# EXPERIMENT NO. 7

Aim: To implement Clustering algorithms (K-means/K-medoids).

Requirements: Windows O.S, Weka tool, Python and Python libraries: Pandas, Numpy, Sklearn and Matplot.

Problem Statement: To implement Clustering algorithms (K-means/K-medoids) on iris data set.

Theory:

## K-Means Clustering:

K-Means algorithm is a centroid based clustering technique. This technique cluster the dataset to k different cluster having an almost equal number of points. Each cluster is k-means clustering algorithm is represented by a centroid point.

***What is a centroid point?***

The centroid point is the point that represents its cluster. Centroid point is the average of all the points in the set and will change in each step and will be computed by:

$$C_i = \frac{1}{||S_i||} \sum_{x_j \epsilon S_i} x_j$$

The idea of the K-Means algorithm is to find k-centroid points and every point in the dataset will belong either of k-sets having minimum Euclidean distance.

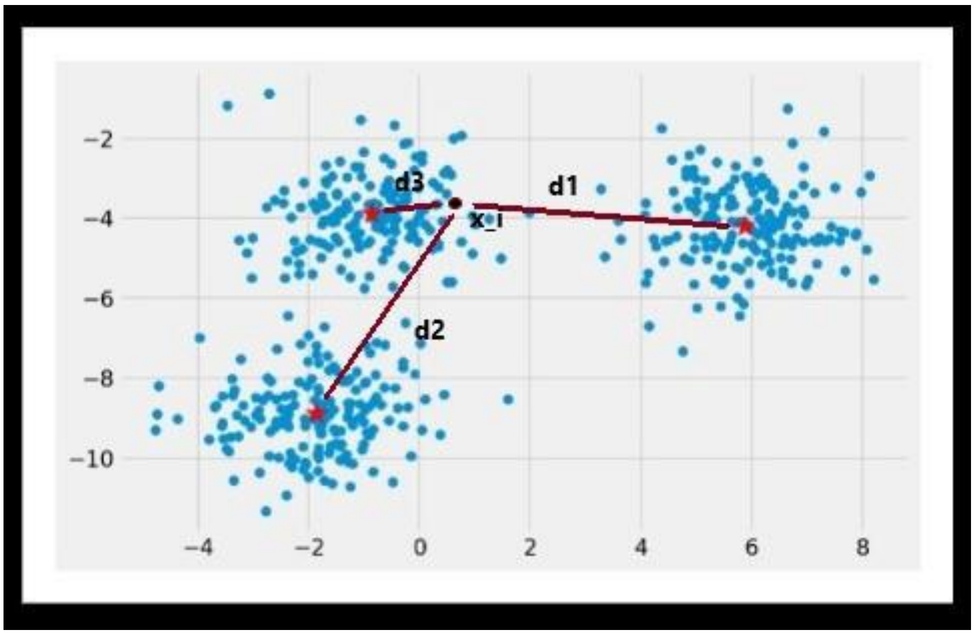


Image a

From the image above (Image a), the distance of point x_i from all three centroids are d1, d2, d3, x_i point is nearest to centroid_3 with distance d3, so the point x_i will belong to the cluster of centroid_3 and this process will continue for all the points in the dataset.

Cost Function of K-Means:

$$C_1, C_2, \cdots, C_k = argmin \sum_{i=1}^{k} \sum_{x \epsilon S_i} ||x - C_i||^2$$

The idea of the K-Means algorithm is to find k centroid points (C_1, C_2, . . . C_k) by minimizing the sum over each cluster of the sum of the square of the distance between the point and its centroid.

***Iterative implementation of the K-Means algorithm:***

**Steps #1: Initialization:**

The initial k-centroids are randomly picked from the dataset of points.

**Steps #2: Assignment:**

For each point in the dataset, find the Euclidean distance between the point and all centroids. The point will be assigned to the cluster with the nearest centroid.

**Steps #3: Updation of Centroid:**

Update the value of the centroid with the new mean value.

**Steps #4: Repeat:**

Repeat steps 2 and 3 unless convergence is achieved. If convergence is achieved then break the loop. Convergence refers to the condition where the previous value of centroids is equal to the updated value.

**K-Medoids clustering**:

It is a clustering algorithm resembling the K-Means clustering technique. It falls under the category of unsupervised machine learning. It majorly differs from the K-Means algorithm in terms of the way it selects the clusters' centres. The former selects the average of a cluster's points as its centre (which may or may not be one of the data points) while the latter always picks the actual data points from the clusters as their centres (also known as '**exemplars**' or '**medoids**'). K-Medoids also differs in this respect from the K-Medians algorithm which is the same as K-means, except that it chooses the medians (instead of means) of the clusters as centres.

