum',
    'InvoiceDate': 'max',
    'Invoice': 'count',
    'Quantity': 'sum'
}).rename(columns={
    'Revenue': 'TotalRevenue',
    'InvoiceDate': 'LastPurchase',
    'Invoice': 'TransactionCount',
    'Quantity': 'TotalQuantity'
})

# Calculate recency (days since last purchase)
latest_date = data['InvoiceDate'].max()  # Define the latest date
customer_data['Recency'] = (latest_date -
customer_data['LastPurchase']).dt.days

# Drop LastPurchase column after calculating recency
customer_data.drop(columns=['LastPurchase'], inplace=True)

# Scale the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(customer_data)

# Display the first few rows
print(customer_data.head())
```

```
             TotalRevenue  TransactionCount  TotalQuantity  Recency
Customer ID
12346.0              0.00                 2              0      325
12347.0           4310.00               182           2458        1
12348.0           1797.24                31           2341       74
12349.0           1757.55                73            631       18
12350.0            334.40                17            197      309
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd

# Apply K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=42)
customer_data['Cluster'] = kmeans.fit_predict(scaled_features)
```

```
# Display cluster centroids
centroids = pd.DataFrame(kmeans.cluster_centers_,
columns=customer_data.columns[:-1])
print("Cluster Centroids:\n", centroids)

# Display the first few rows of the data with cluster labels
print(customer_data.head())

Cluster Centroids:
    TotalRevenue  TransactionCount  TotalQuantity   Recency
0     -0.079562         -0.075751      -0.079459 -0.484738
1     -0.174407         -0.282366      -0.184532  1.563451
2     13.613960         11.055923      13.333843 -0.864721
3      1.024108          1.582632       1.081760 -0.775156
             TotalRevenue  TransactionCount  TotalQuantity  Recency
Cluster
Customer ID

12346.0              0.00                 2              0      325
1
12347.0           4310.00               182           2458        1
0
12348.0           1797.24                31           2341       74
0
12349.0           1757.55                73            631       18
0
12350.0            334.40                17            197      309
1

import matplotlib.pyplot as plt
import seaborn as sns

# Visualize clusters based on TotalRevenue and Recency
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=customer_data['Recency'],
    y=customer_data['TotalRevenue'],
    hue=customer_data['Cluster'],
    palette='viridis',
    s=100,
    alpha=0.8
)
plt.title('Customer Clusters Based on Recency and TotalRevenue',
fontsize=14)
plt.xlabel('Recency (days since last purchase)', fontsize=12)
plt.ylabel('Total Revenue', fontsize=12)
plt.legend(title='Cluster', fontsize=10)
plt.show()
```


```
print(customer_data.columns)

Index(['TotalRevenue', 'TransactionCount', 'TotalQuantity', 'Recency',
       'Cluster'],
      dtype='object')


# Assuming 'customer_data' already contains the relevant aggregated
data
# Create a 'Purchased' column based on Total Quantity
customer_data['Purchased'] = (customer_data['TotalQuantity'] >
0).astype(int)  # 1 if purchased, 0 if not

# Merge customer_data with session-level data
merged_data = data.merge(customer_data, how='left', on='Customer ID')
# Features (X) and Target (y)
X = merged_data[['TotalRevenue', 'Recency', 'TotalQuantity',
'TransactionCount']]
# Use Purchased_y instead of Purchased
y = merged_data['Purchased']  # Using the 'Purchased' column from
customer_data

# Train-test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42)

# Display train-test split shapes
print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (321284, 4)
Testing set shape: (80321, 4)
```

```
purchase_likelihood = customer_data.groupby('Cluster')
['Purchased'].mean()

# Print purchase likelihood for each cluster
print("Purchase Likelihood by Cluster:")
print(purchase_likelihood)
```

```
Purchase Likelihood by Cluster:
Cluster
0    0.994740
1    0.965485
2    1.000000
3    1.000000
Name: Purchased, dtype: float64
```

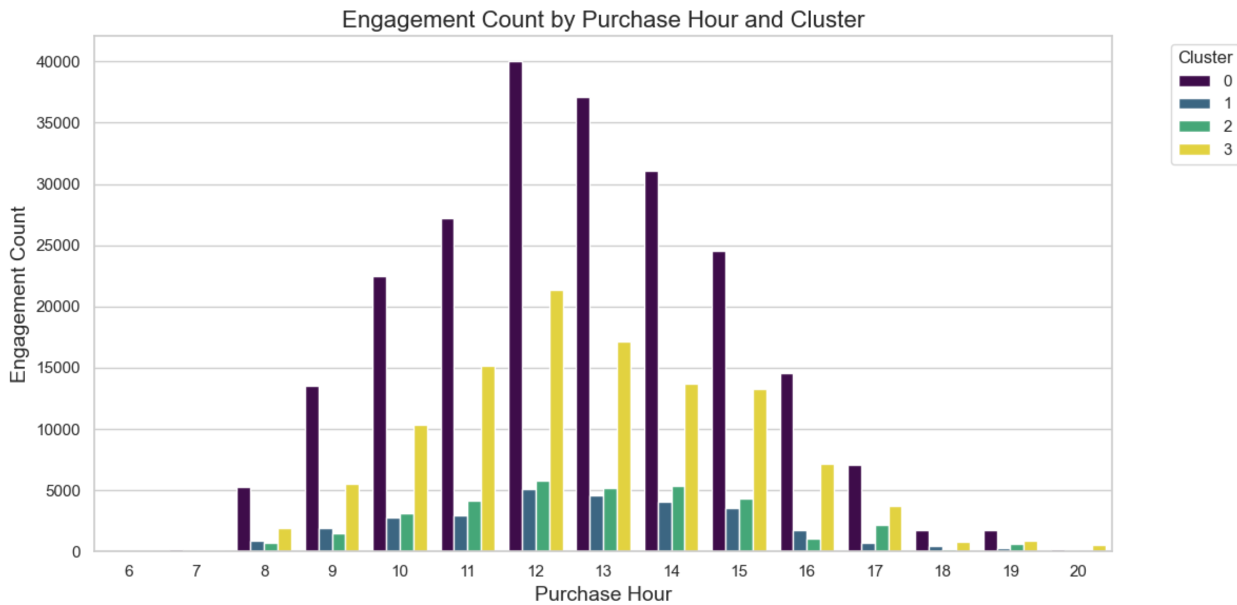
```
# Define marketing strategies for each cluster
marketing_strategies = {
    0: {
        "message": "We miss you! Enjoy 30% off your next purchase with
code WELCOME30.",
        "email_subject": "Come back for an exclusive discount!"
    },
    1: {
        "message": "Refer a friend and get 20% off your next order!",
        "email_subject": "Share the love with friends!"
    },
    2: {
        "message": "Thank you for being a loyal customer! Enjoy 10%
off your next order.",
        "email_subject": "Exclusive offers for our top customers!"
    },
    3: {
        "message": "We value your feedback! Help us improve and
receive a special offer.",
        "email_subject": "Your opinion matters to us!"
    }
}

# Function to assign marketing strategy based on cluster
def assign_marketing_strategy(row):
    cluster = row['Cluster']
```

```
    return marketing_strategies.get(cluster)

# Apply the function to the customer data
customer_data['MarketingMessage'] =
customer_data.apply(assign_marketing_strategy, axis=1)

# Display the first few rows with assigned marketing messages
print(customer_data[['Cluster', 'MarketingMessage']].head())


             Cluster
MarketingMessage
Customer ID

12346.0            1  {'message': 'Refer a friend and get 20% off
yo...
12347.0            0  {'message': 'We miss you! Enjoy 30% off your
n...
12348.0            0  {'message': 'We miss you! Enjoy 30% off your
n...
12349.0            0  {'message': 'We miss you! Enjoy 30% off your
n...
12350.0            1  {'message': 'Refer a friend and get 20% off
yo...

print(merged_data.columns)

Index(['Invoice', 'StockCode', 'Description', 'Quantity',
'InvoiceDate',
       'Price', 'Customer ID', 'Country', 'DayOfWeek', 'TimeOfDay',
'Revenue',
       'TotalRevenue', 'TransactionCount', 'TotalQuantity', 'Recency',
       'Cluster', 'Purchased'],
      dtype='object')

import pandas as pd
# Use the 'InvoiceDate' column to create the 'PurchaseHour' column
if 'InvoiceDate' in merged_data.columns:
    merged_data['PurchaseHour'] =
pd.to_datetime(merged_data['InvoiceDate']).dt.hour
else:
    raise KeyError("The 'InvoiceDate' column does not exist in the
merged data.")

