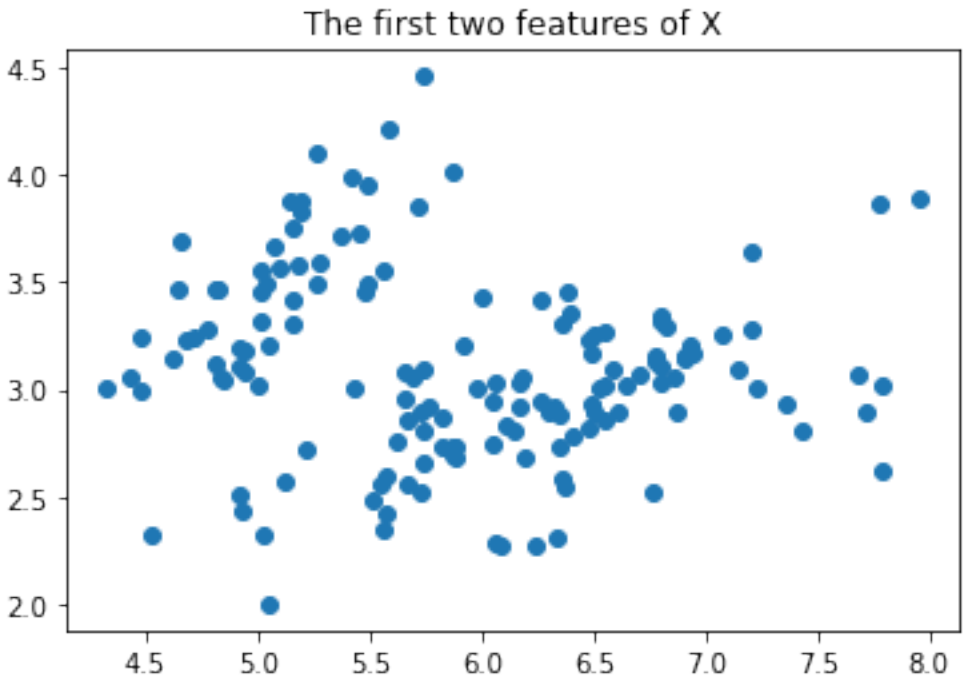
273A hw5

EnYu Huang

March 2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
#Problem 1.1
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X = iris[:,0:2]
plt.scatter(X[:,0], X[:,1])
plt.title('The first two features of X')
plt.show()
```

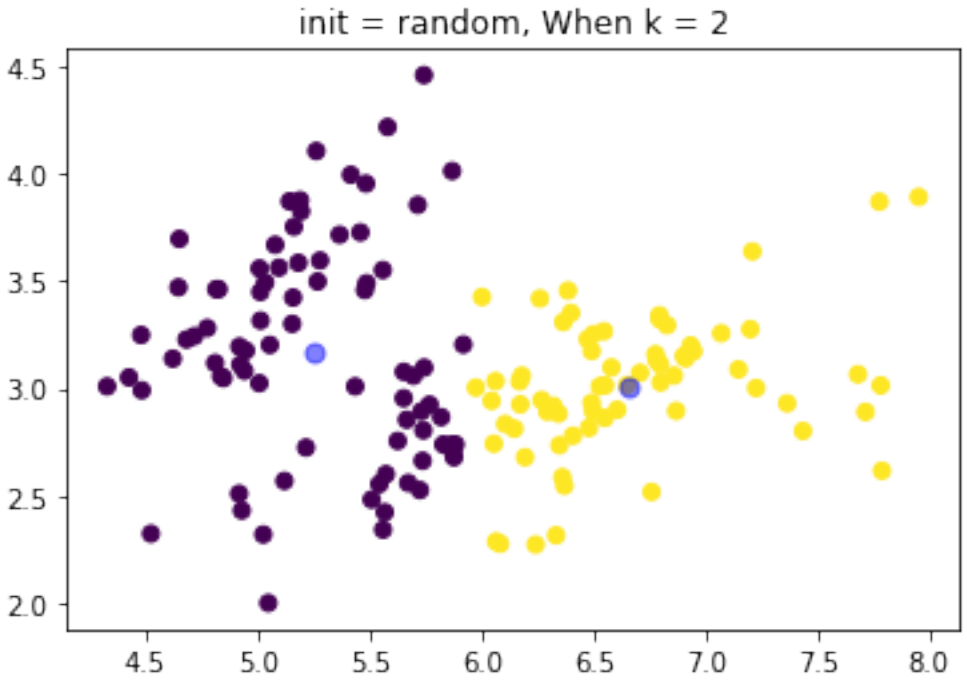


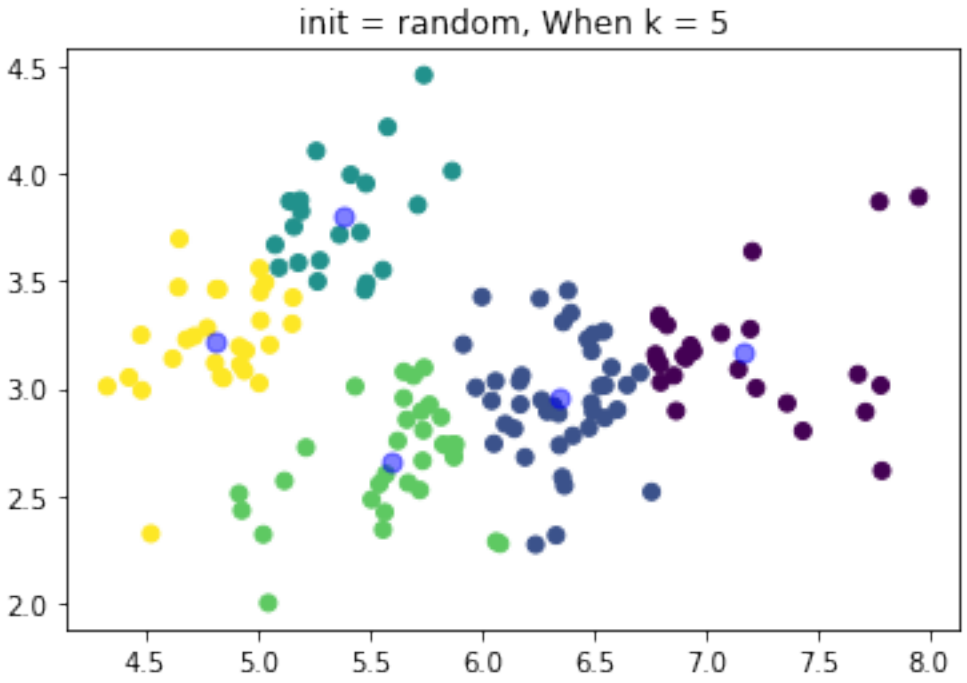
I think there is one cluster for features in orange, two to three clusters for features in blue.

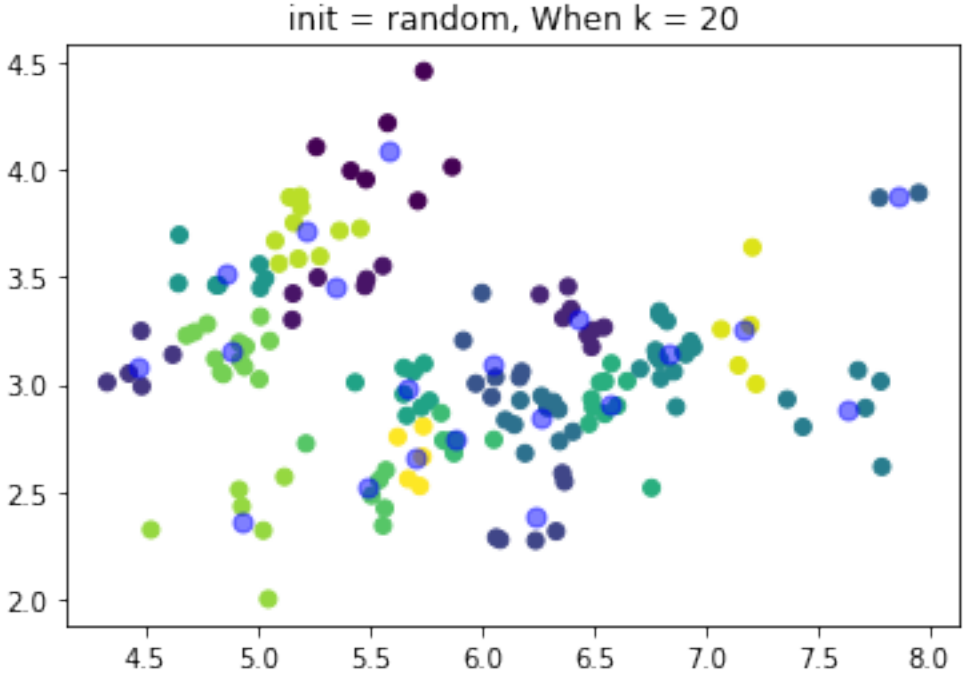
[2]:
```
from kneed import KneeLocator
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