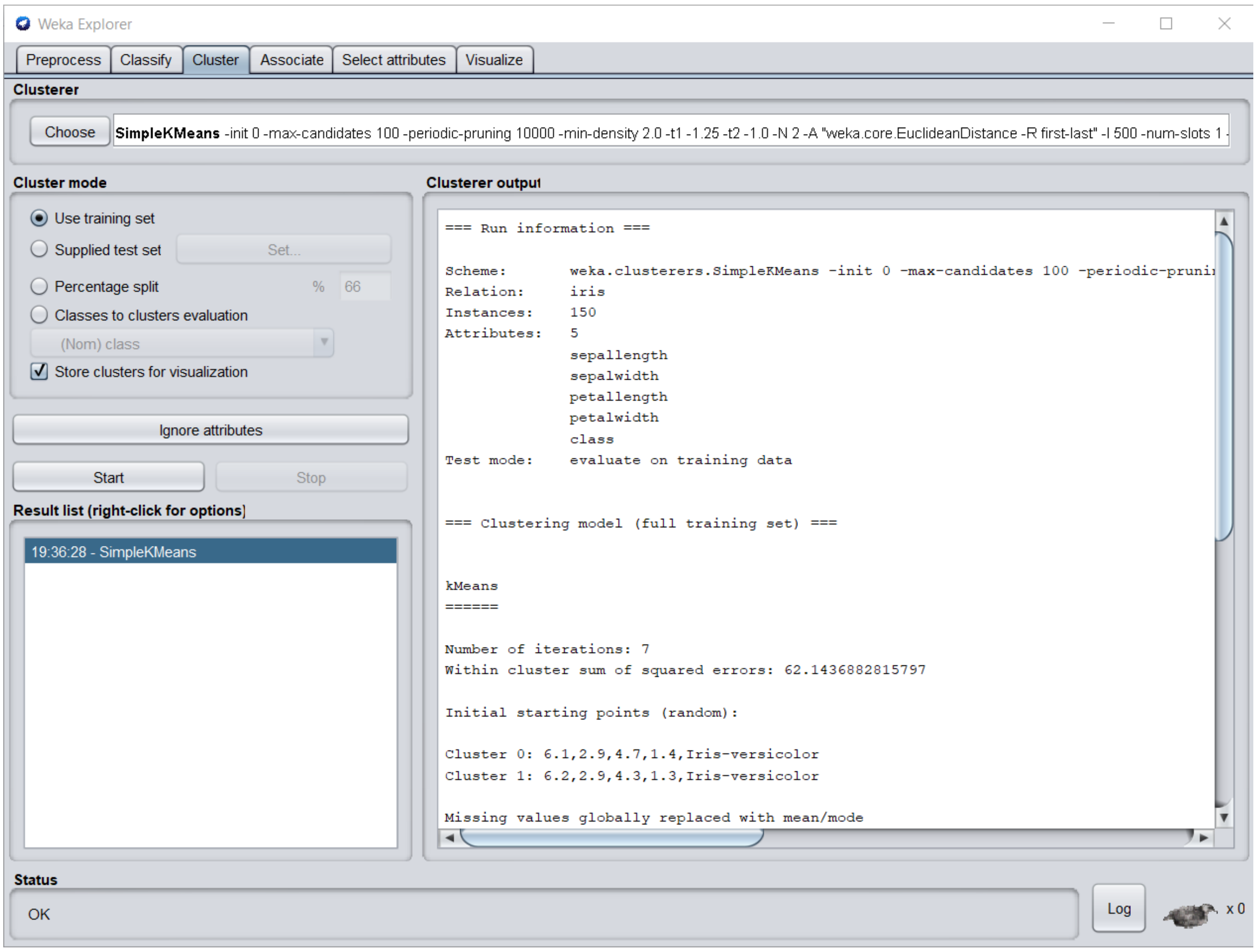
Working of the K-Medoids approach:

The steps followed by the K-Medoids algorithm for clustering are as follows:

1. Randomly choose 'k' points from the input data ('k' is the number of clusters to be formed). The correctness of the choice of k's value can be assessed using methods such as silhouette method.

2. Each data point gets assigned to the cluster to which its nearest medoid belongs.
3. For each data point of cluster i, its distance from all other data points is computed and added. The point of ith cluster for which the computed sum of distances from other points is minimal is assigned as the medoid for that cluster.
4. Steps (2) and (3) are repeated until convergence is reached i.e. the medoids stop moving.

