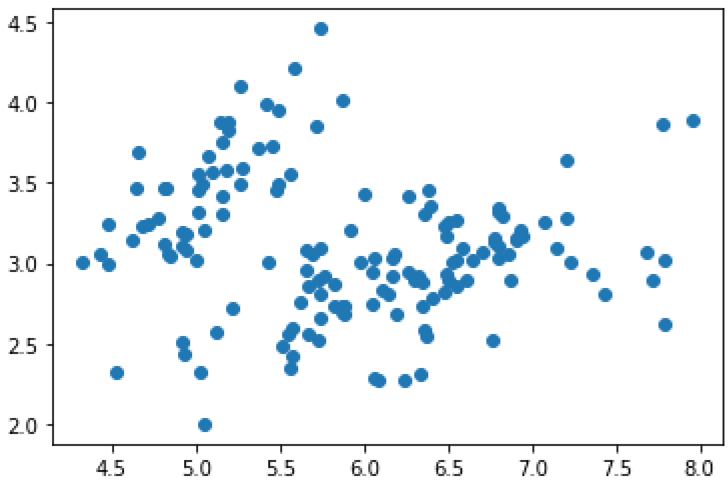
In [1]:
```
import numpy as np
import mltools as ml
import mltools.transforms
%matplotlib inline
import matplotlib.pyplot as plt
import random
import scipy.linalg
```

1: Clustering

1.1:

Loaded the Iris data with its first 2 features and plotted the data.
I believe there are 2 clusters, one at the top left corner that are grouped together, and the other being the rest.

In [2]:
```
iris = np.genfromtxt("data/iris.txt",delimiter=None)

X = iris[:,0:2]

xs = [i[0] for i in X]
ys = [i[1] for i in X]

plt.scatter(xs,ys)
plt.show()
```

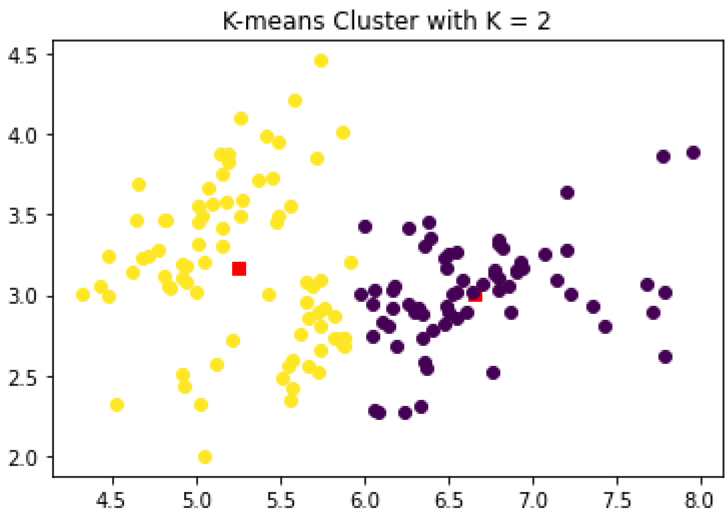


1.2:

Ran K-means clustering for k = 2, 5, and 20. The red boxes in the plots represent the cluster centers.

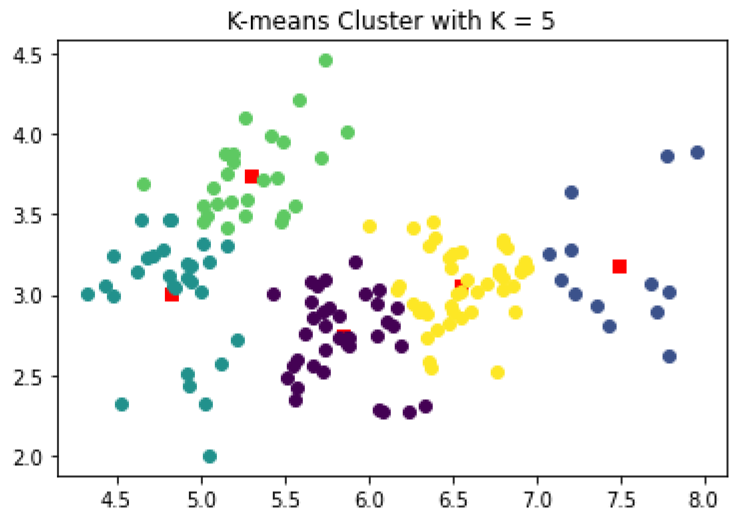
In [3]:
```
inits_2 = ["random", "farthest", "k++"]
for _ in range(2):
    random_center = []
    for _ in range(2):
        random_center.append(list(random.choice(X)))
    inits_2.append(np.array(random_center))

best_2_sum = np.inf
best_2_cluster = None
best_2_center = None
for i in range(len(inits_2)):
    k2_cluster, k2_center, k2_sum = ml.cluster.kmeans(X,2, init = inits_2[i])
    if k2_sum < best_2_sum:
        best_2_cluster, best_2_center, best_2_sum = k2_cluster, k2_center, k2_sum
```

```
    #plt.title('K-means Cluster with K = 2')
    #ml.plotClassify2D(None,X,k2_cluster)
    #center_k_x2s = [i[0] for i in k2_center]
    #center_k_y2s = [i[1] for i in k2_center]
    #plt.scatter(center_k_x2s, center_k_y2s, c = 'Red', marker = 's')
    #plt.show()

plt.title('K-means Cluster with K = 2')
ml.plotClassify2D(None,X,best_2_cluster)
best_2_center_xs = [i[0] for i in best_2_center]
best_2_center_ys = [i[1] for i in best_2_center]
plt.scatter(best_2_center_xs, best_2_center_ys, c = 'Red', marker = 's')
plt.show()
```