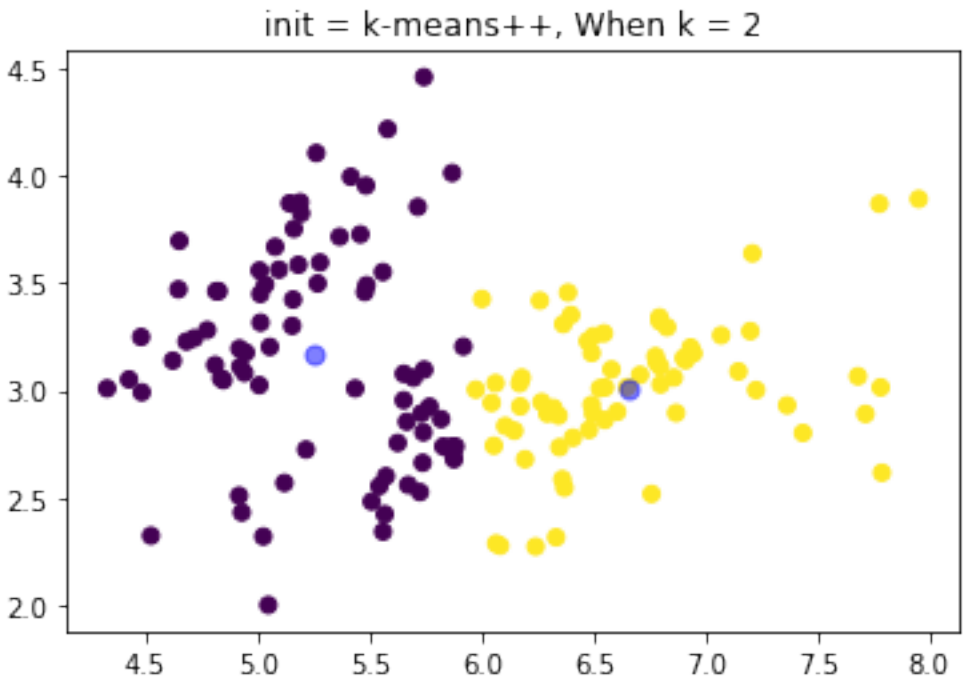
```
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn import datasets
```

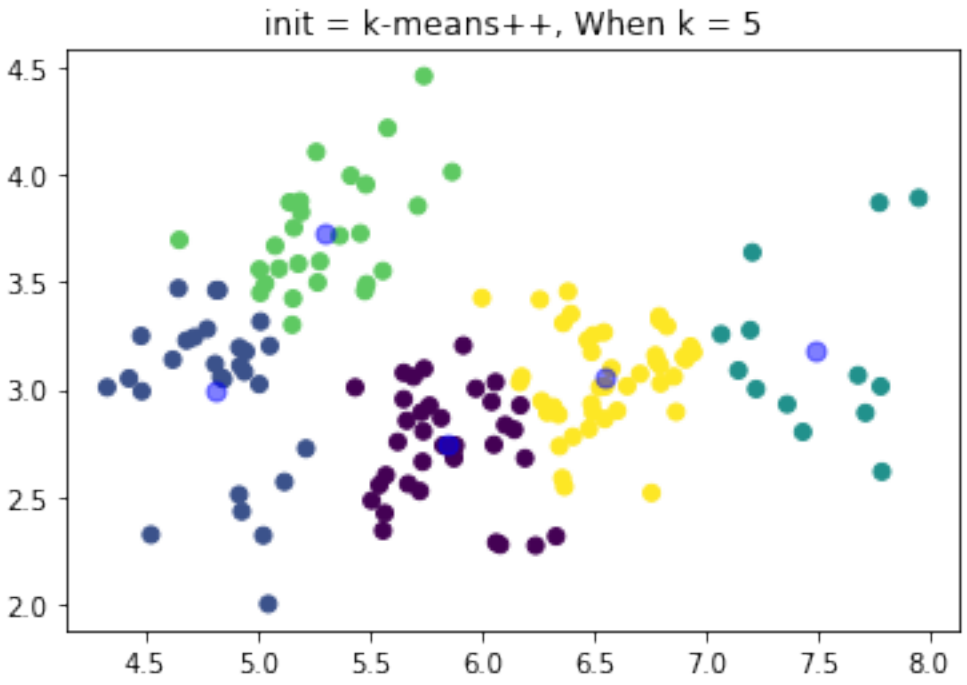
[3]:
```
#Problem 1.2
k = [2,5,20]
for i in k:
    kmeans = KMeans(init="random",n_clusters=i,n_init=10)
    kmeans.fit(X)
    y_kmeans = kmeans.predict(X)
    #ml.plotClassify2D(None,X,y_kmeans)
    plt.title("init = random, " + "When k = "+ str(i))
    plt.scatter(X[:, 0], X[:, 1], c=y_kmeans)
    centers = kmeans.cluster_centers_
    plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=50, alpha=0.5);
    plt.show()
for i in k:
    kmeans = KMeans(init="k-means++",n_clusters=i,n_init=10)
    kmeans.fit(X)
    y_kmeans = kmeans.predict(X)
    #ml.plotClassify2D(None,X,y_kmeans)
    plt.title("init = k-means++, " + "When k = "+ str(i))
    plt.scatter(X[:, 0], X[:, 1], c=y_kmeans)
    centers = kmeans.cluster_centers_
    plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=50, alpha=0.5);
    plt.show()
```

