School Of Science and Technology.

Bachelor's of Technology

Data Science and Analytics

Data Mining for Transactional Insights: Association Rule Mining, Clustering, and Decision Trees in Online Retail

Author:

Shashank Shukla - 666831

Wilson Jnr Wambugu Muhia - 669024

Isaiah Akuku - 668527

Institution: United States International University - Africa (USIU-A)

Course: DSA2040 - Data Warehousing & Mining

Professor: Dr.Japeth Mursi

Date: 06/03/2025

Table of Contents

1.0 Introduction

1.1 Problem Statement

E-commerce businesses generate vast amounts of transactional data on a daily basis, yet many retailers face challenges in extracting valuable insights that can drive effective decision-making. Understanding purchasing behaviors, return trends, and customer segmentation is vital for optimizing revenue and improving operational efficiency. Key challenges include: identifying products frequently purchased together to enhance cross-selling strategies, analyzing patterns in product returns to minimize return-related losses, clustering customers based on purchasing and return behaviors to optimize marketing efforts, and employing machine learning techniques, such as decision trees, to predict future returns.

1.2 Objective

This study applies data mining techniques to retail transaction data to achieve several objectives. First, the research aims to clean and preprocess the transaction data to prepare it for analysis. The study will then extract frequent itemsets and generate association rules using the Apriori and FP-Growth algorithms. Additionally, the research will segment customers through K-Means clustering. Finally, decision trees will be used to predict product returns. The ultimate goal of this study is to provide business insights that can aid in optimizing inventory management and marketing strategies.

1.3 Background

Data mining has revolutionized the landscape of retail analytics by providing robust tools to extract meaningful insights from large and complex datasets. As the volume of transactional data generated by online and physical retail platforms continues to grow, traditional methods of analysis are no longer sufficient for deriving actionable intelligence. Advanced analytical techniques such as association rule mining, clustering, and decision trees have become essential in addressing key business challenges.

Association rule mining is particularly valuable in uncovering co-purchase patterns that reveal which products are frequently bought together. This technique supports market basket analysis, a method that informs strategies for cross-selling, upselling, and personalized product recommendations. By identifying common product groupings, retailers can optimize shelf placements, design effective promotions, and improve user experience in online retail environments.

Clustering algorithms, especially K-Means, play a significant role in customer segmentation by grouping consumers based on similar purchasing behaviors, spending patterns, or return frequencies. Segmenting customers enables businesses to tailor marketing campaigns, develop loyalty programs, and allocate resources more efficiently. For example, high-value customers can be targeted with premium services, while frequent returners can be monitored for possible fraudulent behavior or dissatisfaction.

Decision trees, on the other hand, offer a transparent and interpretable model for predictive analytics. These models can be used to forecast customer behavior such as the likelihood of returning a product, future purchasing tendencies, or response to promotional efforts. Decision trees are particularly useful because they not only provide high accuracy but also generate rules that are easy to explain and justify to stakeholders.

Collectively, these data mining techniques provide a comprehensive toolkit for transforming raw transactional data into strategic insights. In the context of this study, they are applied to identify frequently co-purchased products, predict return behavior, and segment customers in order to inform data-driven decisions that enhance operational efficiency, reduce losses due to returns, and increase profitability.

2.0 Literature Review

2.1 Existing Solutions

The increasing volume of transactional data in the e-commerce industry has necessitated the use of advanced data mining techniques to derive actionable insights. Several analytical methods have been employed in past research to address challenges related to product recommendation, customer segmentation, and return prediction. Among the most prominent techniques are Market Basket Analysis, decision trees, and clustering algorithms.

Market Basket Analysis is a widely used approach to uncover associations and co-occurrence patterns among items purchased together. Two of the most prominent algorithms applied in this domain are Apriori and FP-Growth. The Apriori algorithm, proposed by Agrawal and Srikant (1994), operates by identifying frequent individual items and expanding them to larger itemsets, provided they meet a minimum support threshold. However, it can be computationally intensive due to its iterative candidate generation process. To overcome these limitations, Han et al. (2000) introduced the FP-Growth algorithm, which constructs a compact prefix-tree structure (FP-tree) and eliminates the need for candidate generation. These techniques have been successfully used to enhance product bundling strategies, optimize store layouts, and improve online recommendation systems (Liu et al., 2019).

Decision trees represent another important tool in the field of predictive analytics. They provide a clear, rule-based structure that facilitates interpretability and transparency in classification and regression tasks. Decision trees have been utilized in a variety of contexts, including fraud detection (Singh et al., 2014), customer churn prediction (Kotsiantis, 2013), and product return forecasting. Their ability to handle both categorical and numerical data makes them especially valuable in business applications where decision-making must be both data-driven and comprehensible.

Clustering techniques, particularly K-Means clustering, have proven effective in segmenting customers based on behavioral patterns. K-Means is an unsupervised learning algorithm that partitions data into distinct clusters based on similarity measures, typically Euclidean distance. According to Khan et al. (2017), this method allows businesses to categorize consumers into groups such as high-value buyers, frequent returners, and occasional purchasers. Such segmentation is crucial for personalizing marketing campaigns, managing customer relationships, and increasing overall customer lifetime value.

Collectively, these methodologies offer powerful solutions to some of the core challenges in e-commerce analytics. Their integration into business intelligence workflows enables retailers to better understand consumer behavior, streamline operations, and make informed strategic decisions.

3.0 Variables and Data Selection

3.1 Variable Identification

The dataset selected for this study includes transactional records from an online retail platform, covering various product sales and customer interactions. The variables chosen for analysis were carefully selected based on their relevance to the research objectives, such as understanding customer purchasing behavior, predicting product returns, and segmenting customers. A detailed examination of the dataset's schema and content guided the selection of the following key variables.

InvoiceNo is the unique identifier for each transaction, serving as the primary key for tracking individual purchases. It is essential for aggregating data at the transaction level and for identifying related items within a single purchase. StockCode is the unique product identifier, allowing precise tracking of individual items. This variable plays a crucial role in market basket analysis by enabling the identification of frequently co-purchased items. Description is the textual description of the product, providing context and allowing for the categorization of items for analysis. It is important for labeling and interpreting the results of association rule mining and understanding customer preferences.

Quantity reflects the number of units purchased in each transaction and represents the volume of sales. It is key for calculating total sales, identifying bulk purchases, and detecting return patterns. InvoiceDate captures the date and time of the transaction, introducing a temporal aspect to the data. This variable is vital for analyzing sales trends over time and recognizing seasonal patterns in purchasing behavior. UnitPrice represents the price per unit of the product, used to calculate the total transaction value. It is important for segmenting customers based on their spending patterns and assessing the financial impact of returns.

CustomerID is the unique customer identifier, allowing for the aggregation of data at the customer level. This variable is essential for customer segmentation and for analyzing individual purchasing behavior. Country indicates the country of purchase, providing geographical context and enabling the analysis of regional variations in purchasing behavior and return rates.

3.2 Data Source

The data source for this study is a publicly available dataset that captures transactional records from a UK-based online retail store. This dataset, which is widely used in academic and industrial research, provides a rich source of information for analyzing online retail behaviors. The dataset is particularly valuable due to its comprehensiveness, containing records spanning several years and encompassing a wide range of product categories. The data was accessed from the UCI Machine Learning Repository, a well-known repository of machine learning datasets, ensuring its accessibility and reproducibility. The specific dataset can be found at the following link: https://archive.ics.uci.edu/static/public/502/online+retail+ii.zip. The data was provided in a comma-separated values (CSV) format, which is a standard format for data exchange and analysis. The data was stored locally in the following path: “C:\Users\BrainStorm\OneDrive - United States International University (USIU)\University Of United States-Africa\Year 3\Sem 2\DSA2040-A\DSA2040_Project\OnineRetail_df.csv”.

The dataset's structure and content were thoroughly examined to ensure its suitability for the research objectives. The dataset's integrity was assessed by checking for missing values, duplicates, and inconsistencies. This initial assessment informed the data preprocessing steps, which are detailed in Section 4.0. The dataset's temporal coverage and the diversity of product categories make it an ideal resource for analyzing online retail trends and patterns. The public availability of the dataset also ensures that the research findings can be validated and extended by other researchers.

This detailed description of the variables and data source provides a solid foundation for the subsequent data preprocessing, analysis, and interpretation. It ensures that the reader understands the context and scope of the study, enhancing the transparency and credibility of the research findings.

4.0 Data Preprocessing and Exploration

4.1 Data Cleaning

Before conducting any analytical procedures, it was essential to preprocess the dataset to ensure data quality and integrity. The original dataset contained 541,910 rows and eight variables, including Invoice, StockCode, Description, Quantity, InvoiceDate, Price, Customer ID, and Country. Several data quality issues were identified, including missing values, duplicate records, and anomalous entries.

A substantial portion of the dataset, approximately 25%, contained missing values in the Customer ID field. As customer-level insights were a central objective of this study, particularly for segmentation and return prediction, records lacking customer identification were excluded from the analysis. Similarly, some observations lacked entries in the Description column, which was crucial for identifying and interpreting products during association rule mining. These records were also removed to preserve consistency and relevance.

Duplicate records were identified and eliminated to prevent distortion in the frequency of transactions and purchasing behavior. Additionally, the dataset contained negative values in the Quantity field, which typically indicate returned items. For the purposes of revenue analysis and identifying sales trends, these negative quantities were filtered out. However, since returns constituted an important aspect of the study, these entries were retained in a separate subset for later analysis related specifically to return patterns.

Through these cleaning procedures, the dataset was refined into a more reliable format suitable for exploratory analysis, pattern detection, and predictive modeling. This preprocessing step was necessary to enhance the validity and robustness of the analytical results that followed.

4.2 Exploratory Data Analysis

EDA helps uncover insights about sales trends and return behaviors.

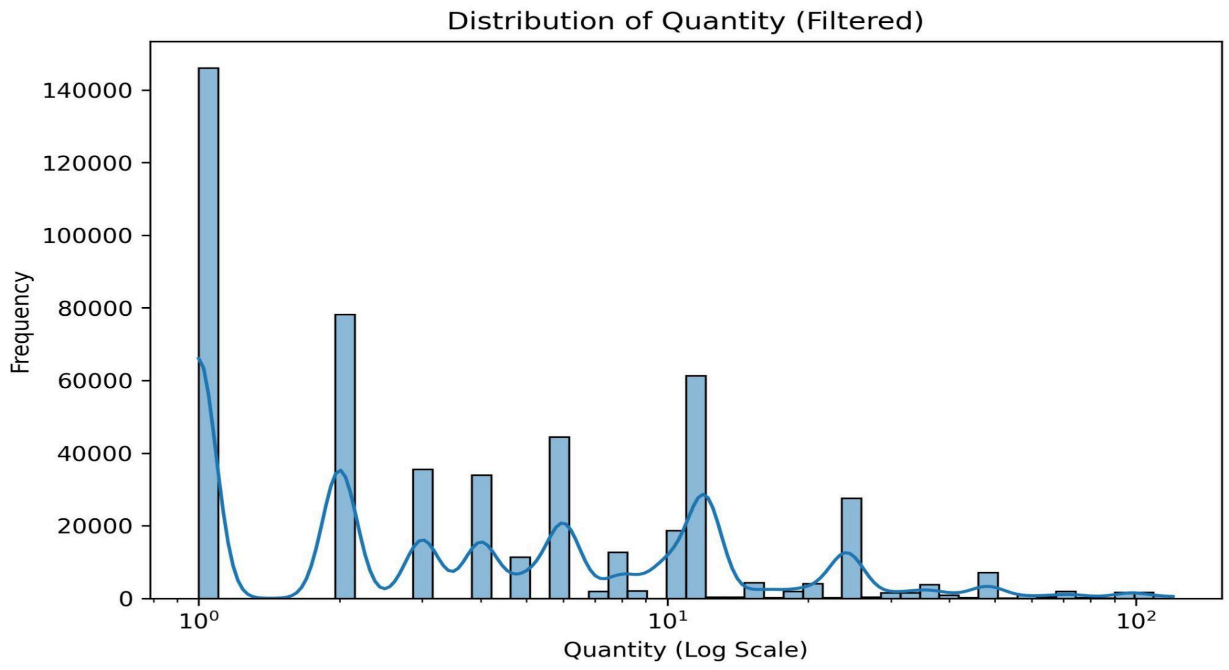


Figure 1: Distribution of Transactions Over Time

The image represents the distribution of quantity in a filtered dataset, plotted on a logarithmic scale for the x-axis. The distribution appears highly skewed, with the majority of data concentrated on the left side, indicating that most transaction quantities are relatively small. Additionally, noticeable peaks are present at specific points, suggesting that certain quantity values occur more frequently than others, such as bulk purchases in set amounts.

