

## Problem 4

```
In [1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import sys
import random
random.seed(0)
np.random.seed(0)
```

```
In [2]: column_names = ['sepal length', 'sepal width', 'petal length', 'petal width', 'class']
```

```
In [3]: # read iris dataset
iris = pd.read_csv('iris.data', names=column_names, index_col=False)
```

```
In [4]: iris.head()
```

```
Out[4]:
```

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [5]: iris_df_new = pd.DataFrame() # creating new dataframe
```

```
In [6]: # this function assigns values
def get_class(x):
    if x=='Iris-setosa': # Iris-setosa =0
        return 0
    elif(x=='Iris-versicolor'): # Iris-versicolor =1
        return 1
    else:
        return 2 # Iris Virginica =3
```

```
In [7]: # Assigning x1 and x2
iris_df_new['x1']=iris['sepal length']/iris['sepal width']
iris_df_new['x2']=iris['petal length']/iris['petal width']
iris_df_new['class']=iris['class']
iris_df_new['class_enc']=iris_df_new['class'].apply(lambda x: get_class(x))
```

```
In [8]: iris_df_new.head()
```

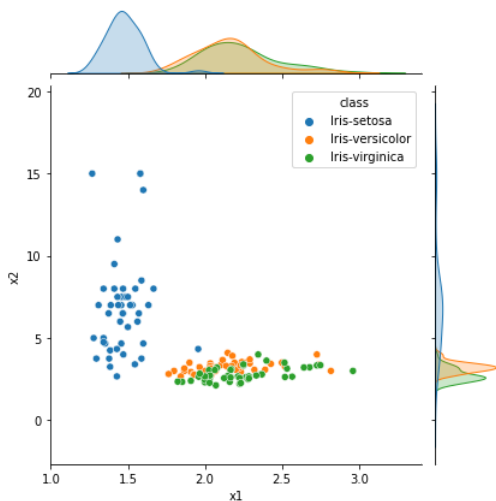
```
Out[8]:
```

	x1	x2	class	class_enc
0	1.457143	7.0	Iris-setosa	0
1	1.633333	7.0	Iris-setosa	0
2	1.468750	6.5	Iris-setosa	0
3	1.483871	7.5	Iris-setosa	0
4	1.388889	7.0	Iris-setosa	0

```
In [9]: # showing the clusters
plt.figure(figsize=(12,8))
data = iris_df_new.drop('class_enc', axis=1)
sns.jointplot(x='x1', y='x2', data=data, hue='class')
```

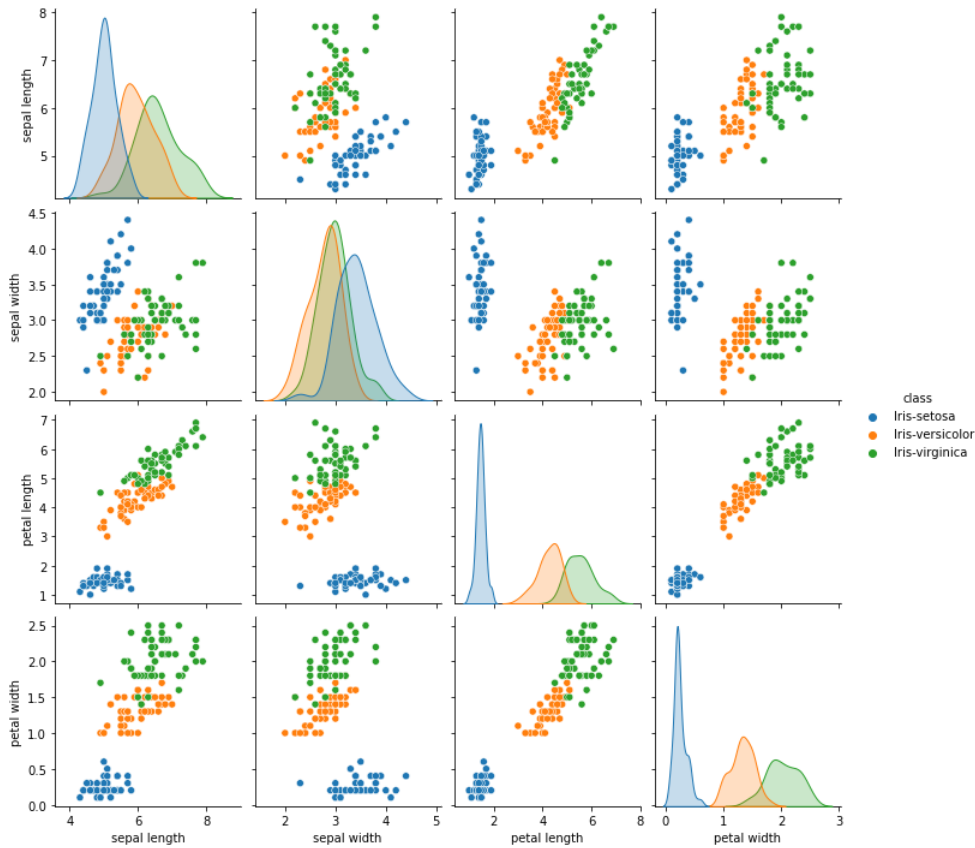
```
Out[9]: <seaborn.axisgrid.JointGrid at 0x7f87c17df0d0>
```

<Figure size 864x576 with 0 Axes>



