```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.colors import LinearSegmentedColormap
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
import plotly.express as px
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.model_selection import train_test_split
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn import tree
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.cluster import  DBSCAN
from sklearn.metrics import silhouette_score
import warnings
from pickle import dump
from sklearn.mixture import GaussianMixture
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA

df = pd.read_csv('OnlineRetail.csv', encoding="ISO-8859-1")
df.head()
```

```
  InvoiceNo 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  UnitPrice  CustomerID         Country
0  12/1/2010 8:26       2.55     17850.0  United Kingdom
1  12/1/2010 8:26       3.39     17850.0  United Kingdom
2  12/1/2010 8:26       2.75     17850.0  United Kingdom
3  12/1/2010 8:26       3.39     17850.0  United Kingdom
4  12/1/2010 8:26       3.39     17850.0  United Kingdom
```

```
df.shape
```

```
(541909, 8)
```

```
print("\nData Types and Missing Values:")
print(df.info())
```

```

Data Types and Missing Values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
None
```

```
df.isnull().sum()
```

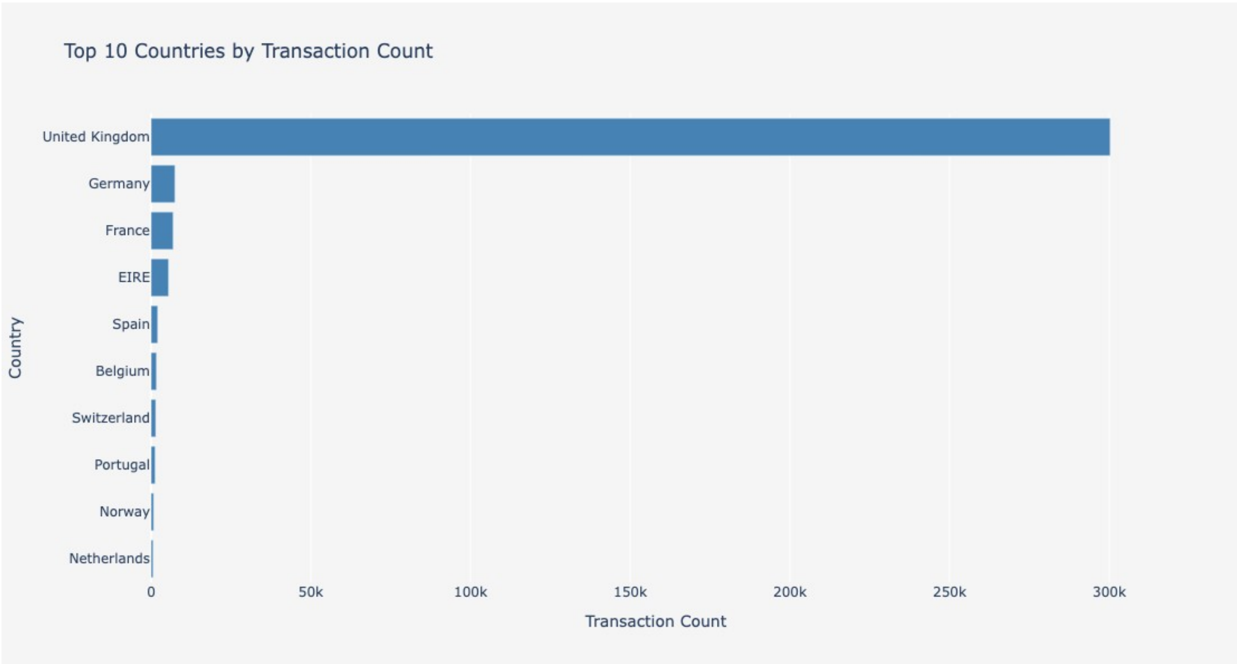
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

```
df = df.dropna(subset=['CustomerID']) #Here removing rows with missing
CustomerID
```

```
df['Description'] = df['Description'].fillna('Unknown') #Here filling
missing description with Unknown

print("Summary Statistics:")
print(df.describe())

Summary Statistics:
            Quantity      UnitPrice     CustomerID
count  406829.000000  406829.000000  406829.000000
mean       12.061303       3.460471   15287.690570
std       248.693370      69.315162    1713.600303
min    -80995.000000       0.000000   12346.000000
25%         2.000000       1.250000   13953.000000
50%         5.000000       1.950000   15152.000000
75%        12.000000       3.750000   16791.000000
max     80995.000000   38970.000000   18287.000000

#negative values handiling
df = df[df['UnitPrice']>0]
df = df[df['Quantity']>0]

print("Summary Statistics:")
print(df.describe())

Summary Statistics:
            Quantity      UnitPrice     CustomerID
count  397884.000000  397884.000000  397884.000000
mean       12.988238       3.116488   15294.423453
std       179.331775      22.097877    1713.141560
min         1.000000       0.001000   12346.000000
25%         2.000000       1.250000   13969.000000
50%         6.000000       1.950000   15159.000000
75%        12.000000       3.750000   16795.000000
max     80995.000000    8142.750000   18287.000000

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

# Ensure CustomerID is integer
df['CustomerID'] = df['CustomerID'].astype(int)

# Convert Price to float
df['UnitPrice'] = df['UnitPrice'].astype(float)

# How many duplicate rows are there?
df.duplicated().sum()

5192

df.drop_duplicates(inplace=True)
```

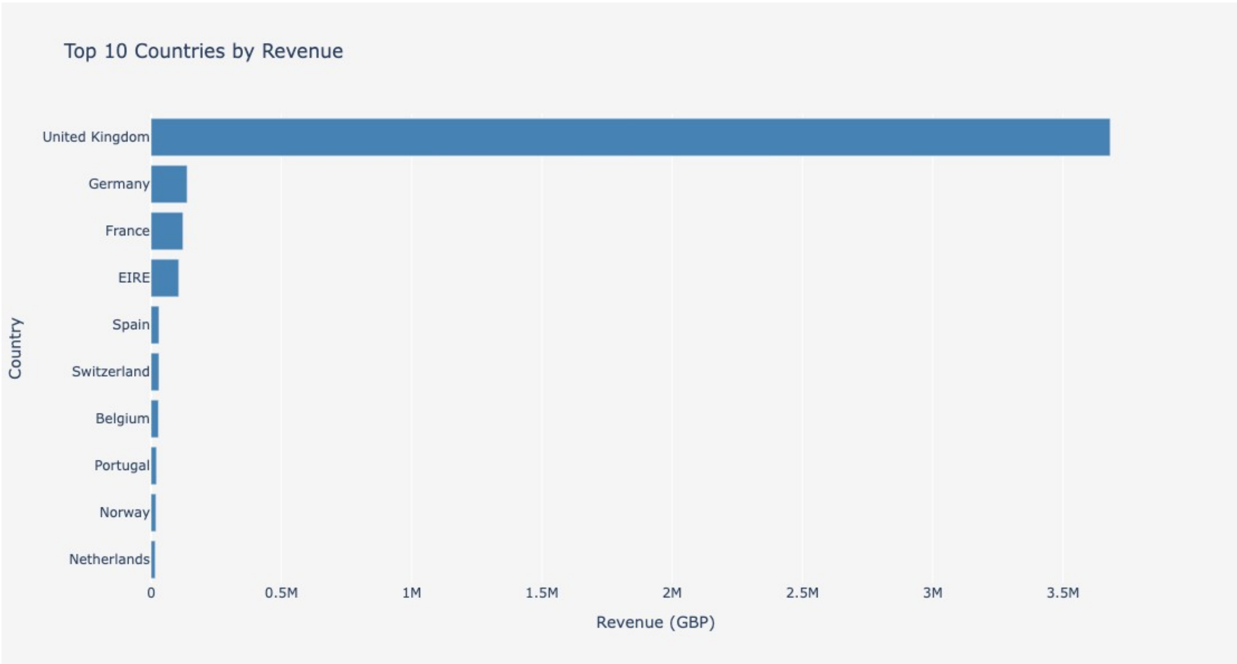
```python
def remove_outliers(df, column_name):

    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter the DataFrame to remove outliers
    return df[(df[column_name] >= lower_bound) & (df[column_name] <=
upper_bound)]

df=remove_outliers(df, 'Quantity')
df=remove_outliers(df, 'UnitPrice')

# Create TotalAmount column
df['Total_Amount'] = df['Quantity'] * df['UnitPrice']
# Extract date components
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month
df['Day'] = df['InvoiceDate'].dt.day
df['DayOfWeek'] = df['InvoiceDate'].dt.dayofweek

# Calculate the transaction count per country
country_counts = df['Country'].value_counts().head(10).reset_index()
country_counts.columns = ['Country', 'TransactionCount']

fig = px.bar(
    country_counts,
    x='TransactionCount',
    y='Country',
    orientation='h',
    title='Top 10 Countries by Transaction Count',
    color='Country',
    color_discrete_sequence=['steelblue']
)

fig.update_layout(
    xaxis_title='Transaction Count',
    yaxis_title='Country',
    yaxis=dict(categoryorder='total ascending'),
    showlegend=False,
    legend=dict(orientation="h", yanchor="bottom", y=-0.3,
xanchor="center", x=0.5),
    plot_bgcolor='rgba(245, 245, 245, 1)',
    paper_bgcolor='rgba(245, 245, 245, 1)',
    width=1200,
    height=600,
)
```

```
fig.show()
```



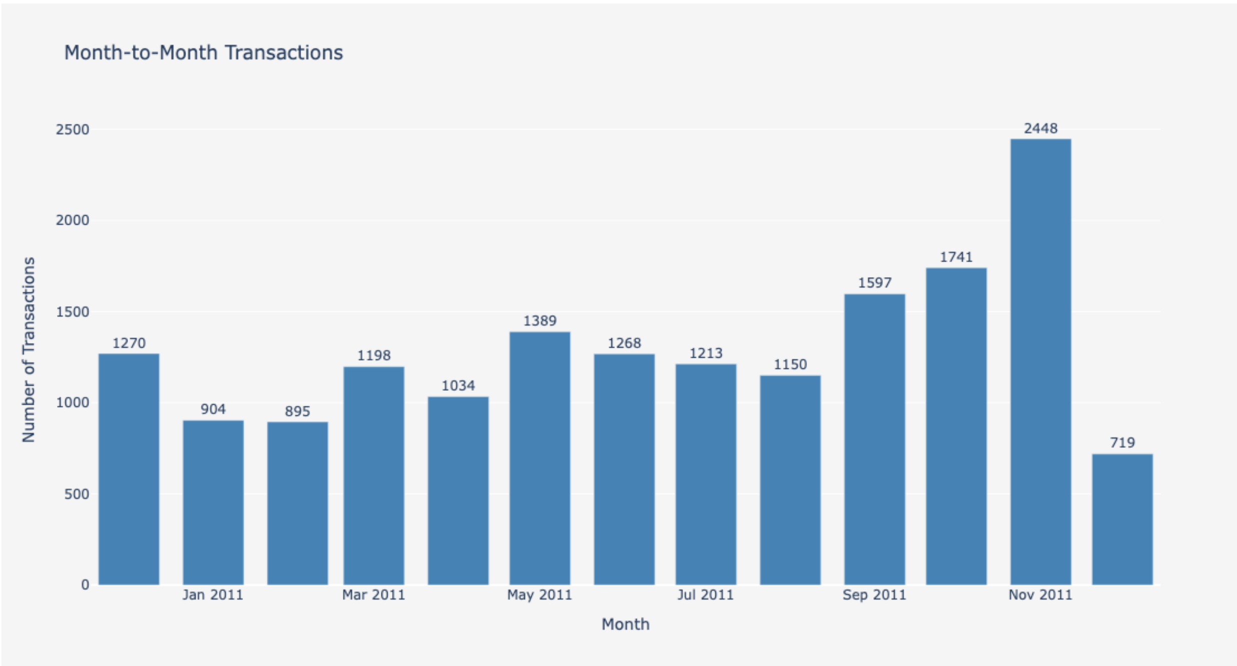

```
# Calculate the total revenue per country
country_revenue = df.groupby('Country')
['Total_Amount'].sum().reset_index()

top_countries_revenue = country_revenue.nlargest(10, 'Total_Amount')
fig = px.bar(
    top_countries_revenue,
    x='Total_Amount',
    y='Country',
    orientation='h',
    title='Top 10 Countries by Revenue',
    color_discrete_sequence=['steelblue']
)

fig.update_layout(
    xaxis_title='Revenue (GBP)',
    yaxis_title='Country',
    yaxis=dict(categoryorder='total ascending'),
    showlegend=False,
     legend=dict(orientation="h", yanchor="bottom", y=-0.3,
xanchor="center", x=0.5),
    plot_bgcolor='rgba(245, 245, 245, 1)',
    paper_bgcolor='rgba(245, 245, 245, 1)',
    width=1200,
    height=600,
)
```

```
fig.show()
```



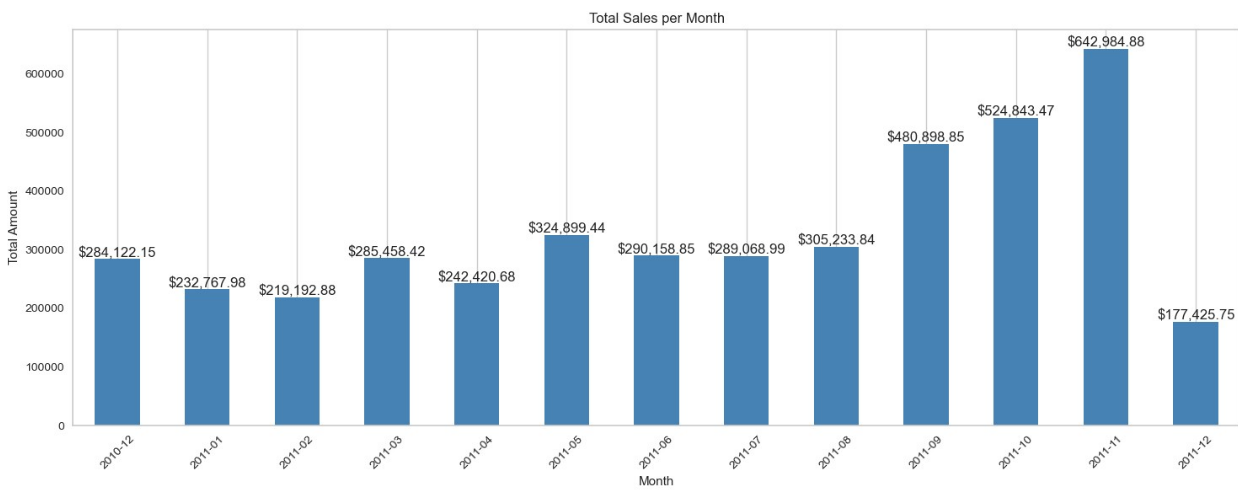

```
# Extract year and month from InvoiceDate
df['YearMonth'] = df['InvoiceDate'].dt.to_period('M')
monthly_transactions = df.groupby('YearMonth')
['InvoiceNo'].nunique().reset_index()
monthly_transactions['YearMonth'] =
monthly_transactions['YearMonth'].dt.to_timestamp()

fig = px.bar(
    monthly_transactions,
    x='YearMonth',
    y='InvoiceNo',
    title='Month-to-Month Transactions',
    labels={'YearMonth': 'Month', 'InvoiceNo': 'Number of
Transactions'},
    text='InvoiceNo'
)
fig.update_traces(marker_color='steelblue', textposition='outside',
texttemplate='%{text}')
fig.update_layout(
    xaxis_title='Month',
    yaxis_title='Number of Transactions',
    showlegend=False,
    bargap=0.2,
    legend=dict(orientation="h", yanchor="bottom", y=-0.3,
xanchor="center", x=0.5),
    plot_bgcolor='rgba(245, 245, 245, 1)',
```

```
    paper_bgcolor='rgba(245, 245, 245, 1)',
    width=1200,
    height=600,

)

fig.show()
```



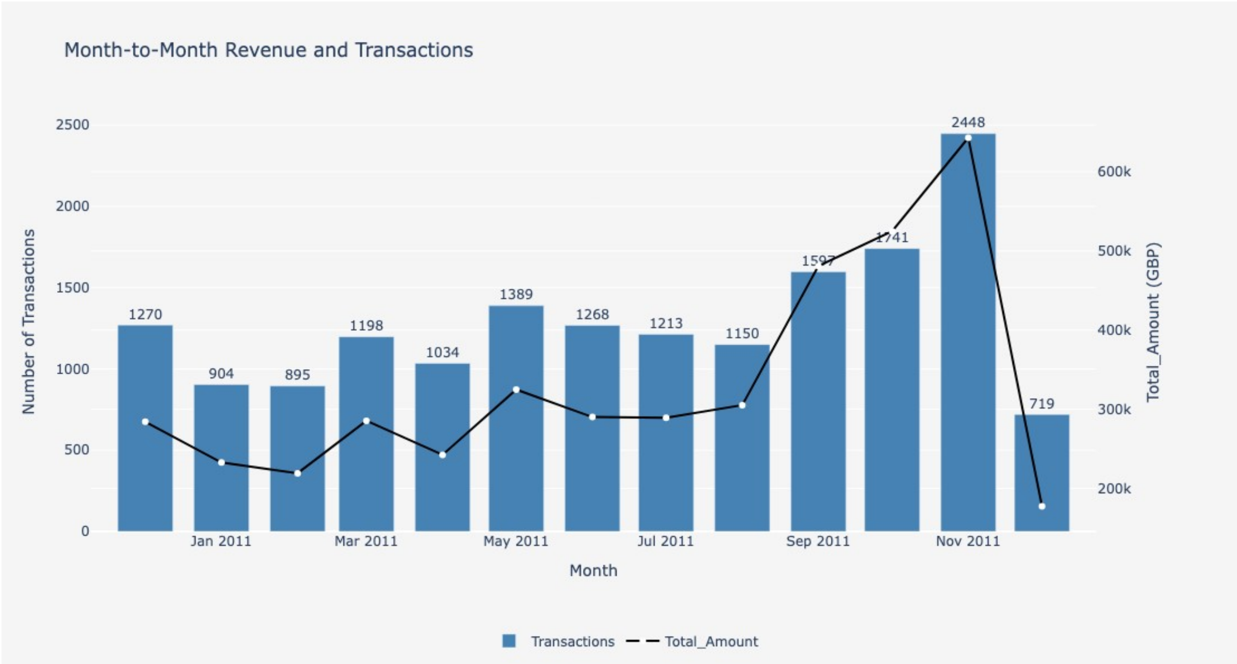

```
df['Total_Amount'] = df['Quantity'] * df['UnitPrice']

sales_per_month = df.groupby(df['InvoiceDate'].dt.to_period('M'))
['Total_Amount'].sum()

plt.figure(figsize=(15, 6))
bars = sales_per_month.plot(kind='bar', color='steelblue')
plt.title('Total Sales per Month')
plt.xlabel('Month')
plt.ylabel('Total Amount')
plt.xticks(rotation=45)
plt.grid(axis='y')

for bar in bars.patches:
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 1,
             f'${bar.get_height():,.2f}', ha='center', va='bottom'),


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

```
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/
pylabtools.py:77: DeprecationWarning:

backend2gui is deprecated since IPython 8.24, backends are managed in
matplotlib and can be externally registered.

/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py
:77: DeprecationWarning:

backend2gui is deprecated since IPython 8.24, backends are managed in
matplotlib and can be externally registered.

/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py
:77: DeprecationWarning:

backend2gui is deprecated since IPython 8.24, backends are managed in
matplotlib and can be externally registered.
```




```
# Extract year and month from InvoiceDate
df['YearMonth'] = df['InvoiceDate'].dt.to_period('M')
monthly_revenue = df.groupby('YearMonth')
['Total_Amount'].sum().reset_index()

monthly_transactions = df.groupby('YearMonth')
['InvoiceNo'].nunique().reset_index()
monthly_revenue['YearMonth'] =
monthly_revenue['YearMonth'].dt.to_timestamp()
monthly_transactions['YearMonth'] =
monthly_transactions['YearMonth'].dt.to_timestamp()

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(
    go.Bar(
```

```
        x=monthly_transactions['YearMonth'],
        y=monthly_transactions['InvoiceNo'],
        name='Transactions',
        text=monthly_transactions['InvoiceNo'],
        textposition='outside',
        marker_color='steelblue'
    ),
    secondary_y=False,
)

fig.add_trace(
    go.Scatter(
        x=monthly_revenue['YearMonth'],
        y=monthly_revenue['Total_Amount'],
        name='Total_Amount',
        mode='lines+markers',
        marker=dict(color='white'),
        line=dict(width=2, color='black'),
        text=monthly_revenue['Total_Amount'],
    ),
    secondary_y=True,
)

fig.update_layout(
    title_text='Month-to-Month Revenue and Transactions',
    xaxis_title='Month',
    bargap=0.2,
    legend=dict(orientation="h", yanchor="bottom", y=-0.3,
xanchor="center", x=0.5),  # Center legend below chart
    plot_bgcolor='rgba(245, 245, 245, 1)',
    paper_bgcolor='rgba(245, 245, 245, 1)',
    width=1200,
    height=600,
)

fig.update_yaxes(title_text="Total_Amount (GBP)", secondary_y=True)
fig.update_yaxes(title_text="Number of Transactions",
secondary_y=False)
fig.show()
```




```
import datetime as dt

snapshot_date = df['InvoiceDate'].max() + dt.timedelta(days=1)

# Calculating Recency, Frequency, and Monetary value for each customer
rfm = df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'nunique',
    'Total_Amount': 'sum'
}).reset_index()

rfm.columns = ['CustomerID', 'Recency', 'Frequency', 'Monetary']
rfm.head()
```

	CustomerID	Recency	Frequency	Monetary
0	12347	2	7	3314.73
1	12348	249	3	90.20
2	12349	19	1	999.15
3	12350	310	1	294.40
4	12352	36	7	1130.94

```
#RFM segmentation
def rfm_segmentation(data):
    # Segment Recency
    data['R'] = pd.qcut(data['Recency'], 5, labels=[5, 4, 3, 2, 1],
duplicates='drop')

    # Segment Frequency
    data['F'] = pd.qcut(data['Frequency'].rank(method='first'), 5,
labels=[1, 2, 3, 4, 5], duplicates='drop')
```

```
    # Segment Monetary
    data['M'] = pd.qcut(data['Monetary'], 5, labels=[1, 2, 3, 4, 5],
duplicates='drop')

    # Calculate RFM Score
    data['RFM_Score'] = data['R'].astype(str) + data['F'].astype(str)
+ data['M'].astype(str)

    return data

rfm = rfm_segmentation(rfm)
rfm.head()
```

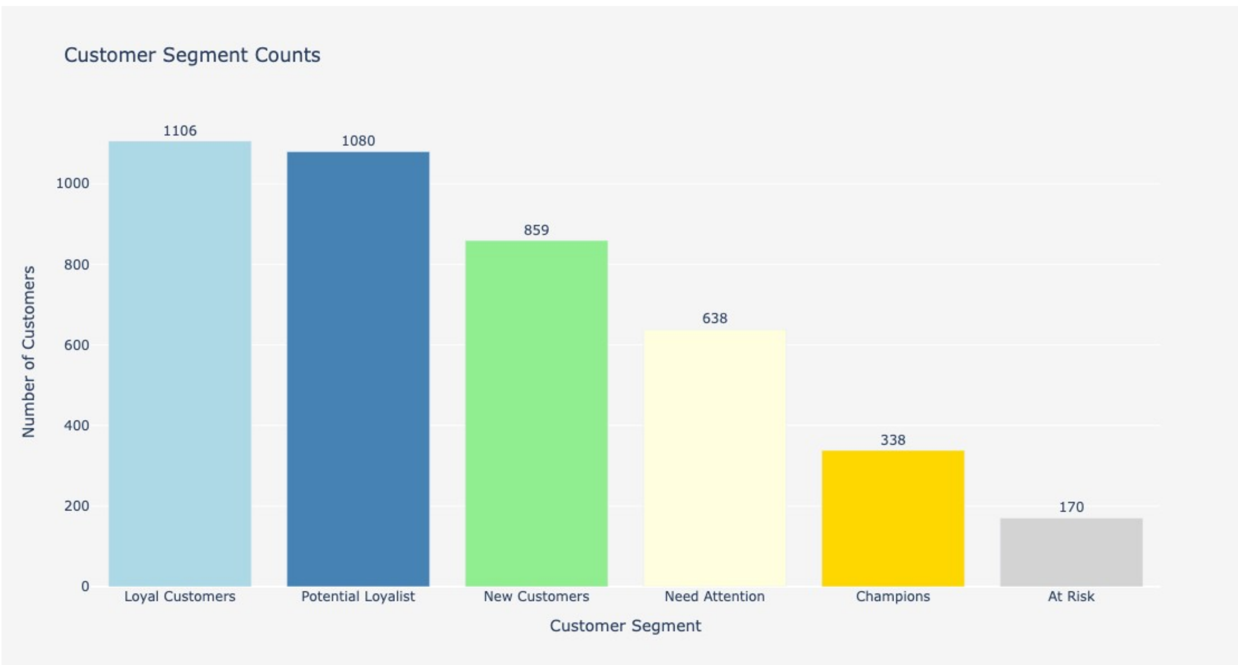
```
   CustomerID  Recency  Frequency  Monetary  R  F  M RFM_Score
0       12347        2          7   3314.73  5  5  5       555
1       12348      249          3     90.20  1  3  1       131
2       12349       19          1    999.15  4  1  4       414
3       12350      310          1    294.40  1  1  2       112
4       12352       36          7   1130.94  3  5  4       354
```

```
# RFM score segments
def rfm_segment(data):
    segment = []
    for score in data['RFM_Score']:
        if score == '555':
            segment.append('Champions')
        elif score[0] == '5' or score[1] == '5' or score[2] == '5':
            segment.append('Loyal Customers')
        elif score[0] == '4' or score[1] == '4' or score[2] == '4':
            segment.append('Potential Loyalist')
        elif score[0] == '3' or score[1] == '3' or score[2] == '3':
            segment.append('New Customers')
        elif score[0] == '2' or score[1] == '2' or score[2] == '2':
            segment.append('Need Attention')
        else:
            segment.append('At Risk')
    data['Segment'] = segment
    return data

rfm = rfm_segment(rfm)
rfm.head(15)
```

```
    CustomerID  Recency  Frequency  Monetary  R  F  M RFM_Score  \
0        12347        2          7   3314.73  5  5  5       555
1        12348      249          3     90.20  1  3  1       131
2        12349       19          1    999.15  4  1  4       414
3        12350      310          1    294.40  1  1  2       112
4        12352       36          7   1130.94  3  5  4       354
5        12353      204          1     29.30  1  1  1       111
```

```
6        12354      232          1    682.69  1  1  4       114
7        12355      214          1    219.00  1  1  2       112
8        12356      246          2   1086.56  1  2  4       124
9        12357       33          1   3315.41  4  1  5       415
10       12358        2          2    878.22  5  2  4       524
11       12359       58          4   3161.58  3  4  5       345
12       12360       52          3   1843.16  3  3  5       335
13       12361      287          1    174.90  1  1  2       112
14       12362        3         10   4098.94  5  5  5       555

               Segment
0            Champions
1        New Customers
2   Potential Loyalist
3       Need Attention
4      Loyal Customers
5              At Risk
6   Potential Loyalist
7       Need Attention
8   Potential Loyalist
9      Loyal Customers
10     Loyal Customers
11     Loyal Customers
12     Loyal Customers
13      Need Attention
14           Champions

color_map = {
    'Potential Loyalist': 'steelblue',
    'Loyal Customers': 'lightblue',
    'New Customers': 'lightgreen',
    'Need Attention': 'lightyellow',
    'Champions': 'gold',
    'At Risk': 'lightgrey'
}

# Count the number of customers in each segment
segment_counts = rfm['Segment'].value_counts().reset_index()
segment_counts.columns = ['Segment', 'Count']

fig = px.bar(
    segment_counts,
    x='Segment',
    y='Count',
    title='Customer Segment Counts',
    labels={'Segment': 'Customer Segment', 'Count': 'Number of
Customers'},
    text='Count',
    color='Segment',
    color_discrete_map=color_map
```

```
)

fig.update_traces(textposition='outside', texttemplate='%{text:.0f}')
fig.update_layout(
    xaxis_title='Customer Segment',
    yaxis_title='Number of Customers',
    showlegend=False,
    bargap=0.2,
    legend=dict(orientation="h", yanchor="bottom", y=-0.3,
xanchor="center", x=0.5),  # Center legend below chart
    plot_bgcolor='rgba(245, 245, 245, 1)',
    paper_bgcolor='rgba(245, 245, 245, 1)',
    width=1200,
    height=600,
)

fig.show()
```

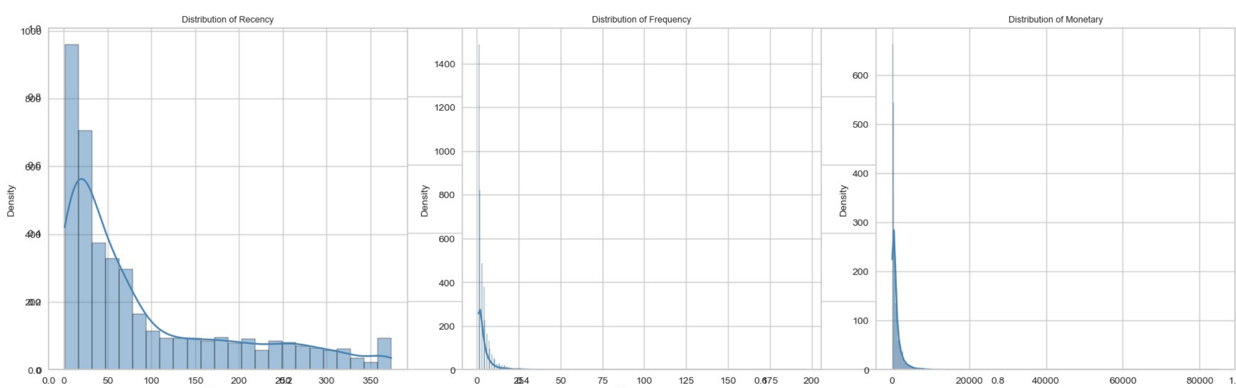



```
df['YearMonth'] = df['YearMonth'].dt.to_timestamp()

# Encode Country
le = LabelEncoder()
df['CountryEncoded'] = le.fit_transform(df['Country'])

# Check for any remaining issues
print(df.isnull().sum())
print(df.dtypes)

print(df.head())
```

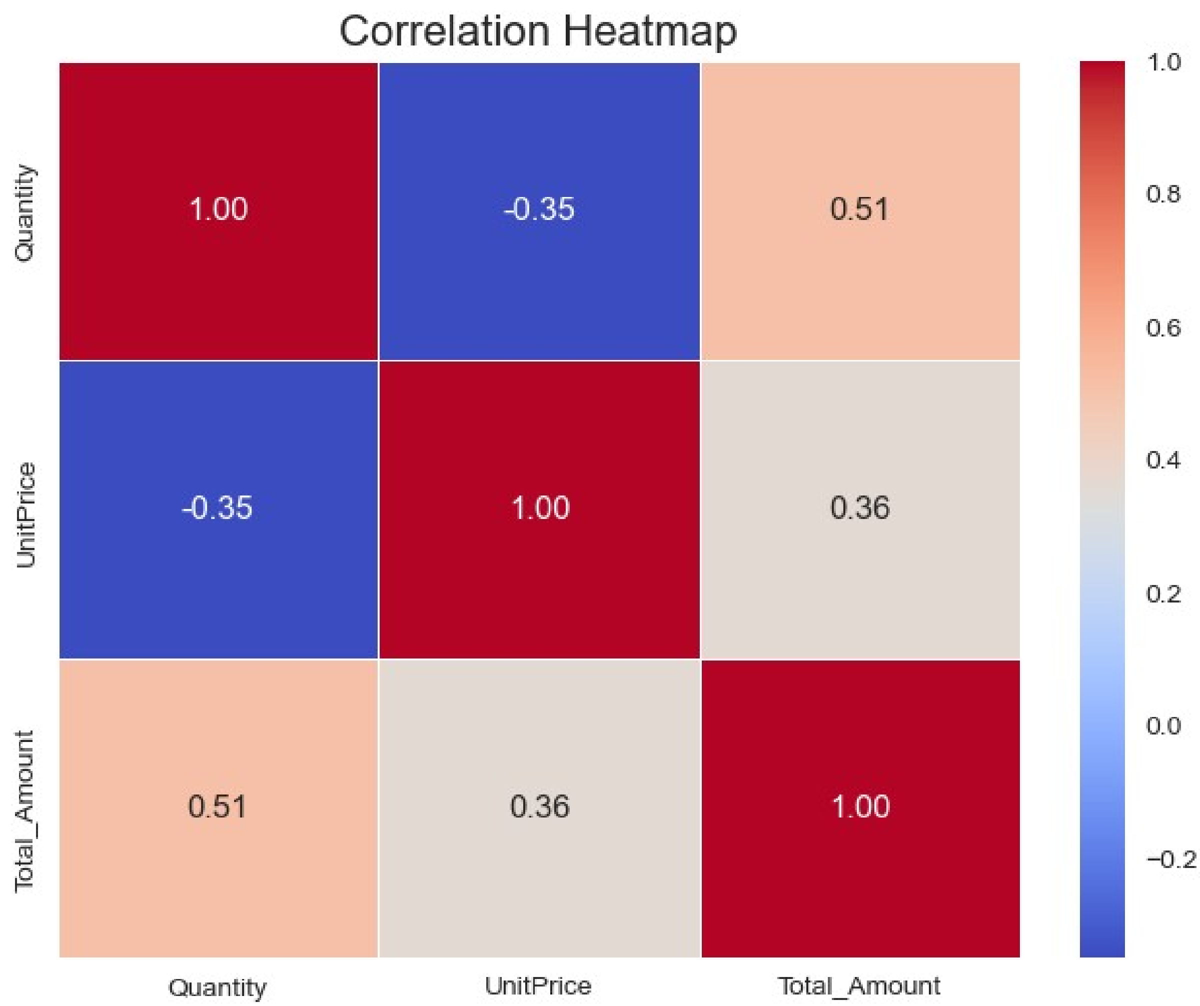
```
InvoiceNo          0
StockCode          0
Description        0
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID         0
Country            0
Total_Amount       0
Year               0
Month              0
Day                0
DayOfWeek          0
YearMonth          0
CountryEncoded     0
dtype: int64
InvoiceNo                 object
StockCode                 object
Description               object
Quantity                   int64
InvoiceDate       datetime64[ns]
UnitPrice                float64
CustomerID                 int64
Country                   object
Total_Amount             float64
Year                       int32
Month                      int32
Day                        int32
DayOfWeek                  int32
YearMonth         datetime64[ns]
CountryEncoded             int64
dtype: object
  InvoiceNo 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  UnitPrice  CustomerID         Country
Total_Amount  \
0 2010-12-01 08:26:00       2.55       17850  United Kingdom
15.30
1 2010-12-01 08:26:00       3.39       17850  United Kingdom
```

```
20.34
2 2010-12-01 08:26:00       2.75       17850  United Kingdom
22.00
3 2010-12-01 08:26:00       3.39       17850  United Kingdom
20.34
4 2010-12-01 08:26:00       3.39       17850  United Kingdom
20.34

   Year  Month  Day  DayOfWeek  YearMonth  CountryEncoded
0  2010     12    1          2 2010-12-01              35
1  2010     12    1          2 2010-12-01              35
2  2010     12    1          2 2010-12-01              35
3  2010     12    1          2 2010-12-01              35
4  2010     12    1          2 2010-12-01              35
```

```
plt.subplots(figsize=(18, 6))
rfm_features = ['Recency', 'Frequency', 'Monetary']
count = 1
for feature in rfm_features:
    plt.subplot(1, 3, count)
    sns.histplot(rfm[feature], kde=True, color='steelblue')
    plt.title(f"Distribution of {feature}", fontsize=9)
    plt.xlabel(feature, fontsize=10)
    plt.ylabel("Density", fontsize=10)
    count += 1
plt.tight_layout()
plt.show()
```




```
correlation_matrix = df[['Quantity', 'UnitPrice',
'Total_Amount']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=0.5)

plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```




```
# apply log transformation on the original rfm dataframe
import math
rfm['Recency_log'] = rfm['Recency'].apply(math.log)
rfm['Frequency_log'] = rfm['Frequency'].apply(math.log)
rfm['Monetary_log'] = rfm['Monetary'].apply(math.log)

rfm.head()
```

```
   CustomerID  Recency  Frequency  Monetary  R  F  M RFM_Score  \
0       12347        2          7   3314.73  5  5  5       555
1       12348      249          3     90.20  1  3  1       131
2       12349       19          1    999.15  4  1  4       414
3       12350      310          1    294.40  1  1  2       112
4       12352       36          7   1130.94  3  5  4       354

             Segment  Recency_log  Frequency_log  Monetary_log
0          Champions     0.693147       1.945910      8.106131
1      New Customers     5.517453       1.098612      4.502029
2  Potential Loyalist     2.944439       0.000000      6.906905
```

```
3      Need Attention     5.736572       0.000000      5.684939
4     Loyal Customers     3.583519       1.945910      7.030804
```

```
features = ['Recency_log', 'Frequency_log', 'Monetary_log']
X_features = rfm[features].values
scaler = StandardScaler()
X = scaler.fit_transform(X_features)

# Using Recency, Frequency, and Monetary as features for clustering
features = rfm[['Recency_log', 'Frequency_log',
'Monetary_log']].values

kmeans = KMeans(n_clusters=3, random_state=42)
rfm['Cluster'] = kmeans.fit_predict(features)

df = df.merge(rfm[['CustomerID', 'Cluster']], on='CustomerID',
how='left')

# Convert TotalAmount to numeric
df = df[pd.to_numeric(df['Total_Amount'], errors='coerce').notna()]
df['Total_Amount'] = pd.to_numeric(df['Total_Amount'])

# Create copy to avoid modifying original
X = df.copy()

# Handling datetime column
if pd.api.types.is_datetime64_any_dtype(X['InvoiceDate']):
    X['InvoiceDate_Year'] = X['InvoiceDate'].dt.year
    X['InvoiceDate_Month'] = X['InvoiceDate'].dt.month
    X['InvoiceDate_Day'] = X['InvoiceDate'].dt.day
    X['InvoiceDate_DayOfWeek'] = X['InvoiceDate'].dt.dayofweek
    X = X.drop('InvoiceDate', axis=1)

if 'YearMonth' in X.columns and
pd.api.types.is_datetime64_any_dtype(X['YearMonth']):
    # Convert YearMonth to a numeric timestamp representation
    X['YearMonth'] = X['YearMonth'].dt.to_period('M').apply(lambda x:
x.to_timestamp().timestamp())

numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Encode categorical columns
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in categorical_cols:
    X[col] = le.fit_transform(X[col].astype(str))

# Preparing features and target
X = X.drop(['CustomerID'], axis=1)
y = X['Total_Amount']
```

```
X = X.drop(['Total_Amount'], axis=1)

# Changed to RandomForestRegressor for continuous target variable
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)

feature_importance = pd.DataFrame({'feature': X.columns, 'importance':
rf.feature_importances_})
feature_importance = feature_importance.sort_values('importance',
ascending=False)
print(feature_importance)
```

```
                 feature    importance
3               Quantity  5.383383e-01
4              UnitPrice  4.616223e-01
1              StockCode  1.102227e-05
2            Description  9.107600e-06
0              InvoiceNo  5.082332e-06
8                    Day  2.172390e-06
15       InvoiceDate_Day  1.940193e-06
10             YearMonth  1.897555e-06
14     InvoiceDate_Month  1.845235e-06
7                  Month  1.799209e-06
16  InvoiceDate_DayOfWeek  1.617316e-06
9              DayOfWeek  1.282470e-06
12               Cluster  7.699384e-07
5                Country  3.915535e-07
11        CountryEncoded  2.928979e-07
13      InvoiceDate_Year  1.126705e-07
6                   Year  7.740572e-08
```

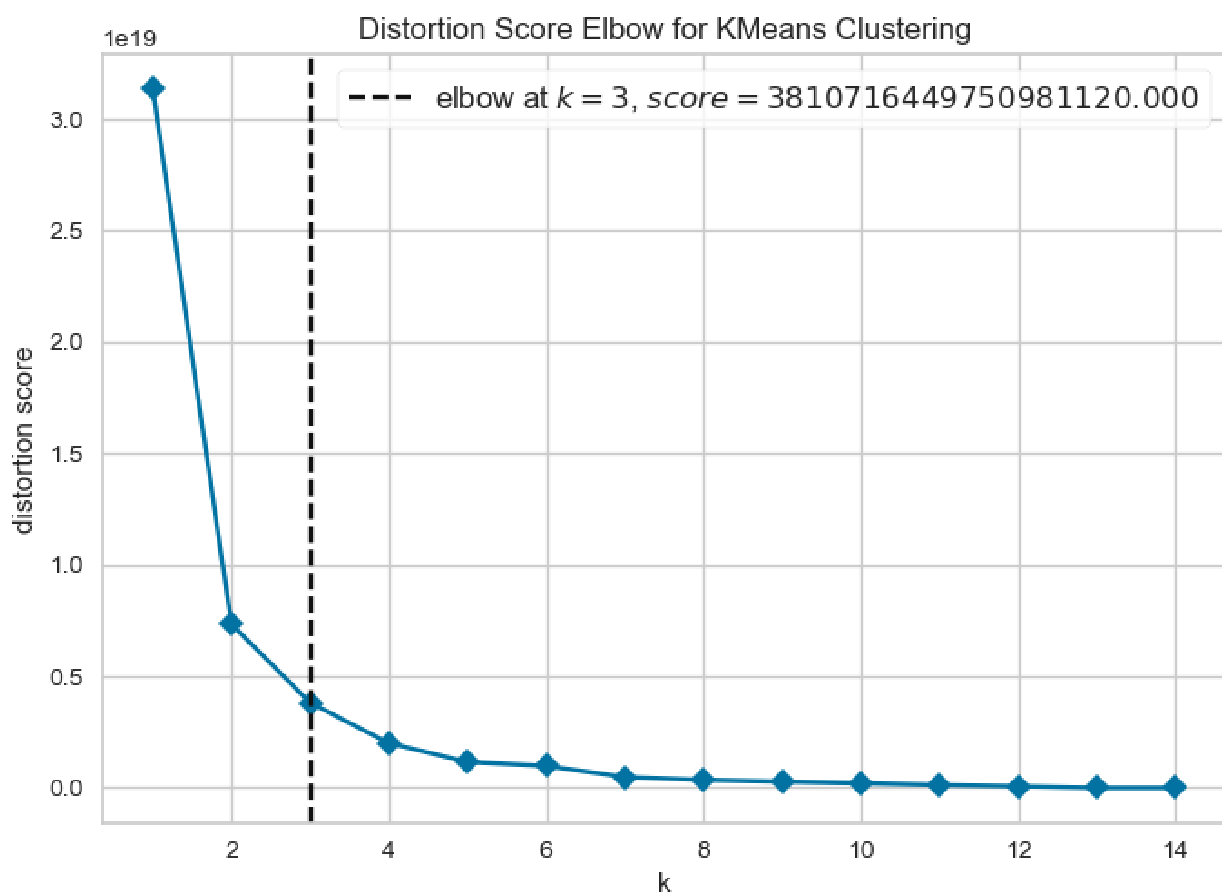
```
numeric_data = df.select_dtypes(include=['number'])

# Drop or fill missing values (if any left)
numeric_data = numeric_data.dropna()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(numeric_data)

pca = PCA(n_components=0.95)  # Preserve 95% of variance
X_pca = pca.fit_transform(X_scaled)

print(f"Original features: {X_scaled.shape[1]}")
print(f"Features after PCA: {X_pca.shape[1]}")
```

```
Original features: 10
Features after PCA: 9
```

```
# chose elbow method to find out the best
SSE = {}
```

```
for k in range(1,15):
  km = KMeans(n_clusters = k, init = 'k-means++', max_iter = 1000)
  km = km.fit(X)
  SSE[k] = km.inertia_

visualizer = KElbowVisualizer(km, k=(1,15), metric='distortion',
timings=False)
visualizer.fit(X)
visualizer.poof()
plt.show()
```

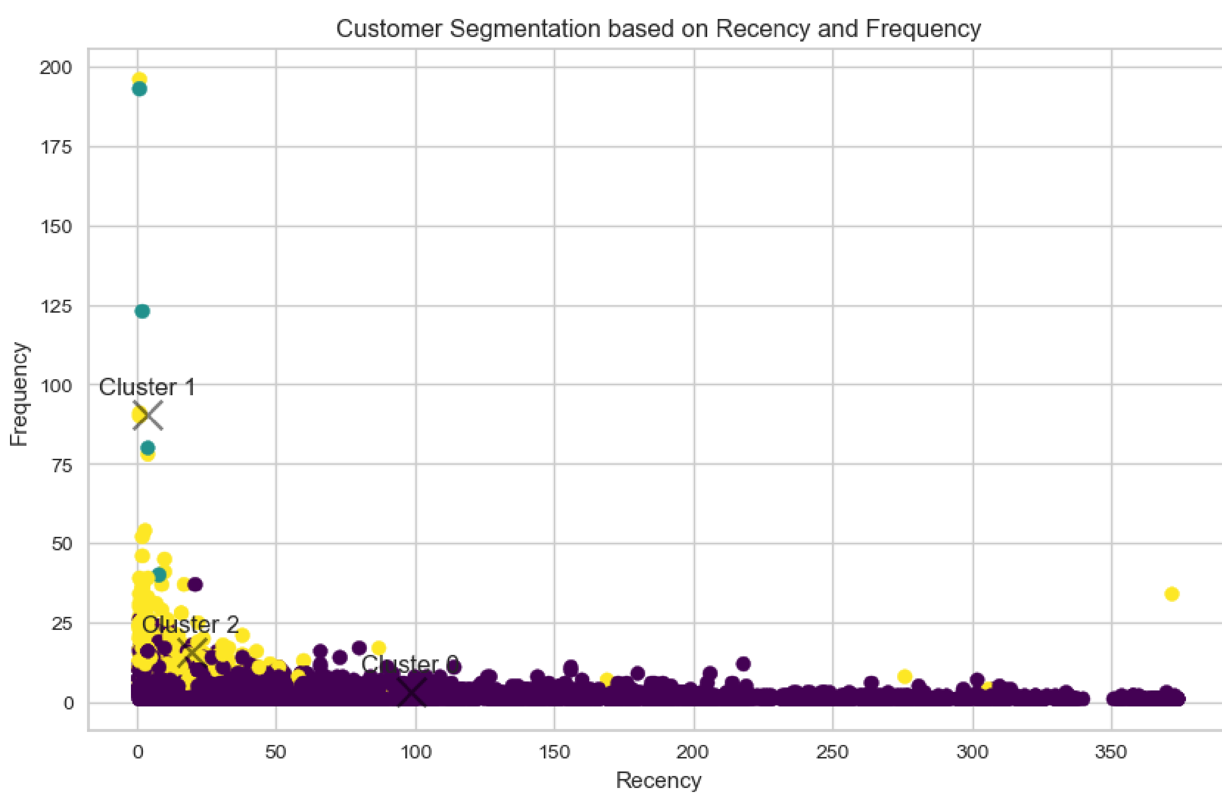
```
/opt/anaconda3/lib/python3.12/site-packages/yellowbrick/base.py:258:
DeprecationWarning:

this method is deprecated, please use show() instead
```




```
print(X.columns)
```

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity',
'UnitPrice',
       'Country', 'Year', 'Month', 'Day', 'DayOfWeek', 'YearMonth',
```

```
       'CountryEncoded', 'Cluster', 'InvoiceDate_Year',
'InvoiceDate_Month',
       'InvoiceDate_Day', 'InvoiceDate_DayOfWeek'],
      dtype='object')

X_rfm = rfm[['Recency', 'Frequency', 'Monetary']].values
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_rfm)
rfm['Cluster'] = kmeans.predict(X_rfm)

plt.figure(figsize=(10, 6))
plt.scatter(rfm['Recency'], rfm['Frequency'], c=rfm['Cluster'], s=50,
cmap='viridis')
plt.title('Customer Segmentation based on Recency and Frequency')
plt.xlabel('Recency')
plt.ylabel('Frequency')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5,
marker='x')
for i, center in enumerate(centers):
    plt.annotate(f'Cluster {i}', (center[0], center[1]),
textcoords="offset points", xytext=(0, 10), ha='center')

plt.show()
```

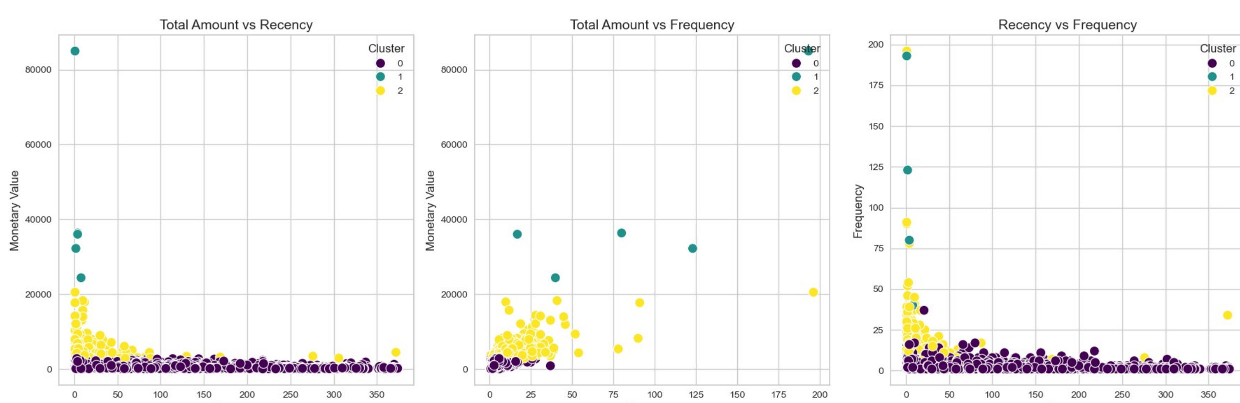


```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_rfm)

rfm['Cluster'] = kmeans.predict(X_rfm)
rfm.head(10)
```

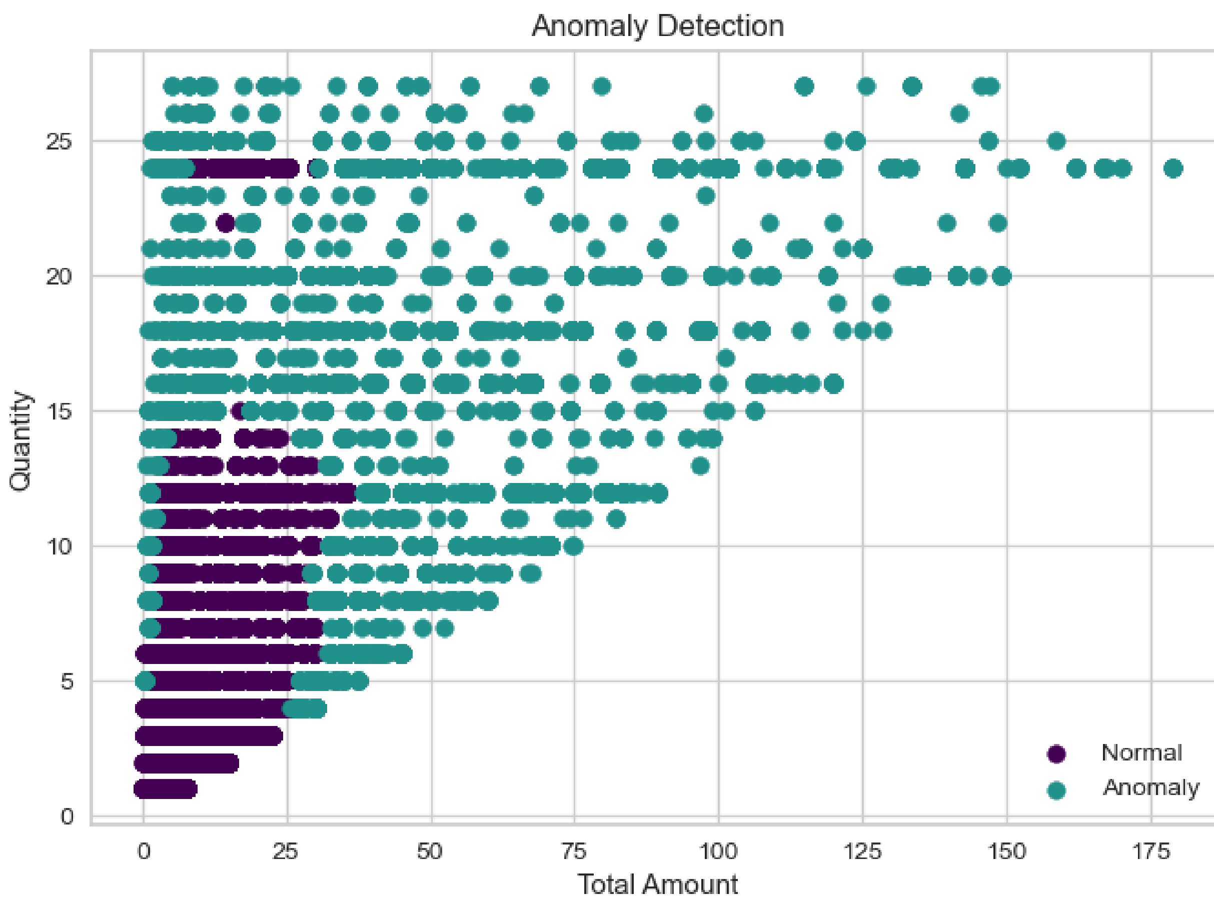
```
   CustomerID  Recency  Frequency  Monetary  R  F  M RFM_Score  \
0       12347        2          7   3314.73  5  5  5       555
1       12348      249          3     90.20  1  3  1       131
2       12349       19          1    999.15  4  1  4       414
3       12350      310          1    294.40  1  1  2       112
4       12352       36          7   1130.94  3  5  4       354
5       12353      204          1     29.30  1  1  1       111
6       12354      232          1    682.69  1  1  4       114
7       12355      214          1    219.00  1  1  2       112
8       12356      246          2   1086.56  1  2  4       124
9       12357       33          1   3315.41  4  1  5       415

              Segment  Recency_log  Frequency_log  Monetary_log
Cluster
0           Champions     0.693147       1.945910      8.106131
2
1       New Customers     5.517453       1.098612      4.502029
0
2  Potential Loyalist     2.944439       0.000000      6.906905
0
3      Need Attention     5.736572       0.000000      5.684939
0
4     Loyal Customers     3.583519       1.945910      7.030804
0
5             At Risk     5.318120       0.000000      3.377588
0
6  Potential Loyalist     5.446737       0.000000      6.526041
0
7      Need Attention     5.365976       0.000000      5.389072
0
8  Potential Loyalist     5.505332       0.693147      6.990772
0
9     Loyal Customers     3.496508       0.000000      8.106337
2
```

```
plt.figure(figsize=(18, 6))
sns.set_style('whitegrid')

#Total Amount vs Recency
plt.subplot(1, 3, 1)
sns.scatterplot(
    x=rfm['Recency'],
    y=rfm['Monetary'],
    hue=rfm['Cluster'],
```

```
    palette='viridis',
    s=100
)
plt.title('Total Amount vs Recency', fontsize=14)
plt.xlabel('Recency', fontsize=12)
plt.ylabel('Monetary Value', fontsize=12)
plt.legend(title='Cluster', loc='upper right')

#Total Amount vs Frequency
plt.subplot(1, 3, 2)
sns.scatterplot(
    x=rfm['Frequency'],
    y=rfm['Monetary'],
    hue=rfm['Cluster'],
    palette='viridis',
    s=100
)
plt.title('Total Amount vs Frequency', fontsize=14)
plt.xlabel('Frequency', fontsize=12)
plt.ylabel('Monetary Value', fontsize=12)
plt.legend(title='Cluster', loc='upper right')

# Total Amount vs Monetary
plt.subplot(1, 3, 3)
sns.scatterplot(
    x=rfm['Recency'],
    y=rfm['Frequency'],
    hue=rfm['Cluster'],
    palette='viridis',
    s=100
)
plt.title('Recency vs Frequency', fontsize=14)
plt.xlabel('Recency', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.legend(title='Cluster', loc='upper right')
plt.tight_layout()
plt.show()
```



```
# Preparing data for anomaly detection
anomaly_data = df[['Total_Amount', 'Quantity']]

# Applied Isolation Forest
iso_forest = IsolationForest(contamination=0.1, random_state=42)
df['Anomaly'] = iso_forest.fit_predict(anomaly_data)

plt.scatter(df[df['Anomaly']==1]['Total_Amount'], df[df['Anomaly']==1]
['Quantity'], c='#440154', label='Normal')
plt.scatter(df[df['Anomaly']==-1]['Total_Amount'], df[df['Anomaly']==-
1]['Quantity'], c='#21918c', label='Anomaly')
plt.xlabel('Total Amount')
plt.ylabel('Quantity')
plt.title('Anomaly Detection')
plt.legend()
plt.show()
```




```
# Define features (X) and target (y)
X = df.drop(columns=['Cluster', 'CustomerID'])
y = df['Cluster']
```

```
# Identify non-numeric columns in our x data frame
non_numeric_cols = X.select_dtypes(include=['object']).columns
print("Non-Numeric Columns:", non_numeric_cols)

le = LabelEncoder()
for col in non_numeric_cols:
    X[col] = le.fit_transform(X[col].astype(str))
```

```
Non-Numeric Columns: Index(['InvoiceNo', 'StockCode', 'Description',
'Country'], dtype='object')
```

```
# Convert datetime columns to numeric components
if 'InvoiceDate' in X.columns:
    X['InvoiceDate_Year'] = X['InvoiceDate'].dt.year
    X['InvoiceDate_Month'] = X['InvoiceDate'].dt.month
    X['InvoiceDate_Day'] = X['InvoiceDate'].dt.day
    X['InvoiceDate_DayOfWeek'] = X['InvoiceDate'].dt.dayofweek
    X = X.drop(columns=['InvoiceDate'])

if 'YearMonth' in X.columns:
    X['YearMonth'] = X['YearMonth'].dt.to_period('M').apply(lambda x:
x.to_timestamp().timestamp())

# Spliting data into training and testing section
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)

# Logistic Regression
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train, y_train)
y_pred_logistic = logistic_model.predict(X_test)

print("Logistic Regression Evaluation:")
print(classification_report(y_test, y_pred_logistic))
print("Accuracy:", accuracy_score(y_test, y_pred_logistic))

# Decision Tree
dt_model = DecisionTreeClassifier(max_depth=4, random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

print("\nDecision Tree Evaluation:")
print(classification_report(y_test, y_pred_dt))
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
```

```
Logistic Regression Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      9516
           1       0.56      1.00      0.72     56293
           2       0.00      0.00      0.00     34162
```

```
    accuracy                           0.56     99971
   macro avg       0.19      0.33      0.24     99971
weighted avg       0.32      0.56      0.41     99971

Accuracy: 0.5630932970561463

/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/
_classification.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.

