CS5542 BIG DATA APPS AND ANALYTICS

In Class Programming – 7

k-means clustering

Kalyani Nikure
Kmn6ng@umkc.edu

Table of Contents

Description

Use a different dataset (than the one we used in class) and use the model provided in ICP7 to perform clustering. You must try 5 different number of clusters (for example n_clusters = 5 or n_clusters =6,7,8, or 9 etc.) based on elbow curve and for each cluster visualize the clustering results and report your findings in detail. Also provide the elbow curve screen shots in your report.

Detailed Steps Explanation

1. K-means clustering using code already provided in the source code

- Importing all the required dependencies

```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.cluster import KMeans
import sklearn.cluster as cluster
import sklearn.metrics as metrics

import warnings
warnings.filterwarnings("ignore")
```

- About dataset
  The Mall Customers dataset contains the basic information (ID, age, gender, income, spending score) about the customers. It has 200 records and 5 columns.

Download Source: https://www.kaggle.com/vjchoudhary7/customer-segmentation-tutorial-in-python

- Import Data by loading the data from csv file

```
# reading the data and looking at the first fifteen rows of the data
data=pd.read_csv("/content/Mall_Customers.csv")
data.head(15)
```

	CustomerID	Gender	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	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13

- Rename columns as to short names for easy usability

```
data.rename(columns={'Annual Income (k$)' : 'Income', 'Spending Score (1-100)' : 'Spending_Score'}, inplace = True)
data.head()
```

	CustomerID	Gender	Age	Income	Spending_Score
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

- Data Statistics to understand how data is distributed in each column

```
# statistics of the data
data.describe()
```

	CustomerID	Age	Income	Spending_Score
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

- K-means clustering using the existing code

```
[21] df_Short = data[['Age', 'Spending_Score','Income']]
```

Since K-Means is a distance-based algorithm, this difference of magnitude can create a problem. So let's first bring all the variables to the same magnitude:

```
[22] # standardizing the data
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     data_scaled = scaler.fit_transform(df_Short)

     # statistics of scaled data
     data_s=pd.DataFrame(data_scaled).describe()
     data_s
```

	0	1	2
count	2.000000e+02	2.000000e+02	2.000000e+02
mean	-9.603429e-17	-1.121325e-16	-6.128431e-16
std	1.002509e+00	1.002509e+00	1.002509e+00
min	-1.496335e+00	-1.910021e+00	-1.738999e+00
25%	-7.248436e-01	-5.997931e-01	-7.275093e-01
50%	-2.045351e-01	-7.764312e-03	3.587926e-02
75%	7.284319e-01	8.851316e-01	6.656748e-01
max	2.235532e+00	1.894492e+00	2.917671e+00

The magnitude looks similar now. Next, let's create a kmeans function and fit it on the data:

```
[23] # defining the kmeans function with initialization as k-means++
     kmeans = KMeans(n_clusters=2, init='k-means++')

     # fitting the k means algorithm on scaled data
     kmeans.fit(data_scaled)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

We have initialized two clusters and evaluate how well the formed clusters are. To do that, we will calculate the inertia of the clusters:

```
[24] # inertia on the fitted data
     kmeans.inertia_
```

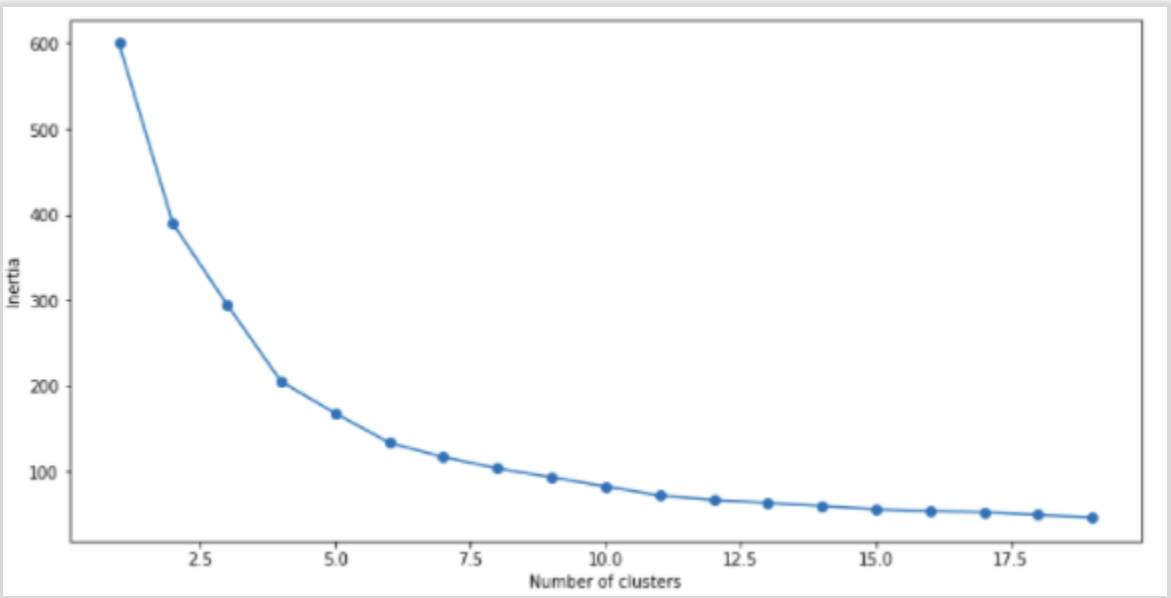
```
389.3861889564372
```

We will first fit multiple k-means models and in each successive model, we will increase the number of clusters. We will store the inertia value of each model and then plot it to visualize the result:

```
[25] # fitting multiple k-means algorithms and storing the values in an empty list
     SSE = []
     for cluster in range(1,20):
         kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
         kmeans.fit(data_scaled)
         SSE.append(kmeans.inertia_)

     # converting the results into a dataframe and plotting them
     frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
     plt.figure(figsize=(12,6))
     plt.plot(frame['Cluster'], frame['SSE'], marker='o')
     plt.xlabel('Number of clusters')
     plt.ylabel('Inertia')
```

- The elbow plot of k values from 1 to 20



2. K-means clustering after performing different values of k and evaluating it for best value

- Import the data and rename the columns

```
df = pd.read_csv("/content/Mall_Customers.csv")
df.head()
```

	CustomerID	Gender	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

```
[30] df.rename(columns={'Annual Income (k$)' : 'Income', 'Spending Score (1-100)' : 'Spending_Score'}, inplace = True)
     df.head()
```

	CustomerID	Gender	Age	Income	Spending_Score
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

2.1 Segmentation using Age and Spending Score

- To implement k-means clustering, use elbow method to identify the clusters and its inertia values.

Elbow Method to Identify Clusters

Run Cluster Analysis 10 times

Selecting N Clusters based in Inertia (Squared Distance between Centroids and data points, should be less)

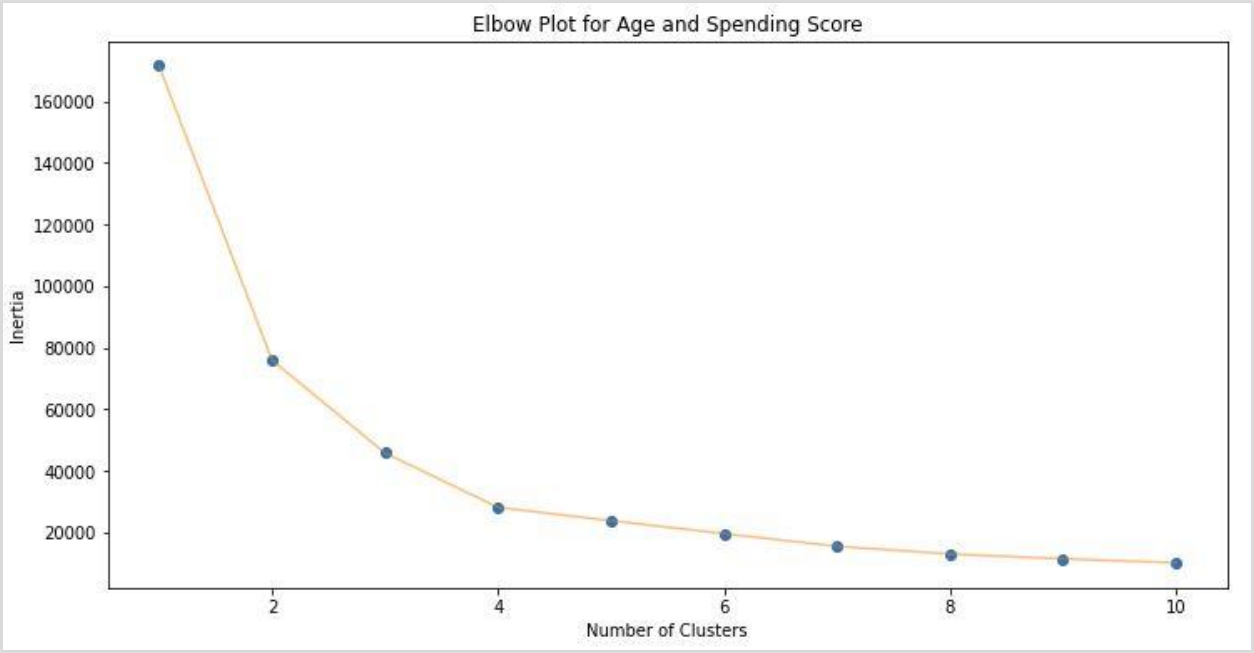
- We run the Cluster Analysis using Cluster as 1 till 10. Also, we store the WSS Scores. The WSS score will be used to create the Elbow Plot
- WSS = Within-Cluster-Sum of Squared

