Homework 5

In [1]:

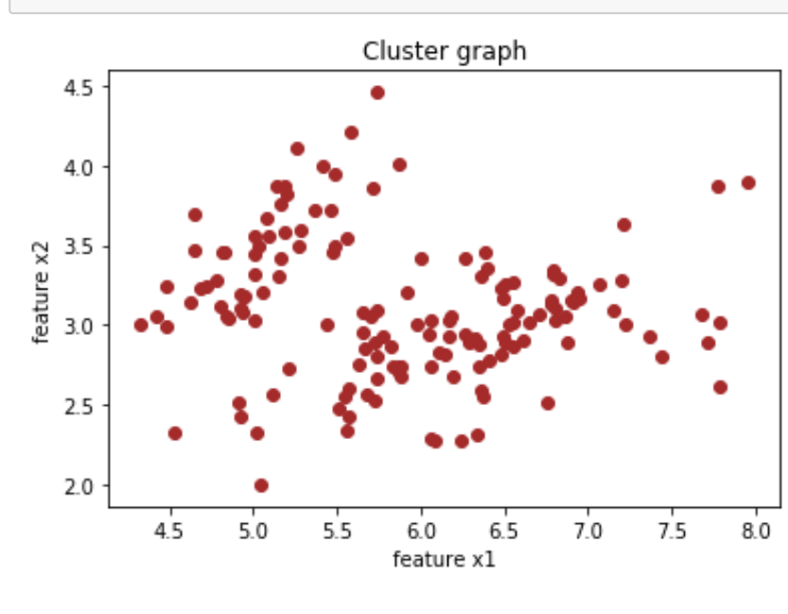
```python
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
from scipy import linalg
import mltools.cluster as clust
```

Problem 1

Problem 1 - Part 1

In [2]:

```python
iris = np.genfromtxt('data/iris.txt', delimiter= None)

X, Y = iris[:, 0:2], iris[:, -1]

plt.scatter(X[:,0],X[:,1], color = 'brown')
plt.xlabel('feature x1')
plt.ylabel('feature x2')
plt.title('Cluster graph')
plt.show()
```



In my opinion, based on the visual representation above of the features X_1 and X_2 there can be two clusters. One cluster is the points gathered in the bottom right corner of the figure and the other cluster is the points on the top left.

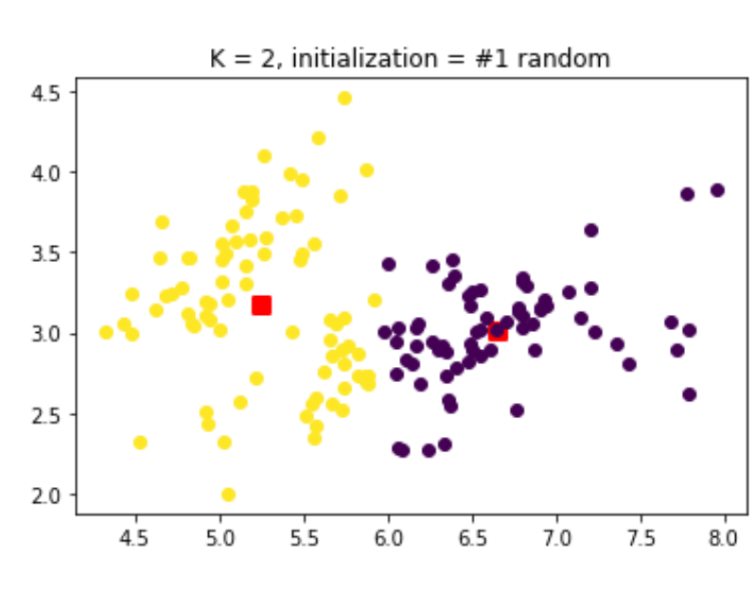
Problem 1 - Part 2

K = 2

In [3]:

```python
# define different k and initializations
cluster = 2
initializations = ['random','random','random','k++','farthest']

z   = [None] * len(initializations)
mu  = [None] * len(initializations)
ssd = [None] * len(initializations)

print("\nk =", cluster, "\n------")
for j, initial in enumerate(initializations):
    np.random.seed(j * 10)
    z[j], mu[j], ssd[j] = ml.cluster.kmeans(X, K = cluster, init = initial, max_iter = 100)
    print("initialization = #", j + 1, initial, "\tsumd =", ssd[j])
if len(set(ssd)) != 1:
    best_solution = ssd.index(np.min(ssd))
    print('\nFor K = %d, initialization #%d/%s, has the smallest sumd'
          % (cluster, best_solution + 1, initializations[best_solution]))
    plt.scatter(mu[best_solution][:, 0], mu[best_solution][:, 1], s = 50, marker= 's',
                facecolor = 'red', lw = 2)
    ml.plotClassify2D(None, X, z[best_solution])
    plt.title('K = %d, initialization = #%d %s' %(cluster, best_solution + 1, initializations[best_s
olution]))
    print("\n")
    plt.show()
else:
    print('These initializations find the same solution')
```

```
k = 2
------
initialization = # 1 random     sumd = 57.877648396983034
initialization = # 2 random     sumd = 57.87966196118197
initialization = # 3 random     sumd = 57.877648396983034
initialization = # 4 k++        sumd = 57.877648396983034
initialization = # 5 farthest   sumd = 57.87966196118197

For K = 2, initialization #1/random, has the smallest sumd
```