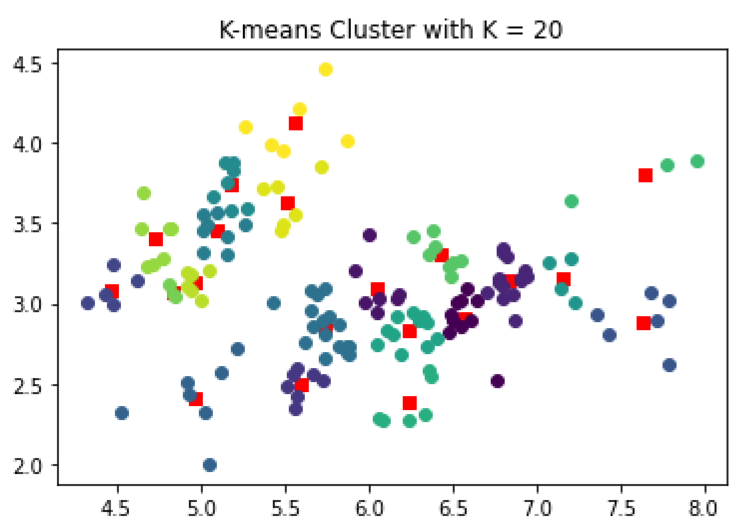


In [4]:

```
inits_5 = ["random", "farthest", "k++"]
for _ in range(2):
    random_center = []
    for _ in range(5):
        random_center.append(list(random.choice(X)))
    inits_5.append(np.array(random_center))

best_5_sum = np.inf
best_5_cluster = None
best_5_center = None
for i in range(len(inits_5)):
    k5_cluster, k5_center, k5_sum = ml.cluster.kmeans(X,5, init = inits_5[i])
    if k5_sum < best_5_sum:
        best_5_cluster, best_5_center, best_5_sum = k5_cluster, k5_center, k5_sum
    #plt.title('K-means Cluster with K = 5')
    #ml.plotClassify2D(None,X,k5_cluster)
    #center_k_x5s = [i[0] for i in k5_center]
    #center_k_y5s = [i[1] for i in k5_center]
    #plt.scatter(center_k_x5s, center_k_y5s, c = 'Red', marker = 's')
    #plt.show()

plt.title('K-means Cluster with K = 5')
ml.plotClassify2D(None,X,best_5_cluster)
best_5_center_xs = [i[0] for i in best_5_center]
best_5_center_ys = [i[1] for i in best_5_center]
plt.scatter(best_5_center_xs, best_5_center_ys, c = 'Red', marker = 's')
plt.show()
```



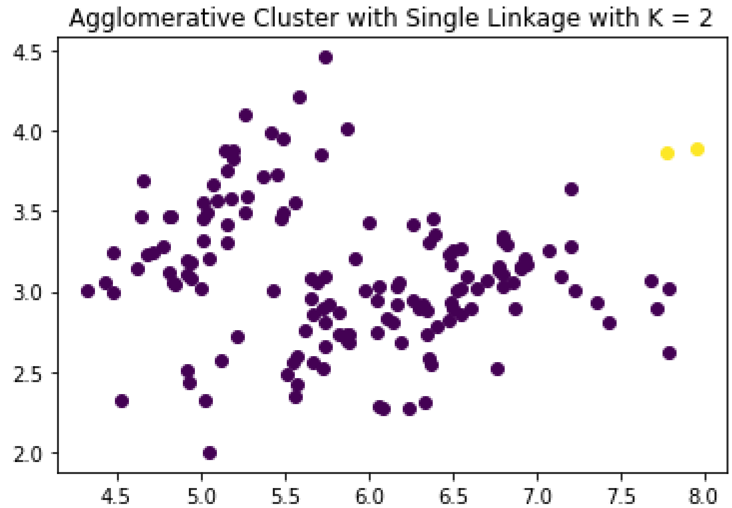
In [5]:
```
inits_20 = ["random", "farthest", "k++"]
for _ in range(2):
    random_center = []
    for _ in range(20):
        random_center.append(list(random.choice(X)))
    inits_20.append(np.array(random_center))

best_20_sum = np.inf
best_20_cluster = None
best_20_center = None
for i in range(len(inits_20)):
    k20_cluster, k20_center, k20_sum = ml.cluster.kmeans(X,20, init = inits_20[i])
    if k20_sum < best_20_sum:
        best_20_cluster, best_20_center, best_20_sum = k20_cluster, k20_center, k20_sum
    #plt.title('K-means Cluster with K = 20')
    #ml.plotClassify2D(None,X,k20_cluster)
    #center_k_x20s = [i[0] for i in k20_center]
    #center_k_y20s = [i[1] for i in k20_center]
    #plt.scatter(center_k_x20s, center_k_y20s, c = 'Red', marker = 's')
    #plt.show()

plt.title('K-means Cluster with K = 20')
ml.plotClassify2D(None,X,best_20_cluster)
best_20_center_xs = [i[0] for i in best_20_center]
best_20_center_ys = [i[1] for i in best_20_center]
plt.scatter(best_20_center_xs, best_20_center_ys, c = 'Red', marker = 's')
plt.show()
```

