UNSUPERVISED MACHINE LEARNING PROJECT

GROUP MEMBERS : Om Pawar(A006) and Reese Pereira(A007)

Topic Name : Customer Segmentation using K-Means Clustering

INTRODUCTION

Customer segmentation is all about strategically dividing your customers into groups that share common characteristics. These characteristics can be based on demographics (age, income), behavior (purchase history, website activity), or even psychographics (interests, values).

The goal is to understand your customers better. By segmenting your base, you can tailor your marketing messages, products, and services to resonate more effectively with each group. This can lead to increased sales, happier customers, and a more strategic business approach.

In this project, we are taking 'Mall Customer' dataset, which is a .csv file.

DATASET LINK : https://www.kaggle.com/datasets/shwetabh123/mall-customers

Data Description : Our dataset contains 200 rows and 5 columns namely CustomerID, Genre, Age, Annual Income(k$) and Spending Score(1-100).

Customer ID – It contains customer Id from 1 to 200.

Gender – Wrongly written as Genre. It is a column which contains Male and Female.

Age - It contains the age of the customers.

Annual Income(k$) - It gives us the annual income of the customers in thousand dollars.

Spending Score (1-100) - It is the spending score of a customer which indicates how much they tend to spend when shopping. It's a way for businesses to understand how valuable a customer is based on their purchasing habits.

Objective : When using the K-Means algorithm for customer segmentation with mall customer data, we are trying to group customers into distinct clusters based on their shopping behavior. The goal is to identify different segments of customers who exhibit similar characteristics, such as spending habits, annual income, gender and age based. By segmenting customers, businesses can better understand their target audience, tailor marketing strategies to specific segments, and improve customer satisfaction and loyalty.

Algorithms Applied : K-Means Clustering Algorithm

Why K Means Algorithm ?

We have opted for the K-means algorithm for our project due to the manageable size of our dataset(i.e. 200 rows which is a small dataset). K-means excels at handling smaller datasets, efficiently grouping data points into clusters based on their similarities. This approach allows us to effectively analyze patterns and relationships within our data, making it an ideal choice for our project.

So, K-Means is an algorithm that partitions data points into K clusters based on their features. It works by iteratively assigning data points to the nearest cluster centroid and then updating the centroid based on the mean of the assigned points. This process continues until the centroids no longer change significantly, resulting in K clusters that represent similar data points.

Colab File :
https://colab.research.google.com/drive/1NSNNa3EJ1vbM23WbHG-W0JhG_-cWI5Gr?usp=sharing

PROCESS

For our project, we have imported libraries such as numpy, pandas, matplotlib, seaborn, standardscaler and K-Means. Then we move further for reading the data, by using pd.read_csv. The data is still raw and uncleaned, so therefore we check outliers (boxplot) and null values and hence do further cleaning of the data. After the data is cleaned, we plotted 3 different Distribution plots of ‘Age Range’, ‘Annual Income’ and ‘Spending Score’ versus ‘Density’ in order to visualize the data. Then we use countplot to count the number of male and female. Then we use scatterplots to find relationships between ‘Age v/s Annual Income’ and ‘Spending Score v/s Annual Income’.

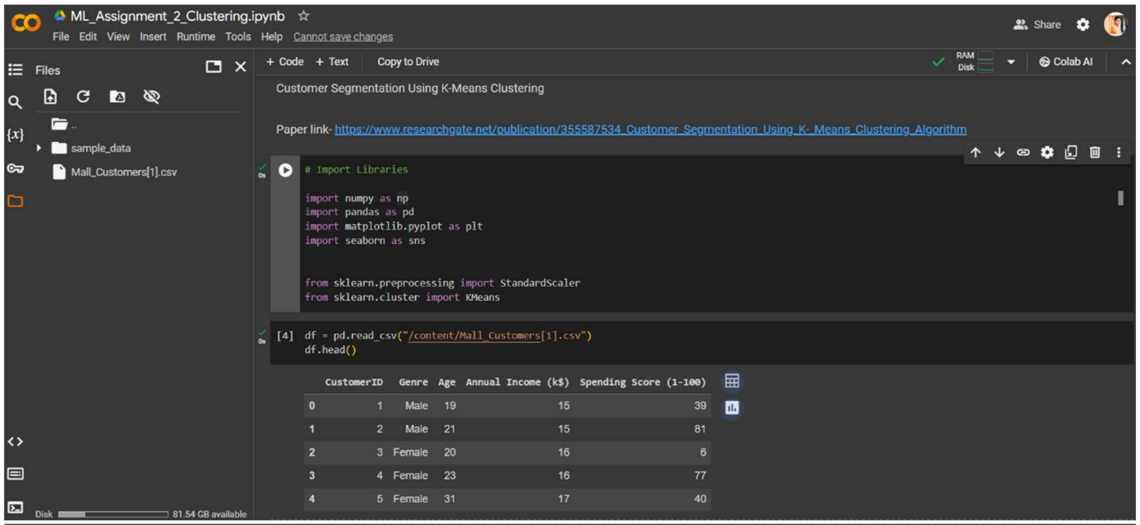
We plotted ‘Number of Customers v/s Age’ barplots to check the number of customers for particular age groups, ‘Number of Customers v/s Spending Score’ to find how much customers have spent and ‘Number of customers v/s Income’ to find the income with respect to number of customers. We used “nipy_spectral_r” palette to assign different colours for different bars. Then we scale the columns ‘Age’, ‘Annual Income’ and ‘Spending Score’ using the StandardScaler function.

Data scaling is the process of transforming the values of the features of a dataset till they are within a specific range. This is to ensure that no single feature dominates the distance calculations in an algorithm, and can help to improve the performance of the algorithm. Then we use df.scaled_fit function because it fits the scaler to the data (df_scaled) and then applies the scaling transformation, resulting in a new DataFrame (df_scaled_fit) containing the scaled data. We have used K Means algorithm, so we are creating a KMeans object with n_clusters=4. This means that we want the algorithm to find 4 clusters in our data. ‘max_iter=50’ sets the maximum number of iterations for the algorithm to converge. If the algorithm doesn't converge within 50 iterations, it stops. ‘kmeans.fit(var_list)’ fits the KMeans model to your data. var_list presumably contains the data points you want to cluster.

The KMeans algorithm will iterate until convergence or until it reaches the maximum number of iterations specified earlier. After this, we will be able to predict cluster labels for new data points or we can analyze the clustering results. We used ‘ssd=[]’, this initializes an empty list called ssd. SSD is commonly used to store the sum of squared distances (SSD) of data points from their respective cluster centroids, which is a measure of how tightly grouped the data points are within clusters. This is used for clustering with different number of clusters ranging from 1 to 10 and storing the inertia values for each case in the ‘ssd’ list. This is typically done by looking for an "elbow" point in the plot of SSD values versus the number of clusters, where the rate of decrease in SSD slows down significantly. Here, the Elbow Curve has been plotted where our x label is K Value and y label is SSD.

From the elbow curve, we can interpret that, the number of clusters is 5. Therefore, we fit KMeans clustering again with number of clusters as 5 and maximum iterations as 50. After that, we use ScatterPlot when K=5 to plot the data into 5 different clusters.

CODE SNAPSHOTS


ML_Assignment_2_Clustering.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
Files
sample_data
Mall_Customers[1].csv
+ Code + Text Copy to Drive
RAM Disk
Colab AI
Disk 81.54 GB available

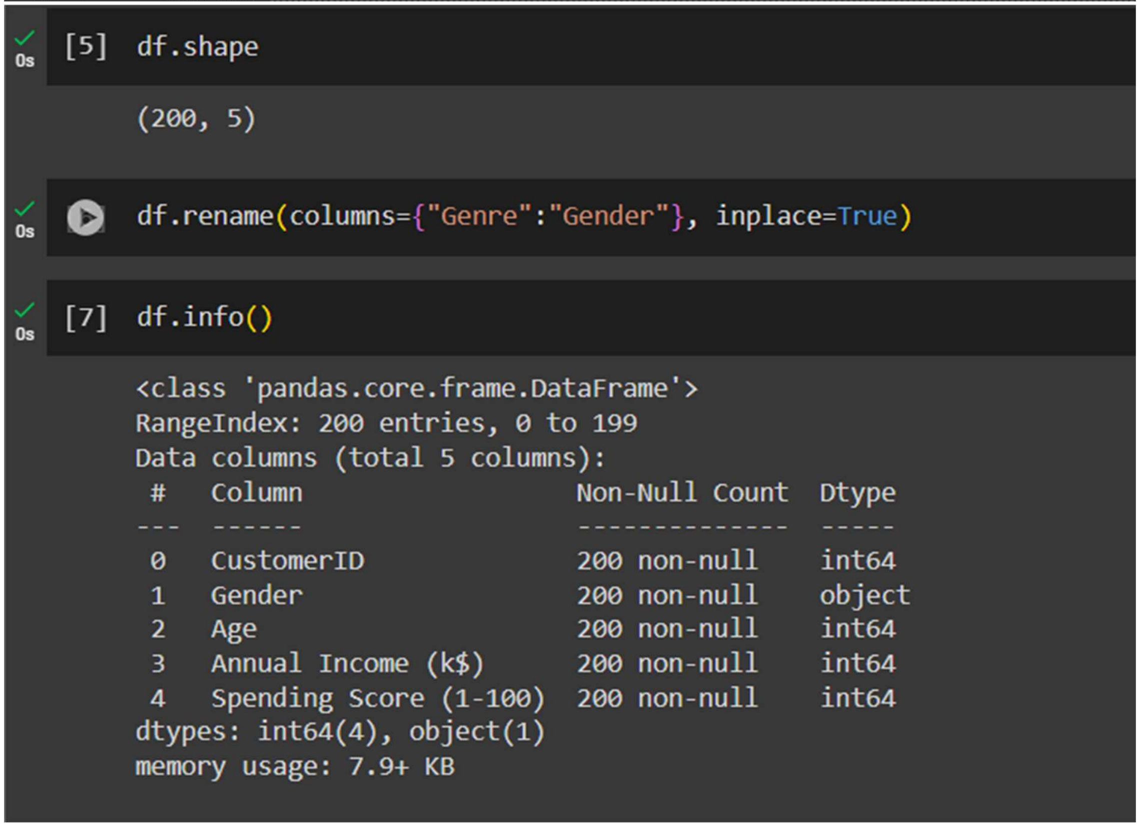
Customer Segmentation Using K-Means Clustering

Paper link- https://www.researchgate.net/publication/355587534_Customer_Segmentation_Using_K-_Means_Clustering_Algorithm

```
# Import Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


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

```
[4] df = pd.read_csv("/content/Mall_Customers[1].csv")
    df.head()
```

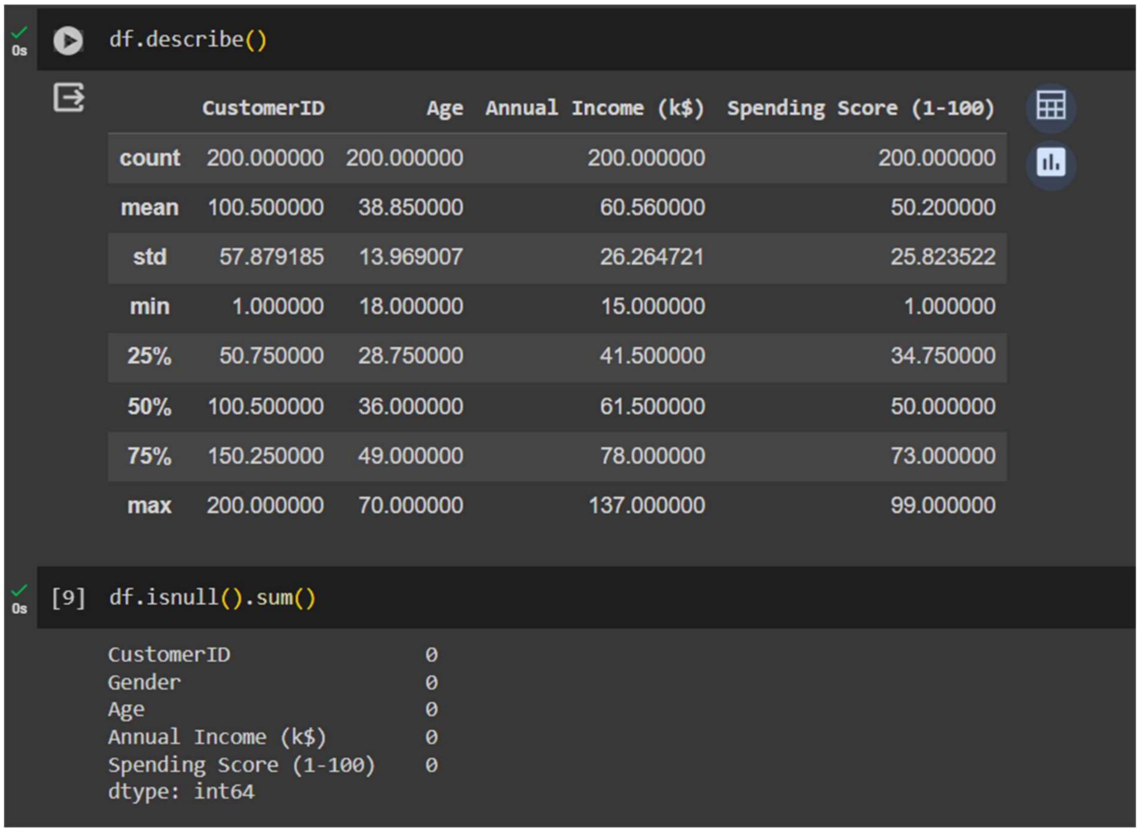
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40



```
[5] df.shape

(200, 5)
```

```
df.rename(columns={"Genre":"Gender"}, inplace=True)
```

```
[7] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

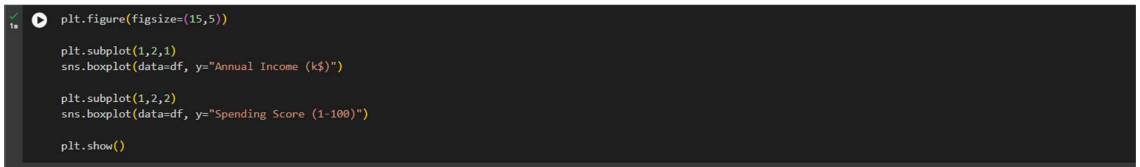


```
df.describe()
```

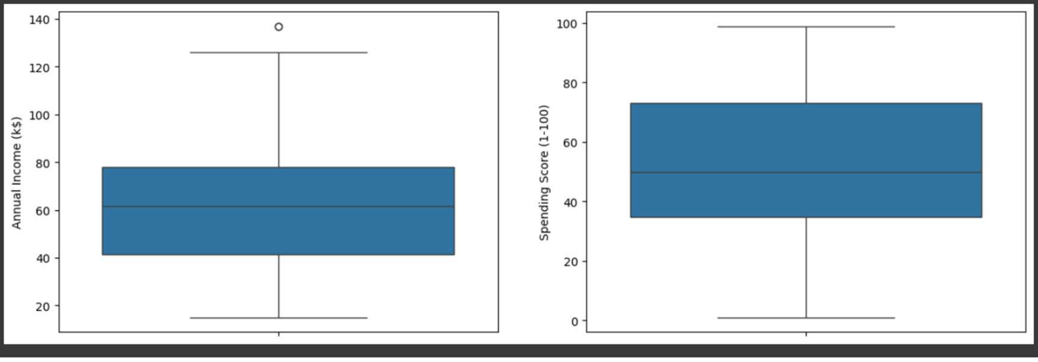
	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

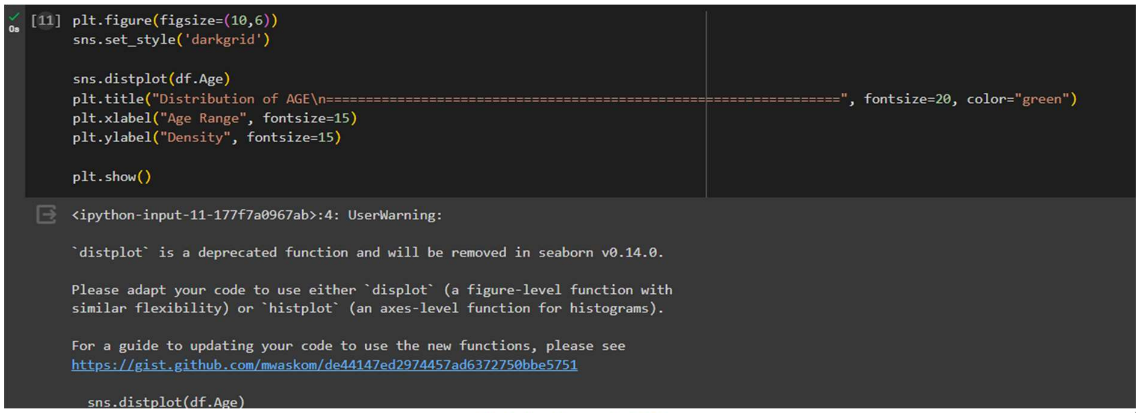
```
[9] df.isnull().sum()
```

```
CustomerID               0
Gender                   0
Age                      0
Annual Income (k$)       0
Spending Score (1-100)   0
dtype: int64
```



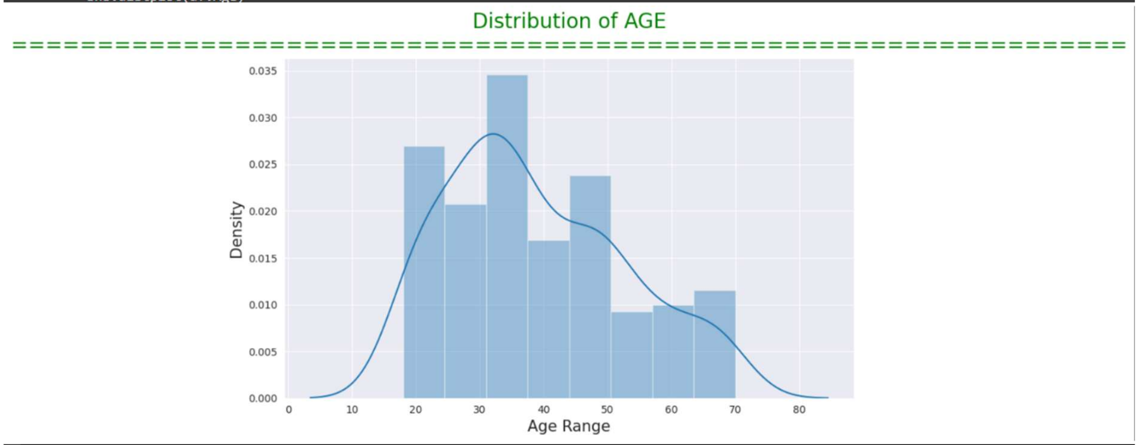
```
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
sns.boxplot(data=df, y="Annual Income (k$)")

plt.subplot(1,2,2)
sns.boxplot(data=df, y="Spending Score (1-100)")

plt.show()
```

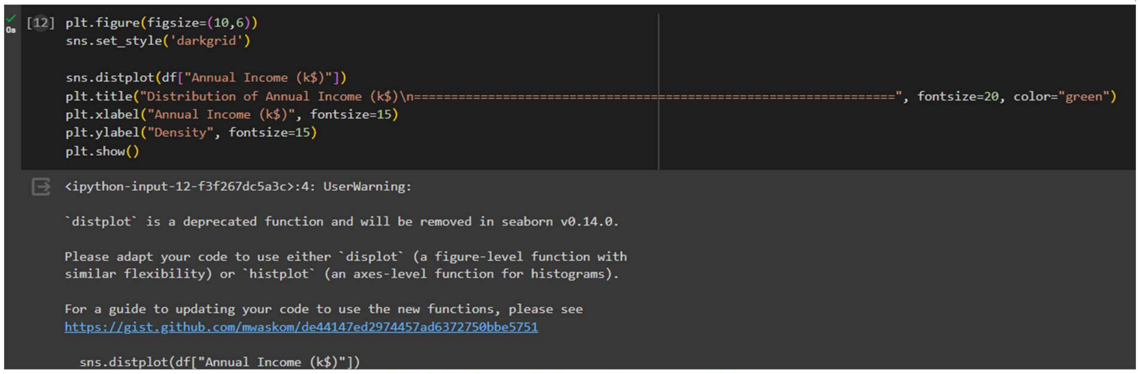




```
[11] plt.figure(figsize=(10,6))
     sns.set_style('darkgrid')

     sns.distplot(df.Age)
     plt.title("Distribution of AGE\n=================================================================", fontsize=20, color="green")
     plt.xlabel("Age Range", fontsize=15)
     plt.ylabel("Density", fontsize=15)

     plt.show()
```

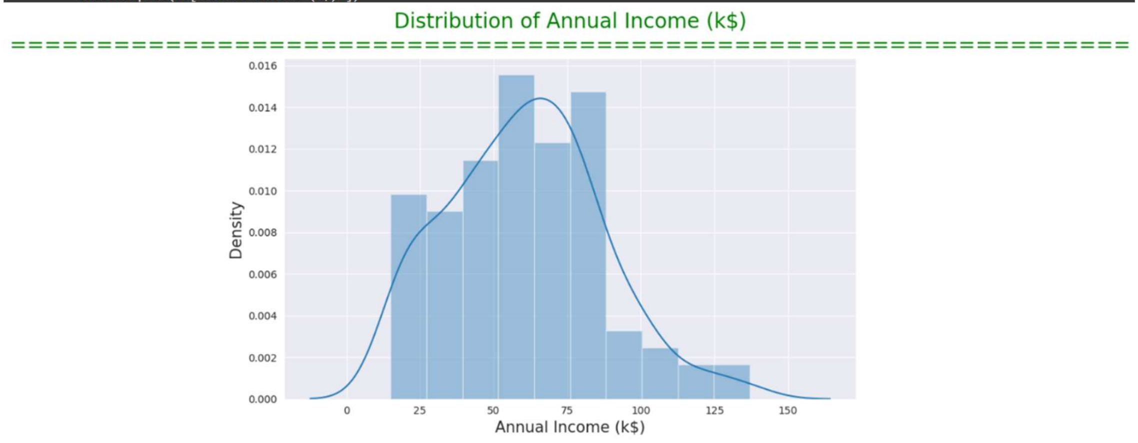
```
<ipython-input-11-177f7a0967ab>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Age)
```

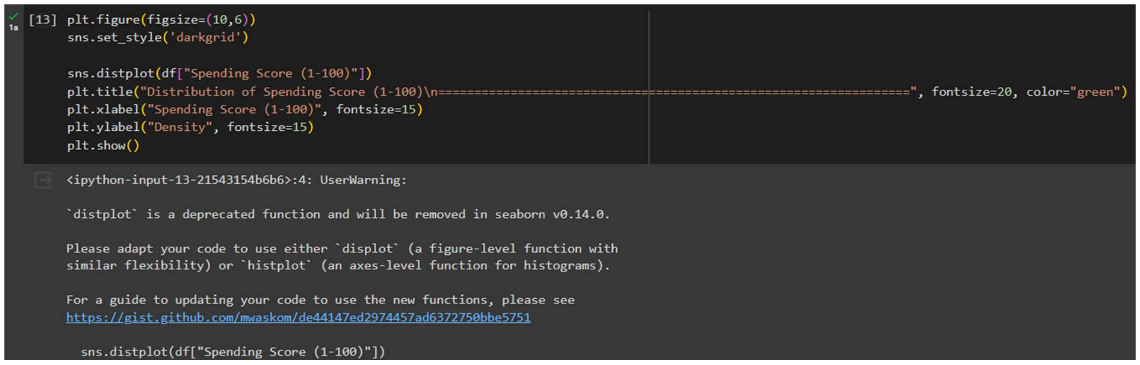




```
[12] plt.figure(figsize=(10,6))
     sns.set_style('darkgrid')

     sns.distplot(df["Annual Income (k$)"])
     plt.title("Distribution of Annual Income (k$)\n=================================================================", fontsize=20, color="green")
     plt.xlabel("Annual Income (k$)", fontsize=15)
     plt.ylabel("Density", fontsize=15)
     plt.show()
```

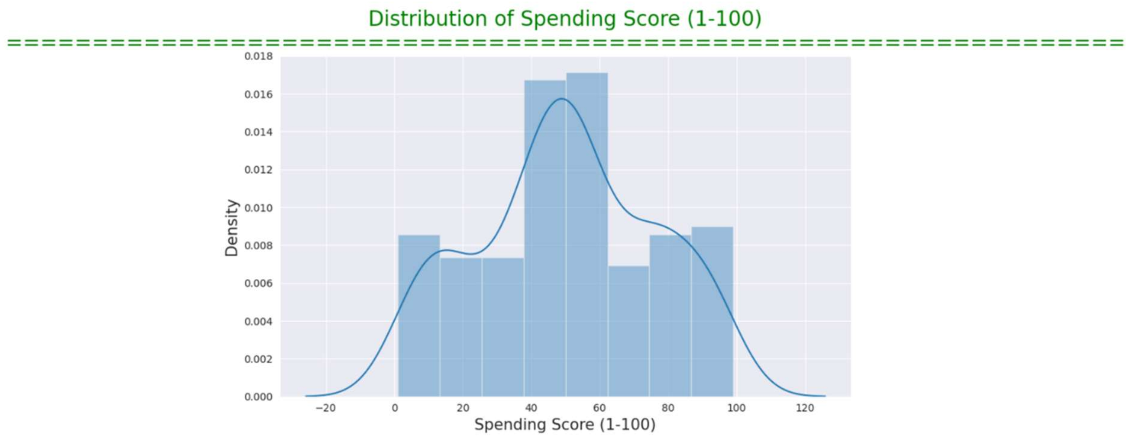
```
<ipython-input-12-f3f267dc5a3c>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Annual Income (k$)"])
```



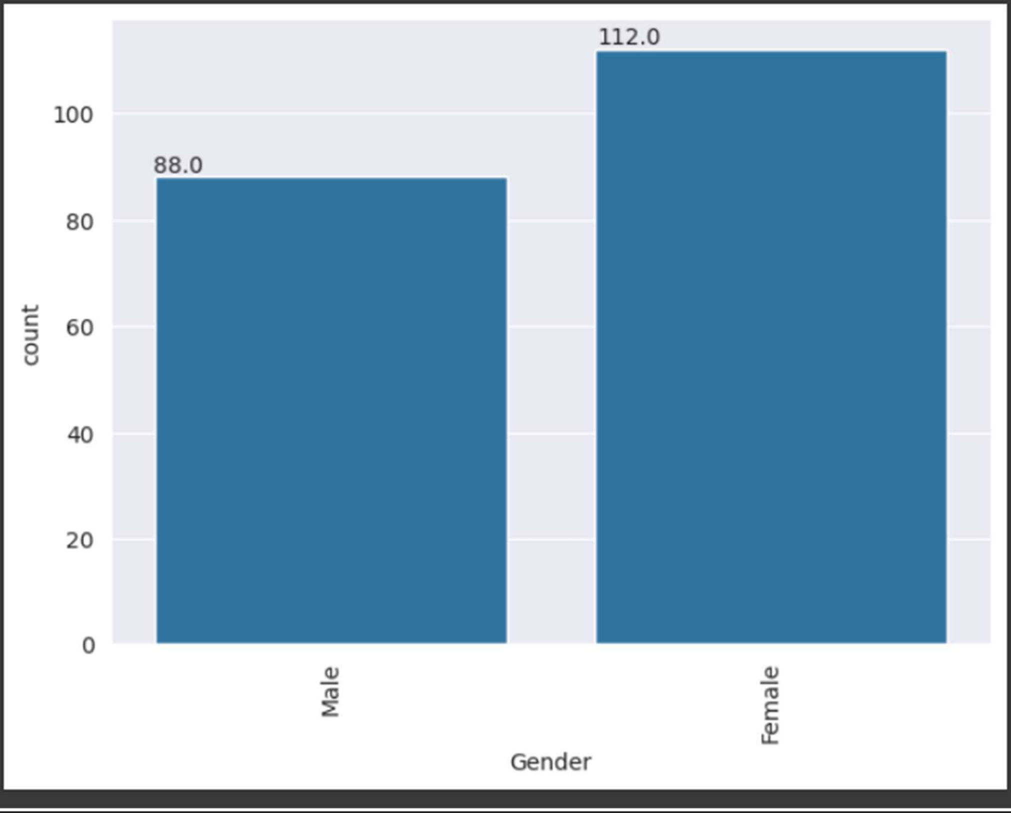


```
[13] plt.figure(figsize=(10,6))
     sns.set_style('darkgrid')

     sns.distplot(df["Spending Score (1-100)"])
     plt.title("Distribution of Spending Score (1-100)\n=================================================================", fontsize=20, color="green")
     plt.xlabel("Spending Score (1-100)", fontsize=15)
     plt.ylabel("Density", fontsize=15)
     plt.show()
```

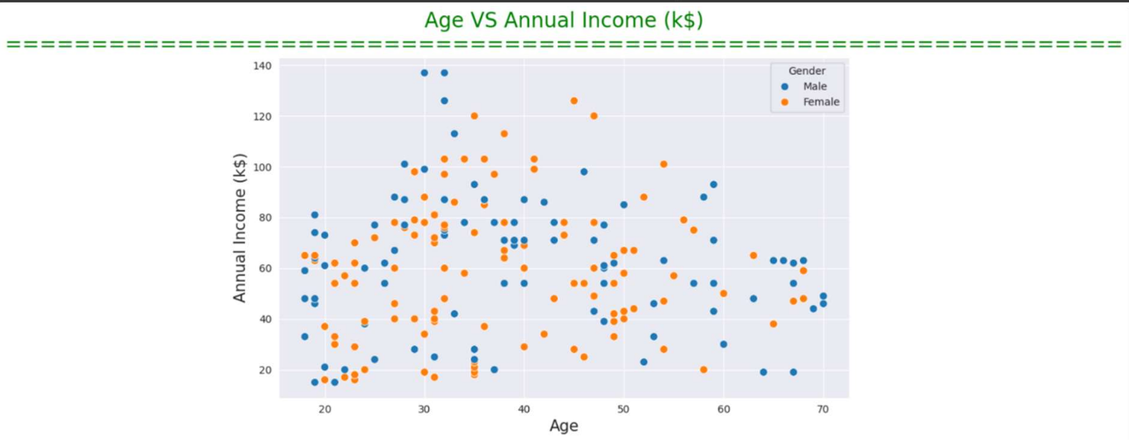
```
<ipython-input-13-21543154b6b6>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Spending Score (1-100)"])
```



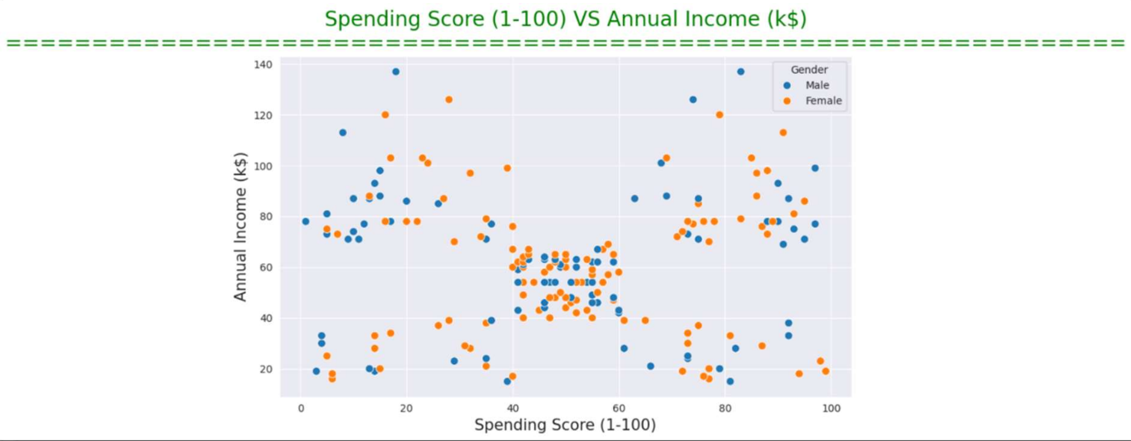
```
plt.figure(figsize = (7,5))
gender = df['Gender'].sort_values(ascending = False)
ax = sns.countplot(x='Gender', data= df)
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.01 , p.get_height() * 1.01))
plt.xticks(rotation=90)
plt.show()
```



```
plt.figure(figsize=(10,6))
sns.set_style('darkgrid')

sns.scatterplot(data=df, x="Age", y= "Annual Income (k$)", hue="Gender", s=60)
plt.title("Age VS Annual Income (k$)\n================================================================", fontsize=20, color="green")
plt.xlabel("Age", fontsize=15)
plt.ylabel("Annual Income (k$)", fontsize=15)
plt.show()
```



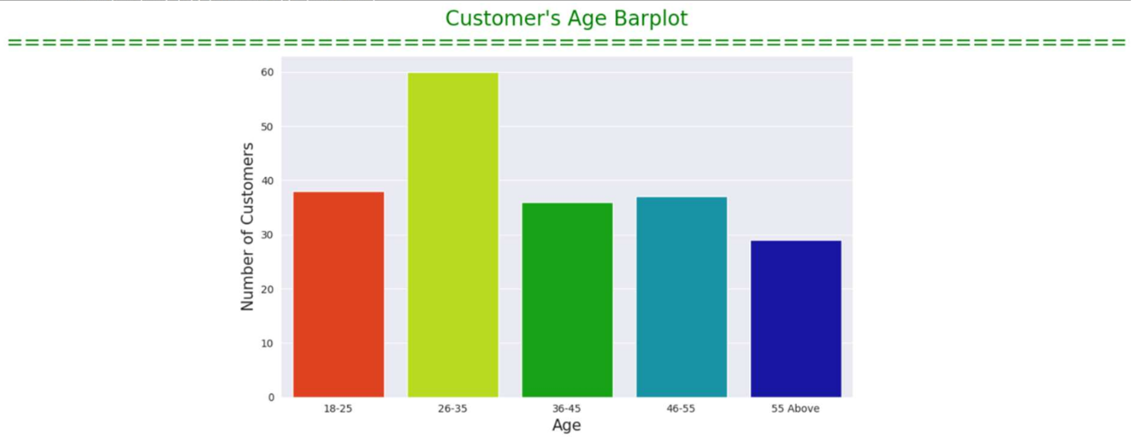
```
plt.figure(figsize=(10,6))
sns.set_style('darkgrid')

sns.scatterplot(data=df, x="Spending Score (1-100)", y= "Annual Income (k$)", hue="Gender", s=60)
plt.title("Spending Score (1-100) VS Annual Income (k$)\n=================================================================", fontsize=20, color="green")
plt.xlabel("Spending Score (1-100)", fontsize=15)
plt.ylabel("Annual Income (k$)", fontsize=15)
plt.show()
```



```
[17] Age_18_25 = df.Age[(df.Age>=18) & (df.Age<=25)]
     Age_26_35 = df.Age[(df.Age>=26) & (df.Age<=35)]
     Age_36_45 = df.Age[(df.Age>=36) & (df.Age<=45)]
     Age_46_55 = df.Age[(df.Age>=46) & (df.Age<=55)]
     Age_55_Above = df.Age[(df.Age>=56)]
```

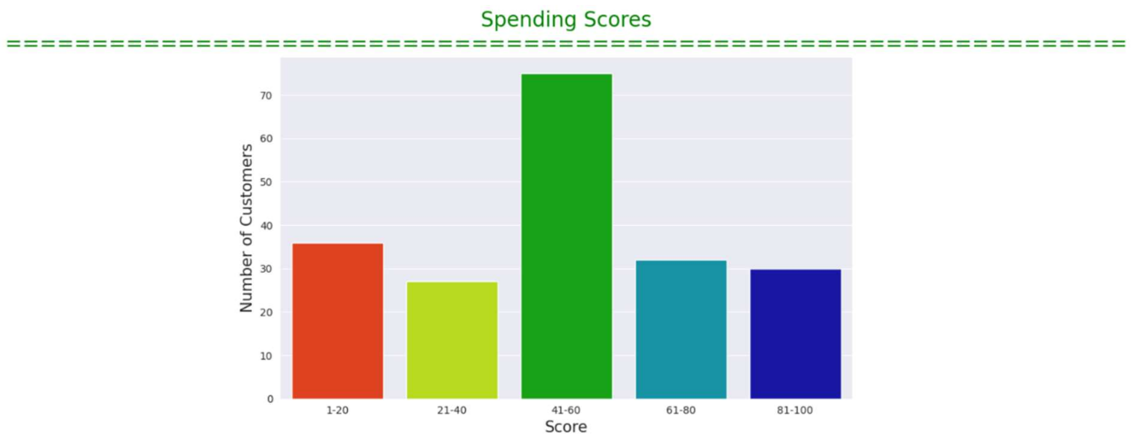
```
[18] x = ["18-25","26-35","36-45","46-55","55 Above"]
     y = [len(Age_18_25.values),len(Age_26_35.values),len(Age_36_45.values),len(Age_46_55.values),len(Age_55_Above.values)]

     plt.figure(figsize=(10,6))
     sns.barplot(x=x, y=y, palette="nipy_spectral_r")
     plt.title("Customer's Age Barplot\n=================================================================", fontsize=20, color="green")
     plt.xlabel("Age", fontsize=15)
     plt.ylabel("Number of Customers", fontsize=15)
     plt.show()
```

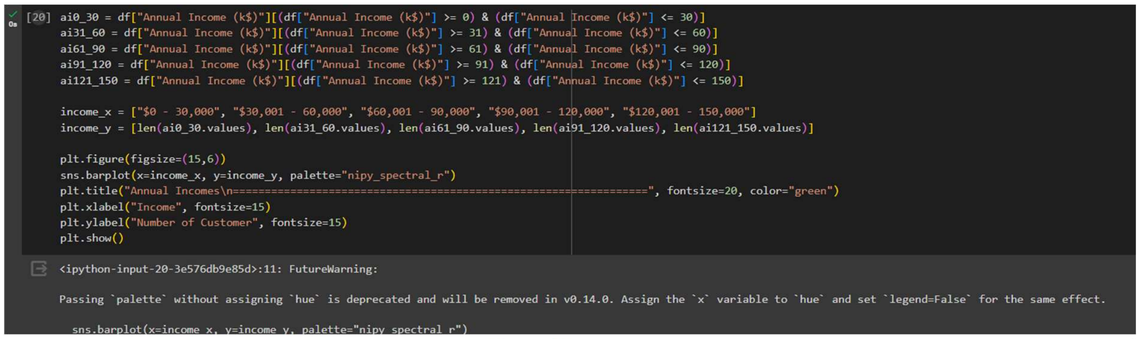
```
<ipython-input-18-1edf9ed38fd9>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=x, y=y, palette="nipy_spectral_r")
```



```
9] ss1_20 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 1) & (df["Spending Score (1-100)"] <= 20)]
   ss21_40 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 21) & (df["Spending Score (1-100)"] <= 40)]
   ss41_60 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 41) & (df["Spending Score (1-100)"] <= 60)]
   ss61_80 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 61) & (df["Spending Score (1-100)"] <= 80)]
   ss81_100 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 81) & (df["Spending Score (1-100)"] <= 100)]

   score_x = ["1-20", "21-40", "41-60", "61-80", "81-100"]
   score_y = [len(ss1_20.values), len(ss21_40.values), len(ss41_60.values), len(ss61_80.values), len(ss81_100.values)]

   plt.figure(figsize=(10,6))
   sns.barplot(x=score_x, y=score_y,palette="nipy_spectral_r")
   plt.title("Spending Scores\n=================================================================", fontsize=20, color="green")
   plt.xlabel("Score", fontsize=15)
   plt.ylabel("Number of Customers", fontsize=15)
   plt.show()
```

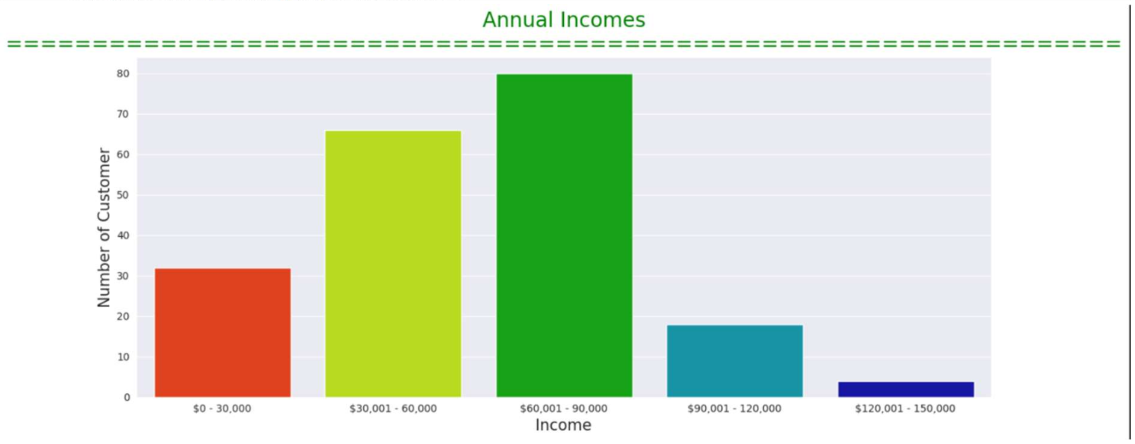
```
<ipython-input-19-8a8243814efe>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=score_x, y=score_y,palette="nipy_spectral_r")
```

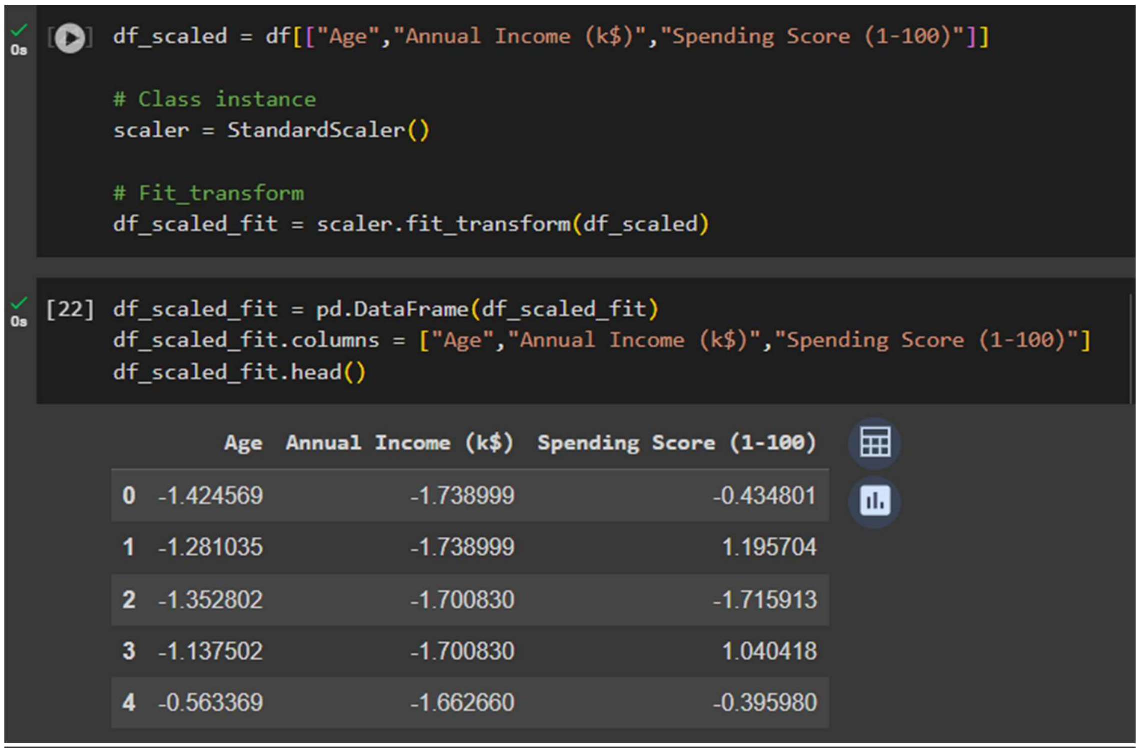




```
[20] ai0_30 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 0) & (df["Annual Income (k$)"] <= 30)]
     ai31_60 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 31) & (df["Annual Income (k$)"] <= 60)]
     ai61_90 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 61) & (df["Annual Income (k$)"] <= 90)]
     ai91_120 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 91) & (df["Annual Income (k$)"] <= 120)]
     ai121_150 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 121) & (df["Annual Income (k$)"] <= 150)]

     income_x = ["$0 - 30,000", "$30,001 - 60,000", "$60,001 - 90,000", "$90,001 - 120,000", "$120,001 - 150,000"]
     income_y = [len(ai0_30.values), len(ai31_60.values), len(ai61_90.values), len(ai91_120.values), len(ai121_150.values)]

     plt.figure(figsize=(15,6))
     sns.barplot(x=income_x, y=income_y, palette="nipy_spectral_r")
     plt.title("Annual Incomes\n==============================================================", fontsize=20, color="green")
     plt.xlabel("Income", fontsize=15)
     plt.ylabel("Number of Customer", fontsize=15)
     plt.show()
```

```
<ipython-input-20-3e576db9e85d>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=income_x, y=income_y, palette="nipy_spectral_r")
```

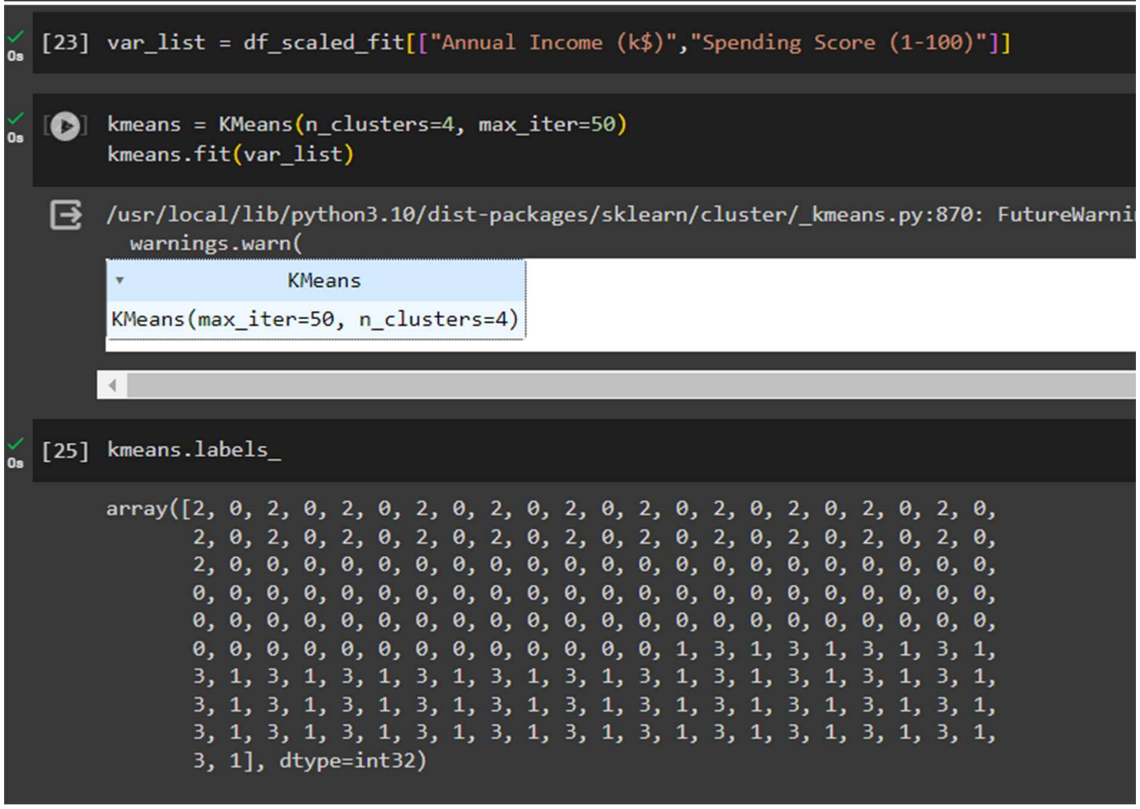




```
df_scaled = df[["Age","Annual Income (k$)","Spending Score (1-100)"]]

# Class instance
scaler = StandardScaler()

# Fit_transform
df_scaled_fit = scaler.fit_transform(df_scaled)
```

```
[22] df_scaled_fit = pd.DataFrame(df_scaled_fit)
     df_scaled_fit.columns = ["Age","Annual Income (k$)","Spending Score (1-100)"]
     df_scaled_fit.head()
```

	Age	Annual Income (k$)	Spending Score (1-100)
0	-1.424569	-1.738999	-0.434801
1	-1.281035	-1.738999	1.195704
2	-1.352802	-1.700830	-1.715913
3	-1.137502	-1.700830	1.040418
4	-0.563369	-1.662660	-0.395980



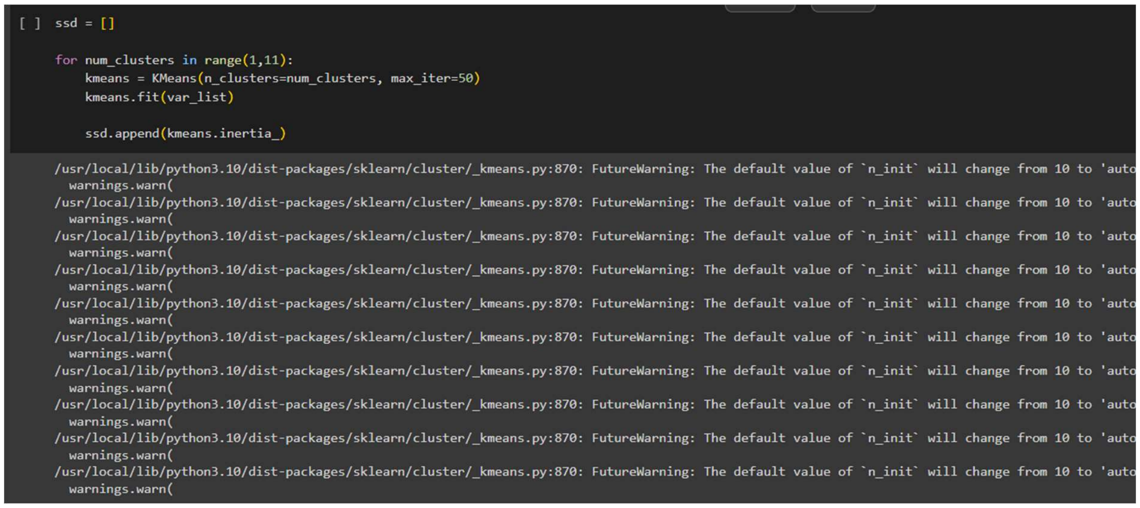
```
[23] var_list = df_scaled_fit[["Annual Income (k$)","Spending Score (1-100)"]]
```

```
kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(var_list)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
KMeans
KMeans(max_iter=50, n_clusters=4)
```

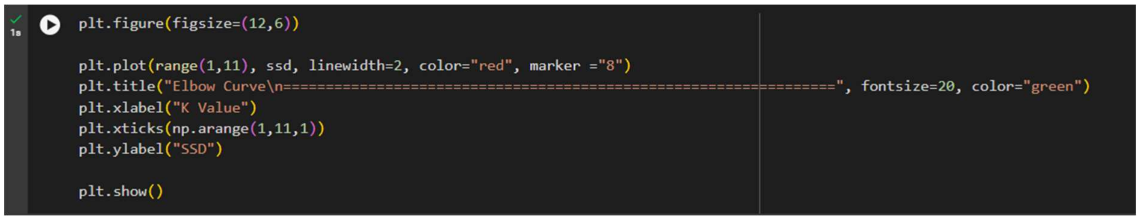
```
[25] kmeans.labels_
```

```
array([2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
       2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
       2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1], dtype=int32)
```

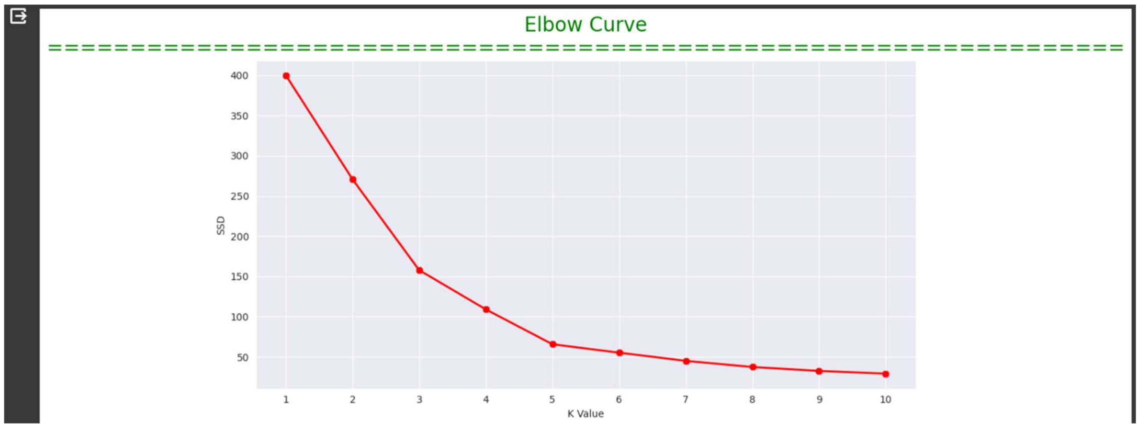


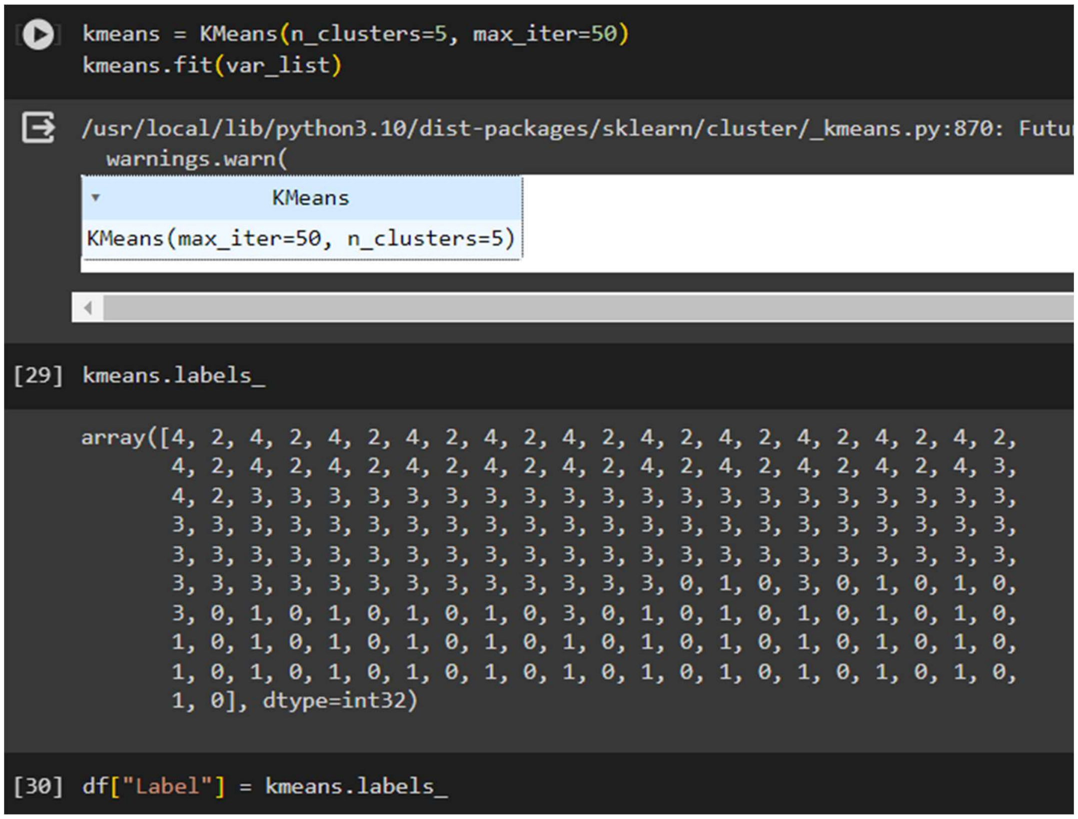
```
[ ]  ssd = []

     for num_clusters in range(1,11):
         kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
         kmeans.fit(var_list)

         ssd.append(kmeans.inertia_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
```



```
plt.figure(figsize=(12,6))

plt.plot(range(1,11), ssd, linewidth=2, color="red", marker ="8")
plt.title("Elbow Curve\n=================================================================", fontsize=20, color="green")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("SSD")

plt.show()
```





```
kmeans = KMeans(n_clusters=5, max_iter=50)
kmeans.fit(var_list)
```

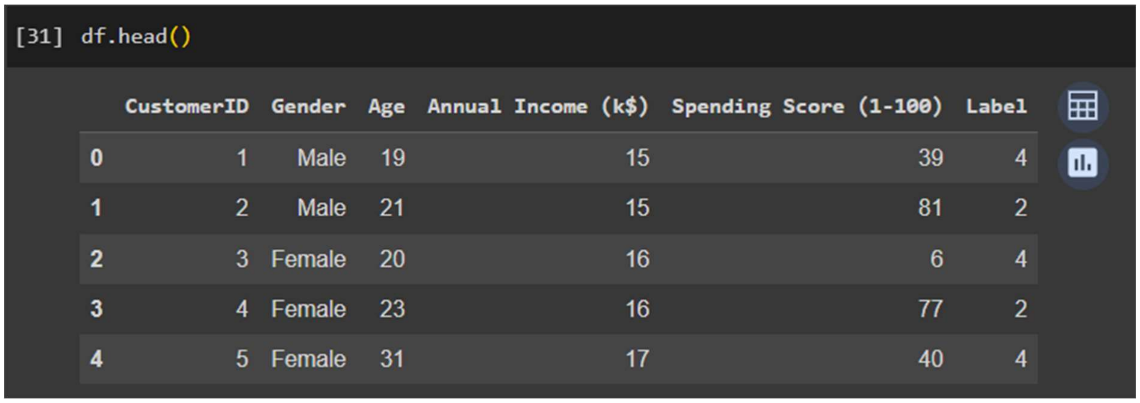
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futu
  warnings.warn(
```

```
KMeans
KMeans(max_iter=50, n_clusters=5)
```

```
[29] kmeans.labels_
```

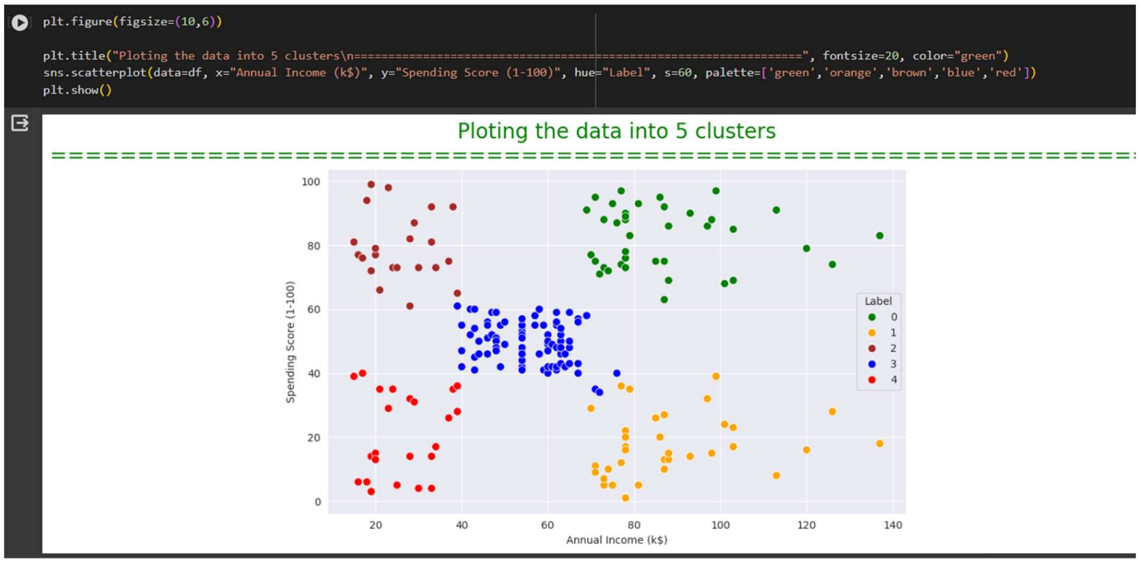
```
array([4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 3,
       4, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 0, 3, 0, 1, 0, 1, 0,
       3, 0, 1, 0, 1, 0, 1, 0, 1, 0, 3, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0], dtype=int32)
```

```
[30] df["Label"] = kmeans.labels_
```



```
[31] df.head()
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Label
0	1	Male	19	15	39	4
1	2	Male	21	15	81	2
2	3	Female	20	16	6	4
3	4	Female	23	16	77	2
4	5	Female	31	17	40	4



```
plt.figure(figsize=(10,6))

plt.title("Ploting the data into 5 clusters\n===================================================================", fontsize=20, color="green")
sns.scatterplot(data=df, x="Annual Income (k$)", y="Spending Score (1-100)", hue="Label", s=60, palette=['green','orange','brown','blue','red'])
plt.show()
```

Results

The elbow curve for the k-means clustering algorithm shows a clear elbow point at k=5, which suggests that using 5 clusters is the optimal choice for clustering our data. This means that dividing our data into 5 distinct groups (clusters) provides a good balance between reducing the sum of squared distances within each cluster and avoiding overfitting (using too many clusters).

Observations

1. Cluster 0 which is at centre, average annual income with average spending score.
2. Cluster 1 which is at top right, highest annual income with highest spending score.
3. Cluster 2 which is at top left, lowest annual income with highest spending score.
4. Cluster 3 which is at bottom right, high annual income with low spending score.
5. Cluster 4 which is at bottom left, lowest annual income with lowest spending score. So high income, high spending are the most beneficial ones to the mall owners which increases the owner's business(Cluster 1) .

Inferences

- The Highest income , high spending can be target these type of customers as they earn more money and spend as much as they want.
- Highest income, low spending can be target these type of customers by asking feedback and advertising the product in a better way.
- Average income,Average spending may or may not be beneficial to the mall owners of this type of customers.
- Low income, High spending can be target these type of customers by providing them with low-cost EMI's etc.
- Low income, Low spending don't target these type of customers because they earn a bit and spend some amount of money.

ADVANTAGES OF K-MEANS for this dataset

1. Customer segmentation using K-means can provide valuable insights into customer behavior and preferences. This information can be used for targeted marketing campaigns, product recommendations, and personalized customer experiences.
2. K-means can effectively identify distinct groups of customers within the dataset. For example, it can identify segments such as young, high-spending females or middle-aged, low-spending males.
3. K-means clustering can be re-run periodically to accommodate changes in customer behavior and preferences over time. This allows for dynamic and adaptive customer segmentation strategies.

Future Considerations

- Dynamic Clustering: Implementing dynamic clustering to adapt to changing customer behavior and preferences over time.
- Integration with CRM: Integrating the clustering results with Customer Relationship Management (CRM) systems to personalize marketing campaigns and promotions.
- Feedback Loop: Establishing a feedback loop to continuously refine the clustering model based on customer feedback and new data.

- Advanced Techniques: Exploring advanced clustering techniques like hierarchical clustering or density-based clustering for more complex segmentation requirements.

Conclusion

Implementing K-means clustering for customer segmentation of Mall data can provide valuable insights into customer behavior and preferences. By segmenting customers into distinct groups, malls can tailor their marketing strategies and offerings to better meet the needs of each segment, ultimately improving customer satisfaction and loyalty. However, it's important to regularly review and update the clustering model to ensure its effectiveness in capturing changing trends and customer preferences.