Retail Analytics Customer Segmentation Analysis

Spring 2025, University of West Georgia

By

Enita Omuvwie

Table of contents

Introduction

- The dataset contains **8068** entries and **9** columns with customer attributes like demographics and behaviors
- Explore customer attributes, segment based on characteristics, provide insights and inform strategic business decisions

Description of Features:

- **Gender:** Categorical variable indicating the gender of the customer (e.g., Male, Female).
- **Ever_Married:** Categorical variable indicating whether the customer has ever been married (e.g., Yes, No).
- **Age:** Numerical variable representing the age of the customer, measured in years.
- **Graduated:** Categorical variable indicating whether the customer has graduated from a higher education institution (e.g., Yes, No).
- **Profession:** Categorical variable representing the customer's profession (e.g., Engineer, Teacher).
- **Work_Experience:** Numerical variable indicating the number of years of work experience the customer has.
- **Spending_Score:** Categorical variable that rates the customer’s spending behavior (e.g., Low, Medium, High).
- **Family_Size:** Numerical variable indicating the size of the customer’s family.
- **Var_1:** Categorical variable for customer

Objective

- The primary objective of this analysis is to perform customer segmentation using K-Means clustering.
- By identifying distinct customer groups, we aim to uncover insights that can inform marketing strategies and enhance customer engagement.

Methodology

Data Preprocessing	• Load in datasets (Test and Train) • Checking for missing values • Handling missing values
Data Exploration	• Generate summary statistics • Visualizing variable distributions for numerical and categorical features • Correlation plot using heatmap
Clustering	• Choose K-means algorithm for segmentation • Determine optimal no. of cluster using elbow method • Trained K-means model on scaled training data • Assign cluster labels to both datasets
Customer Profiling	• Create profile for each segment based on characteristics • Analyze key features like age, income
Result Interpretation	• Summarized findings and insights for cluster analysis • Provide actional recommendations based on customer segments

Code and Output : Data Preprocessing

Loading Libraries and Preprocessing Function to load in datasets

Enita Omuvwie Customer Segmentation

[70]:
```
# Reading in the libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import silhouette_score
```

[71]:
```
# Function to load data
def load_data(train_path, test_path):
    train_data = pd.read_csv(train_path)
    test_data = pd.read_csv(test_path)
    return train_data, test_data
```

[72]:
```
# Load datasets
train_data, test_data = load_data('Train1.csv', 'Test1.csv')

# Preprocessing
train_data = train_data.drop(["Segmentation", "ID"], axis="columns")
test_data = test_data.drop(["ID"], axis="columns")
```

[73]:
```
# Check first few rows of train data
print(train_data.head())
```

```
  Gender Ever_Married  Age Graduated     Profession  Work_Experience  \
0   Male           No   22        No     Healthcare              1.0
1 Female          Yes   38       Yes       Engineer              NaN
2 Female          Yes   67       Yes       Engineer              1.0
3   Male          Yes   67       Yes         Lawyer              0.0
4 Female          Yes   40       Yes  Entertainment              NaN

  Spending_Score  Family_Size  Var_1
0            Low          4.0  Cat_4
1        Average          3.0  Cat_4
2            Low          1.0  Cat_6
3           High          2.0  Cat_6
4           High          6.0  Cat_6
```

Code and Output : Data Preprocessing

Checking the data overview and descriptive statistics

[74]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8068 entries, 0 to 8067
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Gender           8068 non-null   object 
 1   Ever_Married     7928 non-null   object 
 2   Age              8068 non-null   int64  
 3   Graduated        7990 non-null   object 
 4   Profession       7944 non-null   object 
 5   Work_Experience  7239 non-null   float64
 6   Spending_Score   8068 non-null   object 
 7   Family_Size      7733 non-null   float64
 8   Var_1            7992 non-null   object 
dtypes: float64(2), int64(1), object(6)
memory usage: 567.4+ KB
```

[75]:
```
train_data.describe().T
```

[75]:

	count	mean	std	min	25%	50%	75%	max
Age	8068.0	43.466906	16.711696	18.0	30.0	40.0	53.0	89.0
Work_Experience	7239.0	2.641663	3.406763	0.0	0.0	1.0	4.0	14.0
Family_Size	7733.0	2.850123	1.531413	1.0	2.0	3.0	4.0	9.0

[76]:
```
train_data.describe(include='object').T
```

[76]:

	count	unique	top	freq
Gender	8068	2	Male	4417
Ever_Married	7928	2	Yes	4643
Graduated	7990	2	Yes	4968
Profession	7944	9	Artist	2516
Spending_Score	8068	3	Low	4878
Var_1	7992	7	Cat_6	5238

Code and Output : Data Preprocessing

Checking the data overview and descriptive statistics

