Customer Segmentation using K-means clustering algorithm

Customer Segmentation is the subdivision of a market into discrete customer groups that share similar characteristics. Customer Segmentation can be a powerful means to identify unsatisfied customer needs. Using the above data companies can then outperform the competition by developing uniquely appealing products and services.

The most common ways in which businesses segment their customer base are:

1. **Demographic information**, such as gender, age, familial and marital status, income, education, and occupation.
2. **Geographical information**, which differs depending on the scope of the company. For localized businesses, this info might pertain to specific towns or counties. For larger companies, it might mean a customer's city, state, or even country of residence.
3. **Psychographics**, such as social class, lifestyle, and personality traits.
4. **Behavioral data**, such as spending and consumption habits, product/service usage, and desired benefits.

Advantages of Customer Segmentation

1. Determine appropriate product pricing.
2. Develop customized marketing campaigns.
3. Design an optimal distribution strategy.
4. Choose specific product features for deployment.
5. Prioritize new product development efforts.

The Challenge

Suppose we are owing a supermarket mall and through membership cards, we have some basic data about our customers like Customer ID, age, gender, annual income and spending score. We want to understand the customers like who are the target customers so that the sense can be given to marketing team and plan the strategy accordingly.

Data

This project is a part of the Mall Customer Segmentation Data competition held on Kaggle.We have used the dataset from:

Environment and tools

- sklearn
- seaborn
- numpy
- pandas
- matplotlib

Where is the code?

Without much ado, let's get started with the code.

We started with loading all the libraries and dependencies. The columns in the dataset are customer id, gender, age, income and spending score.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

print("libraries implemented")
```

```
In [1]: runfile('C:/Users/mobee/.spyder-py3/version 2 ai code.py', wdir='C:/Users/mobee/.spyder-py3')
libraries implemented
```

Libraries:

Pandas: pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Numpy: numpy is an open-source numerical Python library. NumPy contains a multi-dimensional array and matrix data structures. It can be utilized to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.

Sklearn: Here we use this library to import KMean, metrices and LabelEncoder functions to use in the code.

Matplotlib: Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB.

Seaborn: Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

And when these all the libraries will successfully be loaded and implemented the "**libraries implemented**" will print out on the screen.

```
data = pd.read_csv('C:/Users/mobee/Downloads/Mall_Customers.csv')
print (data)
```

```
     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
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
```

We will load the dataset that we are using for our project "**Mall_Customers.csv**" by giving its full path with the help of **pandas** library.

And after the we print the data on the screen with print statement as shown.

```
df= data.copy()
print (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
```

Now here the copy of the whole dataset is done here in variable name **df** and printing out the head of copied dataset on screen as shown.

```
#Nulls in Dataset.
print(df.isnull().sum())
```

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

Checking whether there is any nulls in the data set or not by using function isnull() and by function sum() we are summing the null in of every column in a table as shown but the results are that there is **no Nulls in Dataset**.

```
print("rows: ",df.shape[0])
print("columns",df.shape[1])
```

```
rows:  200
columns:  5
```

Checking how many rows and columns are in total in the dataset.

```
print(df.dtypes)
```

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

This will tell us the data types of every column in the dataset.

```
print(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

This function will describe the data set and tell the count, mean, standard deviation, minimum, 25% (First quartile), 50% (Second quartile), 75% (Third quartile) and maximum of every column and present it in a table foam as shown.

By which we have predicted that

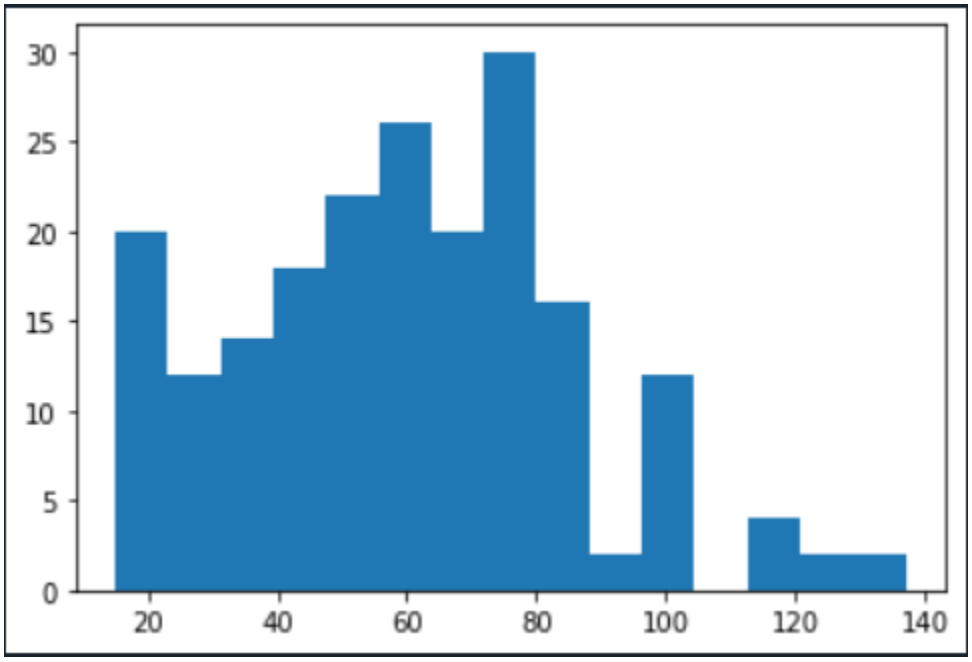
- Age is Positive skew

- Annual Income is Negative Skew

- Spending score is almost symmetric

Histogram of Annual Income:

```
#histogram of Annual Income
plt.hist(df['Annual Income (k$)'],bins=15)
plt.show()
```



This chunk of code will draw the histogram of the Annual Income from the dataset values which will have 15 bins in total to present data.

By which we have predicted that

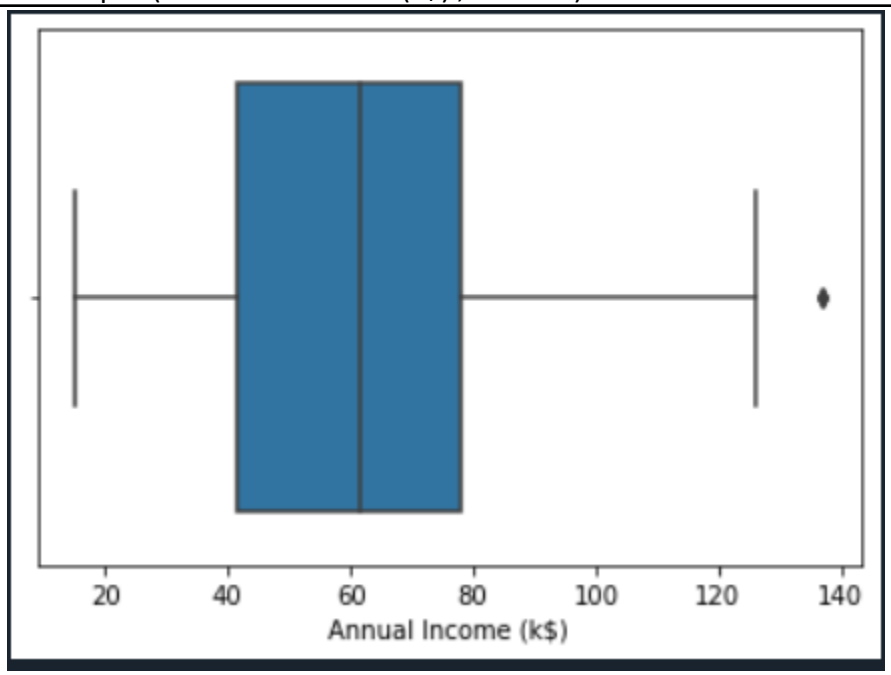
- it is negatively skewed

- Mean is around 60k$

(Negatively skewed distribution refers to the distribution type where the more values are plotted on the right side of the graph, where the tail of the distribution is longer on the left side and the mean is lower than the median and mode which it might be zero or negative due to the nature of the data as negatively)

Boxplot of Annual Income (k$):

```
#boxplot of Annual Income
sns.boxplot(x='Annual Income (k$)', data=df)
```



This chunk of code will draw the boxplot of the Annual Income from the dataset values.

By which we have predicted that

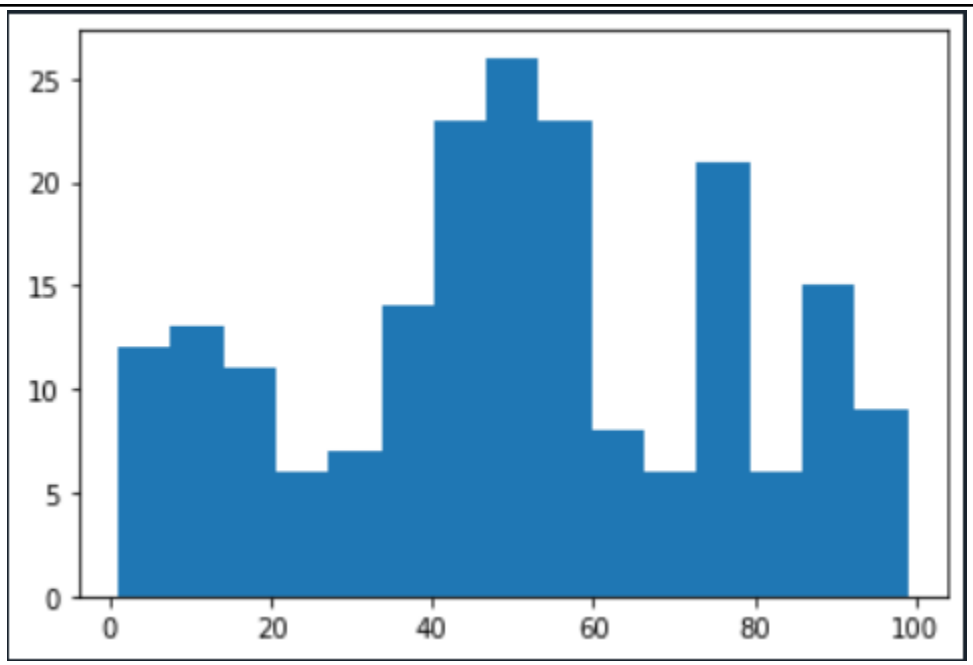
- it is negatively skewed

- outlies exist in data.

(An outlier is an observation that lies an abnormal distance from other values in a random sample from a population)

Histogram of Spending Score:

```
#histogram of Spending Score
plt.hist(df['Spending Score (1-100)'],bins=15)
plt.show()
```



This chunk of code will draw the histogram of the Spending Score from the dataset values which will have 15 bins in total to present data.

By which we have predicted that

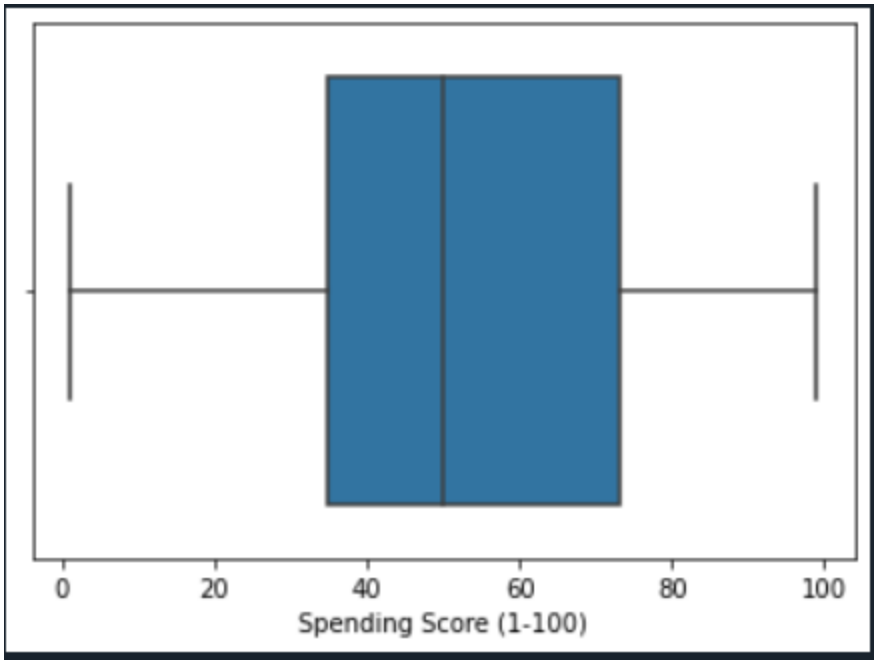
- it is Symmetric

- Mean is around 50

(If the data are symmetric, they have about the same shape on either side of the middle. With symmetric data, the mean and median are close together.)

Boxplot of Spending Score:

```
#boxplot of Spending Score
sns.boxplot(x='Spending Score (1-100)', data=df)
```



This chunk of code will draw the boxplot of the Spending Score from the dataset values.

By which we have predicted that

- Symmetric graph

- no outliers

Date Preprocessing:

```
label_encoder=LabelEncoder()
df['Genre']= label_encoder.fit_transform(df['Genre'])
print(df)
```

```
     CustomerID  Genre  Age  Annual Income (k$)  Spending Score (1-100)
0             1      1   19                  15                      39
1             2      1   21                  15                      81
2             3      0   20                  16                       6
3             4      0   23                  16                      77
4             5      0   31                  17                      40
..          ...    ...  ...                 ...                     ...
195         196      0   35                 120                      79
196         197      0   45                 126                      28
197         198      1   32                 126                      74
198         199      1   32                 137                      18
199         200      1   30                 137                      83

[200 rows x 5 columns]
```

LabelEncoder: Sklearn provides a very efficient tool for encoding the levels of categorical features into numeric values. LabelEncoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels. If a label repeats it assigns the same value to as assigned earlier.

Fit_transform: fit- It is used for calculating the initial filling of parameters on the training data (like mean of the column values) and saves them as an internal objects state. Transform- Use the above calculated values and return modified training data. Transform and fit are used together to remove null values and add mean in these values

We are converting here the categorical values in to numerical values before passing them into the model of genre column and printing the new table with numerical values in the column of genre as shown on screen.

```
X= df.drop(['CustomerID'],axis=1)
print(X)
```

```
     Genre  Age  Annual Income (k$)  Spending Score (1-100)
0        1   19                  15                      39
1        1   21                  15                      81
2        0   20                  16                       6
3        0   23                  16                      77
4        0   31                  17                      40
..     ...  ...                 ...                     ...
195      0   35                 120                      79
196      0   45                 126                      28
197      1   32                 126                      74
198      1   32                 137                      18
199      1   30                 137                      83

[200 rows x 4 columns]
```

We are dropping the 'CustomerID' column from the dataset because it doesn't affect our model and we don't need it during calculations of KMean because it is unique and we use Euclidean distance in KMean and we cannot use Euclidean distance on customer id. So, we drop CustomerID and store the dataset without column of 'CustomerID' in X as shown in result.

```
col_names=["Genre", "Age","Annual Income (k$)","Spending Score (1-100)"]
scaled=pd.DataFrame(columns=col_names,data=X)
print(scaled.head())
```

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

We are assigning all the column names to the variable **col_names** and then passing the col_names and X (dataset without CustomerID column) into DataFrame function as shown below.

Dataframe: Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Silhouette score:

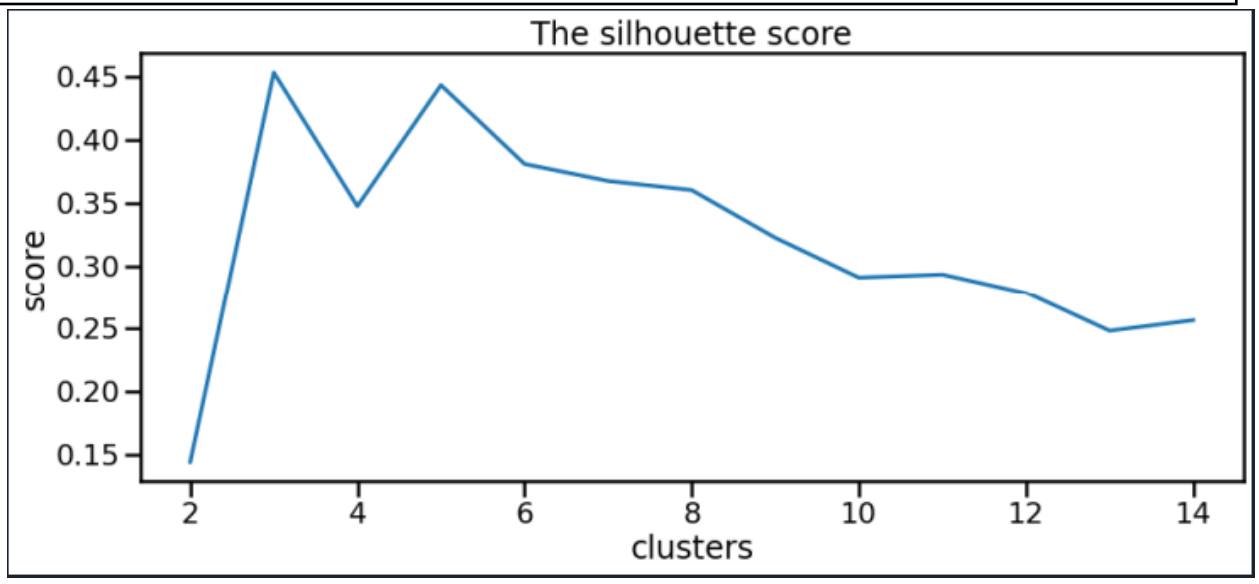
Silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1. 1: Means clusters are well apart from each other and clearly distinguished. 0: Means clusters are indifferent, or we can say that the distance between clusters is not significant. -1: Means clusters are assigned in the wrong way.

```
from sklearn.metrics import silhouette_score
s_score = []
for k in range(2, 15):
   kmeans = KMeans(n_clusters=k, random_state=0).fit(scaled)
   s_score.append([k, silhouette_score(df, kmeans.labels_)])
print(s_score)
```

```
[[2, 0.1443508310946748], [3, 0.4535541830196435], [4, 0.34791801459211613], [5, 0.44364769684095806], [6,
0.38128242115795885], [7, 0.36781895598712466], [8, 0.3606422916765322], [9, 0.3228794890551152], [10, 0.29132121570534336],
[11, 0.2936465950377538], [12, 0.27872087127612966], [13, 0.24816470274640273], [14, 0.2564886135452383]]
```

we are using Silhouette score to find the best no of clusters for this dataset. The best number will be which has high score. Here with the help of a algorithm we have calculated the score for every number from 2 to 15 with the help of a while loop. We use the KMean function to find clusters with the random state=0 and we pass the copy of dataset as scaled. And append the result that are coming from the silhouette_score function in s_score and print out the result as shown. For example: [2,0.1443508] in this 2 is no of clusters and 0.1443508 is the score and so on for all of them.

```
plt.figure(figsize=(15,6))      #(width,height)
plt.plot( pd.DataFrame(s_score)[0], pd.DataFrame(s_score)[1])
plt.xlabel('clusters')
plt.ylabel('score')
plt.title('The silhouette score')
plt.show()
```



Now we are plotting the result that come from the silhouette score algorithm with help of results that are stored in s_score. By that we have the graph as shown above in which it is highest on 6 so

6 clusters are best due to high silhouette_score

```
pred=X.copy()
print(pred)
```

```
     Genre  Age  Annual Income (k$)  Spending Score (1-100)
0        1   19                  15                      39
1        1   21                  15                      81
2        0   20                  16                       6
3        0   23                  16                      77
4        0   31                  17                      40
..     ...  ...                 ...                     ...
195      0   35                 120                      79
196      0   45                 126                      28
197      1   32                 126                      74
198      1   32                 137                      18
199      1   30                 137                      83

[200 rows x 4 columns]
```

We are copying X in to pred so that we can use it for the model or anywhere without making the changes in actual dataset.

KMeans

```
kmean3 = KMeans(n_clusters=6, random_state=0)
#write code to fit
kmean3.fit(pred)
#Write code to assign labels to predicted data
pred['kmean3'] =  kmean3.labels_
print(pred['kmean3'])
```

```
0      5
1      4
2      5
3      4
4      5
      ..
195    3
196    2
197    3
198    2
199    3
Name: kmean3, Length: 200, dtype: int32
```

Now we know the best no of cluster so we will make the cluster of the dataset with the help of KMean functions. We will assign every predicted value to the suitable cluster for it. The results are showing that which row belong to which cluster for example: 0 row belong to 5^{th} cluster, 1 row belong to 4^{th} cluster, 2 row belong also belong to 5^{th} cluster and so on. The list that which row belong to which cluster is shown in results.

```
#Write code to display value counts
print(pred['kmean3'].value_counts())
```

```
1    45
3    39
0    38
2    35
4    22
5    21
Name: kmean3, dtype: int64
```

Now we will count that how many number of rows belong to each cluster and print it as shown above. This is the final KMean cluster and there no of rows list. So dataset clustering is done with help of KMean is done successfully.

```
p_ = pred[["Genre", "Age","Annual Income (k$)","Spending Score (1-100)",'kmean3']]
print(p_)
```

```
     Genre  Age  Annual Income (k$)  Spending Score (1-100)  kmean3
0        1   19                  15                      39       5
1        1   21                  15                      81       4
2        0   20                  16                       6       5
3        0   23                  16                      77       4
4        0   31                  17                      40       5
..     ...  ...                 ...                     ...     ...
195      0   35                 120                      79       3
196      0   45                 126                      28       2
197      1   32                 126                      74       3
198      1   32                 137                      18       2
199      1   30                 137                      83       3

[200 rows x 5 columns]
```

Now this is the same result as above that also shows which row belong to which cluster but here it is detailed and with the rows details as shown above.

```
pivoted =(p_.groupby('kmean3')["Genre", "Age","Annual Income (k$)","Spending Score
(1-100)"].median().reset_index())
print(pivoted)
```

```
   kmean3  Genre   Age  Annual Income (k$)  Spending Score (1-100)
0       0      0  26.5                59.5                    50.0
1       1      0  54.0                54.0                    49.0
2       2      1  43.0                85.0                    16.0
3       3      0  32.0                79.0                    83.0
4       4      0  23.5                24.5                    77.0
5       5      0  45.0                24.0                    15.0
```

This is the median of all the rows present in each cluster for better understanding of customer segmentation. The median of every cluster rows as per column are shown above.

Conclusions

K means clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of K means is to group data points into distinct non-overlapping subgroups. One of the major application of K means clustering is segmentation of customers to get a better understanding of them which in turn could be used to increase the revenue of the company.