ComS 574 HW6 Kanak Choudhury.

Problem 1: Ch 10, #2:

We have, $d(x,y) = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0{\cdot}3 & 0{\cdot}4 & 0{\cdot}7 \\ 2 & 0{\cdot}3 & 0 & 0{\cdot}5 & 0{\cdot}8 \\ 3 & 0{\cdot}4 & 0{\cdot}5 & 0 & 0{\cdot}45 \\ 4 & 0{\cdot}7 & 0{\cdot}8 & 0{\cdot}45 & 0 \end{array}$

a)

$\min d(i,j) = d(1,2) = 0{\cdot}3 \Rightarrow 1 \Leftrightarrow 2$

$d((1,2),3) = \max(D(1,3), D(2,3)) = \max(0{\cdot}4, 0{\cdot}5) = 0{\cdot}5$

$d((1,2),4) = \max(0{\cdot}7, 0{\cdot}8) = 0{\cdot}8$

$d_1(i,j) = \begin{array}{c|ccc} & (1,2) & & \\ \hline (1,2) & 0 & 0{\cdot}5 & 0{\cdot}8 \\ 3 & 0{\cdot}5 & 0 & 0{\cdot}45 \\ 4 & 0{\cdot}8 & 0{\cdot}45 & 0 \end{array}$

$\min d(i,j) = 0{\cdot}45 = d(3,4)$

$d((1,2),(3,4)) = \max(0{\cdot}5, 0{\cdot}8) = 0{\cdot}8$

$d_2(i,j) = \begin{array}{c|cc} & (1,2) & (3,4) \\ \hline (1,2) & 0 & 0{\cdot}8 \\ (3,4) & 0{\cdot}8 & 0 \end{array}$

b) $\min d(i,j) = d(1,2) = 0.3$ ⇒ 1 is connected with 2

$d((1,2),3) = \cancel{d}\ \min(d(1,3), d(2,3)) = \min(0.4, 0.5)$
$= 0.4$

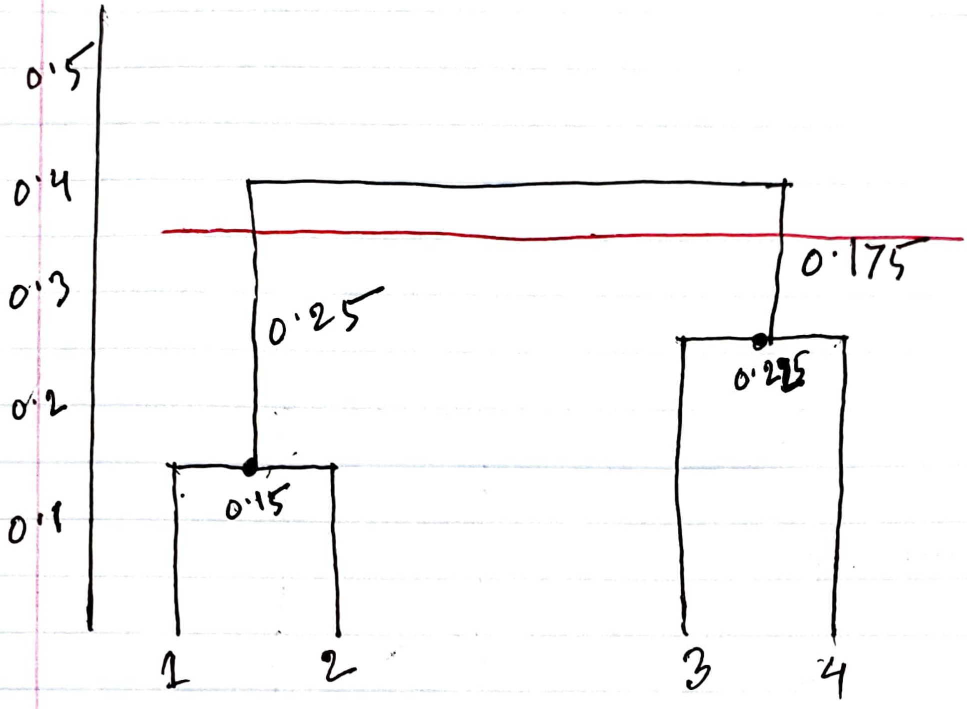
$d((1,2),4) = \min(0.7, 0.8) = 0.7$

$$d_1(i,j) = \begin{array}{c|ccc} & (1,2) & 3 & 4 \\ \hline (1,2) & 0 & 0.4 & 0.7 \\ 3 & 0.4 & 0 & 0.45 \\ 4 & 0.7 & 0.45 & 0 \end{array}$$

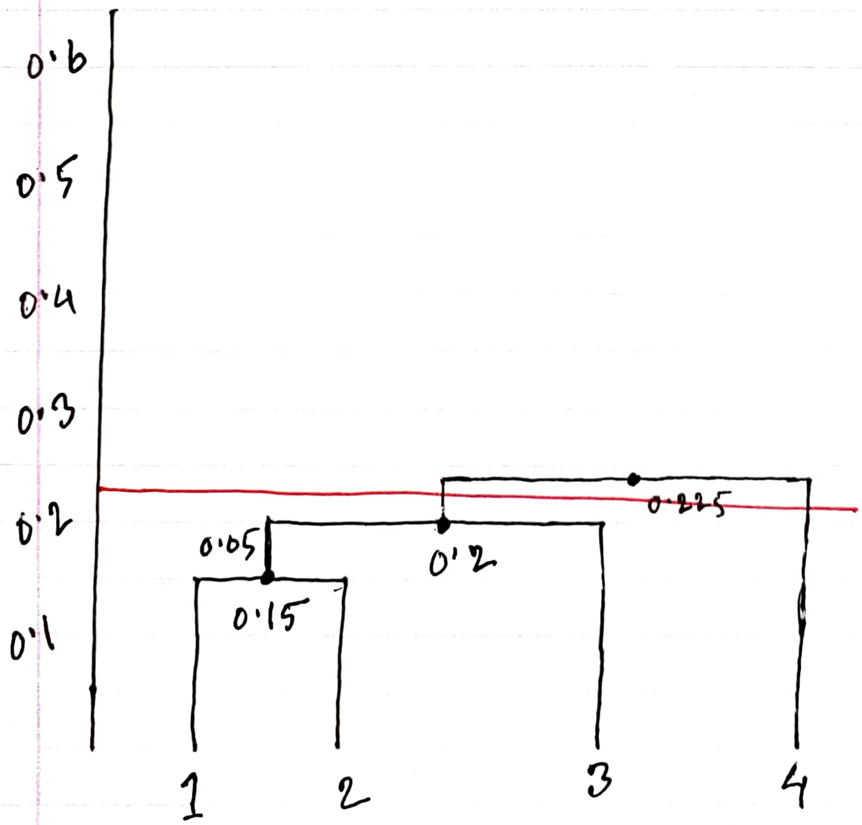
$\min d(i,j) = d((1,2),3) = 0.4$ ⇒ $(1,2) \Leftrightarrow 3$

$d(((1,2),3),4) = \min(d((1,2),4), d(3,4))$
$= \min(0.7, 0.45) = 0.45$

$$d_2(i,j) = \begin{array}{c|cc} ((1,2),3) & 0 & 0.45 \\ 4 & 0.45 & 0 \end{array}$$