```
[77]: test_data.head()
```

[77]:

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
0	Female	Yes	36	Yes	Engineer	0.0	Low	1.0	Cat_6
1	Male	Yes	37	Yes	Healthcare	8.0	Average	4.0	Cat_6
2	Female	Yes	69	No	NaN	0.0	Low	1.0	Cat_6
3	Male	Yes	59	No	Executive	11.0	High	2.0	Cat_6
4	Female	No	19	No	Marketing	NaN	Low	4.0	Cat_6

```
[78]: test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2627 entries, 0 to 2626
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           2627 non-null   object
 1   Ever_Married     2577 non-null   object
 2   Age              2627 non-null   int64
 3   Graduated        2603 non-null   object
 4   Profession       2589 non-null   object
 5   Work_Experience  2358 non-null   float64
 6   Spending_Score   2627 non-null   object
 7   Family_Size      2514 non-null   float64
 8   Var_1            2595 non-null   object
dtypes: float64(2), int64(1), object(6)
memory usage: 184.8+ KB
```

```
[79]: test_data.describe()
```

[79]:

	Age	Work_Experience	Family_Size
count	2627.000000	2358.000000	2514.000000
mean	43.649791	2.552587	2.825378
std	16.967015	3.341094	1.551906
min	18.000000	0.000000	1.000000
25%	30.000000	0.000000	2.000000
50%	41.000000	1.000000	2.000000
75%	53.000000	4.000000	4.000000
max	89.000000	14.000000	9.000000

```
[80]: test_data.describe(include="object").T
```

[80]:

	count	unique	top	freq
Gender	2627	2	Male	1424
Ever_Married	2577	2	Yes	1520
Graduated	2603	2	Yes	1602
Profession	2589	9	Artist	802
Spending_Score	2627	3	Low	1616
Var_1	2595	7	Cat_6	1672

Code and Output : Data Preprocessing

Checking for missing values

```python
# Function to plot missing values
def plot_missing_values(data, dataset_name="Dataset"):
    """
    Plots the percentage of missing values for each column in the dataset.

    Parameters:
    - data: Pandas DataFrame containing the dataset
    - dataset_name: String, name of the dataset to display in the title
    """
    missing_data = data.isnull().sum()
    missing_percent = (missing_data[missing_data > 0] / data.shape[0]) * 100
    missing_percent.sort_values(ascending=True, inplace=True)

    if missing_percent.empty:
        print(f"No missing values in {dataset_name}!")
        return

    fig, ax = plt.subplots(figsize=(15, 4))
    ax.barh(missing_percent.index, missing_percent, color='blue')

    for i, (value, name) in enumerate(zip(missing_percent, missing_percent.index)):
        ax.text(value + 0.5, i, f"{value:.2f}%", ha='left', va='center', fontweight='bold', fontsize=12, color='black')