Weka K-means Output:



SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1

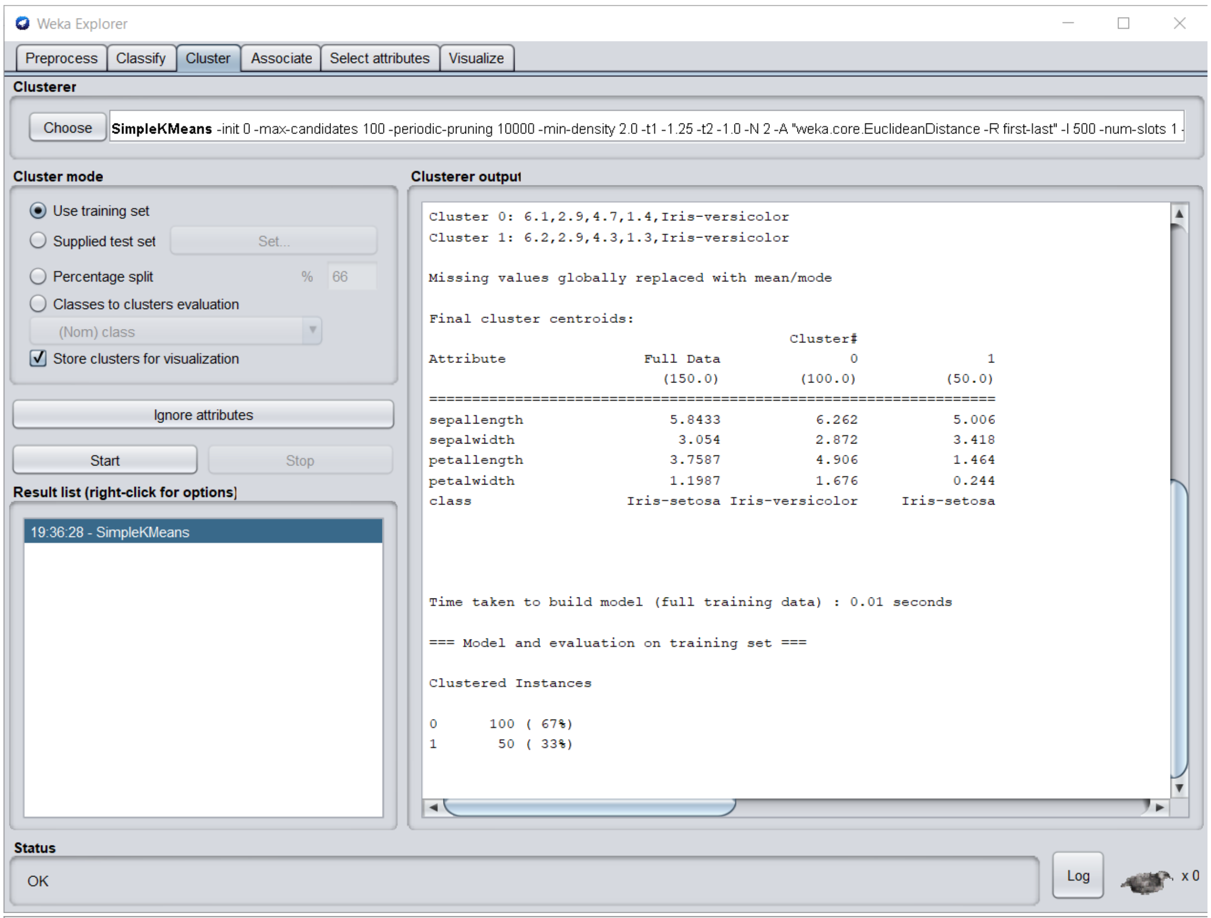
Cluster mode: [✓ Use training set] / Supplied test set / Percentage split % 66 / Classes to clusters evaluation (Nom) class / [✓ Store clusters for visualization]

Result list: 19:36:28 - SimpleKMeans

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruni
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode
```

Status: OK

68_Adnan Shaikh



Clusterer

**SimpleKMeans** -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1

Cluster mode

- (●) Use training set
- ( ) Supplied test set
- ( ) Percentage split % 66
- ( ) Classes to clusters evaluation
- (Nom) class
- [✓] Store clusters for visualization

Result list (right-click for options)

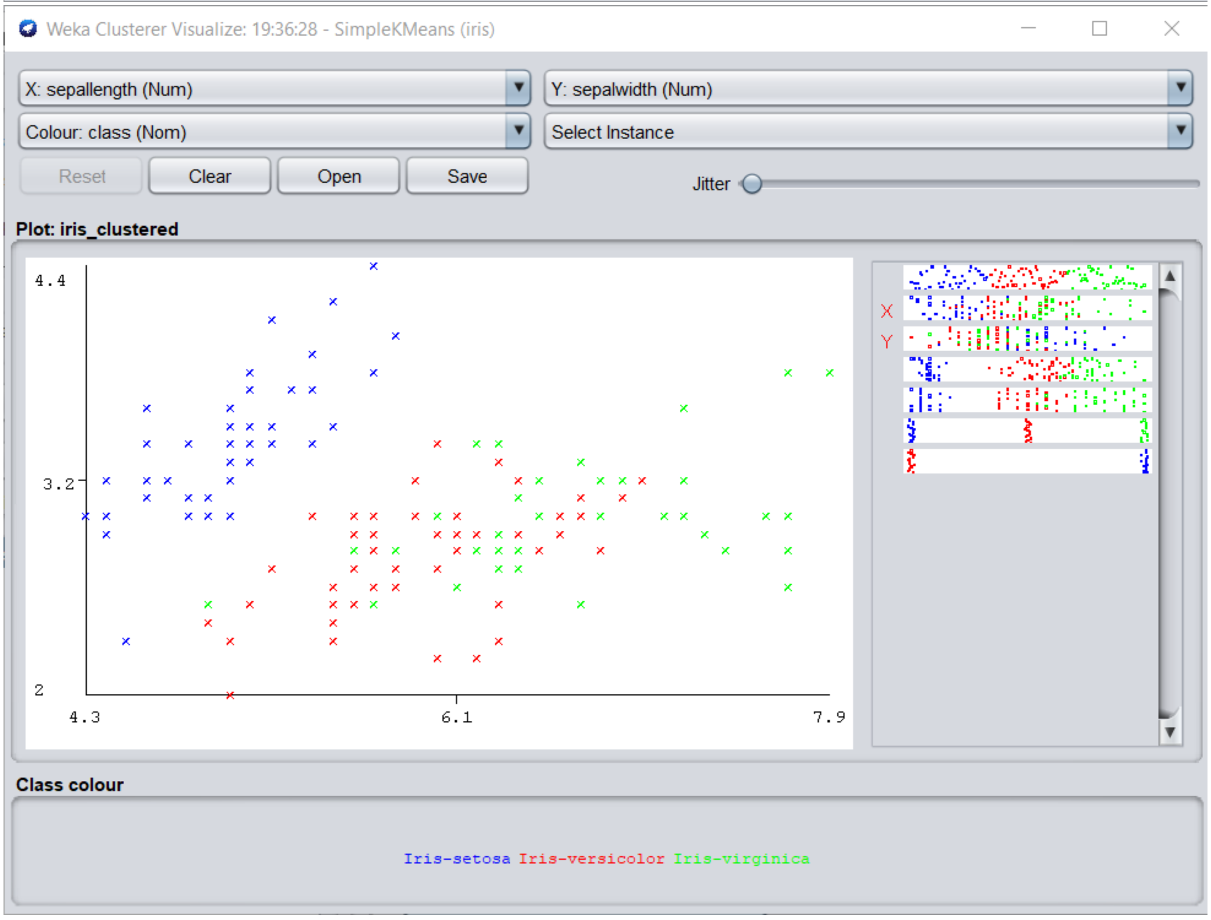
19:36:28 - SimpleKMeans

Clusterer output

```
Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode

Final cluster centroids:
                                        Cluster#
Attribute                 Full Data            0             1
                            (150.0)      (100.0)        (50.0)
================================================================
sepallength                  5.8433        6.262         5.006
sepalwidth                    3.054        2.872         3.418
petallength                  3.7587        4.906         1.464
petalwidth                   1.1987        1.676         0.244
class                   Iris-setosa Iris-versicolor    Iris-setosa




Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       100 ( 67%)
1        50 ( 33%)
```

Status

OK



X: sepallength (Num)

Y: sepalwidth (Num)

Colour: class (Nom)

Select Instance

Plot: iris_clustered

Class colour

Iris-setosa Iris-versicolor Iris-virginica

## Python K-means and K-medoids Output:

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.cluster import KMeans
        from sklearn_extra.cluster import KMedoids
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.datasets import load_iris
        import warnings
        warnings.filterwarnings("ignore")
```

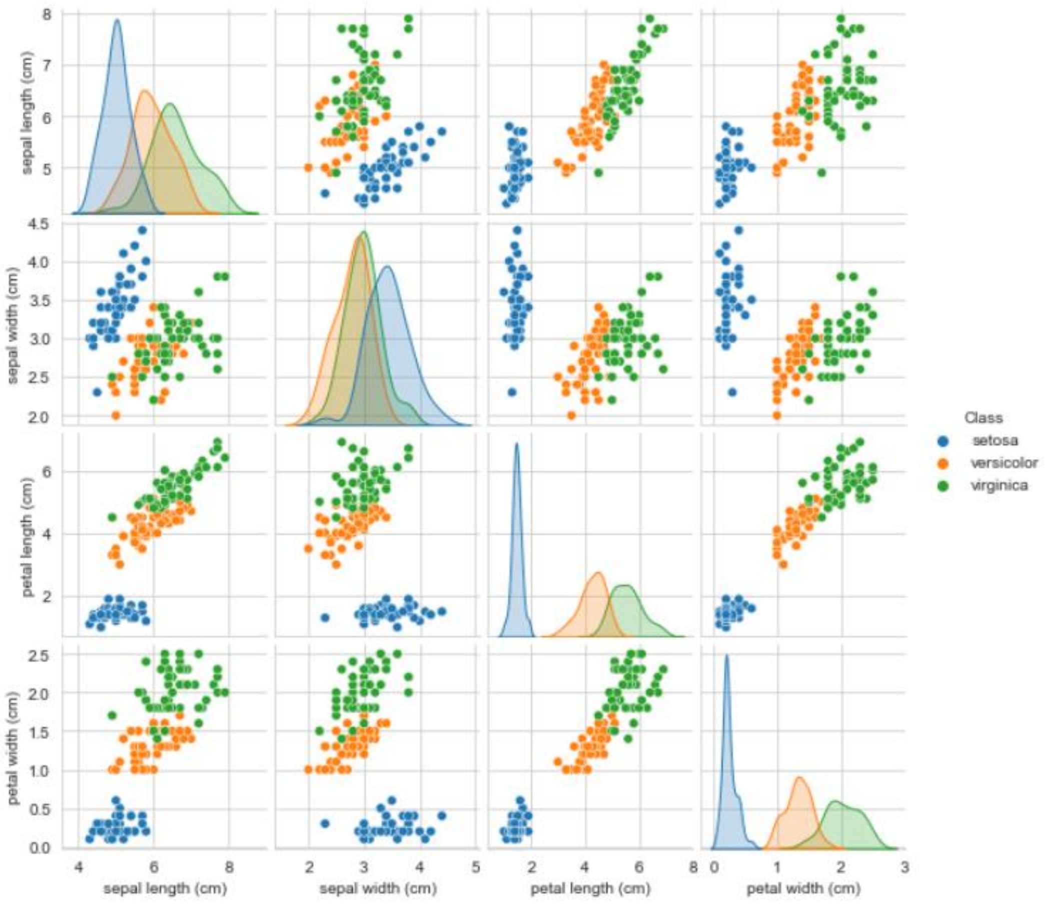
```
In [2]: dt = load_iris()
        X, y = load_iris(return_X_y=True)
```

```
In [3]: df = pd.DataFrame(np.column_stack((X,y)),columns = [*dt.feature_names,"Class"])
        df["Class"].replace([0,1,2],['setosa', 'versicolor', 'virginica'],inplace=True)
```

```
In [4]: df["Class"].value_counts()
```

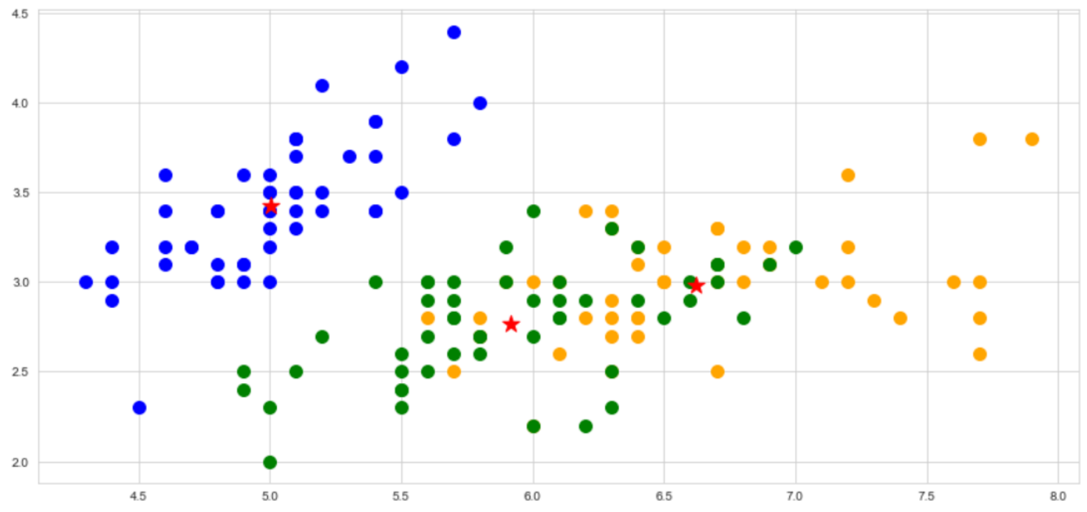
```
Out[4]: versicolor    50
        virginica     50
        setosa        50
        Name: Class, dtype: int64
```

```
In [12]: sns.set_style("whitegrid")
         sns.pairplot(df,hue="Class",size=2)
         plt.show()
```