# Continue with the analysis
engagement = merged_data.groupby(['Cluster',
'PurchaseHour']).size().reset_index(name='EngagementCount')

# Identify peak engagement hours for each cluster
peak_engagement = engagement.loc[engagement.groupby('Cluster')
['EngagementCount'].idxmax()]
```

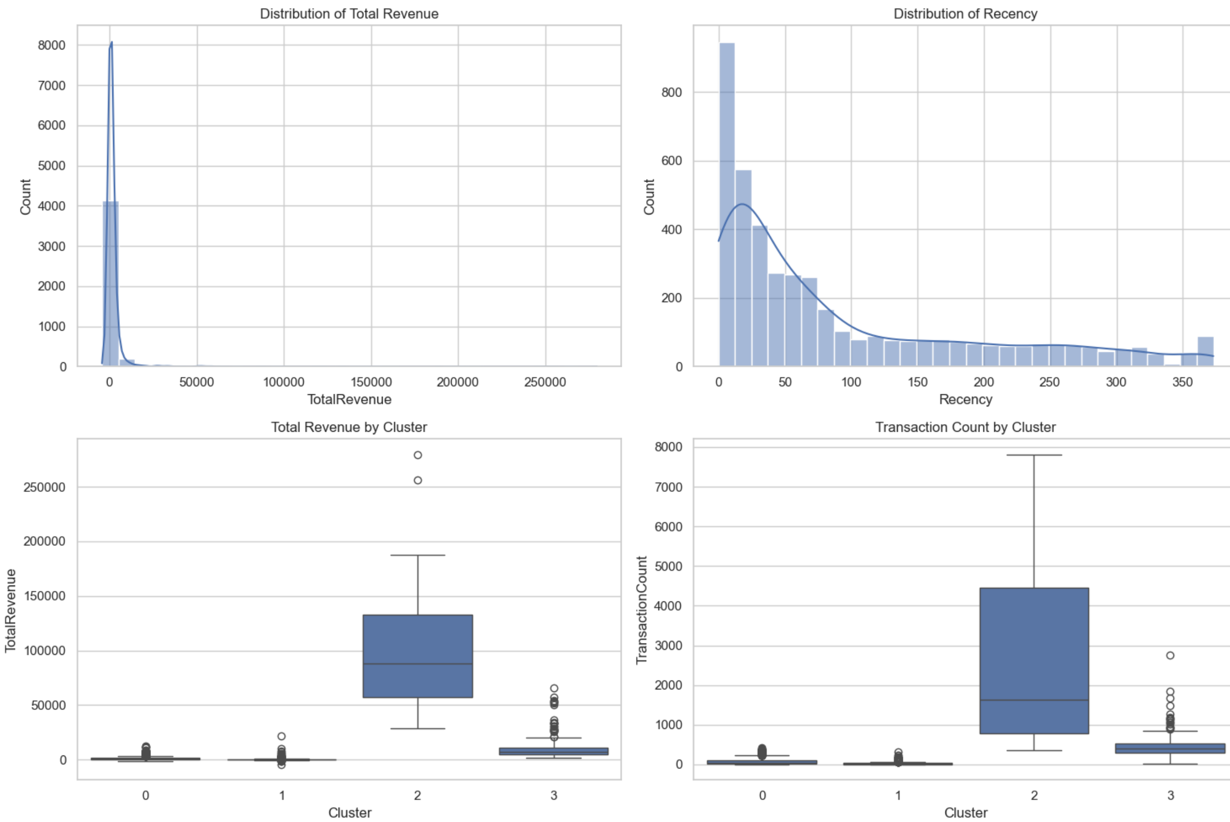
```python
# Display peak engagement hours for each cluster
print("Peak Engagement Hours by Cluster:")
print(peak_engagement)
```

```
Peak Engagement Hours by Cluster:
    Cluster  PurchaseHour  EngagementCount
6         0            12            40054
20        1            12             5064
35        2            12             5755
50        3            12            21341
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set the aesthetic style of the plots
sns.set(style='whitegrid')

# Create a bar plot for engagement counts by Cluster and PurchaseHour
plt.figure(figsize=(12, 6))
sns.barplot(data=engagement, x='PurchaseHour', y='EngagementCount',
hue='Cluster', palette='viridis')

# Customize the plot
plt.title('Engagement Count by Purchase Hour and Cluster',
fontsize=16)
plt.xlabel('Purchase Hour', fontsize=14)
plt.ylabel('Engagement Count', fontsize=14)
plt.xticks(rotation=0)  # Rotate x-axis labels if needed
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.tight_layout()

# Show the plot
plt.show()
```




```
# Histograms for feature distributions
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
sns.histplot(customer_data['TotalRevenue'], bins=30, kde=True)
plt.title('Distribution of Total Revenue')

plt.subplot(2, 2, 2)
sns.histplot(customer_data['Recency'], bins=30, kde=True)
plt.title('Distribution of Recency')

plt.subplot(2, 2, 3)
sns.boxplot(x='Cluster', y='TotalRevenue', data=customer_data)
plt.title('Total Revenue by Cluster')

plt.subplot(2, 2, 4)
sns.boxplot(x='Cluster', y='TransactionCount', data=customer_data)
plt.title('Transaction Count by Cluster')

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




```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Initialize the model
model = RandomForestClassifier(random_state=42)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Print the classification report
print(classification_report(y_test, y_pred))

# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['No Purchase', 'Purchase'], yticklabels=['No Purchase',
'Purchase'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
```

```
plt.title('Confusion Matrix')
plt.show()

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       126
           1       1.00      1.00      1.00     80195

    accuracy                           1.00     80321
   macro avg       1.00      1.00      1.00     80321
weighted avg       1.00      1.00      1.00     80321
```


```
# Create a new column for total spend
merged_data['TotalSpend'] = merged_data['Quantity'] *
merged_data['Price']

# Count unique products viewed by each customer
product_views = merged_data.groupby('Customer ID')
['StockCode'].nunique().reset_index()
product_views.columns = ['Customer ID', 'Stockcode']

# Count previous purchases
```

```
purchase_counts = merged_data.groupby('Customer ID')
['Invoice'].count().reset_index()
purchase_counts.columns = ['Customer ID', 'PreviousPurchases'] #
Aggregate customer data
customer_data = merged_data.groupby('Customer ID').agg({
    'Invoice': 'count',  # Number of purchases
    'TotalSpend': 'sum',   # Total spending
    'Country': 'first'      # Assuming customers might be in one
country
}).reset_index()# Merge additional features into customer data
customer_data = customer_data.merge(product_views, on='Customer ID',
how='left')
customer_data = customer_data.merge(purchase_counts, on='Customer ID',
how='left')# Create a target variable: 1 if a purchase was made, 0
otherwise
customer_data['MadePurchase'] = customer_data['Invoice'].apply(lambda
x: 1 if x > 0 else 0)

# Preparing the data for modeling
X = customer_data.drop(columns=['Customer ID', 'MadePurchase',
'Country', 'Invoice'])
y = customer_data['MadePurchase']

# Split into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a classification model
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize and fit the classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.00
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       875
```

```
    accuracy                           1.00       875
   macro avg       1.00      1.00      1.00       875
weighted avg       1.00      1.00      1.00       875
```

```python
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model, X, y, cv=5)  # 5-fold cross-validation
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean():.2f}")
```

```
Cross-Validation Scores: [1. 1. 1. 1. 1.]
Mean CV Score: 1.00
```

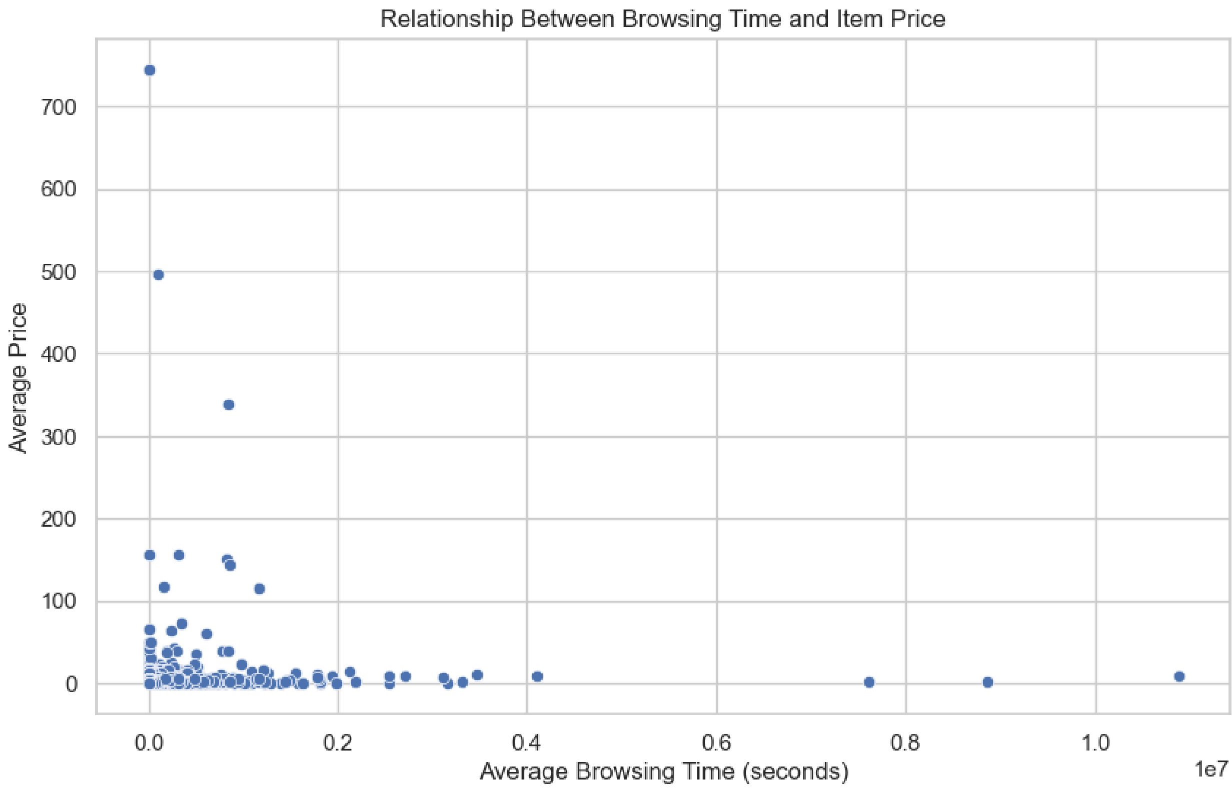
```python
print(customer_data['MadePurchase'].value_counts())
```

```
MadePurchase
1    4372
Name: count, dtype: int64
```

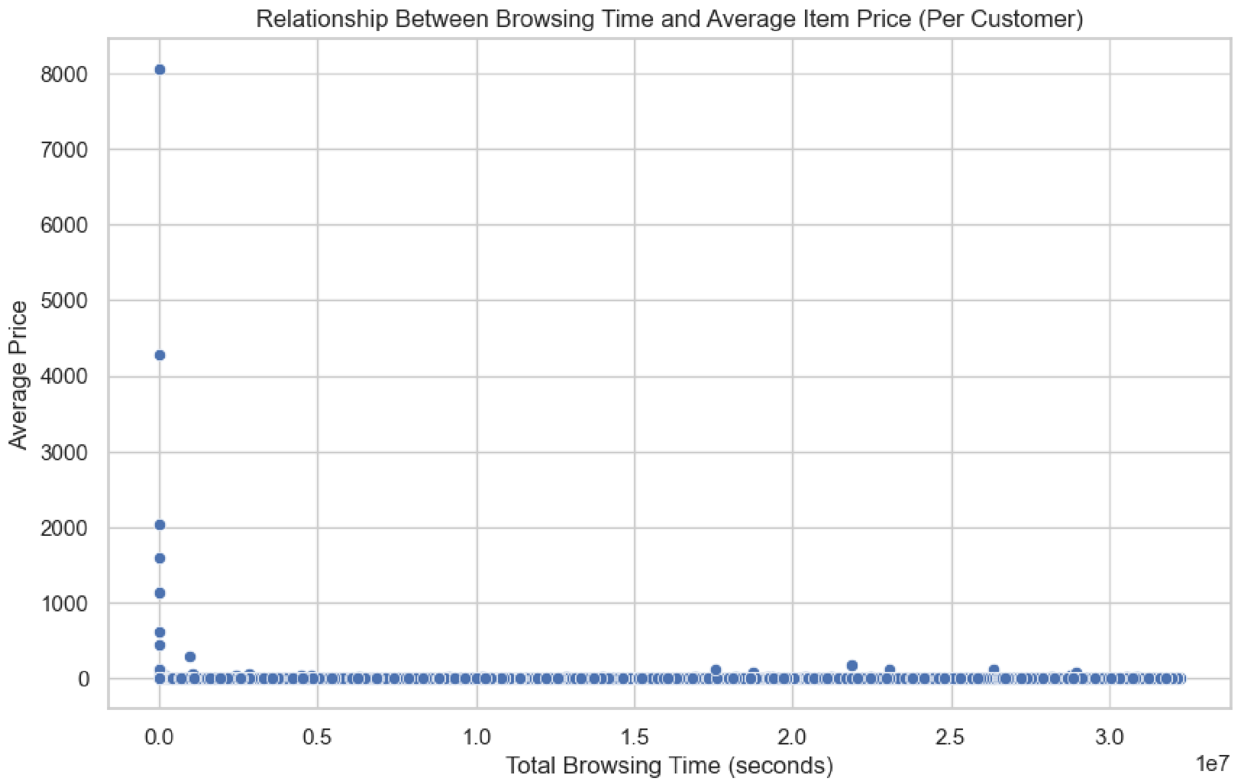
```python
import matplotlib.pyplot as plt

# Get the value counts
class_counts = customer_data['MadePurchase'].value_counts()

# Plot the class distribution
plt.figure(figsize=(8, 5))
class_counts.plot(kind='bar', color=['lightblue', 'salmon'])
plt.title('Class Distribution of MadePurchase')
plt.xlabel('Made Purchase (1 = Yes, 0 = No)')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```


```
# Sort the data by Customer ID and InvoiceDate
merged_data = merged_data.sort_values(by=['Customer ID',
'InvoiceDate'])

# Calculate browsing duration between transactions
merged_data['BrowsingTime'] = merged_data.groupby('Customer ID')
['InvoiceDate'].diff().dt.total_seconds()

# Replace NaN browsing time (first purchase) with a default value
(e.g., 0)
merged_data['BrowsingTime'] = merged_data['BrowsingTime'].fillna(0)

# Average browsing time per item (StockCode)
item_browsing = merged_data.groupby('StockCode').agg({
    'BrowsingTime': 'mean',
    'Price': 'mean'
}).reset_index()

# Visualize the relationship
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(data=item_browsing, x='BrowsingTime', y='Price')
plt.title('Relationship Between Browsing Time and Item Price')
```

```
plt.xlabel('Average Browsing Time (seconds)')
plt.ylabel('Average Price')
plt.grid(True)
plt.show()
```




```
# Total browsing time and average price per customer
customer_browsing = merged_data.groupby('Customer ID').agg({
    'BrowsingTime': 'sum',
    'Price': 'mean'
}).reset_index()

# Visualize the relationship
plt.figure(figsize=(10, 6))
sns.scatterplot(data=customer_browsing, x='BrowsingTime', y='Price')
plt.title('Relationship Between Browsing Time and Average Item Price
(Per Customer)')
plt.xlabel('Total Browsing Time (seconds)')
plt.ylabel('Average Price')
plt.grid(True)
plt.show()
```




```python
# Aggregate browsing time by customer
customer_browsing_time = merged_data.groupby('Customer ID')
['BrowsingTime'].sum().reset_index()
customer_browsing_time.columns = ['Customer ID', 'TotalBrowsingTime']

# Merge with customer data
customer_data = customer_data.merge(customer_browsing_time,
on='Customer ID', how='left')

# Verify the updated customer_data
print(customer_data.head())
```

```
   Customer ID  Invoice  TotalSpend         Country  Stockcode  \
0      12346.0        2        0.00  United Kingdom          1
1      12347.0      182     4310.00         Iceland        103
2      12348.0       31     1797.24         Finland         22
3      12349.0       73     1757.55           Italy         73
4      12350.0       17      334.40          Norway         17

   PreviousPurchases  MadePurchase  TotalBrowsingTime
0                  2             1              960.0
1                182             1         31539300.0
2                 31             1         24429840.0
3                 73             1                0.0
4                 17             1                0.0
```

```python
# Add a feature for average price per customer
average_price = merged_data.groupby('Customer ID')
['Price'].mean().reset_index()
average_price.columns = ['Customer ID', 'AveragePrice']
customer_data = customer_data.merge(average_price, on='Customer ID',
how='left')

# Prepare data for modeling
X = customer_data.drop(columns=['Customer ID', 'MadePurchase',
'Country', 'Invoice'])
y = customer_data['MadePurchase']

# Train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train a classification model
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
from sklearn.metrics import classification_report
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      1312

    accuracy                           1.00      1312
   macro avg       1.00      1.00      1.00      1312
weighted avg       1.00      1.00      1.00      1312
```

```python
def categorize_browsing_time(time):
    if time < 300:  # Short browsing (< 5 minutes)
        return 'Short'
    elif time <= 900:  # Medium browsing (5–15 minutes)
        return 'Medium'
    else:  # Long browsing (> 15 minutes)
        return 'Long'

merged_data['BrowsingCategory'] =
merged_data['BrowsingTime'].apply(categorize_browsing_time)

# Analyze the number of transactions per browsing category
browsing_counts = merged_data['BrowsingCategory'].value_counts()
print(browsing_counts)
```

```
BrowsingCategory
Short     385345
Long       15789
Medium       471
Name: count, dtype: int64
```

```
avg_price_by_category = merged_data.groupby('BrowsingCategory')
['Price'].mean().reset_index()
print(avg_price_by_category)
```

```
  BrowsingCategory      Price
0             Long   8.725475
1           Medium  25.598301
2            Short   3.231890
```

```
merged_data['TotalSpend'] = merged_data['Quantity'] *
merged_data['Price']
revenue_by_category = merged_data.groupby('BrowsingCategory')
['TotalSpend'].sum().reset_index()
print(revenue_by_category)
```

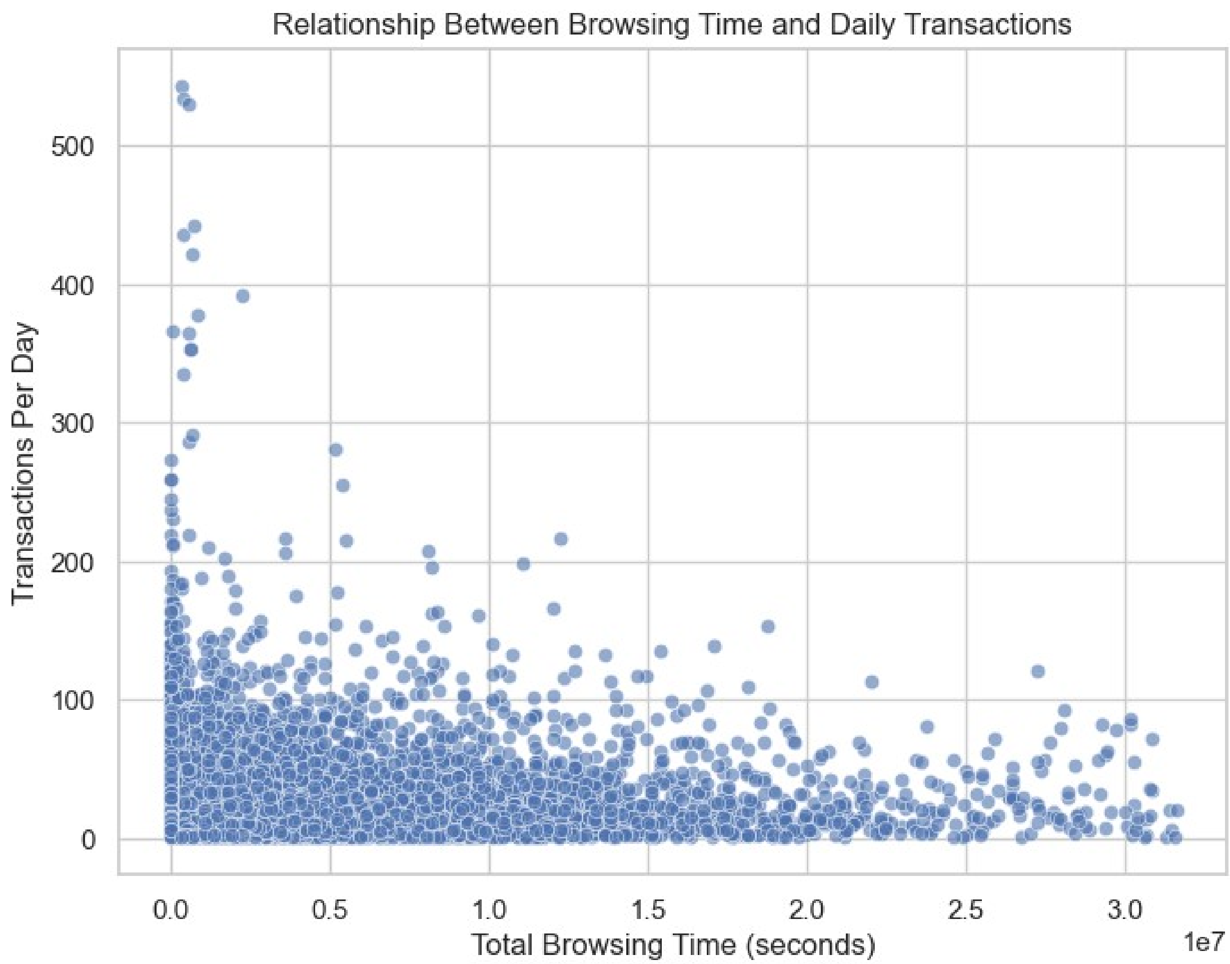
```
  BrowsingCategory   TotalSpend
0             Long   651520.890
1           Medium  -102223.000
2            Short  7729239.534
```

```
def generate_offer(category):
    if category == 'Short':
        return 'Free Shipping'
    elif category == 'Medium':
        return '10% Discount'
    else:  # Long
        return 'Personalized Recommendations'

merged_data['DynamicOffer'] =
merged_data['BrowsingCategory'].apply(generate_offer)

# Validate the offer distribution
offer_distribution = merged_data['DynamicOffer'].value_counts()
print(offer_distribution)
```

```
DynamicOffer
Free Shipping                   385345
Personalized Recommendations     15789
10% Discount                       471
Name: count, dtype: int64
```

```
# Extract the date (day only) from InvoiceDate
merged_data['Day'] = merged_data['InvoiceDate'].dt.date# Group by
Customer ID and Day
customer_daily_data = merged_data.groupby(['Customer ID',
```

```
'Day']).agg({
    'Invoice': 'count',       # Number of transactions in a day
    'BrowsingTime': 'sum'       # Total browsing time in a day
(seconds)
}).reset_index()

# Rename columns for clarity
customer_daily_data.rename(columns={
    'Invoice': 'TransactionsPerDay',
    'BrowsingTime': 'TotalBrowsingTime'
}, inplace=True)


# Merge daily customer data back into the original dataset
merged_data = merged_data.merge(customer_daily_data, on=['Customer
ID', 'Day'], how='left')

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.scatterplot(
    x='TotalBrowsingTime',
    y='TransactionsPerDay',
    data=customer_daily_data,
    alpha=0.6
)
plt.title('Relationship Between Browsing Time and Daily Transactions')
plt.xlabel('Total Browsing Time (seconds)')
plt.ylabel('Transactions Per Day')
plt.show()
```




```
correlation =
customer_daily_data['TotalBrowsingTime'].corr(customer_daily_data['Tra
nsactionsPerDay'])
print(f"Correlation: {correlation}")

Correlation: 0.07562107407483008

# Create a binary target: 1 if more than 3 transactions, else 0
customer_daily_data['HighTransaction'] =
customer_daily_data['TransactionsPerDay'].apply(lambda x: 1 if x > 3
else 0)

X = customer_daily_data[['TotalBrowsingTime']]  # Feature
y = customer_daily_data['HighTransaction']      # Target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

from sklearn.metrics import classification_report, accuracy_score

print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7349740932642487
              precision    recall  f1-score   support

           0       0.30      0.25      0.27       770
           1       0.82      0.86      0.84      3090

    accuracy                           0.73      3860
   macro avg       0.56      0.55      0.55      3860
weighted avg       0.72      0.73      0.73      3860
```

```python
def determine_action(browsing_time):
    if browsing_time <= 300:  # 5 minutes in seconds
        return "No action"
    elif 301 <= browsing_time <= 600:  # 5-10 minutes
        return "Offer personalized recommendations"
    elif 601 <= browsing_time <= 900:  # 10-15 minutes
        return "Offer 10% discount"
    else:  # 15 minutes and above
        return "Offer free shipping"

import pandas as pd

# Sample merged_data DataFrame


# Create a new column for Total Sales per transaction
merged_data['TotalSales'] = merged_data['Quantity'] *
merged_data['Price']

# Ensure BrowsingTime is in datetime format if not already
merged_data['BrowsingTime'] =
pd.to_datetime(merged_data['BrowsingTime'])


merged_data['BrowsingDate'] = merged_data['BrowsingTime'].dt.date
```

```
# Aggregate data to get total sales per customer per day
sales_data = merged_data.groupby(['Customer ID',
'BrowsingDate']).agg({
    'TotalSales': 'sum',          # Total sales per customer per day
    'TotalQuantity': 'sum',            # Total quantity per customer
per day
    'TotalSales': 'sum',          # Total sales per transaction
    'BrowsingTime': 'count'        # Count of transactions
}).reset_index()

# Rename columns for clarity
sales_data.columns = ['Customer ID', 'BrowsingDate', 'TotalSales',
'TotalQuantity', 'TransactionCount']

# Display the first few rows of the aggregated data
print(sales_data.head())
```

```
   Customer ID BrowsingDate  TotalSales  TotalQuantity
TransactionCount
0      12346.0   1970-01-01        0.00              0
2
1      12347.0   1970-01-01     4310.00         447356
182
2      12348.0   1970-01-01     1797.24          72571
31
3      12349.0   1970-01-01     1757.55          46063
73
4      12350.0   1970-01-01      334.40           3349
17
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
X = sales_data[['TotalQuantity', 'Customer ID']]  # Add other features
as necessary
y = sales_data['TotalSales']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')


predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(predictions_df.head())
```

