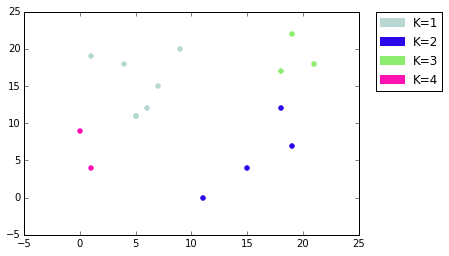
Data Analysis HW4

# Question 1 Answers

 A.) Clustering plot when data points are plotted in order.

s

Cluster Centers:

(Xave, Yave, NumPoints)

{1: (5.2857142857142865, 15.142857142857142, 7),

2: (15.75, 5.75, 4),

3: (19.333333333333332, 19.0, 3),

4: (0.5, 6.5, 2)}

Data point order:

X Y Cluster

0 6 12 1

1 19 7 2

2 15 4 2

3 11 0 3

4 18 12 2

5 9 20 1

6 19 22 2

7 18 17 2

8 5 11 1

9 4 18 1

10 7 15 1

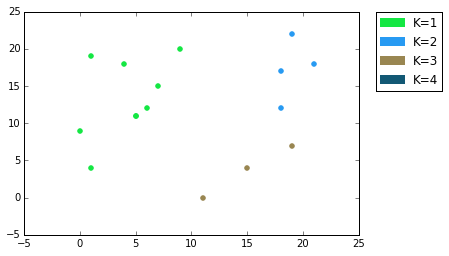
11 21 18 2

12 1 19 4

13 1 4 1

14 0 9 1

15 5 11 1

 B.) Clustering plot when data points are plotted in reverse order.

Cluster Centers:

(Xave, Yave, NumPoints)

{1: (4.2222222222222223, 13.222222222222221, 9),

2: (19.0, 17.25, 4),

3: (15.0, 3.666666666666667, 3)}

Data points order:

X Y Cluster

15 5 11 1

14 0 9 1

13 1 4 1

12 1 19 4

11 21 18 2

10 7 15 1

9 4 18 1

8 5 11 1

7 18 17 2

6 19 22 2

5 9 20 1

4 18 12 2

3 11 0 3

2 15 4 2

1 19 7 2

0 6 12 1

C.) Use Rand index to find the difference between the two clusterings obtained in (a) and (b). Informally, identify the cluster in (a) that has been altered the most in clustering done in (b). Give informal explanation of why this cluster broke apart the most.

**Rand Index**

Rand Index: 0.566666666667

**Cluster identification and explanation:** The cluster from part A that was changed the most was cluster 1 and 2. Clusters 1 and 2 both merged in the second clustering to form one large cluster. This makes sense when analyzing two parts of the algorithm. First is the fact that distance to a cluster is based off of the centroid of the cluster. Second is the order that the points are evaluated. When moving through the original data point list, points (1, 4), (0, 9), and (5, 11) aren’t evaluated until after most of the other points. By this time, the centroid of cluster 1 has moved far enough away that points (1, 4) and (0, 9) are no longer close enough to be considered to be a part of cluster 1. So, they are put in their own cluster. When the algorithm iterates through the list of data points backwards, points (1, 4), (0, 9), and (5, 11) are evaluated first. Thus, the centroid for cluster 1 is close enough to all of these points to include them in the same cluster. Furthermore, because these points are evaluated first, the centroid for cluster 1 is allowed to move to a position close enough to include all of the data points

**Python Code for Question 1**

def prob1():

global prob1Data

global clusterCenters

global clusterCenters2

global prob1Data2

listdat = [(6, 12), (19, 7), (15, 4), (11, 0),

(18, 12), (9, 20), (19, 22), (18, 17),

(5, 11), (4, 18), (7, 15), (21, 18), (1, 19),

(1, 4), (0, 9), (5, 11)]

#Initialize the data

prob1Data = pd.DataFrame(listdat, columns=['X','Y'])

prob1Data['Cluster'] = pd.Series(np.zeros(prob1Data.shape[0]))

#Reverse the data order and create a new data set for it.

listdat.reverse()

prob1Data2 = pd.DataFrame(listdat, columns=['X','Y'])

prob1Data2['Cluster'] = pd.Series(np.zeros(prob1Data.shape[0]))

#RUN PART A

clusterCenters = sequantialClusteringAlgorithm(prob1Data)

#Plot the data on a scatter plot.

colorHandles = []

#Make a list to hold the plotted cluster radi.

clusterCircles = []

#Create each cluster scatter plot.

for cluster in range(1, 5):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob1Data[prob1Data.Cluster == cluster].X, prob1Data[prob1Data.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

clusterCircles.append(plt.Circle((clusterCenters[cluster][0], clusterCenters[cluster][1]), 12, color=randColor, fill=False, clip\_on=False))

#plt.scatter(prob1Data[prob1Data.Cluster == 1].X, prob1Data[prob1Data.Cluster == 1].Y, label='Cluster 1', color=)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#RUN PART B

plt.clf()

clusterCenters2 = sequantialClusteringAlgorithm(prob1Data2)

#Plot the data on a scatter plot.

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 5):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob1Data2[prob1Data2.Cluster == cluster].X, prob1Data2[prob1Data2.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#RUN PART C

randindex = randIndex(prob1Data, prob1Data2)

print("Rand Index: " + str(randindex))

'''

Clusters 1, 2, and 3.

Using incremental average newave = oldave + (an−oldave)/n.

'''

def sequantialClusteringAlgorithm(dataSet):

clusterCenters = {}

#Algorithm Parameters.

theta = 12

maxClusters = 4

numClusters = 1

#Set first point to its own cluster.

#For each data point

#1) Calculate distance to closest cluster center

#2) If dist < alg and numClusters < 4

#Create new cluster with data point.

