Clustering using K-Mean algorithm

In this case study we are generating the data set at run time randomly and apply user defined K-Mean algorithm.

```
import numpy as np
import pandas as pd
from copy import deepcopy
from matplotlib import pyplot as plt

def MarvellousKMean():
    print("_____");
    # Set three centers, the model should predict similar results
    center_1 = np.array([1,1])
    print(center_1)
    print("_____");
    center_2 = np.array([5,5])
    print(center_2)
    print("_____");
    center_3 = np.array([8,1])
    print(center_3)
    print("_____");
    # Generate random data and center it to the three centers
    data_1 = np.random.randn(7, 2) + center_1
    print("Elements of first cluster with size"+str(len(data_1)))
    print(data_1)
    print("_____");
    data_2 = np.random.randn(7,2) + center_2
    print("Elements of second cluster with size"+str(len(data_2)))
    print(data_2)
    print("_____");
    data_3 = np.random.randn(7,2) + center_3
    print("Elements of third cluster with size"+str(len(data_3)))
    print(data_3)
    print("_____");
    data = np.concatenate((data_1, data_2, data_3), axis = 0)
    print("Size of complete data set"+str(len(data)))
    print(data);
    print("_____");
    plt.scatter(data[:,0], data[:,1], s=7)
    plt.title('Marvellous Infosystems : Input Dataset')
    plt.show()
    print("_____");
    # Number of clusters
    k = 3

    # Number of training data
```

```
    n = data.shape[0]
    print("Total number of elements are",n)
    print("_____");
    # Number of features in the data
    c = data.shape[1]
    print("Total number of features are",c)
    print("_____");
        # Generate random centers, here we use sigma and mean to ensure it
represent the whole data
    mean = np.mean(data, axis = 0)
    print("Value of mean",mean)
    print("_____");
    # Calculate standard deviation
    std = np.std(data, axis = 0)
    print("Value of std",std)
    print("_____");
    centers = np.random.randn(k,c)*std + mean
    print("Random points are",centers)
    print("_____");
    # Plot the data and the centers generated as random
    plt.scatter(data[:,0], data[:,1],c='r', s=7)
    plt.scatter(centers[:,0], centers[:,1], marker='*', c='g', s=150)
    plt.title('Marvellous Infosystems : Input Datase with random centroid *')
    plt.show()
    print("_____");

    centers_old = np.zeros(centers.shape)   # to store old centers
    centers_new = deepcopy(centers)         # Store new centers

    print("Values of old centroids")
    print(centers_old)
    print("_____");

    print("Values of new centroids")
    print(centers_new)
    print("_____");

    data.shape
    clusters = np.zeros(n)
    distances = np.zeros((n,k))

    print("Initial distances are")
    print(distances)
    print("_____");

    error = np.linalg.norm(centers_new - centers_old)
    print("value of error is ",error);
    # When, after an update, the estimate of that center stays the same, exit loop
```

```
    while error != 0:
        print("value of error is ",error);
        # Measure the distance to every center
        print("Measure the distance to every center")
        for i in range(k):
            print("Iteration number ",i)
            distances[:,i] = np.linalg.norm(data - centers[i], axis=1)

        # Assign all training data to closest center
        clusters = np.argmin(distances, axis = 1)

        centers_old = deepcopy(centers_new)

        # Calculate mean for every cluster and update the center
        for i in range(k):
            centers_new[i] = np.mean(data[clusters == i], axis=0)
        error = np.linalg.norm(centers_new - centers_old)
    # end of while
    centers_new

    # Plot the data and the centers generated as random
    plt.scatter(data[:,0], data[:,1], s=7)
    plt.scatter(centers_new[:,0], centers_new[:,1], marker='*', c='g', s=150)
    plt.title('Marvellous Infosystems : Final data with Centroid')
    plt.show()

def main():
    print("---- Marvellous Infosystems by Piyush Khairnar-----")

    print("Unsuervised Machine Learning")

    print("Clustering using K Mean Algorithm")

    MarvellousKMean()

if __name__ == "__main__":
    main()
```

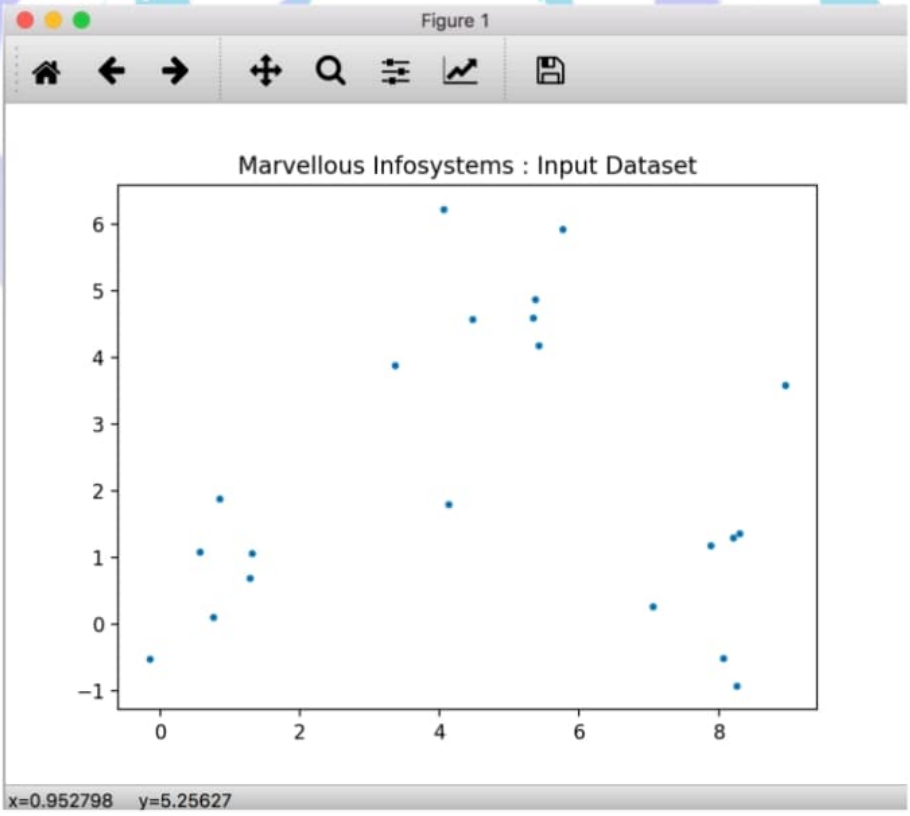
Output of above application :

```
---- Marvellous Infosystems by Piyush Khairnar——

Unsuervised Machine Learning
Clustering using K Mean Algorithm

_____________________________
[1 1]

_____________________________
[5 5]

_____________________________
[8 1]

_____________________________
Elements of first cluster with size7
[[ 0.76455047  0.1057342 ]
 [ 0.57413358  1.07954625]
 [-0.14432049 -0.52217058]
 [ 1.31367866  1.05880002]
 [ 0.85572402  1.87967134]
 [ 1.28881943  0.68895633]
 [ 4.12989909  1.80041537]]
_____________________________
Elements of second cluster with size7
[[5.34096455 4.5997487 ]
 [4.06005167 6.22535108]
 [5.42174515 4.17713132]
 [3.36119862 3.88051506]
 [4.4704562  4.57687409]
 [5.37072318 4.87360285]
 [5.76651027 5.92478869]]
_____________________________
Elements of third cluster with size7
[[ 8.29482744  1.36251521]
 [ 8.20703246  1.29568945]
 [ 8.06619888 -0.50865465]
 [ 8.94870609  3.58106472]
 [ 8.25722023 -0.9248197 ]
 [ 7.05976628  0.26199046]
 [ 7.88080164  1.17437363]]
_____________________________
Size of complete data set21
[[ 0.76455047  0.1057342 ]
 [ 0.57413358  1.07954625]
 [-0.14432049 -0.52217058]
 [ 1.31367866  1.05880002]
 [ 0.85572402  1.87967134]
 [ 1.28881943  0.68895633]
```

```
[ 4.12989909  1.80041537]
[ 5.34096455  4.5997487 ]
[ 4.06005167  6.22535108]
[ 5.42174515  4.17713132]
[ 3.36119862  3.88051506]
[ 4.4704562   4.57687409]
[ 5.37072318  4.87360285]
[ 5.76651027  5.92478869]
[ 8.29482744  1.36251521]
[ 8.20703246  1.29568945]
[ 8.06619888 -0.50865465]
[ 8.94870609  3.58106472]
[ 8.25722023 -0.9248197 ]
[ 7.05976628  0.26199046]
[ 7.88080164  1.17437363]]
```

Total number of elements are 21

Total number of features are 2



Value of mean [4.72803273 2.21862495]

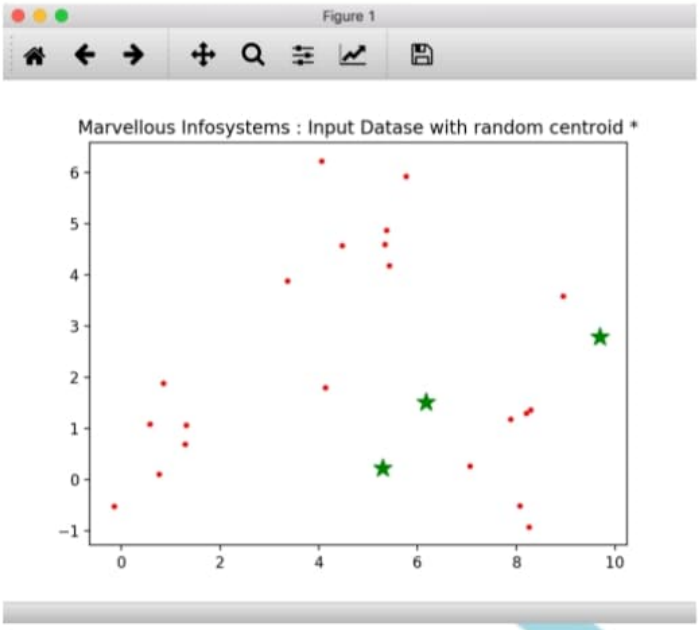
Value of std [2.94300319 2.15577759]

Random points are [[9.68991885 2.78162963]
 [6.17016879 1.5050397]
 [5.29720489 0.21581884]]

Values of old centroids
[[0. 0.]
 [0. 0.]
 [0. 0.]]

Values of new centroids
[[9.68991885 2.78162963]
 [6.17016879 1.5050397]
 [5.29720489 0.21581884]]

Initial distances are
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]



value of error is 13.04128351259481
value of error is 13.04128351259481

Measure the distance to every center
Iteration number 0
Iteration number 1
Iteration number 2
value of error is 3.847400069439621

Measure the distance to every center
Iteration number 0
Iteration number 1
Iteration number 2