LAB 9 : Dimensionality Reduction

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

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
jtplot.style(theme='gruvboxd',context='notebook',grid=False,ticks=
```

PCA

PCA deals with projection of higher dimension data to lower dim.Project data only along the directions of maximum variance. $Cov(X,Y) = \frac{1}{M}\sum_{i=1}^{M}(X_i - \mu_{Xi})(Y_i - \mu_{Yi})$

Σ = [Cov(X, Y)N×N], N × N matrix.

Perform singular value decomposition (SVD) on Σ

[U, S, V] = SVD(Σ)

If we want to reduce dimension from N to K, then consider only first K cols resulting in UK which is N × K matrix.

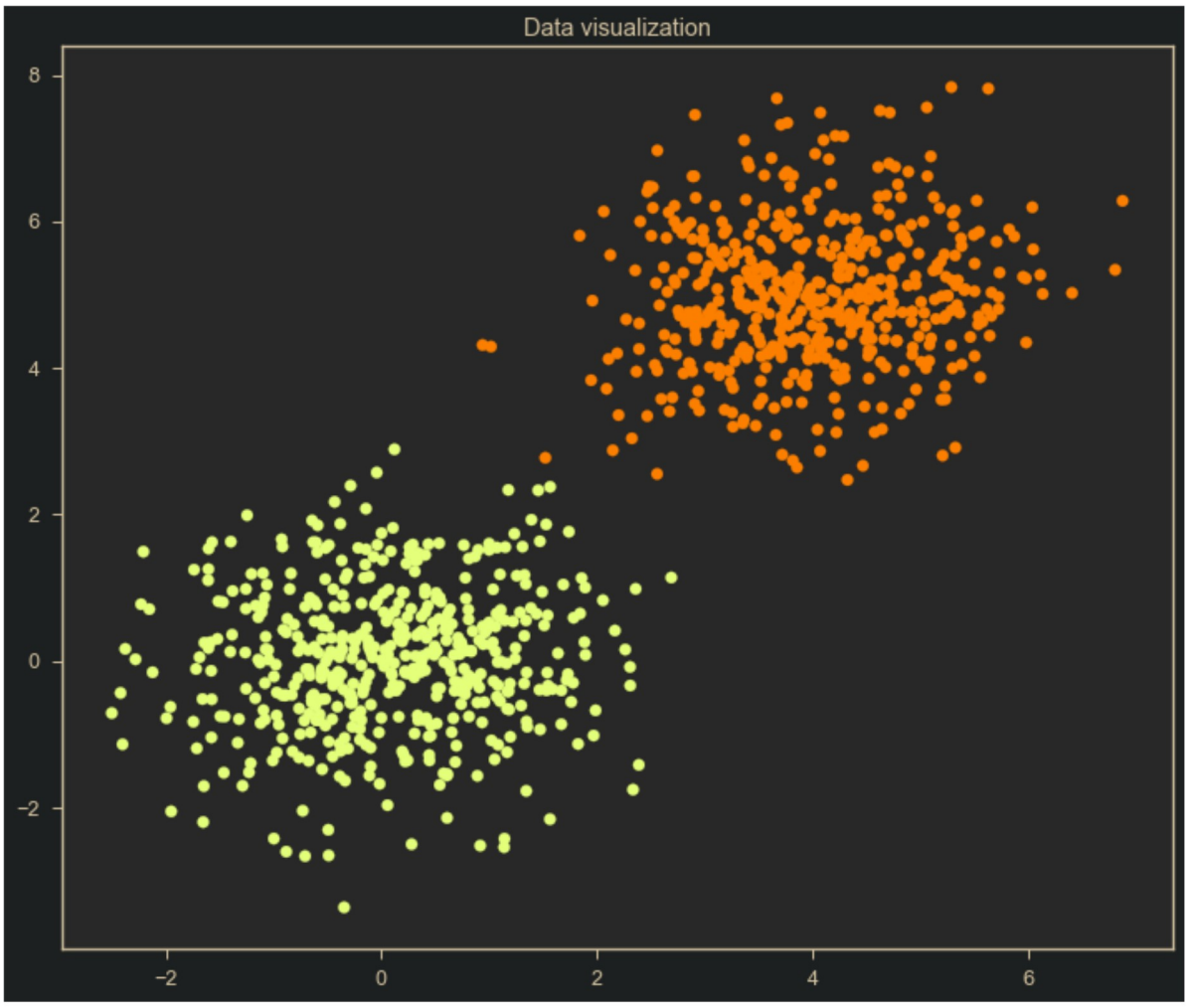
Let X be an M × N matrix

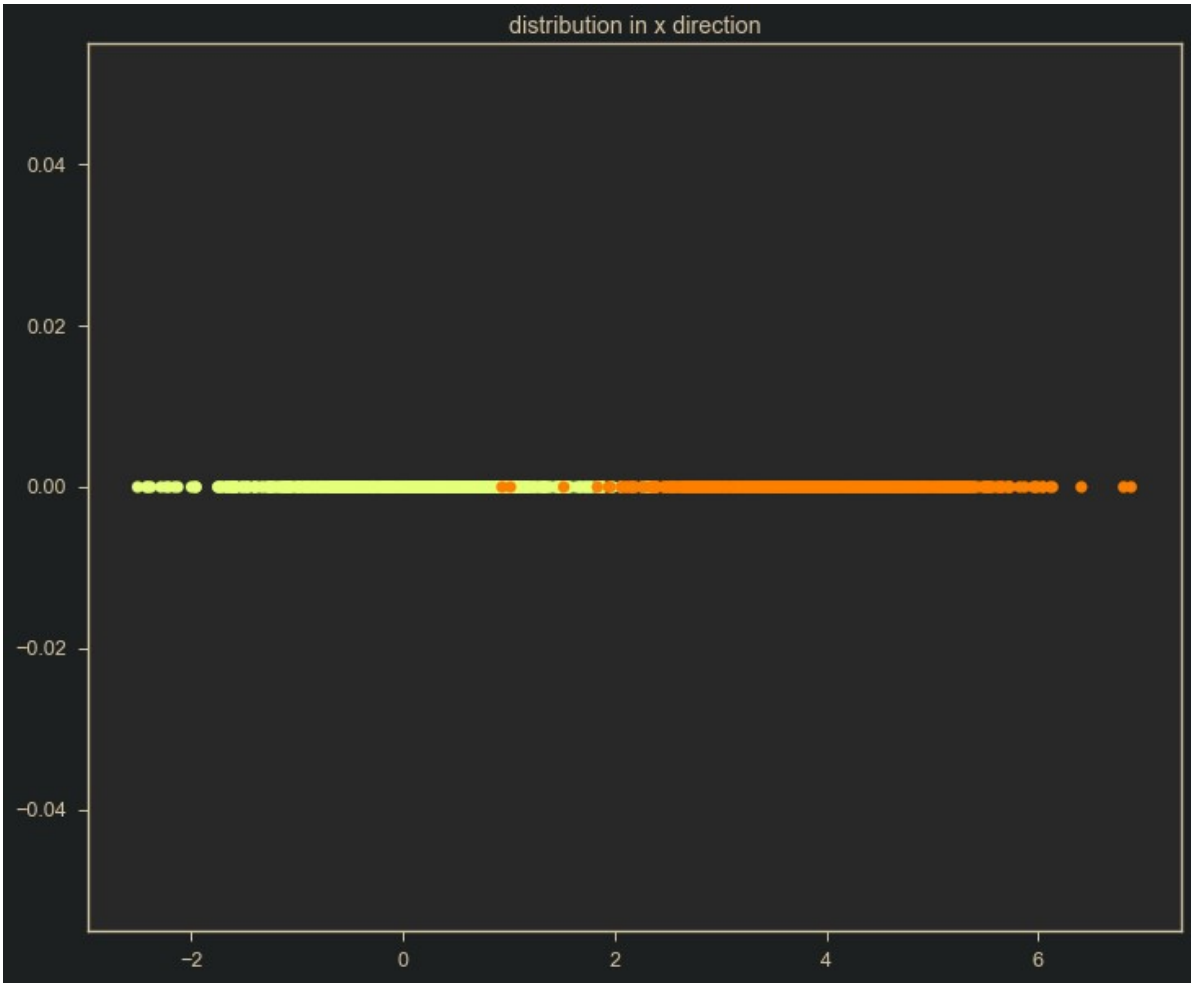
Compressed matrix Z = X × UK

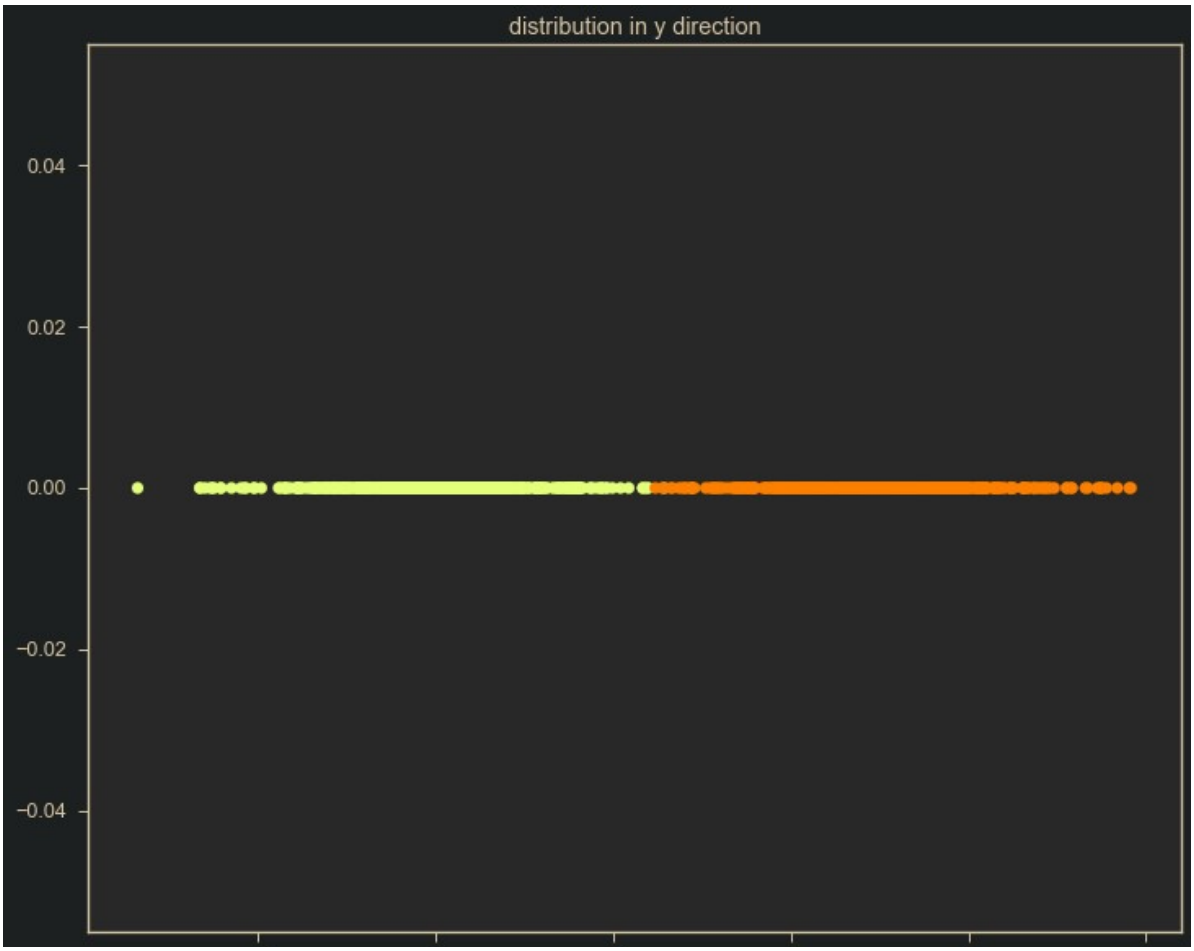
Z will be a M × K matrix.

In [2]:

```
# 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
#plotting data
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],data[:,1],c=label,cmap='Wistia');
plt.title('Data visualization');
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label,cmap='Wistia
plt.title('distribution in x direction');
plt.figure(figsize=(12,10))
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label,cmap='Wistia
plt.title('distribution in y direction');
```

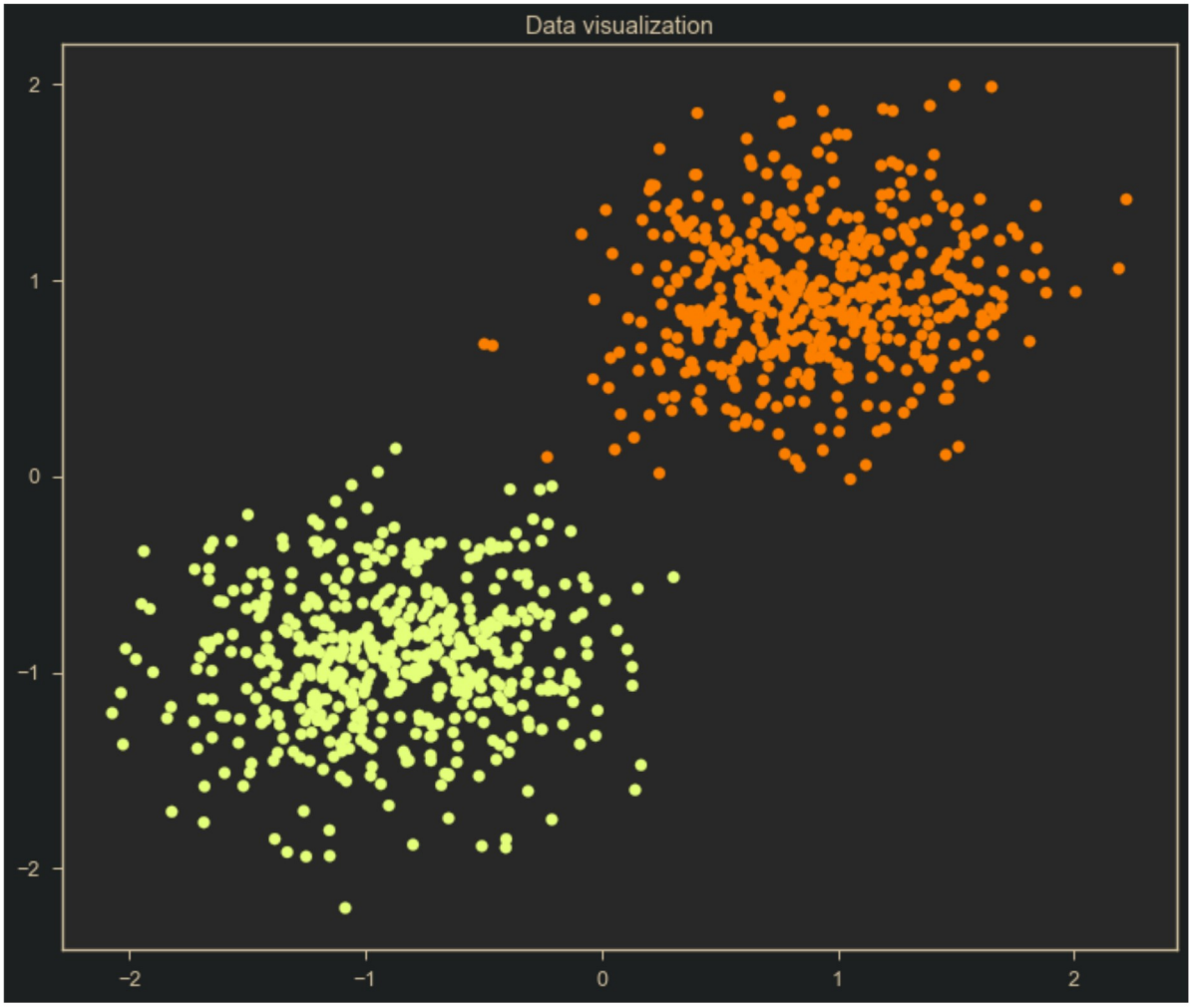



distribution in x direction
0.04
0.02
0.00
−0.02
−0.04
−2
0
2
4
6


distribution in y direction
0.04
0.02
0.00
−0.02
−0.04

In [3]:

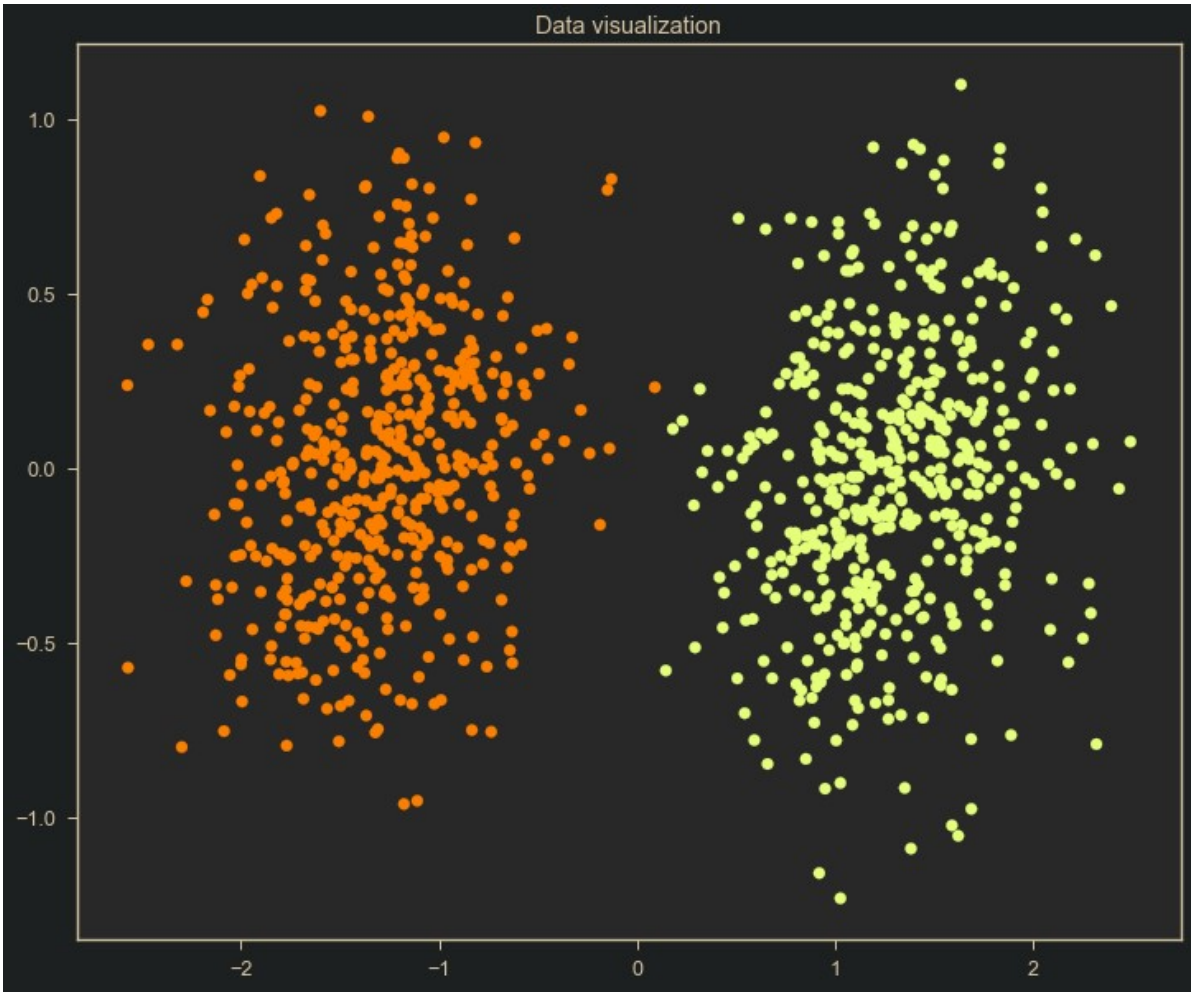
```
# Data normalization
data1=(data-np.mean(data,axis=0));
data=data1/(np.std(data1,axis=0)+10**(-30)); # avoid divide by
zero
# Perform data normalization here using mean substraction and
std division
## Write your code here
#plotting data after normalisation
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],data[:,1],c=label,cmap='Wistia');
plt.title('Data visualization');
```

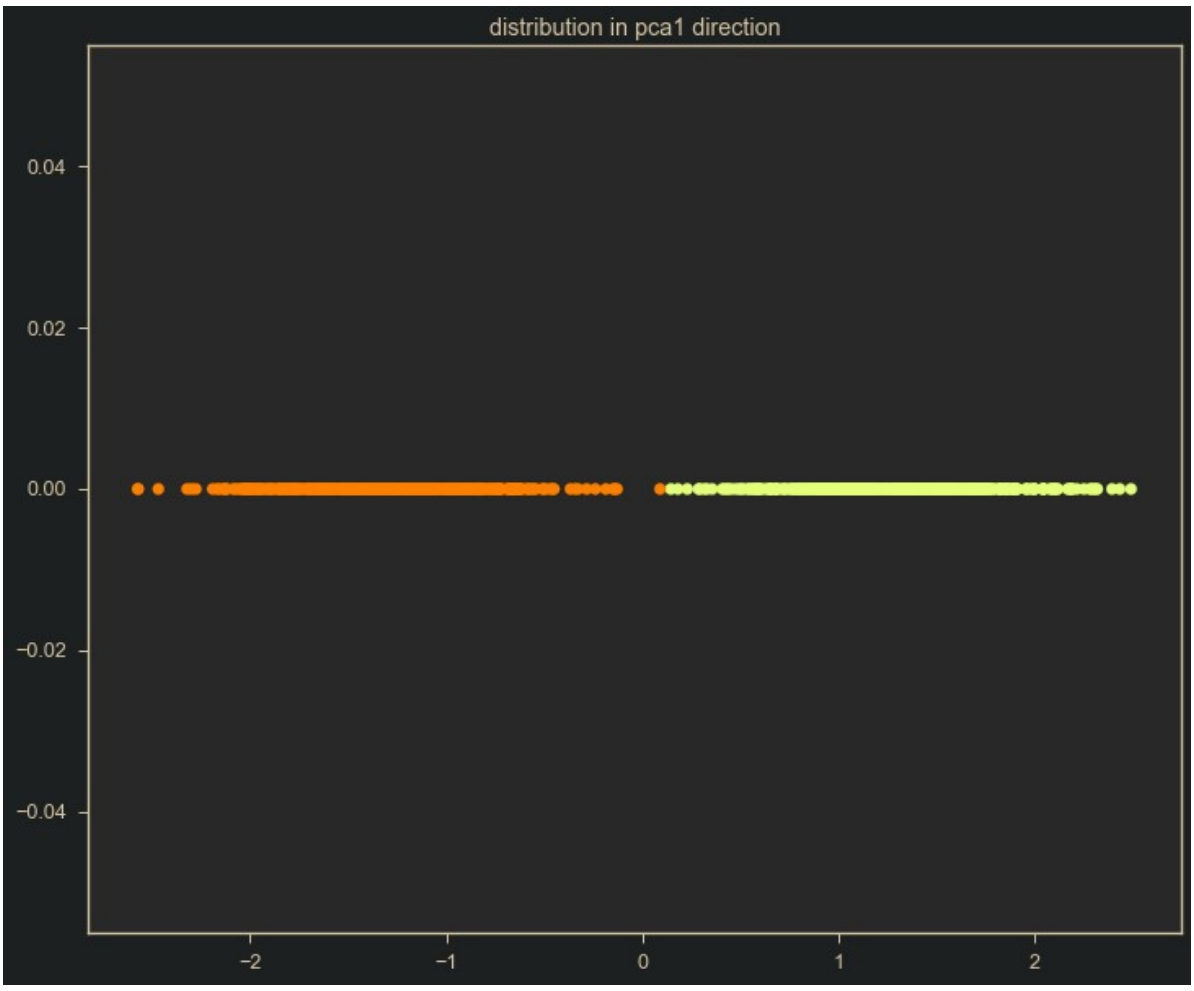


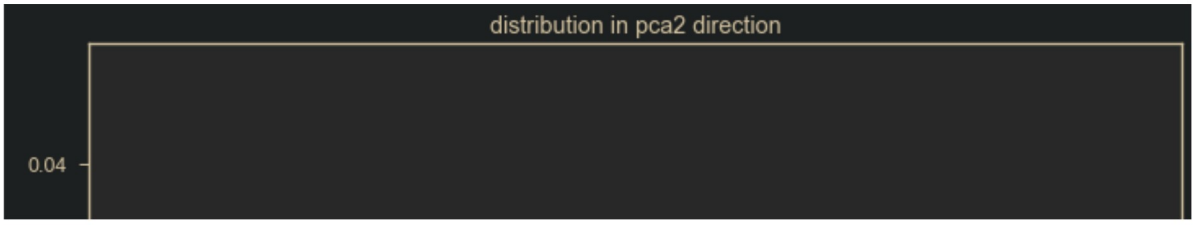
In [4]:

```
# PCA

# coverance matrix
cov=data.T @ data;

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

trans_data = data@(u.T);#projecting data

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);
# plotting transformed data
plt.figure(figsize=(12,10))
plt.scatter(trans_data[:,0],trans_data[:,1],c=label,cmap='Wistia')
plt.title('Data visualization');
plt.figure(figsize=(12,10))
plt.scatter(trans_data[:,0],np.zeros(data.shape[0]),c=label,cmap='
plt.title('distribution in pca1 direction');
plt.figure(figsize=(12,10))
plt.scatter(trans_data[:,1],np.zeros(data.shape[0]),c=label,cmap='
plt.title('distribution in pca2 direction');
```

```
variance along pca1 direction= 1.8477663843459726
variance along pca2 direction= 0.15223361565402704
```


Data visualization
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2


distribution in pca1 direction
0.04
0.02
0.00
−0.02
−0.04
−2
−1
0
1
2

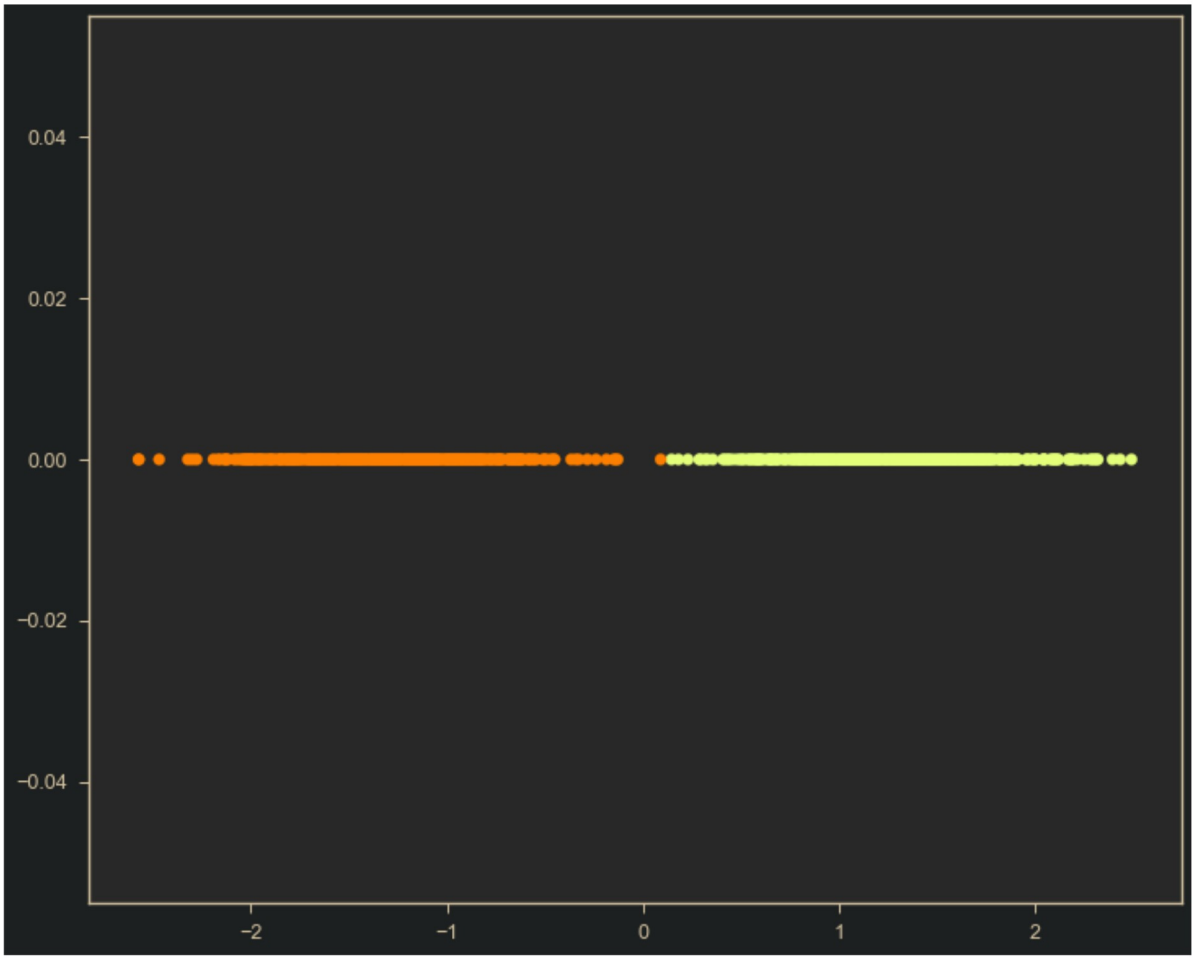


In [5]:

```
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);
        ## Write your code here
        t_var = np.sum(np.square(s));
        for ind in range(s.shape[0]):
            var = np.sum(np.square(s[:ind]));
            if (var >= 0.9*t_var):
                break
        return ind
    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 [6]:

```
# pca transformation
PCA=pca(data=data);
trans_data,trans_mat=PCA.pca_comp(data);
#plots
plt.figure(figsize=(12,10))
plt.scatter(trans_data,np.zeros(trans_data.shape),c=label,cmap="Wi
```



In [7]:

```
#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.s

print('KNN Testing accuracy =',knn.score(PCA.pre_process(data) @
trans_mat,tst_label)*100)
```

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

PCA on MNIST

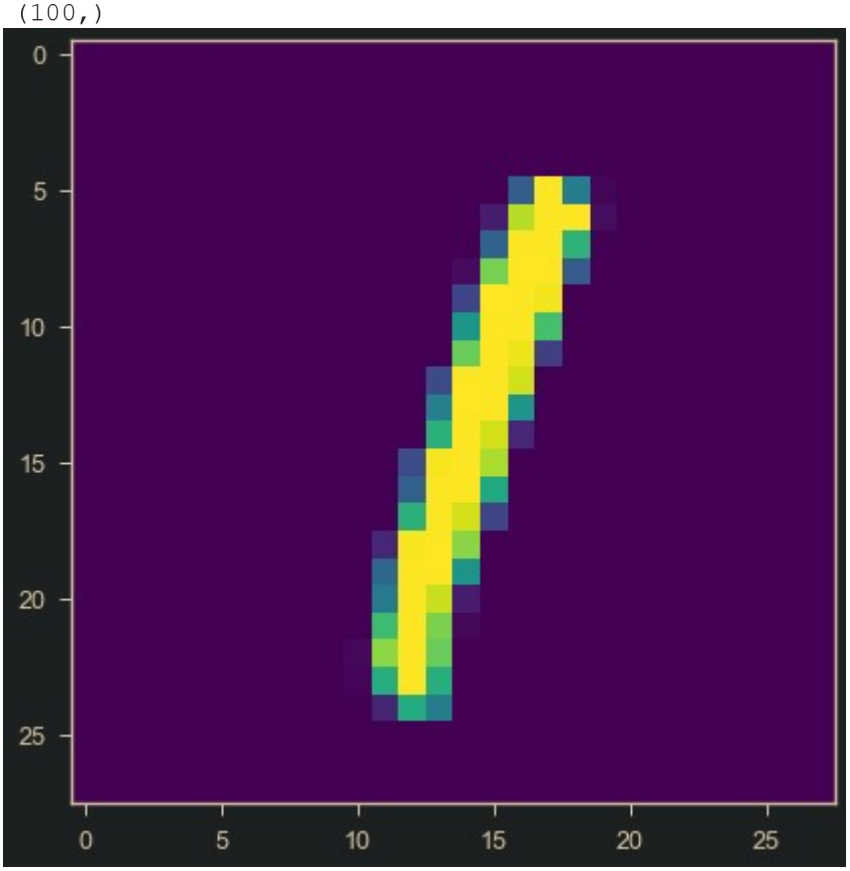
In [8]:

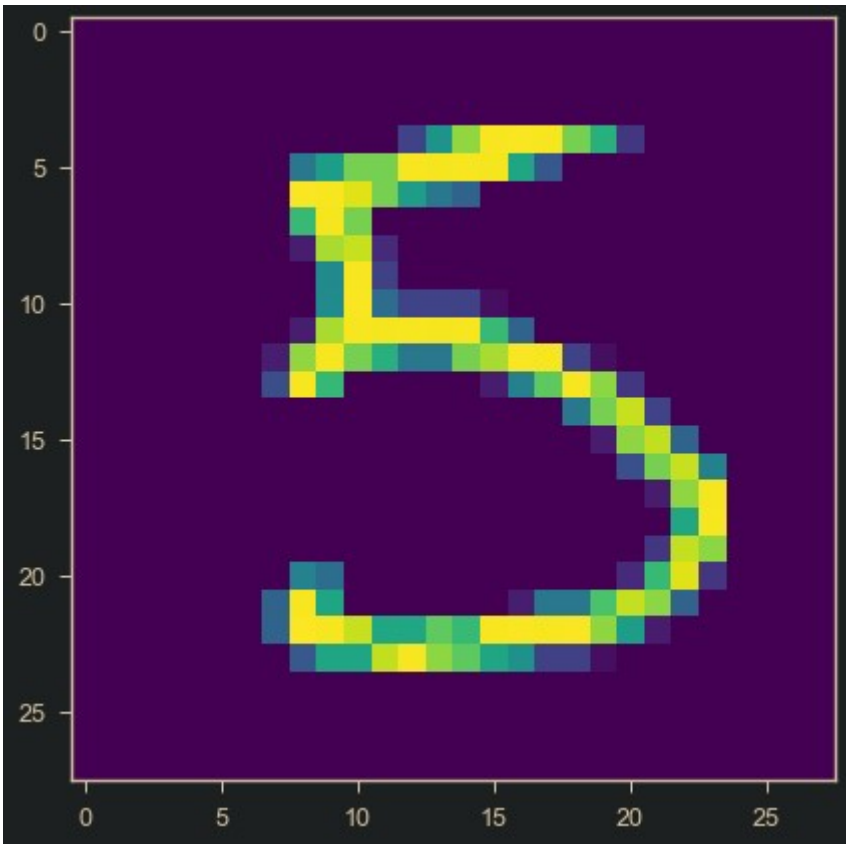
```
# !pip install idx2numpy
```

In [9]:

```
# MNIST data

file1='C:/Users/R3M0/Documents/EE 413 Pattern Recognition and
Machine Learning Laboratory/9/t10k-images-idx3-ubyte'; ## Change
the path accordingly
file2='C:/Users/R3M0/Documents/EE 413 Pattern Recognition and
Machine Learning Laboratory/9/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]
# showing
plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])

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)
```

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

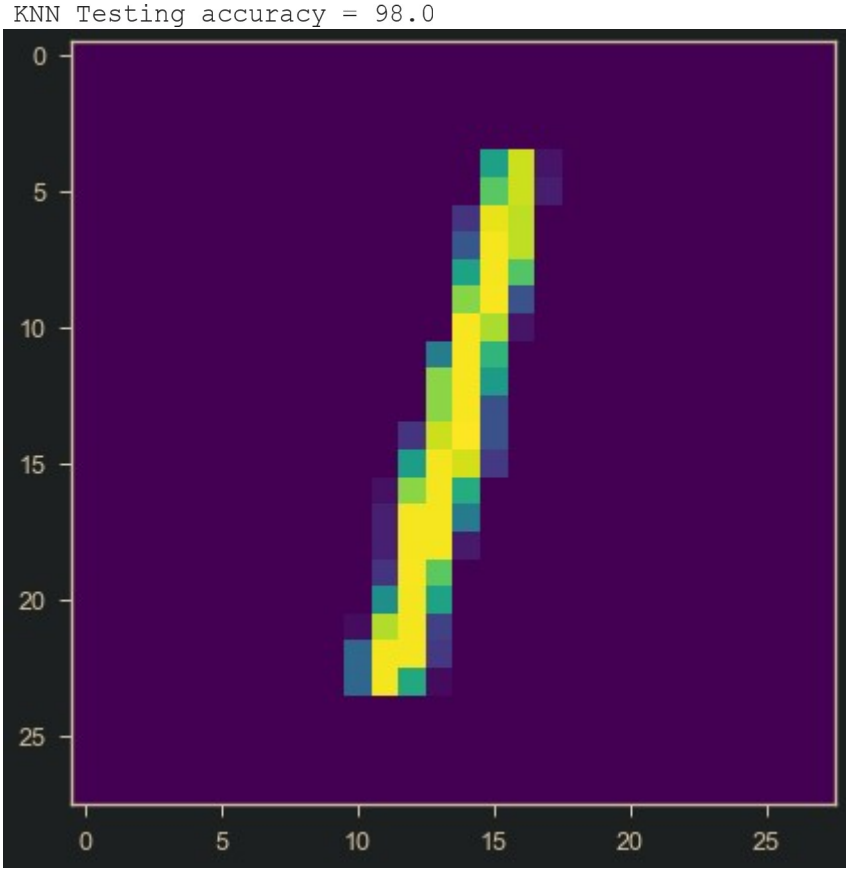


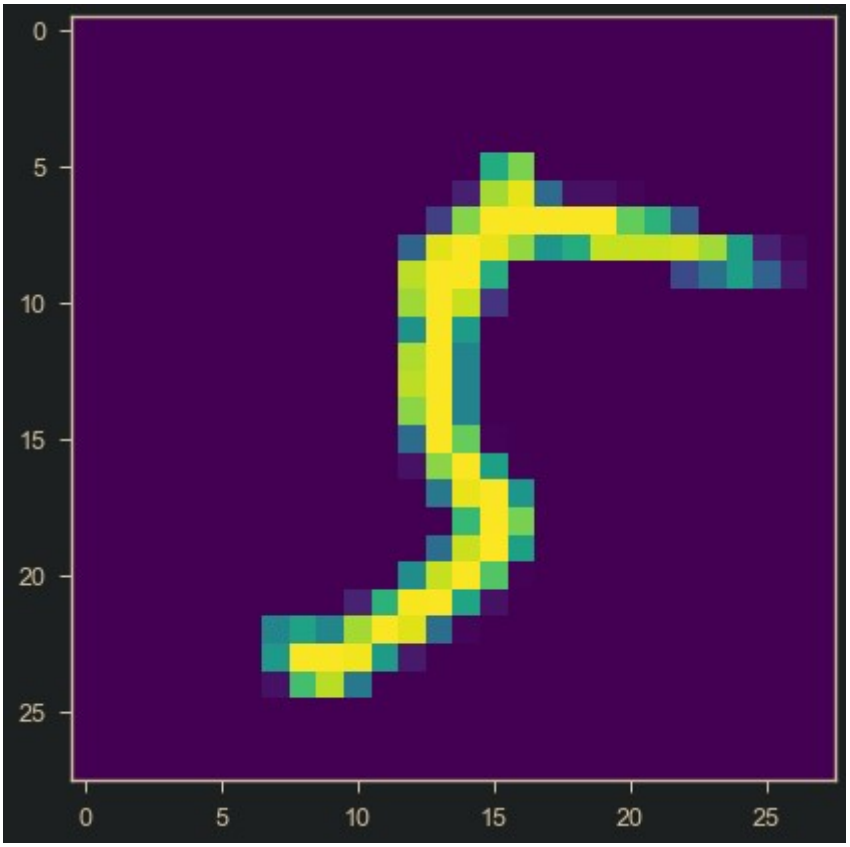


In [10]:

```
#applying knn to reduced data features
print('Initial data dimension=',data.shape[1])
PCA=pca(data=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]
#plotting 1 image from each test class
plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])

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_lab)*100)
```

```
Initial data dimension= 784
Retained dimesion after PCA= 10
KNN Training accuracy = 100.0
KNN Testing accuracy = 98.0
```



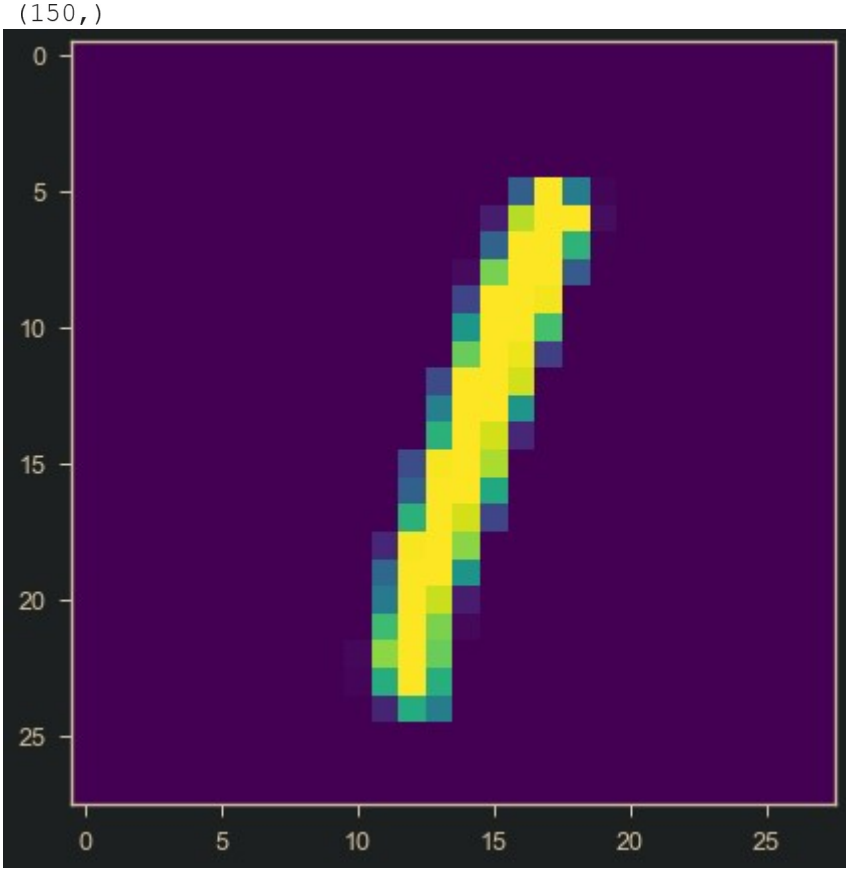


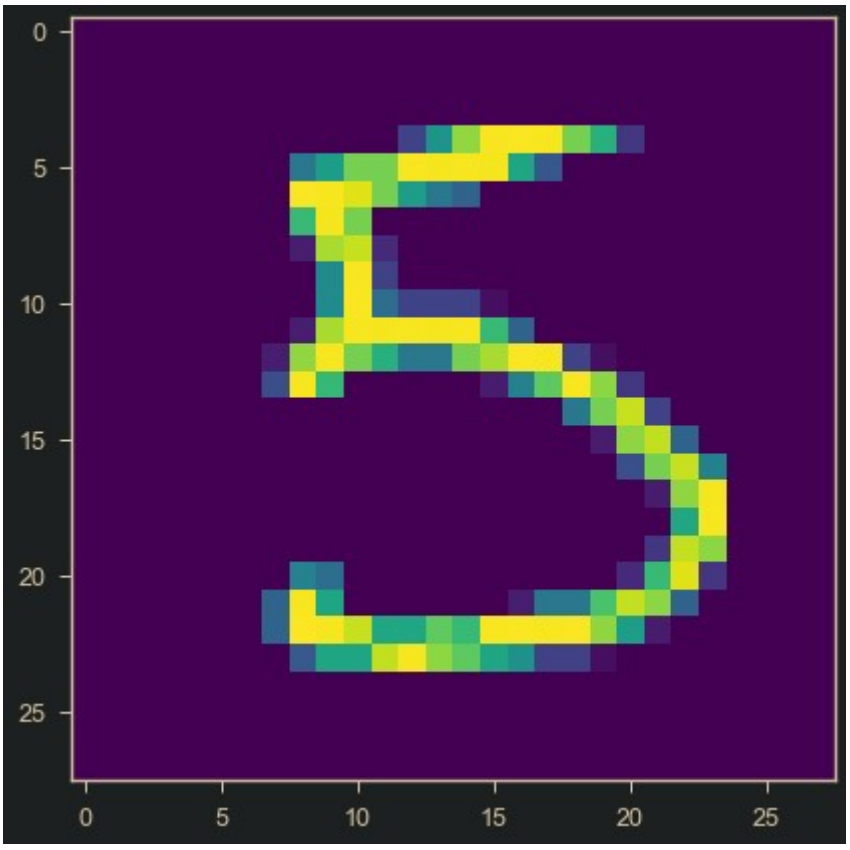
Perform PCA on MNIST and Classify taking the data with any 3 Classes

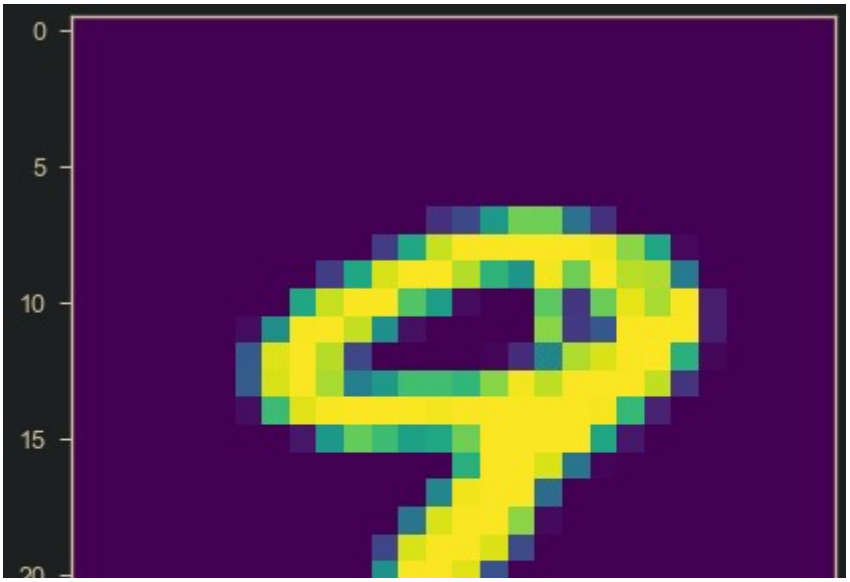
In [11]:

```
## Write your code here
#class array
cl=[1,5,9]

# 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]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[:50]
Im_9=Images[id9,:,:]
lab_9=labels[id9]
# plotting 1 image from each class
plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])

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')

data /= 255
```

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





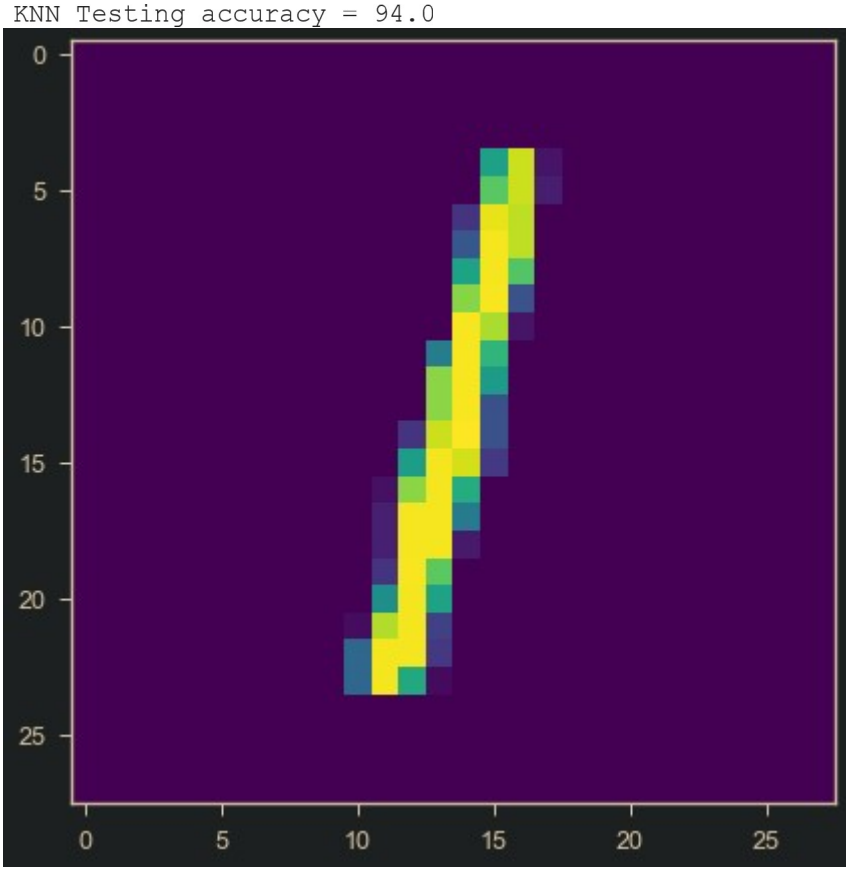

0
5
10
15
20

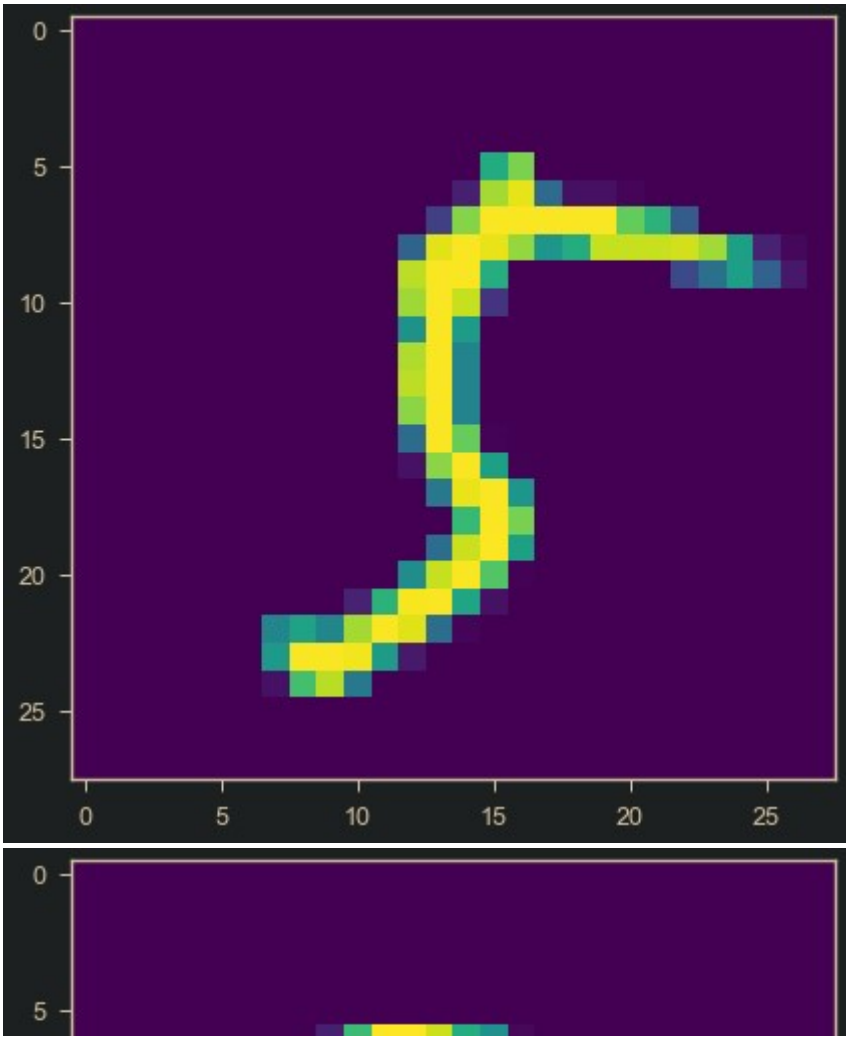
In [12]:

```
# applying knn on reduced data
print('Initial data dimension=',data.shape[1])
PCA=pca(data=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]
#plotting some images from each class
plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])

data_tst=np.concatenate((Im_1,Im_5,Im_9))
```

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

```
Initial data dimension= 784
Retained dimesion after PCA= 13
KNN Training accuracy = 97.33333333333334
KNN Testing accuracy = 94.0
```





LDA

LDA is like PCA. However, while reducing dimensions, it focuses on maximizing the separability among classes.

Given a set of samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$, and their class labels $y_1, \ldots, y_n$:

The within-class scatter matrix is defined as: $S_W = \sum_{i=1}^{n} (x_i - \mu_{y_i})(x_i - \mu_{y_i})^T$

Here, $\boldsymbol{\mu}_k$ is the sample mean of the k-th class. The between-class scatter matrix is defined as: $S_B = (\mu_1 - \mu_2)(\mu_1^T - \mu_2^T)$ for 2 classses

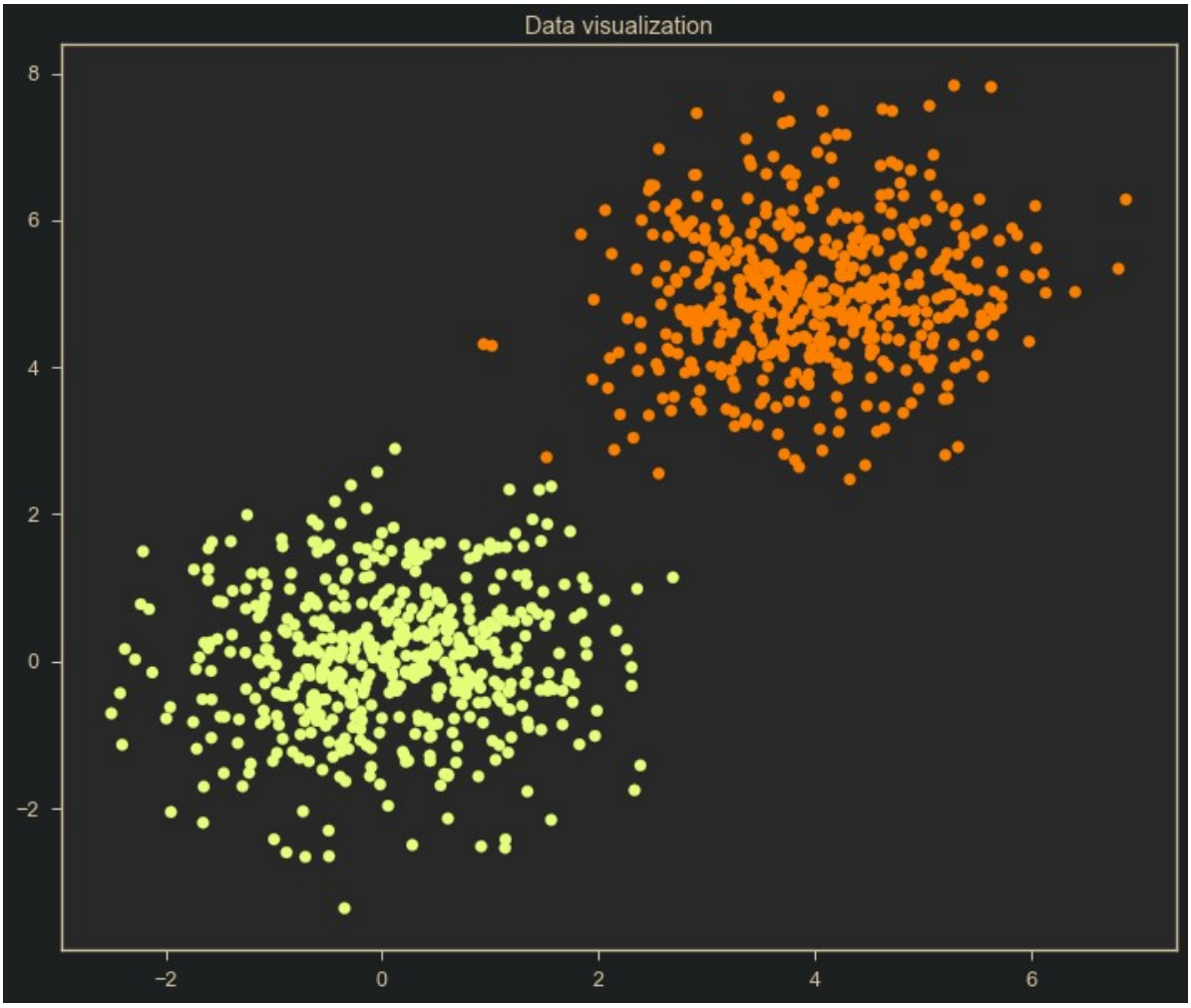
The between-class scatter matrix is defined as: $S_B = \sum_{k=1}^{m} n_k(\mu_k - \mu)(\mu_k - \mu)^T$

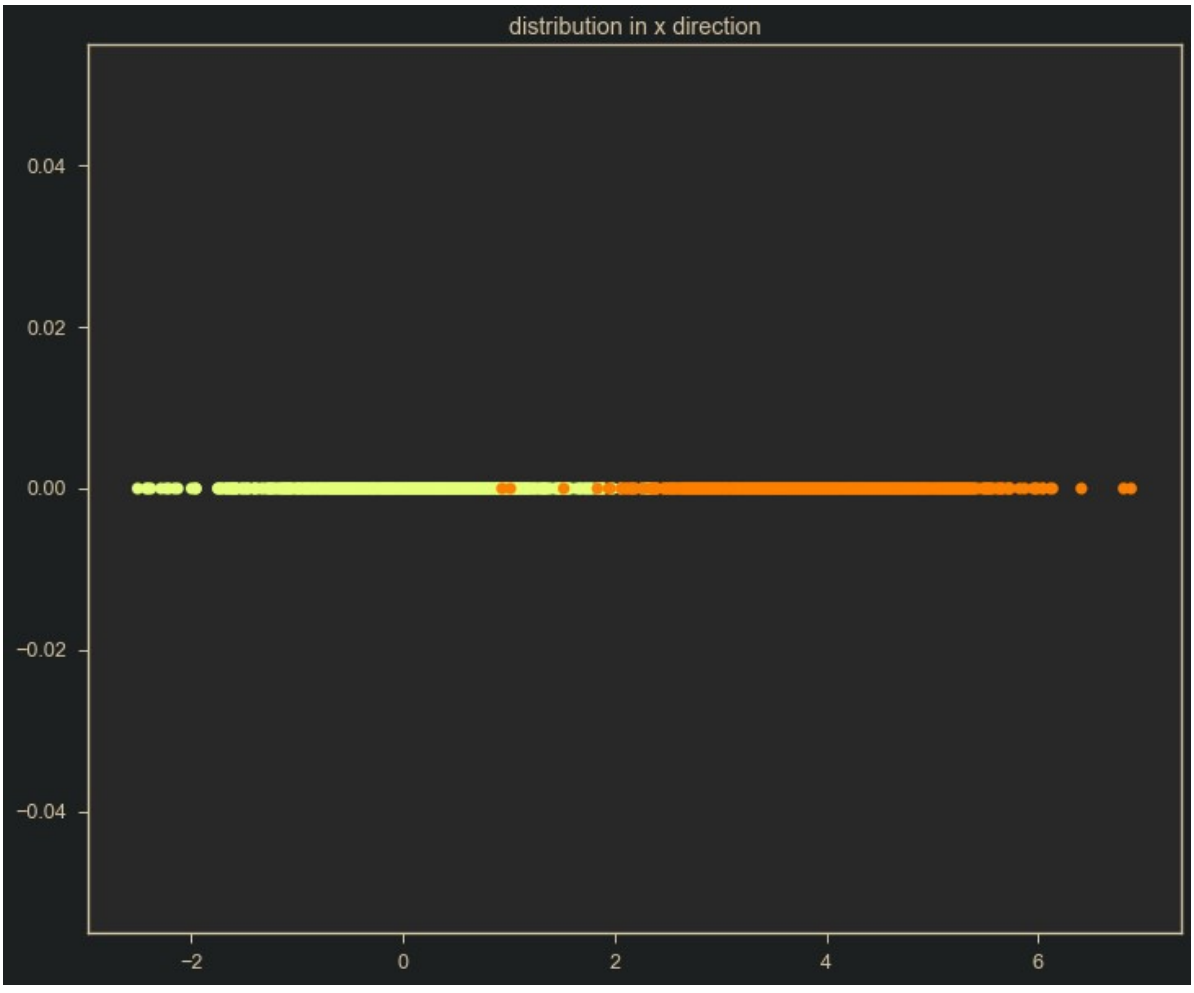
Here, m is the number of classes, μ is the overall sample mean, and n_k is the number of samples in the k-th class.

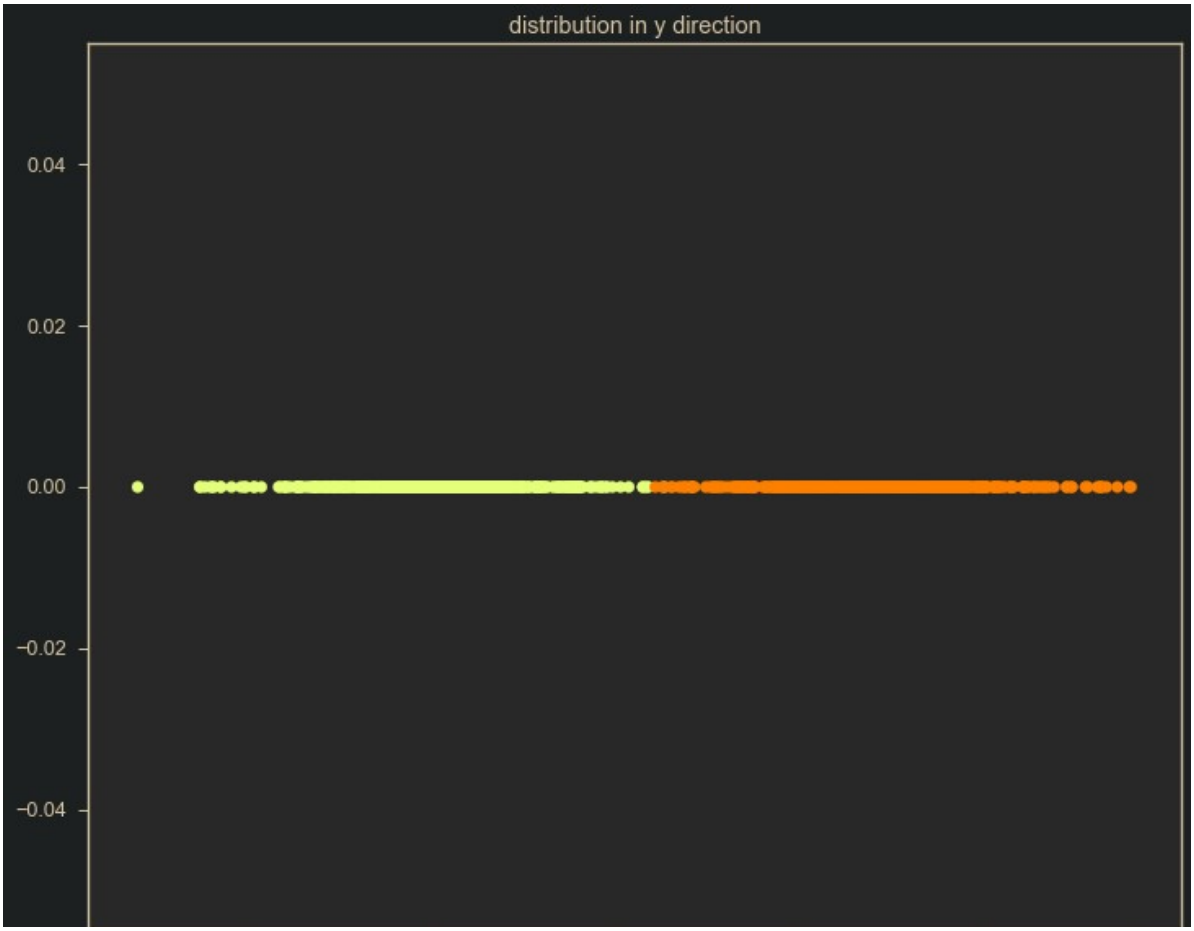
The solution is given by the following generalized eigenvalue problem:$S_B w = \lambda S_w w$

In [13]:

```
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
# plots
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],data[:,1],c=label,cmap='Wistia');
plt.title('Data visualization');
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label,cmap='Wistia
plt.title('distribution in x direction');
plt.figure(figsize=(12,10))
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label,cmap='Wistia
plt.title('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

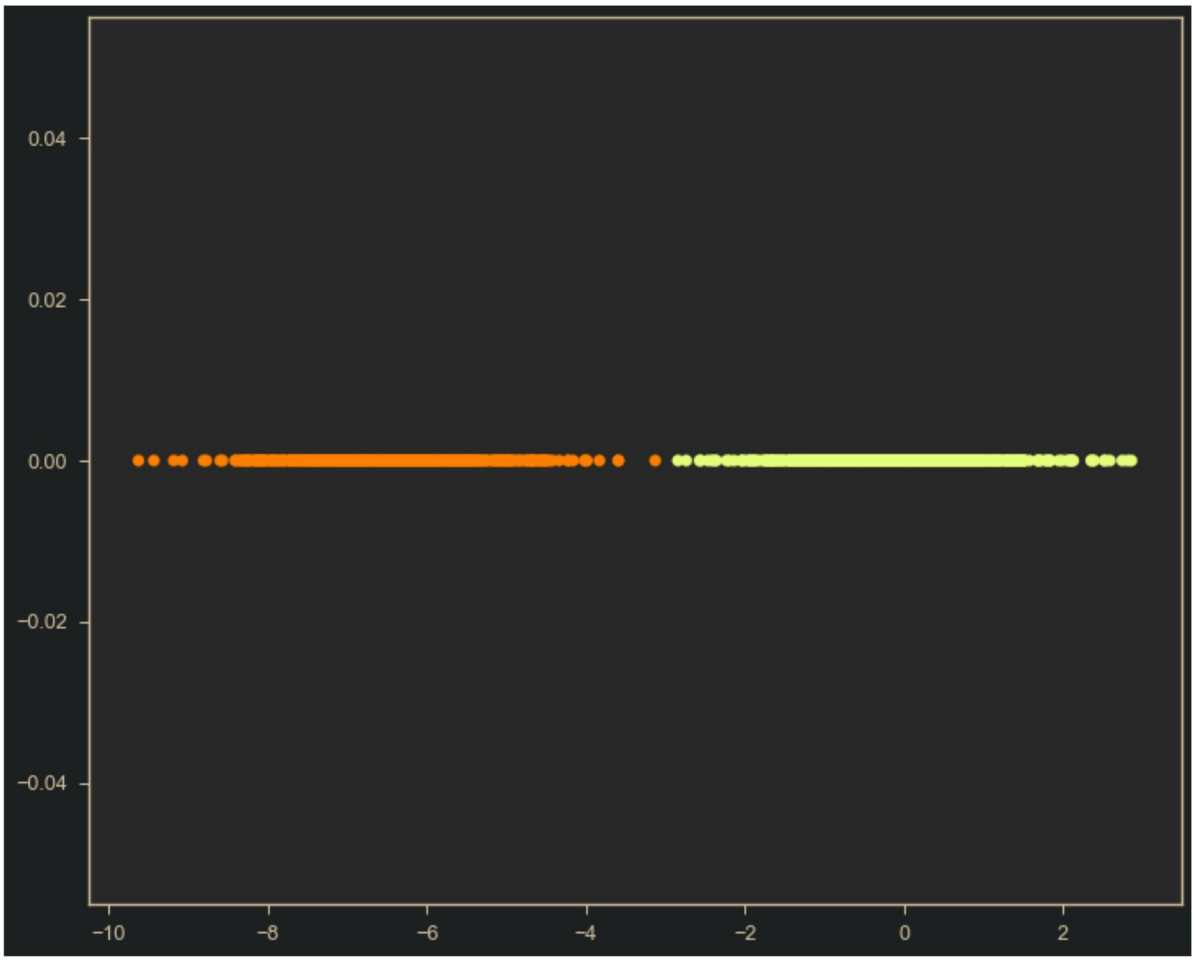

distribution in y direction
0.04
0.02
0.00
−0.02
−0.04

In [14]:

```
# perform 2-class and m-class LDA
def LDA(data,label):
    # some requiered variables
    id={};
    data_l={};
    mean_l={};
    cov_l={};
    S_w=np.zeros((data.shape[1],data.shape[1]));
    # finding Sb and Sw
    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,keepdims=True);
        cov_l[i]= ((data_l[i]-mean_l[i]).T)@(data_l[i]-
mean_l[i]);## Write your code here
        S_w+= cov_l[i];

    S_w=S_w/len(data_l);

    if len(cls)==2:# fro 2 classes
        S_b= ((mean_l[0]-mean_l[1]).T)@(mean_l[0]-mean_l[1]);##
Write your code here
        val,w1= np.linalg.eig(np.linalg.pinv(S_w)@S_b);## Write
your code here
        idx = np.flip(np.argsort(np.abs(val)));
        w=w1[:,idx];
    else:# for more than 2 clases
        S_t=np.cov(data,rowvar=False);
        S_b= S_t-S_w;## Write your code here
        _,val,w1= np.linalg.svd(np.linalg.pinv(S_w)@S_b);##
Write your code here
        idx = np.argsort(np.abs(val));
        w=w1[:,idx];

    return w
```

In [15]:

```
# after LDA projection
w=LDA(data,label);
plt.figure(figsize=(12,10))
plt.scatter(data @
w[:,0],np.zeros(data.shape[0]),c=label,cmap='Wistia');
```



In [16]:

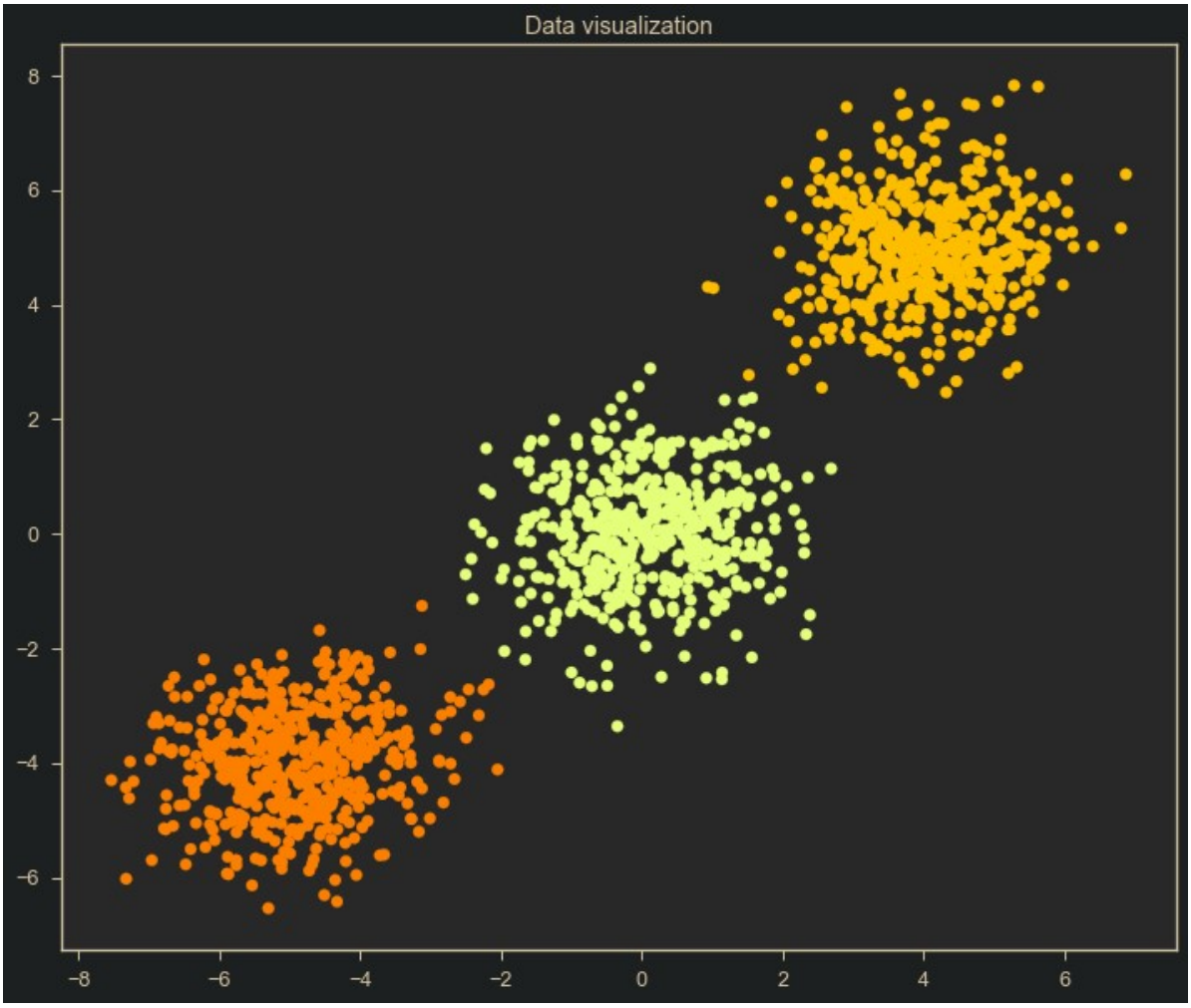
```
# Classification using :DA
# Use k-nearest neighbour classifier (Scikit Learn) after
dimensionality reduction
trans_data= np.reshape(data@w[:,0],(-1,1));
from sklearn.neighbors import KNeighborsClassifier
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, label)

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.s

print('KNN Testing accuracy =',knn.score(np.reshape(data@w[:,0],
(-1,1)),tst_label)*100)
```

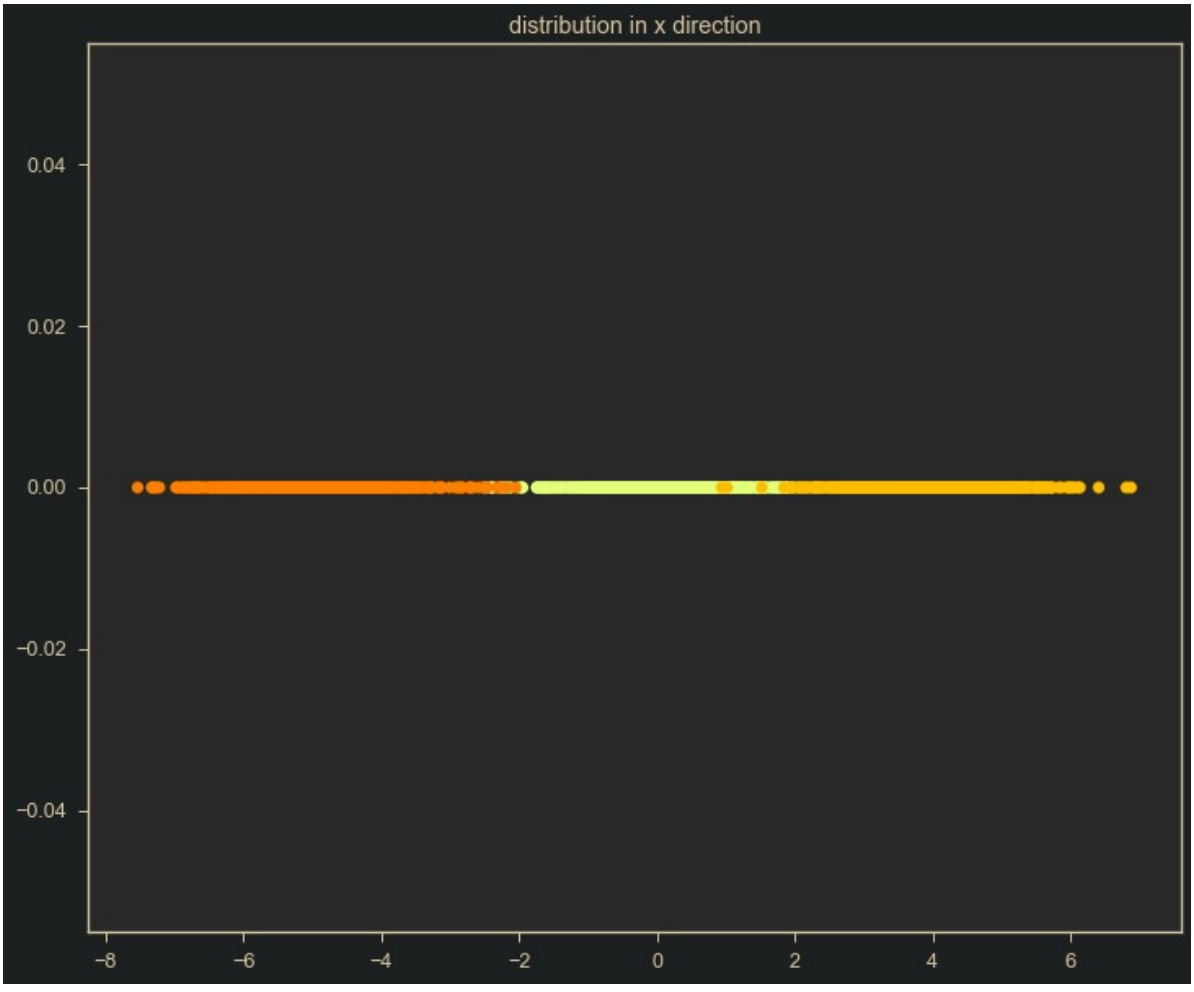
```
KNN Training accuracy = 100.0
KNN Testing accuracy = 100.0
```

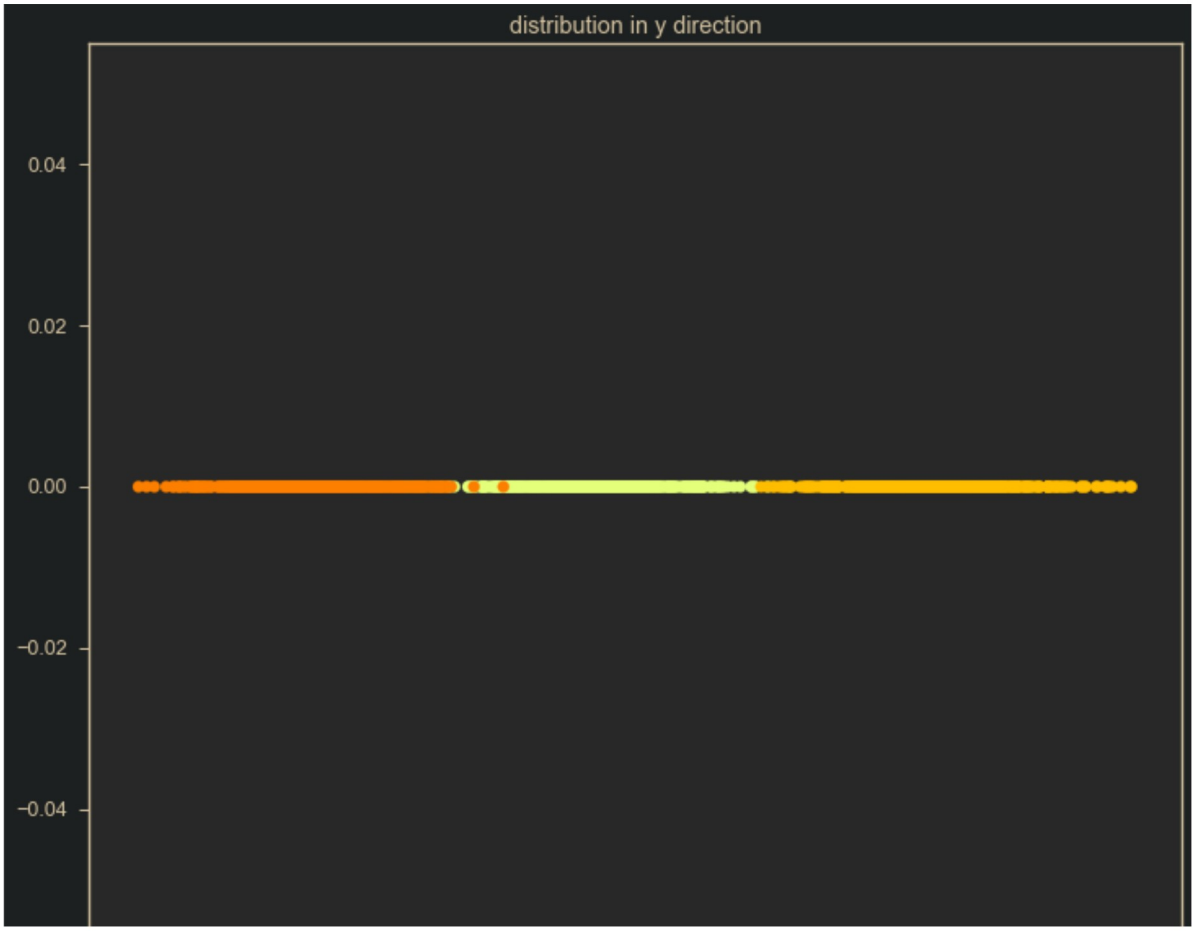
LDA Multiclass

In [17]:

```
# generating data
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
#plots
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],data[:,1],c=label,cmap='Wistia');
plt.title('Data visualization');
plt.figure(figsize=(12,10))
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label,cmap='Wistia
plt.title('distribution in x direction');
plt.figure(figsize=(12,10))
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label,cmap='Wistia
plt.title('distribution in y direction');
```


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

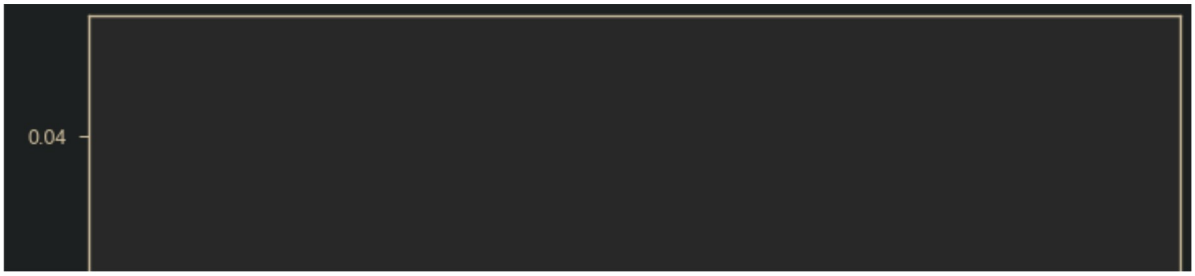

distribution in x direction
0.04
0.02
0.00
−0.02
−0.04
−8
−6
−4
−2
0
2
4
6



In [18]:

```
# after projection
w=LDA(data,label)
print(w.shape)
plt.figure(figsize=(12,10))
plt.scatter(data @
w[:,0],np.zeros(data.shape[0]),c=label,cmap='Wistia'); # by
performing 1D projection
```

```
(2, 2)
```



In [19]:

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

## Write your code here
trans_data= np.reshape(data@w[:,0],(-1,1));
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.s

print('KNN Testing accuracy =',knn.score(np.reshape(data@w[:,0],
(-1,1)),tst_label)*100)
```

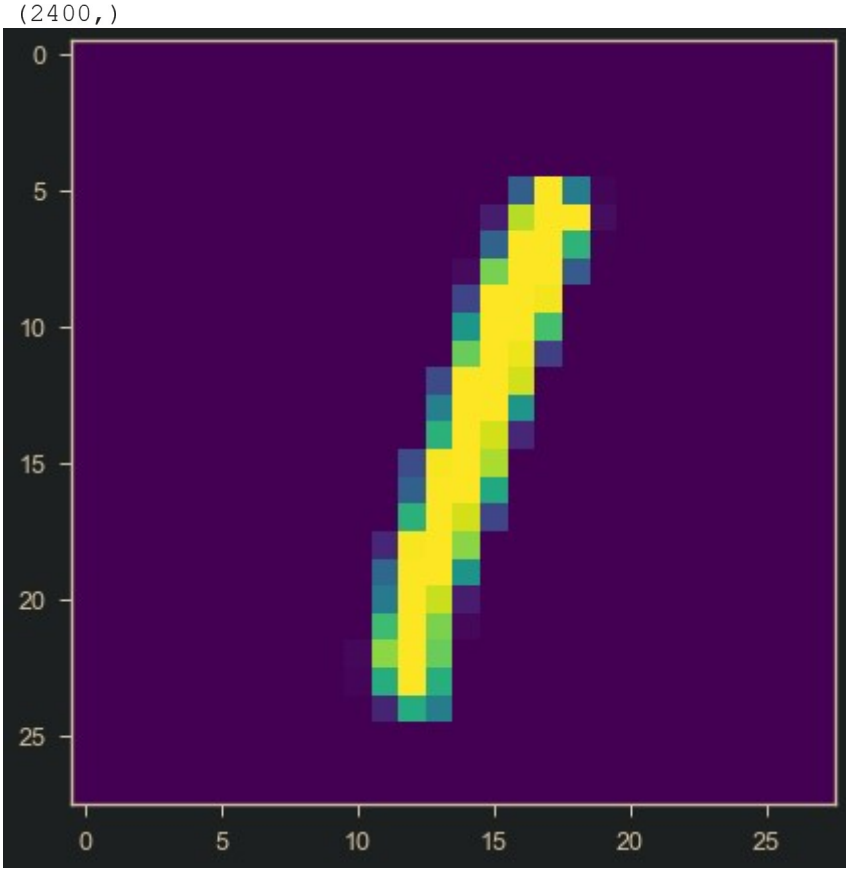
```
KNN Training accuracy = 99.93333333333332
KNN Testing accuracy = 100.0
```

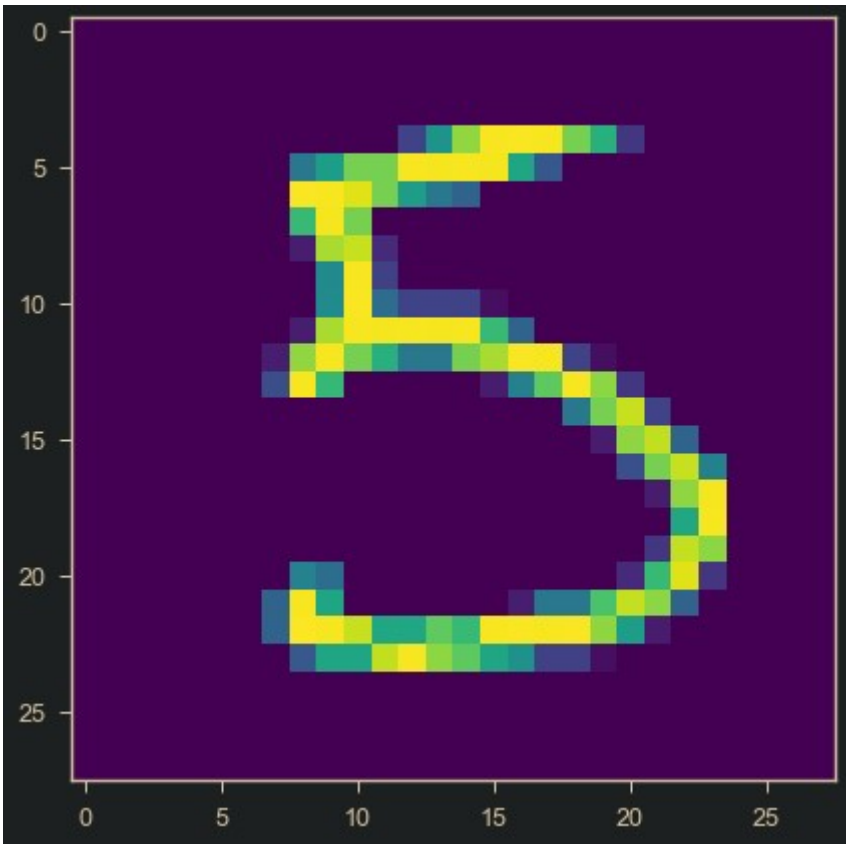
Perform LDA on MNIST and Classify using the data of any 3 classes

In [20]:

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

# for class 1
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[:800]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[:800]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[:800]
Im_9=Images[id9,:,:]
lab_9=labels[id9]
# 1 image from each class
plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])

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')

data /= 255
```

```
(2400, 784)
(2400,)
```





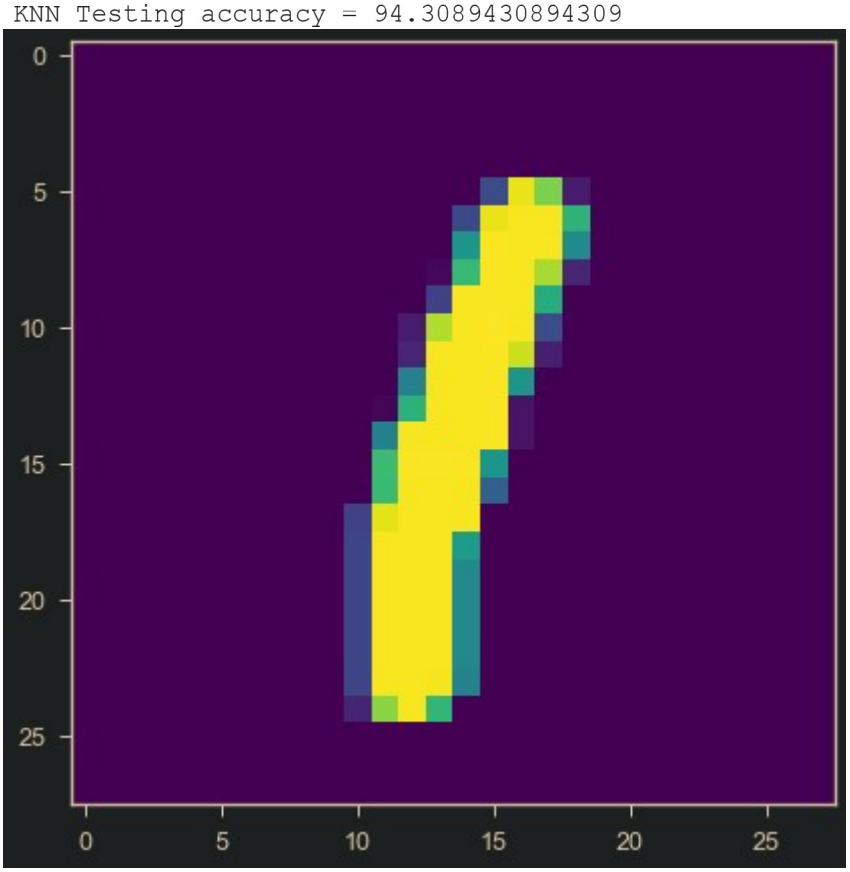

0
5
10
15

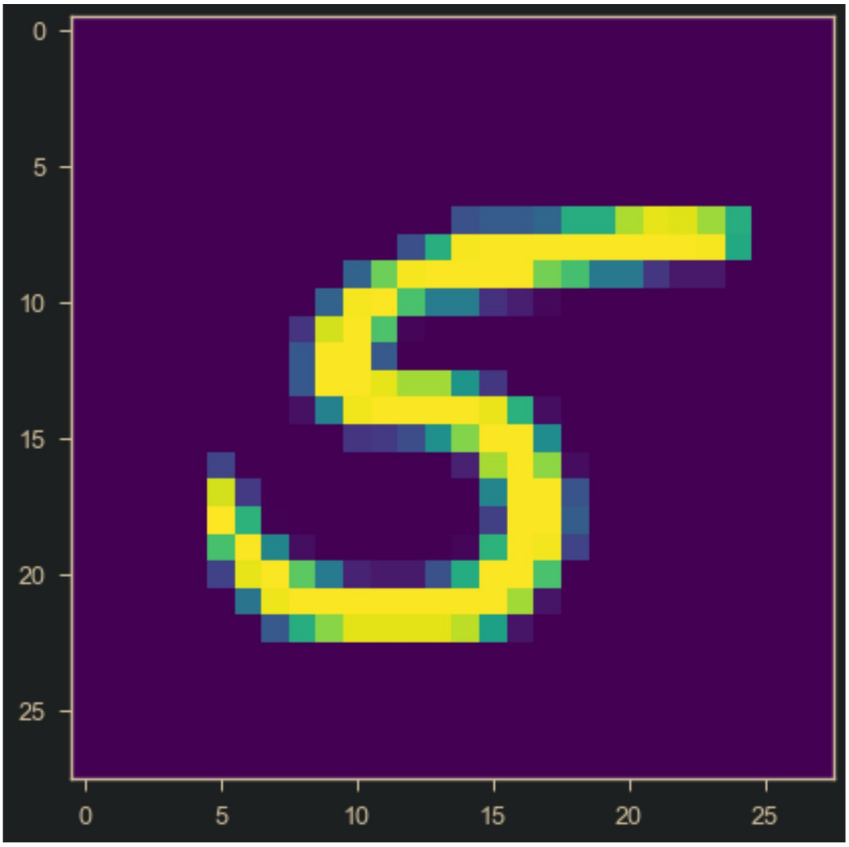
In [21]:

```
# applying knn after reducing dimension
print('Initial data dimension=',data.shape[1])
w=LDA(data,G_lab)
dim=200;
trans_data=np.reshape(data@w[:,:dim],(-1,dim));
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[800:1000]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[800:1000]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[800:1000]
Im_9=Images[id9,:,:]
lab_9=labels[id9]
# show 1 image from each class
plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])

data_tst=np.concatenate((Im_1,Im_5,Im_9))
```

```
# final testing
print('KNN Testing accuracy
=',knn.score(np.reshape(data_tst@w[:,:dim],
(-1,dim)),tst_lab)*100)
```

```
Initial data dimension= 784
Retained dimesion after LDA= 200
KNN Training accuracy = 98.5
KNN Testing accuracy = 94.3089430894309
```





In [ ]: