20f-1114

In [183…
```
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Module 1

In [184…
```
# Data Acquisition:

df = pd.read_json('electronics.json')
df
```

Out[184]:

	Customer_ID	Age	Gender	Income_Level	Address	Transaction_ID	Purchase_Date
0	b81ee6c9-2ae4-48a7-b283-220eaa244f43	40	Female	Medium	43548 Murray Islands Suite 974\nAmyberg, CT 13457	c6a6c712-e36b-406a-bfde-f53bdcf4744f	2022-04-26
1		25	Male	High		0b587838-1e4f-4231-b488-42bcd47c052a	2021-08-10
2	fdf79bcd-5908-4c90-8501-570ffb5b7648	57	Other	Low	79683 Kevin Hill Apt. 555\nJohnshire, AR 39961	462925b1-a5bf-4996-bda2-59749de64eea	2021-12-09
3	878dccba-893a-48f9-8d34-6ed394fa3c9c	38	Female	Medium	02998 Hall Meadows Suite 809\nNorth Robertvill...	3cfafa02-6b34-4d77-9e05-d223dfab64e8	2022-12-03
4	0af0bd81-73cc-494e-aa5e-75c6d0b6d743	68	Other	Medium	21411 Timothy Ford Apt. 320\nDavisborough, AR ...	0d8dc27a-0c8f-4a82-b57e-8bf54cee9759	2020-06-08
...	...	...	...	...	...	...	...
995		70	Male	Medium	566 Butler Turnpike\nPort Holly, OK 22329	776be313-5308-468e-a0ed-7409a4303364	2023-03-17
996	2116266d-8d1c-48cc-ac28-e4e675cb2a4d	78	Female	Low	45710 Wilson Circles Apt. 411\nWalterton, NC 8...	51f771bf-2562-46c1-a25d-2f46f4bb1525	2023-08-30
997	562cee08-f909-4e1c-a811-5711f967bea5	63	Male	High	243 Emily Creek\nSouth Lindaport, CO 81594	74eba598-ee91-4396-a137-6b869702ef29	Hidden
998	84da2eea-6e9e-46d4-8d94-1e9b0c377d78	43	Male	High	1129 Kirby Ferry Suite 743\nBillyfurt, UT 41587	4d2e213e-bcc0-4a8a-9501-6ca8361381c4	2021-05-13
999	87629baf-a138-4374-be37-8bab776379b8	19	Other	High	896 Troy Branch\nAmytown, NJ 62321	69afa592-2658-48ac-9b37-33a3a473d0be	2022-09-13

1000 rows × 18 columns

In [185…

```python
#Data Cleaning

#step 1
#find null data
missing_data = df.isnull().sum()
```

```
data_types = df.dtypes
unique_values = df.nunique()

# Identify missing data, data types, and unique values in the dataset
data_analysis = pd.DataFrame({
    'Data Type': data_types,
    'Missing Values': missing_data,
    'Unique Values': unique_values
})

data_analysis
```

Out[185]:

	Data Type	Missing Values	Unique Values
Customer_ID	object	0	958
Age	object	0	65
Gender	object	0	5
Income_Level	object	0	5
Address	object	0	955
Transaction_ID	object	0	952
Purchase_Date	object	0	700
Product_ID	object	0	953
Product_Category	object	0	5
Brand	object	0	5
Purchase_Amount	object	0	419
Average_Spending_Per_Purchase	object	0	98
Purchase_Frequency_Per_Month	object	0	12
Brand_Affinity_Score	object	0	12
Product_Category_Preferences	object	0	5
Month	object	0	14
Year	object	0	56
Season	object	0	6

In [186…

```
# Replace empty strings and 'Hidden' values with NaN to standardize missing data
df.replace('', np.nan, inplace=True)
df.replace('Hidden', np.nan, inplace=True)

data_analysis = pd.DataFrame({
    'Data Type': data_types,
    'Missing Values': missing_data,
    'Unique Values': unique_values
})

data_analysis
```

Out[186]:

	Data Type	Missing Values	Unique Values
Customer_ID	object	0	958
Age	object	0	65
Gender	object	0	5
Income_Level	object	0	5
Address	object	0	955
Transaction_ID	object	0	952
Purchase_Date	object	0	700
Product_ID	object	0	953
Product_Category	object	0	5
Brand	object	0	5
Purchase_Amount	object	0	419
Average_Spending_Per_Purchase	object	0	98
Purchase_Frequency_Per_Month	object	0	12
Brand_Affinity_Score	object	0	12
Product_Category_Preferences	object	0	5
Month	object	0	14
Year	object	0	56
Season	object	0	6

In [187…

```
numerical_columns_specific = ['Age', 'Purchase_Amount', 'Average_Spending_Per_Purchase
                              'Purchase_Frequency_Per_Month', 'Brand_Affinity_Score',
# Converting specific columns to numeric types for accurate analysis
df[numerical_columns_specific] = df[numerical_columns_specific].apply(pd.to_numeric)


non_numerical_columns_specific = ['Customer_ID', 'Gender', 'Income_Level', 'Address',
                                  'Transaction_ID', 'Product_ID', 'Product_Category',
                                  'Brand', 'Product_Category_Preferences', 'Season']
```

In [188…

```
# Filling missing values in numerical columns with their means
numerical_means = df[numerical_columns_specific].mean()
df[numerical_columns_specific] = df[numerical_columns_specific].fillna(numerical_means

# Filling missing values in non-numerical columns with the most frequent value (mode)
modes = df[non_numerical_columns_specific].mode().iloc[0]
for col in non_numerical_columns_specific:
    df[col] = df[col].fillna(modes[col])

# Converting Purchase_Date to datetime format
df['Purchase_Date'] = pd.to_datetime(df['Purchase_Date'], errors='coerce')



# Removing outliers based on the Interquartile Range (IQR) method
```

```
for col in numerical_columns_specific:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    df = df[~((df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 * IQR)))]









# # Set the size of the plot
# plt.figure(figsize=(12, 8))

# # Plotting boxplots for each numerical column
# for i, col in enumerate(numerical_columns_specific):
#     plt.subplot(len(numerical_columns_specific), 1, i+1)
#     sns.boxplot(x=df[col])
#     plt.title(col)

# plt.tight_layout()
# plt.show()




negative_count = (df['Average_Spending_Per_Purchase'] < 0).sum()

print(f"Number of negative values in the column: {negative_count}")
```

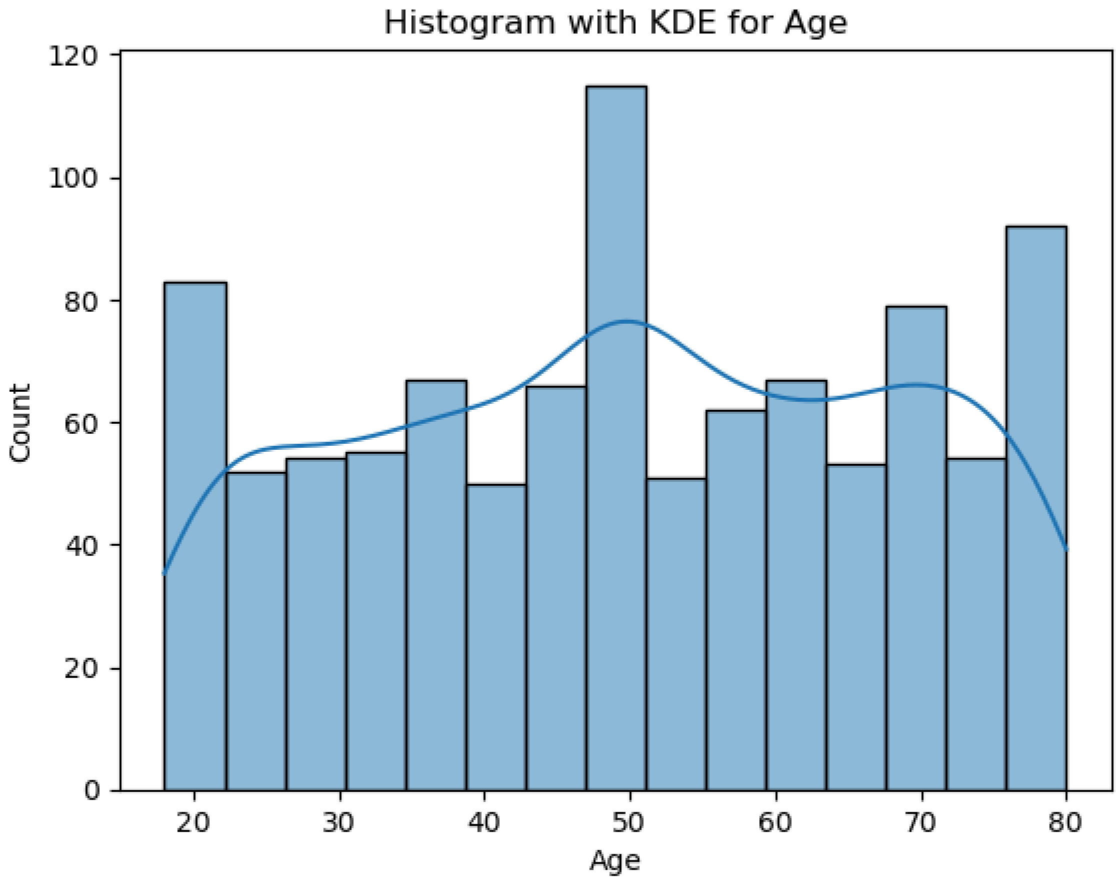
```
Number of negative values in the column: 0
```

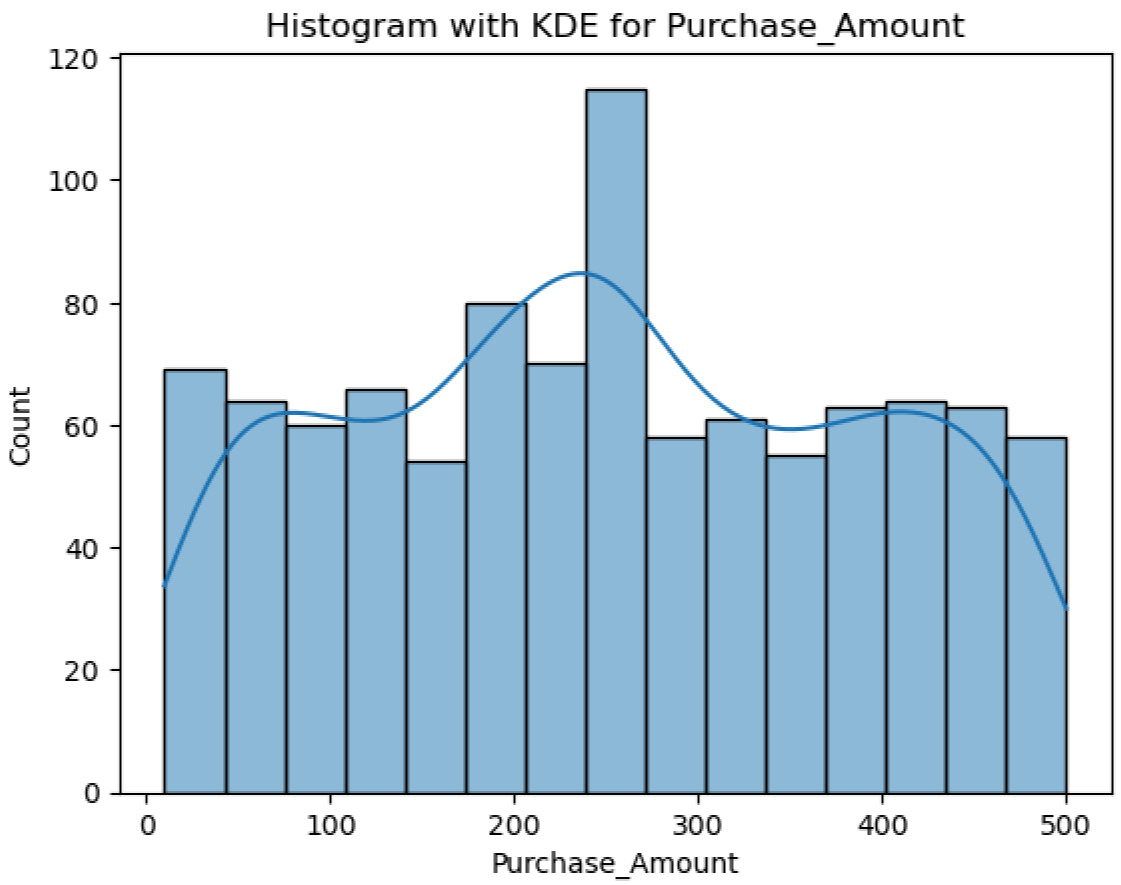
Mdolue 2

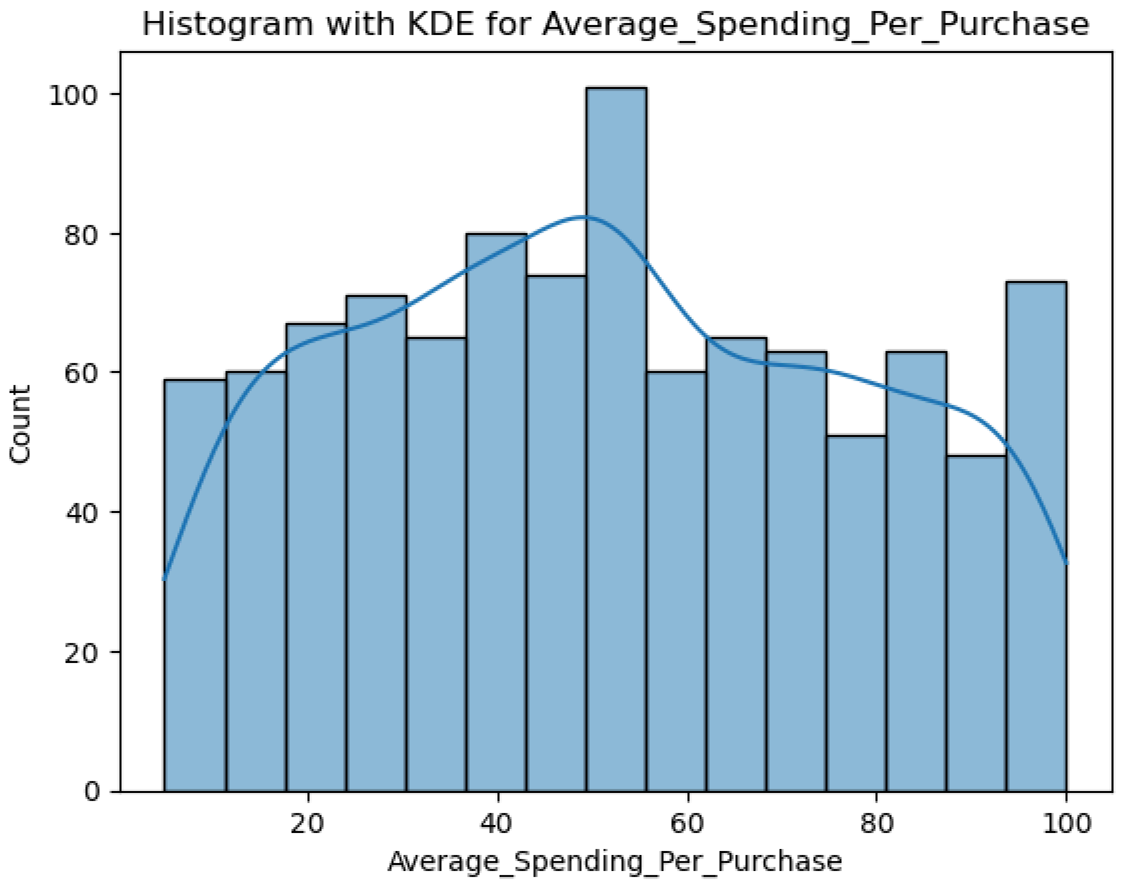
In Module 2, Part 1 of your project, We will conduct Univariate Analysis to understand the distribution and characteristics of individual variables within the electronics section sales data. We will utilize histograms and boxplots to visualize the distribution, skewness, and outliers for key features like customer age and purchase amount. Additionally, We will aggregate data by age groups to analyze how purchase behaviors varied across different demographics. This analysis was crucial for identifying patterns and trends in customer behavior.

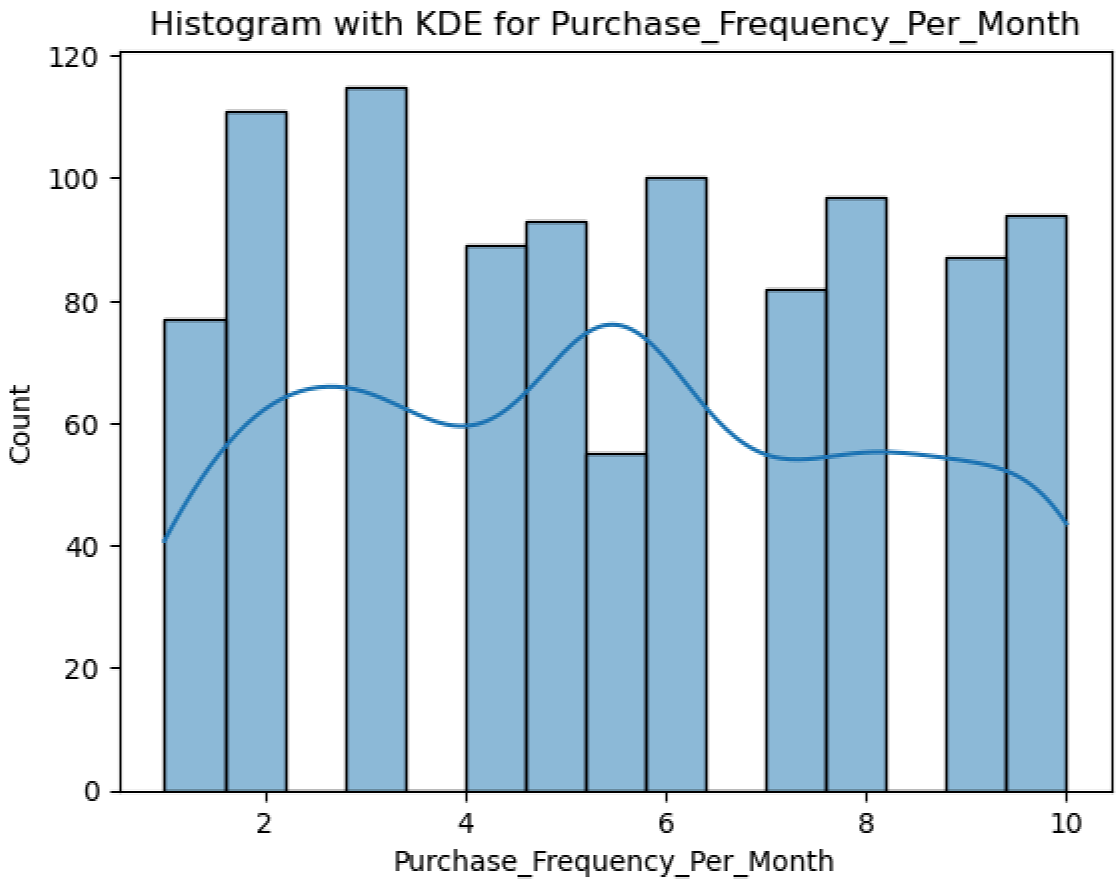
In [189…

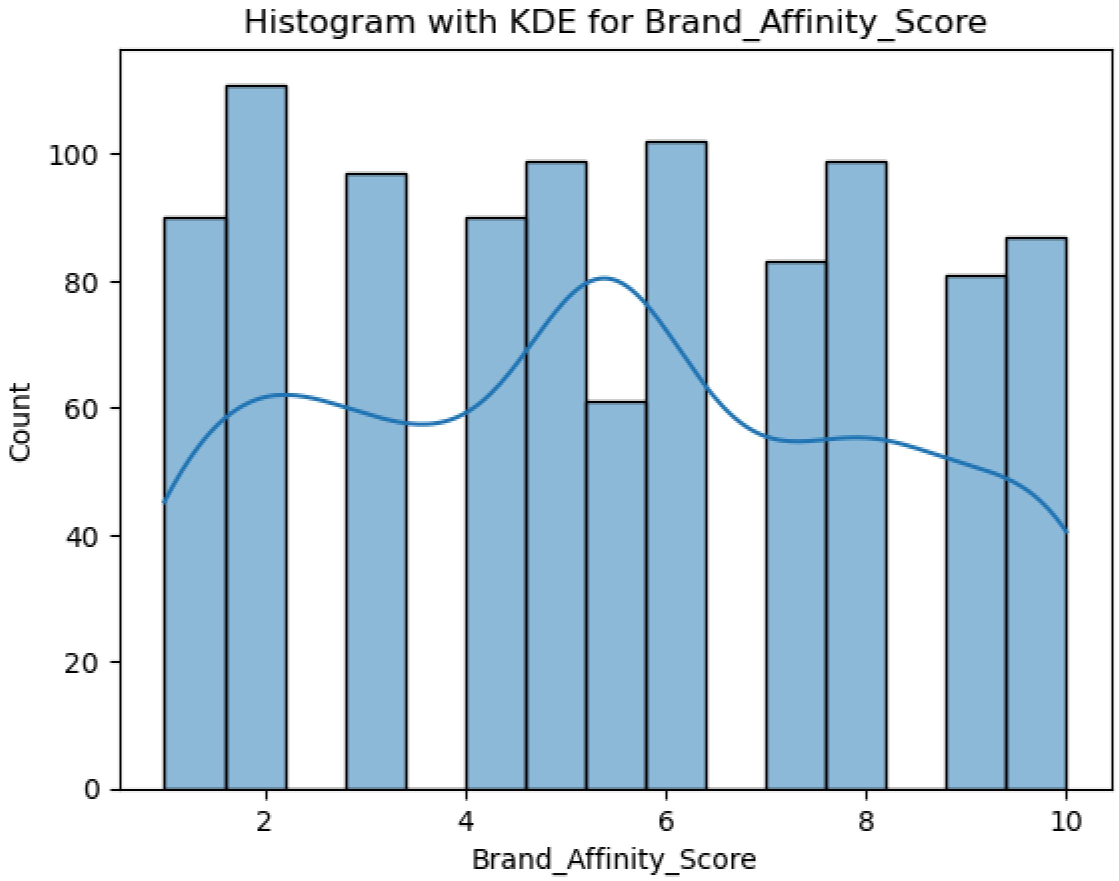
```
#Univariate Analysis



# Histograms for numerical columns
for column in numerical_columns_specific:
    sns.histplot(df[column], bins=15, kde=True)
    plt.title(f'Histogram with KDE for {column}')
    plt.show()

# Boxplots for numerical columns
for col in numerical_columns_specific:
    plt.figure(figsize=(6, 4))
    sns.boxplot(y=df[col])
    sns.stripplot(y=df[col], color= 'orange', alpha= 0.7)
    plt.title(f"Boxplot of {col}")
    plt.show()

# Descriptive statistics
df[numerical_columns_specific].describe()
```

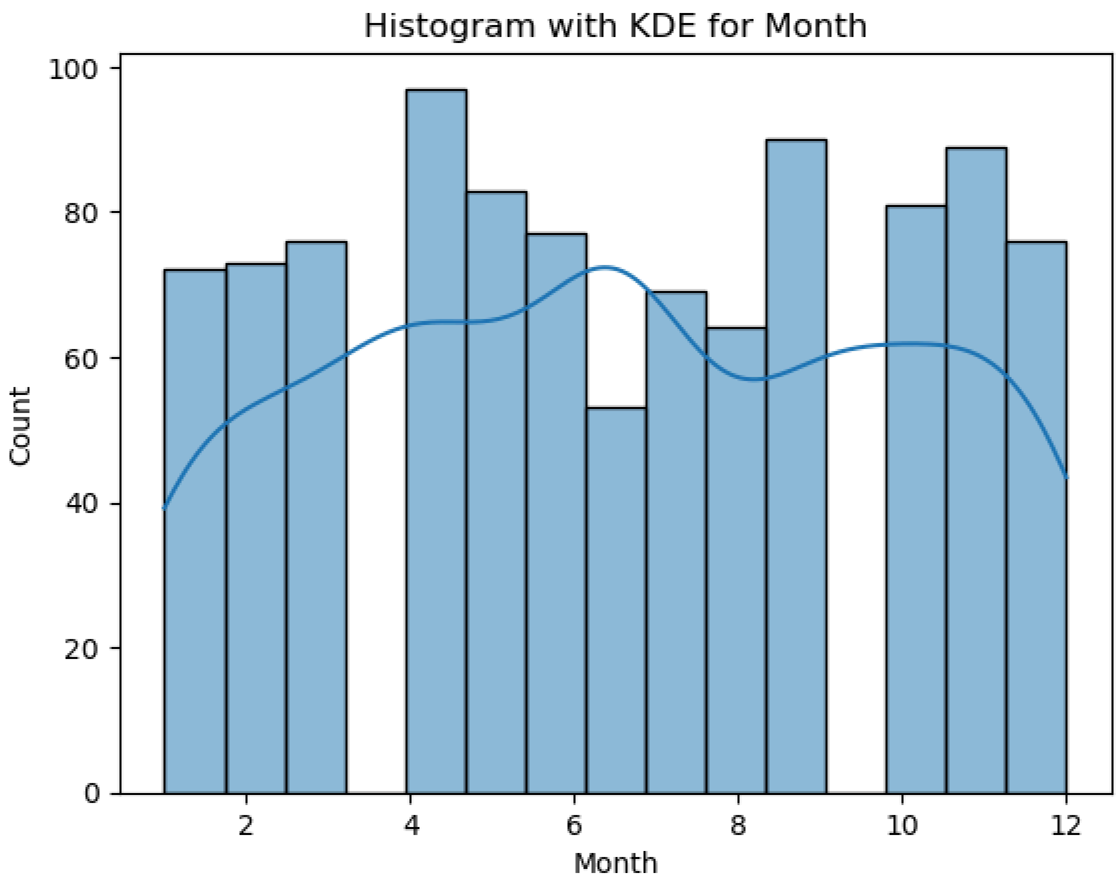


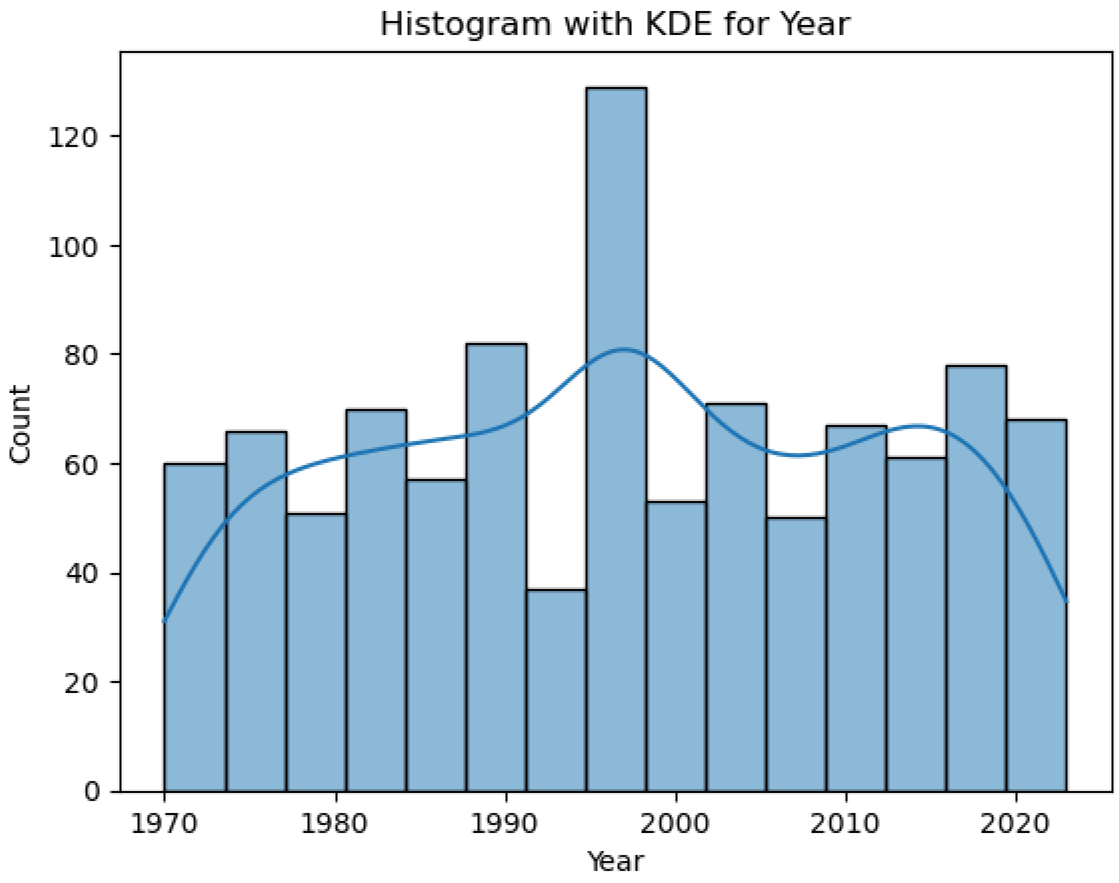

Histogram with KDE for Purchase_Amount
Count
Purchase_Amount
0
20
40
60
80
100
120
0
100
200
300
400
500

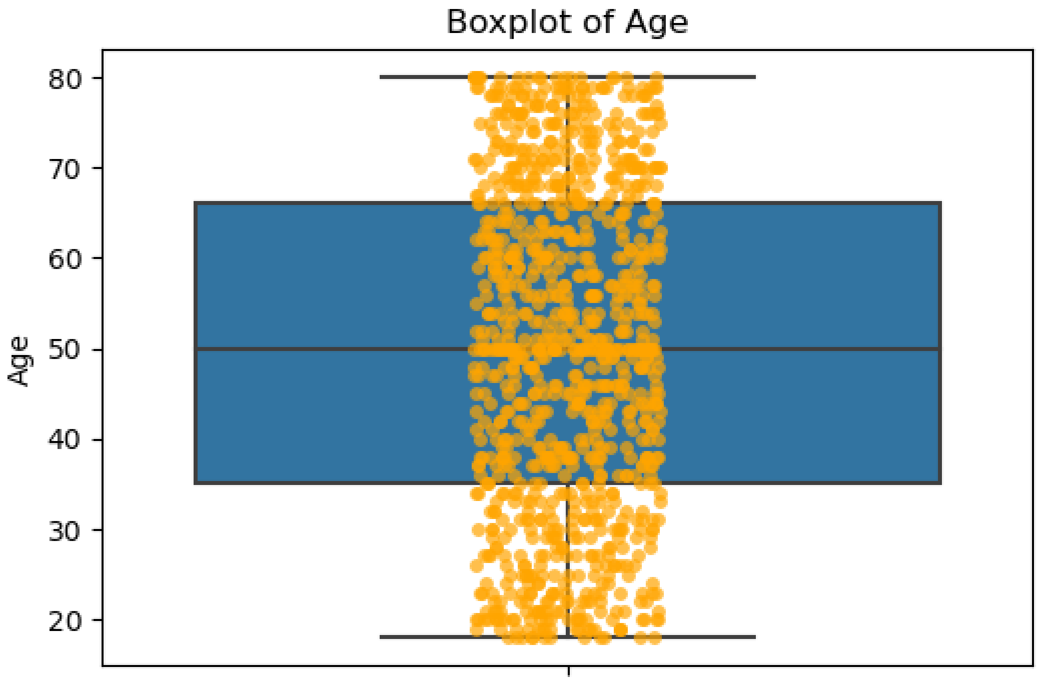

Histogram with KDE for Average_Spending_Per_Purchase
Count
Average_Spending_Per_Purchase
0
20
40
60
80
100
20
40
60
80
100

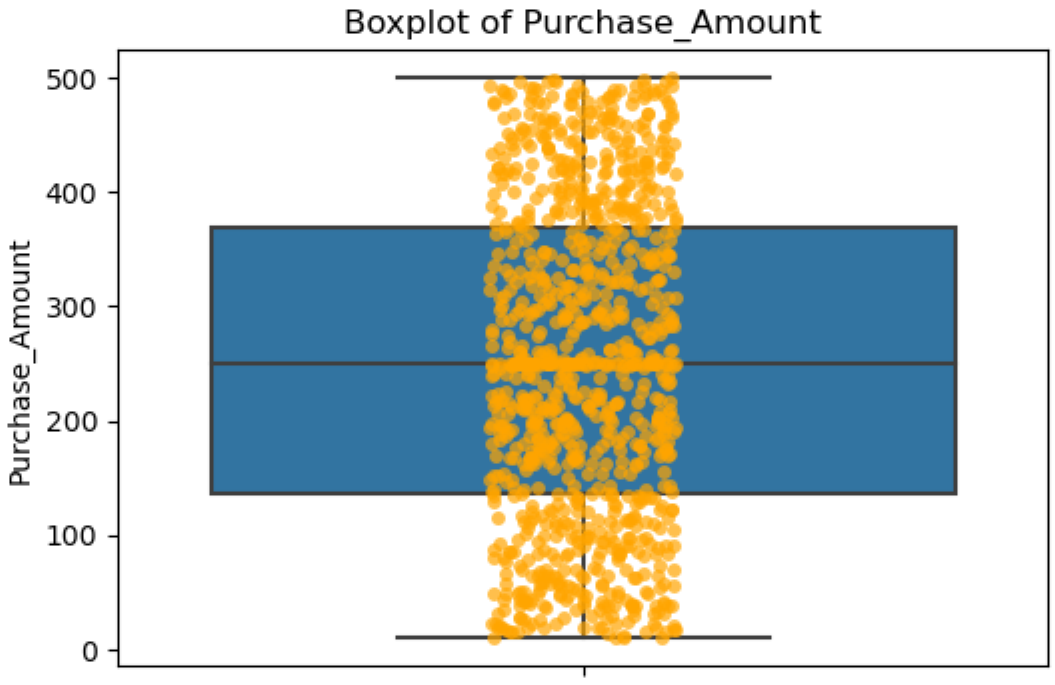

Histogram with KDE for Purchase_Frequency_Per_Month
Count
Purchase_Frequency_Per_Month
0
20
40
60
80
100
120
2
4
6
8
10

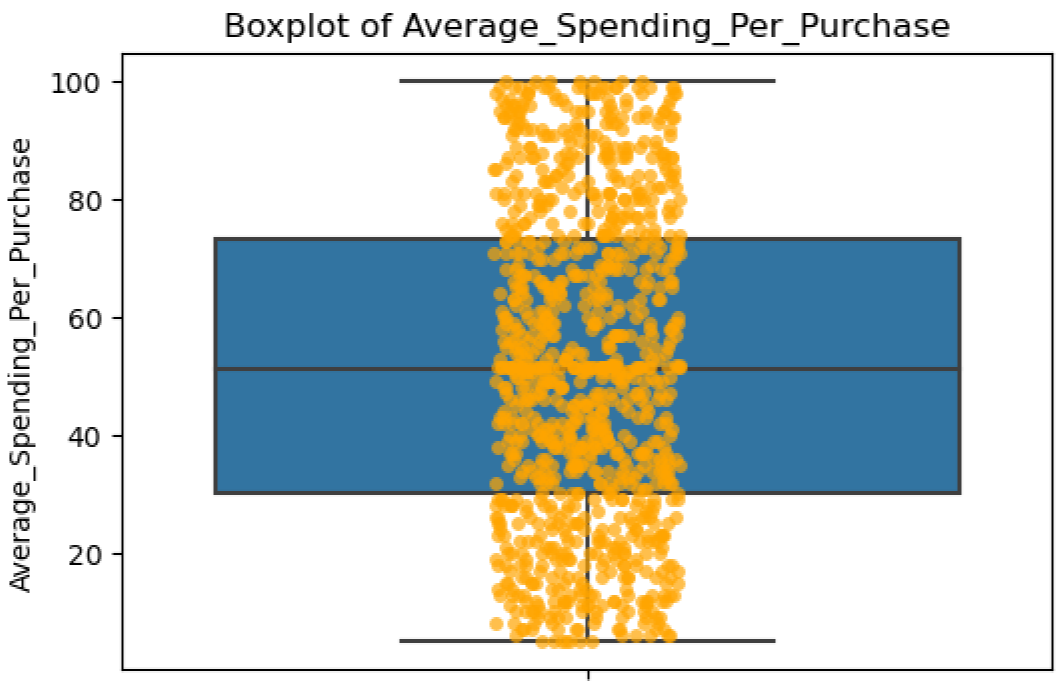

Histogram with KDE for Brand_Affinity_Score
Count
Brand_Affinity_Score
0
20
40
60
80
100
2
4
6
8
10

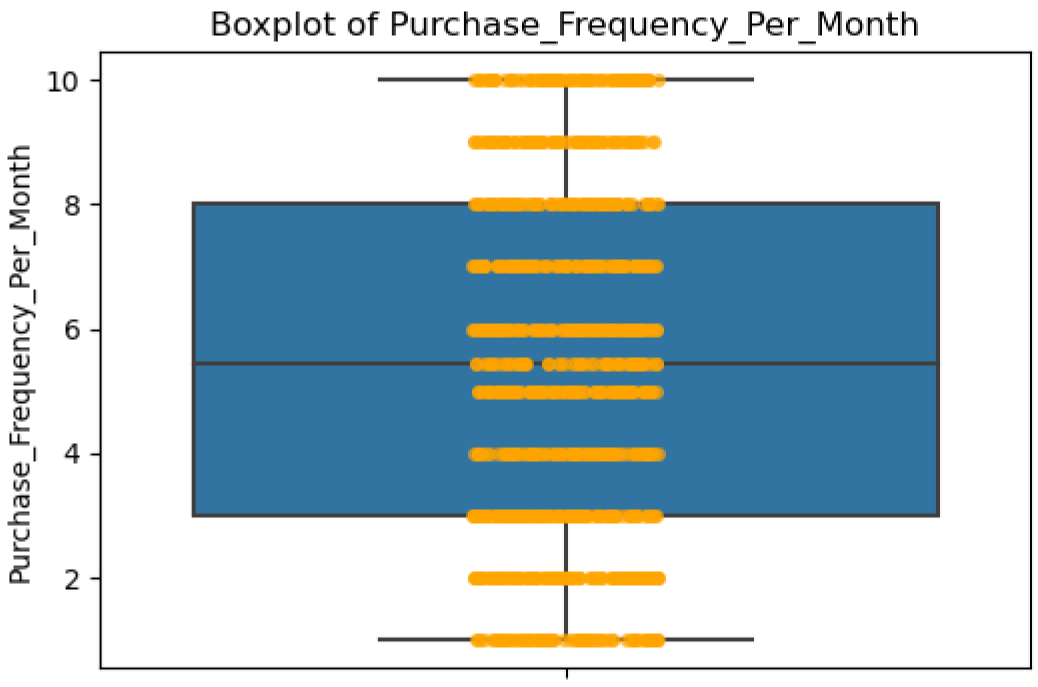

Histogram with KDE for Month
Count
0
20
40
60
80
100
2
4
6
8
10
12
Month

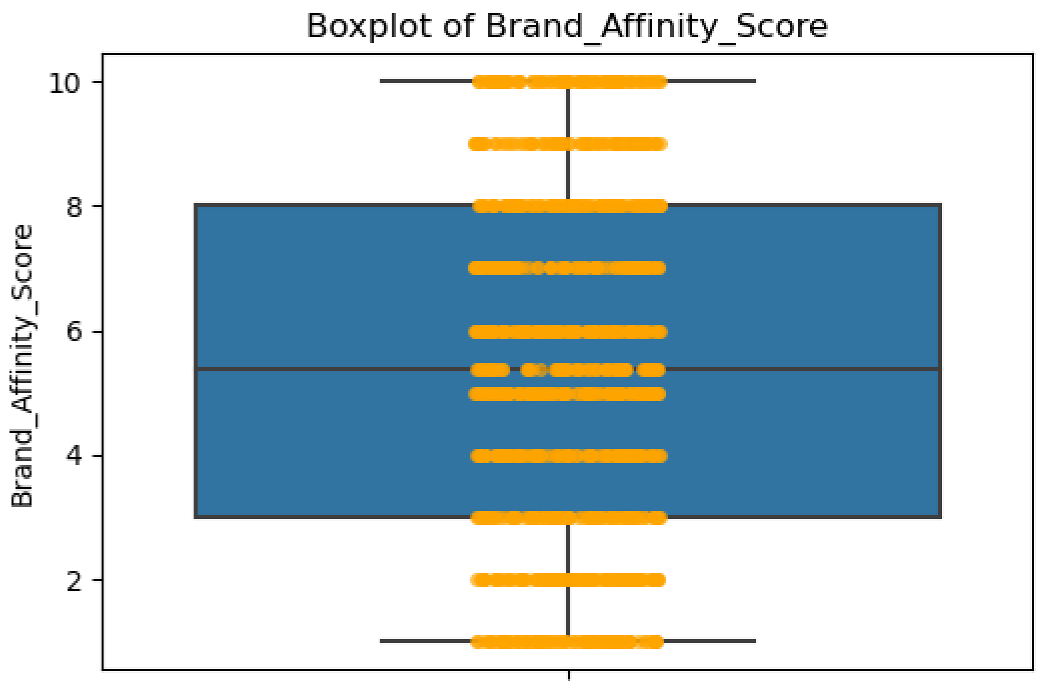

Histogram with KDE for Year
Count
0
20
40
60
80
100
120
1970
1980
1990
2000
2010
2020
Year

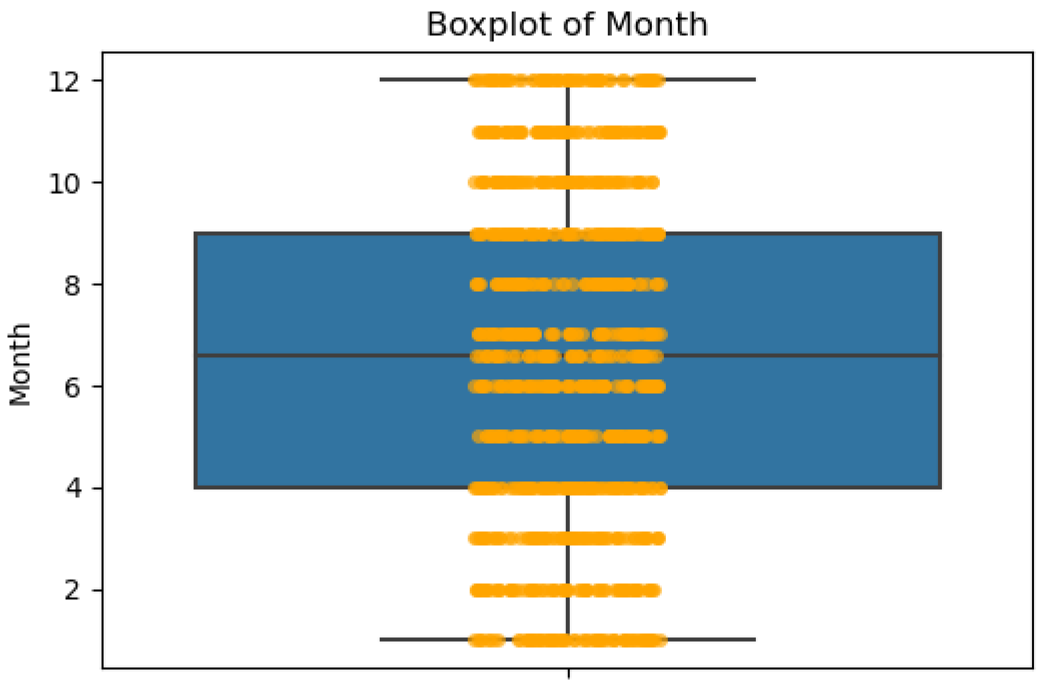

Boxplot of Age
Age
80
70
60
50
40
30
20

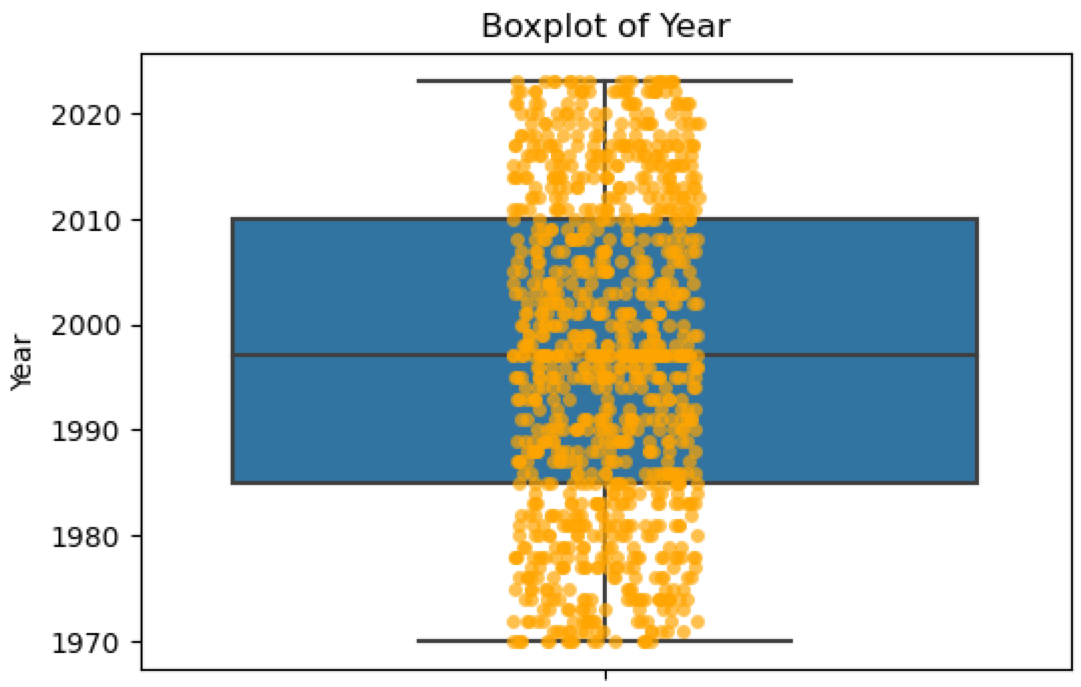

Boxplot of Purchase_Amount
Purchase_Amount
500
400
300
200
100
0


Boxplot of Average_Spending_Per_Purchase
Average_Spending_Per_Purchase
100
80
60
40
20


Boxplot of Purchase_Frequency_Per_Month
Purchase_Frequency_Per_Month
10
8
6
4
2


Boxplot of Brand_Affinity_Score
Brand_Affinity_Score
10
8
6
4
2


Boxplot of Month
Month
12
10
8
6
4
2

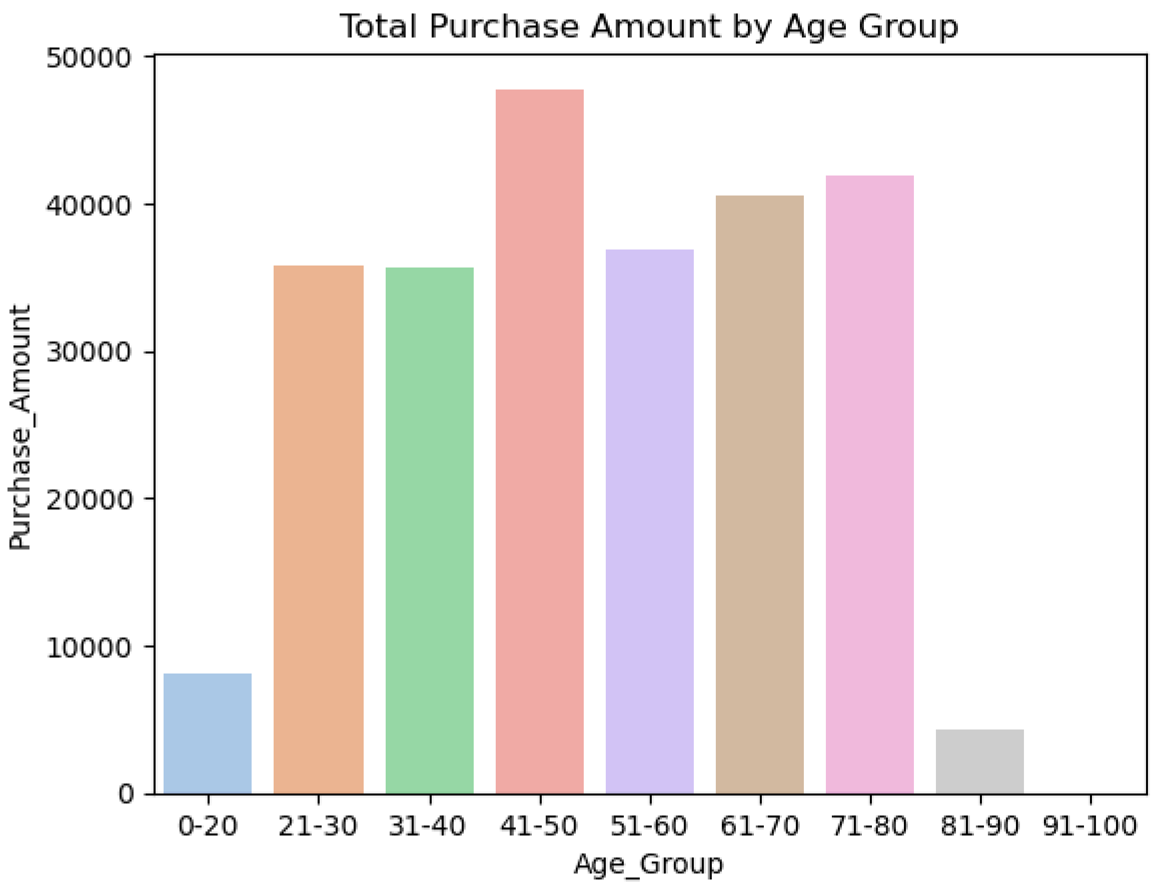


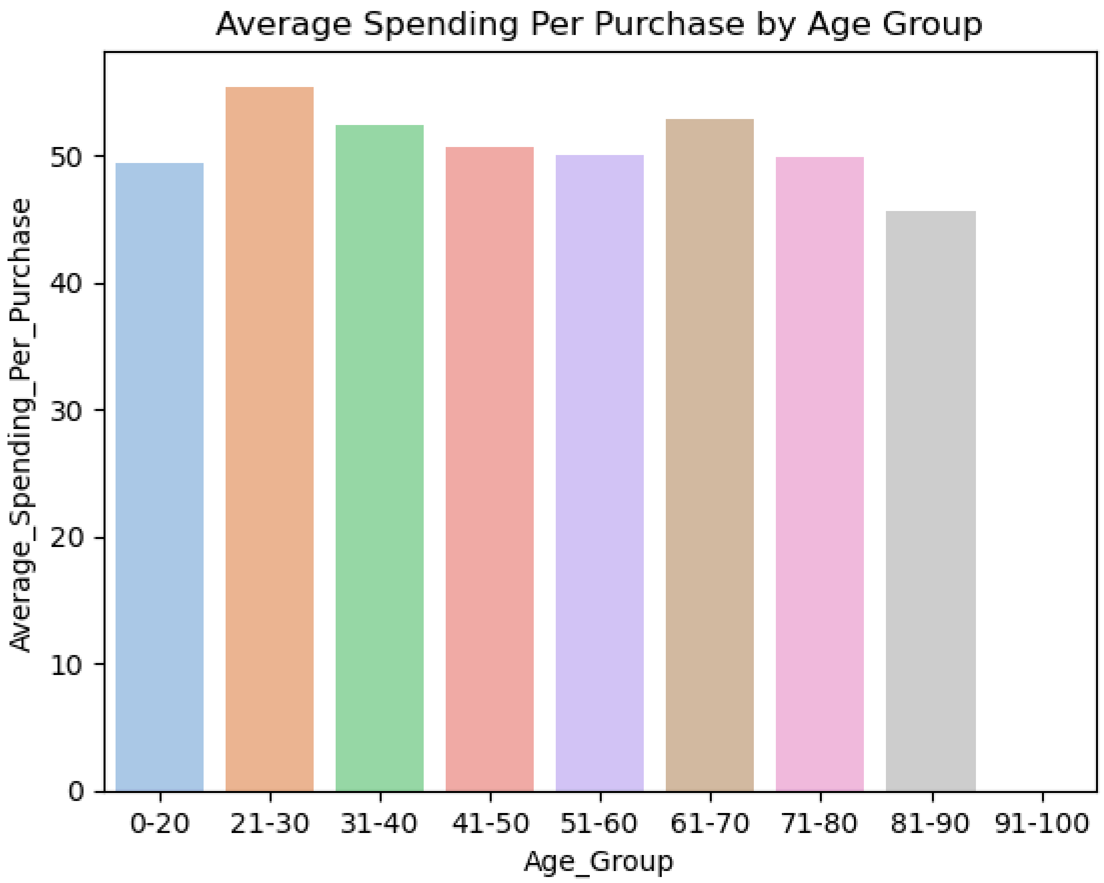
Out[189]:

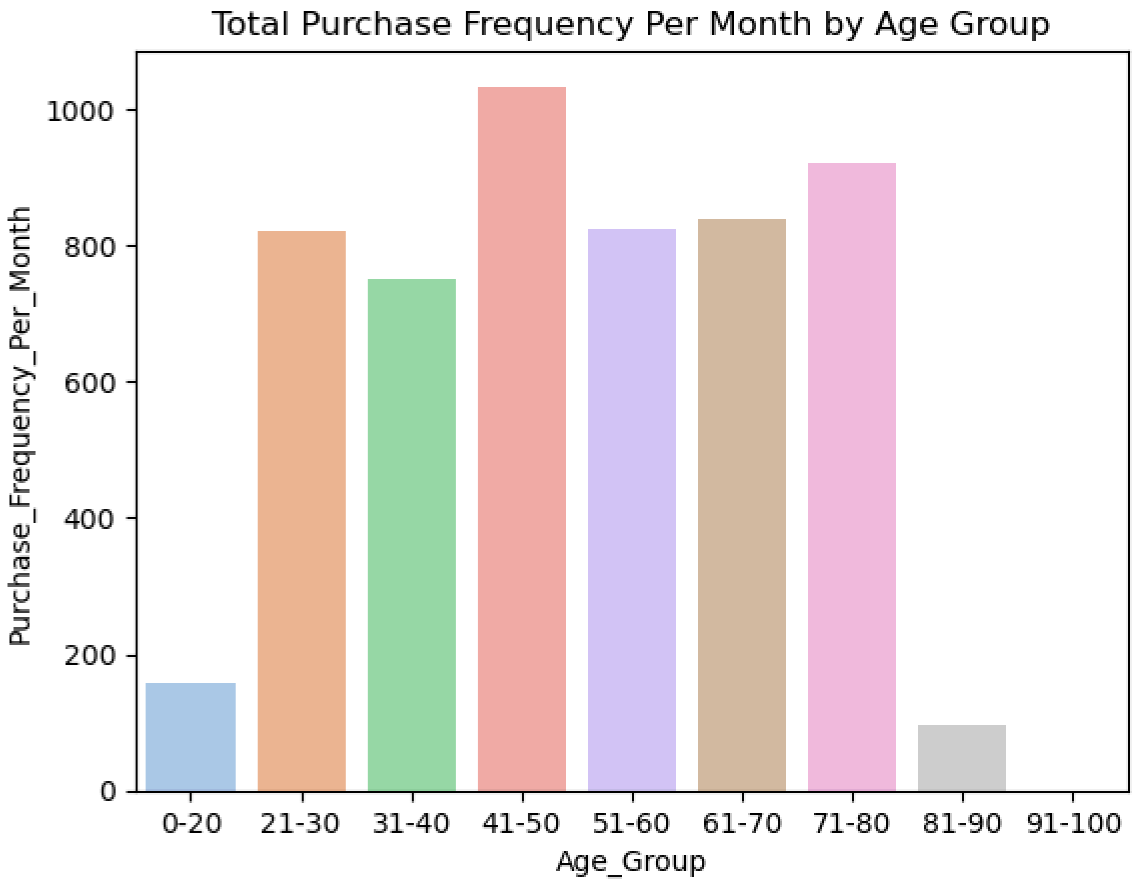
	Age	Purchase_Amount	Average_Spending_Per_Purchase	Purchase_Frequency_Per_Month
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	49.885417	250.629863	51.603125	5.437037
std	18.108487	137.515156	26.556676	2.765891
min	18.000000	10.000000	5.000000	1.000000
25%	35.000000	137.000000	30.000000	3.000000
50%	49.885417	250.629863	51.301563	5.437037
75%	66.000000	369.000000	73.250000	8.000000
max	80.000000	500.000000	100.000000	10.000000

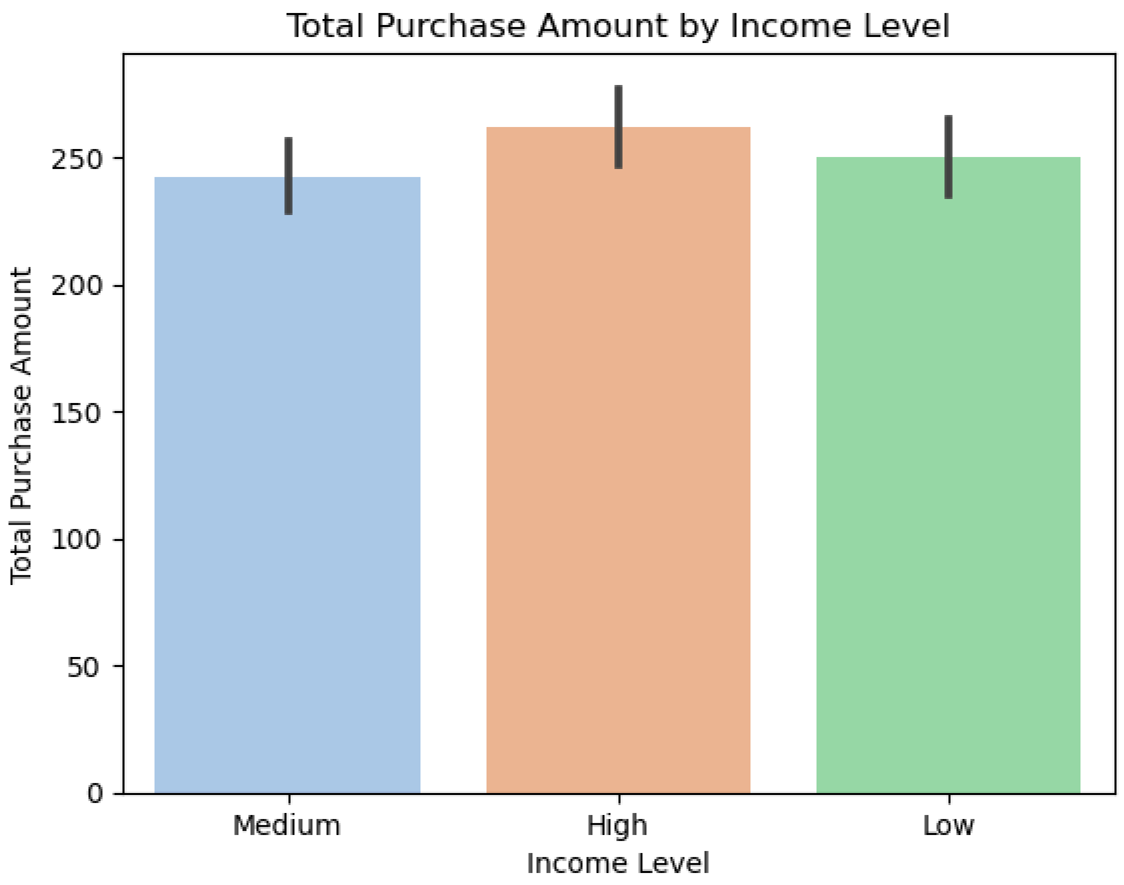
In [190…