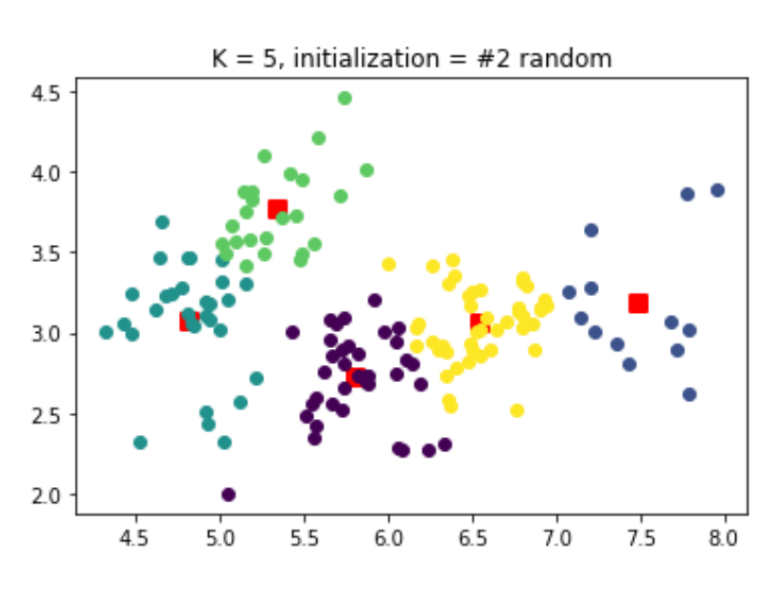


K = 5

In [4]:

```python
# define different k and initializations
cluster = 5
initializations = ['random','random','random','k++','farthest']

z   = [None] * len(initializations)
mu  = [None] * len(initializations)
ssd = [None] * len(initializations)

print("\nk =", cluster, "\n------")
for j, initial in enumerate(initializations):
    np.random.seed(j * 10)
    z[j], mu[j], ssd[j] = ml.cluster.kmeans(X, K = cluster, init = initial, max_iter = 100)
    print("initialization = #", j + 1, initial, "\tsumd =", ssd[j])
if len(set(ssd)) != 1:
    best_solution = ssd.index(np.min(ssd))
    print('\nFor K = %d, initialization #%d/%s, has the smallest sumd'
          % (cluster, best_solution + 1, initializations[best_solution]))
    plt.scatter(mu[best_solution][:, 0], mu[best_solution][:, 1], s = 50, marker= 's',
                facecolor = 'red', lw = 2)
    ml.plotClassify2D(None, X, z[best_solution])
    plt.title('K = %d, initialization = #%d %s' % (cluster, best_solution + 1, initializations[best_
solution]))
    print("\n")
    plt.show()
else:
    print('These initializations find the same solution')
```

```
k = 5
------
initialization = # 1 random     sumd = 23.438643757297644
initialization = # 2 random     sumd = 20.886682200353754
initialization = # 3 random     sumd = 25.138941899584196
initialization = # 4 k++        sumd = 21.0902063018713
initialization = # 5 farthest   sumd = 20.954630196254048

For K = 5, initialization #2/random, has the smallest sumd
```

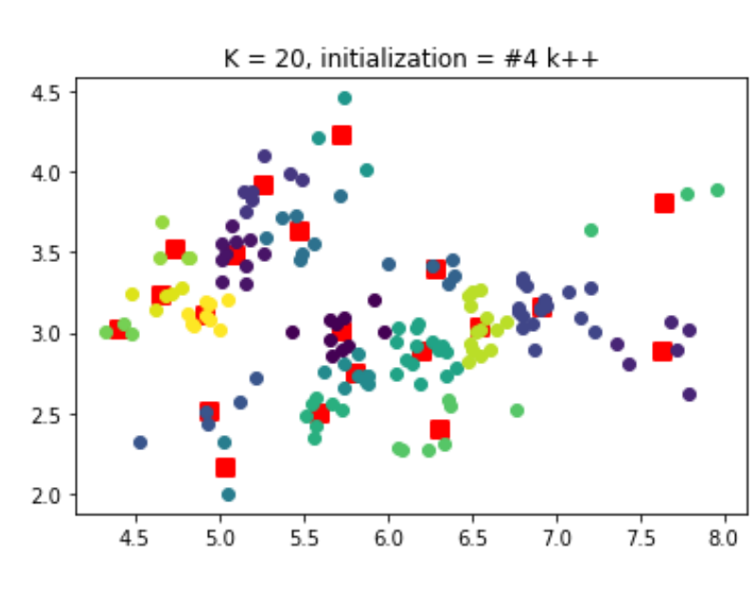


k = 20

In [5]:

```python
# define different k and initializations
cluster = 20
initializations = ['random','random','random','k++','farthest']

z   = [None] * len(initializations)
mu  = [None] * len(initializations)
ssd = [None] * len(initializations)

print("\nk =", cluster, "\n------")
for j, initial in enumerate(initializations):
    np.random.seed(j * 10)
    z[j], mu[j], ssd[j] = ml.cluster.kmeans(X, K = cluster, init = initial, max_iter = 100)
    print("initialization = #", j + 1, initial, "\tsumd =", ssd[j])
if len(set(ssd)) != 1:
    best_solution = ssd.index(np.min(ssd))
    print('\nFor K = %d, initialization #%d/%s, has the smallest sumd'
          % (cluster, best_solution + 1, initializations[best_solution]))
    plt.scatter(mu[best_solution][:, 0], mu[best_solution][:, 1], s = 50, marker= 's',
                facecolor = 'red', lw = 2)
    ml.plotClassify2D(None, X, z[best_solution])
    plt.title('K = %d, initialization = #%d %s' % (cluster, best_solution + 1, initializations[best_
solution]))
    print("\n")
    plt.show()
else:
    print('These initializations find the same solution')
```

```
k = 20
------
initialization = # 1 random     sumd = 4.848796922616994
initialization = # 2 random     sumd = 6.375178437745638
initialization = # 3 random     sumd = 4.901454051727956
initialization = # 4 k++        sumd = 4.431755978617504
initialization = # 5 farthest   sumd = 4.929483325105752

For K = 20, initialization #4/k++, has the smallest sumd
```

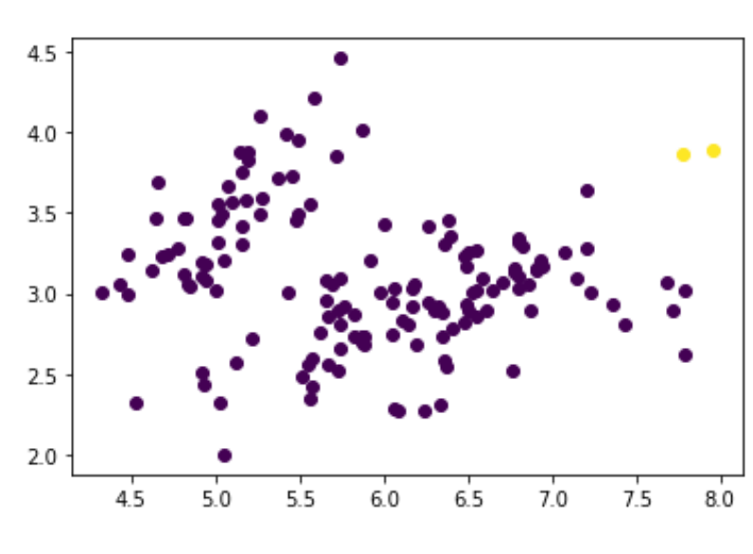


Problem 1 - Part 3

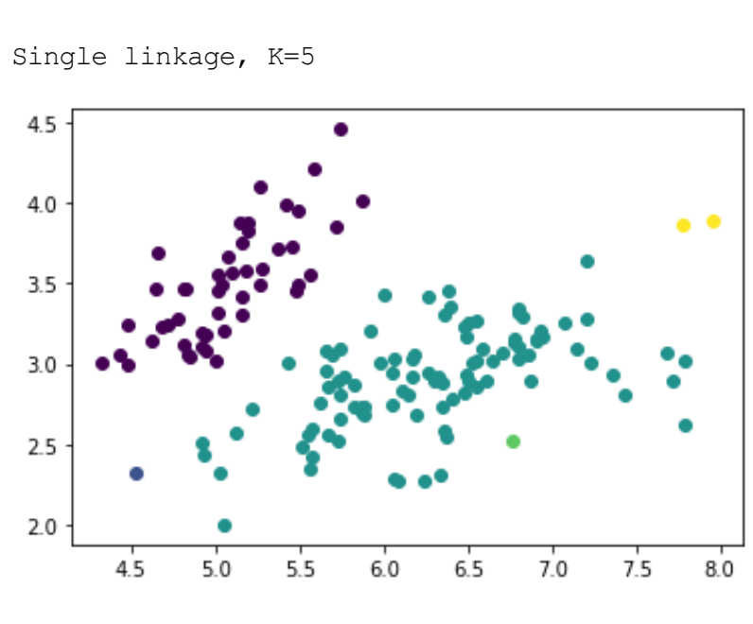
In [6]:

```python
Z,dend=clust.agglomerative(X,K=2,method='min')
plt.figure()
print("Single linkage, K=2")
ml.plotClassify2D(None,X,Z)
plt.show()

Z,dend=clust.agglomerative(X,K=5,method='min')
plt.figure()
print("Single linkage, K=5")
ml.plotClassify2D(None,X,Z)
plt.show()

Z,dend=clust.agglomerative(X,K=20,method='min')
plt.figure()
print("Single linkage, K=20")
ml.plotClassify2D(None,X,Z)
plt.show()

Z,dend=clust.agglomerative(X,K=2,method='max')
plt.figure()
print("Complete linkage, K=2")
ml.plotClassify2D(None,X,Z)
plt.show()

Z,dend=clust.agglomerative(X,K=5,method='max')
plt.figure()
print("Complete linkage, K=5")
ml.plotClassify2D(None,X,Z)
plt.show()

Z,dend=clust.agglomerative(X,K=20,method='max')
plt.figure()
print("Complete linkage, K=20")
ml.plotClassify2D(None,X,Z)
plt.show()
```