#3) Else, add data point to cluster.

clusterCenters[numClusters] = (dataSet.ix[0].X, dataSet.ix[0].Y, 1)

dataSet['Cluster'].loc[0] = numClusters

for index in range(1, dataSet.shape[0]):

distance = 999999999

clusterToAssign = 0

for cluster in clusterCenters.keys():

tempDist = dist(dataSet.ix[index], clusterCenters[cluster])

if tempDist < distance:

distance = tempDist

clusterToAssign = cluster

if distance <= theta and numClusters <= maxClusters:

dataSet['Cluster'].loc[index] = clusterToAssign

newX = clusterCenters[clusterToAssign][0] + (dataSet.ix[index].X - clusterCenters[clusterToAssign][0])/(clusterCenters[clusterToAssign][2]+1.0)

newY = clusterCenters[clusterToAssign][1] + (dataSet.ix[index].Y - clusterCenters[clusterToAssign][1])/(clusterCenters[clusterToAssign][2]+1.0)

newSize = clusterCenters[clusterToAssign][2] + 1

clusterCenters[clusterToAssign] = (newX, newY, newSize)

dataSet['Cluster'].loc[index] = clusterToAssign

elif distance > theta and numClusters < maxClusters:

numClusters += 1

dataSet['Cluster'].loc[index] = numClusters

clusterCenters[numClusters] = (dataSet.ix[index].X, dataSet.ix[index].Y, 1)

dataSet['Cluster'].loc[index] = numClusters

elif numClusters >= maxClusters:

#print("At Max")

dataSet.ix[index].Cluster = clusterToAssign

newX = clusterCenters[clusterToAssign][0] + (dataSet.ix[index].X - clusterCenters[clusterToAssign][0])/(clusterCenters[clusterToAssign][2]+1.0)

newY = clusterCenters[clusterToAssign][1] + (dataSet.ix[index].Y - clusterCenters[clusterToAssign][1])/(clusterCenters[clusterToAssign][2]+1.0)

newSize = clusterCenters[clusterToAssign][2] + 1

clusterCenters[clusterToAssign] = (newX, newY, newSize)

dataSet['Cluster'].loc[index] = clusterToAssign

return clusterCenters

def dist(point1, point2):

sumsq = 0

for index in range(0, len(point1) - 1):

sumsq += math.pow(point1[index] - point2[index], 2)

return math.pow(sumsq,.5)

def randIndex(clustering1, clustering2):

f00 = 0

f01 = 0

f10 = 0

f11 = 0

for firstIndex in range(0, clustering1.shape[0] - 1):

for secondIndex in range(firstIndex + 1, clustering1.shape[0]):

if clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f00 += 1

elif clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f01 += 1

elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f10 += 1

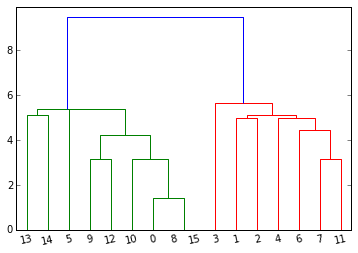
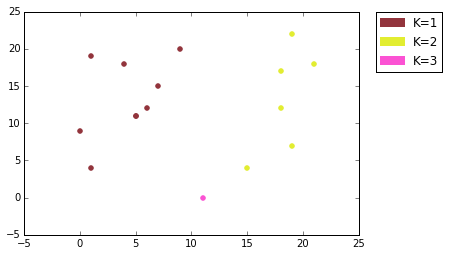
elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f11 += 1

return (f00 + f11) / float(f00 + f01 + f10 + f11)

# Question 2 Answers

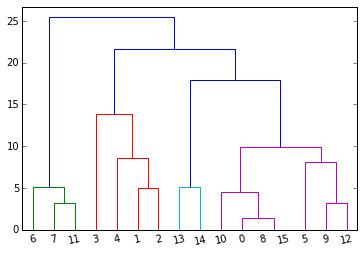
A.) Single Link Hierarchical Clustering

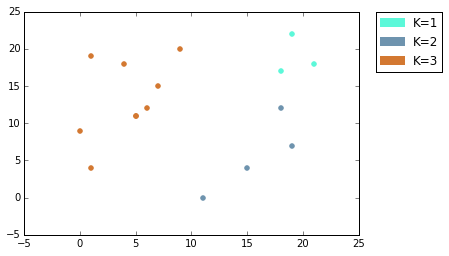


Distance at 5.4 to split data into 3 clusters.

B.) Complete Link Hierarchical Clustering

Distance at 20 to split data into 3 clusters.





C.) Sum Squared Errors

**Comparison and Comment on Clustering SSEs:** The first clustering has a much higher SSE than the second clustering. Looking at the graphs of points, this makes sense because the first clustering has more spread out points in cluster 2 than the second clustering. Since cluster 2 in the first clustering incorporates more points further away from its mean, and since the measurement is a squared value, the first clustering has a higher sum squared error. This also makes sense from the conceptual standpoint. Reduced SSE means that the clusters are more globular. As we can see in the second clustering graph, its clusters are much more globular than the first clustering’s. Thus, we would expect to see a smaller SSE for the second clustering.

Sum squared error for single link: 551.777777778

Cluster **contributing most** to SSE: 1

Cluster SSE: 293.111111111

Sum squared error for complete link: 427.277777778

Cluster **contributing most** to SSE: 3

Cluster SSE: 293.111111111

**Note:** Look at each graph individually above to identify which cluster is labeled 1, and which is labeled 3. Each clustering has labeled the clusters with differing numeric labels.

D.) Find the correlation values between the proximity and incidence matrices for both clusterings.

**Correlation:**

Correlation coeff of single linkage clustering: -0.723477128698

Correlation coeff of complete linkage clustering: -0.741615297576

**Correlation Comments:**

The correlation of complete linkage is -0.7416, and its magnitude is larger than the single link clustering. That means that the clusters in complete linkage are more compact. This makes sense when looking at the graphing of both clustering methods since the clustering of the second method is more well defined.

**Python Code for Question 2**

def prob2():

global prob2Data

global linkageMatrix

global dend

listdat = [(6, 12), (19, 7), (15, 4), (11, 0),

(18, 12), (9, 20), (19, 22), (18, 17),

(5, 11), (4, 18), (7, 15), (21, 18), (1, 19),

(1, 4), (0, 9), (5, 11)]

#Initialize the data

prob2Data = pd.DataFrame(listdat, columns=['X','Y'])

#Perform Clustering.

linkageMatrix = heirarchical.linkage(prob2Data.values, method='single', metric='euclidean')

#Draw Dendrogram.

dend = heirarchical.dendrogram(linkageMatrix)

plt.show()

#Clustering with the distance set to 5.4 so that there are 3 clusters.

prob2Data['Cluster'] = fcluster(linkageMatrix, 5.4, criterion='distance')

#Plot the data on a scatter plot.

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 4):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob2Data[prob2Data.Cluster == cluster].X, prob2Data[prob2Data.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#PART B

#Initialize the data

prob2Data2 = pd.DataFrame(listdat, columns=['X','Y'])

#Perform Clustering.

linkageMatrix = heirarchical.linkage(prob2Data2.values, method='complete', metric='euclidean')

#Draw Dendrogram.

dend = heirarchical.dendrogram(linkageMatrix)

plt.show()

#Clustering with the distance set to 20 so that there are 3 clusters.

prob2Data2['Cluster'] = fcluster(linkageMatrix, 20, criterion='distance')

#Plot the data on a scatter plot.

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 4):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob2Data2[prob2Data2.Cluster == cluster].X, prob2Data2[prob2Data2.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#PART C. Calculate the SSE or both clusterings.

singleLinkSSE, maxClusterContribSingle = sumSquaredError(prob2Data)

completeLinkSSE, maxClusterContribComplete = sumSquaredError(prob2Data2)

print "Sum squared error for single link: " + str(singleLinkSSE)

print "Cluster contributing most to SSE: " + str(maxClusterContribSingle[0])

print "Cluster SSE: " + str(maxClusterContribSingle[1])

print

print "Sum squared error for complete link: " + str(completeLinkSSE)

print "Cluster contributing most to SSE: " + str(maxClusterContribComplete[0])

print "Cluster SSE: " + str(maxClusterContribComplete[1])

#PART D. Build proximity and incidence matricies, and calculate the correlation for each clustering.

print

corr, pm, im = correlationClusterAnalysis(prob2Data)

print("Correlation coeff of single linkage clustering: " + str(corr[0][1]))

corr, pm, im = correlationClusterAnalysis(prob2Data2)

print("Correlation coeff of complete linkage clustering: " + str(corr[0][1]))

def sumSquaredError(dataSet):

totalSum = 0

#Store cluster with maximum contribution to sse as (cluster, SSE contrib)

maxClusterContribution = (0, 0)

#For each cluster

# For each point in each cluster

# Find squared distance between mean and point, and add to cluster sum.

#Sum all cluster values.

for cluster in range(1, 4):

#Get view of all data in the same cluster.

currentClusterData = dataSet[dataSet.Cluster == cluster]

meanX = currentClusterData.X.values.mean()

meanY = currentClusterData.Y.values.mean()

clusterSum = 0

for index, row in currentClusterData.iterrows():

clusterSum += math.pow(meanX-row.X, 2) + math.pow(meanY-row.Y, 2)

if clusterSum > maxClusterContribution[1]:

maxClusterContribution = (cluster, clusterSum)

totalSum += clusterSum

return totalSum, maxClusterContribution

def correlationClusterAnalysis(dataSet):

#Make an mxm matrix where m is the number of data points.

proximityMatrix = np.zeros((dataSet.shape[0], dataSet.shape[0]))

incidenceMatrix = np.zeros((dataSet.shape[0], dataSet.shape[0]))

for i in range(0, dataSet.shape[0]):

for j in range(i, dataSet.shape[0]):

distance = math.pow(math.pow(dataSet.iloc[i].X - dataSet.iloc[j].X, 2) + math.pow(dataSet.iloc[i].Y - dataSet.iloc[j].Y, 2), .5)

proximityMatrix[i,j] = distance

proximityMatrix[j,i] = distance

if dataSet['Cluster'].iloc[i] == dataSet['Cluster'].iloc[j]:

incidenceMatrix[i,j] = 1

incidenceMatrix[j,i] = 1

correlation = np.corrcoef(proximityMatrix.flatten(), incidenceMatrix.flatten())

return correlation, proximityMatrix, incidenceMatrix

# Question 3 Answers

A.) Mark all points as Core, Border, or Noise, and plot the clusters on the number line using Epsilon = 4, and minpts = 3.

**Point Marking.**

Noise points: Pkind = 0

Core points: Pkind = 1

Fringe points: Pkind = 2

X Pkind Cluster

0 1 1 0

1 3 1 0

2 5 1 0

3 6 1 0

4 8 1 0

5 11 1 0

6 12 1 0

7 13 1 0

8 14 1 0

9 15 1 0

10 16 1 0

11 22 0 0

12 28 2 0

13 32 1 0

14 33 1 0

15 34 1 0

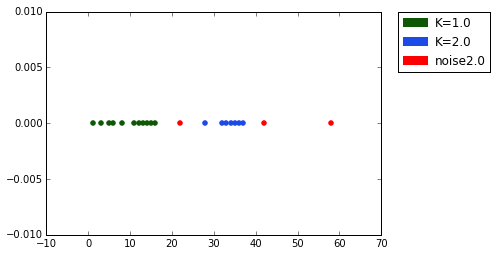
16 35 1 0

17 36 1 0

18 37 1 0

19 42 0 0

20 58 0 0



Clusters

Noise

B.) Mark all points as Core, Border, or Noise, and plot the clusters on the number line using Epsilon = 6, and minpts = 3.