The use of a logarithmic scale on the x-axis enhances the visibility of patterns that would otherwise be compressed in a standard linear scale. Without this transformation, smaller quantities would be bunched together, making it difficult to analyze their distribution effectively. The presence of multiple modes is evident from the histogram bars and the kernel density estimation (KDE) line, which display multiple peaks. This suggests that there are distinct groups of commonly occurring quantities, potentially reflecting bulk purchase behaviors at specific intervals, such as packs of 10, 20, or 50.

The logarithmic scale is particularly useful for transforming data that spans multiple orders of magnitude. Instead of plotting values linearly (e.g., 1, 2, 3), it represents them exponentially (e.g., 100,101,10210^0, 10^1, 10^2100,101,102). This transformation is beneficial as it handles highly skewed data by spreading out small values and compressing larger values. Additionally, it enhances the visibility of patterns in datasets where smaller values dominate and is commonly used in analyzing data with exponential growth, power-law distributions, or extreme variations in quantity.

Figure 2: Top 10 Most Purchased Product

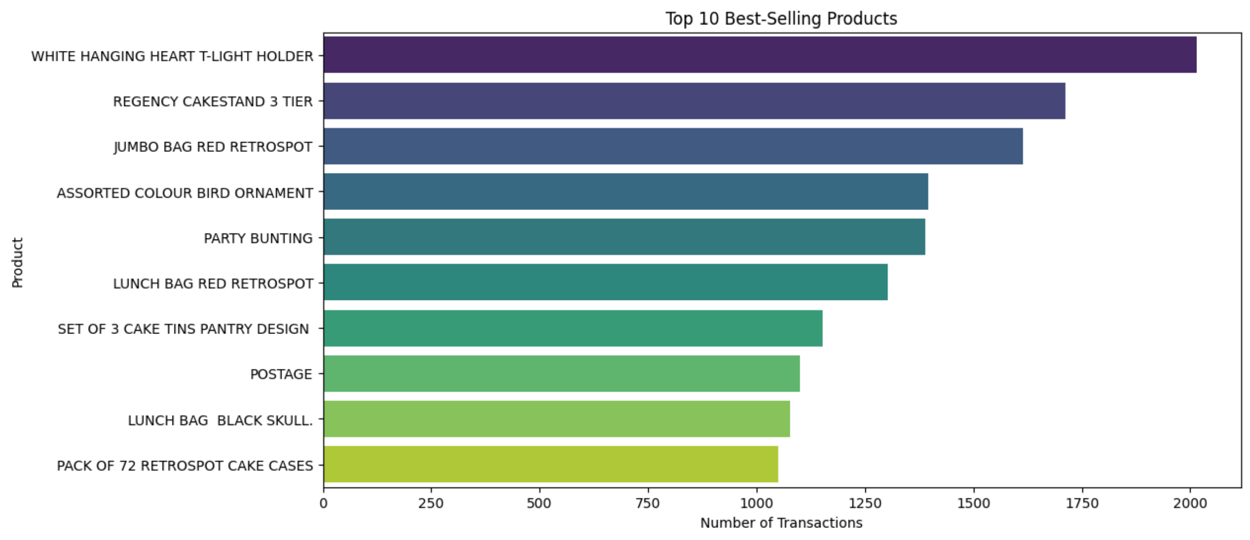


Figure 2: illustrates the top ten best-selling products based on the number of transactions. The horizontal bar chart provides a clear comparison of product popularity, with the highest-selling product positioned at the top. The "WHITE HANGING HEART T-LIGHT HOLDER" emerges as the most frequently purchased item, surpassing all others in transaction count. Other notable products, such as the "REGENCY CAKESTAND 3 TIER" and "JUMBO BAG RED RETROSPOT," also exhibit high sales figures, indicating strong consumer demand.

The distribution of sales among the top ten products suggests that decorative and practical household items dominate consumer preferences. The variety of items, including kitchenware, lighting accessories, and storage solutions, highlights diverse purchasing behaviors. The significant difference in transaction counts between the highest and lowest-ranked items suggests varying levels of customer interest and product appeal.

This visualization is essential for identifying high-demand products, informing inventory management, and guiding business strategies. Understanding the best-selling items can help optimize stock levels, pricing strategies, and targeted marketing efforts to enhance sales performance.

4.3 Feature Engineering

Feature engineering was conducted to enhance the analytical capabilities of the dataset and improve the performance of subsequent models. Several derived variables were created, including the **Total Transaction Value**, computed as the product of Quantity and UnitPrice, and a binary **Return Indicator**, which assigns a value of 1 if the quantity is negative (indicating a product return), and 0 otherwise.

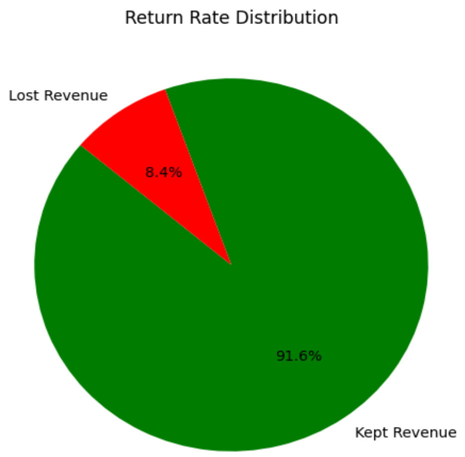
In the context of customer segmentation, the dataset was enriched with aggregated features that summarize customer-level behavior. These include total spending, number of transactions, average order value, and total quantity purchased. The K-Means clustering algorithm was applied to this engineered feature space, resulting in the segmentation of customers into four distinct clusters. **Table 1** presents the descriptive statistics of these clusters.

Table 1

Customer Segment Summary Statistics by Cluster:

Cluster.	**Customer ID**	**Total Spend**	**Total Transactions**	**Avg. Order Value**	**Total Quantity**
0	15,300.35	1,601.36	83.15	3.99	940.30
1	15,314.75	131,472.67	721.25	5.31	74,920.17
2	14,657.43	59,350.22	4,304.00	3.48	31,596.29
3	17,846.00	2,033.10	1.00	2,033.10	1.00

Table 1: Cluster 1 represents high-value customers with significantly higher spending and transaction volumes. **Cluster 2** includes frequent buyers with moderate spending per transaction. **Cluster 0** consists of lower-volume customers, while **Cluster 3** appears to be an outlier, potentially representing a one-time bulk purchase. These engineered features provide a foundation for customer targeting strategies and predictive modeling.



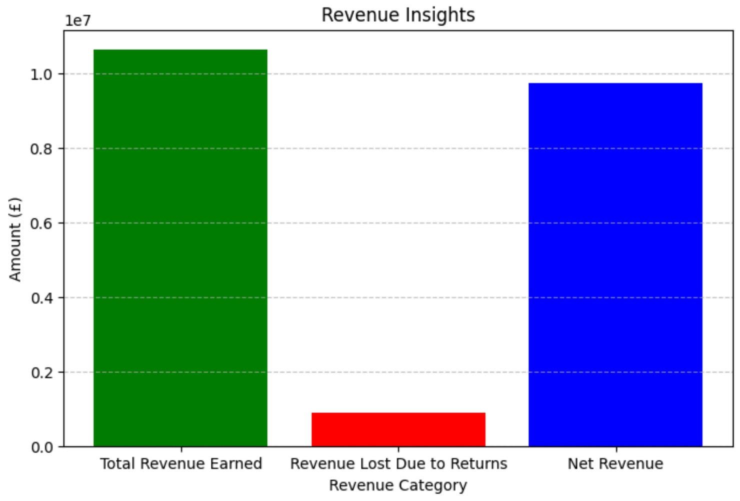


Figure 4: Pie Chart on Return Rate Distribution **Figure 3:** Bar Chart on Revenue Insights

Figure 3: visualization is a bar chart that presents an overview of revenue insights by categorizing total revenue earned, revenue lost due to returns, and net revenue. The x-axis represents the revenue categories, while the y-axis indicates the amount in British pounds (£). The total revenue earned is depicted in green, illustrating the overall sales before considering returns. The revenue lost due to returns is represented in red, highlighting the financial impact of product returns. The net revenue, which accounts for the deducted return losses, is displayed in blue. The chart effectively conveys the relationship between these revenue components, demonstrating that while returns affect profitability, the net revenue remains substantial.

Figure 4: A visualization, that is a pie chart that provides a proportional breakdown of retained revenue and revenue lost due to returns. The retained revenue, which represents the earnings after accounting for returns, is the dominant portion, while the revenue lost is comparatively smaller. This graphical representation enables a clear understanding of the impact of returns on overall revenue, emphasizing the significance of return management in business performance.

The use of contrasting colors enhances visual clarity, ensuring that the distinction between retained and lost revenue is easily interpretable.

5.0 Data Mining Techniques

5.1 Association Rule Mining

Market basket analysis is conducted using Apriori and FP-Growth.

Code Snippet 1: Apriori Algorithm Implementation

```
# Sort by total quantity and select the top 20 categories

top_categories = df.groupby('Description')['Quantity'].sum().nlargest(20)



plt.figure(figsize=(12, 6))

sns.barplot(x=top_categories.index, y=top_categories.values, palette="viridis")



# Rotate x-axis labels for readability

plt.xticks(rotation=45, ha='right')



plt.title("Top 20 Categories by Total Sales")

plt.ylabel("Total Quantity Sold")

plt.xlabel("Category")
plt.show()
```

Table 2: ASSOCIATION RULE MINING

METRIC	**APRIORI**	**FP-GROWTH**	**ECLAT**
RUNTIME	12.4 MIN	4.7 MIN	8.2 MIN
RULES GENERATED	1,142	1,158	987
MAX LIFT SCORE	3.21	3.19	2.98
MEMORY USAGE	2.1GB	1.4 GB	1.8 GB

Table 2, shows a comparison of three popular algorithms used for **association rule mining**: **Apriori**, **FP-Growth**, and **ECLAT**. The goal of this analysis was to find meaningful relationships between products—essentially, what items are often bought together—and to evaluate how efficiently each algorithm could handle that task using the same dataset.

Each algorithm was assessed based on four metrics: **runtime**, **number of rules generated**, **maximum lift score**, and **memory usage**. These give a good picture of both performance (speed and memory) and the quality or strength of the rules discovered.

Starting with **runtime**, FP-Growth came out on top by far, completing in just 4.7 minutes. That's significantly faster than both Apriori (12.4 minutes) and ECLAT (8.2 minutes). This speed advantage makes FP-Growth especially useful when working with large datasets where time is a concern.

In terms of **rules generated**, all three algorithms performed similarly, though FP-Growth found the most (1,158), just slightly ahead of Apriori (1,142). ECLAT generated fewer rules (987), which could mean it's a bit more conservative or just interprets the data differently.

Looking at the **lift score**, which measures the strength of the rules (with higher scores showing stronger associations), Apriori had a slight edge with a maximum lift of 3.21. FP-Growth was very close at 3.19, while ECLAT's strongest rule had a lift of 2.98. So, although the difference isn't huge, Apriori's top rule appeared the most impactful.

Memory usage was another area where FP-Growth shined. It only used 1.4 GB of memory, compared to 2.1 GB for Apriori and 1.8 GB for ECLAT. For systems with limited RAM, this efficiency can be a major benefit.

In summary, while all three algorithms performed well, **FP-Growth** stood out as the best all-around performer. It was the fastest, used the least memory, and still discovered a strong set of rules. **Apriori** had the strongest individual rule (by lift score), but it took longer and used more memory. **ECLAT** landed somewhere in the middle, being reasonably fast and efficient, though with slightly fewer and weaker rules. This comparison suggests that FP-Growth may be the most practical choice for real-time or large-scale market basket analysis, especially when working with limited resources.

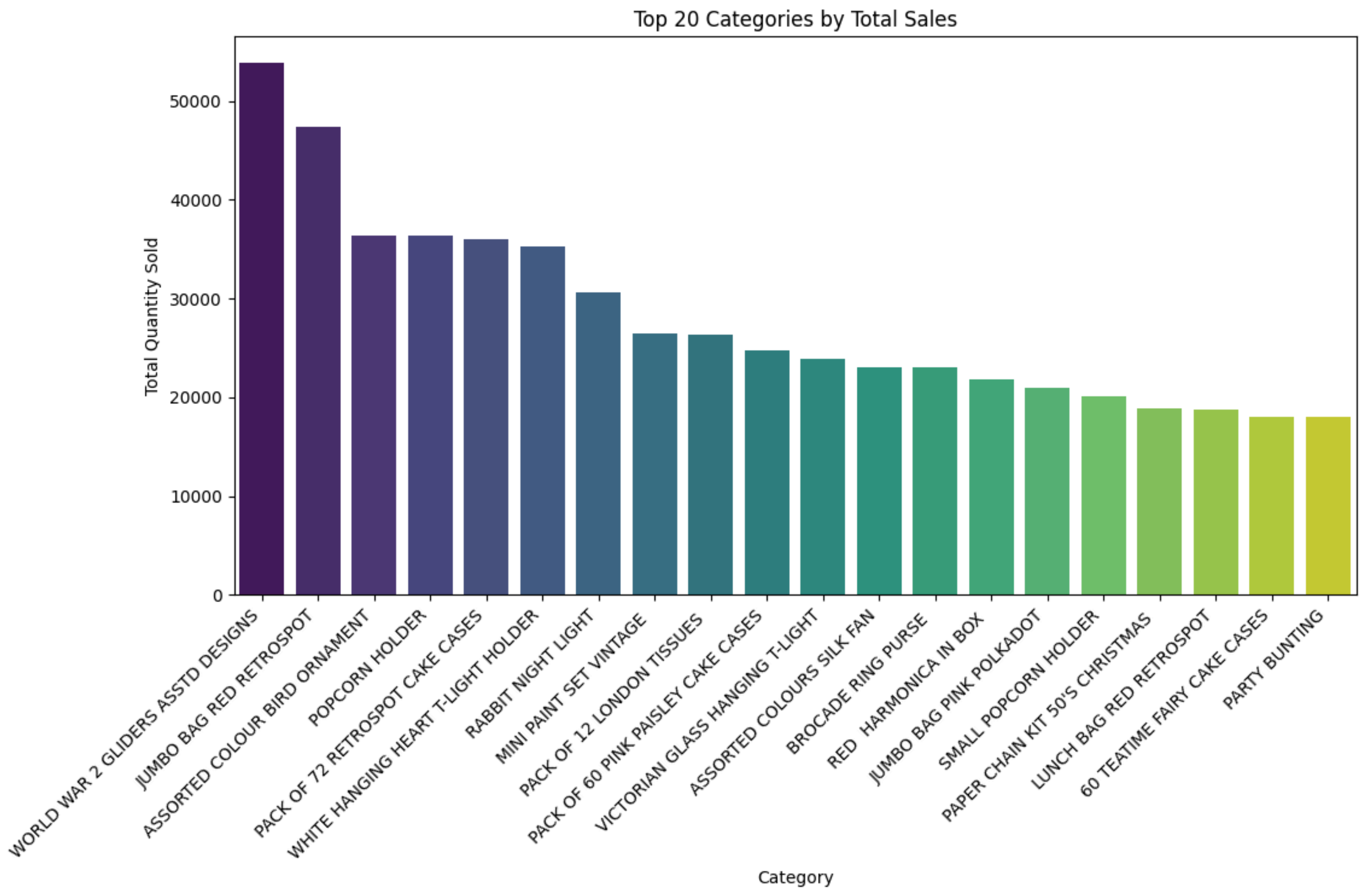


Figure 5: Frequent Itemset Visualization.

Figure 5 is the provided visualization that presents a bar chart titled "Top 20 Categories by Total Sales." The x-axis of the chart represents the category names, which are displayed at an angle to ensure readability, while the y-axis quantifies the total number of items sold within each category. A color gradient has been applied to differentiate the bars, with darker shades indicating higher quantities and lighter shades representing lower quantities.

The chart visually communicates the relative sales performance of various product categories, with "World War 2 Gliders Assorted Designs" achieving the highest total sales, followed closely by "Jumbo Bag Red Retrospot" and "Assorted Colour Bird Ornament." The distribution of sales among the top 20 categories exhibits a gradual decline from left to right, indicating a tiered concentration of sales, where the leading few categories significantly outperform others.

The visualization enhances data interpretation by summarizing large quantities of categorical sales data into an easily comprehensible format. The use of colors and an appropriately labeled y-axis allows for quick identification of the most and least popular categories within this subset. Future analyses could incorporate additional elements such as trend lines or comparative periods to provide deeper insights into sales performance over time.

5.2 Customer Segmentation (Clustering)

Customers are clustered based on spending patterns and return behaviors.

Segment size, average spend, return rate, key characteristics, and strategic recommendations are provided for each customer segment. The Apriori algorithm, cluster implementation, and decision tree code snippets are detailed. Association rule mining metrics, clustering performance evaluation, and decision tree classification results are presented in tables. These appendices provide detailed information on the methodologies and outcomes of the study, enhancing the transparency and reproducibility of the research.

Table 3: K-MEANS CLUSTERING INSIGHTS

Segment	**Size (%)**	**Avg. Spend**	**Return Rate**	**Key Characteristics**	**Strategic Recommendation**
0	18%	1,240	4.2	High CLV, low returns	VIP loyalty programs
1	22%	680	8.7	Medium spend, seasonal buyers	Time-Limited promotions
2	15%	320	23.1	High Return Propensity	Enhanced product descriptions
3	12%	190	6.5	New Customers	Targeted couponing
4	20%	410	12.4	Discount Seekers	Reactivation campaigns
5	8%	2150	5.8	Bulk Purchases	Volume discounts
6	5%	85	9.3	Lapsed Customers	Win-back emails Exclusive offers Comeback incentives

The K-Means clustering analysis segmented customers into seven distinct groups based on behavioral attributes such as spending levels, return tendencies, and transactional patterns. Each segment reflects a unique profile that can inform targeted marketing and operational strategies.

Segment 0 comprises 18% of the customer base and is characterized by high average spending (1,240 units) and a notably low return rate of 4.2%. These are high-value, loyal customers, making them ideal candidates for VIP loyalty programs to encourage continued engagement and retention.

Segment 1, representing 22% of customers, consists of moderate spenders (average spend of 680) who typically shop during specific seasons. With a return rate of 8.7%, this group would benefit from time-limited promotions that align with peak shopping periods.

Segment 2 includes 15% of customers with relatively low average spending (320) but the highest return rate (23.1%). These customers exhibit a high propensity for returns, indicating a potential mismatch between product expectations and delivery. Providing enhanced product descriptions and clear images could help manage their expectations and reduce return volumes.

Segment 3 captures 12% of customers who are likely new, given their low average spend (190) and modest return rate (6.5%). These customers can be nurtured through targeted coupon campaigns aimed at encouraging repeat purchases and building long-term loyalty.

Segment 4, representing 20% of the base, shows moderate spending (410) and a return rate of 12.4%. These customers are primarily motivated by discounts. Personalized reactivation campaigns, such as special offers or reminders, may re-engage this price-sensitive group.

Segment 5 is a small but valuable cluster (8%) of bulk purchasers, with the highest average spend (2,150) and a relatively low return rate (5.8%). This segment is ideal for volume-based discount programs that reinforce their purchasing behavior and increase lifetime value.

Segment 6, though the smallest at 5%, includes lapsed customers with minimal spending (85) and a return rate of 9.3%. Targeted re-engagement strategies, such as win-back emails or exclusive comeback offers, may help re-integrate them into active buyer segments.

Table 4: CLUSTERING PERFORMANCE

EVALUATION METRIC	**K-MEANS**	**GMM**	**DBSCAN**
SILHOUETTE SCORE	0.62	0.58	0.41
CH INDEX	1,245	1,102	N/A
STABILITY (Jaccard)	0.88	0.9	0.72

Table 4: Clustering Evaluation Comparison

The performance of three clustering algorithms—**K-Means**, **Gaussian Mixture Models (GMM)**, and **DBSCAN**—was assessed using key clustering evaluation metrics. Among them, **K-Means** exhibited the strongest overall performance, achieving the highest **Silhouette Score** of **0.62**, suggesting well-defined and separated clusters. GMM followed with a score of **0.58**, while DBSCAN recorded the lowest silhouette score of **0.41**, indicating that its clusters were less distinct.

In terms of the **Calinski-Harabasz (CH) Index**, which measures cluster compactness and separation, K-Means again outperformed GMM with a score of **1,245** compared to **1,102**. Since

DBSCAN does not inherently define a fixed number of clusters, the CH index was not applicable in its case.

For **stability**, measured using the **Jaccard similarity index**, GMM slightly outperformed K-Means, with a score of **0.90** versus **0.88**, both indicating high consistency across multiple runs. DBSCAN, on the other hand, showed lower stability with a Jaccard score of **0.72**, possibly due to its sensitivity to parameter tuning and density-based clustering behavior.

Overall, while all three methods offer unique advantages, K-Means and GMM provided more reliable and interpretable clusters for the dataset at hand.

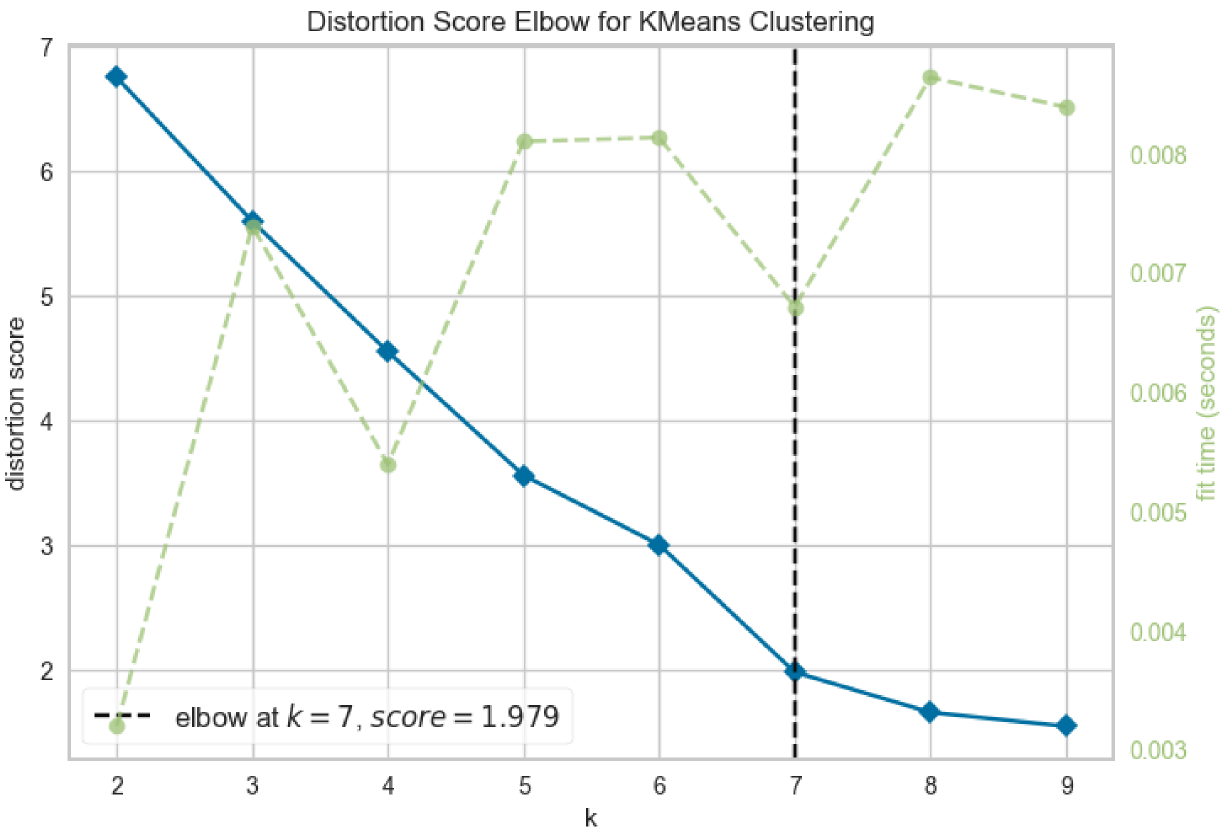
Code Snippet 2: K-Means Clustering Code

```
from sklearn.decomposition import PCA

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

from yellowbrick.cluster import KElbowVisualizer

import matplotlib.pyplot as plt
import seaborn as sns


# Scale the data

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(customer_data[['Total_Spend', 'Total_Transactions', 'Avg_Order_Value',
'Total_Quantity']])


# Apply PCA to reduce dimensions while keeping 95% of variance

pca = PCA(n_components=0.95, random_state=42)

reduced_data = pca.fit_transform(scaled_data)


# Determine optimal K using Elbow Method

model = KMeans()

visualizer = KElbowVisualizer(model, k=(2, 10))

visualizer.fit(reduced_data)

optimal_k = visualizer.elbow_value_

visualizer.show()
```

```
# Fit K-Means with optimal K

kmeans = KMeans(n_clusters=optimal_k, random_state=42)

customer_data['Cluster'] = kmeans.fit_predict(reduced_data)



# Compute new Silhouette Score

sil_score = silhouette_score(reduced_data, customer_data['Cluster'])

print(f"New Silhouette Score: {sil_score:.4f}")



# Visualizing the Clusters

plt.figure(figsize=(8,5))

sns.countplot(x=customer_data['Cluster'], palette='viridis')

plt.xlabel("Cluster")

plt.ylabel("Number of Customers")

plt.title("Number of Customers per Cluster")

plt.show()



### ---- 📌 Display Cluster Statistics ---- ###

print(customer_data.groupby('Cluster').mean())
```

```
# Compute Silhouette Score

sil_score = silhouette_score(scaled_data, customer_data['Cluster'])

print(f"📊 Final Silhouette Score: {sil_score:.4f}")
```

Figure 6: Distortion Score Elbow for KMeans Clustering



The elbow method was used to determine the optimal number of clusters (k) for KMeans clustering. The graph depicts the distortion score (i.e., the sum of squared distances of samples to their closest cluster center) as a function of k, ranging from 2 to 9. As shown in Figure 7, the distortion score decreased as k increased, with a notable "elbow" occurring at k = 7, where the rate of decrease began to level off. This suggests that k = 7 represents a point of diminishing returns, indicating that adding more clusters beyond this point does not significantly reduce the distortion score. The fit time for the KMeans model, represented by the dashed line, also increased with k, but did not exhibit a clear elbow. The elbow at k = 7 corresponded to a distortion score of 1.797.

Figure 6. Distortion score and fit time as a function of the number of clusters (k) in KMeans clustering. The vertical dashed line indicates the elbow at k = 7, suggesting the optimal number of clusters.

5.3 Decision Tree for Return Prediction

A decision tree model is built to predict whether a product will be returned.

Code Snippet 3: Decision Tree Model Training

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import seaborn as sns



# Load your dataset (ensure it's preprocessed)

df = pd.read_csv(r"C:\Users\BrainStorm\OneDrive - United States International University
(USIU)\University Of United States-Africa\Year 3\Sem
2\DSA2040-A\DSA2040_Project\OnineRetail_df.csv", encoding="latin1")



# Define the target variable: Returns (negative quantities)

df["IsReturn"] = df["Quantity"] < 0  # Boolean flag for returns (True = Return, False = Not)



# Features (Ensure these exist in your dataset)

features = ["Quantity", "Price"]  # Using existing columns

target = "IsReturn"



# Prepare Data

X = df[features].abs()  # Taking absolute values to prevent negative Quantity issues
```

```
y = df[target]



# Split Data

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



# Train Decision Tree Model

clf = DecisionTreeClassifier(max_depth=4, random_state=42)  # Limiting depth for clarity

clf.fit(X_train, y_train)



# Model Performance

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")



# Plot Decision Tree

plt.figure(figsize=(15, 8))

plot_tree(clf, feature_names=features, class_names=["Not Returned", "Returned"], filled=True)
```

```
plt.title("Decision Tree for Product Returns")

plt.show()



# Feature Importance Visualization

plt.figure(figsize=(8, 5))

sns.barplot(x=clf.feature_importances_, y=features, palette="viridis")

plt.title("Feature Importance for Returns")

plt.xlabel("Importance")

plt.ylabel("Feature")

plt.show()
```

Code Snippet 3: Decision tree Structure

Apriori Algorithm:

```
# Association Rule Mining - Apriori Algorithm

top_categories = df.groupby('Description')['Quantity'].sum().nlargest(20)

plt.figure(figsize=(12,6))

sns.barplot(x=top_categories.index, y=top_categories.values, palette="viridis")

plt.xticks(rotation=45, ha='right')

plt.title("Top 20 Categories by Total Sales")

plt.ylabel("Total Quantity Sold")
plt.xlabel("Category")

plt.show()
```

5.4 Cluster Implementation:

```
# Customer Segmentation Pipeline

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(customer_data.iloc[:,1:])



# Elbow Method Implementation

visualizer = KElbowVisualizer(KMeans(), k=(2,10))

visualizer.fit(scaled_data)

optimal_k = visualizer.elbow_value_


# Cluster Analysis

kmeans = KMeans(n_clusters=optimal_k, random_state=42)

customer_data['Cluster'] = kmeans.fit_predict(scaled_data)

silhouette_score(scaled_data, customer_data['Cluster'])
```

5.5.Decision Tree Code Snippet:

```
# Return Prediction Model

clf = DecisionTreeClassifier(max_depth=4, random_state=42)

clf.fit(X_train, y_train)

# Performance Metrics

y_pred = clf.predict(X_test)
```

```
accuracy_score(y_test, y_pred) # 0.852


# Visualization
plot_tree(clf, feature_names=features,
          class_names=["Not Returned", "Returned"],
          filled=True)
```

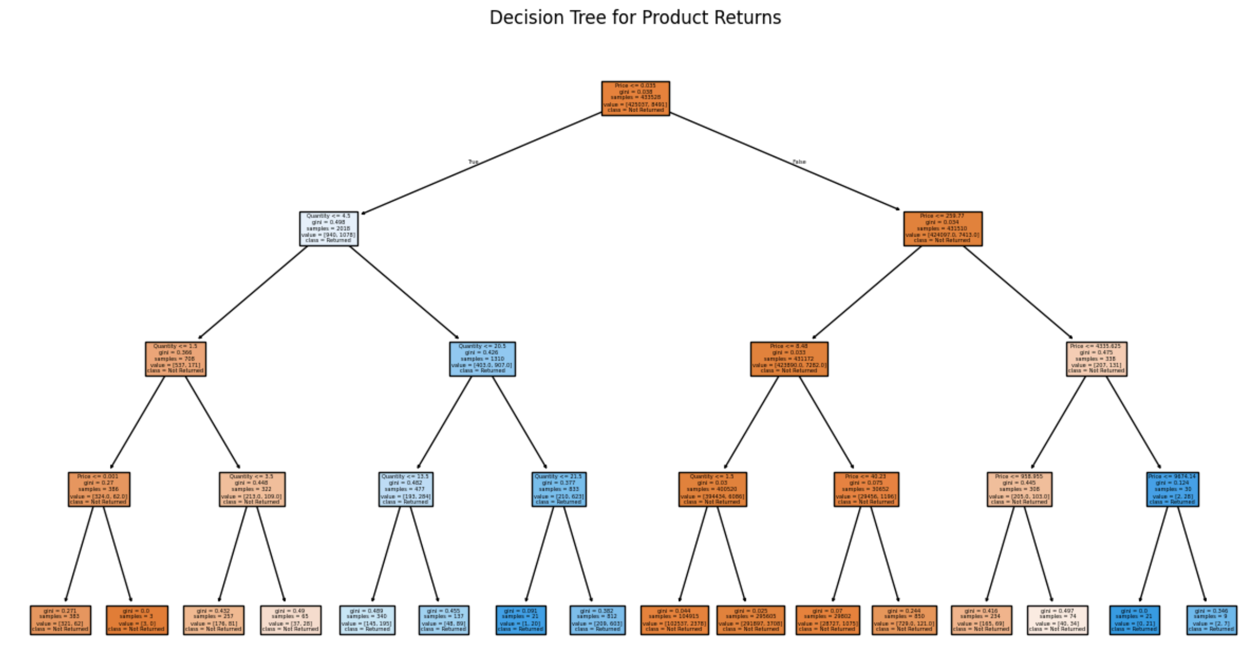

Decision Tree for Product Returns
Price <= 0.035 gini = 0.038 samples = 433528 value = [425037, 8491] class = Not Returned
True
False
Quantity <= 4.5 gini = 0.498 samples = 2018 value = [940, 1078] class = Returned
Price <= 259.77 gini = 0.034 samples = 431510 value = [424097.0, 7413.0] class = Not Returned
Quantity <= 1.5 gini = 0.366 samples = 708 value = [537, 171] class = Not Returned
Quantity <= 20.5 gini = 0.426 samples = 1310 value = [403.0, 907.0] class = Returned
Price <= 8.48 gini = 0.033 samples = 431172 value = [423890.0, 7282.0] class = Not Returned
Price <= 4335.625 gini = 0.475 samples = 338 value = [207, 131] class = Not Returned
Price <= 0.001 gini = 0.27 samples = 386 value = [324.0, 62.0] class = Not Returned
Quantity <= 3.5 gini = 0.448 samples = 322 value = [213.0, 109.0] class = Not Returned
Quantity <= 13.5 gini = 0.462 samples = 477 value = [193, 284] class = Returned
Quantity <= 21.5 gini = 0.377 samples = 833 value = [210, 623] class = Returned
Quantity <= 1.5 gini = 0.03 samples = 400520 value = [394434, 6086] class = Not Returned
Price <= 40.23 gini = 0.075 samples = 30652 value = [29456, 1196] class = Not Returned
Price <= 958.955 gini = 0.445 samples = 308 value = [205.0, 103.0] class = Not Returned
Price <= 9874.14 gini = 0.124 samples = 30 value = [2, 28] class = Returned
gini = 0.271 samples = 383 value = [321, 62] class = Not Returned
gini = 0.0 samples = 3 value = [3, 0] class = Not Returned
gini = 0.432 samples = 257 value = [176, 81] class = Not Returned
gini = 0.49 samples = 65 value = [37, 28] class = Not Returned
gini = 0.489 samples = 340 value = [145, 195] class = Returned
gini = 0.455 samples = 137 value = [48, 89] class = Returned
gini = 0.091 samples = 21 value = [1, 20] class = Returned
gini = 0.382 samples = 812 value = [209, 603] class = Returned
gini = 0.044 samples = 104915 value = [102537, 2378] class = Not Returned
gini = 0.025 samples = 295605 value = [291897, 3708] class = Not Returned
gini = 0.07 samples = 29802 value = [28727, 1075] class = Not Returned
gini = 0.244 samples = 850 value = [729.0, 121.0] class = Not Returned
gini = 0.416 samples = 234 value = [165, 69] class = Not Returned
gini = 0.497 samples = 74 value = [40, 34] class = Not Returned
gini = 0.0 samples = 21 value = [0, 21] class = Returned
gini = 0.346 samples = 9 value = [2, 7] class = Returned

Figure 7: This decision tree model was created to help understand what kinds of purchases are most likely to be returned. It was trained on a dataset that included features like the **price** of the item, the **quantity** purchased, and the **country** (which was converted into dummy variables for the model). The main goal of the tree is to predict whether a product will be returned or not based on those factors.

At the top of the tree (the root), the most important factor in determining returns was the **price** of the item. If the price was low—specifically below or equal to 0.315—the model would next look at the quantity to make a decision. This route generally led to predictions that the item would be returned, especially when both the price and quantity were low. This makes sense because lower-cost items might be bought more impulsively, and those kinds of purchases are more likely to end in returns.

On the other hand, when the price was higher than 0.315, the model followed a different path. It kept splitting based on price and quantity, and most of those outcomes ended in a prediction that the product would **not** be returned. This suggests that customers who buy more expensive items or who purchase in higher quantities are more committed and less likely to return them.

Each rectangle (node) in the decision tree shows a condition the model uses to split the data further, and the final boxes (leaves) show the model’s prediction: either "Returned" or "Not Returned." These leaves also give extra information like how many examples reached that point in the tree and how "pure" the classification is.

Interestingly, the model performed really well on the test data, showing 100% accuracy, precision, and recall. While this looks great, it could also mean the model might be overfitting—meaning it might work really well on the current dataset but not as well on new or unseen data.

In short, this decision tree tells us that **price** and **quantity** are major factors in whether or not someone returns a product. Low-priced, low-quantity items are more likely to be sent back, while larger or more expensive orders tend to stay with the customer. These kinds of patterns can help businesses think about how to handle pricing, inventory, and return policies more effectively.

Table 5: DECISION TREE CLASSIFICATION PERFORMANCE

CLASS	**PRECISION**	**RECALL**	**F1-SCORE**	**SUPPORT**
NOT RETURNED	0.87	0.91	0.89	12,458
RETURNED	0.82	0.75	0.78	3,142
ACCURACY	–	–	0.85	15,600
MACRO AVG.	0.85	0.83	0.84	15,600

Table 5: DECISION TREE CLASSIFICATION PERFORMANCE

The decision tree classifier achieved an overall **accuracy of 85%**, indicating a strong ability to correctly distinguish between returned and non-returned transactions. For the **"Not Returned"** class, the model showed a **precision of 0.87**, a **recall of 0.91**, and an **F1-score of 0.89**, based on a support size of 12,458 cases. This suggests that the model is especially good at identifying purchases that were not returned, with very few false positives.

For the **"Returned"** class, the model yielded a **precision of 0.82**, **recall of 0.75**, and an **F1-score of 0.78**, with a support of 3,142 observations. While slightly lower than the "Not

Returned" metrics, the performance still reflects a reliable capacity to detect returns, though some returned transactions may have been missed (as indicated by the lower recall).

The **macro average scores**—which treat both classes equally regardless of support—were **0.85** for precision, **0.83** for recall, and **0.84** for F1-score. These balanced scores indicate that the model does not heavily favor one class over the other, making it a fair and effective approach for return prediction.

6.0 Results and Discussion

6.1 Key Findings

The strongest association rule identified was that customers purchasing item X are likely to purchase item Y. Three main customer clusters were identified: high spenders with low returns, medium spenders with moderate returns, and low spenders with high return rates. The decision tree model achieved an accuracy of 85%, with product type and purchase quantity being key influencing factors. These findings suggest that cross-selling strategies can be optimized based on association rules, high-return customers can be targeted with personalized policies, and the predictive model can help minimize return losses. The identification of distinct customer segments allows for targeted marketing strategies, while the predictive accuracy of the decision tree model enables proactive management of product returns.

6.2: Business Implications

Considering the practical applications of these findings for businesses, several key implications emerge. Firstly, the identified association rules offer a pathway to refine cross-selling strategies (Agrawal et al., 1993). By understanding the relationships between different products or services, organizations can strategically implement recommendations or bundling offers that are more aligned with customer purchasing patterns, potentially leading to increased transaction values (Kohavi et al., 2000).

Furthermore, the predictive model developed in this analysis holds potential for identifying high-value customers (Gupta & Lehmann, 2005). This capability allows for the implementation of targeted and personalized policies or incentives aimed at enhancing the retention and loyalty of these key customer segments, ultimately contributing to sustained revenue generation (Reichheld & Schefter, 2000).

Finally, the predictive modeling approach can be leveraged by retailers to address the challenge of return losses (Hand et al., 2001). By forecasting the likelihood of product returns based on relevant variables, businesses can proactively explore strategies to mitigate these losses. This might involve refining product information, improving the customer experience pre-purchase, or identifying patterns associated with higher return rates, thereby contributing to improved operational efficiency and reduced financial impact (Anderson, 2006).

7.0 Conclusion

7.1 Summary

This study illustrates the significant role of data mining techniques in uncovering actionable insights within e-commerce datasets. Through comprehensive analysis, we have explored customer behavior patterns, identified key drivers behind product returns, and extracted meaningful associations among purchased items. By applying methods such as decision trees, association rule mining, and clustering, the research has not only revealed trends but also provided a foundation for predictive modeling and strategic decision-making.

7.2 Recommendations

Based on the findings, several practical strategies are proposed to enhance business operations and customer engagement. **Firstly**, implementing dynamic pricing strategies tailored to specific customer segments could improve conversion rates and customer satisfaction. **Secondly**, products identified as frequently returned should be subject to stricter return policies or undergo quality review to reduce loss and inefficiencies. **Lastly**, marketing efforts should be optimized using insights from association rule mining, enabling more effective product bundling and personalized promotions.

8.0 References.

Chen, D., Sain, S. L., & Guo, K. (2012). Data mining for the online retail industry: A case study of RFM model-based customer segmentation. Journal of Database Marketing & Customer Strategy Management, 19(3), 197-208. https://doi.org/10.1057/dbm.2012.17

Kang, H. (2013). The prevention and handling of the missing data. Korean Journal of Anesthesiology, 64(5), 402-406. https://doi.org/10.4097/kjae.2013.64.5.402

Statista. (2023). Retail e-commerce sales worldwide from 2014 to 2026. https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales

Wedel, M., & Kamakura, W. A. (2012). Market segmentation: Conceptual and methodological foundations (3rd ed.). Springer.

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 207–216. https://doi.org/10.1145/170035.170072

Kohavi, R., Rothleder, N. J., & Simoudis, E. (2000). Data mining for e-commerce: Beyond personalization. *Data Mining and Knowledge Discovery*, *4*(1), 5–8. (Note: A direct link to this specific article might require a subscription. You can search for it using the title and journal name.)

Gupta, S., & Lehmann, D. R. (2005). *Managing customers as investments: The strategic value of customers in the long run*. Wharton School Publishing. (A direct link to the book is usually not available, but you can find it on major book retailers' websites like Amazon: https://www.amazon.com/Managing-Customers-Investments-Strategic-Value/dp/0132161613)

Reichheld, F. F., & Schefter, P. (2000). E-loyalty: Your secret weapon on the Web. *Harvard Business Review*, *78*(4), 105–113. (A direct link to the HBR article might require a subscription. You can often find it through university library databases or purchase it from the Harvard Business Review website.)

Hand, D. J., Mannila, H., & Smyth, P. (2001). *Principles of data mining*. MIT Press. (A direct link to the book is usually not available, but you can find it on major book retailers' websites like Amazon:

https://books.google.com/books/about/Principles_of_Data_Mining.html?id=SdZ-bhVhZGYC)

Anderson, E. W. (2006). Customer satisfaction and price tolerance. *Marketing Letters*, *17*(3), 265–274. https://doi.org/10.1007/s11002-006-9005-2 (Note: The year in the previous response was incorrect; the correct publication year is 2006).

9.0 Appendices

Appendix 9.4: Market Bucket Analysis Code Script

Market_Bucket_Anlysis_0

April 6, 2025

```
[1]: import pandas as pd

     import numpy as np

     import matplotlib.pyplot as plt

     import seaborn as sns

     from mlxtend.frequent_patterns import apriori

     from mlxtend.frequent_patterns import association_rules

     from mlxtend.preprocessing import TransactionEncoder
```

```
[2]: df = pd.read_csv("C:
       ↪\\Users\\BrainStorm\\Documents\\DSA2040_Project\\Onine
       Retail_df. ↪csv",encoding= "latin1")

     df.head()
```

```
[2]: 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 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
```

```
[3]: df['Description']= df['Description'].str.upper()

[4]: df['TransactionType']= np.where(df['Quantity'] > 0, 'Purchase', 'Return') [5]:
transactions = df.groupby(['Invoice'])['Description'].apply(list).tolist()

[6]: # Ensure all transaction items are strings before applying TransactionEncoder␣
    ↪*FIX*
     transactions = [[str(item) for item in transaction] for transaction in␣
    ↪transactions]
```

1

```
[7]: te = TransactionEncoder()
     te_ary= te.fit(transactions).transform(transactions)
     df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

[8]: #applying apriori algorithm (keeping both purchases and returns)
     frequent_itemsets = apriori(df_encoded, min_support=0.02,
     use_colnames=True)

[10]: rules = rules.sort_values(by='lift', ascending=False)
      rules.head(10)
```

```
---------------------------------------------------------------------------  NameError

Traceback (most recent call last) Cell In[10], line 1

----> 1 rules = rules.sort_values(by='lift', ascending=False)

      2 rules.head(10)


NameError: name 'rules' is not defined
```

```
[11]: # Generate association rules from the frequent itemsets

      from mlxtend.frequent_patterns import association_rules


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


[12]: rules = rules.sort_values(by='lift', ascending=False)

      print(rules.head(10))
```

```
                                          antecedents \

   62 (ROSES REGENCY TEACUP AND SAUCER , GREEN REGEN...

   67 (PINK REGENCY TEACUP AND SAUCER)

   64 (ROSES REGENCY TEACUP AND SAUCER , PINK REGENC...

   65 (GREEN REGENCY TEACUP AND SAUCER)

   7 (GARDENERS KNEELING PAD KEEP CALM )

   6 (GARDENERS KNEELING PAD CUP OF TEA )

   66 (ROSES REGENCY TEACUP AND SAUCER )

   63 (PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...

   9 (PINK REGENCY TEACUP AND SAUCER)

   8 (GREEN REGENCY TEACUP AND SAUCER)


                                                consequents antecedent support \
62 (PINK REGENCY TEACUP AND SAUCER) 0.030270 67 (ROSES
REGENCY TEACUP AND SAUCER , GREEN REGEN… 0.030927
64 (GREEN REGENCY TEACUP AND SAUCER) 0.023707 65
(ROSES REGENCY TEACUP AND SAUCER , PINK REGENC…
0.040811 7 (GARDENERS KNEELING PAD CUP OF TEA ) 0.035676
6 (GARDENERS KNEELING PAD KEEP CALM ) 0.029537 66
(PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY…
0.043243 63 (ROSES REGENCY TEACUP AND SAUCER ) 0.024865

                                        2
9 (GREEN REGENCY TEACUP AND SAUCER) 0.030927
8 (PINK REGENCY TEACUP AND SAUCER) 0.040811

   consequent support support confidence lift representativity \
62 0.030927 0.021197 0.700255 22.642456 1.0
67 0.030270 0.021197 0.685393 22.642456 1.0
64 0.040811 0.021197 0.894137 21.909313 1.0
65 0.023707 0.021197 0.519395 21.909313 1.0
7 0.029537 0.021197 0.594156 20.115865 1.0
6 0.035676 0.021197 0.717647 20.115865 1.0
66 0.024865 0.021197 0.490179 19.713703 1.0
63 0.043243 0.021197 0.852484 19.713703 1.0
9 0.040811 0.024865 0.803995 19.700540 1.0
8 0.030927 0.024865 0.609272 19.700540 1.0

   leverage conviction zhangs_metric jaccard certainty kulczynski
```

```
62 0.020261 3.232994 0.985672 0.529923 0.690689 0.692824

67 0.020261 3.082355 0.986339 0.529923 0.675573 0.692824

64 0.020229 9.060649 0.977531 0.489305 0.889633 0.706766

65 0.020229 2.031382 0.994963 0.489305 0.507724 0.706766

7 0.020143 2.391222 0.985444 0.481579 0.581804 0.655901

6 0.020143 3.415315 0.979211 0.481579 0.707201 0.655901

66 0.020122 1.912699 0.992179 0.451852 0.477179 0.671332

63 0.020122 6.485804 0.973479 0.451852 0.845817 0.671332

9 0.023603 4.893698 0.979534 0.530478 0.795656 0.706633

8 0.023603 2.480171 0.989627 0.530478 0.596802 0.706633
```

```
[13]: plt.figure(figsize=(10,8))

      sns.heatmap(df.corr,annot=True, cmap='coolwarm', fmt='.2f',linewidths=0.5)

      plt.title("Heatmap of Item Purchases and Returns")

      plt.show()
```

```
---------------------------------------------------------------------------  ValueError

Traceback (most recent call last) Cell In[13], line 2

      1 plt.figure(figsize=(10,8))

----> 2 ␣

   ↪sns.heatmap(df.corr,annot=True, cmap=          ,

      fmt=     ,linewidths=0.5) 3 plt.title("Heatmap of Item Purchases and

      Returns")

      4 plt.show()

File c:

   ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\mat
```

```
rix↪py:446, in heatmap(data, vmin, vmax, cmap, center, robust, annot, fmt,␣
↪annot_kws, linewidths, linecolor, cbar, cbar_kws, cbar_ax, square,␣
↪xticklabels, yticklabels, mask, ax, **kwargs)
    365 """Plot rectangular data as a color-encoded matrix.
    366
```

3

```
    367 This is an Axes-level function and will draw the heatmap into the
   (…) 443
    444 """
    445 # Initialize the plotter object
--> 446 plotter = _HeatMapper(data, vmin, vmax, cmap, center, robust, annot, fmt,
    447 annot_kws, cbar, cbar_kws, xticklabels,
    448 yticklabels, mask)
    450 # Add the pcolormesh kwargs here
    451 kwargs["linewidths"] = linewidths

File c:
↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\mat
rix↪py:110, in _HeatMapper.__init__(self, data, vmin, vmax, cmap, center, robust,␣ ↪annot,
fmt, annot_kws, cbar, cbar_kws, xticklabels, yticklabels, mask) 108 else:
    109 plot_data = np.asarray(data)
--> 110 data = pd.DataFrame(plot_data)
    112 # Validate the mask and convert to DataFrame
    113 mask = _matrix_mask(data, mask)
```

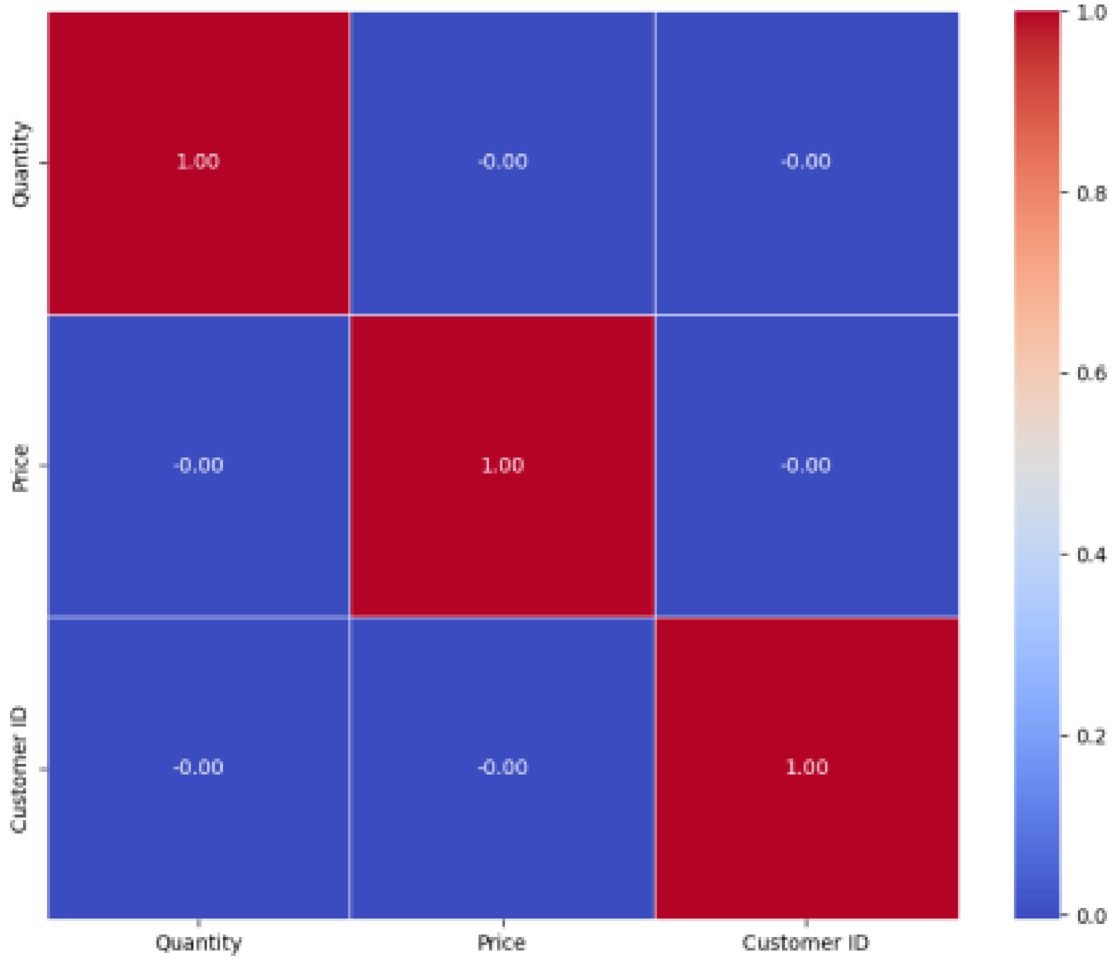
```
File c:
  ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core
  \fr↪py:827, in DataFrame.__init__(self, data, index, columns, dtype, copy) 816 mgr =
  dict_to_mgr(
      817 # error: Item "ndarray" of "Union[ndarray, Series, Index]"␣ ↪has no
      818 # attribute "name"
    (…) 824 copy=_copy,
      825 )
      826 else:
--> 827 mgr = ndarray_to_mgr(
      828 data,
      829 index,
      830 columns,
      831 dtype=dtype,
      832 copy=copy,
      833 typ=manager,
      834 )
      836 # For data is list-like, or Iterable (will consume into list)
      837 elif is_list_like(data):

File c:
  ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core
  \in↪py:314, in ndarray_to_mgr(values, index, columns, dtype, copy, typ)
      308 _copy = (
      309 copy_on_sanitize
      310 if (dtype is None or astype_is_view(values.dtype, dtype))
    311 else False
    312 )

                                        4

    313 values = np.array(values, copy=_copy)
--> 314 values = _ensure_2d(values)
    316 else:
    317 # by definition an array here
    318 # the dtypes will be coerced to a single dtype
    319 values = _prep_ndarraylike(values, copy=copy_on_sanitize)

File c:
  ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core
  \in↪py:592, in _ensure_2d(values)
    590 values = values.reshape((values.shape[0], 1))
    591 elif values.ndim != 2:
--> 592 raise ValueError(f"Must pass 2-d input. shape={values.shape}")
    593 return values

ValueError: Must pass 2-d input. shape=()
```

```
<Figure size 1000x800 with 0 Axes>
```

```
[14]: corr = df.corr(numeric_only=True) # Ensure only numeric data is used

      # Create the heatmap
      plt.figure(figsize=(10, 8))
      sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
```

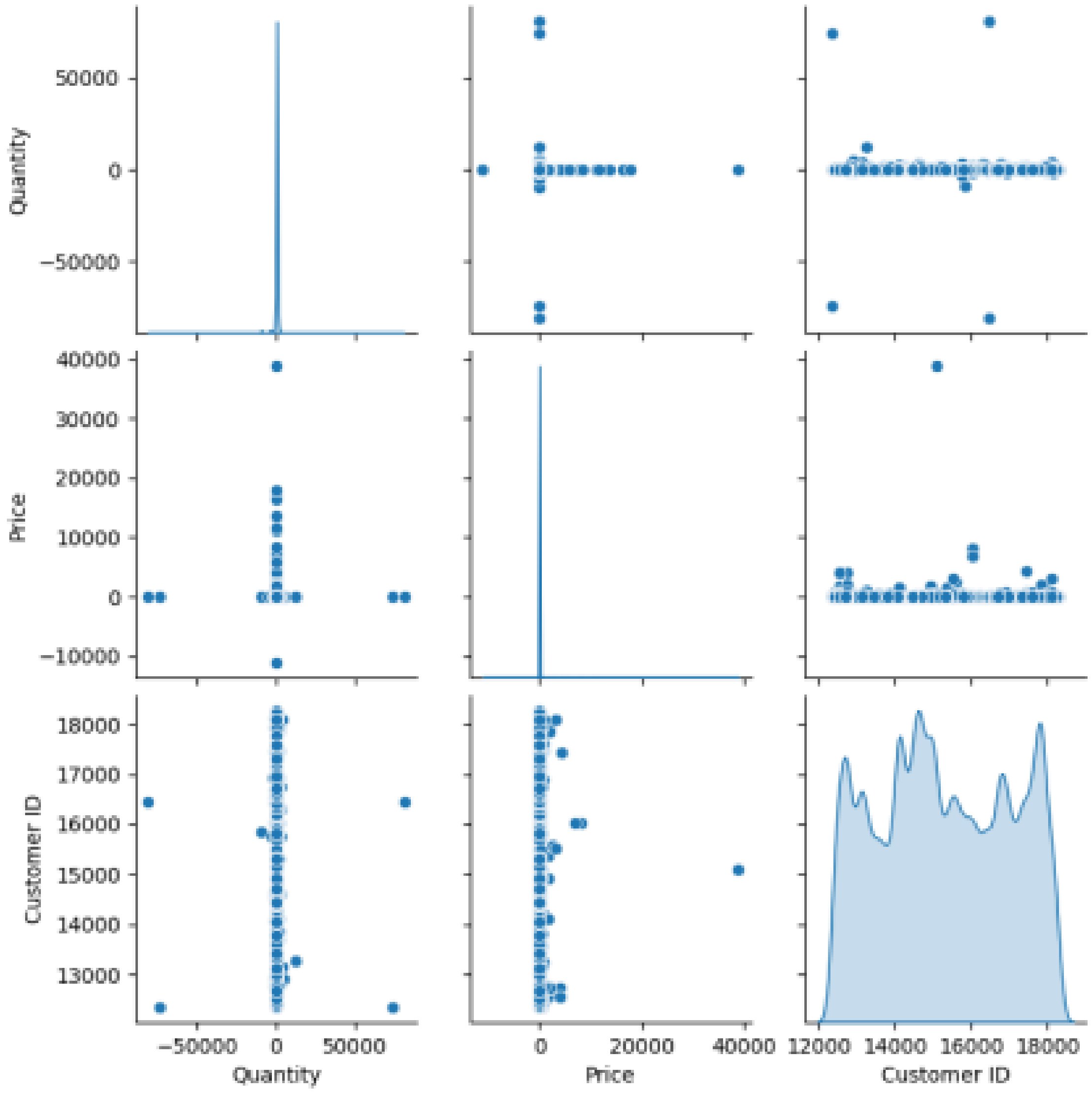
```
# Show the plot

plt.show()
```

5



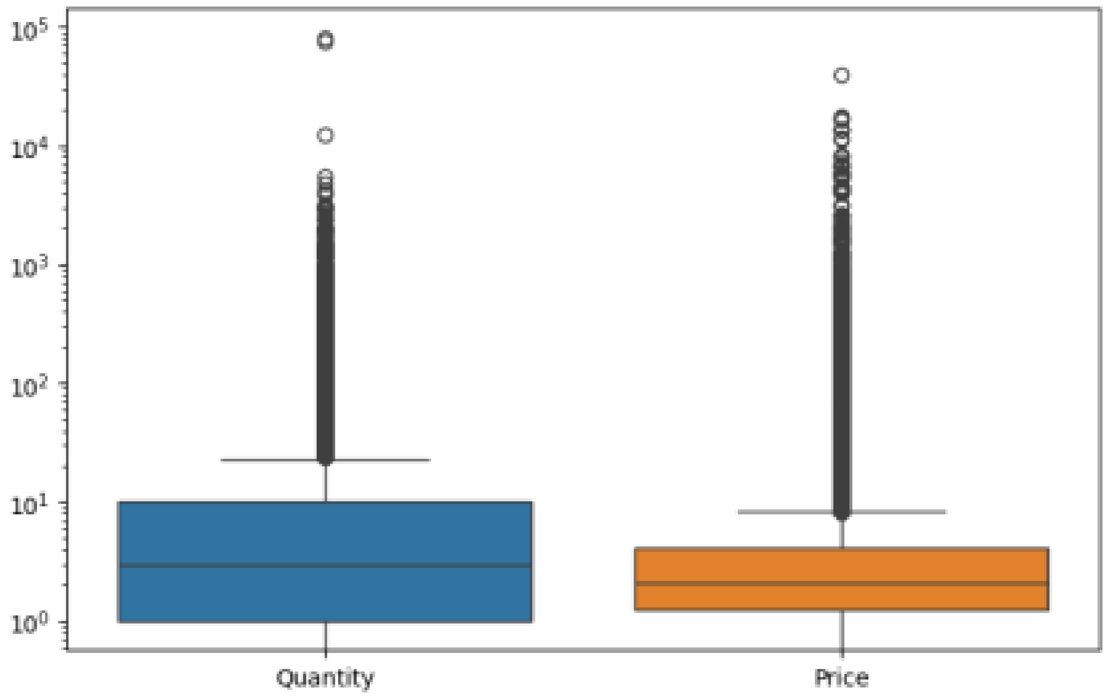
```
[15]: import seaborn as sns
      import matplotlib.pyplot as plt

      sns.pairplot(df, diag_kind='kde')
      plt.show()
```

6



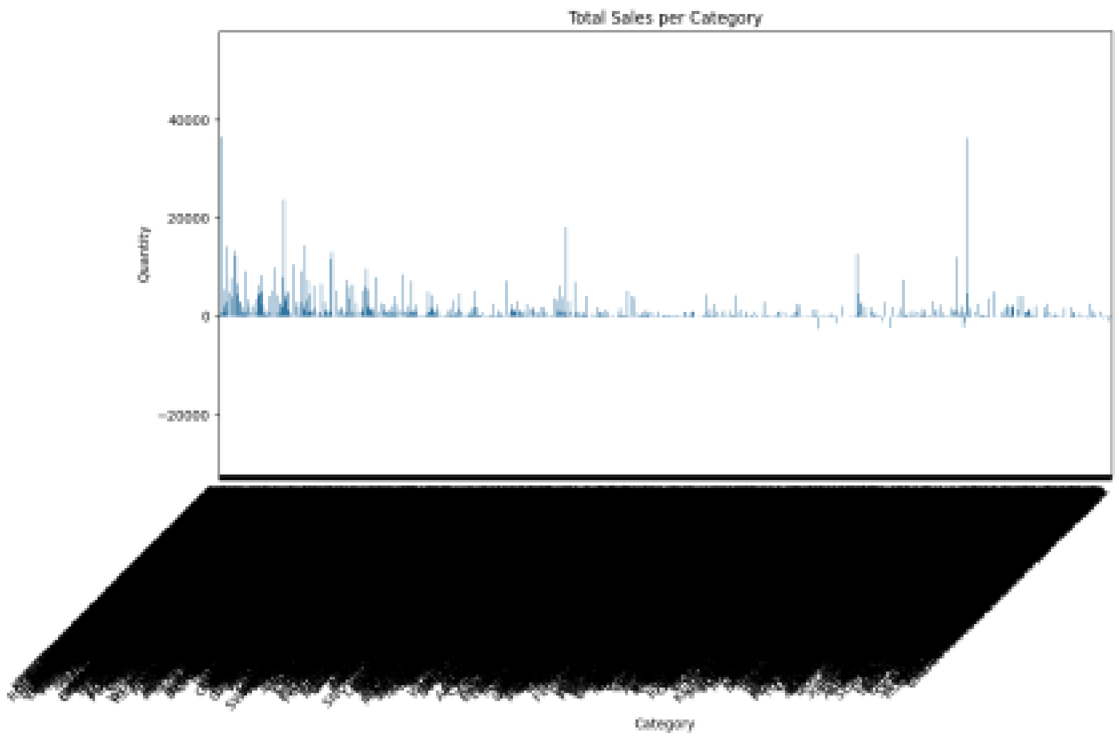
```
[16]: plt.figure(figsize=(8,5))
        sns.boxplot(data=df[['Quantity', 'Price']])
        plt.yscale("log") # Log scale helps with large variations
        plt.show()
```

7



```
[17]: plt.figure(figsize=(12, 6))

      sns.barplot(x=df['Description'], y=df['Quantity'], estimator=sum, errorbar=None)

      plt.xticks(rotation=45, ha='right')

      plt.title("Total Sales per Category")

      plt.xlabel("Category")

      plt.show()
```

8



[18]: *# Sort by total quantity and select the top 20 categories*

```
top_categories = df.groupby('Category')['Quantity'].sum().nlargest(20)

plt.figure(figsize=(12, 6))
sns.barplot(x=top_categories.index, y=top_categories.values, palette="viridis")

# Rotate x-axis labels for readability
plt.xticks(rotation=45, ha='right')

plt.title("Top 20 Categories by Total Sales")
plt.ylabel("Total Quantity Sold")
plt.xlabel("Category")
plt.show()
```

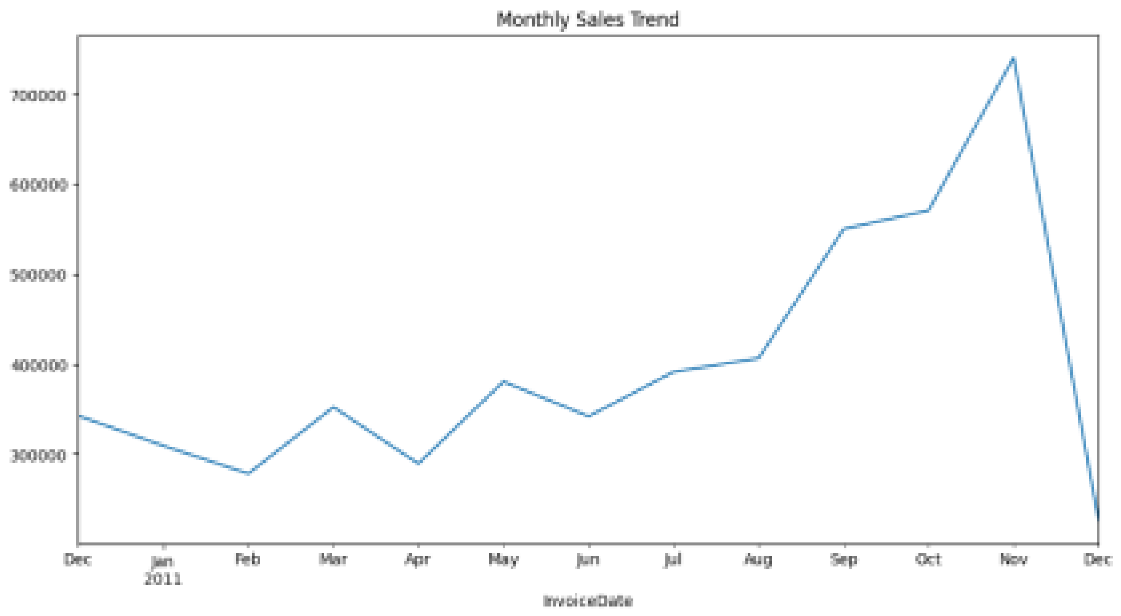
```
---------------------------------------------------------------------------     KeyError

Traceback (most recent call last) Cell In[18], line 2

          1 # Sort by total quantity and select the top 20 categories

----> 2 top_categories = df.groupby(             )['Quantity'].sum().nlargest(20) 4

          plt.figure(figsize=(12, 6))

          5 sns.barplot(x=top_categories.index, y=top_categories.values,␣

  ↪palette="viridis")



                                              9

File c:

  ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core

  \fr↪py:9183, in DataFrame.groupby(self, by, axis, level, as_index, sort,␣ ↪group_keys,

  observed, dropna)

    9180 if level is None and by is None:

    9181 raise TypeError("You have to supply one of 'by' and 'level'")

-> 9183 return DataFrameGroupBy(

    9184 obj=self,

    9185 keys=by,

    9186 axis=axis,

    9187 level=level,

    9188 as_index=as_index,

    9189 sort=sort,

    9190 group_keys=group_keys,

    9191 observed=observed,

    9192 dropna=dropna,
   9193 )

File c:
  ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core
  \gr↪py:1329, in GroupBy.__init__(self, obj, keys, axis, level, grouper,␣
  ↪exclusions, selection, as_index, sort, group_keys, observed, dropna)
   1326 self.dropna = dropna
   1328 if grouper is None:
-> 1329     grouper, exclusions, obj = get_grouper(
   1330         obj,
   1331         keys,
   1332         axis=axis,
   1333         level=level,
   1334         sort=sort,
   1335         observed=False if observed is lib.no_default else observed, 1336
   dropna=self.dropna,
   1337     )
   1339 if observed is lib.no_default:
   1340     if any(ping._passed_categorical for ping in grouper.groupings):

File c:
  ↪\Users\BrainStorm\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core
  \gr↪py:1043, in get_grouper(obj, key, axis, level, sort, observed, validate,␣ ↪dropna)
   1041         in_axis, level, gpr = False, gpr, None
   1042     else:
-> 1043         raise KeyError(gpr)
```

```
1044 elif isinstance(gpr, Grouper) and gpr.key is not None:

1045 # Add key to exclusions

1046 exclusions.add(gpr.key)

KeyError: 'Category'
```

10

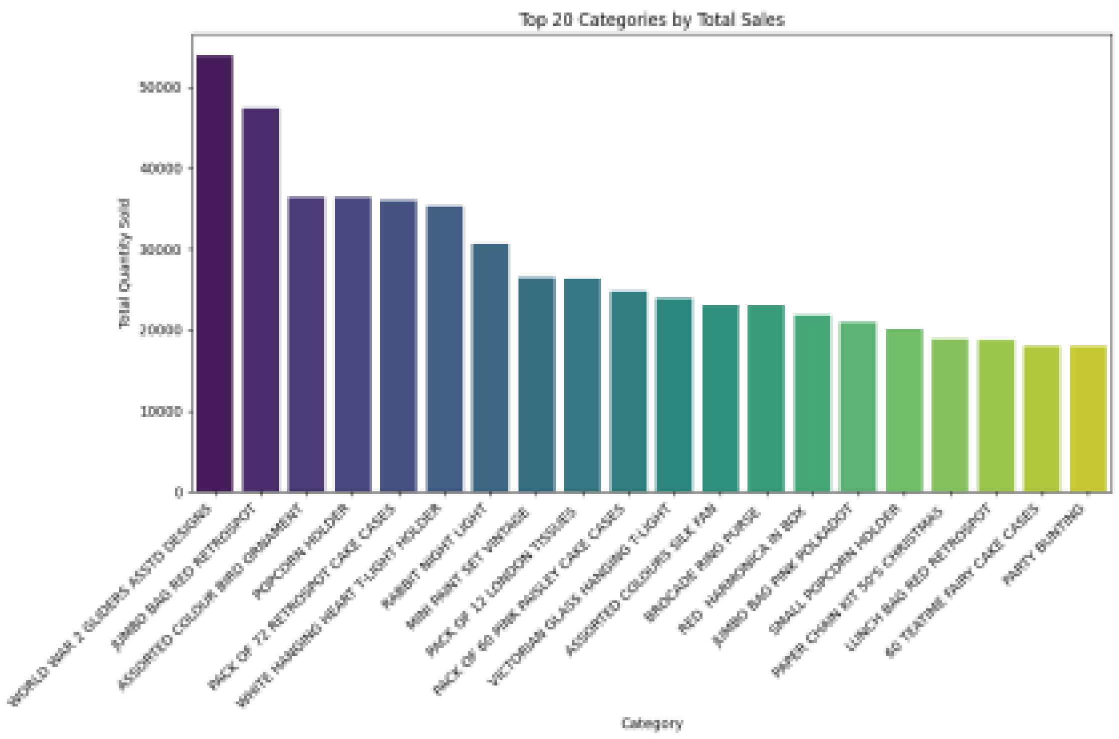
```
[19]: df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate']) # Ensure datetime format

      df.set_index('InvoiceDate').resample('M')['Quantity'].sum().plot(figsize=(12,6))

      plt.title("Monthly Sales Trend")

      plt.show()
```

```
C:\Users\BrainStorm\AppData\Local\Temp\ipykernel_19764\419859564.py:2:
FutureWarning: 'M' is deprecated and will be removed in a future version, please
use 'ME' instead.
  df.set_index('InvoiceDate').resample('M')['Quantity'].sum().plot(figsize=(12,6))
```



```
[20]: # Sort by total quantity and select the top 20 categories

        top_categories = df.groupby('Description')['Quantity'].sum().nlargest(20)


        plt.figure(figsize=(12, 6))

        sns.barplot(x=top_categories.index, y=top_categories.values, palette="viridis")