```
In [10]: sns.pairplot(data=iris,hue='class')
```

```
Out[10]: <seaborn.axisgrid.PairGrid at 0x7f87c374dbd0>
```



```
In [11]: # Euclidean Distance - to calculate distance
```

```
def get_distance(x1,x2,y1,y2):  
    d1=(y1-x1)**2  
    d2=(y2-x2)**2  
    return (d1+d2)
```

```
In [12]: ## Initialize the centroids function
```

```
num_rows = iris_df_new.shape[0]  
def initialize_kmeans_plus_plus(k):  
    centroids_x1=[]  
    centroids_x2=[]  
    # first point is selected randomly  
    rand_num = np.random.randint(num_rows-1)  
    first_centroid_x1 = iris_df_new['x1'].iloc[rand_num]  
    first_centroid_x2 = iris_df_new['x2'].iloc[rand_num]  
    centroids_x1.append(first_centroid_x1)  
    centroids_x2.append(first_centroid_x2)  
  
    for k_id in range(k-1):  
        dist=[]  
        # for each of the point  
        for ind in range(num_rows):  
            x1=iris_df_new['x1'].iloc[ind]  
            x2=iris_df_new['x2'].iloc[ind]  
            min_dist = sys.maxsize  
            # we determine the minimum distance from current all centroids  
            for cd in range(len(centroids_x1)):  
                y1=centroids_x1[cd]  
                y2=centroids_x2[cd]  
                distance = get_distance(x1,x2,y1,y2)  
                min_dist = min(min_dist,distance)  
            dist.append(min_dist)  
        # then select the data point which is at maximum distance,  
        # and assign that data point as new centroid  
        mx_ind = np.argmax(np.array(dist))  
        centroids_x1.append(iris_df_new['x1'].iloc[mx_ind])  
        centroids_x2.append(iris_df_new['x2'].iloc[mx_ind])  
  
    return centroids_x1,centroids_x2
```

```

In [13]: ## Kmean++ algorithm
def kmeans_plus_plus(k,centroids_x1,centroids_x2):
    # number of datapoints
    num_rows = iris_df_new.shape[0]
    cent_map=[] # stores centroids and corresponding assigned points mapping
    sums_x1=[] # stores sum of x1 feature of all points within a cluster
    sums_x2=[] # stores sum of x2 feature of all points within a cluster
    for ind in range(k):
        cent_map.append([])
        sums_x1.append(0)
        sums_x2.append(0)
    # for each data point
    for ind in range(num_rows):
        x1=iris_df_new['x1'].iloc[ind]
        x2=iris_df_new['x2'].iloc[ind]
        min_dist = sys.maxsize
        mid=-1
        # we calculate the distances between point and all centroids
        # point will be mapped to the centroid with minimum distance from it
        for cd in range(len(centroids_x1)):
            y1=centroids_x1[cd]
            y2=centroids_x2[cd]
            distance = get_distance(x1,x2,y1,y2)
            if mid== -1:
                mid=cd
                min_dist = distance
            else:
                if(distance<min_dist):
                    mid=cd
                    min_dist = distance
        cent_map[mid].append(ind)
    # calculating new centroids
    for ind in range(len(cent_map)):
        for cd in range(len(cent_map[ind])):
            index = cent_map[ind][cd]
            sums_x1[ind]+=iris_df_new['x1'].iloc[index]
            sums_x2[ind]+=iris_df_new['x2'].iloc[index]

    for ind in range(len(cent_map)):
        length = len(cent_map[ind])
        #print(length)
        if(length>0):
            sums_x1[ind]=sums_x1[ind]/float(length)
            sums_x2[ind]=sums_x2[ind]/float(length)

    centroids_x1=sums_x1
    centroids_x2=sums_x2
    # returning new centroids, mapping of points to corresponding centroids
    return centroids_x1,centroids_x2,cent_map

```

