Assignment 1

Presented by: -
Ashutosh Samal (2403res15)

K-Means and K-medoids

List of user defined functions: -

1-

```
# we need 3 centroids so select one as minimum , max, and median
def get_initial_centroids(data):
    c1=data.apply(lambda x: (x.max()),axis=0)    ## Axis =0 means it will take data from each column
    c2=data.apply(lambda x: (x.min()),axis=0)    ## Axis =0 means it will take data from each column
    c3=data.apply(lambda x: (x.median()),axis=0) ## Axis =0 means it will take data from each column
    centroid=pd.concat([c1,c2,c3],axis=1)        ## Intial centroids
    return centroid
```

This function initializes 3 centroids which looks like this: -

	0	1	2
SepalLengthCm	1.0	0.0	0.416667
SepalWidthCm	1.0	0.0	0.416667
PetalLengthCm	1.0	0.0	0.567797
PetalWidthCm	1.0	0.0	0.500000

2-

```
2]: #return squered sum distance of each points in a dataset fr
    def SSE(data,centroids):
        return ((data-centroids)**2).sum(axis=1)
```

It finds SSE from one given centroid and given pandas data frame. We will get SSE for each data points.

4-

```
[30]: ## Returns Mean of a given column. Used in calculating centroids by using mean of all points
      def updateByMean(groupdata):
          return groupdata.sum()/groupdata.count()

[14]: #Retrun the median row wich have lowest intra cluster distance among all the points
      def calculatemedioids(groupdata):
          total_distance=10**10
          for i in groupdata.index:
              dist_i=0
              for j in groupdata.index:
                  dist_i+=((groupdata.loc[i]-groupdata.loc[j])**2).sum()

              if dist_i<total_distance:
                  centroid=groupdata.loc[i]
                  total_distance=dist_i
          return centroid
```

This two-function used to calculate medoids. Given a pandas data frame 1st function calculate mean of each column/feature. And other function returns the median of the data frame given.
It takes data frame of each cluster and returns the new centroids.

5-

```
# Assign a cluster to each points (0,1,2)
def get_labels(data,centroids):

    distance=centroids.apply(lambda x: SSE(data,x)) # calculate distance or SSE  for each points from each centroids


    return distance.idxmin(axis=1) # return the cluster with minimum SSE for each points
```

This function will return labels as per the given data frame and the 3 centroids.
Here for each centroid, we apply SSE to whole dataset. For each record we will get sse w.r.t to each cluster as shown below: -

Distance dataframe look like this: -

	0	1	2
149	0.094235	0.892130	0.051493
13	1.790214	0.076200	0.820471
35	1.461069	0.010592	0.649156
125	0.028179	1.391084	0.268991
139	0.002983	1.337441	0.211388
...	...	...	...
105	0.068228	1.905364	0.458157
74	0.183111	0.621176	0.027446
141	0.019152	1.408017	0.250984
15	1.399995	0.208843	0.903449
80	0.491135	0.482721	0.059236

Idxmin will return the index with lowest value (lowest SSE value) along axis =1 ,for each record which will be its cluster (0,1,2)

Returned data frame looks like this: -

```
149    2
13     1
35     1
125    0
139    0
      ..
105    0
74     2
141    0
15     1
80     2
Length: 150, dtype: int64
```

Labeled for each row/record.

6-

```
]: def update_centroids(data,labels,type):
    """
     type could be "Mean" or "Median"
     By "Mean" function will calculate centroids by mean of all the points
     By "Median" function will calculate centroids by medain of all the points
    """

    #Did groupby labels to group all the data points in each cluster and the calculate mean or median of the points within the cluster

    if type=="Mean":
        return data.groupby(labels).apply(lambda x: updateByMean(x) ).T  #Transposed to get the same structure of centroids as  initial
    elif type=="Median":
        # return data.groupby(labels).apply(lambda x: updateByMediods(x) ).T
        return data.groupby(labels).apply(lambda x:calculatemedioids(x)).T
```

This function used to calculate centroid as per the inputs (mean or median).
I used group by to group data by labels so 3 groups (3 labels are there) will be created and will be passed one be one to calculatemedioids or updateByMean methods and will get 3 new centroids.
Finally transposed it to make it dimension correct and we get centroids as shown in 1st function.

7: -

```
def plot_clusters(data,labels):

    """
    Total 4 plots for the given data and their labels.
    1:- SepalLengthCm vs PetalLengthCm
    2:- SepalLengthCm vs PetalWidthCm
    3:- SepalWidthCm vs PetalLengthCm
    4:- SepalWidthCm vs PetalWidthCm
    """

    fig=plt.figure()


    plot1=fig.add_subplot(221)
    plot2=fig.add_subplot(222)
    plot3=fig.add_subplot(223)
    plot4=fig.add_subplot(224)


    plot1.scatter(data["SepalLengthCm"] ,data["PetalLengthCm"] , c=labels)

    # plot1.set_title("SepalLengthCm vs PetalLengthCm")
    plot1.set_xlabel("SepalLengthCm")
    plot1.set_ylabel("PetalLengthCm")

    plot2.scatter(data["SepalLengthCm"] ,data["PetalWidthCm"] , c=labels)

    # plot2.set_title("SepalLengthCm vs PetalWidthCm")
    plot2.set_xlabel("SepalLengthCm")
    plot2.set_ylabel("PetalWidthCm")

    plot3.scatter(data["SepalWidthCm"] ,data["PetalLengthCm"] , c=labels)

    # plot3.set_title("SepalWidthCm vs PetalLengthCm")
    plot3.set_xlabel("SepalWidthCm")
    plot3.set_ylabel("PetalLengthCm")

    plot4.scatter(data["SepalWidthCm"] ,data["PetalWidthCm"] , c=labels)

    # plot4.set_title("SepalWidthCm vs PetalWidthCm")
    plot4.set_xlabel("SepalWidthCm")
    plot4.set_ylabel("PetalWidthCm")
    fig.tight_layout()

    fig.show()
```