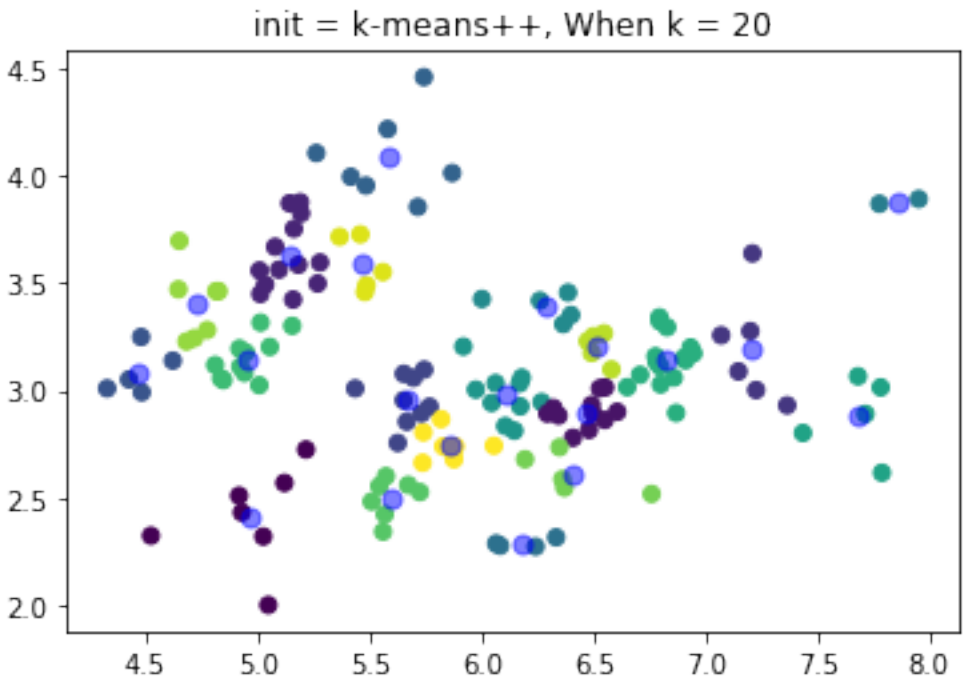



init = random, When k = 5
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

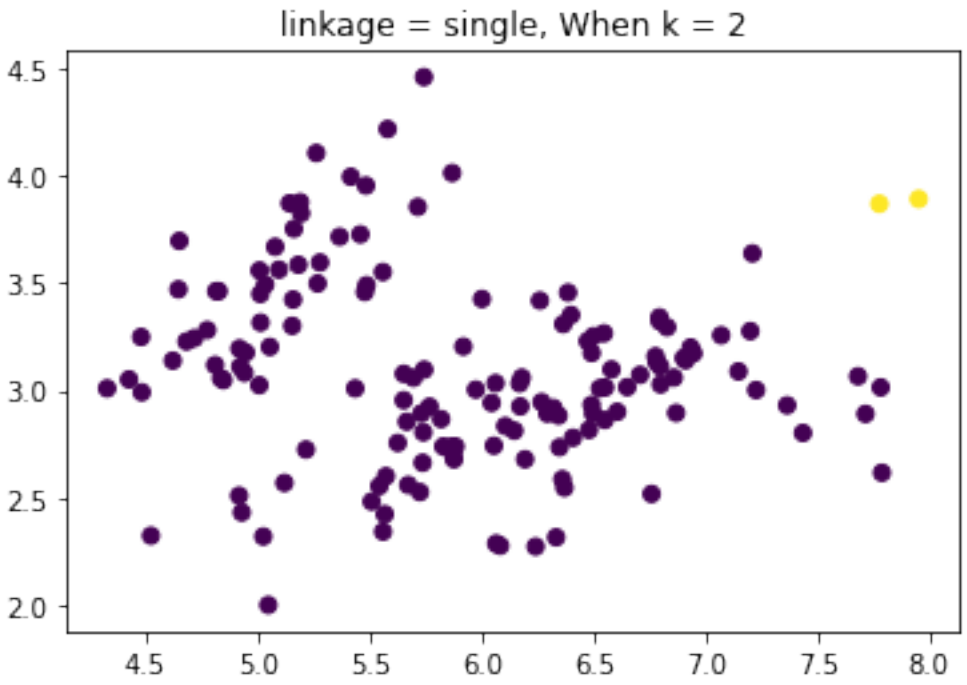

init = random, When k = 20
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

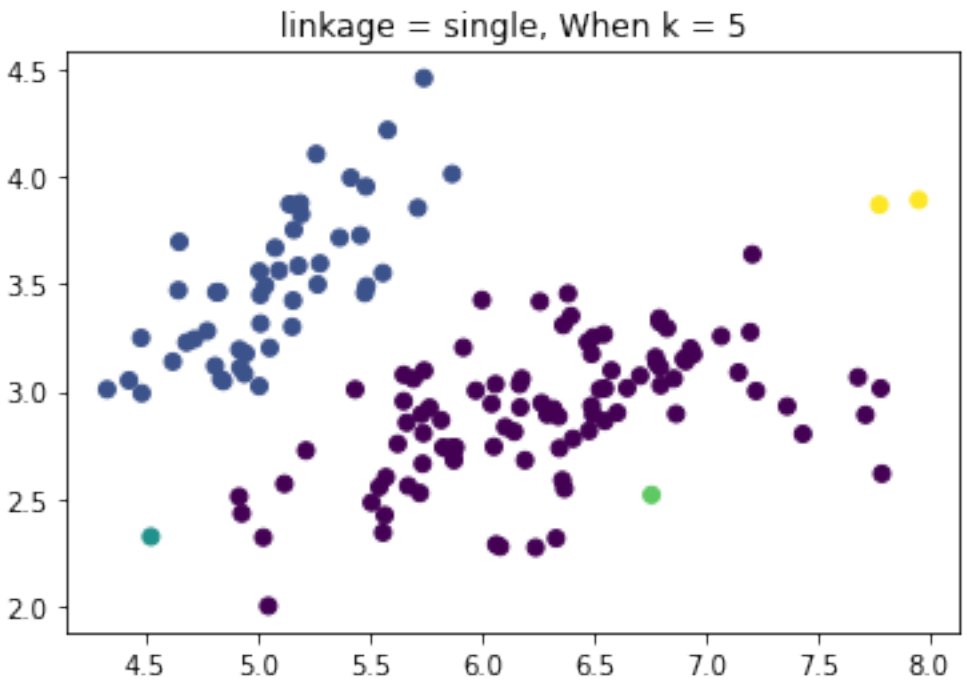

init = k-means++, When k = 2
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

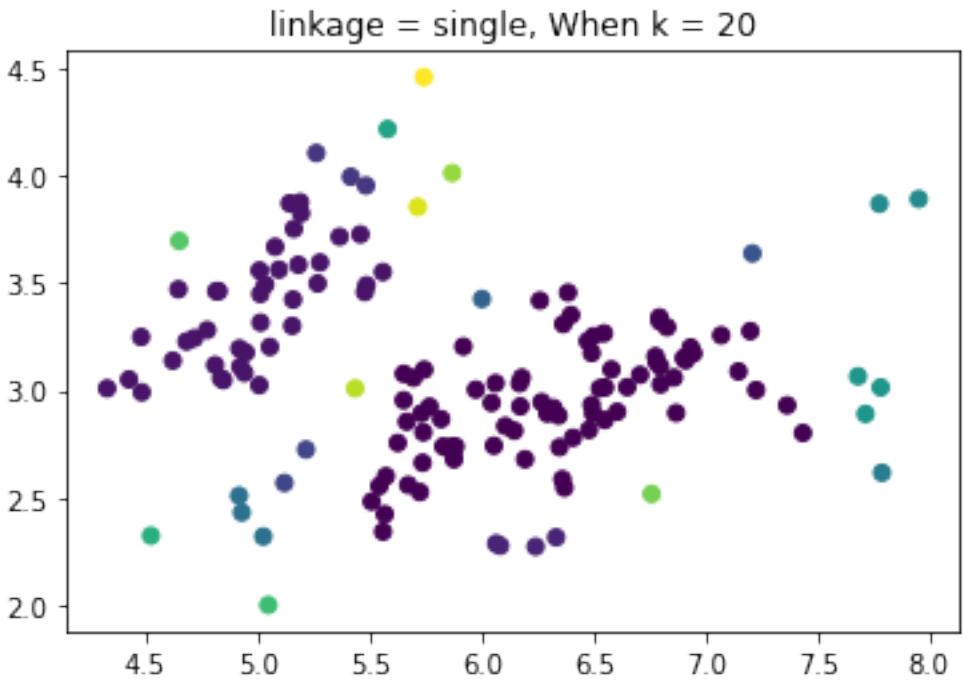

init = k-means++, When k = 5
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

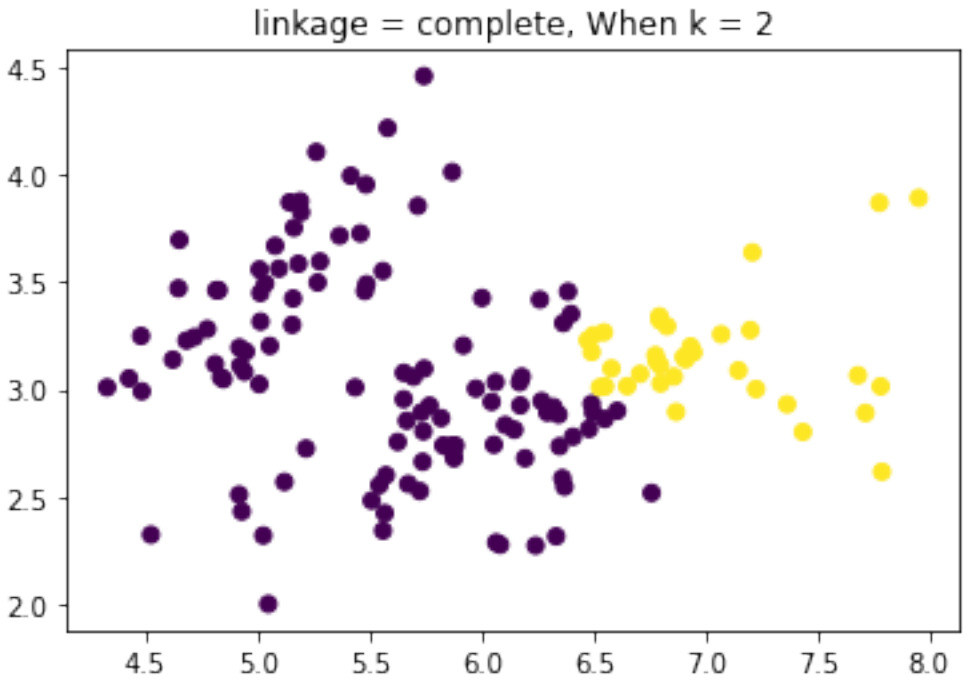


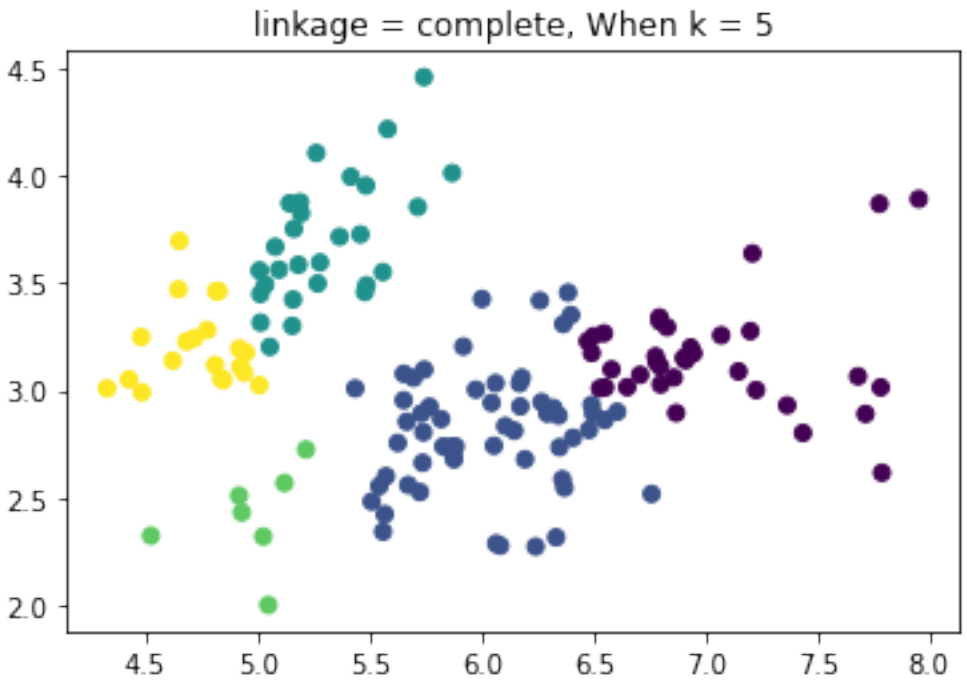
[4]:
```
#Problem 1.3
for i in k:
    model = AgglomerativeClustering(n_clusters = i, linkage = 'single')
    model.fit(X)
    labels = model.fit_predict(X)
    plt.scatter(X[:,0], X[:,1], c = labels)
    plt.title('linkage = single, ' + "When k = "+ str(i))
    plt.show()
for i in k:
    plt.title("When k = "+ str(i))
    model = AgglomerativeClustering(n_clusters = i, linkage = 'complete')
    model.fit(X)
    labels = model.fit_predict(X)
    plt.scatter(X[:,0], X[:,1], c = labels)
    plt.title('linkage = complete, ' + "When k = "+ str(i))
    plt.show()
```

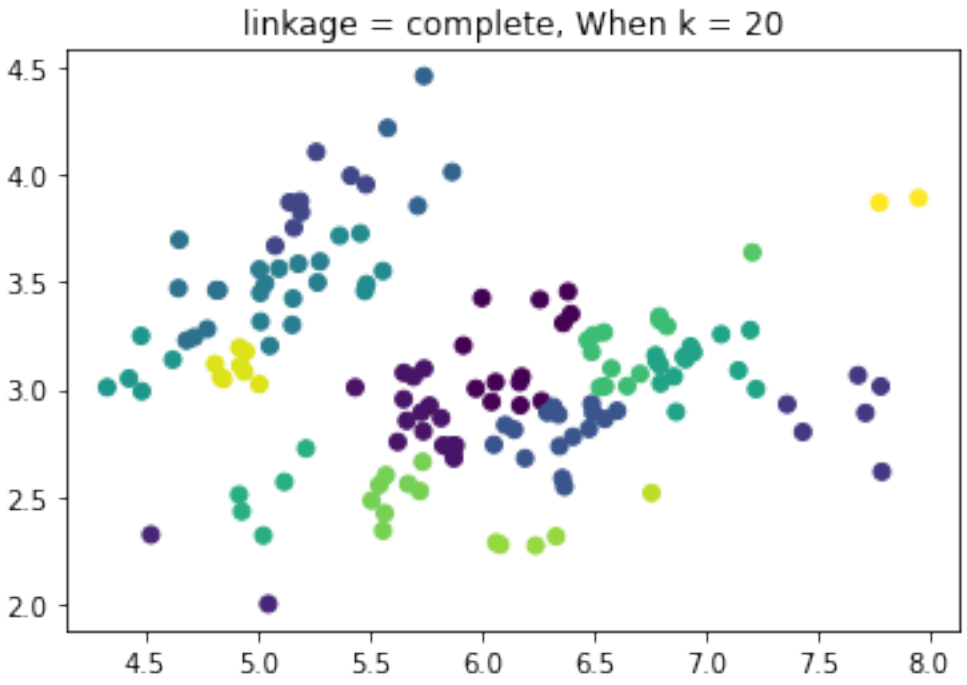

linkage = single, When k = 2
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


linkage = single, When k = 5
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


linkage = single, When k = 20
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


linkage = complete, When k = 2
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0





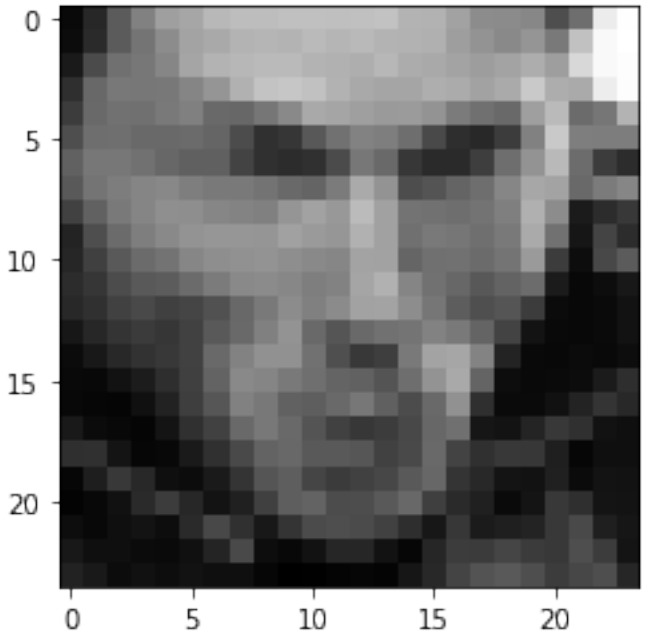
Problem 1.4 For agglomerative clustering, when linkage = complete, the output clusters are more close to k-means, however, whem linkage = single, the result is far from k-means. Moreover,

since initializations of k-means is "random", the results are different from previous one everytime.

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
from scipy import linalg
import random
```

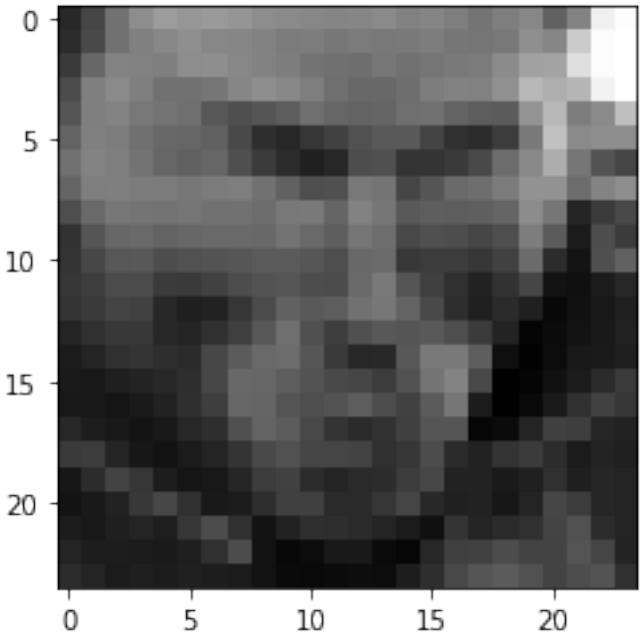
[6]:
```
X = np.genfromtxt("data/faces.txt", delimiter=None) # load face dataset
plt.figure()
# pick a data point i for display
img = np.reshape(X[i,:],(24,24)) # convert vectorized data to 24x24 imagepatches
plt.imshow( img.T , cmap="gray") # display image patch; you may have tosquint
```

[6]:
```
<matplotlib.image.AxesImage at 0x2505a008c40>
```



[7]:
```
#Problem 2.1
X_mean = np.mean(X, axis=0)
X = X - X_mean # Converting the data to zero mean
img = np.reshape(X[i,:],(24,24)) # convert vectorized data to 24x24 imagepatches
plt.imshow( img.T , cmap="gray") # display image patch; you may have tosquint