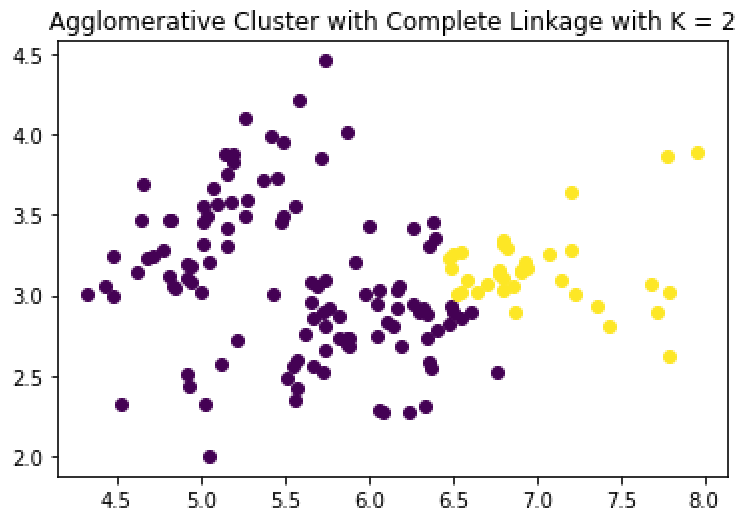


1.3:

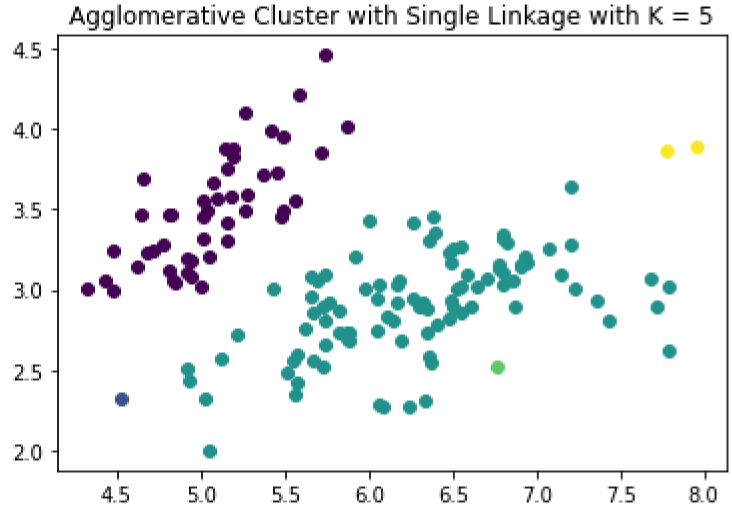
Ran Agglomerative clustering for k = 2, 5, and 20 and for single and complete linkages.

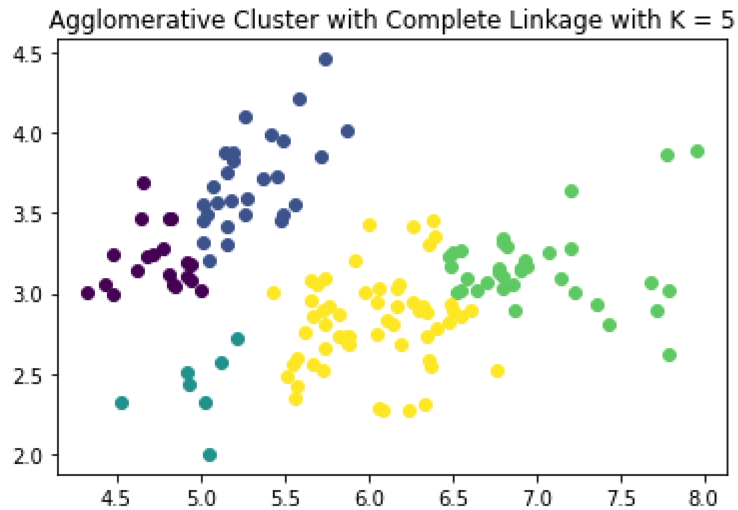
In [6]:
```
a2_single_cluster, a2_single_join = ml.cluster.agglomerative(X,2,'min')
plt.title('Agglomerative Cluster with Single Linkage with K = 2')
ml.plotClassify2D(None,X,a2_single_cluster)
plt.show()

a2_complete_cluster, a2_complete_join = ml.cluster.agglomerative(X,2,'max')
plt.title('Agglomerative Cluster with Complete Linkage with K = 2')
ml.plotClassify2D(None,X,a2_complete_cluster)
plt.show()
```





