HW3-Q1

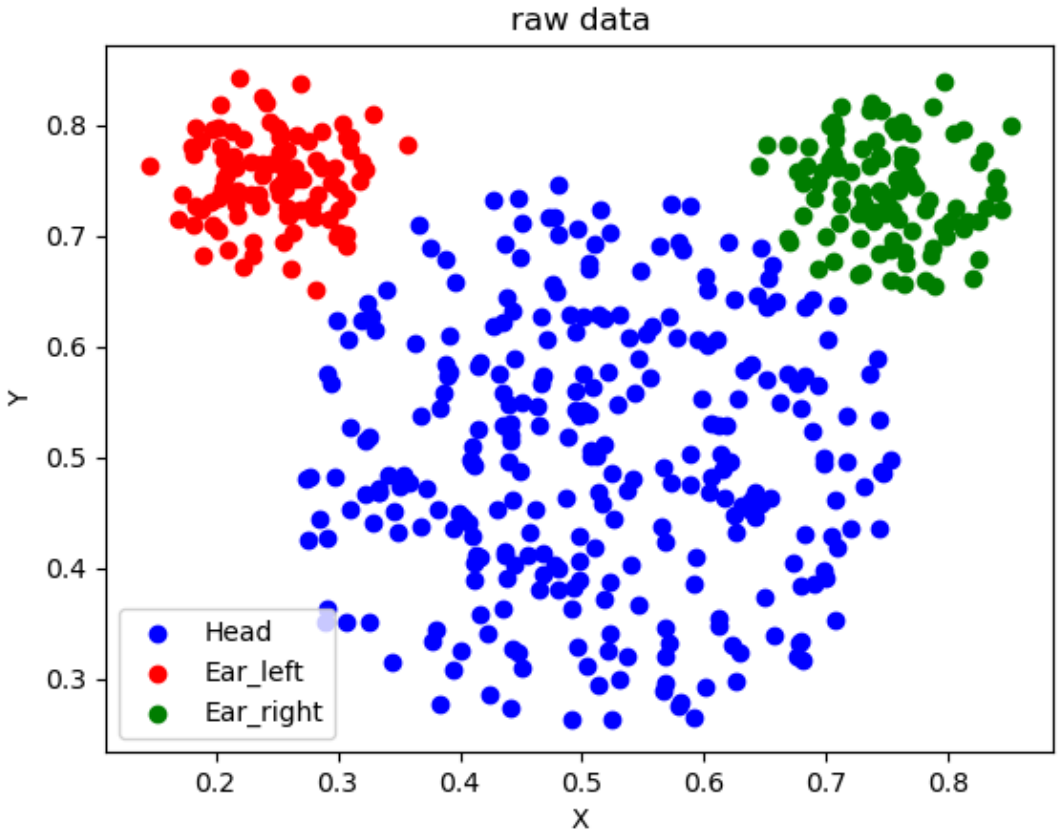
K-means

In [1]:
```
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.vq import kmeans2
```

In [2]:
```
#read data
with open('cluster.txt','r') as f:
    lines=f.readlines()
pattern=re.compile(r'^-?\d+(\.\d+)?\s+-?\d+(\.\d+)?\s+[a-zA-Z]+(_[a-zA-Z]+)*$'
new_lines=[i for i in lines if pattern.match(i.strip())]
with open('new_lines.txt','w') as f:
    f.writelines(new_lines)
pairs=pd.read_table('new_lines.txt',sep=' ',header=None,names=['x','y','label
pairs_3=pairs.to_numpy()
```

In [3]:

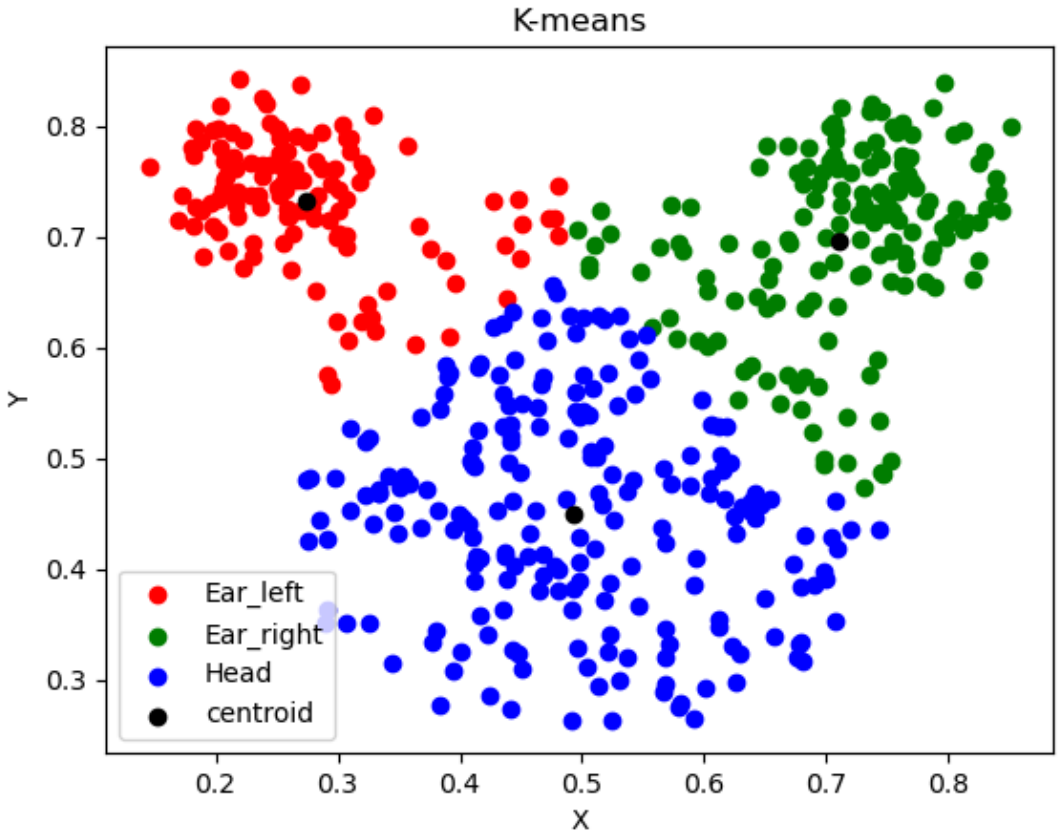
```
#draw raw data
plt.figure()
idx=np.where(pairs_3[:,2]=='Head')[0]
plt.scatter(pairs_3[idx,0],pairs_3[idx,1],color='blue')
idx=np.where(pairs_3[:,2]=='Ear_left')[0]
plt.scatter(pairs_3[idx,0],pairs_3[idx,1],color='red')
idx=np.where(pairs_3[:,2]=='Ear_right')[0]
plt.scatter(pairs_3[idx,0],pairs_3[idx,1],color='green')
plt.legend(['Head','Ear_left','Ear_right'])
plt.xlabel('X')
plt.ylabel('Y')
plt.title('raw data')
plt.show()
```



In [16]:

```
#kmean
K=3
pairs_2=pairs_3[:,:2]
pairs_2=pairs_2.astype(float)
centroid,label=kmeans2(pairs_2,K,minit='points')
sorted_centroid=centroid[centroid[:,0].argsort()]
sorted_centroid[[1,2]]=sorted_centroid[[2,1]]
sorted_label=np.zeros_like(label)
for i in range(len(label)):
    for j in range(3):
        if centroid[label[i],0]==sorted_centroid[j,0]:
            sorted_label[i]=j
#print(sorted_label)
```

In [17]:
```
color_dic={0:'red',1:'green',2:'blue'}
plt.figure()
for i in range(3):
    idx=np.where(sorted_label==i)[0]
    plt.scatter(pairs_2[idx,0],pairs_2[idx,1],color=color_dic[i])
plt.scatter(centroid[:,0],centroid[:,1],color='black')
plt.legend(['Ear_left','Ear_right','Head','centroid'])
plt.xlabel('X')
plt.ylabel('Y')
plt.title('K-means')
plt.show()
```



In [18]:
```
#confusion matrix
confusion_matrix = pd.crosstab(pairs_3[:,2],sorted_label,rownames=['Actual'],c
print("Confusion Matrix")
print("(0:Ear_left, 1:Ear_right, 2:Head)")
print(confusion_matrix)
```

```
Confusion Matrix
(0:Ear_left, 1:Ear_right, 2:Head)
Predicted    0    1    2
Actual
Ear_left   100    0    0
Ear_right    0  100    0
Head        25   54  211
```

GMM

In [7]:
```
def draw_label_scater(X,label,title):
    color_dic={0:'red',1:'green',2:'blue'}
    plt.figure()
    for i in range(3):
        idx=np.where(label==i)[0]
        plt.scatter(X[idx,0],X[idx,1],color=color_dic[i])
    plt.legend(['Ear_left','Ear_right','Head'])
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title(title)
    plt.show()
```

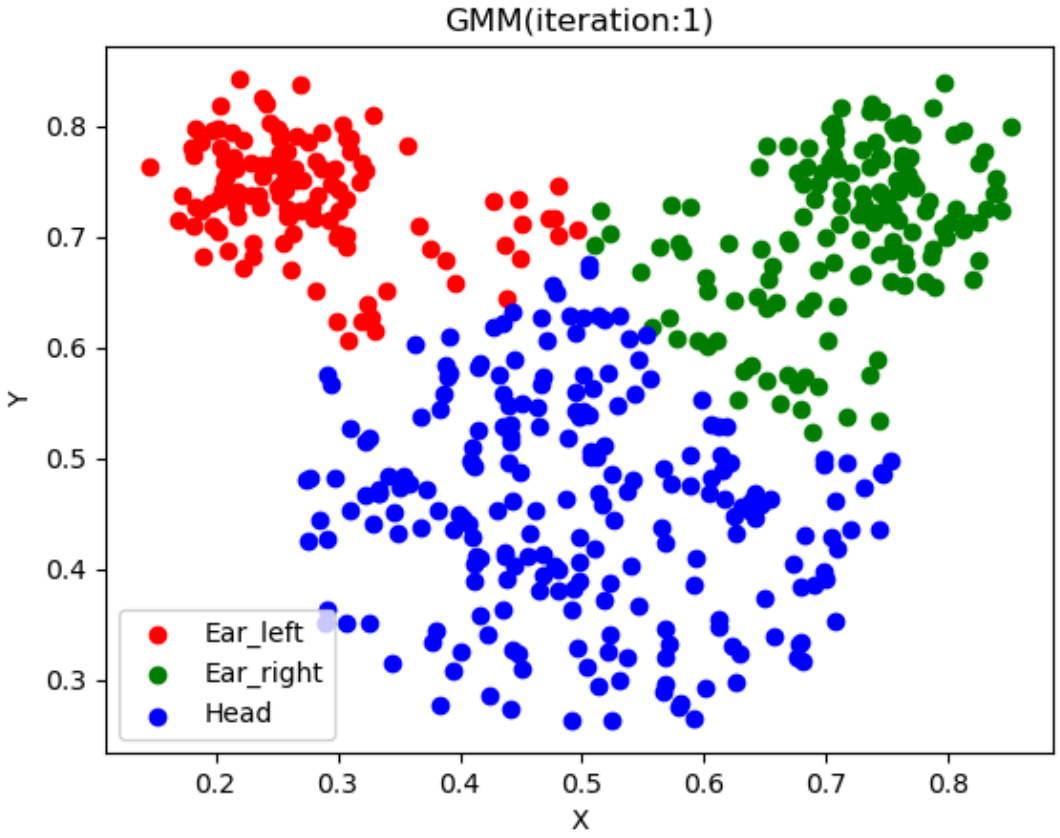
In [8]:
```
#PDF
def jointed_gaussian_pdf(X,mu,sigma):
    N=len(X)
    D=len(X[0])
    sigma_inv=np.linalg.inv(sigma)
    sigma_det=np.linalg.det(sigma)
    lowwer=1/(np.sqrt(((2*np.pi)**D)*sigma_det))
    pdf=np.zeros(N)
    for i in range(N):
        diff=X[i]-mu
        exponent=-0.5*np.dot(diff.T,np.dot(sigma_inv,diff))
        pdf[i]=lowwer*np.exp(exponent)
    return pdf
```

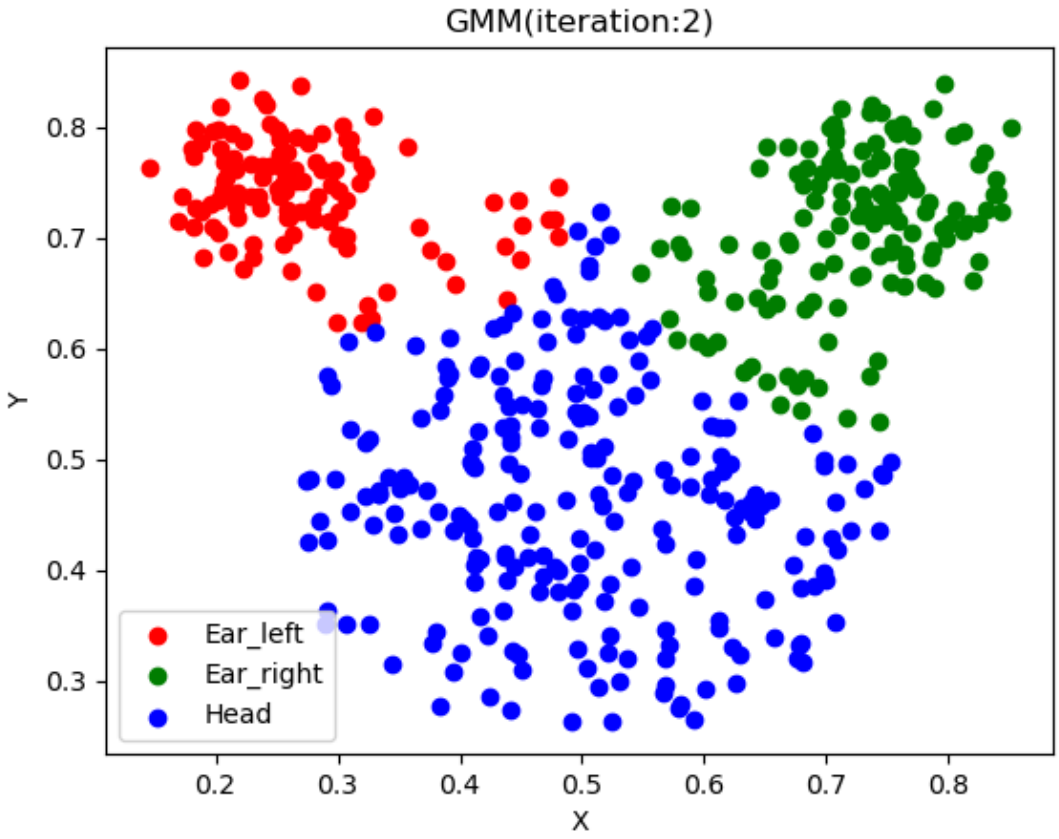
In [9]:

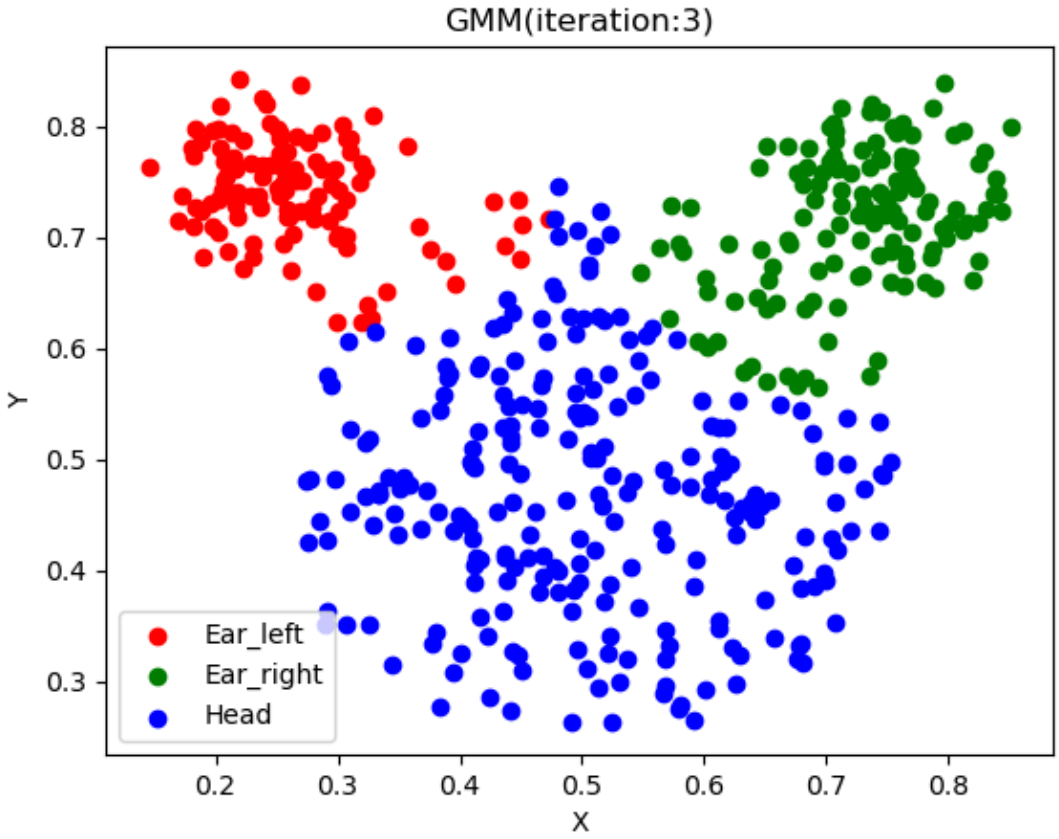
```
#GMM
def gmm(X,K,gamma,pis,mus,sigmas,max_iter=100,tol=1e-4):
    N=len(X)
    D=len(X[0])
    ll_old=0#-np.inf    #negative log-likelihood
    new_label=np.zeros_like(label)
    for iter in range(max_iter):
        #E-step
        for k in range(K):
            gamma[:,k]=pis[k]*jointed_gaussian_pdf(X,mus[k],sigmas[k])
        gamma/=np.sum(gamma,axis=1,keepdims=True)
        #for k in range(K):
        #    gamma[:,k]/=sum(gamma[:,k])

        # M-step
        for k in range(K):
            pis[k]=sum(gamma[:,k])/N
            for d in range(D):
                mus[k,d]=sum(gamma[:,k]*X[:,d])/sum(gamma[:,k])
            diff=X-mus[k]
            sigmas[k]=np.dot(gamma[:,k]*diff.T,diff)/sum(gamma[:,k])

        #update new_label and draw first four iterations
        for n in range(N):
            new_label[n]=np.argmax(gamma[n])
        if iter<4:
            draw_label_scater(pairs_2,new_label,f"GMM(iteration:{iter+1})")

        #negative log-likelihood
        ll_new=0
        for k in range(K):
            ll_new+=pis[k]*jointed_gaussian_pdf(X,mus[k],sigmas[k])
        ll_new=sum(np.log(ll_new))
        if abs(ll_new-ll_old)<tol:
            draw_label_scater(pairs_2,new_label,f"Final result of GMM(iteratio
            break
        ll_old=ll_new
    return new_label
```

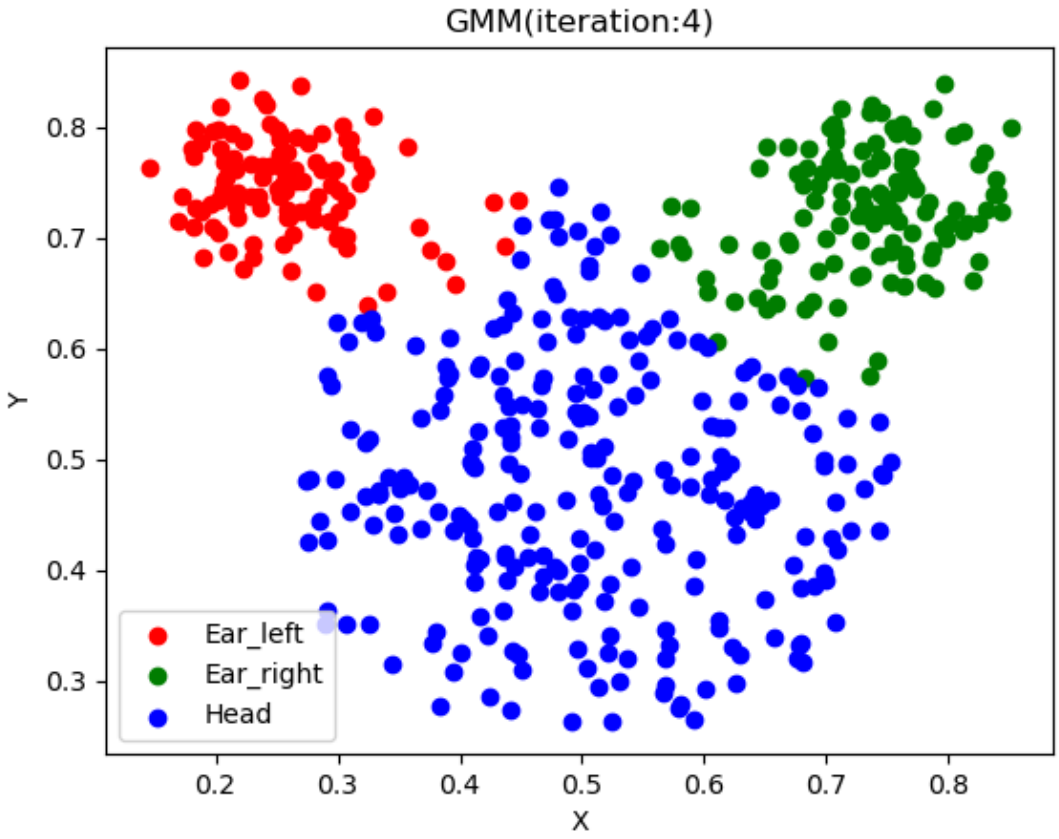
In [10]:

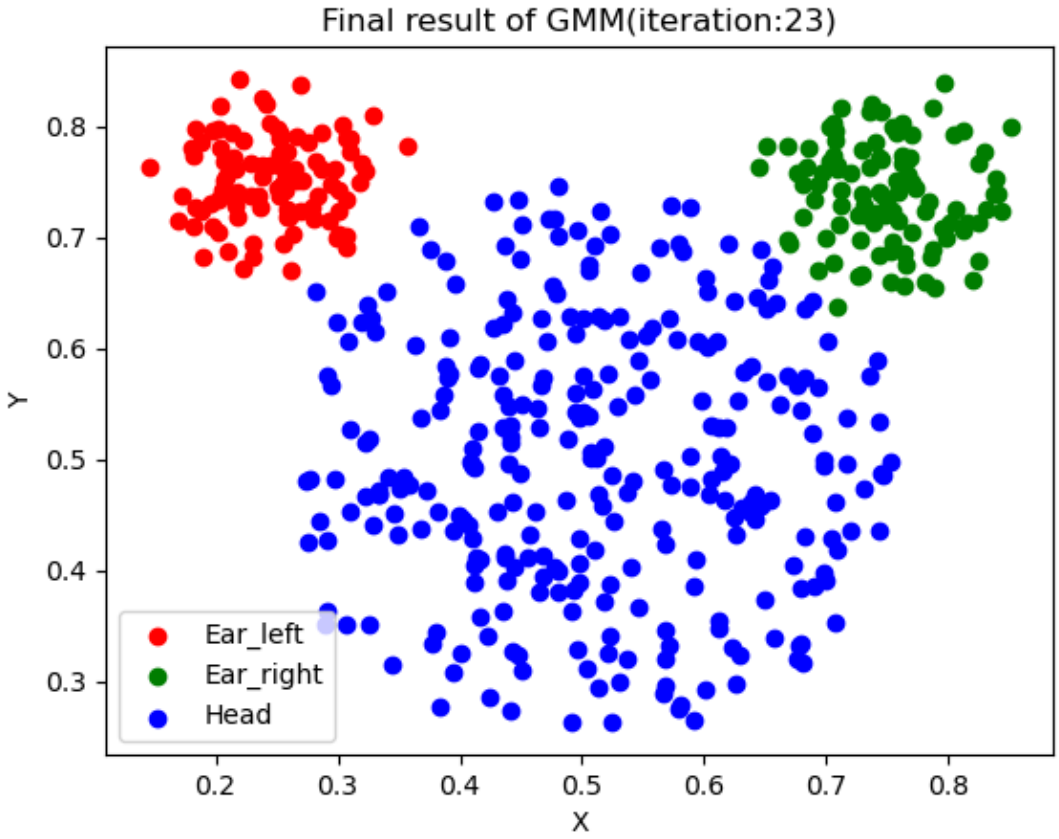
```
#one-hot
one_hot=np.zeros((len(pairs_3),K))
for i in range(len(sorted_label)):
    one_hot[i,sorted_label[i]]=1
#print(one_hot)

cluster_means=np.zeros((K,2))    #mu
cluster_covs=np.zeros((K,2,2))   #sigma
cluster_weights=np.zeros((K))    #pi
for k in range(K):
    cluster_weights[k]=sum(one_hot[:,k])/len(pairs_3)
    cluster_means[k,0]=sum(one_hot[:,k]*pairs_2[:,0])/sum(one_hot[:,k]) #x_mea
    cluster_means[k,1]=sum(one_hot[:,k]*pairs_2[:,1])/sum(one_hot[:,k]) #y_mea
    diff=pairs_2-cluster_means[k]
    cluster_covs[k,0,0]=sum(one_hot[:,k]*(diff[:,0]**2))/sum(one_hot[:,k])  #v
    cluster_covs[k,1,1]=sum(one_hot[:,k]*(diff[:,1]**2))/sum(one_hot[:,k])  #v
    cluster_covs[k,0,1]=cluster_covs[k,1,0]=sum(one_hot[:,k]*diff[:,0]*diff[:,

new_label=gmm(pairs_2,K,one_hot,cluster_weights,cluster_means,cluster_covs)
```




GMM(iteration:2)
Ear_left
Ear_right
Head
X
Y
0.2
0.3
0.4
0.5
0.6
0.7
0.8


GMM(iteration:3)
Ear_left
Ear_right
Head
X
Y
0.2
0.3
0.4
0.5
0.6
0.7
0.8


GMM(iteration:4)
Ear_left
Ear_right
Head
X
Y
0.2
0.3
0.4
0.5
0.6
0.7
0.8


Final result of GMM(iteration:23)
Ear_left
Ear_right
Head
X
Y
0.2
0.3
0.4
0.5
0.6
0.7
0.8

In [11]:
```
#confusion matrix
confusion_matrix = pd.crosstab(pairs_3[:,2],new_label,rownames=['Actual'],coln
print("Confusion Matrix")
print("(0:Ear_left, 1:Ear_right, 2:Head)")
print(confusion_matrix)
```

```
Confusion Matrix
(0:Ear_left, 1:Ear_right, 2:Head)
Predicted   0    1    2
Actual
Ear_left   99    0    1
Ear_right   0  100    0
Head        0    1  289
```

Comment

different

- K-means based purely on distance, so some "Head" points may be misclassified as "Ear_left" or "Ear_right" if thay are closer to those clusters.
- GMM can handle more complex situation, becase it consider both the mean and the covarance matrix of each cluster, allowing for clusters with different shapes and sizes.

perform

- In this problem, I think GMM performs better. GMM takes into account the variations in size and shape of different cluster.
- From the confusion matrices of two methods, we can see that the result of GMM are closer to the true labels compared to K-means.
- When clusters have the same shape and size, K-means may perform well and it computationally faster than GMM.

hw3q2

(revised) Python code template — Python

```python
## copyright, Keith Chugg, Brandon Franzke
##  EE599, 2020

#################################################
## this is a template to illustrate hd5 files
##
## also can be used as template for HW1 problem
#################################################

import h5py
import numpy as np
import matplotlib.pyplot as plt

#DEBUG = True
DEBUG = False
DATA_FNAME = 'brandon_franzke_hw1_1.hd5'

if DEBUG:
    num_sequences = 3
    sequence_length = 4
else:
    num_sequences = 10
    sequence_length = 50

### Enter your data here...
### Be sure to generate the data by hand.  DO NOT:
###     copy-n-paste
###     use a random number generator
###
"""
x_list = [
    [ 0, 1, 1, 0],
    [ 1, 1, 0, 0],
    [ 0, 0, 0, 1]
]
"""
x_list=[[0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,1
,1,1,1,1,1,1,1,0,0,0,0,0,0,1,0,1,0],
        [0,0,0,0,1,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,1,1,1,1,0,1,1,1,1,0
,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0],
        [1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0
,1,0,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1],
        [1,1,0,0,0,1,1,1,1,1,1,0,0,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],

```

```
        [1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,0,0,0],
        [0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0],
        [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1]]

# convert list to a numpy array...
human_binary = np.asarray(x_list)

### do some error trapping:

assert human_binary.shape[0] == num_sequences, 'Error: the number of sequences was entered incorrectly'
assert human_binary.shape[1] == sequence_length, 'Error: the length of the seqeunces is incorrect'

# the with statement opens the file, does the business, and close it up for us...
with h5py.File(DATA_FNAME, 'w') as hf:
    hf.create_dataset('human_binary', data = human_binary)
    ## note you can write several data arrays into one hd5 file, just give each a different name.

###################
# Let's read it back from the file and then check to make sure it is as we wrote...
with h5py.File(DATA_FNAME, 'r') as hf:
    hb_copy = hf['human_binary'][:]

### this will throw and error if they are not the same...
np.testing.assert_array_equal(human_binary, hb_copy)
```

HW3-Q3

In [3]:
```
import numpy as np
import h5py
import matplotlib.pyplot as plt
```

In [4]:
```
with h5py.File('mnist_network_params.hdf5','r') as f:
    W1=np.array(f['W1'])
    W2=np.array(f['W2'])
    W3=np.array(f['W3'])
    b1=np.array(f['b1'])
    b2=np.array(f['b2'])
    b3=np.array(f['b3'])

assert W1.shape==(200, 784),"Error: W1's size incorrect"
assert b1.shape==(200,),"Error: b1's size incorrect"
assert W2.shape==(100, 200),"Error: W2's size incorrect"
assert b2.shape==(100,),"Error: b2's size incorrect"
assert W3.shape==(10, 100),"Error: W3's size incorrect"
assert b3.shape==(10,),"Error: b3's size incorrect"
```

In [5]:
```
with h5py.File('mnist_testdata.hdf5','r') as f:
    xdata=np.array(f['xdata'])
    ydata=np.array(f['ydata'])

assert xdata.shape==(10000, 784),"Error: xdata's size incorrect"
assert ydata.shape==(10000, 10),"Error: ydata's size incorrect"
```

In [6]:
```
def ReLU(x):
    return np.maximum(np.zeros_like(x),x)

def Softmax(x):
    return np.exp(x)/np.sum(np.exp(x))

def MLP(x):
    z1=np.dot(W1,x)+b1
    a1=ReLU(z1)
    z2=np.dot(W2,a1)+b2
    a2=ReLU(z2)
    z3=np.dot(W3,a2)+b3
    result=Softmax(z3)
    return result
```

In [7]:
```
result=np.zeros_like(ydata)
pre_label=np.zeros(len(ydata))
true_label=np.zeros_like(pre_label)
data=[]
for i in range(len(xdata)):
    result[i]=MLP(xdata[i])
    pre_label[i]=np.argmax(result[i])
    true_label[i]=np.argmax(ydata[i])
    data.append({
        "index": i,
        "activations": result[i].tolist(),
        "classification": pre_label[i]
    })
```

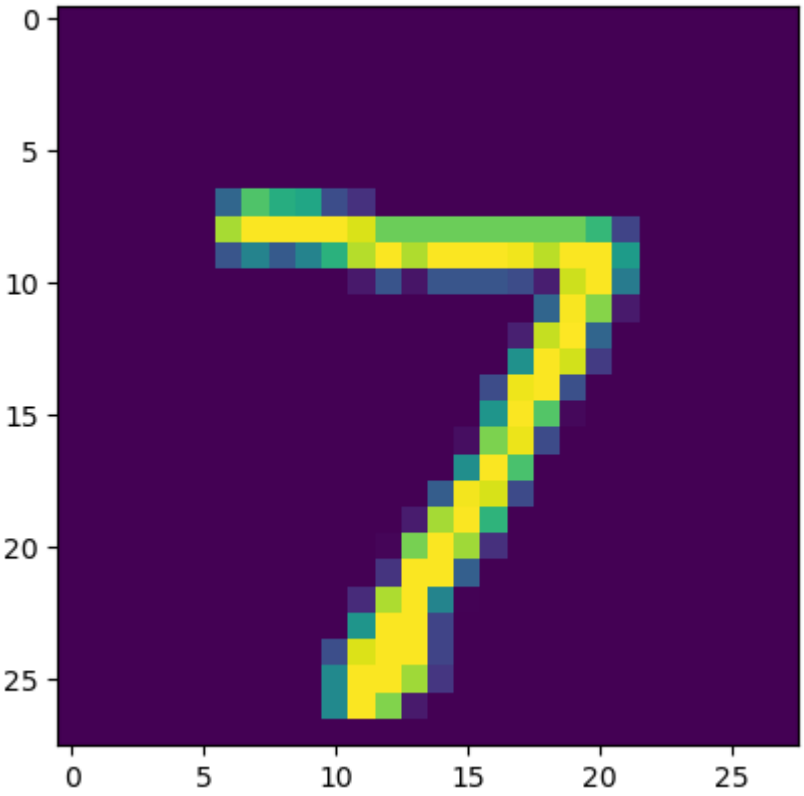
In [8]:
```
import json
with open("result.json","w") as f:
    f.write(json.dumps(data))
```

In [10]:
```
classified=[]
misclassified=[]
for i in range(len(true_label)):
    if pre_label[i]==true_label[i]:
        classified.append(i)
    else:
        misclassified.append(i)
print("The number of correctly classified images:",len(classified))
```

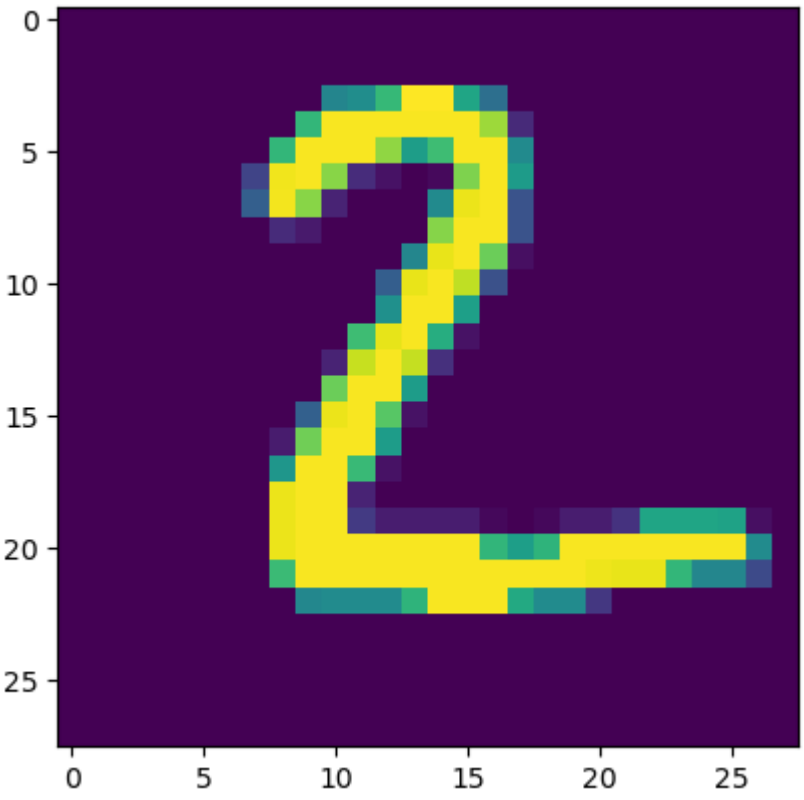
```
The number of correctly classified images: 9790
```

In [12]:
```
#several it classified correctly
for i in range(3):
    plt.figure()
    idx=classified[i]
    print("Picture",idx,"is classified correctly:")
    print("The label is:",int(true_label[idx]))
    plt.imshow(xdata[idx].reshape(28,28))
    plt.show()
```

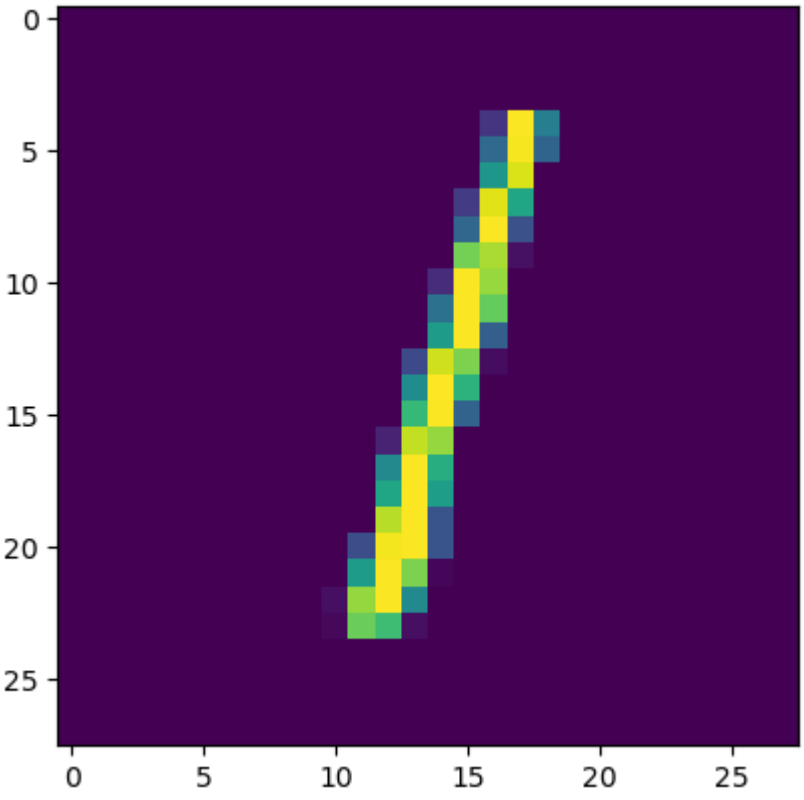
```
Picture 0 is classified correctly:
The label is: 7
```




```
Picture 1 is classified correctly:
The label is: 2
```

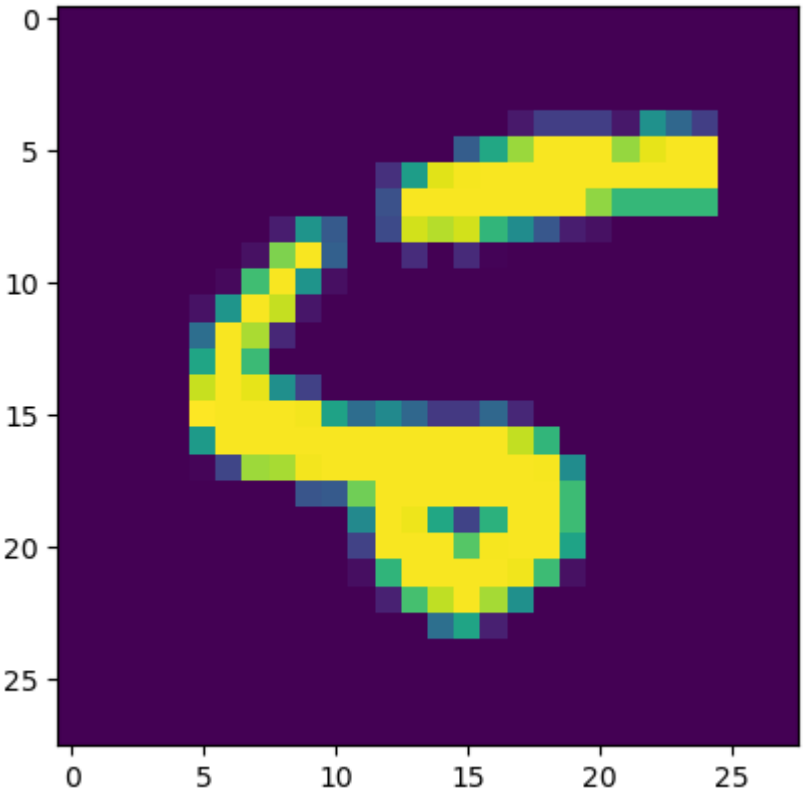



```
Picture 2 is classified correctly:
The label is: 1
```

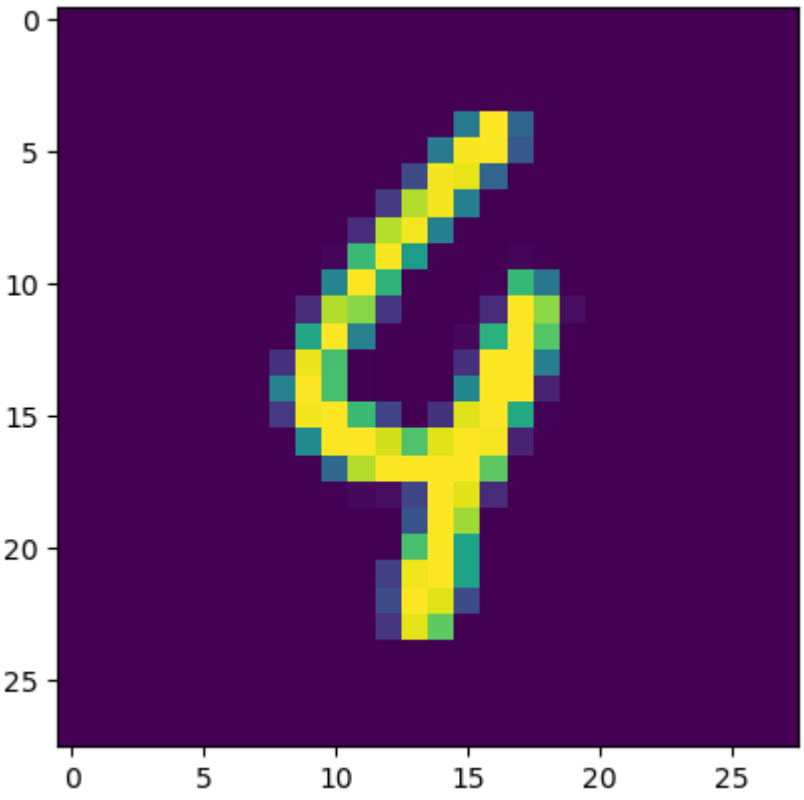


In [15]:
```
#several it classified incorrectly
for i in range(5):
    plt.figure()
    idx=misclassified[i]
    print("Picture",idx,"is classified incorrectly:")
    print("The true label is:",int(true_label[idx]))
    print("The predicted label is:",int(pre_label[idx]))
    plt.imshow(xdata[idx].reshape(28,28))
    plt.show()
```

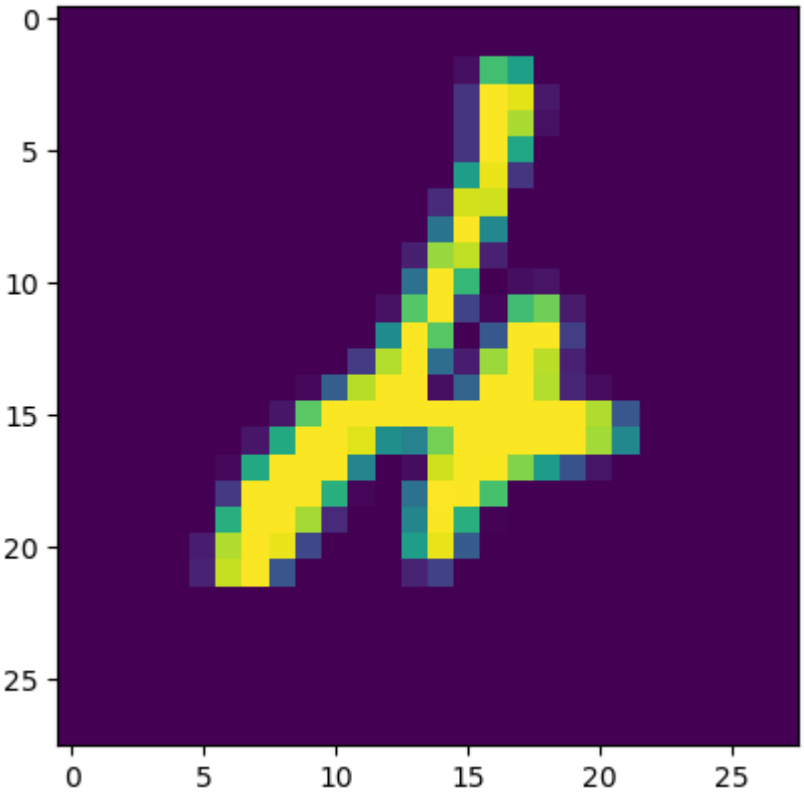
```
Picture 8 is classified incorrectly:
The true label is: 5
The predicted label is: 6
```



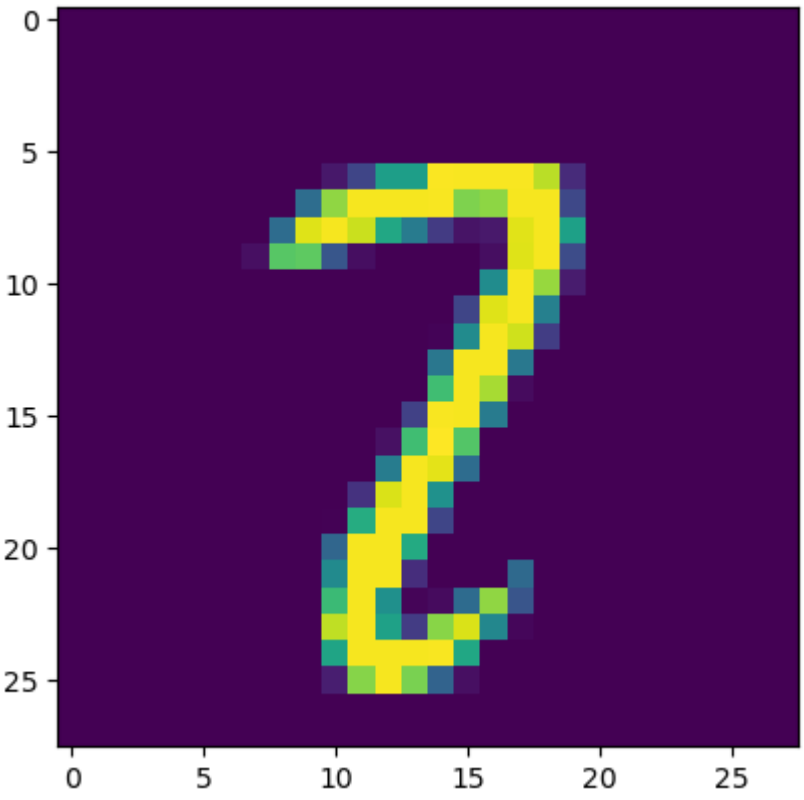

```
Picture 115 is classified incorrectly:
The true label is: 4
The predicted label is: 9
```



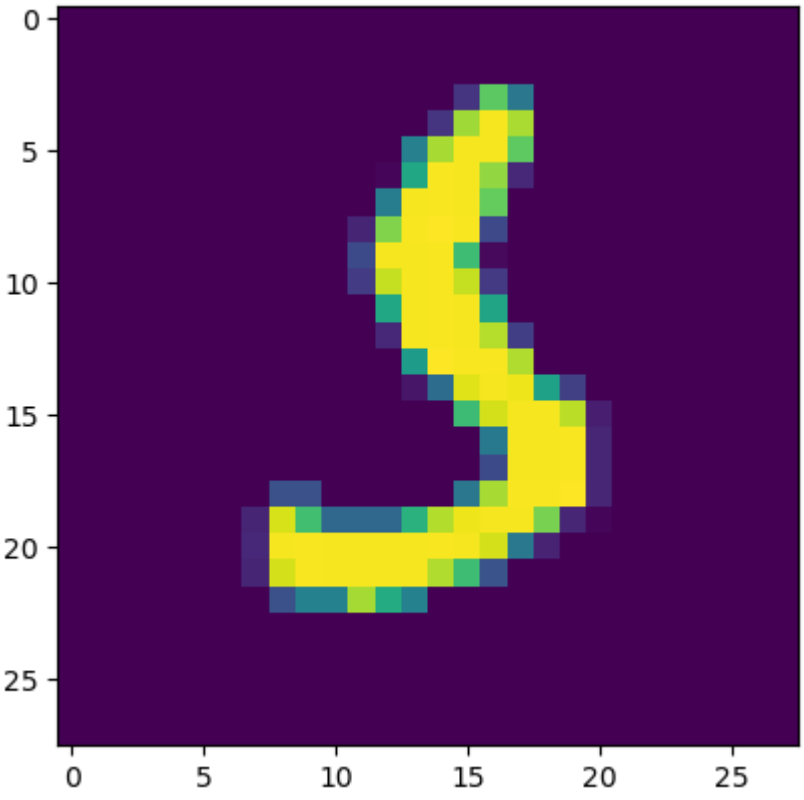

```
Picture 247 is classified incorrectly:
The true label is: 4
The predicted label is: 6
```




```
Picture 321 is classified incorrectly:
The true label is: 2
The predicted label is: 7
```




```
Picture 340 is classified incorrectly:
The true label is: 5
The predicted label is: 3
```



Comment

I think the incorrect cases are also challenging for me to identified. For instance, in picture 115, the digit '4' is misclassified as '9'. This is understandable, because two digits share similar stuctures, and when they are not written clearly, it becomes easy to confuse them.