This function is used to plot the graphs. We have 4 features in 2d we can plot 6 types of graphs, but I only plotted 4 graphs in one figure for each iteration.
I will be calling this function in each iteration.
Took color (c) as labels so for each index in dataset it will pick color as per labels.
So that we can have a graph with 3 different color one for each cluster.

8: -

[18]:
```
"""
Function takes an array of index that belongs to a particular cluster and check the labels of each index from main dataframe provided to us
Print count of each labels in the given cluster
Used to undestand how the clustering has performed
"""
def label_for_each_group(index):
    count_Iris_setosa=0
    count_Iris_virginica=0
    count_Iris_versicolor=0
    for i in index:
        if iris_data.loc[i]["Species"]=='Iris-versicolor':
            count_Iris_versicolor+=1
        elif iris_data.loc[i]["Species"]=='Iris-virginica':
            count_Iris_virginica+=1
        else :
            count_Iris_setosa+=1

    print("Count of each labels in given cluster:-")
    print("Iris-versicolor:-",count_Iris_versicolor)
    print("Iris-virginica:-",count_Iris_virginica)
    print("Iris-setosa:-",count_Iris_setosa)
```

As given data is pre labeled, we want to see how many records are correctly clustered.
This function will take index of records belonging to a particular cluster and print all the labeled count present in that cluster.

9:-

```
def Clustering(initial_centroids,Type):
    """
     Type could be "Mean" or "Median"
     By "Mean" function will calculate K-means
     By "Median" function will calculate K-Mediods
    """

    delta=10**-7  ## centroid different must be less than or equal to this
    current_centroid=initial_centroids
    centroid_deff=10*4
    iteration=1

    while centroid_deff>delta:
        print("interation is :- ",iteration)
        #assign each point to centroids
        labels=get_labels(train_data,current_centroid)

        SSE_error = current_centroid.apply(lambda x : SSE(train_data,x)).min(axis=1).sum()

        """
        SSE function will return SSE for each points from each centroids.
        By using Min we take the SSE from nearest centroid(with lowest SSE) of each points and add them up
        By this we are calclulating sum of intra-cluster SSE
        """

        print("SSE is :-" ,SSE_error)



        #Update centroids
        new_centroids=update_centroids(train_data,labels,Type)

        #centroids diff
        centroid_deff=abs((current_centroid-new_centroids)).sum().sum()

        print("centroid diff:- " ,centroid_deff)

        iteration+=1


        plot_clusters(train_data,labels)
        current_centroid=new_centroids
    return current_centroid
```

This is main function to evaluate k-means or k-medoids algorithm.

Uses all the helper function defined earlier and returns the final centroids, also prints SSE, plot clustered graph at each iteration.

Code Logic and Output: -

Reading and preprocessing: -

```
[2]: iris_data=pd.read_csv("Iris.csv")

[3]: features=["SepalLengthCm","SepalWidthCm","PetalLengthCm","PetalWidthCm"]

[4]: iris_data=iris_data.dropna(subset=features)

[5]: iris_data.describe()
```

[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Here we can see range of all the column/features are different.
SepalLengthCm is between 4.3 to 7.9
SepalWidthCm is between 3.05 to 4.4
PetalLengthCm is between 3.75 to 6.9
PetalWidthCm is between 1.19 to 2.5

So we need to normalize this before proceeding to have the range of all the features between 0-1.

And we can see that standard deviation is also very low(compared to mean) so we don't have risk of any outliers

```
[6]: iris_data = iris_data.sample(frac=1)  ## Randomize the dataset

[7]: ## Train and Test/validate split

[8]: train_size=150
     train_data=iris_data[0:train_size][features]
     test_data=iris_data[train_size:][features]

[9]: ##Scale all data between 0 to 1 as the value of all the column have different ranges

[10]: train_data=(train_data-train_data.min())/(train_data.max()-train_data.min())
      test_data=(test_data-test_data.min())/(test_data.max()-test_data.min())

[ ]:
```

Line 6 – Randomized the data and then

split the data in train test (I have taken all data in train but we can split if needed)

line 10th :-normalized the data by using formula (x-min(x))/(max(x)-min(x))

All data will be in the range of 0-1

K- medoids: -

```
[27]: final_centroids=Clustering(get_initial_centroids(train_data),"Median")

      interation is :-  1
      SSE is :- 32.968947001979
      centroid diff:-  1.7243408662900193
      interation is :-  2
      SSE is :- 10.430157173775802
      centroid diff:-  0.5783898305084744
      interation is :-  3
      SSE is :- 7.629981407003094
      centroid diff:-  0.18196798493408645
      interation is :-  4
      SSE is :- 7.557347636286577
      centroid diff:-  0.08639359698681742
      interation is :-  5
      SSE is :- 7.528701664414581
      centroid diff:-  0.0
```