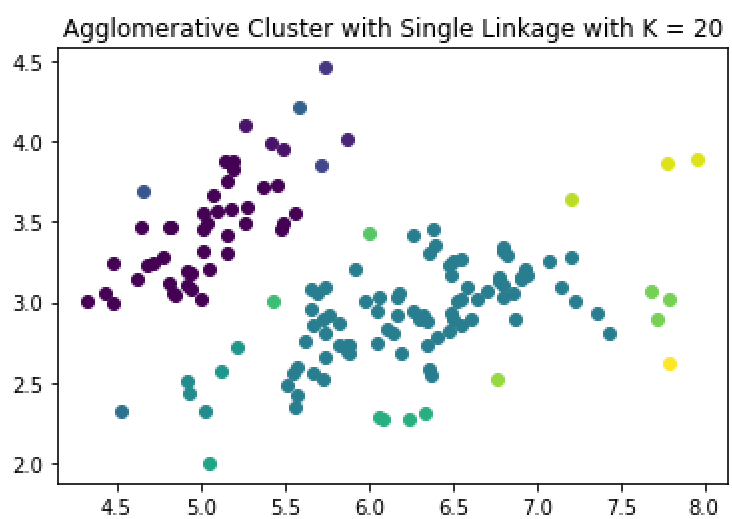
In [7]:
```
a5_single_cluster, a5_single_join = ml.cluster.agglomerative(X,5,'min')
plt.title('Agglomerative Cluster with Single Linkage with K = 5')
ml.plotClassify2D(None,X,a5_single_cluster)
plt.show()

a5_complete_cluster, a5_complete_join = ml.cluster.agglomerative(X,5,'max')
plt.title('Agglomerative Cluster with Complete Linkage with K = 5')
ml.plotClassify2D(None,X,a5_complete_cluster)
plt.show()
```

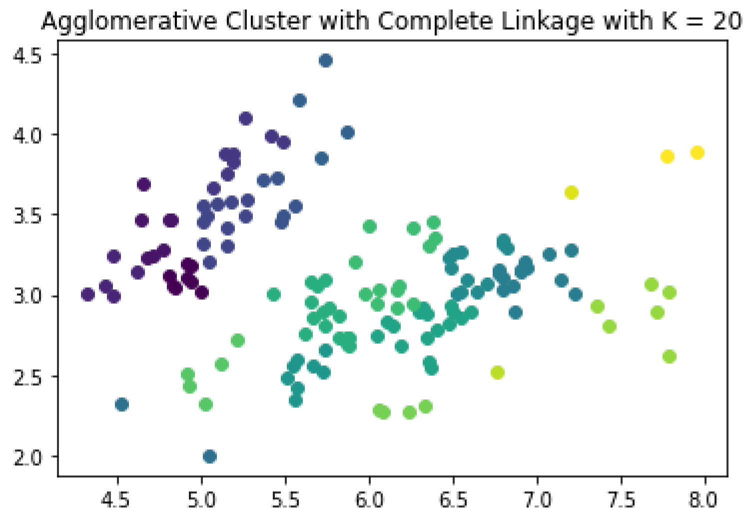




In [8]:

```
a20_single_cluster, a20_single_join = ml.cluster.agglomerative(X,20,'min')
plt.title('Agglomerative Cluster with Single Linkage with K = 20')
ml.plotClassify2D(None,X,a20_single_cluster)
plt.show()

a20_complete_cluster, a20_complete_join = ml.cluster.agglomerative(X,20,'max')
plt.title('Agglomerative Cluster with Complete Linkage with K = 20')
ml.plotClassify2D(None,X,a20_complete_cluster)
plt.show()
```

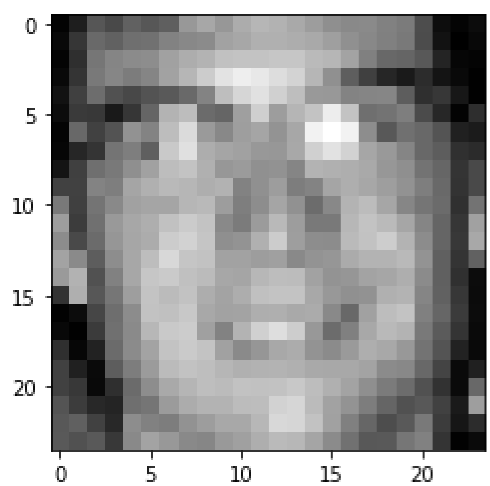




1.4:
Similarities: Both K-means and Agglomerative clusterings had similar areas be different clusterings.
Differences: Even though K-means and Agglomerative clusterings had similar areas be different clusterings, the sizes of those clusters were very different, the most noticeable being with K = 2. The region with K-means were more of an even split, but with Agglomerative, the single linkage had a vry cluster on the right, and the compelte linkage has a cluster about the size between single linkage and K-means. With the K-means the clusterings are more clear compared to Agglomerative clusterings becasue of the sizes, K-means had a more even distribution. Especially in the agglomerative clusterings for K = 5, 20 there were more small clusterings 1 or 2 points compared to K-means.

2: EigenFaces

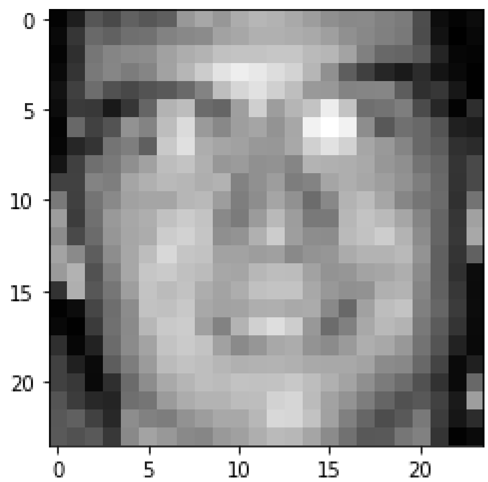
In [9]:
```
X = np.genfromtxt("data/faces.txt", delimiter=None) # load face dataset
plt.figure()
# pick a data point i for display
img = np.reshape(X[2002,:],(24,24)) # convert vectorized data to 24x24 image patches
plt.imshow( img.T , cmap="gray") # display image patch; you may have to squint
plt.show()
```



2.1:
Plot the mean face after subtracting mean from face images to get data zero-mean

In [10]:
```
mean = np.mean(X)
```

```
X_0 = X - mean
img_0 = np.reshape(X_0[2002,:],(24,24)) # convert vectorized data to 24x24 image patches
plt.imshow( img_0.T , cmap="gray") # display image patch; you may have to squint
plt.show()
```



2.2:

Take SVD of data to compute X0 = U *diag(S)* Vh

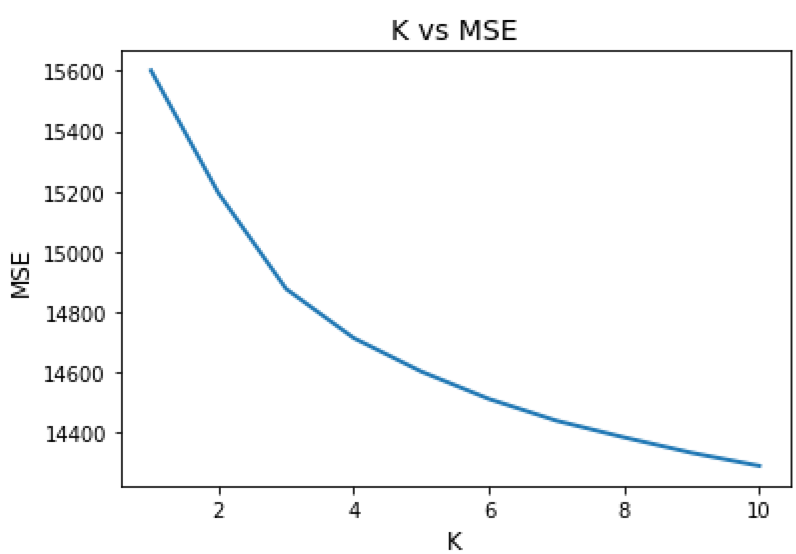
In [11]:
```
U, S, V = scipy.linalg.svd(X, full_matrices=False)
W = U.dot(np.diag(S))
print(W.shape)
print(V.shape)
```

```
(4916, 576)
(576, 576)
```

2.3:

Mean Square Errors for K's from 1 to 10 inclusive

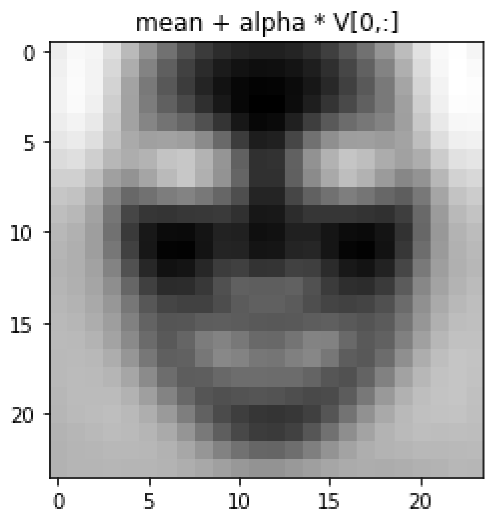
In [12]:
```
MSE = []
for i in range(1,11):
    X_0_hat = W[:,:i].dot(V[: i,:])
    MSE.append(np.mean((X_0-X_0_hat)**2))
plt.plot(range(1,11), MSE, linewidth=2)
plt.title("K vs MSE", fontsize=14)
plt.xlabel("K", fontsize=12)
plt.ylabel("MSE", fontsize=12)
plt.show()
```

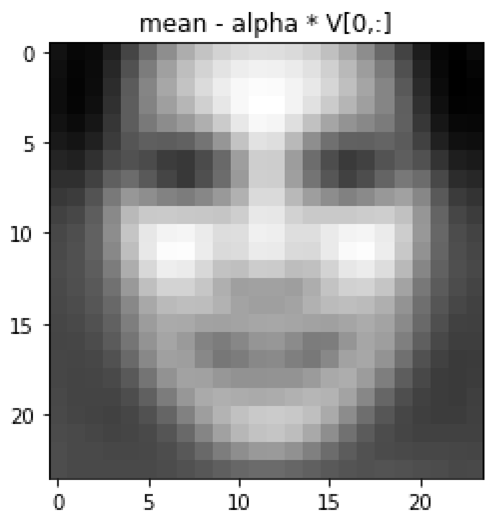


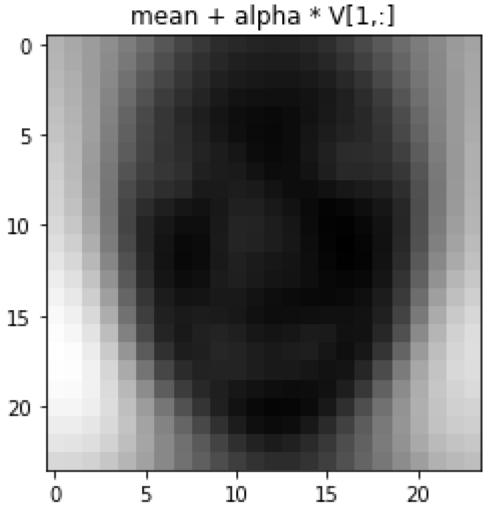
2.4:

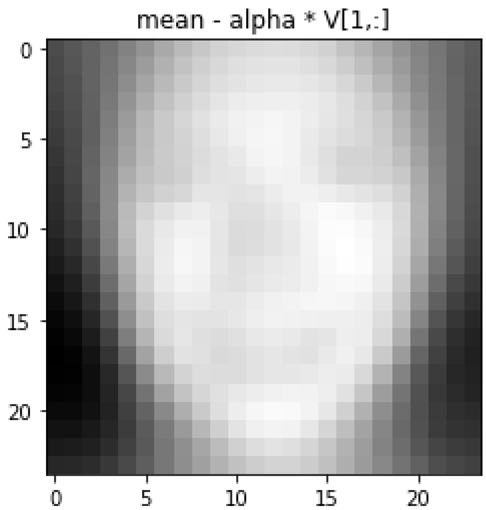
First three principal directions of the data

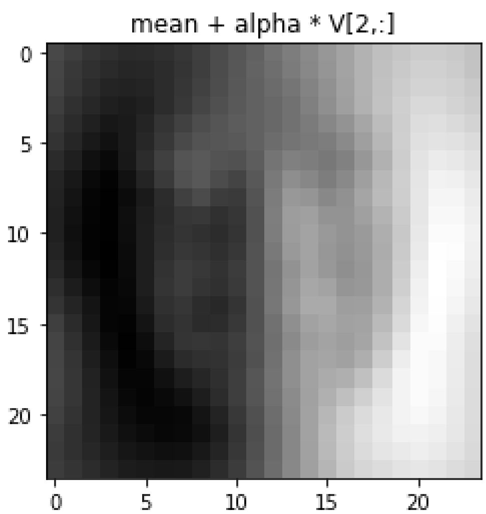
In [13]:
```
for i in range(3):
    alpha = 2 * np.median(np.abs(W[:,i]))

    pd_1 = mean + alpha * V[i,:]
    img_1 = np.reshape(pd_1,(24,24))
    title_1 = f'mean + alpha * V[{i},:]'.format(i)
    plt.title(title_1)
    plt.imshow( img_1.T , cmap="gray")
    plt.show()

    pd_2 = mean - alpha * V[i,:]
    img_2 = np.reshape(pd_2,(24,24))
    title_2 = f'mean - alpha * V[{i},:]'.format(i)
    plt.title(title_2)
    plt.imshow( img_2.T , cmap="gray")
    plt.show()
```



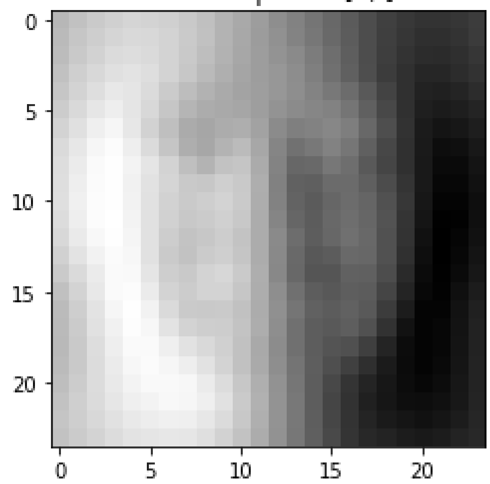



mean + alpha * V[1,:]
0
5
10
15
20
0
5
10
15
20


mean - alpha * V[1,:]
0
5
10
15
20
0
5
10
15
20


mean + alpha * V[2,:]
0
5
10
15
20
0
5
10
15
20

mean - alpha * V[2,:]

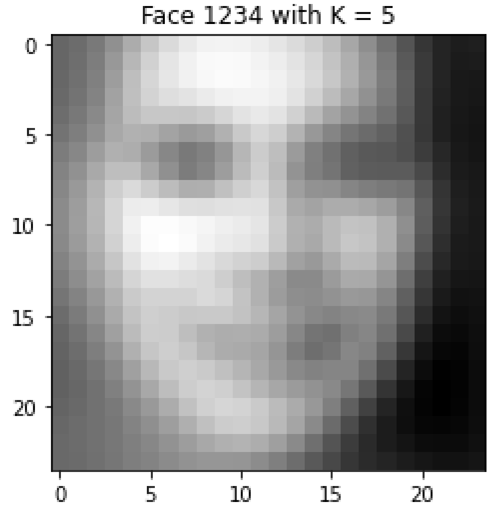


2.5:

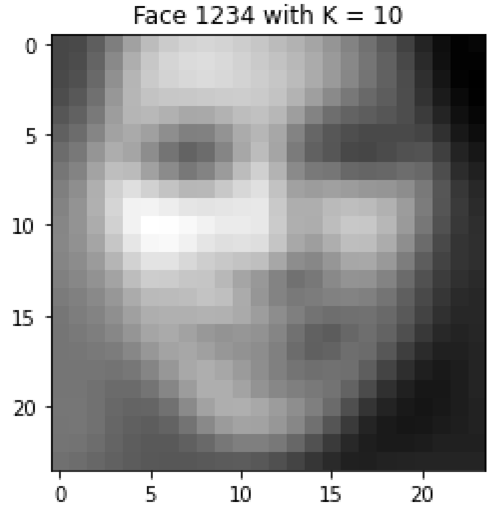
Face 1234 and 4321 and their first 5, 10, 50, 100 principal directions

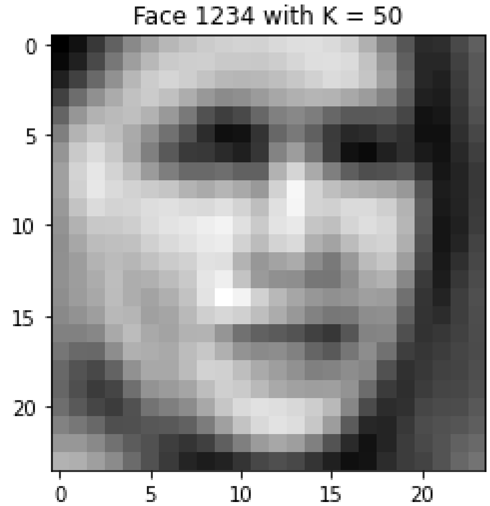
In [14]:
```
for k in [5,10,50,100]:
    face_1 = W[1234,:k].dot(V[: k,:])
    img_1 = np.reshape(face_1,(24,24))
    title_1 = f'Face 1234 with K = {k}'.format(k)
    plt.title(title_1)
    plt.imshow( img_1.T , cmap="gray")
    plt.show()
```

