BDAA-ICP7-wiki report : Yousef Almutairi
Date: 10/18/2021

1. <u>What you learned in the ICP:</u>

   I learned to segment the customers of a Mall customers dataset based on their annual income and spending score. Spending Score is something you assign to the customer based on your defined parameters like customer behavior and purchasing data. Also, I learned how to use K means algorithm to groups all unlabeled dataset into different clusters.

2. <u>Screen shots that shows the successful execution of each required step of your code:</u>

   Read the dataset and remove the 'Genre' column in order to have only float dataset.

[92]
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
```

Dataset : https://www.kaggle.com/kandij/mall-customers

[93]
```
# reading the data and looking at the first five rows of the data
orginial_data=pd.read_csv("Mall_Customers.csv")
#drop Gender column
data = orginial_data.drop(['Genre'], axis=1)
```

```
data.head()
```

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

Then as a part to ensure the dataset is cleaned and readable, I removed all NULL and NA values. In addition, brought all the variables to the same magnitude using the StandardScaler.

```
[96] #remove null values
     data =data[~data.isin([np.nan, np.inf, -np.inf]).any(1)]
```

Here, we see that there is a lot of variation in the magnitude of the data. Variables like Channel and Region have low magnitude whereas variables like Fresh, Milk, Grocery, etc. have a higher magnitude.

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:

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

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

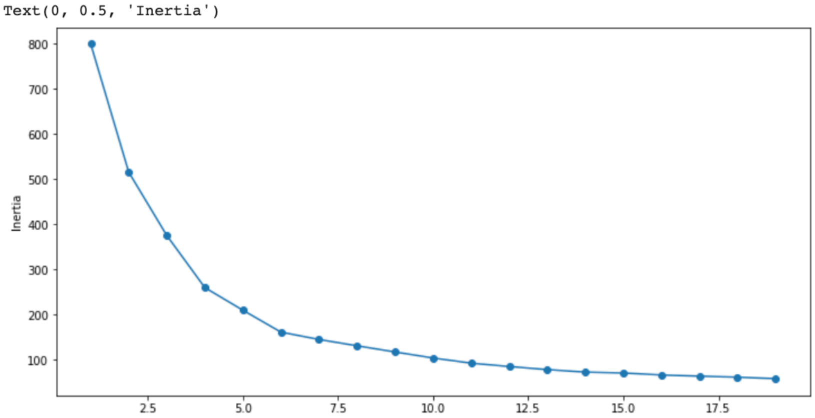
	0	1	2	3
count	2.000000e+02	2.000000e+02	2.000000e+02	2.000000e+02
mean	-6.661338e-18	-9.603429e-17	-6.128431e-16	-1.121325e-16
std	1.002509e+00	1.002509e+00	1.002509e+00	1.002509e+00
min	-1.723412e+00	-1.496335e+00	-1.738999e+00	-1.910021e+00
25%	-8.617060e-01	-7.248436e-01	-7.275093e-01	-5.997931e-01
50%	0.000000e+00	-2.045351e-01	3.587926e-02	-7.764312e-03
75%	8.617060e-01	7.284319e-01	6.656748e-01	8.851316e-01
max	1.723412e+00	2.235532e+00	2.917671e+00	1.894492e+00

Plot the elbow curve to determine the optimum number of clusters.

```
[100] # 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')
```

```
Text(0, 0.5, 'Inertia')
```



BDAA-ICP7-wiki report : Yousef Almutairi
Date: 10/18/2021

I started with 4 clusters and print out the value numbers and prediction of the clusters.

```
[101] # k means using 5 clusters and k-means++ initialization
      kmeans = KMeans(n_jobs = -1, n_clusters = 4, init='k-means++')
      kmeans.fit(data_scaled)
      pred = kmeans.predict(data_scaled)
```

Finally, let's look at the value count of points in each of the above-formed clusters:

```
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

```
1    60
0    59
2    41
3    40
Name: cluster, dtype: int64
```

So, there are 234 data points belonging to cluster 4 (index 3), then 125 points in cluster 2 (index 1), and so on. This is how we can implement K-Means Clustering in Python.

```
[103] frame
```

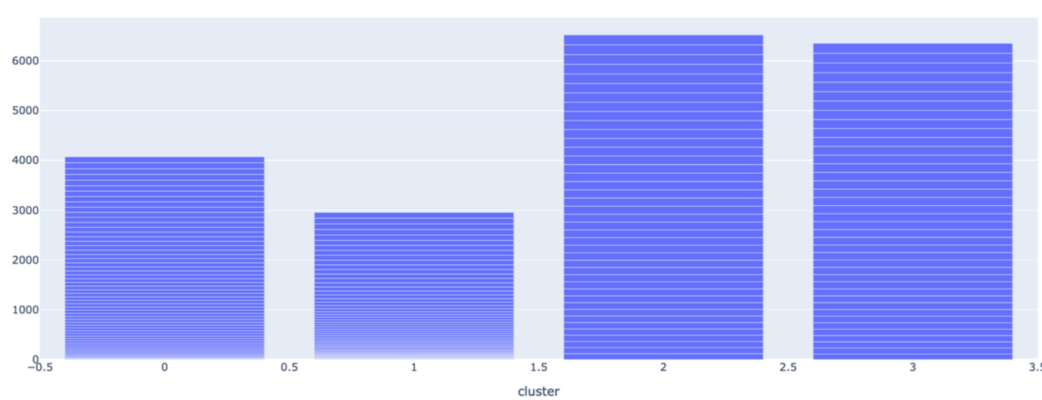
	0	1	2	3	cluster
0	-1.723412	-1.424569	-1.738999	-0.434801	1
1	-1.706091	-1.281035	-1.738999	1.195704	1
2	-1.688771	-1.352802	-1.700830	-1.715913	1
3	-1.671450	-1.137502	-1.700830	1.040418	1
4	-1.654129	-0.563369	-1.662660	-0.395980	1
...	...	...	...	...	...
195	1.654129	-0.276302	2.268791	1.118061	2
196	1.671450	0.441365	2.497807	-0.861839	3
197	1.688771	-0.491602	2.497807	0.923953	2
198	1.706091	-0.491602	2.917671	-1.250054	3
199	1.723412	-0.635135	2.917671	1.273347	2

200 rows × 5 columns

Then, plot the result and prediction score of the cluster.

```
import plotly.express as px
fig = px.bar(frame, x='cluster')
fig.show()

# predict the cluster
from sklearn import metrics
score = metrics.silhouette_score(frame, pred)
print('cluster prediction: ', score)
```




```
cluster prediction:  0.4774952527322249
```

BDAA-ICP7-wiki report : Yousef Almutairi
Date: 10/18/2021

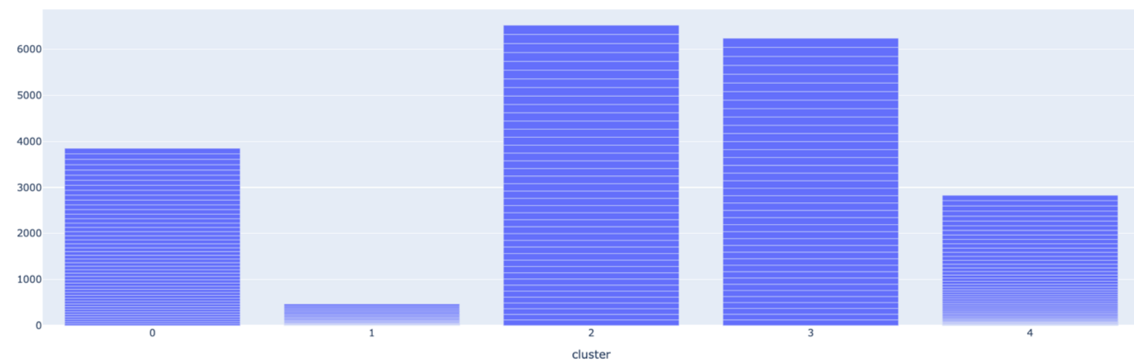
As shown in the screenshot below, increased the cluster to n_clusters= 5

```
[105] # clusters 5
kmeans5 = KMeans(n_jobs = -1, n_clusters = 5, init='k-means++')
kmeans5.fit(data_scaled)
pred5 = kmeans5.predict(data_scaled)
frame5 = pd.DataFrame(data_scaled)
frame5['cluster'] = pred5
frame5['cluster'].value_counts()
```

```
4    52
0    47
2    41
3    39
1    21
Name: cluster, dtype: int64
```

```
#Visualize it
fig = px.bar(frame5, x='cluster')
fig.show()

# predict the cluster
score5 = metrics.silhouette_score(frame5, pred5)
print('cluster prediction: ', score5)
```




```
cluster prediction:  0.5391675319585535
```

BDAA-ICP7-wiki report : Yousef Almutairi
Date: 10/18/2021

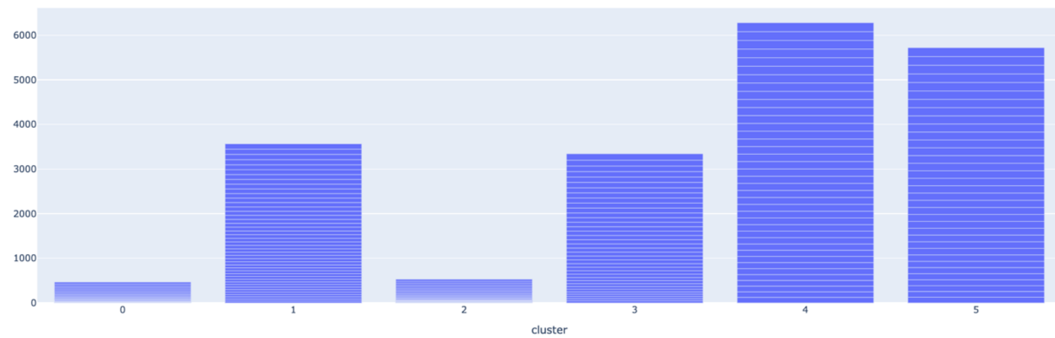
n_clusters= 6

```
[107] #  cluster 6
kmeans6 = KMeans(n_jobs = -1, n_clusters = 6, init='k-means++')
kmeans6.fit(data_scaled)
pred6 = kmeans6.predict(data_scaled)
frame6 = pd.DataFrame(data_scaled)
frame6['cluster'] = pred6
frame6['cluster'].value_counts()
```

```
1    44
4    39
3    37
5    35
2    24
0    21
Name: cluster, dtype: int64
```

```
[108] #Visualize
fig = px.bar(frame6, x='cluster')
fig.show()

# predict the cluster
score6 = metrics.silhouette_score(frame6, pred6)
print('cluster prediction: ', score6)
```




```
cluster prediction:  0.5348927420131205
```

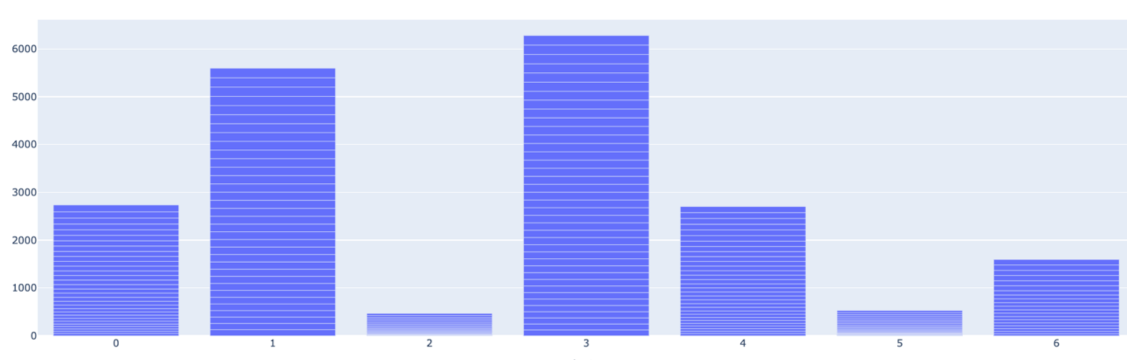
n_clusters= 7

```
# clusters 7
kmeans7 = KMeans(n_jobs = -1, n_clusters = 7, init='k-means++')
kmeans7.fit(data_scaled)
pred7 = kmeans7.predict(data_scaled)
frame7 = pd.DataFrame(data_scaled)
frame7['cluster'] = pred7
frame7['cluster'].value_counts()
```

```
3    39
1    34
4    31
0    31
5    24
2    21
6    20
Name: cluster, dtype: int64
```

```
#Visualize
fig = px.bar(frame7, x='cluster')
fig.show()

# predict the cluster
score7 = metrics.silhouette_score(frame7, pred7)
print('cluster prediction: ', score7)
```




```
cluster prediction:  0.6001021030193721
```

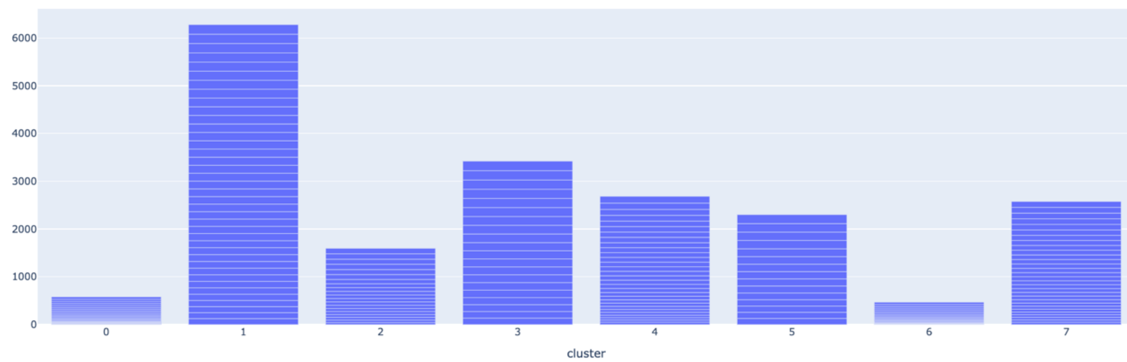
n_clusters= 8

```
# Cluster 8
kmeans8 = KMeans(n_jobs = -1, n_clusters = 8, init='k-means++')
kmeans8.fit(data_scaled)
pred8 = kmeans8.predict(data_scaled)
frame8 = pd.DataFrame(data_scaled)
frame8['cluster'] = pred8
frame8['cluster'].value_counts()
```

```
1    39
7    30
4    30
0    25
6    21
3    20
2    20
5    15
Name: cluster, dtype: int64
```

```
#Visualize it
fig = px.bar(frame8, x='cluster')
fig.show()

# predict the cluster
score8 = metrics.silhouette_score(frame8, pred8)
print('cluster prediction: ', score8)
```




```
cluster prediction:  0.6227910411782872
```

3. <u>Conclusion:</u>

I have Successfully executed the code and made clusters with different values of K between 4 - 8. In order to determine the goodness of the cluster, I used the silhouette score for each cluster. For the silhouette metric, the result must be between 1 and -1 which positive result means all samples fit in the correct cluster. However, in my work above, I noticed that the more clusters I have, the better score I got. I meant that less score means that cluster might have received wrong samples/info.

4. <u>Video link:</u>

https://drive.google.com/file/d/1t1HEyukfx4jJdpIDJkZx1t2goahUwrGH/view?usp=sharing