Problem Set 4: Clustering

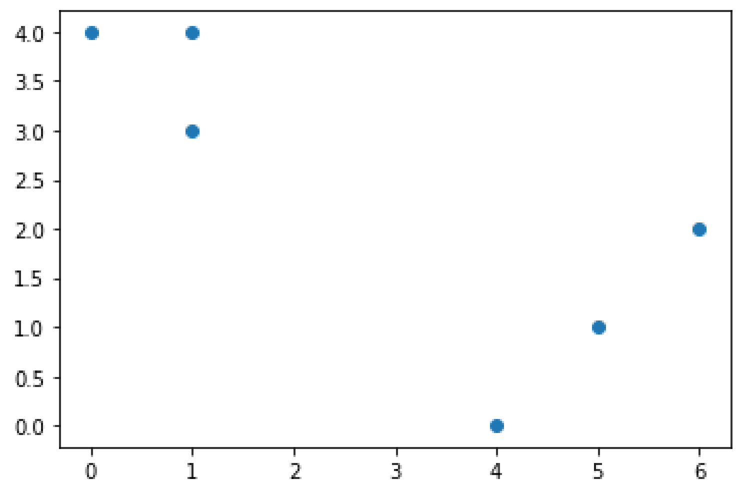
Performing k-Means by hand

In [35]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pyreadr
```

1. (5 points) Plot the observations.

In [43]:
```
sample = pd.DataFrame({'x': [1, 1, 0, 5, 6, 4], 'y': [4, 3, 4, 1, 2, 0]})
plt.scatter(sample['x'],sample['y'])
```

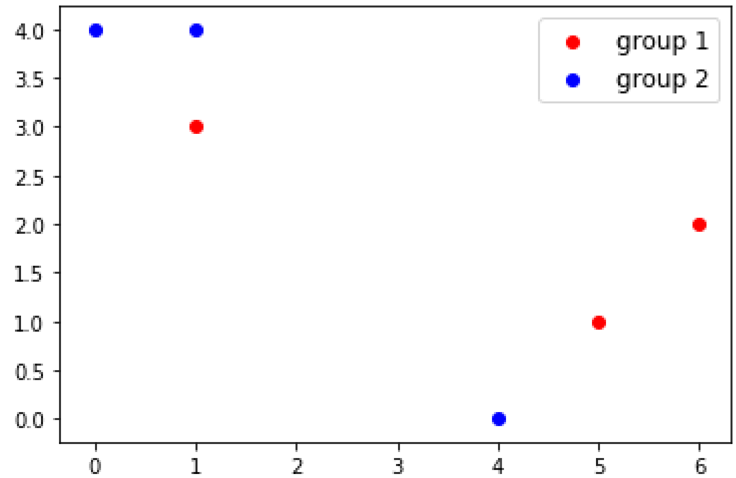
Out[43]: `<matplotlib.collections.PathCollection at 0x12c0d4590>`



2. (5 points) Randomly assign a cluster label to each observation. Report the cluster labels for each observation and plot the results with a different color for each cluster (remember to set your seed first).

In [79]:
```
np.random.seed(6)
rc = np.random.binomial(1, 0.5, size=6)
group1 = sample[rc == 0]
group2 = sample[rc == 1]
group1_ = plt.scatter(group1['x'], group1['y'], color='red')
group2_ = plt.scatter(group2['x'], group2['y'], color='blue')
plt.legend((group1_, group2_),
           ('group 1', 'group 2'),
           scatterpoints=1,
           loc='upper right')
plt.show()
```

Out[79]: <matplotlib.legend.Legend at 0x130dace50>



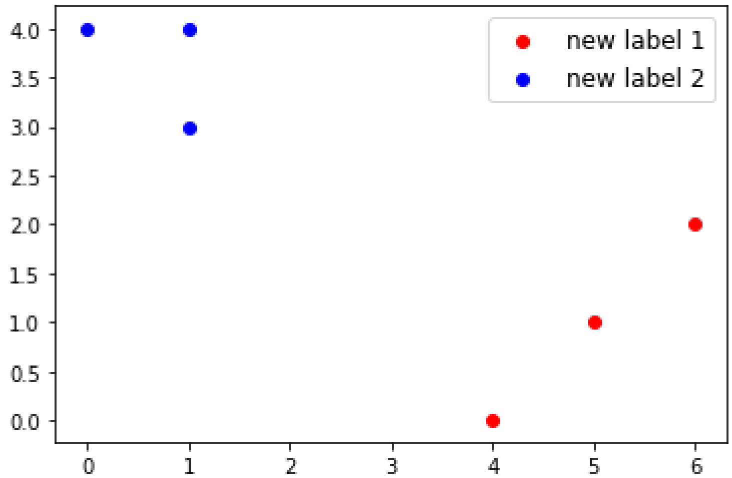
3. (10 points) Compute the centroid for each cluster.

In [83]:
```
c1, c2 = group1.mean(), group2.mean()
```

4. (10 points) Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

In [82]:
```
cond = (sample['x'] - c1['x'])**2 + (sample['y'] - c1['y'])**2 <= (sample['
new1 = sample[cond]
new2 = sample[~cond]
new1_ = plt.scatter(new1['x'], new1['y'], color='red')
new2_ = plt.scatter(new2['x'], new2['y'], color='blue')
plt.legend((new1_, new2_),
           ('new label 1', 'new label 2'),
            scatterpoints=1,
            loc='upper right')
plt.show()
```

Out[82]: <matplotlib.legend.Legend at 0x130e36790>



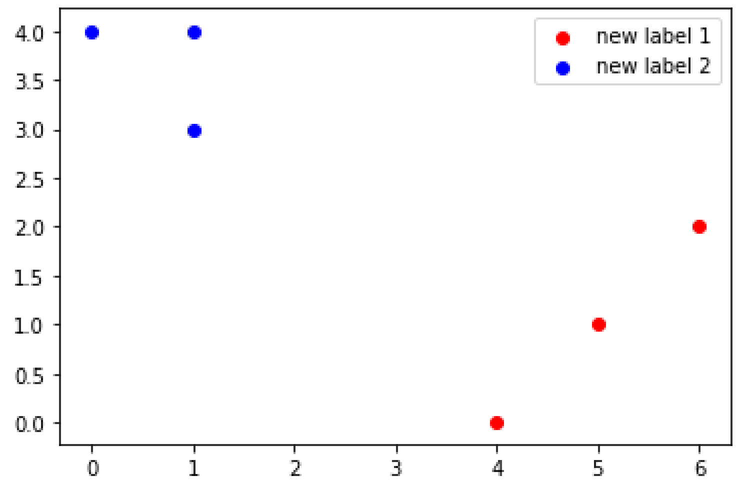
5. (5 points) Repeat (3) and (4) until the answers/clusters stop changing.

In [85]:
```
newC1, newC2 = new1.mean(), new2.mean()
newCond = (sample['x'] - newC1['x'])**2 + (sample['y'] - newC1['y'])**2 <=
newN1 = sample[newCond]
newN2 = sample[~newCond]
while newN1.equals(new1)!= True:
    new1 = newN1
    new2 = newN2
    newC1, newC2 = new1.mean(), new2.mean()
    newCond = (sample['x'] - newC1['x'])**2 + (sample['y'] -newC0['y'])**2
    newN1 = sample[newCond]
    newN2 = sample[~newCond]
```

6. (10 points) Reproduce the original plot from (1), but this time color the observations according to the clusters labels you obtained by iterating the cluster centroid calculation and assignments.

In [76]:
```
new1_ = plt.scatter(new1['x'],new1['y'], color='red')
new2_ = plt.scatter(new2['x'],new2['y'], color='blue')
plt.legend((new1_, new2_),
           ('new label 1', 'new label 2'),
            loc='upper right')
plt.show()
```

Out[76]: <matplotlib.legend.Legend at 0x130a36d90>



Clustering State Legislative Professionalism

1. Load the state legislative professionalism data. See the codebook (or above) for further reference.

In [113]:
```
data = pyreadr.read_r('legprof-components.v1.0.RData')
data['x']
```

In [114]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 11 columns):
fips           950 non-null int32
stateabv       950 non-null object
state          950 non-null object
sessid         950 non-null category
t_slength      889 non-null float64
slength        889 non-null float64
salary_real    945 non-null float64
expend         945 non-null float64
year           950 non-null float64
mds1           889 non-null float64
mds2           889 non-null float64
dtypes: category(1), float64(7), int32(1), object(2)
memory usage: 72.3+ KB
```

2. (5 points) Munge the data:
   a. select only the continuous features that should capture a state legislature's level of "professionalism" (session length (total and regular), salary, and expenditures)

In [119]:
```
data = data[['t_slength', 'slength', 'salary_real', 'expend', 'sessid']]
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49 entries, 18 to 949
Data columns (total 5 columns):
t_slength      49 non-null float64
slength        49 non-null float64
salary_real    49 non-null float64
expend         49 non-null float64
sessid         49 non-null category
dtypes: category(1), float64(4)
memory usage: 2.7 KB
```

b. restrict the data to only include the 2009/10 legislative session for consistency

In [120]:
```
dataNew = data[data.sessid.isin(['2009/10'])]
#data = data[['t_slength', 'slength', 'salary_real', 'expend']]
```

c. omit all missing values

In [121]:
```
dataNew = dataNew.dropna()
```

d. standardize the input features

In [122]:
```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
column_names_to_normalize = ['t_slength', 'slength', 'salary_real', 'expend
x = dataNew[column_names_to_normalize].values
x_scaled = min_max_scaler.fit_transform(x)
df_temp = pd.DataFrame(x_scaled, columns=column_names_to_normalize, index =
dataNew=pd.DataFrame()
dataNew[column_names_to_normalize] = df_temp
```

In [123]:
```
dataNew.describe()
```

Out[123]:

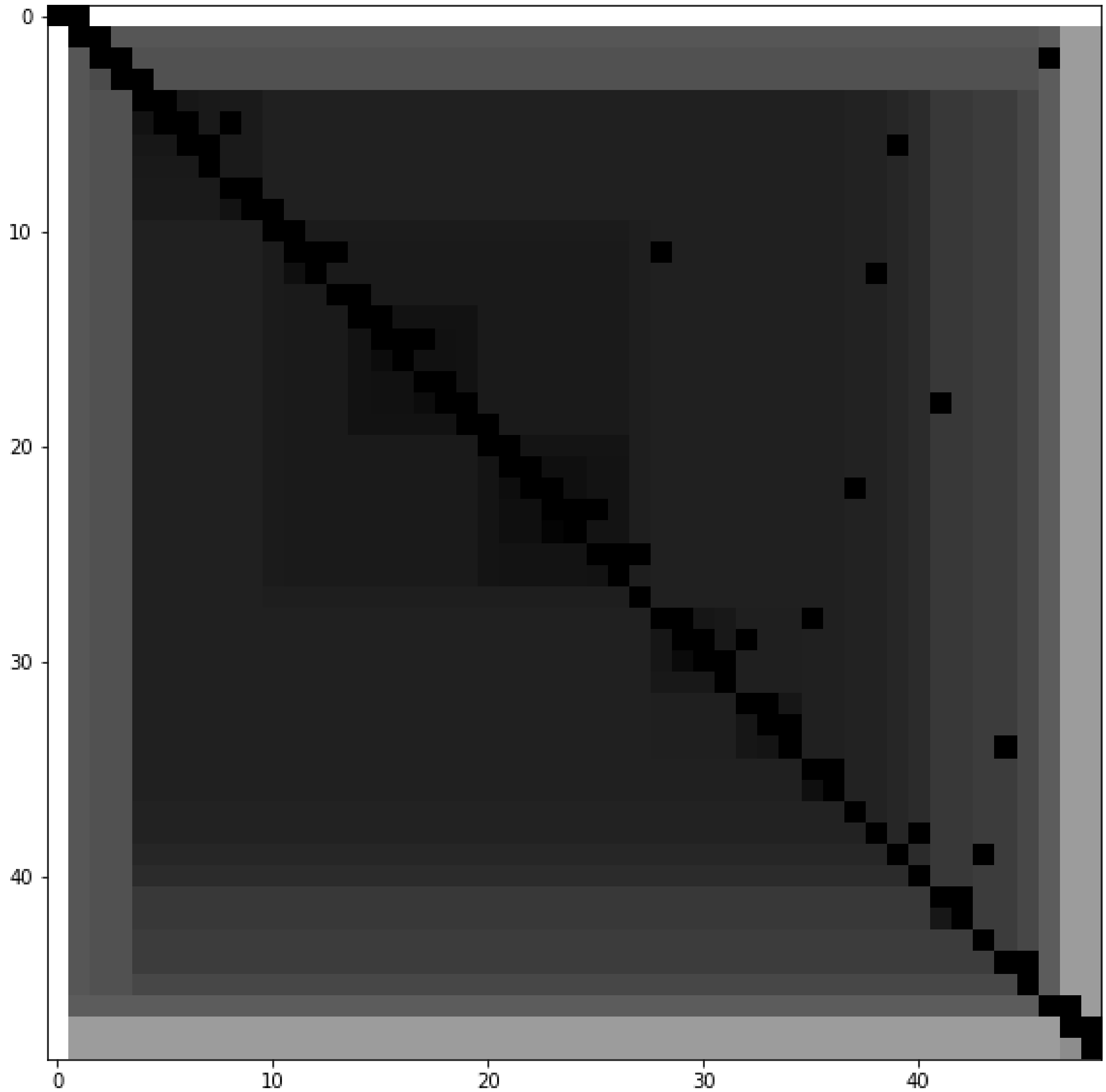
	t_slength	slength	salary_real	expend
count	49.000000	49.000000	49.000000	49.000000
mean	0.257797	0.254540	0.257685	0.123617
std	0.201068	0.191109	0.231468	0.159966
min	0.000000	0.000000	0.000000	0.000000
25%	0.137319	0.136898	0.092285	0.037899
50%	0.209901	0.214387	0.188974	0.085227
75%	0.284587	0.287305	0.362830	0.120030
max	1.000000	1.000000	1.000000	1.000000

e. and anything else you think necessary to get this subset of data into workable form (hint: consider storing the state names as a separate object to be used in plotting later)

3. (5 points) Diagnose clusterability in any way you'd prefer (e.g., sparse sampling, ODI, etc.); display the results and discuss the likelihood that natural, non-random structure exist in these data.

In [124]:
```
from pyclustertend import ivat
```

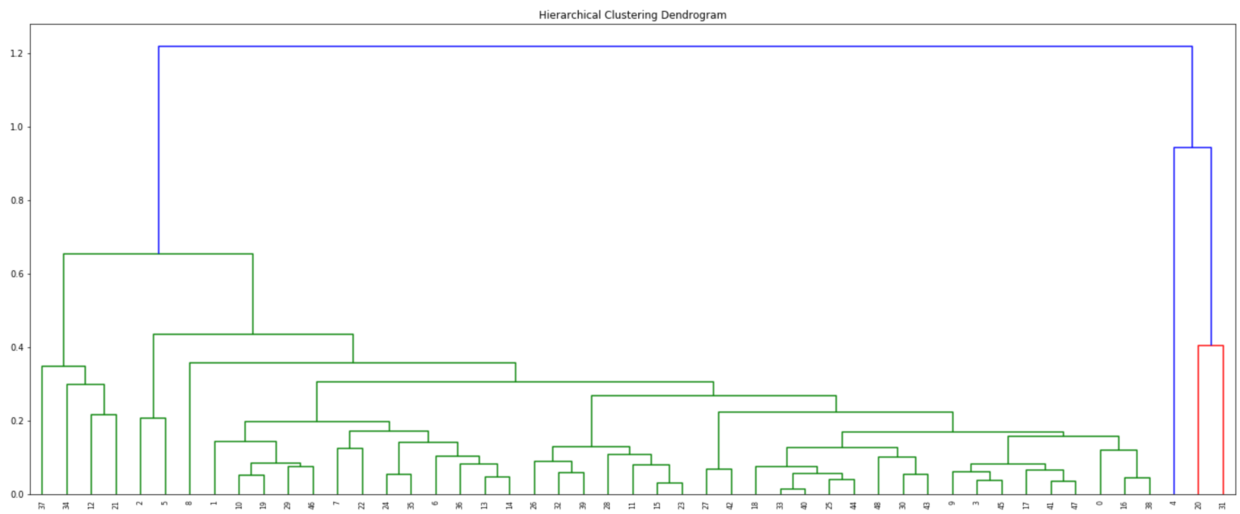
In [125]:
```
ivat(dataNew[['t_slength', 'slength', 'salary_real', 'expend']])
```



Based on the plot, we can tell there are a few clusters that can be explored.

4. Fit an agglomerative hierarchical clustering algorithm using any linkage method you prefer, to these data and present the results. Give a quick, high level summary of the output and general patterns.

In [134]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
dataDD=dataNew.merge(data['stateabv'], how = 'left', left_index=True, right
Z = linkage(dataDD, 'average')
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(Z,
           leaf_rotation=90.,
           leaf_font_size=8., )
plt.show()
```



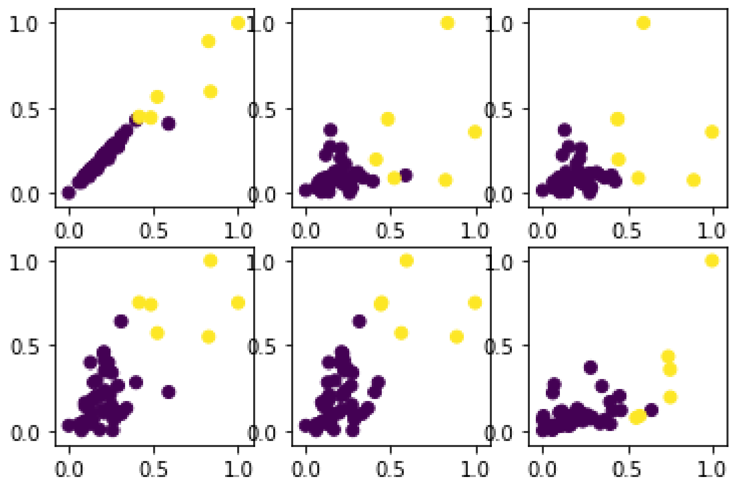
From the dendrogram, CA, MA and NY seem to be different from other states at the beginning, other states seem to have a pretty good clustering pattern in professionalism.

5. (5 points) Fit a k-means algorithm to these data and present the results. Give a quick, high level summary of the output and general patterns. Initialize the algorithm at k = 2, and then check this assumption in the validation questions below.

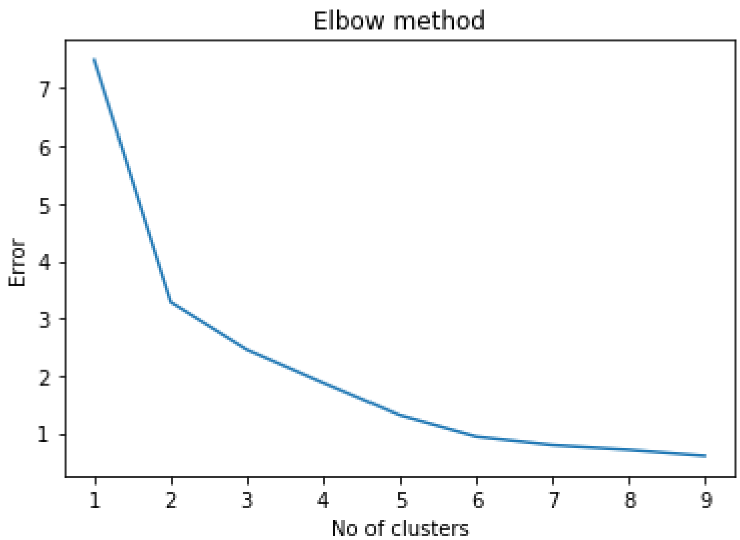
In [103]:
```
from sklearn.cluster import KMeans
kmeans2 = KMeans(n_clusters=2)
y_kmeans2 = kmeans2.fit_predict(data4)
kmean_result = pd.DataFrame(index = dataDD.index)
kmean_result['kmean_classification']=y_kmeans2
```

In [108]:
```
fig, axs = plt.subplots(2, 3)
axs[0, 0].scatter(dataDD['t_slength'], dataDD['slength'], c=y_kmeans2)
axs[1, 0].scatter(dataDD['t_slength'], dataDD['salary_real'], c=y_kmeans2)
axs[0, 1].scatter(dataDD['t_slength'], dataDD['expend'], c=y_kmeans2)
axs[1, 1].scatter(dataDD['slength'], dataDD['salary_real'], c=y_kmeans2)
axs[0, 2].scatter(dataDD['slength'], dataDD['expend'], c=y_kmeans2)
axs[1, 2].scatter(dataDD['salary_real'], dataDD['expend'], c=y_kmeans2)
```

Out[108]: <matplotlib.collections.PathCollection at 0x1322f00d0>



In [27]:
```
Error =[]
for i in range(1, 10):
    kmeans = KMeans(n_clusters = i).fit(data4)
    Error.append(kmeans.inertia_)
import matplotlib.pyplot as plt
plt.plot(range(1, 10), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```

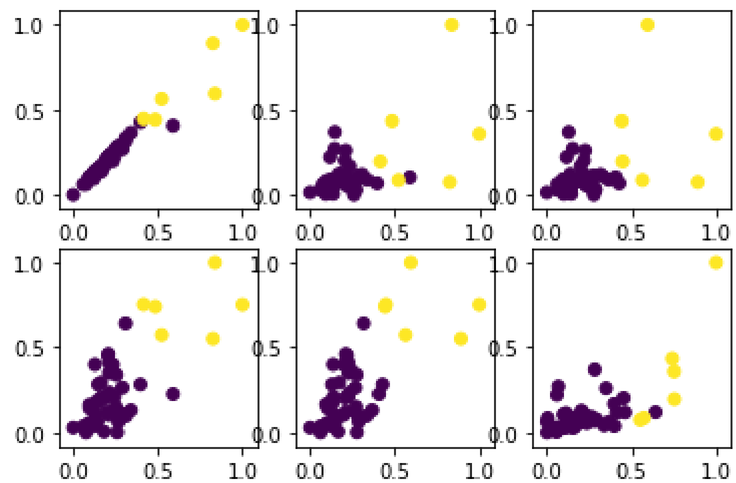



```
Based on the elbow method, I think k = 2 seems to be a good number for
clustering.
```

6. (5 points) Fit a Gaussian mixture model via the EM algorithm to these data and present the results. Give a quick, high level summary of the output and general patterns. Initialize the algorithm at k = 2, and then check this assumption in the validation questions below.

In [110]:
```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=2).fit(dataDD)
gmmL = gmm.predict(dataDD)
fig, axs = plt.subplots(2, 3)
axs[0, 0].scatter(dataDD['t_slength'], dataDD['slength'], c=gmmL)
axs[1, 0].scatter(dataDD['t_slength'], dataDD['salary_real'], c=gmmL)
axs[0, 1].scatter(dataDD['t_slength'], dataDD['expend'], c=gmmL)
axs[1, 1].scatter(dataDD['slength'], dataDD['salary_real'], c=gmmL)
axs[0, 2].scatter(dataDD['slength'], dataDD['expend'], c=gmmL)
axs[1, 2].scatter(dataDD['salary_real'], dataDD['expend'], c=gmmL)
```

Out[110]: <matplotlib.collections.PathCollection at 0x132994810>



7. (15 points) Compare output of all in visually useful, simple ways (e.g., present the dendrogram, plot by state cluster assignment across two features like salary and expenditures, etc.). There should be several plots of comparison and output.

The dendrogram and kmeans are showing pretty the same results. The only difference is if Illinois can be recongized in larger or samller clusters.

8. (5 points) Select a single validation strategy (e.g., compactness via min(WSS), average silhouette width, etc.), and calculate for all three algorithms. Display and compare your results for all three algorithms you fit (hierarchical, k-means, GMM). Hint: Here again, we didn't cover this in R in class, but think about using the clValid package, though there are many other packages and ways to validate cluster patterns across iterations.

In [111]:

```
c1 = dataDD[hierarchical_result.hierarchical_classification==0].mean()
c2 = dataDD[hierarchical_result.hierarchical_classification==1].mean()
agglomerative_wss = ((dataDD[hierarchical_result.hierarchical_classificatic
print("Agglomerative clustering's WSS is", agglomerative_wss)
kmeans2 = KMeans(n_clusters = 2).fit(dataDD)
kmeans2.inertia_
print("Kmeans' WSS is", kmeans2.inertia_)
gmm_wss = ((dataDD[gmm_result.gmm_classification==0]-gmm.means_[0])**2).sum
print("Gmm' WSS is", gmm_wss)
```

```
Agglomerative clustering's WSS is 3.321908124470716
Kmeans' WSS is 3.2842452937295263
Gmm' WSS is 3.284262677128642
```

9. (10 points) Discuss the validation output, e.g.,

- What can you take away from the fit?
- Which approach is optimal? And optimal at what value of k?
- What are reasons you could imagine selecting a technically “sub-optimal” clustering method, regardless of the validation statistics?

```
Firstly, when k = 2, it is hard to tell which method is better. We
probably need to tune the hyperparameter k to gain better insights.
Secondly, based the result above, the optimal approach is kmeans with k =
2. Lastly, I think the reasons can be stemmed from interpretability or
other charateristics. Lke, kmeans is pretty friendly method because it can
be easily interpreted (based on spacial distance to cluster).
```

In [ ]: