LAB 9 : Dimensionality Reduction

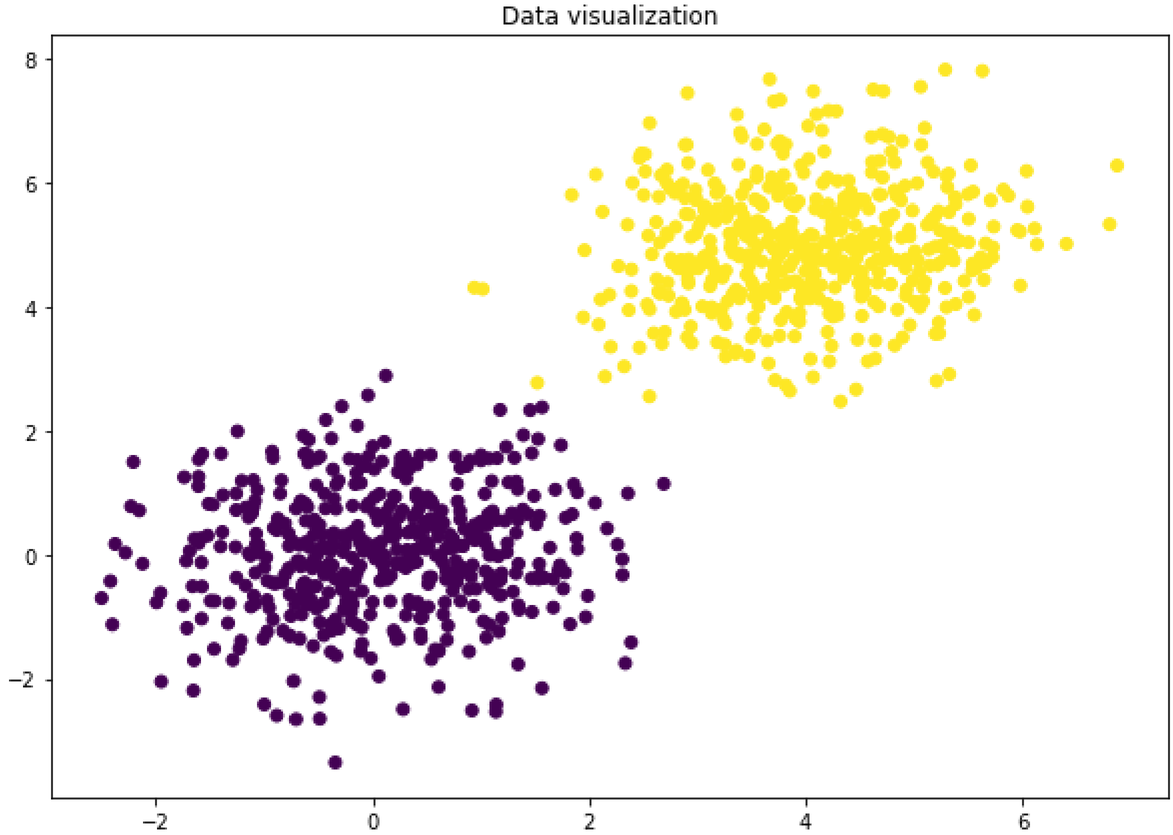
1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)

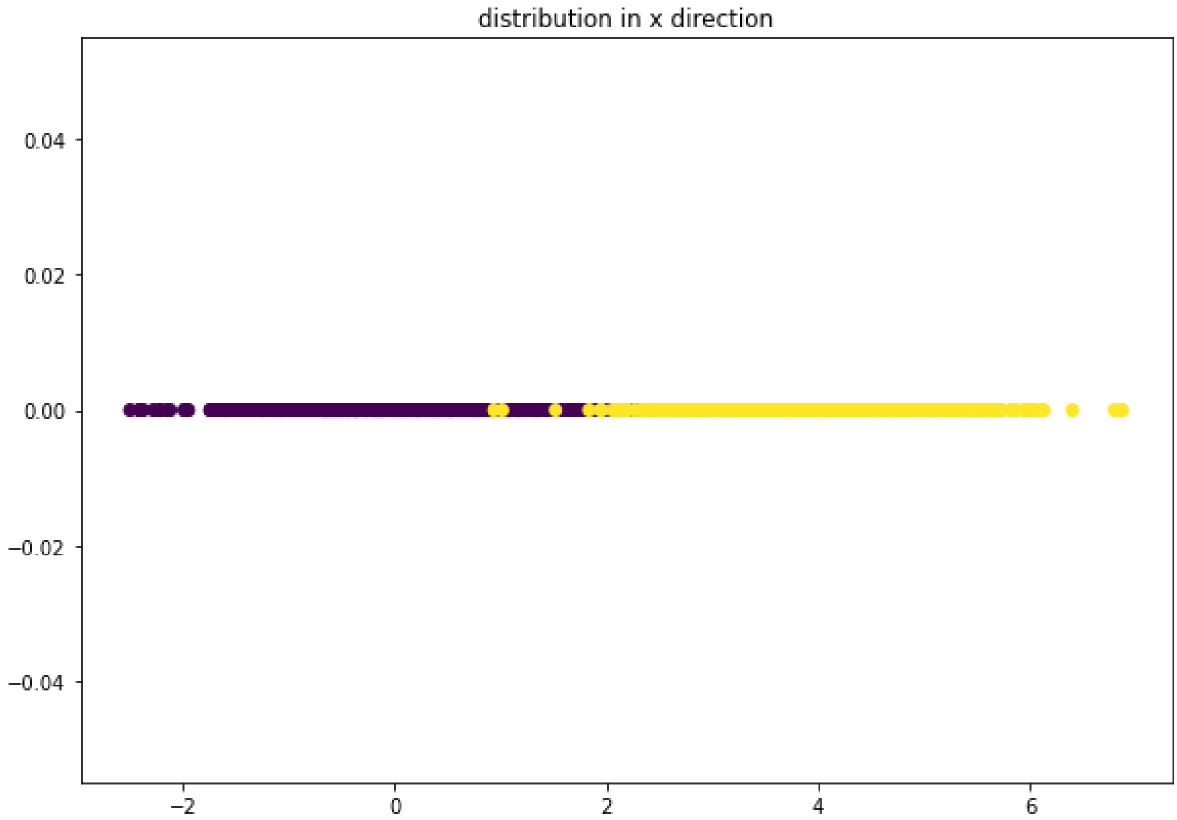
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

matplotlib.rcParams['figure.figsize'] = (10, 7)
```

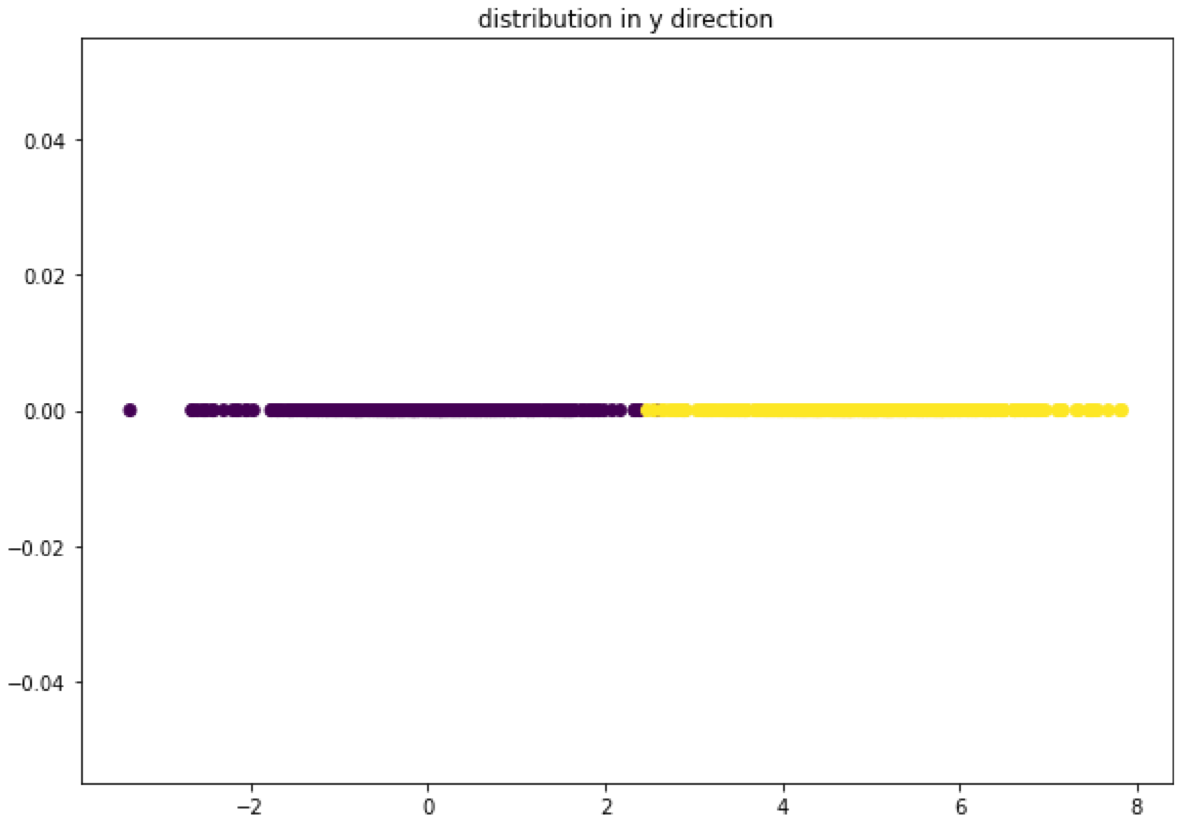
PCA

In [ ]:
```
mean1=np.array([0,0])
mean2=np.array([4,5])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data=np.concatenate((data1,data2))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in x direction')
plt.figure()
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in y direction')
```

Out[ ]:
```
Text(0.5, 1.0, 'distribution in y direction')
```

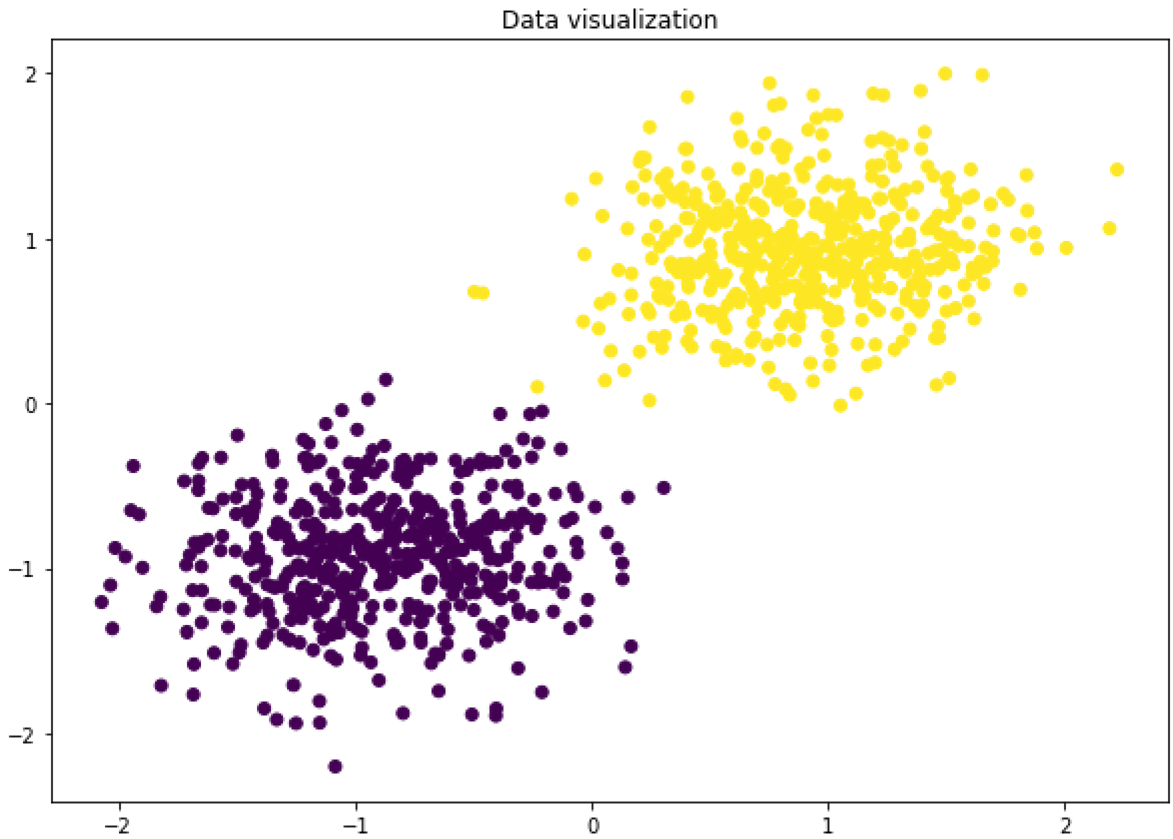

Data visualization
8
6
4
2
0
−2
−2
0
2
4
6


distribution in x direction
0.04
0.02
0.00
−0.02
−0.04
−2
0
2
4
6



In [ ]:
```
# Data normalization

# Perform data normalization here using mean substraction and std division
## Write your code here

mean = np.mean(data, axis=0)
std = np.std(data, axis=0)
mean, std

data = (data-mean)/std

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
```

Out[ ]:
```
Text(0.5, 1.0, 'Data visualization')
```



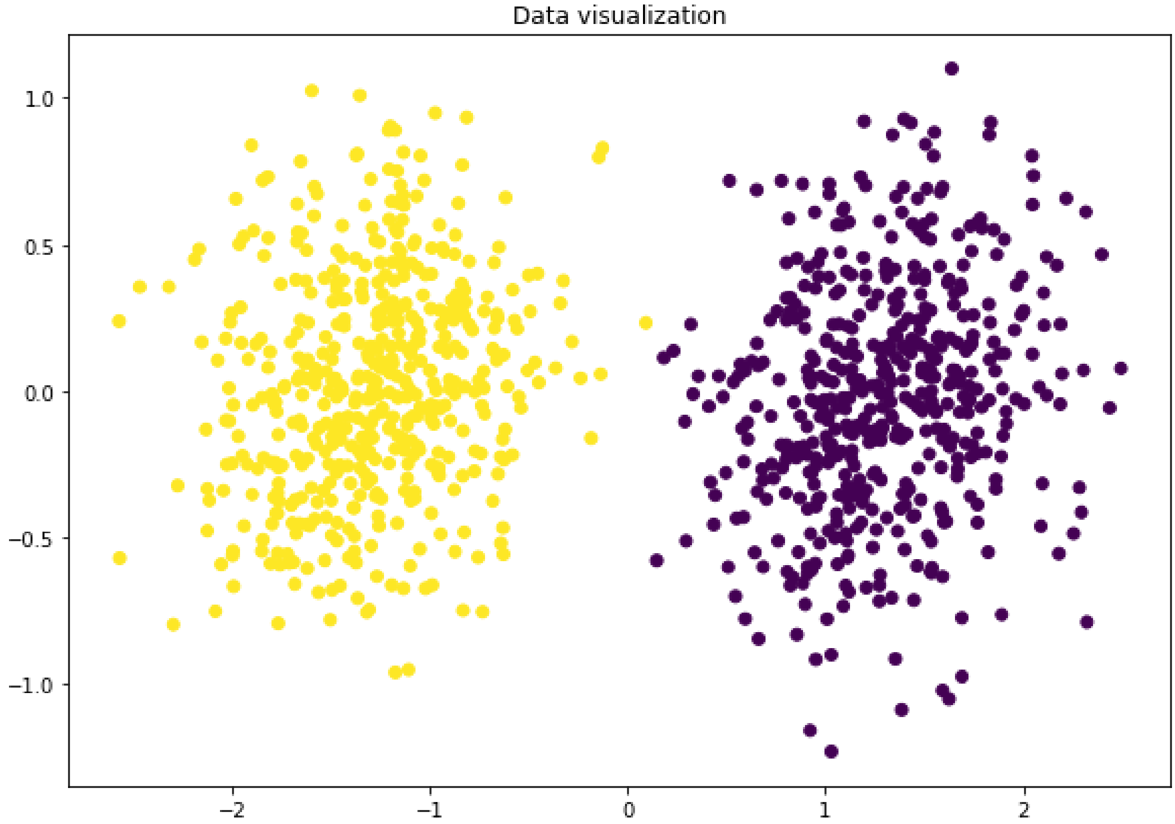
In [ ]:

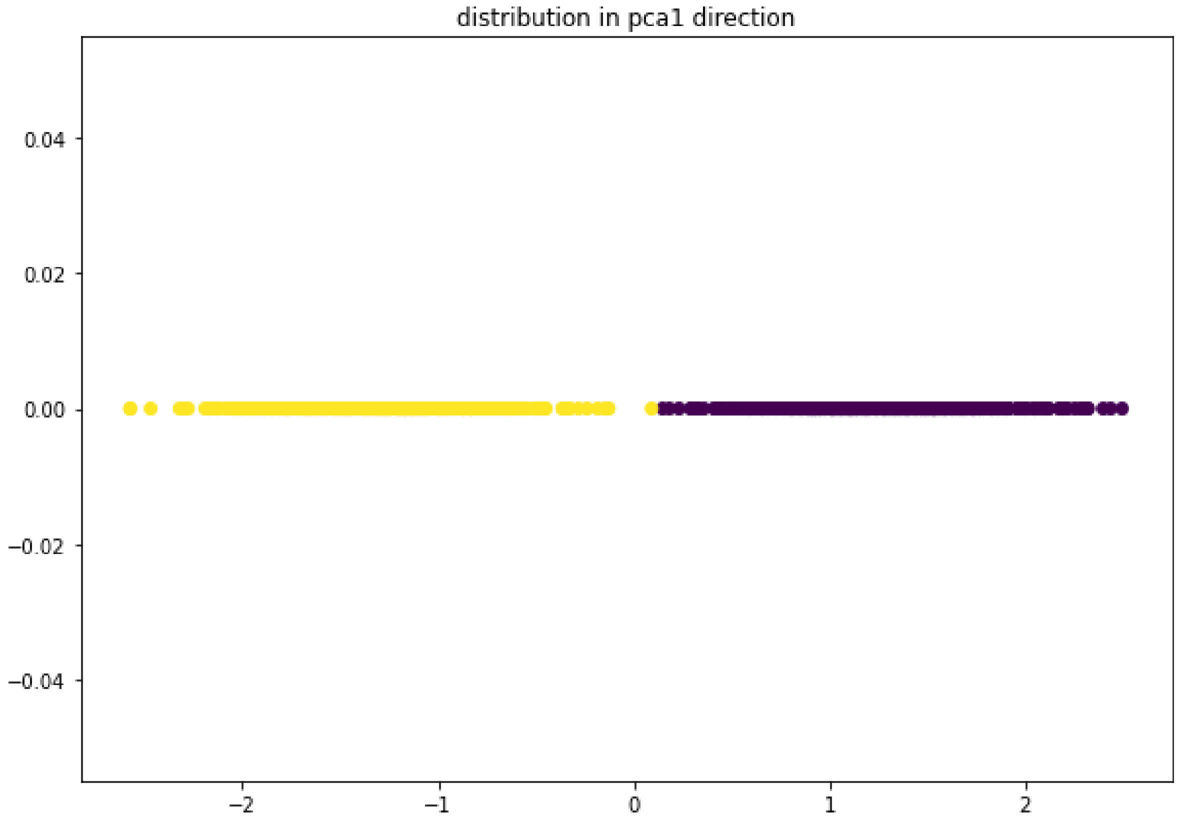
```
# PCA

# coverance matrix
print(data.shape)
cov=data.T @ data

print(cov.shape)


# using sigular value decomposition
u, s, v=np.linalg.svd(cov)

print(u.shape, s.shape, v.shape)

trans_data= data @ u ## Write your code here

var_pca1=np.var(trans_data[:,0])
var_pca2=np.var(trans_data[:,1])

print('variance along pca1 direction=',var_pca1)
print('variance along pca2 direction=',var_pca2)


plt.figure()
plt.scatter(trans_data[:,0],trans_data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(trans_data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in pca1 direction')
plt.figure()
plt.scatter(trans_data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in pca2 direction')
```

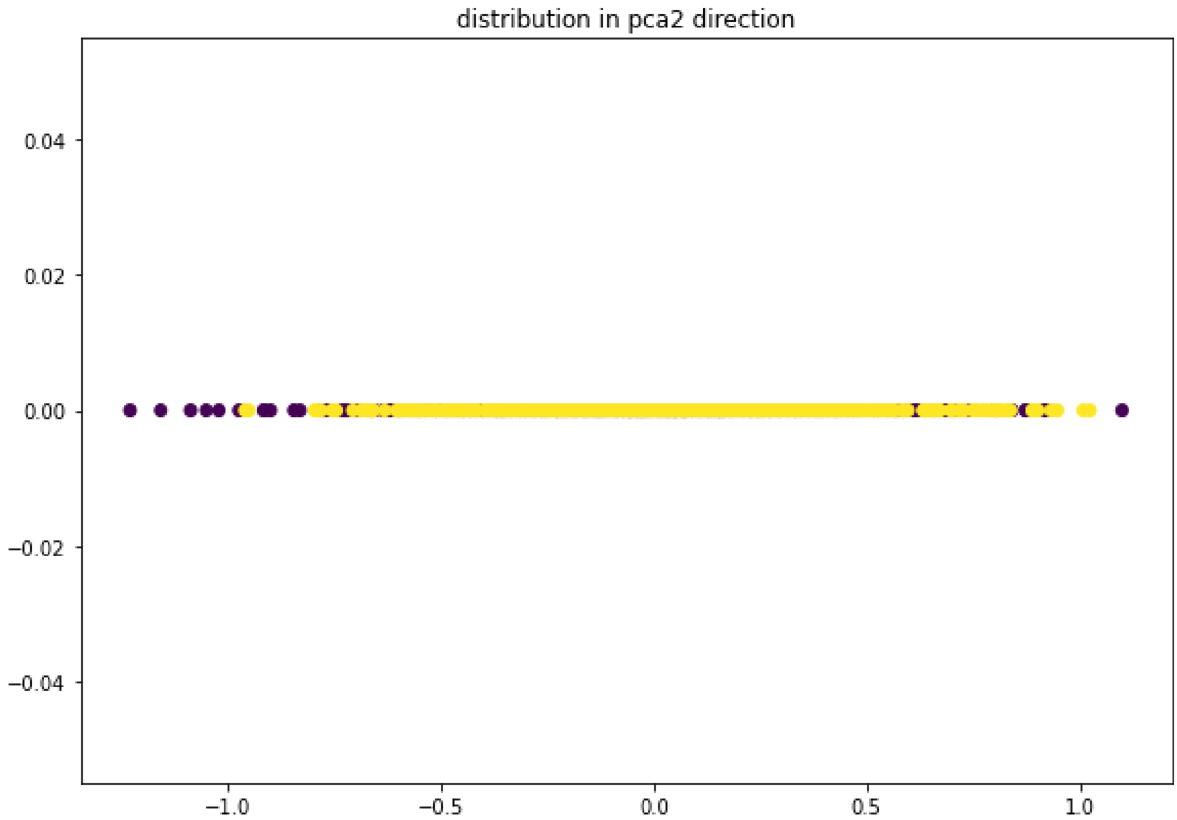
```
(1000, 2)
(2, 2)
(2, 2) (2,) (2, 2)
variance along pca1 direction= 1.8477663843459724
variance along pca2 direction= 0.152233615654027
```

Out[ ]:

```
Text(0.5, 1.0, 'distribution in pca2 direction')
```



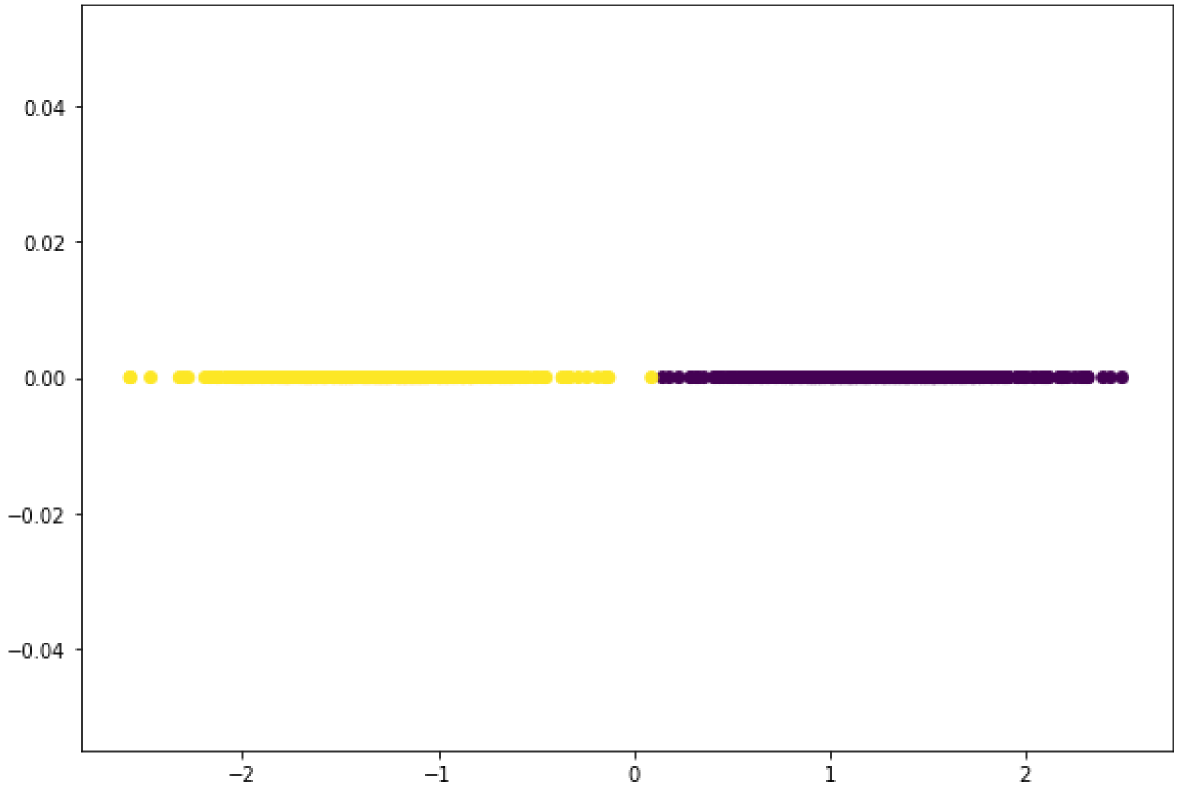




In [ ]:
```
class pca:
  # Constructor
  def __init__(self, name='reg',data=None,retain_dim=None):
    self.name = name  # Create an instance variable
    self.data=data
    self.retain_dim=retain_dim if retain_dim is not None else self.ret_dim(self.data
    # compute pca transform value

  def pca_comp(self,data):
    data = self.pre_process(data)
    cov = data.T @ data         ## Write your code here
    u,_,_ = np.linalg.svd(cov) # singular value decomposition
    u_req = u[:, :self.retain_dim] ## Write your code here
    trans_data= data @ u_req ## Write your code here
    return trans_data, u_req
    # compute the required retain dimension

  def ret_dim(self,data):
    data=self.pre_process(data)
    cov=data.T @ data
    _,s,_=np.linalg.svd(cov)
    ind = np.where(np.cumsum(s) >= 0.9*np.sum(s))[0][0] ## Write your code here
    return ind+1
  def pre_process(self,data):
    data1=(data-np.mean(data,axis=0))

    data=data1/(np.std(data1,axis=0)+10**(-30)) # avoid divide by zero
    return data
```

In [ ]:
```
# pca transformation
PCA = pca(data=data)
trans_data,trans_mat=PCA.pca_comp(data)
plt.scatter(trans_data,np.zeros(trans_data.shape),c=label)
```

Out[ ]: <matplotlib.collections.PathCollection at 0x1a7f65a7370>



In [ ]:
```
#classification using pca
#use k-nearest neighbour classifier after dimensionality reduction

from sklearn.neighbors import KNeighborsClassifier
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, label)

print('KNN Training accuracy =',knn.score(trans_data,label)*100)

# test data
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data=np.concatenate((data1,data2))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

print('KNN Testing accuracy =',knn.score(PCA.pre_process(data) @ trans_mat,tst_label
```

```
KNN Training accuracy = 99.9
KNN Testing accuracy = 100.0
```

PCA on MNIST

In [ ]:
```
!pip install idx2numpy
```

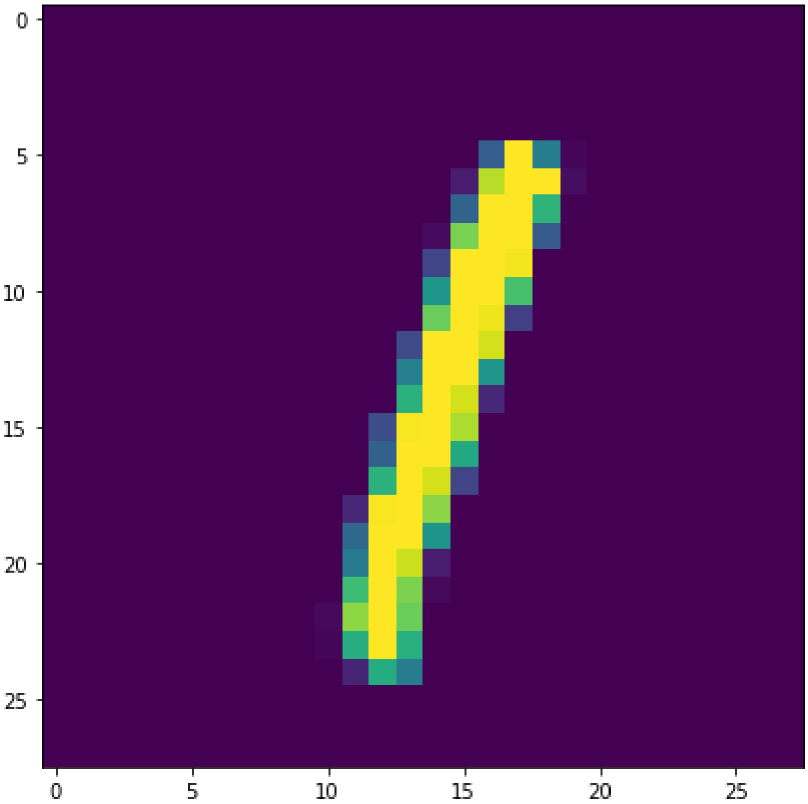
```
Requirement already satisfied: idx2numpy in c:\users\shashank\.env\.mldev\lib\site-p
ackages (1.2.3)
WARNING: Error parsing requirements for cryptography: [Errno 2] No such file or dire
ctory: 'c:\\users\\shashank\\.env\\.mldev\\lib\\site-packages\\cryptography-38.0.1.d
ist-info\\METADATA'
Requirement already satisfied: numpy in c:\users\shashank\.env\.mldev\lib\site-packa
ges (from idx2numpy) (1.22.4)
Requirement already satisfied: six in c:\users\shashank\.env\.mldev\lib\site-package
s (from idx2numpy) (1.16.0)
```

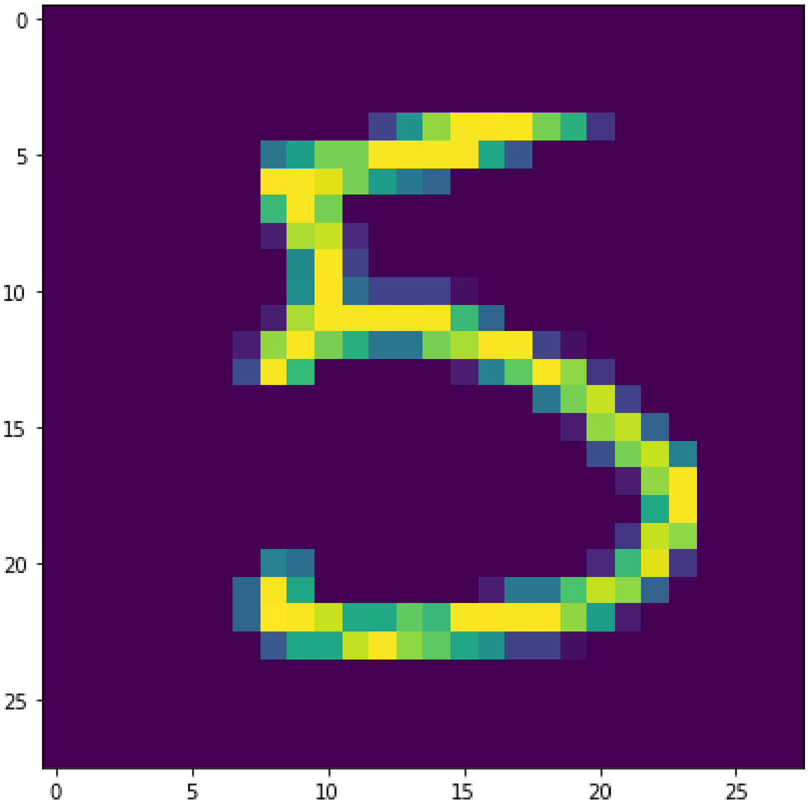
```
# MNIST data
```

In [ ]:

```
file1='./t10k-images-idx3-ubyte' ## Change the path accordingly
file2='./t10k-labels-idx1-ubyte' ## Change the path accordingly

import idx2numpy

Images= idx2numpy.convert_from_file(file1)
labels= idx2numpy.convert_from_file(file2)

cl=[1,5]

# for class 1

id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[:50]
Im_1=Images[id1]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[:50]
Im_5=Images[id5]
lab_5=labels[id5]

plt.imshow(Im_1[1])
plt.figure()
plt.imshow(Im_5[1])

#print(Im_5.shape)

data=np.concatenate((Im_1,Im_5))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
G_lab=np.concatenate((lab_1,lab_5))
print(G_lab.shape)

data = data.astype('float32')
```

```
(100, 784)
(100,)
```



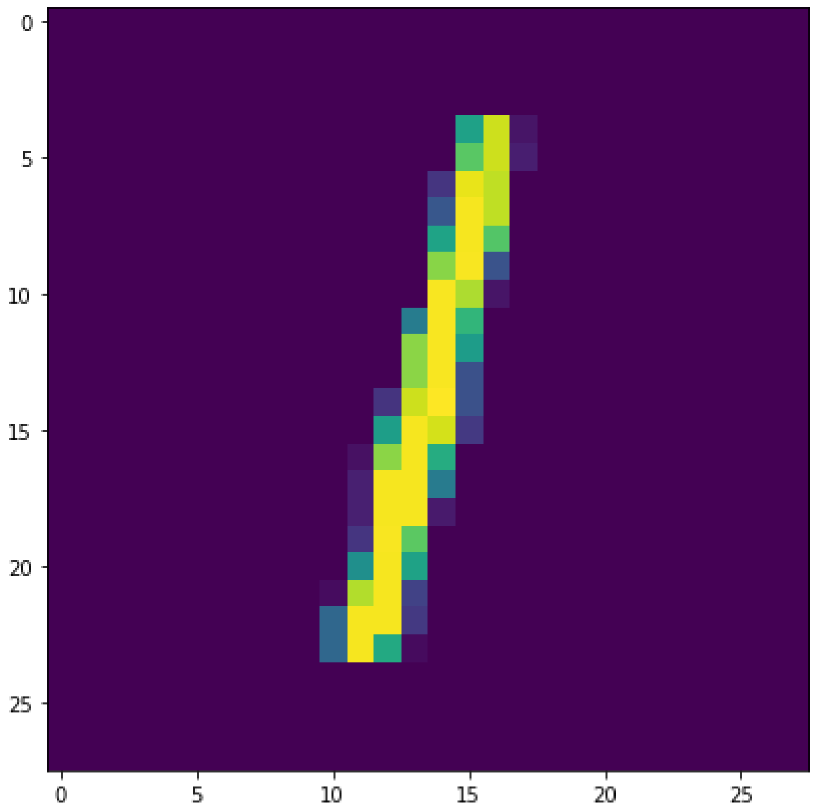


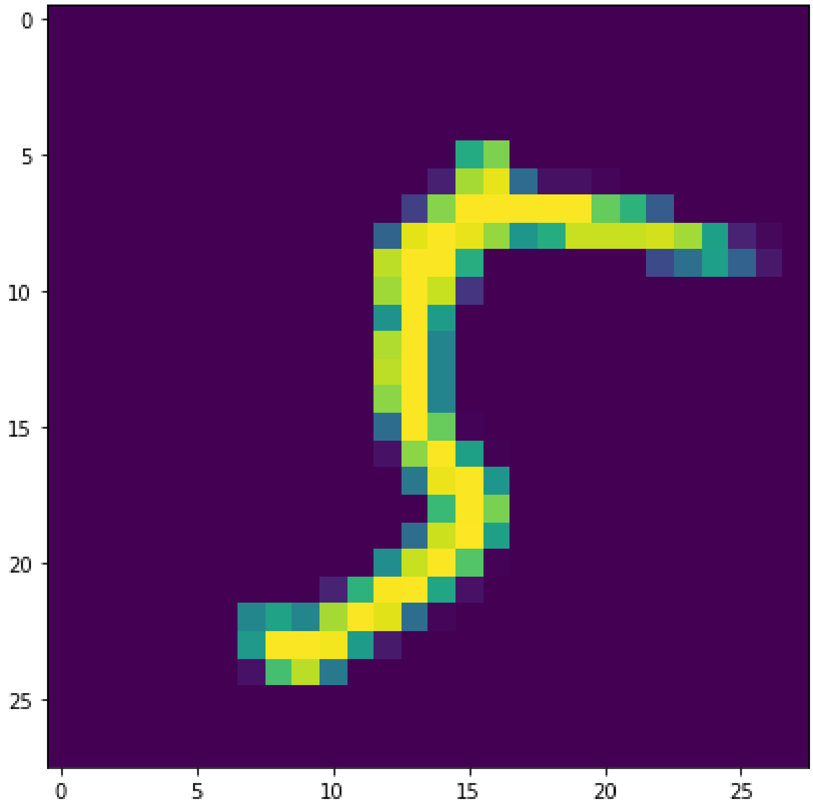
In [ ]:

```
print('Initial data dimension=',data.shape[1])
PCA=pca(data=data)
data = PCA.pre_process(data)

trans_data, trans_mat=PCA.pca_comp(data)
print('Retained dimesion after PCA=',trans_mat.shape[1])
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, G_lab)

print('KNN Training accuracy =',knn.score(trans_data,G_lab)*100)
```

```
## testing
## data preparation
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[100:150]
Im_1=Images[id1]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[100:150]
Im_5=Images[id5]
lab_5=labels[id5]

plt.imshow(Im_1[1])
plt.figure()
plt.imshow(Im_5[1])

print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5))
data_tst=np.reshape(data_tst,(data_tst.shape[0],data_tst.shape[1]*data_tst.shape[2]))

tst_lab=np.concatenate((lab_1,lab_5))


# final testing
print('KNN Testing accuracy =',knn.score(PCA.pre_process(data_tst) @ trans_mat, tst_
```

```
Initial data dimension= 784
Retained dimesion after PCA= 34
KNN Training accuracy = 96.0
(50, 28, 28)
KNN Testing accuracy = 98.0
```



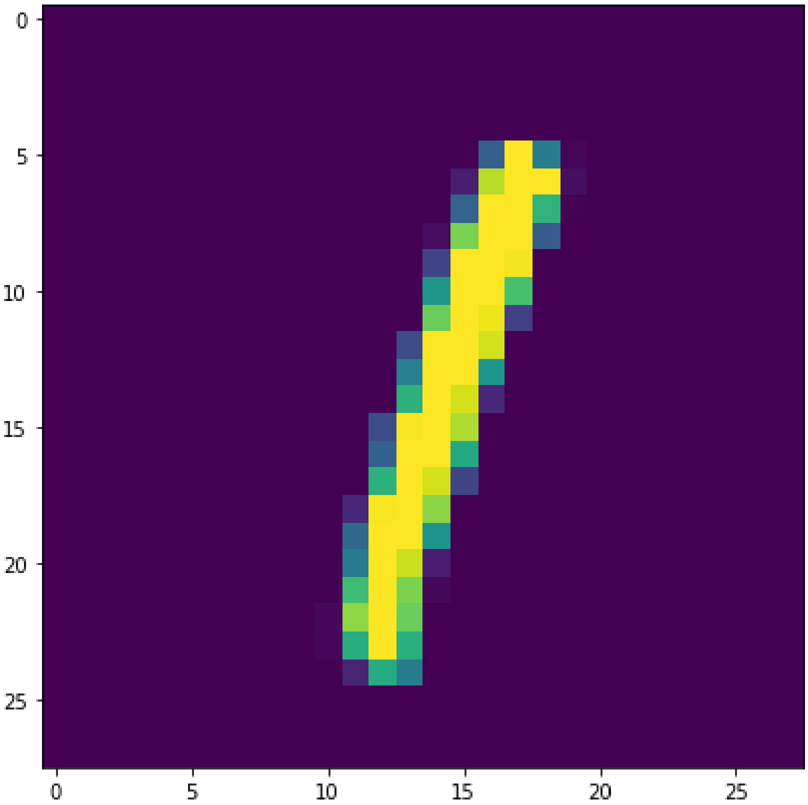


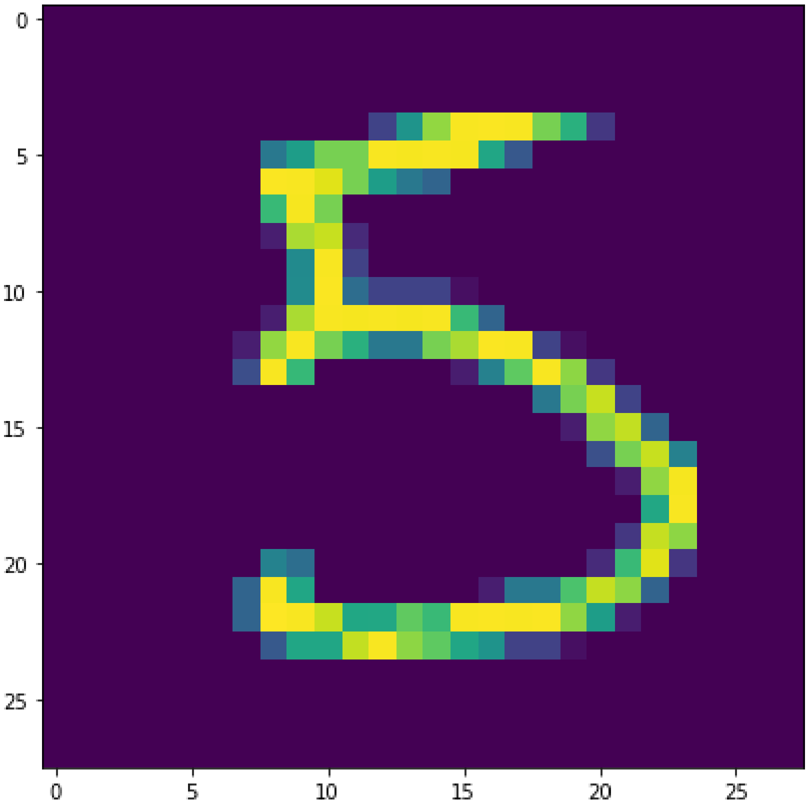
Perform PCA on MNIST and Classify taking the data with any 3 Classes

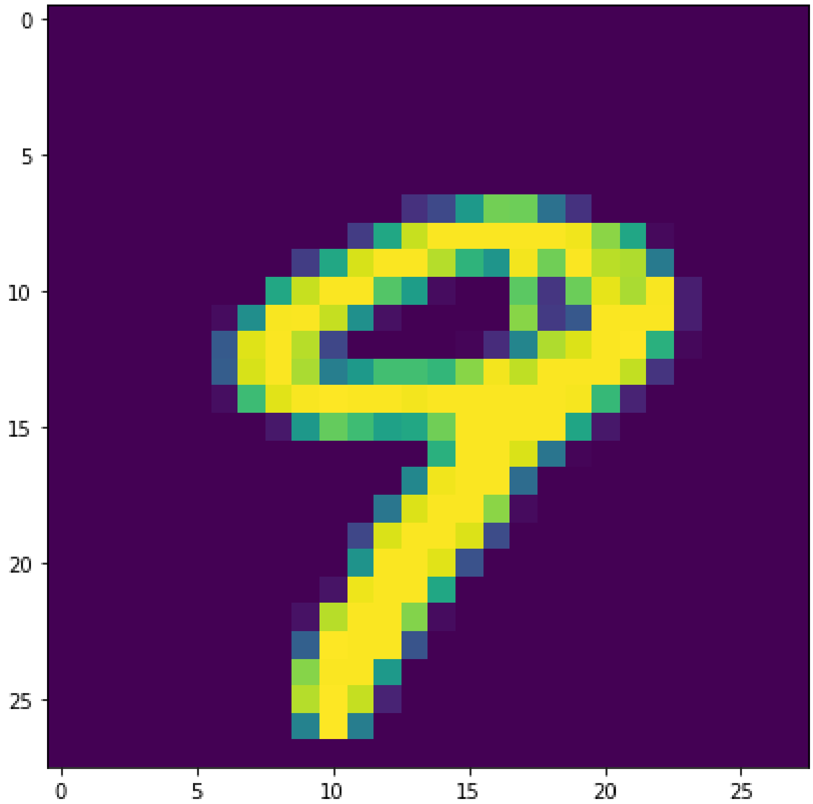
In [ ]:
```
## Write your code here
cl = [1, 5, 9]

id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[:50]
Im_1=Images[id1]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[:50]
Im_5=Images[id5]
lab_5=labels[id5]

id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[:50]
Im_9=Images[id9]
lab_9=labels[id9]

plt.imshow(Im_1[1])
plt.figure()
plt.imshow(Im_5[1])
plt.figure()
plt.imshow(Im_9[1])

#print(Im_5.shape)

data=np.concatenate((Im_1,Im_5,Im_9))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
G_lab=np.concatenate((lab_1,lab_5,lab_9))
print(G_lab.shape)
```

```
data = data.astype('float32')
```

```
(150, 784)
(150,)
```



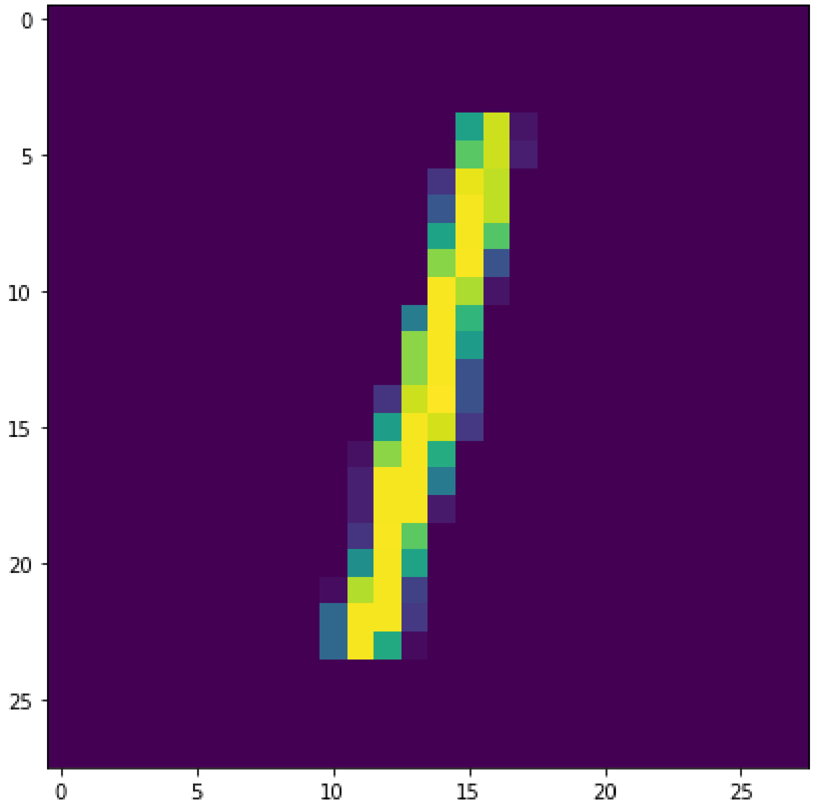


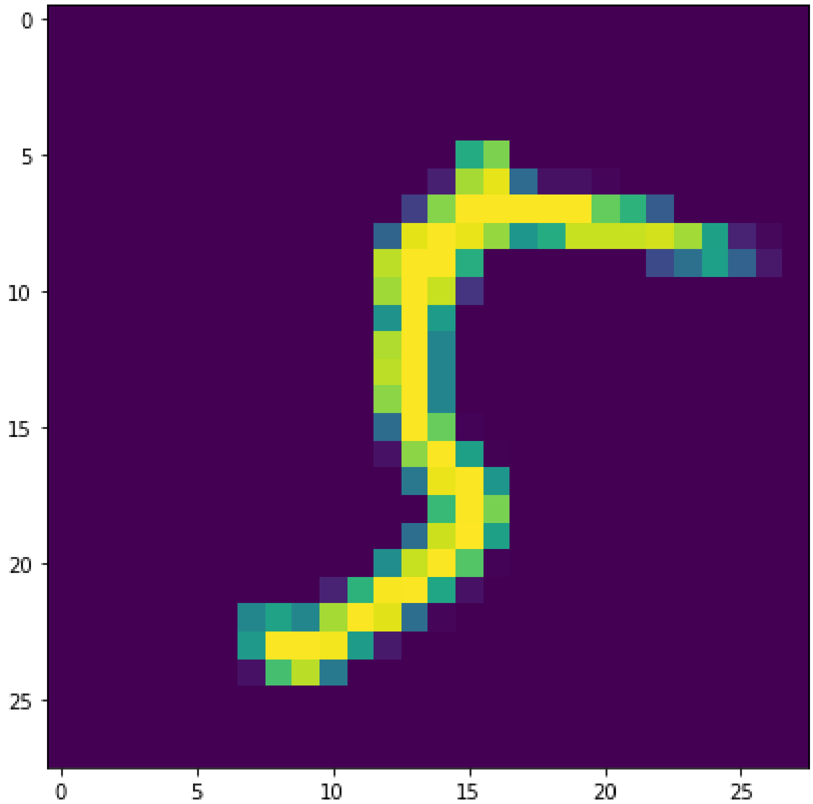


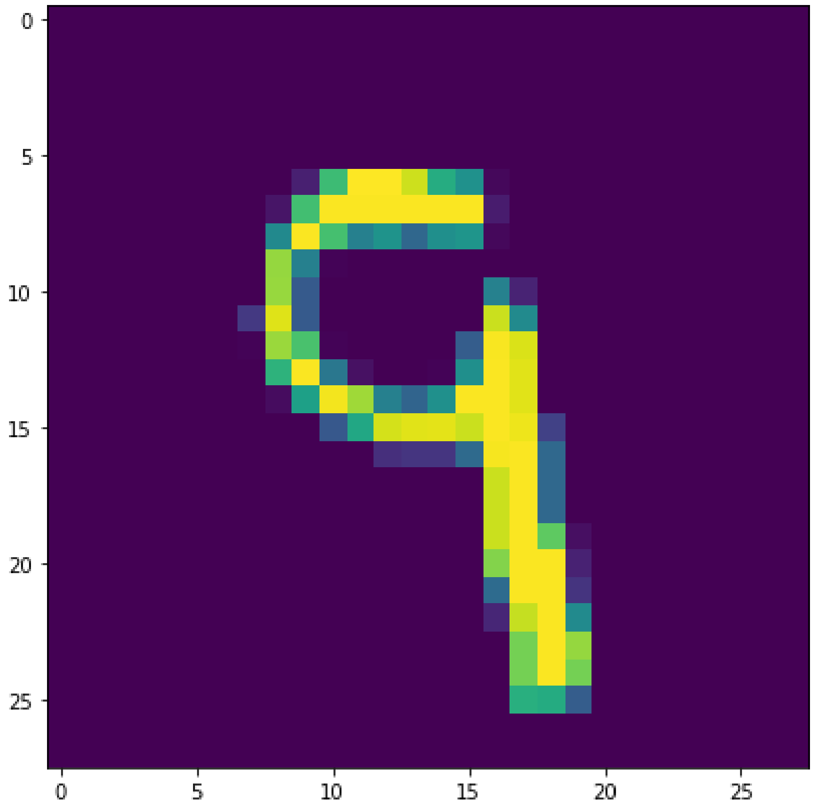
In [ ]:
```
print('Initial data dimension=',data.shape[1])
PCA=pca(data=data)
data = PCA.pre_process(data)

trans_data, trans_mat=PCA.pca_comp(data)
print('Retained dimesion after PCA=',trans_mat.shape[1])
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, G_lab)

print('KNN Training accuracy =',knn.score(trans_data,G_lab)*100)


## testing
## data preparation
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[100:150]
Im_1=Images[id1]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[100:150]
Im_5=Images[id5]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[100:150]
Im_9=Images[id9]
lab_9=labels[id9]

plt.imshow(Im_1[1])
plt.figure()
```

```
plt.imshow(Im_5[1])
plt.figure()
plt.imshow(Im_9[1])

print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5,Im_9))
data_tst=np.reshape(data_tst,(data_tst.shape[0],data_tst.shape[1]*data_tst.shape[2])

tst_lab=np.concatenate((lab_1,lab_5,lab_9))


# final testing
print('KNN Testing accuracy =',knn.score(PCA.pre_process(data_tst) @ trans_mat, tst_
```

```
Initial data dimension= 784
Retained dimesion after PCA= 50
KNN Training accuracy = 94.66666666666667
(50, 28, 28)
KNN Testing accuracy = 90.66666666666666
```





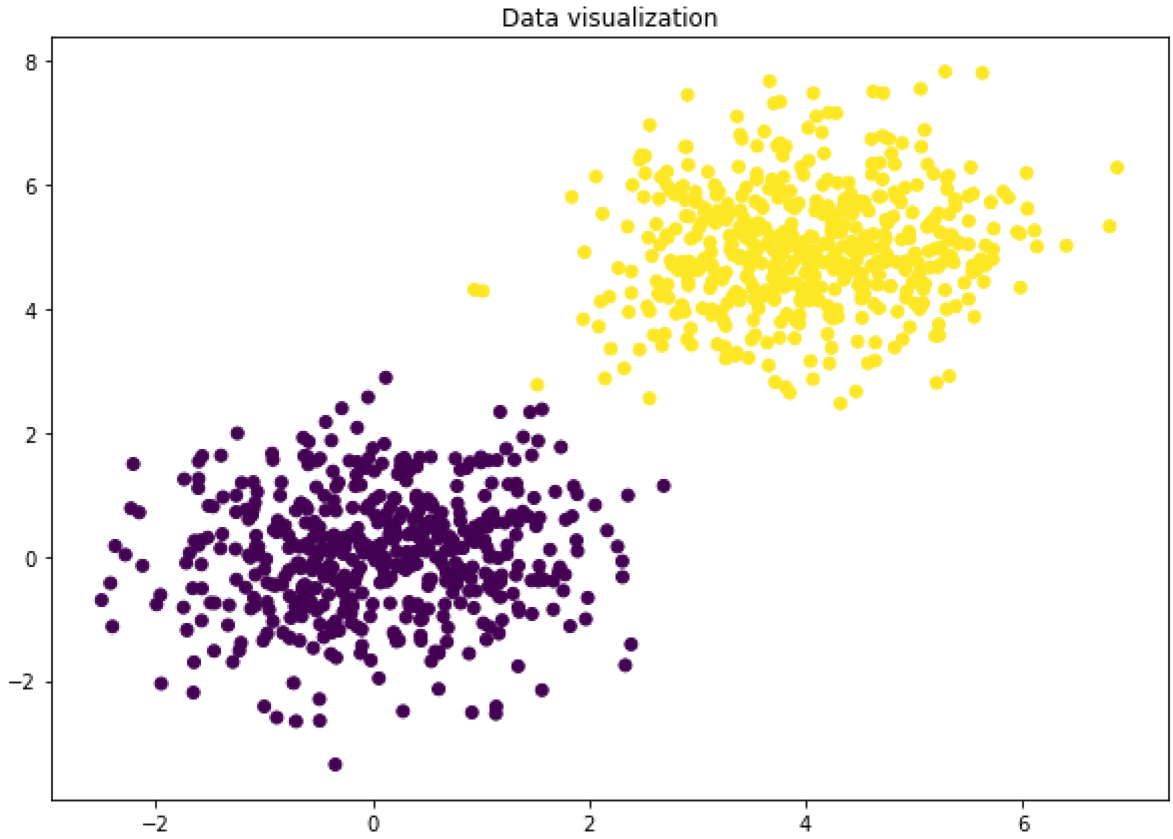


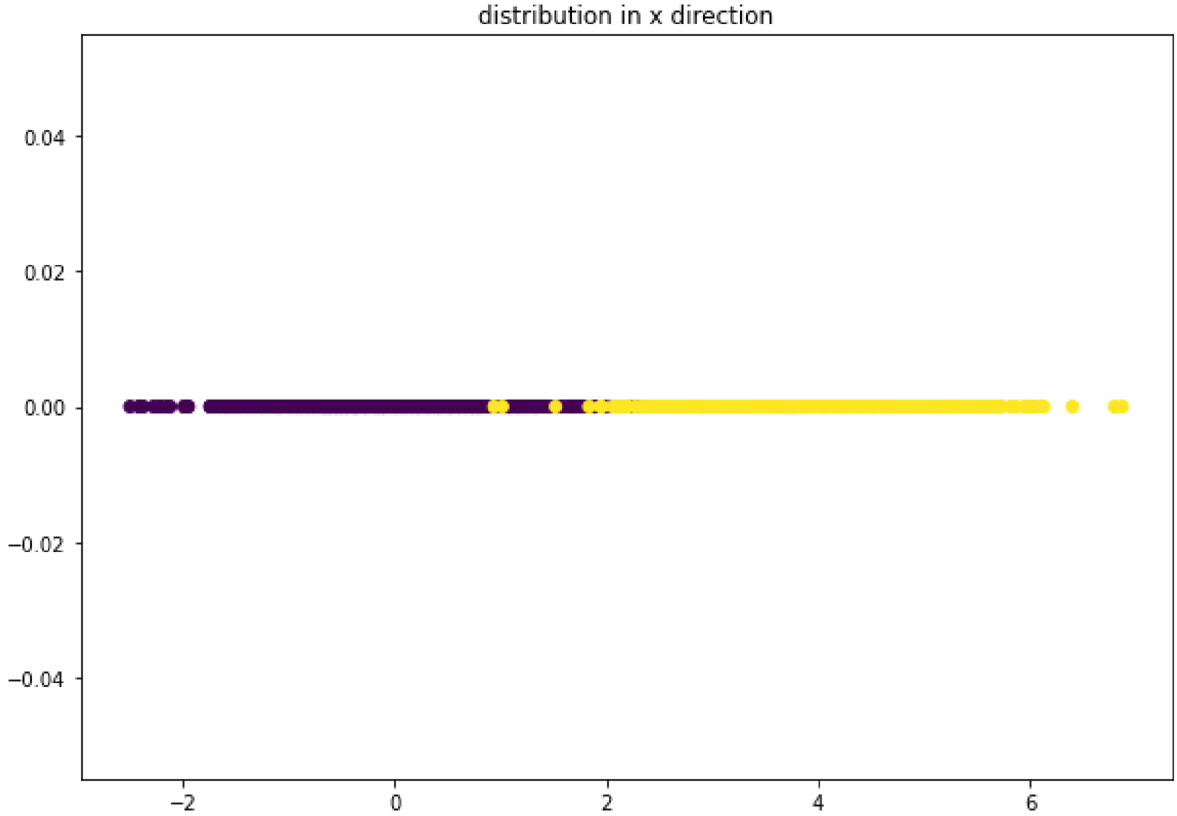
LDA

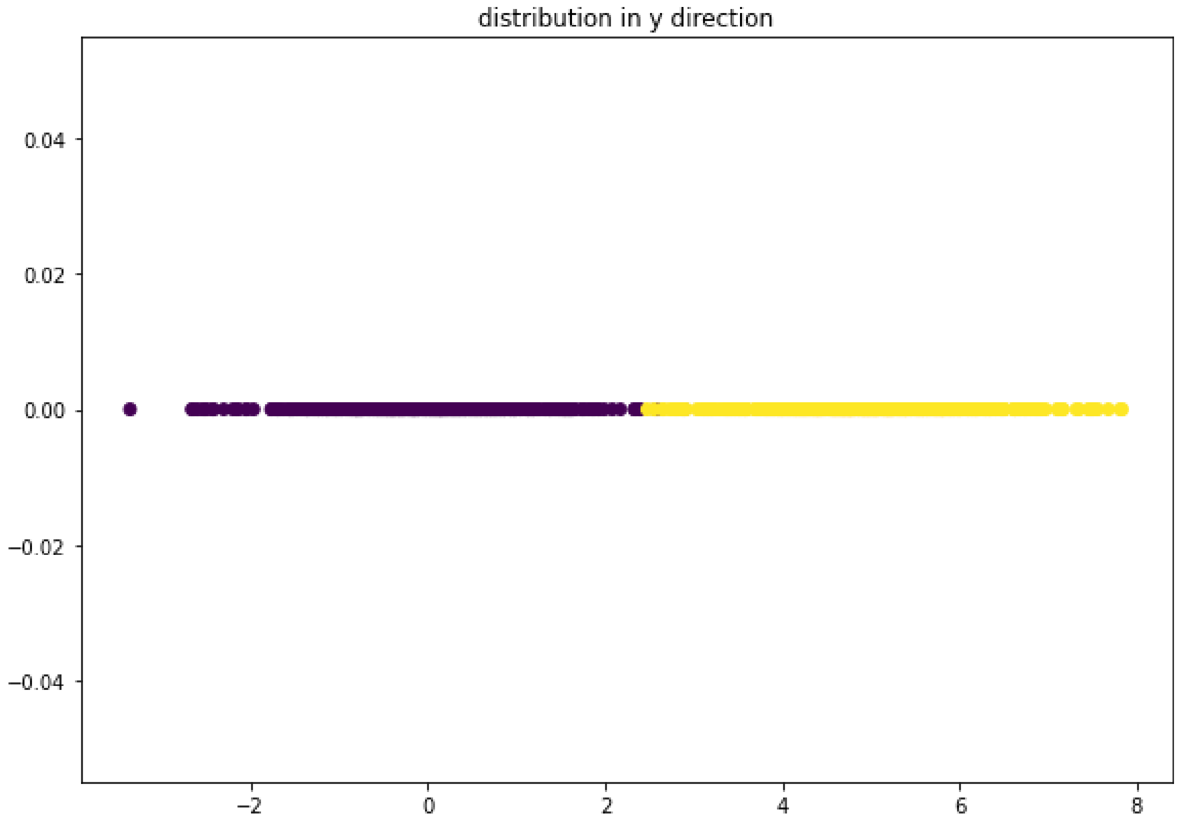
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# data generation

mean1=np.array([0,0])
mean2=np.array([4,5])
var=np.array([[1,0.1],[0.1,1]])
```

```
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data=np.concatenate((data1,data2))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in x direction')
plt.figure()
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in y direction')
```

Out[ ]:
```
Text(0.5, 1.0, 'distribution in y direction')
```







In [ ]:
```
# perform 2-class and m-class LDA
def LDA(data,label):
  id={}
  data_l={}
  mean_l={}
  cov_l={}
  S_w=np.zeros((data.shape[1],data.shape[1]))


  cls=np.unique(label)
  for i in cls:
```

```
    id[i]=np.where(label==i)[0]
    data_l[i]=data[id[i],:]
    mean_l[i]=np.mean(data_l[i],axis=0)
    cov_l[i]= ((data_l[i] - mean_l[i]).T @ (data_l[i] - mean_l[i])) / (data_l[i].sha
    S_w=S_w+cov_l[i]


  S_w=S_w/len(data_l)
  print("S_w Shape:", S_w.shape)

  if len(data_l)==2:
    print("mean Shape:", mean_l[1].shape)
    S_b= (mean_l[1] - mean_l[0]).T @ (mean_l[1] - mean_l[0]) ## Write your code here
    print("S_b Shape:", S_b.shape)
    w = np.linalg.pinv(S_w) @ (mean_l[1] - mean_l[0]).T ## Write your code here

  else:
    S_t = np.cov(data,rowvar=False)
    print("S_t Shape:", S_t.shape)
    S_b = S_t - S_w ## Write your code here
    u,s,_= np.linalg.svd(np.linalg.pinv(S_w) @ S_b) ## Write your code here
    print("u Shape:", u.shape)
    print(s)
    max_index = max(len(data_l)-1, int(u.shape[1]*0.2))
    w=u[:, :max_index]

  print("w Shape:", w.shape)

  return w
```

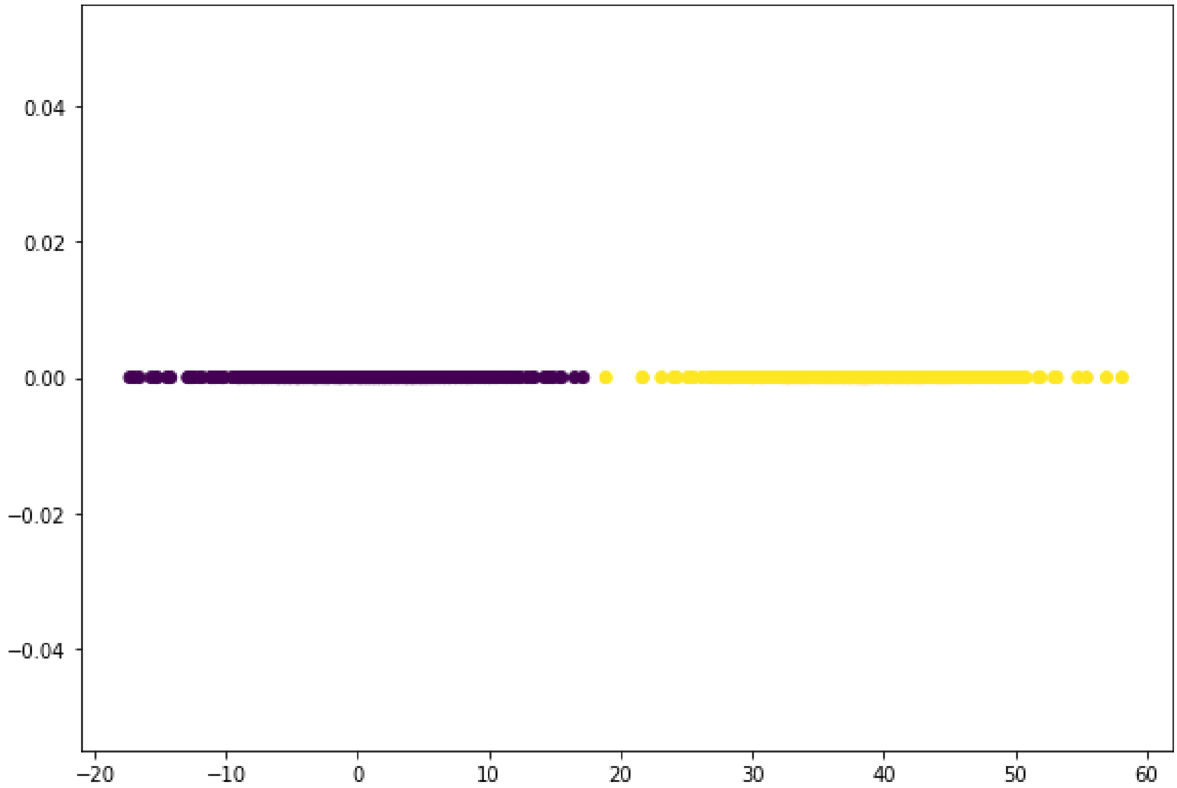
In [ ]:

```
# after LDA projection

w = LDA(data, label)
print(data.shape, label.shape)
plt.figure()
plt.scatter(data @ w, np.zeros(data.shape[0]),c=label)
```

```
S_w Shape: (2, 2)
mean Shape: (2,)
S_b Shape: ()
w Shape: (2,)
(1000, 2) (1000,)
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x1a784522560>
```



In [ ]:
```
# Classification using :LDA
# Use k-nearest neighbour classifier (Scikit Learn) after dimensionality reduction

## Write your code here

from sklearn.neighbors import KNeighborsClassifier
k=5
knn = KNeighborsClassifier(n_neighbors=k)
trans_data = (data @ w).reshape((-1, 1))

knn.fit(trans_data, label)

print('KNN Training accuracy =',knn.score(trans_data, label)*100)

# test data
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data=np.concatenate((data1,data2))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

trans_data = (data @ w).reshape((-1, 1))

print('KNN Testing accuracy =',knn.score(trans_data, tst_label)*100)
```

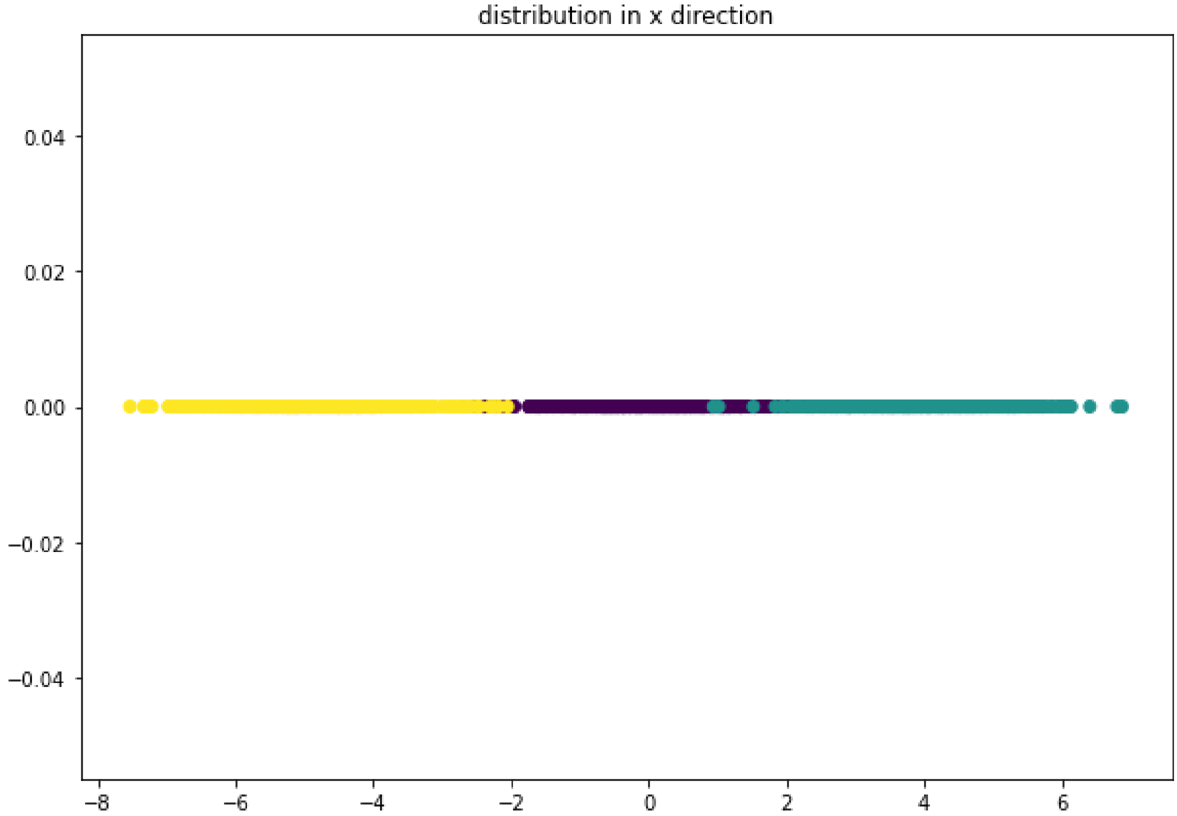
```
KNN Training accuracy = 100.0
KNN Testing accuracy = 100.0
```

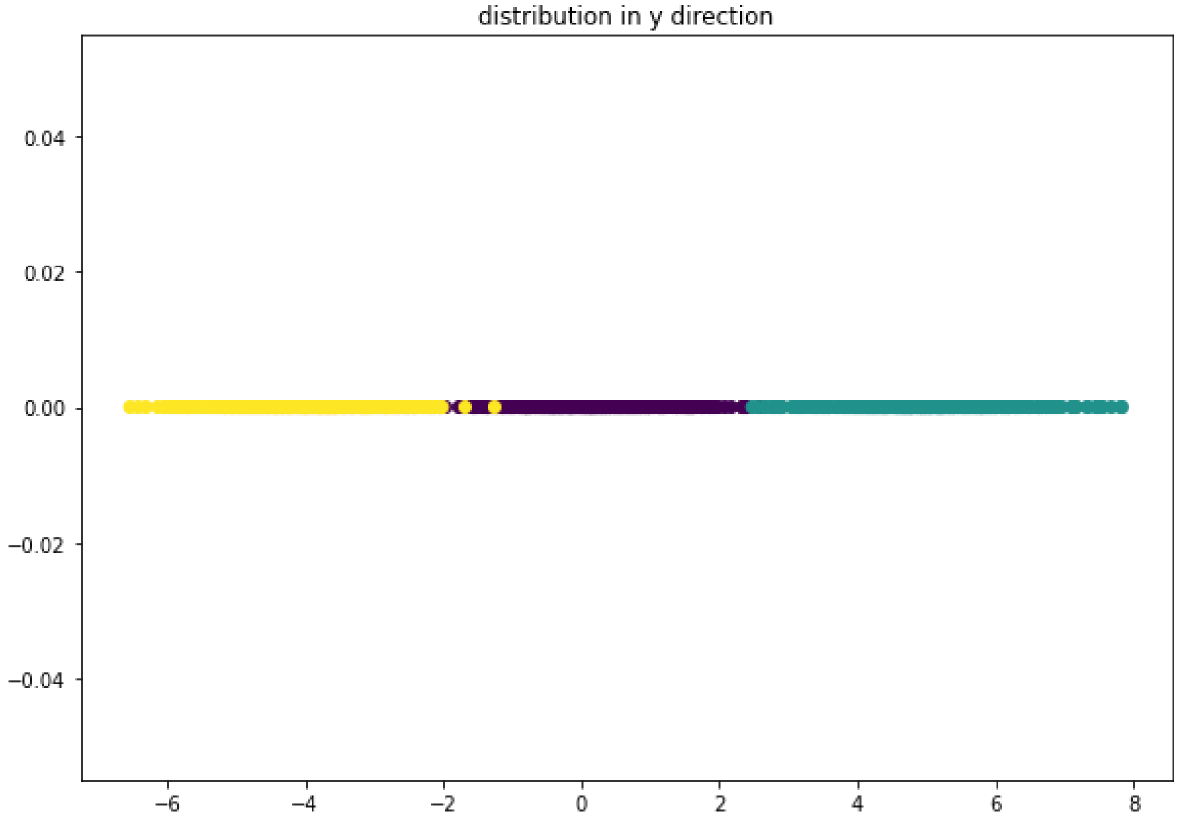
LDA Multiclass

In [ ]:
```
mean1=np.array([0,0])
mean2=np.array([4,5])
mean3=np.array([-5,-4])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
```

```
data3=np.random.multivariate_normal(mean3,var,500)
data=np.concatenate((data1,data2,data3))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0]),np.ones(data3

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in x direction')
plt.figure()
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in y direction')
```

Out[ ]: `Text(0.5, 1.0, 'distribution in y direction')`





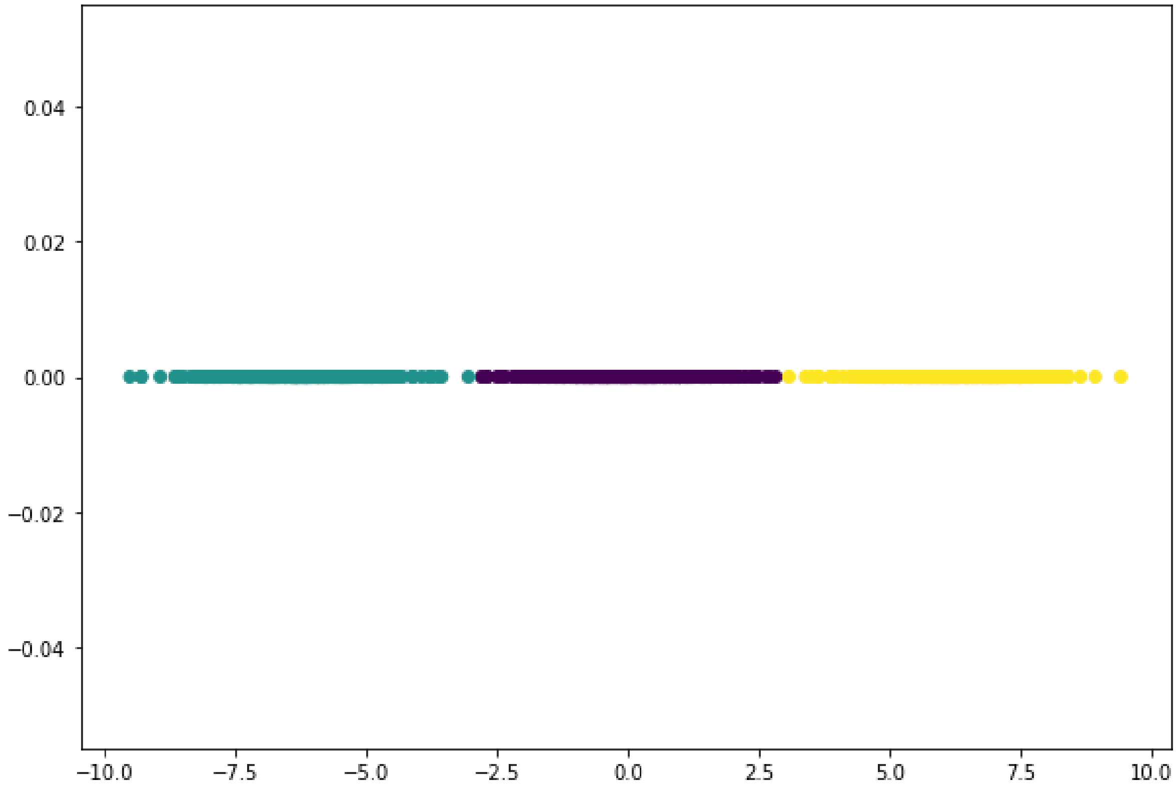
In [ ]:

```
# after projection
w=LDA(data,label)
print(w.shape)
plt.figure()
plt.scatter(data @ w[:,0],np.zeros(data.shape[0]),c=label) # by performing 1D projec
```

```
S_w Shape: (2, 2)
S_t Shape: (2, 2)
u Shape: (2, 2)
[25.44474058  0.13632272]
w Shape: (2, 2)
(2, 2)
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x1a785809150>
```