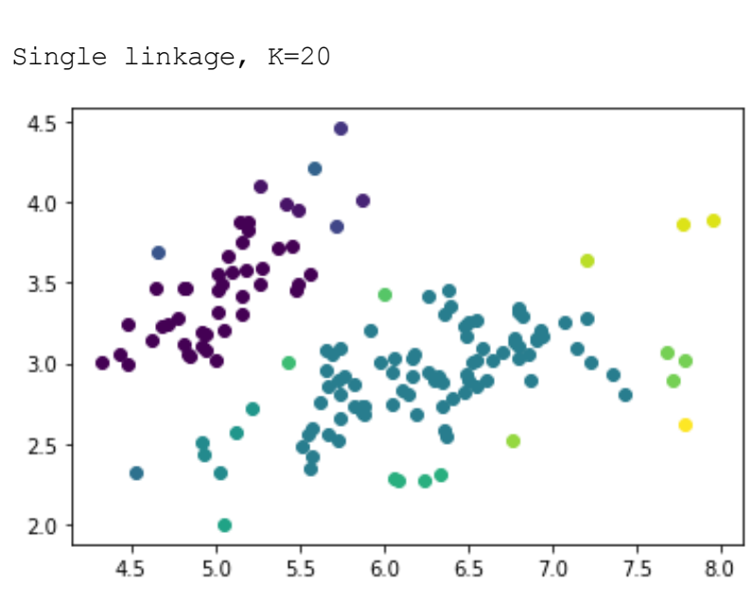
```
Single linkage, K=2
```



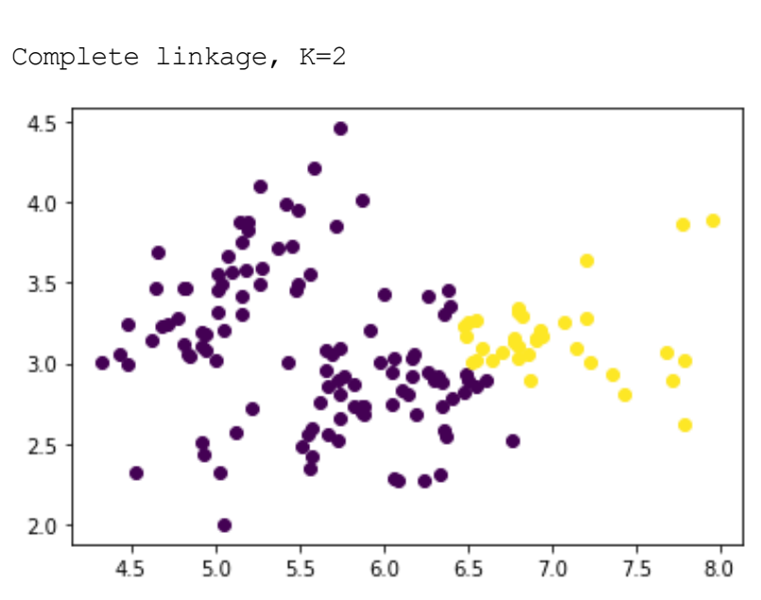
```
Single linkage, K=5
```



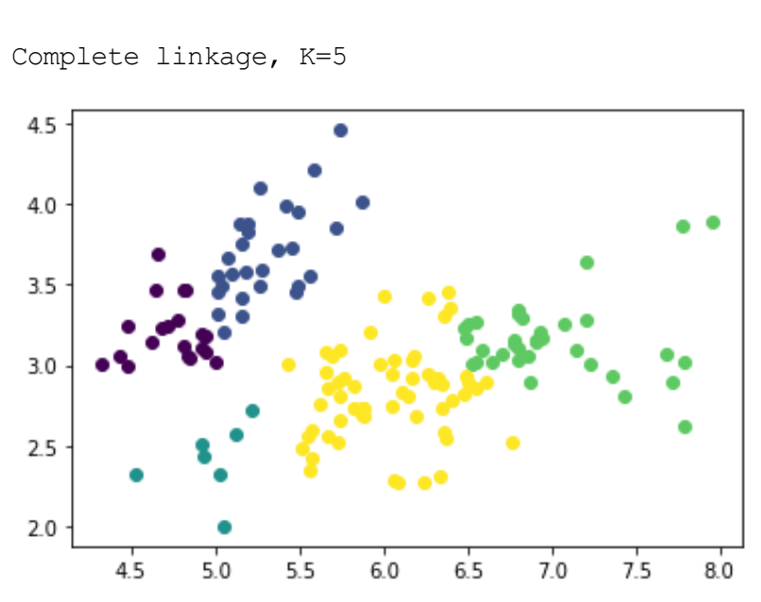
```
Single linkage, K=20
```



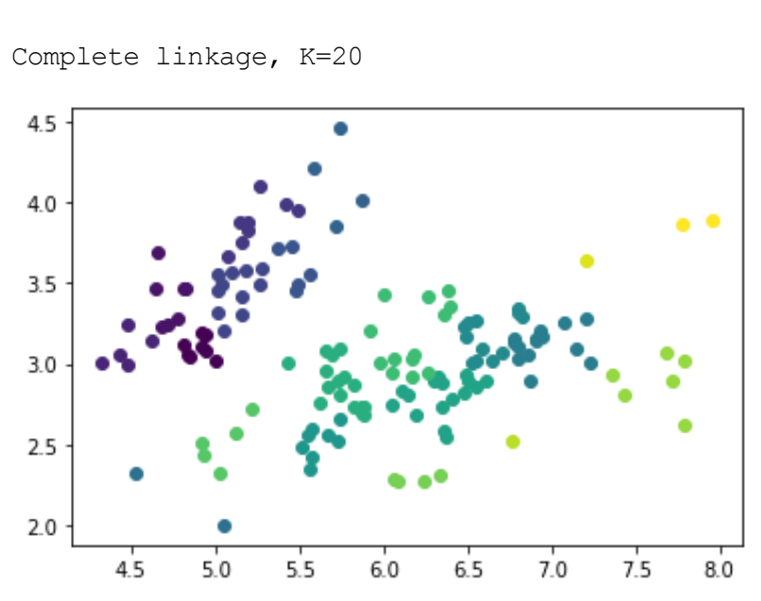
```
Complete linkage, K=2
```



```
Complete linkage, K=5
```



```
Complete linkage, K=20
```



Problem 1 - Part 4

Differences:

K-means clustering are more likely to group the points into different clusters evenly with similar size.
However, for agglomerative clustering, 'single linkage' groups density points into some clusters with large size while group some points as clusters with quite small size so it produces MST. However, 'complete linkage' avoids elongated clusters.

Similarities:

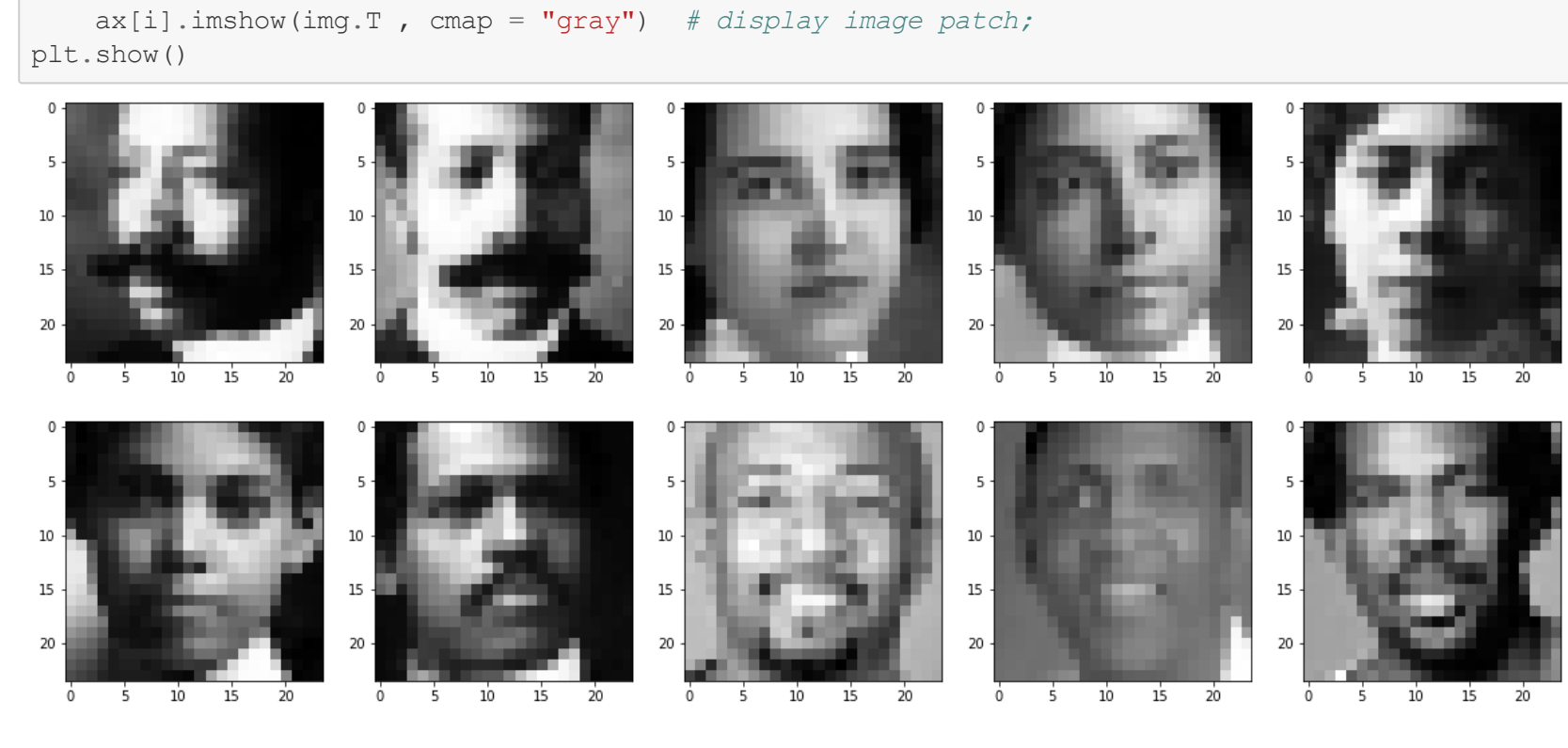
the result of 'complete linkage' is almost similar to K-means