    ax.set_xlim([0, min(100, max(missing_percent) + 5)])  # Dynamically adjust x-axis limit
    plt.title(f"Percentage of Missing Values - {dataset_name}", fontweight='bold', fontsize=16)
    plt.xlabel('Percentages (%)', fontsize=12)
    plt.show()
```

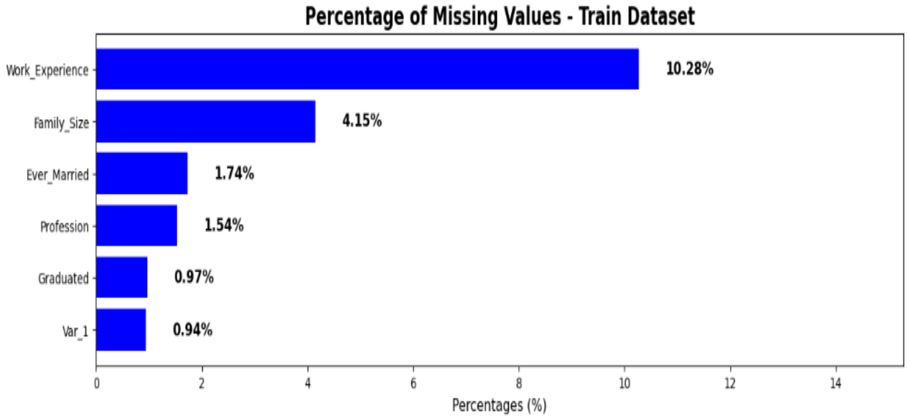
```python
# Plot missing values
plot_missing_values(train_data, "Train Dataset")
plot_missing_values(test_data, "Test Dataset")
```

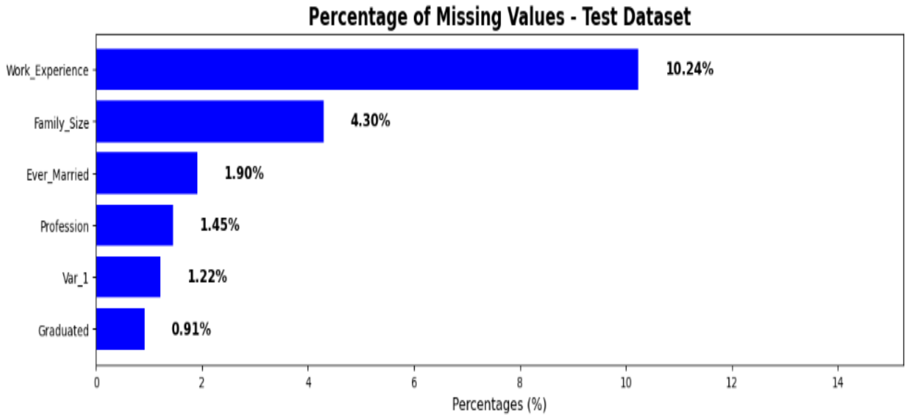
Code and Output : Data Preprocessing

Checking for missing values

```python
# Function to plot missing values
def plot_missing_values(data, dataset_name="Dataset"):
    """
    Plots the percentage of missing values for each column in the dataset.

    Parameters:
    - data: Pandas DataFrame containing the dataset
    - dataset_name: String, name of the dataset to display in the title
    """
    missing_data = data.isnull().sum()
    missing_percent = (missing_data[missing_data > 0] / data.shape[0]) * 100
    missing_percent.sort_values(ascending=True, inplace=True)

    if missing_percent.empty:
        print(f"No missing values in {dataset_name}!")
        return

    fig, ax = plt.subplots(figsize=(15, 4))
    ax.barh(missing_percent.index, missing_percent, color='blue')

    for i, (value, name) in enumerate(zip(missing_percent, missing_percent.index)):
        ax.text(value + 0.5, i, f"{value:.2f}%", ha='left', va='center', fontweight='bold', fontsize=12, color='black')

    ax.set_xlim([0, min(100, max(missing_percent) + 5)])  # Dynamically adjust x-axis limit
    plt.title(f"Percentage of Missing Values - {dataset_name}", fontweight='bold', fontsize=16)
    plt.xlabel('Percentages (%)', fontsize=12)
    plt.show()
```

```python
# Plot missing values
plot_missing_values(train_data, "Train Dataset")
plot_missing_values(test_data, "Test Dataset")
```





Code and Output : Data Preprocessing

Handling missing values

```python
# Function to handle missing values
def handle_missing_values(data, categorical_cols, numerical_cols):
    for col in categorical_cols:
        data[col].fillna(data[col].mode()[0], inplace=True)
    for col in numerical_cols:
        data[col].fillna(data[col].median(), inplace=True)
    return data


