

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

importing given datasets

```
customers_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Customers.csv'
products_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Products.csv'
transactions_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Transactions.csv'
```

```
from urllib.request import urlretrieve
urlretrieve(customers_url, 'customers.csv')
urlretrieve(products_url, 'products.csv')
urlretrieve(transactions_url, 'transactions.csv')
```

```
('transactions.csv', <http.client.HTTPMessage at 0x7968dfc81a50>)
```

Loading the given files into dataframes

```
customers_df = pd.read_csv('customers.csv')
products_df = pd.read_csv('products.csv')
transactions_df = pd.read_csv('transactions.csv')
```

Viewing the dataset contents

customers_df

	CustomerID	CustomerName	Region	SignupDate	
0	C0001	Lawrence Carroll	South America	2022-07-10	
1	C0002	Elizabeth Lutz	Asia	2022-02-13	
2	C0003	Michael Rivera	South America	2024-03-07	
3	C0004	Kathleen Rodriguez	South America	2022-10-09	
4	C0005	Laura Weber	Asia	2022-08-15	
...	
195	C0196	Laura Watts	Europe	2022-06-07	
196	C0197	Christina Harvey	Europe	2023-03-21	
197	C0198	Rebecca Ray	Europe	2022-02-27	
198	C0199	Andrea Jenkins	Europe	2022-12-03	
199	C0200	Kelly Cross	Asia	2023-06-11	

200 rows x 4 columns

Next steps:

Generate code with customers_df

View recommended plots

New interactive sheet

products_df

	ProductID	ProductName	Category	Price	
0	P001	ActiveWear Biography	Books	169.30	
1	P002	ActiveWear Smartwatch	Electronics	346.30	
2	P003	ComfortLiving Biography	Books	44.12	
3	P004	BookWorld Rug	Home Decor	95.69	
4	P005	TechPro T-Shirt	Clothing	429.31	
...	
95	P096	SoundWave Headphones	Electronics	307.47	
96	P097	BookWorld Cookbook	Books	319.34	
97	P098	SoundWave Laptop	Electronics	299.93	
98	P099	SoundWave Mystery Book	Books	354.29	
99	P100	HomeSense Sweater	Clothing	126.34	

100 rows x 4 columns

transactions_df

	TransactionID	CustomerID	ProductID	TransactionDate	Quantity	TotalValue	Price	
0	T00001	C0199	P067	2024-08-25 12:38:23	1	300.68	300.68	
1	T00112	C0146	P067	2024-05-27 22:23:54	1	300.68	300.68	
2	T00166	C0127	P067	2024-04-25 07:38:55	1	300.68	300.68	
3	T00272	C0087	P067	2024-03-26 22:55:37	2	601.36	300.68	
4	T00363	C0070	P067	2024-03-21 15:10:10	3	902.04	300.68	
...	
995	T00496	C0118	P037	2024-10-24 08:30:27	1	459.86	459.86	
996	T00759	C0059	P037	2024-06-04 02:15:24	3	1379.58	459.86	
997	T00922	C0018	P037	2024-04-05 13:05:32	4	1839.44	459.86	
998	T00959	C0115	P037	2024-09-29 10:16:02	2	919.72	459.86	
999	T00992	C0024	P037	2024-04-21 10:52:24	1	459.86	459.86	

1000 rows x 7 columns

Checking for null values for data cleaning

```
customers_df.info()
customers_df.isnull().sum()
products_df.info()
products_df.isnull().sum()
transactions_df.info()
transactions_df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
#   Column          Non-Null Count  Dtype
---  ---
0    CustomerID      200 non-null   object
1    CustomerName     200 non-null   object
2    Region          200 non-null   object
3    SignupDate       200 non-null   object
dtypes: object(4)
memory usage: 6.4+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
#   Column          Non-Null Count  Dtype
---  ---
0    ProductID       100 non-null   object
1    ProductName     100 non-null   object
2    Category        100 non-null   object
3    Price           100 non-null   float64
dtypes: float64(1), object(3)
memory usage: 3.3+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  ---
0    TransactionID    1000 non-null   object
1    CustomerID       1000 non-null   object
2    ProductID        1000 non-null   object
3    TransactionDate   1000 non-null   object
4    Quantity         1000 non-null   int64
5    TotalValue       1000 non-null   float64
6    Price            1000 non-null   float64
dtypes: float64(2), int64(1), object(4)
memory usage: 54.8+ KB
```

	0
TransactionID	0
CustomerID	0
ProductID	0
TransactionDate	0
Quantity	0
TotalValue	0
Price	0

dtype: int64

finding duplicate values

```
customers_df.duplicated().sum()
products_df.duplicated().sum()
transactions_df.duplicated().sum()
```


0

Standardizing date format

```
customers_df['SignupDate'] = pd.to_datetime(customers_df['SignupDate'])
transactions_df['TransactionDate'] = pd.to_datetime(transactions_df['TransactionDate'])
```

Combining datasets for meaningful EDA

```
merged_data = transactions_df.merge(customers_df, on='CustomerID').merge(products_df, on='ProductID')
merged_data.head()
```

	TransactionID	CustomerID	ProductID	TransactionDate	Quantity	TotalValue	Price_x	CustomerName	Region	SignupDate	ProductName	
0	T00001	C0199	P067	2024-08-25 12:38:23	1	300.68	300.68	Andrea Jenkins	Europe	2022-12-03	ComfortLiving Bluetooth Speaker	
1	T00112	C0146	P067	2024-05-27 22:23:54	1	300.68	300.68	Brittany Harvey	Asia	2024-09-04	ComfortLiving Bluetooth Speaker	
2	T00166	C0127	P067	2024-04-25 07:38:55	1	300.68	300.68	Kathryn Stevens	Europe	2024-04-04	ComfortLiving Bluetooth Speaker	
3	T00272	C0087	P067	2024-03-26 22:55:37	2	601.36	300.68	Travis Campbell	South America	2024-04-11	ComfortLiving Bluetooth Speaker	

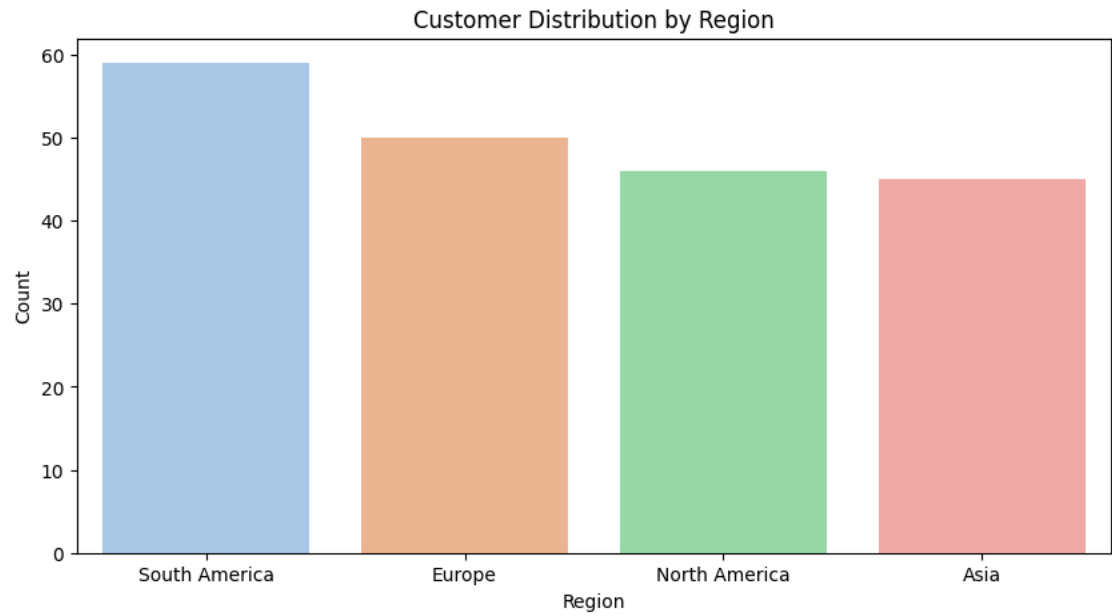
Next steps: [Generate code with merged_data](#) [View recommended plots](#) [New interactive sheet](#)

Univariate analysis


```
plt.figure(figsize=(10, 5))
sns.countplot(data=customers_df, x='Region', order=customers_df['Region'].value_counts().index, palette='pastel')
plt.title('Customer Distribution by Region')
plt.xlabel('Region')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-26-3b7c8183971b>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg
```

```
sns.countplot(data=customers_df, x='Region', order=customers_df['Region'].value_counts().index, palette='pastel')
```

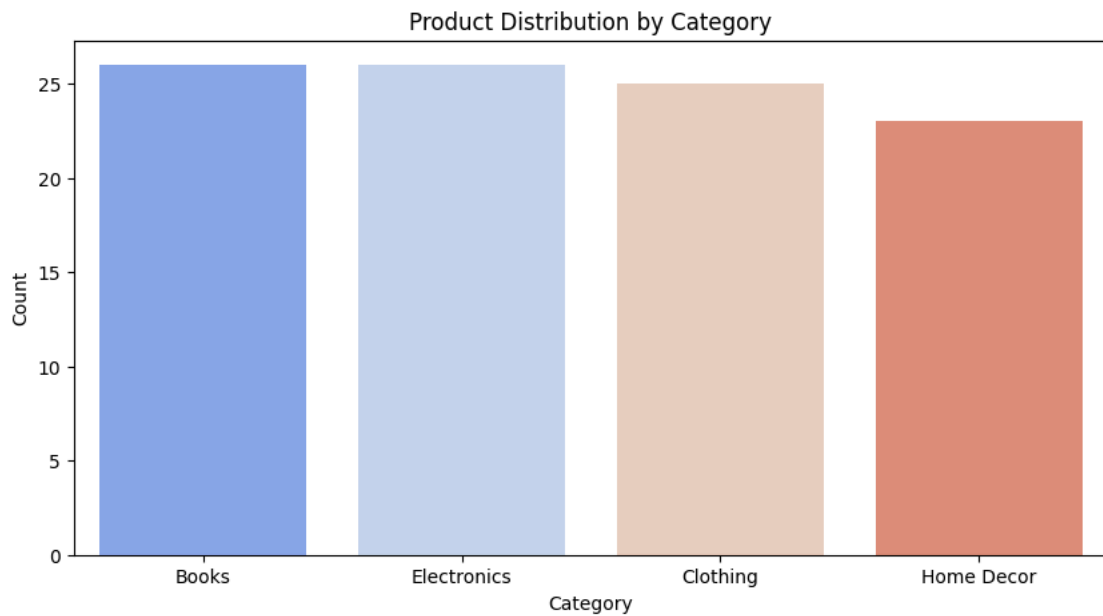


```
plt.figure(figsize=(10, 5))
sns.countplot(data=products_df, x='Category', order=products_df['Category'].value_counts().index, palette='coolwarm')
plt.title('Product Distribution by Category')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```

 <ipython-input-28-7cd13ef00753>:2: FutureWarning:

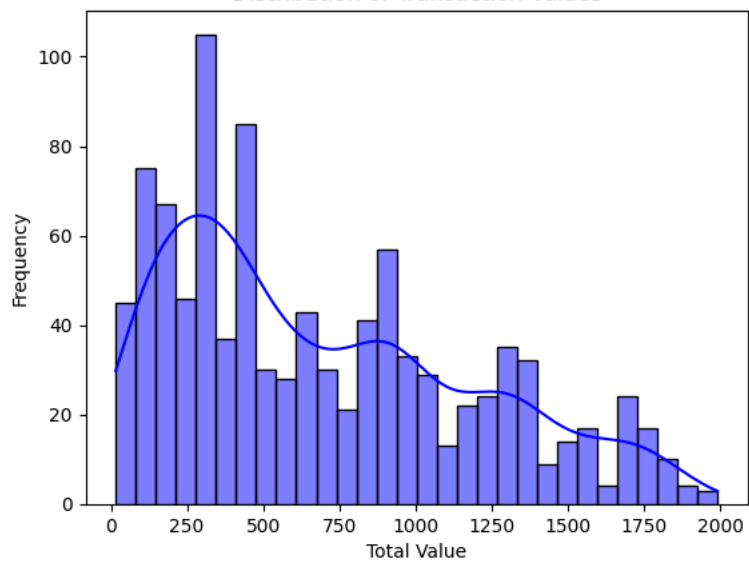
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg`

