load data and store in df

In [1]:
```
from data_processing import load_data # Load data in df and store to loaded.csv
df = load_data()

print(f"Data loaded: {df.shape[0]} rows, {df.shape[1]} columns.")
```

```
2025-02-06 18:32:47,238 - INFO - Loading all rows from the October dataset...
2025-02-06 18:33:20,305 - INFO - Data loaded successfully: 424487 rows, 9 columns.
2025-02-06 18:33:24,063 - INFO - Saved to 'loaded.csv'.
Data loaded: 424487 rows, 9 columns.
```

Preprocess data

In [2]:
```
from data_processing import preprocess_data

df = preprocess_data(df)
df.head()
```

```
2025-02-06 18:33:24,077 - INFO - Starting preprocessing...
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\data_proces
sing.py:46: FutureWarning: A value is trying to be set on a copy of a DataFrame or S
eries through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['category_code'].fillna('unknown', inplace=True)
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\data_proces
sing.py:47: FutureWarning: A value is trying to be set on a copy of a DataFrame or S
eries through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['brand'].fillna('unknown', inplace=True)
2025-02-06 18:33:30,490 - INFO - Preprocessing complete. Final dataset shape: 424487
rows, 11 columns.
2025-02-06 18:33:36,296 - INFO - Saved to 'cleaned.csv'.
```

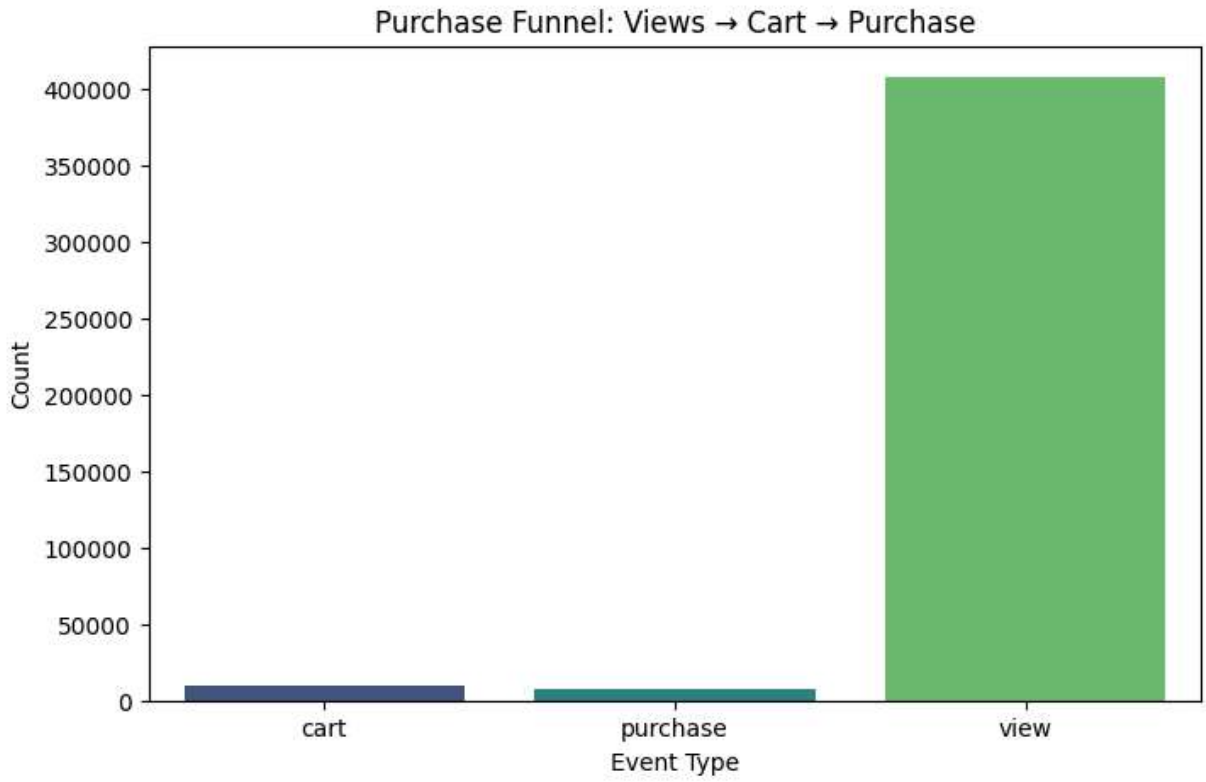
Out[2]:

vent_time	event_type	product_id	category_id	category_code	b
019-10-01 :28+00:00	view	4300070	2053013552385491165	unknown	tim
019-10-01 :46+00:00	view	3200321	2053013555321504139	appliances.kitchen.meat_grinder	redn
019-10-01 :24+00:00	view	12703421	2053013553559896355	unknown	cor
019-10-01 :25+00:00	view	3700766	2053013565983425517	appliances.environment.vacuum	sam
019-10-01 :23+00:00	view	34800106	2062461754293617058	unknown	unkr

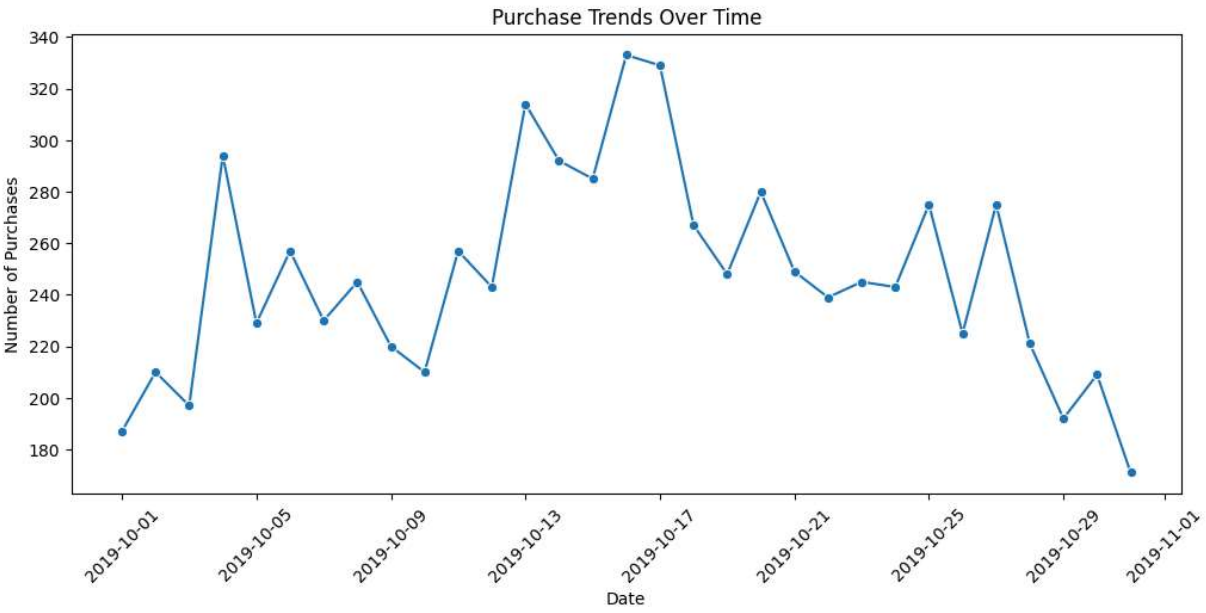
EDA

In [3]:
```
# Import the EDA functions
from eda import perform_eda

# Perform Exploratory Data Analysis
perform_eda(df)
```

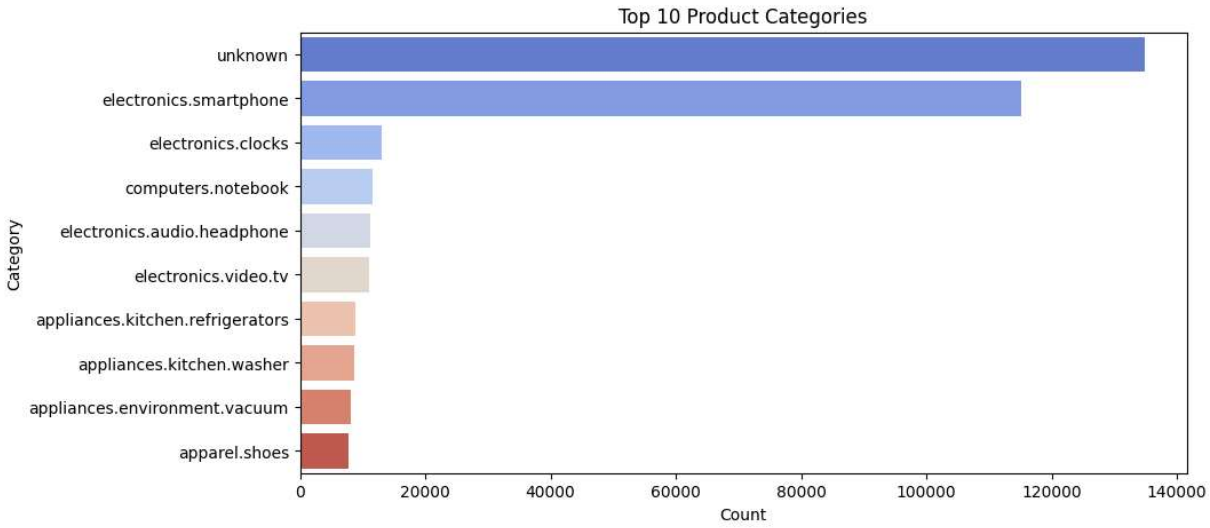
```
2025-02-06 18:33:39,520 - INFO - Starting Exploratory Data Analysis (EDA)...
2025-02-06 18:33:39,520 - INFO - Starting Exploratory Data Analysis (EDA)...
2025-02-06 18:33:39,520 - INFO - Starting Exploratory Data Analysis (EDA)...
2025-02-06 18:33:39,523 - INFO - Generating purchase funnel visualization...
2025-02-06 18:33:39,523 - INFO - Generating purchase funnel visualization...
2025-02-06 18:33:39,523 - INFO - Generating purchase funnel visualization...
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\eda.py:88:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

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



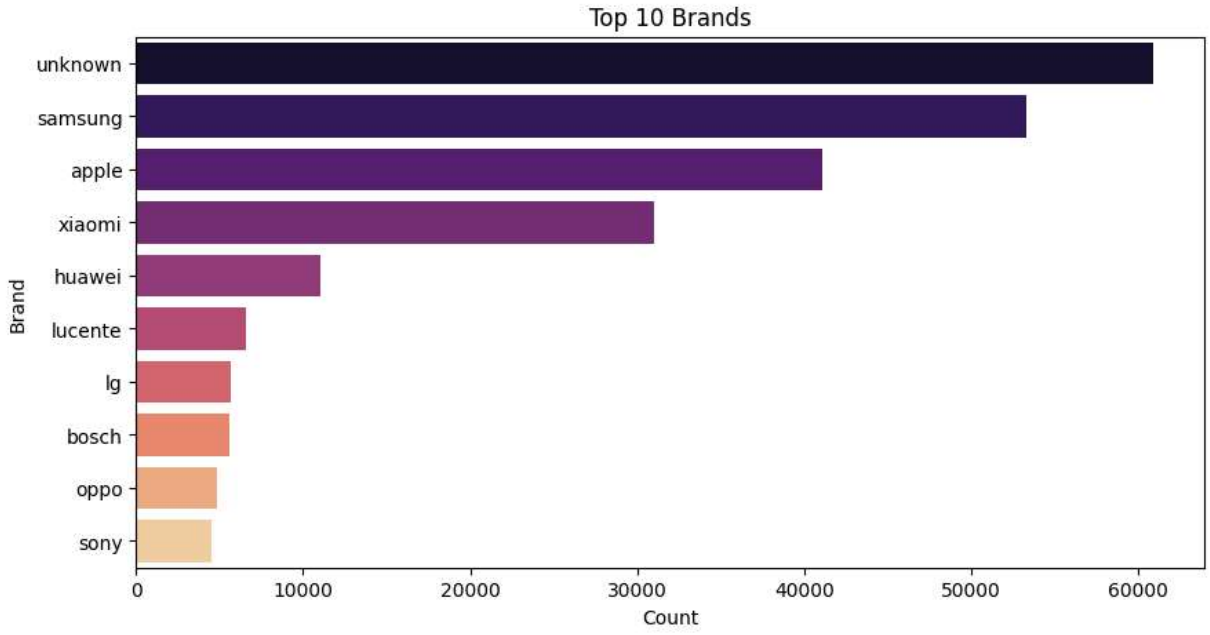

```
2025-02-06 18:33:39,906 - INFO - Generating time series analysis of purchases...
2025-02-06 18:33:39,906 - INFO - Generating time series analysis of purchases...
2025-02-06 18:33:39,906 - INFO - Generating time series analysis of purchases...
```



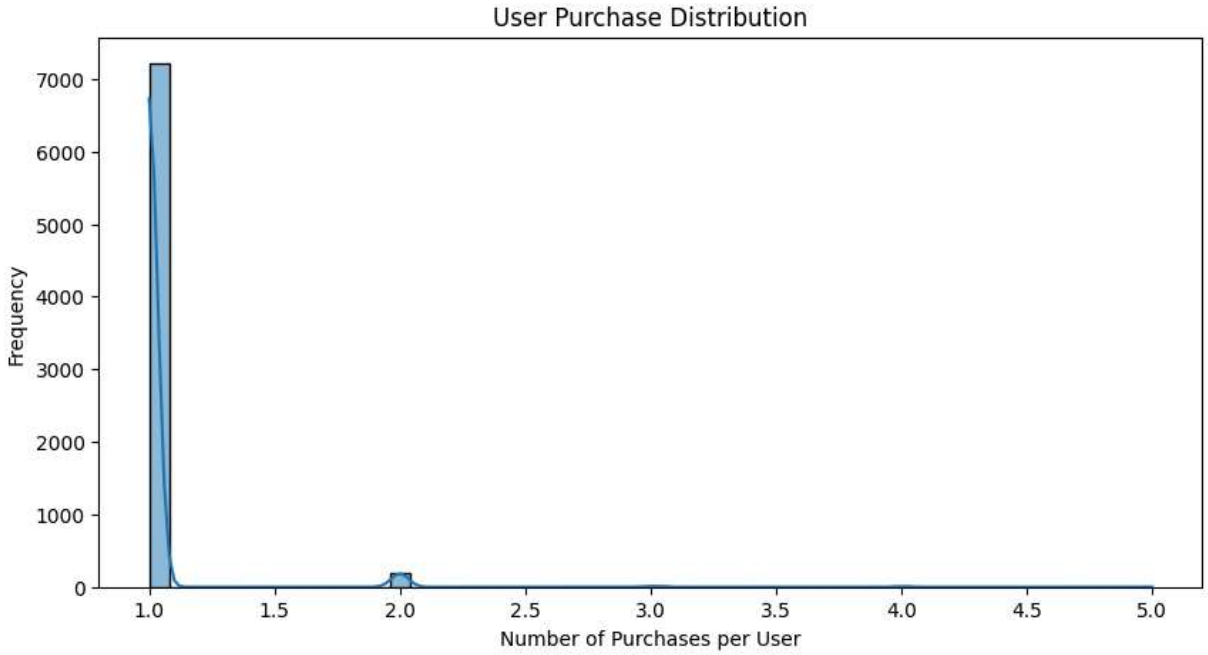

```
2025-02-06 18:33:40,251 - INFO - Generating top product categories visualization...
2025-02-06 18:33:40,251 - INFO - Generating top product categories visualization...
2025-02-06 18:33:40,251 - INFO - Generating top product categories visualization...
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\eda.py:118:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

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




```
2025-02-06 18:33:40,779 - INFO - Generating top brands visualization...
2025-02-06 18:33:40,779 - INFO - Generating top brands visualization...
2025-02-06 18:33:40,779 - INFO - Generating top brands visualization...
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\eda.py:131:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(y=top_brands.index, x=top_brands.values, palette="magma")
```




```
2025-02-06 18:33:41,255 - INFO - Generating user activity distribution visualizatio
n...
2025-02-06 18:33:41,255 - INFO - Generating user activity distribution visualizatio
n...
2025-02-06 18:33:41,255 - INFO - Generating user activity distribution visualizatio
n...
```




```
2025-02-06 18:33:41,640 - INFO - EDA completed successfully.
2025-02-06 18:33:41,640 - INFO - EDA completed successfully.
2025-02-06 18:33:41,640 - INFO - EDA completed successfully.
2025-02-06 18:33:41,640 - INFO - EDA completed successfully.
2025-02-06 18:33:41,640 - INFO - EDA completed successfully.
```

implement feature engineering to enhance the dataset for machine learning models

In [4]:
```
from feature_engineering import feature_engineering
import pandas as pd
df = df.iloc[::10]
df = feature_engineering(df)
df.head()
```

```
2025-02-06 18:33:41,928 - INFO - Starting feature engineering...
2025-02-06 18:33:41,940 - INFO - Computing session duration...
2025-02-06 18:33:42,055 - INFO - Computing interaction counts per session...
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\feature_eng
ineering.py:17: FutureWarning: The default value of observed=False is deprecated and
will change to observed=True in a future version of pandas. Specify observed=False t
o silence this warning and retain the current behavior
  interaction_counts = df.pivot_table(index='user_session', columns='event_type', va
lues='product_id', aggfunc='count', fill_value=0)
2025-02-06 18:33:42,300 - INFO - Computing product popularity...
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\feature_eng
ineering.py:23: FutureWarning: The default value of observed=False is deprecated and
will change to observed=True in a future version of pandas. Specify observed=False t
o silence this warning and retain the current behavior
  product_counts = df.pivot_table(index='product_id', columns='event_type', values
='user_id', aggfunc='count', fill_value=0)
2025-02-06 18:33:42,349 - INFO - Computing user activity patterns...
2025-02-06 18:33:42,377 - INFO - Computing price sensitivity...
2025-02-06 18:33:42,402 - INFO - Preprocessing categorical and numerical features...
2025-02-06 18:33:42,434 - INFO - Creating user-item interaction matrix...
2025-02-06 18:45:28,010 - INFO - Feature engineering complete.
2025-02-06 18:45:29,476 - INFO - Saved to 'featured.csv'.
```

Out[4]:

	event_time	event_type	product_id	category_id	category_code	brand
0	2019-10-01 00:01:28+00:00	view	4300070	2053013552385491165	1.006221	0.934610
1	2019-10-01 01:09:16+00:00	view	1004857	2053013555631882655	0.094739	0.613749
2	2019-10-01 02:21:08+00:00	view	5100575	2053013553341792533	0.063309	-1.632278
3	2019-10-01 02:23:55+00:00	view	4300400	2053013552385491165	1.006221	0.632500
4	2019-10-01 02:26:18+00:00	view	8800401	2053013555573162395	0.157600	0.180378