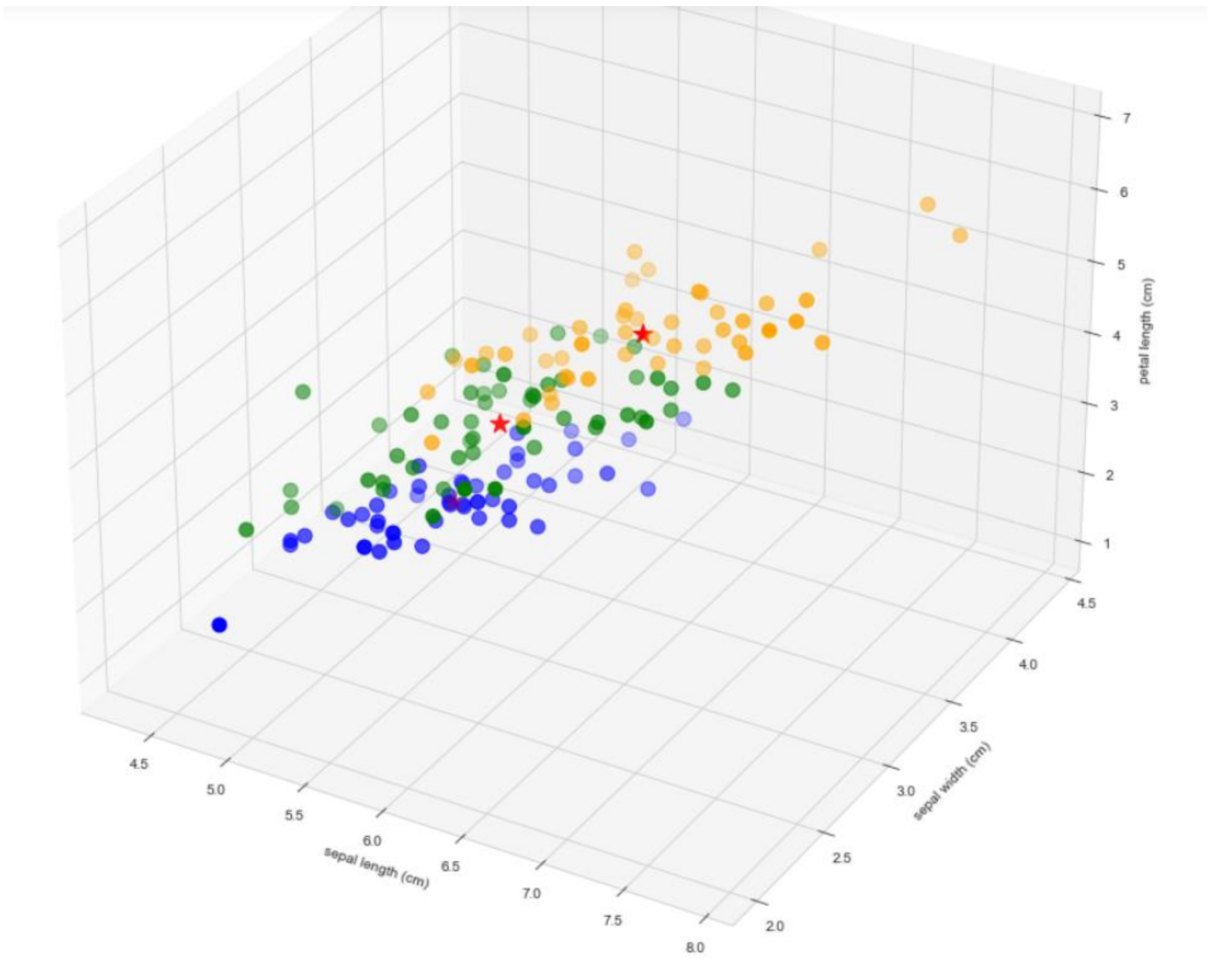


```
In [6]: kmeans = KMeans(n_clusters=3,init = "random",random_state=77,algorithm="auto")
        temp_kmeans = kmeans.fit_predict(df.replace(['setosa', 'versicolor', 'virginica'],[0,1,2]))
```

```
In [7]: plt.subplots(figsize=(15,7))
        plt.scatter(df.iloc[temp_kmeans == 0,0],df.iloc[temp_kmeans == 0,1],label="Setosa",color="blue",s=100)
        plt.scatter(df.iloc[temp_kmeans == 1,0],df.iloc[temp_kmeans == 1,1],label="Versicolor",color="orange",s=100)
        plt.scatter(df.iloc[temp_kmeans == 2,0],df.iloc[temp_kmeans == 2,1],label="Virginica",color="green",s=100)
        plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=200,marker="*",color="red")
        plt.show()
```

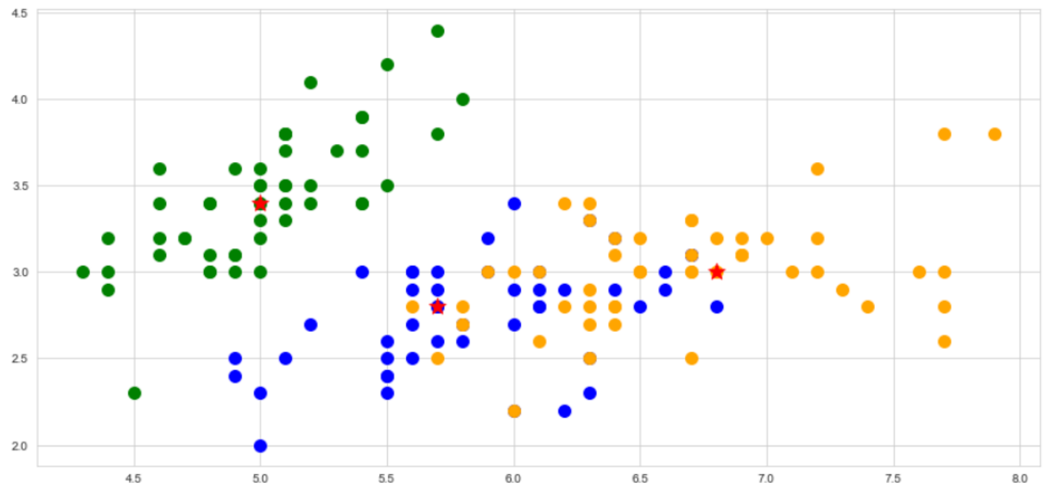


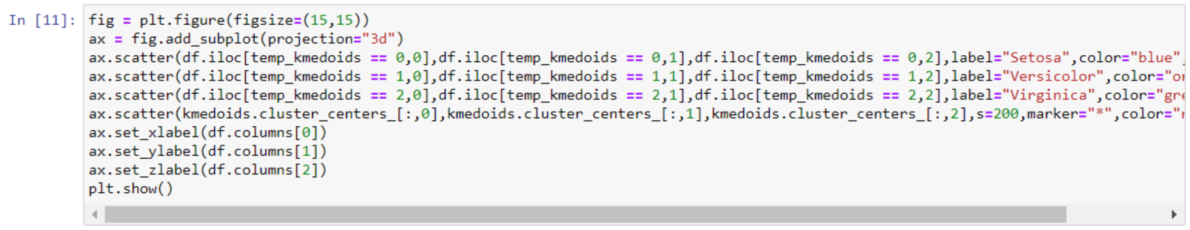
```
In [8]: fig = plt.figure(figsize=(15,15))
        ax = fig.add_subplot(projection="3d")
        ax.scatter(df.iloc[temp_kmeans == 0,0],df.iloc[temp_kmeans == 0,1],df.iloc[temp_kmeans == 0,2],label="Setosa",color="blue",s=100)
        ax.scatter(df.iloc[temp_kmeans == 1,0],df.iloc[temp_kmeans == 1,1],df.iloc[temp_kmeans == 1,2],label="Versicolor",color="orange",
        ax.scatter(df.iloc[temp_kmeans == 2,0],df.iloc[temp_kmeans == 2,1],df.iloc[temp_kmeans == 2,2],label="Virginica",color="green",s=
        ax.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],kmeans.cluster_centers_[:,2],s=200,marker="*",color="red")
        ax.set_xlabel(df.columns[0])
        ax.set_ylabel(df.columns[1])
        ax.set_zlabel(df.columns[2])
        plt.show()
```



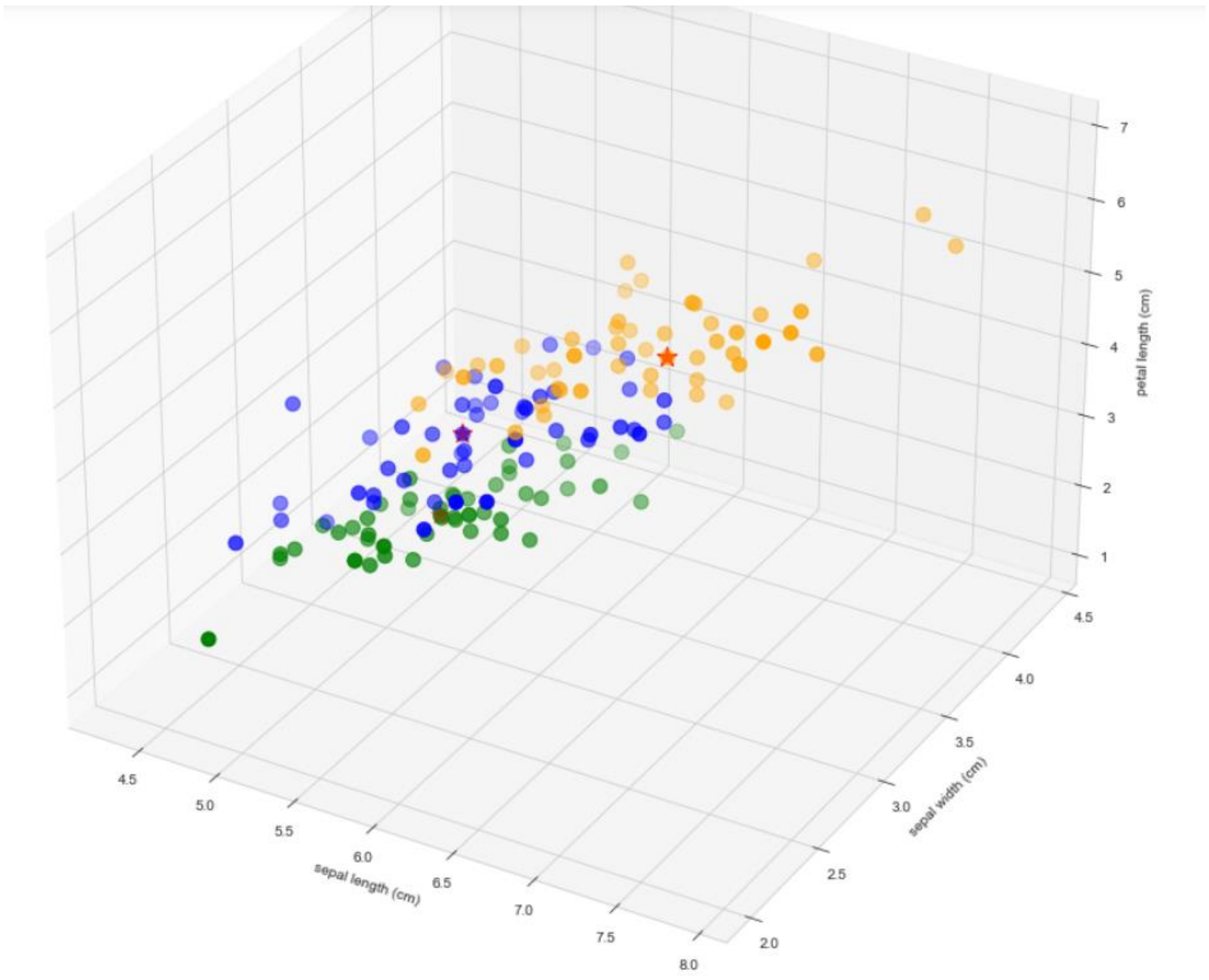
```
In [9]: kmedoids = KMedoids(n_clusters=3,init = "random",random_state=77)
        temp_kmedoids = kmedoids.fit_predict(df.replace(['setosa', 'versicolor', 'virginica'],[0,1,2]))
```

```
In [10]: plt.subplots(figsize=(15,7))
         plt.scatter(df.iloc[temp_kmedoids== 0,0],df.iloc[temp_kmedoids == 0,1],label="Setosa",color="blue",s=100)
         plt.scatter(df.iloc[temp_kmedoids == 1,0],df.iloc[temp_kmedoids == 1,1],label="Versicolor",color="orange",s=100)
         plt.scatter(df.iloc[temp_kmedoids == 2,0],df.iloc[temp_kmedoids == 2,1],label="Virginica",color="green",s=100)
         plt.scatter(kmedoids.cluster_centers_[:,0],kmedoids.cluster_centers_[:,1],s=200,marker="*",color="red")
         plt.show()
```





```
In [11]: fig = plt.figure(figsize=(15,15))
         ax = fig.add_subplot(projection="3d")
         ax.scatter(df.iloc[temp_kmedoids == 0,0],df.iloc[temp_kmedoids == 0,1],df.iloc[temp_kmedoids == 0,2],label="Setosa",color="blue"
         ax.scatter(df.iloc[temp_kmedoids == 1,0],df.iloc[temp_kmedoids == 1,1],df.iloc[temp_kmedoids == 1,2],label="Versicolor",color="or
         ax.scatter(df.iloc[temp_kmedoids == 2,0],df.iloc[temp_kmedoids == 2,1],df.iloc[temp_kmedoids == 2,2],label="Virginica",color="gre
         ax.scatter(kmedoids.cluster_centers_[:,0],kmedoids.cluster_centers_[:,1],kmedoids.cluster_centers_[:,2],s=200,marker="*",color="r
         ax.set_xlabel(df.columns[0])
         ax.set_ylabel(df.columns[1])
         ax.set_zlabel(df.columns[2])
         plt.show()
```



Conclusion: We have successfully implemented K-means and K-medoids algorithms on iris data set using Weka Tool and Python.