AIGC 5503 - Midterm

Daniel Mehta

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

Data Preprocessing & Exploration

Load Data

In [5]:
```
transactions = pd.read_csv('Transactions.csv',low_memory=False)
customers = pd.read_csv('Customers.csv')
products = pd.read_csv('Products.csv')
```

In [6]:
```
print("Transactions:\n", transactions.head())
print("Customers:\n", customers.head())
print("Products:\n", products.head())
```

```
Transactions:
   TransactionNo       Date ProductNo  Quantity  CustomerNo
0         536365  12/1/2018     22752         2     17850.0
1         536365  12/1/2018    85123A         6     17850.0
2         536365  12/1/2018     71053         6     17850.0
3         536365  12/1/2018    84029G         6     17850.0
4         536365  12/1/2018    84029E         6     17850.0
Customers:
    CustomerNo         Country  Gender  Age
0      17490.0  United Kingdom  Female   20
1      13069.0  United Kingdom  Female   18
2      12433.0          Norway    Male   18
3      13426.0  United Kingdom    Male   54
4      17364.0  United Kingdom    Male   48
Products:
   ProductNo  Price Product Category
0      22485  21.47         Clothing
1      22596  10.65      Electronics
2      23235  11.53         Clothing
3      23272  10.65           Beauty
4      23239  11.94         Clothing
```

Clean Data

In [8]:

```
print(transactions.info())
print(customers.info())
print(products.info())

print("-"*25)
print("Missing values:\n")
print("Transactions:\n", transactions.isnull().sum())
print("-"*25)
print("\nCustomers:\n", customers.isnull().sum())
print("-"*25)
print("\nProducts:\n", products.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 536350 entries, 0 to 536349
Data columns (total 5 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   TransactionNo  536350 non-null  object
 1   Date           536350 non-null  object
 2   ProductNo      536337 non-null  object
 3   Quantity       536350 non-null  int64
 4   CustomerNo     536282 non-null  float64
dtypes: float64(1), int64(1), object(3)
memory usage: 20.5+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4739 entries, 0 to 4738
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   CustomerNo  4738 non-null   float64
 1   Country     4739 non-null   object
 2   Gender      4739 non-null   object
 3   Age         4739 non-null   int64
dtypes: float64(1), int64(1), object(2)
memory usage: 148.2+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3768 entries, 0 to 3767
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ProductNo         3768 non-null   object
 1   Price             3768 non-null   float64
 2   Product Category  3768 non-null   object
dtypes: float64(1), object(2)
memory usage: 88.4+ KB
None
-------------------------
Missing values:

Transactions:
 TransactionNo     0
Date              0
ProductNo        13
Quantity          0
CustomerNo       68
dtype: int64
-------------------------
```

```
Customers:
 CustomerNo    1
Country       0
Gender        0
Age           0
dtype: int64
-------------------------

Products:
 ProductNo           0
Price               0
Product Category    0
dtype: int64
```

In [9]:
```
transactions_clean = transactions.dropna(subset=['ProductNo','CustomerNo'])

customers_clean = customers.dropna(subset=['CustomerNo'])

print(transactions_clean.isnull().sum())
print(customers_clean.isnull().sum())
```

```
TransactionNo    0
Date             0
ProductNo        0
Quantity         0
CustomerNo       0
dtype: int64
CustomerNo    0
Country       0
Gender        0
Age           0
dtype: int64
```

In [10]:
```
#convert CustomerNo to int
transactions_clean.loc[:, 'CustomerNo'] = transactions_clean['CustomerNo'].astype(int)
customers_clean.loc[:, 'CustomerNo'] = customers_clean['CustomerNo'].astype(int)
transactions_clean.loc[:, 'Date'] =pd.to_datetime(transactions_clean['Date'],errors='coerce')
```

Merge Datasets

In [12]:
```
#Merge product details
df = transactions_clean.merge(products, on='ProductNo', how='left')

#merge customer details
df = df.merge(customers_clean, on='CustomerNo', how='left')
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 536269 entries, 0 to 536268
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   TransactionNo     536269 non-null  object 
 1   Date              536269 non-null  object 
 2   ProductNo         536269 non-null  object 
 3   Quantity          536269 non-null  int64  
 4   CustomerNo        536269 non-null  float64
 5   Price             536269 non-null  float64
 6   Product Category  536269 non-null  object 
 7   Country           536269 non-null  object 
 8   Gender            536269 non-null  object 
 9   Age               536269 non-null  int64  
dtypes: float64(2), int64(2), object(6)
memory usage: 40.9+ MB
None
```

Initial Exploration & Feature Checks

In [14]:
```
print(df.describe(include='all'))
```

```
/var/folders/xb/2tg9ddl94wl284px7ngj8hn40000gn/T/ipykernel_51491/115838729.py:1: FutureWarning: The behavior of value_c
ounts with object-dtype is deprecated. In a future version, this will *not* perform dtype inference on the resulting in
dex. To retain the old behavior, use `result.index = result.index.infer_objects()`
  print(df.describe(include='all'))
```

