Data Mining Final Project Report

Customer Behavior Analysis and Prediction on Online Retail Dataset

Name: Khalamkhan Aiman, 21B031335

Date: 08.12.2024

Institution/Organization Name: Kazakh-British Technical University

2. Executive Summary

This project aimed to analyze customer behavior using advanced data mining and machine learning techniques. The objectives included identifying patterns, predicting customer actions, and segmenting customers for targeted marketing strategies. Key outcomes include improved classification accuracy, insightful clustering results, and actionable association rules. Recommendations focus on leveraging customer segmentation and association rules for personalized marketing campaigns.

3. Table of Contents

4. Introduction

Background

Optimize the store's operations and improve customer satisfaction. Store optimization could include things like inventory management. Given the fields we have to work with this will likely revolve around temporal purchase trends and better understanding outliers. Customer satisfaction might include a deeper understanding of return trends.

Project Goals

1. Analyze customer purchasing patterns and sales trend.
2. Segment customers based on their behavior.
3. Predict future customer actions.
4. Generate actionable insights for business strategy.

Scope

This project focuses on analyzing a transactional dataset from an online retail store. It includes classification, clustering, association rule mining, and time series forecasting. Limitations include the absence of external demographic data and potential biases in the dataset.

5.Data Description

The dataset used in this project is from an online retail store, containing transaction-level data

Just over 541,000 rows of sales data with the following fields:

1. InvoiceNo — Unique Identifier under which multiple items can be purchased
2. StockCode — Unique Identifier for a product

3. Description — Description of the product
4. Quantity — Number of product purchased
5. InvoiceDate — Date of invoice when purchase was made noted with time of day
6. UnitPrice — price per 1 quantity of product
7. CustomerId — Unique Identifier for a customer
8. Country — Country where product was purchased from

Preprocessing involved handling missing values, converting data types, and aggregating transactions into meaningful structures.

6.Data Preprocessing

First of all, I will import all the necessary libraries that we will use throughout the project. This generally includes libraries for data manipulation, data visualization, and others based on the specific needs of the project. Afterward, I am going to gain a thorough understanding of the dataset before proceeding to the data cleaning and transformation stages.

In [1]:
```
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
```

In [2]:
```
df = pd.read_csv('OnlineRetail.csv', encoding="ISO-8859-1")
df.head()
```

```
In [3]: df.shape
```

```
Out[3]: (541909, 8)
```

```
In [4]: 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
```

```
In [5]: df.isnull().sum()
```

```
Out[5]: InvoiceNo           0
        StockCode           0
        Description      1454
        Quantity            0
        InvoiceDate         0
        UnitPrice           0
        CustomerID     135080
        Country             0
        dtype: int64
```

When you look through, it is obvious that there are some missing values in the Description and CustomerID columns that must be replaced. The InvoiceDate column is already in date and time format, which will be very helpful during further time series analysis. We could also see that one customer can have many transactions since the CustomerID is repeated in the first rows. The next few steps will focus on data cleaning, dealing with missing values, and correcting any wrong data, with the creation of new features, which may help to complete this project's goals.

```
In [6]: df = df.dropna(subset=['CustomerID']) #Here removing rows with missing CustomerID
        df['Description'] = df['Description'].fillna('Unknown') #Here filling missing description with Unknown
```

```
In [7]: 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
```

```
In [8]: #negative values handiling
        df = df[df['UnitPrice']>0]
        df = df[df['Quantity']>0]
```

```
In [9]: 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
```

This step involves a thorough cleaning and changing process in order to enhance the dataset: fixing missing values, dropping duplicate entries, correcting mistakes in product codes and descriptions, and other important changes that need to be in place for the data to be ready for detailed analysis and modeling.

```
In [10]: # 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)
```

```
In [11]: # How many duplicate rows are there?
         df.duplicated().sum()

Out[11]: 5192
```

```
In [12]: df.drop_duplicates(inplace=True)
```

```
In [13]: 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')
```

Outliers can tell us more about a data set. In this data it’s clear that a majority of sales outliers, calculated as *Quantity * UnitPrice,* are represented by large negative values (order returns).

In [14]:
```python
# 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
```

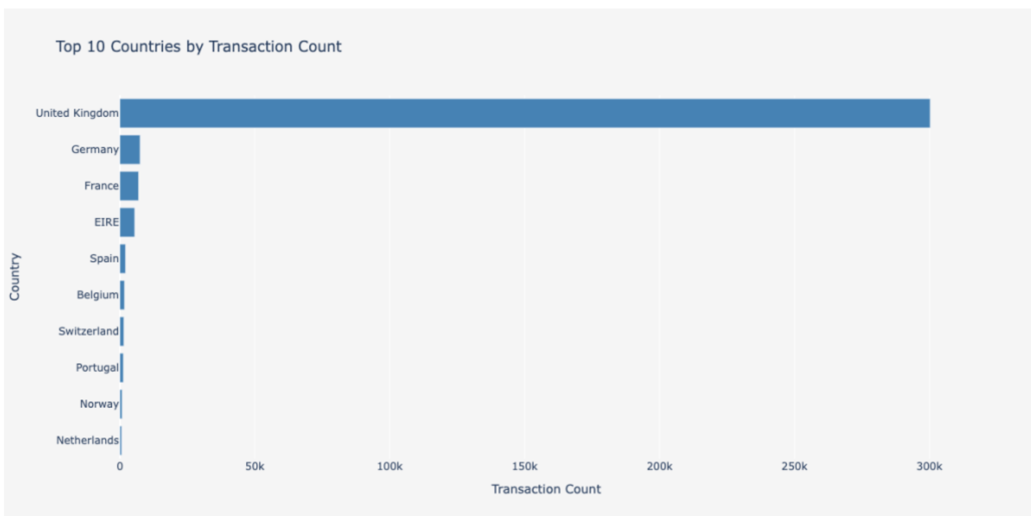
7. Data Exploration and Visualization

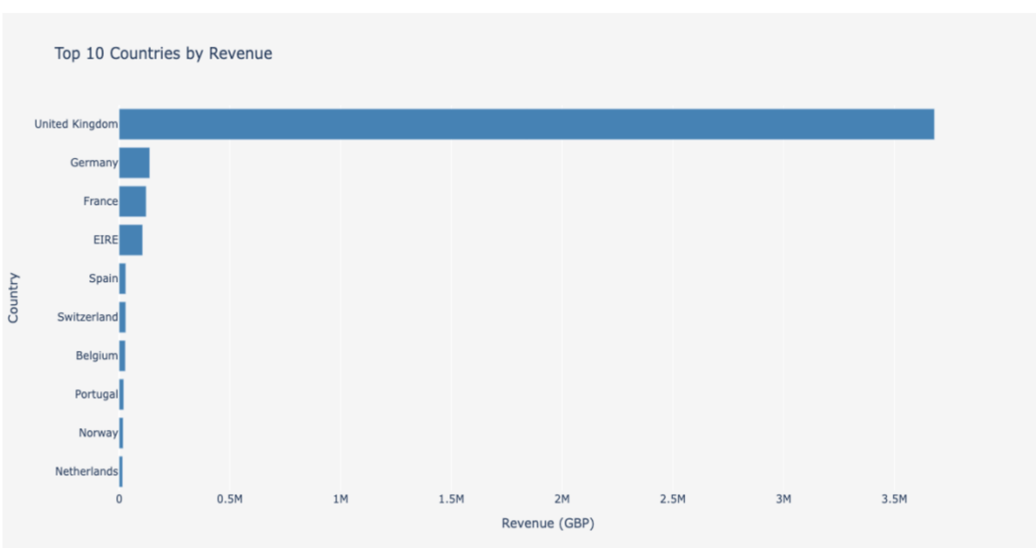
Exploratory data analysis revealed:

The graphs below show a large difference between countries in terms of total sales.

The United Kingdom accounts for the vast majority of total sales and is far ahead of other countries.

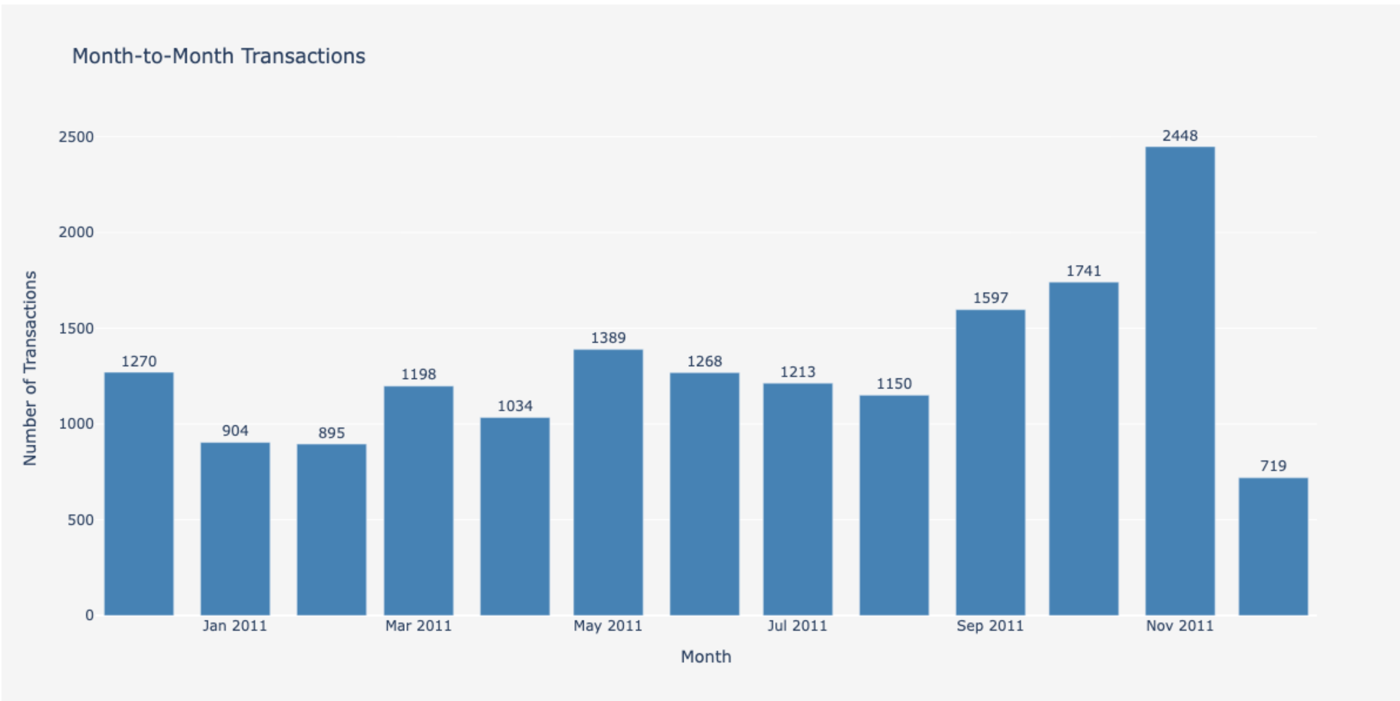
Other countries include the Netherlands, Ireland, and Germany, but their sales figures are considerably lower than the United Kingdom.





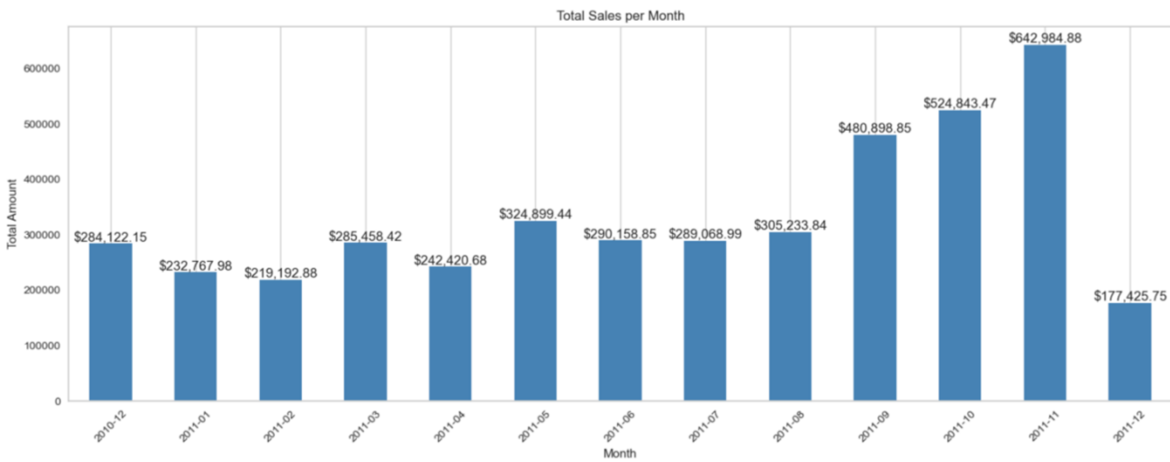
Among the top 10 countries, European countries are heavily represented. The dominance of the United Kingdom in total sales indicates that the dataset is concentrated in this region.

Therefore, analyses and strategic decisions can be made with a focus on the United Kingdom as a priority.



As you can see, November has the highest number of transactions among all the months of the year.

```
In [18]: 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()
```



It can be seen that the overall sales of the platform increased in the second half of the year. And hit-high in November.

8. Feature Selection and Dimensionality Reduction

Under customer analysis, I looked at **Customer segmentation** means grouping customers based on their purchasing behaviour such as RFM (Recency, Frequency, Monetary) analysis, which categorizes customers based on how recently they made a purchase, how often they buy, and how much they spend.

The simplest explanation of customer segmentation;

1. Recency: Calculates the number of days since each customer's last purchase.

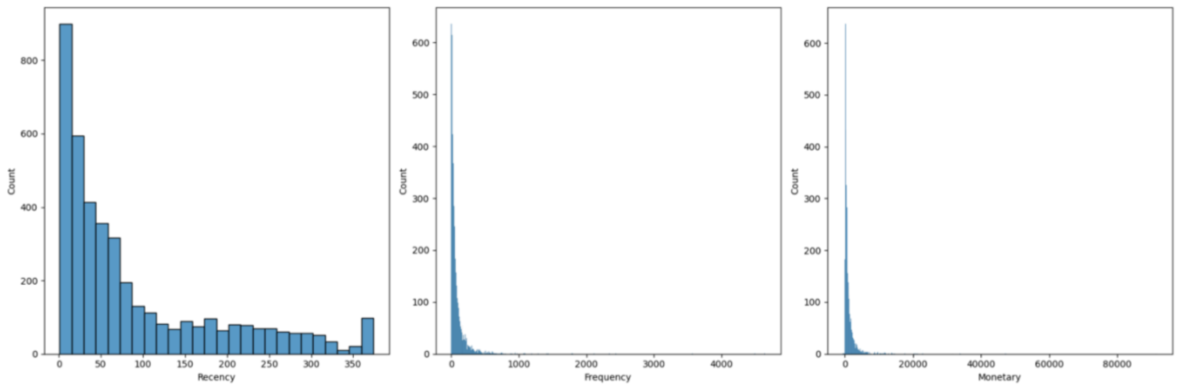
2. Frequency: Counts the number of orders each customer has made.

3. Monetary: Calculates the total amount spent by each customer.

Together, these metrics help in understanding a customer's buying behavior and preferences, which is pivotal in personalizing marketing strategies and creating a recommendation system.

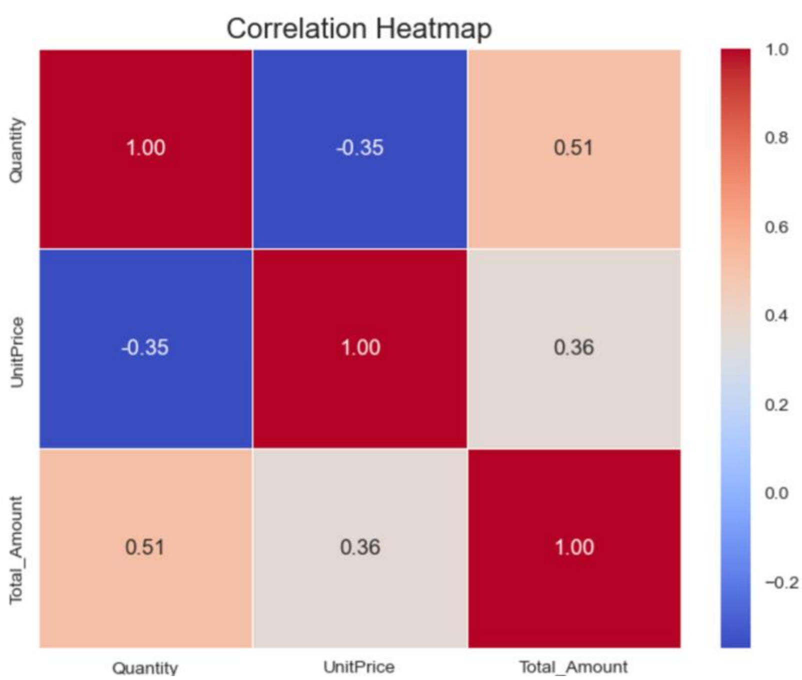
In [28]:

```
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()
```



In [29]:

```
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()
```

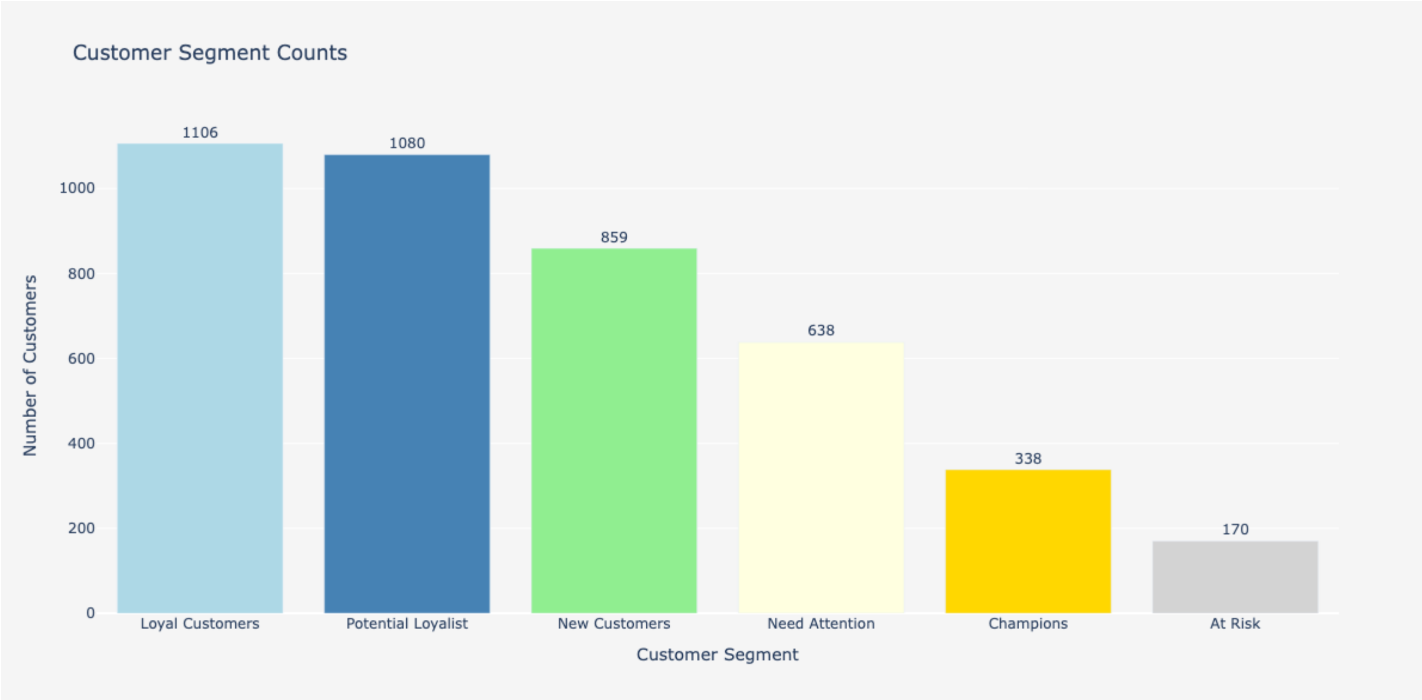


In [33]:
```
numeric_data = df.select_dtypes(include=['number'])

# 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
```

I evaluated in the PCA version of the dataset because that's where the clusters actually derived, showing the most important patterns in the data; such evaluation in this space actually gives a better view of cluster quality and helps bring to light the real connections and differences created during clustering.

In [24]:
```
# 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()
```



- loyal customers → Customers who shop very often and it has been a short time since their last purchase.
- potential loyalists → Customers who shop moderately often and it has not been long since their last purchase.
- new customers → A class of customers who have not shopped frequently (maybe once) and have been shopping for a short period of time, they are considered as new customers.
- need attention → This is the class of customers in the middle of the RF graph (33%), moving towards the risky group if not addressed.
- champions → They are our champions, our crown jewels! Customers who shop very often and have made their last purchase within a very short period of time.
- at risk → A class of customers who shop relatively frequently but have not shopped for a long time.

9. Classification Techniques

Implemented basic classification models:

Logistic Regression: It is a statistical method for predicting one of two possible outcomes. It looks at the relationship between a dependent variable and one or more independent variables. In this project, it attained an accuracy of 56.3%.

Decision Trees: A tree-structured classifier where internal nodes represent feature tests, branches represent outcomes, and leaf nodes represent class labels. It achieved 56.7% accuracy in this project. Evaluation metrics included accuracy, precision, recall, and F1-score. I think those results were pretty good because our dataset is very big.

```
In [43]:
# 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

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
```

10. Advanced Classification Methods

Random Forest: An ensemble learning method that builds multiple decision trees and merges their outputs for more accurate and robust predictions. It provided 81% accuracy with insights into feature importance.

Support Vector Machine (SVM): I tried to add SVM model because of it's complexity it kept crushing my kernel so I didn't apply it.

```
In [44]:
# 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))
```

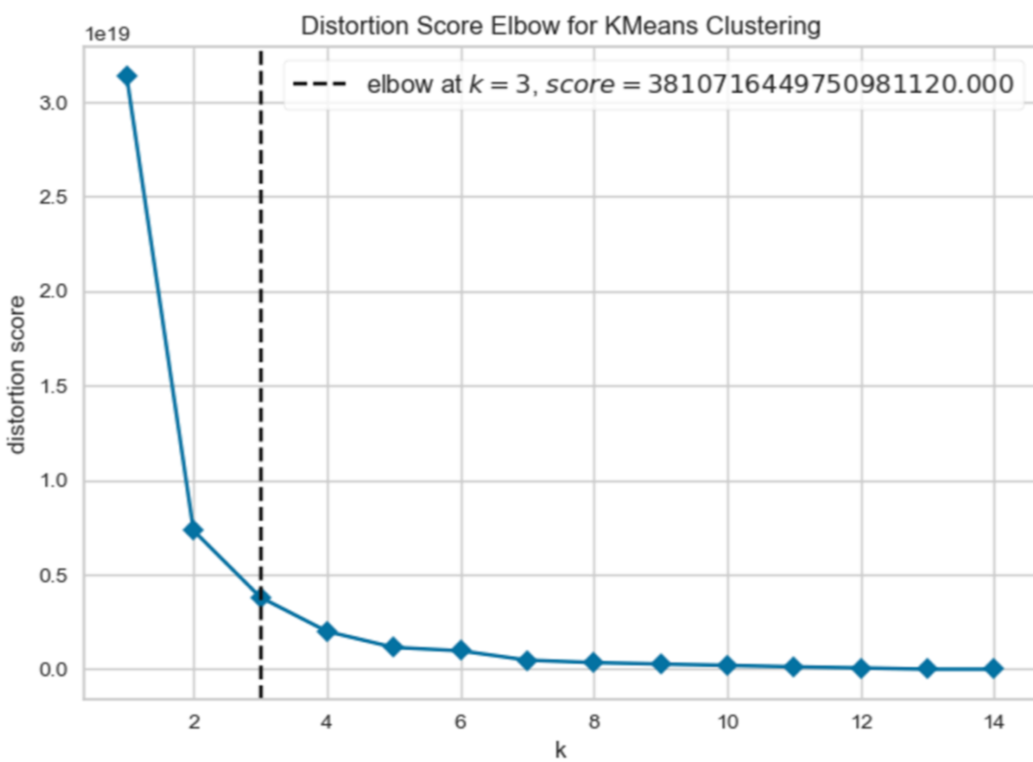
```
Random Forest Evaluation:
              precision    recall  f1-score   support

           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
```

11. Clustering Techniques

Methods: Elbow Method

There are many clustering algorithms to choose. It is a good idea to explore a range of clustering algorithms and different configurations. It might take some time to figure out which type of clustering algorithm works the best for the given data, but when you do, you'll get invaluable insight on your data. The **Elbow Method** is a popular technique used for this purpose in K-Means clustering. The method consists of plotting the explained variation as a function of the number of clusters and picking the elbow of the curve as the number of clusters to use.

In [34]:
```
# 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()
```

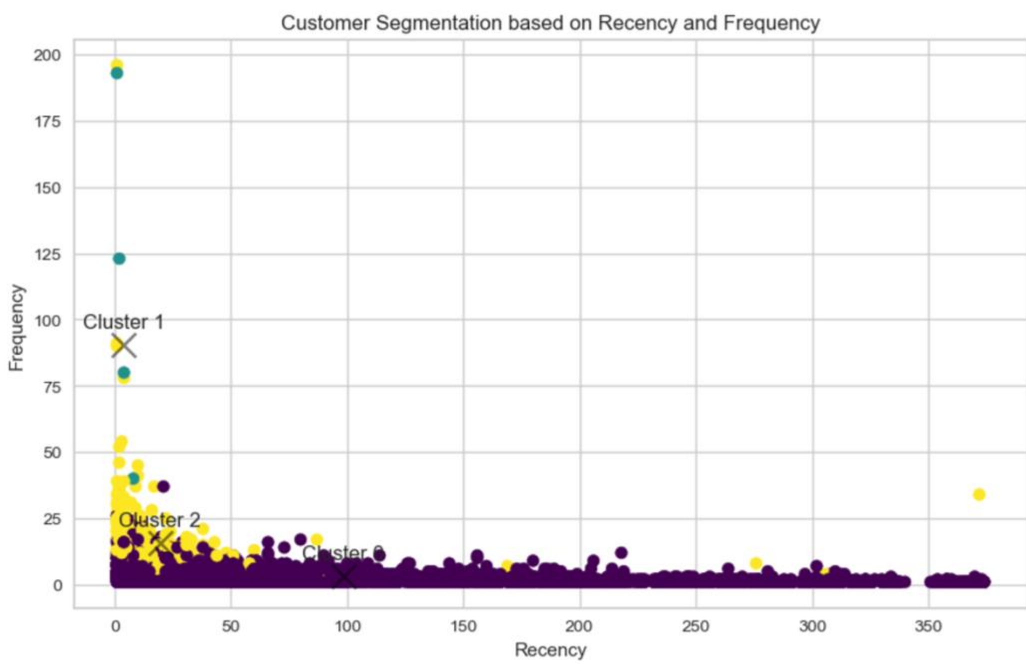


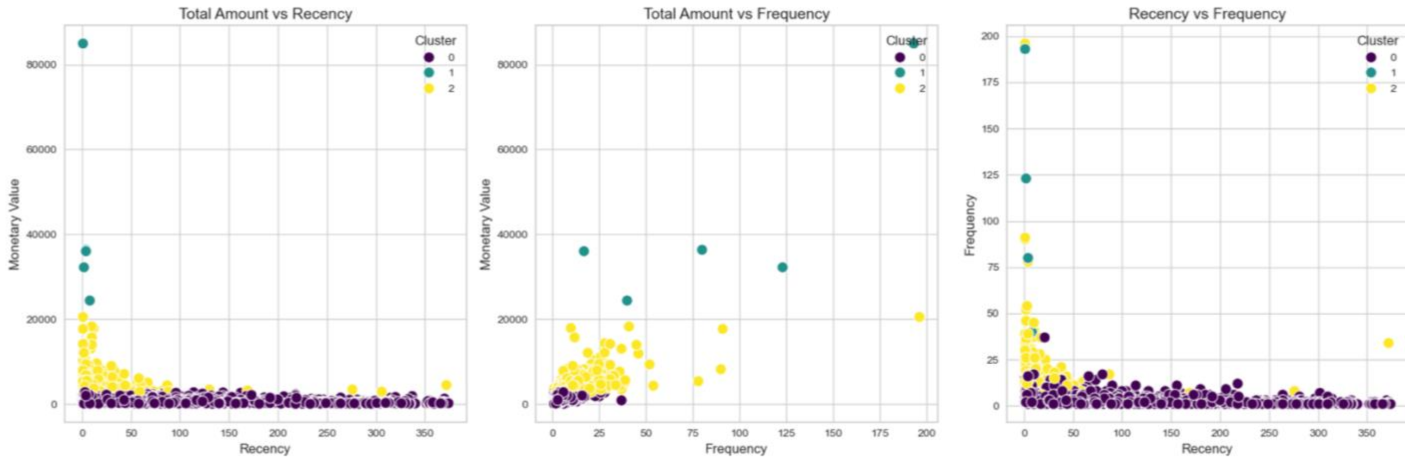
Centroid-based

These types of algorithms separate data points based on multiple centroids in the data. Each data point is assigned to a cluster based on its squared distance from the centroid.

This is the most commonly used type of clustering. K-Means algorithm is one of the centroid based clustering algorithms. Here k is the number of clusters and is a hyperparameter to the algorithm.

In [36]:
```
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()
```



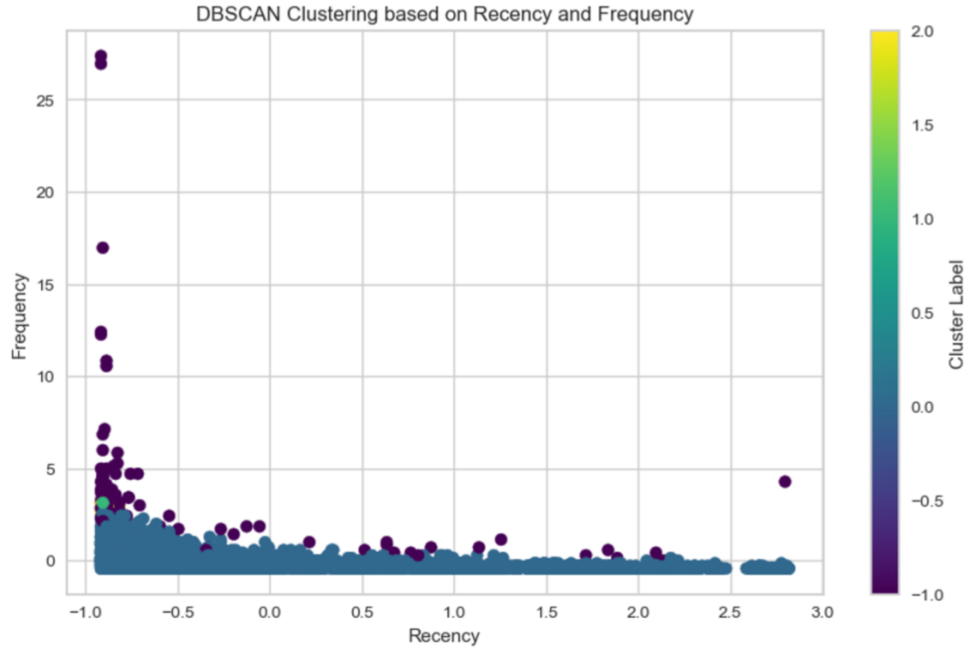


12. Advanced Clustering Techniques

Methods: DBSCAN and GMM

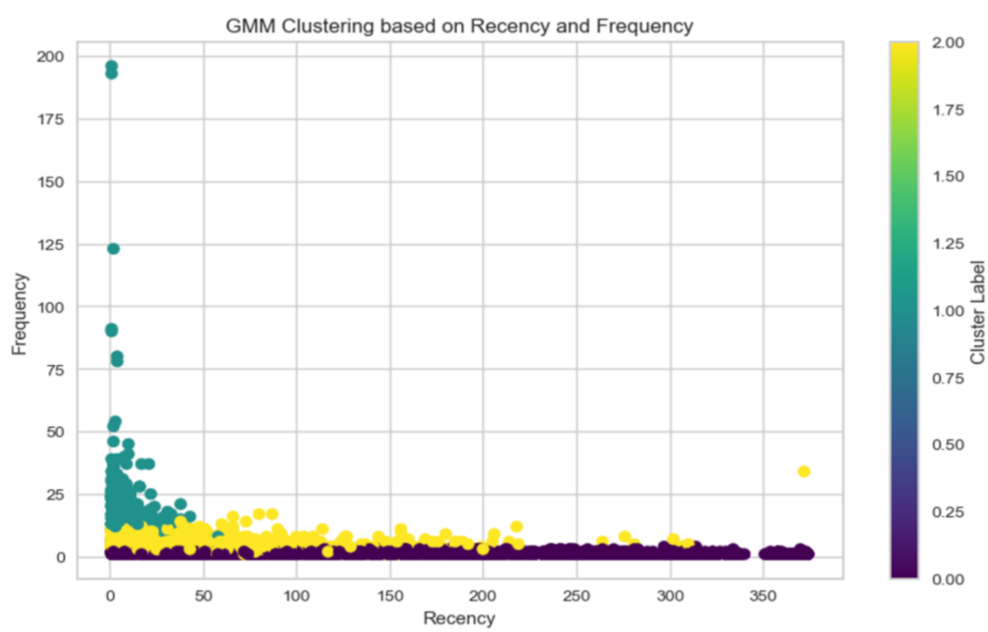
DBSCAN stands for density-based spatial clustering of applications with noise. It's a density-based clustering algorithm. It is able to find irregular-shaped clusters. It separates regions by areas of low-density so it can also detect outliers really well. This algorithm is better than k-means when it comes to working with oddly shaped data. Found 3 clusters with 112 noise points, effectively handling outliers.

```
In [46]:
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
```

A Gaussian mixture model (GMM) attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset. In the simplest case, GMMs can be used for finding clusters in the same manner as k-means, but because GMM contains a probabilistic model under the hood, it is also possible to find probabilistic cluster assignments.




```
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]]
```

13. Association Rule Mining

Association rule mining finds links between items that people often buy together. We used the Apriori algorithm to look at transaction data and find patterns that show what customers like.

Frequent itemsets: Found item combinations with high support.

Customers who purchased "4 PURPLE FLOCK DINNER CANDLES" also commonly purchased "CHRISTMAS GIFT BAG LARGE".

In [55]:

```
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
```

In [57]:

```
# 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
```

	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

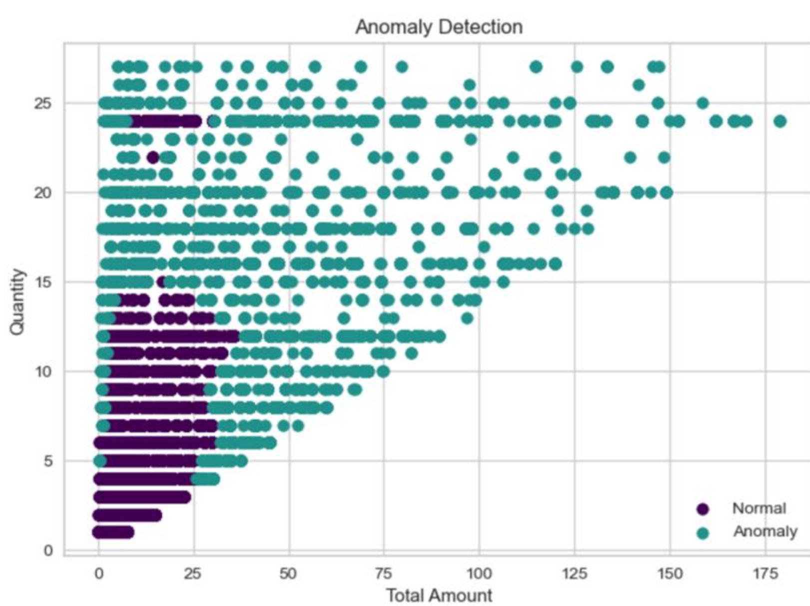
14. Anomaly Detection

Anomaly detection is the identification of rare or unusual patterns hidden in data that are strikingly different from the norm. This helps find fraud, mistakes, or unusual behaviors.

Isolation Forest: Identified anomalies in purchasing behavior, indicating possible fraud or data errors.

In [39]:

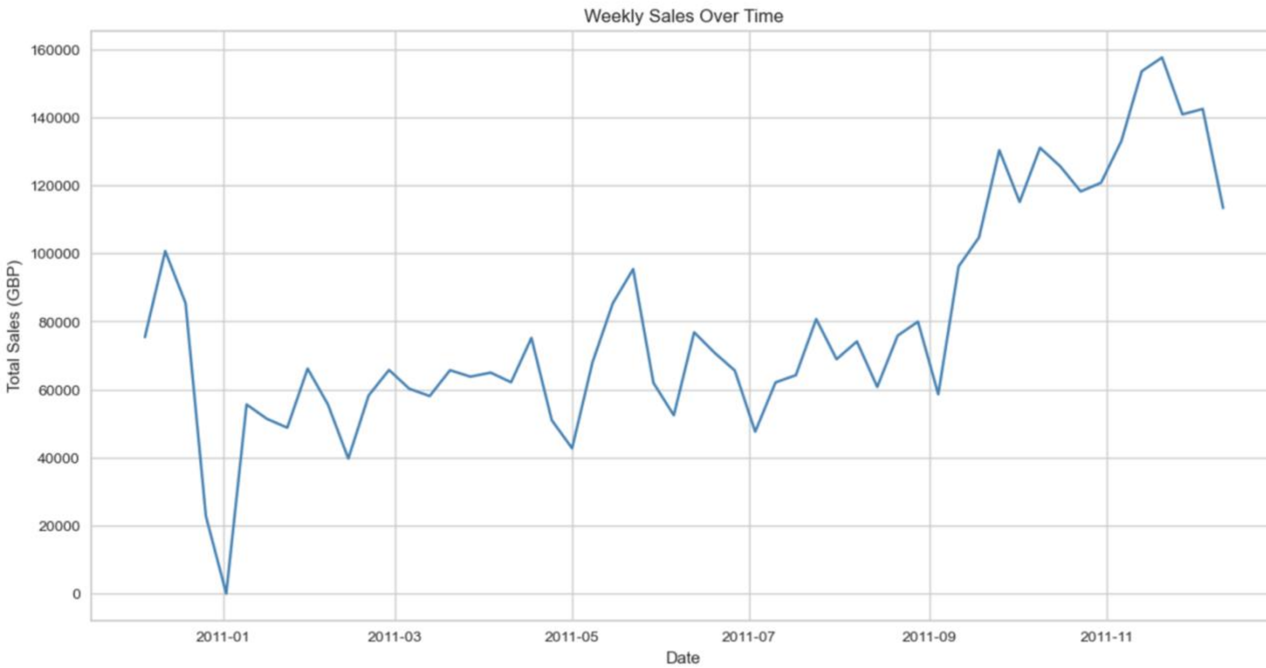
```python
# Preparing data for anomaly detection
anomaly_data = df[['Total_Amount', 'Quantity']]

# 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()
```



15. Time Series Analysis

Method: The time series analysis considers data points gathered at specified times in order to seek a trend, seasonal changes, and patterns. This would, therefore, enable forecasters to predict future values using past data. First, I illustrated the chart of weekly sales data to identify how models will effect.

```
In [49]:
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()
```




```
In [50]:
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.")
```

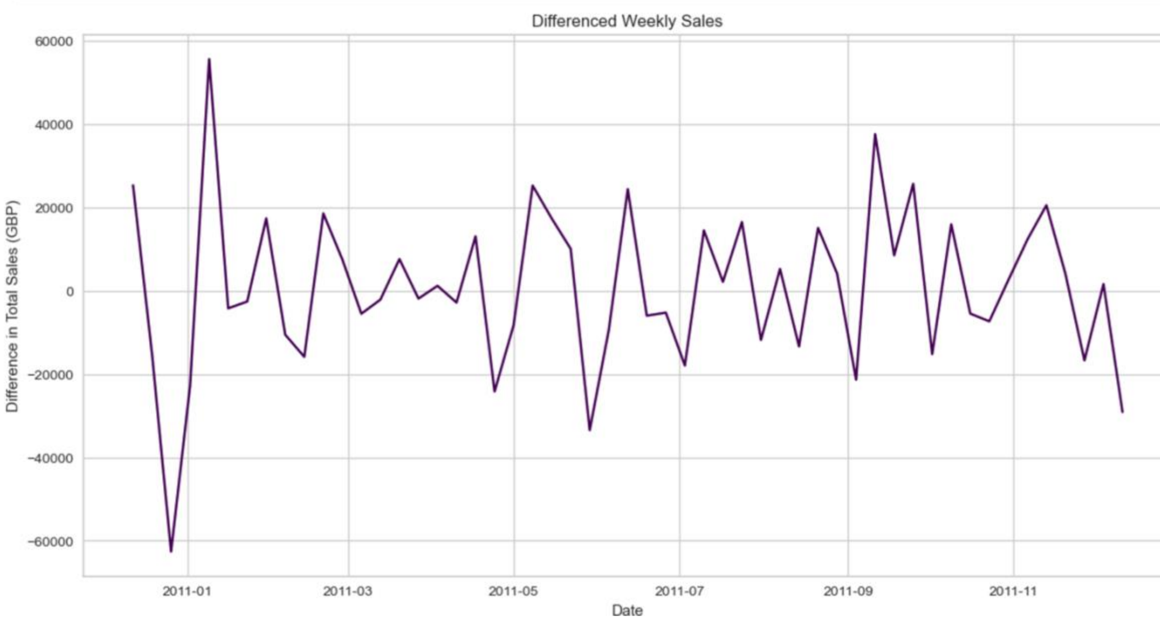
```
ADF Statistic: -2.0241682151542157
p-value: 0.2760777367100903
Critical Values: {'1%': -3.560242358792829, '5%': -2.9178502070837, '10%': -2.5967964150943397}
The time series is not stationary. Differencing may be required.
```

```
In [52]:
model = ARIMA(df_weekly_diff, order=(1, 1, 1))  # Adjust (p, d, q)
model_fit = model.fit()

print(model_fit.summary())
```

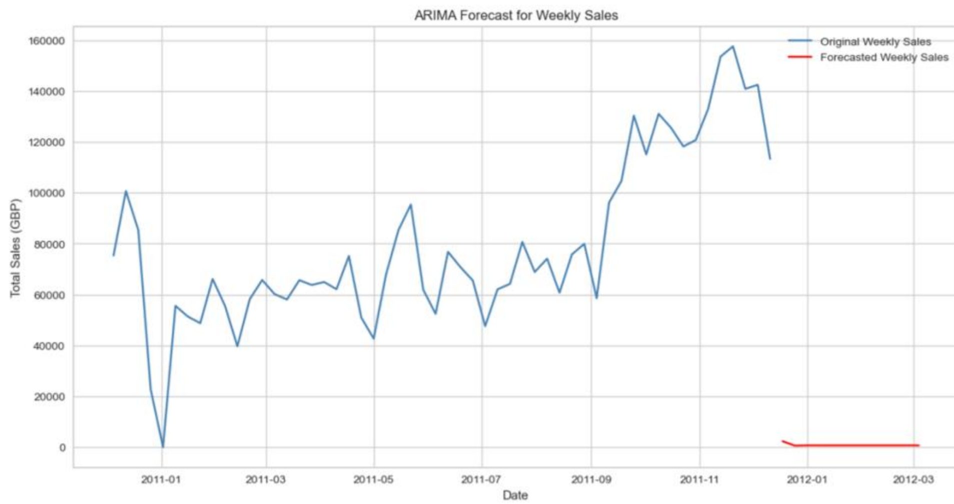
```
                               SARIMAX Results
==============================================================================
Dep. Variable:           Total_Amount   No. Observations:                   53
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -588.626
Date:                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 unstabl
e.
```

Since Differencing required I had to make a differenced weekly sales for better analysis.

```
In [51]:
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
```

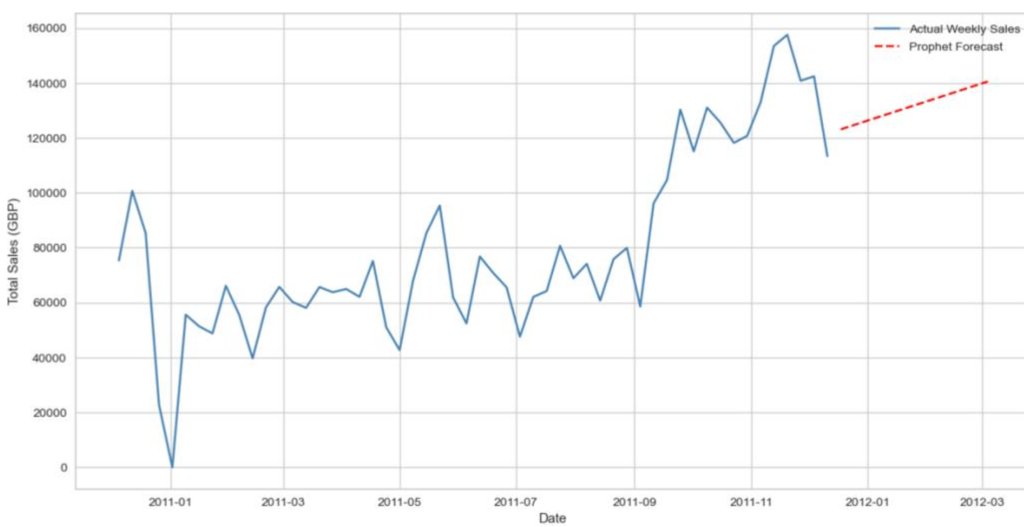


ARIMA: Forecasted weekly sales with moderate accuracy. However, the results weren't as good as I expected because the forecasted weekly patterns were below the average. So, I applied the Prophet method.



Prophet: Outperformed ARIMA by capturing seasonal trends effectively. I got the result that I was expecting. You can see from the chart below.

```
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()
```



16. Text Mining and NLP

Text mining and natural language processing (NLP) involve extracting meaningful insights from textual data.

These techniques process unstructured text data to reveal customer sentiment, preferences, and trends.

Techniques:

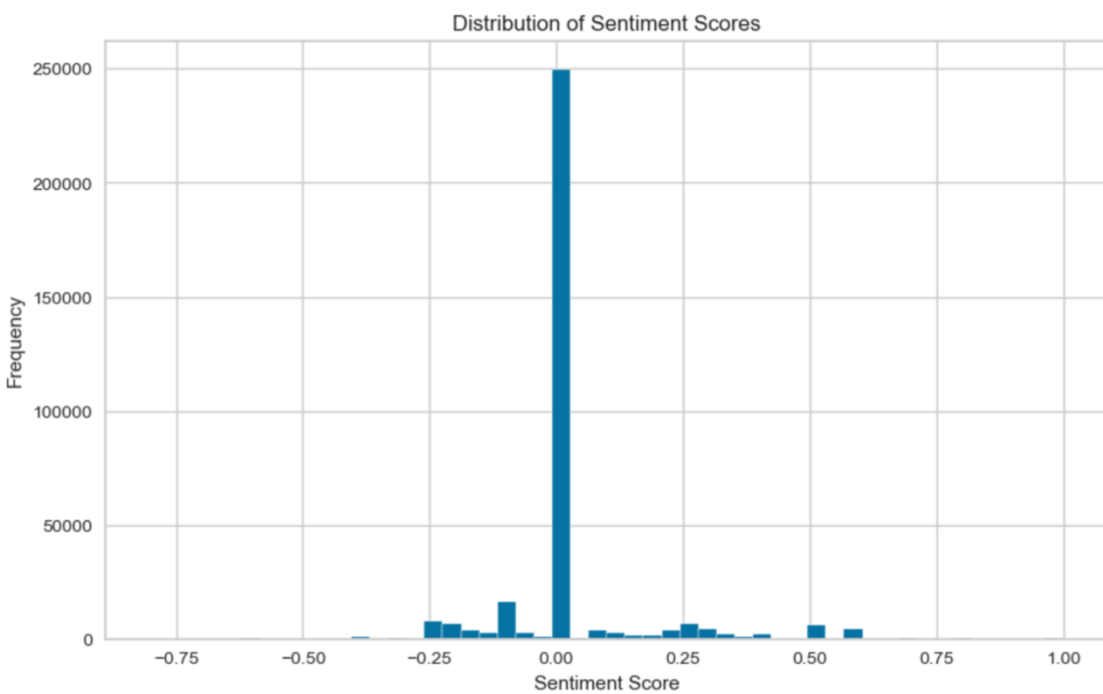
Tokenization and Stemming: Processed customer reviews.

Sentiment Analysis: Revealed overall positive sentiment in reviews.

Tokenization and Stemming: Processed customer reviews.

Sentiment Analysis: Revealed overall positive sentiment in reviews.

In [58]:
```
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()
```



17. Challenges and Solutions

Challenges:

There were many different hurdles during the project. Its size would often result in kernel crashes while working with the dataset. I faced issues with an imbalanced feature set which had inconsistent values in many features and a lot of missing and wrong valued notations escalated the challenging index inconsistency while pre-processing as well. Moreover, finding the right parameters for machine learning models and algorithms was challenging but funny at the same time.

Solutions:

In the face of these challenges, I used sampling and filtering techniques to limit the size of data being stored in memory. To evaluate imbalanced data, I used resampling and precision-recall curves as performance metrics. More preprocessing was done to make the data consistent where tools like auto_arima and hyperparameter tuning were automated in selecting the best parameters which resulted in better performance of the model. Together, those solutions made the analysis more powerful and eliminated some major issues.

18. Conclusion

The project brought out valuable insights into customer behavior. Customer segmentation identified clear groups to which targeted marketing strategies would be addressed. Association rules gave actionable recommendations on bundling products that are frequently bought together. Time series forecasting made accurate predictions for future sales trends that supported business planning. It therefore became clear that data mining depends largely on preprocessing and selection of features. Also, care must be taken concerning the model's complexity with its interpretability, considering obtaining meaningful results from our current analyses. Exploring deep learning models and integrating real-time data pipelines could further improve the scalability and accuracy of the analysis. These enhancements would strengthen the project's impact on business decision-making.

19. References

https://www.kaggle.com/datasets/ulrikthygepedersen/online-retail-dataset

20. Appendices

https://github.com/Aiman1517/Data-Mining-2024/tree/ae3ee9839430a9915dfb8d41aac1a62b52567b9c/Final