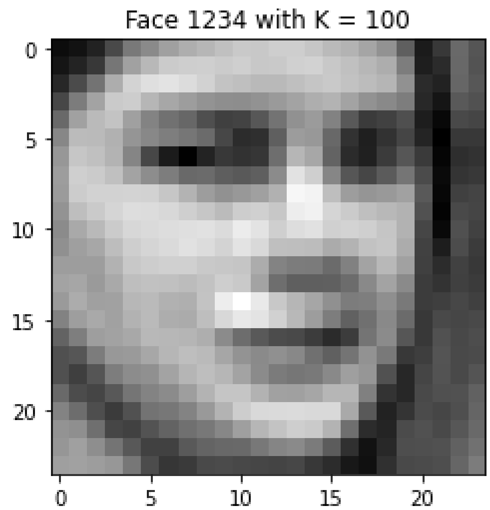
Face 1234 with K = 5



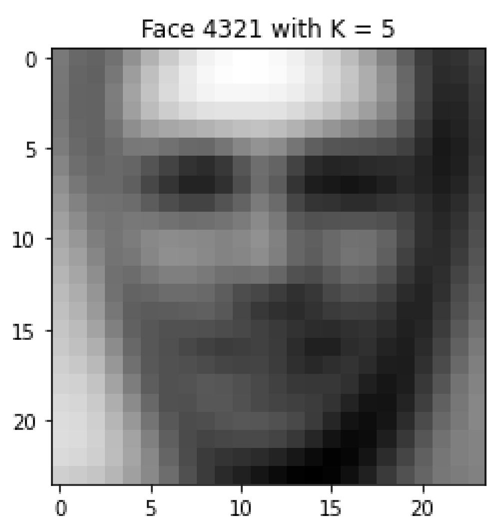
Face 1234 with K = 10

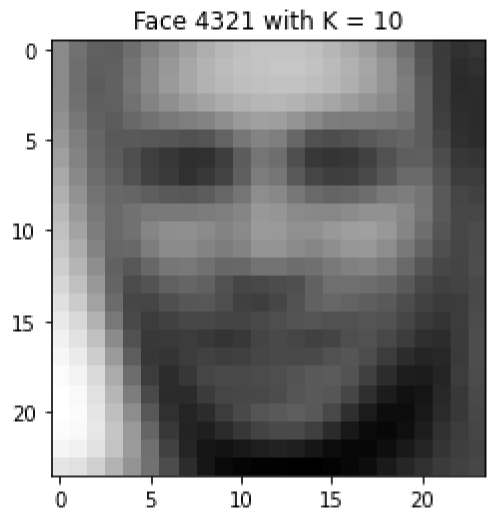


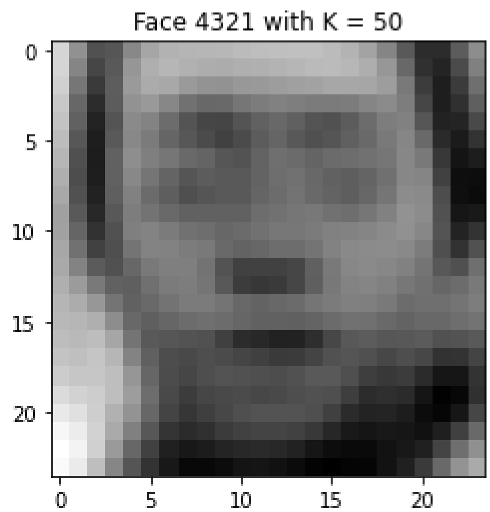


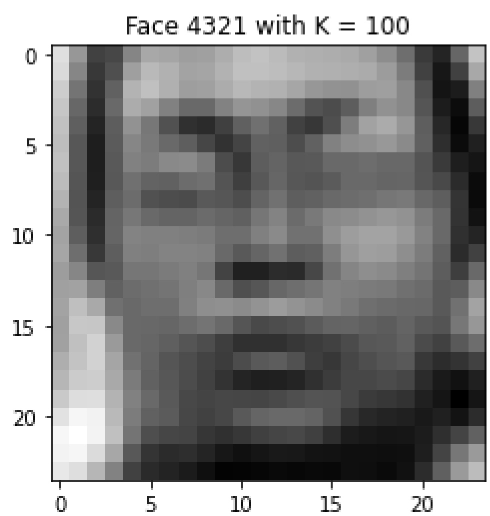


In [15]:
```
for k in [5,10,50,100]:
    face_1 = W[4321,:k].dot(V[: k,:])
    img_1 = np.reshape(face_1,(24,24))
    title_1 = f'Face 4321 with K = {k}'.format(k)
    plt.title(title_1)
    plt.imshow( img_1.T , cmap="gray")
    plt.show()
```









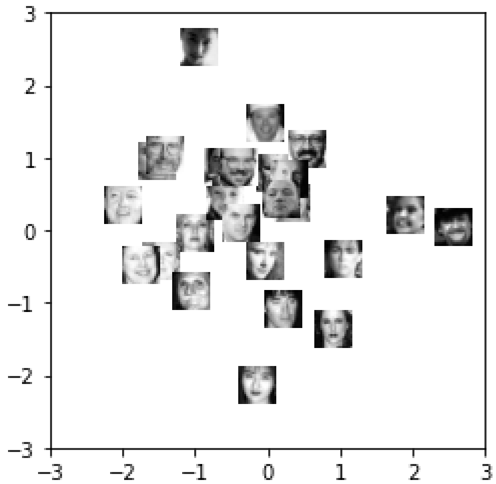
2.6:

25 random faces

In [16]:

```
idx = [] # pick some data (randomly or otherwise); an array of integer indices
while len(idx) < 25:
    face = random.randrange(0, 4916)
    if face not in idx:
        idx.append(face)

coord, params = ml.transforms.rescale( W[:,0:2] ) # normalize scale of "W" locations
plt.figure(); #plt.hold(True); # you may need this for pyplot
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( (-3,3,-3,3) ) # set axis to a reasonable scale
```



Statement of Collaboration

I, Andy Quoc Anh Dinh Tran, did this assignmnet by myself.