Called the function Clustering with type “Median” to compute K-Medoids.
SSE for each iteration is printed.

```
[40]: final_centroids
```

[40]:

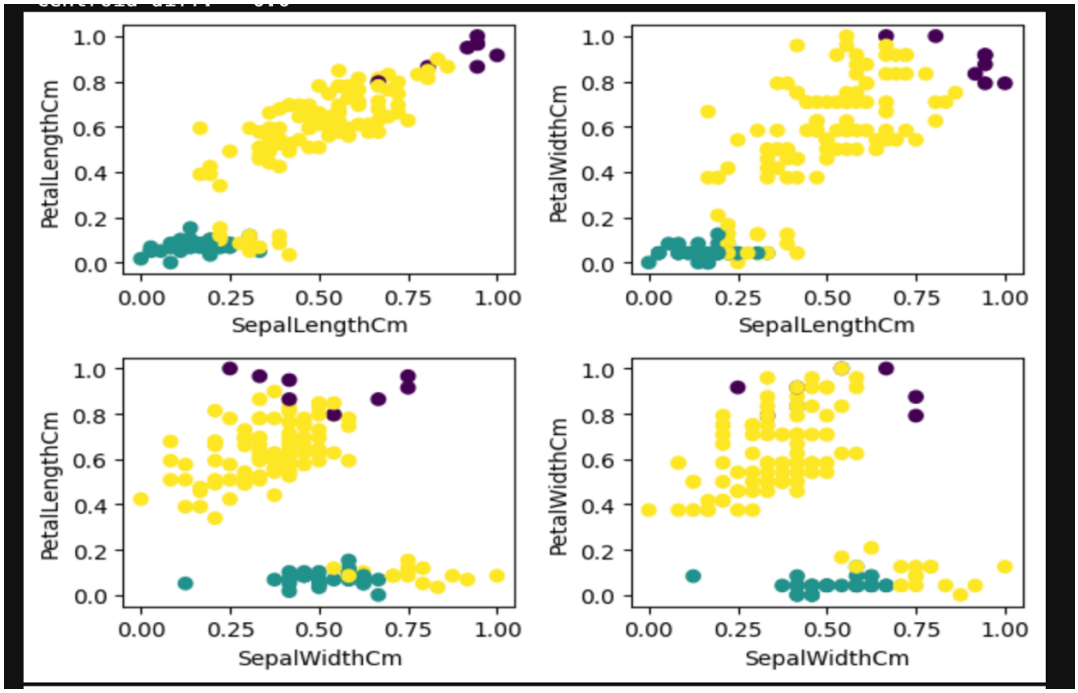
	0	1	2
SepalLengthCm	0.694444	0.194444	0.472222
SepalWidthCm	0.416667	0.583333	0.375000
PetalLengthCm	0.762712	0.084746	0.593220
PetalWidthCm	0.833333	0.041667	0.583333

Final centroid for K-medoids

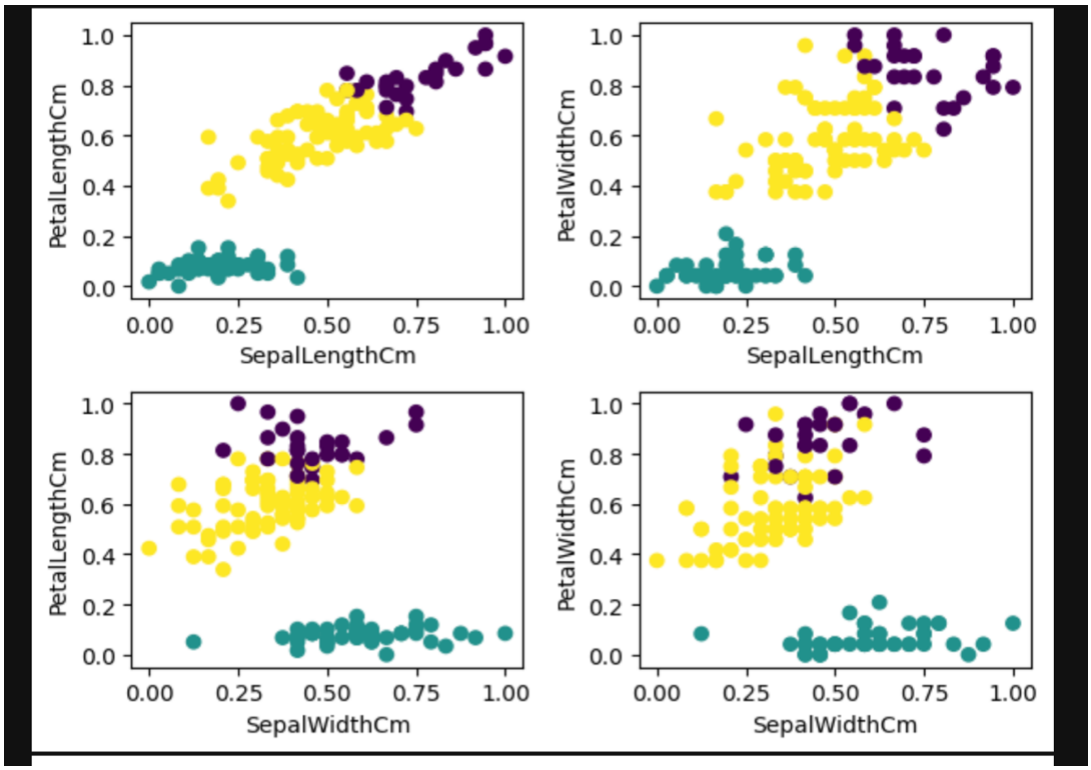
```
[28]: train_lables=get_labels(train_data,final_centroids)

[29]: label_for_each_group(train_data.groupby(train_lables).apply(lambda x:x.index)[0])

      Count of each labels in given cluster:-
      Iris-versicolor:- 1
      Iris-virginica:- 36
      Iris-setosa:- 0

[30]: label_for_each_group(train_data.groupby(train_lables).apply(lambda x:x.index)[1])

      Count of each labels in given cluster:-
      Iris-versicolor:- 0
      Iris-virginica:- 0
      Iris-setosa:- 50

[31]: label_for_each_group(train_data.groupby(train_lables).apply(lambda x:x.index)[2])

      Count of each labels in given cluster:-
      Iris-versicolor:- 49
      Iris-virginica:- 14
      Iris-setosa:- 0
```

This shows how the algorithm grouped the data set.
Graphs for each Iteration: -

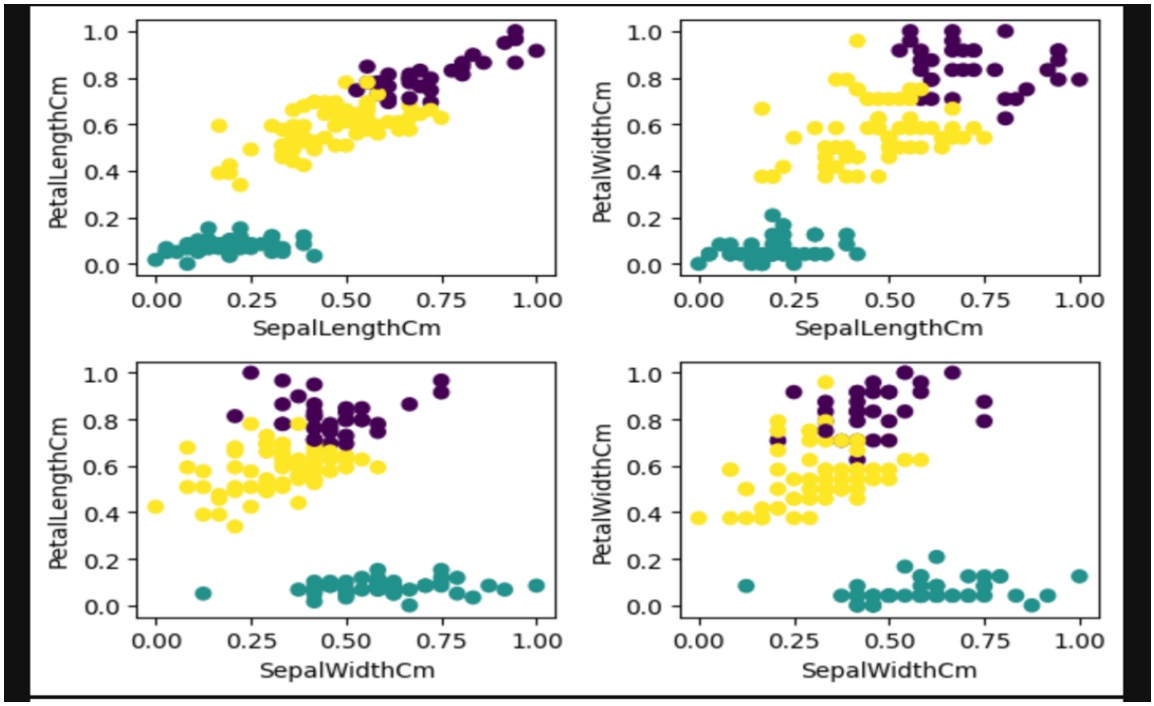
Iteration 1: -



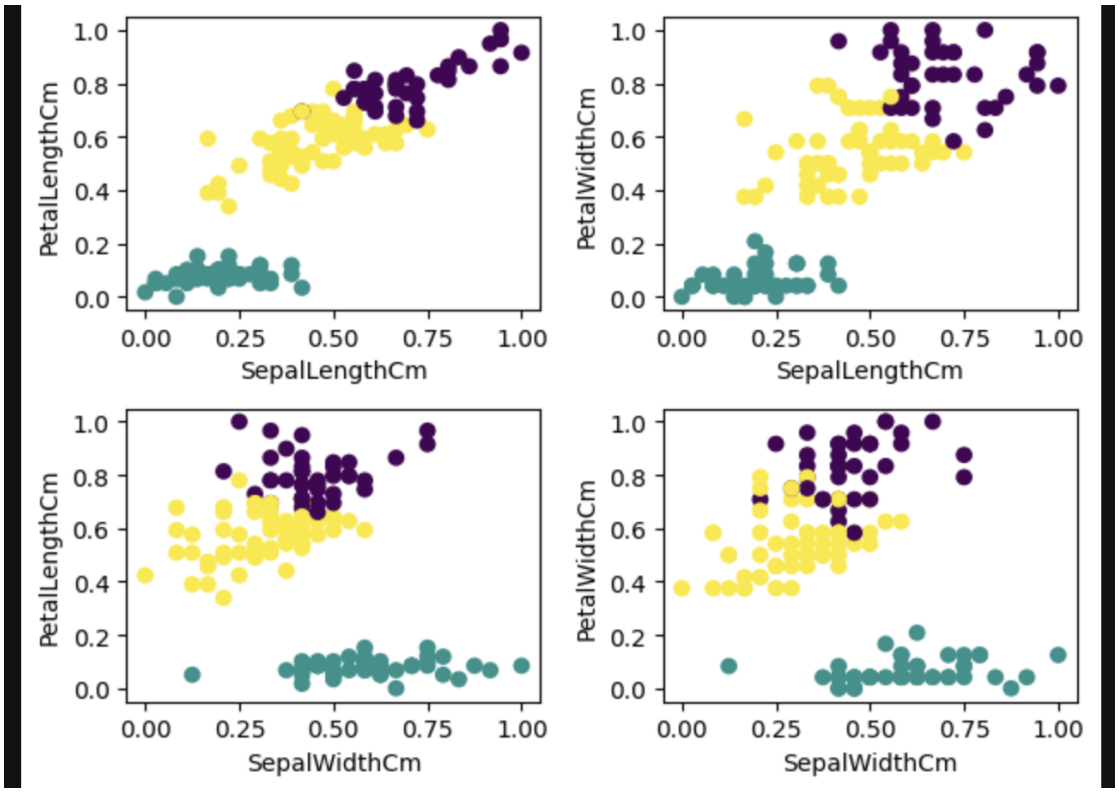
Iteration 2: -



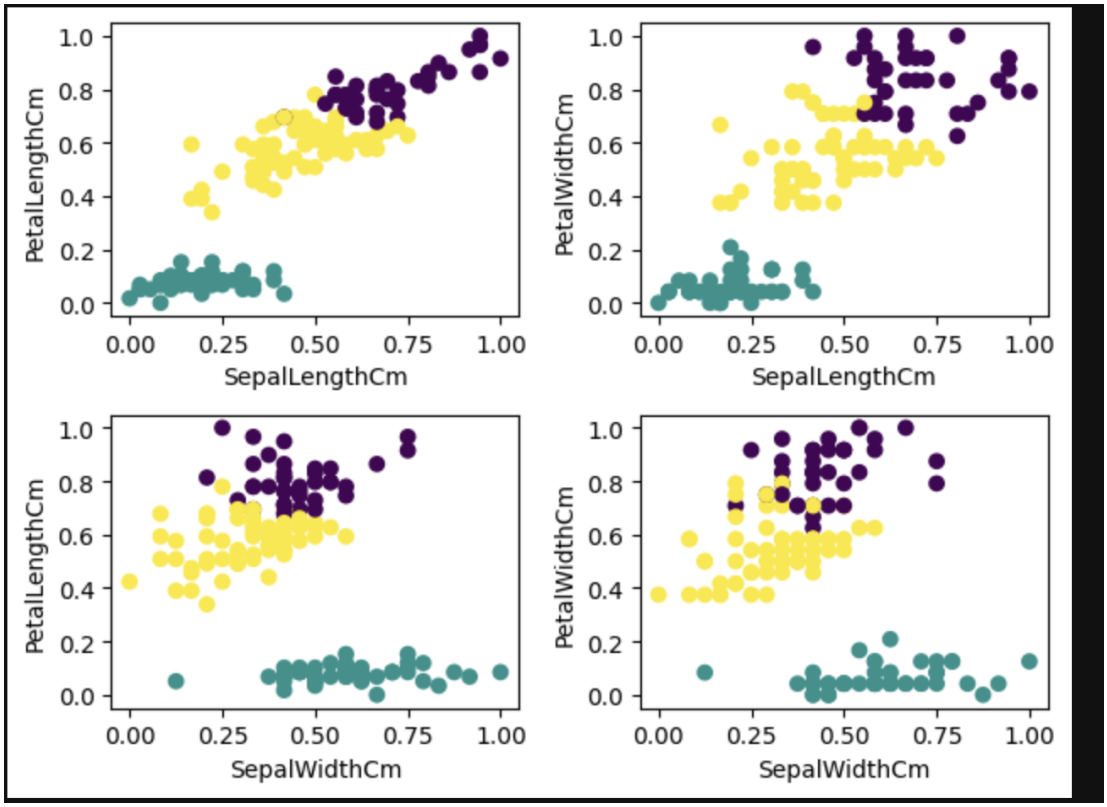
Iteration 3: -



Iteration 4: -



Iteration 5: -



K -Means: -

```
[22]: final_centroids=Clustering(get_initial_centroids(train_data),"Mean")

      interation is :-  1
      SSE is :- 32.96894700197899
      centroid diff:-  1.7022824274091306
      interation is :-  2
      SSE is :- 9.442510388534416
      centroid diff:-  0.6008211311580789
      interation is :-  3
      SSE is :- 7.240162537959597
      centroid diff:-  0.12323638840453027
      interation is :-  4
      SSE is :- 7.073931926882713
      centroid diff:-  0.09720285898508213
      interation is :-  5
      SSE is :- 7.003317600098681
      centroid diff:-  0.023376982785922706
      interation is :-  6
      SSE is :- 6.998114004826761
      centroid diff:-  0.0
```

SSE for each iteration is printed for k-means.

```
[38]: final_centroids

[38]:
```

	0	1	2
SepalLengthCm	0.707265	0.196111	0.441257
SepalWidthCm	0.450855	0.590833	0.307377
PetalLengthCm	0.797045	0.078644	0.575715
PetalWidthCm	0.824786	0.060000	0.549180

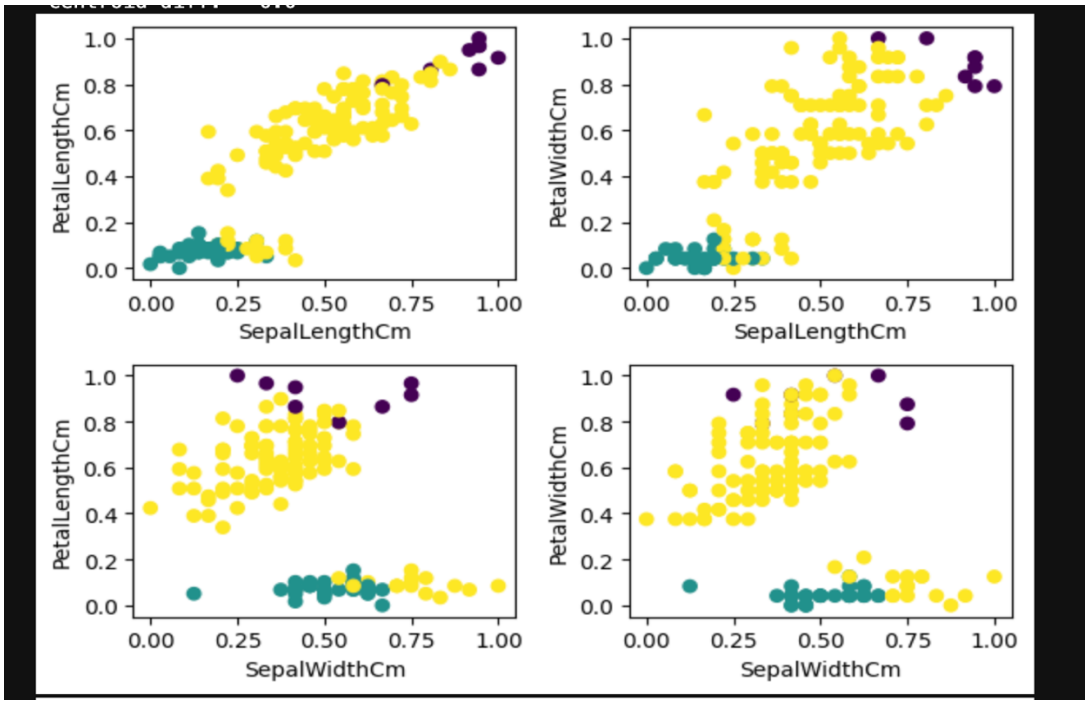
Final centroids for K -means.

```
[27]: train_lables=get_labels(train_data,final_centroids)

[28]: label_for_each_group(train_data.groupby(train_lables).apply(lambda x:x.index)[0])

      Count of each labels in given cluster:-
      Iris-versicolor:- 3
      Iris-virginica:- 36
      Iris-setosa:- 0

[29]: label_for_each_group(train_data.groupby(train_lables).apply(lambda x:x.index)[1])

      Count of each labels in given cluster:-
      Iris-versicolor:- 0
      Iris-virginica:- 0
      Iris-setosa:- 50

[30]: label_for_each_group(train_data.groupby(train_lables).apply(lambda x:x.index)[2])

      Count of each labels in given cluster:-
      Iris-versicolor:- 47
      Iris-virginica:- 14
      Iris-setosa:- 0
```

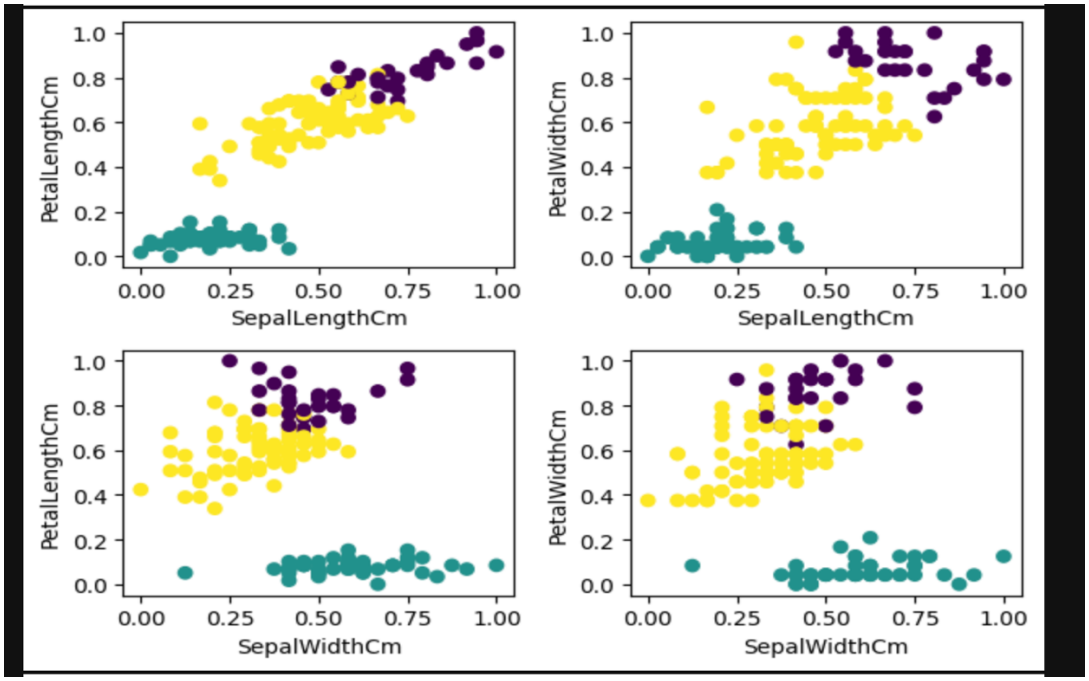
How data is grouped

GRAPHS for each iteration: -

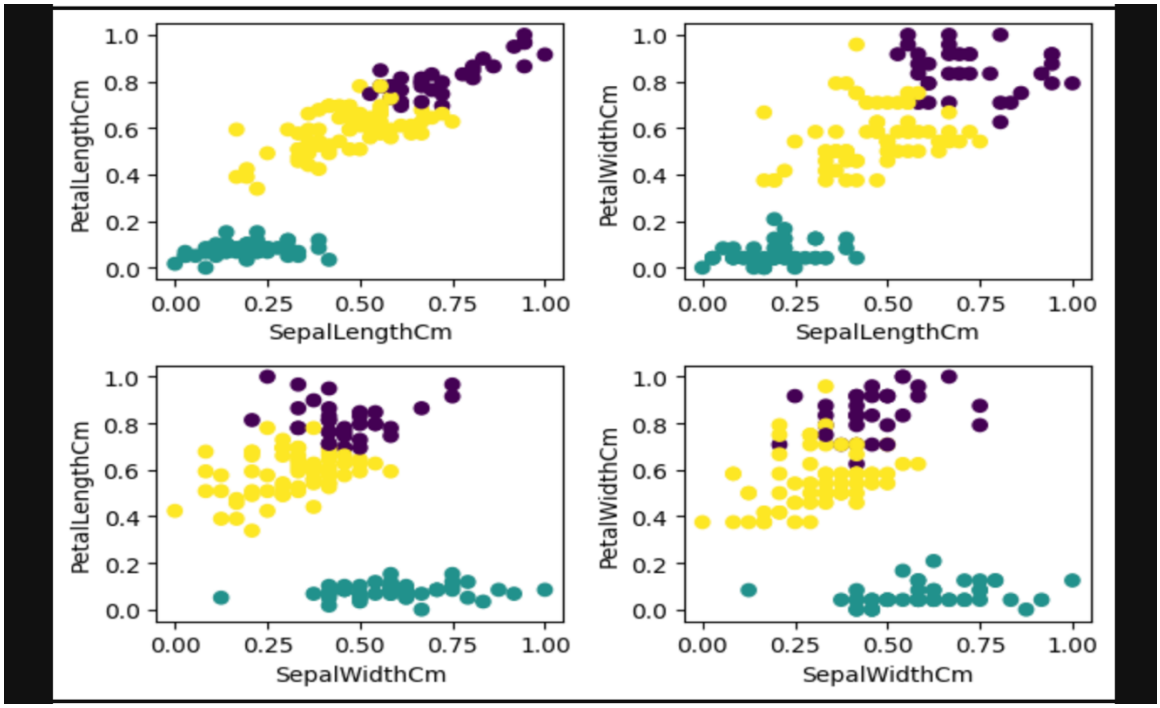
1st Iterations