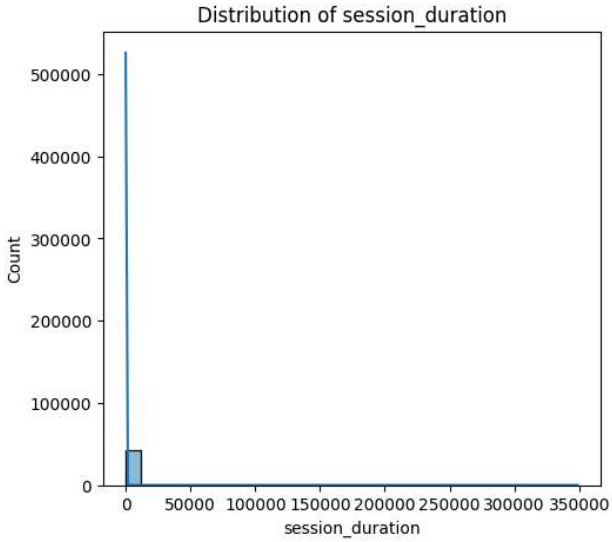
5 rows × 21 columns

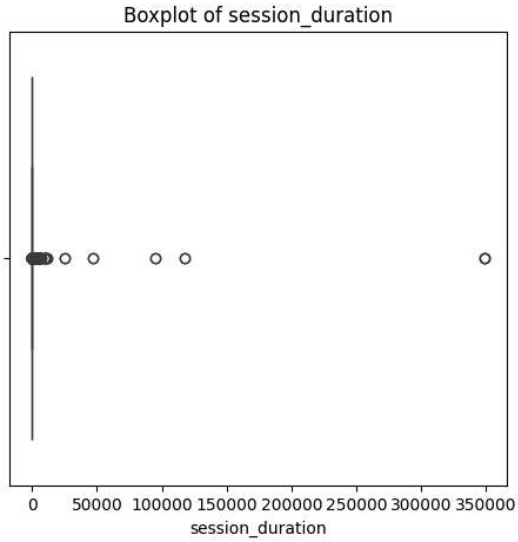
analyze features

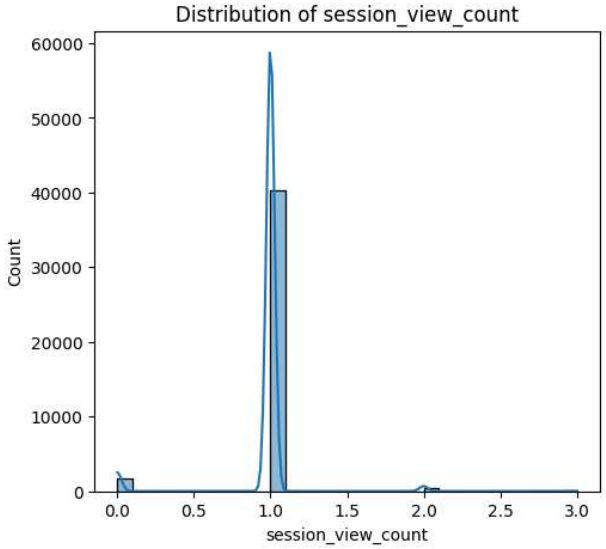
In [5]:
```
from feature_eda import analyze_features

analyze_features(df)
```

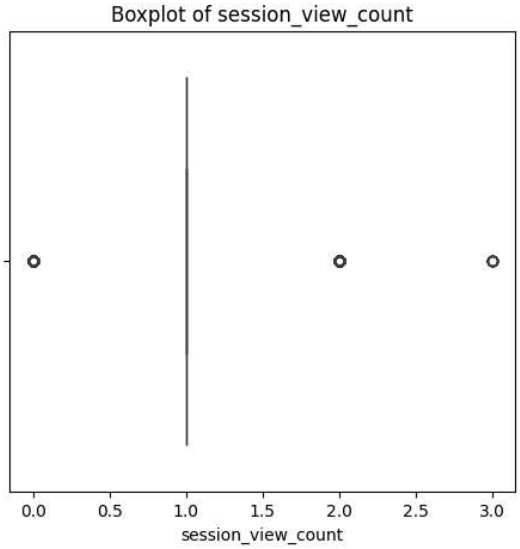
```
2025-02-06 18:45:29,651 - INFO - Starting EDA on engineered features...
2025-02-06 18:45:29,653 - INFO - Generating summary statistics...
```

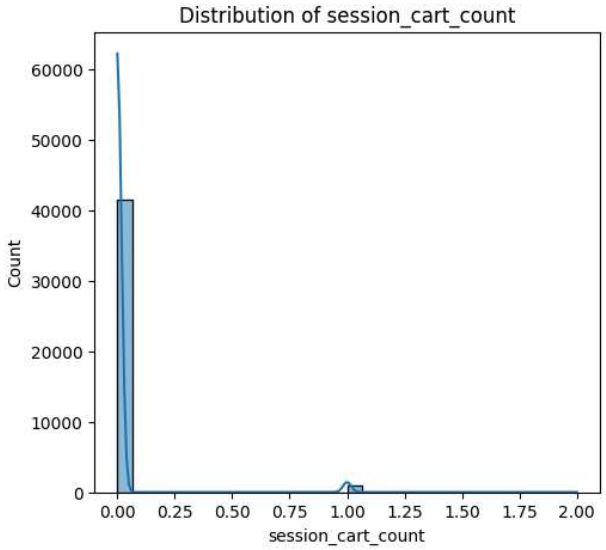
```
         product_id   category_id  category_code         brand         price  \
count  4.244900e+04  4.244900e+04   4.244900e+04  4.244900e+04  4.244900e+04
mean   1.052422e+07  2.057389e+18  -5.825081e-17 -6.578324e-17 -1.707351e-17
std    1.185439e+07  1.841851e+16   1.000012e+00  1.000012e+00  1.000012e+00
min    1.000978e+06  2.053014e+18  -2.796860e+00 -1.796876e+00 -8.027559e-01
25%    1.005159e+06  2.053014e+18  -3.452867e-01 -9.009652e-01 -6.223036e-01
50%    5.000342e+06  2.053014e+18   9.473923e-02  3.366416e-01 -3.578381e-01
75%    1.590216e+07  2.053014e+18   1.006221e+00  9.887812e-01  1.829964e-01
max    6.040000e+07  2.175420e+18   1.006221e+00  1.251304e+00  6.273674e+00

            user_id          hour     dayofweek      purchase  \
count  4.244900e+04  42449.000000  42449.000000  42449.000000
mean   5.334873e+08     11.208627      2.962049      0.017904
std    1.832143e+07      5.299605      1.957556      0.132604
min    3.186112e+08      0.000000      0.000000      0.000000
25%    5.159200e+08      7.000000      1.000000      0.000000
50%    5.297030e+08     11.000000      3.000000      0.000000
75%    5.513130e+08     16.000000      5.000000      0.000000
max    5.662765e+08     23.000000      6.000000      1.000000

       session_duration  session_cart_count  session_purchase_count  \
count      42449.000000        42449.000000            42449.000000
mean          48.474075            0.022662                0.018139
std         3137.133491            0.149459                0.133810
min            0.000000            0.000000                0.000000
25%            0.000000            0.000000                0.000000
50%            0.000000            0.000000                0.000000
75%            0.000000            0.000000                0.000000
max       348783.000000            2.000000                2.000000

       session_view_count  product_cart_count  product_purchase_count  \
count        42449.000000        42449.000000            42449.000000
mean             0.971637            2.699451                1.534524
std              0.226613            8.346460                4.498570
min              0.000000            0.000000                0.000000
25%              1.000000            0.000000                0.000000
50%              1.000000            0.000000                0.000000
75%              1.000000            1.000000                1.000000
max              3.000000           56.000000               28.000000

       product_view_count  user_event_count  price_sensitivity
count        42449.000000      42449.000000        8534.000000
mean            32.935169          1.088059           0.458510
std             74.071905          0.347669           7.718888
min              0.000000          1.000000         -40.826667
25%              1.000000          1.000000          -1.160000
50%              4.000000          1.000000           0.059821
75%             21.000000          1.000000           1.284606
max            419.000000          8.000000          42.275000
```

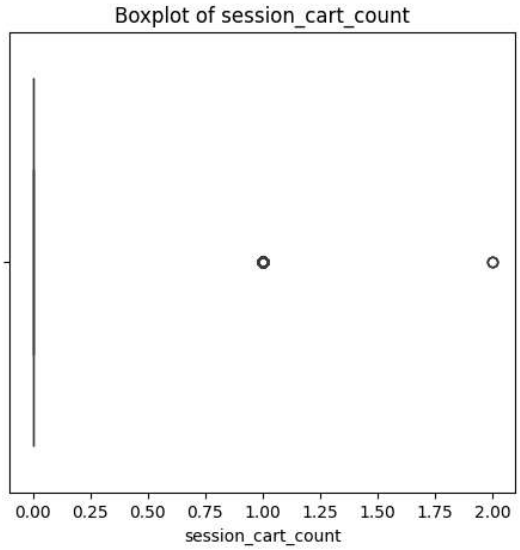

Distribution of session_duration
Count
session_duration
0
100000
200000
300000
400000
500000
0
50000
100000
150000
200000
250000
300000
350000

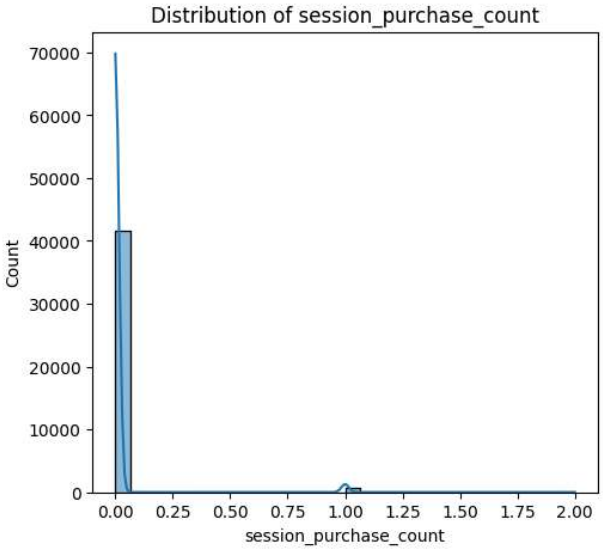

Boxplot of session_duration
session_duration
0
50000
100000
150000
200000
250000
300000
350000

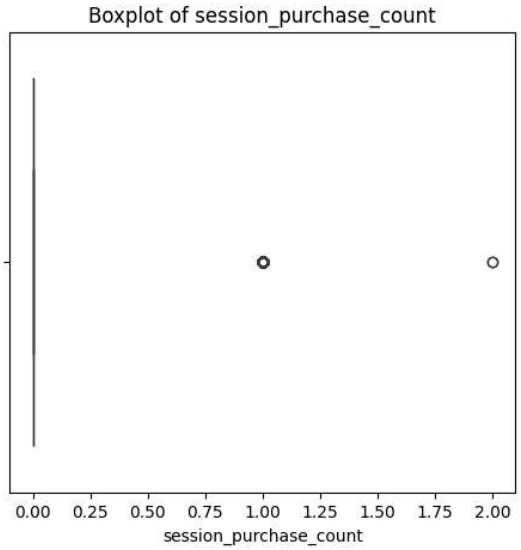

Distribution of session_view_count
Count
session_view_count
0
10000
20000
30000
40000
50000
60000
0.0
0.5
1.0
1.5
2.0
2.5
3.0

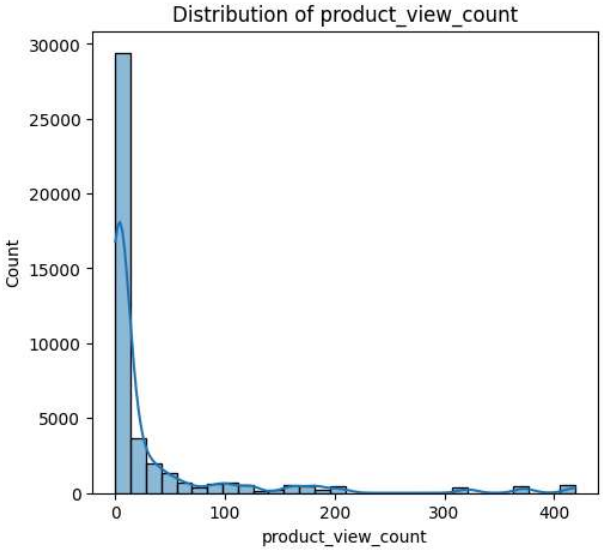

Boxplot of session_view_count
session_view_count
0.0
0.5
1.0
1.5
2.0
2.5
3.0

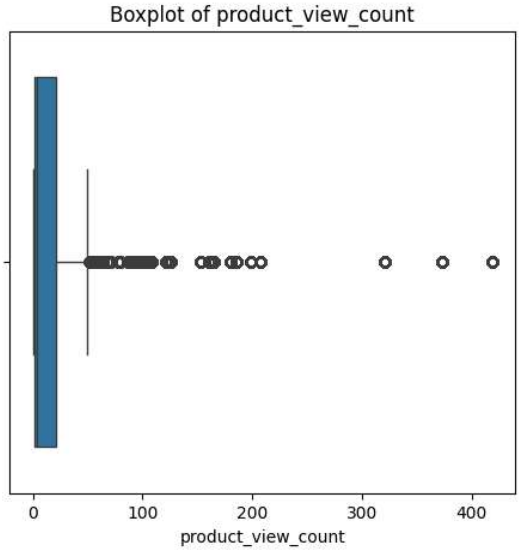

Distribution of session_cart_count
Count
session_cart_count
0
10000
20000
30000
40000
50000
60000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

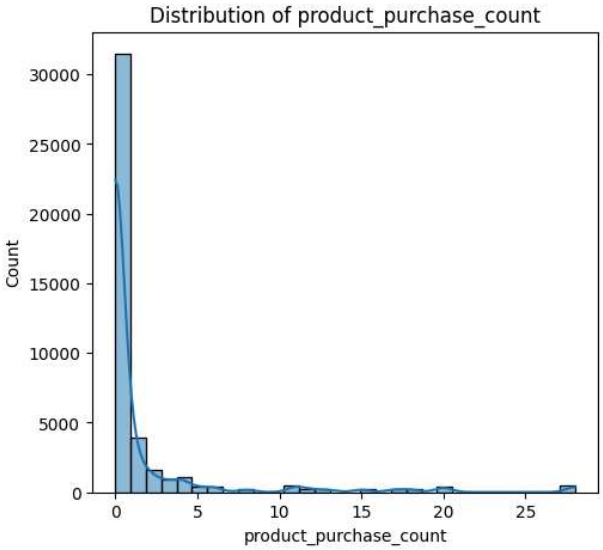

Boxplot of session_cart_count
session_cart_count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

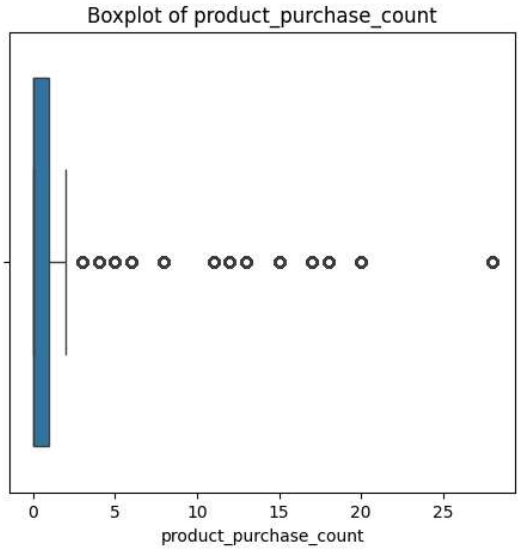

Distribution of session_purchase_count
Count
session_purchase_count
0
10000
20000
30000
40000
50000
60000
70000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

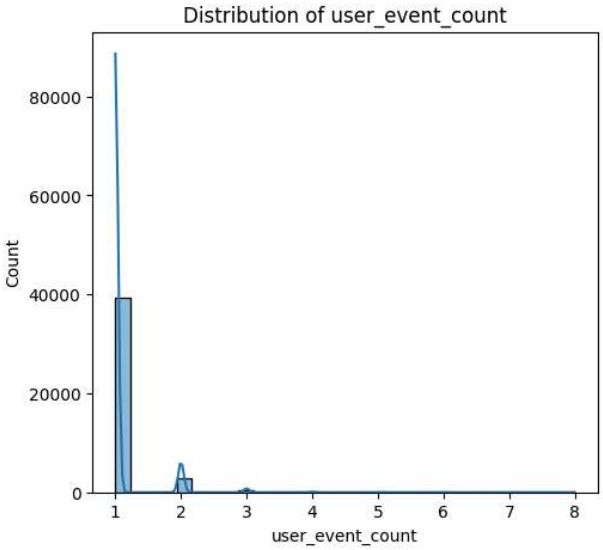

Boxplot of session_purchase_count
session_purchase_count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

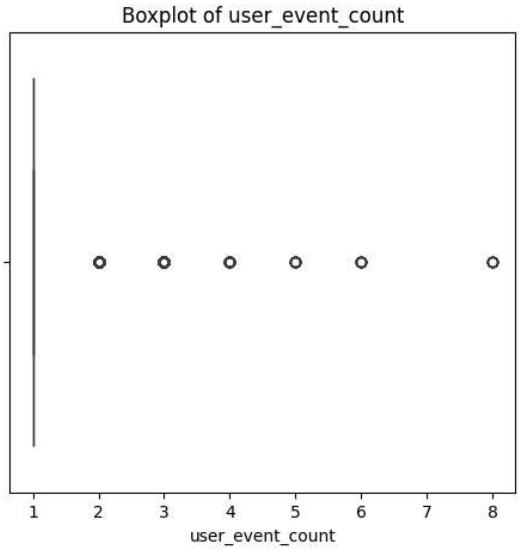

Distribution of product_view_count
Count
product_view_count
0
5000
10000
15000
20000
25000
30000
0
100
200
300
400

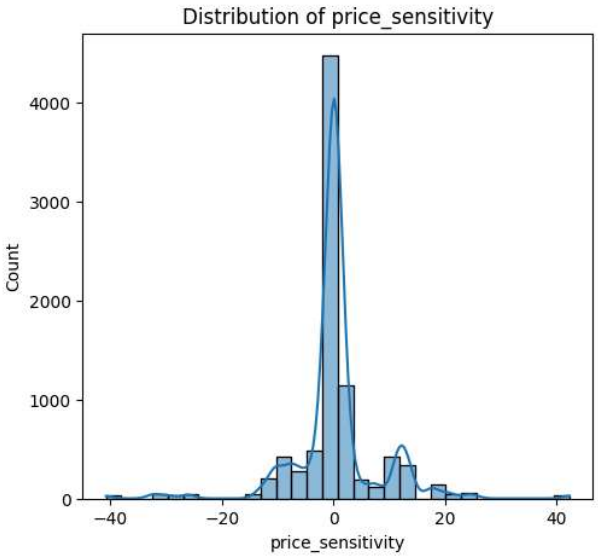

Boxplot of product_view_count
product_view_count
0
100
200
300
400

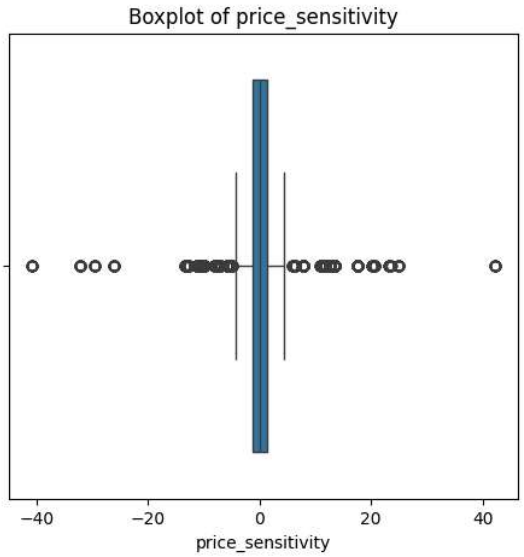

Distribution of product_purchase_count
Count
product_purchase_count
0
5000
10000
15000
20000
25000
30000
0
5
10
15
20
25


Boxplot of product_purchase_count
product_purchase_count
0
5
10
15
20
25

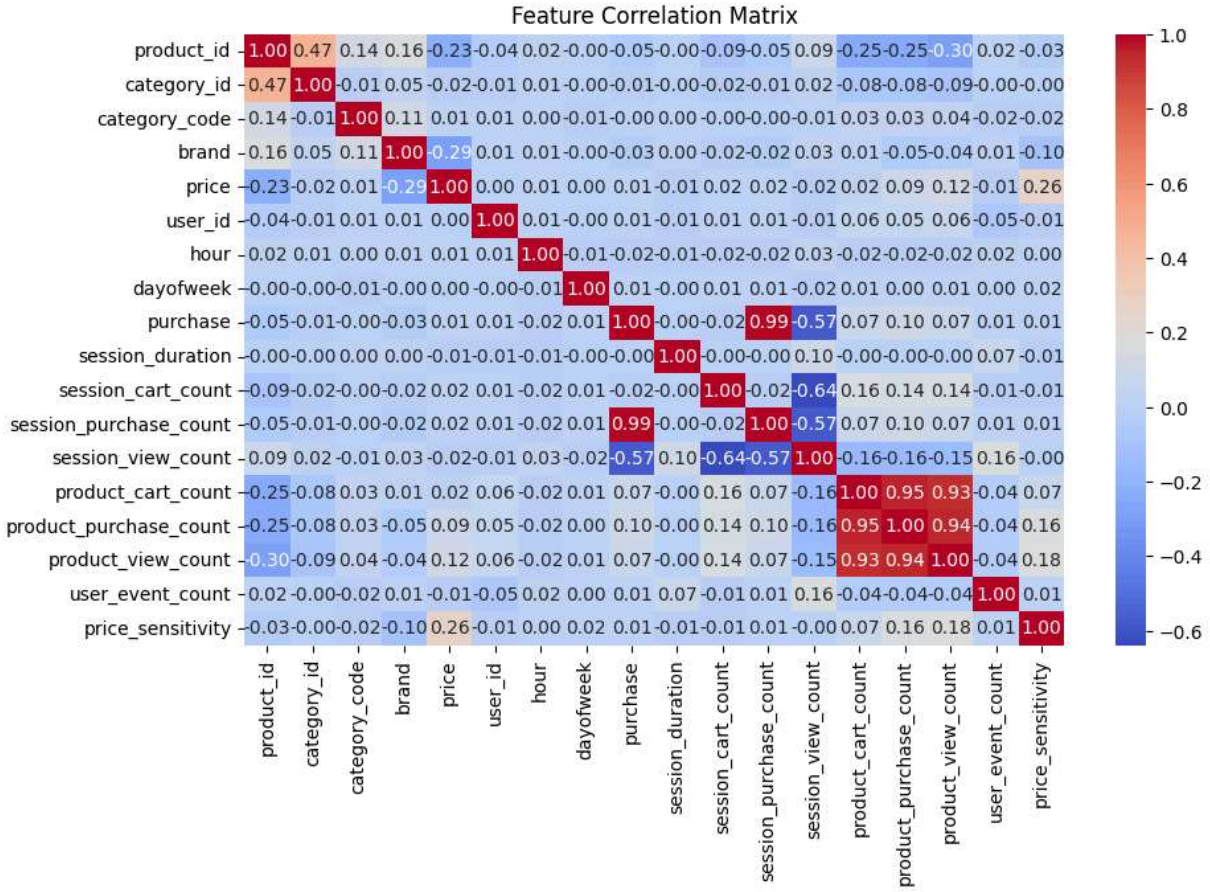









```
2025-02-06 18:45:36,014 - INFO - Generating correlation matrix...
```




```
2025-02-06 18:45:37,152 - INFO - EDA complete.
```

ml models:

Purchase Prediction Model

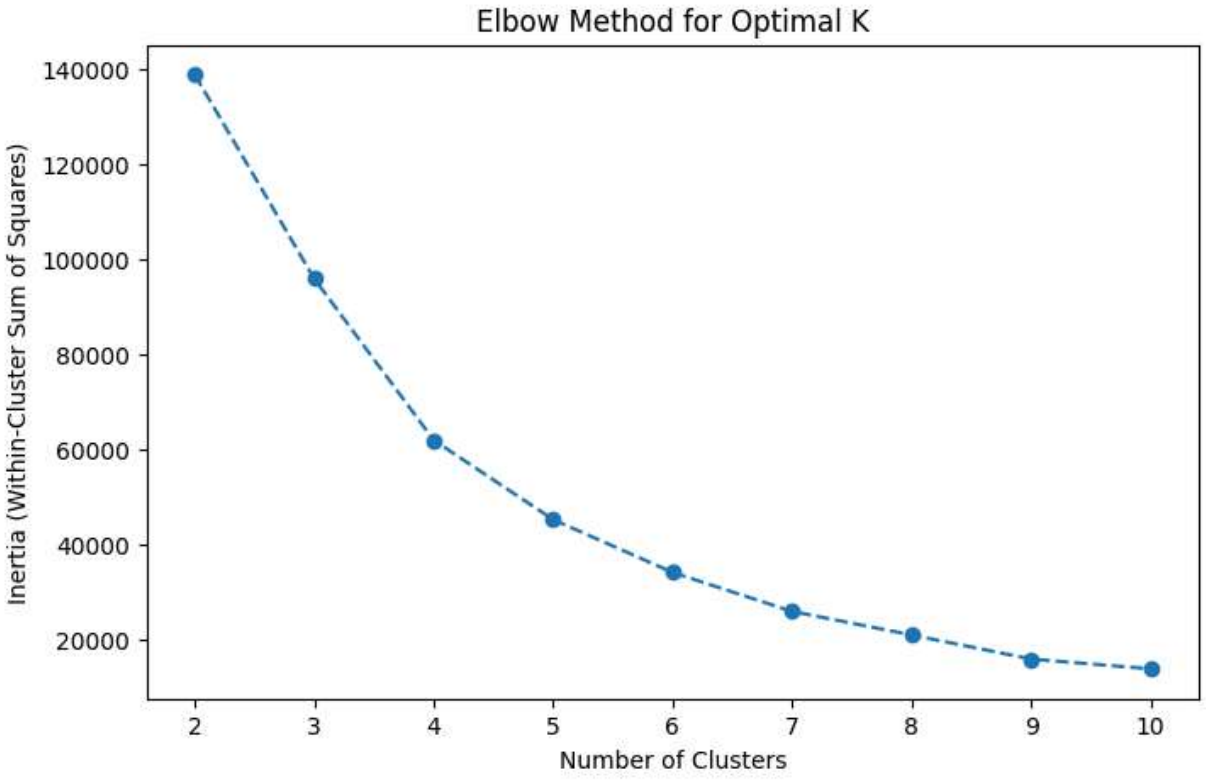
In [6]:
```python
from ml_models import train_and_evaluate

# Train and evaluate the purchase prediction model
model, scaler, metrics = train_and_evaluate(df, model_type='random_forest')
```

```
2025-02-06 18:45:37,844 - INFO - Preprocessing data for purchase prediction...
2025-02-06 18:45:37,866 - INFO - Training random_forest model...
2025-02-06 18:45:38,183 - INFO - Evaluating model performance...
2025-02-06 18:45:38,212 - INFO - Model Accuracy: 0.9994
2025-02-06 18:45:38,213 - INFO - Precision: 1.0000
2025-02-06 18:45:38,214 - INFO - Recall: 0.9873
2025-02-06 18:45:38,215 - INFO - F1 Score: 0.9936
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1628
           1       1.00      0.99      0.99        79

    accuracy                           1.00      1707
   macro avg       1.00      0.99      1.00      1707
weighted avg       1.00      1.00      1.00      1707
```

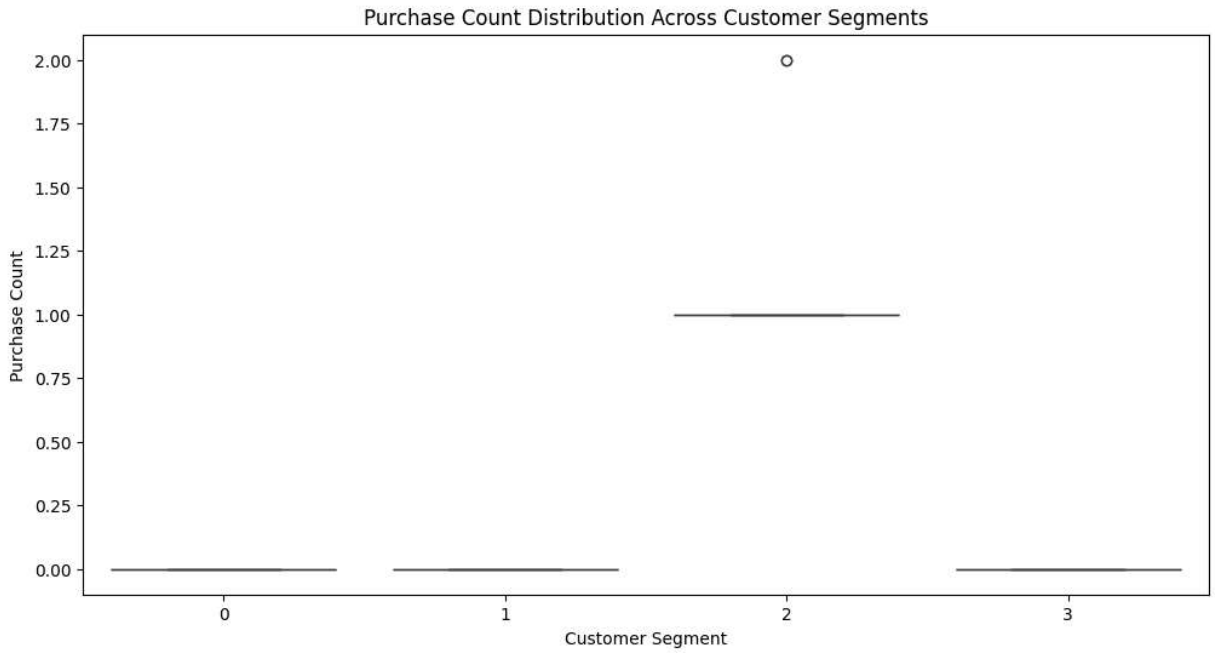
Customer Segmentation (Clustering)

In [10]:
```
# Import clustering functions
from customer_segmentation import segment_customers, plot_elbow_method, analyze_seg
import pandas as pd

df = pd.read_csv("featured.csv")
# Determine optimal K
plot_elbow_method(df)

# Apply clustering
df_segmented, kmeans_model, cluster_scaler = segment_customers(df, num_clusters=4)

# Analyze segments
analyze_segments(df_segmented)
```

```
2025-02-06 20:08:34,379 - INFO - Determining the optimal number of clusters using th
e Elbow Method.
```




```
2025-02-06 20:08:37,200 - INFO - Segmenting customers using K-Means clustering based
on their purchasing behavior.
2025-02-06 20:08:37,399 - INFO - Finished Sccessfully
2025-02-06 20:08:37,400 - INFO - Analyzing customer segments with visualization
2025-02-06 20:08:37,484 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:37,521 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```

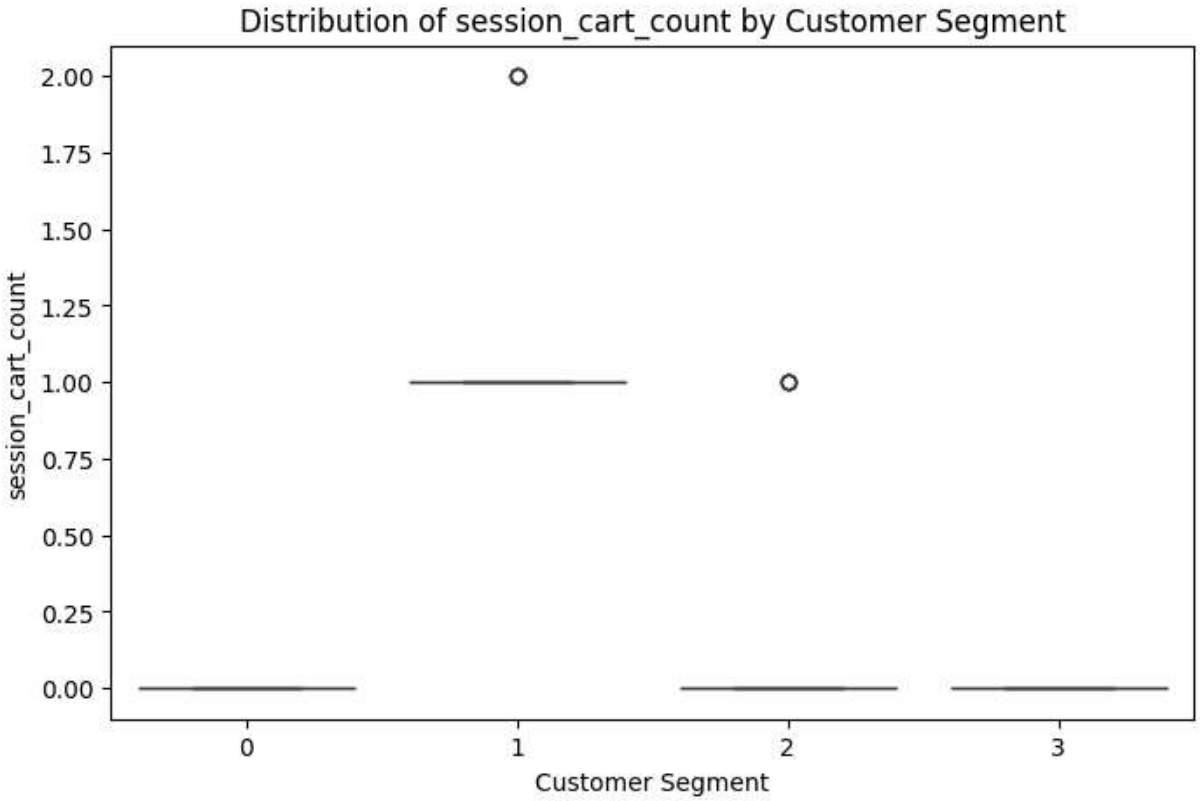



```
2025-02-06 20:08:37,737 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:37,771 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```

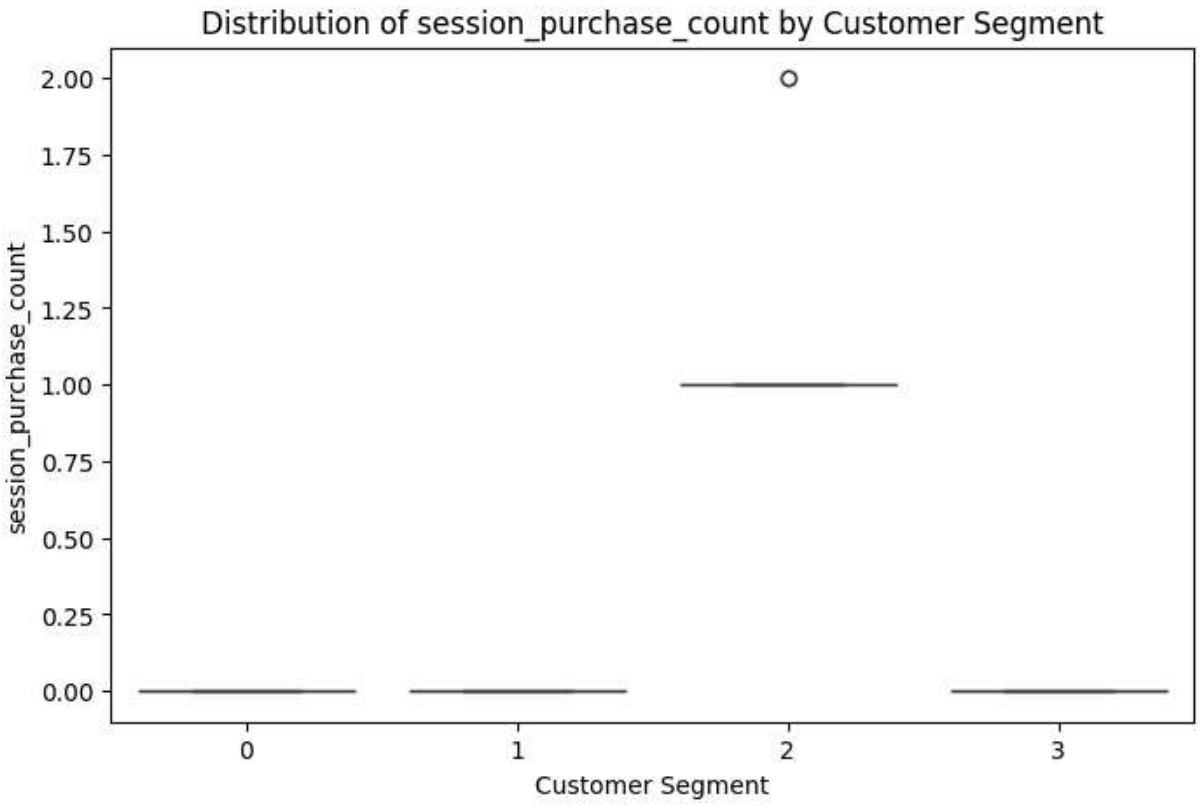
visualisation

In [11]:
```python
from customer_segmentation import visualize_segment_distribution

# Visualize segment distribution across important features
features = ['session_cart_count', 'session_purchase_count', 'session_view_count', '
visualize_segment_distribution(df_segmented, features)
```

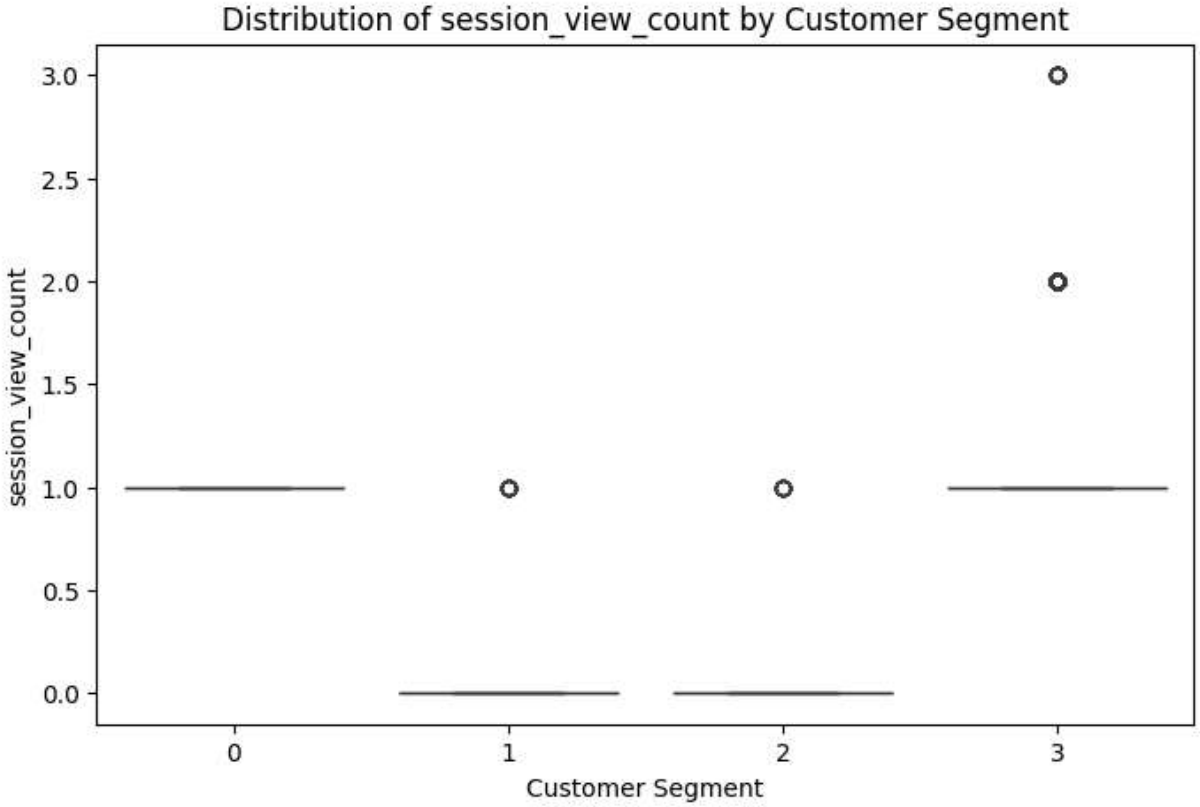
```
2025-02-06 20:08:42,190 - INFO - Visualizing the distribution of customer segments a
cross key features.
2025-02-06 20:08:42,257 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:42,290 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```



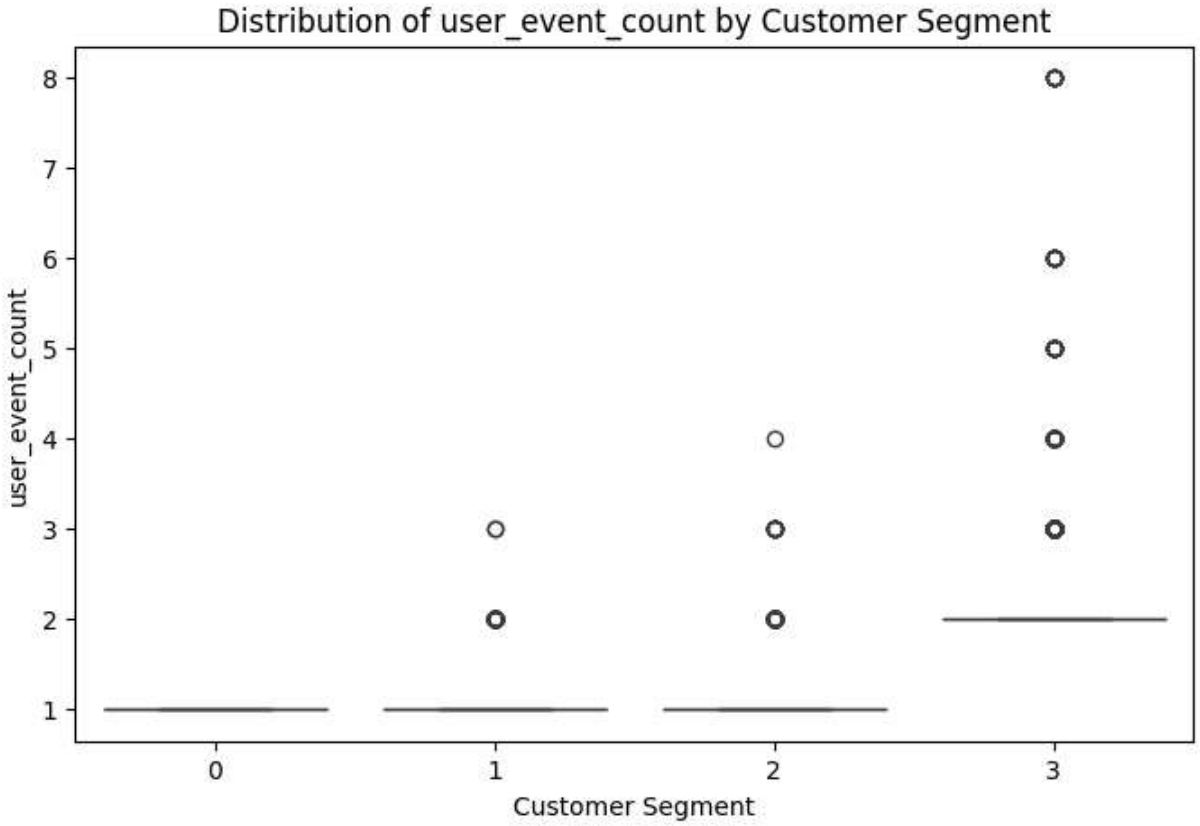

```
2025-02-06 20:08:42,570 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:42,625 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```



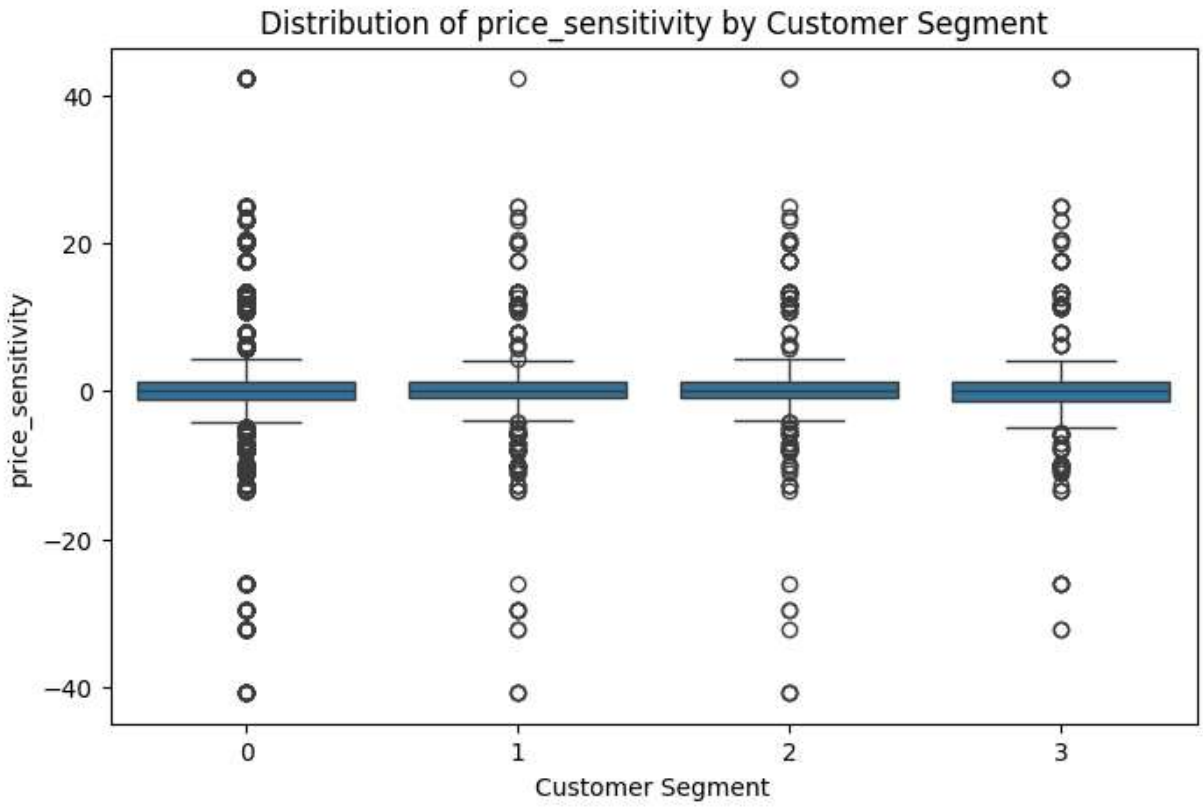

```
2025-02-06 20:08:42,831 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:42,865 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```



```
2025-02-06 20:08:43,057 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:43,088 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```

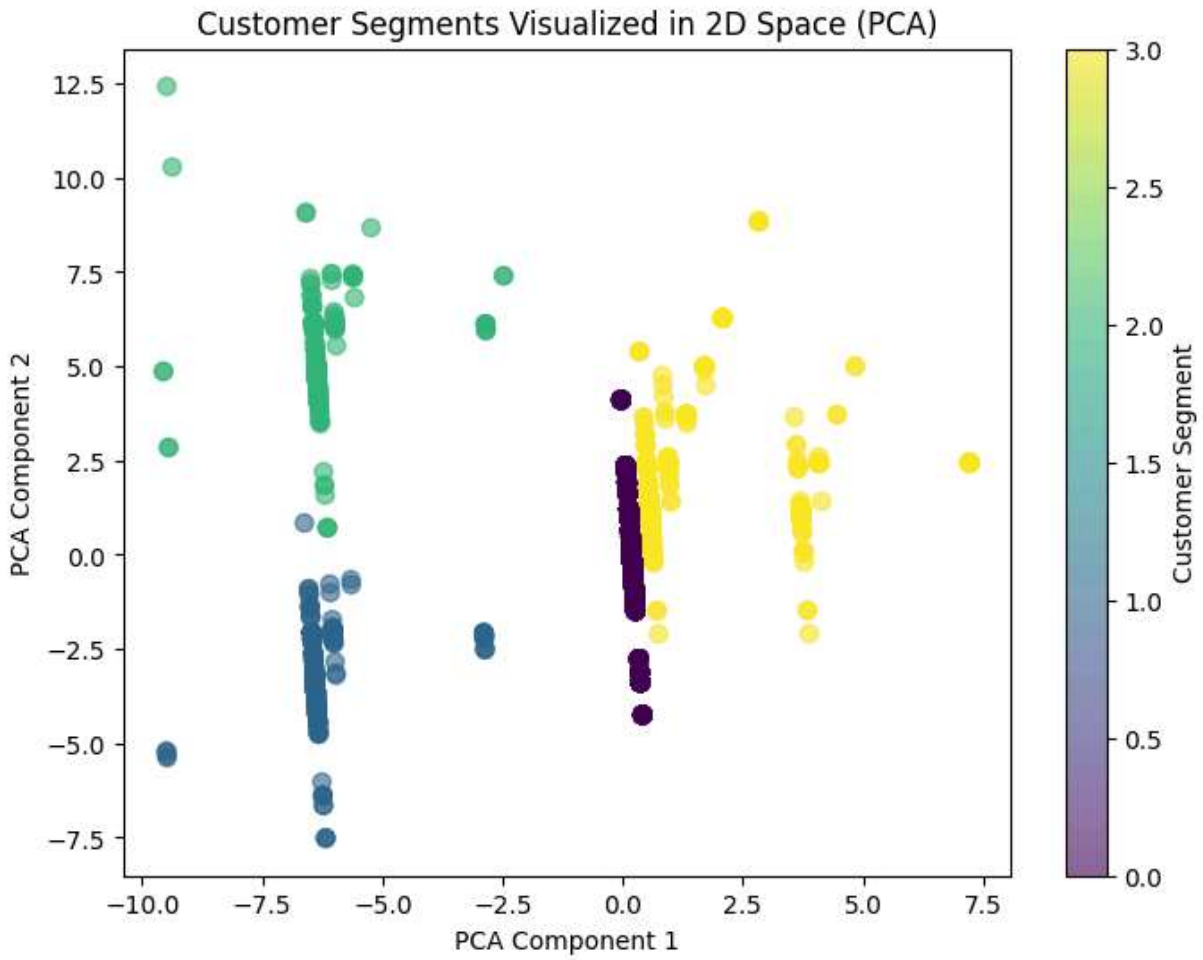



```
2025-02-06 20:08:43,274 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
2025-02-06 20:08:43,309 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```



In [12]:
```
from customer_segmentation import visualize_clusters_2d