```
age_bins = [0, 20, 30, 40, 50, 60, 70, 80, 90, 100]
age_labels = ['0-20', '21-30', '31-40', '41-50', '51-60', '61-70', '71-80', '81-90', '

df['Age_Group'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels, right=False)

# Aggregating data by age group to summarize key variables
age_group_stats = df.groupby('Age_Group').agg({
    'Purchase_Amount': 'sum',
    'Average_Spending_Per_Purchase': 'mean',
    'Purchase_Frequency_Per_Month': 'sum'
}).reset_index()

sns.barplot(x='Age_Group', y='Purchase_Amount', data=age_group_stats, palette='pastel'
plt.title('Total Purchase Amount by Age Group')
plt.show()

sns.barplot(x='Age_Group', y='Average_Spending_Per_Purchase', data=age_group_stats, pa
plt.title('Average Spending Per Purchase by Age Group')
plt.show()

sns.barplot(x='Age_Group', y='Purchase_Frequency_Per_Month', data=age_group_stats, pal
plt.title('Total Purchase Frequency Per Month by Age Group')
plt.show()

# Bar plot to show total purchase amount by different income levels
sns.barplot(x='Income_Level', y='Purchase_Amount', data=df, palette='pastel')
plt.title('Total Purchase Amount by Income Level')
plt.xlabel('Income Level')
plt.ylabel('Total Purchase Amount')
plt.show()
```




Average Spending Per Purchase by Age Group
Average_Spending_Per_Purchase
0
10
20
30
40
50
0-20
21-30
31-40
41-50
51-60
61-70
71-80
81-90
91-100
Age_Group


Total Purchase Frequency Per Month by Age Group
Purchase_Frequency_Per_Month
0
200
400
600
800
1000
0-20
21-30
31-40
41-50
51-60
61-70
71-80
81-90
91-100
Age_Group



We have observed that the numerical columns displayed normal distributions in their histograms and did not present significant outliers. This indicates a relatively standard and expected pattern in our data, which is beneficial for accurate analysis. By segmenting the data into age groups, We have effectively analyzed how purchase behaviors varied across different demographics. This step is crucial in understanding customer segments and tailoring strategies to different age categories. The bar plot showcasing the total purchase amount by different income levels revealed a clear trend: higher income levels correspond to higher purchase amounts.

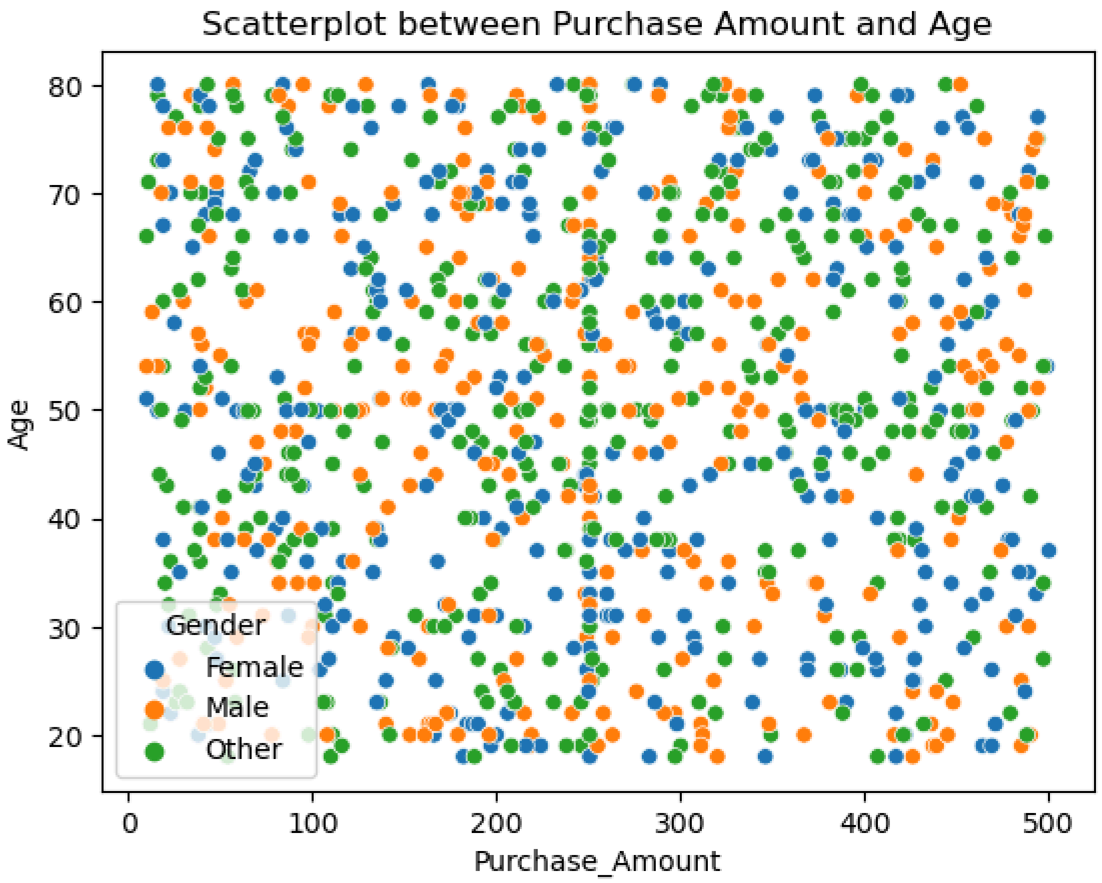
PART 2

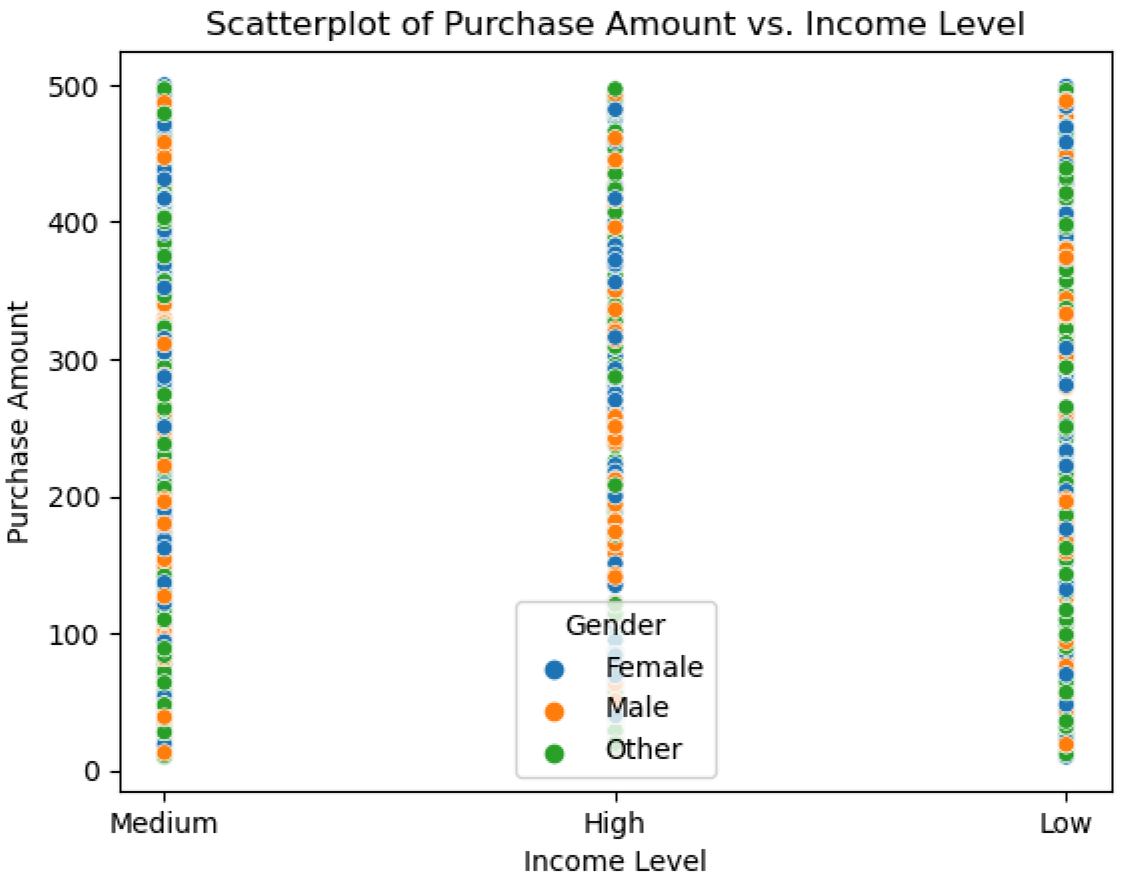
In Module 2, Part 2, We will conduct Bivariate Analysis to explore and understand the relationships between different variables in the dataset. Through scatterplots, we will visually examine correlations between pairs of variables like Purchase Amount, Age, Income Level, and Brand Affinity. Heatmaps will utilize to further analyze these relationships, particularly focusing on aggregate data like Purchase Amount by Income Level and Brand Affinity by Product Category.

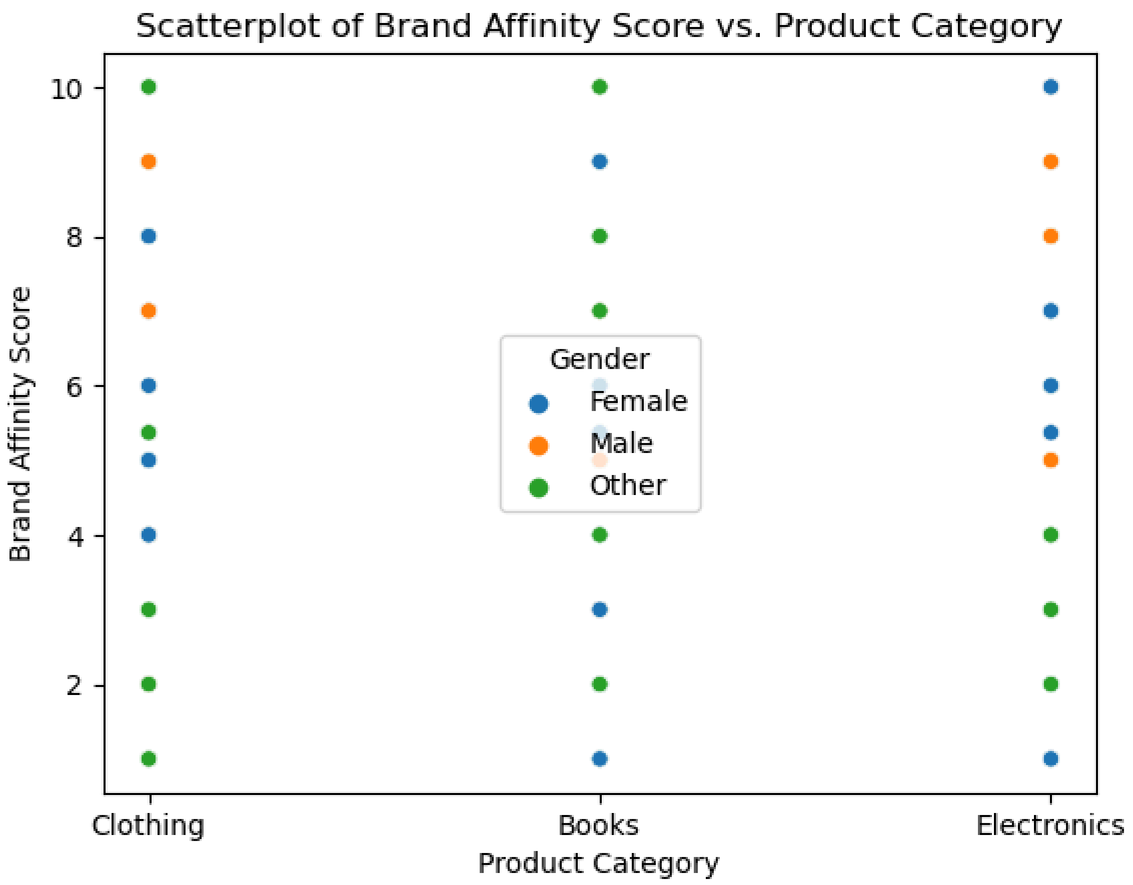
In [191…

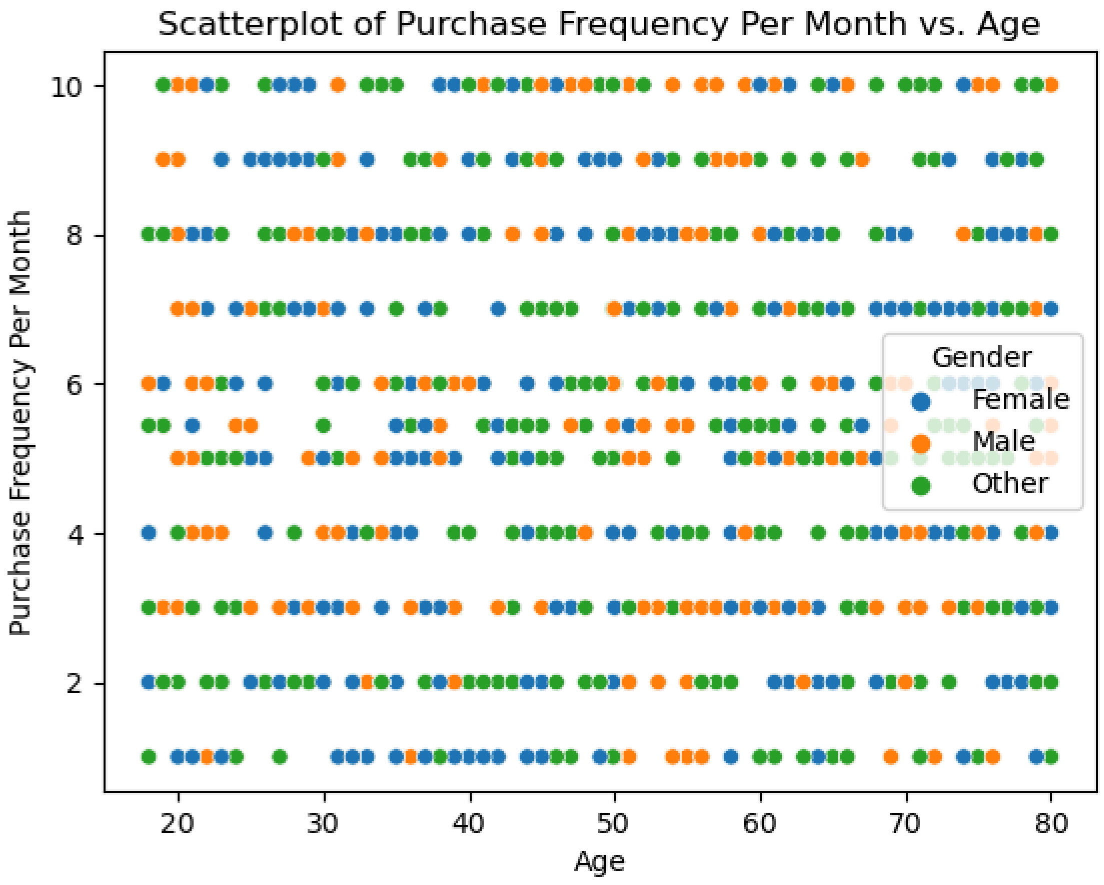
```
#Bivariate Analysis

sns.scatterplot(data=df, x='Purchase_Amount', y='Age', hue='Gender')
plt.title("Scatterplot between Purchase Amount and Age")
plt.show()
```

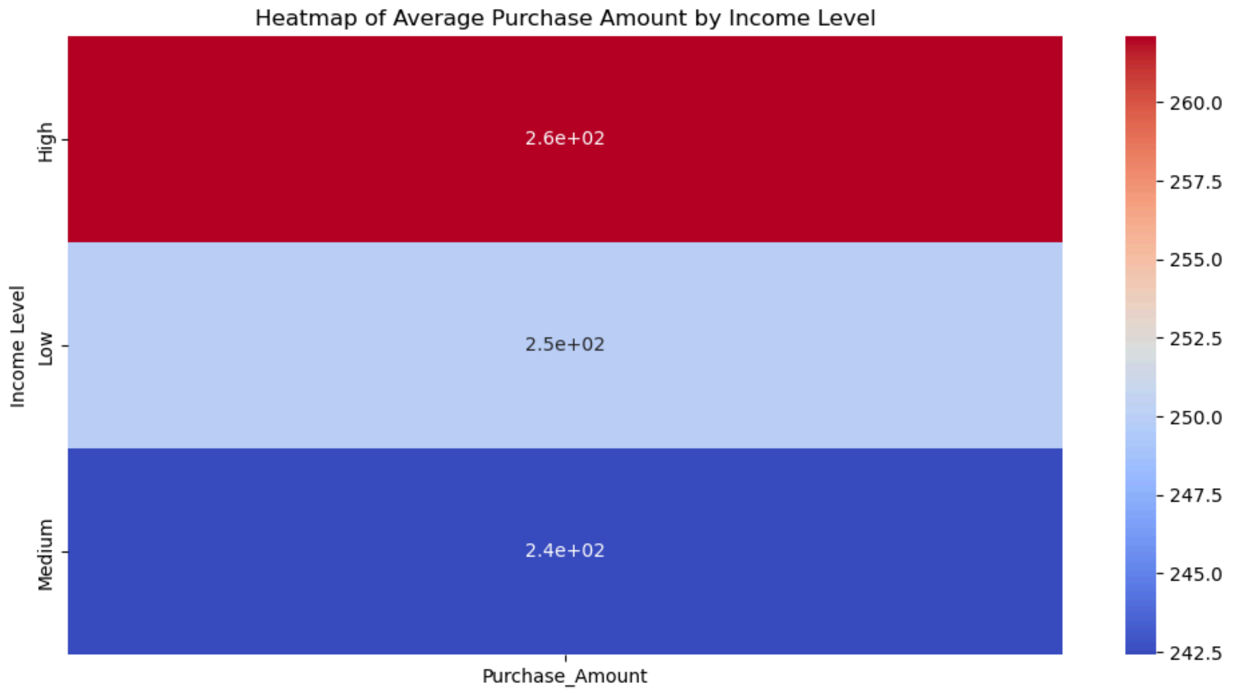
```
sns.scatterplot(data=df, x='Income_Level', y='Purchase_Amount', hue='Gender')
plt.title("Scatterplot of Purchase Amount vs. Income Level")
plt.xlabel("Income Level")
plt.ylabel("Purchase Amount")
plt.show()

# Scatterplot for Brand Affinity Score vs. Product Category
sns.scatterplot(data=df, x='Product_Category', y='Brand_Affinity_Score', hue='Gender')
plt.title("Scatterplot of Brand Affinity Score vs. Product Category")
plt.xlabel("Product Category")
plt.ylabel("Brand Affinity Score")
plt.show()

# Scatterplot for Purchase Frequency vs. Age
sns.scatterplot(data=df, x='Age', y='Purchase_Frequency_Per_Month', hue='Gender')
plt.title("Scatterplot of Purchase Frequency Per Month vs. Age")
plt.xlabel("Age")
plt.ylabel("Purchase Frequency Per Month")
plt.show()




# Aggregating Purchase Amount for each Income Level
purchase_income_agg = df.groupby('Income_Level')['Purchase_Amount'].mean().reset_index
# Creating a pivot table for the heatmap
pivot_purchase_income = purchase_income_agg.pivot_table(index='Income_Level', values='
# Heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(pivot_purchase_income, annot=True, cmap='coolwarm')
plt.title("Heatmap of Average Purchase Amount by Income Level")
plt.ylabel("Income Level")
plt.show()



# Aggregating Brand Affinity Score for each Product Category
brand_product_agg = df.groupby('Product_Category')['Brand_Affinity_Score'].mean().rese
# Creating a pivot table for the heatmap
pivot_brand_product = brand_product_agg.pivot_table(index='Product_Category', values='
# Heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(pivot_brand_product, annot=True, cmap='coolwarm')
plt.title("Heatmap of Average Brand Affinity Score by Product Category")
plt.ylabel("Product Category")
plt.show()


# Heatmap for Purchase Frequency and Age
plt.figure(figsize=(12, 6))
sns.heatmap(df[['Purchase_Frequency_Per_Month', 'Age']].corr(), annot=True, cmap='cool
plt.title("Correlation Heatmap between Purchase Frequency and Age")
plt.show()


heatmap_data = df.pivot_table(index='Age_Group', columns=['Product_Category', 'Brand']
plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm')
plt.title('Count of Purchases by Age Group, Product Category, and Brand')
plt.xlabel('Product Category - Brand')
plt.ylabel('Age Group')
plt.show()



# Encoding categorical variables for correlation analysis
le_income = LabelEncoder()
le_category = LabelEncoder()
df_encoded['Income_Level_Encoded'] = le_income.fit_transform(df['Income_Level'])
df_encoded['Product_Category_Encoded'] = le_category.fit_transform(df['Product_Categor


correlation_purchase_income = df_encoded['Purchase_Amount'].corr(df_encoded['Income_Le
correlation_brand_product = df_encoded['Brand_Affinity_Score'].corr(df_encoded['Produc
correlation_frequency_age = df_encoded['Purchase_Frequency_Per_Month'].corr(df_encodec

# Displaying the correlations
print("Correlation between Purchase Amount and Income Level:", correlation_purchase_ir
print("Correlation between Brand Affinity and Product Category:", correlation_brand_pr
print("Correlation between Purchase Frequency and Age:", correlation_frequency_age)


# Heatmap for correlations
correlation_matrix = df[numerical_columns_specific].corr()
plt.figure(figsize=(12, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()



correlation_matrix
```

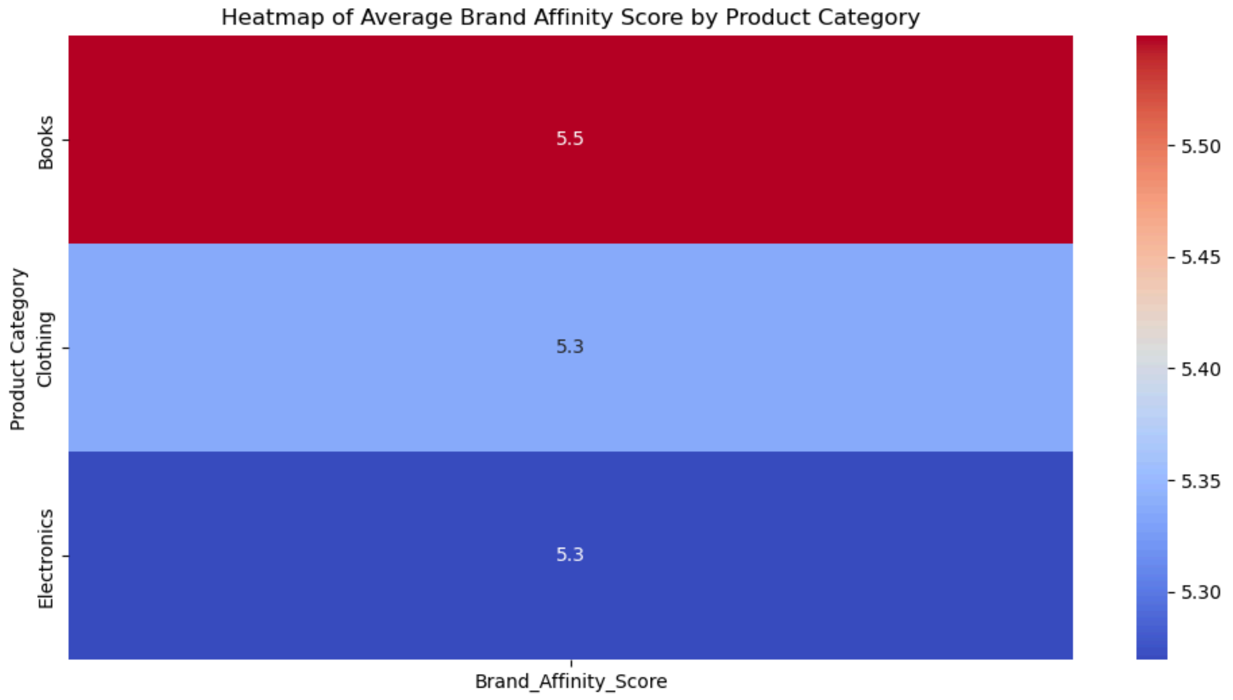

Scatterplot between Purchase Amount and Age
Age
Purchase_Amount
Gender
Female
Male
Other
0
100
200
300
400
500
20
30
40
50
60
70
80

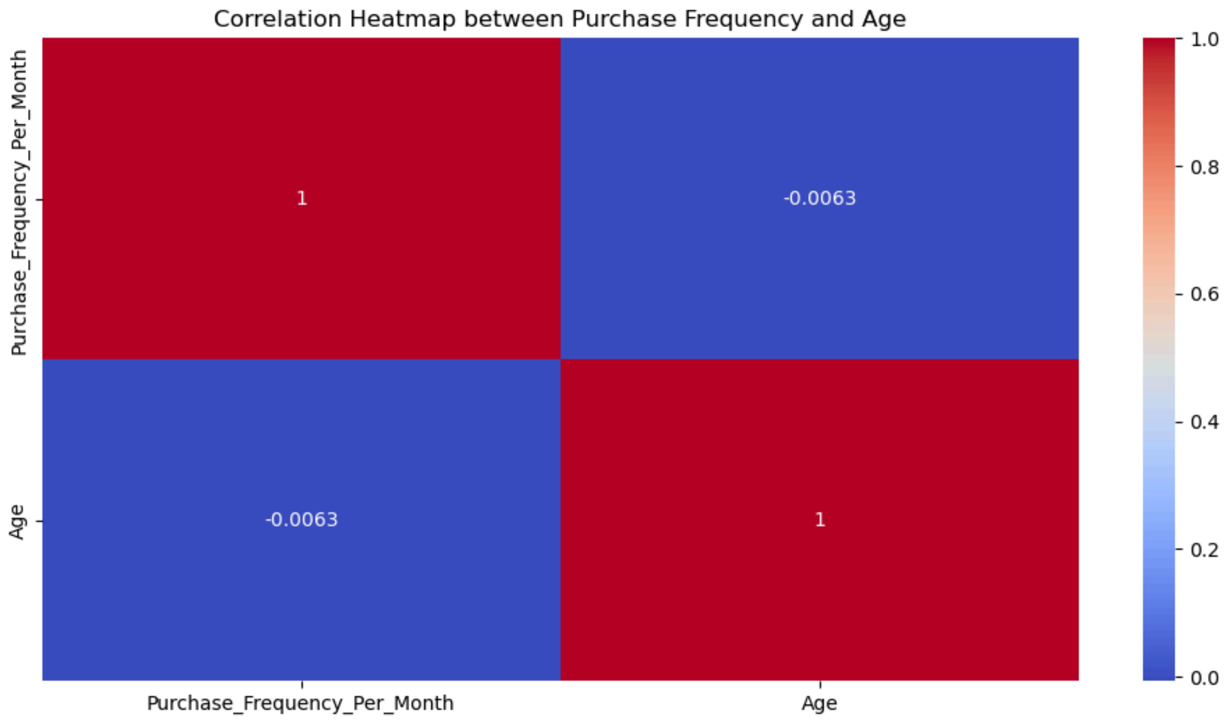

Scatterplot of Purchase Amount vs. Income Level
Purchase Amount
Income Level
Medium
High
Low
Gender
Female
Male
Other
0
100
200
300
400
500

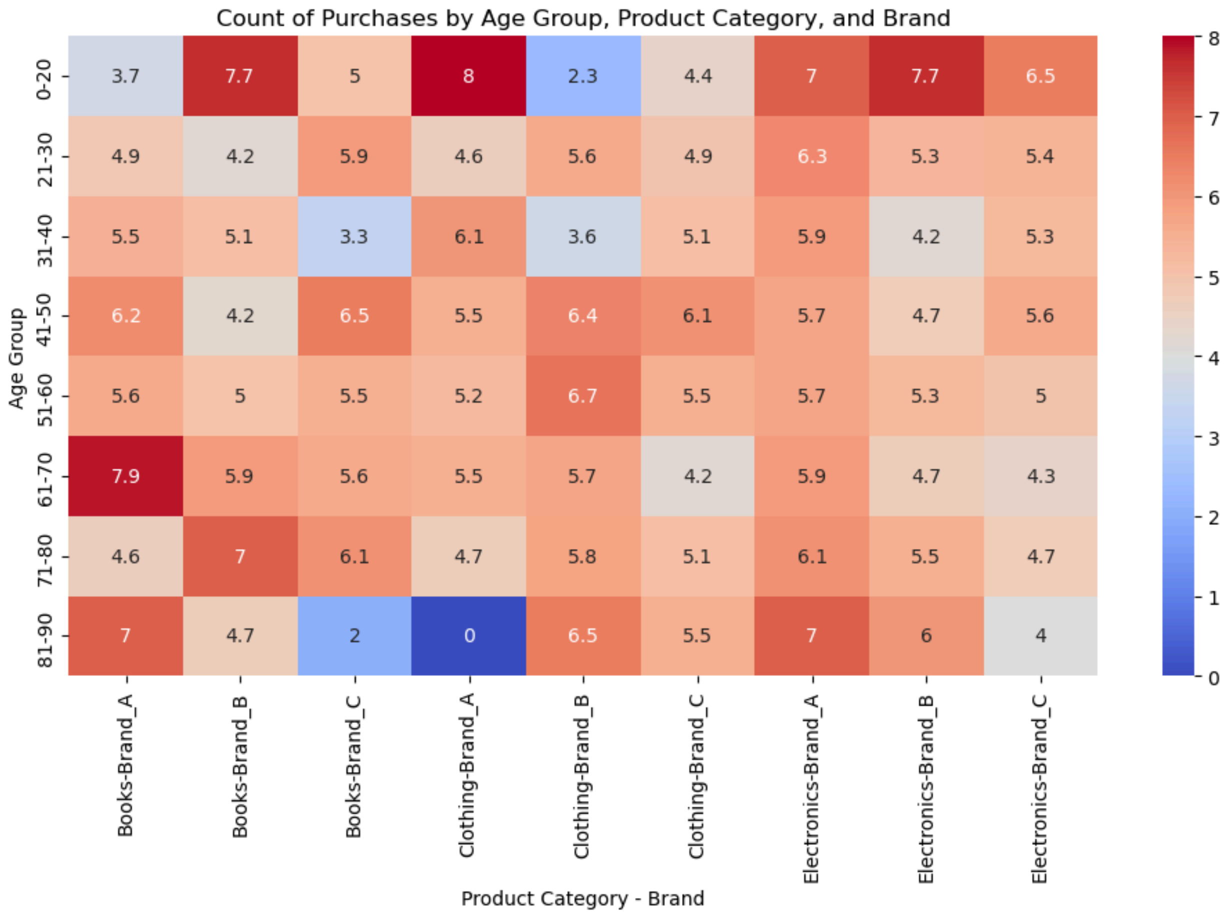

Scatterplot of Brand Affinity Score vs. Product Category
Brand Affinity Score
10
8
6
4
2
Clothing
Books
Electronics
Product Category
Gender
Female
Male
Other


Scatterplot of Purchase Frequency Per Month vs. Age
Purchase Frequency Per Month
10
8
6
4
2
20
30
40
50
60
70
80
Age
Gender
Female
Male
Other


Heatmap of Average Purchase Amount by Income Level
Income Level
High
Low
Medium
2.6e+02
2.5e+02
2.4e+02
Purchase_Amount
260.0
257.5
255.0
252.5
250.0
247.5
245.0
242.5

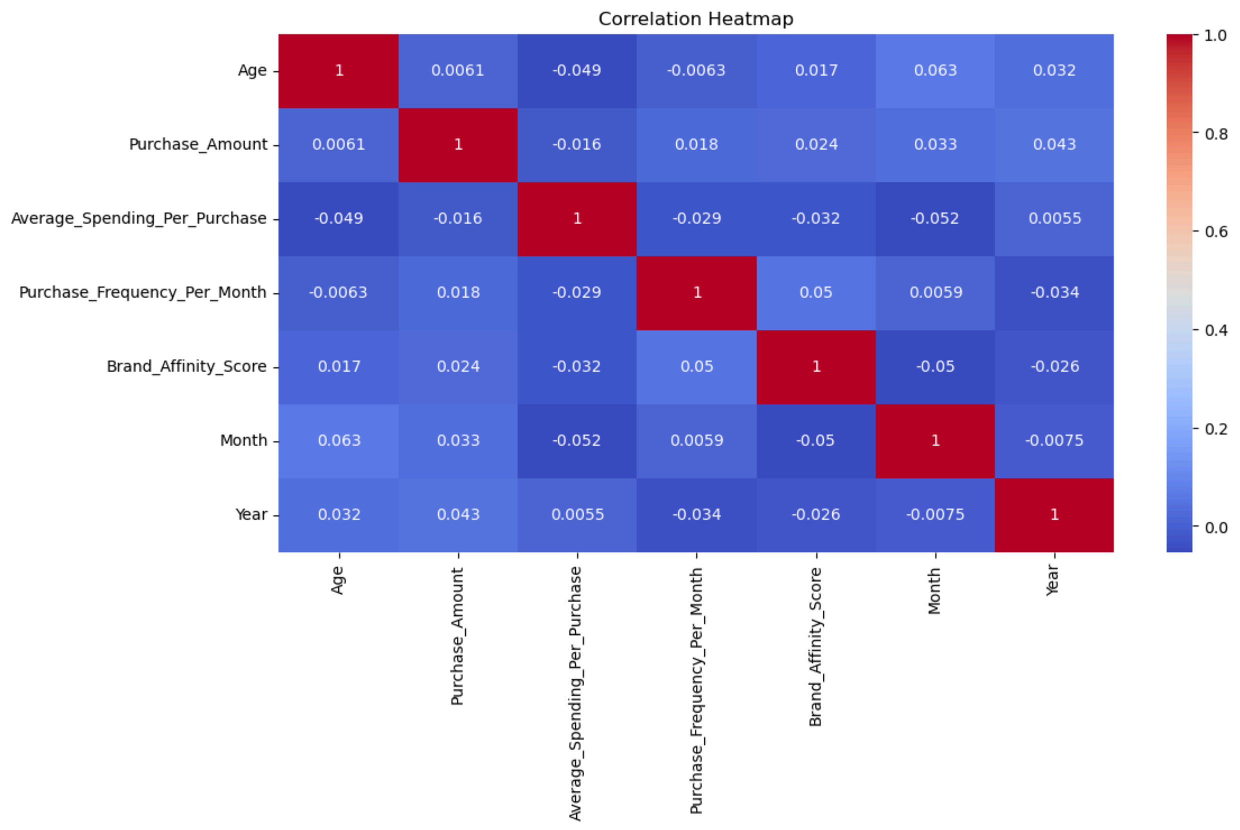

Heatmap of Average Brand Affinity Score by Product Category
Product Category
Books
Clothing
Electronics
5.5
5.3
5.3
Brand_Affinity_Score
5.50
5.45
5.40
5.35
5.30





Age Group	Books-Brand_A	Books-Brand_B	Books-Brand_C	Clothing-Brand_A	Clothing-Brand_B	Clothing-Brand_C	Electronics-Brand_A	Electronics-Brand_B	Electronics-Brand_C
0-20	3.7	7.7	5	8	2.3	4.4	7	7.7	6.5
21-30	4.9	4.2	5.9	4.6	5.6	4.9	6.3	5.3	5.4
31-40	5.5	5.1	3.3	6.1	3.6	5.1	5.9	4.2	5.3
41-50	6.2	4.2	6.5	5.5	6.4	6.1	5.7	4.7	5.6
51-60	5.6	5	5.5	5.2	6.7	5.5	5.7	5.3	5
61-70	7.9	5.9	5.6	5.5	5.7	4.2	5.9	4.7	4.3
71-80	4.6	7	6.1	4.7	5.8	5.1	6.1	5.5	4.7
81-90	7	4.7	2	0	6.5	5.5	7	6	4

```
Correlation between Purchase Amount and Income Level: -0.05874510213269463
Correlation between Brand Affinity and Product Category: -0.04067402363111571
Correlation between Purchase Frequency and Age: -0.0062595046999421085
```



Out[191]:

	Age	Purchase_Amount	Average_Spending_Per_Purchase	Purchas
Age	1.000000	0.006050	-0.049116	
Purchase_Amount	0.006050	1.000000	-0.015922	
Average_Spending_Per_Purchase	-0.049116	-0.015922	1.000000	
Purchase_Frequency_Per_Month	-0.006260	0.017931	-0.029020	
Brand_Affinity_Score	0.016884	0.023574	-0.031535	
Month	0.063402	0.032568	-0.052471	
Year	0.031937	0.042759	0.005484	

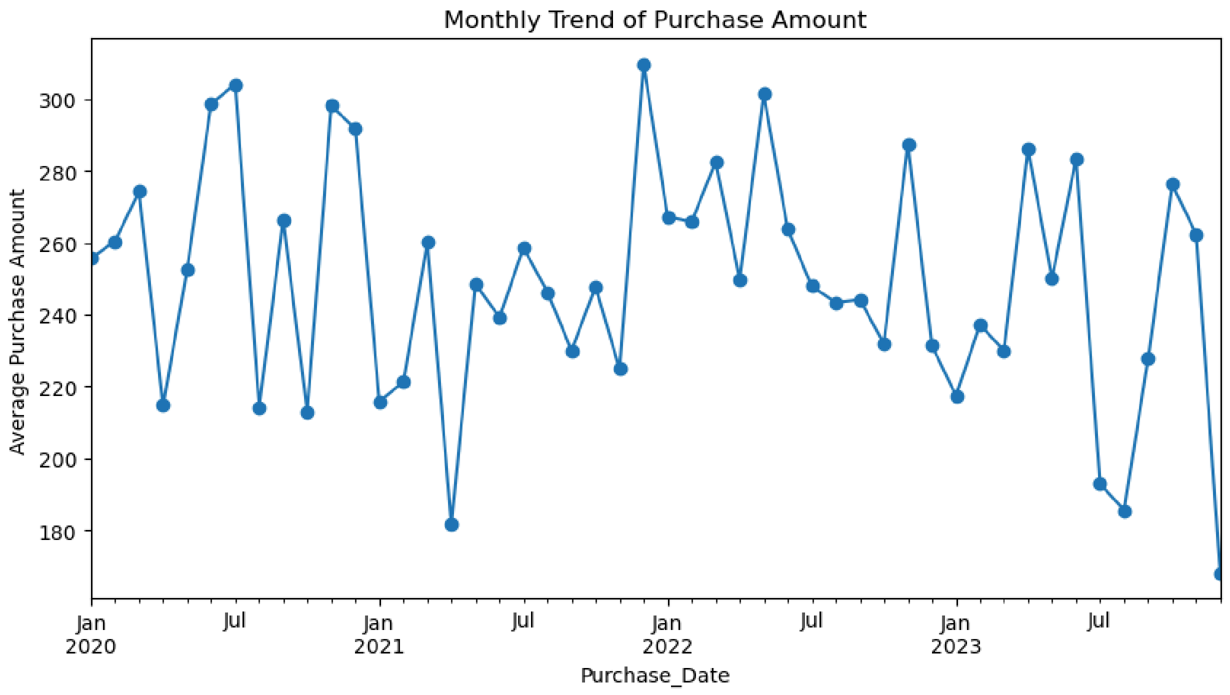
PART 3

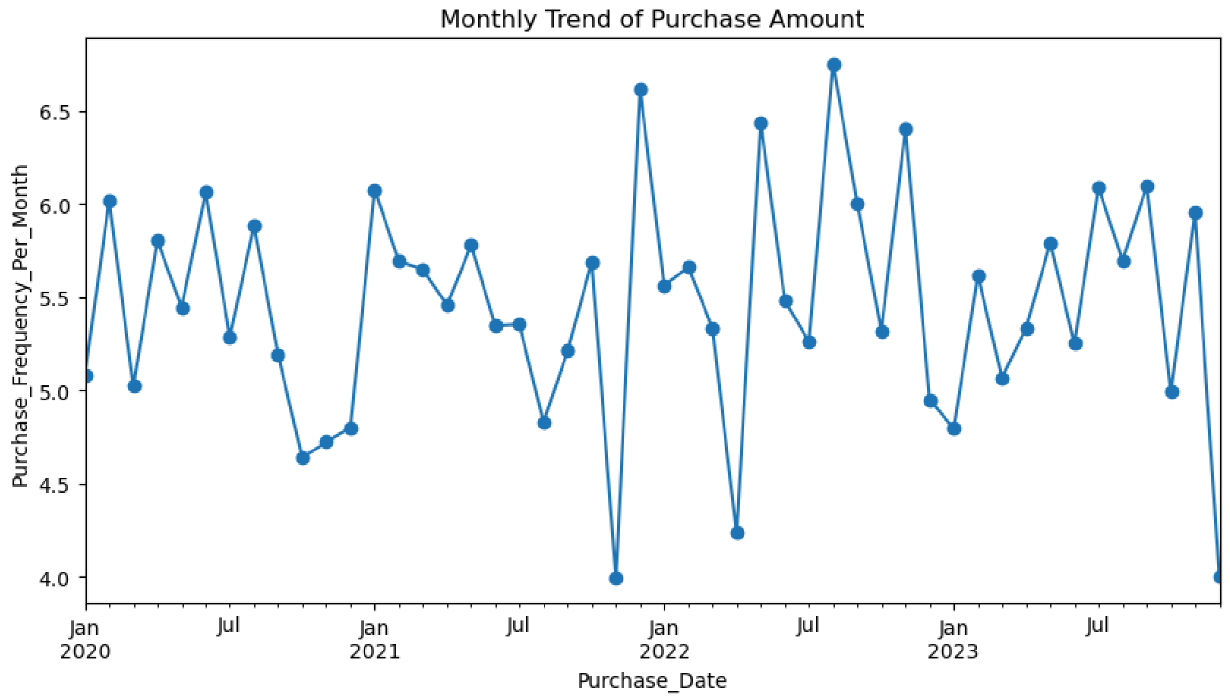
In Module 2, Part 3 of your project, We will conduct Temporal Analysis to examine trends and patterns in customer behavior over time. We will analyze the monthly trends in purchase amounts, revealing how average spending will fluctuate across different months. Additionally, We will explore seasonal variations in spending, providing insights into how customer purchase behavior will changes with seasons. Lastly, We will assessed the popularity of different brands by counting transactions, which will help in understanding brand preferences among customers. This analysis will be crucial for identifying time-related trends and patterns, aiding in strategic planning and decision-making for better customer engagement and sales optimization.

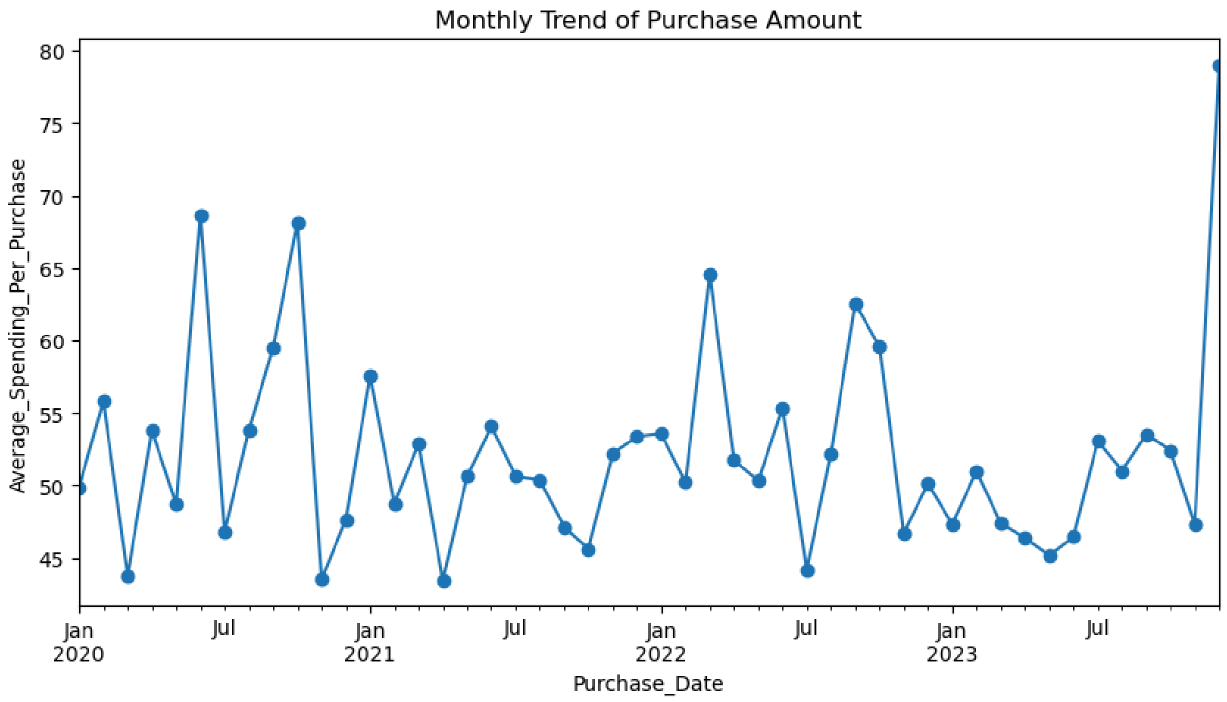
In [192…

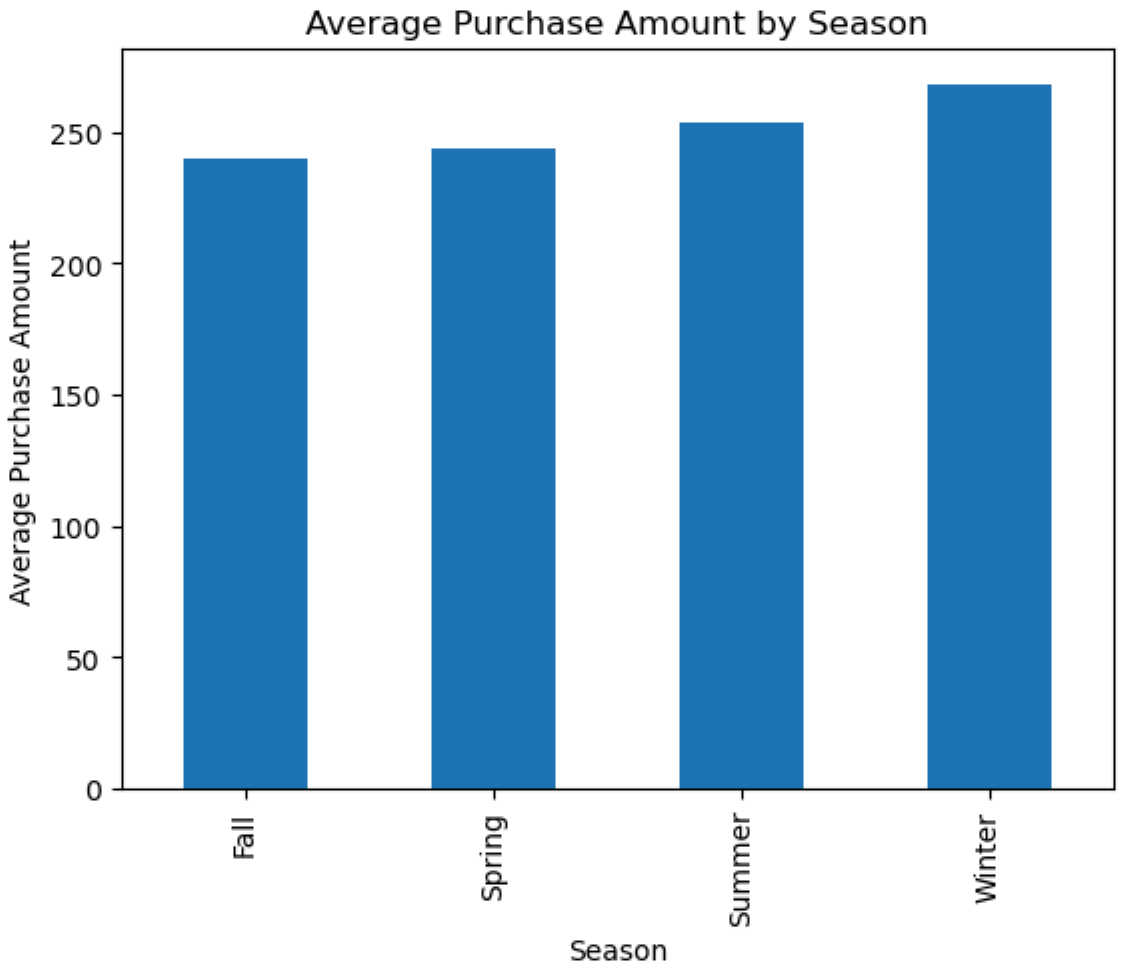
```
#  Temporal Analysis
```

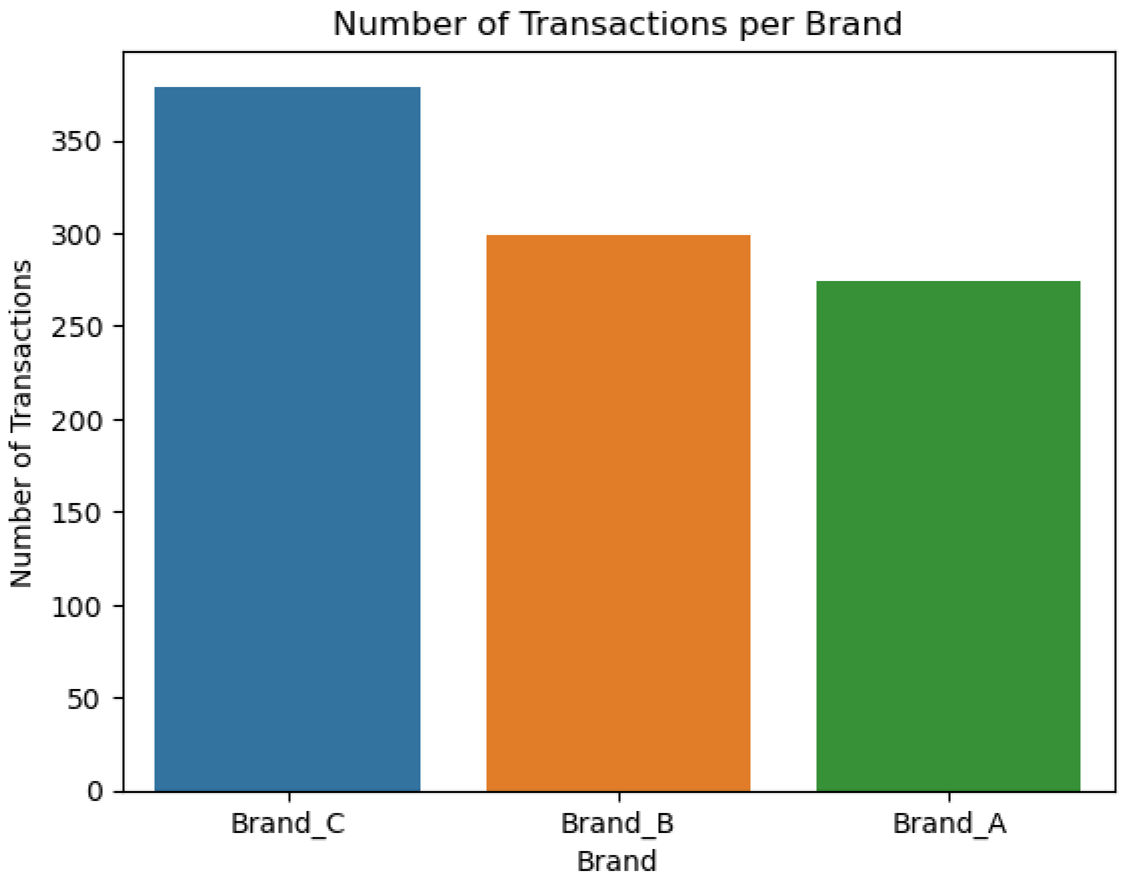
```
#ternd over time
df.set_index('Purchase_Date')['Purchase_Amount'].resample('M').mean().plot(marker='o',
plt.title('Monthly Trend of Purchase Amount')
plt.ylabel('Average Purchase Amount')
plt.show()


df.set_index('Purchase_Date')['Purchase_Frequency_Per_Month'].resample('M').mean().plo
plt.title('Monthly Trend of Purchase Amount')
plt.ylabel('Purchase_Frequency_Per_Month')
plt.show()



df.set_index('Purchase_Date')['Average_Spending_Per_Purchase'].resample('M').mean().pl
plt.title('Monthly Trend of Purchase Amount')
plt.ylabel('Average_Spending_Per_Purchase')
plt.show()





#Seasonal Variations:
df.groupby('Season')['Purchase_Amount'].mean().plot(kind='bar')
plt.title('Average Purchase Amount by Season')
plt.ylabel('Average Purchase Amount')
plt.show()


brand_sales = df.groupby('Brand')['Transaction_ID'].nunique()
brand_sales_sorted = brand_sales.sort_values(ascending=False)
sns.barplot(x=brand_sales_sorted.index, y=brand_sales_sorted.values)
plt.title('Number of Transactions per Brand')
plt.xlabel('Brand')
plt.ylabel('Number of Transactions')
plt.show()



age_group_counts = df['Age_Group'].value_counts()
plt.figure(figsize=(10, 6))
bars = plt.bar(age_group_counts.index, age_group_counts.values, color=['red', 'blue',
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.title('Distribution of Age Groups')
plt.show()
```

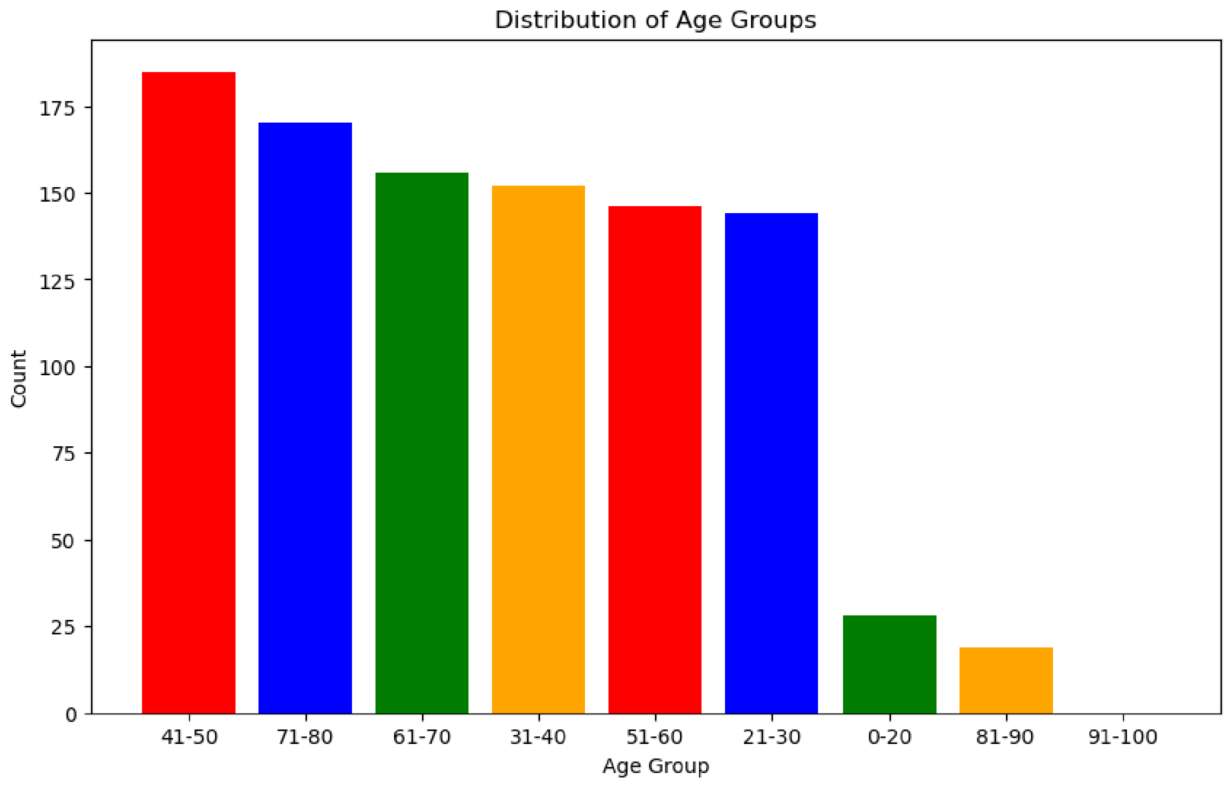

Monthly Trend of Purchase Amount
Average Purchase Amount
300
280
260
240
220
200
180
Jan 2020
Jul
Jan 2021
Jul
Jan 2022
Jul
Jan 2023
Jul
Purchase_Date


Monthly Trend of Purchase Amount
Purchase_Frequency_Per_Month
6.5
6.0
5.5
5.0
4.5
4.0
Jan 2020
Jul
Jan 2021
Jul
Jan 2022
Jul
Jan 2023
Jul
Purchase_Date


Monthly Trend of Purchase Amount
Average_Spending_Per_Purchase
80
75
70
65
60
55
50
45
Jan 2020
Jul
Jan 2021
Jul
Jan 2022
Jul
Jan 2023
Jul
Purchase_Date


Average Purchase Amount by Season
Average Purchase Amount
250
200
150
100
50
0
Fall
Spring
Summer
Winter
Season


Number of Transactions per Brand
Number of Transactions
350
300
250
200
150
100
50
0
Brand_C
Brand_B
Brand_A
Brand


Distribution of Age Groups
Count
175
150
125
100
75
50
25
0
41-50
71-80
61-70
31-40
51-60
21-30
0-20
81-90
91-100
Age Group

Summery

We have implemented a line graph to observe customer purchasing patterns over time on a monthly basis. This visualization helped in identifying any significant trends or shifts in customer spending habits across different months. The bar plot showing average spending by season revealed that the winter season exhibited higher spending. This insight is valuable for understanding how customer spending habits are influenced by seasonal factor. Finally, We have analyzed brand popularity through a bar plot, which indicated that Brand C experienced the highest number of sales. This information is crucial for understanding brand preferences and customer choices within the electronics section.

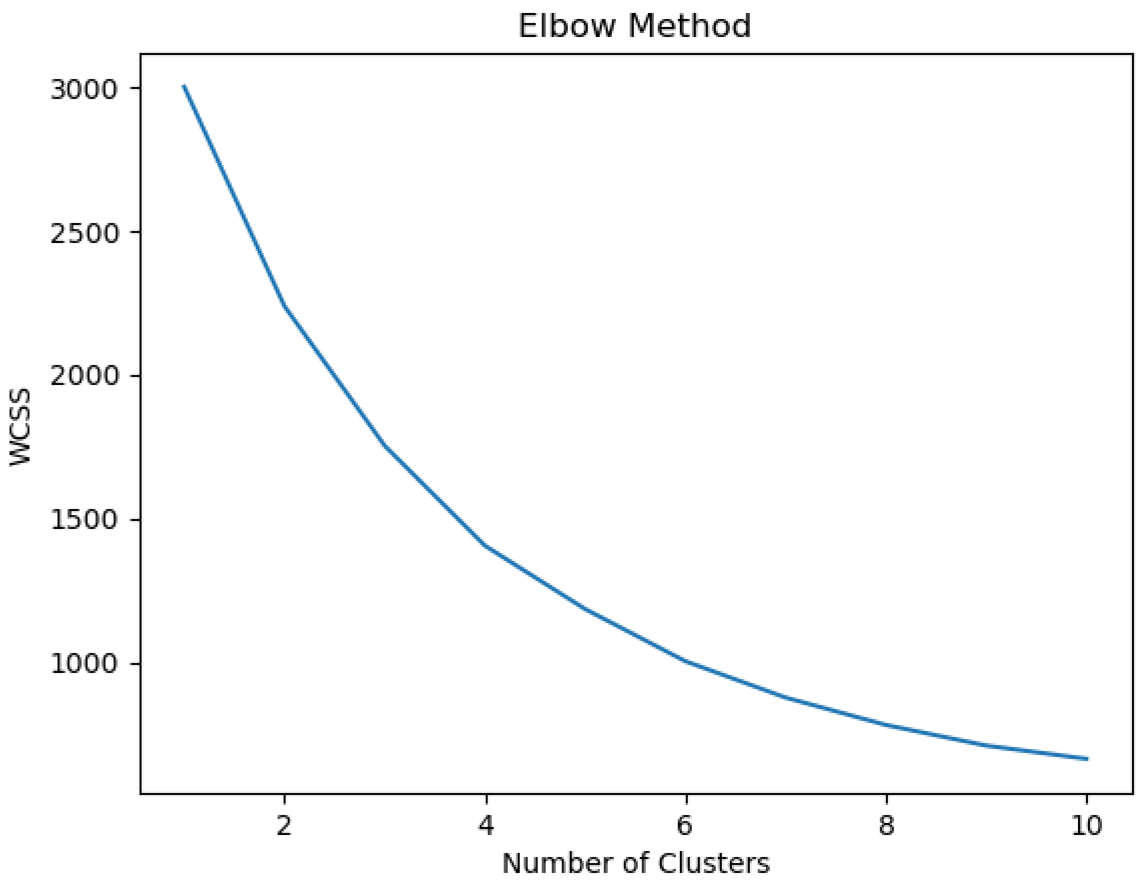
MODULE 3

K MEANS

In [193…
```
from sklearn.cluster import KMeans  # Clustering algorithm
from sklearn.metrics import silhouette_score  # Cluster quality metric
from sklearn.preprocessing import StandardScaler  # Feature scaling
from sklearn.decomposition import PCA  # Dimensionality reduction
from sklearn.neighbors import NearestNeighbors  # Find nearest neighbors
from sklearn.cluster import DBSCAN  # Density-based clustering
```

In [194…
```
#Standardize the features to ensure equal weightage in clustering.
values = {'Low': 0, 'Medium': 0.5, 'High': 1}
df['Product_Category_Preferences_Value'] = df['Product_Category_Preferences'].map(valu
df['Product_Category_Preferences_Value'].fillna(df['Product_Category_Preferences_Value

average_purchase_amount = df['Purchase_Amount'].mean()
df['avg_Purchase_Amount'] = average_purchase_amount

df



features_to_scale = [ 'Average_Spending_Per_Purchase',
                     'Brand_Affinity_Score', 'Product_Category_Preferences_Value']
# features_to_scale = [ 'avg_Purchase_Amount',
#                      'Brand_Affinity_Score', 'Product_Category_Preferences_Value']
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features_to_scale])
```

In [195…
```
#Determine the number of clusters
#Elbow Method


wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    kmeans.fit(scaled_features)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



In [151…

```
# Silhouette Analysis

for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    cluster_labels = kmeans.fit_predict(scaled_features)
    silhouette_avg = silhouette_score(scaled_features, cluster_labels)
    print(f"Silhouette Score for {i} clusters: {silhouette_avg}")
```

```
Silhouette Score for 2 clusters: 0.2494942410320902
Silhouette Score for 3 clusters: 0.25960331712687773
Silhouette Score for 4 clusters: 0.2725464503944895
Silhouette Score for 5 clusters: 0.27567709283846836
Silhouette Score for 6 clusters: 0.2885893848357499
Silhouette Score for 7 clusters: 0.2861792063923574
Silhouette Score for 8 clusters: 0.2920102783956405
Silhouette Score for 9 clusters: 0.30513997031177964
Silhouette Score for 10 clusters: 0.3053673995886255
```

In [162…

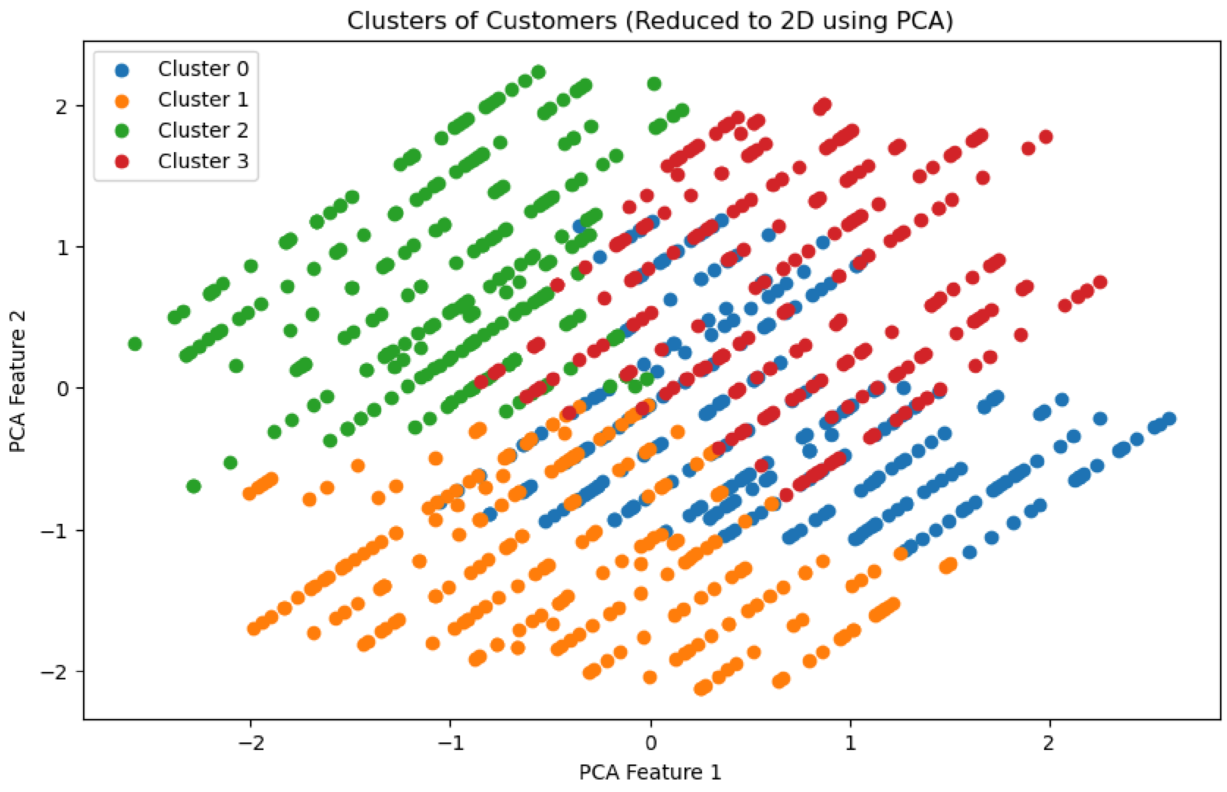
```
#Apply the K-Means Algorithm

optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, init='random', n_init=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_features)
cluster_analysis = df.groupby('Cluster')[features_to_scale].mean()
cluster_analysis
```

Out[162]:

Cluster	Average_Spending_Per_Purchase	Brand_Affinity_Score	Product_Category_Preferences_Value
0	31.302187	5.162892	0.103448
1	81.373913	5.671454	0.180435
2	56.058070	2.904748	0.831325
3	42.645157	7.985224	0.811688

In [163…

```
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_features)

# Plotting the clusters
plt.figure(figsize=(10, 6))
for i in range(optimal_k):
    plt.scatter(reduced_data[df['Cluster'] == i, 0], reduced_data[df['Cluster'] == i,
plt.title('Clusters of Customers (Reduced to 2D using PCA)')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.legend()
plt.show()
```

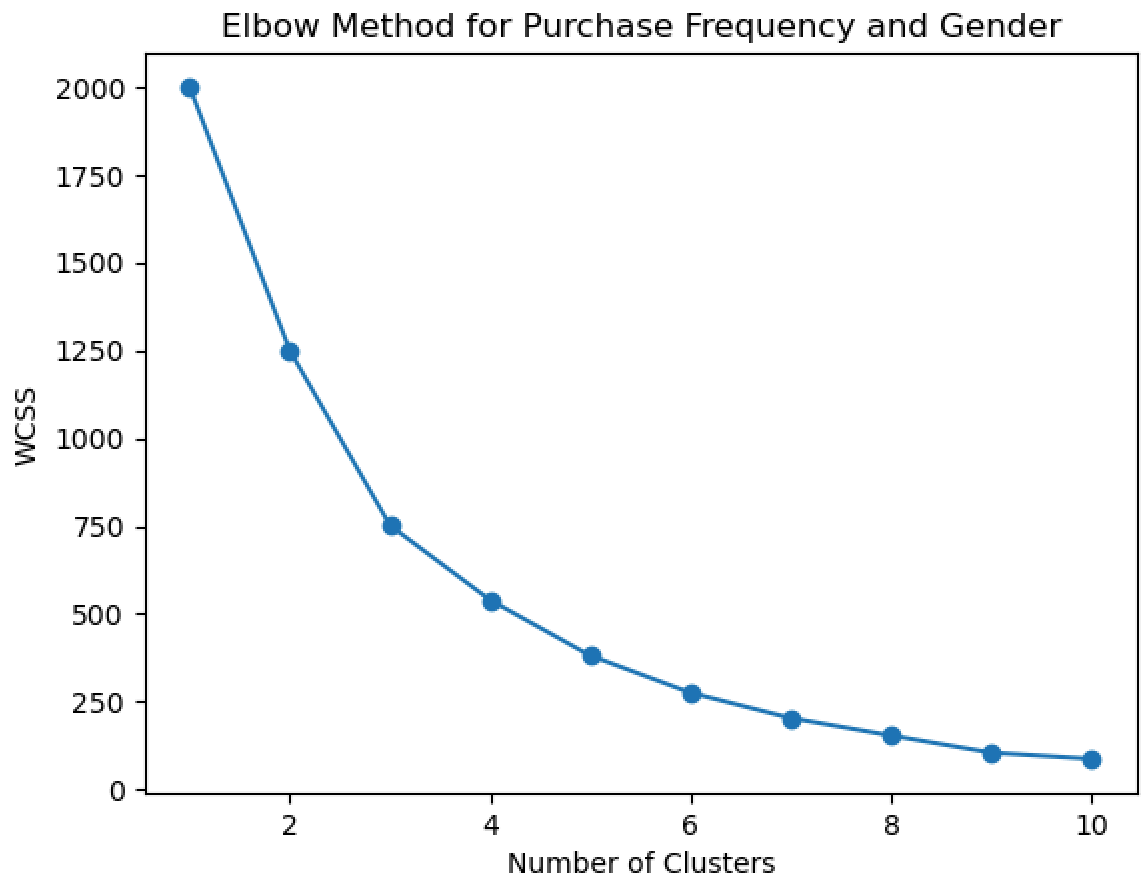


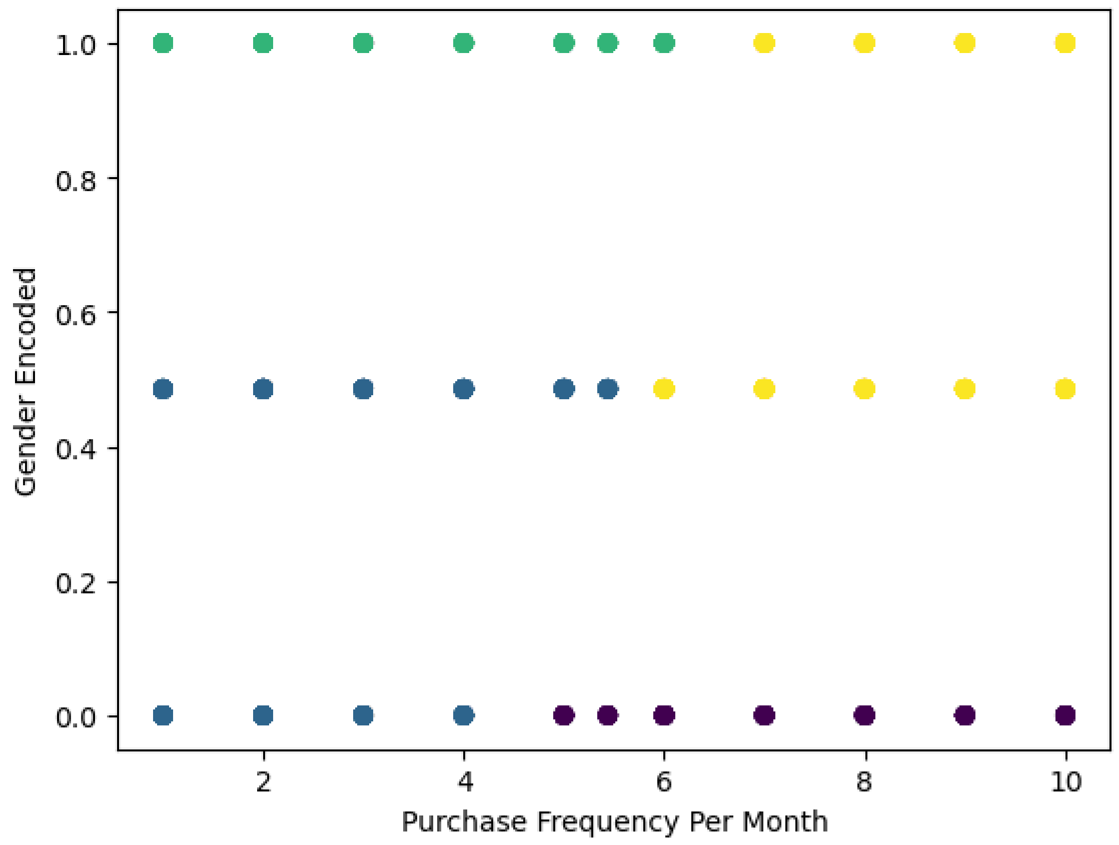
Summary

In Module 3, Part 1, we implemented K-Means Clustering to segment customers based on their purchase behaviors and preferences. we began by standardizing selected features for equal weighting in clustering. Then, we determined the optimal number of clusters using the Elbow Method and Silhouette Analysis, which helped us to choose the most appropriate cluster number for your data. Finally, we applied the K-Means algorithm with the chosen number of clusters and visualized the resulting customer segments using PCA for dimensionality reduction. This analysis is crucial for identifying distinct customer groups, understanding their characteristics, and tailoring strategies to address their specific needs and preferences.

In [172…

```
# Encoding Gender
df['Gender_Encoded'] = df['Gender'].map({'Male': 1, 'Female': 0})  # Update mapping as
df[['Purchase_Frequency_Per_Month', 'Gender_Encoded']] = df[['Purchase_Frequency_Per_M

# Features to cluster: 'Purchase_Frequency_Per_Month', 'Gender_Encoded'
features_to_cluster3 = df[['Purchase_Frequency_Per_Month', 'Gender_Encoded']]
scaler3 = StandardScaler()
scaled_features3 = scaler3.fit_transform(features_to_cluster3)

# Elbow Method
wcss3 = []
for i in range(1, 11):
    kmeans3 = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    kmeans3.fit(scaled_features3)
    wcss3.append(kmeans3.inertia_)

plt.plot(range(1, 11), wcss3, marker='o')
plt.title('Elbow Method for Purchase Frequency and Gender')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# Apply K-Means
optimal_k3 = 4
kmeans3 = KMeans(n_clusters=optimal_k3, n_init=10, random_state=42)
df['Cluster3'] = kmeans3.fit_predict(scaled_features3)

# Scatter plot
plt.scatter(df['Purchase_Frequency_Per_Month'], df['Gender_Encoded'], c=df['Cluster3']
plt.xlabel('Purchase Frequency Per Month')
plt.ylabel('Gender Encoded')
plt.show()
```

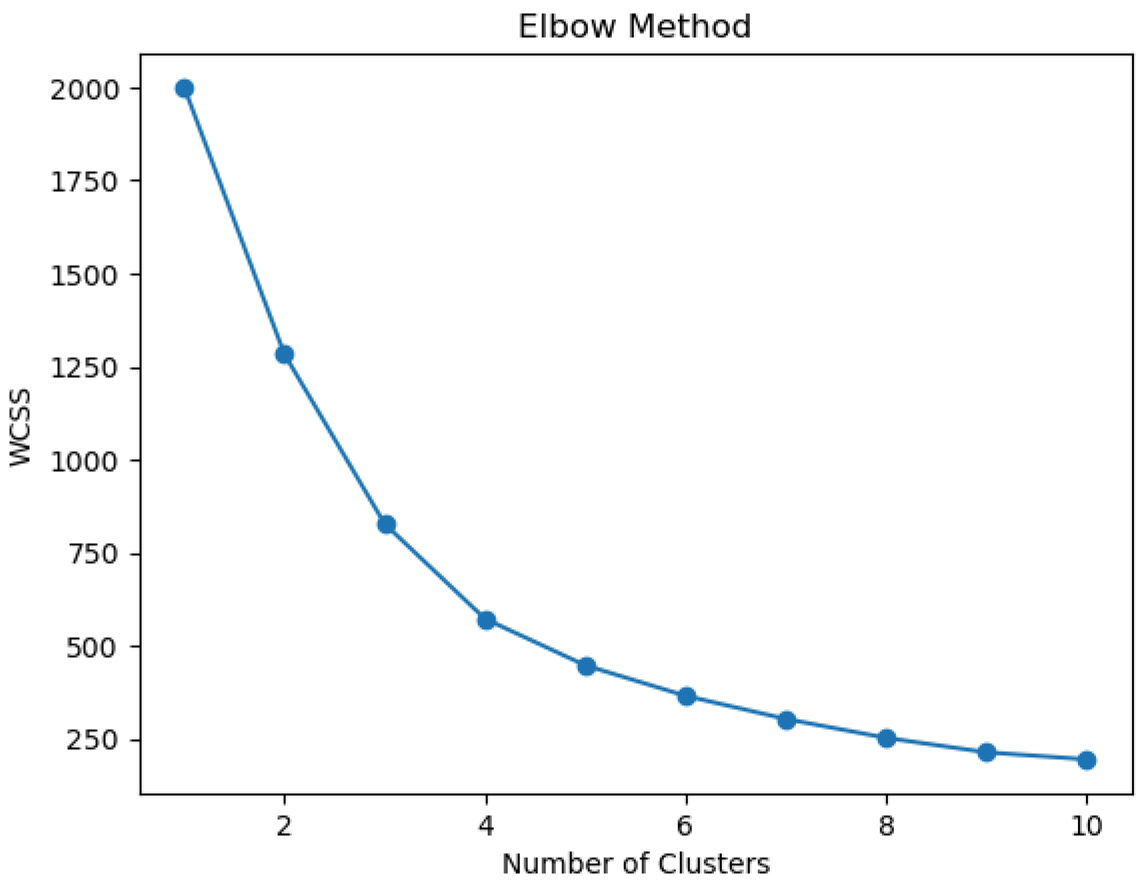


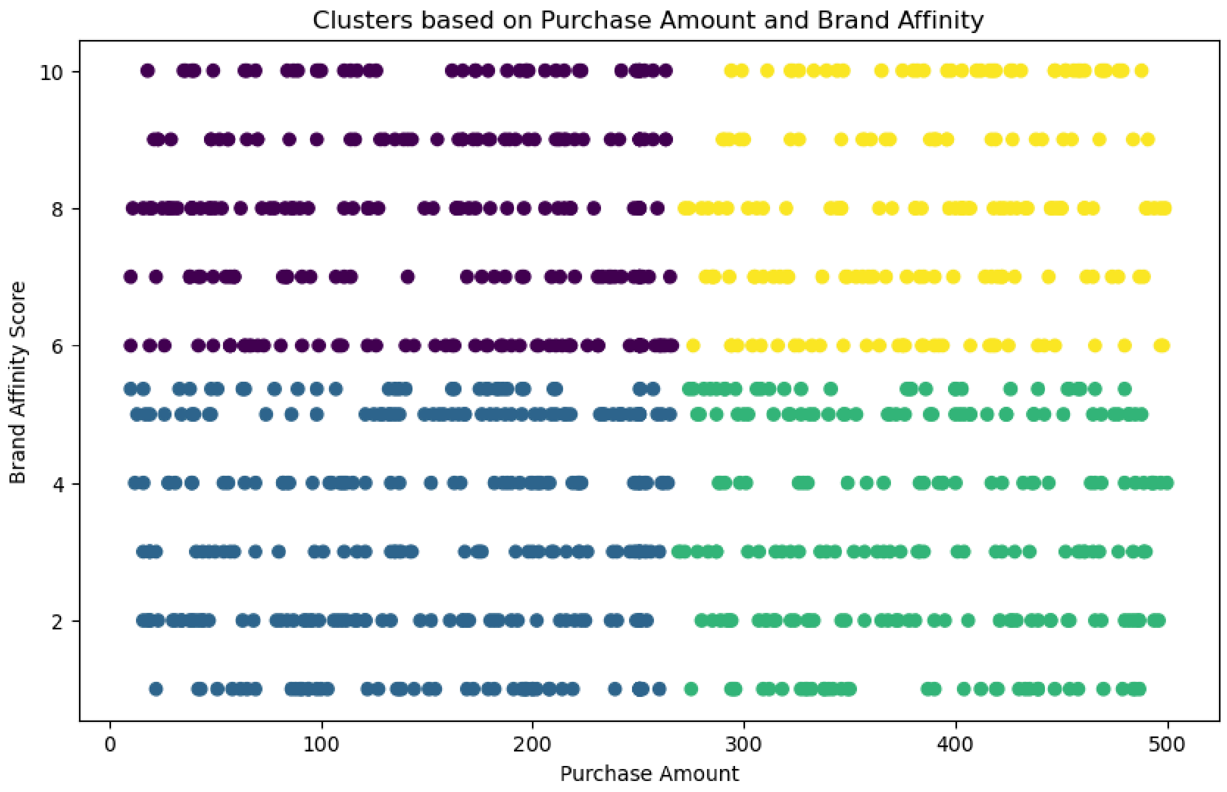


In [173…

```
#---------------------------------------#


features_to_cluster1 = df[['Purchase_Amount', 'Brand_Affinity_Score']]
```

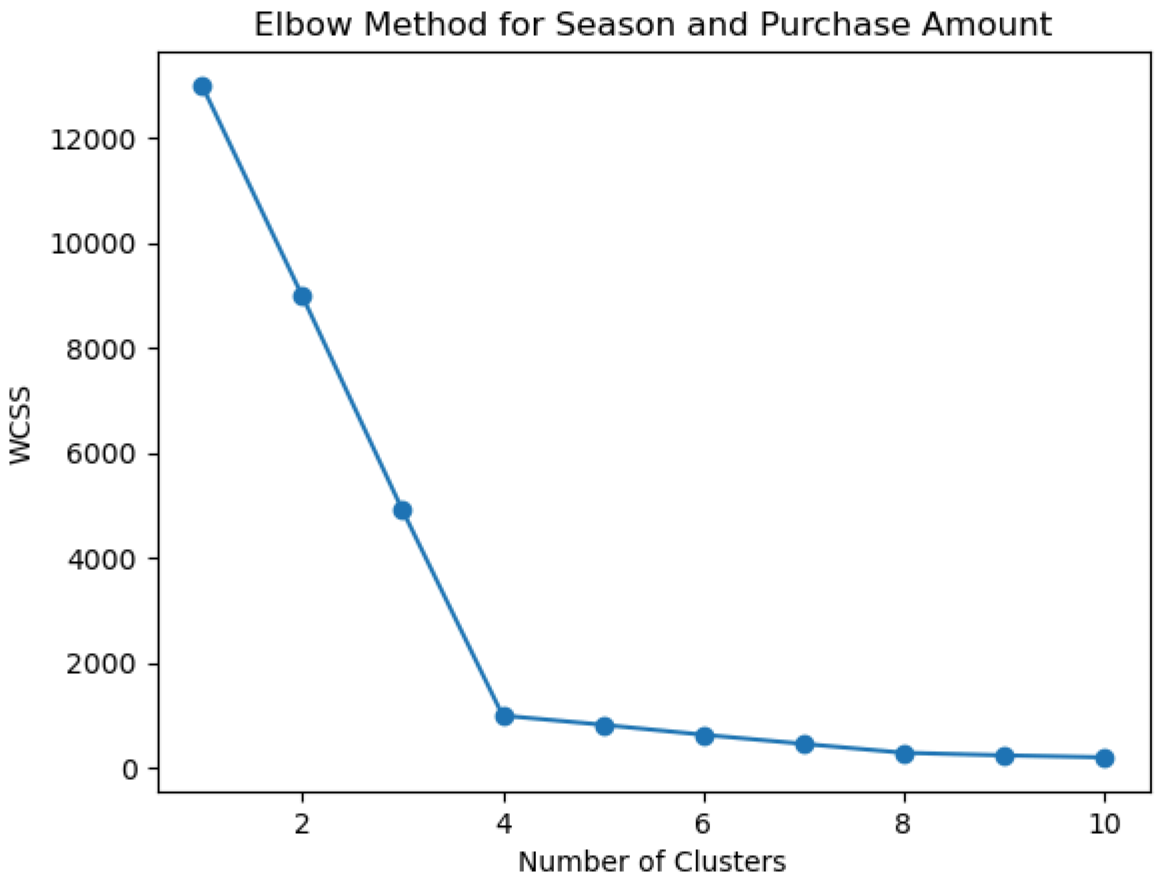
```
scaler1 = StandardScaler()
scaled_features1 = scaler1.fit_transform(features_to_cluster1)

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    kmeans.fit(scaled_features1)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# Apply K-Means
optimal_k = 4  # Assuming 6 is optimal from your analysis
kmeans = KMeans(n_clusters=optimal_k,n_init=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_features1)

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['Purchase_Amount'], df['Brand_Affinity_Score'], c=df['Cluster'], cmap='
plt.title('Clusters based on Purchase Amount and Brand Affinity')
plt.xlabel('Purchase Amount')
plt.ylabel('Brand Affinity Score')
plt.show()
```

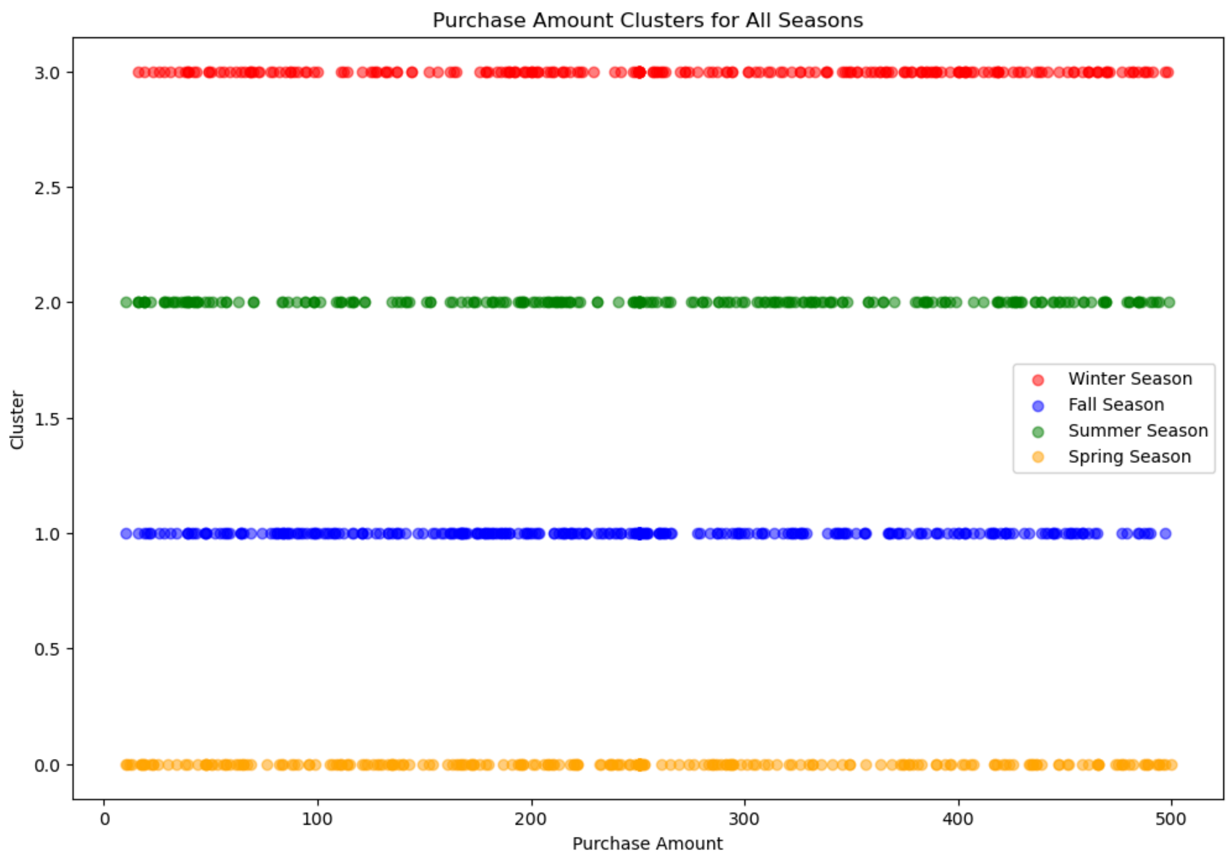




In [174…

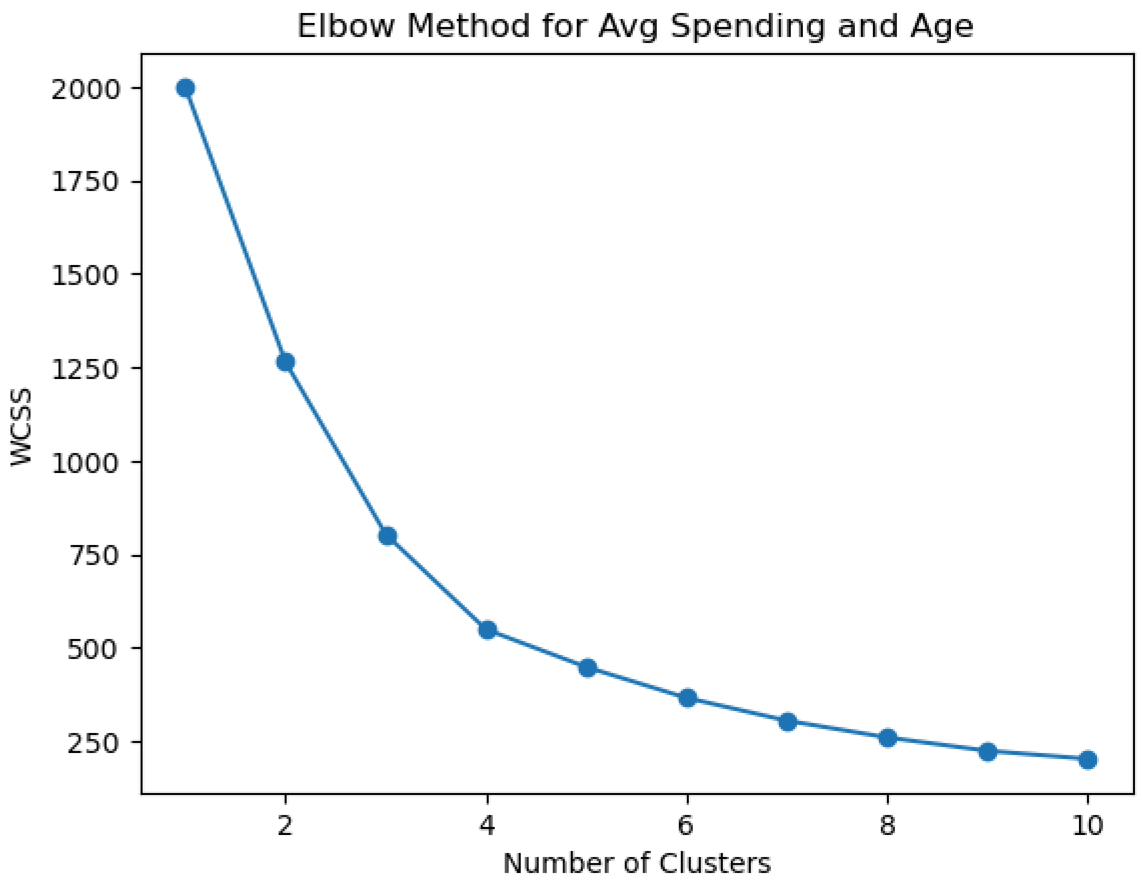
```
# Encoding 'Season'
season_encoded = pd.get_dummies(df['Season'], prefix='Season')
df = pd.concat([df, season_encoded], axis=1)

# Features to cluster: Encoded 'Season' and 'Purchase_Amount'
features_to_cluster = pd.concat([df[season_encoded.columns], df['Purchase_Amount']], a
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features_to_cluster)

# Elbow Method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    kmeans.fit(scaled_features)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method for Season and Purchase Amount')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()


optimal_k = 4  # Adjust based on your analysis
kmeans = KMeans(n_clusters=optimal_k,n_init=10, random_state=42)
df['Cluster_Season_Purchase'] = kmeans.fit_predict(scaled_features)

# Plotting a scatter plot for all seasons
plt.figure(figsize=(12, 8))
seasons = df['Season'].unique()
colors = ['red', 'blue', 'green', 'orange']  # Adjust the colors as needed
for i, season in enumerate(seasons):
    season_cluster_data = df[df['Season'] == season]
    plt.scatter(season_cluster_data['Purchase_Amount'], season_cluster_data['Cluster_S
                color=colors[i], label=f'{season} Season', alpha=0.5)

plt.title('Purchase Amount Clusters for All Seasons')
plt.xlabel('Purchase Amount')
plt.ylabel('Cluster')
plt.legend()
plt.show()
```

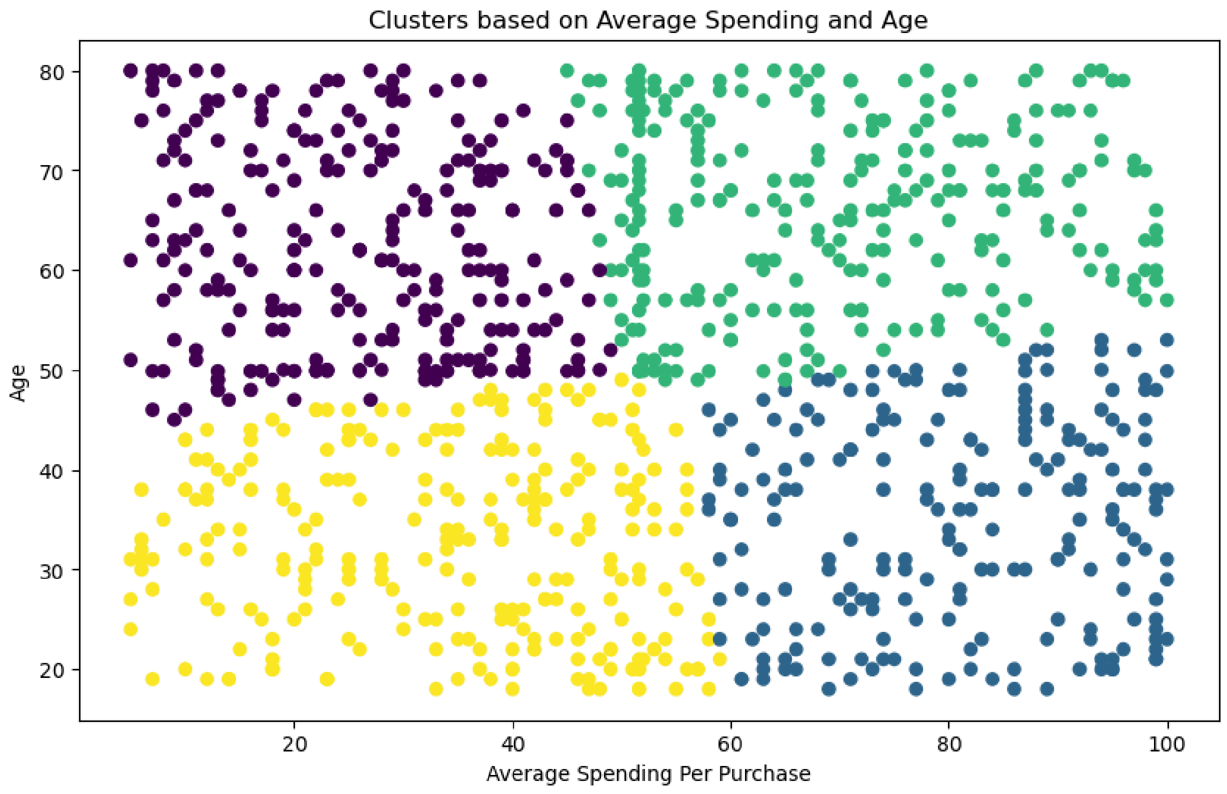




In [176…

```
# Features to cluster: 'Average_Spending_Per_Purchase', 'Age'
features_to_cluster2 = df[['Average_Spending_Per_Purchase', 'Age']]
scaler2 = StandardScaler()
scaled_features2 = scaler2.fit_transform(features_to_cluster2)

# Elbow Method
wcss2 = []
for i in range(1, 11):
    kmeans2 = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    kmeans2.fit(scaled_features2)
    wcss2.append(kmeans2.inertia_)

plt.plot(range(1, 11), wcss2, marker='o')
plt.title('Elbow Method for Avg Spending and Age')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# Apply K-Means
optimal_k2 = 4
kmeans2 = KMeans(n_clusters=optimal_k2,n_init=10, random_state=42)
df['Cluster2'] = kmeans2.fit_predict(scaled_features2)

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['Average_Spending_Per_Purchase'], df['Age'], c=df['Cluster2'], cmap='vi
plt.title('Clusters based on Average Spending and Age')
plt.xlabel('Average Spending Per Purchase')
plt.ylabel('Age')
plt.show()
```

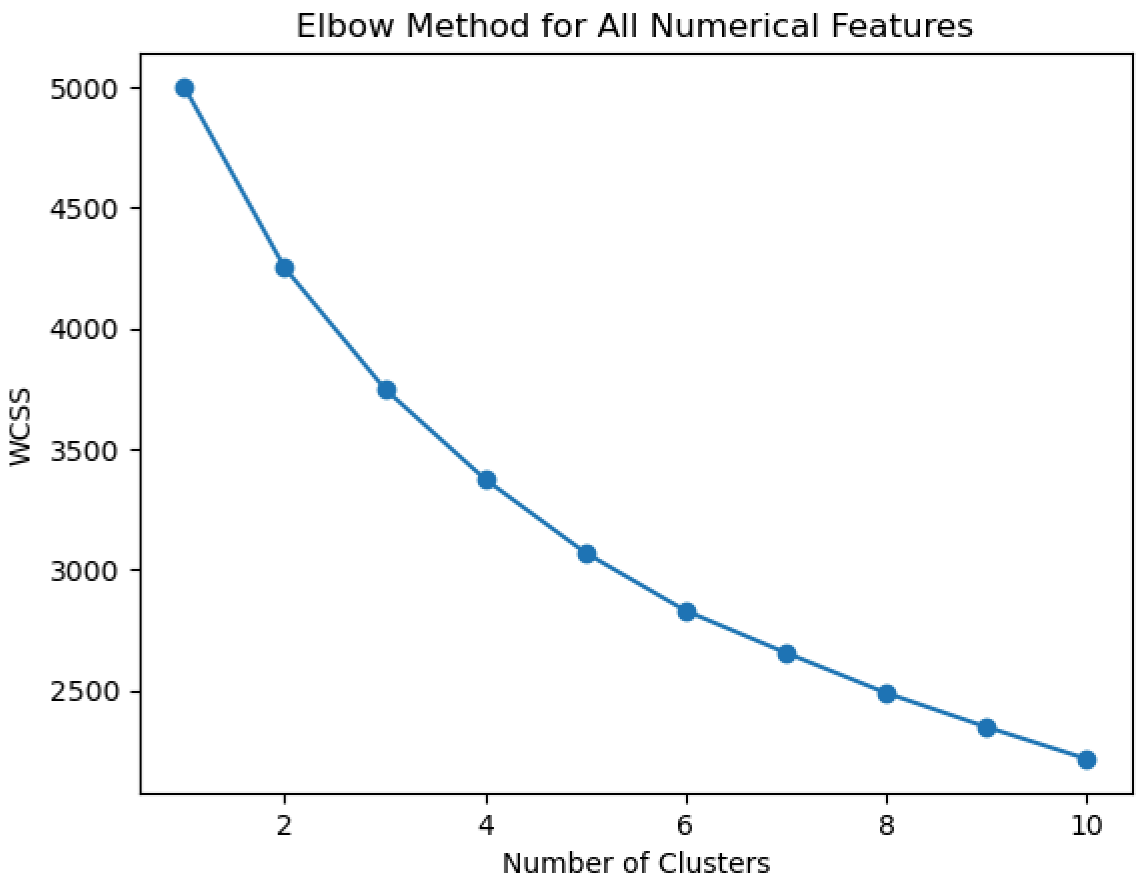


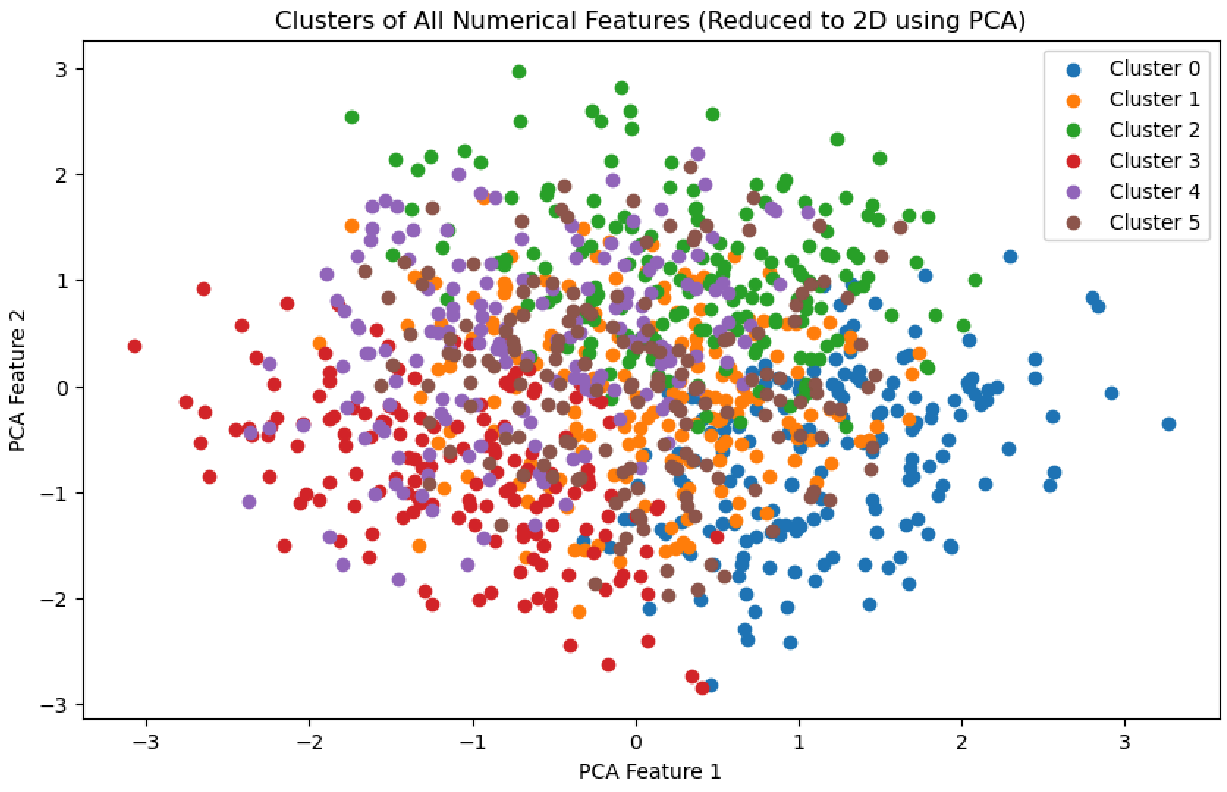


In [175…

```
# Assuming 'numerical_columns_specific' contains the list of numerical columns you wan
numerical_columns_specific = ['Age', 'Purchase_Amount', 'Average_Spending_Per_Purchase
                              'Purchase_Frequency_Per_Month', 'Brand_Affinity_Score']

# Standardize the features
# Standardize all numerical features
```

```
scaler_all = StandardScaler()
scaled_features_all = scaler_all.fit_transform(df[numerical_columns_specific])

# Elbow Method to determine the optimal number of clusters
wcss_all = []
for i in range(1, 11):
    kmeans_all = KMeans(n_clusters=i, init='random', n_init=10, random_state=42)
    kmeans_all.fit(scaled_features_all)
    wcss_all.append(kmeans_all.inertia_)

plt.plot(range(1, 11), wcss_all, marker='o')
plt.title('Elbow Method for All Numerical Features')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# Assuming the optimal number of clusters is found to be 4
optimal_k_all = 6
kmeans_all = KMeans(n_clusters=optimal_k_all, n_init=10, random_state=42)
df['All_Numerical_Cluster'] = kmeans_all.fit_predict(scaled_features_all)

# Using PCA for visualization
pca_all = PCA(n_components=2)
reduced_data_all = pca_all.fit_transform(scaled_features_all)

plt.figure(figsize=(10, 6))
for i in range(optimal_k_all):
    plt.scatter(reduced_data_all[df['All_Numerical_Cluster'] == i, 0], reduced_data_al
plt.title('Clusters of All Numerical Features (Reduced to 2D using PCA)')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.legend()
plt.show()
```





Kmeans ++

In [167…
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features_to_scale])
```

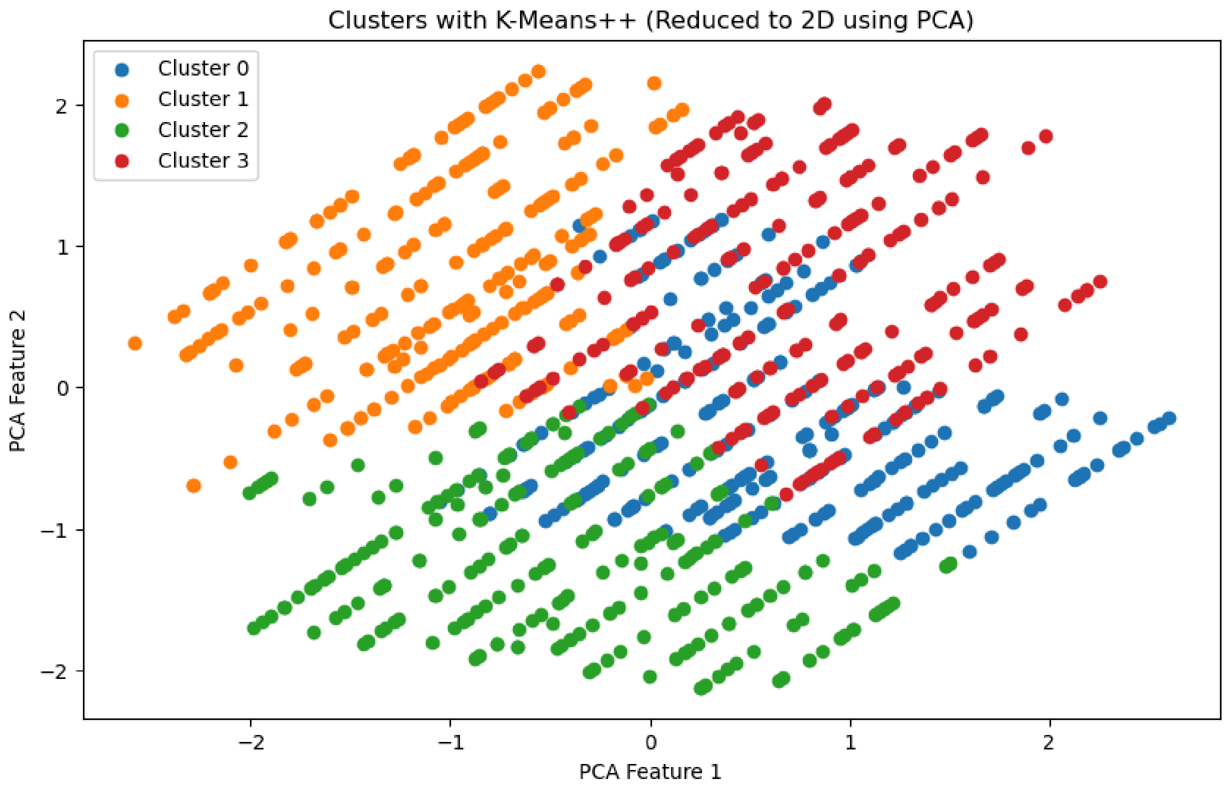
```
scaled_features

desired_k = 4  # Example value, adjust as needed

# Implement K-Means++ with the desired number of clusters
kmeans_plus = KMeans(n_clusters=desired_k, init='k-means++', n_init=10, random_state=4
df['Cluster_KMeans'] = kmeans_plus.fit_predict(scaled_features)
Cluster_KMeans = df.groupby('Cluster_KMeans')[features_to_scale].mean()
Cluster_KMeans
```

Out[167]:

	Average_Spending_Per_Purchase	Brand_Affinity_Score	Product_Category_Preferences_Va
Cluster_KMeans			
0	31.327455	5.184216	0.100
1	55.922149	2.913475	0.828
2	81.493450	5.656919	0.181
3	42.645157	7.985224	0.811

In [168…

```
# Reduce the dimensions for visualization (if necessary)
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_features)

# Scatter plot of the clusters
plt.figure(figsize=(10, 6))
for i in range(desired_k):
    plt.scatter(reduced_data[df['Cluster_KMeans'] == i, 0], reduced_data[df['Cluster_K
plt.title('Clusters with K-Means++ (Reduced to 2D using PCA)')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.legend()
plt.show()
```



In [169…

```
kmeans_regular = KMeans(n_clusters=desired_k, init='random', n_init=10, max_iter=300,
df['Cluster_KMeans'] = kmeans_regular.fit_predict(scaled_features)

# K-Means++
kmeans_plus = KMeans(n_clusters=desired_k, init='k-means++', n_init=10, max_iter=300,
df['Cluster_KMeans++'] = kmeans_plus.fit_predict(scaled_features)



silhouette_regular = silhouette_score(scaled_features, df['Cluster_KMeans'])
# Silhouette Score for K-Means++
silhouette_plus = silhouette_score(scaled_features, df['Cluster_KMeans++'])

iterations_regular = kmeans_regular.n_iter_
# Number of iterations for K-Means++
iterations_plus = kmeans_plus.n_iter_

print(f"Number of iterations for Regular K-Means: {iterations_regular}")
print(f"Number of iterations for K-Means++: {iterations_plus}")

print(f"Silhouette Score for Regular K-Means: {silhouette_regular}")
print(f"Silhouette Score for K-Means++: {silhouette_plus}")
```

```
Number of iterations for Regular K-Means: 18
Number of iterations for K-Means++: 32
Silhouette Score for Regular K-Means: 0.2725464503944895
Silhouette Score for K-Means++: 0.27263006605502776
```

Based on the data you provided:

K-Means++ shows a very slight improvement in cluster quality over Regular K-Means, as evidenced by the marginally higher silhouette score. However, it required more iterations to converge (32 for K-Means++ vs. 18 for Regular K-Means). This indicates that while K-Means++ may offer a slight edge in defining more cohesive clusters, it does so at the cost of additional computational effort. In scenarios involving large and complex datasets, this minor improvement in cluster quality can be significant, making K-Means++ a preferable choice despite its slightly longer convergence time.

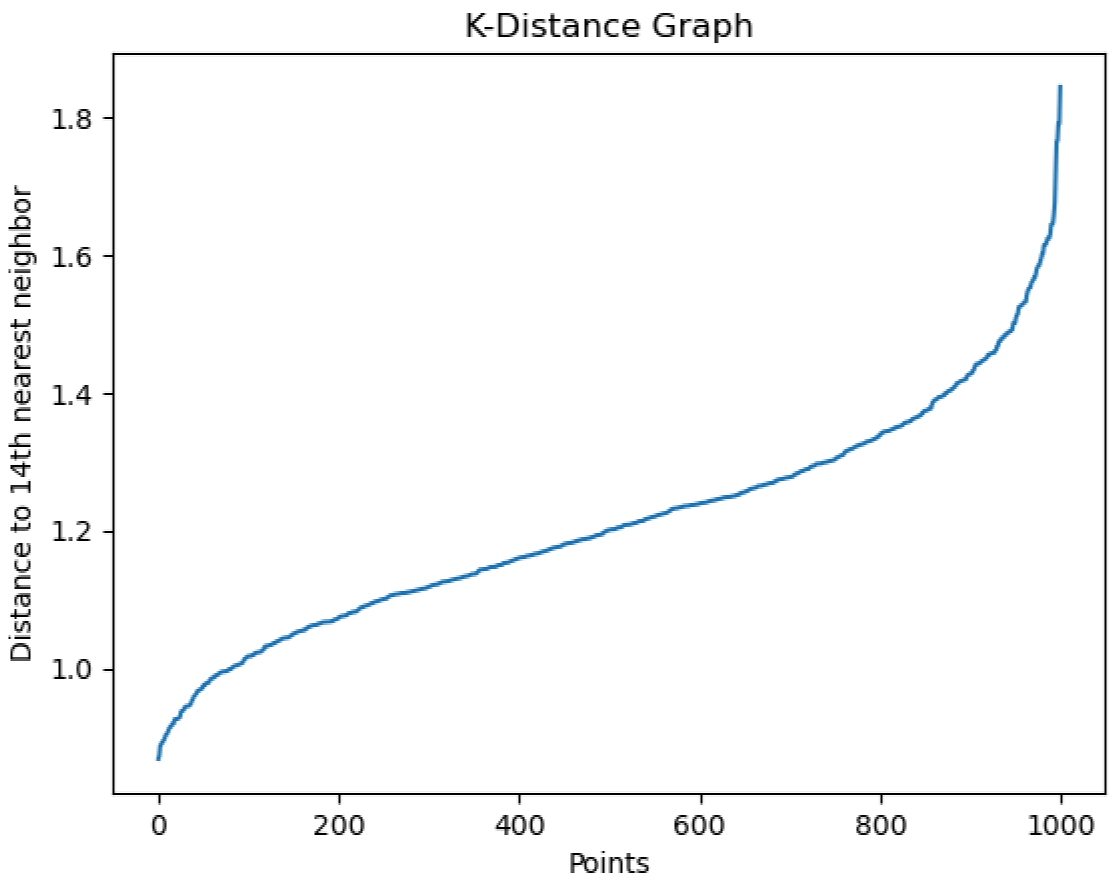
Summary

In Module 3, Part 3, we implemented and analyzed K-Means++ clustering to segment customers based on selected features. we standard scaling to preprocess the data, ensuring effective clustering. The application of K-Means++ with a specified cluster number and the subsequent dimensionality reduction using PCA allowed for a clear visualization and understanding of the clusters. Additionally, we compared the performance of K-Means++ with regular K-Means using silhouette scores and iteration counts, demonstrating the efficiency of K-Means++ in forming more distinct and well-separated clusters.

DB Scan

In [116…

```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[numerical_columns_specific])

# Use NearestNeighbors to find the average distance to the kth nearest neighbor
min_pts = 14
nearest_neighbors = NearestNeighbors(n_neighbors=min_pts)
neighbors = nearest_neighbors.fit(scaled_features)
distances, indices = neighbors.kneighbors(scaled_features)

# Sort distance values and plot
sorted_distances = np.sort(distances[:, min_pts - 1])
plt.plot(sorted_distances)
plt.xlabel("Points")
plt.ylabel("Distance to " + str(min_pts) + "th nearest neighbor")
plt.title("K-Distance Graph")
plt.show()
```



In [119…
```
# Example values for experimentation
eps_values = [1.3, 1.4, 1.5]  # Adjust these based on your data
min_samples_values = [14, 15, 16]

# Store the number of clusters for each configuration
configurations = {}

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        clusters = dbscan.fit_predict(scaled_features)
        num_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)  # -1 is for
        configurations[(eps, min_samples)] = num_clusters

# Print the number of clusters for each configuration
for config, num_clusters in configurations.items():
    print(f"eps: {config[0]}, MinPts: {config[1]}, Clusters: {num_clusters}")
```

```
eps: 1.3, MinPts: 14, Clusters: 1
eps: 1.3, MinPts: 15, Clusters: 1
eps: 1.3, MinPts: 16, Clusters: 1
eps: 1.4, MinPts: 14, Clusters: 1
eps: 1.4, MinPts: 15, Clusters: 1
eps: 1.4, MinPts: 16, Clusters: 1
eps: 1.5, MinPts: 14, Clusters: 1
eps: 1.5, MinPts: 15, Clusters: 1
eps: 1.5, MinPts: 16, Clusters: 1
```

In [120…
```
silhouette_scores = {}

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        clusters = dbscan.fit_predict(scaled_features)

        # We calculate silhouette score only when there are 2 or more clusters and les
        if len(set(clusters)) >= 2 and len(set(clusters)) < len(scaled_features) - 1:
            score = silhouette_score(scaled_features, clusters)
            silhouette_scores[(eps, min_samples)] = score
        else:
            silhouette_scores[(eps, min_samples)] = None

# Print the silhouette scores for each configuration
for config, score in silhouette_scores.items():
    print(f"eps: {config[0]}, MinPts: {config[1]}, Silhouette Score: {score}")
```

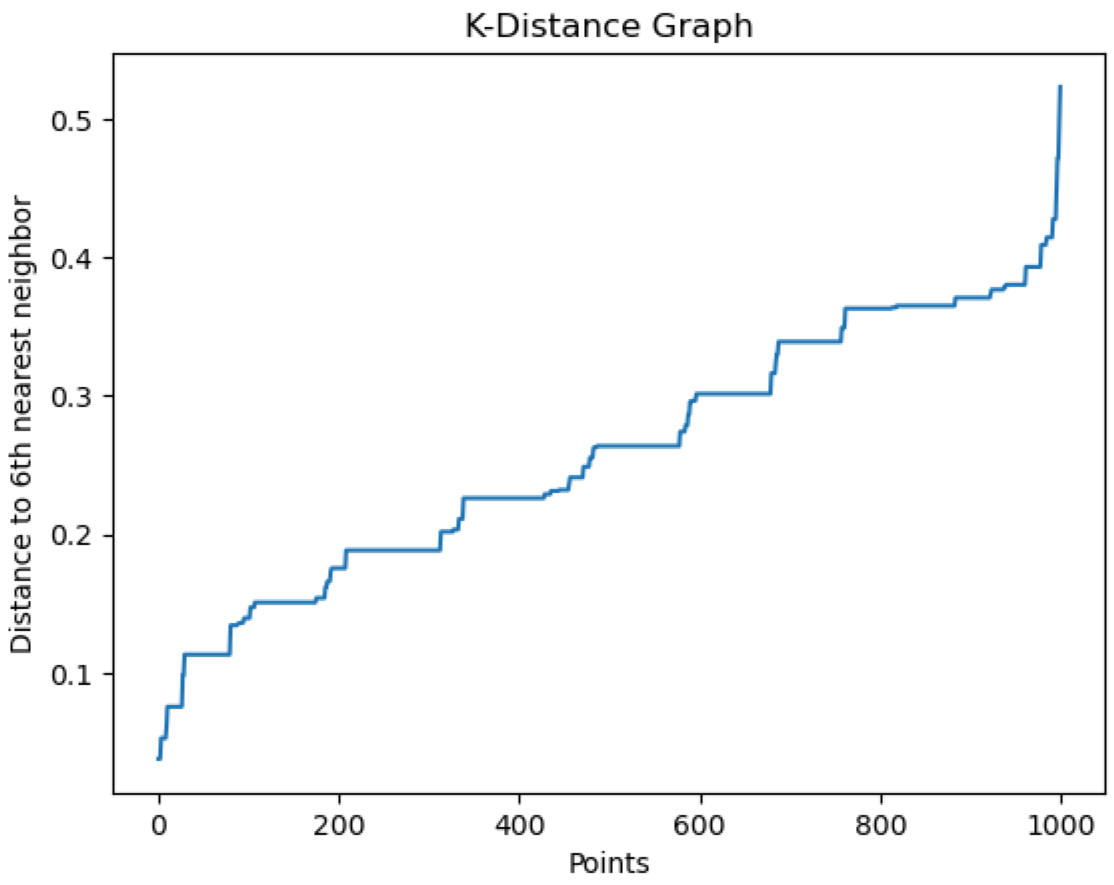
```
eps: 1.3, MinPts: 14, Silhouette Score: 0.14473131400170478
eps: 1.3, MinPts: 15, Silhouette Score: 0.14808810290945473
eps: 1.3, MinPts: 16, Silhouette Score: 0.1539879770633721
eps: 1.4, MinPts: 14, Silhouette Score: 0.1788875763465445
eps: 1.4, MinPts: 15, Silhouette Score: 0.17914804775228657
eps: 1.4, MinPts: 16, Silhouette Score: 0.17914804775228657
eps: 1.5, MinPts: 14, Silhouette Score: None
eps: 1.5, MinPts: 15, Silhouette Score: None
eps: 1.5, MinPts: 16, Silhouette Score: None
```

In [ ]:

Actual DBSCAN

In [170…

```
features_to_scale = ['Average_Spending_Per_Purchase', 'Brand_Affinity_Score', 'Product
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features_to_scale])
# Use NearestNeighbors to find the average distance to the kth nearest neighbor
min_pts = 6
nearest_neighbors = NearestNeighbors(n_neighbors=min_pts)
neighbors = nearest_neighbors.fit(scaled_features)
distances, indices = neighbors.kneighbors(scaled_features)

# Sort distance values and plot
sorted_distances = np.sort(distances[:, min_pts - 1])
plt.plot(sorted_distances)
plt.xlabel("Points")
plt.ylabel("Distance to " + str(min_pts) + "th nearest neighbor")
plt.title("K-Distance Graph")
plt.show()
```



In [159…

```
# Example ranges for experimenting - these ranges might need to be adjusted
eps_values = [0.3,0.4, 0.5]
min_samples_values = [6,7,8]
# Store the number of clusters for each configuration
configurations = {}

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        clusters = dbscan.fit_predict(scaled_features)
        num_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)  # -1 is for
        configurations[(eps, min_samples)] = num_clusters

# Print the number of clusters for each configuration
for config, num_clusters in configurations.items():
    print(f"eps: {config[0]}, MinPts: {config[1]}, Clusters: {num_clusters}")
```

```
eps: 0.3, MinPts: 6, Clusters: 55
eps: 0.3, MinPts: 7, Clusters: 44
eps: 0.3, MinPts: 8, Clusters: 30
eps: 0.4, MinPts: 6, Clusters: 3
eps: 0.4, MinPts: 7, Clusters: 4
eps: 0.4, MinPts: 8, Clusters: 3
eps: 0.5, MinPts: 6, Clusters: 3
eps: 0.5, MinPts: 7, Clusters: 3
eps: 0.5, MinPts: 8, Clusters: 3
```

In [160…

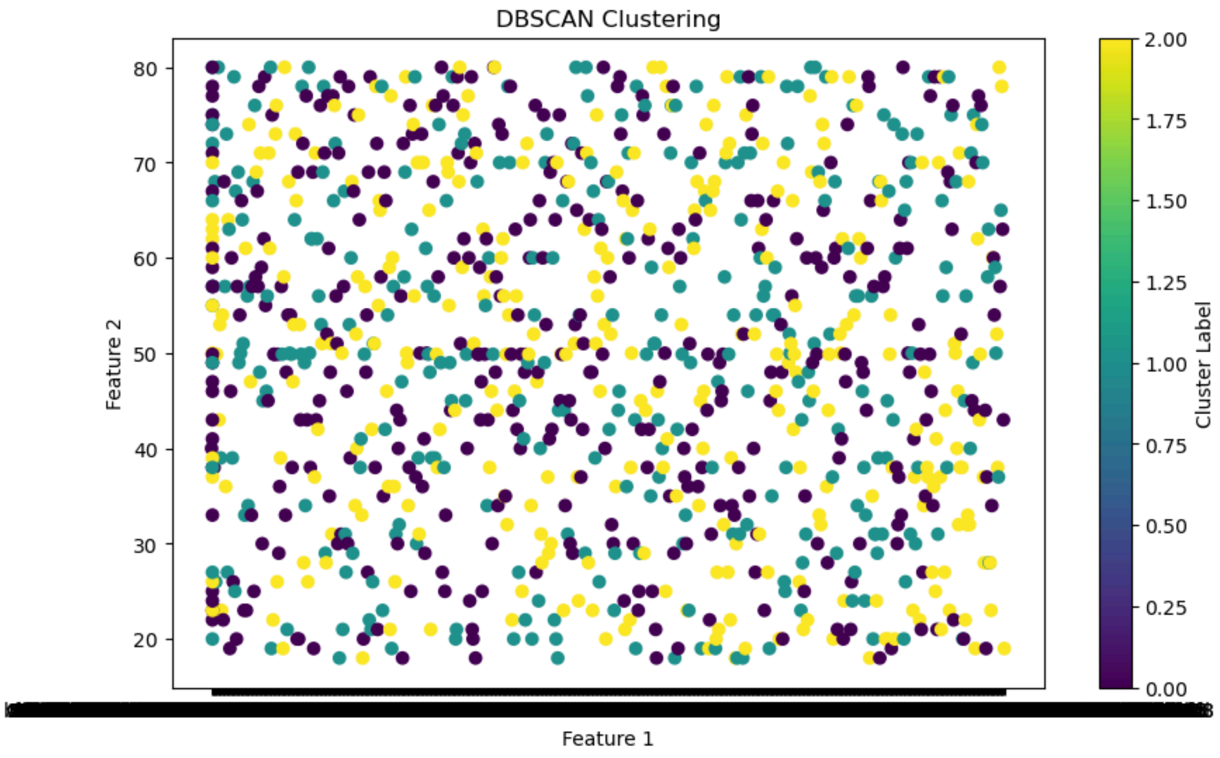
```
for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(scaled_features)
        # Silhouette score can only be computed if more than one cluster is formed, ex
        if len(set(labels)) > 1:
            silhouette_avg = silhouette_score(scaled_features, labels)
            print(f"DBSCAN eps={eps}, min_samples={min_samples}, Silhouette Score: {si
        else:
            print(f"DBSCAN eps={eps}, min_samples={min_samples}, less than 2 clusters
```

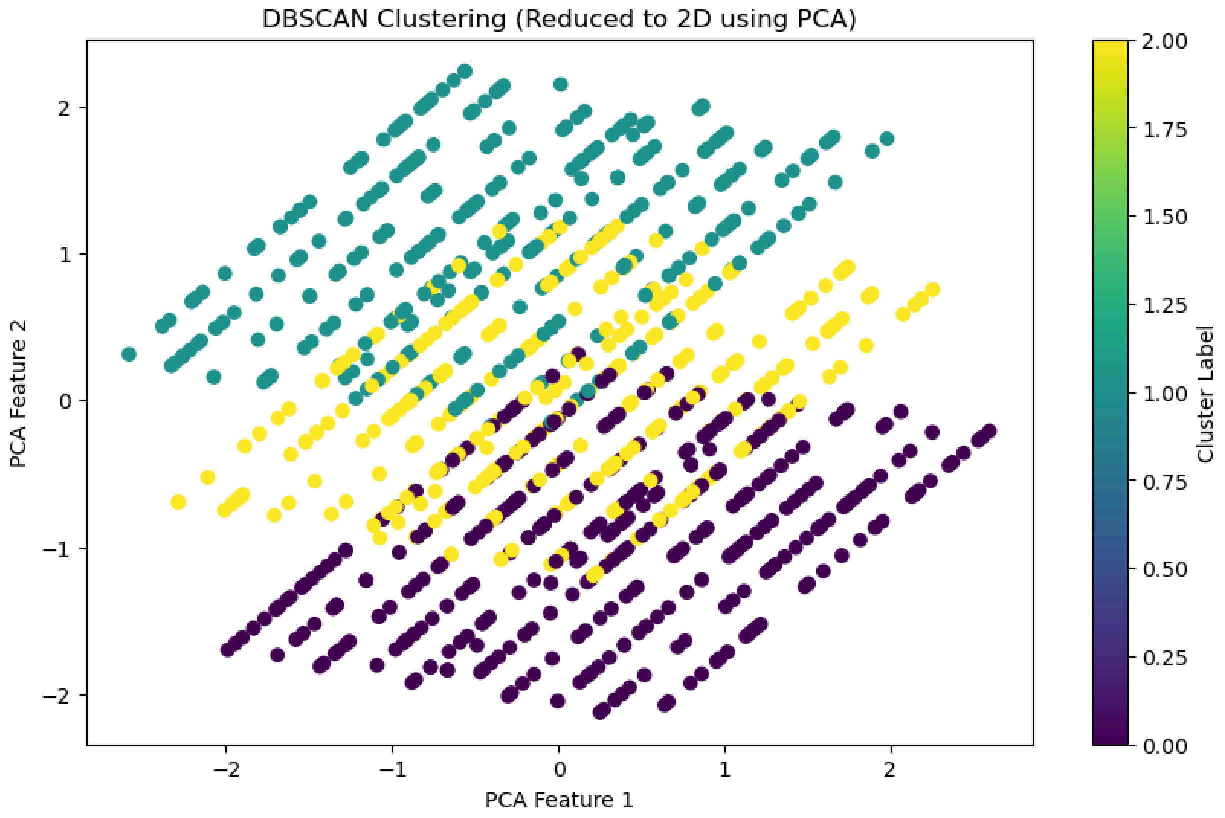
```
DBSCAN eps=0.3, min_samples=6, Silhouette Score: -0.0043917927149448276
DBSCAN eps=0.3, min_samples=7, Silhouette Score: -0.018511038618861737
DBSCAN eps=0.3, min_samples=8, Silhouette Score: -0.1186164066024163
DBSCAN eps=0.4, min_samples=6, Silhouette Score: 0.1928901556852146
DBSCAN eps=0.4, min_samples=7, Silhouette Score: 0.08609125347362846
DBSCAN eps=0.4, min_samples=8, Silhouette Score: 0.13644012415344964
DBSCAN eps=0.5, min_samples=6, Silhouette Score: 0.1928901556852146
DBSCAN eps=0.5, min_samples=7, Silhouette Score: 0.1928901556852146
DBSCAN eps=0.5, min_samples=8, Silhouette Score: 0.1928901556852146
```

In [161…

```
dbscan = DBSCAN(eps=0.4, min_samples=6)
df['DBSCAN_Cluster'] = dbscan.fit_predict(scaled_features)

# Count the number of points in each cluster including noise (-1)
cluster_distribution = df['DBSCAN_Cluster'].value_counts()
print("Cluster Distribution:\n", cluster_distribution)



plt.figure(figsize=(10, 6))
plt.scatter(df.iloc[:, 0], df.iloc[:, 1], c=df['DBSCAN_Cluster'], cmap='viridis', labe
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')  # Adjust based on your dataset
plt.ylabel('Feature 2')  # Adjust based on your dataset
plt.colorbar(label='Cluster Label')
plt.show()





dbscan = DBSCAN(eps=0.4, min_samples=6)
df['DBSCAN_Cluster'] = dbscan.fit_predict(scaled_features)
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_features)

plt.figure(figsize=(10, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=df['DBSCAN_Cluster'], cmap='viri
plt.title('DBSCAN Clustering (Reduced to 2D using PCA)')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.colorbar(label='Cluster Label')
plt.show()
```

```
Cluster Distribution:
 DBSCAN_Cluster
0    377
2    314
1    309
Name: count, dtype: int64
```





Summary

In Module 3, Part 3, we implemented DBSCAN clustering to segment the customer data. We began by scaling the features and then used the K-Distance Graph to determine optimal eps and min_samples parameters for DBSCAN. After experimenting with different parameter combinations and evaluating the cluster quality using silhouette scores, you applied DBSCAN with the selected parameters. Finally, you visualized the clusters in 2D space, both in the original feature space and reduced PCA space.

Module 4

Compare the Results of All Three Clustering Algorithms

Based on the values for the silhouette scores and the number of iterations for each clustering algorithm (DBSCAN, Regular K-Means, and K-Means++) here is the comparison:

Silhouette Scores:

- **DBSCAN:** Silhouette Score = 0.1929
- **Regular K-Means:** Silhouette Score = 0.2725
- **K-Means++:** Silhouette Score = 0.2726

Number of Iterations:

- **Regular K-Means:** 18 iterations
- **K-Means++:** 32 iterations

Analysis and Comparison:

- **Silhouette Scores:** Both versions of K-Means show better cluster definition than DBSCAN, as indicated by their higher silhouette scores. K-Means++ offers a marginally better score than Regular K-Means, suggesting slightly more distinct clustering.
- **Convergence Speed:** K-Means++ required more iterations compared to Regular K-Means. This suggests a trade-off between cluster quality and computational effort in K-Means++.
- **DBSCAN Performance:** The lower silhouette score for DBSCAN reflects its nature as a density-based clustering algorithm, which might form clusters differently than centroid-based methods.

Conclusions:

- **K-Means++:** Slightly more effective than Regular K-Means in terms of silhouette score but requires more iterations.
- **Regular K-Means:** Converges faster, making it a more efficient option where computation time is a concern.

- **DBSCAN:** Robust choice for datasets with non-standard cluster shapes or when identifying outliers is important, despite its lower silhouette score.

Additional Considerations:

- The choice of clustering algorithm should consider the nature of dataset and the specific objectives of analysis. DBSCAN might be more suitable for detecting outliers or handling non-spherical clusters, despite its lower silhouette score.

Draw conclusions and recommendations

Conclusions and Recommendations

Customer Segmentation and Purchasing Patterns:

- Age Group 41-50: Key customer segment with most purchases. Strategies should focus on this demographic.
- Age Group 21-30: Highest average spending per purchase. Target for high-value transaction campaigns.
- Income Levels: High income correlates with higher purchasing. Tailor strategies for different income levels.
- Seasonal Variations: Winter shows highest average purchase amount. Plan season-specific promotions.

Temporal Trends:

- December 2021 vs. December 2023: Decline in average purchases over time needs investigation.
- Purchase Frequency: High in August, low in November. Adjust marketing and stock accordingly.

Brand Preference:

- Brand C vs. Brand A: Utilize Brand C's popularity in marketing, investigate and address Brand A's underperformance.

Cluster Analysis:

- K-Means and K-Means++: Identified 4 customer clusters.
- DBSCAN: Identified 3 clusters. Utilize these insights for targeted marketing.

Data-Driven Strategies

Targeted Marketing and Promotions:

- Focus on the most active customer segments with tailored campaigns.
- Season-specific promotions, especially in high-spending periods.

Focus on Underperforming Segments and Metrics

- **81-90 Age Group:** Investigate the reasons behind minimal engagement in this age group and explore strategies to increase their participation.
- **Fall Season Spending:** Analyze the factors contributing to lower spending in the fall season and develop targeted campaigns to boost sales during this period.
- **Brand A's Performance:** Conduct a deep dive into why Brand A is underperforming in terms of transaction count and consider strategies to enhance its market presence and appeal.