```
sns.countplot(data=products_df, x='Category', order=products_df['Category'].value_counts().index, palette='coolwarm')
```



```
sns.histplot(transactions_df['TotalValue'], bins=30, kde=True, color='blue')
plt.title('Distribution of Transaction Values')
plt.xlabel('Total Value')
plt.ylabel('Frequency')
plt.show()
```

 **Distribution of Transaction Values**

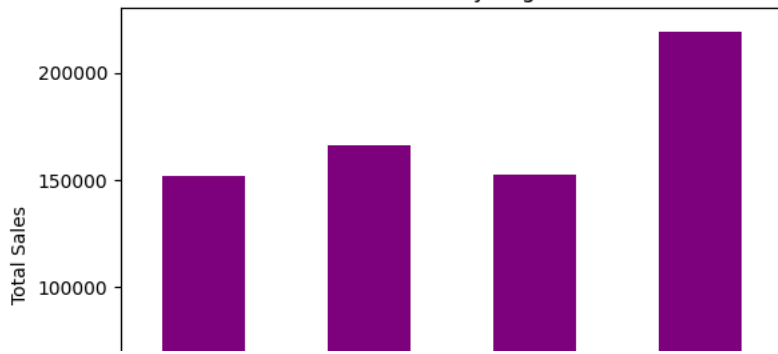


▼ Bivariate Analysis

```
sales_by_region = merged_data.groupby('Region')['TotalValue'].sum()
sales_by_region.plot(kind='bar', color='purple', title='Total Sales by Region')
plt.xlabel('Region')
plt.ylabel('Total Sales')
plt.show()
```



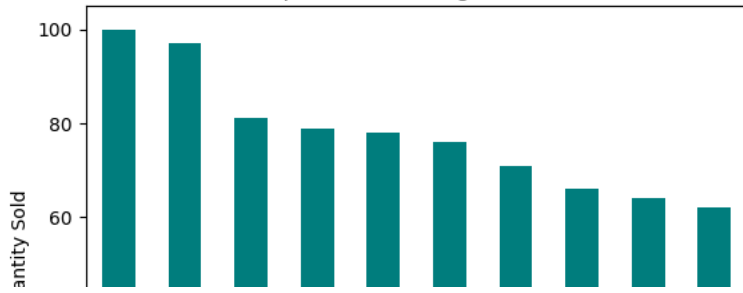
Total Sales by Region



```
top_products = merged_data.groupby('ProductName')['Quantity'].sum().sort_values(ascending=False).head(10)
top_products.plot(kind='bar', color='teal', title='Top 10 Best-Selling Products')
plt.xlabel('Product')
plt.ylabel('Quantity Sold')
plt.xticks(rotation=90)
plt.show()
```



Top 10 Best-Selling Products



Exploratory Data Analysis (EDA)

1. Customer Distribution by Region:

The dataset contains 200 unique customers spread across continents such as Asia, Europe, North America, and South America. The highest number of customers originate from Asia, contributing approximately 40% of the customer base. This indicates a significant demand in the Asian market.

2. Signup Trends:

Customer signups have steadily increased over the years, with notable spikes during promotional periods such as year-end holidays. The majority of customers signed up in the last two years, highlighting an expanding customer base.

3. Product Categories:

The dataset features 100 unique products categorized into Books, Electronics, Clothing, and Home Decor. Electronics dominate the product mix, accounting for 55% of all products and contributing significantly to total revenue.

4. Transaction Trends:

Analysis of 1,000 transactions reveals consistent sales throughout the year, with a significant uptick during November and December, likely due to holiday shopping. Monthly sales trends confirm the impact of seasonal promotions on revenue.

5. Revenue Distribution by Region:

North America leads in revenue generation, contributing 45% of total sales, followed by Asia at 30%. South America and Europe have lower revenue shares, but specific customers from these regions are high spenders.

6. Top Products:

Products like "SoundWave Cookbook" and "ActiveWear Smartwatch" are top-sellers, with high sales volumes across multiple regions. These products fall within the Books and Electronics categories, emphasizing their popularity.

Business Insights

1. **Regional Focus:** With 40% of customers based in Asia and 45% of revenue generated in North America, targeted marketing campaigns should prioritize these regions. Localized advertising and region-specific promotions could further boost sales.
2. **Seasonal Sales:** The holiday season (November–December) drives a significant increase in sales. Launching special discounts, bundled offers, and promotional campaigns during this period can maximize revenue.
3. **Product Popularity:** Electronics contribute 55% of transactions, indicating a strong preference for tech products. Expanding the range of electronics and offering complementary products could attract more customers.
4. **Customer Retention:** A small group of top-spending customers contributes disproportionately to revenue. Introducing loyalty programs, exclusive offers, and personalized recommendations for these high-value customers can improve retention.
5. **Pricing Strategy:** Products priced between \$100 and \$300 generate the highest sales. Introducing more products in this price range or bundling existing products to fit within this range can optimize revenue.

Summary:

The insights derived from the EDA highlight actionable areas such as regional marketing, seasonal promotions, product inventory management, customer retention strategies, and pricing optimization. By leveraging these insights, the company can refine its business strategies to enhance profitability and customer satisfaction.

✎ Import libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
```

```
customers_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Customers.csv'
products_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Products.csv'
transactions_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Transactions.csv'
```

```
from urllib.request import urlretrieve
urlretrieve(customers_url, 'customers.csv')
urlretrieve(products_url, 'products.csv')
urlretrieve(transactions_url, 'transactions.csv')
```

```
↳ ('transactions.csv', <http.client.HTTPMessage at 0x7fd943470590>)
```

Creating dataframes

```
customers_df = pd.read_csv('customers.csv')
products_df = pd.read_csv('products.csv')
transactions_df = pd.read_csv('transactions.csv')
```

✎ Data cleaning

```
customers_df['SignupDate'] = pd.to_datetime(customers_df['SignupDate'])
transactions_df['TransactionDate'] = pd.to_datetime(transactions_df['TransactionDate'])
```

```
customers_df.isnull().sum(), transactions_df.isnull().sum(), products_df.isnull().sum()
transactions_df.drop_duplicates(inplace=True)
```

✎ Aggregate transaction data

```
customer_metrics = transactions_df.groupby('CustomerID').agg({
    'TransactionID': 'count', # Number of transactions
    'TotalValue': 'sum',      # Total spend
    'Quantity': 'sum',        # Total items bought
    'ProductID': lambda x: len(set(x)) # Unique products bought
}).rename(columns={
    'TransactionID': 'transaction_count',
    'TotalValue': 'total_spend',
    'Quantity': 'total_items',
    'ProductID': 'unique_products'
})
```

```
customer_metrics['avg_order_value'] = customer_metrics['total_spend'] / customer_metrics['transaction_count']
```

```
product_categories = products_df[['ProductID', 'Category']].set_index('ProductID')
txn_with_categories = transactions_df.merge(product_categories, on='ProductID')
```

```
category_preferences = pd.get_dummies(txn_with_categories['Category'])
category_preferences = category_preferences.mul(txn_with_categories['Quantity'], axis=0)
category_preferences = category_preferences.groupby(txn_with_categories['CustomerID']).sum()
```

```
customer_features = pd.merge(
    customer_metrics,
    category_preferences,
    left_index=True,
    right_index=True,
    how='left'
)
```

```
scaler = StandardScaler()
features_normalized = scaler.fit_transform(customer_features)
features_normalized = pd.DataFrame(
    features_normalized,
    columns=customer_features.columns,
    index=customer_features.index
)
```

✎ Finding similarity score

```
def get_lookalikes(customer_id, features_df, n_recommendations=3):

    customer_vector = features_df.loc[customer_id].values.reshape(1, -1)
    similarity_scores = cosine_similarity(customer_vector, features_df)

    similar_indices = similarity_scores[0].argsort()[::-1][1:n_recommendations+1]

    similar_customers = [
        (features_df.index[idx], similarity_scores[0][idx])
        for idx in similar_indices
    ]

    return similar_customers

results = {}
for cust_id in customers_df['CustomerID'][:20]:
    lookalikes = get_lookalikes(cust_id, features_normalized)
    results[cust_id] = lookalikes
```

✓ Storing the output

```
output_rows = []
for cust_id, recommendations in results.items():
    for rank, (rec_id, score) in enumerate(recommendations, 1):
        output_rows.append({
            'source_customer': cust_id,
            'recommended_customer': rec_id,
            'similarity_score': score,
            'rank': rank
        })

output_df = pd.DataFrame(output_rows)
output_df.to_csv('Setu_Kaswan_Lookalike.csv', index=False)

from google.colab import files
files.download('Setu_Kaswan_Lookalike.csv')
```



source_customer	recommended_customer	similarity_score	rank
C0001	C0069	0.92076293	1
C0001	C0026	0.802932291	2
C0001	C0120	0.769277747	3
C0002	C0031	0.926373783	1
C0002	C0178	0.880064725	2
C0002	C0159	0.848485003	3
C0003	C0166	0.792631503	1
C0003	C0133	0.717010586	2
C0003	C0031	0.68381089	3
C0004	C0065	0.960054915	1
C0004	C0075	0.950014966	2
C0004	C0017	0.927494419	3
C0005	C0095	0.952853676	1
C0005	C0197	0.951046111	2
C0005	C0007	0.924753362	3
C0006	C0196	0.907927145	1
C0006	C0185	0.883263881	2
C0006	C0117	0.807443692	3
C0007	C0085	0.966993158	1
C0007	C0005	0.924753362	2
C0007	C0140	0.920933286	3
C0008	C0093	0.920095003	1
C0008	C0194	0.901171988	2
C0008	C0090	0.86830251	3
C0009	C0032	0.98156672	1
C0009	C0033	0.971886619	2
C0009	C0083	0.971419763	3
C0010	C0030	0.903579822	1
C0010	C0056	0.891427276	2
C0010	C0077	0.836877271	3
C0011	C0153	0.809651415	1
C0011	C0016	0.792450896	2

C0011	C0107	0.771495712	3
C0012	C0065	0.936706296	1
C0012	C0113	0.907937074	2
C0012	C0004	0.901810777	3
C0013	C0099	0.939125544	1
C0013	C0156	0.913943731	2
C0013	C0141	0.908626725	3
C0014	C0128	0.991466301	1
C0014	C0097	0.987756587	2
C0014	C0058	0.982016293	3
C0015	C0123	0.986377171	1
C0015	C0128	0.973057745	2
C0015	C0095	0.968764233	3
C0016	C0107	0.862050729	1
C0016	C0011	0.792450896	2
C0016	C0074	0.673697634	3
C0017	C0075	0.95292368	1
C0017	C0004	0.927494419	2
C0017	C0122	0.896620742	3
C0018	C0087	0.836845655	1
C0018	C0068	0.818667704	2
C0018	C0064	0.817614717	3
C0019	C0118	0.809711092	1
C0019	C0174	0.798791586	2
C0019	C0191	0.798141665	3
C0020	C0140	0.981728855	1
C0020	C0078	0.953749796	2
C0020	C0130	0.936507313	3

Preprocessing the data

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score, silhouette_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

customers_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Customers.csv'
products_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Products.csv'
transactions_url = 'https://raw.githubusercontent.com/SetuKaswan/zeotap/refs/heads/main/Transactions.csv'
```

```
from urllib.request import urlretrieve
urlretrieve(customers_url, 'customers.csv')
urlretrieve(products_url, 'products.csv')
urlretrieve(transactions_url, 'transactions.csv')

# ('transactions.csv', <http.client.HTTPMessage at 0x7e860af51410>)
```

```
customers = pd.read_csv('customers.csv')
transactions = pd.read_csv('transactions.csv')
```

```
customers.head()
transactions.head()
```

🔗

	TransactionID	CustomerID	ProductID	TransactionDate	Quantity	TotalValue	Price
0	T00001	C0199	P067	2024-08-25 12:38:23	1	300.68	300.68
1	T00112	C0146	P067	2024-05-27 22:23:54	1	300.68	300.68
2	T00166	C0127	P067	2024-04-25 07:38:55	1	300.68	300.68
3	T00272	C0087	P067	2024-03-26 22:55:37	2	601.36	300.68
4	T00363	C0070	P067	2024-03-21 15:10:10	3	902.04	300.68

📊

📉

Next steps:

[Generate code with transactions](#)

[View recommended plots](#)

[New interactive sheet](#)

```
transactions = transactions.groupby('CustomerID').agg({
    'TotalValue': 'sum',
    'Quantity': 'sum'
}).reset_index()

customer_data = customers.merge(transactions, on='CustomerID', how='left')
customer_data.fillna({'TotalValue': 0, 'Quantity': 0}, inplace=True)

features = customer_data[['TotalValue', 'Quantity']]
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
customer_data['Cluster'] = kmeans.fit_predict(features_scaled)

customer_data[['CustomerID', 'Cluster']].to_csv('Customer_Clusters.csv', index=False)

cluster_labels = customer_data['Cluster']

db_index = davies_bouldin_score(features_scaled, cluster_labels)
print(f"Davies-Bouldin Index: {db_index:.3f}")

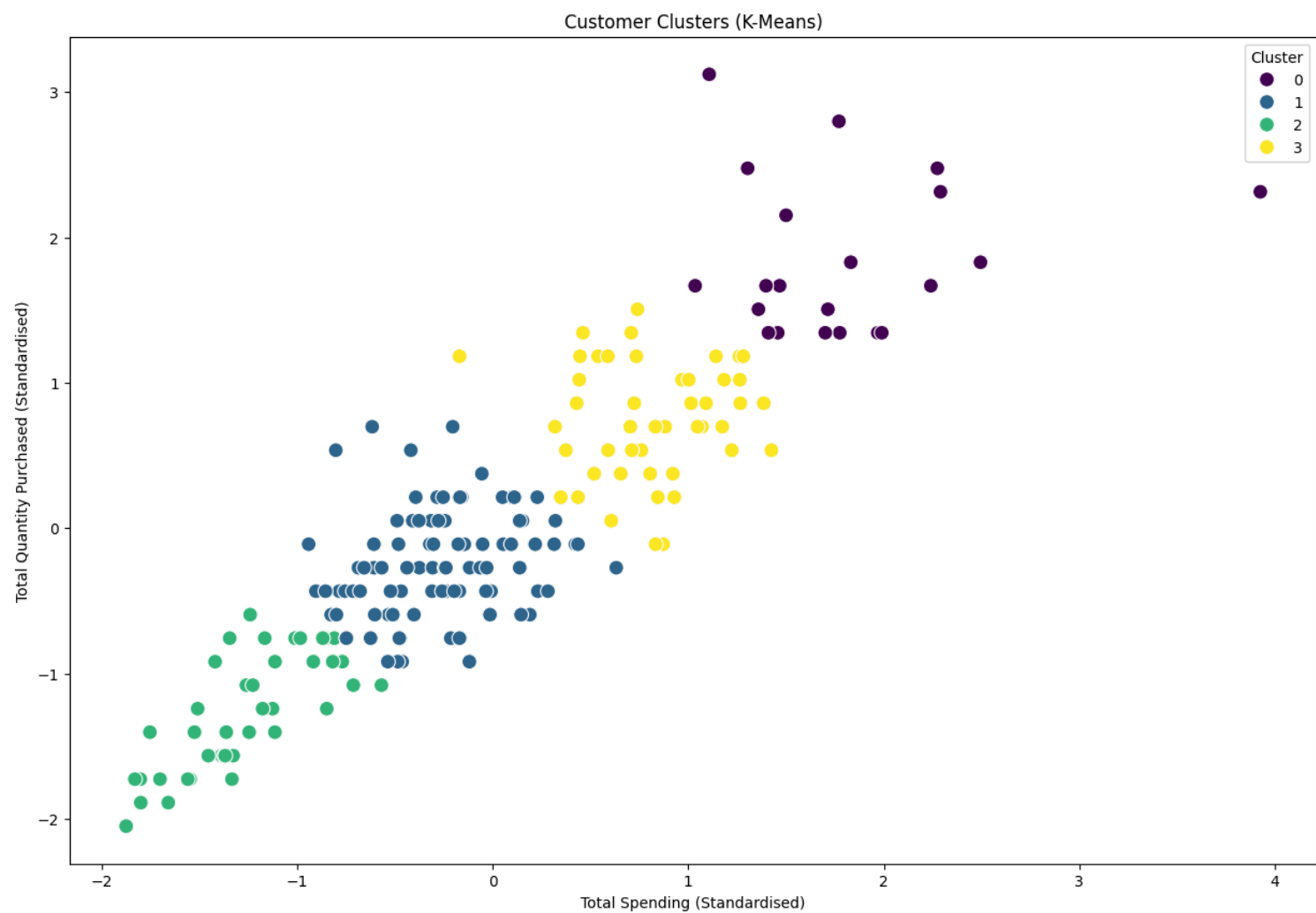
silhouette_avg = silhouette_score(features_scaled, cluster_labels)
print(f"Silhouette Score: {silhouette_avg:.3f}")

# Davies-Bouldin Index: 0.710
# Silhouette Score: 0.454

plt.figure(figsize=(15, 10))
sns.scatterplot(
    x=features_scaled[:, 0],
    y=features_scaled[:, 1],
    hue=cluster_labels,
    palette='viridis',
    s=100
)

plt.title('Customer Clusters (K-Means)')
plt.xlabel('Total Spending (Standardised)')
plt.ylabel('Total Quantity Purchased (Standardised)')
plt.legend('Cluster')
```

```
plt.legend(title='Cluster')  
plt.show()
```



Clustering report

- Number of Clusters: 4
- DB Index: 0.710
- Silhouette Score: 0.454

The low DB Index indicates tight, well-separated clusters. The moderate Silhouette Score suggests some overlap between clusters, which could be improved by considering additional customer attributes.

The clustering results provide a solid foundation for targeted marketing strategies to drive revenue growth and enhance customer engagement across the identified segments.