Problem 2

We load the data first and display the first 10 faces to gain knowledge of the data format

In [7]:

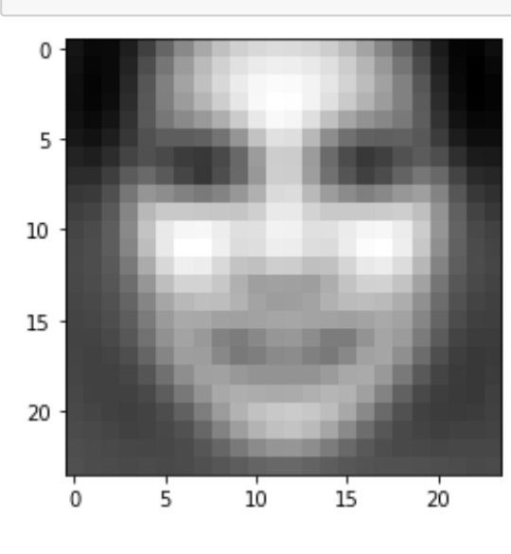
```python
X = np.genfromtxt("data/faces.txt", delimiter = None) # load face dataset

# pick the first 10 faces for display
fig, ax = plt.subplots(2, 5, figsize = (20, 8))
ax = ax.flatten()
for i in range(len(ax)):
    # convert vectorized data to 24x24 image patches
    img = np.reshape(X[i,:] ,(24, 24))
    ax[i].imshow(img.T , cmap = "gray")  # display image patch;
plt.show()
```



Problem 2 - Part 1

In [8]:

```python
mu = np.mean(X, axis = 0, keepdims = True ) # find mean over data points
X0 = X - mu # zero-center the data

# Plot the mean face
plt.figure()
img = np.reshape(mu, (24, 24))
plt.imshow(img.T , cmap="gray")
plt.show()
```



Problem 2 - Part 2

In [9]:

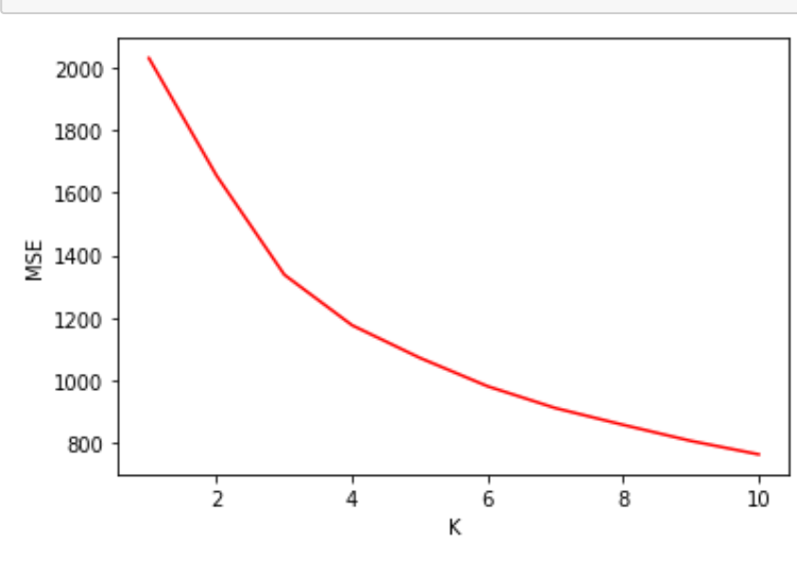
```python
U,S,Vh = linalg.svd(X0, full_matrices = False) # X0 = U * diag(S) * Vh
W = U.dot(np.diag(S)) # W = U * diag(S)
print('Shapes of W  :', W.shape)
print('Shapes of V_h:', Vh.shape)
```

```
Shapes of W  : (4916, 576)
Shapes of V_h: (576, 576)
```

Problem 2 - Part 3

In [10]:

```python
K = range(1,11)
mse = np.zeros(len(K))
for i,k in enumerate(K):
    # approx using k largest eigendirections
    X0_hat = W[:,:k].dot(Vh[:k,:])
    # compute the mean squared error in the SVD's approximation
    mse[i] = np.mean((X0 - X0_hat) ** 2)

plt.plot(K, mse, color = 'red')
plt.xlabel('K')
plt.ylabel('MSE')
plt.show()
```

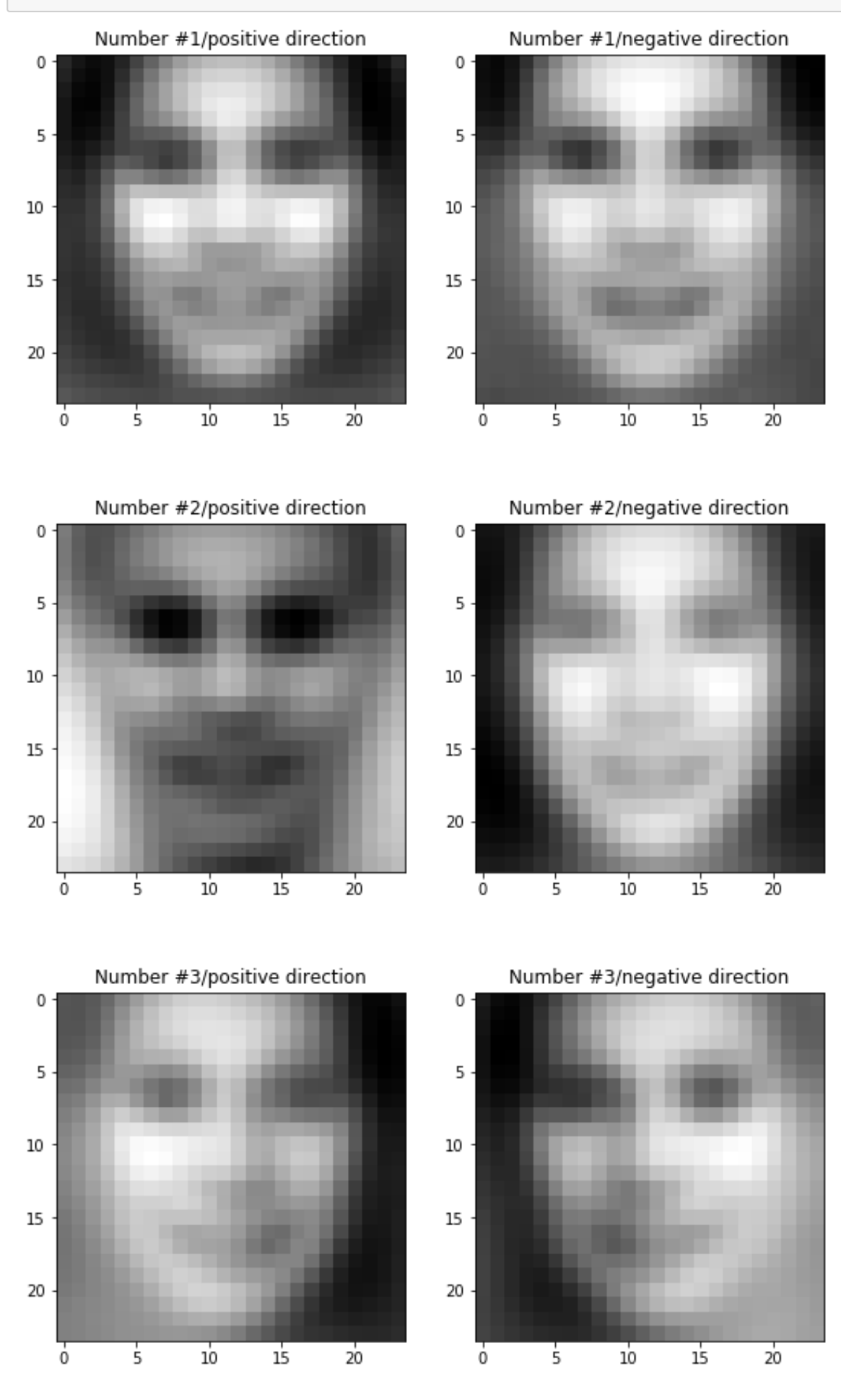


Based on the plot above, as k increases, MSE in the SVD's approximation decreases

Problem 2 - Part 4

In [11]:

```python
fig, ax = plt.subplots(3, 2, figsize = (9, 16))
for i in range(3):
    alpha = 2 * np.median(np.abs(W[:,i]))
    imgage_1  = np.reshape(mu + alpha*Vh[i,:],(24, 24))
    imgage_2  = np.reshape(mu - alpha*Vh[i,:],(24, 24))
    ax[i][0].imshow( imgage_1.T , cmap = "gray")
    ax[i][0].set_title('Number #%d/positive direction' % (i + 1))
    ax[i][1].imshow( imgage_2.T , cmap = "gray")
    ax[i][1].set_title('Number #%d/negative direction' % (i + 1))
plt.show()
```

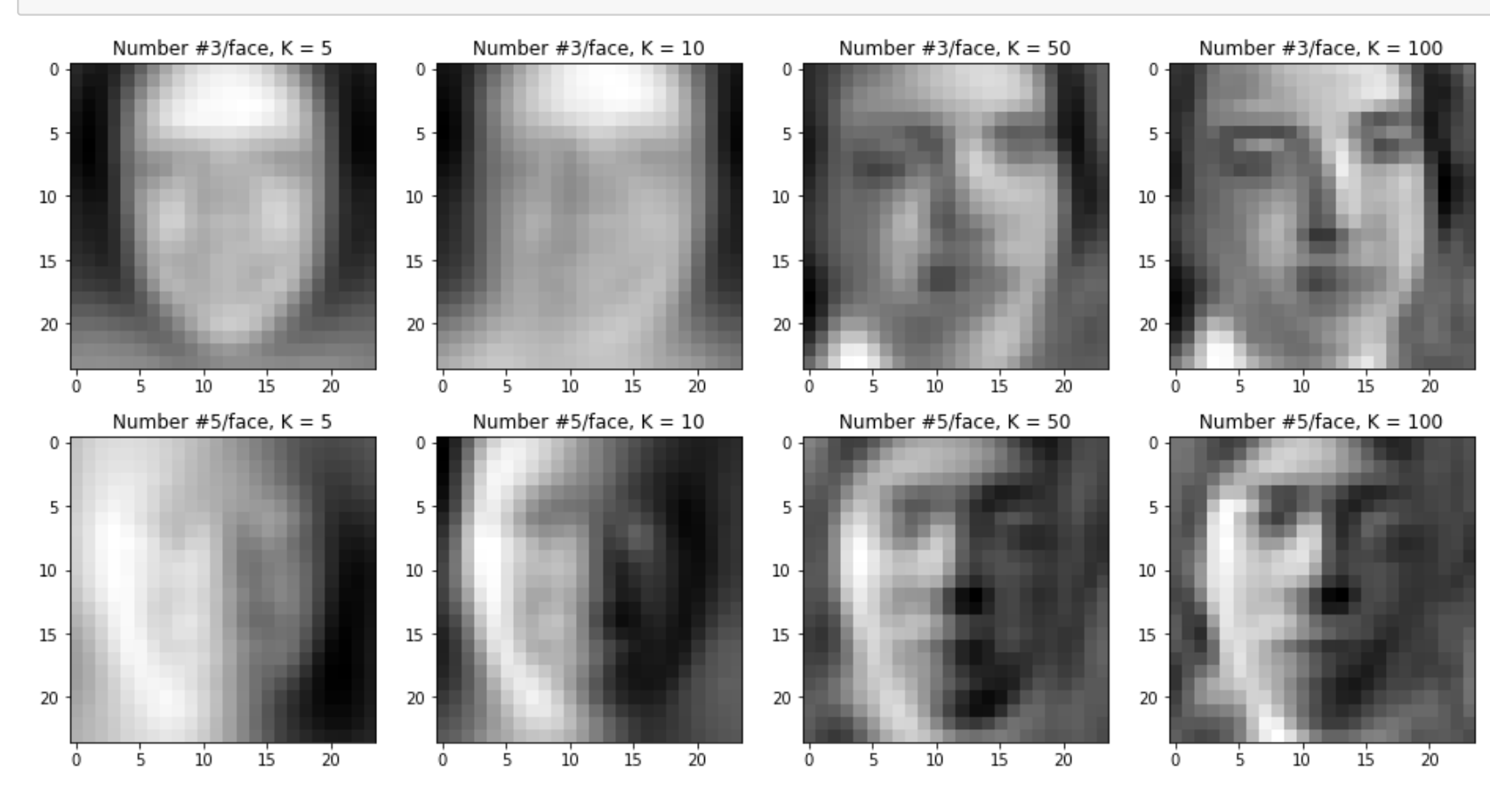


Problem 2 - Part 5

I select the 3rd and 5th faces and reconstruct them for K = 5, 10, 50, 100.

In [12]:

```python
fig, ax = plt.subplots(2,4,figsize = (16, 8))
K = [5, 10, 50, 100]
Xn = [3, 5]
for i,n in enumerate(Xn):
    for j,k in enumerate(K):
        Xnhat = W[n-1,:k].dot(Vh[:k,:])
        img = np.reshape(Xnhat,(24,24))
        ax[i][j].imshow( img.T , cmap="gray")
        ax[i][j].set_title('Number #%d/face, K = %d' % (n,k))
plt.show()
```



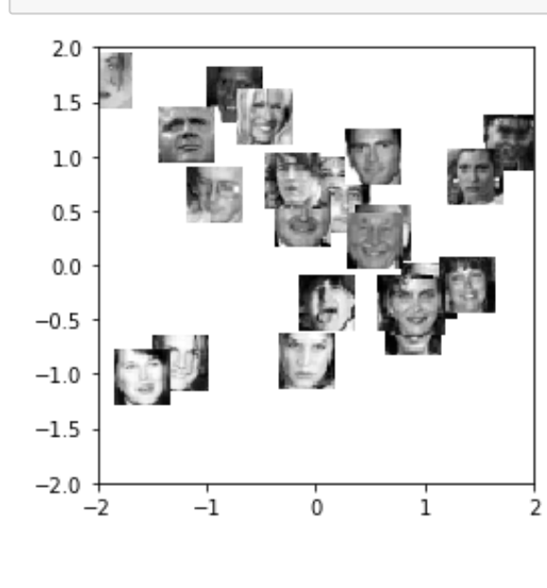
Based on the resluts shown above, when K becomes larger, the face becomes closer to the original

Problem 2 - Part 6

I randomly choose 25 faces, and display them as images with the coordinates given by their coefficients on the first two principal components.

In [13]:

```python
idx = np.random.randint(0, len(X), 25)
# pick some data randomly; an array of integer indices

coord, params = ml.transforms.rescale( W[:, 0:2] )
# normalize scale of "W" locations
plt.figure()
for i in idx:
    # compute where to place image (scaled W values) & size
    loc = (coord[i, 0],coord[i, 0] + 0.5, coord[i, 1], coord[i, 1] + 0.5)
    img = np.reshape(X[i,:], (24, 24) ) # reshape to square
    plt.imshow( img.T , cmap = "gray", extent = loc) # draw each image
    plt.axis((-2,2,-2,2)) # set axis to a reasonable scale
plt.show()
```



Problem 3

For this assignemnt, I mostly used the lecture notes, discussion and Piazza. Piazza and lecture notes were two really helpful

resources and I asked questions on Piazza and used my friends help as well

Problem 4

I submitted the course evaluation on EEE-Legacy