hw4_aml_task1_2

April 4, 2018

```
In [175]: import warnings
          warnings.filterwarnings('ignore')
          %matplotlib inline
          import numpy as np
          import matplotlib.pyplot as plt
          import pandas as pd
          from sklearn.decomposition import PCA
          from sklearn.manifold import TSNE
          from scipy.io import loadmat
          from sklearn.preprocessing import StandardScaler
          from sklearn.pipeline import make_pipeline
          from sklearn.cluster import KMeans
          import itertools
          from scipy.spatial import Voronoi, voronoi_plot_2d
          from scipy.cluster.hierarchy import dendrogram, linkage
          from sklearn.cluster import DBSCAN
          from sklearn.metrics import normalized_mutual_info_score as nmi
          from sklearn.metrics import adjusted_rand_score as ari
          from sklearn.cluster import AgglomerativeClustering

In [5]: data = loadmat('annthyroid.mat')

In [9]: data

Out[9]: {'X': array([[  7.30000000e-01,   6.00000000e-04,   1.50000000e-02,
                        1.20000000e-01,   8.20000000e-02,   1.46000000e-01],
                    [  2.40000000e-01,   2.50000000e-04,   3.00000000e-02,
                        1.43000000e-01,   1.33000000e-01,   1.08000000e-01],
                    [  4.70000000e-01,   1.90000000e-03,   2.40000000e-02,
                        1.02000000e-01,   1.31000000e-01,   7.80000000e-02],
                    ...,
                    [  5.10000000e-01,   7.60000000e-04,   2.01000000e-02,
                        9.00000000e-02,   6.70000000e-02,   1.34000000e-01],
                    [  3.50000000e-01,   2.80000000e-03,   2.01000000e-02,
                        9.00000000e-02,   8.90000000e-02,   1.01000000e-01],
                    [  7.30000000e-01,   5.60000000e-04,   2.01000000e-02,
                        8.10000000e-02,   9.00000000e-02,   9.00000000e-02]]),
           '__globals__': [],
```

```
'__header__': b'MATLAB 5.0 MAT-file, Platform: MACI64, Created on: Thu Aug 18 14:56:12
'__version__': '1.0',
'y': array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]], dtype=uint8)}
```

In [17]:
```
col_names = ['a','b','c','d','e','f']
all_data = pd.DataFrame(data['X'],columns=col_names)
all_data.head()
```

Out[17]:

	a	b	c	d	e	f
0	0.73	0.00060	0.015	0.120	0.082	0.146
1	0.24	0.00025	0.030	0.143	0.133	0.108
2	0.47	0.00190	0.024	0.102	0.131	0.078
3	0.64	0.00090	0.017	0.077	0.090	0.085
4	0.23	0.00025	0.026	0.139	0.090	0.153

In [19]:
```
y_truth = pd.DataFrame(data['y'],columns=['label'])
y_truth.head()
```

Out[19]:

	label
0	0
1	0
2	0
3	0
4	0

In [22]:
```
set(y_truth['label'])
```

Out[22]: {0, 1}

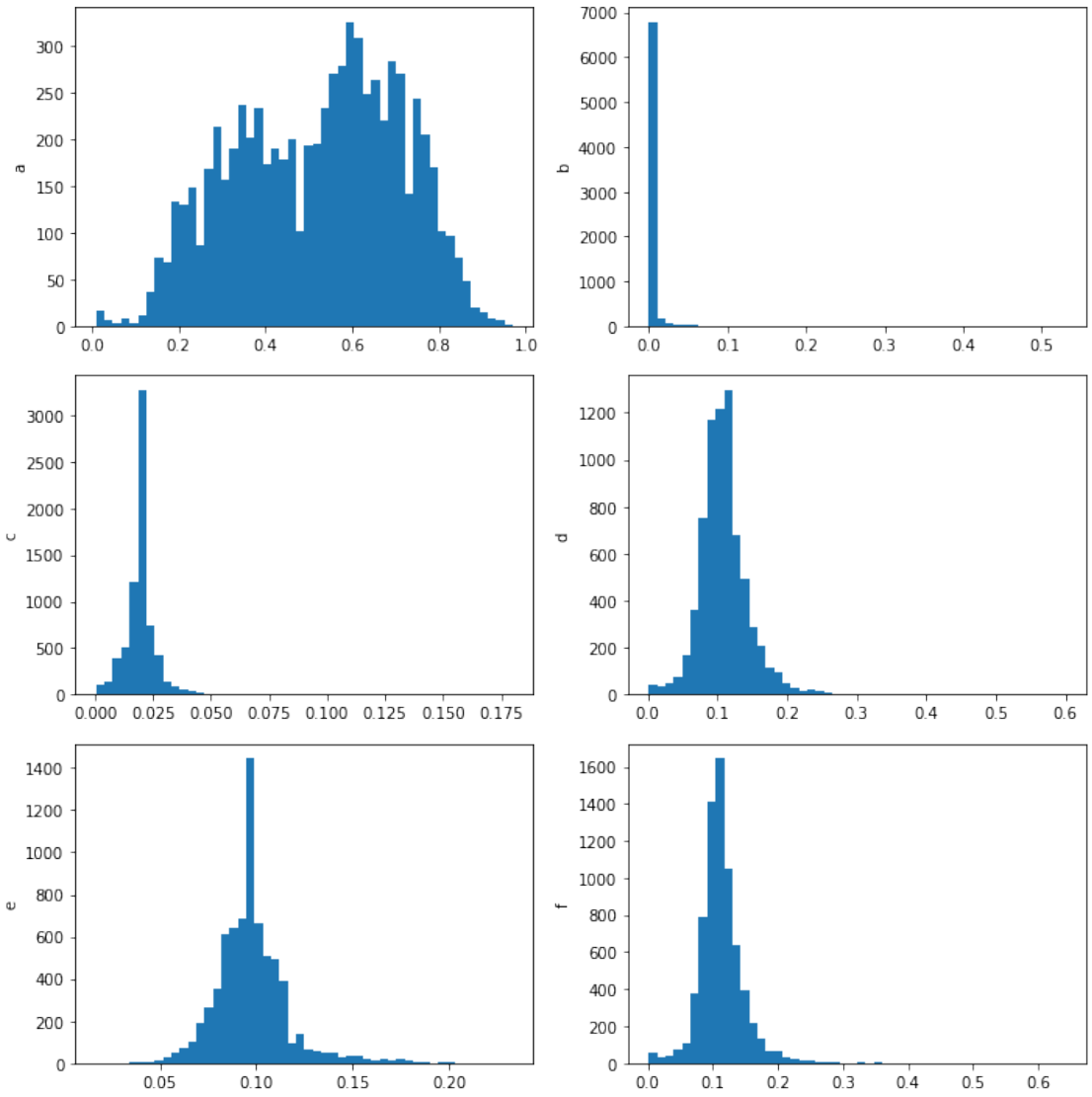
0.1 Task 1.1

In [23]:
```
all_data0 = all_data.loc[y_truth['label']==0]
all_data1 = all_data.loc[y_truth['label']==1]
print (all_data0.shape,all_data1.shape,)
```

```
(6666, 6) (534, 6)
```

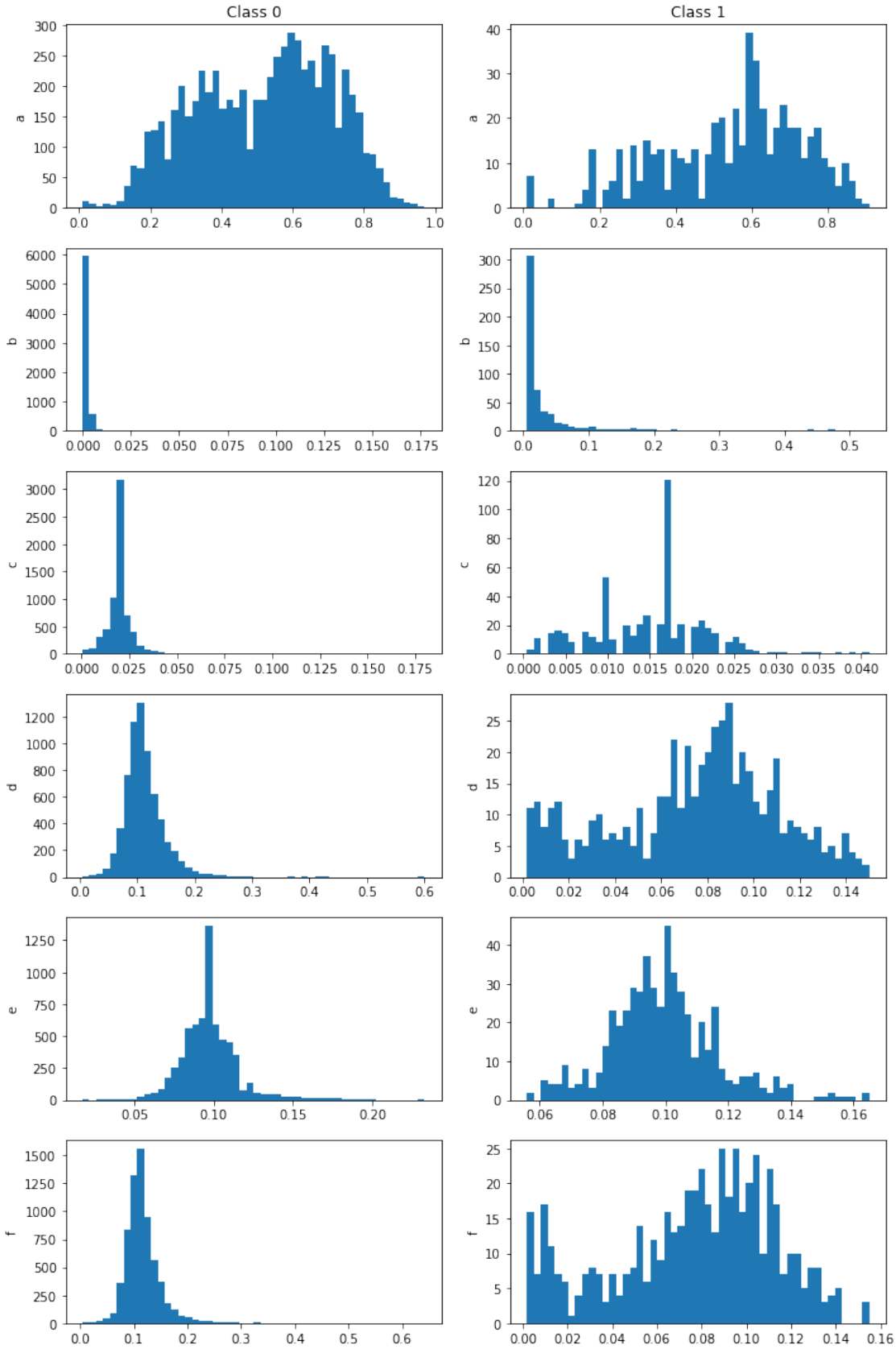
In [24]:
```
#Visualize the univariate distributions of all features, jointly
fig,ax = plt.subplots(3,2, figsize=(10,10))
p = 0
for i in range(3):
    for j in range(2):
```

```
            ax[i,j].hist(all_data[col_names[p]], bins=50)
            ax[i,j].set_ylabel(col_names[p])
            p+=1
    #ax[12].set_xlabel('MEDV')

    plt.tight_layout()
    plt.show()
```




```
In [25]: #Visualize the univariate distributions of all features per class.
         fig,ax = plt.subplots(6,2, figsize=(10,15))
         p = 0
         for i in range(6):
```

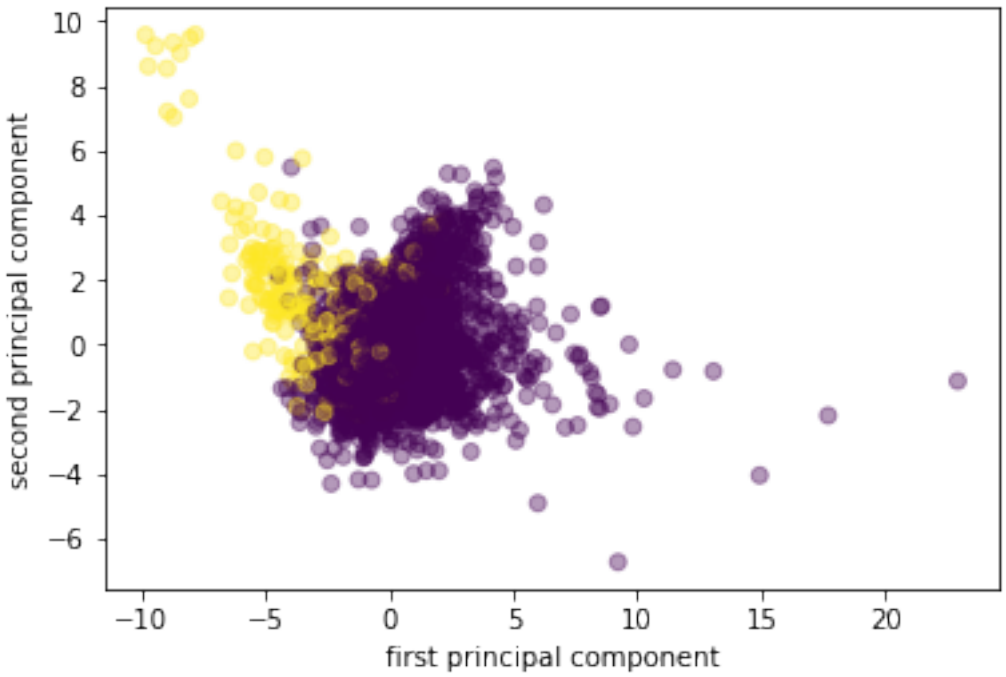
```
    for j in range(2):
        if j == 0:
            ax[i,j].hist(all_data0[col_names[p]], bins=50)
            ax[i,j].set_ylabel(col_names[p])
        else:
            ax[i,j].hist(all_data1[col_names[p]], bins=50)
            ax[i,j].set_ylabel(col_names[p])
    p+=1

ax[0,0].set_title('Class 0')
ax[0,1].set_title('Class 1')

plt.tight_layout()
plt.show()
```


Class 0
Class 1
a
b
c
d
e
f

In [44]:
```
# all_data_x = all_data.loc[:,all_data.columns != 'ann_th']
# all_data_y = pd.DataFrame(all_data.loc[:,all_data.columns == 'ann_th'])
```

In [40]:
```
pca_scaled = make_pipeline(StandardScaler(), PCA(n_components=2))
all_data_pca = pca_scaled.fit_transform(all_data)
print(all_data_pca.shape)
plt.scatter(all_data_pca[:, 0], all_data_pca[:, 1], c=y_truth['label'],alpha = 0.4)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
plt.show()
```

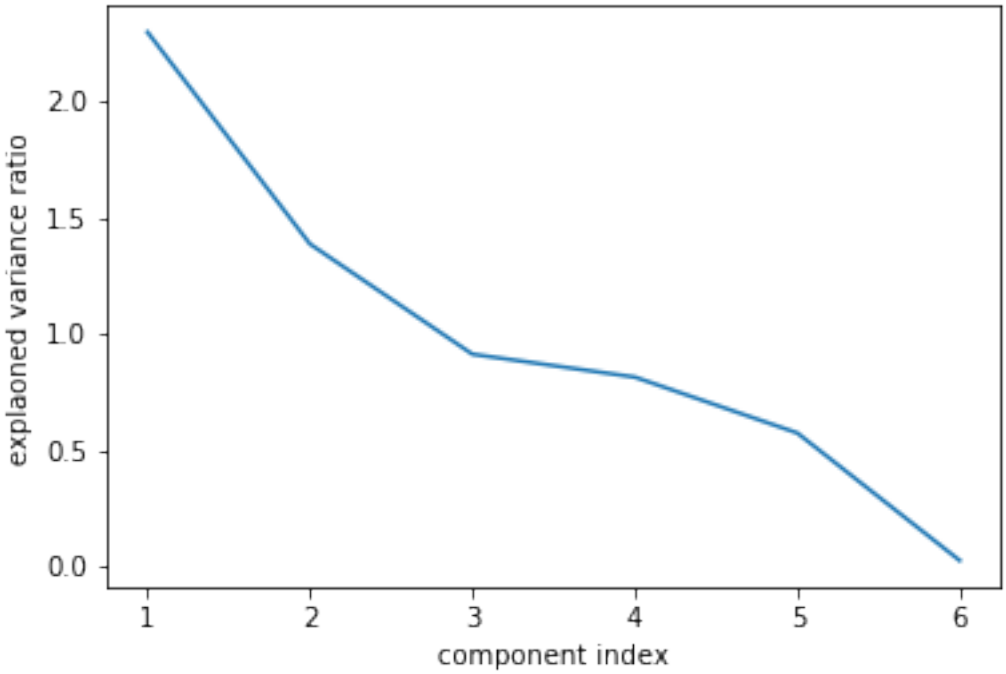
```
(7200, 2)
```



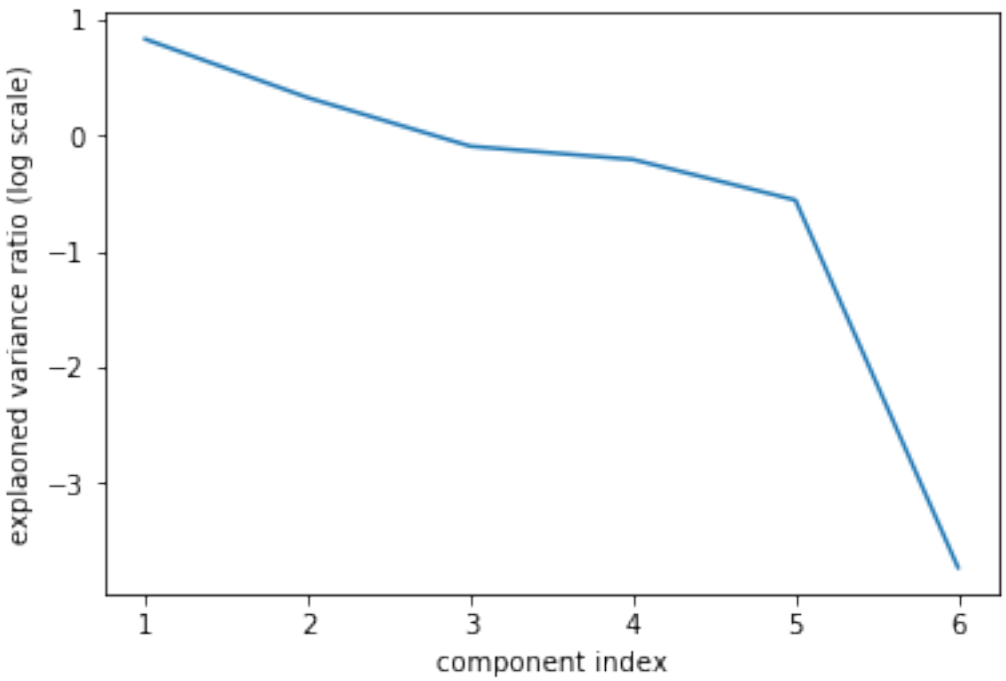
In [47]:
```
scaler = StandardScaler()
all_data_scaled = scaler.fit_transform(all_data) #scaling data
pca2 = PCA(n_components=6)
pca2.fit_transform(all_data_scaled)
pca2.explained_variance_
```

Out[47]:
```
array([ 2.29601111,  1.38494736,  0.91098472,  0.81249316,  0.57237237,
        0.02402473])
```

In [48]:
```
plt.plot(range(1,7),pca2.explained_variance_)
plt.xlabel("component index")
plt.ylabel("explaoned variance ratio")
plt.xticks(np.arange(1, 7, step=1))
plt.show()
```



In [49]:
```
plt.plot(range(1,7),np.log(pca2.explained_variance_))
plt.xlabel("component index")
plt.ylabel("explaoned variance ratio (log scale)")
plt.xticks(np.arange(1, 7, step=1))
plt.show()
```



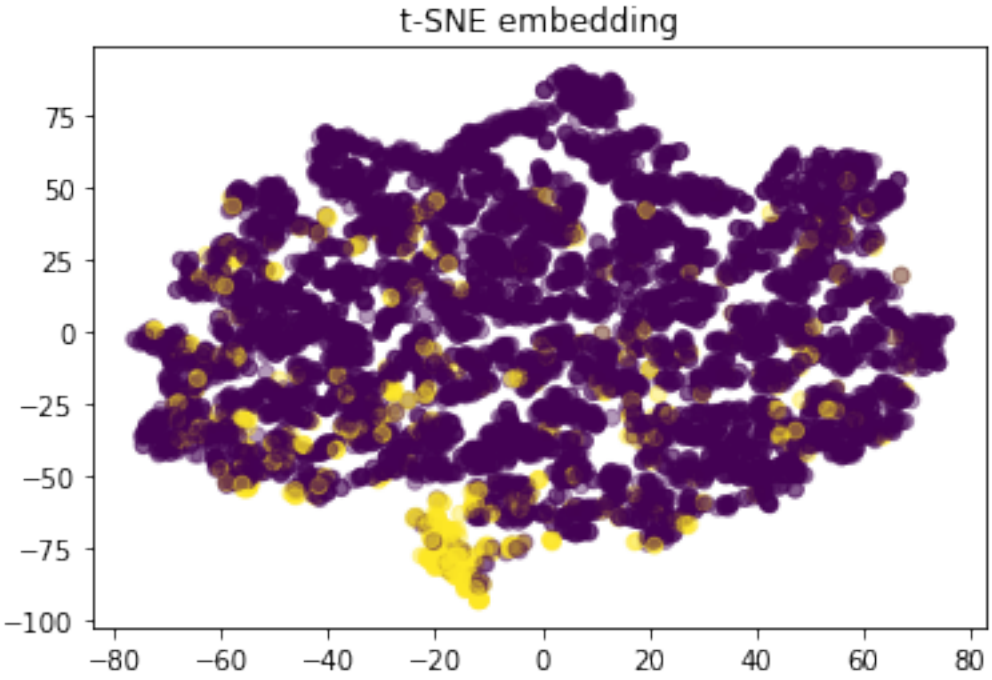
As seen from above 2 plots, 5 components seem to cover major variance. Therfore, we can use 5 components to reduce our data

0.2 Task 1.2

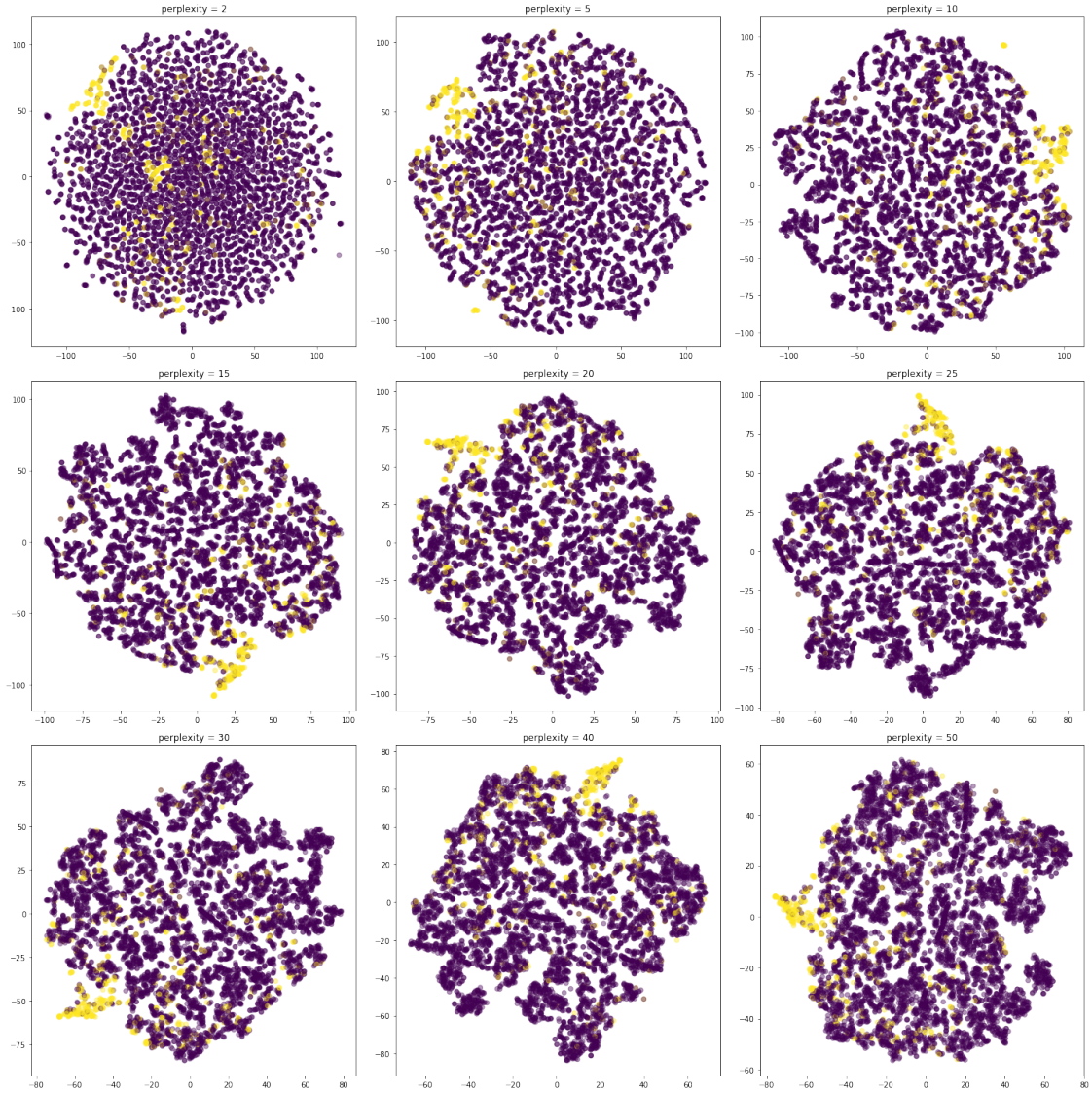
Visualize the data using t-SNE. See if tuning the perplexity parameter helps obtaining a better visualization.

```
In [51]: tsne = TSNE()
         all_data_sc_tsne = tsne.fit_transform(all_data_scaled)
         all_data_sc_tsne

Out[51]: array([[-51.26181412,  32.24555588],
                [ 52.93050003,  34.06492996],
                [  5.26826572,  37.53252411],
                ...,
                [  2.89794707, -28.22206306],
                [ 47.72912598, -40.4499054 ],
                [-27.30810356, -40.97901154]], dtype=float32)

In [52]: print(all_data_sc_tsne.shape)
         plt.scatter(all_data_sc_tsne[:, 0], all_data_sc_tsne[:, 1],c=y_truth['label'],alpha = 0
         plt.title('t-SNE embedding')
         plt.show()
```

```
(7200, 2)
```




```
In [126]: perplexity = [2,5,10,15,20,25,30,40,50]
          p = 0
          fig,ax = plt.subplots(3,3, figsize=(20,20))
          for i in range(3):
              for j in range(3):
                  all_data_sc_tsne = TSNE(perplexity=perplexity[p]).fit_transform(all_data_scale
                  ax[i,j].scatter(all_data_sc_tsne[:, 0], all_data_sc_tsne[:, 1],c=y_truth['labe
                  ax[i,j].set_title('perplexity = '+str(perplexity[p]))
                  p+=1
          plt.tight_layout()
          plt.show()
```



Yes, adjusting perplexity helps a bit in visualizing data but still it's not easy to separate the outlier and inlier clusters clearly.

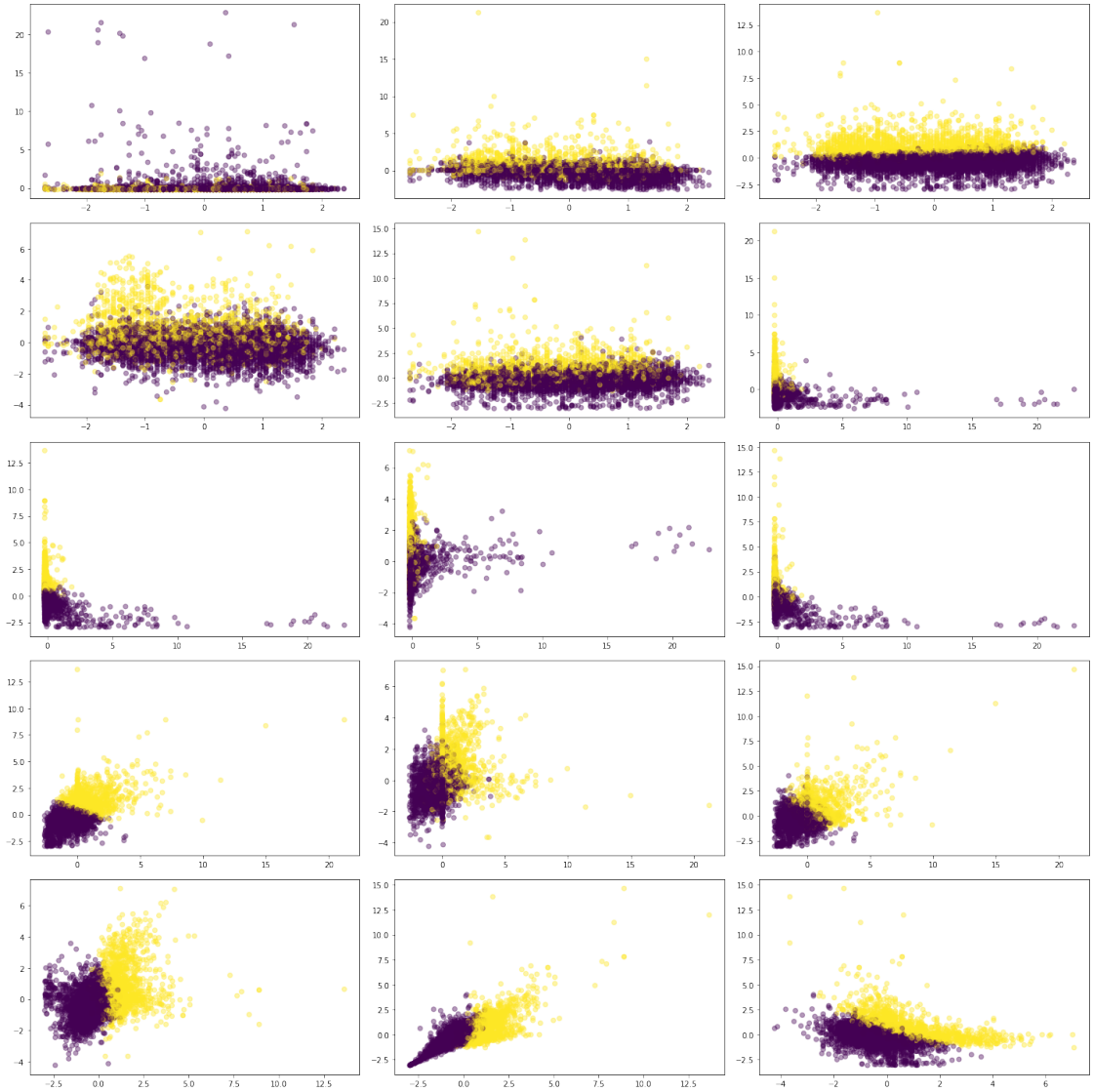
0.3 Task 2.1

2.1: Use KMeans, Agglomerative Clustering and DBSCAN to cluster the data. For each algorithm, try to manually tune the parameters for a reasonable outcome and document how you tuned the parameters. In particular pay attention to the sizes of the clusters created. Create a dendrogram for agglomerative clustering (the truncate_mode='level' might be useful). Manually inspect the outcomes as good as you can and identify if any of the resulting clusters are semantically meaningful (as far as you can tell)

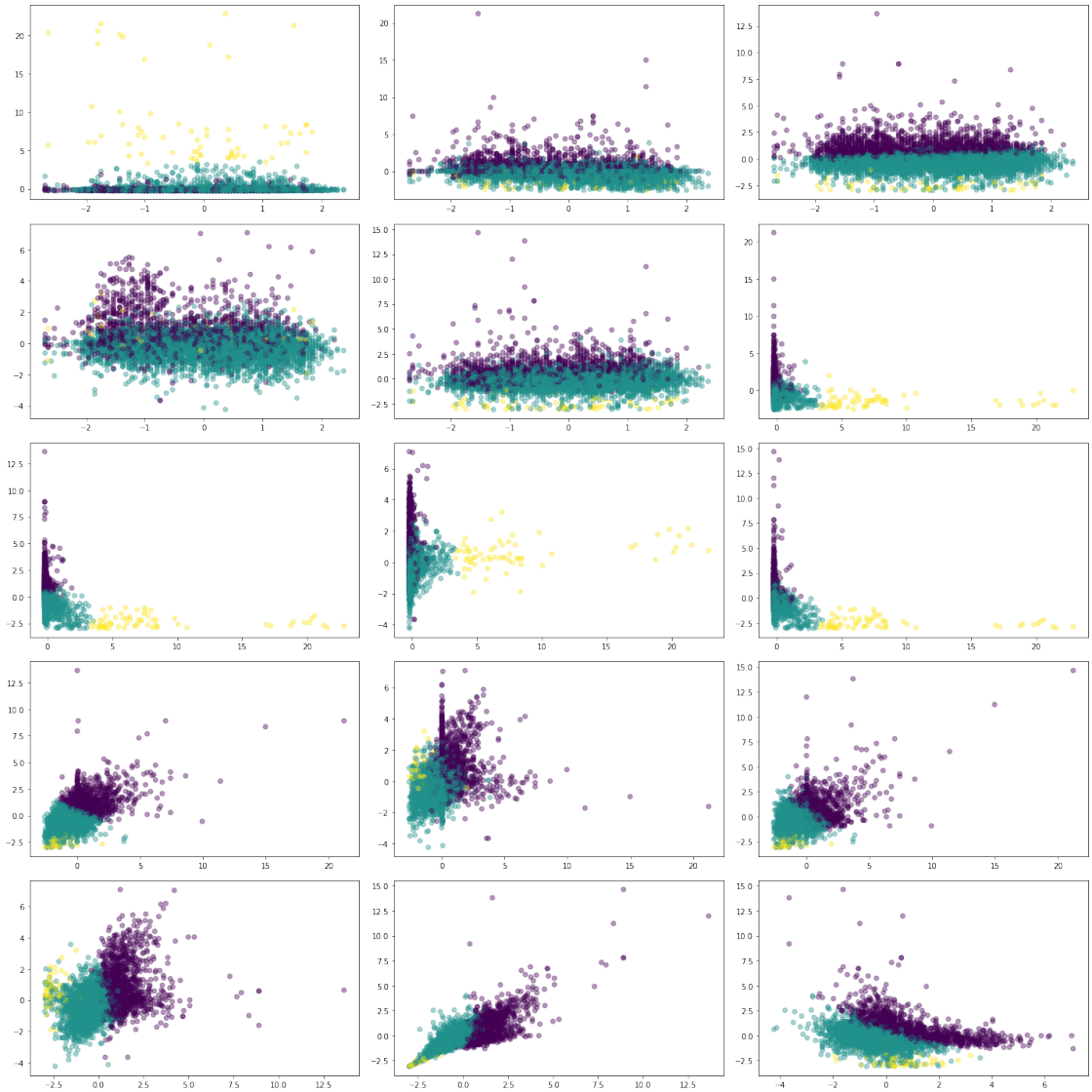
```
In [234]: km_dict = {}
          clusters = [2,3,4,5,6]
          for i in clusters:
              km = KMeans(n_clusters=i, random_state=0)
              km.fit(all_data_scaled)
              km_dict['km'+str(i)] = km
              print(km.cluster_centers_.shape)
              print(km.labels_.shape)
              print('sum of sq. distances : ',km.inertia_)

(2, 6)
(7200,)
sum of sq. distances :  35401.8381629
(3, 6)
(7200,)
sum of sq. distances :  30075.6213259
(4, 6)
(7200,)
sum of sq. distances :  25755.2933391
(5, 6)
(7200,)
sum of sq. distances :  22836.1289766
(6, 6)
(7200,)
sum of sq. distances :  20368.8827189


In [232]: # km_dict['km4'].cluster_centers_

In [230]: # lst = km_dict['km4'].cluster_centers_
          # [item[[1,2]] for item in lst]

In [231]: # vor = Voronoi([item[[4,5]] for item in lst])
          # plt2 = voronoi_plot_2d(vor)
          # plt2.show()

In [85]: comb = []
         for i in itertools.combinations(range(6), 2):
             comb.append(i)

[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5),


In [93]: k = [2,3,4,5]

         for q in k:
             print ('clusters=',q)
             p = 0
             fig,ax = plt.subplots(5,3, figsize=(20,20))
```

```
            for i in range(5):
                for j in range(3):
                    ax[i,j].scatter(all_data_scaled[:, comb[p][0]], all_data_scaled[:, comb[p][
                                    c=km_dict['km'+str(q)].labels_,alpha = 0.4)
            #         ax[i,j].set_title('perplexity = '+str(perplexity[p]))
                    p+=1
            plt.tight_layout()
            plt.show()
```

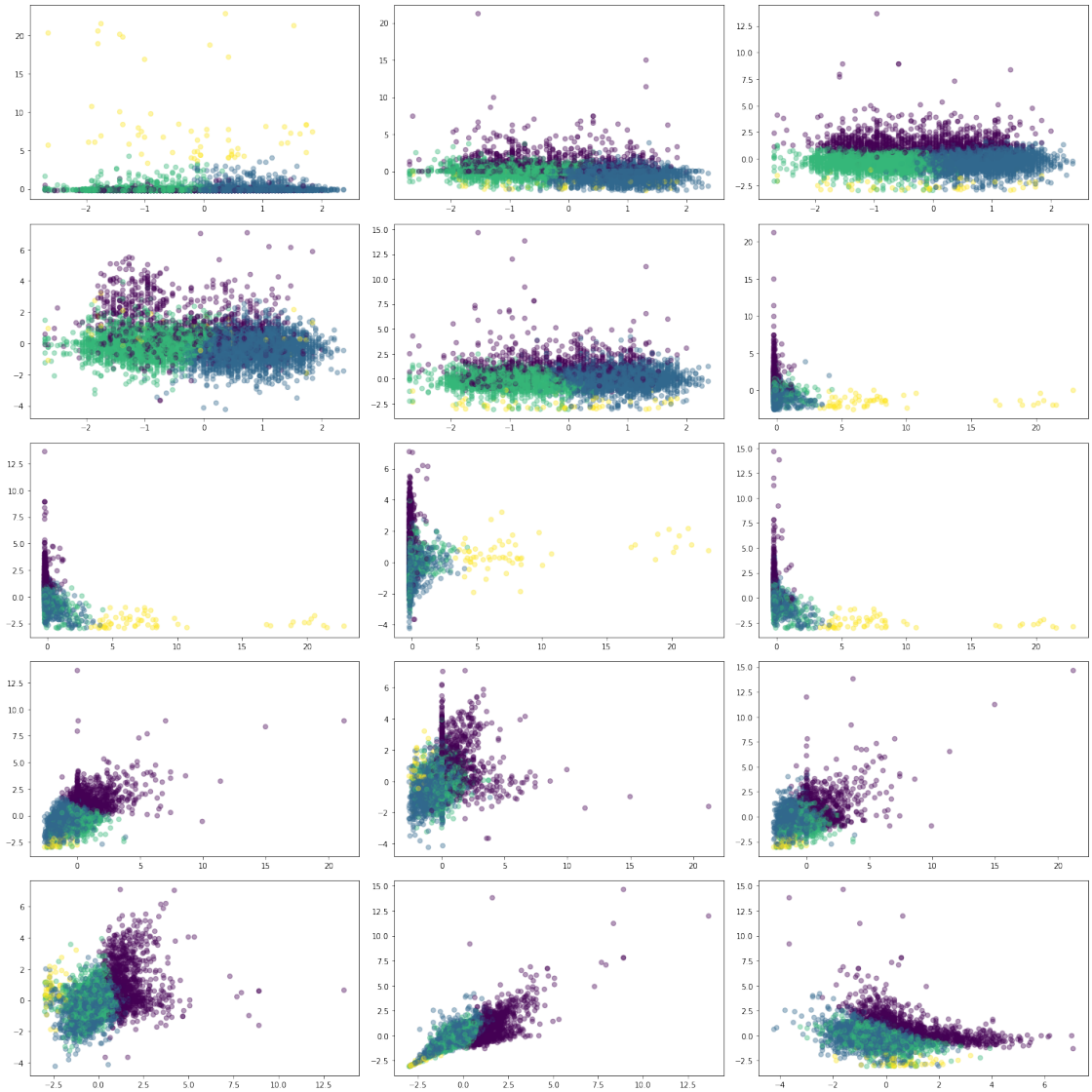
```
clusters= 2
```



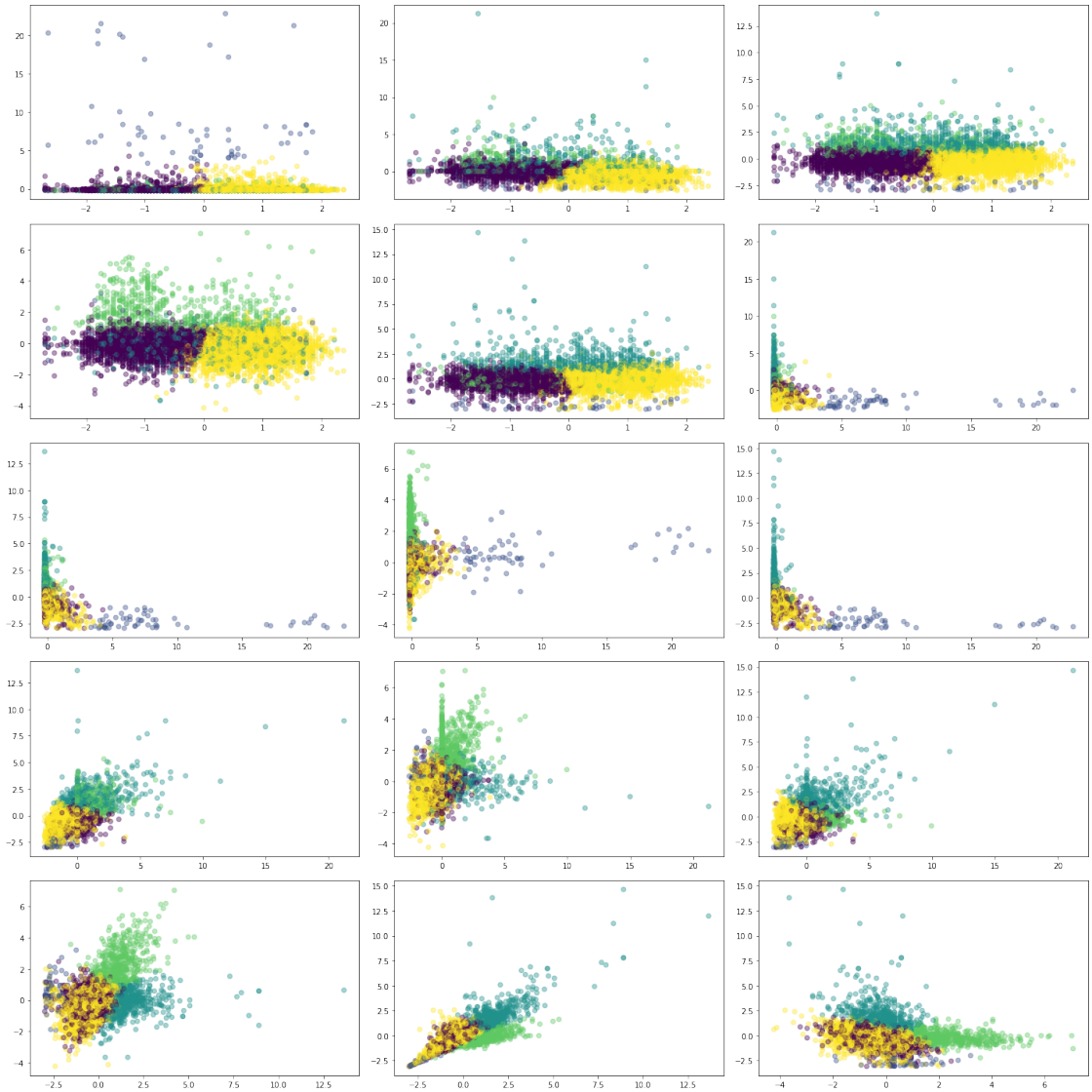
```
clusters= 3
```



```
clusters= 4
```

