K-means Clustering in Machine Learning

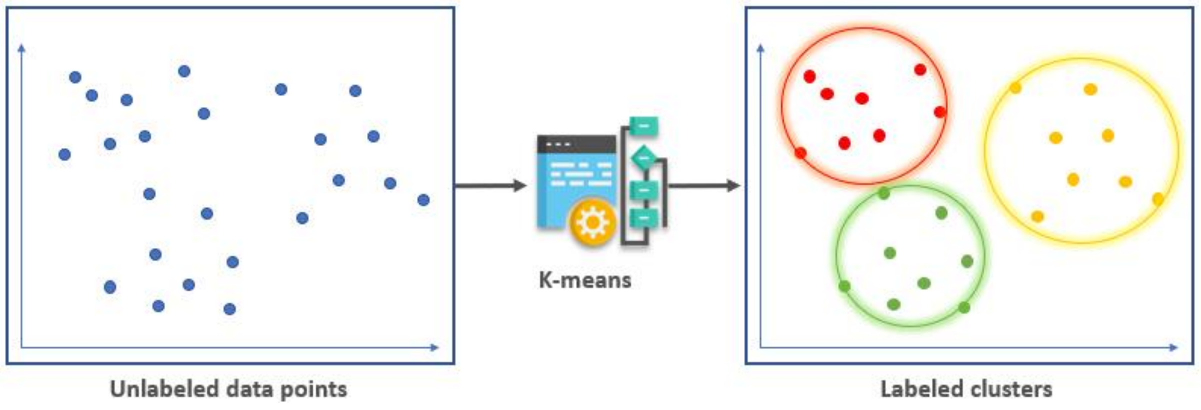
When you are dealing with Machine Learning problems that work with unlabeled training datasets, the most common learning algorithms you will come across are clustering algorithms. Amongst them, the simplest and the most popular is K-means clustering.

Typically, a good clustering algorithm groups the data into clusters such that the data points within each cluster are more similar to each other than the data points belonging to other clusters. Clustering algorithms find their usage in many applications including pattern detection, medical diagnosis, market segmentation, customer segmentation, etc.

What is K-means Clustering?

K-means is a clustering algorithm designed to divide data points into distinct clusters depending on the similarities between the observations. Data points that share certain relevant characteristics are grouped together, implying that data points in different clusters would be dissimilar from each other.

This is an unsupervised learning algorithm, which means you can train a model to create clusters on some given data without needing to label it first.



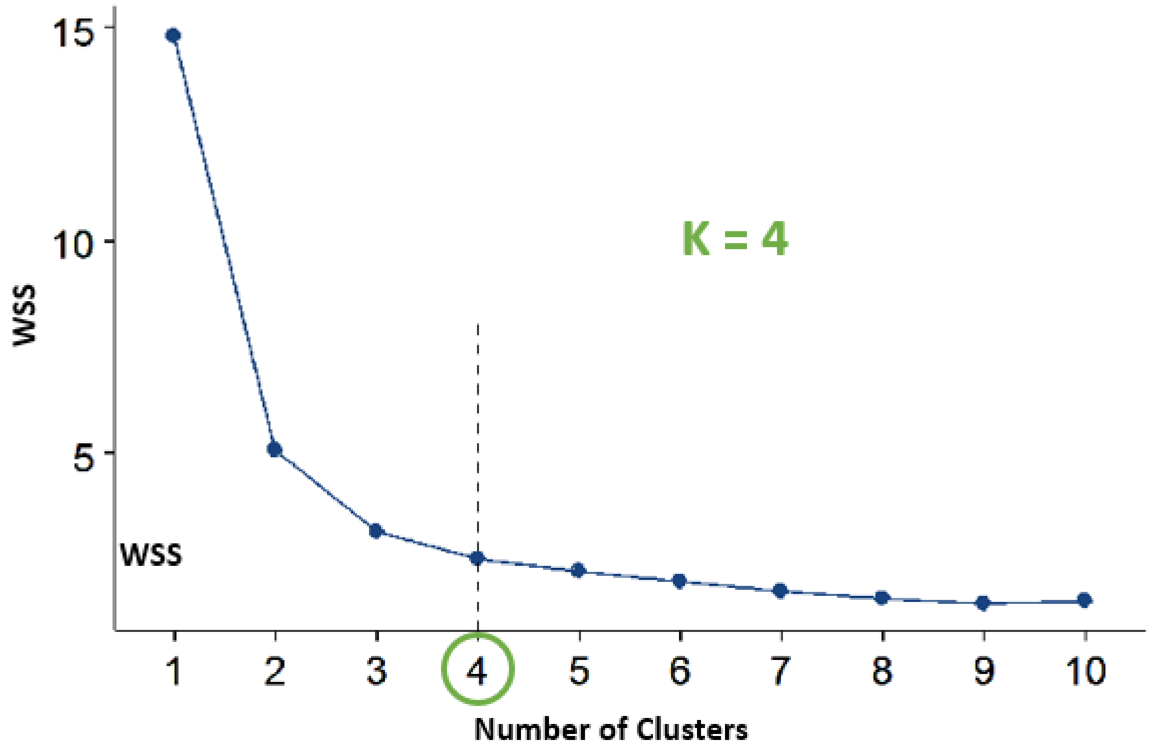
How to choose the value of K?

The number of clusters 'K' you want to group your data points into, has to be pre-defined. Determining the optimal number of clusters for a given data is a fundamental issue in K-means clustering. The answer to this question is somewhat subjective and mostly depends on the methods used for partitioning. The most commonly used method in the industry to identify the value of K is the elbow method.

The Elbow Method

In this method, the idea behind partitioning is to define clusters such that total **intra-cluster variation** or the total **within the sum of squares** (WSS) is minimized for each cluster. WSS measures the compactness of the clusters – so you'd want the value to be as small as possible.

The method consists of plotting the total WSS as a function of the various number of clusters. And then picking the 'elbow' of the curve as the appropriate number of clusters to use. See the example below.



The motive behind this method is the number of clusters should be chosen such that adding another cluster would not increase the total WSS.

The code to generate a graph and the modeling component of this algorithm is provided below in the *Demo: Implementing K-means Clustering* section of this blog.

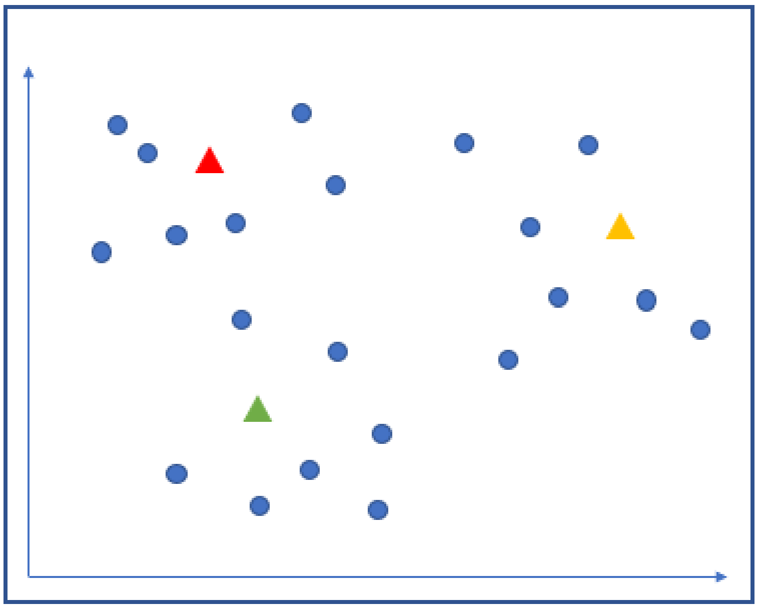
How does the K-means algorithm work?

- **Choose the value of K.**

For example, let's assume the value of K is 3.

- **Select ‘K’ data points each of which represents a cluster centroid.**

Since K=3, we will choose 3 random data points as centroids.



- **Assign the rest of the data points to their nearest cluster centroids.**

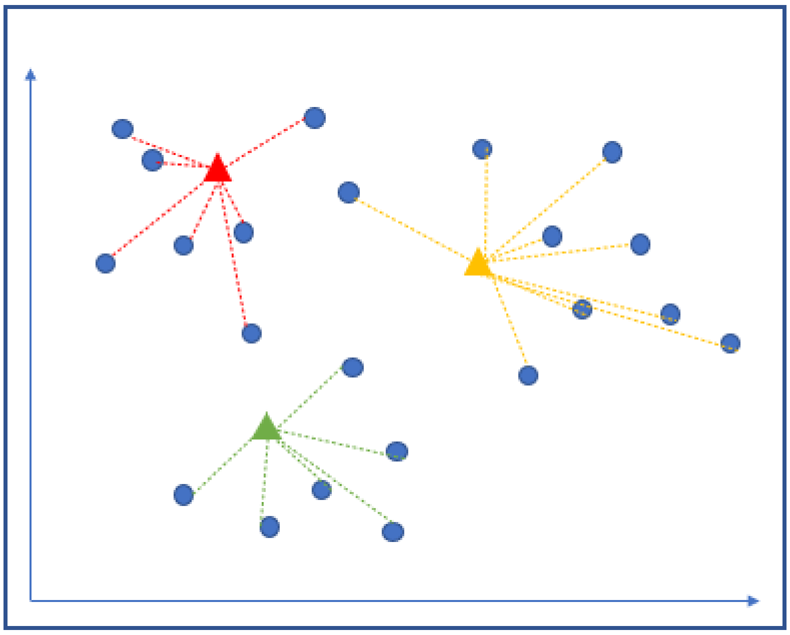
We will calculate the *Euclidean distance* between the data points and the centroids. The data point at a minimum distance with a given centroid is assigned to its cluster.

If we consider two points P(p1,p2) and Q(q1,q2) in a 2D space, the shortest distance between these two points is given as the length of the hypotenuse of a right triangle. This is the Euclidean distance between the two points.

$$h^2 = (p1 - q1)^2 + (p2 - q2)^2$$

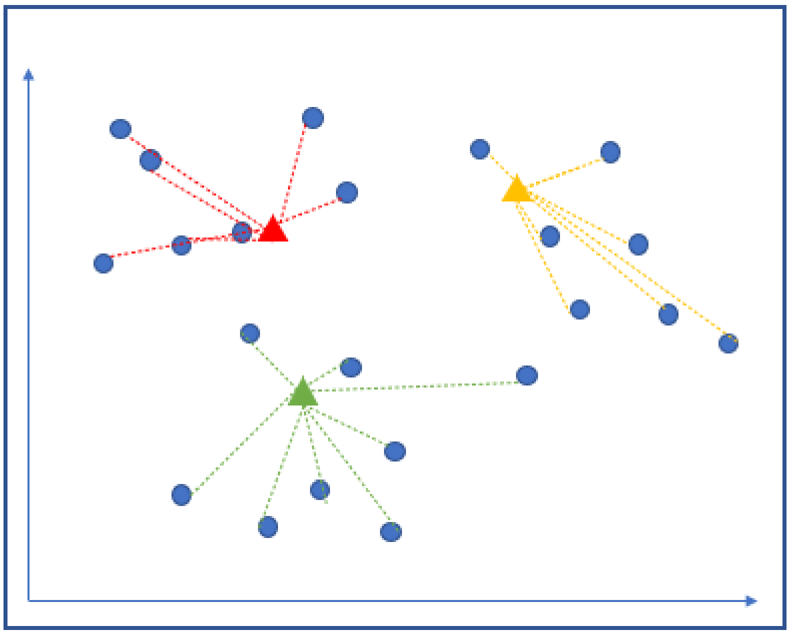
The value of h is the Euclidean distance.

The value of h is the Euclidean distance.

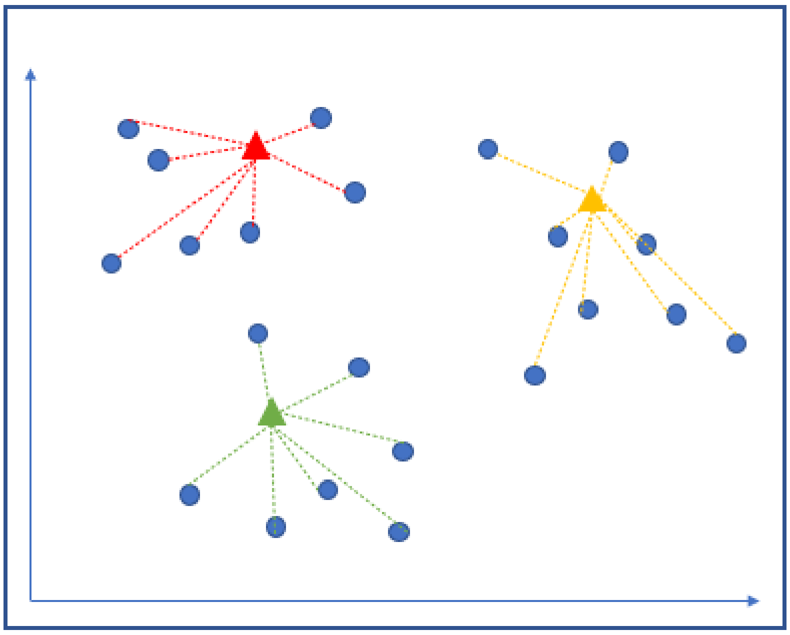


- **Reposition the cluster centroid after each data point assignment till it's the average of all data points in that cluster.**

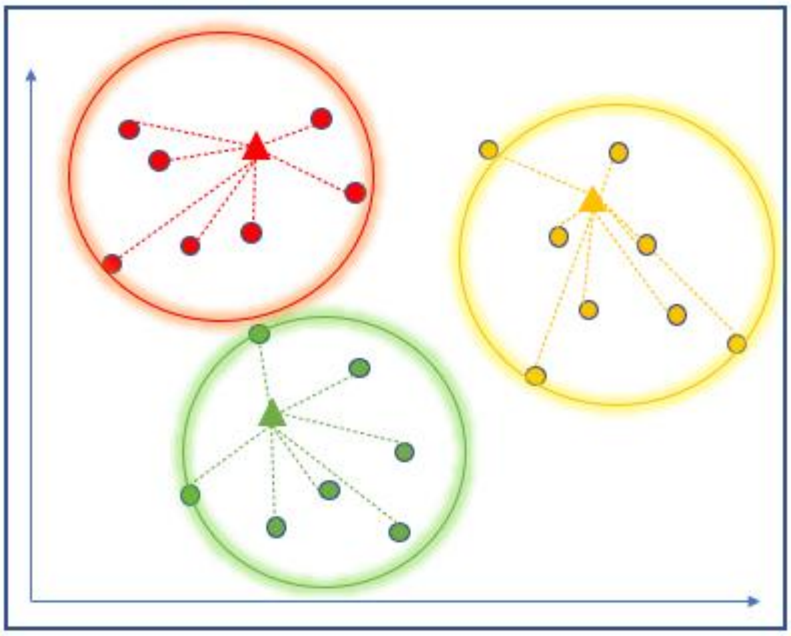
Once a data point is assigned to the nearest cluster, the cluster centroid is re-calculated, including the new point.



- **Repeat steps 3 and 4 till there are no more changes in each cluster.**



After the final iteration, this is how the clusters are formed:



Implementing K-means Clustering

Please keep in mind that K-means clustering is **not performed on categorical data**.

Dataset Description:

We will be creating a dataset using the make_blobs API from the **Scikit-learn** library. This API is used to create multi-class datasets by allocating each class to one or more normally distributed clusters.

Tasks to be performed:

- Generate the data
- Visualize the data clusters
- Plot the Elbow Curve
- Build and Visualize the K-means Clustering Model

Step 1 – Generate the data

Let us generate a dataset with, say, 12 random clusters in a 2D space. For that, we will initialize this AdaBoost ensemble model with the following parameters:

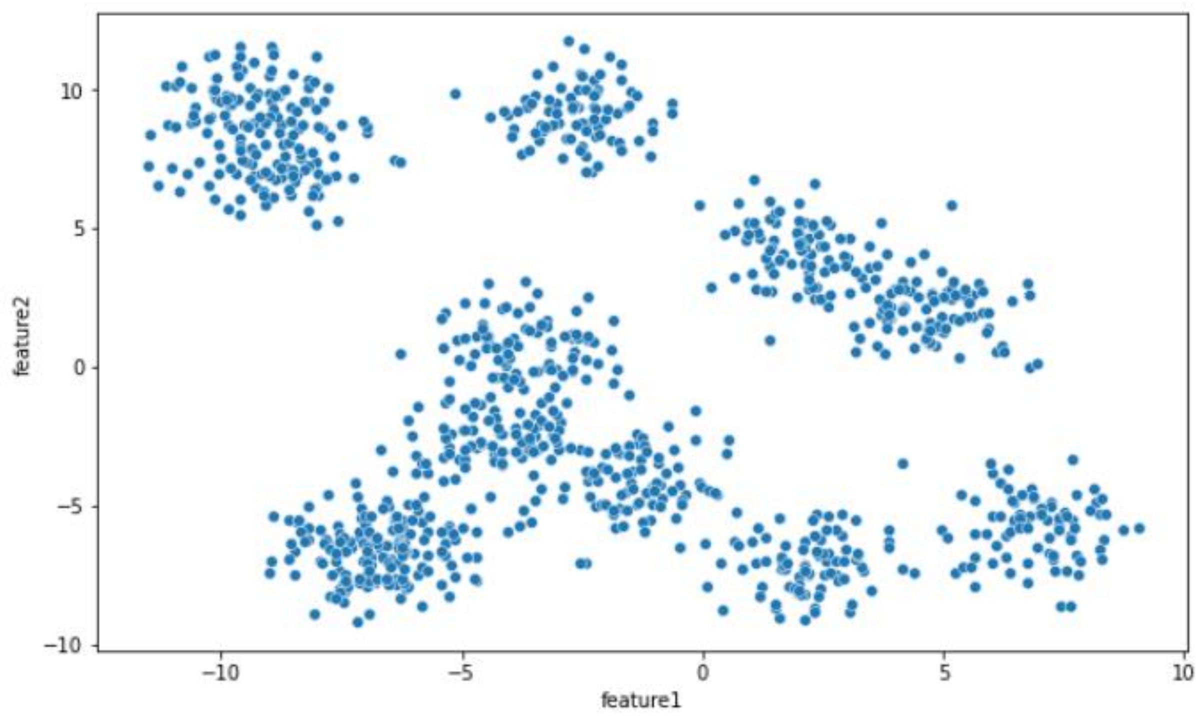
- *n_samples* = 1000 – Create 1000 samples
- *n_features* = 2 – The dataset will have two features
- *centers* = 12 – Create 12 random clusters with 12 centroids

```
import pandas as pd
from sklearn.datasets import make_blobs

#Generate the data
X, y = make_blobs(n_samples=1000, n_features=2, centers=12, random_state=42)

#Create a DataFrame for the dataset features
df = pd.DataFrame(X, columns=['feature1','feature2'])
```

Step 2 – Visualize the data clusters

We will use a **Seaborn** scatter plot to visualize our data with its 2 features:

```
import seaborn as sns

import matplotlib.pyplot as plt



#Create a scatter plot using seaborn

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

sns.scatterplot(x='feature1', y='feature2', data=df)
```

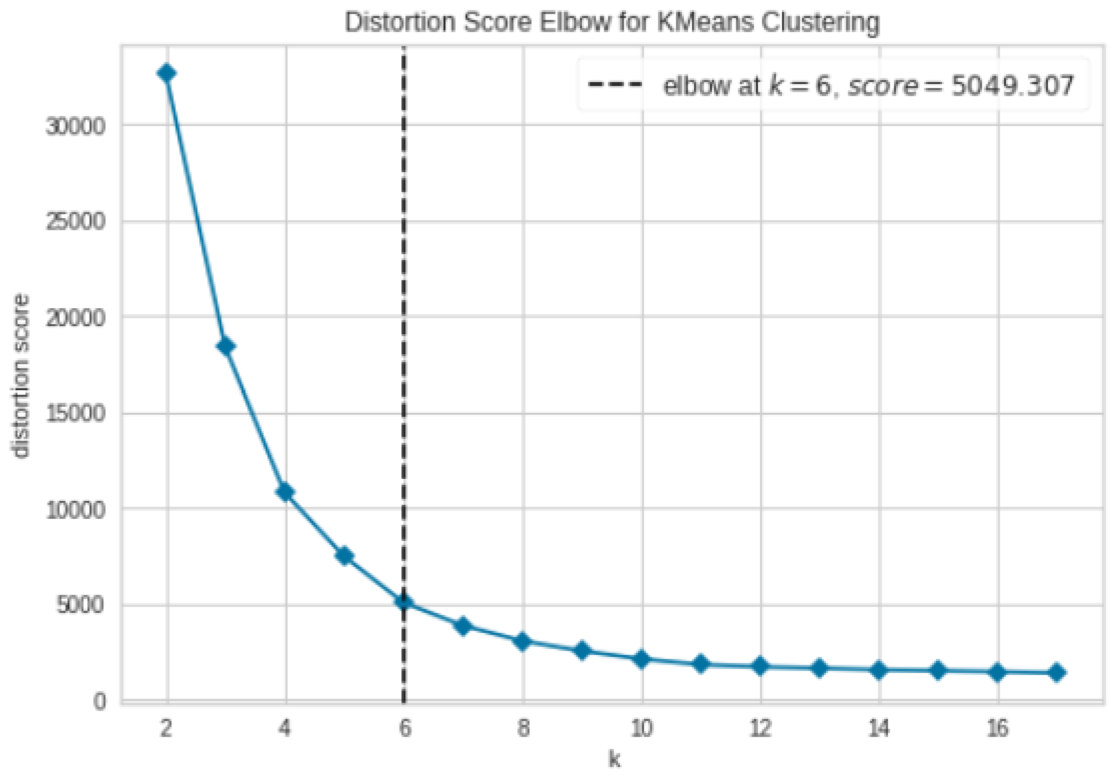


Though we specified 12 centers, the plot above shows an overlap between some clusters. So, have only 7-8 separable clusters right now.

Step 3 – Plot the Elbow Curve

```
from sklearn.cluster import KMeans
Sum_of_squared_distances = []
K = range(1,10)
for num_clusters in K :
   kmeans = KMeans(n_clusters=num_clusters)
   kmeans.fit(df)
   Sum_of_squared_distances.append(kmeans.inertia_)
plt.plot(K,Sum_of_squared_distances,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Elbow Method For Optimal k')
```

plt.show()



We found that the optimal value of the number of clusters using the Elbow Method is 6. Let's see how our model clusters the data when K=6.

Step 4 – Build and visualize the K-means Clustering Model

Let's see how our model clusters the data when K=6:

from sklearn.cluster import KMeans

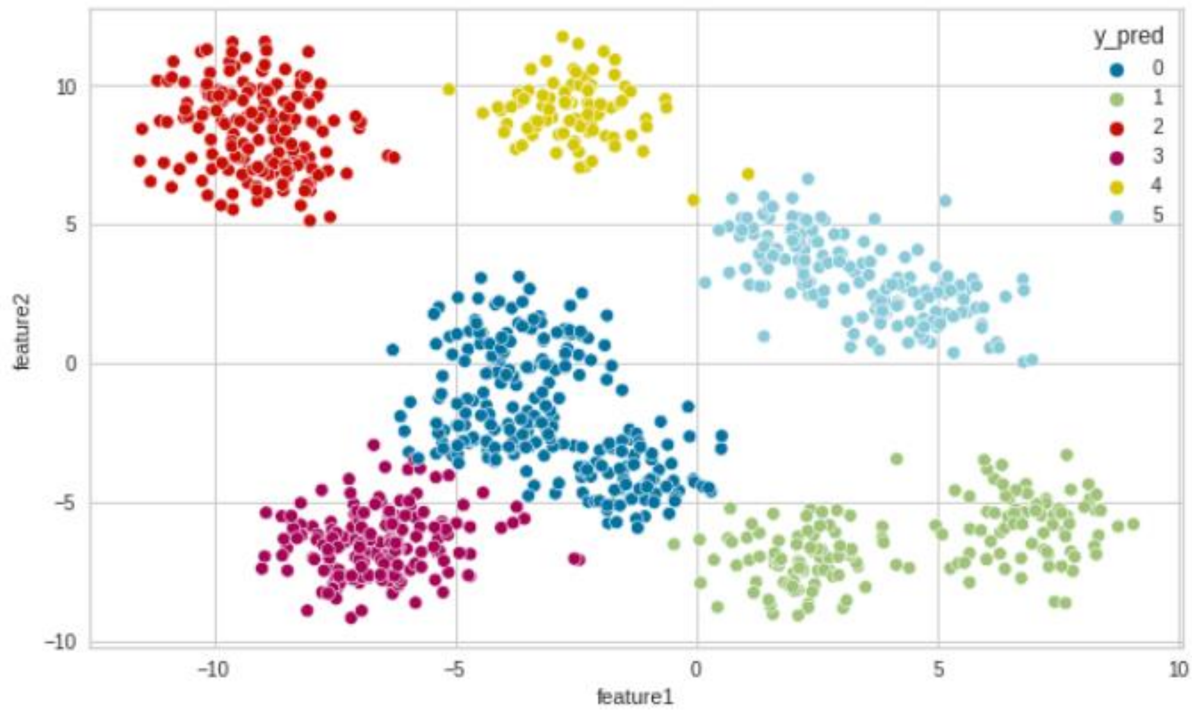
#Create the model

model = KMeans(n_clusters=6)

#Fit values to the model

model.fit(X)

```
#Predict the output labels

df['y_pred'] = model.predict(X)


#Convert the output column to category

df['y_pred'] = df['y_pred'].astype('category')


#Plot the clusterd data using seaborn

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

sns.scatterplot(x='feature1', y='feature2', hue='y_pred', data=df)
```



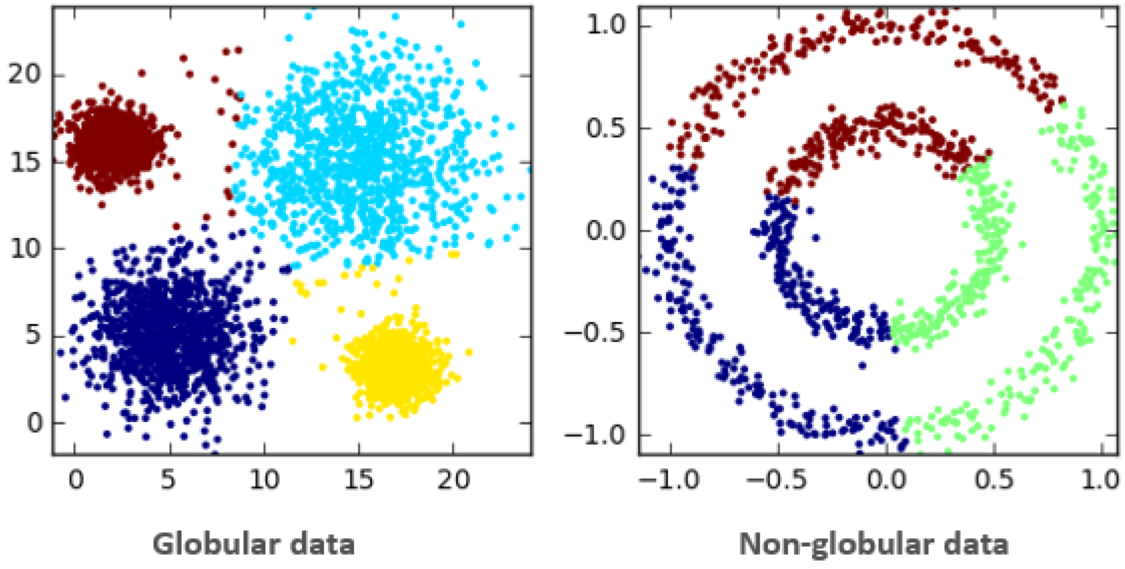
Thus, we have a perfectly grouped data divided into 6 defined clusters.

Assumptions of K-means Clustering

Like most machine learning models, K-means clustering also holds certain assumptions about the data they fit on. It is important to check on these assumptions before we rely on our trained model:

- **K-means cannot handle non-globular structure**

A given data can have any type or number of patterns that can be interpreted visually. Since the K-means algorithm uses centroids as a reference, it does not capture the information well if the data has a non-globular pattern.



- **K-means is susceptible to outliers.**

We already know that in K-means, centroids are re-calculated repeatedly by averaging the data points in the cluster. Averages are sensitive to outliers. So, it becomes necessary to spot and remove outliers before training our model.

To measure accuracy we may use silhouette score

- if we do not have the actual labels of the data points, we will have to use the *intrinsic methods* like silhouette score which is based on the *silhouette coefficient*.
- The equation for calculating the silhouette coefficient for a particular data point:

$$s(\boldsymbol{o}) = \frac{b(\boldsymbol{o}) - a(\boldsymbol{o})}{\max\{a(\boldsymbol{o}), b(\boldsymbol{o})\}}$$

where,

- ***s(o)*** is the silhouette coefficient of the data point **o**

- ***a(o)*** is the *average distance* between **o** and all the other data points in the cluster to which **o** belongs

- ***b(o)*** is the *minimum average distance* from **o** to all clusters to which **o** does not belong

There are main points that we should remember during calculating silhouette coefficient . The value of the silhouette coefficient is between [-1, 1]. A score of 1 denotes the best meaning that the data point **o** is very compact within the cluster to which it belongs and far away from the other clusters**.** The worst value is -1. Values near 0 denote overlapping clusters.

Elbow Curve Method

Recall that the basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total wss measures the compactness of the clustering, and we want it to be as small as possible. The elbow method runs k-means clustering (kmeans number of clusters) on the dataset for a range of values of k (say 1 to 10) In the elbow method, we plot mean distance and look for the elbow point where the rate of decrease shifts. For each k, calculate the total within-cluster sum of squares (WSS). This elbow point can be used to determine K.

- Perform K-means clustering with all these different values of K. For each of the K values, we calculate average distances to the centroid across all data points.
- Plot these points and find the point where the average distance from the centroid falls suddenly ("Elbow").

At first, clusters will give a lot of information (about variance), but at some point, the marginal gain will drop, giving an angle in the graph. The number of clusters is chosen at this point, hence the "elbow criterion". This "elbow" can't always be unambiguously identified.

Inertia: Sum of squared distances of samples to their closest cluster center.

we always do not have clear clustered data. This means that the elbow may not be clear and sharp.

Let us see the python code with the help of an example.

Python Code:

Visually we can see that the optimal number of clusters should be around 3. But ***visualizing/visualization of the data alone cannot always give the right answer.***

```
Sum_of_squared_distances = []
```

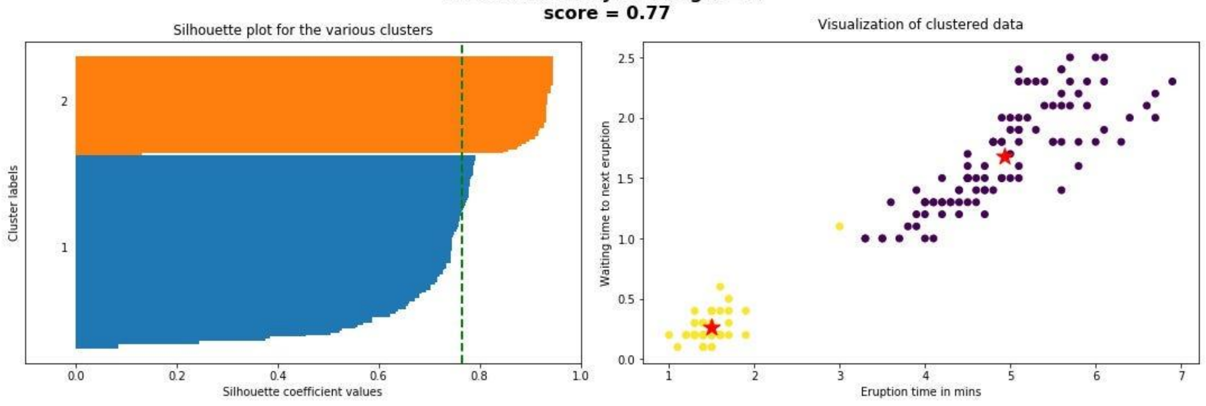
```
K = range(1,10)
for num_clusters in K :
  kmeans = KMeans(n_clusters=num_clusters)
  kmeans.fit(data_frame)
  Sum_of_squared_distances.append(kmeans.inertia_)
plt.plot(K,Sum_of_squared_distances,’bx-’)
plt.xlabel(‘Values of K’)
plt.ylabel(‘Sum of squared distances/Inertia’)
plt.title(‘Elbow Method For Optimal k’)
plt.show()
```

Python Code

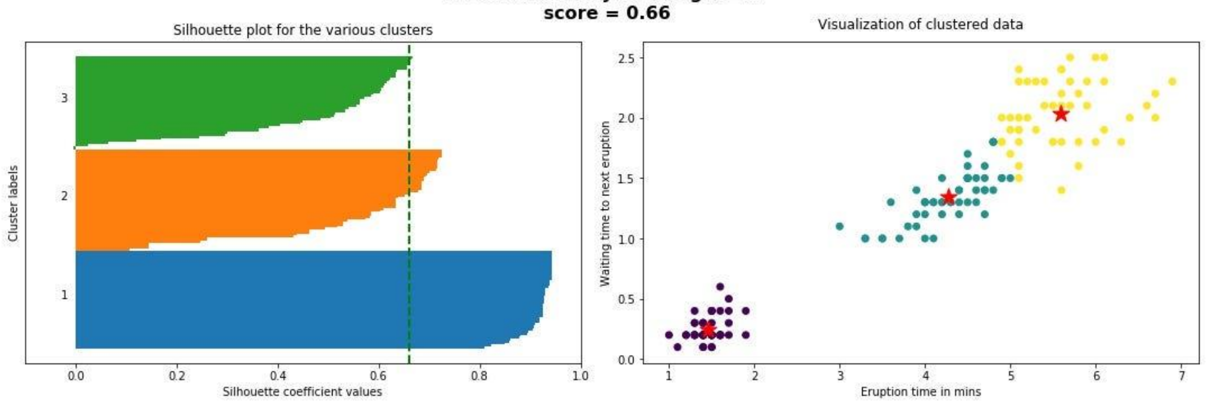
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples,silhouette_score
# load petal data
data = datasets.load_iris()
dir(data)# load into Dataframe
df = pd.DataFrame(data.data,columns = data.feature_names)
print(df.shape)
df.head()
df1 = df.drop(['sepal length (cm)', 'sepal width (cm)'],axis =
'columns')
df1.head()
# plot scatter plot
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'])
# Now check silhouette coefficient

for i,k in enumerate([2,3,4,5]):
    fig, ax = plt.subplots(1,2,figsize=(15,5))

    # Run the kmeans algorithm
    km = KMeans(n_clusters=k)
    y_predict = km.fit_predict(df1)
    centroids  = km.cluster_centers_

       # get silhouette
    silhouette_vals = silhouette_samples(df1,y_predict)
    #silhouette_vals
        # silhouette ploty_ticks = []
    y_lower = y_upper = 0for i,cluster in
enumerate(np.unique(y_predict)):
   cluster_silhouette_vals = silhouette_vals[y_predict
==cluster]
   cluster_silhouette_vals.sort()
   y_upper += len(cluster_silhouette_vals)

   ax[0].barh(range(y_lower,y_upper),
   cluster_silhouette_vals,height =1);   ax[0].text(-
0.03,(y_lower+y_upper)/2,str(i+1))
   y_lower += len(cluster_silhouette_vals)
   # Get the average silhouette score    avg_score =
np.mean(silhouette_vals)
   ax[0].axvline(avg_score,linestyle ='--',
   linewidth =2,color = 'green')
   ax[0].set_yticks([])
   ax[0].set_xlim([-0.1, 1])
   ax[0].set_xlabel('Silhouette coefficient values')
   ax[0].set_ylabel('Cluster labels')
   ax[0].set_title('Silhouette plot for the various
clusters');


    # scatter plot of data colored with labels

    ax[1].scatter(df2['petal length (cm)'],
    df2['petal width (cm)'] , c = y_predict);
ax[1].scatter(centroids[:,0],centroids[:,1],
    marker = '*' , c= 'r',s =250);
    ax[1].set_xlabel('Eruption time in mins')
    ax[1].set_ylabel('Waiting time to next eruption')
    ax[1].set_title('Visualization of clustered data',
y=1.02)

    plt.tight_layout()
    plt.suptitle(f' Silhouette analysis using k =
{k}',fontsize=16,fontweight = 'semibold')
    plt.savefig(f'Silhouette_analysis_{k}.jpg')
```

Output:

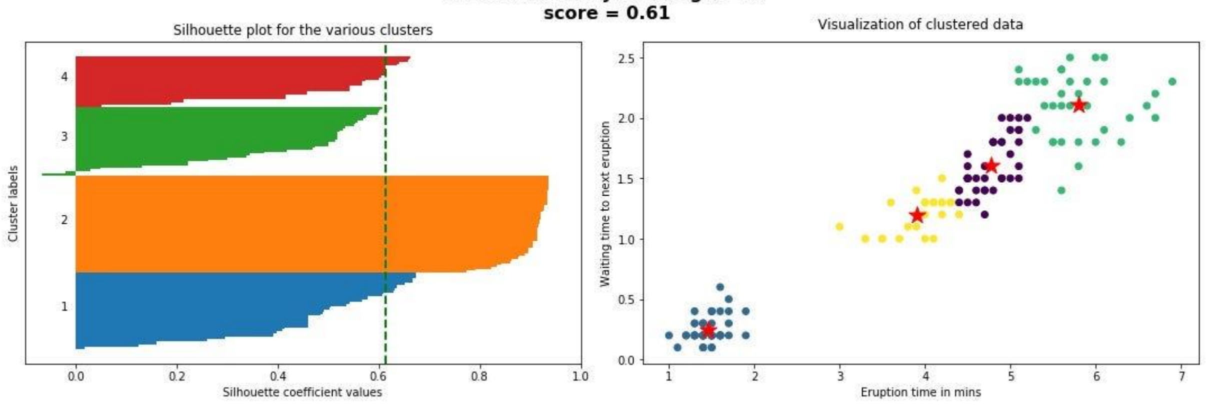
Silhouette analysis using k = 2
score = 0.77



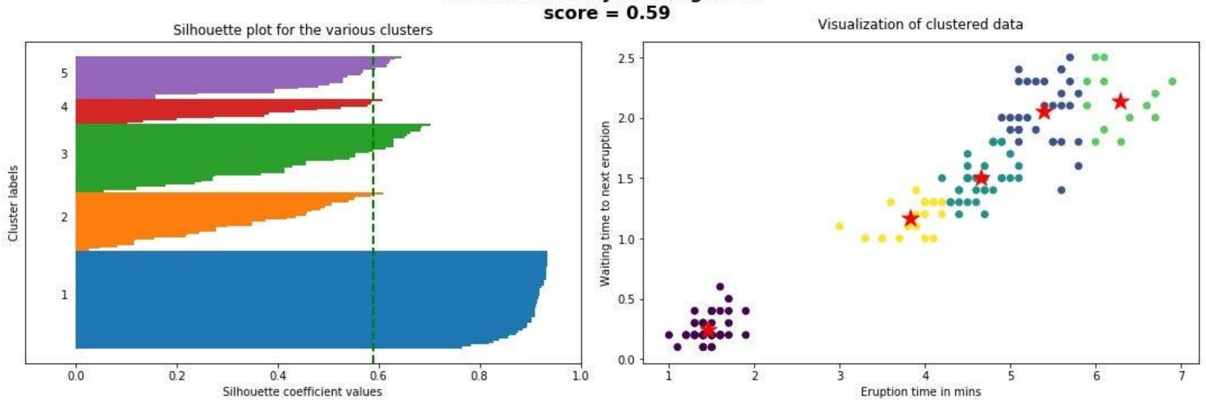
Silhouette analysis using k = 3
score = 0.66



Silhouette analysis using k = 4
score = 0.61



Silhouette analysis using k = 5
score = 0.59



In above all pictures , we can clearly see that how plot and score are different according to n_cluster(k) . So, we can easily choose high score and number of k via silhouette analysis technique instead of elbow technique.