# Visualize customer segments in 2D using PCA
visualize_clusters_2d(df_segmented, features)
```

```
2025-02-06 20:08:52,191 - INFO - Visualizing customer segments on a 2D plot using PC
A.
```

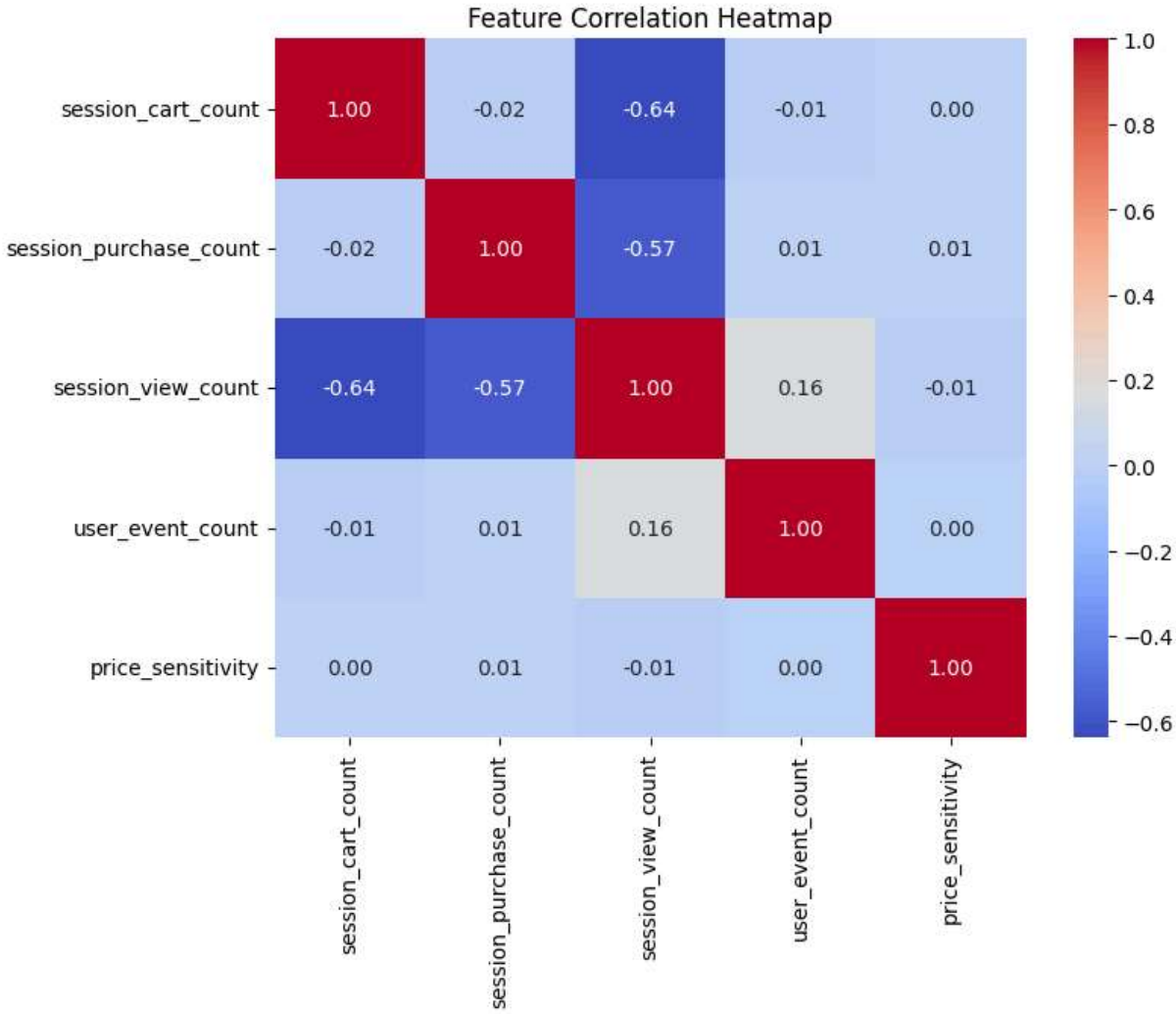


In [13]:
```
from customer_segmentation import calculate_silhouette_score

# Calculate and print silhouette score
calculate_silhouette_score(df_segmented, features, n_clusters =3)
```

```
2025-02-06 20:08:58,493 - INFO - Calculating the silhouette score for the clustering quality.
Silhouette Score for 3 clusters: 0.891
```

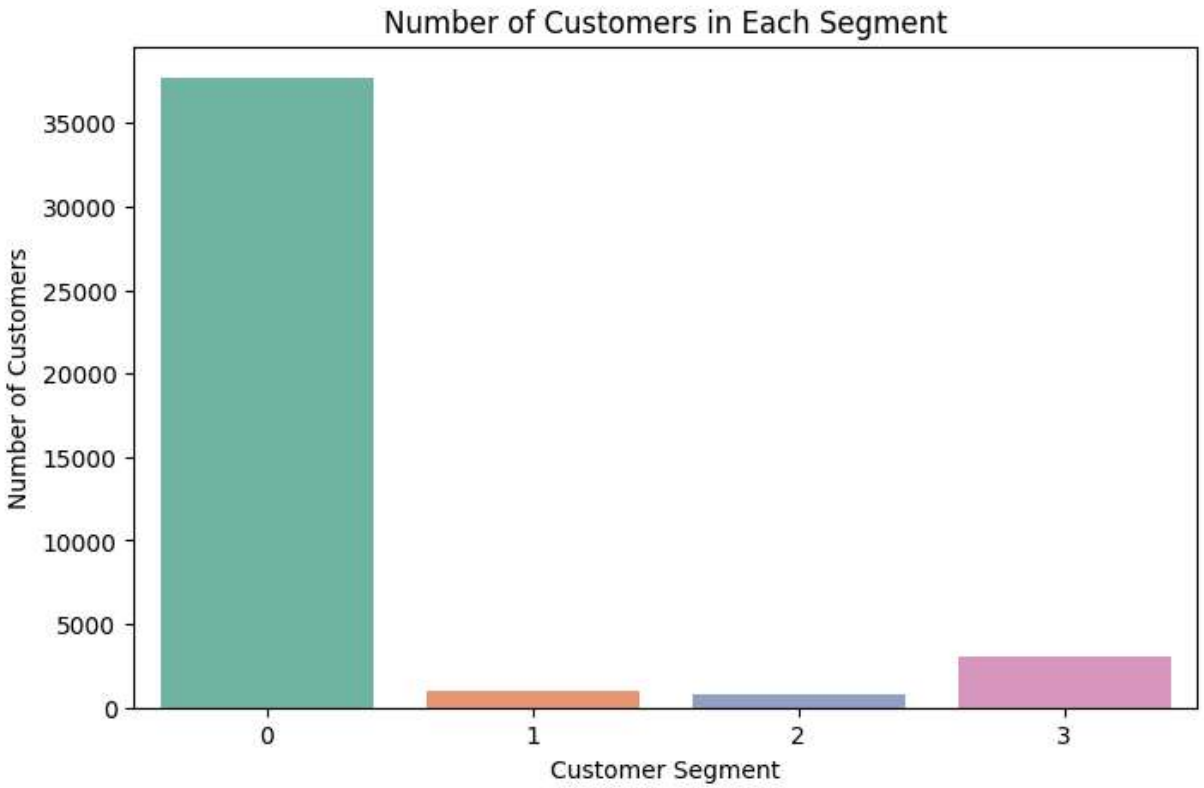
In [14]:
```
from customer_segmentation import plot_feature_correlation_heatmap

# Plot correlation heatmap of selected features
plot_feature_correlation_heatmap(df_segmented, features)
```

```
2025-02-06 20:09:35,000 - INFO - Plotting a heatmap of the correlations between selected features.
```



In [15]:
```
from customer_segmentation import visualize_cluster_size

# Visualize the number of customers in each segment
visualize_cluster_size(df_segmented)
```

```
2025-02-06 20:09:41,390 - INFO - Visualizing the number of customers in each cluste
r.
2025-02-06 20:09:41,414 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
c:\Users\Kgomi.DELL-K\Desktop\KIU\Data Science with Python\Final project\customer_se
gmentation.py:166: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=cluster_size.index, y=cluster_size.values, palette='Set2')
2025-02-06 20:09:41,431 - INFO - Using categorical units to plot a list of strings t
hat are all parsable as floats or dates. If these strings should be plotted as numbe
rs, cast to the appropriate data type before plotting.
```



Recommendation

In [ ]:
```
from recommendation import content_based_recommendations, collaborative_filtering_r
import pandas as pd

# Content-Based Recommendation
sample_product_id = df['product_id'].iloc[0]
recommended_products_content = content_based_recommendations(df, sample_product_id,

print(f"Content-Based Recommendations for Product {sample_product_id}: {recommended

# Collaborative Filtering Recommendation
sample_user_id = df['user_id'].iloc[0]
recommended_products_collab = collaborative_filtering_recommendations(sample_user_i

print(f"Collaborative Filtering Recommendations for User {sample_user_id}: {recomme
```

```
Content-Based Recommendations for Product 37900112: [np.int64(26400208), np.int64(38
900028), np.int64(17303026), np.int64(17700802), np.int64(38900028)]
Collaborative Filtering Recommendations for User 518673182: ['60400001', '1000978',
'1001588', '1002062', '1002098']
```

In [9]:
```
import random
import pandas as pd
from recommendation import collaborative_filtering_recommendations
from recommendation_check import precision_at_k, recall_at_k, hit_rate, plot_precis

df = pd.read_csv("featured.csv",  skiprows=lambda i: i % 10 != 0)  # Load precomput

# --- Evaluating Recommendation Performance ---
# Get actual purchases per user
```

```python
actual_purchases_dict = df[df['purchase'] == 1].groupby('user_id')['product_id'].ap

# Select 5 random users (We shall chek for 5 random users as my PC runs out of memo
random_users = random.sample(list(actual_purchases_dict.keys()), min(5, len(actual_

# Generate predicted recommendations for 5 users
predicted_products_dict = {user_id: collaborative_filtering_recommendations(user_id
                           for user_id in random_users}

# Convert to lists for evaluation
actual_purchases_list = [actual_purchases_dict[user_id] for user_id in random_users
predicted_products_list = list(predicted_products_dict.values())

# Compute Precision@K, Recall@K, and Hit Rate
precision_scores = [precision_at_k(actual, predicted, k=5) for actual, predicted in
recall_scores = [recall_at_k(actual, predicted, k=5) for actual, predicted in zip(a
hit_rates = [hit_rate(actual, predicted) for actual, predicted in zip(actual_purcha

print(f"Average Precision@5: {sum(precision_scores) / len(precision_scores):.2f}")
print(f"Average Recall@5: {sum(recall_scores) / len(recall_scores):.2f}")
print(f"Hit Rate: {sum(hit_rates) / len(hit_rates):.2f}")

# --- Visualizing Performance ---
plot_precision_recall_curve(actual_purchases_list, predicted_products_list, k_value
```

```
Average Precision@5: 0.00
Average Recall@5: 0.00
Hit Rate: 0.00
```