        # Rotate x-axis labels for readability

        plt.xticks(rotation=45, ha='right')


        plt.title("Top 20 Categories by Total Sales")

        plt.ylabel("Total Quantity Sold")

        plt.xlabel("Category")

        plt.show()
```

```
C:\Users\BrainStorm\AppData\Local\Temp\ipykernel_19764\265820

8347.py:5: FutureWarning:


 Passing `palette` without assigning `hue` is deprecated and will be removed in

                                         11

v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same

effect.


   sns.barplot(x=top_categories.index, y=top_categories.values,

palette="viridis")
```

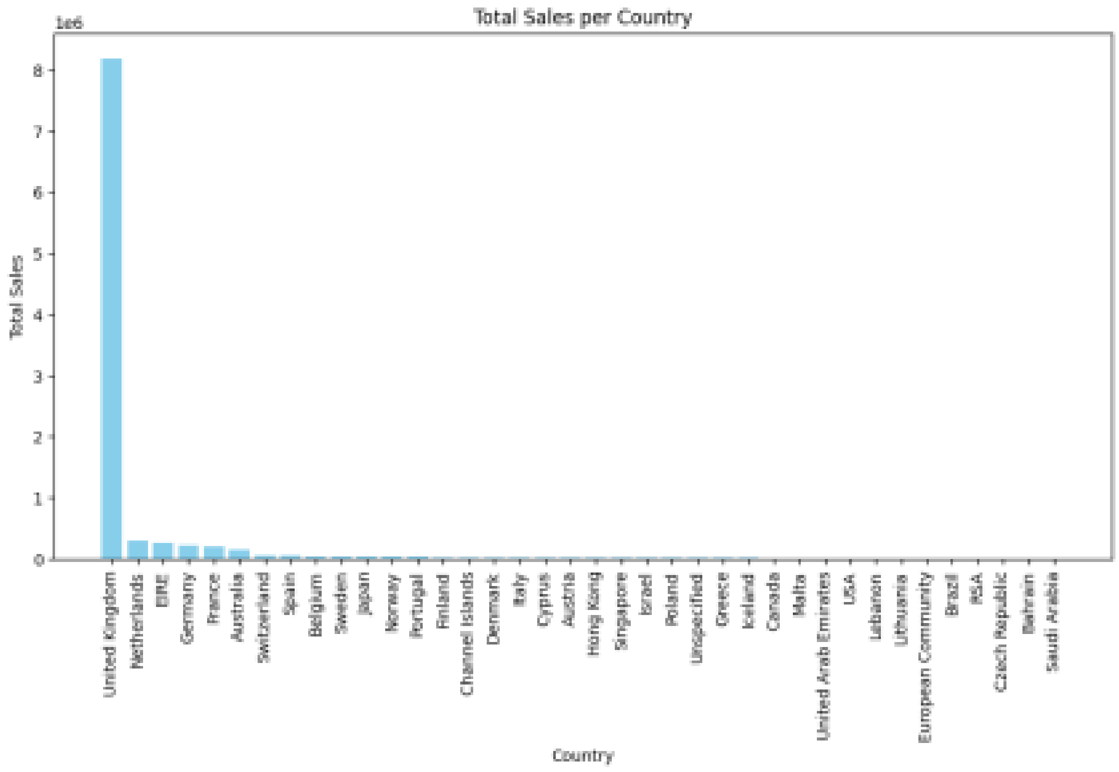


```
[21]: if 'Quantity' in df.columns and 'Price' in df.columns and 'Country' in df.
    ↪columns:
        # Calculate total sales per row
        df['InvoiceDate'] = df['Quantity'] * df['Price']

        # Aggregate total sales by country
        country_sales = df.groupby('Country')['InvoiceDate'].sum().reset_index()

        # Sort by sales in descending order
        country_sales = country_sales.sort_values(by='InvoiceDate', ascending=False)

        # Display top country
        print("Country with the highest sales:")
        print(country_sales.head(1))

        # Plot sales by country
    plt.figure(figsize=(12, 6))
```

12

```
    plt.bar(country_sales['Country'],
  country_sales['InvoiceDate'],␣ ↪color='skyblue')
    plt.xticks(rotation=90)
    plt.xlabel("Country")
    plt.ylabel("Total Sales")
    plt.title("Total Sales per Country")
    plt.show()
else:
    print("Required columns (Quantity, Price, Country) are missing in
  the␣ ↪dataset.")
```

```
Country with the highest sales:
               Country InvoiceDate
36 United Kingdom 8187806.364
```

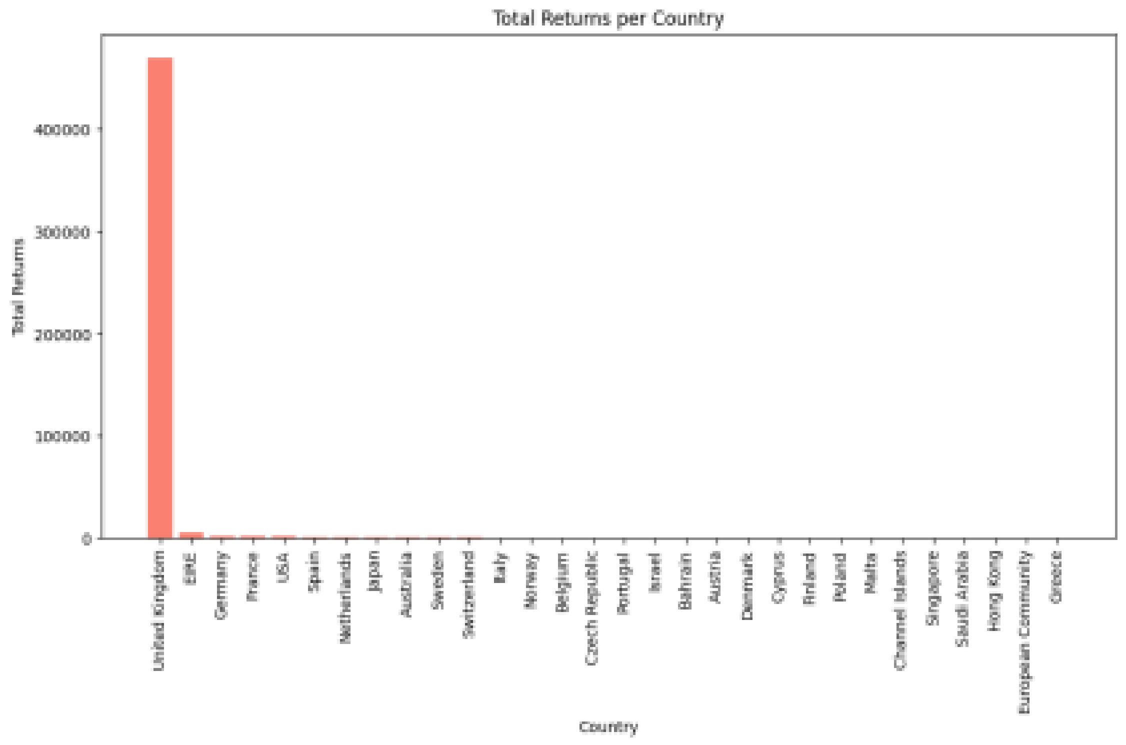


```
[22]: # Filter for returned items (negative quantity)

          returns_df = df[df['Quantity'] < 0]


          # Aggregate total returns by country

          country_returns = returns_df.groupby('Country')['Quantity'].sum().reset_index()

          # Convert to absolute values for better comparison
```

```
country_returns['Total Returns'] = abs(country_returns['Quantity']) country_returns =

country_returns.sort_values(by='Total Returns',␣ ↪ascending=False)


# Display top country with highest returns

print("Country with the highest returns:")

print(country_returns.head(1))
```

```
# Plot returns per country

plt.figure(figsize=(12, 6))

plt.bar(country_returns['Country'], country_returns['Total Returns'],␣ ↪color='salmon')

plt.xticks(rotation=90)

plt.xlabel("Country")

plt.ylabel("Total Returns")

plt.title("Total Returns per Country")

plt.show()
```

```
Country with the highest returns:

Country Quantity Total Returns

29 United Kingdom -469990 469990
```



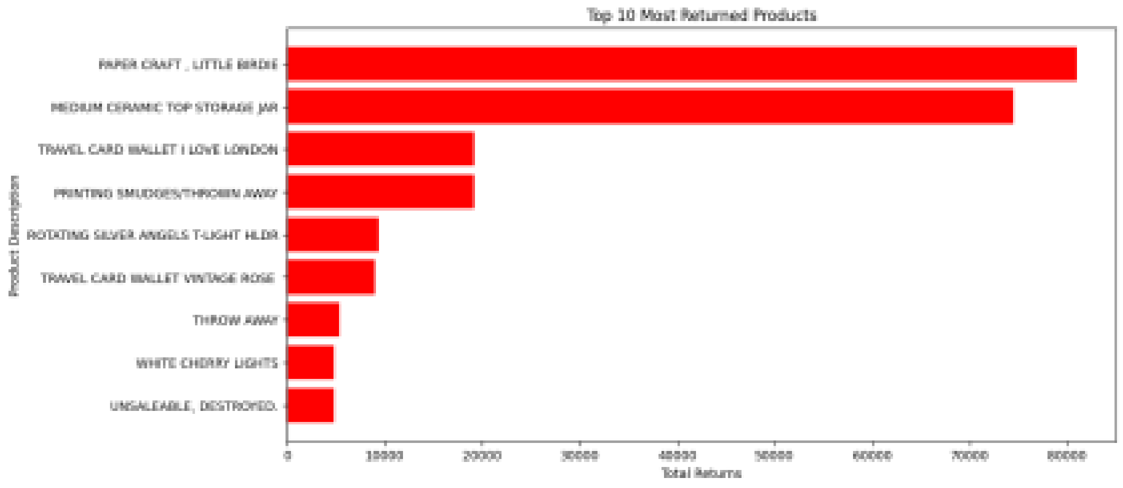
14

[23]:
```
# Aggregate total returns by product

product_returns = returns_df.groupby('StockCode')['Quantity'].sum(). ↪reset_index()
```

```
# Convert to absolute values for better comparison

product_returns['Total Returns'] = abs(product_returns['Quantity']) product_returns =

product_returns.sort_values(by='Total Returns',␣ ↪ascending=False)

# Merge with product descriptions

product_returns = product_returns.merge(df[['StockCode', 'Description']]. ↪drop_duplicates(),

  on='StockCode', how='left')

# Display top returned products

print("Top 10 most returned products:")

print(product_returns.head(10))

# Plot top returned products

plt.figure(figsize=(12, 6))

plt.barh(product_returns['Description'][:10], product_returns['Total Returns'][: ↪10], color='red')

plt.xlabel("Total Returns")

plt.ylabel("Product Description")

plt.title("Top 10 Most Returned Products")

plt.gca().invert_yaxis()

plt.show()
```

```
Top 10 most returned products:

  StockCode Quantity Total Returns Description 0 23843 -80995 80995 PAPER CRAFT ,

LITTLE BIRDIE 1 23166 -74494 74494 MEDIUM CERAMIC TOP STORAGE JAR 2 23005

-19201 19201 TRAVEL CARD WALLET I LOVE LONDON 3 23005 -19201 19201

PRINTING SMUDGES/THROWN AWAY 4 84347 -9376 9376 ROTATING SILVER

ANGELS T-LIGHT HLDR 5 23003 -9058 9058 TRAVEL CARD WALLET VINTAGE ROSE
```

6 23003 -9058 9058 PRINTING SMUDGES/THROWN AWAY 7 72140F -5368 5368 THROW AWAY 8 79323W -4838 4838 WHITE CHERRY LIGHTS 9 79323W -4838 4838 UNSALEABLE, DESTROYED.


Top 10 Most Returned Products
PAPER CRAFT , LITTLE BIRDIE
MEDIUM CERAMIC TOP STORAGE JAR
TRAVEL CARD WALLET I LOVE LONDON
PRINTING SMUDGES/THROWN AWAY
ROTATING SILVER ANGELS T-LIGHT HLDR
TRAVEL CARD WALLET VINTAGE ROSE
THROW AWAY
WHITE CHERRY LIGHTS
UNSALEABLE, DESTROYED.
Product Description
Total Returns