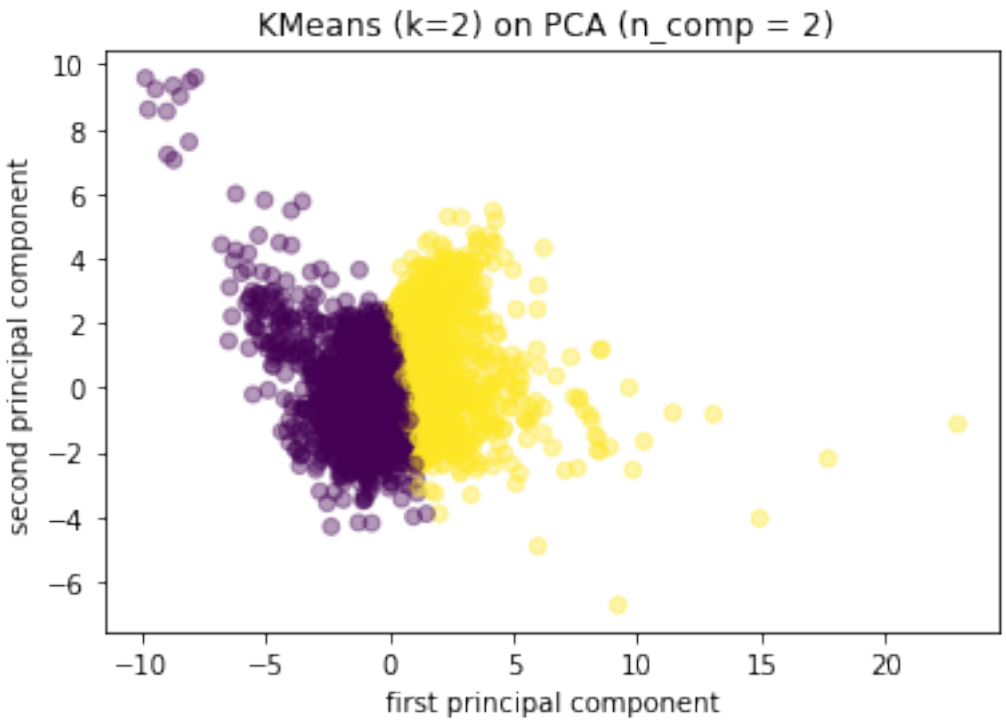


```
clusters= 5
```

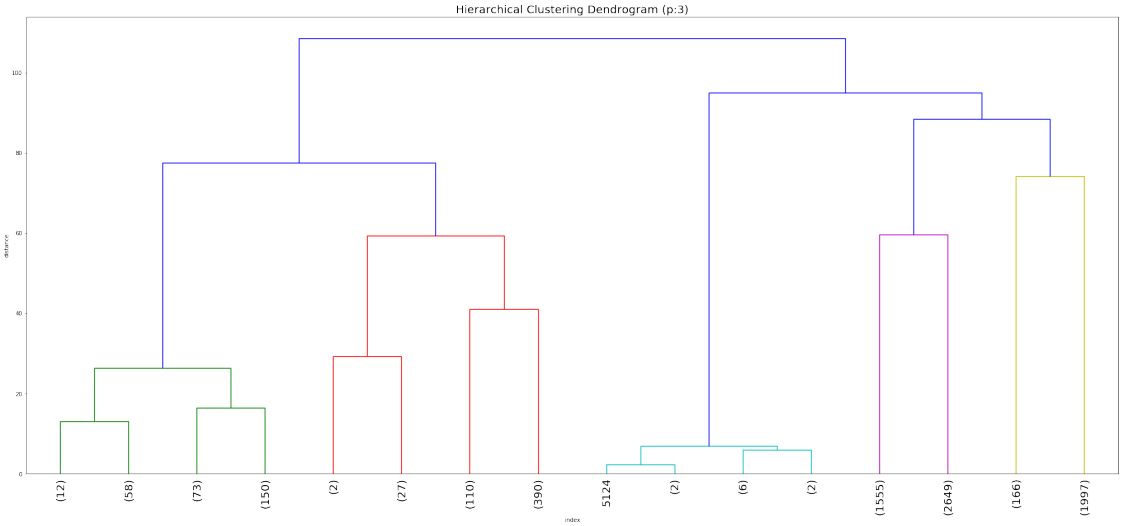


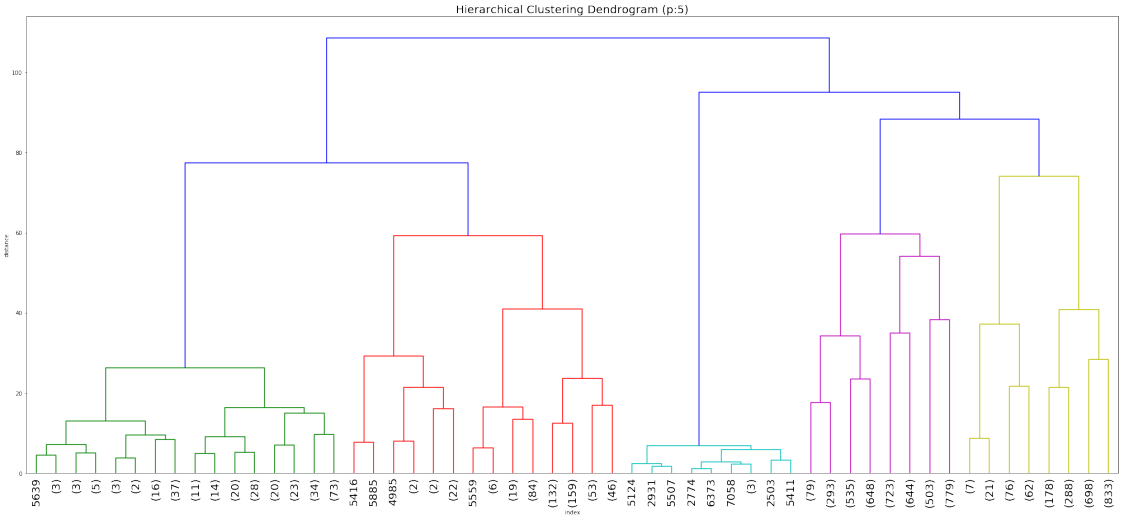
On plotting all combinations of scatter plots with k = 2,3,4,5 clusters, I observed k=2 seems to be a clustering the data in well fashion. So, in KMeans I would choose 2 clusters. Even in the 2 cluster case the supposed clusters are not clearly separated. But it looks the best of the lot lets do PCA on 2 components and run KMeans with k=2 and see the scatter plot

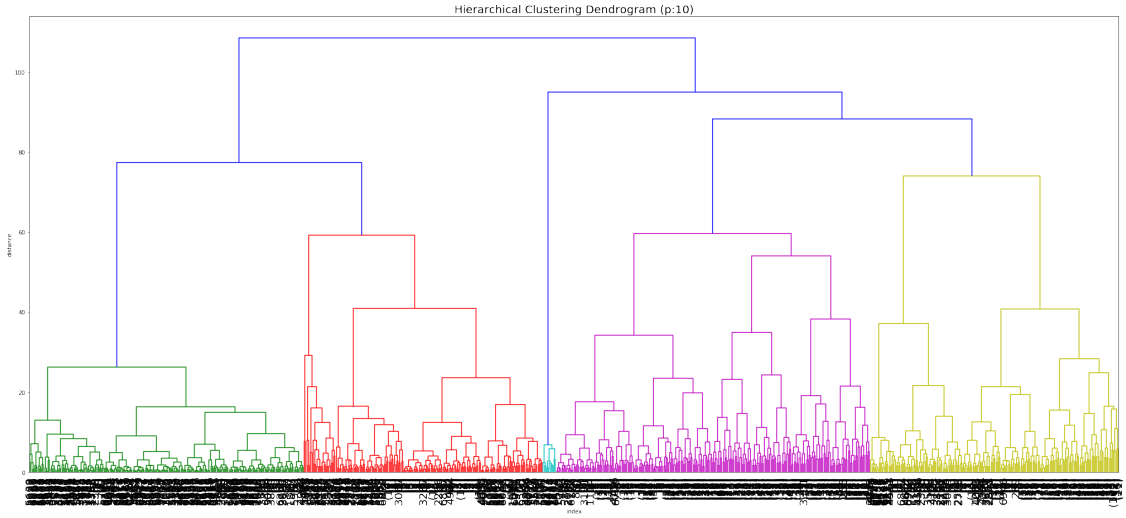
```
In [237]: pca = PCA(n_components=2)
          all_data_scaled_pca = pca.fit_transform(all_data_scaled)
          plt.scatter(all_data_scaled_pca[:, 0], all_data_scaled_pca[:, 1], c=km_dict['km2'].lab
          plt.title('KMeans (k=2) on PCA (n_comp = 2) ')
          plt.xlabel("first principal component")
          plt.ylabel("second principal component")
          plt.show()
```



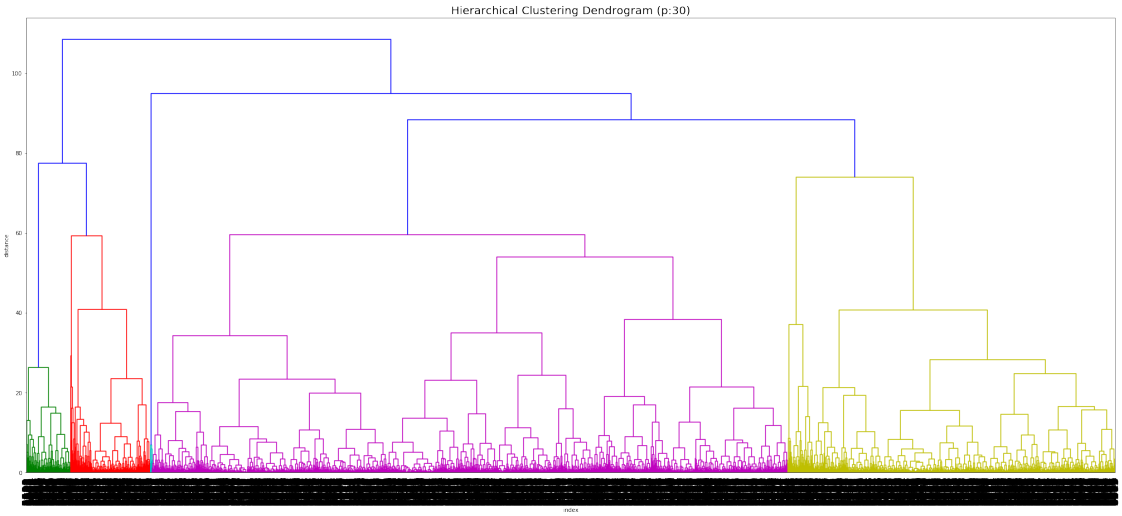

```
In [210]: #agglomerative clustering sklearn package
          ac_dict={}
          clusters = [2,3,4,5,6]
          for i in clusters:
              ac = AgglomerativeClustering(n_clusters = i,linkage='ward')
              ac.fit(all_data_scaled)
              ac_dict['ac'+str(i)]=ac

In [128]: #agglomerative clustering
          z = linkage(all_data_scaled, 'ward')

In [244]: for i in [3,5,10,20,30]:
              plt.figure(figsize=(35, 15))
              plt.title('Hierarchical Clustering Dendrogram (p:'+str(i)+')',fontsize=20)
              plt.xlabel('index')
              plt.ylabel('distance')
              dendrogram(
                  z,
                  truncate_mode='level',
                  p=i,
                  leaf_rotation=90.,
                  leaf_font_size=20.,
              )
              plt.show()
```


Hierarchical Clustering Dendrogram (p:3)
distance
index
0
20
40
60
80
100
(12)
(58)
(73)
(150)
(2)
(27)
(110)
(390)
5124
(2)
(6)
(2)
(1555)
(2649)
(166)
(1997)


Hierarchical Clustering Dendrogram (p:5)
distance
index
0
20
40
60
80
100
5639
(3)
(3)
(5)
(3)
(2)
(16)
(37)
(11)
(14)
(20)
(28)
(20)
(23)
(34)
(73)
5416
5885
4985
(2)
(2)
(22)
5559
(6)
(19)
(84)
(132)
(159)
(53)
(46)
5124
2931
5507
2774
6373
7058
(3)
2503
5411
(79)
(293)
(535)
(648)
(723)
(644)
(503)
(779)
(7)
(21)
(76)
(62)
(178)
(288)
(698)
(833)


Hierarchical Clustering Dendrogram (p:10)
distance
index


Hierarchical Clustering Dendrogram (p:20)
distance
index



I ran the dendogram plots over different p-values ([3,5,10,20,30]) i.e. level. Through observation, I can identify 3 clusters in these plots.

```
In [258]: #DBSCAN
          db_dict = {}
          eps = [1,1.5,2,2.5,3]
          min_samples = [2,3,4,5,6]
          for i in eps:
              for j in min_samples:
                  db = DBSCAN(eps=i, min_samples=j).fit(all_data_scaled)
                  print('No. of Noise Points')
                  print('eps = ',i,' and min_samples = ',j,' ',sum(db.labels_ ==-1))
                  print('No. of non noise and non zero class:')
                  print('eps = ',i,' and min_samples = ',j,' ',sum(db.labels_ !=0)-sum(db.labels
                  print(set(db.labels_))
                  db_dict['db'+str(i)+str(j)]=db
          #_predict
```

```
No. of Noise Points
eps =  1  and min_samples =  2   179
No. of non noise and non zero class:
eps =  1  and min_samples =  2   68
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -
No. of Noise Points
eps =  1  and min_samples =  3   213
No. of non noise and non zero class:
eps =  1  and min_samples =  3   34
{0, 1, 2, 3, 4, 5, 6, 7, 8, -1}
No. of Noise Points
eps =  1  and min_samples =  4   244
```

```
No. of non noise and non zero class:
eps =  1  and min_samples =  4   16
{0, 1, 2, 3, -1}
No. of Noise Points
eps =  1  and min_samples =  5   277
No. of non noise and non zero class:
eps =  1  and min_samples =  5   11
{0, 1, 2, -1}
No. of Noise Points
eps =  1  and min_samples =  6   303
No. of non noise and non zero class:
eps =  1  and min_samples =  6   17
{0, 1, 2, -1}
No. of Noise Points
eps =  1.5  and min_samples =  2   59
No. of non noise and non zero class:
eps =  1.5  and min_samples =  2   32
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, -1}
No. of Noise Points
eps =  1.5  and min_samples =  3   69
No. of non noise and non zero class:
eps =  1.5  and min_samples =  3   22
{0, 1, 2, 3, 4, 5, -1}
No. of Noise Points
eps =  1.5  and min_samples =  4   84
No. of non noise and non zero class:
eps =  1.5  and min_samples =  4   13
{0, 1, 2, -1}
No. of Noise Points
eps =  1.5  and min_samples =  5   95
No. of non noise and non zero class:
eps =  1.5  and min_samples =  5   7
{0, 1, -1}
No. of Noise Points
eps =  1.5  and min_samples =  6   100
No. of non noise and non zero class:
eps =  1.5  and min_samples =  6   7
{0, 1, -1}
No. of Noise Points
eps =  2  and min_samples =  2   23
No. of non noise and non zero class:
eps =  2  and min_samples =  2   18
{0, 1, 2, 3, 4, 5, 6, -1}
No. of Noise Points
eps =  2  and min_samples =  3   29
No. of non noise and non zero class:
eps =  2  and min_samples =  3   12
{0, 1, 2, 3, -1}
```

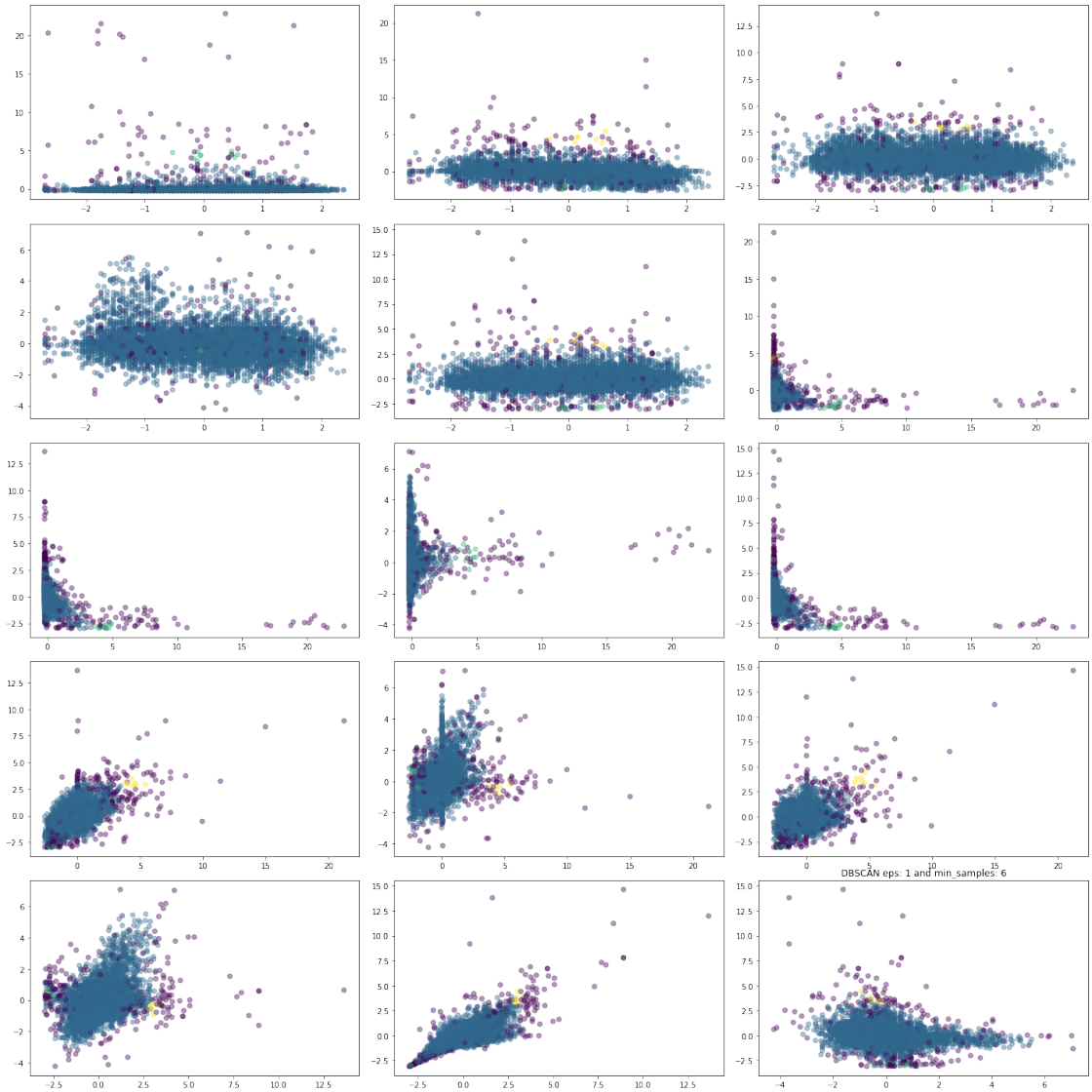
```
No. of Noise Points
eps =  2  and min_samples =  4   36
No. of non noise and non zero class:
eps =  2  and min_samples =  4   6
{0, 1, -1}
No. of Noise Points
eps =  2  and min_samples =  5   36
No. of non noise and non zero class:
eps =  2  and min_samples =  5   6
{0, 1, -1}
No. of Noise Points
eps =  2  and min_samples =  6   41
No. of non noise and non zero class:
eps =  2  and min_samples =  6   6
{0, 1, -1}
No. of Noise Points
eps =  2.5  and min_samples =  2   14
No. of non noise and non zero class:
eps =  2.5  and min_samples =  2   16
{0, 1, 2, 3, 4, -1}
No. of Noise Points
eps =  2.5  and min_samples =  3   18
No. of non noise and non zero class:
eps =  2.5  and min_samples =  3   12
{0, 1, 2, -1}
No. of Noise Points
eps =  2.5  and min_samples =  4   21
No. of non noise and non zero class:
eps =  2.5  and min_samples =  4   9
{0, 1, -1}
No. of Noise Points
eps =  2.5  and min_samples =  5   23
No. of non noise and non zero class:
eps =  2.5  and min_samples =  5   8
{0, 1, -1}
No. of Noise Points
eps =  2.5  and min_samples =  6   24
No. of non noise and non zero class:
eps =  2.5  and min_samples =  6   8
{0, 1, -1}
No. of Noise Points
eps =  3  and min_samples =  2   9
No. of non noise and non zero class:
eps =  3  and min_samples =  2   13
{0, 1, 2, 3, -1}
No. of Noise Points
eps =  3  and min_samples =  3   13
No. of non noise and non zero class:
```

```
eps =  3  and min_samples =  3   9
{0, 1, -1}
No. of Noise Points
eps =  3  and min_samples =  4   14
No. of non noise and non zero class:
eps =  3  and min_samples =  4   9
{0, 1, -1}
No. of Noise Points
eps =  3  and min_samples =  5   14
No. of non noise and non zero class:
eps =  3  and min_samples =  5   9
{0, 1, -1}
No. of Noise Points
eps =  3  and min_samples =  6   14
No. of non noise and non zero class:
eps =  3  and min_samples =  6   9
{0, 1, -1}
```

When I ran with eps = 0.1 and 0.5, the no. of noise points ran into thousands. Because of this I have started eps minimum value as 1

- eps = [1,1.5,2,2.5,3]
- min_samples = [2,3,4,5,6]
- The above mentioned combination of eps and min_value are taken to observe number of noise points. Larger eps seems to produce lower noise points.
- Going through all possible labels manually, DBSCAN results 2 or 3 size clusters (other than noise points).
- On further analysis on number of data points in the clusters and noise sets, eps with 1 and min_samples = 6 seems to be better settings for DBSCAN.

```
In [260]: #DBSCAN with eps=1 and min_samples = 6
          p = 0
          fig,ax = plt.subplots(5,3, figsize=(20,20))
          for i in range(5):
              for j in range(3):
                  ax[i,j].scatter(all_data_scaled[:, comb[p][0]], all_data_scaled[:, comb[p][1]]
                                  c=db_dict['db16'].labels_,alpha = 0.4)
          #         ax[i,j].set_title('perplexity = '+str(perplexity[p]))
                  p+=1
          plt.tight_layout()
          plt.title('DBSCAN eps: 1 and min_samples: 6')
          plt.show()
```



0.4 Task 2.2

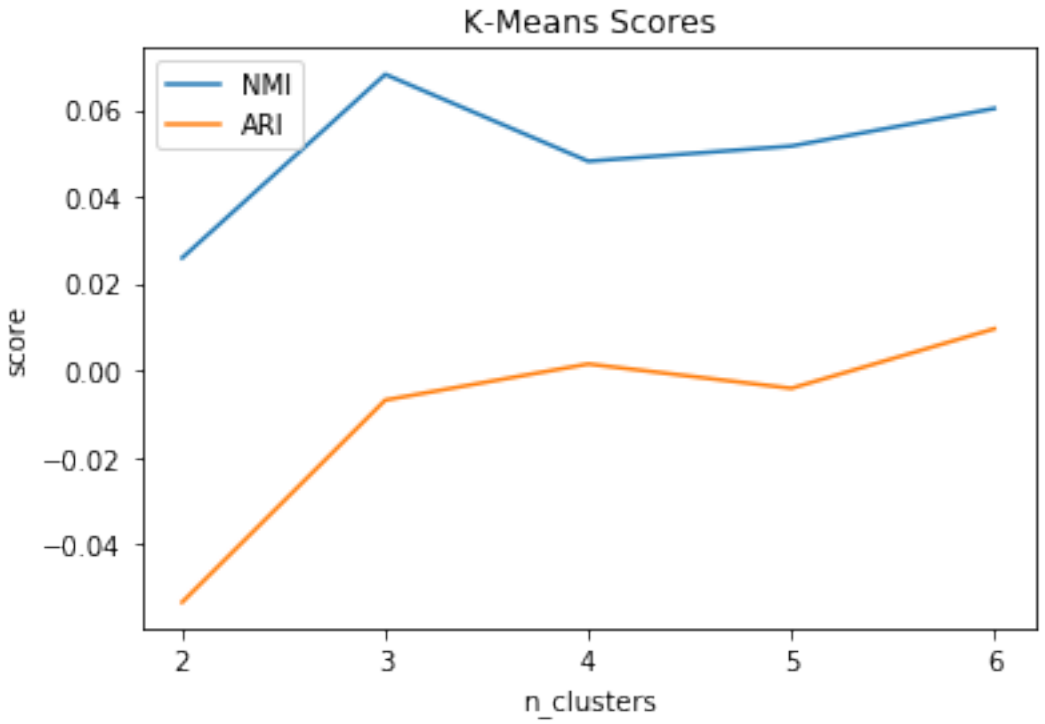
2.2: Use the known ground truth labels of the outlier vs inlier class to evaluate your clustering approaches using NMI and ARI scores. How well did they do? Can you adjust parameters so they can detect the outliers better?

```
In [168]: y_truth.head()
```

```
Out[168]:    label
          0      0
          1      0
          2      0
          3      0
          4      0
```

```
In [195]: #K-Means scoring
          km_nmi = []
          km_ari = []
          for i in km_dict:
              km_nmi.append(nmi(y_truth['label'],km_dict[i].predict(all_data_scaled)))
              km_ari.append(ari(y_truth['label'],km_dict[i].predict(all_data_scaled)))
```

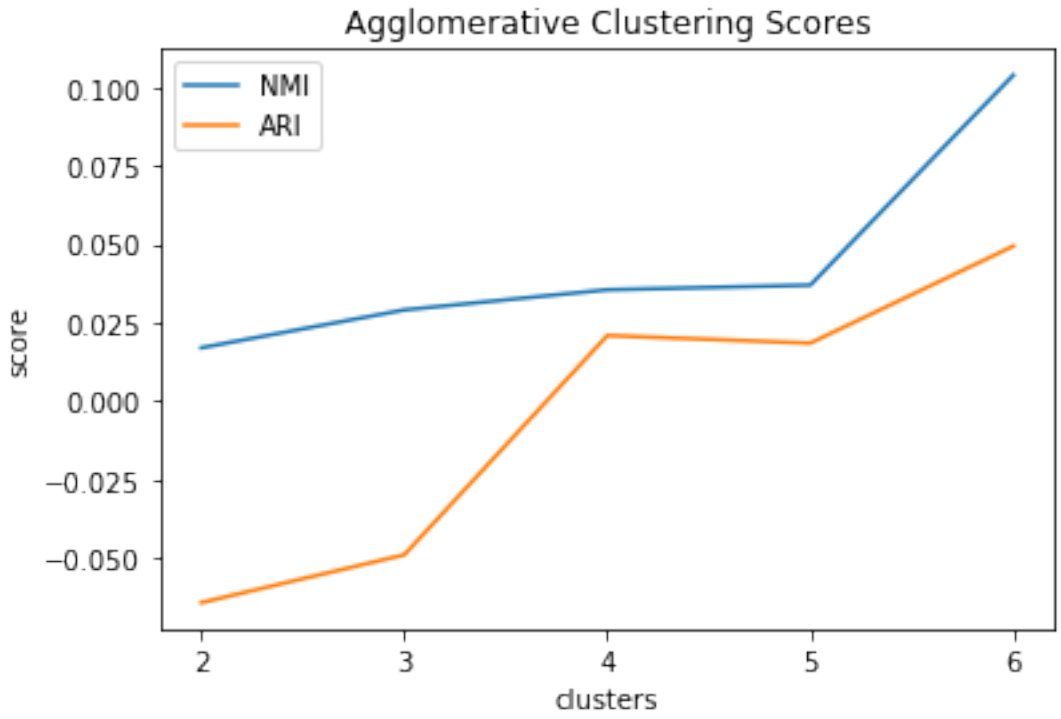
```
In [261]: plot1, = plt.plot(clusters, km_nmi)
          plot2, = plt.plot(clusters,km_ari)
          plt.legend([plot1,plot2],["NMI", "ARI"])
          plt.xlabel("n_clusters")
          plt.ylabel("score")
          plt.title('K-Means Scores')
          plt.xticks(np.arange(2, 7, step=1))
          plt.show()
```



The above plot provides the variation of NMI and ARI for different n_clusters and k=3 seems to be giving optimal score

```
In [213]: #Agglomerative Clustering (with ward linkage)
          ac_nmi = []
          ac_ari = []
          for i in ac_dict:
              ac_nmi.append(nmi(y_truth['label'],ac_dict[i].fit_predict(all_data_scaled)))
              ac_ari.append(ari(y_truth['label'],ac_dict[i].fit_predict(all_data_scaled)))
```

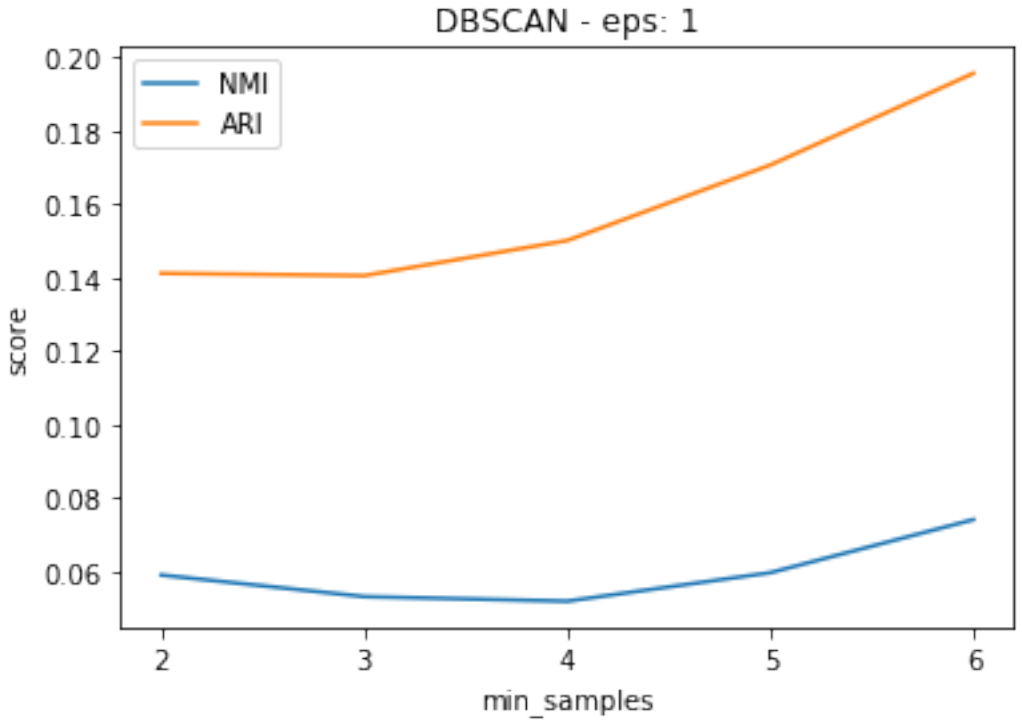
```
In [215]: plot1, = plt.plot(clusters, ac_nmi)
          plot2, = plt.plot(clusters, ac_ari)
          plt.legend([plot1,plot2],["NMI", "ARI"])
          plt.xlabel("clusters")
          plt.ylabel("score")
          plt.title('Agglomerative Clustering Scores')
          plt.xticks(np.arange(2, 7, step=1))
          plt.show()
```

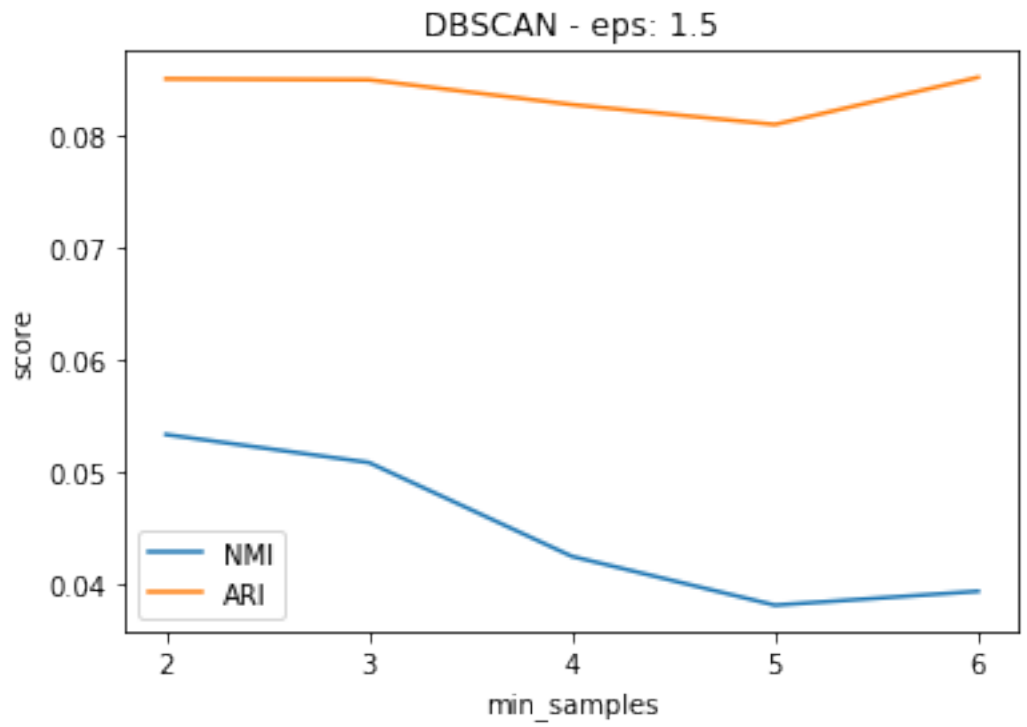


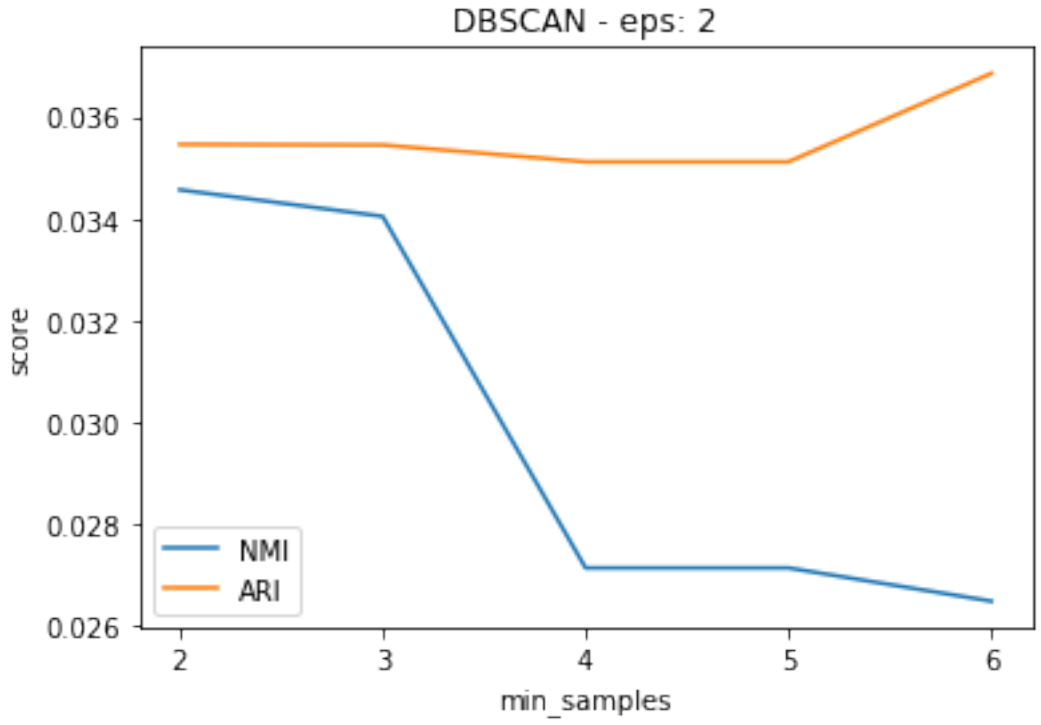
clusters = 6 giving higher NMI and ARI scores for agglomerative clustering

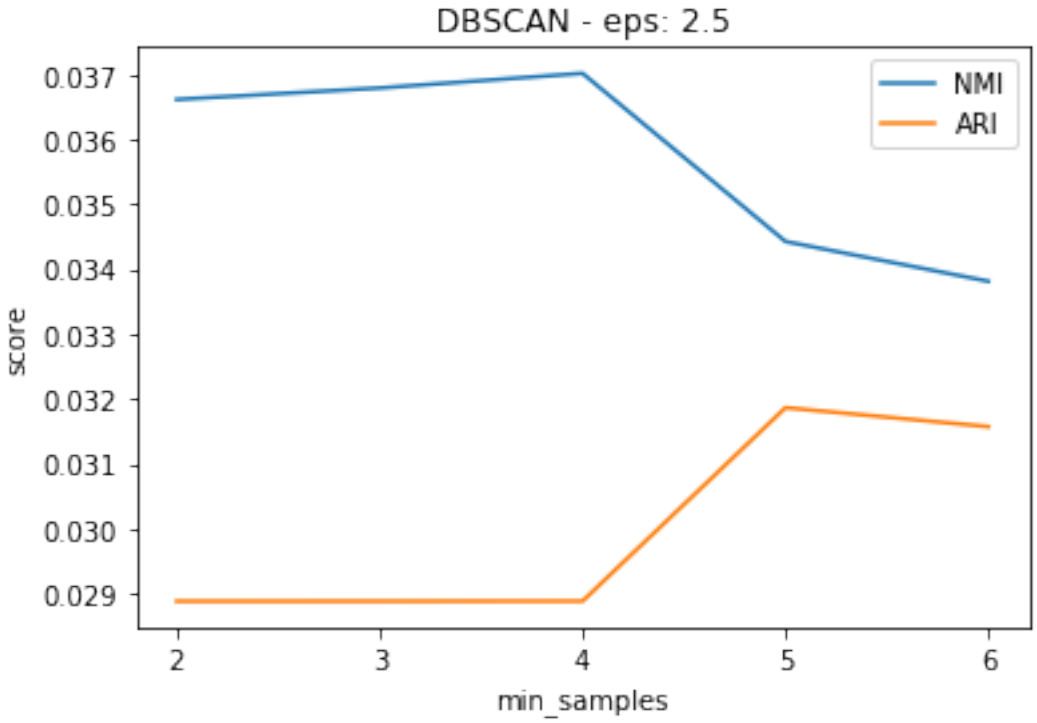
```
In [216]: #Agglomerative Clustering (with ward linkage)
          db_nmi = []
          db_ari = []
          for i in db_dict:
              db_nmi.append(nmi(y_truth['label'],db_dict[i].fit_predict(all_data_scaled)))
              db_ari.append(ari(y_truth['label'],db_dict[i].fit_predict(all_data_scaled)))

In [222]: eps = [1,1.5,2,2.5,3]
          min_samples = [2,3,4,5,6]
          a = 0
          for i in eps:
              plot1, = plt.plot(min_samples, db_nmi[a:a+5])
              plot2, = plt.plot(min_samples, db_ari[a:a+5])
```

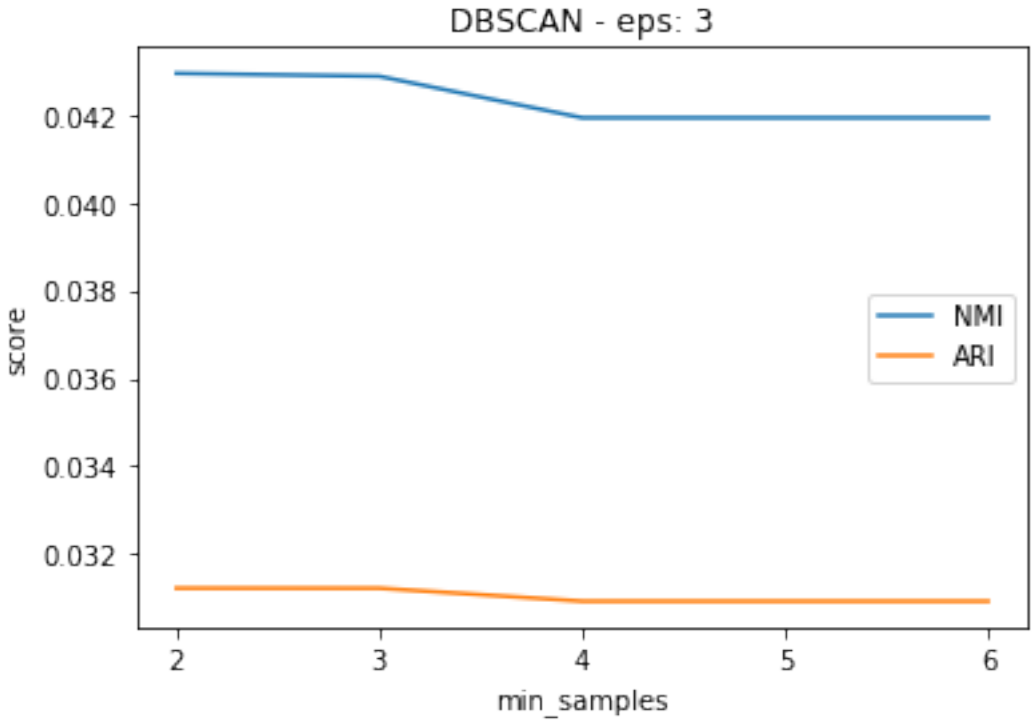
```
plt.legend([plot1,plot2],["NMI", "ARI"])
plt.xlabel("min_samples")
plt.ylabel("score")
plt.title('DBSCAN - eps: '+str(i))
plt.xticks(np.arange(min(min_samples), max(min_samples)+1, step=1))
plt.show()
a+=5
```




DBSCAN - eps: 1.5
0.08
0.07
0.06
0.05
0.04
score
NMI
ARI
2
3
4
5
6
min_samples


DBSCAN - eps: 2
0.036
0.034
0.032
0.030
0.028
0.026
score
NMI
ARI
2
3
4
5
6
min_samples


DBSCAN - eps: 2.5
NMI
ARI
score
0.037
0.036
0.035
0.034
0.033
0.032
0.031
0.030
0.029
2
3
4
5
6
min_samples


DBSCAN - eps: 3
NMI
ARI
score
0.042
0.040
0.038
0.036
0.034
0.032
2
3
4
5
6
min_samples

For different combinations, highest NMI and ARI scoreset combination is for eps = 1 and min_samples = 6 which is what we predicted earlier (task 2.1)

In [ ]:

aml_hw4_task3_4

April 4, 2018

```
In [167]: import scipy.io as sio
          import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          % matplotlib inline
          #plt.rcParams["figure.dpi"] = 300
          np.set_printoptions(precision=3, suppress=True)
```

1 Task 1

```
In [12]: data = sio.loadmat("annthyroid.mat")
```

```
In [13]: X_data = pd.DataFrame(data["X"])
         y_data = pd.DataFrame(data["y"])
```

```
In [14]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         X_sc = scaler.fit_transform(X_data)
```

```
In [15]: y_sc = np.array(y_data)
```

```
In [16]: y_sc = y_sc.reshape(7200,)
```

```
In [19]: y_sc.shape
```

```
Out[19]: (7200,)
```

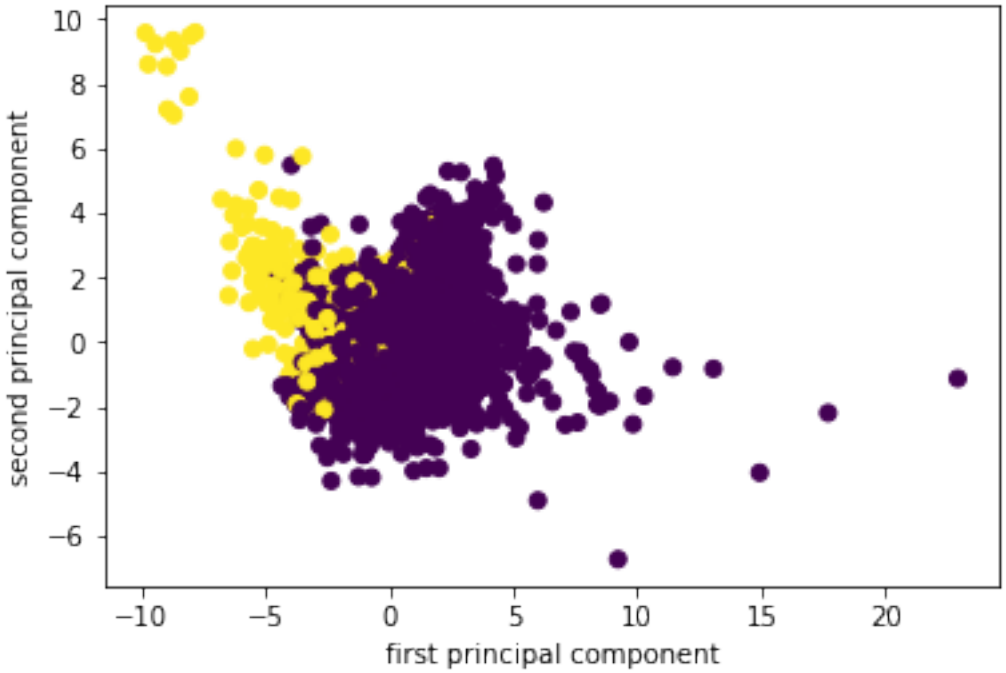
```
In [161]: from sklearn.decomposition import PCA
          import matplotlib.pyplot as plt
          %matplotlib
          #print(cancer.data.shape)
          pca = PCA(n_components=2)
          X_pca = pca.fit_transform(X_sc)
```

```
Using matplotlib backend: TkAgg
```

```
In [8]: print(X_pca.shape)
        print(y_data.shape)
```

```
(7200, 2)
(7200, 1)
```

```
In [9]: plt.scatter(X_pca[:, 0], X_pca[:, 1], c = y_sc)
        plt.xlabel("first principal component")
        plt.ylabel("second principal component")
        components = pca.components_
        #plt.imshow(components.T)
        #plt.yticks(range(len(y_data.columns)), y_data.columns)
        #plt.colorbar()
```



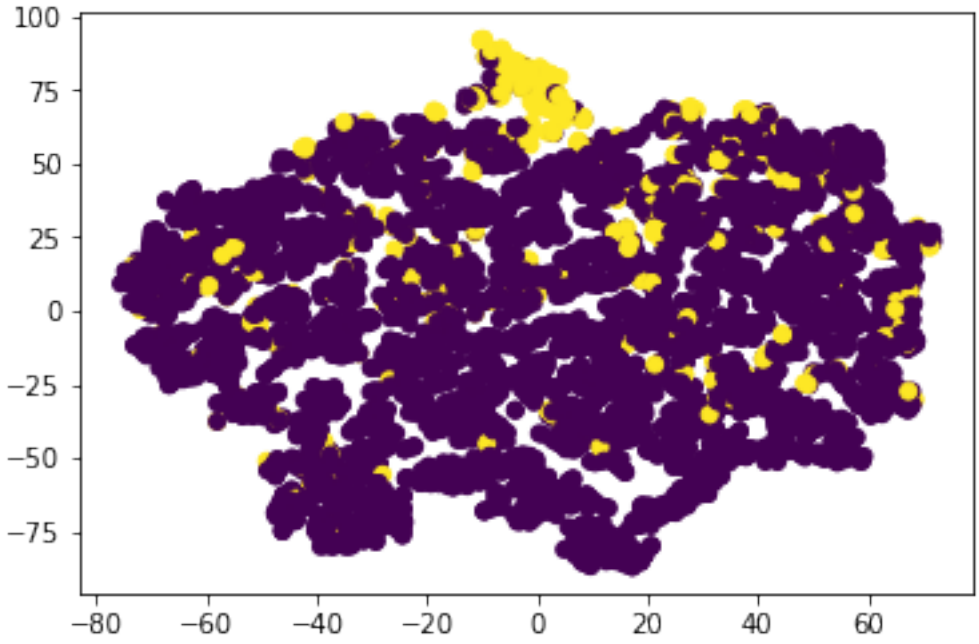

```
In [207]: from sklearn.manifold import TSNE
          #from sklearn.datasets import load_digits
          #digits = load_digits()
          #X = digits.data / 16.
          X_tsne = TSNE().fit_transform(X_sc)
          #X_pca = PCA(n_components=2).fit_transform(X_sc)

In [208]: X_tsne

Out[208]: array([[ 60.845, -18.318],
                 [-39.394, -51.906],
                 [  4.52 , -37.725],
                 ...,
```

```
                [  0.374,  30.941],
                [-59.743,  24.545],
                [ 16.716,  49.189]], dtype=float32)
```

In [ ]:

In [209]:
```
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c = y_sc)
#plt.xlabel("first principal component")
#plt.ylabel("second principal component")
```

Out[209]: <matplotlib.collections.PathCollection at 0x7f7592fc0c18>



In [27]:
```
X_sc
```

Out[27]:
```
array([[ 1.107, -0.185, -0.66 ,  0.294, -0.83 ,  0.908],
       [-1.482, -0.201,  1.33 ,  0.933,  1.843, -0.144],
       [-0.267, -0.129,  0.534, -0.207,  1.738, -0.976],
       ...,
       [-0.056, -0.178,  0.016, -0.54 , -1.616,  0.576],
       [-0.901, -0.09 ,  0.016, -0.54 , -0.463, -0.338],
       [ 1.107, -0.187,  0.016, -0.79 , -0.411, -0.643]])
```

In [ ]:

In [ ]:

2 Task 3 Outlier Detection

2.0.1 3.1

```
In [37]: #Since we can assume that we know the proportion of outliers in the data
         print("Actual proportion of outliers in the data: ", np.mean(y_sc))
```

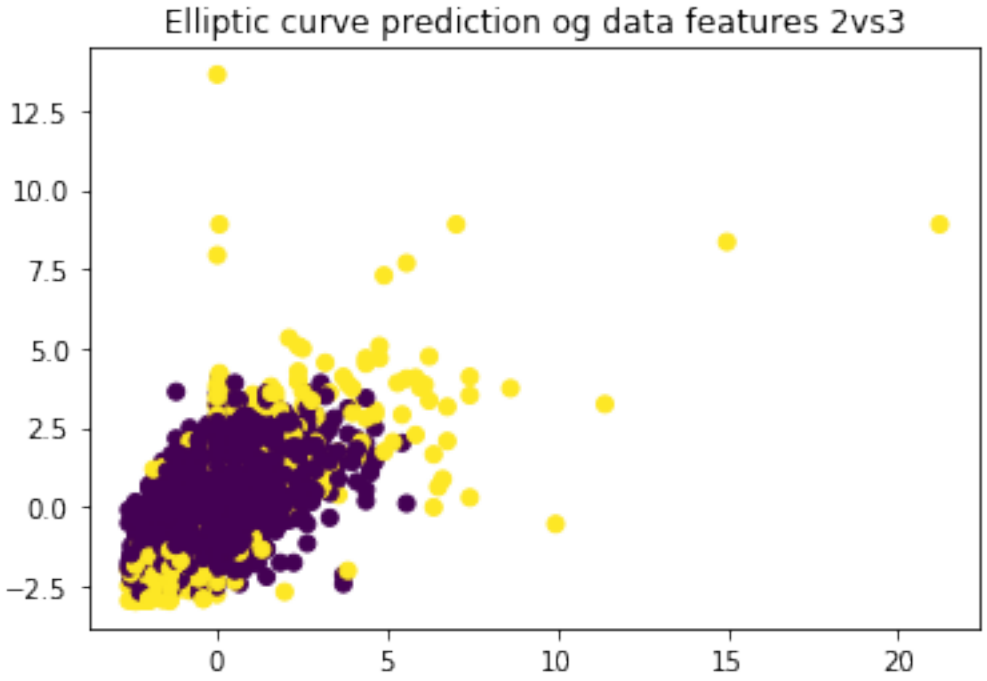
```
Actual proportion of outliers in the data:  0.07416666666666667
```

```
In [158]: from sklearn.covariance import EllipticEnvelope
          ee = EllipticEnvelope(contamination=0.07416666666666667).fit(X_sc)
          pred_ee = ee.predict(X_sc)
          print(pred_ee)
          print("Predicted proportion of outliers in the data using Elliptic Envelope: ", np.mea
```

```
[1 1 1 ... 1 1 1]
Predicted proportion of outliers in the data using Elliptic Envelope:  0.07416666666666667
```

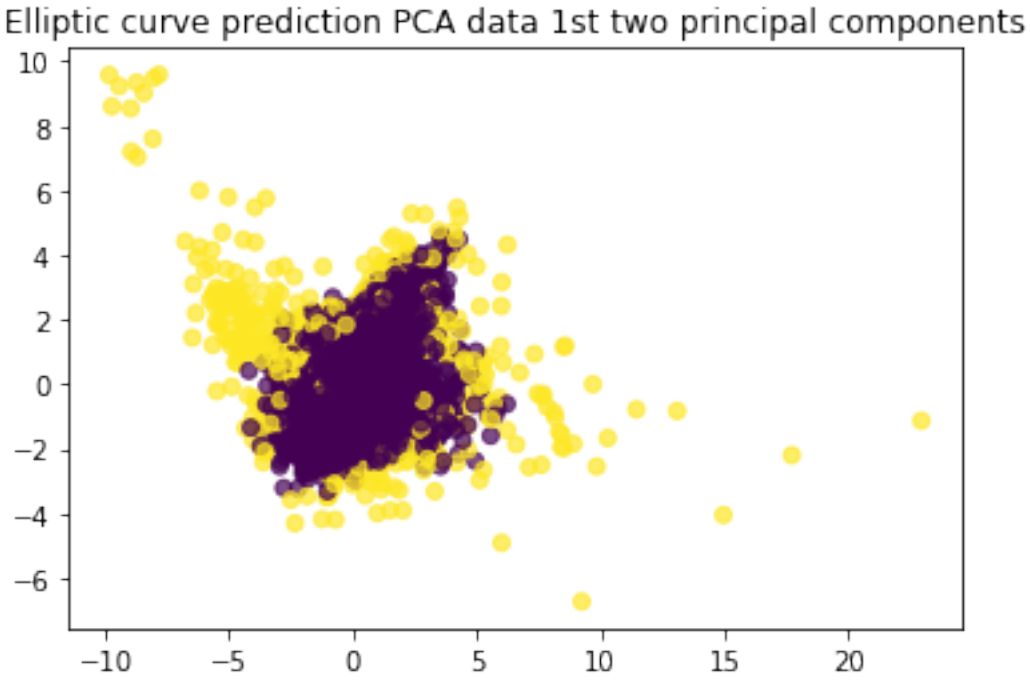
```
In [211]: plt.title("Elliptic curve prediction og data features 2vs3")
          plt.scatter(X_sc[:, 2], X_sc[:, 3], c=pred_ee)
```

```
Out[211]: <matplotlib.collections.PathCollection at 0x7f7592f5bf98>
```

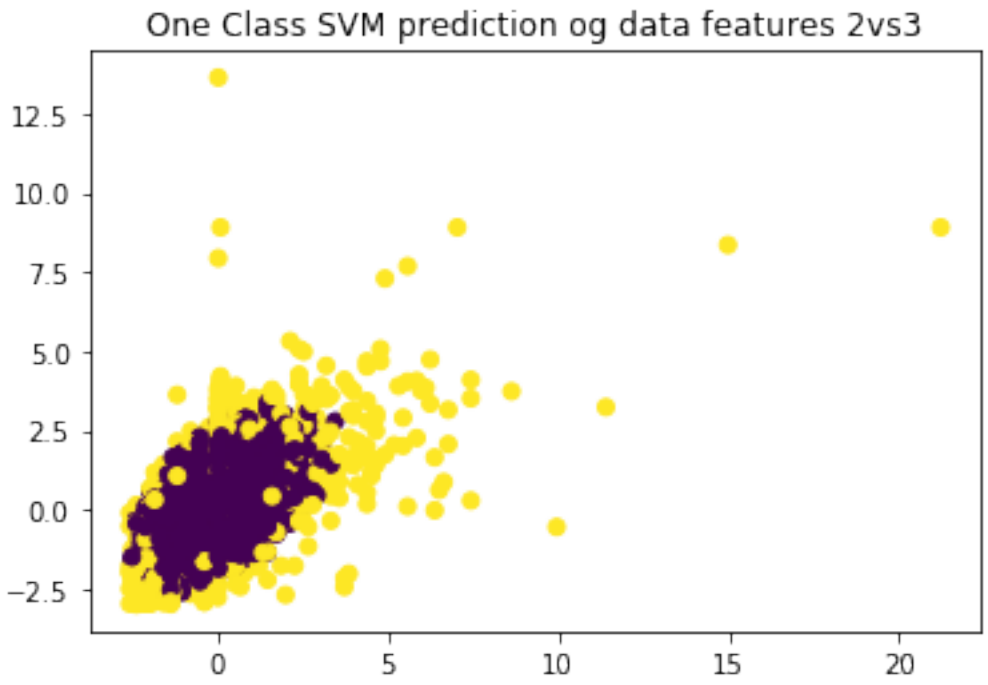



```
In [212]: plt.title("Elliptic curve prediction PCA data 1st two principal components")
          plt.scatter(X_pca[:, 0], X_pca[:, 1], c=pred_ee, alpha = 0.7)
```

```
Out[212]: <matplotlib.collections.PathCollection at 0x7f7592efaac8>
```

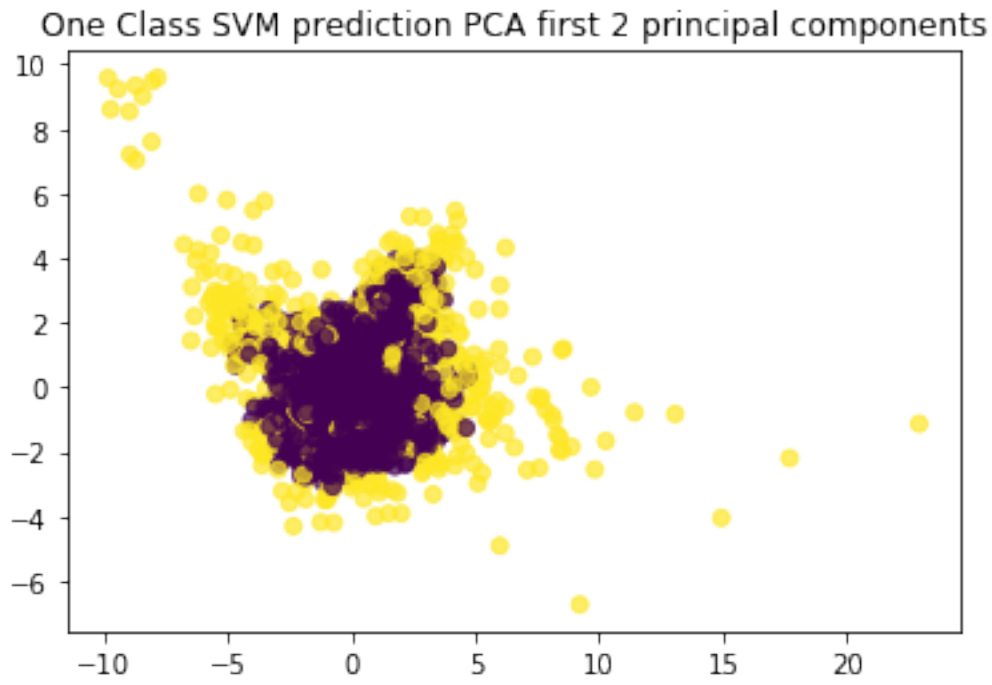



```
In [68]: from sklearn.svm import OneClassSVM
         oneclass = OneClassSVM(nu=0.07416666666666667).fit(X_sc)
         pred_oc = oneclass.predict(X_sc)

In [50]: np.mean(pred_oc == -1)

Out[50]: 0.07458333333333333

In [213]: plt.title("One Class SVM prediction og data features 2vs3")
          plt.scatter(X_sc[:, 2], X_sc[:, 3], c=pred_oc)

Out[213]: <matplotlib.collections.PathCollection at 0x7f7592efafd0>
```

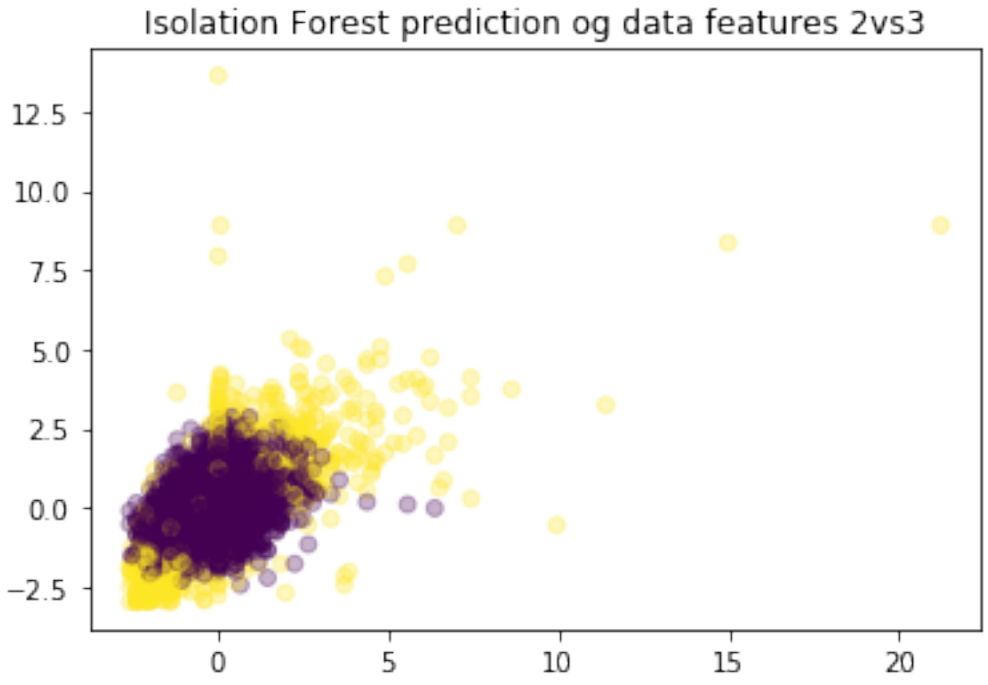



```
In [217]: plt.title("One Class SVM prediction PCA first 2 principal components")
          plt.scatter(X_pca[:, 0], X_pca[:, 1], c=pred_oc, alpha =0.7)
```

```
Out[217]: <matplotlib.collections.PathCollection at 0x7f7592e052b0>
```

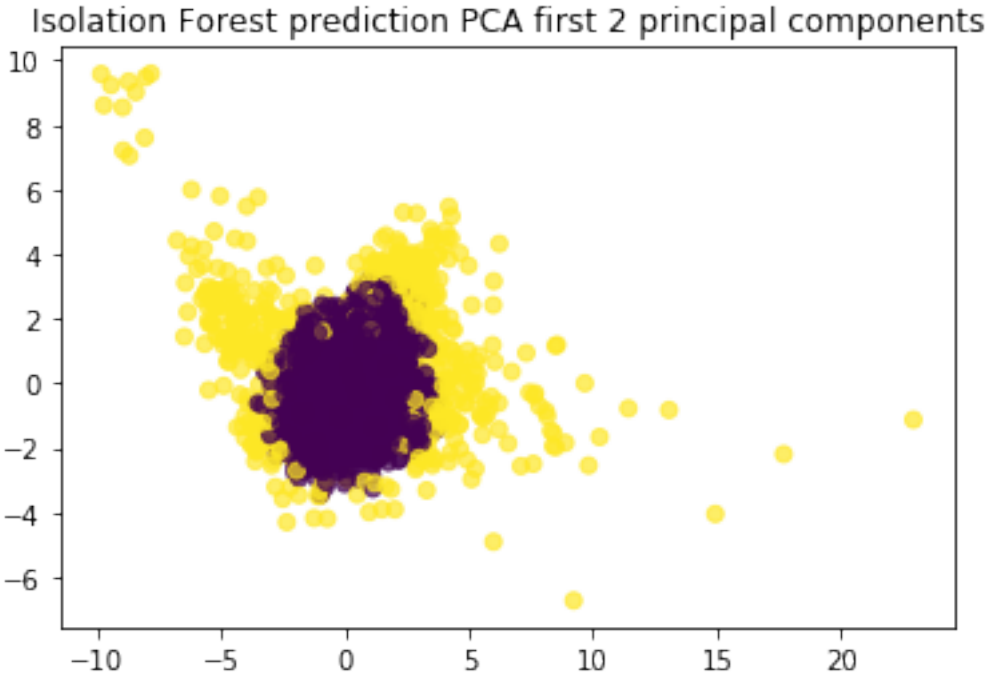


```
In [52]: from sklearn.ensemble import IsolationForest

In [110]: ifr = IsolationForest(contamination=0.07416666666666667)

In [111]: ifr.fit(X_sc)

Out[111]: IsolationForest(bootstrap=False, contamination=0.07416666666666667,
                  max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=1,
                  random_state=None, verbose=0)

In [112]: pred_if = ifr.predict(X_sc)

In [113]: np.mean(pred_if == -1)

Out[113]: 0.07416666666666667

In [216]: plt.title("Isolation Forest prediction og data features 2vs3")
          plt.scatter(X_sc[:, 2], X_sc[:, 3], c=pred_if, alpha = 0.3)

Out[216]: <matplotlib.collections.PathCollection at 0x7f7592e78b38>
```




```
In [218]: plt.title("Isolation Forest prediction PCA first 2 principal components")

          plt.scatter(X_pca[:, 0], X_pca[:, 1], c=pred_if, alpha = 0.7)
```

```
Out[218]: <matplotlib.collections.PathCollection at 0x7f7592e05cc0>
```



2.0.2 Without using the ground truth we cannot really gauge how good an outlier detection algorithm works as the visualizing the predictions of all three models on X_pca leads to similar looking plots. Not much can be inferred from them

```
In [219]: ee.decision_function(X_sc)

Out[219]: array([2.553, 2.419, 1.104, ..., 2.056, 3.501, 3.143])

In [220]: oneclass.decision_function(X_sc)

Out[220]: array([[ 9.077],
                 [ 5.277],
                 [ 7.151],
                 ...,
                 [ 9.115],
                 [11.846],
                 [ 8.598]])

In [221]: ifr.decision_function(X_sc)

Out[221]: array([0.113, 0.053, 0.093, ..., 0.103, 0.137, 0.132])
```

As we can see only the Isolation Forest decision function provides output values between 0 and 1

2.0.3 3.2

```
In [65]: from sklearn.metrics import average_precision_score
```

```
In [ ]: for i in range(len(y_sc)):
            #print(y_sc[i])

        for i in range(len(y_sc)):
            if(pred_ee[i] == 1):
                pred_ee[i] = 0

            else:
                pred_ee[i] = 1
            print(pred_ee[i])

        for i in range(len(y_sc)):
            if(pred_oc[i] == 1):
                pred_oc[i] = 0

            else:
                pred_oc[i] = 1
            print(pred_oc[i])

        for i in range(len(y_sc)):
            if(pred_if[i] == 1):
                pred_if[i] = 0

            else:
                pred_if[i] = 1
            print(pred_if[i])
```

```
In [230]: average_precision_ee = average_precision_score(y_sc, pred_ee)
          average_precision_oc = average_precision_score(y_sc, pred_oc)
          average_precision_if = average_precision_score(y_sc, pred_if)


          print('Elliptic Envelope Average precision-recall score: {0:0.2f}'.format(
                average_precision_ee))

          print('One Class SVM Average precision-recall score: {0:0.2f}'.format(
                average_precision_oc))

          print('Isolation Forest Average precision-recall score: {0:0.2f}'.format(
                average_precision_if))
```

```
Elliptic Envelope Average precision-recall score: 0.25
One Class SVM Average precision-recall score: 0.11
Isolation Forest Average precision-recall score: 0.15
```

```
In [121]: average_precision_ee = average_precision_score(y_sc, pred_ee)
          average_precision_oc = average_precision_score(y_sc, pred_oc)
          average_precision_if = average_precision_score(y_sc, pred_if)


          print('Elliptic Envelope Average precision-recall score: {0:0.2f}'.format(
                average_precision_ee))

          print('One Class SVM Average precision-recall score: {0:0.2f}'.format(
                average_precision_oc))

          print('Isolation Forest Average precision-recall score: {0:0.2f}'.format(
                average_precision_if))
```

```
Elliptic Envelope Average precision-recall score: 0.25
One Class SVM Average precision-recall score: 0.11
Isolation Forest Average precision-recall score: 0.15
```

```
In [122]: from sklearn.metrics import roc_auc_score
          from sklearn.metrics import zero_one_loss
          ee_auc = roc_auc_score(y_sc, pred_ee)
          oc_auc = roc_auc_score(y_sc, pred_oc)
          if_auc = roc_auc_score(y_sc, pred_if)

          ee_loss = zero_one_loss(y_sc, pred_ee)
          oc_loss = zero_one_loss(y_sc, pred_oc)
          if_loss = zero_one_loss(y_sc, pred_if)

          print("AUC for Elliptic Envelope is ", ee_auc)
          print("AUC for One Class SVM: ", oc_auc)
          print("AUC for Isolated Forest: ", if_auc)


          print("Zero-one loss for Elliptic Envelope is ", ee_loss)
          print("Zero-one loss for One Class SVM: ", oc_loss)
          print("Zero-one loss for Isolated Forest: ", if_loss)
```

```
AUC for Elliptic Envelope is  0.7077235813468987
AUC for One Class SVM:  0.5841154340153115
AUC for Isolated Forest:  0.6288392884232243
Zero-one loss for Elliptic Envelope is  0.08027777777777778
Zero-one loss for One Class SVM:  0.11458333333333337
Zero-one loss for Isolated Forest:  0.1019444444444444
```

2.0.4 The AUC and average precision readings suggest that EllipticEnvelope is the best

The clustering approaches from task 2 yield poor results with in some cases 3 clusters giving the best results. Thus, the approaches in Task 3 which assume the additional information that you know the number of classes and their proportion gives better results

However comaparing the visualzations in Task 3 with the ground truth clustering of the PCA visualization in Task 1 we notice that even Task 3 is quite off. Therefore there is no tangible way of comparing one model from another without knowing the ground truth and just via visualization. This could be the result of the data being highly non Gaussian

3 Task 4

```
In [191]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X_sc, y_sc, stratify = y_sc)
```

Imbalanced Learning

```
In [195]: from sklearn.model_selection import cross_validate
          from sklearn.linear_model import LogisticRegression
          scores = cross_validate(LogisticRegression(),
                                  X_sc, y_sc, cv=10, scoring=('roc_auc', 'average_precision'))
          scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()

Out[195]: (0.9726447870177892, 0.7637355829525652)

In [196]: from sklearn.ensemble import RandomForestClassifier
          scores = cross_validate(RandomForestClassifier(n_estimators=100),
                                  X_sc, y_sc, cv=10, scoring=('roc_auc', 'average_precision'))
          scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()

Out[196]: (0.9957853471700322, 0.9321881275718397)
```

Balanced weights

```
In [197]: scores = cross_validate(LogisticRegression(class_weight='balanced'),
                                  X_sc, y_sc, cv=10, scoring=('roc_auc', 'average_precision'))
          print("Test ROC AUC for Logistic :", scores['test_roc_auc'].mean(),"| Average Precisio

Test ROC AUC for Logistic : 0.9884794372602356 | Average Precision for Logistic:  0.828698915304


In [198]: scores = cross_validate(RandomForestClassifier(n_estimators=100, class_weight='balance
                                  X_sc, y_sc, cv=10, scoring=('roc_auc', 'average_precision'))
          print("Test ROC AUC for Random Forest :", scores['test_roc_auc'].mean(),"| Average Pre

Test ROC AUC for Random Forest : 0.9957258147797312 | Average Precision for Random Forest:  0.92
```

Grid Searching C for LogReg and other pruning parameters for the trees

```
In [133]: from sklearn.model_selection import GridSearchCV

In [134]: param_grid_rf = {

              'max_depth': [80, 90, 100, 110],
              'max_features': [2, 3],

              'n_estimators': [100, 200, 300, 1000]
          }

In [135]: rf_grid = GridSearchCV(RandomForestClassifier(), param_grid=param_grid_rf, cv=3)

In [136]: rf_grid.fit(X_train,y_train)

Out[136]: GridSearchCV(cv=3, error_score='raise',
                 estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                      oob_score=False, random_state=None, verbose=0,
                      warm_start=False),
                 fit_params=None, iid=True, n_jobs=1,
                 param_grid={'max_depth': [80, 90, 100, 110], 'max_features': [2, 3], 'n_estimat
                 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                 scoring=None, verbose=0)

In [137]: rf_grid.score(X_test,y_test)

Out[137]: 0.9833333333333333

In [143]: print("AUC for RF is ", roc_auc_score(y_test, rf_grid.predict(X_test)))
          print("Avg Precision for RF : ", average_precision_score(y_test, rf_grid.predict(X_tes

AUC for RF is  0.9737045072188569
Avg Precision for RF :  0.8079279080053073


In [199]: param_grid_lr = {

              'C': [ 0.1, 1,2,5 ,10,100]
          }

In [200]: lr_grid = GridSearchCV(LogisticRegression(), param_grid=param_grid_lr, cv=10)

In [201]: lr_grid.fit(X_train,y_train)
```

```
Out[201]: GridSearchCV(cv=10, error_score='raise',
                  estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercep
                     intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                     penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                     verbose=0, warm_start=False),
                  fit_params=None, iid=True, n_jobs=1,
                  param_grid={'C': [0.1, 1, 2, 5, 10, 100]}, pre_dispatch='2*n_jobs',
                  refit=True, return_train_score='warn', scoring=None, verbose=0)

In [202]: lr_grid.score(X_test,y_test)

Out[202]: 0.9533333333333334

In [206]: print("AUC for LogReg is ", roc_auc_score(y_test, lr_grid.predict(X_test)))
          print("Avg Precision for LogReg : ", average_precision_score(y_test, lr_grid.predict(X

AUC for LogReg is  0.7208793965347333
Avg Precision for LogReg :  0.4249064202795546
```

3.0.1 Changing to balanced weights gives the best avg precision and recall scores. So yes. It helps

```
In [ ]:
```