In [83]:
```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from scipy.stats import zscore
import matplotlib.pyplot as plt
from datetime import timedelta
```

In [86]:
```
# Load dataset (use a sample for performance)
df = pd.read_excel(r"C:\Users\mukki\OneDrive\Desktop\Online Retail.xlsx", sheet_nam

df
```

Out[86]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	C
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	K
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	K
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	K
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	K
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	K
...	...	...	...	...	...	...	...	
4995	536836	21843	RED RETROSPOT CAKE STAND	2	2010-12-02 18:08:00	10.95	18168.0	K
4996	536836	21531	RED RETROSPOT SUGAR JAM BOWL	2	2010-12-02 18:08:00	2.55	18168.0	K
4997	536836	21539	RED RETROSPOT BUTTER DISH	3	2010-12-02 18:08:00	4.95	18168.0	K
4998	536836	22198	LARGE POPCORN HOLDER	2	2010-12-02 18:08:00	1.65	18168.0	K
4999	536836	22197	SMALL POPCORN HOLDER	2	2010-12-02 18:08:00	0.85	18168.0	K

5000 rows × 8 columns

In [19]:
```
# ----------------------------
# Handle Missing Values
# ----------------------------
# Using KNN Imputer for demonstration
knn_imputer = KNNImputer(n_neighbors=5)
df['CustomerID'] = knn_imputer.fit_transform(df[['CustomerID']])
df['CustomerID']
```

Out[19]:
```
0       17850.0
1       17850.0
2       17850.0
3       17850.0
4       17850.0
         ...
4995    18168.0
4996    18168.0
4997    18168.0
4998    18168.0
4999    18168.0
Name: CustomerID, Length: 4988, dtype: float64
```

In [23]:
```
# Drop rows with missing 'Description' as they can't be imputed meaningfully
df.dropna(subset=['Description'], inplace=True)
```

In [81]:
```
# ----------------------------
# Outlier Detection & Handling
# ----------------------------
# Robust Z-Score Method
z_scores = np.abs(zscore(df[['Quantity', 'UnitPrice']]))
outliers = (z_scores > 3)
df[['Quantity', 'UnitPrice']] = df[['Quantity', 'UnitPrice']].mask(outliers, np.nan
df[['Quantity', 'UnitPrice']]
outliers
```

Out[81]:

	Quantity	UnitPrice
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
...	...	...
4995	False	False
4996	False	False
4997	False	False
4998	False	False
4999	False	False

4988 rows × 2 columns

In [45]:

```
# ----------------------------
# Normalization & Standardization
# ----------------------------
scaler_minmax = MinMaxScaler()
scaler_standard = StandardScaler()
df[['Quantity_scaled']] = scaler_minmax.fit_transform(df[['Quantity']])
df[['UnitPrice_scaled']] = scaler_standard.fit_transform(df[['UnitPrice']])
df[['Quantity_scaled']]
df[['UnitPrice_scaled']]
```

Out[45]:

	UnitPrice_scaled
0	-0.247739
1	-0.011391
2	-0.191465
3	-0.011391
4	-0.011391
...	...
4995	2.115740
4996	-0.247739
4997	0.427541
4998	-0.500969
4999	-0.726062

4988 rows × 1 columns

In [47]:
```
# ----------------------------
# Feature Engineering
# ----------------------------
# Total price per transaction
df['TotalPrice'] = df['Quantity'] * df['UnitPrice']
df['TotalPrice']
```

Out[47]:
```
0       15.30
1       20.34
2       22.00
3       20.34
4       20.34
        ...
4995    21.90
4996     5.10
4997    14.85
4998     3.30
4999     1.70
Name: TotalPrice, Length: 4988, dtype: float64
```

In [51]:
```
# Recency (days since last purchase)
latest_date = df['InvoiceDate'].max() + timedelta(days=1)
recency_df = df.groupby('CustomerID')['InvoiceDate'].max().reset_index()
recency_df['Recency'] = (latest_date - recency_df['InvoiceDate']).dt.days
recency_df['Recency']
```

Out[51]:
```
0      2
1      2
2      1
3      2
4      2
      ..
193    2
194    2
195    1
196    2
197    1
Name: Recency, Length: 198, dtype: int64
```

In [55]:
```
# Frequency (number of purchases)
frequency_df = df.groupby('CustomerID')['InvoiceNo'].nunique().reset_index()
frequency_df.columns = ['CustomerID', 'Frequency']
frequency_df.columns
```

Out[55]:
```
Index(['CustomerID', 'Frequency'], dtype='object')
```

In [59]:
```
# Monetary value (total spend)
monetary_df = df.groupby('CustomerID')['TotalPrice'].sum().reset_index()
monetary_df.columns = ['CustomerID', 'Monetary']
monetary_df.columns
```

Out[59]:
```
Index(['CustomerID', 'Monetary'], dtype='object')
```

In [69]:
```
# Merge RFM metrics
rfm = recency_df.merge(frequency_df, on='CustomerID').merge(monetary_df, on='Custom
rfm
```

Out[69]:

	CustomerID	InvoiceDate	Recency	Frequency	Monetary
0	12431.0	2010-12-01 10:03:00	2	1	358.25
1	12433.0	2010-12-01 13:24:00	2	1	1919.14
2	12471.0	2010-12-02 10:37:00	1	1	-17.00
3	12472.0	2010-12-01 14:33:00	2	1	-122.30
4	12583.0	2010-12-01 08:45:00	2	1	855.86
...	...	...	...	...	...
193	18085.0	2010-12-01 12:08:00	2	1	303.90
194	18144.0	2010-12-01 13:45:00	2	1	165.05
195	18168.0	2010-12-02 18:08:00	1	2	139.60
196	18229.0	2010-12-01 16:25:00	2	1	344.20
197	18239.0	2010-12-02 17:48:00	1	1	438.10

198 rows × 5 columns

In [71]:
```
# Calculate Loyalty Score: proxy using Frequency
rfm['LoyaltyScore'] = scaler_minmax.fit_transform(rfm[['Frequency']])
rfm['LoyaltyScore']
```

Out[71]:
```
0      0.000000
1      0.000000
2      0.000000
3      0.000000
4      0.000000
         ...
193    0.000000
194    0.000000
195    0.030303
196    0.000000
197    0.000000
Name: LoyaltyScore, Length: 198, dtype: float64
```

In [73]:
```
# Customer Lifetime Value (simple proxy: Monetary * Frequency)
rfm['CLV'] = rfm['Monetary'] * rfm['Frequency']
rfm['CLV']
```

Out[73]:
```
0       358.25
1      1919.14
2       -17.00
3      -122.30
4       855.86
         ...
193     303.90
194     165.05
195     279.20
196     344.20
197     438.10
Name: CLV, Length: 198, dtype: float64
```

In [75]:
```
# ------------------------------
# Final merged dataset for modeling or analysis
# ------------------------------
df_final = df.merge(rfm, on='CustomerID', how='left')
df_final
```

Out[75]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate_x	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6.0	2010-12-01 08:26:00	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6.0	2010-12-01 08:26:00	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8.0	2010-12-01 08:26:00	2.75	17850.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	2010-12-01 08:26:00	3.39	17850.0
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6.0	2010-12-01 08:26:00	3.39	17850.0
...	...	...	...	...	...	...	...
4983	536836	21843	RED RETROSPOT CAKE STAND	2.0	2010-12-02 18:08:00	10.95	18168.0
4984	536836	21531	RED RETROSPOT SUGAR JAM BOWL	2.0	2010-12-02 18:08:00	2.55	18168.0
4985	536836	21539	RED RETROSPOT BUTTER DISH	3.0	2010-12-02 18:08:00	4.95	18168.0
4986	536836	22198	LARGE POPCORN HOLDER	2.0	2010-12-02 18:08:00	1.65	18168.0
4987	536836	22197	SMALL POPCORN HOLDER	2.0	2010-12-02 18:08:00	0.85	18168.0

4988 rows × 17 columns

In [77]:
```
# Save to CSV if needed
df_final.to_csv("preprocessed_retail_data.csv", index=False)
```

In [79]:
```
print("Data preprocessing and feature engineering completed.")
```

```
Data preprocessing and feature engineering completed.
```

In [ ]: