

Project Title: A/B Testing for Customer Segmentation and Targeting

Introduction

Objective: The purpose of this project is to perform customer segmentation and evaluate targeted marketing strategies through A/B testing. By clustering customers based on purchasing behaviors, I can identify distinct customer groups and analyze the impact of specific marketing strategies within each segment. This process aims to provide actionable insights for optimizing customer engagement and boosting sales.

Tools Used: This project leverages Python, with the Pandas library for data manipulation, Seaborn and Matplotlib for visualizations, and Scipy for statistical analysis. K-means clustering is applied for segmentation, while T-tests are used to evaluate the statistical significance of the A/B testing results.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import OneHotEncoder
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
from scipy.stats import ttest_ind
warnings.filterwarnings('ignore')
```

Data Loading and Exploration

In this section, I loaded the initial dataset and conducted exploratory data analysis (EDA) to understand the structure and characteristics of the data. This includes checking for missing values, data types, and summary statistics.

Steps:

- 1. Loaded the dataset using Pandas and inspect the first few rows to understand its structure.
- 2. Displaying data types and non-null counts.
- 3. Used describe() to generate summary statistics for numerical features.

```
df = pd.read_csv('/content/drive/MyDrive/Sentiment2.csv')
df.head()
```

↗

	id	first_name	age	gender	state	traffic_source	order_id	status	created_at	product_id	sale_price	num_of_item
0	1260	Pamela	12	F	Hunan	Email	1588	Complete	2024-01-24 17:11:00 UTC	10467	34.000000	1
1	1260	Pamela	12	F	Hunan	Email	1589	Complete	2024-01-30 17:11:00 UTC	3385	49.990002	1
2	1350	Virginia	12	F	Jiangsu	Search	1695	Complete	2022-08-04 11:13:00 UTC	1016	67.959999	1
3	1550	Glenda	12	F	Minnesota	Organic	1960	Complete	2023-08-05 00:26:00 UTC	8788	219.000000	1

```
df.info()
```

↗

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 154021 entries, 0 to 154020
Data columns (total 14 columns):
#   Column              Non-Null Count  Dtype
---  -
0   id                  154021 non-null  int64
```

```

1  first_name      154021 non-null object
2  age            154021 non-null int64
3  gender         154021 non-null object
4  state          154021 non-null object
5  traffic_source 154021 non-null object
6  order_id       154021 non-null int64
7  status         154021 non-null object
8  created_at     154021 non-null object
9  product_id     154021 non-null int64
10 sale_price     154021 non-null float64
11 num_of_item    154021 non-null int64
12 category       154021 non-null object
13 brand          153928 non-null object
dtypes: float64(1), int64(5), object(8)
memory usage: 16.5+ MB

```

```
df.isnull().sum()
```

```

0
id      0
first_name  0
age      0
gender    0
state     0
traffic_source  0
order_id   0
status     0
created_at  0
product_id  0
sale_price  0
num_of_item  0
category    0
brand      93

```

```
dtype: int64
```

✓ Data Cleaning and Preparation

Before proceeding with analysis, I cleaned the data to handle missing values and filtered out irrelevant records. Data cleaning ensures that only relevant and complete data points are used in further steps, improving the accuracy of our analysis.

Steps:

1. **Handle Missing Values:** Removing or filling missing values in essential columns such as `age`, `gender`, and `sale_price`.
2. **Filter Rows:** Retained only rows where `status` is 'Complete' or 'Shipped' to focus on actual purchases.

```

df = df.dropna(subset=['age', 'gender', 'sale_price'])
df = df[df['status'].isin(['Complete', 'Shipped'])]

df['created_at'] = df['created_at'].replace('', np.nan)
df.dropna(subset=['created_at'], inplace=True)
df['created_at'] = df['created_at'].str.replace(' UTC', '', regex=False)
df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce')

```

```

df['purchase_date'] = df['created_at'].dt.date
df['purchase_time'] = df['created_at'].dt.time

```

```


def get_time_of_day(hour):
    if pd.isna(hour):
        return 'Unknown'
    elif 5 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 17:

```

```
        return 'Afternoon'
    elif 17 <= hour < 21:
        return 'Evening'
    else:
        return 'Night'

df['time_of_day'] = df['created_at'].dt.hour.apply(get_time_of_day)

df.head()
```



	id	first_name	age	gender	state	traffic_source	order_id	status	created_at	product_id	sale_price	num_of_item
0	1260	Pamela	12	F	Hunan	Email	1588	Complete	2024-01-24 17:11:00	10467	34.000000	1
1	1260	Pamela	12	F	Hunan	Email	1589	Complete	2024-01-30 17:11:00	3385	49.990002	1
2	1350	Virginia	12	F	Jiangsu	Search	1695	Complete	2022-08-04 11:13:00	1016	67.959999	1
3	1550	Glenda	12	F	Minnesota	Organic	1960	Complete	2023-08-05 00:26:00	8788	219.000000	1
4	1550	Glenda	12	F	Minnesota	Organic	1961	Complete	2024-08-09 00:26:00	12805	112.000000	1

Next steps:

Generate code with df

 View recommended plots

New interactive sheet

```
df['Total_spent'] = df['sale_price'] * df['num_of_item']
```

Feature Engineering

To better understand and segment the customers, I created several new features based on existing data. These engineered features capture key aspects of customer behavior, which can then be used for clustering and testing.


Features Created:



- 1. **Average Order Value (AOV):** Calculated by dividing the total amount spent by the number of orders.
- 2. **Preferred Product Category:** The product category each customer purchases most frequently.
- 3. **Most Frequent Traffic Source:** The most common traffic source for each customer (e.g., search, email).
- 4. **Purchase Recency:** Number of days since the last purchase. This helps identify active vs. inactive customers.
- 5. **Order Frequency:**
- 6. **Time of Day Analysis (morning, afternoon, evening, night)**

Average Order Value (AOV)

```
average_order_value = df.groupby('id').apply(
    lambda x: (x['sale_price'] * x['num_of_item']).sum() / len(x)
).reset_index(name='avg_order_value')
```

```
average_order_value.head()
```



	id	avg_order_value	
0	1	78.000000	
1	3	51.189999	
2	5	19.990000	
3	7	150.500000	
4	10	112.952000	

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

▼ Preferred Product Category

```
# Product Category Preferences: Identify the product categories that each customer purchases most frequently
category_preferences = df.groupby(['id', 'category']).size().reset_index(name='category_count')
category_preferences = category_preferences.loc[category_preferences.groupby('id')['category_count'].idxmax()]
category_preferences = category_preferences[['id', 'category']].rename(columns={'category': 'Preferred_Category'})

category_preferences.head()
```

	id	Preferred_Category
0	1	Maternity
1	3	Tops & Tees
2	5	Maternity
3	7	Jeans
7	10	Sweaters

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

▼ Most Frequent Traffic Source

```
# Assuming you have a 'traffic_source' column in your DataFrame
customer_traffic_source = df.groupby('id')['traffic_source'].agg(pd.Series.mode).reset_index()
customer_traffic_source = customer_traffic_source.rename(columns={'traffic_source': 'most_frequent_traffic_source'})
customer_traffic_source.head()
```

	id	most_frequent_traffic_source
0	1	Facebook
1	3	Search
2	5	Search
3	7	Search
4	10	Search

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

▼ Purchase Recency

```
last_purchase = df.groupby('id')['purchase_date'].max().reset_index().rename(columns={'purchase_date': 'Last_Purchase_Date'})
last_purchase['Last_Purchase_Date'] = pd.to_datetime(last_purchase['Last_Purchase_Date'])
last_purchase['Purchase_Recency'] = (pd.to_datetime('now').tz_localize(None) - last_purchase['Last_Purchase_Date']).dt.days
last_purchase = last_purchase[['id', 'Purchase_Recency']]
last_purchase['Purchase_Recency'] = last_purchase['Purchase_Recency'].fillna(-1)
last_purchase.head()
```

	id	Purchase_Recency
0	1	58.0
1	3	235.0
2	5	989.0
3	7	1330.0
4	10	316.0

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

Order Frequency

```
order_frequency = df.groupby('id').size().reset_index(name='total_orders')
order_frequency.nlargest(10, 'total_orders')
```

	id	total_orders
9171	17330	12
10763	20336	12
12076	22853	12
30677	58121	12
9035	17084	11
11591	21933	11
13465	25563	11
42171	79874	11
13197	25071	10
27073	51398	10

Time of Day Analysis (morning, afternoon, evening, night)

```
time_of_day_preference = df.groupby(['id', 'time_of_day'])['order_id'].count().reset_index().sort_values(by='order_id', ascending=False)
time_of_day_preference = time_of_day_preference.loc[time_of_day_preference.groupby('id')['order_id'].idxmax()][['id', 'time_of_day']]
time_of_day_preference.head()
```

	id	Preferred_Time_of_Day
0	1	Night
1	3	Evening
2	5	Night
3	7	Afternoon
4	10	Night

Next steps: [Generate code with time_of_day_preference](#)

[View recommended plots](#)

[New interactive sheet](#)

Final Customer Features

```
customer_features = pd.merge(average_order_value, category_preferences, on='id', how='left')
customer_features = pd.merge(customer_features, customer_traffic_source, on='id', how='left')
customer_features = pd.merge(customer_features, last_purchase, on='id', how='left')
customer_features = pd.merge(customer_features, order_frequency, on='id', how='left')
customer_features = pd.merge(customer_features, time_of_day_preference, on='id', how='left')
customer_features = pd.merge(customer_features, df[['id', 'state']], on='id', how='left')
```

```
customer_features.head()
```

	id	avg_order_value	Preferred_Category	most_frequent_traffic_source	Purchase_Recency	total_orders	Preferred_Time_of_Day
0	1	78.000000	Maternity	Facebook	58.0	1	Night
1	3	51.189999	Tops & Tees	Search	235.0	1	Evening
2	5	19.990000	Maternity	Search	989.0	1	Night
3	7	150.500000	Jeans	Search	1330.0	3	Afternoon
4	7	150.500000	Jeans	Search	1330.0	3	Afternoon

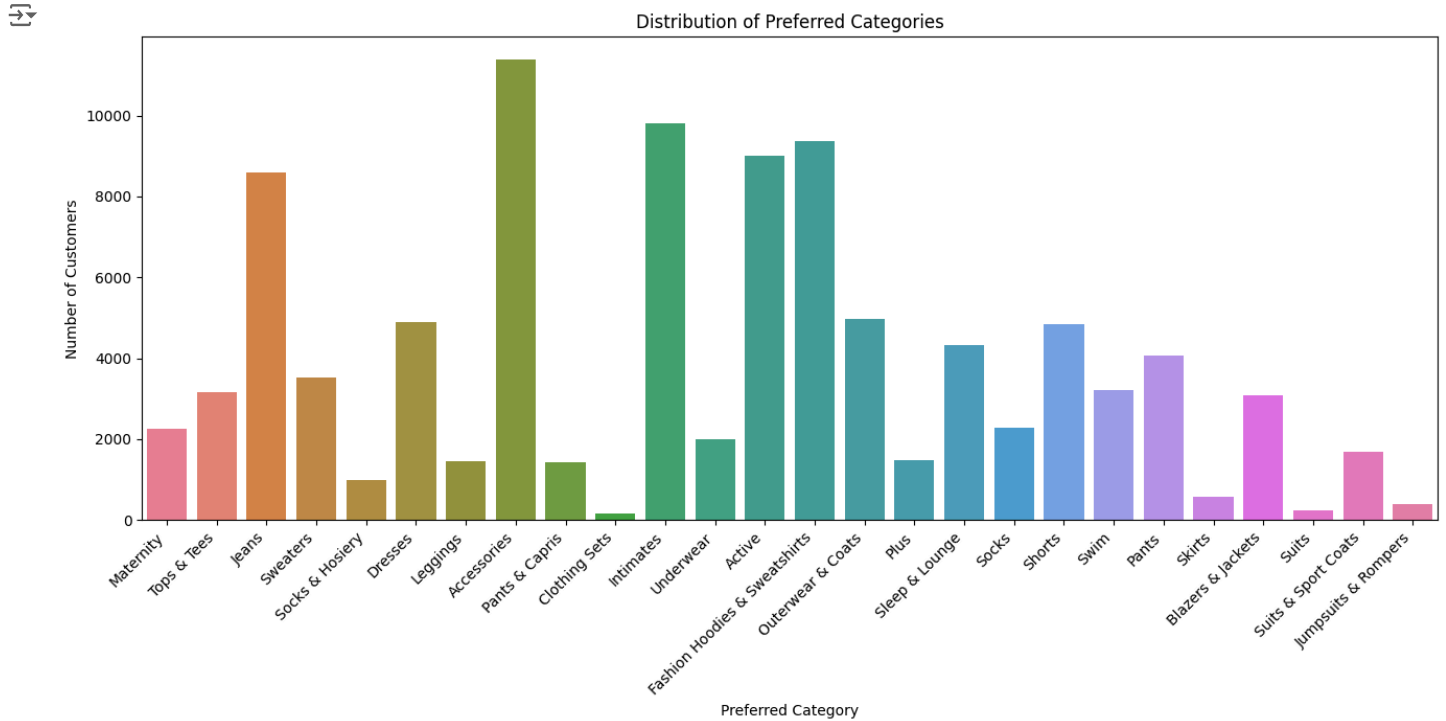
Next steps: [Generate code with customer_features](#)

[View recommended plots](#)

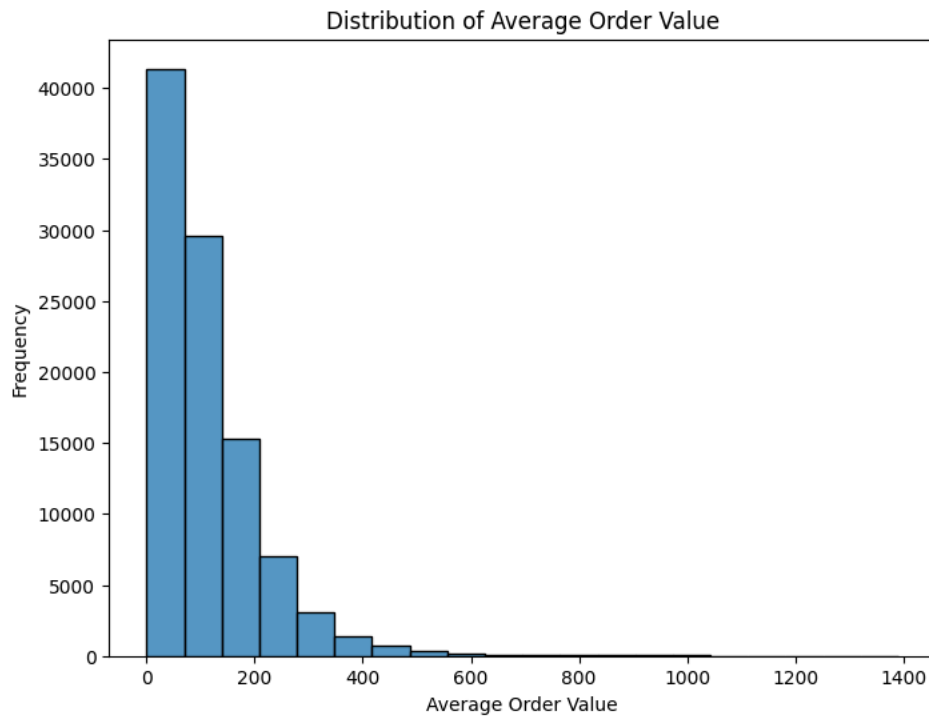
[New interactive sheet](#)

5. Basic EDA

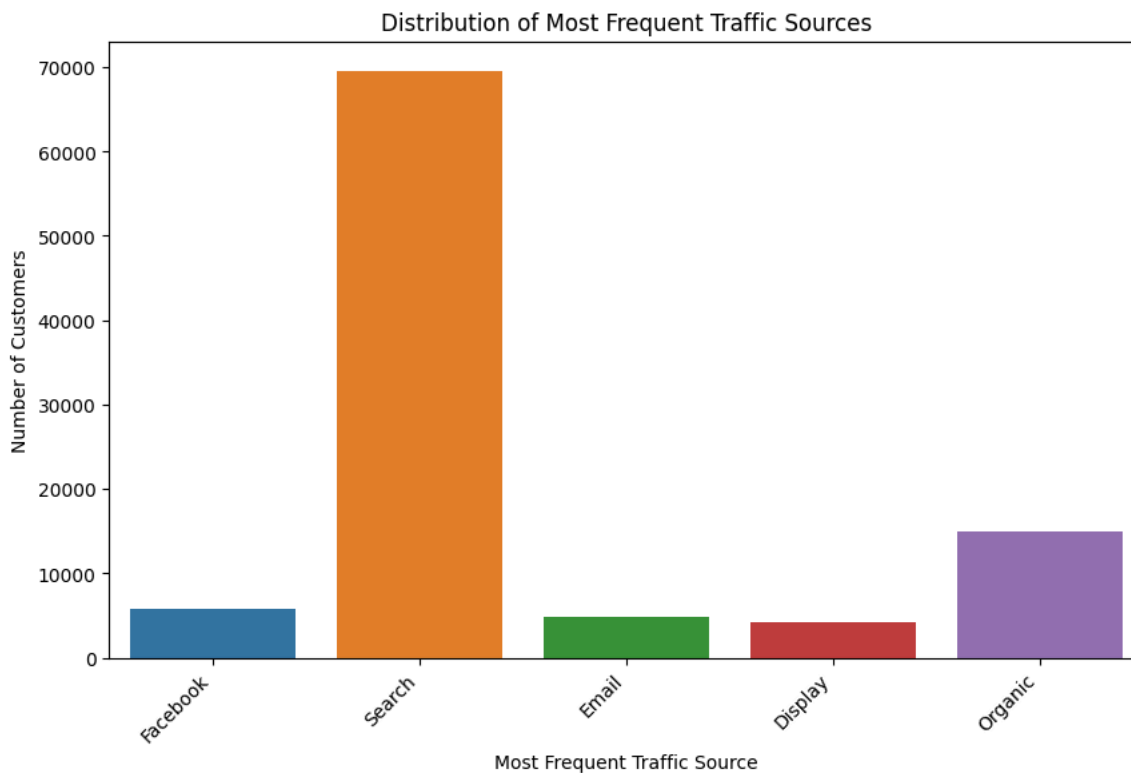
```
plt.figure(figsize=(16, 6))
sns.countplot(x='Preferred_Category', data=customer_features, hue='Preferred_Category')
plt.title('Distribution of Preferred Categories')
plt.xlabel('Preferred Category')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
# Distribution of Average Order Value
plt.figure(figsize=(8, 6))
sns.histplot(customer_features['avg_order_value'], bins=20)
plt.title('Distribution of Average Order Value')
plt.xlabel('Average Order Value')
plt.ylabel('Frequency')
plt.show()
```

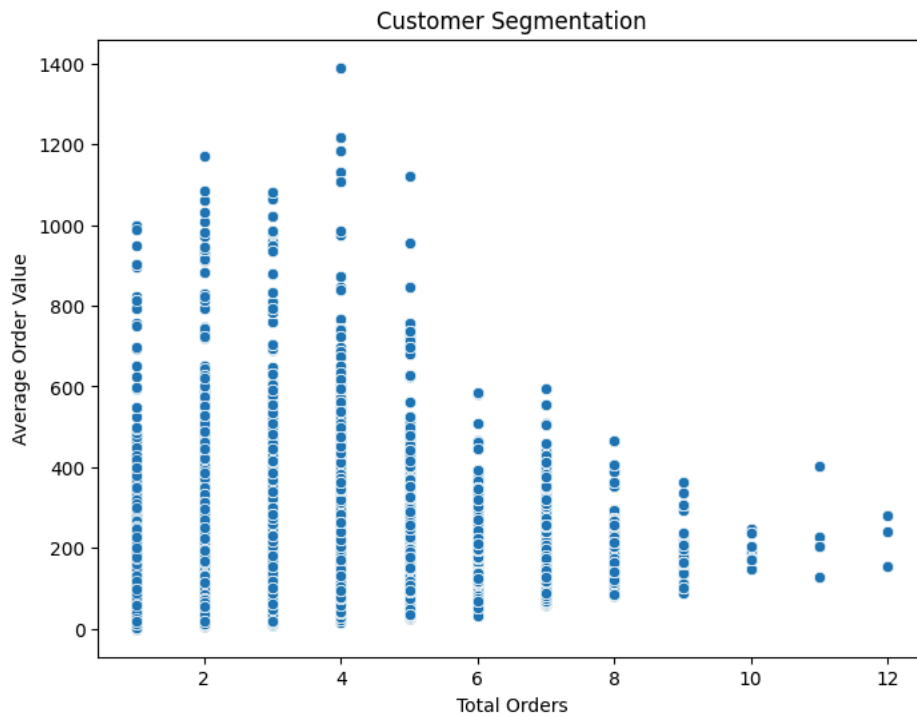


```
# Traffic Source Analysis
plt.figure(figsize=(10, 6))
sns.countplot(x='most_frequent_traffic_source', data=customer_features, hue='most_frequent_traffic_source')
plt.title('Distribution of Most Frequent Traffic Sources')
plt.xlabel('Most Frequent Traffic Source')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
# Customer Segmentation (Example: Based on Total Orders and Average Order Value)
plt.figure(figsize=(8, 6))
sns.scatterplot(x='total_orders', y='avg_order_value', data=customer_features)
plt.title('Customer Segmentation')
plt.xlabel('Total Orders')
```

```
plt.ylabel('Average Order Value')
plt.show()
```



```
customer_features.head()
```

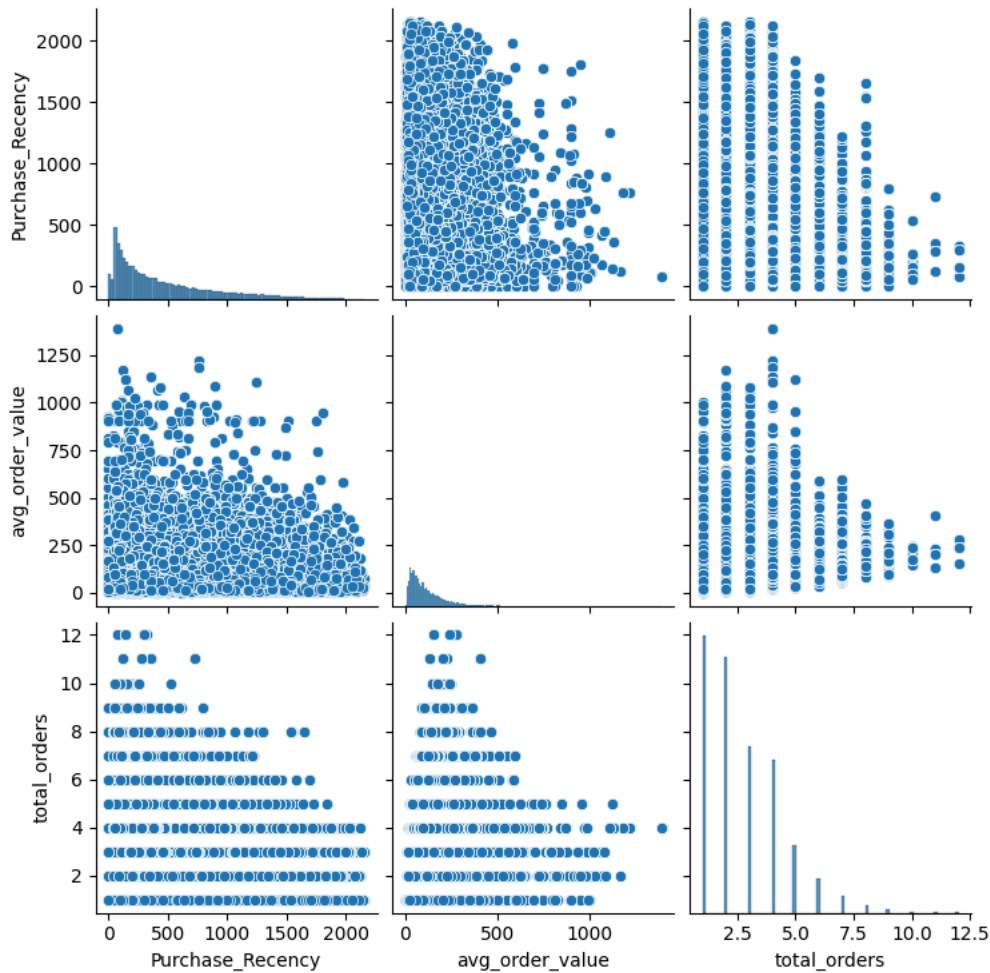


	id	avg_order_value	Preferred_Category	most_frequent_traffic_source	Purchase_Recency	total_orders	Preferred_Time_of_Da
0	1	78.000000	Maternity	Facebook	58.0	1	Nig
1	3	51.189999	Tops & Tees	Search	235.0	1	Evenin
2	5	19.990000	Maternity	Search	989.0	1	Nig
3	7	150.500000	Jeans	Search	1330.0	3	Afterno
4	7	150.500000	Jeans	Search	1330.0	3	Afterno

Next steps:

[Generate code with customer_features](#)
☒ [View recommended plots](#)
[New interactive sheet](#)

```
sns.pairplot(customer_features[['Purchase_Recency', 'avg_order_value', 'total_orders']])
plt.show()
```

6. Customer Segmentation

I applied clustering to segment customers based on their purchasing behavior and demographic information. This segmentation allows me to identify distinct customer groups, each with unique characteristics and needs.

Clustering Method:

- **K-means Clustering:** I used the K-means algorithm to group customers based on similarity in their behavior and demographics.

Steps:

1. **Feature Scaling:** Scale features to ensure that they contribute equally to the clustering process.
2. **Elbow Method:** Use the Elbow method to determine the optimal number of clusters by plotting inertia against different k values.
3. **Apply K-means:** Perform K-means clustering with the optimal k value.

```
customer_features.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99299 entries, 0 to 99298
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     99299 non-null  int64
1   avg_order_value                       99299 non-null  float64
2   Preferred_Category                   99299 non-null  object
3   most_frequent_traffic_source         99299 non-null  object
4   Purchase_Recency                     99299 non-null  float64
5   total_orders                         99299 non-null  int64
6   Preferred_Time_of_Day                 99299 non-null  object
7   state                                99299 non-null  object
dtypes: float64(2), int64(2), object(4)
memory usage: 6.1+ MB
```

Feature Scaling and Encoding

```
categorical_features = ['Preferred_Category', 'most_frequent_traffic_source', 'Preferred_Time_of_Day', 'state']
```

```
encoder = OneHotEncoder(drop='first', sparse_output=False)
X_cat_encoded = encoder.fit_transform(customer_features[categorical_features])
```

```
encoded_feature_names = encoder.get_feature_names_out(categorical_features)
X_cat_df = pd.DataFrame(X_cat_encoded, columns=encoded_feature_names)
```

```
X_cat_df.reset_index(drop=True, inplace=True)
```

```
numerical_features = ['total_orders', 'avg_order_value', 'Purchase_Recency']
```

```
X_num = customer_features[numerical_features].fillna(0)
scaler = StandardScaler()
X_num_scaled = scaler.fit_transform(X_num)
```

```
X_num_df = pd.DataFrame(X_num_scaled, columns=numerical_features)
X_num_df.reset_index(drop=True, inplace=True)
```

```
X_combined = pd.concat([X_num_df, X_cat_df], axis=1)
X_combined.head()
```



	total_orders	avg_order_value	Purchase_Recency	Preferred_Category_Active	Preferred_Category_Blazers & Jackets	Preferred_Category
0	-1.036891	-0.349674	-0.964353	0.0	0.0	
1	-1.036891	-0.617802	-0.555804	0.0	0.0	
2	-1.036891	-0.929834	1.184571	0.0	0.0	
3	0.174116	0.375402	1.971664	0.0	0.0	
4	0.174116	0.375402	1.971664	0.0	0.0	

5 rows x 263 columns

Dimensionality Reduction with PCA

```
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_combined)
```

```
explained_variance_ratio = pca.explained_variance_ratio_
print(f'Explained variance by PCA components: {explained_variance_ratio}')
```



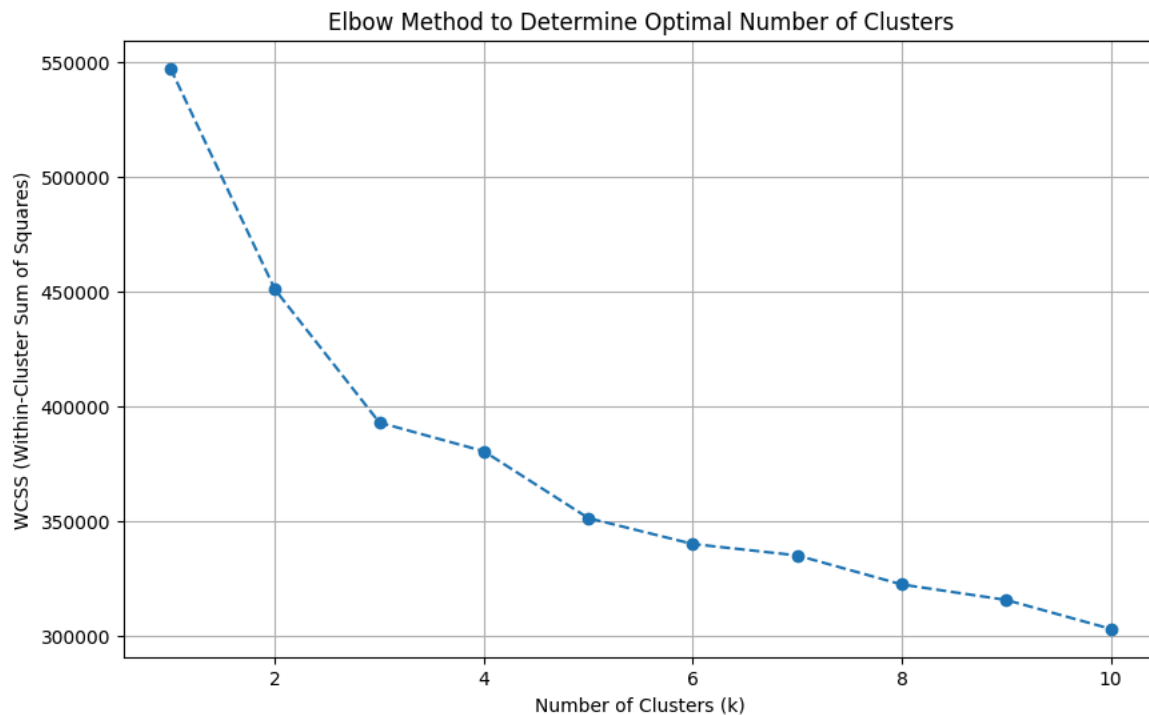
```
Explained variance by PCA components: [0.24813661 0.17293719 0.09801558 0.05295194 0.04963377 0.0255209
0.0166728 0.01597805 0.01539898 0.0149674 0.0121518 0.01033482
0.00914568 0.00865938 0.00849823 0.00828063 0.00777219 0.00728766
0.00682791 0.00654721 0.00629613 0.00589111 0.00553626 0.00537294
0.00433987 0.00427545 0.0041371 0.00394856 0.00379669 0.00375456
0.00364053 0.00359081 0.00343346 0.00334962 0.00323549 0.00308516
0.00300781 0.00287621 0.00271926 0.00264351 0.00255415 0.00255091
0.00250011 0.00243597 0.00240865 0.00233167 0.00228504 0.00225981
0.00222596 0.00218359 0.00214853 0.0020835 0.0020272 0.00196225
0.00188454 0.0018086 0.00178627 0.00172748 0.00164868 0.00159648
0.00155559 0.00154789 0.00151937 0.00148332 0.00143445 0.00136153
0.00134171 0.00130743 0.00129587 0.00127995 0.00126365 0.00122704
0.00118217 0.00116408 0.00114154 0.00113794 0.00112181 0.00111591
0.00110319 0.0010876 0.00106347 0.00103671 0.00101812 0.00100344
0.00099855 0.00099311 0.00098913 0.0009725 ]
```

Elbow Method to Determine Optimal Clusters

```
wcss = []
k_values = range(1, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.title('Elbow Method to Determine Optimal Number of Clusters')
plt.grid(True)
plt.show()
```



Implementing KMeans Clustering

```
optimal_k = 4

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
customer_features['Cluster'] = kmeans.fit_predict(X_pca)

customer_features.head()
```



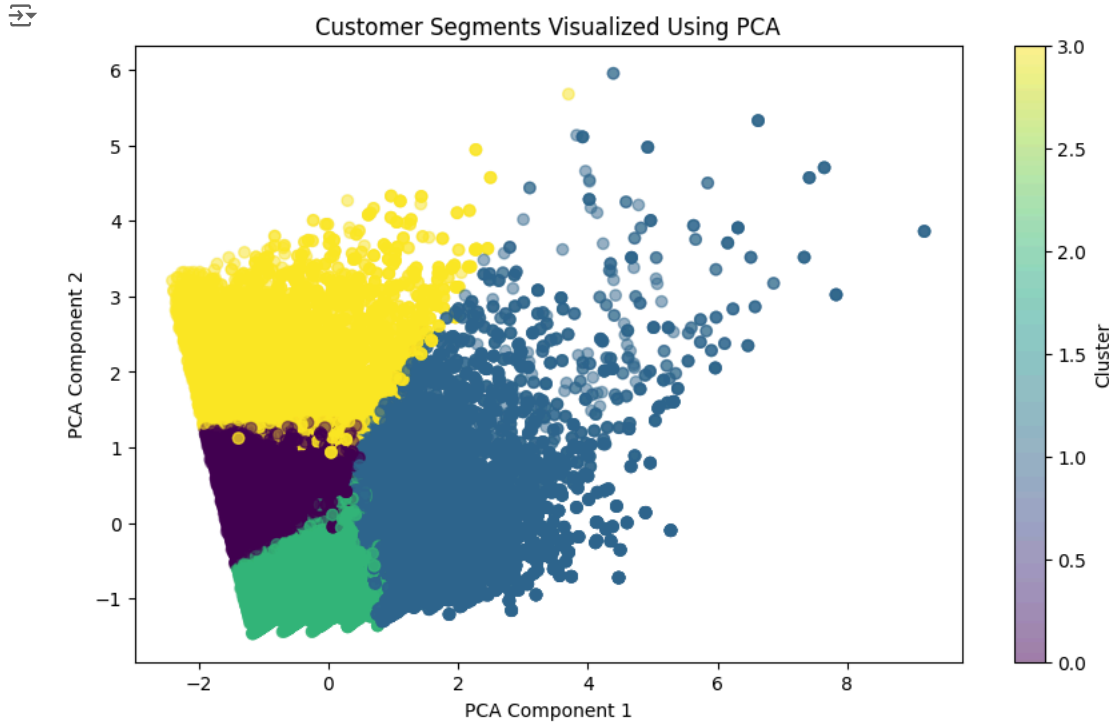
	id	avg_order_value	Preferred_Category	most_frequent_traffic_source	Purchase_Recency	total_orders	Preferred_Time_of_D
0	1	78.000000	Maternity	Facebook	58.0	1	Nig
1	3	51.189999	Tops & Tees	Search	235.0	1	Evenin
2	5	19.990000	Maternity	Search	989.0	1	Nig
3	7	150.500000	Jeans	Search	1330.0	3	Afterno
4	7	150.500000	Jeans	Search	1330.0	3	Afterno

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

```
pca_viz = PCA(n_components=2)
X_viz_pca = pca_viz.fit_transform(X_combined)


plt.figure(figsize=(10, 6))
plt.scatter(X_viz_pca[:, 0], X_viz_pca[:, 1], c=customer_features['Cluster'], cmap='viridis', alpha=0.5)
plt.xlabel('PCA Component 1')
```

```
plt.ylabel('PCA Component 2')
plt.title('Customer Segments Visualized Using PCA')
plt.colorbar(label='Cluster')
plt.show()
```



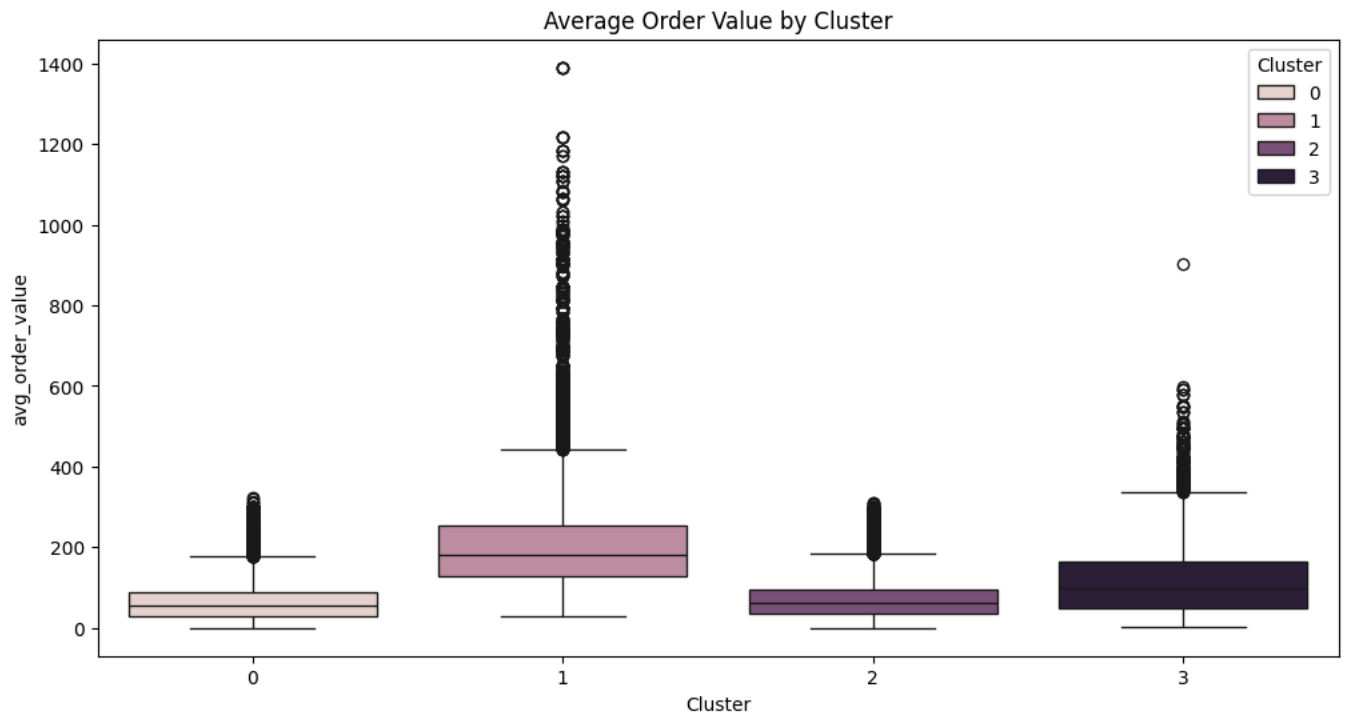
▼ Analyzing Clusters

```
cluster_summary = customer_features.groupby('Cluster')[['avg_order_value', 'total_orders', 'Purchase_Recency']].mean()
print(cluster_summary)
```

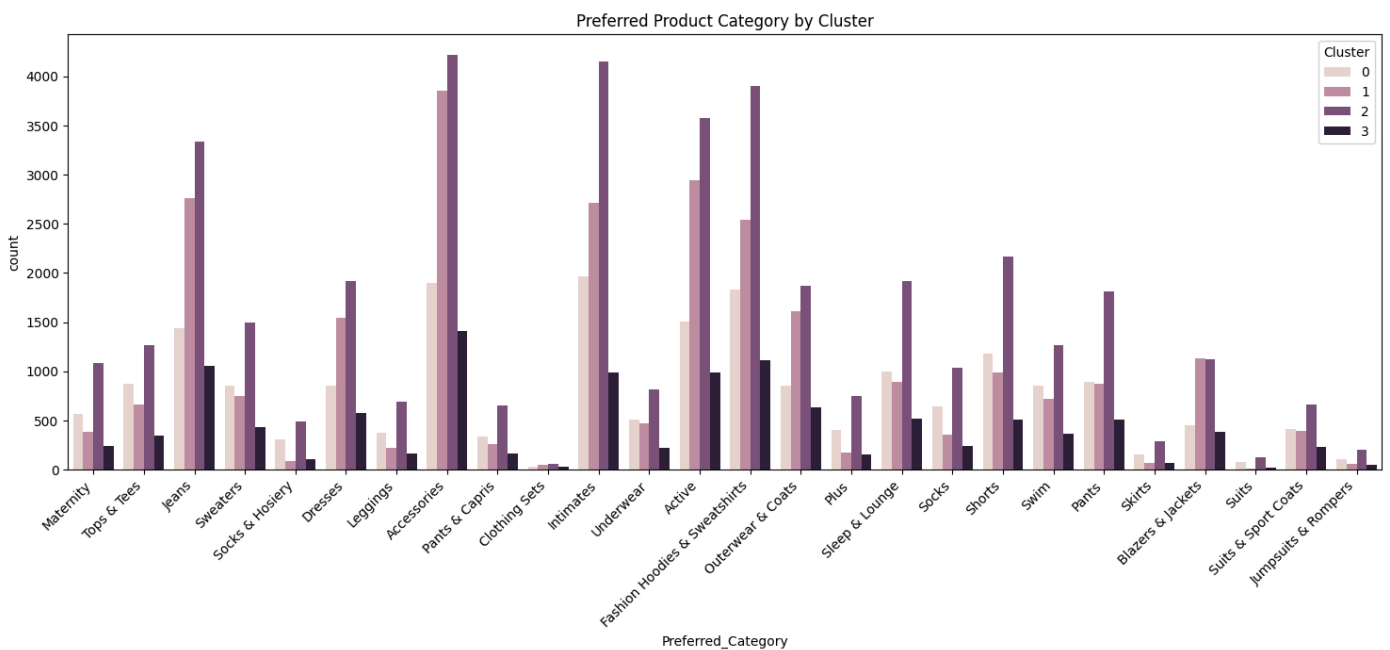


Cluster	avg_order_value	total_orders	Purchase_Recency
0	66.263959	1.687856	724.740288
1	210.022402	4.694536	329.577931
2	71.753391	2.059152	192.385722
3	118.366129	2.281020	1376.017602

```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Cluster', y='avg_order_value', data=customer_features, hue="Cluster")
plt.title('Average Order Value by Cluster')
plt.show()
```




```
plt.figure(figsize=(18, 6))
sns.countplot(x='Preferred_Category', hue='Cluster', data=customer_features)
plt.xticks(rotation=45, ha='right')
plt.title('Preferred Product Category by Cluster')
plt.show()
```



```
cluster_summary = customer_features.groupby('Cluster').agg({
    'avg_order_value': ['mean', 'median', 'std'],
    'total_orders': ['mean', 'median', 'std'],
    'Purchase_Reency': ['mean', 'median', 'std'],
```

```
'Preferred_Category': lambda x: x.mode()[0], # Most frequent category
'most_frequent_traffic_source': lambda x: x.mode()[0], #Most frequent traffic source
'Preferred_Time_of_Day': lambda x: x.mode()[0] #Most frequent time of day
})

cluster_summary
```



	avg_order_value			total_orders			Purchase_Recency			Preferred_Category	most_freq
	mean	median	std	mean	median	std	mean	median	std	<lambda>	<lambda>
Cluster											
0	66.263959	53.995000	47.369236	1.687856	1.0	0.813854	724.740288	697.0	190.598775		Intimates
1	210.022402	182.382001	123.497794	4.694536	4.0	1.494417	329.577931	248.0	272.228588		Accessories
2	71.753391	60.663333	46.988291	2.059152	2.0	0.946850	192.385722	167.0	126.306099		Accessories
3	118.366129	97.990002	87.560402	2.281020	2.0	1.189626	1376.017602	1340.0	266.950628		Accessories

Next steps:

[Generate code with cluster_summary](#)

 [View recommended plots](#)

[New interactive sheet](#)


```
preferred_categories_per_cluster = customer_features.groupby(['Cluster', 'Preferred_Category']).size().unstack(fill_value=0)
preferred_categories_per_cluster
```



Preferred_Category	Accessories	Active	Blazers & Jackets	Clothing Sets	Dresses	Fashion Hoodies & Sweatshirts	Intimates	Jeans	Jumpsuits & Rompers	Leggings	...	S
Cluster												
0		1894	1507	451	32	859	1832	1969	1443	103	377	...
1		3856	2941	1134	52	1545	2536	2712	2760	57	223	...
2		4216	3577	1124	56	1920	3899	4154	3333	199	688	...
3		1413	989	385	32	577	1115	984	1052	47	168	...

4 rows x 26 columns

```
traffic_source_per_cluster = customer_features.groupby(['Cluster', 'most_frequent_traffic_source']).size().unstack(fill_value=0)
traffic_source_per_cluster
```



most_frequent_traffic_source	Display	Email	Facebook	Organic	Search
Cluster					
0	829	1033	1305	3008	14213
1	1110	1238	1477	4002	18690
2	1698	1949	2378	6119	28717
3	485	610	694	1761	7983

Next steps:

[Generate code with traffic_source_per_cluster](#)

 [View recommended plots](#)

[New interactive sheet](#)

Cluster Summary Insights

1. Cluster 0: "Low Value, Dormant Shoppers" - Low spenders who have not purchased in a long time.
2. Cluster 1: "High Value, Frequent Shoppers" - High spenders with frequent purchases and reasonable engagement.
3. Cluster 2: "Moderate Value, Engaged Shoppers" - Moderate spenders who are relatively recent purchasers.
4. Cluster 3: "Moderate Value, Highly Dormant Shoppers" - Moderate spenders who have not been active for a very long time.

```
def label_clusters(row):
    if row['Cluster'] == 0:
        return 'Low Value, Dormant Shoppers'
    elif row['Cluster'] == 1:
        return 'High Value, Frequent Shoppers'
    elif row['Cluster'] == 2:
        return 'Moderate Value, Engaged Shoppers'
    elif row['Cluster'] == 3:
        return 'Moderate Value, Highly Dormant Shoppers'

customer_features['Cluster_Label'] = customer_features.apply(label_clusters, axis=1)

customer_features = customer_features.rename(columns={'Cluster_Label': 'Customer_Profile'})

df = pd.merge(df, customer_features[['id', 'Customer_Profile']], on='id', how='left')

df.head()
```

	id	first_name	age	gender	state	traffic_source	order_id	status	created_at	product_id	sale_price	num_of_item	ca
0	1260	Pamela	12	F	Hunan	Email	1588	Complete	2024-01-24 17:11:00	10467	34.000000	1	li
1	1260	Pamela	12	F	Hunan	Email	1588	Complete	2024-01-24 17:11:00	10467	34.000000	1	li
2	1260	Pamela	12	F	Hunan	Email	1589	Complete	2024-01-30 17:11:00	3385	49.990002	1	
3	1260	Pamela	12	F	Hunan	Email	1589	Complete	2024-01-30 17:11:00	3385	49.990002	1	
4	1350	Virginia	12	F	Jiangsu	Search	1695	Complete	2022-08-04 11:13:00	1016	67.959999	1	S

```
# customer_features.to_csv('customer_features.csv', index=False)
```

✓ A/B Testing Setup

To evaluate the effectiveness of different strategies, I simulated an A/B test by dividing each cluster into Control and Test groups. This setup allows me to assess how changes (e.g., increasing order frequency or AOV) impact customer behavior within each segment.

Hypothetical Campaign

- Control Group: This group will receive no discount.
- Treatment Group: This group will receive a 10% discount.
- Segments Chosen:
 - Cluster 1: High Value, Frequent Shoppers.
 - Cluster 0: Low Value, Dormant Shoppers.

Objective

The goal is to predict the impact of a 10% discount on different customer groups:

- Cluster 1 - To further engage existing high-value customers.
- Cluster 0 - To reactivate dormant customers and encourage them to make a purchase.

I'll compare how the treatment group (with the discount) and the control group (without any discount) could hypothetically differ in terms of average order value, total orders, and purchase recency.

✓ Splitting Segments

```
high_value_frequent_shoppers = customer_features[customer_features['Cluster'] == 1]
low_value_dormant_shoppers = customer_features[customer_features['Cluster'] == 0]
```

✓ Control & Treatment Groups

```
high_value_control = high_value_frequent_shoppers.sample(frac=0.5, random_state=42)
high_value_treatment = high_value_frequent_shoppers.drop(high_value_control.index)
```

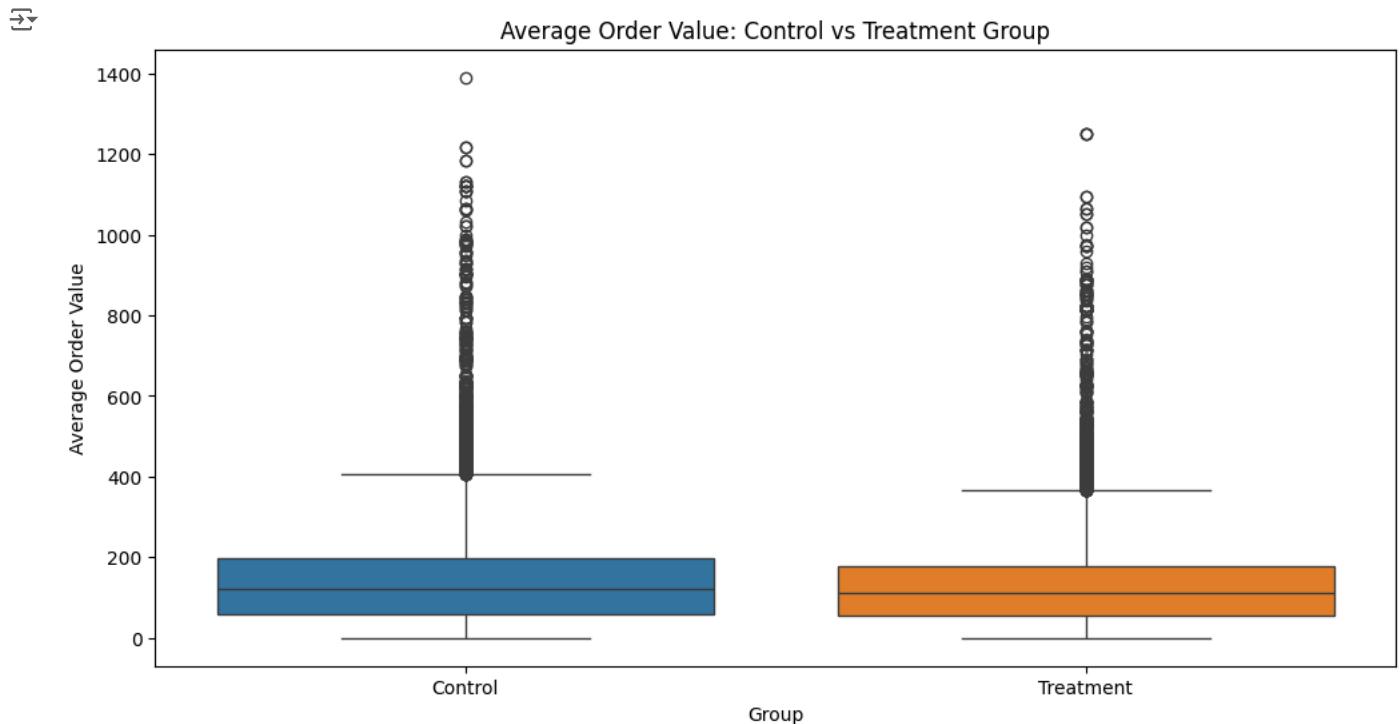
```
low_value_control = low_value_dormant_shoppers.sample(frac=0.5, random_state=42)
low_value_treatment = low_value_dormant_shoppers.drop(low_value_control.index)
```

✓ Discount to Treatment Group

```
high_value_treatment['avg_order_value'] = high_value_treatment['avg_order_value'] * 0.90
low_value_treatment['avg_order_value'] = low_value_treatment['avg_order_value'] * 0.90
```

```
combined_df = pd.concat([high_value_control, high_value_treatment, low_value_control, low_value_treatment])
combined_df['Group'] = np.where(combined_df.index.isin(high_value_treatment.index) | combined_df.index.isin(low_value_treatment.index),
```

```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Group', y='avg_order_value', data=combined_df, hue='Group')
plt.xlabel('Group')
plt.ylabel('Average Order Value')
plt.title('Average Order Value: Control vs Treatment Group')
plt.show()
```



✓ Scipy Analysis

```
control_avg_order = combined_df[combined_df['Group'] == 'Control']['avg_order_value']
treatment_avg_order = combined_df[combined_df['Group'] == 'Treatment']['avg_order_value']
```

```
t_stat, p_value = ttest_ind(control_avg_order, treatment_avg_order)
print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

```
if p_value < 0.05:
    print("The difference between the Control and Treatment groups is statistically significant.")
```