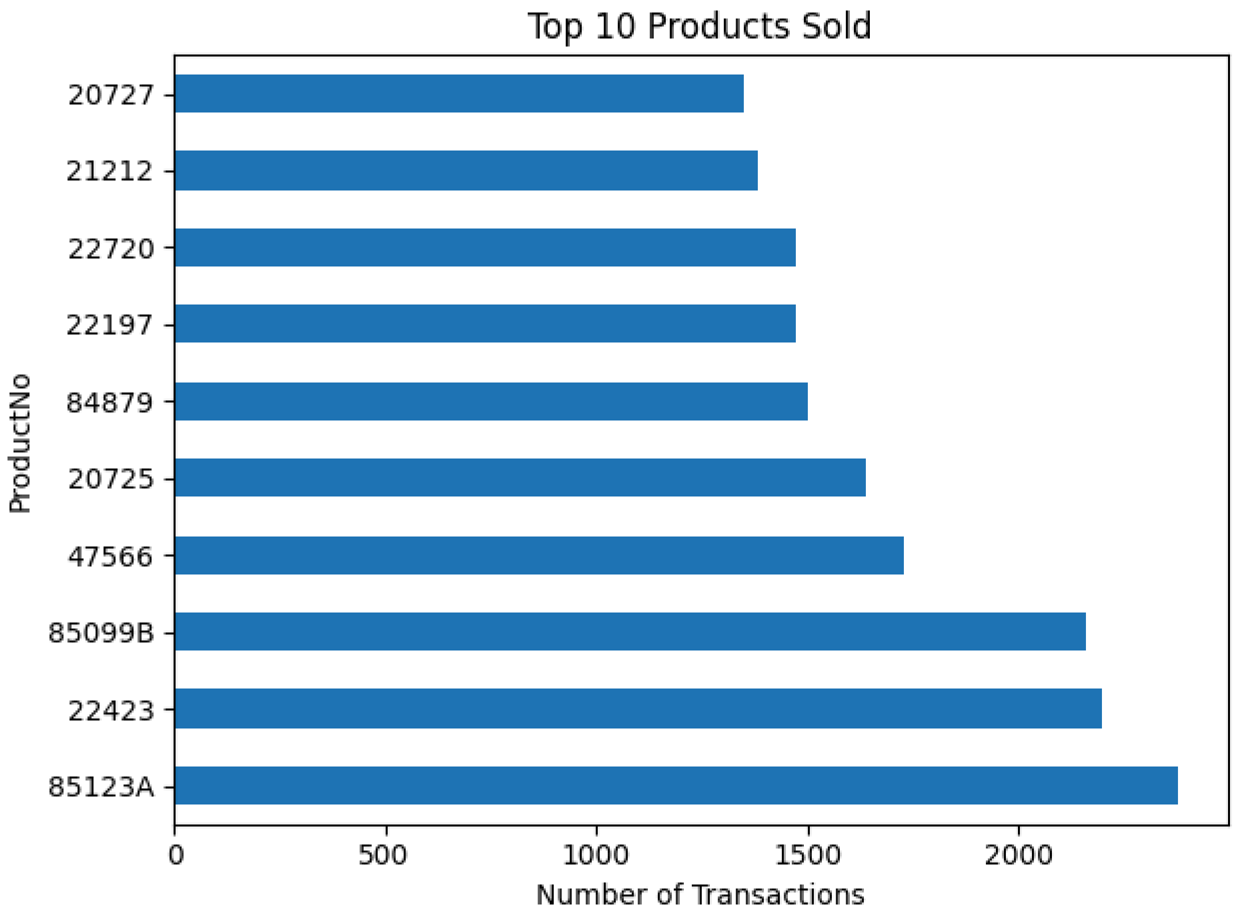
```
       TransactionNo                 Date ProductNo      Quantity  \
count         536269               536269    536269  5.362690e+05
unique         23163                  305      3767           NaN
top           573585  2019-12-05 00:00:00    85123A           NaN
freq            1111                 5299      2378           NaN
mean             NaN                  NaN       NaN  3.049953e+01
std              NaN                  NaN       NaN  1.059661e+04
min              NaN                  NaN       NaN -8.099500e+04
25%              NaN                  NaN       NaN  1.000000e+00
50%              NaN                  NaN       NaN  4.000000e+00
75%              NaN                  NaN       NaN  1.100000e+01
max              NaN                  NaN       NaN  6.487413e+06

           CustomerNo          Price Product Category         Country  Gender  \
count   536269.000000  536269.000000           536269          536269  536269
unique            NaN            NaN                3              38       2
top               NaN            NaN         Clothing  United Kingdom  Female
freq              NaN            NaN           187229          485020  269632
mean     15227.898590       7.624801              NaN             NaN     NaN
std       1716.575041       7.542364              NaN             NaN     NaN
min      12004.000000       5.460000              NaN             NaN     NaN
25%      13807.000000       6.190000              NaN             NaN     NaN
50%      15152.000000       6.190000              NaN             NaN     NaN
75%      16729.000000       7.240000              NaN             NaN     NaN
max      18287.000000     594.500000              NaN             NaN     NaN

                  Age
count   536269.000000
unique            NaN
top               NaN
freq              NaN
mean        40.802530
std         13.434077
min         18.000000
25%         29.000000
50%         41.000000
75%         52.000000
max         64.000000
```

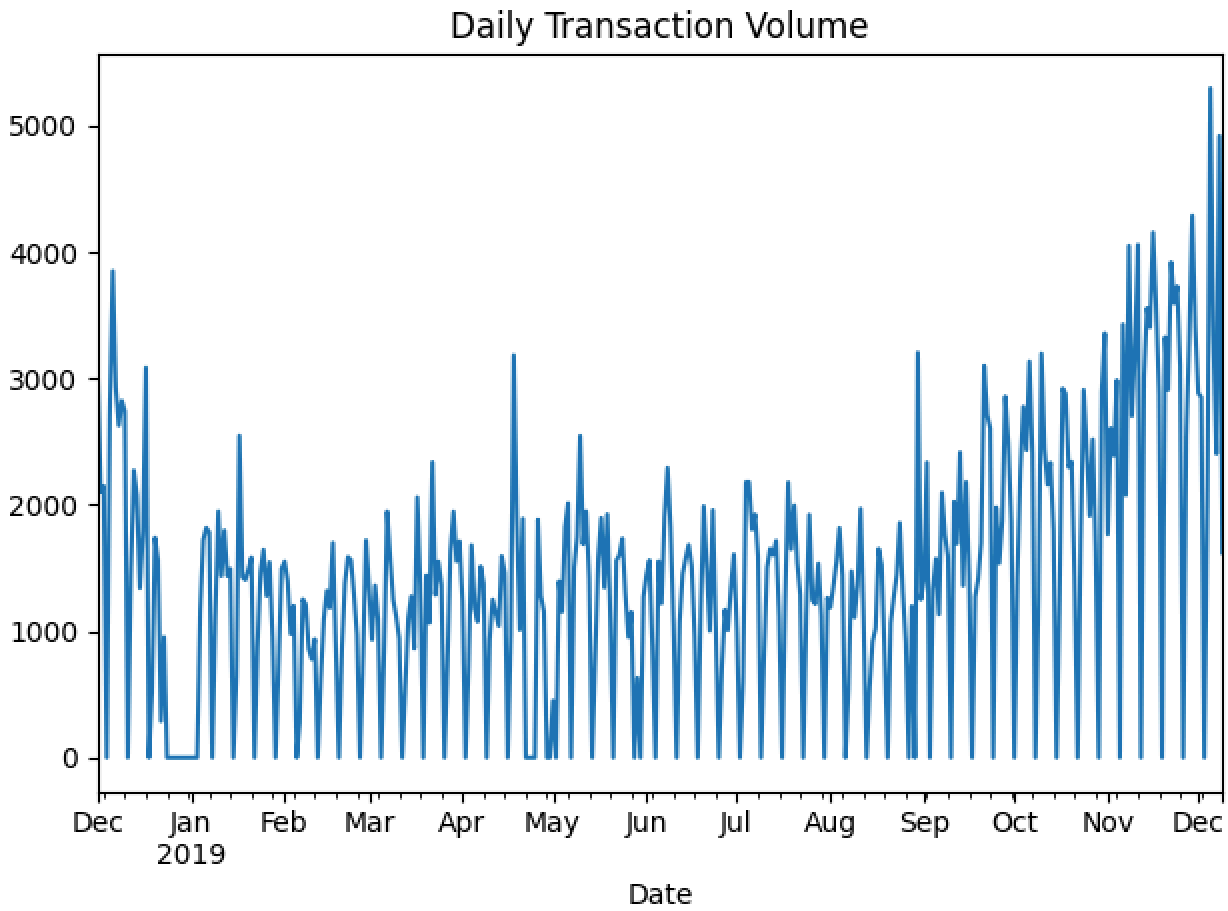
In [15]:

```
#Top 10 Products Sold
top_products = df['ProductNo'].value_counts().head(10)
top_products.plot(kind='barh', title='Top 10 Products Sold')
plt.xlabel('Number of Transactions')
plt.tight_layout()
plt.show()
```



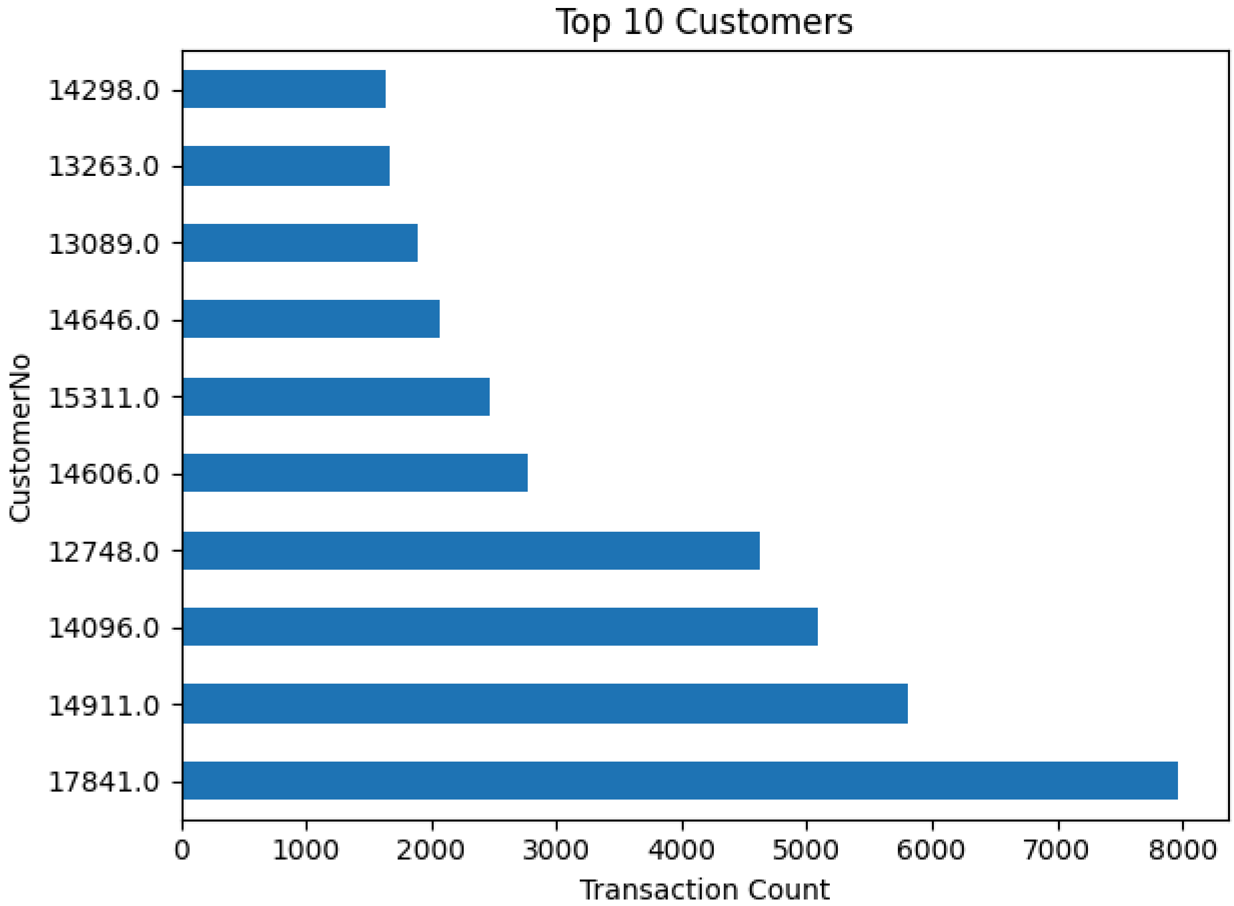
In [16]:
```
#Daily Transaction Volume
df.set_index('Date').resample('D').size().plot(title='Daily Transaction Volume')
plt.tight_layout()
plt.show()
```

```
/opt/anaconda3/envs/moflow/lib/python3.10/site-packages/pandas/core/indexes/base.py:7588: FutureWarning: Dtype inferenc
e on a pandas object (Series, Index, ExtensionArray) is deprecated. The Index constructor will keep the original dtype
in the future. Call `infer_objects` on the result to get the old behavior.
  return Index(sequences[0], name=names)
```

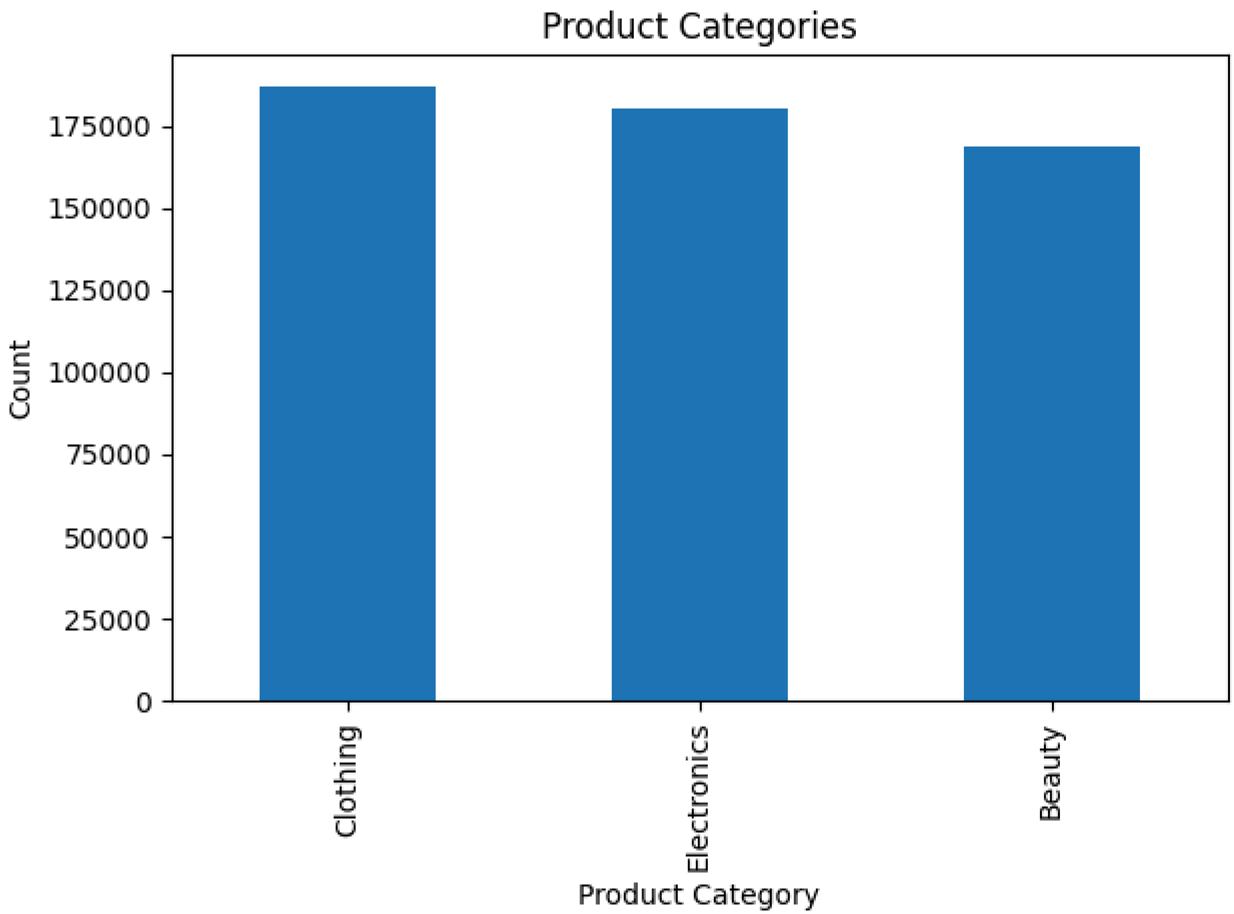


In [17]:

```
# top 10 Customers by num of purchases
df['CustomerNo'].value_counts().head(10).plot(kind='barh', title='Top 10 Customers')
plt.xlabel('Transaction Count')
plt.tight_layout()
plt.show()
```



In [18]:
```
# Product Category distribution
df['Product Category'].value_counts().plot(kind='bar', title='Product Categories')
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```



Implement AI Models (2 Core + 1 Additional Insight)

Problem 1: Market Basket Analysis (Apriori)

In [21]:
```
# basket list format
basket_df = df.groupby(['TransactionNo'])['ProductNo'].apply(list).values.tolist()
```

In [22]:
```
# hotone encode
te = TransactionEncoder()
te_ary = te.fit(basket_df).transform(basket_df)
basket_encoded =pd.DataFrame(te_ary,columns=te.columns_)
```

In [23]:
```
# run Apriori
frequent_itemsets = apriori(basket_encoded, min_support=0.01, use_colnames=True)
```

In [24]:
```
# make rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
```

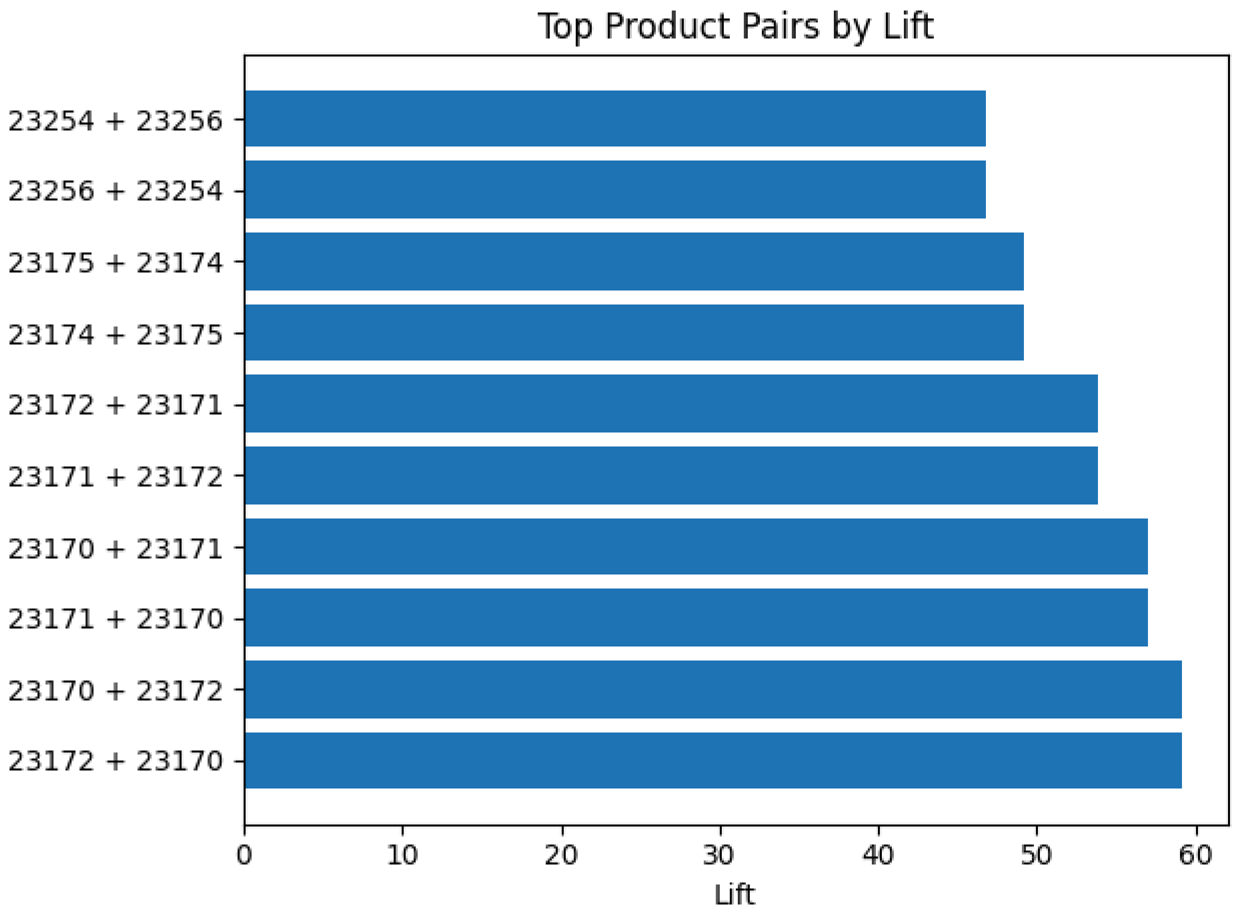
In [25]:
```
# strong rules
strong_rules = rules[(rules['lift'] >= 1.2) & (rules['confidence'] >= 0.4)]
```

In [26]:
```
# view top 10 rules
print(strong_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(10))
```

```
   antecedents consequents   support  confidence       lift
0      (20711)     (20712)  0.012477    0.545283  14.484393
3      (20711)     (21931)  0.012865    0.562264  10.844067
5      (20711)     (22386)  0.012175    0.532075  10.011750
6      (20711)     (22411)  0.011484    0.501887   9.793769
8      (20711)    (85099B)  0.015024    0.656604   7.123613
10     (20713)     (20712)  0.012218    0.426848  11.338386
25     (20712)     (21931)  0.019428    0.516055   9.952858
28     (22385)     (20712)  0.012606    0.421965  11.208696
31     (20712)     (22386)  0.018607    0.494266   9.300312
33     (20712)     (22411)  0.016837    0.447248   8.727547
```

In [27]:
```
top_lift = strong_rules.sort_values(by='lift', ascending=False).head(10)
labels = []
for i, row in top_lift.iterrows():
    item1 = list(row['antecedents'])[0]
    item2 = list(row['consequents'])[0]
    labels.append(f"{item1} + {item2}")

plt.barh(labels, top_lift['lift'])
plt.xlabel("Lift")
plt.title("Top Product Pairs by Lift")
plt.tight_layout()
plt.show()
```



Analysis

I used the Apriori algorithm to identify frequently co-purchased products and generate association rules.

- The rules highlight strong relationships between items based on **support**, **confidence**, and **lift**
- For example, `{20711} -> {20712}` has:
  - **Support**: 1.25% of all transactions include this pair
  - **Confidence**: When item 20711 is bought, 54.5% of the time 20712 is also bought
  - **Lift**: 14.48 - meaning customers are **14x more likely** to buy 20712 if they buy 20711

High-lift product pairs like these can guide **cross-sell bundles or combo discounts**.

Recommendation:

- Create product bundles or targeted combo discounts for high-lift item pairs (ex: 20711 + 20712) to boost average order value
- Promote these combinations via personalized emails or on-site recommendations during checkout to encourage cross-sells

Problem 2: Customer Segmentation (RFM + K-Means)

Creating the RFM Table

In [31]:
```
# the snapshot/reference date is is 1 day after the last transaction
snapshot_date = df['Date'].max() +pd.Timedelta(days=1)
```

In [32]:
```
# group by customer
rfm = df.groupby('CustomerNo').agg({
    'Date': lambda x: (snapshot_date - x.max()).days, # Recency
    'TransactionNo': 'nunique', # Frequency
    'Price': 'sum' # Monetary
}).reset_index()
```

In [33]:
```
# Rename Columns
rfm.columns = ['CustomerNo', 'Recency', 'Frequency', 'Monetary']
```

In [34]:
```
# remove Negative monetary values
rfm = rfm[rfm['Monetary']>0]
```

Normalizing the Data

In [36]:
```
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm[['Recency', 'Frequency', 'Monetary']])
```

Apply K-Means Clustering

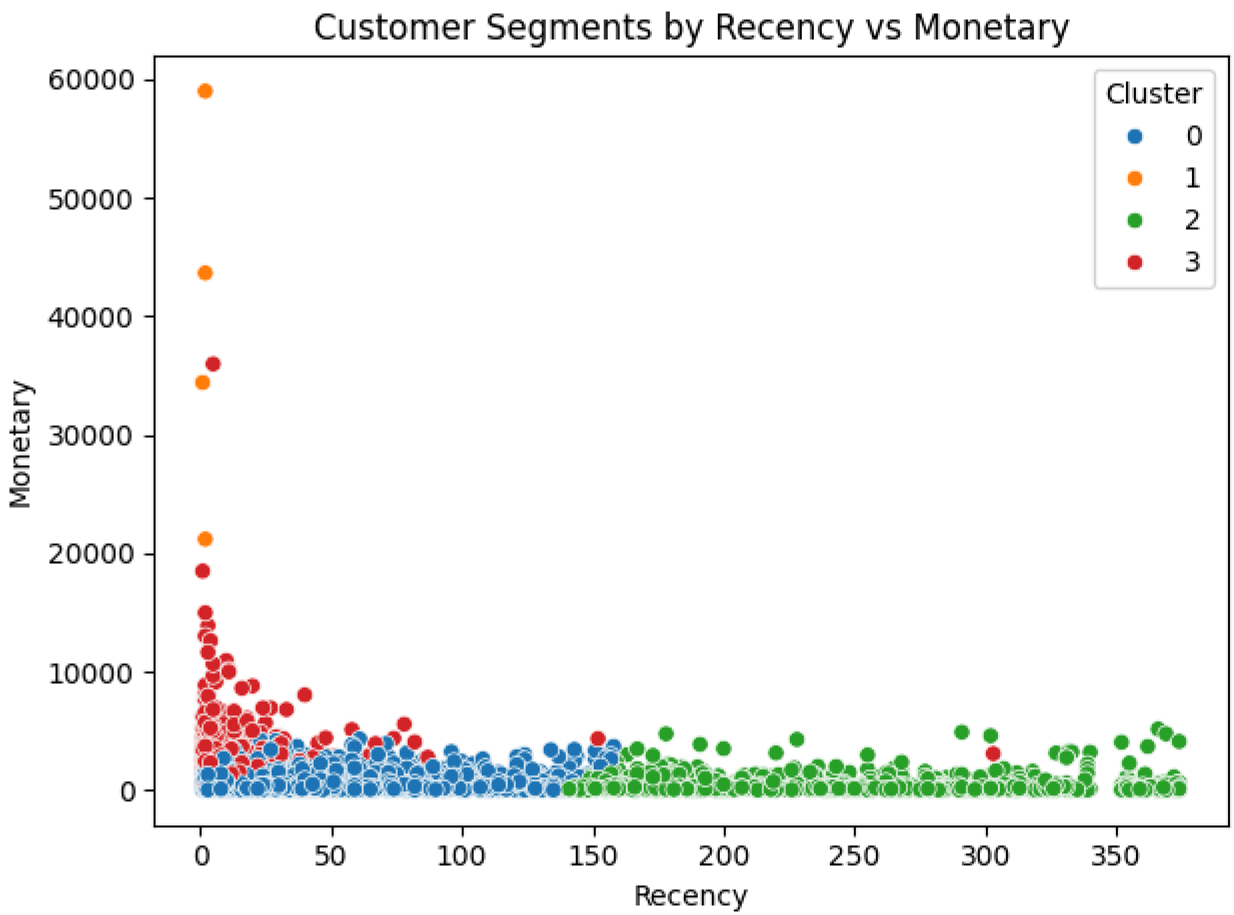
In [38]:
```
# using 4 clustors for simplicity
kmeans = KMeans(n_clusters=4,random_state=5503)
rfm['Cluster'] = kmeans.fit_predict(rfm_scaled)
```

In [39]:
```
## Analyize Clusters
cluster_summary =rfm.groupby('Cluster')[['Recency', 'Frequency', 'Monetary']].mean().round(1)
print(cluster_summary)
```

```
         Recency  Frequency  Monetary
Cluster
0           44.7        4.1     702.6
1            1.8      188.5   39551.2
2          247.2        1.7     376.0
3           13.3       22.2    3742.8
```

In [40]:

```
sns.scatterplot(data=rfm, x='Recency', y='Monetary', hue='Cluster', palette='tab10')
plt.title("Customer Segments by Recency vs Monetary")
plt.tight_layout()
plt.show()
```



Analysis

I applied K-Means clustering on standardized RFM features (Recency, Frequency, Monetary) to identify key customer segments. The results reveal four distinct behavioral groups:

Cluster Summaries:

- **Cluster 1**
    - Very recent purchases
    - Very frequent buyers
    - Highest total spend
    - -> Active, loyal customers - best overall
- **Cluster 3**
    - Fairly recent
    - Moderate frequency
    - Decent total spend
    - -> Good customers - might be worth upselling
- **Cluster 0**
    - Not very recent
    - Low frequency
    - Low total spend
    - -> Average group - not high priority
- **Cluster 2**
    - Long time since last purchase
    - Very few transactions
    - Low total spend
    - -> At-risk or inactive - may be dropped or re-engaged

Recommendation:

These insights allow for **targeted marketing strategies**:

- Reward and retain Cluster 1
- Upsell Cluster 3
- Reactivate or drop Cluster 2

Problem 3: Country-Level Sales

Group by Country and Sum Monetary Value

In [44]:
```
# Group by country and sum total revenue
country_sales = df.groupby('Country')['Price'].sum().sort_values(ascending=False).reset_index()
```

In [45]:
```
# rename columns
country_sales.columns = ['Country', 'TotalRevenue']
```

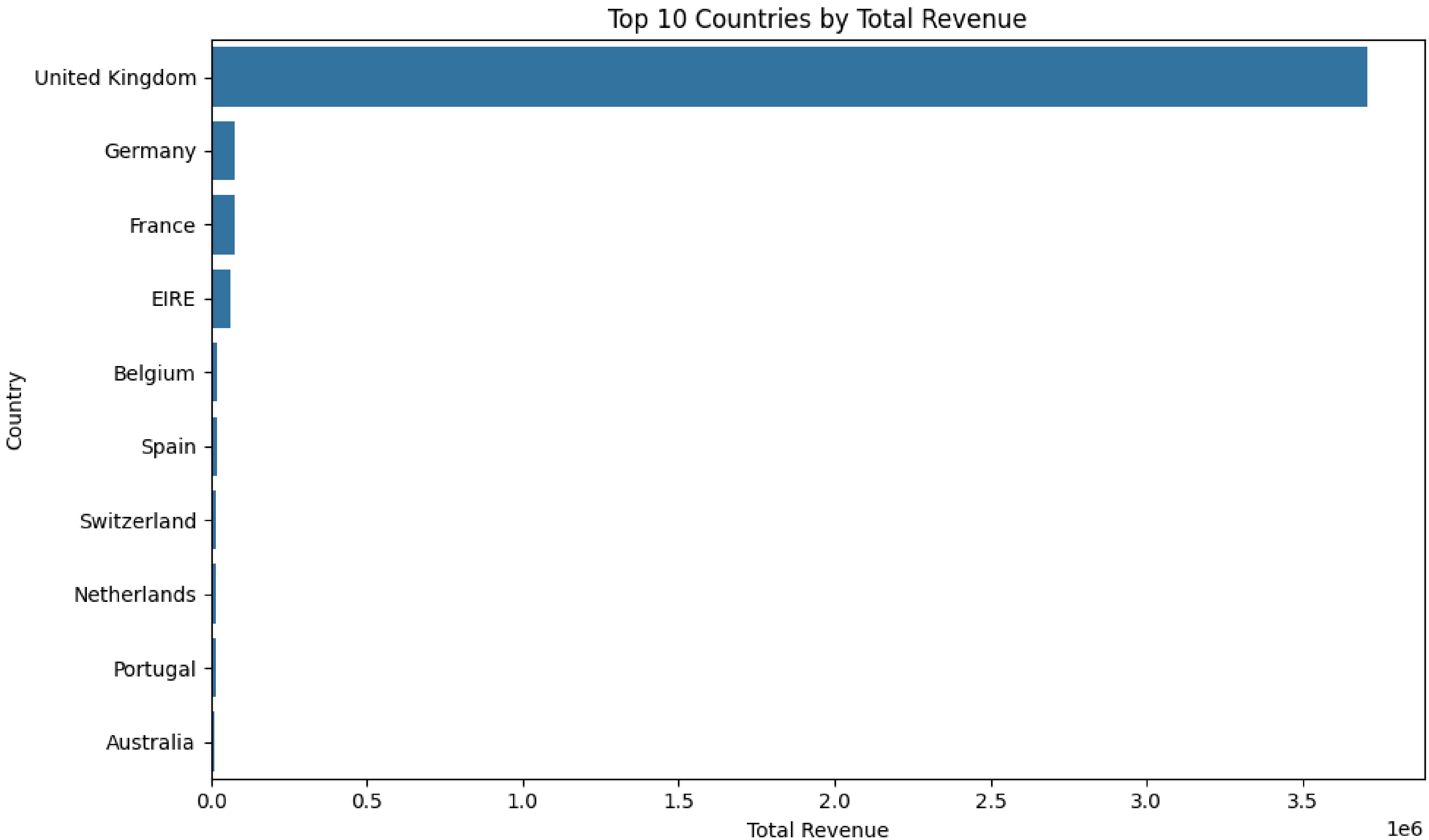
In [46]:
```
country_sales.head(10)
```

Out[46]:

	Country	TotalRevenue
0	United Kingdom	3704989.59
1	Germany	78864.67
2	France	75927.33
3	EIRE	61082.72
4	Belgium	18854.47
5	Spain	18539.21
6	Switzerland	16977.69
7	Netherlands	16835.35
8	Portugal	13530.34
9	Australia	12905.39

In [47]:
```
top_countries = country_sales.head(10)

plt.figure(figsize=(10, 6))
sns.barplot(data=top_countries, x='TotalRevenue', y='Country')
plt.title('Top 10 Countries by Total Revenue')
plt.xlabel('Total Revenue')
plt.ylabel('Country')
plt.tight_layout()
plt.show()
```



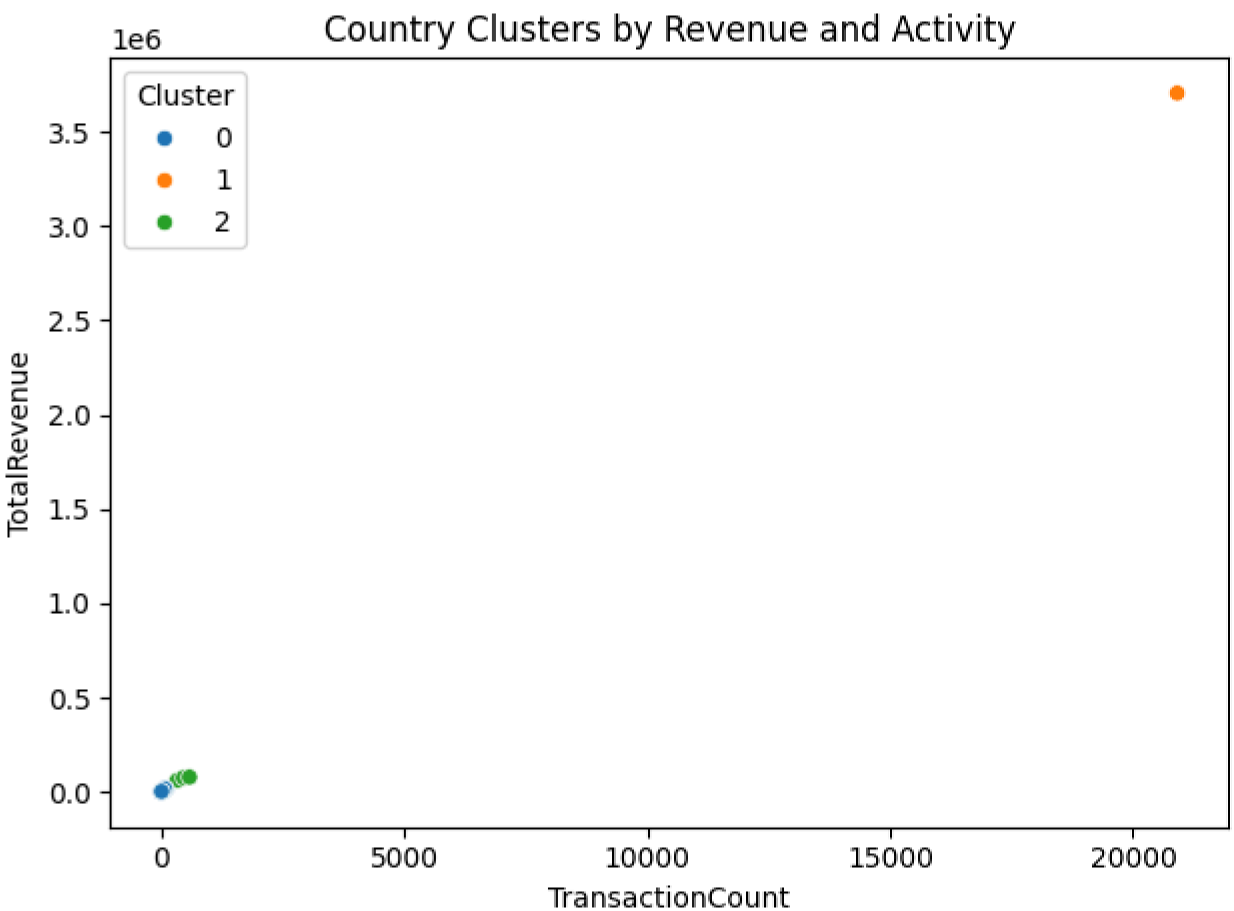
In [48]:
```
# Features per Country
country_stats = df.groupby('Country').agg({
    'CustomerNo': 'nunique',
    'TransactionNo': 'nunique',
    'Price': 'sum'
}).reset_index()

country_stats.columns =['Country','UniqueCustomers', 'TransactionCount', 'TotalRevenue']
```

In [49]:
```python
# scale
features = country_stats[['UniqueCustomers', 'TransactionCount', 'TotalRevenue']]
scaler = StandardScaler()
scaled = scaler.fit_transform(features)
```

In [50]:
```python
#K-means
kmeans = KMeans(n_clusters=3, random_state=5503)
country_stats['Cluster'] = kmeans.fit_predict(scaled)
```

In [51]:
```python
sns.scatterplot(data=country_stats, x='TransactionCount', y='TotalRevenue', hue='Cluster', palette='tab10')
plt.title("Country Clusters by Revenue and Activity")
plt.tight_layout()
plt.show()
```



Analysis

I analyzed customer revenue by country to understand geographic performance and market potential.

Key Findings:

- **United Kingdom** generated over **$3.7 million**, dominating all other regions
- Other countries like **Germany**, **France**, and **EIRE (Ireland)** had moderate revenue, ranging from ~60K to 78K.
- Remaining countries (Belgium, Spain, Australia, etc) contributed under 20K each.

K-Means Country Clustering

To further explore international potential, I used **K-Means clustering** on three country level metrics:

- Total Revenue
- Transaction Count
- Unique Customer Count

This grouped countries into **3 distinct clusters**:

- **Cluster 1**: High-performing (like United Kingdom) - high spend, lots of activity
- **Cluster 2**: Mid-range markets - Germany, France, EIRE
- **Cluster 0**: Low-volume countries - potential areas for marketing tests or deprioritization

Recommendation:

- Focus resources and promotions in the **UK**, where sales are strongest.
- Consider **Germany and France** for expansion - they show early traction.
- Test small campaigns in low-performing regions (Portugal, Australia) or deprioritize.

AI Models used for the problems:

- Apriori (1)
- K-Means (2 & 3)

In [ ]:

```
```