/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classific
ation.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.

/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classific
ation.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.



Decision Tree Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      9516
           1       0.57      0.99      0.72     56293
           2       0.63      0.02      0.04     34162

    accuracy                           0.57     99971
   macro avg       0.40      0.34      0.25     99971
weighted avg       0.53      0.57      0.42     99971

Accuracy: 0.5671044602934852

/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/
_classification.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.

/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classific
```

```
ation.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.

/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classific
ation.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("\nRandom Forest Evaluation:")
print(classification_report(y_test, y_pred_rf))
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

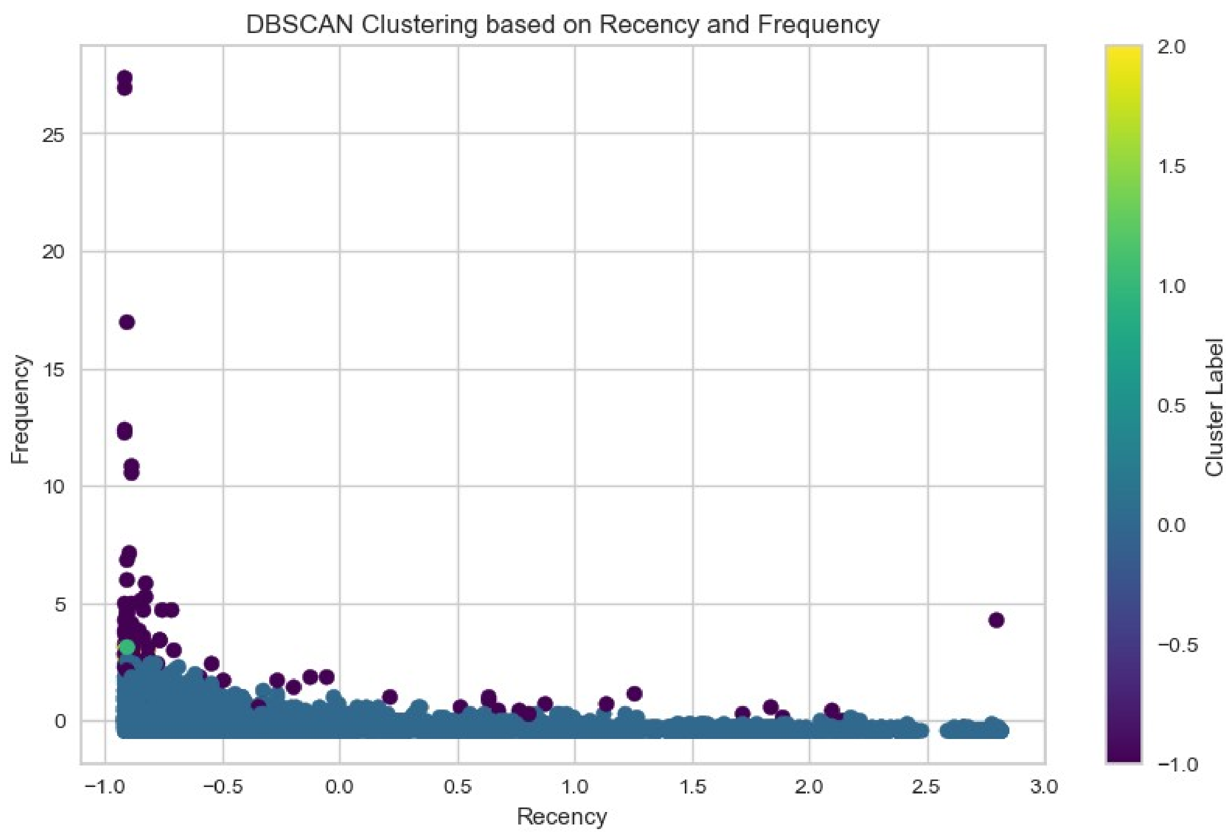
```
Random Forest Evaluation:
              precision    recall  f1-score   support

           0       0.79      0.54      0.64      9516
           1       0.83      0.90      0.87     56293
           2       0.81      0.76      0.78     34162

    accuracy                           0.82     99971
   macro avg       0.81      0.73      0.76     99971
weighted avg       0.82      0.82      0.82     99971

Accuracy: 0.8193176021046104
```

```
dbscan = DBSCAN(eps=.5, min_samples=5)
dbscan.fit(X_rfm)
```

```
DBSCAN()
```

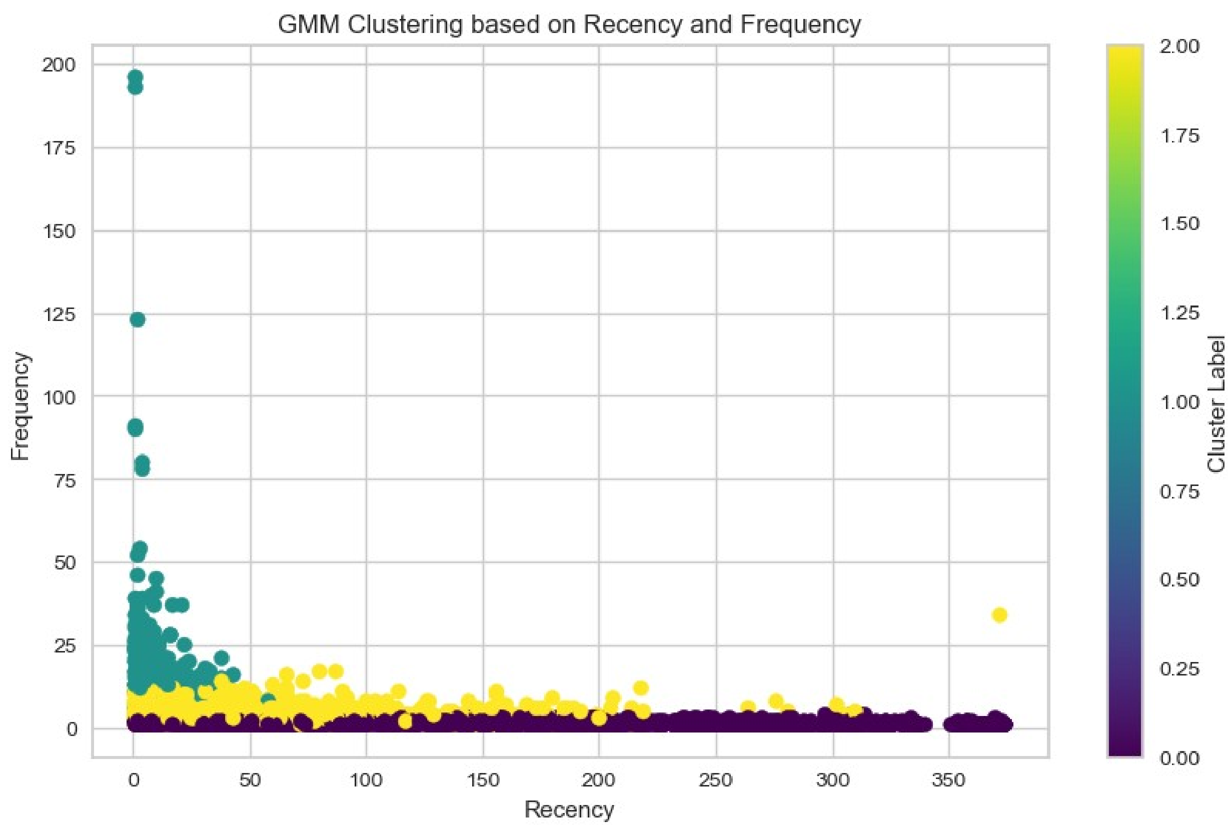
```
scaler = StandardScaler()
X_rfm_scaled = scaler.fit_transform(X_rfm)

from sklearn.neighbors import NearestNeighbors

neighbors = NearestNeighbors(n_neighbors=4)
neighbors.fit(X_rfm_scaled)
distances, indices = neighbors.kneighbors(X_rfm_scaled)
```

```
distances = np.sort(distances[:, 3])
dbscan = DBSCAN(eps=0.3, min_samples=5)
dbscan.fit(X_rfm_scaled)
rfm['DBSCAN_Cluster'] = dbscan.labels_

plt.figure(figsize=(10, 6))
plt.scatter(X_rfm_scaled[:, 0], X_rfm_scaled[:, 1],
c=rfm['DBSCAN_Cluster'], cmap='viridis', s=50)
plt.title('DBSCAN Clustering based on Recency and Frequency')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.colorbar(label='Cluster Label')
plt.show()
print(f"Number of clusters found: {len(set(dbscan.labels_)) - (1 if -1
in dbscan.labels_ else 0)}")
print(f"Number of noise points: {list(dbscan.labels_).count(-1)}")
```




```
Number of clusters found: 3
Number of noise points: 112

# Apply GMM with 3 components
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(X_rfm)
```

```
rfm['GMM_Cluster'] = gmm.predict(X_rfm)

plt.figure(figsize=(10, 6))
plt.scatter(X_rfm[:, 0], X_rfm[:, 1], c=rfm['GMM_Cluster'],
cmap='viridis', s=50)
plt.title('GMM Clustering based on Recency and Frequency')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.colorbar(label='Cluster Label')
plt.show()

print("Cluster means (centroids):")
print(gmm.means_)
```



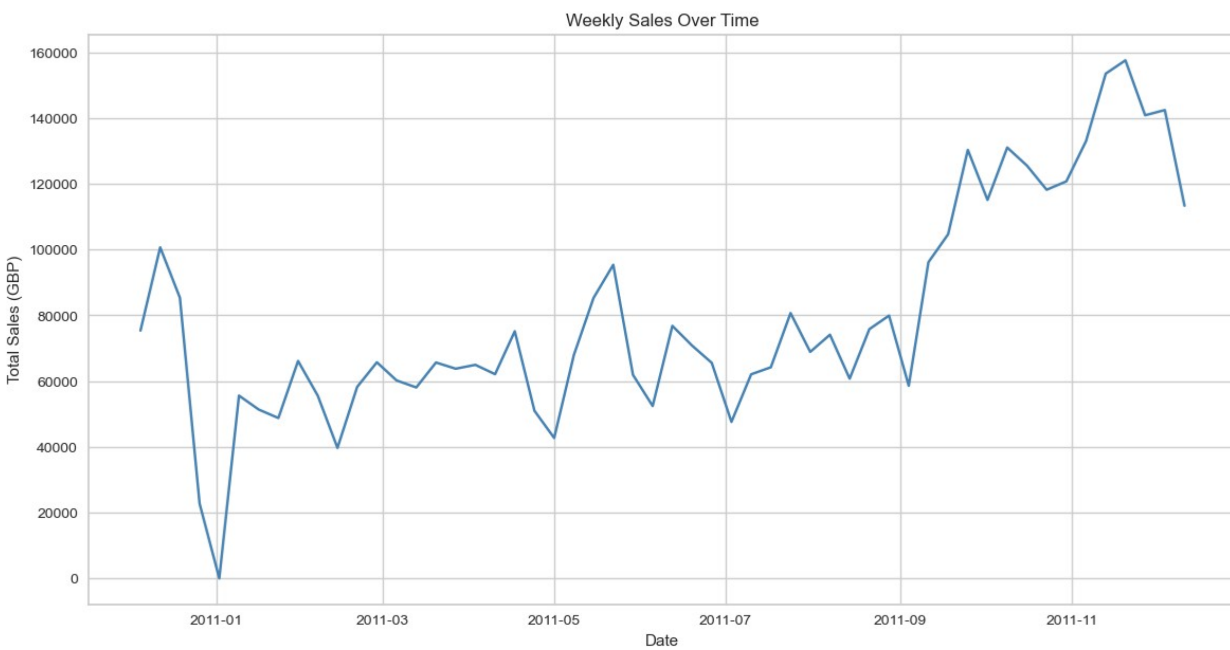

```
Cluster means (centroids):
[[1.40619706e+02 1.44426907e+00 2.73441777e+02]
 [1.11240150e+01 1.78486201e+01 5.23727449e+03]
 [4.52923861e+01 4.75301270e+00 1.21153821e+03]]
```

```
df = df[df['Quantity'] > 0]
df_daily = df.groupby('InvoiceDate')['Total_Amount'].sum()

df_weekly = df_daily.resample('W').sum()
```

```
print(df_weekly.head())
```

```
InvoiceDate
2010-12-05     75357.92
2010-12-12    100657.99
2010-12-19     85348.52
2010-12-26     22757.72
2011-01-02         0.00
Freq: W-SUN, Name: Total_Amount, dtype: float64
```

```
plt.figure(figsize=(14, 7))
plt.plot(df_weekly.index, df_weekly, label='Weekly Sales',
color='steelblue')
plt.title('Weekly Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales (GBP)')
plt.grid(True)
plt.show()
```

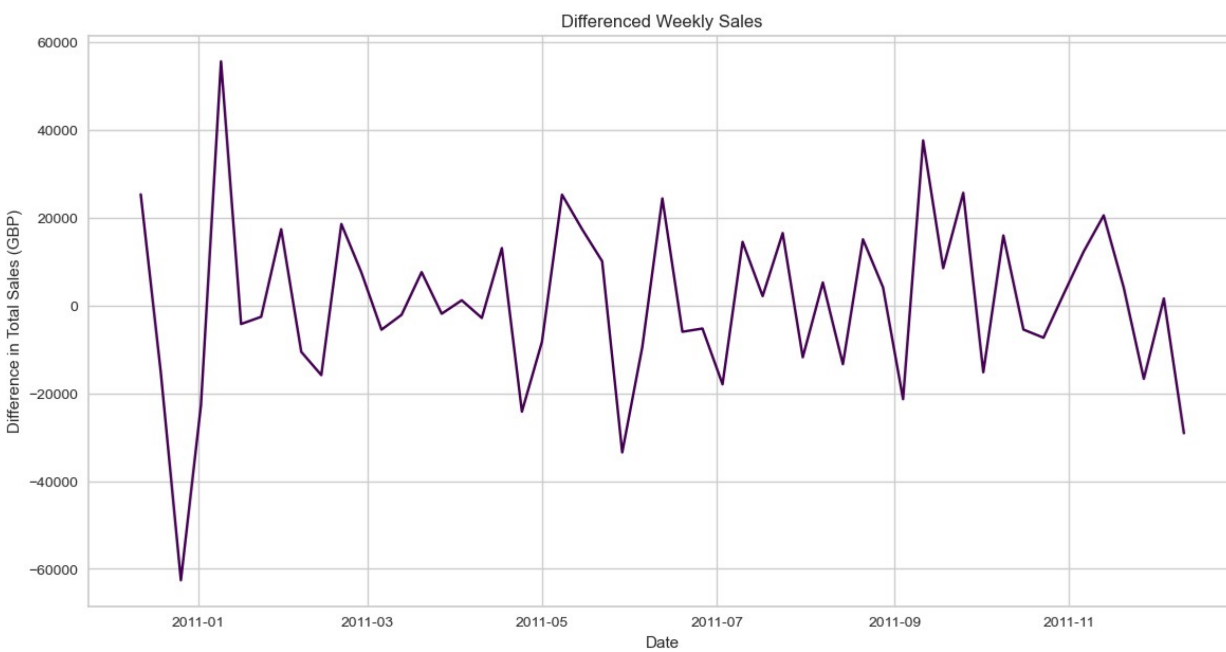



```
result = adfuller(df_weekly)
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', result[4])

if result[1] <= 0.05:
    print("The time series is stationary.")
else:
```

```
    print("The time series is not stationary. Differencing may be required.")
```

```
ADF Statistic: -2.0241682151542157
p-value: 0.2760777367100903
Critical Values: {'1%': -3.560242358792829, '5%': -2.9178502070837, '10%': -2.5967964150943397}
The time series is not stationary. Differencing may be required.
```

```
if result[1] > 0.05:
    df_weekly_diff = df_weekly.diff().dropna()
    plt.figure(figsize=(14, 7))
    plt.plot(df_weekly_diff.index, df_weekly_diff, label='Differenced Series', color='#440154')
    plt.title('Differenced Weekly Sales')
    plt.xlabel('Date')
    plt.ylabel('Difference in Total Sales (GBP)')
    plt.grid(True)
    plt.show()
else:
    df_weekly_diff = df_weekly
```

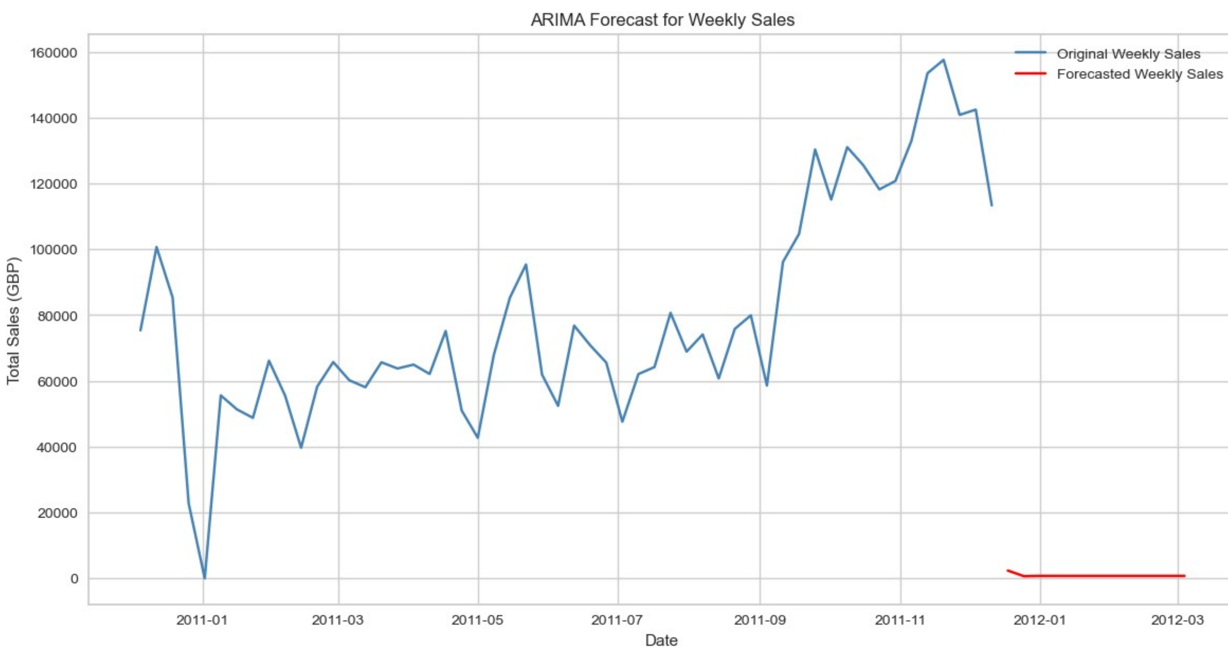



```
model = ARIMA(df_weekly_diff, order=(1, 1, 1))  # Adjust (p, d, q)
model_fit = model.fit()

print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
```

```
========
Dep. Variable:           Total_Amount   No. Observations:
53
Model:                 ARIMA(1, 1, 1)   Log Likelihood
-588.626
Date:                Sat, 07 Dec 2024   AIC
1183.252
Time:                        16:02:26   BIC
1189.106
Sample:                    12-12-2010   HQIC
1185.496
                         - 12-11-2011

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -0.0546      0.162     -0.338      0.735      -0.372
0.262
ma.L1         -0.9989      0.121     -8.281      0.000      -1.235
-0.763
sigma2      3.205e+08   3.76e-10   8.52e+17      0.000    3.21e+08
3.21e+08
======================================================================
=============
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
8.32
Prob(Q):                              0.99   Prob(JB):
0.02
Heteroskedasticity (H):               0.54   Skew:
0.10
Prob(H) (two-sided):                  0.21   Kurtosis:
4.95
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number 2.72e+32. Standard errors may be unstable.

/opt/anaconda3/lib/python3.12/site-packages/statsmodels/tsa/
statespace/sarimax.py:978: UserWarning:

Non-invertible starting MA parameters found. Using zeros as starting
```

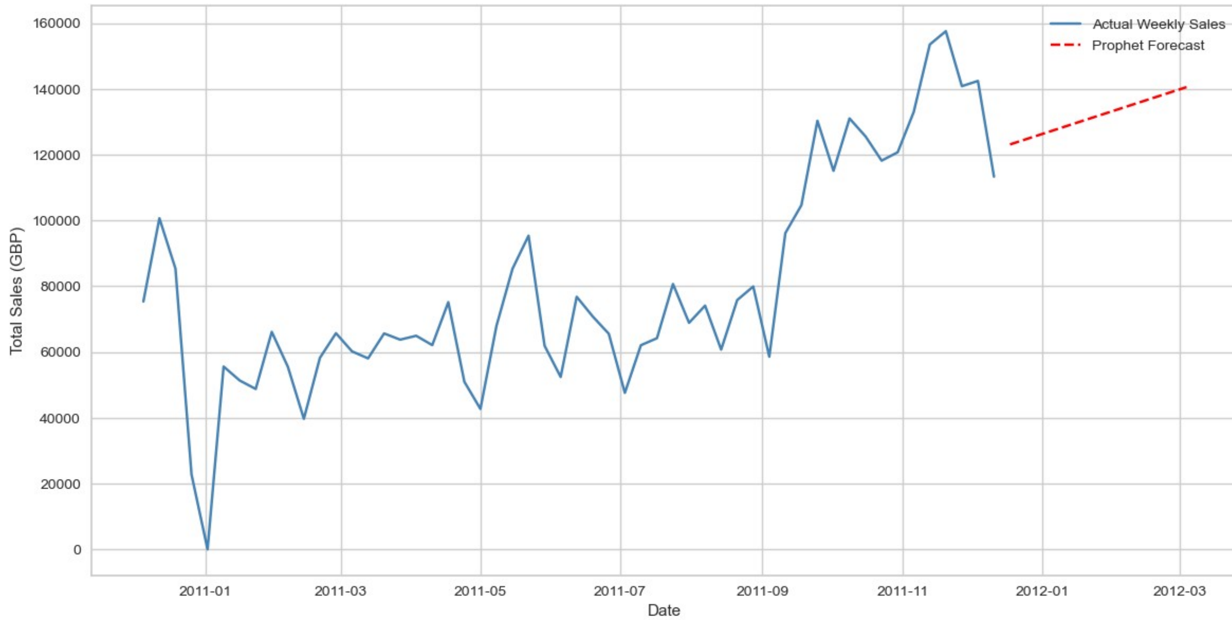
```
parameters.

# Predicting the next 12 weeks with Arima Forecast
forecast = model_fit.forecast(steps=12)

forecast_index = pd.date_range(start=df_weekly.index[-1], periods=13,
freq='W')[1:]
forecast_series = pd.Series(forecast, index=forecast_index)

plt.figure(figsize=(14, 7))
plt.plot(df_weekly.index, df_weekly, label='Original Weekly Sales',
color='steelblue')
plt.plot(forecast_series.index, forecast_series, label='Forecasted
Weekly Sales', color='red')
plt.title('ARIMA Forecast for Weekly Sales')
plt.xlabel('Date')
plt.ylabel('Total Sales (GBP)')
plt.legend()
plt.grid(True)
plt.show()
```




```
from prophet import Prophet
#Predicting the next 12 weeks Prophet model and forecast
prophet_data = df_weekly.reset_index().rename(columns={'InvoiceDate':
'ds', 'Total_Amount': 'y'})
prophet_model = Prophet()
prophet_model.fit(prophet_data)
future = prophet_model.make_future_dataframe(periods=12, freq='W')
```

```
prophet_forecast = prophet_model.predict(future)
prophet_forecast_series = prophet_forecast.set_index('ds')['yhat'][-
12:]
plt.figure(figsize=(14, 7))
plt.plot(df_weekly.index, df_weekly, label='Actual Weekly Sales',
color='steelblue')
plt.plot(prophet_forecast_series.index, prophet_forecast_series,
label='Prophet Forecast', color='red', linestyle='--')

plt.xlabel('Date')
plt.ylabel('Total Sales (GBP)')
plt.legend()
plt.grid(True)
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/holidays/deprecations/
v1_incompatibility.py:40: FutureIncompatibilityWarning:



This is a future version incompatibility warning from Holidays v0.62
to inform you about an upcoming change in our API versioning strategy
that may affect your
project's dependencies. Starting from version 1.0 onwards, we will be
following a loose form of
Semantic Versioning (SemVer, https://semver.org) to provide clearer
communication regarding any
potential breaking changes.

This means that while we strive to maintain backward compatibility,
there might be occasional
updates that introduce breaking changes to our API. To ensure the
stability of your projects,
we highly recommend pinning the version of our API that you rely on.
You can pin your current
holidays v0.x dependency (e.g., holidays==0.62) or limit it (e.g.,
holidays<1.0) in order to
avoid potentially unwanted upgrade to the version 1.0 when it's
released (ETA 2025Q1-Q2).

If you have any questions or concerns regarding this change, please
don't hesitate to reach out
to us via https://github.com/vacanza/holidays/discussions/1800.


16:02:27 - cmdstanpy - INFO - Chain [1] start processing
16:02:27 - cmdstanpy - INFO - Chain [1] done processing
```




```
df = df[df['Quantity'] > 0]

basket = df.pivot_table(index='InvoiceNo', columns='Description',
values='Quantity', aggfunc='sum', fill_value=0)

basket = basket > 0

print(basket.head(2))

Description   4 PURPLE FLOCK DINNER CANDLES   50'S CHRISTMAS GIFT BAG
LARGE  \
InvoiceNo

536365                                False
False
536366                                False
False

Description   DOLLY GIRL BEAKER   I LOVE LONDON MINI BACKPACK  \
InvoiceNo
536365                     False                        False
536366                     False                        False

Description   I LOVE LONDON MINI RUCKSACK   OVAL WALL MIRROR DIAMANTE
\
InvoiceNo

536365                              False                        False

536366                              False                        False
```

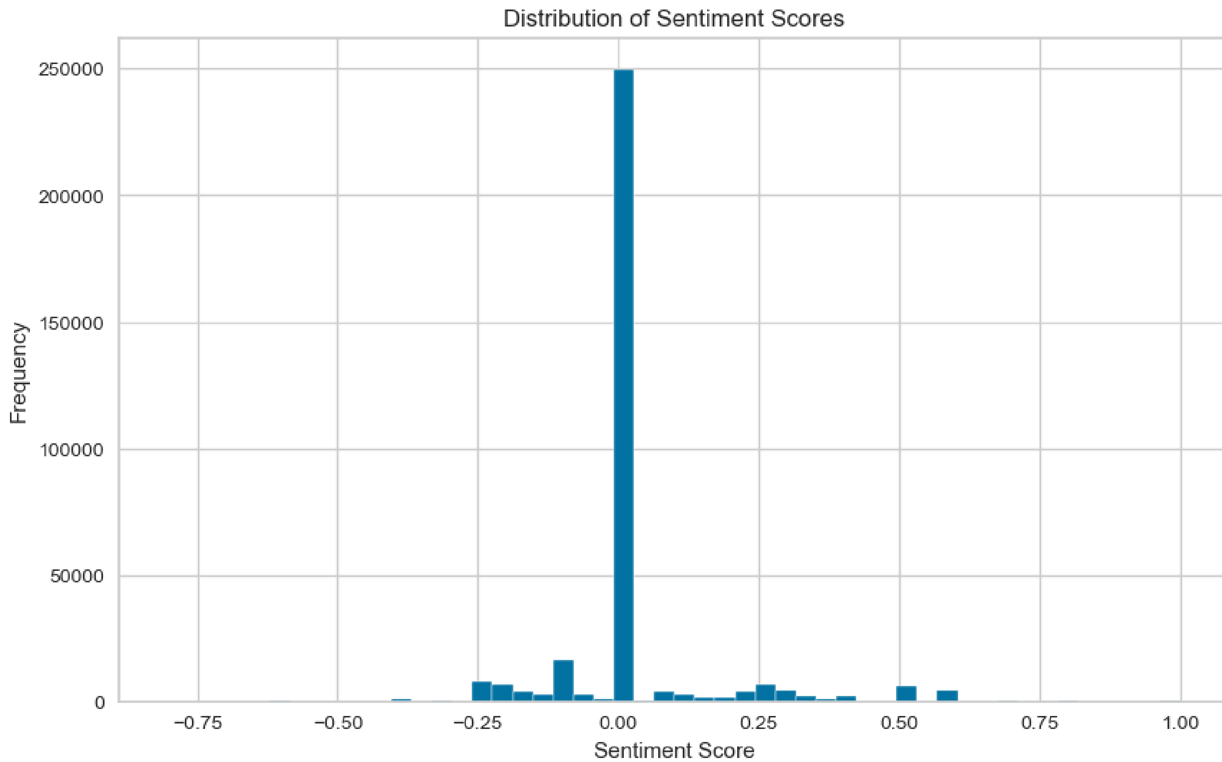
```
Description   RED SPOT GIFT BAG LARGE   SET 2 TEA TOWELS I LOVE LONDON
\
InvoiceNo

536365                          False
False
536366                          False
False

Description   TRELLIS COAT RACK  10 COLOUR SPACEBOY PEN  ...  \
InvoiceNo                                                ...
536365                    False                   False  ...
536366                    False                   False  ...

Description  ZINC PLANT POT HOLDER  ZINC STAR T-LIGHT HOLDER   \
InvoiceNo
536365                       False                      False
536366                       False                      False

Description  ZINC SWEETHEART SOAP DISH  ZINC SWEETHEART WIRE LETTER
RACK  \
InvoiceNo

536365                           False
False
536366                           False
False

Description  ZINC T-LIGHT HOLDER STAR LARGE  ZINC T-LIGHT HOLDER STARS
LARGE  \
InvoiceNo

536365                                False
False
536366                                False
False

Description  ZINC T-LIGHT HOLDER STARS SMALL  \
InvoiceNo
536365                                 False
536366                                 False

Description  ZINC WILLIE WINKIE  CANDLE STICK  ZINC WIRE KITCHEN
ORGANISER  \
InvoiceNo

536365                                  False
False
536366                                  False
```

```
False

Description  ZINC WIRE SWEETHEART LETTER TRAY
InvoiceNo                                    
536365                                  False
536366                                  False

[2 rows x 3575 columns]

# Removing items purchased fewer than 50 times
basket = basket.loc[:, (basket.sum(axis=0) >= 50)]
# Use a random sample of 10,000 invoices for testing
basket = basket.sample(n=10000, random_state=42)

# Find itemsets and calculate the number of itemsets
frequent_itemsets = apriori(basket, min_support=0.01,
use_colnames=True)
num_itemsets = frequent_itemsets.shape[0]

rules = association_rules(frequent_itemsets,
num_itemsets=num_itemsets, metric='confidence', min_threshold=0.7)

print("Association Rules:")
print(rules)

Association Rules:
                                          antecedents  \
0                         (BAKING SET SPACEBOY DESIGN)   
1                                 (KITCHEN METAL SIGN)   
2                                  (TOILET METAL SIGN)   
3                        (PINK HAPPY BIRTHDAY BUNTING)   
4                    (CANDLEHOLDER PINK HANGING HEART)   
..                                                ...   
61  (REGENCY TEA PLATE ROSES , REGENCY TEA PLATE G...   
62  (REGENCY TEA PLATE ROSES , REGENCY TEA PLATE P...   
63  (REGENCY TEA PLATE GREEN , REGENCY TEA PLATE P...   
64                            (REGENCY TEA PLATE PINK)   
65  (WOODEN FRAME ANTIQUE WHITE , WHITE HANGING HE...   

                                          consequents  antecedent
support  \
0                      (BAKING SET 9 PIECE RETROSPOT )
0.0242   
1                                (BATHROOM METAL SIGN)
0.0128   
2                                (BATHROOM METAL SIGN)
0.0170   
3                        (BLUE HAPPY BIRTHDAY BUNTING)
0.0196   
4                 (WHITE HANGING HEART T-LIGHT HOLDER)
```

```
0.0186
..                                                ...
...
61                           (REGENCY TEA PLATE PINK)
0.0126
62                         (REGENCY TEA PLATE GREEN )
0.0106
63                         (REGENCY TEA PLATE ROSES )
0.0111
64  (REGENCY TEA PLATE ROSES , REGENCY TEA PLATE G...
0.0121
65                  (WOODEN PICTURE FRAME WHITE FINISH)
0.0141

    consequent support  support  confidence       lift
representativity  \
0                0.0494   0.0174    0.719008  14.554823
1.0
1                0.0206   0.0100    0.781250  37.924757
1.0
2                0.0206   0.0122    0.717647  34.837236
1.0
3                0.0206   0.0139    0.709184  34.426392
1.0
4                0.0987   0.0136    0.731183   7.408134
1.0
..                  ...      ...         ...        ...               .
..
61               0.0121   0.0102    0.809524  66.902794
1.0
62               0.0147   0.0102    0.962264  65.460146
1.0
63               0.0164   0.0102    0.918919  56.031641
1.0
64               0.0126   0.0102    0.842975  66.902794
1.0
65               0.0477   0.0100    0.709220  14.868341
1.0

    leverage  conviction  zhangs_metric   jaccard  certainty
kulczynski
0   0.016205    3.383018       0.954391  0.309609   0.704406
0.535617
1   0.009736    4.477257       0.986256  0.427350   0.776649
0.633343
2   0.011850    3.468708       0.988093  0.480315   0.711708
0.654940
3   0.013496    3.367761       0.990364  0.528517   0.703067
0.691970
```

```
4   0.011764    3.352836       0.881407  0.131148   0.701745
0.434487
..       ...         ...              ...       ...         ...          .
..
61  0.010048    5.186475       0.997623  0.703448   0.807191
0.826250
62  0.010044   26.110450       0.995273  0.675497   0.961701
0.828071
63  0.010018   12.131067       0.993177  0.589595   0.917567
0.770435
64  0.010048    6.288179       0.997118  0.703448   0.840971
0.826250
65  0.009327    3.274983       0.946083  0.193050   0.694655
0.459432

[66 rows x 14 columns]
```

```python
from textblob import TextBlob

def get_sentiment(text):
    return TextBlob(text).sentiment.polarity

df['sentiment'] = df['Description'].apply(get_sentiment)

plt.figure(figsize=(10, 6))
plt.hist(df['sentiment'], bins=50)
plt.title('Distribution of Sentiment Scores')
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.show()
```


Distribution of Sentiment Scores
Frequency
250000
200000
150000
100000
50000
0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Sentiment Score