In [ ]:

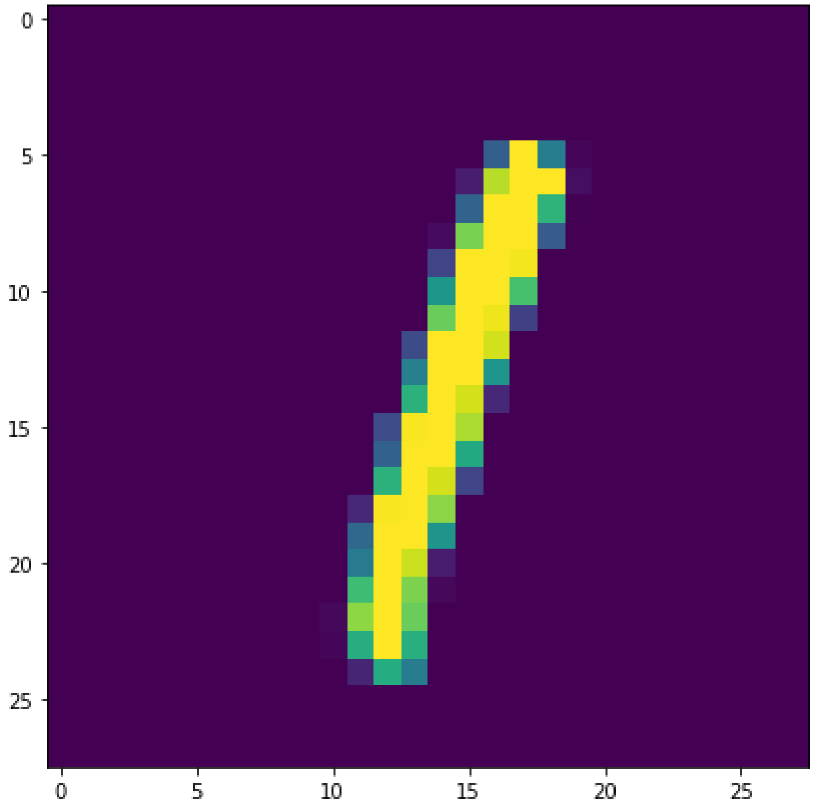
```
# Testing (using KNN)
# Use k-nearest neighbour classifier (Scikit Learn) after dimensionality reduction

## Write your code here

from sklearn.neighbors import KNeighborsClassifier
k=5
knn = KNeighborsClassifier(n_neighbors=k)
trans_data = data @ w

knn.fit(trans_data, label)

print('KNN Training accuracy =',knn.score(trans_data, label)*100)

# test data
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data=np.concatenate((data1,data2))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

trans_data = data @ w

print('KNN Testing accuracy =',knn.score(trans_data, tst_label)*100)
```

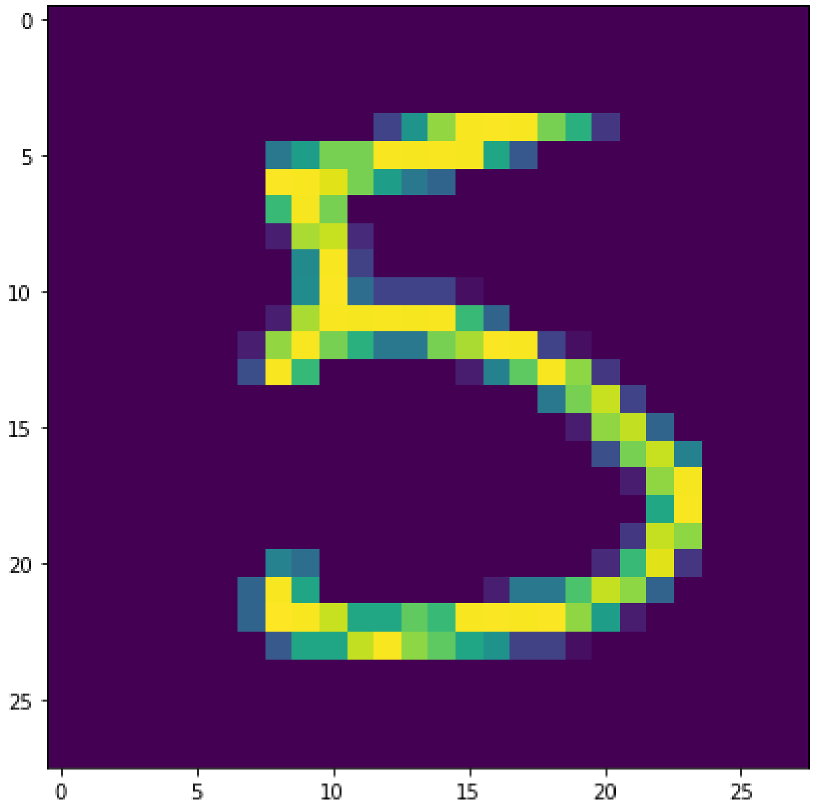
```
KNN Training accuracy = 99.93333333333332
KNN Testing accuracy = 100.0
```

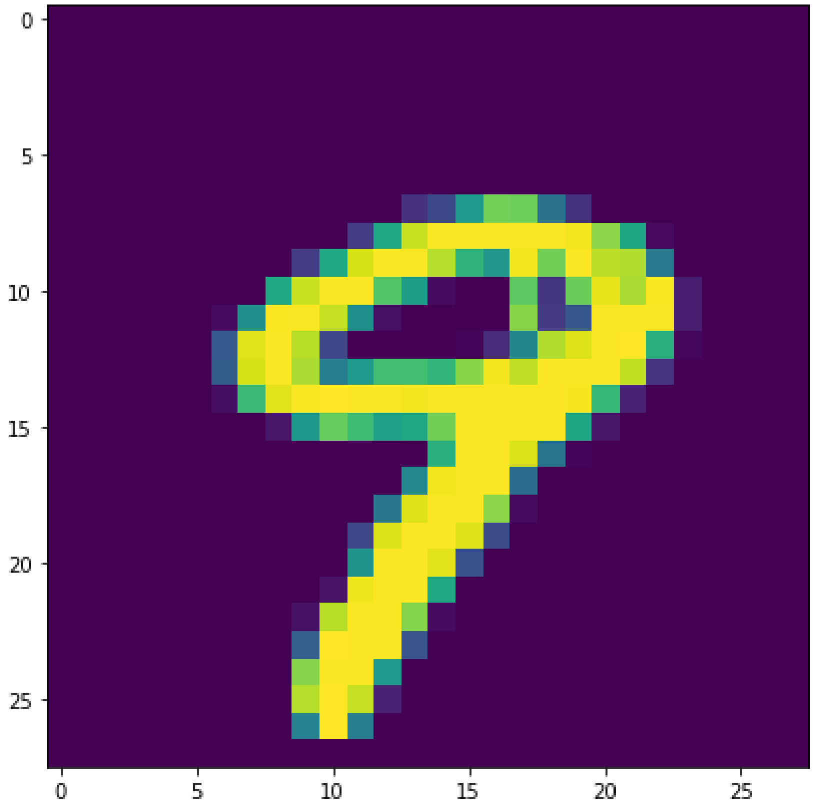
Perform LDA on MNIST and Classify using the data of any 3 classes

In [ ]:
```
## Write your code here
cl = [1, 5, 9]

id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[:50]
Im_1=Images[id1]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[:50]
Im_5=Images[id5]
lab_5=labels[id5]

id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[:50]
Im_9=Images[id9]
lab_9=labels[id9]

plt.imshow(Im_1[1])
plt.figure()
plt.imshow(Im_5[1])
plt.figure()
plt.imshow(Im_9[1])

#print(Im_5.shape)

data=np.concatenate((Im_1,Im_5,Im_9))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
G_lab=np.concatenate((lab_1,lab_5,lab_9))
print(G_lab.shape)

data = data.astype('float32')
```

```
(150, 784)
(150,)
```


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



In [ ]:
```
print('Initial data dimension=',data.shape[1])
w = LDA(data, G_lab)

trans_data = data @ w
print('Retained dimesion after LDA=',trans_data.shape[1])
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, G_lab)

print('KNN Training accuracy =',knn.score(trans_data,G_lab)*100)


## testing
## data preparation
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[100:150]
Im_1=Images[id1]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[100:150]
Im_5=Images[id5]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[100:150]
Im_9=Images[id9]
lab_9=labels[id9]

plt.imshow(Im_1[1])
plt.figure()
plt.imshow(Im_5[1])
```

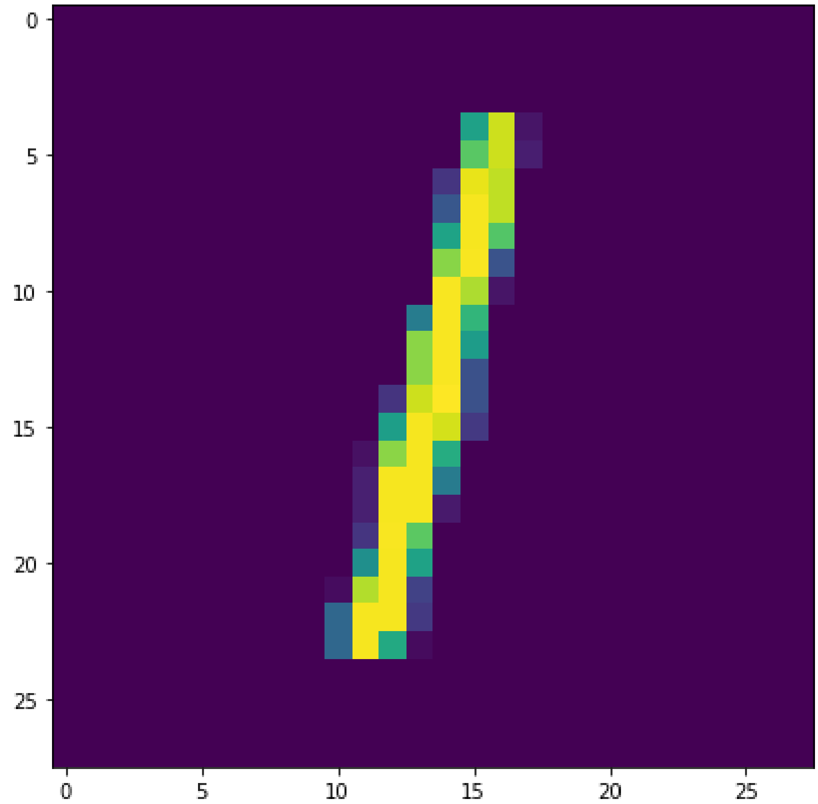
```
plt.figure()
plt.imshow(Im_9[1])

print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5,Im_9))
data_tst=np.reshape(data_tst,(data_tst.shape[0],data_tst.shape[1]*data_tst.shape[2])

tst_lab=np.concatenate((lab_1,lab_5,lab_9))


# final testing
print('KNN Testing accuracy =', knn.score(data_tst @ w, tst_lab)*100)
```

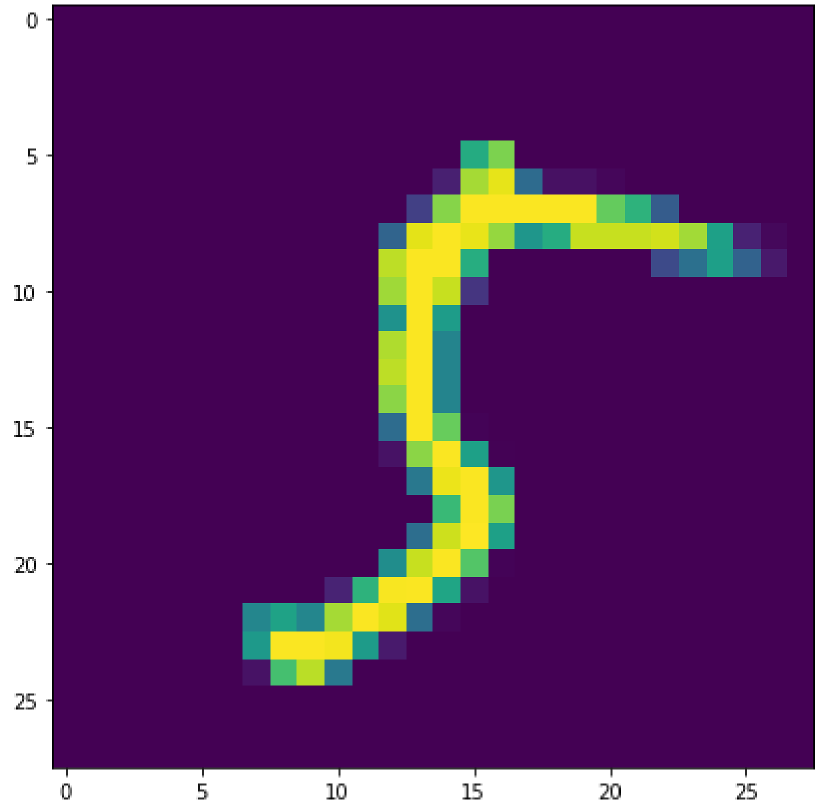
```
Initial data dimension= 784
S_w Shape: (784, 784)
S_t Shape: (784, 784)
u Shape: (784, 784)
[1.46050290e+10 5.34680679e+09 2.24347056e+03 6.75006455e+02
 4.34327414e+02 2.86929102e+02 2.19858780e+02 1.49953942e+02
 9.89413888e+01 8.25454711e+01 5.75857866e+01 5.03469018e+01
 4.33379733e+01 3.84586094e+01 3.13796519e+01 2.95175218e+01
 2.75203697e+01 2.55127917e+01 2.42411898e+01 2.12988629e+01
 1.98217655e+01 1.85170929e+01 1.64743726e+01 1.50146611e+01
 1.44561260e+01 1.39497350e+01 1.31461480e+01 1.21373600e+01
 1.14867511e+01 1.08950899e+01 1.03220136e+01 9.88573378e+00
 9.24290521e+00 8.93920063e+00 8.69936247e+00 8.24120616e+00
 7.93800801e+00 7.35709688e+00 7.14681889e+00 6.88867370e+00
 6.52186669e+00 6.31452610e+00 6.05385207e+00 5.91487317e+00
 5.65459379e+00 5.44898901e+00 5.29487796e+00 5.00564498e+00
 4.73157480e+00 4.64250044e+00 4.60733667e+00 4.42000696e+00
 4.35240259e+00 4.13711440e+00 4.08327399e+00 4.02886507e+00
 3.95915708e+00 3.78073565e+00 3.66811758e+00 3.62206018e+00
 3.48540911e+00 3.43724987e+00 3.35467453e+00 3.22005460e+00
 3.17280180e+00 3.16286028e+00 3.00555229e+00 2.95878130e+00
 2.94537011e+00 2.87549032e+00 2.81861269e+00 2.74337994e+00
 2.69689654e+00 2.65948859e+00 2.65213287e+00 2.61677972e+00
 2.59084357e+00 2.49538665e+00 2.43790951e+00 2.38888897e+00
 2.35650537e+00 2.34347771e+00 2.30949773e+00 2.29099138e+00
 2.21491917e+00 2.17968395e+00 2.14048303e+00 2.10729025e+00
 2.05709414e+00 2.04098682e+00 2.02661806e+00 1.95471065e+00
 1.94325965e+00 1.91312846e+00 1.89155365e+00 1.87461669e+00
 1.83063349e+00 1.82177125e+00 1.80939111e+00 1.78132189e+00
 1.77664720e+00 1.71783825e+00 1.69511579e+00 1.66437329e+00
 1.64035336e+00 1.61254465e+00 1.59910673e+00 1.57194008e+00
 1.55765663e+00 1.55240996e+00 1.52314524e+00 1.51076030e+00
 1.49621497e+00 1.47551364e+00 1.47417584e+00 1.45037854e+00
 1.43952215e+00 1.42290485e+00 1.40748171e+00 1.39322510e+00
 1.38238873e+00 1.37891666e+00 1.34584417e+00 1.33479052e+00
 1.32477912e+00 1.31304992e+00 1.29892313e+00 1.29867830e+00
 1.28550380e+00 1.26832212e+00 1.26384894e+00 1.25467628e+00
 1.24149554e+00 1.22968027e+00 1.21930260e+00 1.20333283e+00
 1.19414240e+00 1.18600047e+00 1.17567194e+00 1.17298717e+00
 1.16519245e+00 1.15089671e+00 1.14684656e+00 1.13219838e+00
 1.12872667e+00 1.12562640e+00 1.10536995e+00 1.09681907e+00
 1.06253701e+00 1.00000072e+00 1.00000072e+00 1.00000072e+00
 1.00000071e+00 1.00000065e+00 1.00000020e+00 1.00000019e+00
 1.00000011e+00 1.00000010e+00 1.00000009e+00 1.00000007e+00
 1.00000005e+00 1.00000005e+00 1.00000005e+00 1.00000005e+00
 1.00000004e+00 1.00000004e+00 1.00000004e+00 1.00000004e+00
 1.00000004e+00 1.00000003e+00 1.00000003e+00 1.00000003e+00
 1.00000003e+00 1.00000003e+00 1.00000002e+00 1.00000002e+00
 1.00000002e+00 1.00000002e+00 1.00000002e+00 1.00000002e+00
 1.00000002e+00 1.00000002e+00 1.00000002e+00 1.00000002e+00
 1.00000002e+00 1.00000002e+00 1.00000002e+00 1.00000002e+00
 1.00000002e+00 1.00000001e+00 1.00000001e+00 1.00000001e+00
 1.00000001e+00 1.00000001e+00 1.00000001e+00 1.00000001e+00
 1.00000001e+00 1.00000001e+00 1.00000001e+00 1.00000001e+00
 1.00000001e+00 1.00000001e+00 1.00000001e+00 1.00000001e+00
 1.00000001e+00 1.00000001e+00 1.00000001e+00 1.00000001e+00
 1.00000001e+00 1.00000001e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 9.99999999e-01 9.99999999e-01
 9.99999999e-01 9.99999999e-01 9.99999998e-01 9.99999998e-01
 9.99999996e-01 9.99999996e-01 9.99999996e-01 9.99999995e-01
```

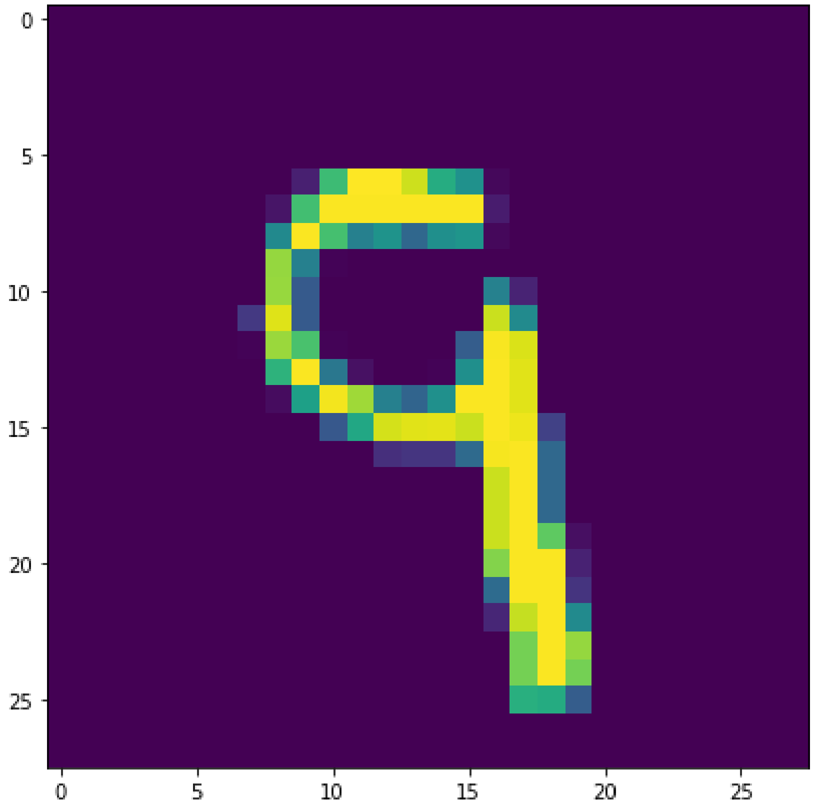
```
 9.99999994e-01 9.99999994e-01 9.99999993e-01 9.99999992e-01
 9.99999992e-01 9.99999992e-01 9.99999992e-01 9.99999991e-01
 9.99999991e-01 9.99999991e-01 9.99999990e-01 9.99999990e-01
 9.99999990e-01 9.99999990e-01 9.99999990e-01 9.99999989e-01
 9.99999988e-01 9.99999987e-01 9.99999986e-01 9.99999986e-01
 9.99999985e-01 9.99999985e-01 9.99999985e-01 9.99999984e-01
 9.99999984e-01 9.99999983e-01 9.99999982e-01 9.99999981e-01
 9.99999981e-01 9.99999981e-01 9.99999979e-01 9.99999979e-01
 9.99999978e-01 9.99999978e-01 9.99999977e-01 9.99999976e-01
 9.99999975e-01 9.99999975e-01 9.99999975e-01 9.99999974e-01
 9.99999971e-01 9.99999970e-01 9.99999968e-01 9.99999967e-01
 9.99999964e-01 9.99999964e-01 9.99999959e-01 9.99999954e-01
 9.99999954e-01 9.99999951e-01 9.99999951e-01 9.99999950e-01
 9.99999935e-01 9.99999925e-01 9.99999897e-01 9.99999893e-01
 9.99999801e-01 9.99999642e-01 9.99999355e-01 9.99999290e-01
 9.99999283e-01 9.99999280e-01 9.99999280e-01 8.99759502e-01
 8.27688483e-01 1.22589134e-02 1.21385134e-02 1.19196504e-02
 1.18509327e-02 1.17051645e-02 1.16699937e-02 1.15391206e-02
 1.14615498e-02 1.13892248e-02 1.13639333e-02 1.12565181e-02
 1.11776456e-02 1.10729529e-02 1.09984227e-02 1.08561590e-02
 1.07544647e-02 1.06119327e-02 1.05729915e-02 1.04498869e-02
 1.03713407e-02 1.03261528e-02 1.02106232e-02 1.01979490e-02
 1.00799774e-02 9.97379812e-03 9.80123969e-03 9.71606811e-03
 9.60162940e-03 9.53254358e-03 9.41599897e-03 9.37482469e-03
 9.29662152e-03 9.14975571e-03 9.10096119e-03 8.97883184e-03
 8.89932136e-03 8.83213040e-03 8.63817510e-03 8.63064335e-03
 8.54699040e-03 8.43266124e-03 8.37120720e-03 8.19584263e-03
 8.04739701e-03 7.95371346e-03 7.83560446e-03 7.58183297e-03
 7.54539737e-03 7.41647162e-03 7.37652912e-03 7.31227120e-03
 7.15412748e-03 7.12502123e-03 6.99364449e-03 6.95964028e-03
 6.88743847e-03 6.66326926e-03 6.56731461e-03 6.55744986e-03
 6.34981675e-03 6.28884102e-03 6.21497527e-03 6.13956246e-03
 5.86987668e-03 5.80413096e-03 5.73509112e-03 5.71835986e-03
 5.65393608e-03 5.55224520e-03 5.38920221e-03 5.20253974e-03
 5.14969258e-03 5.09920114e-03 5.04628182e-03 4.96694606e-03
 4.93823211e-03 4.75798151e-03 4.70760715e-03 4.55854959e-03
 4.53393297e-03 4.41754818e-03 4.27195711e-03 4.22362343e-03
 4.15339952e-03 3.99798660e-03 3.90331183e-03 3.85988547e-03
 3.70714261e-03 3.65475534e-03 3.58764622e-03 3.44350887e-03
 3.33454667e-03 3.28616961e-03 3.23721739e-03 3.06325258e-03
 3.04446873e-03 2.91243828e-03 2.87448331e-03 2.80998563e-03
 2.68835456e-03 2.56071510e-03 2.48396755e-03 2.37063138e-03
 2.28583751e-03 2.19163039e-03 2.12727315e-03 2.05647715e-03
 1.95750612e-03 1.87571361e-03 1.81399510e-03 1.77779681e-03
 1.67204538e-03 1.53887255e-03 1.52443154e-03 1.46220766e-03
 1.40170045e-03 1.28956755e-03 1.23352972e-03 1.16077622e-03
 1.10105265e-03 9.99370038e-04 9.44128254e-04 9.16157818e-04
 9.01208462e-04 8.08293526e-04 7.36528103e-04 7.07102487e-04
 6.47503916e-04 5.94274004e-04 5.45665299e-04 4.98337801e-04
 4.61298781e-04 4.27692589e-04 3.61828275e-04 3.14158703e-04
 2.67718928e-04 2.40795089e-04 1.63891544e-04 1.37406092e-04
 9.95343265e-05 6.94683759e-05 5.68445889e-05 3.57044621e-05
 2.58867946e-05 6.48301514e-06 4.93556604e-06 1.73006957e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
```

```
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
```

```
 1.43951618e-06 1.43951618e-06 1.43951618e-06 1.43951618e-06
 1.43951618e-06 1.43951618e-06 1.43944362e-06 1.21216867e-06
 1.20210452e-06 1.07273580e-06 3.89437387e-07 3.64090532e-07
 2.47312025e-07 2.05029629e-07 1.95480135e-07 1.35314288e-07
 1.32404593e-07 1.20036596e-07 7.41814806e-08 3.86927508e-08
 3.48569163e-08 3.39473747e-08 2.14695006e-08 1.80451001e-08
 1.15860207e-08 5.34711506e-09 1.99140180e-12 9.12205708e-13
 8.60881897e-13 4.17212224e-13 3.88828612e-13 3.50597930e-13]
w Shape: (784, 156)
Retained dimesion after LDA= 156
KNN Training accuracy = 100.0
(50, 28, 28)
KNN Testing accuracy = 94.0
```








```
In [ ]:
```