Plot: Complete Linkage

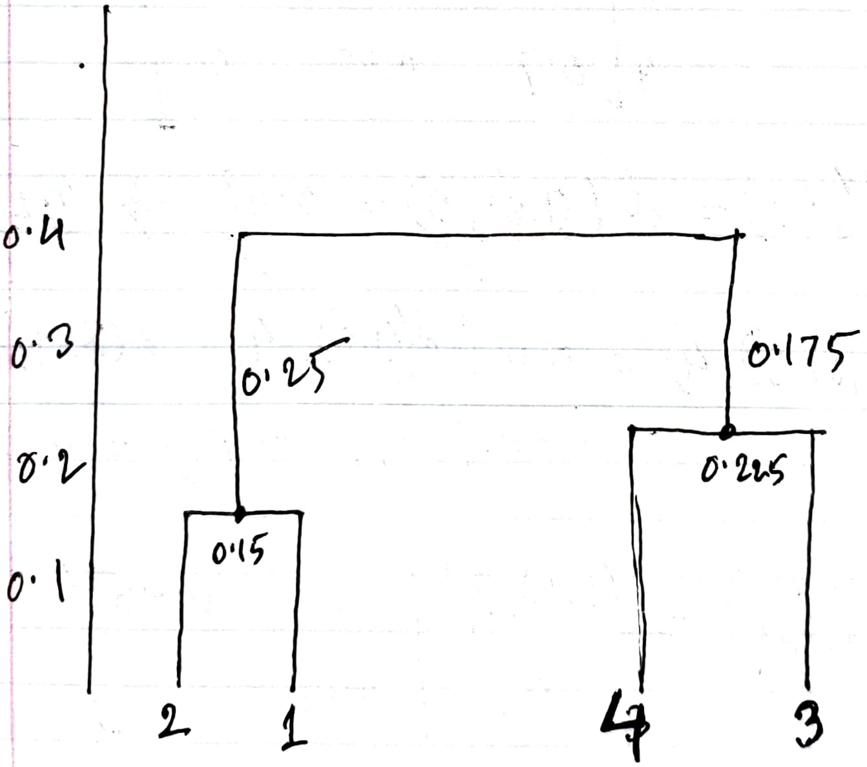


Plot: Single Linkage.

(c) cluster 1 : 1, 2
cluster 2 : 3, 4

d) Cluster 1 : 1, 2, 3
Cluster 2 : 4

e)



ComS574 HW6

April 19, 2020

ComS 574

HW 6

Kanak Choudhury

1 Problem 2

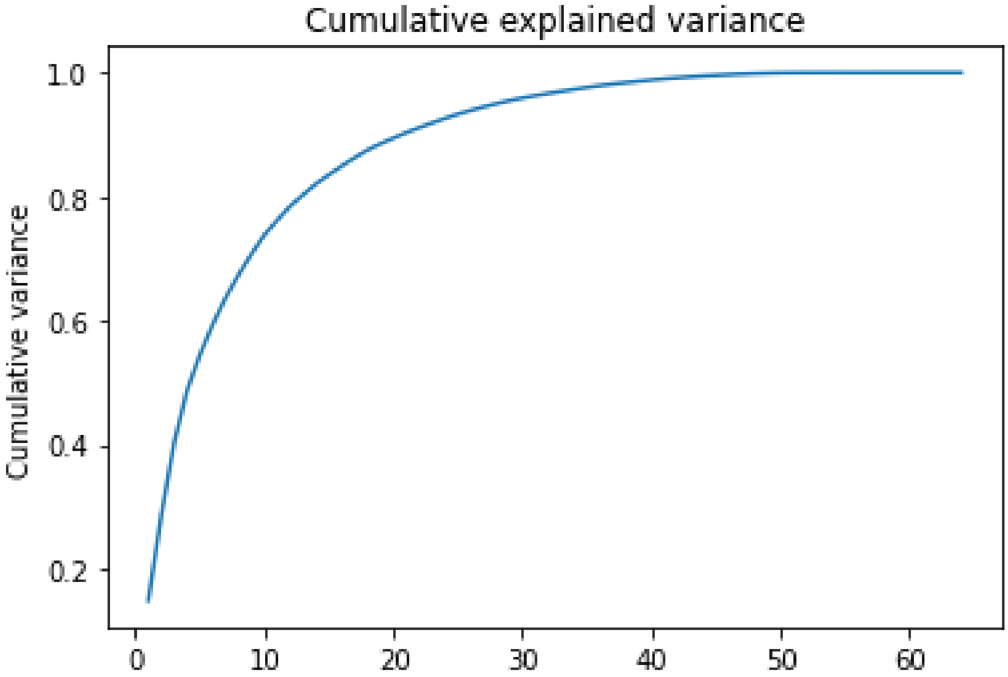
[137]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import datasets
```

[138]:
```
digits = datasets.load_digits()

print(digits.data.shape)

#normalize values to [0,1]
X = digits.data / 255.
```

```
(1797, 64)
```

1.1 A.

[139]:
```
pca = PCA(n_components=64)
X_transformed = pca.fit_transform(X)
cumsum_var=np.cumsum(pca.explained_variance_ratio_)

plt.figure()
plt.plot(range(1,65), cumsum_var)
plt.title('Cumulative explained variance')
plt.ylabel('Cumulative variance')
plt.show()
```



1.2 B.

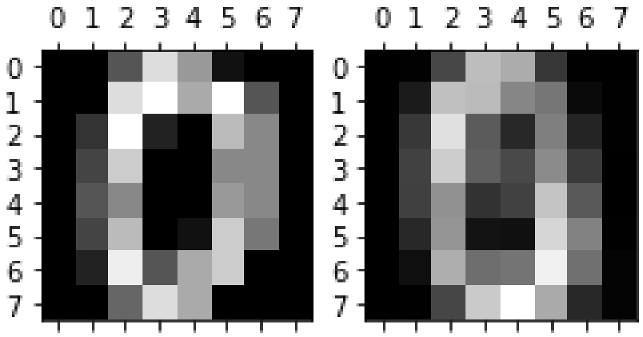
[140]:
```
print('\n\nCumulative explained variance for the best two components: \n' ,np.
↪round(cumsum_var[:2],4))
```

```
Cumulative explained variance for the best two components:
 [0.1489 0.2851]
```

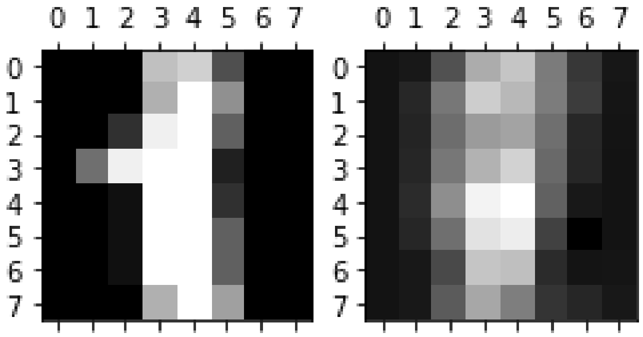
1.3 C.

[141]:
```
pca = PCA(n_components=2)
X_transformed = pca.fit_transform(X)
X_reproduce = pca.inverse_transform(X_transformed)
for i in range(5):
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(4, 3))
    axes[0].matshow(X[i].reshape(8, 8))

    axes[1].matshow(X_reproduce[i].reshape(8, 8))
    fig.suptitle('Original and Reconstructed image sample # = %i'%(i+1))
    plt.show()
```

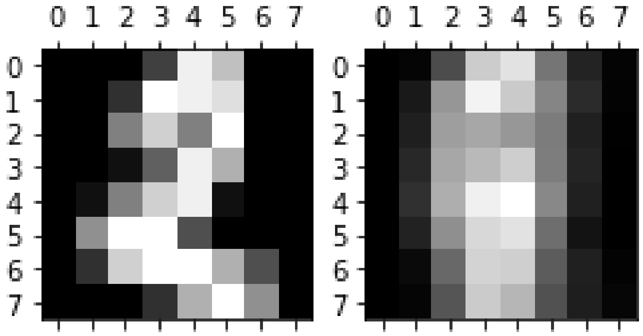
Original and Reconstructed image sample # = 1

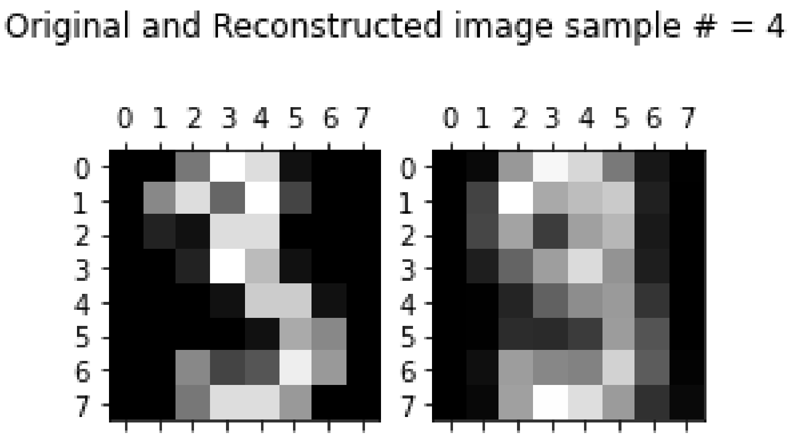


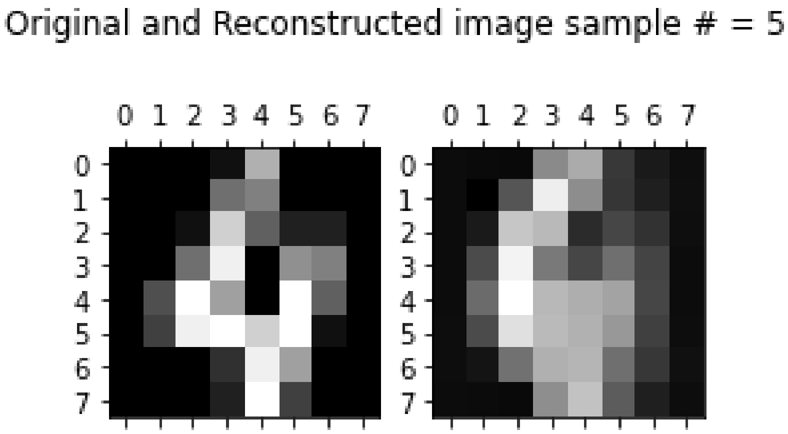
Original and Reconstructed image sample # = 2



Original and Reconstructed image sample # = 3







1.4 D.

All five figures are little distorted than the original one. However, for most figures, it is still possible to see the shape of the figure though we are just using two features compare to 64 features. This is the advantages of PCA that with only two features we can still explain about 29% of total variation.

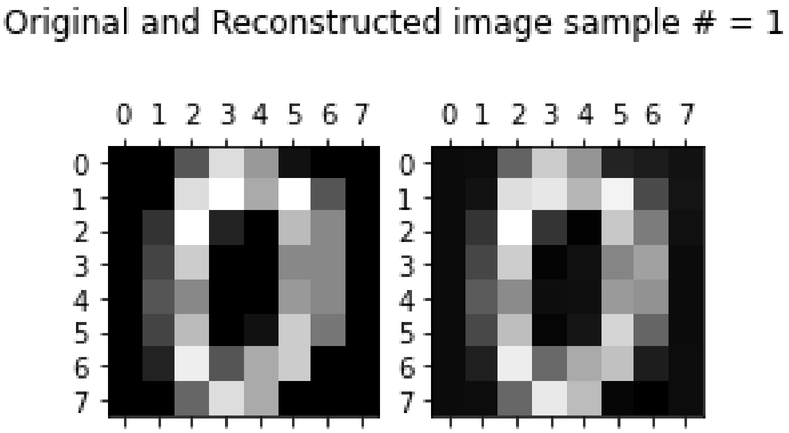
1.5 E.

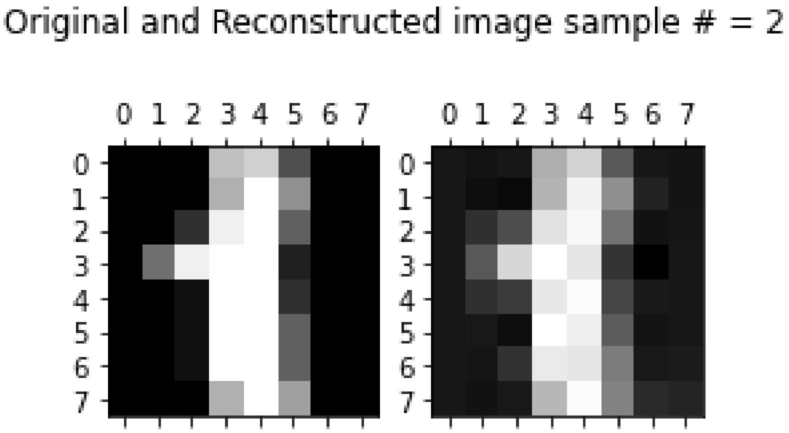
```
[142]: pca = PCA(n_components=32)
       X_transformed = pca.fit_transform(X)
       cumsum_var=np.cumsum(pca.explained_variance_ratio_)
       print('\n\ncumulative explained variance for the best two components: \n\n', np.
        ↪round(cumsum_var,4))
```

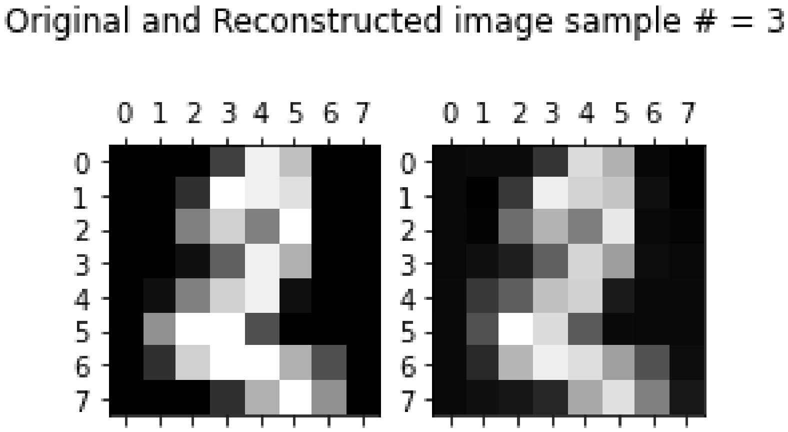
```
cumulative explained variance for the best two components:

 [0.1489 0.2851 0.403  0.4871 0.545  0.5941 0.6373 0.6739 0.7074 0.7382
 0.762  0.7847 0.8029 0.8206 0.8353 0.8494 0.8626 0.8751 0.8852 0.8943
 0.9032 0.9112 0.9188 0.9261 0.933  0.939  0.9447 0.9499 0.9548 0.9591
 0.9628 0.9664]
```

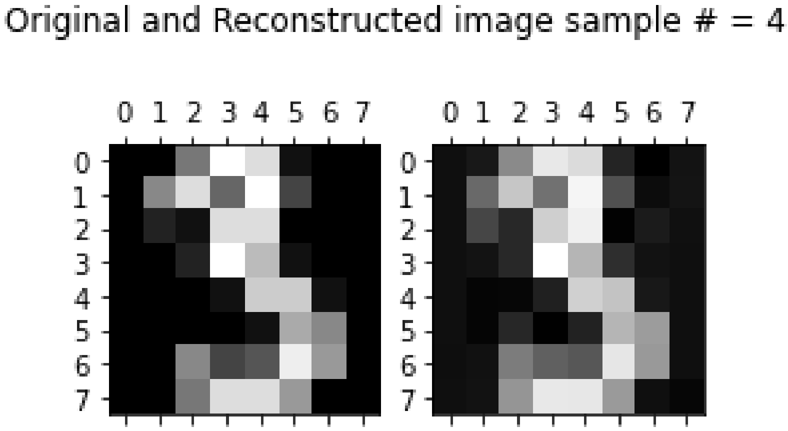
[143]:

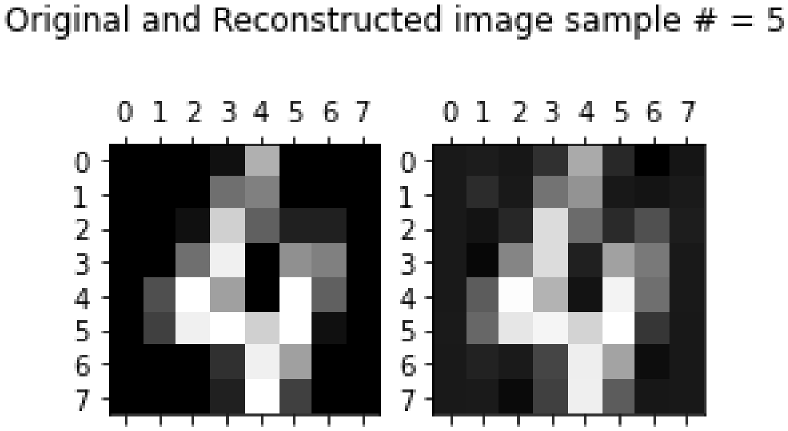
```python
X_reproduce = pca.inverse_transform(X_transformed)
for i in range(5):
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(4, 3))
    axes[0].matshow(X[i].reshape(8, 8))

    axes[1].matshow(X_reproduce[i].reshape(8, 8))
    fig.suptitle('Original and Reconstructed image sample # = %i'%(i+1))
    plt.show()
```






Original and Reconstructed image sample # = 3
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7


Original and Reconstructed image sample # = 4
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7


Original and Reconstructed image sample # = 5
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

Using 32 PCA components, all figures are almost similar with the original one and better than using two components. Using 32 components we can explain about 97% of total variation but we can reduce 32 variables.

2 Problem 3

[144]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering #to make scatterplot
from scipy.cluster import hierarchy # to make dendogram
```

[145]:

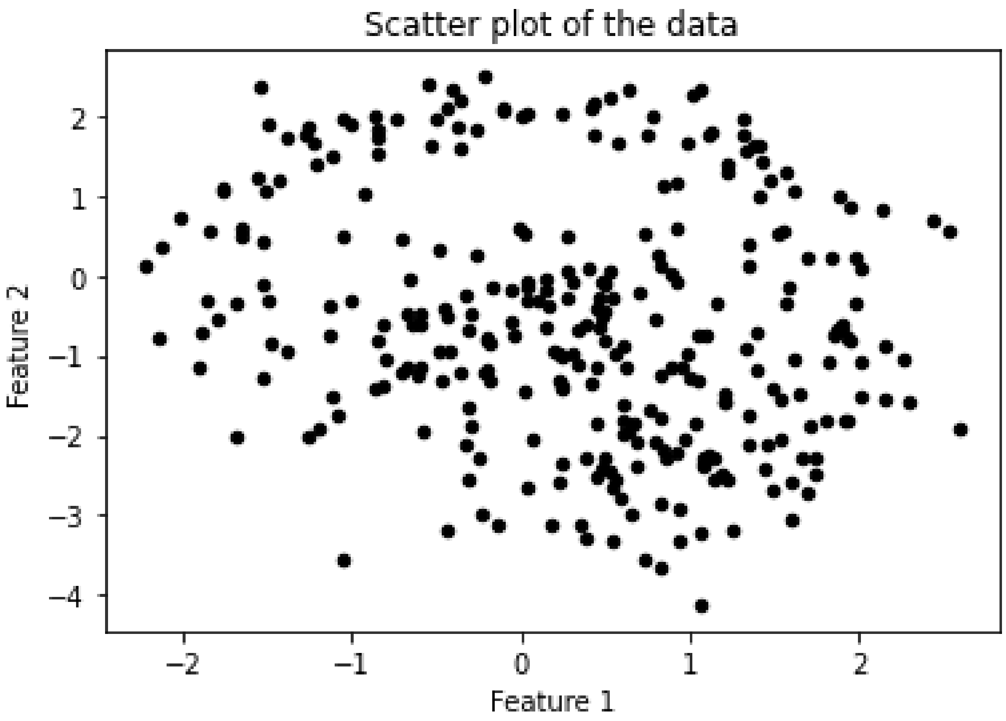
```
path  = 'D:/ISU/COMS 574 - Introduction to Machine Learning/HW/HW3/'

df_train = pd.read_csv(path + 'HW3train.csv', sep=',',
                       header=None, names=['Y', 'X1', 'X2'])
df_test = pd.read_csv(path + 'HW3test.csv', sep=',',
                      header=None, names=['Y', 'X1', 'X2'])
tr_size = df_train.shape
ts_size = df_test.shape

x_train = np.array(df_train[['X1', 'X2']])
y_train = np.array(df_train['Y'])

x_test = np.array(df_test[['X1', 'X2']])
y_test = np.array(df_test['Y'])
```

[146]:

```
#define colors to be used for plotting cluters (using hex values)
# you can pick more/different using https://htmlcolorcodes.com/
#  or https://www.w3schools.com/colors/colors_picker.asp
colors = np.array(['#377eb8', '#ff7f00', '#4daf4a', \
                         '#f781bf', '#a65628', '#984ea3', \
                         '#999999', '#e41a1c', '#dede00',\
                         '#707b7c', '#58d68d', '#1a5276',\
                         '#641e16', '#f5cba7', '#212f3d',\
                  '#da7e6b', '#e9c6bf', '#cd9227', '#74cd27',\
                  '#40cd27', '#27cd7f', '#27cdbe', '#27b8cd',\
                  '#27a2cd', '#5594b2', '#5574b2', '#6655b2', \
                   '#5855b2', '#6655b2', '#9c55b2', '#aa55b2', \
                   '#b255b1', '#b25594'])
```