```
'''Age and spending Score'''
X1 = df[['Age' , 'Spending_Score']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10, max_iter=300,
                        tol=0.0001, random_state= 111, algorithm='elkan') )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

Elbow Plot

```
[32] plt.figure(1 , figsize = (12 ,6))
     plt.plot(np.arange(1 , 11) , inertia , 'o')
     plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
     plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
     plt.title("Elbow Plot for Age and Spending Score")
     plt.show()
```

- Below is the elbow plot for the best clustering achieved over the data



- Implement Silhouette method to get the score values for each cluster value. The coeffient varies between -1 to 1. The Maximum scores gives the best clustering technique.

```
for i in range(3,11):
    labels= cluster.KMeans(n_clusters=i, init="k-means++", random_state=200).fit(X1).labels_
    print ("Silhouette score for k(clusters) = "+str(i)+" is "
           +str(metrics.silhouette_score(X1, labels,metric="euclidean", sample_size=1000, random_state=200)))
```

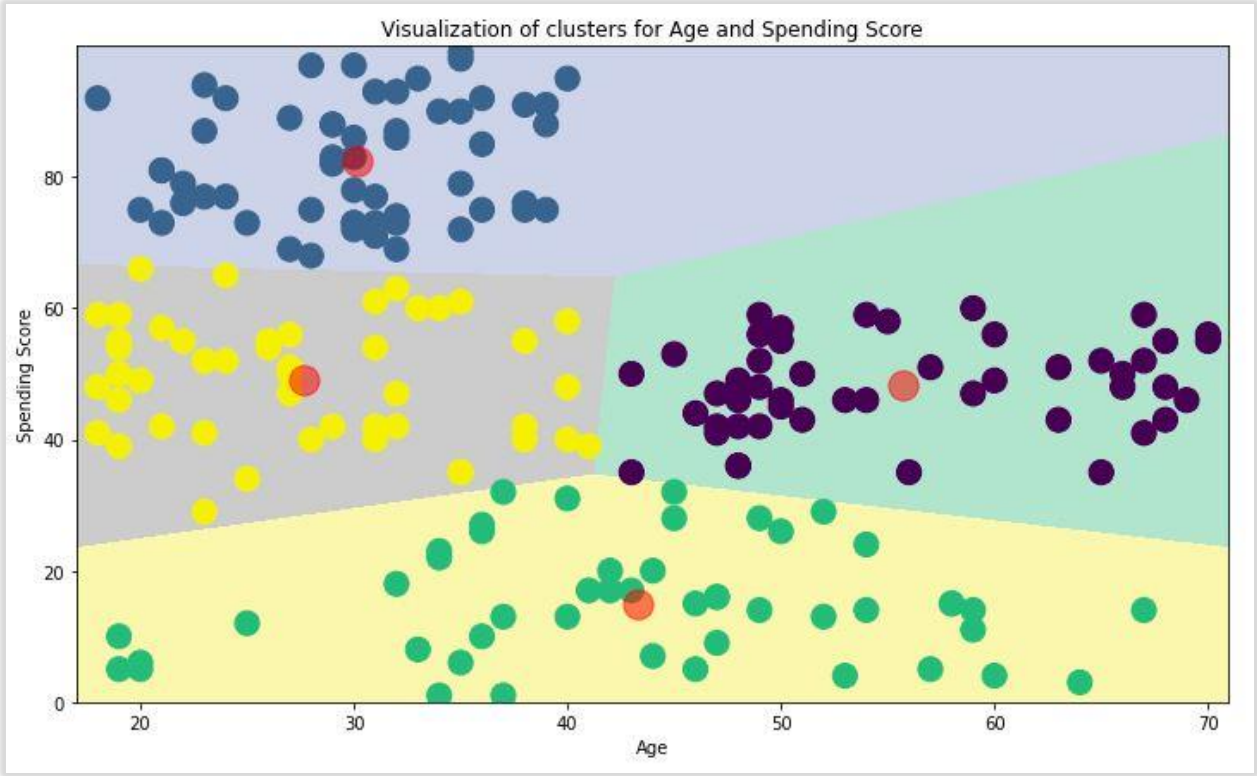
```
Silhouette score for k(clusters) = 3 is 0.4530012701652126
Silhouette score for k(clusters) = 4 is 0.49973941540141753
Silhouette score for k(clusters) = 5 is 0.44526667994351227
Silhouette score for k(clusters) = 6 is 0.43906811854494876
Silhouette score for k(clusters) = 7 is 0.42313509747504796
Silhouette score for k(clusters) = 8 is 0.4305548672407991
Silhouette score for k(clusters) = 9 is 0.41038847746650914
Silhouette score for k(clusters) = 10 is 0.4150825941572905
```

From above values of Silhouette Scores, it is cleara that k=4 has the best score of all.

- Now, refering to above Silhouette Scores, we see that n_clusters= 4 is the best value. Implement k-means clustering using the same value to get best results.

```
algorithm = (KMeans(n_clusters = 4 ,init='k-means++', n_init = 10 ,max_iter=300,
                    tol=0.0001,  random_state= 111  , algorithm='elkan') )
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

- Below is the screenshot of the Visualization the 4 clusters. I have also plotted the centroids of each cluster.



2.2 Segmentation using Annual Income and Spending Score

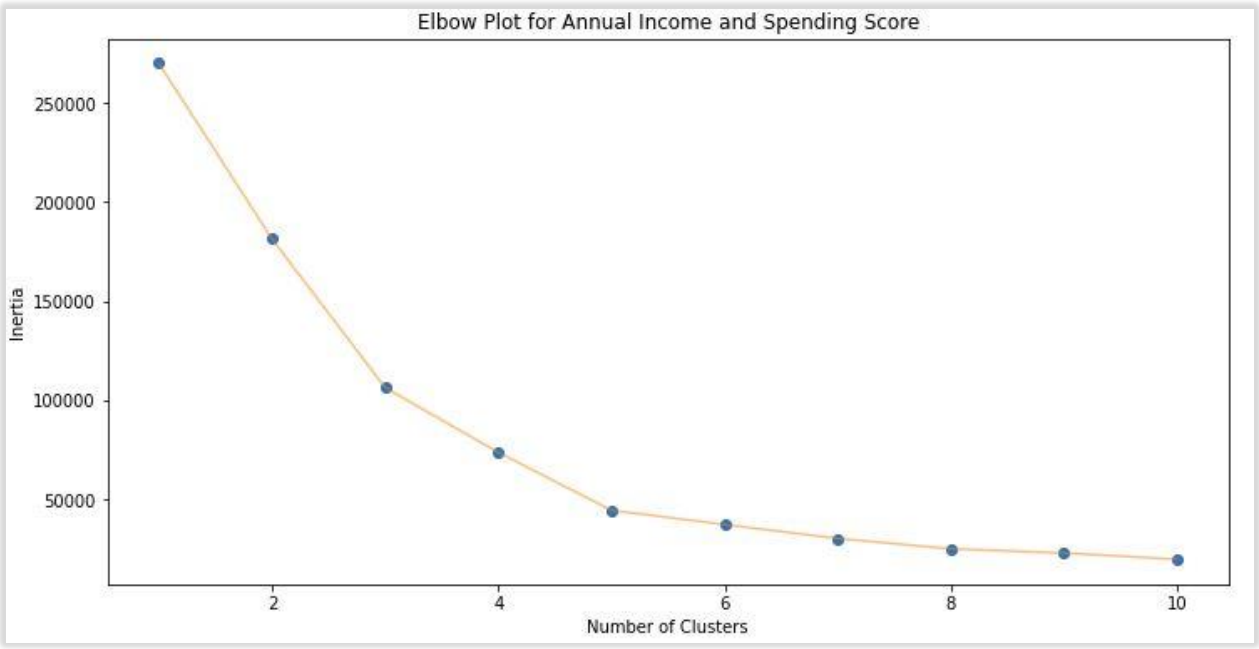
- To implement k-means clustering, use elbow method to identify the clusters and its inertia values.

```
'''Annual Income and spending Score'''
X2 = df[['Income' , 'Spending_Score']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X2)
    inertia.append(algorithm.inertia_)
```

Elbow Method to Identify Clusters

```
plt.figure(1 , figsize = (12 ,6))
plt.plot(np.arange(1 , 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.title("Elbow Plot for Annual Income and Spending Score")
plt.show()
```

- Below is the elbow plot for the best clustering achieved over the data



- Implement Silhouette method to get the score values for each cluster value. The coeffient varies between -1 to 1. The Maximum scores gives the best clustering technique.

```
for i in range(3,11):
    labels=cluster.KMeans(n_clusters=i, init="k-means++", random_state=200).fit(X2).labels_
    print ("Silhouette score for k(clusters) = "+str(i)+" is "
           +str(metrics.silhouette_score(X2, labels,metric="euclidean", sample_size=1000, random_state=200)))
```

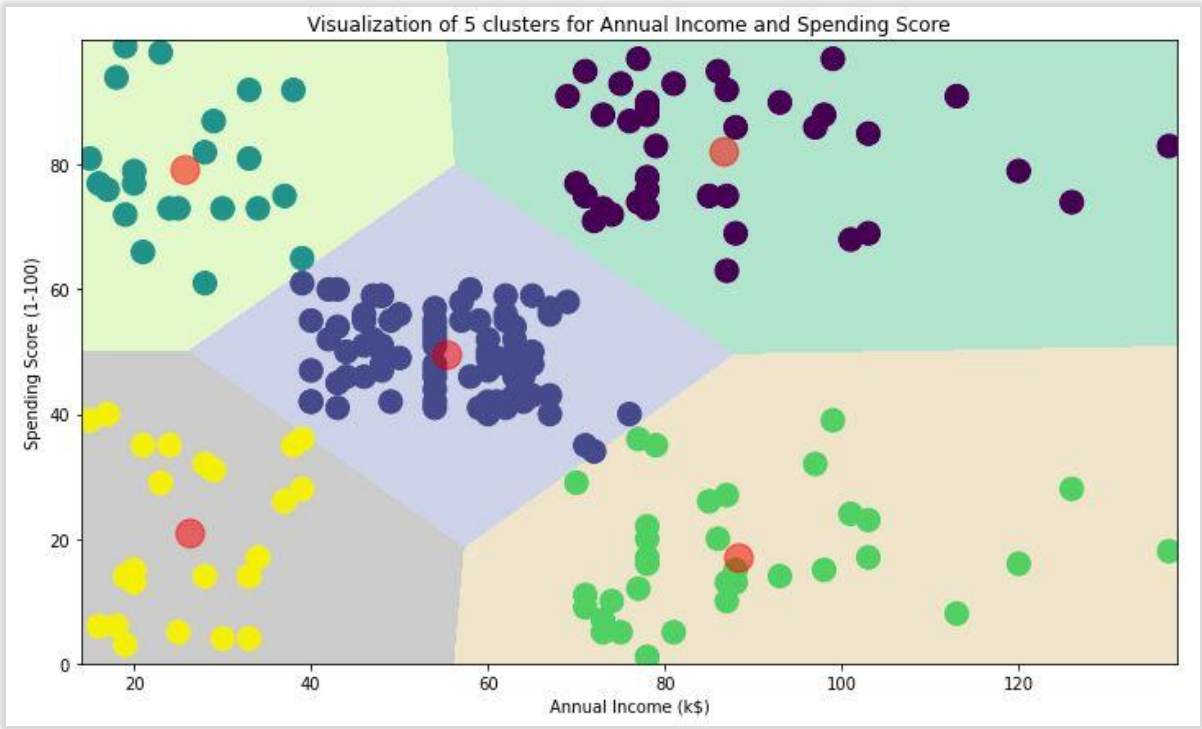
```
Silhouette score for k(clusters) = 3 is 0.46761358158775423
Silhouette score for k(clusters) = 4 is 0.4931963109249047
Silhouette score for k(clusters) = 5 is 0.553931997444648
Silhouette score for k(clusters) = 6 is 0.53976103063432
Silhouette score for k(clusters) = 7 is 0.5288104473798049
Silhouette score for k(clusters) = 8 is 0.4572211842776841
Silhouette score for k(clusters) = 9 is 0.45819645551960536
Silhouette score for k(clusters) = 10 is 0.45275118302579015
```

From above values of Silhouette Scores, it is cleara that k=5 has the best score of all.

- Now, refering to above Silhouette Scores, we see that n_clusters= 5 is the best value. Implement k-means clustering using the same value to get best results.

```
[42] algorithm = (KMeans(n_clusters = 5 ,init='k-means++', n_init = 10 ,max_iter=300,
                         tol=0.0001,  random_state= 111  , algorithm='elkan') )
     algorithm.fit(X2)
     labels2 = algorithm.labels_
     centroids2 = algorithm.cluster_centers_
```

- Below is the screenshot of the Visualization the 5 clusters. I have also plotted the centroids of each cluster.



2.3 Segmentation using Age, Annual Income and Spending Score

- To implement k-means clustering, use elbow method to identify the clusters and its inertia values.

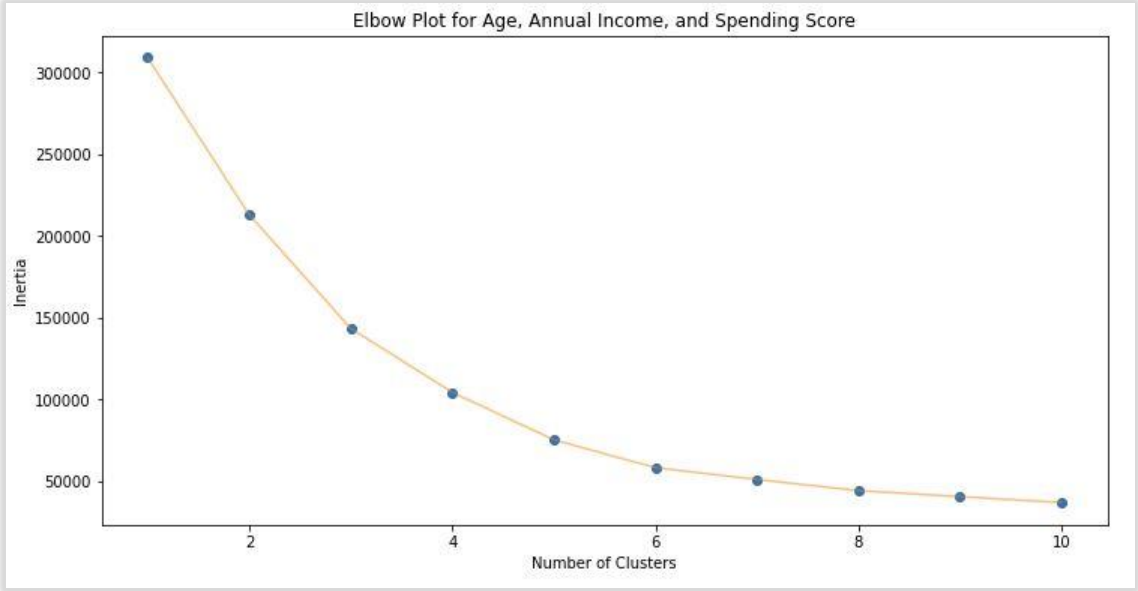
```
'''Age, Annual Income, and spending Score'''