# Handle missing values
categorical_cols = ['Ever_Married', 'Graduated', 'Profession', 'Var_1']
numerical_cols = ['Work_Experience', 'Family_Size']
train_data = handle_missing_values(train_data, categorical_cols, numerical_cols)
test_data = handle_missing_values(test_data, categorical_cols, numerical_cols)
```

Code and Output : Data Exploration

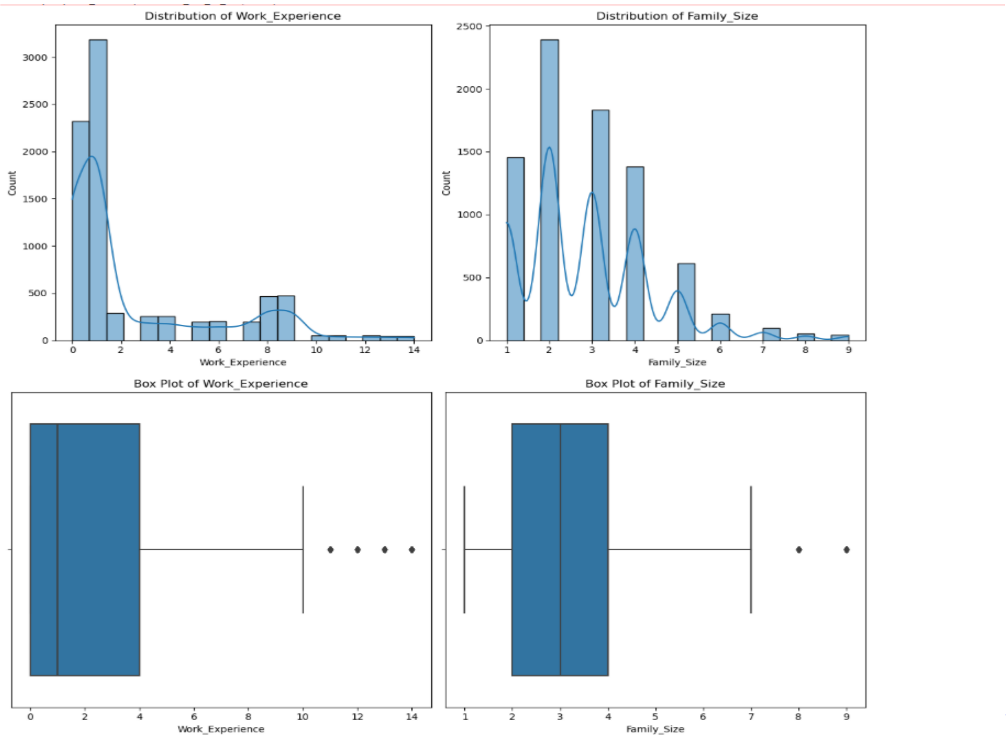
Visualizing distribution of work experience among customers and family size

```
# Visualize distributions of numerical features
plt.figure(figsize=(12, 6))
for i, col in enumerate(numerical_cols):
    plt.subplot(1, 2, i + 1)
    sns.histplot(train_data[col], bins=20, kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()

# Box plots for numerical features
plt.figure(figsize=(12, 6))
for i, col in enumerate(numerical_cols):
    plt.subplot(1, 2, i + 1)
    sns.boxplot(x=train_data[col])
    plt.title(f'Box Plot of {col}')
plt.tight_layout()
plt.show()
```

Code and Output : Data Exploration

Visualizing distribution of work experience among customers, showing right skewed pattern for customers less than 5 years of experience. Box plot shows a central tendency for work experience.

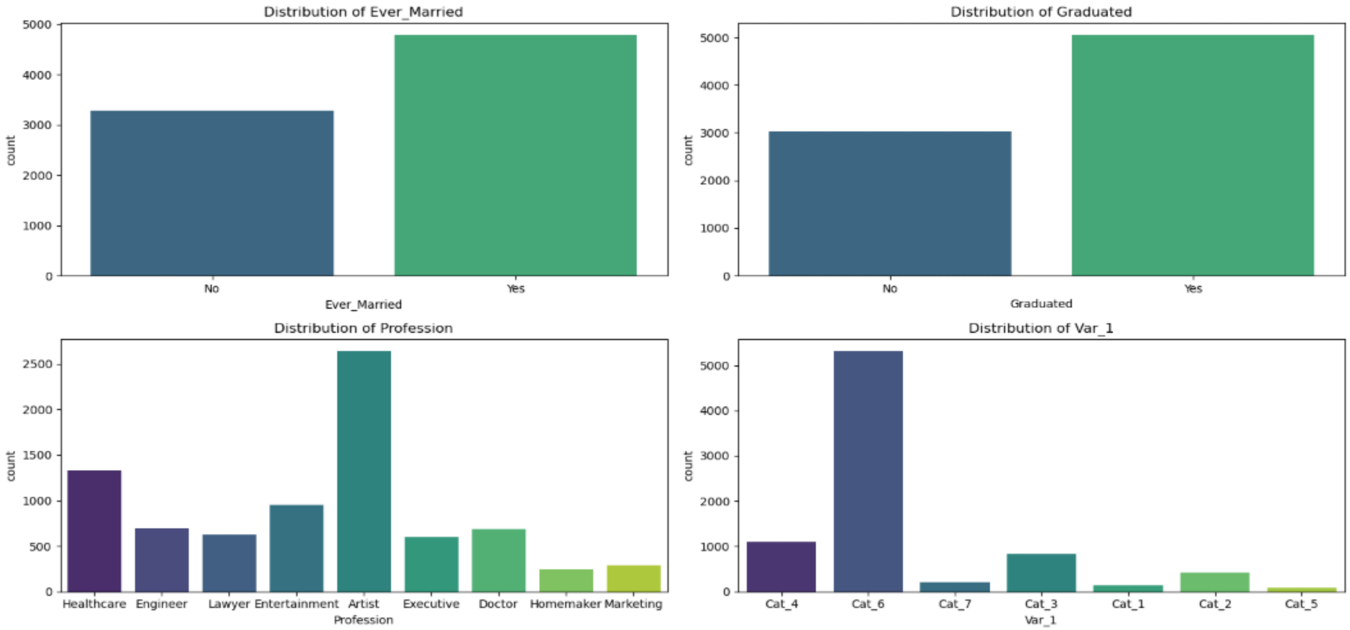
Family size indicates customers have between 2-4 members and the box plot shows the median of the data size



Code and Output : Data Exploration

Visualizing categorical variables showing higher proportion of counts in different variable values

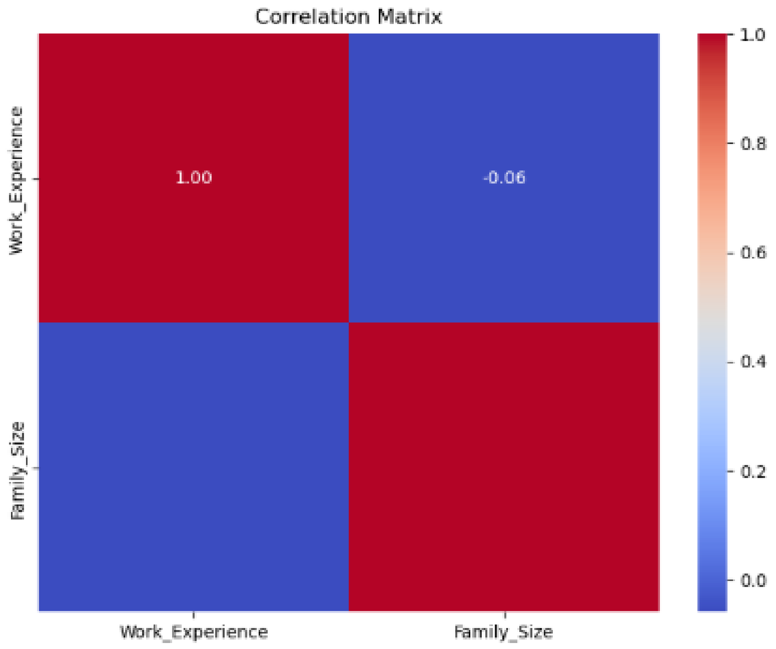
```
# Visualize distributions of categorical features
plt.figure(figsize=(16, 8))
for i, col in enumerate(categorical_cols):
    plt.subplot(2, 2, i + 1)
    sns.countplot(data=train_data, x=col, palette='viridis')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