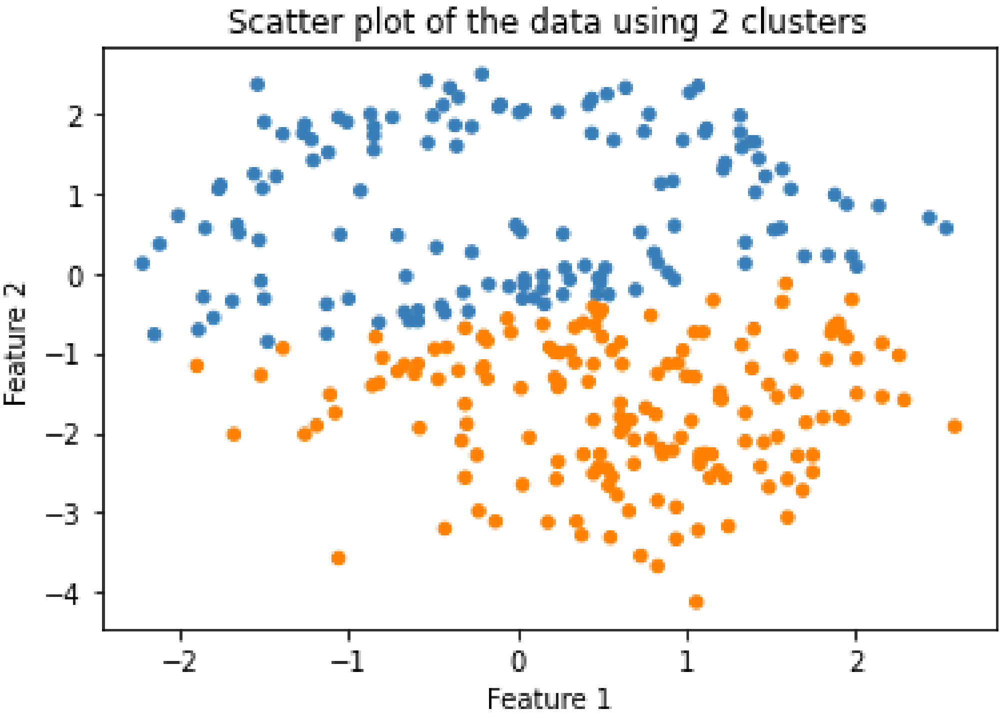
2.1 A.

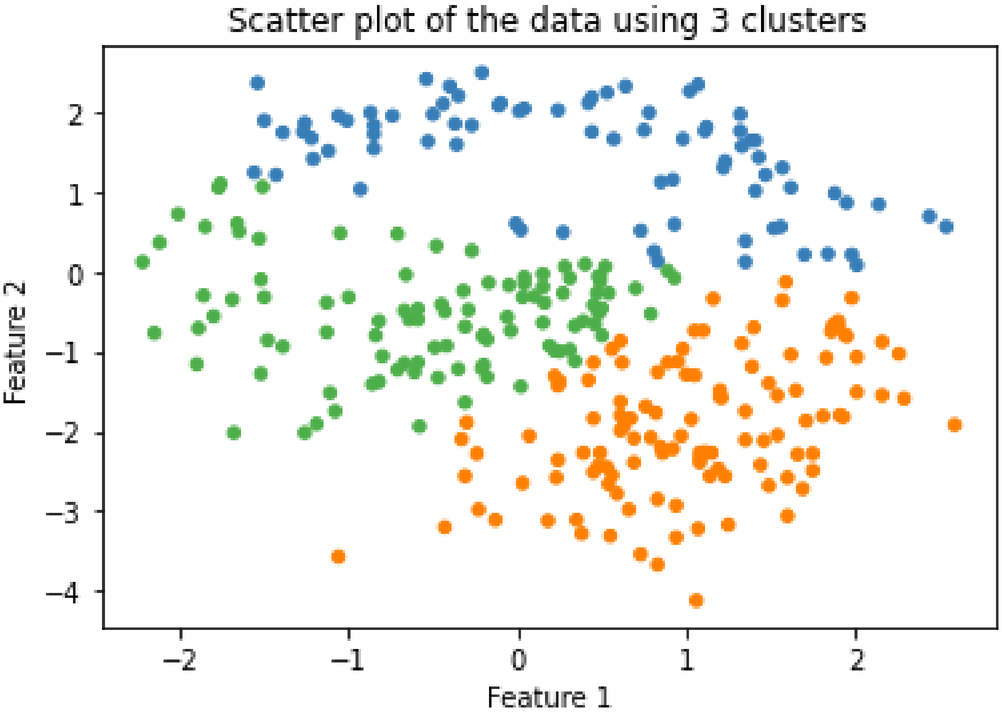
[147]:
```
plt.figure()
plt.scatter(x_train[:, 0], x_train[:, 1], s=20, color='k')

plt.title('Scatter plot of the data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
```

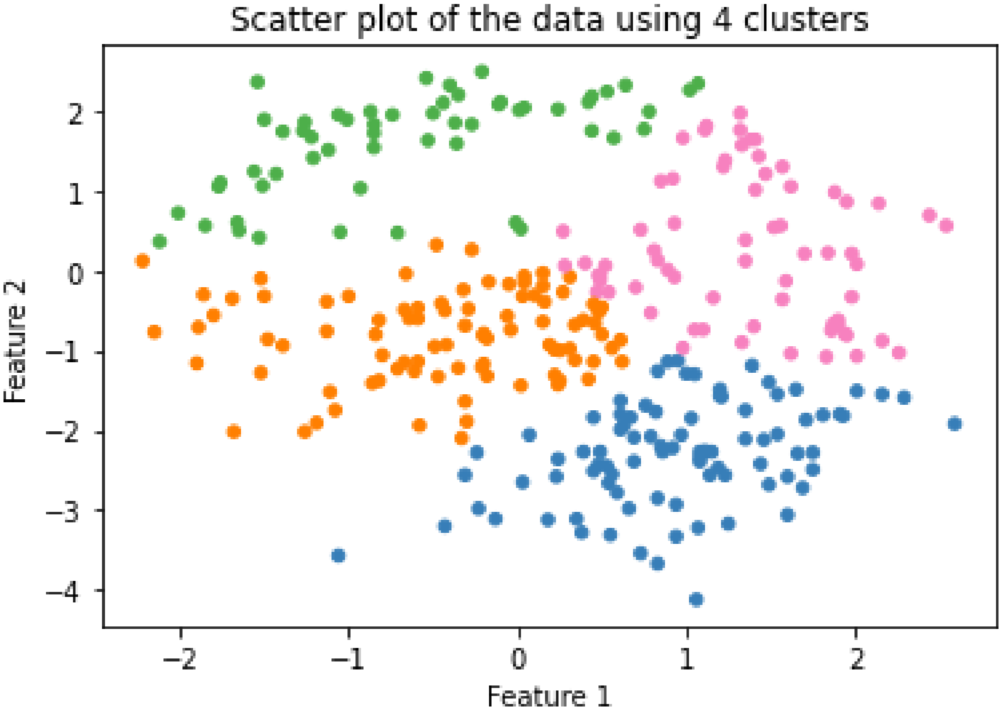


2.2 B.

[148]:
```
k = range(1,16)
for i in [2,3,4]:
    clustering = KMeans(n_clusters=i, n_init=50, max_iter=250, n_jobs=-1,␣
 ↪init='random')
    clustering.fit(x_train)

    # Plot the training points
    plt.figure()
    plt.scatter(x_train[:, 0], x_train[:, 1], s=20,color=colors[clustering.
 ↪labels_])
    plt.title('Scatter plot of the data using %i clusters'%(i))
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

    plt.show()
```

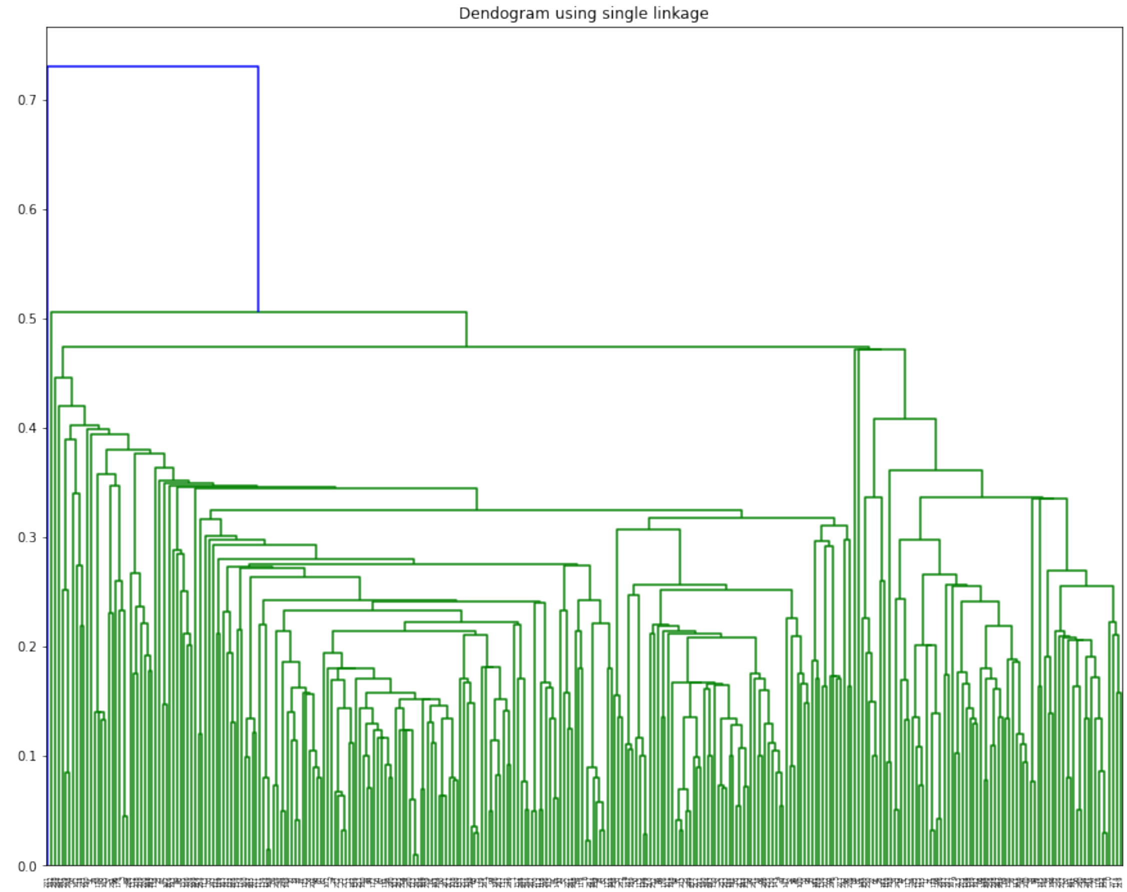



Scatter plot of the data using 3 clusters
Feature 2
Feature 1
2
1
0
−1
−2
−3
−4
−2
−1
0
1
2


Scatter plot of the data using 4 clusters
Feature 2
Feature 1
2
1
0
−1
−2
−3
−4
−2
−1
0
1
2

2.3 C.

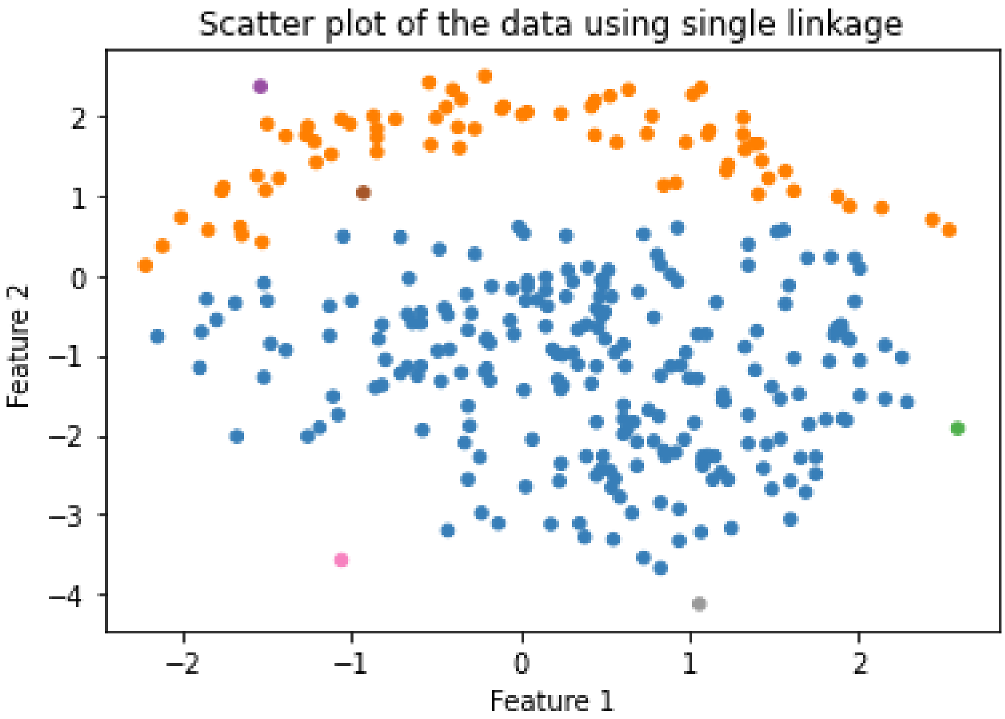
```
[149]: link = 'single'

       Xlink = hierarchy.linkage(x_train, link)
       plt.figure(figsize=(15, 12))
       plt.title('Dendogram using %s linkage'%(link))
       dn = hierarchy.dendrogram(Xlink)
```



3 D.

```
[150]: thresh=0.43

       clustering = AgglomerativeClustering(n_clusters=None, \
                           affinity='euclidean',linkage=link, \
                           distance_threshold=thresh,compute_full_tree=True)

       clustering.fit(x_train)
```

```
# Plot the training points
plt.figure()
plt.scatter(x_train[:, 0], x_train[:, 1], s=20, color=colors[clustering.
 ↪labels_])
plt.title('Scatter plot of the data using %s linkage'%(link))
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
print('Distance threshold for '+link+' linkage: %f' %thresh)
```



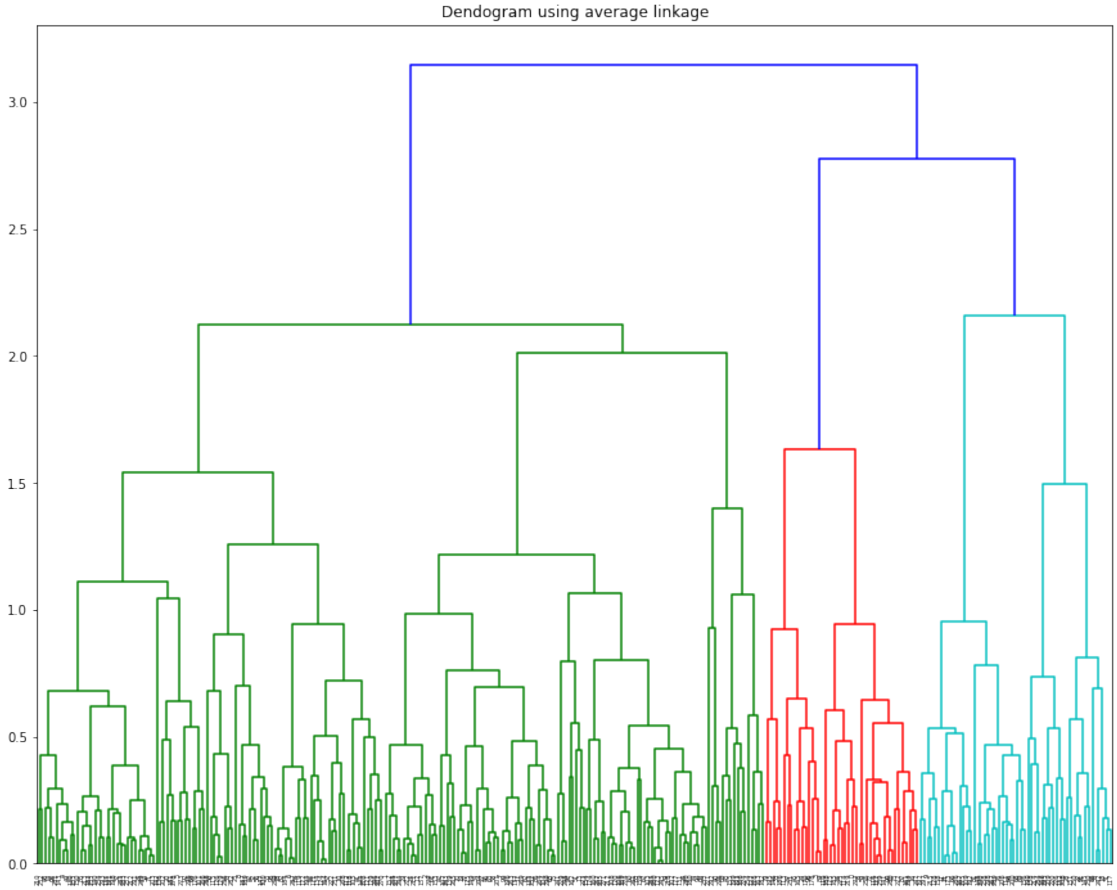
```
Distance threshold for single linkage: 0.430000
```

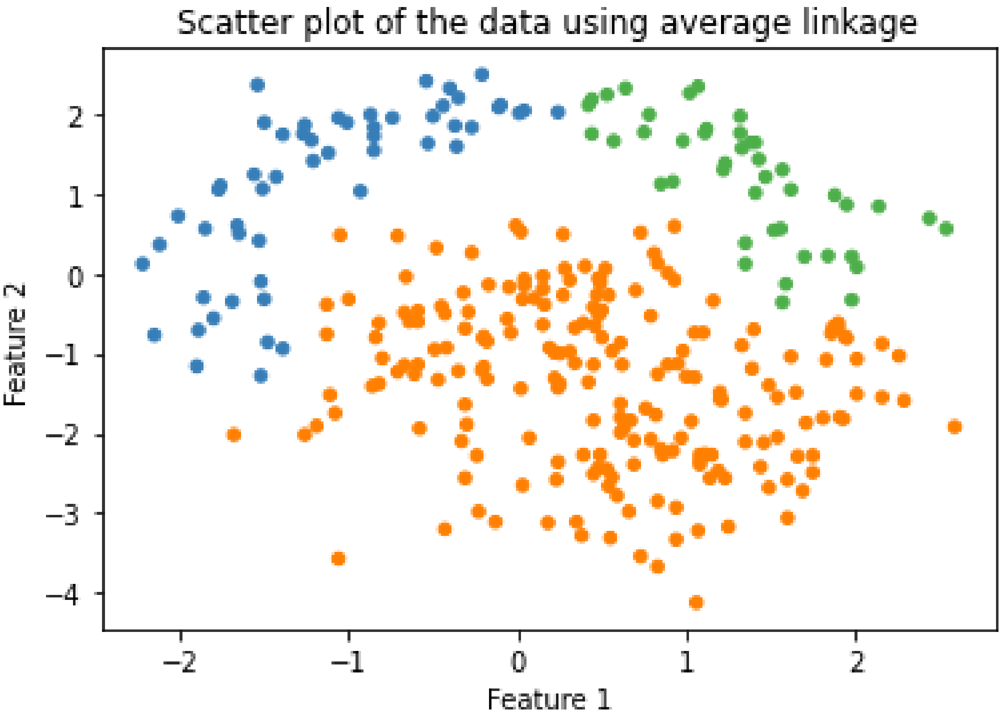
4 E.

4.1 'average' Linkage

[151]:
```
link = 'average'
thresh=2.5

Xlink = hierarchy.linkage(x_train, link)
plt.figure(figsize=(15, 12))
plt.title('Dendogram using %s linkage'%(link))
```

```
dn = hierarchy.dendrogram(Xlink)


clustering = AgglomerativeClustering(n_clusters=None, \
                    affinity='euclidean',linkage=link, \
                    distance_threshold=thresh,compute_full_tree=True)

clustering.fit(x_train)

# Plot the training points
plt.figure()
plt.scatter(x_train[:, 0], x_train[:, 1], s=20, color=colors[clustering.
 ↪labels_])
plt.title('Scatter plot of the data using %s linkage'%(link))
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
print('Distance threshold for '+link+' linkage: %f' %thresh)
```



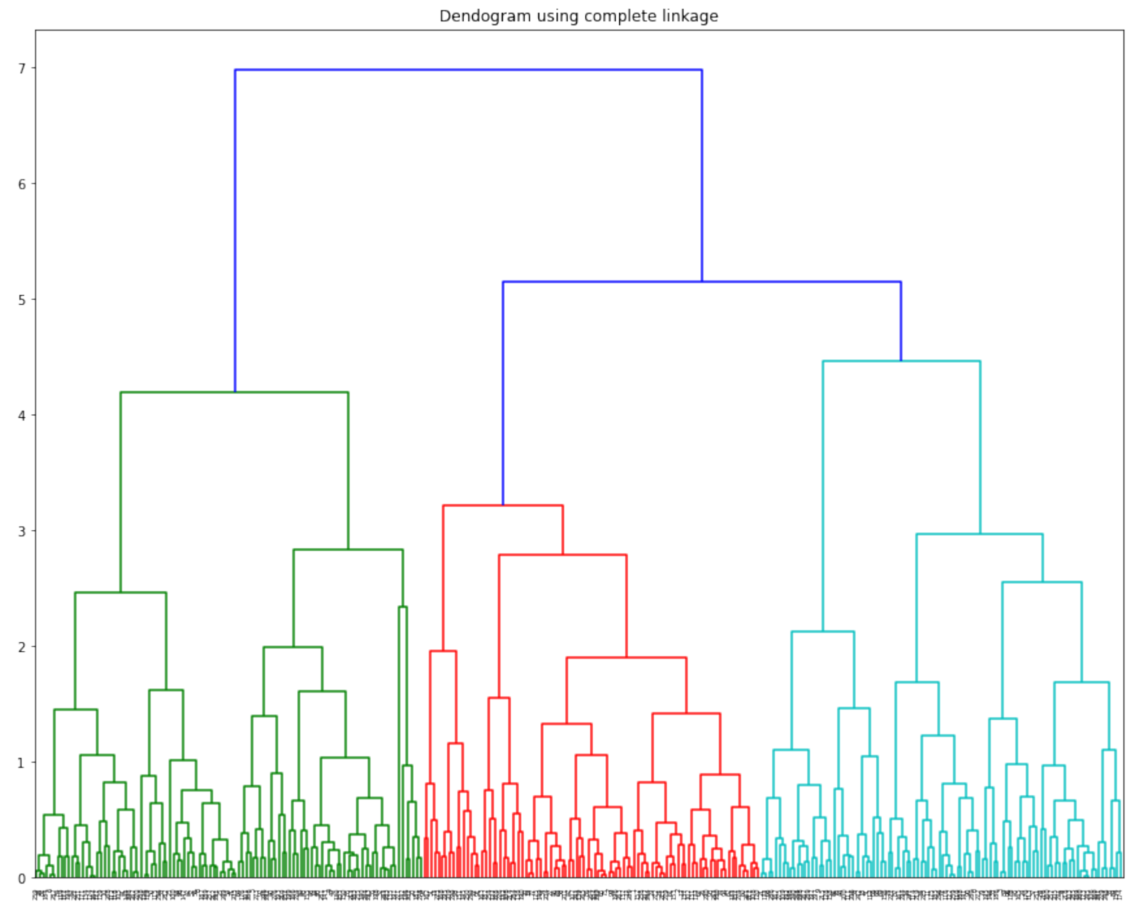


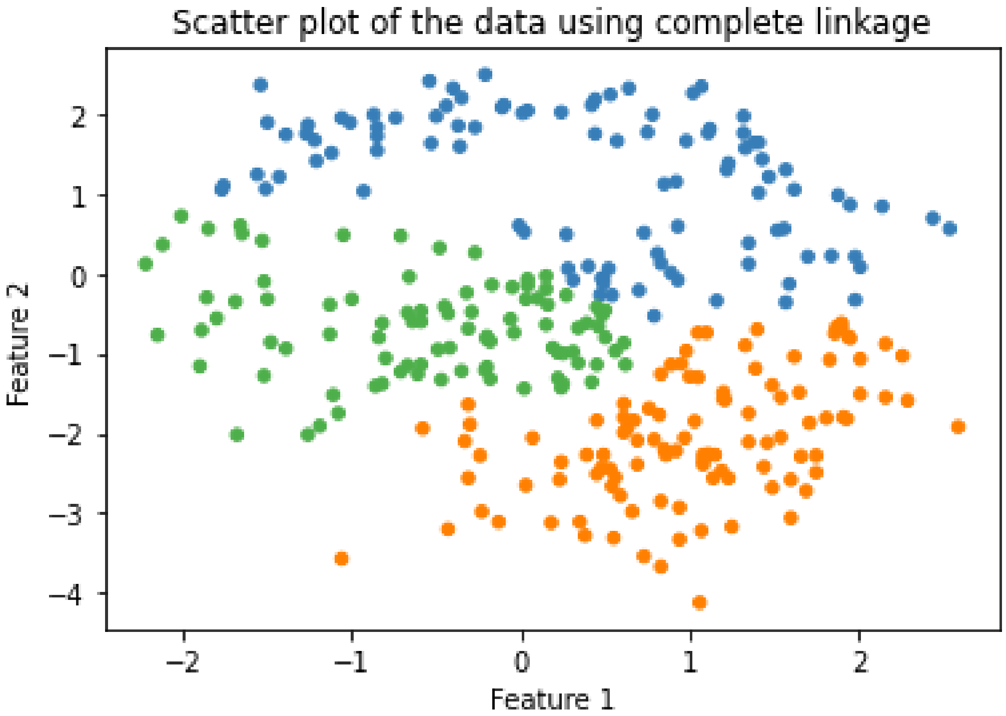
```
Distance threshold for average linkage: 2.500000
```

5 F.

5.1 ‘complete’ Linkage

[152]:
```
link = 'complete'
thresh=4.5

Xlink = hierarchy.linkage(x_train, link)
plt.figure(figsize=(15, 12))
plt.title('Dendogram using %s linkage'%(link))
dn = hierarchy.dendrogram(Xlink)


clustering = AgglomerativeClustering(n_clusters=None, \
                     affinity='euclidean',linkage=link, \
                     distance_threshold=thresh,compute_full_tree=True)

clustering.fit(x_train)

# Plot the training points
plt.figure()
plt.scatter(x_train[:, 0], x_train[:, 1], s=20, color=colors[clustering.
 ↪labels_])
plt.title('Scatter plot of the data using %s linkage'%(link))
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
print('Distance threshold for '+link+' linkage: %f' %thresh)
```





```
Distance threshold for complete linkage: 4.500000
```

6 G.

K-means: To do clustering using K-means, we need to know the number of cluster but for Hierarchical clustering we don't need to know the number of clusters in advance. For this data, k-means cluster with $k = 2$ looks almost similar with single linkage cluster. Similarly, for three center k-means looks similar with complete linkage cluster. It also works better for large number of observations.

Single Linkage: It is clearly seen from the dendogram, that it is difficult to find distinct cluster within data. Single linkage grouped observations based on the nearest neighbor technique. It uses the most similar items into a cluster and tends to include observations into existing clusters rather than creating new cluster though it can detect irregularly shaped clusters. It is possible to take decision about the number of clusters based on visual inspection. For this data it seems there is no similar pattern found by any other methods that can be close to single linkage. It not a good method for large data.

Average Linkage: Average linkage methods clearly shows that there are some group of observations that belongs to the same group compare to single linkage method. It also looks (visually) almost similar to complete linkage method. Average linkage method considers the distance from the center of a cluster to new observations.

Complete Linkage: For this data, it clearly shows three distinct and balanced cluster. It uses the furthest neighbor technique to find pairwise distance.

For this data, complete linkage resulted in the best results. Because, in this data set there are no well separated cluster and since complete linkage includes observation in the same group if they are closed from the farthest distance. However, k-means with $k = 3$ show also better performance for this data. If data size is relatively small, we can use complete linkage or average linkage method, especially if there are no well separated cluster in the observations.

[ ]:

```
```