**Point Marking.**

Noise points: Pkind = 0

Core points: Pkind = 1

Fringe points: Pkind = 2

X Pkind Cluster

0 1 1 0

1 3 1 0

2 5 1 0

3 6 1 0

4 8 1 0

5 11 1 0

6 12 1 0

7 13 1 0

8 14 1 0

9 15 1 0

10 16 1 0

11 22 1 0

12 28 1 0

13 32 1 0

14 33 1 0

15 34 1 0

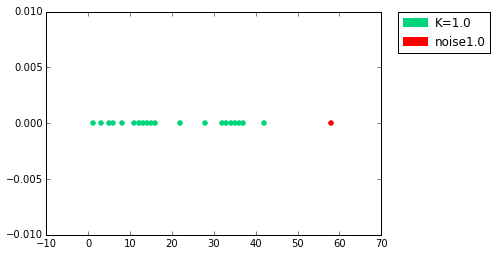
16 35 1 0

17 36 1 0

18 37 1 0

19 42 1 0

20 58 0 0



Cluster

Noise

C.) Compare the two clusterings using Rand Index and show all your work.

**Comparison based on rand index:**

f00 = 0

f01 = 77

f10 = 0

f11 = 76

Rand index = 0.496732026144

**Work shown as list of each point comparison when calculating the rand index:**

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 3, Cluster 1, Name: 1, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 5, Cluster 1, Name: 2, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 6, Cluster 1, Name: 3, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 8, Cluster 1, Name: 4, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 11, Cluster 1, Name: 5, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 12, Cluster 1, Name: 6, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 1, Cluster 1, Name: 0, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 5, Cluster 1, Name: 2, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 6, Cluster 1, Name: 3, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 8, Cluster 1, Name: 4, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 11, Cluster 1, Name: 5, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 12, Cluster 1, Name: 6, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 3, Cluster 1, Name: 1, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 6, Cluster 1, Name: 3, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 8, Cluster 1, Name: 4, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 11, Cluster 1, Name: 5, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 12, Cluster 1, Name: 6, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 5, Cluster 1, Name: 2, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 8, Cluster 1, Name: 4, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 11, Cluster 1, Name: 5, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 12, Cluster 1, Name: 6, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 6, Cluster 1, Name: 3, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 11, Cluster 1, Name: 5, dtype: float64

Assigned to f11

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 12, Cluster 1, Name: 6, dtype: float64

Assigned to f11

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 8, Cluster 1, Name: 4, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 12, Cluster 1, Name: 6, dtype: float64

Assigned to f11

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 11, Cluster 1, Name: 5, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 13, Cluster 1, Name: 7, dtype: float64

Assigned to f11

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 12, Cluster 1, Name: 6, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 14, Cluster 1, Name: 8, dtype: float64

Assigned to f11

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 13, Cluster 1, Name: 7, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 15, Cluster 1, Name: 9, dtype: float64

Assigned to f11

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 14, Cluster 1, Name: 8, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 16, Cluster 1, Name: 10, dtype: float64

Assigned to f11

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 15, Cluster 1, Name: 9, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 28, Cluster 2, Name: 12, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f01

Point A: X 16, Cluster 1, Name: 10, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f01

Point A: X 28, Cluster 2, Name: 12, dtype: float64

Point B: X 32, Cluster 2, Name: 13, dtype: float64

Assigned to f11

Point A: X 28, Cluster 2, Name: 12, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f11

Point A: X 28, Cluster 2, Name: 12, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f11

Point A: X 28, Cluster 2, Name: 12, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f11

Point A: X 28, Cluster 2, Name: 12, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f11

Point A: X 28, Cluster 2, Name: 12, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f11

Point A: X 32, Cluster 2, Name: 13, dtype: float64

Point B: X 33, Cluster 2, Name: 14, dtype: float64

Assigned to f11

Point A: X 32, Cluster 2, Name: 13, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f11

Point A: X 32, Cluster 2, Name: 13, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f11

Point A: X 32, Cluster 2, Name: 13, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f11

Point A: X 32, Cluster 2, Name: 13, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f11

Point A: X 33, Cluster 2, Name: 14, dtype: float64

Point B: X 34, Cluster 2, Name: 15, dtype: float64

Assigned to f11

Point A: X 33, Cluster 2, Name: 14, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f11

Point A: X 33, Cluster 2, Name: 14, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f11

Point A: X 33, Cluster 2, Name: 14, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f11

Point A: X 34, Cluster 2, Name: 15, dtype: float64

Point B: X 35, Cluster 2, Name: 16, dtype: float64

Assigned to f11

Point A: X 34, Cluster 2, Name: 15, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f11

Point A: X 34, Cluster 2, Name: 15, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f11

Point A: X 35, Cluster 2, Name: 16, dtype: float64

Point B: X 36, Cluster 2, Name: 17, dtype: float64

Assigned to f11

Point A: X 35, Cluster 2, Name: 16, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f11

Point A: X 36, Cluster 2, Name: 17, dtype: float64

Point B: X 37, Cluster 2, Name: 18, dtype: float64

Assigned to f11

**Python Code for this section:**

def prob3():

global dataSet

global dataSet2

radius = 4

minpts = 3

pointsDict = {}

numClusters = 0

datalist = [1, 3, 5, 6, 8, 11, 12, 13, 14, 15, 16, 22, 28, 32, 33, 34, 35, 36, 37, 42, 58]

dataSet = pd.DataFrame(datalist, columns=['X'])

#Core points are 1, fringe points are 2, noise points are 0

dataSet['Pkind'] = np.zeros(len(datalist))

dataSet['Cluster'] = np.zeros(len(datalist))

#Find all core points.

for pointIndex in range(0, dataSet.shape[0]):

#Initialize num points to 0.

pointsDict[pointIndex] = 0

dist = 0

for comparisonIndex in range(0, dataSet.shape[0]):

dist = math.sqrt(math.pow(dataSet.iloc[pointIndex].X - dataSet.iloc[comparisonIndex].X, 2))

if dist <= radius:

pointsDict[pointIndex] = pointsDict[pointIndex] + 1

if pointsDict[pointIndex] >= minpts:

#Core point identified. Mark it and move to next point.

dataSet['Pkind'].iloc[pointIndex] = 1

continue

#Find all fringe points.

for fringePointIndex in dataSet[dataSet.Pkind == 0].index.tolist():

for comparisonIndex in dataSet[dataSet.Pkind == 1].index.tolist():

dist = math.sqrt(math.pow(dataSet.iloc[fringePointIndex].X - dataSet.iloc[comparisonIndex].X, 2))

if dist <= radius:

dataSet['Pkind'].iloc[fringePointIndex] = 2

continue

print("Noise points: Pkind = 0")

print("Core points: Pkind = 1")

print("Fringe points: Pkind = 2")

print dataSet

print

#Eliminate all points not core or fringe.

noise = dataSet.copy()[dataSet.Pkind == 0]

dataSet = dataSet[dataSet.Pkind != 0]

#For each core point

for corePointIndex in range(0, dataSet.shape[0]):

#If non-core point, continue.

if dataSet['Pkind'].iloc[corePointIndex] != 1:

continue

#If core point and no cluster, assign new cluster.

if dataSet['Cluster'].iloc[corePointIndex] == 0:

numClusters += 1

dataSet['Cluster'].iloc[corePointIndex] = numClusters

#Assign all points in vicinity of core to same cluster as core.

for comparisonIndex in range(0, dataSet.shape[0]):

dist = math.sqrt(math.pow(dataSet.iloc[corePointIndex].X - dataSet.iloc[comparisonIndex].X, 2))

if dist <= radius and dataSet['Cluster'].iloc[comparisonIndex] == 0:

dataSet['Cluster'].iloc[comparisonIndex] = dataSet['Cluster'].iloc[corePointIndex]

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in set(dataSet.Cluster.tolist()):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(dataSet[dataSet.Cluster == cluster].X, np.zeros(dataSet[dataSet.Cluster == cluster].shape[0]), label="Cluster " + str(cluster), color=randColor)

#Plot noise

colorHandles.append(matplotlib.patches.Patch(color='r', label='noise' + str(cluster)))

plt.scatter(noise.X, np.zeros(noise.shape[0]), color='r')

#Plot final graph of data.

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#PART B

radius = 6

minpts = 3

pointsDict = {}

numClusters = 0

dataSet2 = pd.DataFrame(datalist, columns=['X'])

#Core points are 1, fringe points are 2, noise points are 0

dataSet2['Pkind'] = np.zeros(len(datalist))

dataSet2['Cluster'] = np.zeros(len(datalist))

#Find all core points.

for pointIndex in range(0, dataSet2.shape[0]):

#Initialize num points to 0.

pointsDict[pointIndex] = 0

dist = 0

for comparisonIndex in range(0, dataSet2.shape[0]):

dist = math.sqrt(math.pow(dataSet2.iloc[pointIndex].X - dataSet2.iloc[comparisonIndex].X, 2))

if dist <= radius:

pointsDict[pointIndex] = pointsDict[pointIndex] + 1

if pointsDict[pointIndex] >= minpts:

#Core point identified. Mark it and move to next point.

dataSet2['Pkind'].iloc[pointIndex] = 1

continue

#Find all fringe points.

for fringePointIndex in dataSet2[dataSet2.Pkind == 0].index.tolist():

for comparisonIndex in dataSet2[dataSet2.Pkind == 1].index.tolist():

dist = math.sqrt(math.pow(dataSet2.iloc[fringePointIndex].X - dataSet2.iloc[comparisonIndex].X, 2))

if dist <= radius:

dataSet2['Pkind'].iloc[fringePointIndex] = 2

continue

print("Noise points: Pkind = 0")

print("Core points: Pkind = 1")

print("Fringe points: Pkind = 2")

print dataSet2

print

#Eliminate all points not core or fringe.

noise = dataSet2.copy()[dataSet2.Pkind == 0]

dataSet2 = dataSet2[dataSet2.Pkind != 0]

#For each core point

for corePointIndex in range(0, dataSet2.shape[0]):

#If non-core point, continue.

if dataSet2['Pkind'].iloc[corePointIndex] != 1:

continue

#If core point and no cluster, assign new cluster.

if dataSet2['Cluster'].iloc[corePointIndex] == 0:

numClusters += 1

dataSet2['Cluster'].iloc[corePointIndex] = numClusters

#Assign all points in vicinity of core to same cluster as core.

for comparisonIndex in range(0, dataSet2.shape[0]):

dist = math.sqrt(math.pow(dataSet2.iloc[corePointIndex].X - dataSet2.iloc[comparisonIndex].X, 2))

if dist <= radius and dataSet2['Cluster'].iloc[comparisonIndex] == 0:

dataSet2['Cluster'].iloc[comparisonIndex] = dataSet2['Cluster'].iloc[corePointIndex]

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in set(dataSet2.Cluster.tolist()):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(dataSet2[dataSet2.Cluster == cluster].X, np.zeros(dataSet2[dataSet2.Cluster == cluster].shape[0]), label="Cluster " + str(cluster), color=randColor)

#Plot noise

colorHandles.append(matplotlib.patches.Patch(color='r', label='noise' + str(cluster)))

plt.scatter(noise.X, np.zeros(noise.shape[0]), color='r')

#Plot final graph of data.

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#Part C

columnList = dataSet.columns.tolist()

columnList.remove('Pkind')

rindex = randIndex(dataSet[columnList], dataSet2[columnList])

print "Rand index = " + str(rindex)

def randIndex(clustering1, clustering2):

f00 = 0

f01 = 0

f10 = 0

f11 = 0

f = open(os.getcwd() + "\\randIndexFile.txt", 'w+')

for firstIndex in range(0, clustering1.shape[0] - 1):

for secondIndex in range(firstIndex + 1, clustering1.shape[0]):

if clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f00 += 1

f.write("\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f00\n")

elif clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f01 += 1

f.write( "\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f01\n")

elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f10 += 1

f.write( "\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f10\n")

elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f11 += 1

f.write( "\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f11\n")

f.close()

print "f00 = " + str(f00)

print "f01 = " + str(f01)

print "f10 = " + str(f10)

print "f11 = " + str(f11)

return (f00 + f11) / float(f00 + f01 + f10 + f11)

Full Python Code

# -\*- coding: utf-8 -\*-

"""

Created on Sun Nov 15 11:52:02 2015

@author: DAN

"""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib

import math

import random

import scipy.cluster.hierarchy as heirarchical

from scipy.cluster.hierarchy import fcluster

import os

prob1Data = None

prob1Data2 = None

clusterCenters = None

clusterCenters2 = None

linkageMatrix = None

dend = None

dataSet = None

dataSet2 = None

def prob1():

global prob1Data

global clusterCenters

global clusterCenters2

global prob1Data2

listdat = [(6, 12), (19, 7), (15, 4), (11, 0),

(18, 12), (9, 20), (19, 22), (18, 17),

(5, 11), (4, 18), (7, 15), (21, 18), (1, 19),

(1, 4), (0, 9), (5, 11)]

#Initialize the data

prob1Data = pd.DataFrame(listdat, columns=['X','Y'])

prob1Data['Cluster'] = pd.Series(np.zeros(prob1Data.shape[0]))

#Reverse the data order and create a new data set for it.

listdat.reverse()

prob1Data2 = pd.DataFrame(listdat, columns=['X','Y'])

prob1Data2['Cluster'] = pd.Series(np.zeros(prob1Data.shape[0]))

#RUN PART A

clusterCenters = sequantialClusteringAlgorithm(prob1Data)

#Plot the data on a scatter plot.

colorHandles = []

#Make a list to hold the plotted cluster radi.

clusterCircles = []

#Create each cluster scatter plot.

for cluster in range(1, 5):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob1Data[prob1Data.Cluster == cluster].X, prob1Data[prob1Data.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

clusterCircles.append(plt.Circle((clusterCenters[cluster][0], clusterCenters[cluster][1]), 12, color=randColor, fill=False, clip\_on=False))

#plt.scatter(prob1Data[prob1Data.Cluster == 1].X, prob1Data[prob1Data.Cluster == 1].Y, label='Cluster 1', color=)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#RUN PART B

plt.clf()

clusterCenters2 = sequantialClusteringAlgorithm(prob1Data2)

#Plot the data on a scatter plot.

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 5):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob1Data2[prob1Data2.Cluster == cluster].X, prob1Data2[prob1Data2.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#RUN PART C

randindex = randIndex(prob1Data, prob1Data2)

print("Rand Index: " + str(randindex))

'''

Clusters 1, 2, and 3.

Using incremental average newave = oldave + (an−oldave)/n.

'''

def sequantialClusteringAlgorithm(dataSet):

clusterCenters = {}

#Algorithm Parameters.

theta = 12

maxClusters = 4

numClusters = 1

#Set first point to its own cluster.

#For each data point

#1) Calculate distance to closest cluster center

#2) If dist < alg and numClusters < 4

#Create new cluster with data point.

#3) Else, add data point to cluster.

clusterCenters[numClusters] = (dataSet.ix[0].X, dataSet.ix[0].Y, 1)

dataSet['Cluster'].loc[0] = numClusters

for index in range(1, dataSet.shape[0]):

distance = 999999999

clusterToAssign = 0

for cluster in clusterCenters.keys():

tempDist = dist(dataSet.ix[index], clusterCenters[cluster])

if tempDist < distance:

distance = tempDist

clusterToAssign = cluster

if distance <= theta and numClusters <= maxClusters:

dataSet['Cluster'].loc[index] = clusterToAssign

newX = clusterCenters[clusterToAssign][0] + (dataSet.ix[index].X - clusterCenters[clusterToAssign][0])/(clusterCenters[clusterToAssign][2]+1.0)

newY = clusterCenters[clusterToAssign][1] + (dataSet.ix[index].Y - clusterCenters[clusterToAssign][1])/(clusterCenters[clusterToAssign][2]+1.0)

newSize = clusterCenters[clusterToAssign][2] + 1

clusterCenters[clusterToAssign] = (newX, newY, newSize)

dataSet['Cluster'].loc[index] = clusterToAssign

elif distance > theta and numClusters < maxClusters:

numClusters += 1

dataSet['Cluster'].loc[index] = numClusters

clusterCenters[numClusters] = (dataSet.ix[index].X, dataSet.ix[index].Y, 1)

dataSet['Cluster'].loc[index] = numClusters

elif numClusters >= maxClusters:

#print("At Max")

dataSet.ix[index].Cluster = clusterToAssign

newX = clusterCenters[clusterToAssign][0] + (dataSet.ix[index].X - clusterCenters[clusterToAssign][0])/(clusterCenters[clusterToAssign][2]+1.0)

newY = clusterCenters[clusterToAssign][1] + (dataSet.ix[index].Y - clusterCenters[clusterToAssign][1])/(clusterCenters[clusterToAssign][2]+1.0)

newSize = clusterCenters[clusterToAssign][2] + 1

clusterCenters[clusterToAssign] = (newX, newY, newSize)

dataSet['Cluster'].loc[index] = clusterToAssign

return clusterCenters

def dist(point1, point2):

sumsq = 0

for index in range(0, len(point1) - 1):

sumsq += math.pow(point1[index] - point2[index], 2)

return math.pow(sumsq,.5)

def randIndex(clustering1, clustering2):

f00 = 0

f01 = 0

f10 = 0

f11 = 0

f = open(os.getcwd() + "\\randIndexFile.txt", 'w+')

for firstIndex in range(0, clustering1.shape[0] - 1):

for secondIndex in range(firstIndex + 1, clustering1.shape[0]):

if clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f00 += 1

f.write("\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f00\n")