```
Mean Squared Error: 53130896.37934003
      Actual    Predicted
2014  893.66  1572.508790
457   409.90  1602.404006
478   108.07  1598.747793
438   496.06  1610.289333
3728  748.94  1530.209482
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R² Score: {r2}')
```

```
Mean Squared Error: 53130896.37934003
R² Score: 0.24238147425682788
```

```
merged_data = pd.DataFrame(data)

# Convert 'InvoiceDate' to datetime format
merged_data['InvoiceDate'] =
pd.to_datetime(merged_data['InvoiceDate'])

# Extract month, day of the week, and date
merged_data['Month'] = merged_data['InvoiceDate'].dt.to_period('M')
merged_data['DayOfWeek'] = merged_data['InvoiceDate'].dt.day_name()  #
Gets the day of the week
merged_data['Date'] = merged_data['InvoiceDate'].dt.date
```

```python
# Aggregate quantity data by month
monthly_quantity = merged_data.groupby('Month').agg({'Quantity':
'sum'}).reset_index()
print("Monthly Quantity Sold:")
print(monthly_quantity)

# Identify months with zero or low quantities sold
low_quantity_months = monthly_quantity[monthly_quantity['Quantity'] ==
0]
print("\nMonths with Zero Quantity Sold:")
print(low_quantity_months)

# Aggregate quantity data by weekday
weekly_quantity = merged_data.groupby('DayOfWeek').agg({'Quantity':
'sum'}).reindex([
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'
]).reset_index()
print("\nWeekly Quantity Sold:")
print(weekly_quantity)

# Identify weekdays with zero or low quantities sold
low_quantity_weekdays = weekly_quantity[weekly_quantity['Quantity'] ==
0]
print("\nWeekdays with Zero Quantity Sold:")
print(low_quantity_weekdays)

# Aggregate quantity data by date
daily_quantity = merged_data.groupby('Date').agg({'Quantity':
'sum'}).reset_index()
print("\nDaily Quantity Sold:")
print(daily_quantity)

# Identify days with zero or low quantities sold
low_quantity_days = daily_quantity[daily_quantity['Quantity'] == 0]
print("\nDays with Zero Quantity Sold:")
print(low_quantity_days)
```

```
Monthly Quantity Sold:
      Month  Quantity
0   2010-12    295177
1   2011-01    268755
2   2011-02    262243
3   2011-03    343095
4   2011-04    277730
5   2011-05    367115
6   2011-06    356239
7   2011-07    361359
8   2011-08    385865
9   2011-09    536350
```

```
10  2011-10    568898
11  2011-11    666813
12  2011-12    203213

Months with Zero Quantity Sold:
Empty DataFrame
Columns: [Month, Quantity]
Index: []

Weekly Quantity Sold:
   DayOfWeek   Quantity
0     Monday   739603.0
1    Tuesday   912081.0
2  Wednesday   938243.0
3   Thursday  1115666.0
4     Friday   729509.0
5   Saturday        NaN
6     Sunday   457750.0

Weekdays with Zero Quantity Sold:
Empty DataFrame
Columns: [DayOfWeek, Quantity]
Index: []

Daily Quantity Sold:
           Date  Quantity
0    2010-12-01     23931
1    2010-12-02     20790
2    2010-12-03     11507
3    2010-12-05     16186
4    2010-12-06     15919
..          ...       ...
300  2011-12-05     38224
301  2011-12-06     26641
302  2011-12-07     40903
303  2011-12-08     26837
304  2011-12-09      9523

[305 rows x 2 columns]

Days with Zero Quantity Sold:
Empty DataFrame
Columns: [Date, Quantity]
Index: []

import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Create the basket DataFrame
basket = (merged_data
```

```
          .groupby(['Invoice', 'Description'])['StockCode']
          .count()
          .unstack(fill_value=0)
          .reset_index()
          .set_index('Invoice'))

# Convert counts to boolean values
basket = basket.apply(lambda x: (x > 0), axis=1)

# Generate frequent itemsets
frequent_itemsets = apriori(basket, min_support=0.01,
use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)

# Filter rules based on confidence and lift
filtered_rules = rules[(rules['confidence'] > 0.7) & (rules['lift'] >
1.2)]

# Display filtered rules
print(filtered_rules[['antecedents', 'consequents', 'support',
'confidence', 'lift']])
```

```
                                           antecedents  \
29                         (PAINTED METAL PEARS ASSORTED)
34                           (BAKING SET SPACEBOY DESIGN)
38                                    (TOILET METAL SIGN)
40                          (PINK HAPPY BIRTHDAY BUNTING)
41                          (BLUE HAPPY BIRTHDAY BUNTING)
44                      (CANDLEHOLDER PINK HANGING HEART)
74                   (GARDENERS KNEELING PAD CUP OF TEA )
80                        (PINK REGENCY TEACUP AND SAUCER)
85                       (GREEN REGENCY TEACUP AND SAUCER)
329                       (PINK REGENCY TEACUP AND SAUCER)
336                            (POPPY'S PLAYHOUSE KITCHEN)
337                           (POPPY'S PLAYHOUSE BEDROOM )
356                              (REGENCY TEA PLATE GREEN )
368                         (SET/6 RED SPOTTY PAPER PLATES)
370                         (SET/6 RED SPOTTY PAPER PLATES)
371                           (SET/6 RED SPOTTY PAPER CUPS)
373                          (SMALL MARSHMALLOWS PINK BOWL)
383                    (WOODEN STAR CHRISTMAS SCANDINAVIAN)
384                    (WOODEN TREE CHRISTMAS SCANDINAVIAN)
386   (ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...
392   (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
393   (REGENCY CAKESTAND 3 TIER, GREEN REGENCY TEACU...
398   (ROSES REGENCY TEACUP AND SAUCER , PINK REGENC...
399   (ROSES REGENCY TEACUP AND SAUCER , GREEN REGEN...
```

```
400  (PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...
404  (ROSES REGENCY TEACUP AND SAUCER , REGENCY CAK...
406  (REGENCY CAKESTAND 3 TIER, GREEN REGENCY TEACU...
412    (JUMBO BAG STRAWBERRY, JUMBO BAG PINK POLKADOT)
417  (JUMBO STORAGE BAG SUKI, JUMBO BAG PINK POLKADOT)
519  (LUNCH BAG PINK POLKADOT, LUNCH BAG SUKI DESIGN )
525      (LUNCH BAG WOODLAND, LUNCH BAG PINK POLKADOT)
543       (LUNCH BAG WOODLAND, LUNCH BAG SUKI DESIGN )
549  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
554  (ROSES REGENCY TEACUP AND SAUCER , REGENCY CAK...
555  (ROSES REGENCY TEACUP AND SAUCER , REGENCY CAK...
557  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
561  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...

                                           consequents   support
confidence  \
29                 (ASSORTED COLOUR BIRD ORNAMENT)  0.011537
0.723164
34                 (BAKING SET 9 PIECE RETROSPOT )  0.014060
0.710706
38                           (BATHROOM METAL SIGN)  0.010230
0.739414
40                   (BLUE HAPPY BIRTHDAY BUNTING)  0.011537
0.705234
41                   (PINK HAPPY BIRTHDAY BUNTING)  0.011537
0.715084
44            (WHITE HANGING HEART T-LIGHT HOLDER)  0.011492
0.732759
74             (GARDENERS KNEELING PAD KEEP CALM )  0.021000
0.725857
80               (GREEN REGENCY TEACUP AND SAUCER)  0.021226
0.796954
85              (ROSES REGENCY TEACUP AND SAUCER )  0.025101
0.759891
329             (ROSES REGENCY TEACUP AND SAUCER )  0.020324
0.763113
336                   (POPPY'S PLAYHOUSE BEDROOM )  0.011492
0.730659
337                    (POPPY'S PLAYHOUSE KITCHEN)  0.011492
0.799373
356                      (REGENCY TEA PLATE ROSES )  0.010455
0.843636
368          (SET/20 RED RETROSPOT PAPER NAPKINS )  0.010320
0.704615
370                    (SET/6 RED SPOTTY PAPER CUPS)  0.010635
0.726154
371                  (SET/6 RED SPOTTY PAPER PLATES)  0.010635
0.828070
373            (SMALL DOLLY MIX DESIGN ORANGE BOWL)  0.010185
```

```
0.782007
383            (WOODEN HEART CHRISTMAS SCANDINAVIAN)  0.014421
0.733945
384             (WOODEN STAR CHRISTMAS SCANDINAVIAN)  0.010230
0.819495
386                        (ALARM CLOCK BAKELIKE RED )  0.012078
0.779070
392                (GREEN REGENCY TEACUP AND SAUCER)  0.012348
0.858934
393                 (PINK REGENCY TEACUP AND SAUCER)  0.012348
0.715405
398                (GREEN REGENCY TEACUP AND SAUCER)  0.017891
0.880266
399                 (PINK REGENCY TEACUP AND SAUCER)  0.017891
0.712747
400               (ROSES REGENCY TEACUP AND SAUCER )  0.017891
0.842887
404                (GREEN REGENCY TEACUP AND SAUCER)  0.014241
0.731481
406               (ROSES REGENCY TEACUP AND SAUCER )  0.014241
0.825065
412                          (JUMBO BAG RED RETROSPOT)  0.010500
0.792517
417                          (JUMBO BAG RED RETROSPOT)  0.010050
0.785211
519                          (LUNCH BAG RED RETROSPOT)  0.011492
0.704420
525                          (LUNCH BAG RED RETROSPOT)  0.010816
0.743034
543                          (LUNCH BAG RED RETROSPOT)  0.010140
0.725806
549               (ROSES REGENCY TEACUP AND SAUCER )  0.012213
0.849530
554                (GREEN REGENCY TEACUP AND SAUCER)  0.010861
0.889299
555                 (PINK REGENCY TEACUP AND SAUCER)  0.010861
0.762658
557               (ROSES REGENCY TEACUP AND SAUCER )  0.010861
0.879562
561  (ROSES REGENCY TEACUP AND SAUCER , GREEN REGEN...  0.010861
0.755486

          lift
29   11.586286
34   17.900760
38   42.287602
40   43.712698
41   43.712698
44    8.077453
```

```
74    20.999687
80    24.126079
85    20.169830
329   20.255366
336   50.825466
337   50.825466
356   55.059679
368   21.301656
370   56.538084
371   56.538084
373   47.935728
383   35.024169
384   41.707763
386   19.060152
392   26.002386
393   26.860965
398   26.648164
399   26.761172
400   22.372815
404   22.144030
406   21.899759
412   10.703562
417   10.604892
519   11.761533
525   12.406265
543   12.118619
549   22.549122
554   26.921613
555   28.635171
557   23.346270
561   30.097364
```

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Assuming 'basket' is already prepared
print(frequent_itemsets.head())

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)

# Filter rules based on confidence and lift
filtered_rules = rules[(rules['confidence'] > 0.7) & (rules['lift'] >
1.2)]

# Display filtered rules
print(filtered_rules[['antecedents', 'consequents', 'support',
'confidence', 'lift']])
```

```
    support                                      itemsets
0  0.011221                  (10 COLOUR SPACEBOY PEN)
1  0.012528           (12 PENCIL SMALL TUBE WOODLAND)
2  0.014015  (12 PENCILS SMALL TUBE RED RETROSPOT)
3  0.013249             (12 PENCILS SMALL TUBE SKULL)
4  0.010680   (12 PENCILS TALL TUBE RED RETROSPOT)
                                            antecedents  \
29                        (PAINTED METAL PEARS ASSORTED)
34                          (BAKING SET SPACEBOY DESIGN)
38                                   (TOILET METAL SIGN)
40                         (PINK HAPPY BIRTHDAY BUNTING)
41                         (BLUE HAPPY BIRTHDAY BUNTING)
44                    (CANDLEHOLDER PINK HANGING HEART)
74                 (GARDENERS KNEELING PAD CUP OF TEA )
80                      (PINK REGENCY TEACUP AND SAUCER)
85                     (GREEN REGENCY TEACUP AND SAUCER)
329                     (PINK REGENCY TEACUP AND SAUCER)
336                          (POPPY'S PLAYHOUSE KITCHEN)
337                         (POPPY'S PLAYHOUSE BEDROOM )
356                           (REGENCY TEA PLATE GREEN )
368                        (SET/6 RED SPOTTY PAPER PLATES)
370                        (SET/6 RED SPOTTY PAPER PLATES)
371                          (SET/6 RED SPOTTY PAPER CUPS)
373                       (SMALL MARSHMALLOWS PINK BOWL)
383                  (WOODEN STAR CHRISTMAS SCANDINAVIAN)
384                  (WOODEN TREE CHRISTMAS SCANDINAVIAN)
386  (ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...
392  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
393  (REGENCY CAKESTAND 3 TIER, GREEN REGENCY TEACU...
398  (ROSES REGENCY TEACUP AND SAUCER , PINK REGENC...
399  (ROSES REGENCY TEACUP AND SAUCER , GREEN REGEN...
400  (PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...
404  (ROSES REGENCY TEACUP AND SAUCER , REGENCY CAK...
406  (REGENCY CAKESTAND 3 TIER, GREEN REGENCY TEACU...
412    (JUMBO BAG STRAWBERRY, JUMBO BAG PINK POLKADOT)
417  (JUMBO STORAGE BAG SUKI, JUMBO BAG PINK POLKADOT)
519  (LUNCH BAG PINK POLKADOT, LUNCH BAG SUKI DESIGN )
525      (LUNCH BAG WOODLAND, LUNCH BAG PINK POLKADOT)
543        (LUNCH BAG WOODLAND, LUNCH BAG SUKI DESIGN )
549  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
554  (ROSES REGENCY TEACUP AND SAUCER , REGENCY CAK...
555  (ROSES REGENCY TEACUP AND SAUCER , REGENCY CAK...
557  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
561  (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...

                                            consequents   support
confidence  \
29                      (ASSORTED COLOUR BIRD ORNAMENT)  0.011537
0.723164
34                      (BAKING SET 9 PIECE RETROSPOT )  0.014060
```

```
0.710706
38                              (BATHROOM METAL SIGN)  0.010230
0.739414
40                      (BLUE HAPPY BIRTHDAY BUNTING)  0.011537
0.705234
41                      (PINK HAPPY BIRTHDAY BUNTING)  0.011537
0.715084
44               (WHITE HANGING HEART T-LIGHT HOLDER)  0.011492
0.732759
74                (GARDENERS KNEELING PAD KEEP CALM )  0.021000
0.725857
80                  (GREEN REGENCY TEACUP AND SAUCER)  0.021226
0.796954
85                 (ROSES REGENCY TEACUP AND SAUCER )  0.025101
0.759891
329                (ROSES REGENCY TEACUP AND SAUCER )  0.020324
0.763113
336                      (POPPY'S PLAYHOUSE BEDROOM )  0.011492
0.730659
337                       (POPPY'S PLAYHOUSE KITCHEN)  0.011492
0.799373
356                         (REGENCY TEA PLATE ROSES )  0.010455
0.843636
368           (SET/20 RED RETROSPOT PAPER NAPKINS )  0.010320
0.704615
370                      (SET/6 RED SPOTTY PAPER CUPS)  0.010635
0.726154
371                    (SET/6 RED SPOTTY PAPER PLATES)  0.010635
0.828070
373               (SMALL DOLLY MIX DESIGN ORANGE BOWL)  0.010185
0.782007
383             (WOODEN HEART CHRISTMAS SCANDINAVIAN)  0.014421
0.733945
384              (WOODEN STAR CHRISTMAS SCANDINAVIAN)  0.010230
0.819495
386                       (ALARM CLOCK BAKELIKE RED )  0.012078
0.779070
392                 (GREEN REGENCY TEACUP AND SAUCER)  0.012348
0.858934
393                  (PINK REGENCY TEACUP AND SAUCER)  0.012348
0.715405
398                 (GREEN REGENCY TEACUP AND SAUCER)  0.017891
0.880266
399                  (PINK REGENCY TEACUP AND SAUCER)  0.017891
0.712747
400                (ROSES REGENCY TEACUP AND SAUCER )  0.017891
0.842887
404                 (GREEN REGENCY TEACUP AND SAUCER)  0.014241
0.731481
406                (ROSES REGENCY TEACUP AND SAUCER )  0.014241
```

```
0.825065
412                           (JUMBO BAG RED RETROSPOT)  0.010500
0.792517
417                           (JUMBO BAG RED RETROSPOT)  0.010050
0.785211
519                           (LUNCH BAG RED RETROSPOT)  0.011492
0.704420
525                           (LUNCH BAG RED RETROSPOT)  0.010816
0.743034
543                           (LUNCH BAG RED RETROSPOT)  0.010140
0.725806
549                  (ROSES REGENCY TEACUP AND SAUCER )  0.012213
0.849530
554                   (GREEN REGENCY TEACUP AND SAUCER)  0.010861
0.889299
555                    (PINK REGENCY TEACUP AND SAUCER)  0.010861
0.762658
557                  (ROSES REGENCY TEACUP AND SAUCER )  0.010861
0.879562
561  (ROSES REGENCY TEACUP AND SAUCER , GREEN REGEN...  0.010861
0.755486

          lift
29   11.586286
34   17.900760
38   42.287602
40   43.712698
41   43.712698
44    8.077453
74   20.999687
80   24.126079
85   20.169830
329  20.255366
336  50.825466
337  50.825466
356  55.059679
368  21.301656
370  56.538084
371  56.538084
373  47.935728
383  35.024169
384  41.707763
386  19.060152
392  26.002386
393  26.860965
398  26.648164
399  26.761172
400  22.372815
404  22.144030
406  21.899759
```

```
412  10.703562
417  10.604892
519  11.761533
525  12.406265
543  12.118619
549  22.549122
554  26.921613
555  28.635171
557  23.346270
561  30.097364

# Display the rules with support, confidence, and lift
rules['support'] = rules['support'].round(4)
rules['confidence'] = rules['confidence'].round(4)
rules['lift'] = rules['lift'].round(4)

# Sort rules by lift
sorted_rules = rules.sort_values(by='lift', ascending=False)

# Display the sorted rules
print(sorted_rules[['antecedents', 'consequents', 'support',
'confidence', 'lift']])

                              antecedents  \
370       (SET/6 RED SPOTTY PAPER PLATES)
371         (SET/6 RED SPOTTY PAPER CUPS)
356            (REGENCY TEA PLATE GREEN )
357            (REGENCY TEA PLATE ROSES )
336           (POPPY'S PLAYHOUSE KITCHEN)
..                                    ...
297  (WHITE HANGING HEART T-LIGHT HOLDER)
193  (WHITE HANGING HEART T-LIGHT HOLDER)
192             (JUMBO BAG RED RETROSPOT)
354            (REGENCY CAKESTAND 3 TIER)
355  (WHITE HANGING HEART T-LIGHT HOLDER)

                              consequents  support  confidence
lift
370         (SET/6 RED SPOTTY PAPER CUPS)   0.0106      0.7262
56.5381
371       (SET/6 RED SPOTTY PAPER PLATES)   0.0106      0.8281
56.5381
356            (REGENCY TEA PLATE ROSES )   0.0105      0.8436
55.0597
357            (REGENCY TEA PLATE GREEN )   0.0105      0.6824
55.0597
336          (POPPY'S PLAYHOUSE BEDROOM )   0.0115      0.7307
50.8255
..                                    ...      ...         ...       ..
.
```

```
297              (LUNCH BAG RED RETROSPOT)   0.0102      0.1128
1.8828
193              (JUMBO BAG RED RETROSPOT)   0.0114      0.1262
1.7042
192  (WHITE HANGING HEART T-LIGHT HOLDER)   0.0114      0.1546
1.7042
354  (WHITE HANGING HEART T-LIGHT HOLDER)   0.0108      0.1269
1.3984
355             (REGENCY CAKESTAND 3 TIER)   0.0108      0.1187
1.3984

[568 rows x 5 columns]
```

```python
# Filter for strong association rules for bundling
bundles = sorted_rules[(sorted_rules['lift'] > 1.5) &
(sorted_rules['confidence'] > 0.5)]
print("Recommended Product Bundles:")
print(bundles[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

```
Recommended Product Bundles:
                                           antecedents  \
370                      (SET/6 RED SPOTTY PAPER PLATES)
371                        (SET/6 RED SPOTTY PAPER CUPS)
356                           (REGENCY TEA PLATE GREEN )
357                           (REGENCY TEA PLATE ROSES )
336                          (POPPY'S PLAYHOUSE KITCHEN)
..                                                  ...
394  (PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...
405  (ROSES REGENCY TEACUP AND SAUCER , GREEN REGEN...
327                     (PINK REGENCY TEACUP AND SAUCER)
83                     (GREEN REGENCY TEACUP AND SAUCER)
349                    (ROSES REGENCY TEACUP AND SAUCER )

                         consequents  support  confidence     lift
370    (SET/6 RED SPOTTY PAPER CUPS)   0.0106      0.7262  56.5381
371  (SET/6 RED SPOTTY PAPER PLATES)   0.0106      0.8281  56.5381
356       (REGENCY TEA PLATE ROSES )   0.0105      0.8436  55.0597
357       (REGENCY TEA PLATE GREEN )   0.0105      0.6824  55.0597
336     (POPPY'S PLAYHOUSE BEDROOM )   0.0115      0.7307  50.8255
..                               ...      ...         ...      ...
394       (REGENCY CAKESTAND 3 TIER)   0.0123      0.5817   6.8518
405       (REGENCY CAKESTAND 3 TIER)   0.0142      0.5673   6.6820
327       (REGENCY CAKESTAND 3 TIER)   0.0144      0.5398   6.3574
83        (REGENCY CAKESTAND 3 TIER)   0.0173      0.5225   6.1542
349       (REGENCY CAKESTAND 3 TIER)   0.0195      0.5167   6.0863

[170 rows x 5 columns]
```

Project B :

In this analysis, we looked at customer behavior using data from an online store. Our main goal was to find insights that could help improve marketing strategies and get customers more engaged. We started by cleaning the data and creating some useful metrics, like how long customers spend browsing and their total sales. We used K-Means clustering to group customers based on their buying habits and we try to apply a Random Forest classifier to predict how likely they were to make a purchase. We also used linear regression to estimate total sales based on the number of items bought and customer IDs. Our analysis revealed important patterns, such as how browsing time affects buying behavior. Overall, this study highlights how understanding customer interactions can enhance marketing efforts and boost sales.

Data preprocessing is an important step in getting the Online Retail II dataset ready for analysis. First, we loaded the dataset from an Excel file using the pandas library. After that, we looked at the data to see how many rows and columns there were, what types of data were included, and if there were any missing values. To deal with missing data, we removed any rows without a Customer ID since that information is crucial for our analysis. For entries where the Description was missing, we filled those with the label 'Unknown'. We also filtered out any transactions with negative quantities or prices because they don't make sense. Finally, we checked for and removed any duplicate rows to ensure that each transaction was unique. Feature engineering was key to improving our dataset for analysis. We converted the InvoiceDate into a datetime format so that we could pull out useful features like the day of the week and the hour when transactions occurred. We also created a new feature called TimeOfDay, which categorized the transactions into morning, afternoon, evening, and night based on the hour. This helps us understand when customers are shopping the most. Additionally, we added a Revenue column by multiplying the Quantity by the Price, which shows us how much money each transaction brought in. Once all these steps were done, we visualized the data to look for patterns in sales and revenue throughout the week, helping us make informed decisions about marketing and operations.

In our project, we aimed to segment customers based on their browsing and purchasing habits using K-Means clustering. We started by gathering and organizing our customer data. First, we aggregated the data at the customer level, calculating key metrics like total revenue, the number of transactions, and total quantity purchased. We also computed the recency, which measures the number of days since the customer's last purchase. This step helped us create a clearer picture of each customer's behavior. After that, we scaled the features using StandardScaler, which ensured that all metrics had the same influence on the clustering process. This is important because K-Means works by calculating distances between points, and we wanted to make sure that larger numbers didn’t unfairly dominate the results. Next, we applied the K-Means algorithm to our valued customer data. We chose to create four clusters, which helps in identifying different types of customers based on their behaviors. The algorithm works by first randomly selecting initial cluster centroids and then assigning each customer to the nearest centroid. After assigning the customers, K-Means recalculates the centroids based on the current group of points in each cluster and repeats this process until the centroids no longer change significantly. Once the clustering was done, we analyzed the characteristics of each group. For instance, we created personalized marketing messages for each segment based on their purchasing habits. This way, we could tailor promotions to specific groups, like offering discounts to less active customers to encourage them to return. Additionally, we examined the best times to engage with these customers by analyzing when they were most active, ensuring our communication efforts were optimized. Overall, this approach not only helped us understand our customers better but also enabled us to improve our marketing strategies effectively.

In the next steps, we conducted an analysis of customer engagement and purchasing behavior using various data visualization and machine learning techniques. We created bar plots and histograms to explore the distribution of engagement counts and customer features like total revenue and recency. The bar plot displayed engagement counts segmented by purchase hour and customer clusters, helping us identify peak purchasing times and how different customer groups interact with our products. The histogram for total revenue showed a right-skewed distribution, indicating that most customers spent relatively low amounts, while a few spent significantly more. For example, we noticed that many customers made purchases in lower revenue ranges, but a small number of customers generated a substantial portion of the total revenue. For the modeling part, I used a Random Forest classifier, an ensemble learning method that works well for classification tasks. This algorithm builds multiple decision trees during training and predicts the most common class for classification, which helps reduce overfitting and improves performance. My model achieved an impressive accuracy of 1.00, meaning it correctly classified all instances in the test set, which included 875 samples. The classification report showed perfect precision, recall, and F1-scores of 1.00 for the "MadePurchase" class, indicating that the model effectively identified customers who made purchases. Additionally, cross-validation scores confirmed the model's reliability, achieving perfect results across all five folds. However, a major limitation of this analysis was that the dataset only included transaction data, lacking customer viewing data or information on abandoned carts. This meant I couldn't determine what items customers had in their carts or predict how likely they were to make a purchase based on their browsing behavior. This limitation explains why all 4,372 instances were classified as '1' for the "MadePurchase" target variable since the dataset consisted only of completed transactions. Given these challenges, we decided to shift our focus to analyzing other aspects of customer behavior that we could accurately evaluate with the available data.

Since I couldn't create a classification model to predict whether a customer's browsing session would end in a purchase due to incorrect data, I decided to rethink how I could analyze the data to still provide value for a retail store. Given the available data, which included InvoiceNo, StockCode, Description, Quantity, InvoiceDate, and UnitPrice, I thought it would be helpful for a retail owner to know the probability of a customer making a purchase based on their browsing time. If I knew that customers who browse longer are more likely to buy something, I could suggest ways to enhance their experience on the website.

I started by preprocessing the dataset, sorting it by Customer ID and InvoiceDate to ensure the transactions were in chronological order. This allowed me to accurately calculate the browsing durations between purchases. I calculated browsing time as the difference between consecutive transactions for each customer and converted this duration into seconds. To handle any missing values from the first transaction (which doesn't have a previous transaction to compare), I filled those NaN values with 0 seconds, creating a new column called BrowsingTime, which was crucial for my analysis.

Next, I aggregated the average browsing time and average price for each item using the mean. This created a DataFrame, item_browsing, that showed the relationship between how long customers spent browsing and the prices of items. I visualized this analysis with a scatter plot, which depicted average browsing time against average item prices, helping to identify any trends between these variables.

After that, I focused on customer-level browsing metrics. I aggregated the total browsing time for each customer to create the customer_browsing_time DataFrame, which I then merged with customer_data. This merge introduced a new feature, TotalBrowsingTime, indicating how much time each customer spent browsing before making purchases. For example, customer ID 12347 had a TotalBrowsingTime of over 31 million seconds, suggesting significant browsing activity that could relate to their higher spending.

Continuing with the analysis, I added a feature for the average price per customer and prepared the data for modeling. I dropped irrelevant columns and created a target variable, MadePurchase, to indicate whether a purchase was made. I trained a Random Forest classifier to predict this outcome based on customer features. The model performed extremely well, achieving perfect precision, recall, and F1-score of 1.00 on the test dataset, indicating it classified all transactions accurately. However, this perfect performance raised concerns about potential overfitting or a lack of complexity in the dataset since all transactions were purchases.

I then categorized browsing time into three groups: 'Short', 'Medium', and 'Long', based on specific time thresholds. This allowed for a more detailed analysis of customer behavior. The browsing_counts showed that most sessions were classified as 'Short' (385,345 instances), while only a small fraction were 'Long' (15,789 instances). I also calculated the average price by browsing categories and total spending. The results showed that 'Long' browsing sessions had an average price of about $8.73, while 'Short' sessions averaged $3.23, indicating that customers who browse longer tend to look at higher-priced items. Interestingly, 'Short' browsing sessions generated the highest revenue at over $7.72 million, despite their lower average prices, while 'Medium' sessions had a negative total spend, suggesting possible data anomalies that need further investigation.

Lastly, I created dynamic offers based on browsing categories and checked the offer distribution. The results confirmed that 'Free Shipping' was the most common offer, mainly given to customers in the 'Short' browsing category. I also explored daily transaction patterns by extracting the day from InvoiceDate and aggregating customer transactions per day. A scatter plot of total browsing time against transactions per day showed a weak positive correlation of about 0.076, indicating that more browsing time didn't necessarily lead to more daily transactions. To dig deeper, I created a binary target variable for customers with more than three transactions per day. The subsequent classification model revealed an accuracy of around 73.5%. Overall, these analyses provide valuable insights into customer behavior and spending patterns, highlighting potential marketing strategies based on browsing behavior.

For linear regression, we created a new column called TotalSales in the merged_data DataFrame. This column calculates total sales by multiplying the number of items sold by their price. We also changed the BrowsingTime column to a datetime format and made a new column called BrowsingDate to only include the date. By grouping the data, we found the total sales, total quantity sold, and number of transactions for each customer on a daily basis. This resulted in a new DataFrame called sales_data, which gives us a clear picture of each customer's sales performance day by day. After that, we set up a linear regression model to predict TotalSales based on TotalQuantity and Customer ID. We split the data into training and testing sets, using 80% for training and 20% for testing. We trained the model on the training data and then made predictions on the test set. We evaluated the model's performance using Mean Squared Error (MSE), which measures how close the predicted sales are to the actual sales. The model had a Mean Squared Error of about 53,130,896, indicating the average squared difference between actual and predicted sales. The R^2 score was 0.24, meaning the model explains about 24% of the variance in sales data, which shows there's still room for improvement. The results weren't great, likely because the data we used, especially BrowsingTime and Customer ID, didn't have a strong connection to TotalSales. Ideally, we would want a different dataset that shows stronger relationships. Still, this exercise helped us practice various analyses and improve our skills in working with data.]

When we looked at seasonal discount timing, we wanted to identify periods of low sales to suggest discounts or exclusive deals during off-peak hours. Our monthly sales figures showed that February 2011 had the lowest quantity sold at 262,243 units, while January 2011 followed closely with 268,755 units. Although there were no months with zero sales, these lower numbers suggest that these months are less popular for sales compared to others, especially when looking at peak sales in later months like October and November 2011. In our weekly sales analysis, Saturday stood out as the least utilized day, showing a recorded quantity of NaN (Not a Number), meaning there was no sales data available for Saturdays. This indicates that Saturdays might either have very few sales or are not recorded properly. On the other hand, weekdays like Monday, Tuesday, and Thursday show higher sales volumes, suggesting that customers prefer to shop during the week. In summary, February 2011 is the month with the least sales activity, and

Saturday is identified as the day with potentially zero or very minimal sales. This information can help us plan marketing strategies and discount timings to improve sales during these underperforming times. By offering discounts during these slower periods, especially on weekends, retailers can boost sales. Understanding these trends allows retailers to optimize their marketing strategies and manage their inventory more effectively, ultimately leading to better overall sales performance.

In my analysis of customer purchase behavior, I used the Apriori algorithm and association rule mining to find patterns in the buying data. I started by creating a basket DataFrame that organized the transaction data by invoice and product description. I converted the counts into boolean values to show whether products were included in each basket. This transformation helped us identify frequent itemsets, from which we derived association rules based on metrics like support, confidence, and lift. The filtered rules showed strong connections between different products, which can be useful for marketing strategies. For example, we found that the Pink Regency Teacup and Saucer is often bought together with the Green Regency Teacup and Saucer, as well as the Roses Regency Teacup and Saucer. Additionally, the Set of Red Spotty Paper Plates is frequently linked to the Set of Red Spotty Paper Cups. These insights can help businesses boost sales and create better discounts for customers who abandon their carts. Even though we didn't have specific data on what items were in customers' carts, we can still use this information. For example, when someone adds an item to their cart, the system could suggest other products that people usually buy together. This would not only improve the shopping experience but also encourage customers to buy more items. By using these strategies, businesses can increase their sales and engage customers more effectively.

Some more examples of promotions:

Customers who bought the Painted Metal Pears Assorted also bought the Assorted Colour Bird Ornament.

Customers who bought the Baking Set Spaceboy Design also bought the Baking Set 9 Piece Retrosport.

Customers who bought the Toilet Metal Sign also bought the Bathroom Metal Sign.

Customers who bought the Happy Birthday Bunting (Pink) also bought the Happy Birthday Bunting (Blue).

Customers who bought the Candleholder Pink Hanging Heart also bought the White Hanging Heart T-Light Holder.

Customers who bought the Gardener's Kneeling Pad also bought the Gardener's Kneeling Pad Keep Calm.

Customers who bought the Pink Regency Teacup and Saucer also bought the Green Regency Teacup and Saucer.

In conclusion, analyzing customer behavior using data from an online store has given us some really valuable insights that can help shape marketing strategies and boost customer engagement. By carefully cleaning the data and using techniques like K-Means clustering and Random Forest classification, we were able to identify different customer segments and predict their buying behavior effectively. Our findings show that how long customers browse is a key factor in their purchasing decisions.

It was a bit disappointing that the data we received wasn't what we expected—it mainly focused on transactions and didn't include the customer interaction details we needed. This limited our ability to understand what influences customers when they decide to buy something. Still, this exercise turned out to be beneficial as it pushed us to explore different data analysis methods and think creatively about the kinds of data that could be useful for retailers.

By using data visualization, we were able to see trends in customer engagement, like when people are most likely to make purchases and how much they usually spend based on how long they browse. These insights can help retailers improve their marketing strategies, especially by offering targeted discounts during slower times. Plus, our exploration of association rules showed us how different products are connected, which can create great opportunities for cross-selling and promotional strategies. For example, figuring out which items are often bought together can help businesses design effective marketing campaigns that make customers happier and boost sales.