X3 = df[['Age' , 'Income' ,'Spending_Score']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X3)
    inertia.append(algorithm.inertia_)
```

Elbow Method to Indetify Clusters

```
[5]  plt.figure(1 , figsize = (12 ,6))
     plt.plot(np.arange(1 , 11) , inertia , 'o')
     plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
     plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
     plt.title("Elbow Plot for Age, Annual Income, and Spending Score")
     plt.show()
```

- Below is the elbow plot for the best clustering achieved over the data



- Implement Silhouette method to get the score values for each cluster value. The coeffient varies between -1 to 1. The Maximum scores gives the best clustering technique.

```
[46] for i in range(3,11):
         labels=cluster.KMeans(n_clusters=i, init="k-means++", random_state=200).fit(X3).labels_
         print ("Silhouette score for k(clusters) = "+str(i)+" is "
                +str(metrics.silhouette_score(X3, labels,metric="euclidean", sample_size=1000, random_state=200)))
```

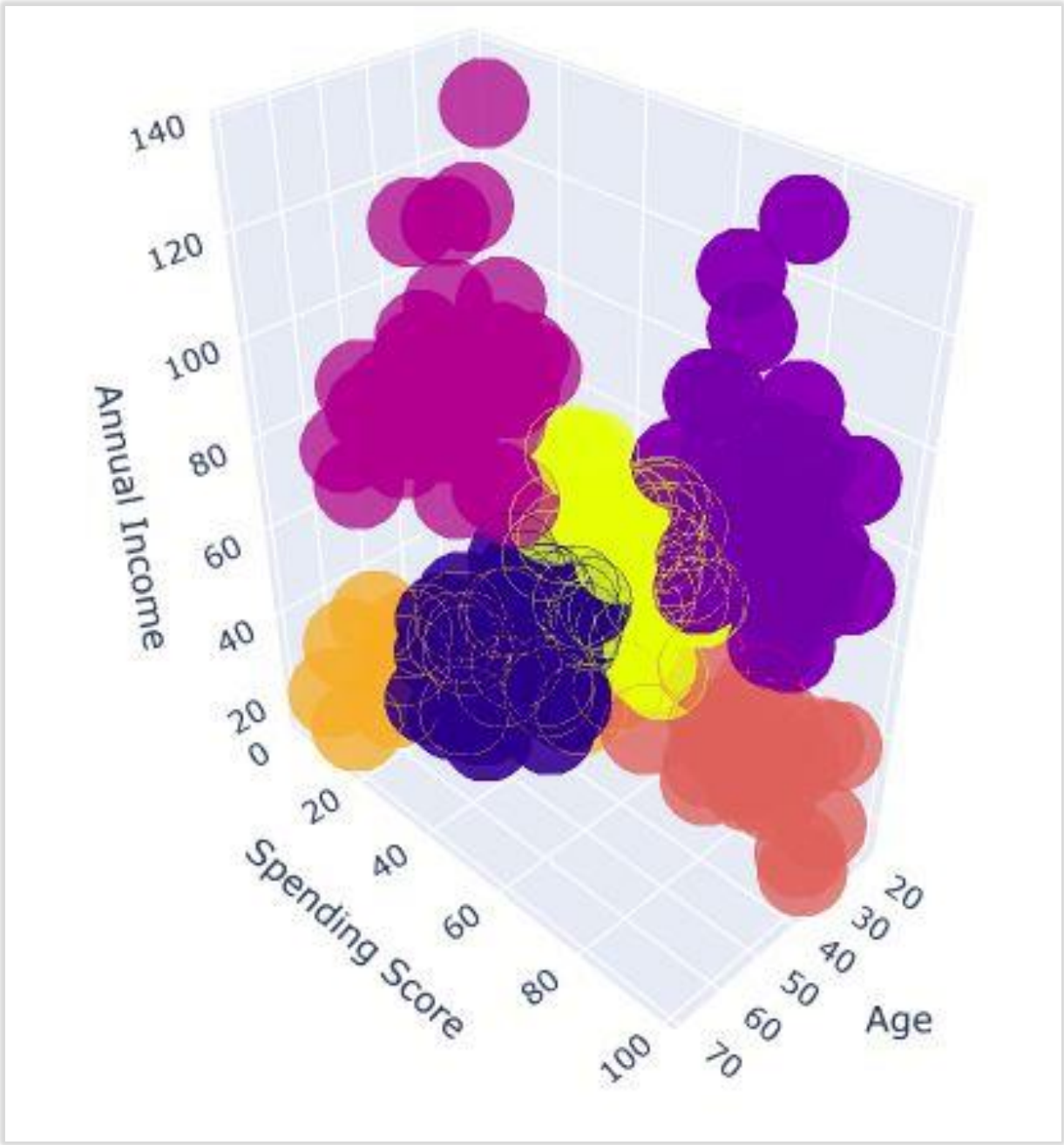
```
Silhouette score for k(clusters) = 3 is 0.3839349967742105
Silhouette score for k(clusters) = 4 is 0.4054630207773331
Silhouette score for k(clusters) = 5 is 0.44428597560893024
Silhouette score for k(clusters) = 6 is 0.45234439477240534
Silhouette score for k(clusters) = 7 is 0.44125523526699084
Silhouette score for k(clusters) = 8 is 0.42786362446871096
Silhouette score for k(clusters) = 9 is 0.4096301136432198
Silhouette score for k(clusters) = 10 is 0.37672414426446066
```

From above values of Silhouette Scores, it is cleara that k=6 has the best score of all.

- Now, refering to above Silhouette Scores, we see that n_clusters= 6 is the best value. Implement k-means clustering using the same value to get best results.