```

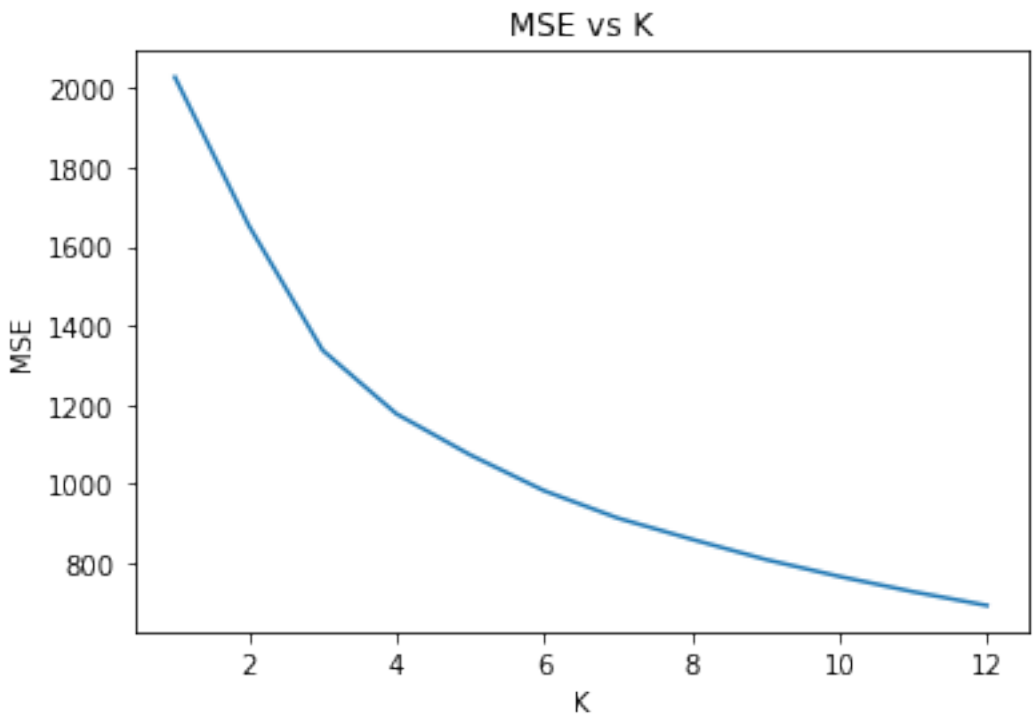
[7]:
```
<matplotlib.image.AxesImage at 0x25057199580>
```



[8]:
```
#Problem 2.2
U, s, Vhat = linalg.svd(X, full_matrices=False)
W = U.dot(np.diag(s)).dot(Vhat)
print("W = ",W.shape,"Vhat = ", Vhat.shape)
```

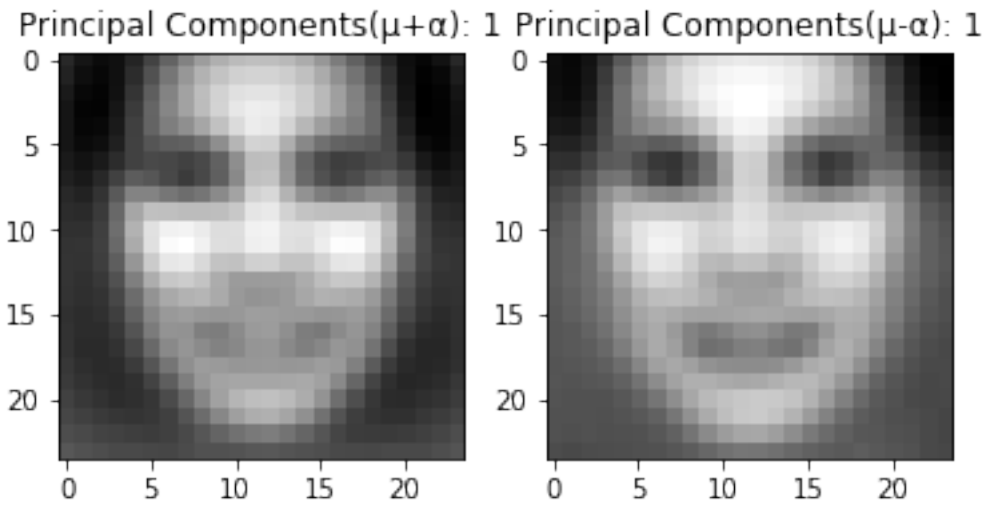
```
W =  (4916, 576) Vhat =  (576, 576)
```

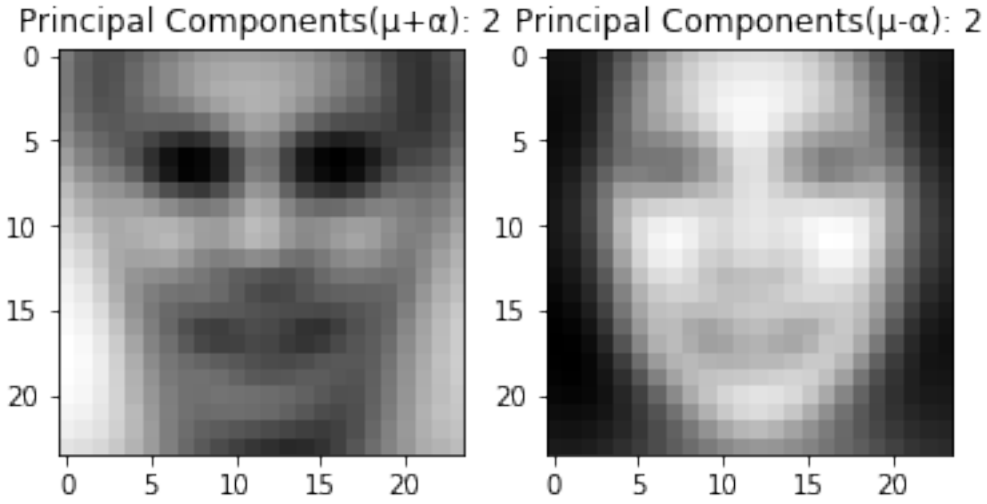
[9]:
```
#Problem 2.3
eigenvectors = range(1,13,1)
error = []
for k in eigenvectors:
    Xhat = U[:,0:k].dot(np.diag(s[0:k])).dot(Vhat[0:k,:])
    error.append(np.mean((W-Xhat) ** 2))
plt.plot(eigenvectors, error)
plt.xlabel('K')
plt.ylabel('MSE')
plt.title('MSE vs K')
plt.show()
```

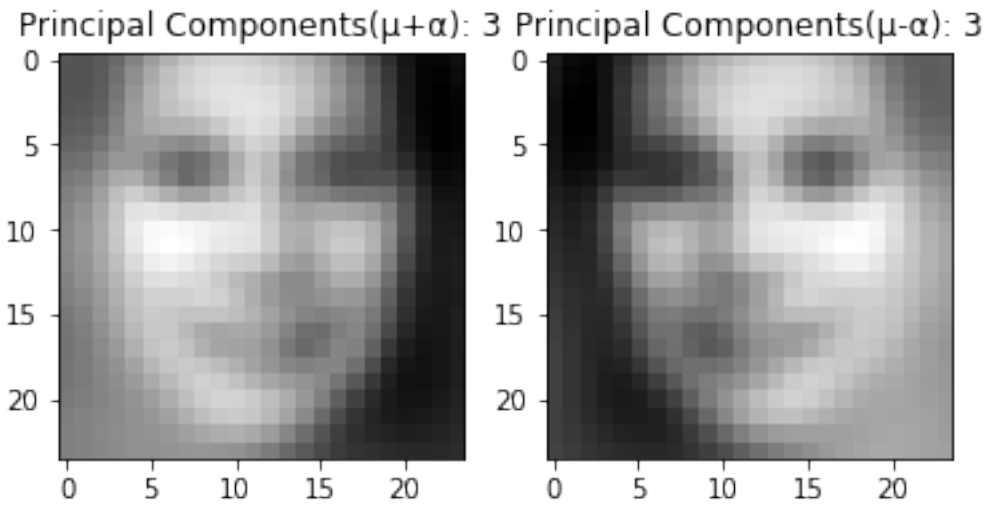


[10]:
```
#Problem 2.4
W = U.dot(np.diag(s)) # Calculating W as U.s for convenience
array = [0,1,2]
for j in array:
    alpha = 2*np.median(np.abs(W[:,j]))
    principal_image = X_mean + alpha*Vhat[j,:]
    img = np.reshape(principal_image, (24, 24))
    plt.subplot(1,2,1)
    plt.imshow(img.T, cmap='gray')
    plt.title('Principal Components(+): ' + str(j+1))

    principal_image = X_mean - alpha*Vhat[j,:]
    img = np.reshape(principal_image, (24, 24))
    plt.subplot(1,2,2)
    plt.imshow(img.T, cmap='gray')
    plt.title('Principal Components(-): ' + str(j+1))
    plt.show()
```


Principal Components(μ+α): 1
Principal Components(μ-α): 1
0
5
10
15
20


Principal Components(μ+α): 2
Principal Components(μ-α): 2
0
5
10
15
20



[11]:
```
#Problem 2.5
images=[]
for i in range(0,2,1):
    images.append(random.randrange(0, X.shape[0], 1))
for i in images:
    for k in [5, 10, 50, 100]:
        plt.subplot(1,2,1)
        img = np.reshape(X[i,:],(24,24))
        plt.title('Image number: ' + str(i))
        plt.imshow(img.T, cmap='gray')

        img = W[i:i+1,0:k].dot(Vhat[0:k,:])
        img = X_mean + img
        img = np.reshape(img, (24,24))
        plt.subplot(1,2,2)
        plt.title('Reconstructed Image with k = ' + str(k))
        plt.imshow(img.T, cmap='gray')
        plt.show()
```

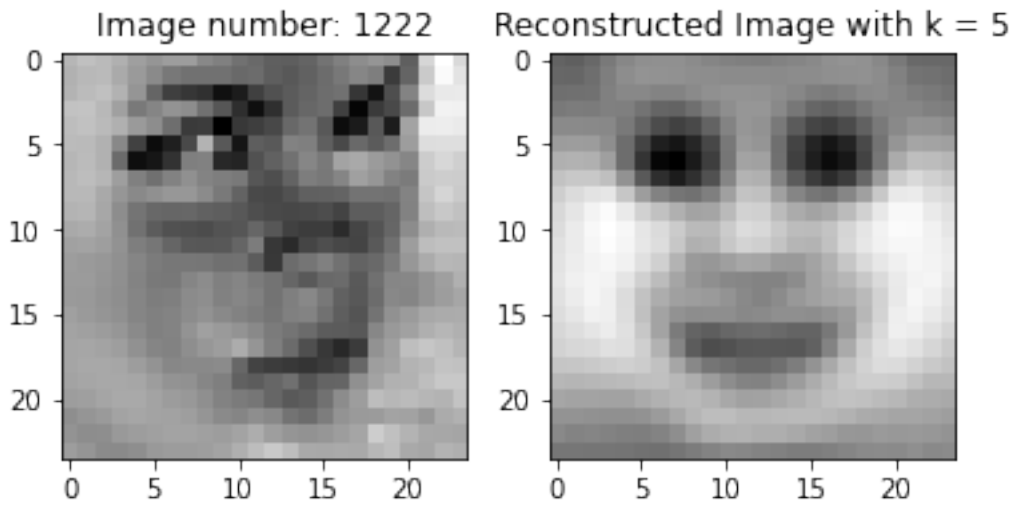


Image number: 1222
Reconstructed Image with k = 5
0
5
10
15
20

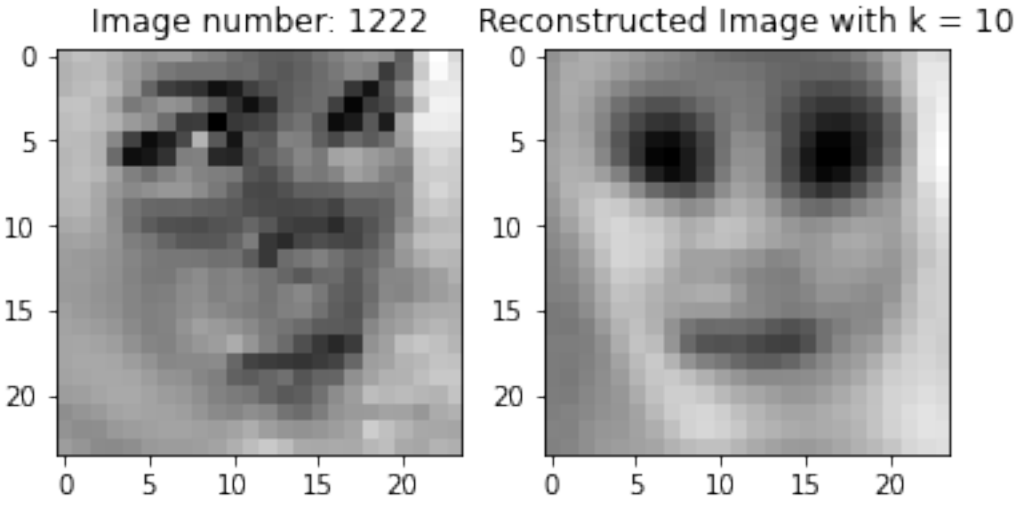


Image number: 1222
Reconstructed Image with k = 10
0
5
10
15
20

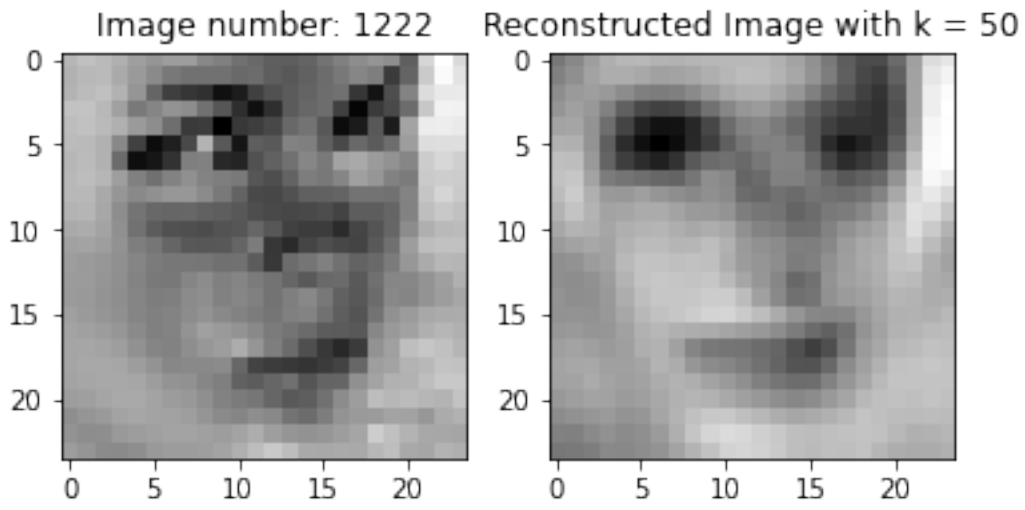


Image number: 1222
Reconstructed Image with k = 50
0
5
10
15
20

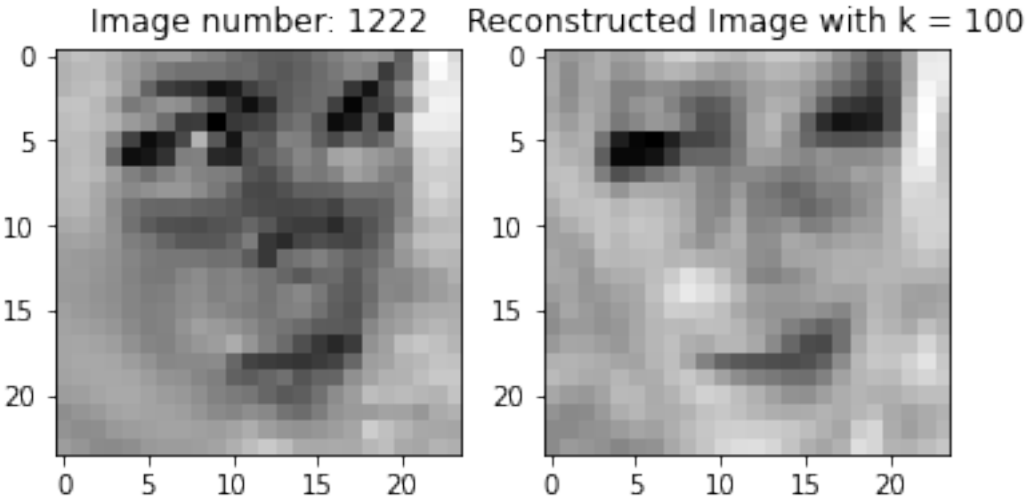


Image number: 1222
Reconstructed Image with k = 100
0
5
10
15
20

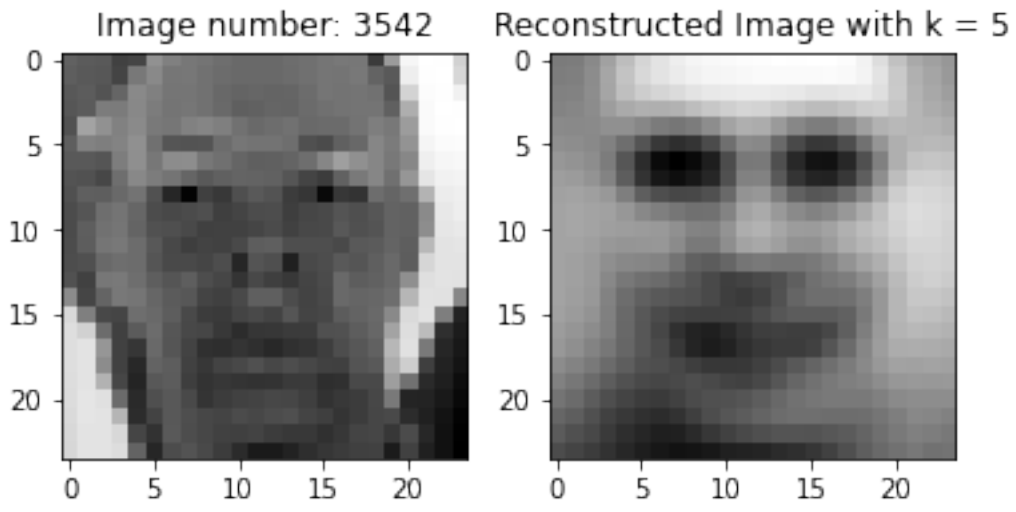


Image number: 3542
Reconstructed Image with k = 5
0
5
10
15
20

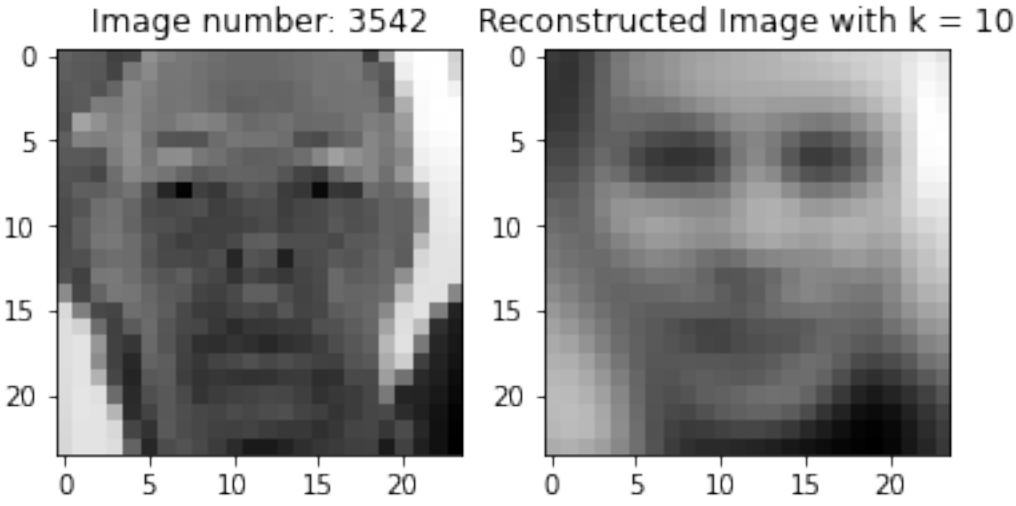
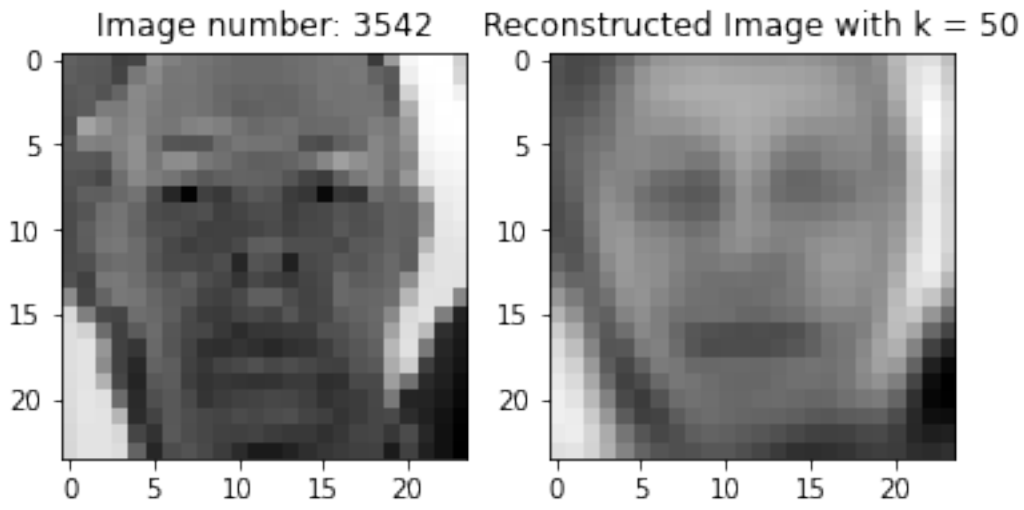
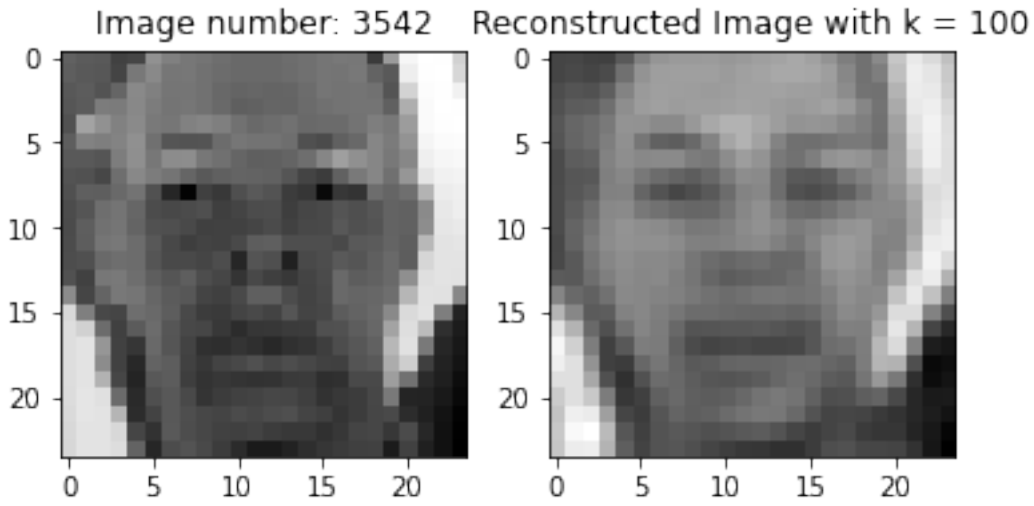


Image number: 3542
Reconstructed Image with k = 10
0
5
10
15
20

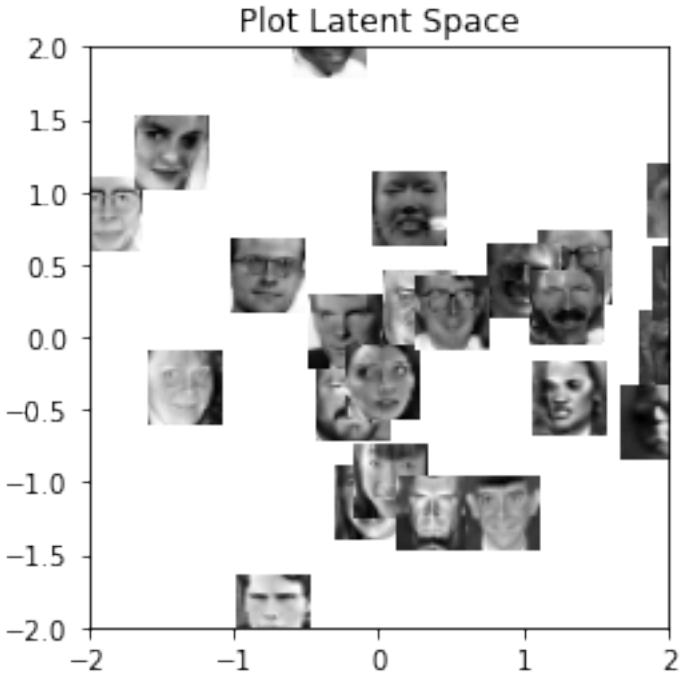




[12]:
```
#Problem 2.6
idx=[]
for i in range(0,25,1):
    idx.append(random.randrange(0, X.shape[0], 1))
print(idx)
coord,params = ml.transforms.rescale( W[:,0:2] ) # normalize scale of "W" locations
for i in idx:
    loc = (coord[i,0],coord[i,0]+0.5, coord[i,1],coord[i,1]+0.5) # where to place the image & size
    img = np.reshape( X[i,:], (24,24) )
```

```
    plt.imshow( img.T , cmap="gray", extent=loc ) # draw each image
    plt.title('Plot Latent Space')
    plt.axis((-2,2,-2,2))# set axis to a reasonable scale
plt.show()
```

```
[4620, 2388, 1839, 4460, 3961, 2235, 3211, 3756, 345, 3870, 1509, 1072, 4295,
3798, 4833, 4270, 1367, 4617, 3286, 2879, 92, 220, 4775, 96, 2737]
```



1 Statement of Collaboration

I finished this assignment by myself without discussing any specfic solution with others.