Project: Customer Segmentation using K-Means Clustering

Done By : *Mr. Aridweep Majumder, M.Tech*

This project applies K-Means clustering to a marketing dataset to segment customers based on demographics and purchase behavior.

Dataset source: *marketing_campaign.csv*

Tools used: *Python, Pandas, Matplotlib, Seaborn, Scikit-learn*

1. Importing Necessary Libraries & Load Datasets

```
# Basic imports for analysis and modeling
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Enable inline plots
%matplotlib inline

# Load the dataset (already uploaded to Colab)
df = pd.read_csv('/content/marketing_campaign.csv')

# Check shape and preview
print(f"Total Rows: {df.shape[0]}")
print(f"Total Columns: {df.shape[1]}")
df.head(3)
```

```
Total Rows: 2240
Total Columns: 29
```

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26

3 rows × 29 columns

2. Data Overview & Summary

```
# Quick look at column types and names
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Year_Birth           2240 non-null   int64
 2   Education            2240 non-null   object
 3   Marital_Status       2240 non-null   object
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64
 6   Teenhome             2240 non-null   int64
 7   Dt_Customer          2240 non-null   object
 8   Recency              2240 non-null   int64
 9   MntWines             2240 non-null   int64
 10  MntFruits            2240 non-null   int64
 11  MntMeatProducts      2240 non-null   int64
 12  MntFishProducts      2240 non-null   int64
 13  MntSweetProducts     2240 non-null   int64
 14  MntGoldProds         2240 non-null   int64
 15  NumDealsPurchases    2240 non-null   int64
 16  NumWebPurchases      2240 non-null   int64
 17  NumCatalogPurchases  2240 non-null   int64
 18  NumStorePurchases    2240 non-null   int64
 19  NumWebVisitsMonth    2240 non-null   int64
 20  AcceptedCmp3         2240 non-null   int64
 21  AcceptedCmp4         2240 non-null   int64
 22  AcceptedCmp5         2240 non-null   int64
 23  AcceptedCmp1         2240 non-null   int64
 24  AcceptedCmp2         2240 non-null   int64
 25  Complain             2240 non-null   int64
 26  Z_CostContact        2240 non-null   int64
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

```
# Listing all columns
print("Columns:")
print(df.columns.tolist())
```

```
Columns:
['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome', 'Teenhome', 'Dt_Custo
```

```
# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
ID                      0
Year_Birth              0
Education               0
Marital_Status          0
Income                 24
Kidhome                 0
Teenhome                0
Dt_Customer             0
Recency                 0
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
```

```
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
AcceptedCmp3            0
AcceptedCmp4            0
AcceptedCmp5            0
AcceptedCmp1            0
AcceptedCmp2            0
Complain                0
Z_CostContact           0
Z_Revenue               0
Response                0
dtype: int64
```

```
# Basic statistical summary
df.describe()
```

	ID	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	Mnt
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250	49.109375	303.935714	26.
std	3246.662198	11.984069	25173.076661	0.538398	0.544538	28.962453	336.597393	39.
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000	24.000000	23.750000	1.
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000	49.000000	173.500000	8.
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000	74.000000	504.250000	33.
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.000000	199.

8 rows × 26 columns

3. Data Cleaning and Preprocessing

```
data = df.copy() # Make a copy to avoid modifying the original dataframe
```

```
data.isnull().sum() # Checking again for missing values
```

	0
ID	0
Year_Birth	0
Education	0
Marital_Status	0
Income	24
Kidhome	0
Teenhome	0
Dt_Customer	0
Recency	0
MntWines	0
MntFruits	0
MntMeatProducts	0
MntFishProducts	0
MntSweetProducts	0
MntGoldProds	0
NumDealsPurchases	0
NumWebPurchases	0
NumCatalogPurchases	0
NumStorePurchases	0
NumWebVisitsMonth	0
AcceptedCmp3	0
AcceptedCmp4	0
AcceptedCmp5	0
AcceptedCmp1	0
AcceptedCmp2	0
Complain	0
Z_CostContact	0
Z_Revenue	0
Response	0

dtype: int64

```
# 'Income' has some missing values - dropping those rows
data = data.dropna(subset=['Income'])

# Convert 'Dt_Customer' to datetime format
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'], format='%d-%m-%Y')
```

```
# Create a new feature: year the customer joined
data['Customer_Year'] = data['Dt_Customer'].dt.year


# Create a new feature: year the customer joined
data['Customer_Year'] = data['Dt_Customer'].dt.year


# Drop unused or duplicate columns for clustering
data = data.drop(['ID', 'Dt_Customer'], axis=1)
```

```
/tmp/ipython-input-139-3013036707.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
  data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'], format='%d-%m-%Y')
/tmp/ipython-input-139-3013036707.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
  data['Customer_Year'] = data['Dt_Customer'].dt.year
/tmp/ipython-input-139-3013036707.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
  data['Customer_Year'] = data['Dt_Customer'].dt.year
```

```
# Convert 'Education' and 'Marital_Status' to numeric using one-hot encoding
data = pd.get_dummies(data, columns=['Education', 'Marital_Status'], drop_first=True)

# Convert all boolean columns to integer (so True/False becomes 1/0)
data = data.astype(int)# Final structure check
print(f"Data shape after cleaning: {data.shape}")
data.head(4)
```

```
Data shape after cleaning: (2216, 37)
```

	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishI
0	1957	58138	0	0	58	635	88	546	
1	1954	46344	1	1	38	11	1	6	
2	1965	71613	0	0	26	426	49	127	
3	1984	26646	1	0	26	11	4	20	

4 rows × 37 columns

```
# Final structure check
print(f"Data shape after cleaning: {data.shape}")
data.head()
```

```
Data shape after cleaning: (2216, 37)
```

	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishI
0	1957	58138	0	0	58	635	88	546	
1	1954	46344	1	1	38	11	1	6	
2	1965	71613	0	0	26	426	49	127	
3	1984	26646	1	0	26	11	4	20	
4	1981	58293	1	0	94	173	43	118	

5 rows × 37 columns

4. Feature Selection for Clustering

```
# Selecting numerical features for clustering
features = data.copy()

# Standardizing the data to bring all features to the same scale
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Convert back to DataFrame for easier inspection if needed
scaled_df = pd.DataFrame(scaled_features, columns=features.columns)

scaled_df.head(4)
```

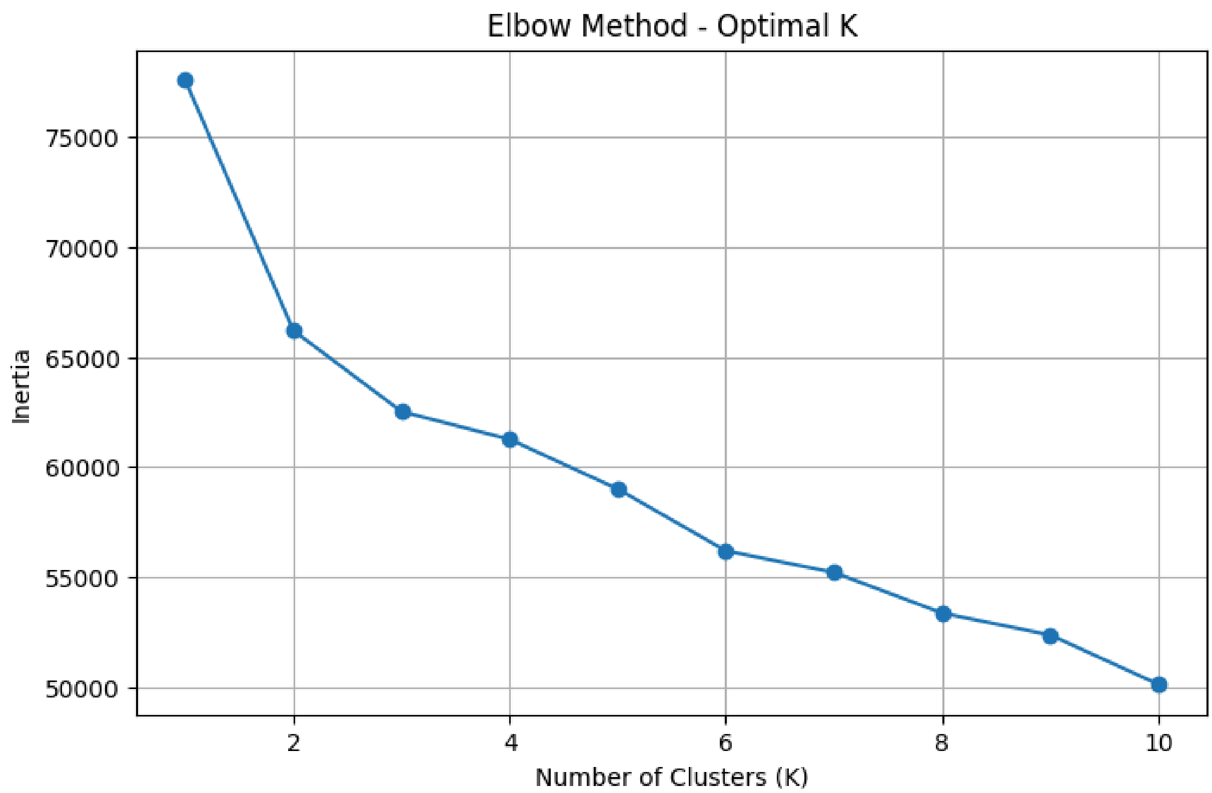
	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFi
0	-0.986443	0.234063	-0.823039	-0.928972	0.310532	0.978226	1.549429	1.690227	
1	-1.236801	-0.234559	1.039938	0.909066	-0.380509	-0.872024	-0.637328	-0.717986	
2	-0.318822	0.769478	-0.823039	-0.928972	-0.795134	0.358511	0.569159	-0.178368	
3	1.266777	-1.017239	1.039938	-0.928972	-0.795134	-0.872024	-0.561922	-0.655551	

4 rows × 37 columns

5. Finding Optimal Clusters (Elbow Method)

```
# Trying different values of K to see where inertia drops sharply
inertia = []
k_range = range(1, 11)

for k in k_range:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(scaled_df)
    inertia.append(model.inertia_)

# Plotting the elbow curve
plt.figure(figsize=(8, 5))
```

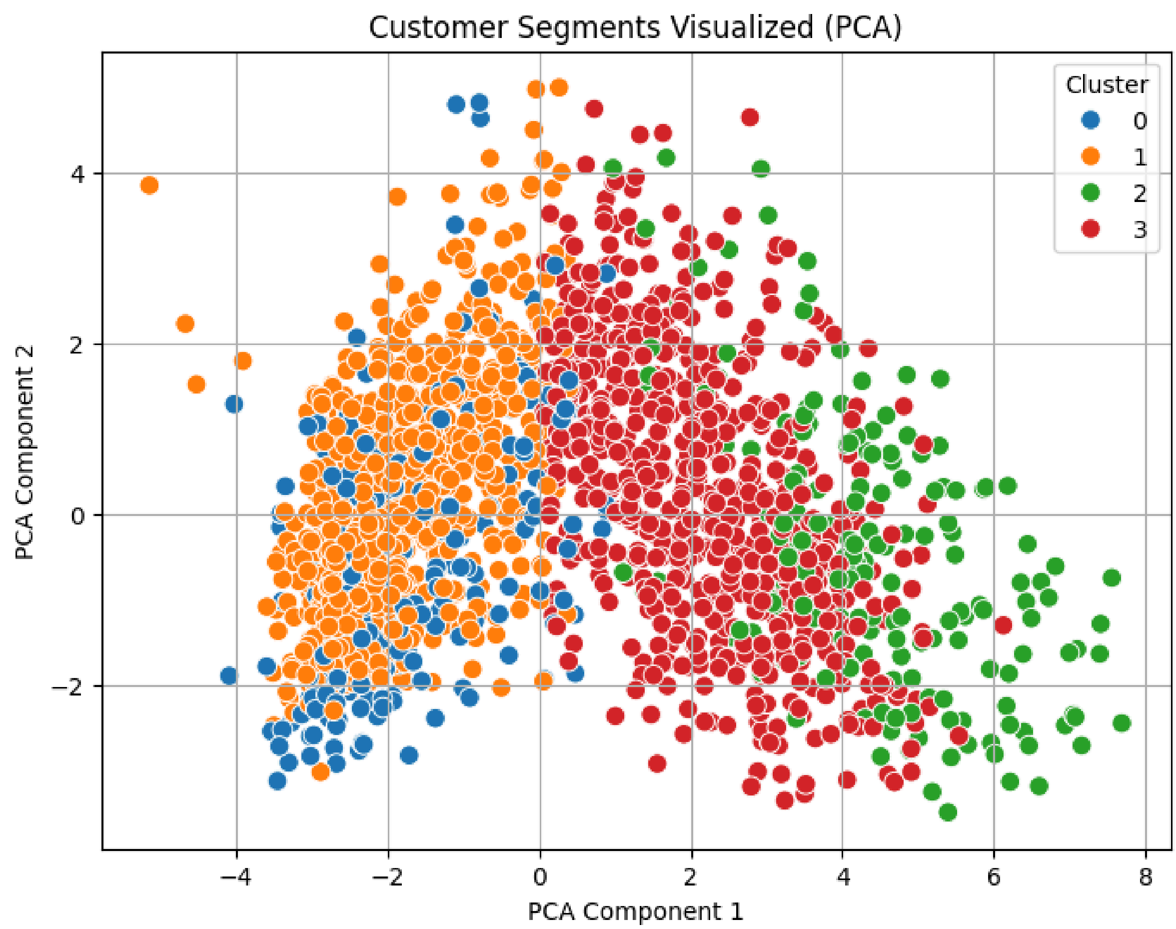
```
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Method - Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```



6. Apply K-Means Clustering & Visualize Clusters

```
# Let's choose 4 clusters (you can change this based on the elbow plot)
k = 4

# Fit the model
kmeans = KMeans(n_clusters=k, random_state=42)
clusters = kmeans.fit_predict(scaled_df)

# Add cluster labels to original data
data['Cluster'] = clusters


# Visualizing clusters using PCA

pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_df)


# DataFrame for plotting
pca_df = pd.DataFrame(data=pca_data, columns=['PCA1', 'PCA2'])
pca_df['Cluster'] = clusters

plt.figure(figsize=(8, 6))
sns.scatterplot(data=pca_df, x='PCA1', y='PCA2', hue='Cluster', palette='tab10', s=70)
```

```
sns.scatterplot(data=pca_df, x='PCA1', y='PCA2', hue='Cluster', palette='tab10', s=70)
plt.title('Customer Segments Visualized (PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



7. Cluster Analysis

```
# Mean profile of each cluster
cluster_profile = data.groupby('Cluster').mean(numeric_only=True).round(2)
cluster_profile.T  # Transposed for easier reading
```

Cluster	0	1	2	3
Year_Birth	1972.44	1969.62	1969.09	1966.21
Income	37261.54	38559.86	81055.89	69113.32
Kidhome	0.71	0.71	0.05	0.07
Teenhome	0.48	0.56	0.15	0.53
Recency	47.41	49.10	48.05	49.79
MntWines	92.84	96.33	863.22	524.26
MntFruits	7.75	5.82	55.62	53.81
MntMeatProducts	38.91	34.30	465.02	318.65
MntFishProducts	9.99	8.91	82.08	75.77
MntSweetProducts	7.36	5.84	64.61	53.59
MntGoldProds	25.09	20.87	77.81	73.78
NumDealsPurchases	2.48	2.55	1.14	2.27
NumWebPurchases	2.88	2.87	5.44	5.85
NumCatalogPurchases	0.94	0.82	6.07	4.97
NumStorePurchases	3.83	3.82	8.23	8.63
NumWebVisitsMonth	6.53	6.44	3.11	3.90
AcceptedCmp3	0.10	0.06	0.15	0.06
AcceptedCmp4	0.02	0.04	0.38	0.06
AcceptedCmp5	0.00	0.00	0.83	0.00
AcceptedCmp1	0.01	0.00	0.49	0.05
AcceptedCmp2	0.01	0.00	0.13	0.00
Complain	0.01	0.01	0.01	0.01
Z_CostContact	3.00	3.00	3.00	3.00
Z_Revenue	11.00	11.00	11.00	11.00
Response	0.16	0.08	0.58	0.12
Customer_Year	2013.08	2013.07	2013.03	2012.95
Education_Basic	0.06	0.04	0.00	0.00
Education_Graduation	0.51	0.48	0.52	0.53
Education_Master	0.15	0.19	0.16	0.14
Education_PhD	0.20	0.19	0.24	0.25
Marital_Status_Alone	0.01	0.00	0.00	0.00
Marital_Status_Divorced	0.00	0.13	0.08	0.12
Marital_Status_Married	0.00	0.51	0.42	0.37
Marital_Status_Single	0.98	0.00	0.20	0.19
Marital_Status_Together	0.00	0.33	0.25	0.26

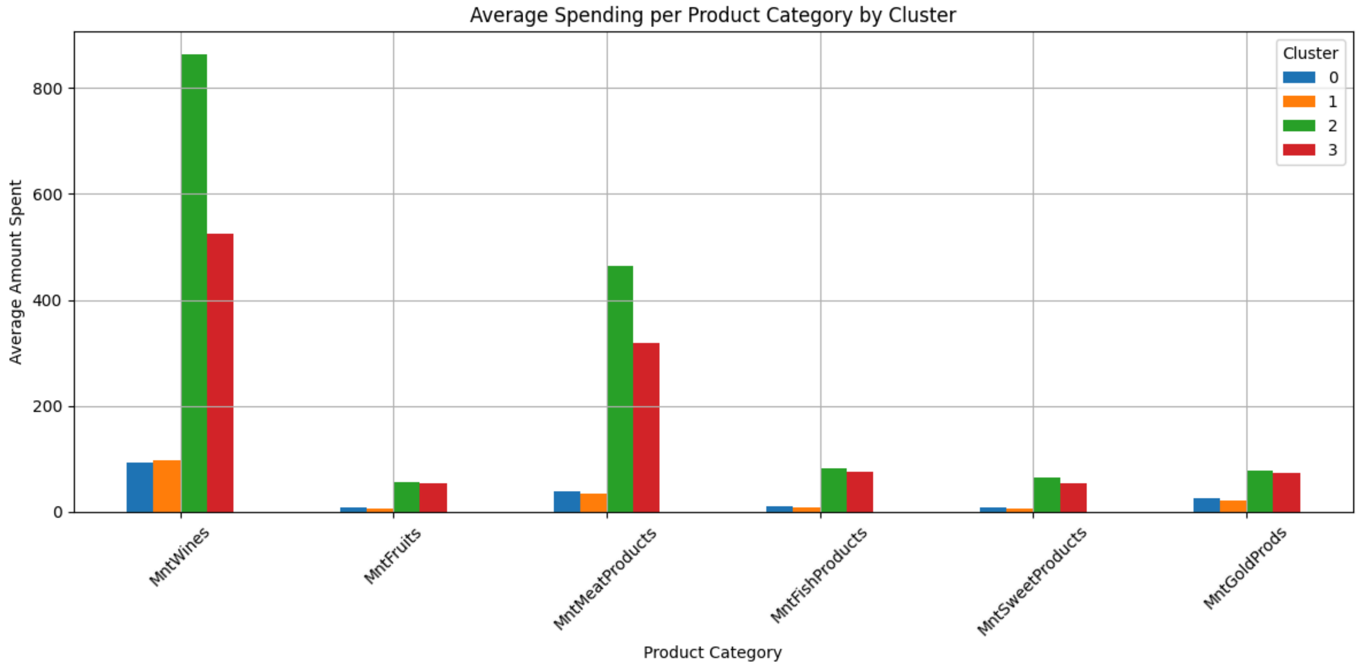
Marital_Status_Widow	0.00	0.03	0.05	0.05
Marital_Status_YOLO	0.01	0.00	0.00	0.00

```
# Number of customers in each cluster
cluster_counts = data['Cluster'].value_counts().sort_index()
print("Number of customers per cluster:")
print(cluster_counts)
```

```
Number of customers per cluster:
Cluster
0    294
1    993
2    195
3    734
Name: count, dtype: int64
```

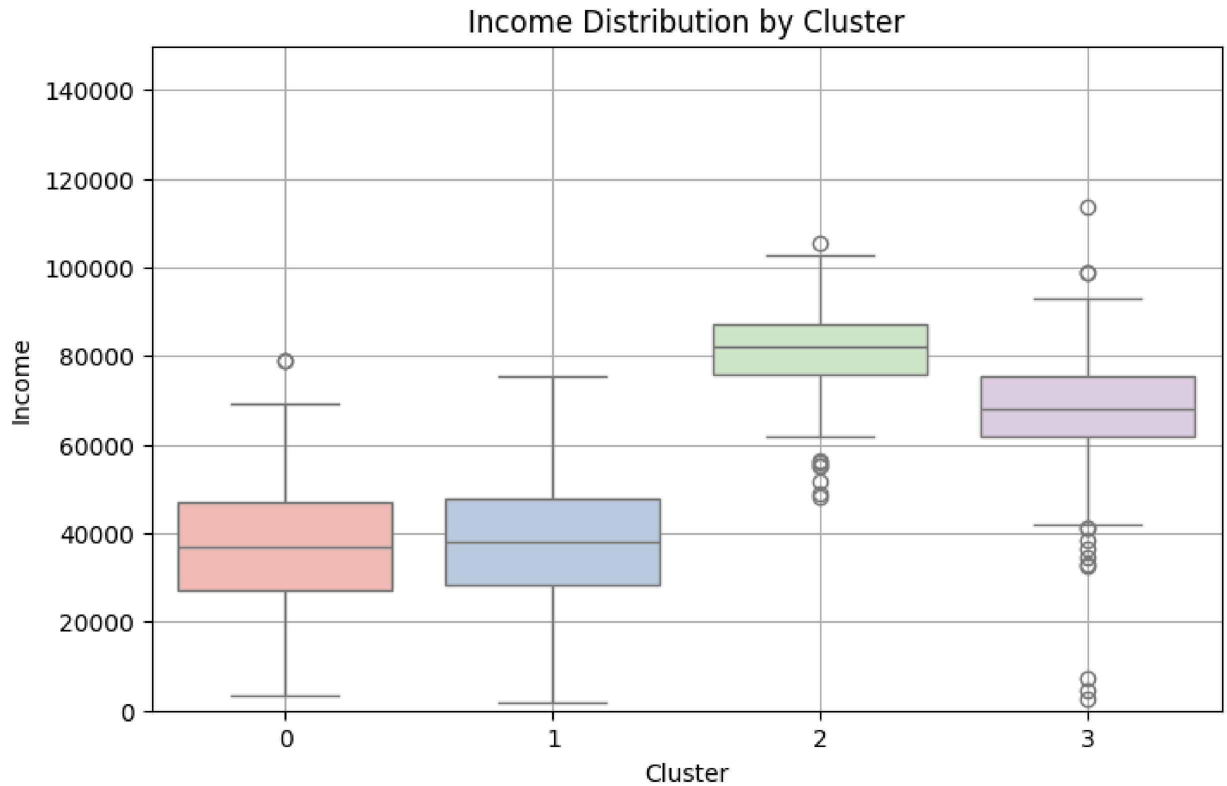
```
# Visualize average spending per cluster
spending_features = [
    'MntWines', 'MntFruits', 'MntMeatProducts',
    'MntFishProducts', 'MntSweetProducts', 'MntGoldProds'
]

avg_spending = data.groupby('Cluster')[spending_features].mean()

plt.figure(figsize=(10, 6))
avg_spending.T.plot(kind='bar', figsize=(12, 6))
plt.title('Average Spending per Product Category by Cluster')
plt.xlabel('Product Category')
plt.ylabel('Average Amount Spent')
plt.xticks(rotation=45)
plt.legend(title='Cluster', loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

<Figure size 1000x600 with 0 Axes>




```
# Comparing income distribution by cluster visual
plt.figure(figsize=(8, 5))
sns.boxplot(data=data, x='Cluster', y='Income', palette='Pastel1')
plt.title('Income Distribution by Cluster')
plt.ylim(0, 150000)  # Adjustable the upper limit as needed
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-149-3784089105.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

  sns.boxplot(data=data, x='Cluster', y='Income', palette='Pastel1')
```




```
#Cluster vs Web Activity
plt.figure(figsize=(8, 5))
sns.barplot(data=data, x='Cluster', y='NumWebPurchases', ci=None, palette='Set3')
plt.title('Average Web Purchases per Cluster')
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-150-486649454.py:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=data, x='Cluster', y='NumWebPurchases', ci=None, palette='Set3')
/tmp/ipython-input-150-486649454.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

  sns.barplot(data=data, x='Cluster', y='NumWebPurchases', ci=None, palette='Set3')
```