```
[47] algorithm = (KMeans(n_clusters = 6 ,init='k-means++', n_init = 10 ,max_iter=300,
                         tol=0.0001,  random_state= 111  , algorithm='elkan') )
     algorithm.fit(X3)
     labels3 = algorithm.labels_
     centroids3 = algorithm.cluster_centers_
```

- Below is the screenshot of the Visualization the 6 clusters. I have also plotted the centroids of each cluster.



3. Final observation comments

We saw that k-means clustering improved after I made changes to KMeans() function like added few more parameters.

```
algorithm = KMeans(n_clusters = n,
            init='k-means++', n_init = 10, max_iter=300,
            tol=0.0001, random_state= 111, algorithm='elkan')
```

- `algorithm='elkan'`
  Use of Elkan's algorithm which accelerates k-means **by avoiding redundant distance calculations**. Elkan's algorithm reduces the number of distance calculations in practice closer to n.
- `n_init = 10`
  Number of times the k-means algorithm will be run with different centroid seeds. The results will be the best output of n_init consecutive runs in terms of inertia.
- `random_state= 111`
  Used a constant number of random_state every time so that it generates results with integrity.
- `tol=0.0001`
  Tolerance - A value to see if algorithm has converged. If the error is greater than the give tolerance value, continue running the algorithm until it gets below the tolerance value.
  Maximum Iterations - Maximum number of times we need to run the algorithm until the error gets below the give tolerance value.
- Silhouette Scores helped us to take the specific value of cluster using the highest value among all. This was the best evaluation of clustering and get the right number of clusters.

Video Link

- https://youtu.be/5fljWr0tPNg

Conclusion

1. Lessons Learnt

- I developed a good understanding of the k-means clustering technique which is mostly suited for unsupervised datasets.
- The understood how to find the best number of clusters for a given dataset using elbow method and silhouette scores for different values of k.

2. Challenges Faced

- To get the best number of clusters, I searched for techniques and criterions to decide.
- I found that silhouette scores can be used to evaluate a particular clustering technique.