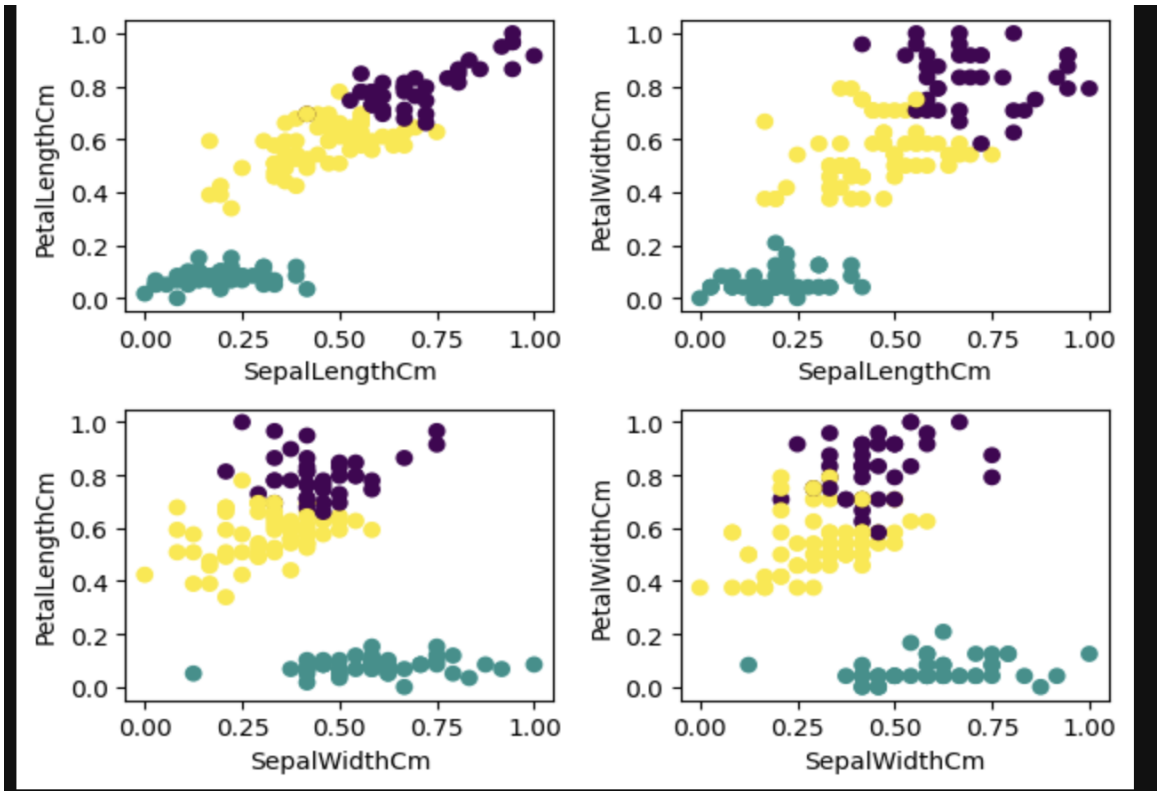
2nd Iteration: -



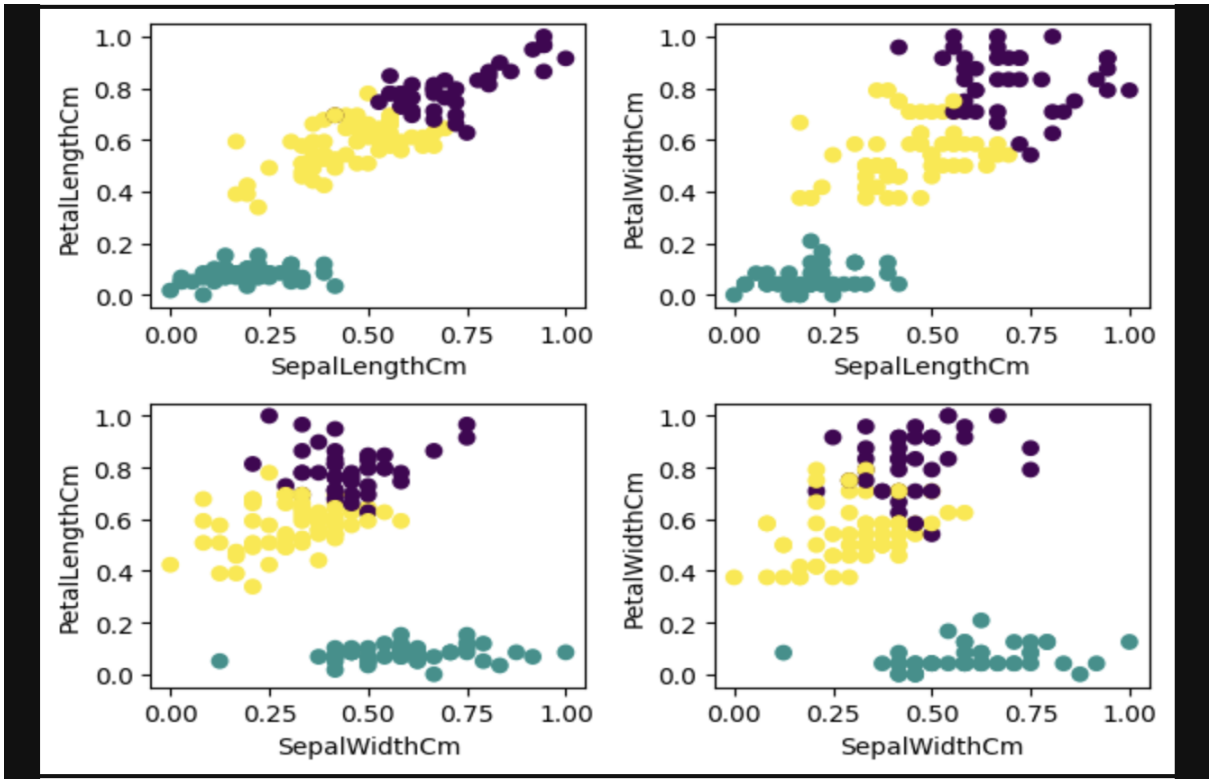
3rd Iteration: -



4th Iteration: -



5th Iteration: -



6th Iterations:-