

# Customer Segmentation

## Objective

Develop a customer segmentation model to understand customers behavior and separate them in different groups or clusters according to their preferences, and once the division is done, this information can be given to marketing team so they can plan the strategy accordingly.

## Data Description

The sample Dataset summarizes the usage behavior of about 200 active customers during the last 3 months. The file is at a customer level with 5 behavioral variables.

## Attribute Information

Following is the description of the columns for the dataset

- **CustomerID** : Unique ID assigned to the customer
- **Gender** : Gender of the customer
- **Age** : Age of the customer
- **Annual Income (k\$)** : Annual Income of the customer
- **Spending Score** : Score assigned by the mall based on customer behavior and spending nature

## Import required libraries/packages

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

## Load Dataset

```
In [ ]: customer_df = pd.read_csv("Mall_Customers.csv")
```

## Exploratory Data Analysis

### Data Exploration

For the dataset, We'll explore following things:

- First 5 rows
- Data shape
- Data information

- Statistical description
- Data types
- Null values

## First 5 records

```
In [ ]: customer_df.head()
```

## Data Shape

```
In [ ]: customer_df.shape
```

## Data Information

```
In [ ]: customer_df.info()
```

## Statistical description

```
In [ ]: customer_df.describe()
```

## Data Types

```
In [ ]: customer_dtype = customer_df.dtypes
customer_dtype
```

## Null Values

```
In [ ]: customer_df.isnull().sum().sort_values(ascending = False).head()
```

## Observations from Data Exploration

From the above data exploration we saw that

- There is no missing value present
- Shape of the dataset is (200, 5) - 200 rows and 5 columns
- Memory usage by dataset is about 7.9 KB
- There are 4 integer and 1 object type feature present

## Check Distribution

### Continuous Features

```
In [ ]: customer_df.columns
```

```
In [ ]: continuous_features = [ 'Age', 'Annual Income (k$)', 'Spending Score (1-100)']
```

```
In [ ]: f, axes = plt.subplots(2,2 , figsize=(20, 7), sharex=False)
pos = 1
for i, feature in enumerate(continuous_features):

    plt.subplot(1 , 3 , pos)
    ax = sns.histplot(data=customer_df, x = feature,kde=True,palette="husl")
    ax.lines[0].set_color('crimson')
    pos = pos + 1
```

Above distribution shows that:

- The distribution of continuous features are normally distributed.

## Categorical Features

```
In [ ]: sns.countplot(x='Gender', data=customer_df, palette="Set2")
plt.show()
```

- Let's see how gender of customers affects to all other features.

```
In [ ]: #Pairplot
sns.pairplot(customer_df,
              vars=["Age", "Annual Income (k$)", "Spending Score (1-100)"],
              kind="reg",
              hue="Gender",
              palette="husl",
              markers=['o','D'])

plt.show()
```

- From the above pairplot we observe that green colour has higher ratio than pink colour as there are more female customers than male.

## Data Correlation

Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate in relation to each other.

- A **positive** correlation indicates the extent to which those variables increase or decrease in parallel.
- A **negative** correlation indicates the extent to which one variable increases as the other decreases.

```
In [ ]: customer_corr = customer_df.corr()
plt.figure(figsize=(8,8))
sns.heatmap(customer_corr, cmap="icefire", linewidths=.5)
plt.show()
```

## Feature Engineering

- All machine learning algorithms use input data to train a model. This input data comprise features, which are usually in the form of structured columns.
- Algorithms require features with some specific characteristic to work properly. Here, the need for feature engineering arises.

- Feature engineering mainly have two goals:
  1. Preparing the proper input dataset, compatible with the machine learning algorithm requirements.
  2. Improving the performance of machine learning models.

## Drop Columns

```
In [ ]: customer_df.drop(columns='CustomerID',axis=1,inplace=True)
```

## Encoding Categorical Features

### What is Categorical Data?

- Categorical data are variables that contain label values rather than numeric values.
- The number of possible values is often limited to a fixed set.
- Categorical variables are often called nominal.
- Some examples include:
  - A “pet” variable with the values: “dog” and “cat”.
  - A “gender” variable with the values: “male” and “female”
  - A “place” variable with the values: “first”, “second” and “third”.Each value represents a different category.
- Some categories may have a natural relationship to each other, such as a natural ordering. The “place” variable above does have a natural ordering of values. This type of categorical variable is called an ordinal variable.

### What is the Problem with Categorical Data?

- Some algorithms can work with categorical data directly.
- Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.
- In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.
- This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

### How to Convert Categorical Data to Numerical Data?

There are two ways:

- Integer Encoding
- One-Hot Encoding

#### 1. Integer Encoding


- As a first step, each unique category value is assigned an integer value.
- For example, “red” is 1, “green” is 2, and “blue” is 3.
- This is called a **label encoding** or integer encoding and is easily reversible.
- For some variables, this may be enough.

- The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.
- For example, ordinal variables like the “place” example above would be a good example where a label encoding would be sufficient

## 2. One-Hot Encoding

- For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.
- In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).
- In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.
- In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors.

Color		
Red		
Red		
Yellow		
Green		
Yellow		



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

```
In [ ]: from sklearn.preprocessing import OneHotEncoder
```

```
In [ ]: enc = OneHotEncoder()
```

```
In [ ]: enc.fit(customer_df.Gender.values.reshape(-1, 1))
```

```
In [ ]: enc.get_feature_names(['gender'])
```

```
In [ ]: enc.transform(customer_df.Gender.values.reshape(-1, 1)).toarray()
```

```
In [ ]: encoded_array = enc.transform(customer_df.Gender.values.reshape(-1, 1)).toarray()
customer_df['gender_Female'] = encoded_array[:,0]
customer_df['gender_Male'] = encoded_array[:,1]
```

- By reshaping array with `(-1, 1)`, the array gets reshaped in such a way that the resulting array has only 1 column.

```
In [ ]: customer_df.head(10)
```

```
In [ ]: customer_df.drop(columns='Gender',axis=1,inplace=True)
```

```
In [ ]: customer_df.head()
```



# Model Development

```
In [ ]: model = KMeans(n_clusters=3, random_state=21)
```

## Elbow Method

The elbow method finds the value of the optimal number of clusters using the total within-cluster sum of square values.

```
In [ ]: inertia = []
range_val = range(1,15)
for i in range_val:
    kmeans_model = KMeans(n_clusters=i, random_state=21)
    kmeans_model.fit(customer_df)
    inertia.append(kmeans_model.inertia_)

fig = plt.figure(figsize=(6,5),dpi=100)
plt.plot(range_val,inertia,'bx-')
plt.grid(color="#cccccc", linestyle='--')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```

## Silhouette Score

Silhouette refers to a method of interpretation and validation of consistency within clusters of data.

- The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).
- The silhouette ranges from  $-1$  to  $+1$ , where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.
- If most objects have a high value, then the clustering configuration is appropriate.
- If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

```
In [ ]: from sklearn.metrics import silhouette_score
```

```
In [ ]: kmeans_4 = KMeans(n_clusters=4, random_state=21)
kmeans_4.fit(customer_df)
```

```
In [ ]: # Let's plot the silhouette score as a function of k:
silhouette_score(customer_df, kmeans_4.labels_)
```

```
In [ ]: # Create a list of hypothetical scenarios for different number of clusters
kmeans_per_k = [KMeans(n_clusters=k, random_state=21).fit(customer_df) for k in range(1, 10)]
```

```
In [ ]: silhouette_scores = [silhouette_score(customer_df, model.labels_) for model in kmeans_per_k]
```

```
In [ ]: silhouette_scores
```

```
In [ ]: # Plot the silhoutee scores graph
fig = plt.figure(figsize=(16,5),dpi=200)
plt.plot(range(2, 10), silhouette_scores, "b", marker = 'o', linewidth=2, markersize=8, la
plt.xlabel("$k$", fontsize=14, family='Arial')
plt.ylabel("Silhouette score", fontsize=14, family='Arial')
plt.grid(which='major', color="#cccccc", linestyle='--')
plt.title('Silhoutte curve for predict optimal number of clusters', family='Arial', fontsi

# # Find the optimal number of cluster
# # Draw a vertical Line to mark optimal number of clusters
k=6
plt.axvline(x=6, linestyle='--', c='green', linewidth=1,
            label='Optimal number of clusters ({}).format(6))
plt.scatter(6, silhouette_scores[k-2], c='red', s=400)
plt.legend(shadow=True)
plt.show()
```

- From the above elbow method we see that **K = 6** is the best K value for our clustering

## Cluster Plots

```
In [ ]: # apply kmeans algorithm
kmeans_model=KMeans(6, random_state=21)
kmeans_clusters = kmeans_model.fit(customer_df)
```

```
In [ ]: kmeans_model.labels_
```

```
In [ ]: fig = plt.figure(figsize=(5,5), dpi=100)
plt.scatter(customer_df['Age'], customer_df['Spending Score (1-100)'],
            c=kmeans_model.labels_, cmap='gist_rainbow', s=10)
plt.grid(linewidth = 0.2)
plt.xlabel("Age")
plt.ylabel("Spending Score")
plt.show()
```

```
In [ ]: from mpl_toolkits.mplot3d import Axes3D
```

```
In [ ]: customer_df["label"] = kmeans_clusters.labels_

fig = plt.figure(figsize=(20,13), dpi=300)
ax = fig.add_subplot(111, projection='3d')
ax.scatter(customer_df.Age[customer_df.label == 0],
            customer_df["Annual Income (k$)"][customer_df.label == 0],
            customer_df["Spending Score (1-100)"][customer_df.label == 0],
            c='blue', s=60)

ax.scatter(customer_df.Age[customer_df.label == 1],
            customer_df["Annual Income (k$)"][customer_df.label == 1],
            customer_df["Spending Score (1-100)"][customer_df.label == 1],
            c='red', s=60)
```

```
ax.scatter(customer_df.Age[customer_df.label == 2],
           customer_df["Annual Income (k$)"][customer_df.label == 2],
           customer_df["Spending Score (1-100)"][customer_df.label == 2],
           c='green', s=60)

ax.scatter(customer_df.Age[customer_df.label == 3],
           customer_df["Annual Income (k$)"][customer_df.label == 3],
           customer_df["Spending Score (1-100)"][customer_df.label == 3],
           c='orange', s=60)

ax.scatter(customer_df.Age[customer_df.label == 4],
           customer_df["Annual Income (k$)"][customer_df.label == 4],
           customer_df["Spending Score (1-100)"][customer_df.label == 4],
           c='purple', s=60)

ax.view_init(30, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.title("3D Cluster Plot")
plt.show()
```