elif clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f01 += 1

f.write( "\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f01\n")

elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f10 += 1

f.write( "\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f10\n")

elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f11 += 1

f.write( "\nPoint A: " + (str(clustering1.iloc[firstIndex]).replace('\n', ', ')))

f.write( "\nPoint B: " + (str(clustering1.iloc[secondIndex]).replace('\n', ', ')))

f.write( "\nAssigned to f11\n")

f.close()

print "f00 = " + str(f00)

print "f01 = " + str(f01)

print "f10 = " + str(f10)

print "f11 = " + str(f11)

return (f00 + f11) / float(f00 + f01 + f10 + f11)

def prob2():

global prob2Data

global linkageMatrix

global dend

listdat = [(6, 12), (19, 7), (15, 4), (11, 0),

(18, 12), (9, 20), (19, 22), (18, 17),

(5, 11), (4, 18), (7, 15), (21, 18), (1, 19),

(1, 4), (0, 9), (5, 11)]

#Initialize the data

prob2Data = pd.DataFrame(listdat, columns=['X','Y'])

#Perform Clustering.

linkageMatrix = heirarchical.linkage(prob2Data.values, method='single', metric='euclidean')

#Draw Dendrogram.

dend = heirarchical.dendrogram(linkageMatrix)

plt.show()

#Clustering with the distance set to 5.4 so that there are 3 clusters.

prob2Data['Cluster'] = fcluster(linkageMatrix, 5.4, criterion='distance')

#Plot the data on a scatter plot.

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 4):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob2Data[prob2Data.Cluster == cluster].X, prob2Data[prob2Data.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#PART B

#Initialize the data

prob2Data2 = pd.DataFrame(listdat, columns=['X','Y'])

#Perform Clustering.

linkageMatrix = heirarchical.linkage(prob2Data2.values, method='complete', metric='euclidean')

#Draw Dendrogram.

dend = heirarchical.dendrogram(linkageMatrix)

plt.show()

#Clustering with the distance set to 20 so that there are 3 clusters.

prob2Data2['Cluster'] = fcluster(linkageMatrix, 20, criterion='distance')

#Plot the data on a scatter plot.

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 4):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob2Data2[prob2Data2.Cluster == cluster].X, prob2Data2[prob2Data2.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#PART C. Calculate the SSE or both clusterings.

singleLinkSSE, maxClusterContribSingle = sumSquaredError(prob2Data)

completeLinkSSE, maxClusterContribComplete = sumSquaredError(prob2Data2)

print "Sum squared error for single link: " + str(singleLinkSSE)

print "Cluster contributing most to SSE: " + str(maxClusterContribSingle[0])

print "Cluster SSE: " + str(maxClusterContribSingle[1])

print

print "Sum squared error for complete link: " + str(completeLinkSSE)

print "Cluster contributing most to SSE: " + str(maxClusterContribComplete[0])

print "Cluster SSE: " + str(maxClusterContribComplete[1])

#PART D. Build proximity and incidence matricies, and calculate the correlation for each clustering.

print

corr, pm, im = correlationClusterAnalysis(prob2Data)

print("Correlation coeff of single linkage clustering: " + str(corr[0][1]))

corr, pm, im = correlationClusterAnalysis(prob2Data2)

print("Correlation coeff of complete linkage clustering: " + str(corr[0][1]))

def sumSquaredError(dataSet):

totalSum = 0

#Store cluster with maximum contribution to sse as (cluster, SSE contrib)

maxClusterContribution = (0, 0)

#For each cluster

# For each point in each cluster

# Find squared distance between mean and point, and add to cluster sum.

#Sum all cluster values.

for cluster in range(1, 4):

#Get view of all data in the same cluster.

currentClusterData = dataSet[dataSet.Cluster == cluster]

meanX = currentClusterData.X.values.mean()

meanY = currentClusterData.Y.values.mean()

clusterSum = 0

for index, row in currentClusterData.iterrows():

clusterSum += math.pow(meanX-row.X, 2) + math.pow(meanY-row.Y, 2)

if clusterSum > maxClusterContribution[1]:

maxClusterContribution = (cluster, clusterSum)

totalSum += clusterSum

return totalSum, maxClusterContribution

def correlationClusterAnalysis(dataSet):

#Make an mxm matrix where m is the number of data points.

proximityMatrix = np.zeros((dataSet.shape[0], dataSet.shape[0]))

incidenceMatrix = np.zeros((dataSet.shape[0], dataSet.shape[0]))

for i in range(0, dataSet.shape[0]):

for j in range(i, dataSet.shape[0]):

distance = math.pow(math.pow(dataSet.iloc[i].X - dataSet.iloc[j].X, 2) + math.pow(dataSet.iloc[i].Y - dataSet.iloc[j].Y, 2), .5)

proximityMatrix[i,j] = distance

proximityMatrix[j,i] = distance

if dataSet['Cluster'].iloc[i] == dataSet['Cluster'].iloc[j]:

incidenceMatrix[i,j] = 1

incidenceMatrix[j,i] = 1

correlation = np.corrcoef(proximityMatrix.flatten(), incidenceMatrix.flatten())

return correlation, proximityMatrix, incidenceMatrix

def prob3():

global dataSet

global dataSet2

radius = 4

minpts = 3

pointsDict = {}

numClusters = 0

datalist = [1, 3, 5, 6, 8, 11, 12, 13, 14, 15, 16, 22, 28, 32, 33, 34, 35, 36, 37, 42, 58]

dataSet = pd.DataFrame(datalist, columns=['X'])

#Core points are 1, fringe points are 2, noise points are 0

dataSet['Pkind'] = np.zeros(len(datalist))

dataSet['Cluster'] = np.zeros(len(datalist))

#Find all core points.

for pointIndex in range(0, dataSet.shape[0]):

#Initialize num points to 0.

pointsDict[pointIndex] = 0

dist = 0

for comparisonIndex in range(0, dataSet.shape[0]):

dist = math.sqrt(math.pow(dataSet.iloc[pointIndex].X - dataSet.iloc[comparisonIndex].X, 2))

if dist <= radius:

pointsDict[pointIndex] = pointsDict[pointIndex] + 1

if pointsDict[pointIndex] >= minpts:

#Core point identified. Mark it and move to next point.

dataSet['Pkind'].iloc[pointIndex] = 1

continue

#Find all fringe points.

for fringePointIndex in dataSet[dataSet.Pkind == 0].index.tolist():

for comparisonIndex in dataSet[dataSet.Pkind == 1].index.tolist():

dist = math.sqrt(math.pow(dataSet.iloc[fringePointIndex].X - dataSet.iloc[comparisonIndex].X, 2))

if dist <= radius:

dataSet['Pkind'].iloc[fringePointIndex] = 2

continue

print("Noise points: Pkind = 0")

print("Core points: Pkind = 1")

print("Fringe points: Pkind = 2")

print dataSet

print

#Eliminate all points not core or fringe.

noise = dataSet.copy()[dataSet.Pkind == 0]

dataSet = dataSet[dataSet.Pkind != 0]

#For each core point

for corePointIndex in range(0, dataSet.shape[0]):

#If non-core point, continue.

if dataSet['Pkind'].iloc[corePointIndex] != 1:

continue

#If core point and no cluster, assign new cluster.

if dataSet['Cluster'].iloc[corePointIndex] == 0:

numClusters += 1

dataSet['Cluster'].iloc[corePointIndex] = numClusters

#Assign all points in vicinity of core to same cluster as core.

for comparisonIndex in range(0, dataSet.shape[0]):

dist = math.sqrt(math.pow(dataSet.iloc[corePointIndex].X - dataSet.iloc[comparisonIndex].X, 2))

if dist <= radius and dataSet['Cluster'].iloc[comparisonIndex] == 0:

dataSet['Cluster'].iloc[comparisonIndex] = dataSet['Cluster'].iloc[corePointIndex]

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in set(dataSet.Cluster.tolist()):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(dataSet[dataSet.Cluster == cluster].X, np.zeros(dataSet[dataSet.Cluster == cluster].shape[0]), label="Cluster " + str(cluster), color=randColor)

#Plot noise

colorHandles.append(matplotlib.patches.Patch(color='r', label='noise' + str(cluster)))

plt.scatter(noise.X, np.zeros(noise.shape[0]), color='r')

#Plot final graph of data.

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#PART B

radius = 6

minpts = 3

pointsDict = {}

numClusters = 0

dataSet2 = pd.DataFrame(datalist, columns=['X'])

#Core points are 1, fringe points are 2, noise points are 0

dataSet2['Pkind'] = np.zeros(len(datalist))

dataSet2['Cluster'] = np.zeros(len(datalist))

#Find all core points.

for pointIndex in range(0, dataSet2.shape[0]):

#Initialize num points to 0.

pointsDict[pointIndex] = 0

dist = 0

for comparisonIndex in range(0, dataSet2.shape[0]):

dist = math.sqrt(math.pow(dataSet2.iloc[pointIndex].X - dataSet2.iloc[comparisonIndex].X, 2))

if dist <= radius:

pointsDict[pointIndex] = pointsDict[pointIndex] + 1

if pointsDict[pointIndex] >= minpts:

#Core point identified. Mark it and move to next point.

dataSet2['Pkind'].iloc[pointIndex] = 1

continue

#Find all fringe points.

for fringePointIndex in dataSet2[dataSet2.Pkind == 0].index.tolist():

for comparisonIndex in dataSet2[dataSet2.Pkind == 1].index.tolist():

dist = math.sqrt(math.pow(dataSet2.iloc[fringePointIndex].X - dataSet2.iloc[comparisonIndex].X, 2))

if dist <= radius:

dataSet2['Pkind'].iloc[fringePointIndex] = 2

continue

print("Noise points: Pkind = 0")

print("Core points: Pkind = 1")

print("Fringe points: Pkind = 2")

print dataSet2

print

#Eliminate all points not core or fringe.

noise = dataSet2.copy()[dataSet2.Pkind == 0]

dataSet2 = dataSet2[dataSet2.Pkind != 0]

#For each core point

for corePointIndex in range(0, dataSet2.shape[0]):

#If non-core point, continue.

if dataSet2['Pkind'].iloc[corePointIndex] != 1:

continue

#If core point and no cluster, assign new cluster.

if dataSet2['Cluster'].iloc[corePointIndex] == 0:

numClusters += 1

dataSet2['Cluster'].iloc[corePointIndex] = numClusters

#Assign all points in vicinity of core to same cluster as core.

for comparisonIndex in range(0, dataSet2.shape[0]):

dist = math.sqrt(math.pow(dataSet2.iloc[corePointIndex].X - dataSet2.iloc[comparisonIndex].X, 2))

if dist <= radius and dataSet2['Cluster'].iloc[comparisonIndex] == 0:

dataSet2['Cluster'].iloc[comparisonIndex] = dataSet2['Cluster'].iloc[corePointIndex]

plt.clf()

colorHandles = []

#Create each cluster scatter plot.

for cluster in set(dataSet2.Cluster.tolist()):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(dataSet2[dataSet2.Cluster == cluster].X, np.zeros(dataSet2[dataSet2.Cluster == cluster].shape[0]), label="Cluster " + str(cluster), color=randColor)

#Plot noise

colorHandles.append(matplotlib.patches.Patch(color='r', label='noise' + str(cluster)))

plt.scatter(noise.X, np.zeros(noise.shape[0]), color='r')

#Plot final graph of data.

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#Part C

columnList = dataSet.columns.tolist()

columnList.remove('Pkind')

rindex = randIndex(dataSet[columnList], dataSet2[columnList])

print "Rand index = " + str(rindex)

def main():

print("In Main.")

prob1()

prob2()

prob3()

if \_\_name\_\_ == "\_\_main\_\_":

main()