Code and Output : Data Exploration

Correlation Analysis: Indicating weak correlation between Family Size and Work Experience

```
# Correlation Analysis
correlation_matrix = train_data[numerical_cols].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

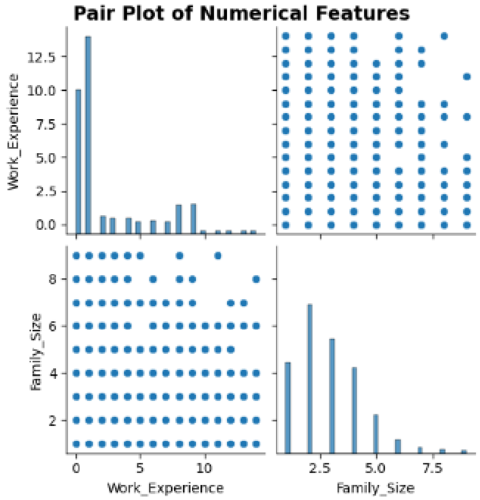


Code and Output : Data Exploration

Pair Plot Analysis showing the relationship between numerical features allowing the identification of potential correlations and distributions

```
]: # Pair plot
plt.figure(figsize=(10, 6))
sns.pairplot(train_data[numerical_cols])  # Pair plot

# Set title correctly for the entire figure
plt.suptitle('Pair Plot of Numerical Features', fontweight='bold', fontsize=14, y=1.02)

plt.show()
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is d
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is d
  with pd.option_context('mode.use_inf_as_na', True):
<Figure size 1000x600 with 0 Axes>
```



Code and Output : Clustering

Label Encoding: Converted variables into numerical format using one-hot encoding

Feature Scaling-Standardization: Standardized the dataset by transforming features to have a mean of 0 and standard deviation of 1

Label Encoding

[85]:
```
# Function to encode categorical variables
def encode_data(data):
    return pd.get_dummies(data, drop_first=True)
```

[86]:
```
# Encode categorical variables
train_encoded = encode_data(train_data)
test_encoded = encode_data(test_data)
```

Feature Scaling - Standardization

[87]:
```
# Function to scale data
def scale_data(data):
    scaler = StandardScaler()
    return scaler.fit_transform(data)
```

[88]:
```
# Scale data
scaled_train = scale_data(train_encoded)
scaled_test = scale_data(test_encoded)
```

Code and Output : Clustering

K-Means Clustering Function: This function trains the model and returns predicted cluster label

Clustering Execution: The model trained on 3 clusters on the scaled training data and resulting cluster labels added

Cluster Prediction for Test Data: The trained model is applied to the scaled test to predict cluster labels added to test dataset

```
# Function to perform K-Means clustering
def perform_kmeans(data, n_clusters):
    kmeans_model = KMeans(n_clusters=n_clusters, random_state=42)
    return kmeans_model.fit_predict(data)
```

```
# Perform K-Means clustering
n_clusters = 3
train_clusters = perform_kmeans(scaled_train, n_clusters)

# Add cluster labels to the dataset
train_encoded['Cluster'] = train_clusters

# Predict clusters for test data
test_clusters = perform_kmeans(scaled_test, n_clusters)
test_encoded['Cluster'] = test_clusters
```

Code and Output : Clustering

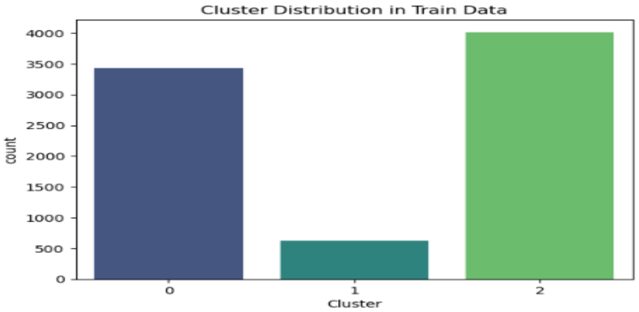
Cluster Distribution Visualization: Body of Code

```
# Visualize cluster distribution
sns.countplot(x=train_encoded['Cluster'], palette='viridis')
plt.title("Cluster Distribution in Train Data")
plt.show()

# Summary statistics per cluster
cluster_summary = train_encoded.groupby('Cluster').mean()
print(cluster_summary)

# Display the count of customers in each cluster
cluster_counts = train_encoded['Cluster'].value_counts()
print(cluster_counts)

# Elbow method to determine optimal k
ssd = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(scaled_train)
    ssd.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), ssd, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('SSD')
plt.title('Elbow Method to Determine Optimal k')
plt.show()
```

Code and Output : Clustering

Summary Statistics per Cluster: Shows the statistics for each cluster providing insights into the average customer characteristics within each segment

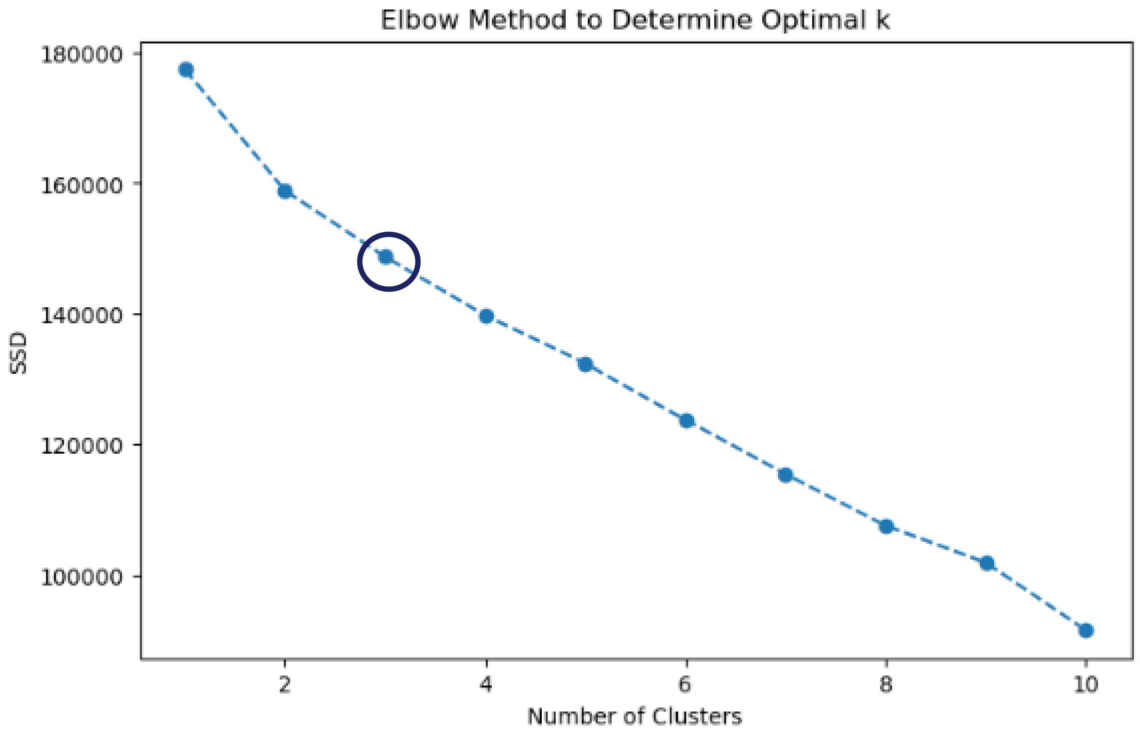
Customer Count in Each Cluster: The count is displayed offering a view of the size of each segment




```
               Age  Work_Experience  Family_Size  Gender_Male  \
Cluster
0        31.715701         2.881154     3.001165     0.481794
1        75.464516         1.191935     1.975806     0.509677
2        48.573599         2.321793     2.868493     0.609465

         Ever_Married_Yes  Graduated_Yes  Profession_Doctor  \
Cluster
0                0.055928       0.503059           0.116807
1                0.938710       0.633871           0.000000
2                0.998506       0.728767           0.071482

         Profession_Engineer  Profession_Entertainment  Profession_Executive  \
Cluster
0                   0.082726                  0.105738              0.009904
1                   0.000000                  0.000000              0.000000
2                   0.103362                  0.145953              0.140722

         ...  Profession_Lawyer  Profession_Marketing  Spending_Score_High  \
Cluster  ...
0        ...           0.000874              0.061462             0.000000
1        ...           1.000000              0.000000             0.522581
2        ...           0.000000              0.020174             0.222167

         Spending_Score_Low  Var_1_Cat_2  Var_1_Cat_3  Var_1_Cat_4  \
Cluster
0                  0.998835     0.073405     0.113603     0.147684
1                  0.448387     0.012903     0.045161     0.058065
2                  0.291656     0.040349     0.100623     0.135990

         Var_1_Cat_5  Var_1_Cat_6  Var_1_Cat_7
Cluster
0           0.014856     0.603554     0.029420
1           0.004839     0.862903     0.000000
2           0.007721     0.674222     0.025405
```

Code and Output : Clustering

Elbow Method for Optimal K: The elbow method is implemented to determine optimal number of clusters using sum of squared distances (SSD) for different k values



Code and Output : Clustering

Customer Profiles: Containing a detail of each cluster profile with a summary mean statistics, providing more insights to customer characteristics in each segment

```
# Create a detailed profile for each cluster
for cluster in range(cluster_summary.shape[0]):
    print(f"Profile for Cluster {cluster}:")
    print(f"Average Age: {cluster_summary.loc[cluster, 'Age']:.2f}")
    print(f"Average Work Experience: {cluster_summary.loc[cluster, 'Work_Experience']:.2f} years")
    print(f"Average Family Size: {cluster_summary.loc[cluster, 'Family_Size']:.2f}")
    print(f"Proportion Ever Married: {cluster_counts[cluster] / train_encoded.shape[0]:.2%}")
    print("\n")
```

```
Profile for Cluster 0:
Average Age: 31.72
Average Work Experience: 2.88 years
Average Family Size: 3.00
Proportion Ever Married: 42.55%


Profile for Cluster 1:
Average Age: 75.46
Average Work Experience: 1.19 years
Average Family Size: 1.98
Proportion Ever Married: 7.68%


Profile for Cluster 2:
Average Age: 48.57
Average Work Experience: 2.32 years
Average Family Size: 2.87
Proportion Ever Married: 49.76%
```

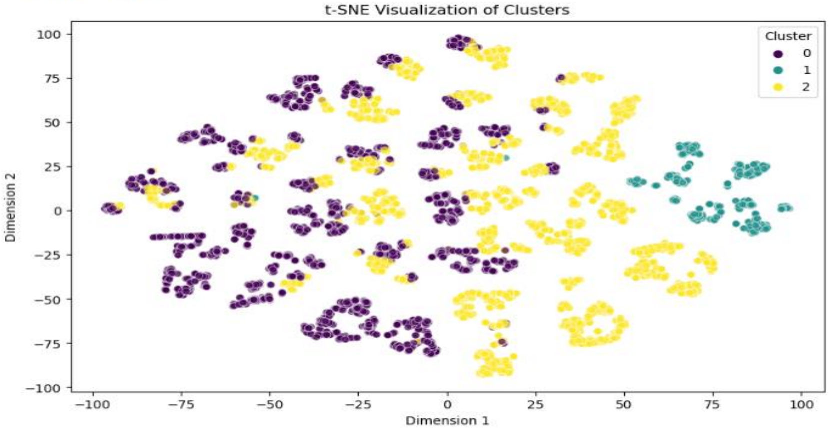
Code and Output : Clustering

Silhouette Score Calculation: Used to assess the quality of clustering with 0.13 indicating clusters are not well separated

T-SNE: Shows the cluster in 2D space showing the distribution of clusters

```
# Calculate the silhouette score
silhouette_avg = silhouette_score(scaled_train, train_clusters)
print(f"Silhouette Score: {silhouette_avg:.2f}")

# Optional: Visualize the silhouette scores for each sample
from sklearn.manifold import TSNE

# Reduce dimensions for visualization
tsne = TSNE(n_components=2, random_state=42)
tsne_results = tsne.fit_transform(scaled_train)

# Create a DataFrame for visualization
tsne_df = pd.DataFrame(tsne_results, columns=['Dimension 1', 'Dimension 2'])
tsne_df['Cluster'] = train_clusters

# Plot the t-SNE results
plt.figure(figsize=(10, 6))
sns.scatterplot(data=tsne_df, x='Dimension 1', y='Dimension 2', hue='Cluster', palette='viridis', alpha=0.7)
plt.title("t-SNE Visualization of Clusters")
plt.show()
```

```
Silhouette Score: 0.13
```



Conclusion

- **Optimal Clusters:** The Elbow method suggests 3 clusters for effective customer segmentation
- **Distinct Profiles:** Significant differences was identified in demographics and behaviors among clusters
- **Silhouette Score:** a score of 0.13 indicates the clusters may not be properly separated
- **Further analysis:** other algorithms can be used to improve the model

- **Recommendations:** Insights can be used for marketing strategies and targeted customers-based advertising based on the results

Thanks!

Do you have any questions?

eomuvwi1@my.westga.edu