```

In [14]: # 3. K=1 to 5 and 5 iterations for each
maps_of_points=[]
final_centroids=[]
initial_centroids=[]
for k_value in range(1,6):
    centroids_x1,centroids_x2=initialize_kmeans_plus_plus(k_value)
    first_cent=[]
    first_cent.append(centroids_x1)
    first_cent.append(centroids_x2)
    initial_centroids.append(first_cent)
    final_map=[]
    for turn in range(50):
        centroids_x1,centroids_x2,cent_map = kmeans_plus_plus(k_value,centroids_x1,centroids_x2)
        final_map=cent_map
    maps_of_points.append(final_map)
    centroids =[]
    centroids.append(centroids_x1)
    centroids.append(centroids_x2)
    final_centroids.append(centroids)

```

```
In [15]: for k in range(len(final_centroids)):
print("centroids for k_value = ",k+1)
print("Initial Centroids")
for t in range(len(initial_centroids[k][0])):
    print(initial_centroids[k][0][t],initial_centroids[k][1][t])
print("\nFinal Centroids")
for t in range(len(final_centroids[k][0])):
    print(final_centroids[k][0][t],final_centroids[k][1][t])
print("\n\n*****\n\n")
```

```
centroids for k_value = 1
Initial Centroids
1.4374999999999998 6.999999999999999
```

```
Final Centroids
1.9551444308694617 4.367166423691872
```

\*\*\*\*\*

```
centroids for k_value = 2
Initial Centroids
2.0263157894736845 3.0454545454545454
1.2682926829268295 15.0
```

```
Final Centroids
1.9777366323385024 3.920104640236229
1.4936180294304975 13.5
```

\*\*\*\*\*

```
centroids for k_value = 3
Initial Centroids
2.148148148148148 4.1
1.2682926829268295 15.0
1.411764705882353 9.499999999999998
```

```
Final Centroids
2.0928950450949397 3.1673129041992576
1.5220456333595596 14.8
1.4788141804347774 7.3678160919540225
```

\*\*\*\*\*

```
centroids for k_value = 4
Initial Centroids
2.1724137931034484 3.1111111111111107
1.2682926829268295 15.0
1.411764705882353 9.499999999999998
1.5714285714285714 6.5
```

```
Final Centroids
2.1066310718427834 3.1351604990097712
1.5220456333595596 14.8
1.4838758832268053 8.625
1.4623690090317294 6.724637681159421
```

\*\*\*\*\*

```
centroids for k_value = 5
Initial Centroids
1.5806451612903227 15.0
2.0714285714285716 2.125
1.5882352941176472 8.5
1.2777777777777777 5.0
1.4333333333333333 11.0
```

```
Final Centroids
1.5220456333595596 14.8
2.172285144648706 2.984749635537806
1.48232743051511 7.211538461538462
1.5910182803613162 4.3882352941176475
1.4225490196078432 10.25
```

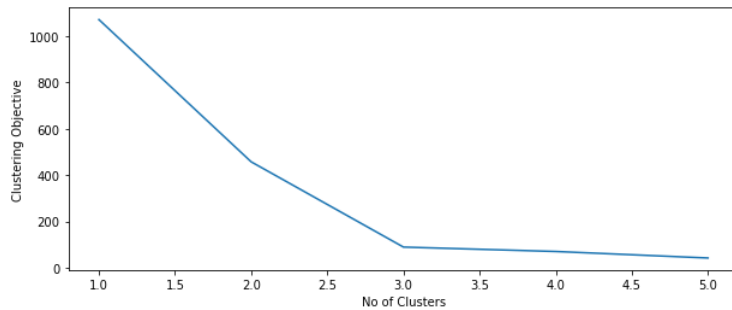
\*\*\*\*\*

```
In [16]: # calculating inertia (sum of square of distance of points from their cluster centroids)
acc=[]
for u in range(len(maps_of_points)):
    fmap=maps_of_points[u]
    centroids = final_centroids[u]
    sum_dist=0
    for t in range(len(fmap)):
        pts = fmap[t]
        mn_dist=0
        for y in range(len(pts)):
            dist = get_distance(centroids[0][t],centroids[1][t],
                                iris_df_new['x1'].iloc[pts[y]],iris_df_new['x2'].iloc[pts[y]])
            mn_dist+=dist
        sum_dist+=mn_dist
    acc.append(sum_dist)
    print("mean =", sum_dist, "for K = ",u+1)
```

```
mean = 1071.228686156884 for K = 1
mean = 457.22352154601197 for K = 2
mean = 89.17750273047949 for K = 3
mean = 70.1996474420087 for K = 4
mean = 42.03957396126128 for K = 5
```

```
In [17]: plt.figure(figsize=(10,4))
index=range(1,6)
sns.lineplot(x=index, y=acc)
plt.xlabel('No of Clusters')
plt.ylabel('Clustering Objective')
```

```
Out[17]: Text(0, 0.5, 'Clustering Objective')
```



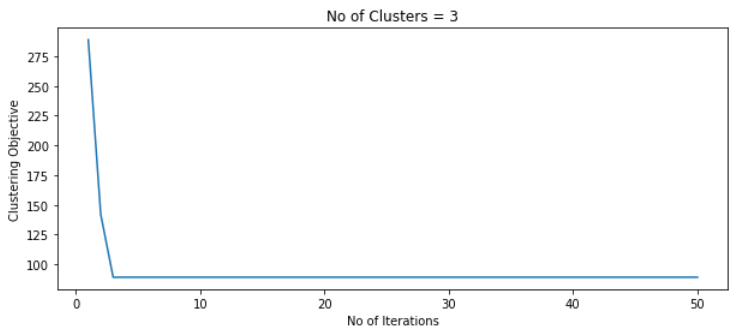
**4) From K =1 to 3, the inertia (sum of square of distance of points from their cluster centroids) decreases highly, then the decrease rate is very less. So, I will choose K=3**

```
In [18]: #chosen cluster = 3
centroids_x13,centroids_x23=initialize_kmeans_plus_plus(3)
print("initial chosen centroids with k=3 ")
for u in range(len(centroids_x13)):
    print(centroids_x13[u],centroids_x23[u])
```

```
initial chosen centroids with k=3
1.3783783783783783 3.75
1.5806451612903227 15.0
1.411764705882353 9.499999999999998
```

[illegible]

```
Text(0.5, 1.0, 'No of Clusters = 3')
```



```

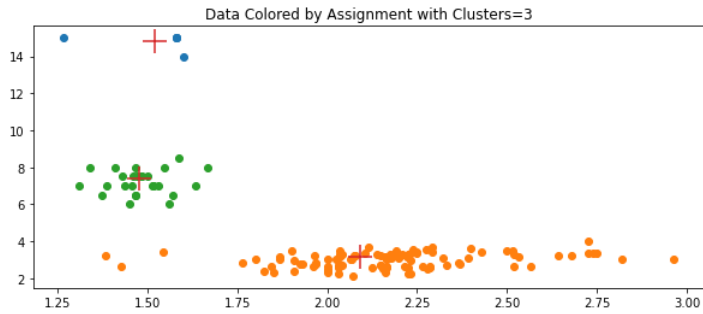
[15, 6, 15, 16, 17, 19, 21, 22, 23, 26, 31, 40, 41, 43, 44, 45, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 1
06, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 13
5, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149], [9, 12, 32, 34, 37], [0, 1, 2, 3, 4, 7, 8, 10, 11, 13, 14, 18, 20, 24, 2
5, 27, 28, 29, 30, 33, 35, 36, 38, 39, 42, 46, 47, 48, 49]]
Final Cluster Centers for K=3
2.0928950450949397 3.1673129041992576
1.5220456333595596 14.8
1.4788141804347774 7.3678160919540225

```

```
In [22]: x3=[]
y3=[]
for t in range(len(final_map3)):
    pts = fmap[t]
    lx=[]
    ly=[]
    for y in range(len(pts)):
        lx.append(iris_df_new['x1'].iloc[pts[y]])
        ly.append(iris_df_new['x2'].iloc[pts[y]])
    x3.append(lx)
    y3.append(ly)
```

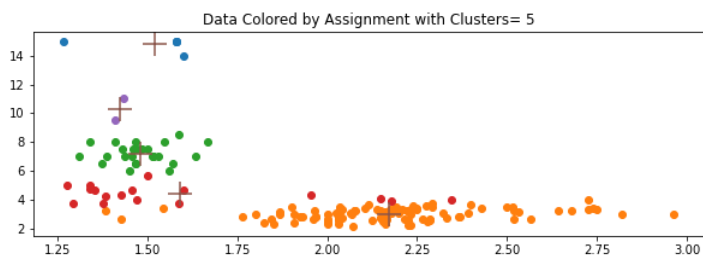
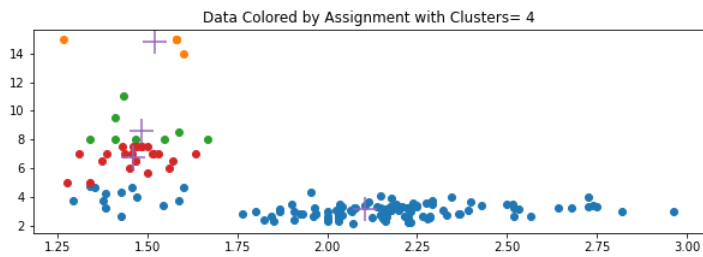
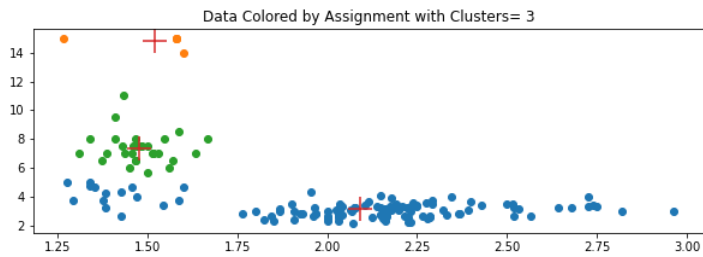
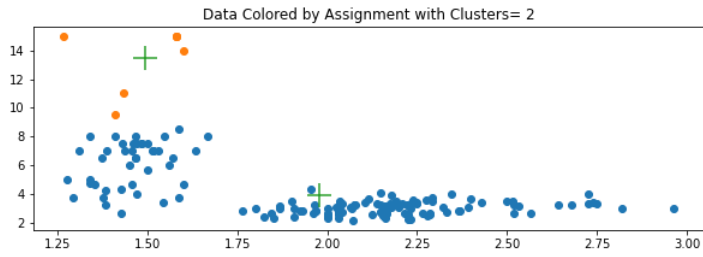
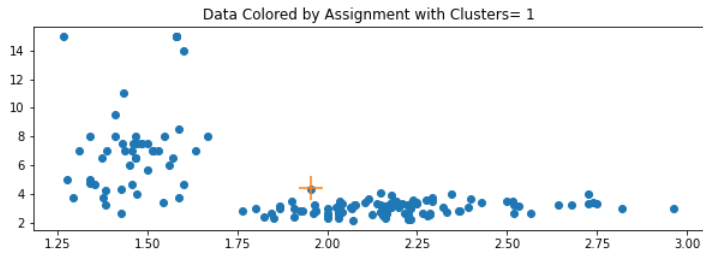
```
In [23]: # centroids are in + sign
```

```
plt.figure(figsize=(10,4))
for u in range(len(x3)):
    plt.scatter(x3[u],y3[u])
plt.scatter(centroids_x13,centroids_x23,marker='+',s=400)
plt.title('Data Colored by Assignment with Clusters=3')
plt.show()
```



```
In [24]: X=[]
Y=[]
for u in range(len(maps_of_points)):
    fmap=maps_of_points[u]
    centroids = final_centroids[u]
    mx=[]
    my=[]
    for t in range(len(fmap)):
        pts = fmap[t]
        mn_dist=0
        lx=[]
        ly=[]
        for y in range(len(pts)):
            lx.append(iris_df_new['x1'].iloc[pts[y]])
            ly.append(iris_df_new['x2'].iloc[pts[y]])
        mx.append(lx)
        my.append(ly)
    X.append(mx)
    Y.append(my)
```

```
In [25]: for y in range(5):
plt.figure(figsize=(10,18))
plt.subplot(5,1,y+1)
for u in range(len(X[y])):
plt.scatter(X[y][u],Y[y][u])
plt.scatter(final_centroids[y][0],final_centroids[y][1],marker='+',s=400)
plt.title('Data Colored by Assignment with Clusters= '+str(y+1))
plt.show()
```



In [ ]: