LAB 11 : Dimensionality Reduction

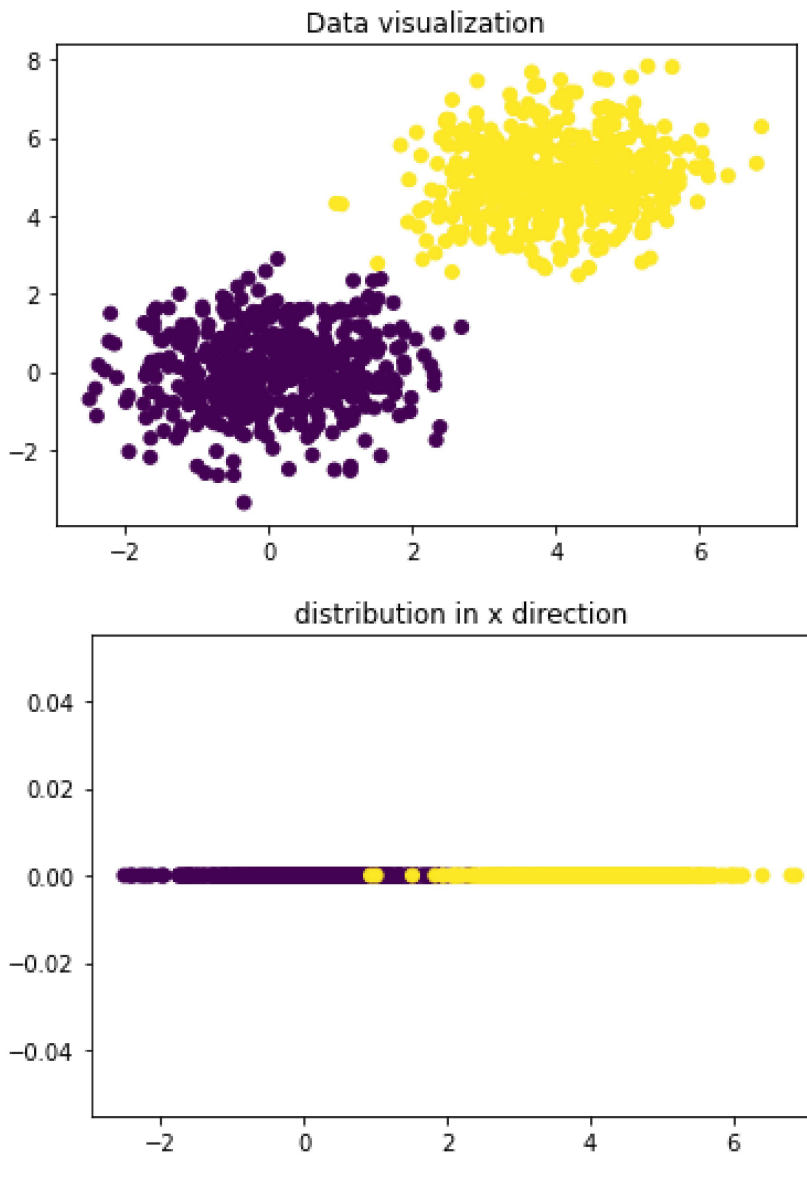
1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)

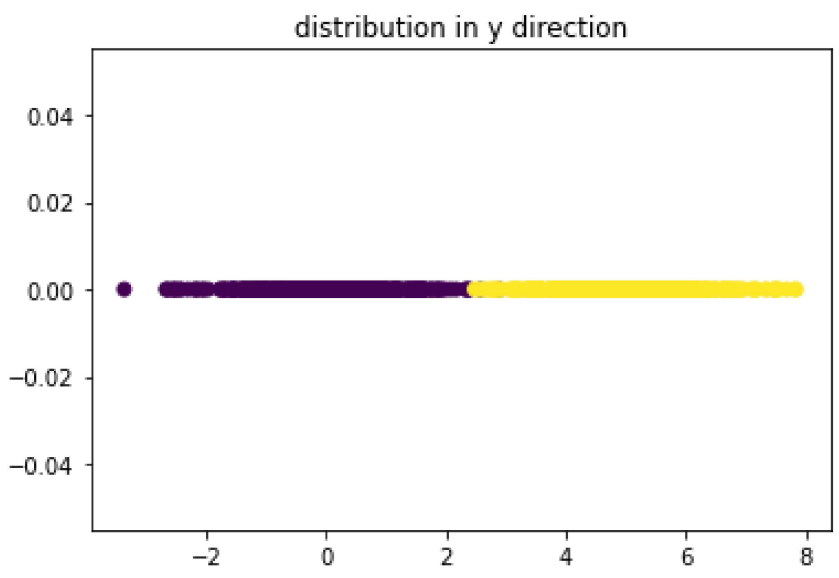
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Task 1: PCA

1.1 Data Generation

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.show()

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

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

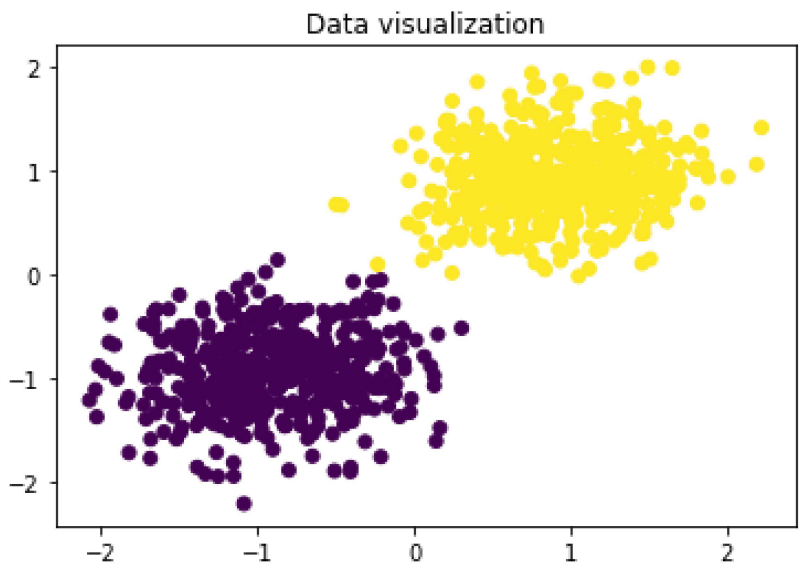




1.2 Data Normalization

$$X \rightarrow \frac{X-\mu}{\sigma}$$

In [ ]:
```
# Data normalization

# Perform data normalization here using mean substraction and std division
## Write your code here
avg = np.mean(data, axis=0)
std = np.std(data, axis=0)
data = (data-avg)/std

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



1.3 Obtain PCA directions and variances

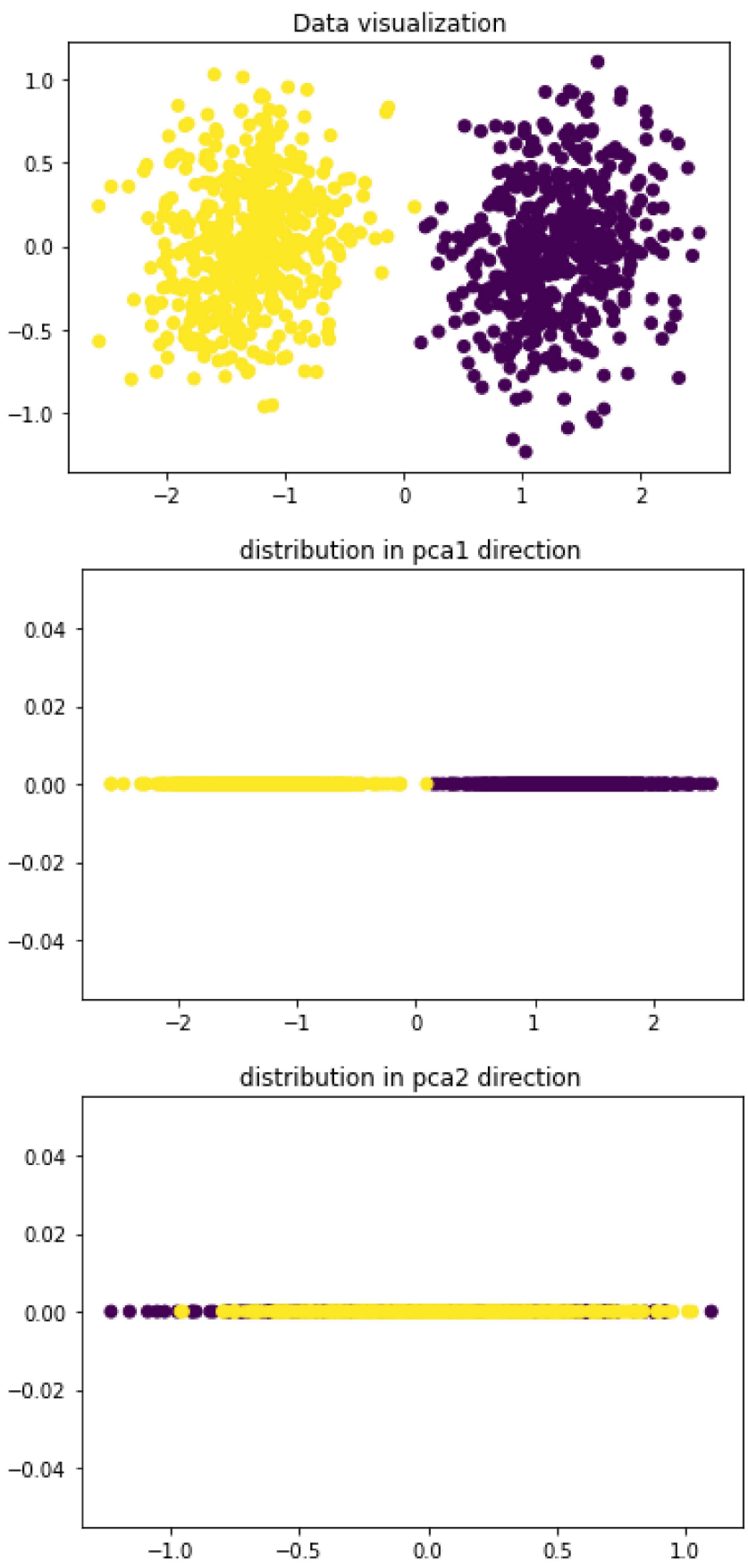
Steps:

1. After normalizing data, compute covariance matrix.
2. Use svd() on covariance matrix (m x n) to get U, S, VT. (U and VT are orthonormal sets of orders m x m and n x n respectively). S is a diagonal matrix with variances in each direction.
3. Transform the data using U, and check variances along the 2 PCA directions chosen.

In [ ]:
```
# PCA

# covariance matrix
cov=data.T @ data

# using singular value decomposition
u,s,vh=np.linalg.svd(cov)

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.show()

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

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

```
variance along pca1 direction= 1.8477663843459724
variance along pca2 direction= 0.152233615654027
```



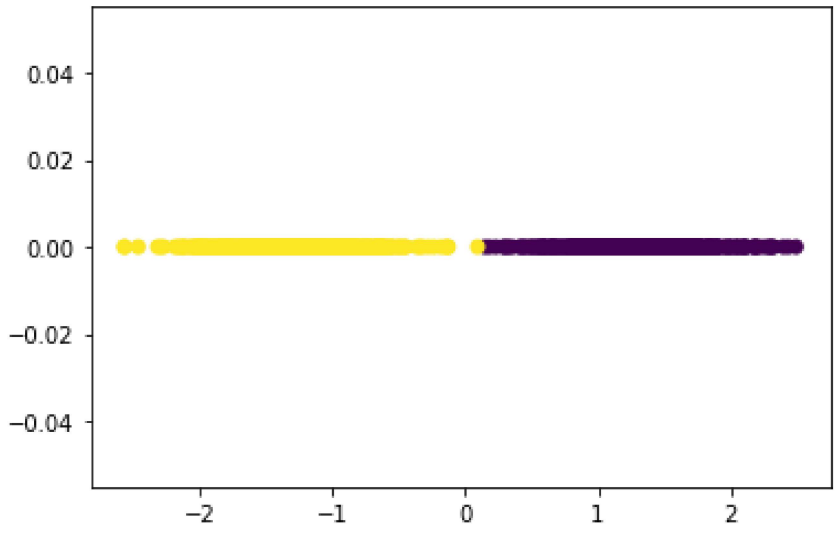
In [ ]:

```
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)/np.sum(s))>0.9))[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 0x23a8ba6ab20>
```



1.4 Conduct Classification using PCA with K-NN

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)*100)
```

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

1.5 Apply PCA on MNIST Digit dataset

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

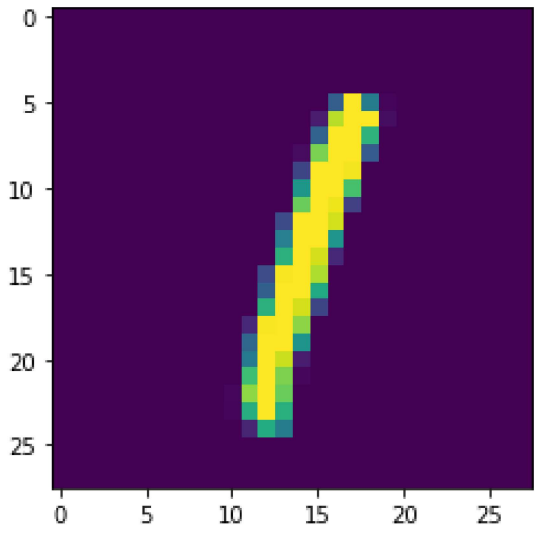
```
Requirement already satisfied: idx2numpy in c:\users\bsidd\appdata\local\programs\python\python39\lib\site-packages (1.2.3)
Requirement already satisfied: numpy in c:\users\bsidd\appdata\local\programs\python\python39\lib\site-packages (from idx2nu
mpy) (1.21.2)
Requirement already satisfied: six in c:\users\bsidd\appdata\local\programs\python\python39\lib\site-packages (from idx2nump
y) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

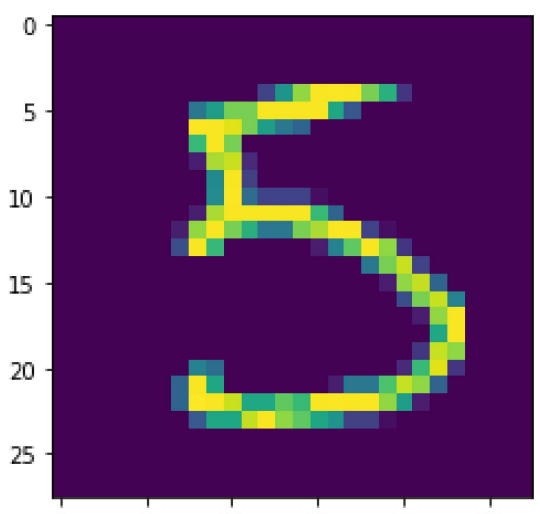
```
WARNING: You are using pip version 22.0.4; however, version 22.3 is available.
You should consider upgrading via the 'c:\Users\bsidd\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upg
rade pip' command.
```

In [ ]:
```
# MNIST data

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')

data /= 255
```

```
(50, 28, 28)
(100, 784)
(100,)
```





In [ ]:

```
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]
```

```
# 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_lab)*100)
```

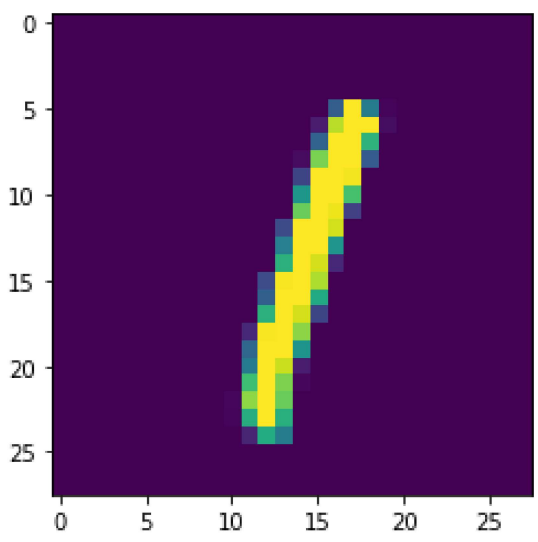
```
Initial data dimension= 784
Retained dimesion after PCA= 34
KNN Training accuracy = 96.0
(50, 28, 28)
KNN Testing accuracy = 98.0
```

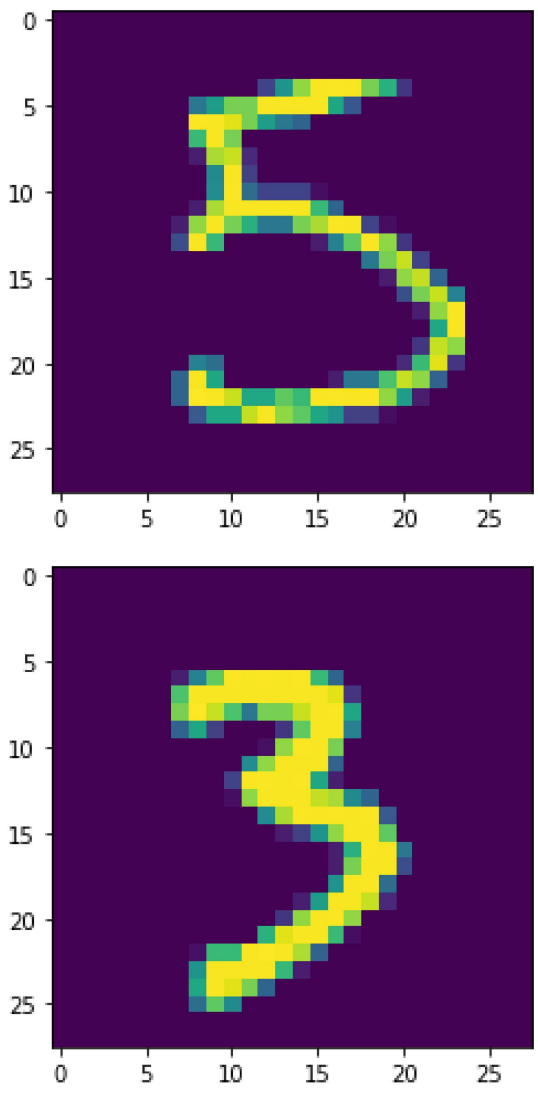
1.6 Perform PCA on MNIST and Classify taking the data with any 3 Classes

In [ ]:

```
## Write your code here
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=[1,5,3]

# 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 3
id_3=np.where(labels==cl[2])
id3=id_3[0]
id3=id3[:50]
Im_3=Images[id3,:,:]
lab_3=labels[id3]

plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_3[1,:,:])

print(Im_5.shape)

data=np.concatenate((Im_1,Im_5,Im_3))
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_3))
print(G_lab.shape)

data = data.astype('float32')

data /= 255
```

```
(50, 28, 28)
(150, 784)
(150,)
```





In [ ]:
```
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=7
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 5
id_3=np.where(labels==cl[2])
id3=id_3[0]
id3=id3[100:150]
Im_3=Images[id3,:,:]
lab_3=labels[id3]

# plt.imshow(Im_1[1,:,:])
# plt.figure()
# plt.imshow(Im_5[1,:,:])

print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5,Im_3))
```

```
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_3))


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

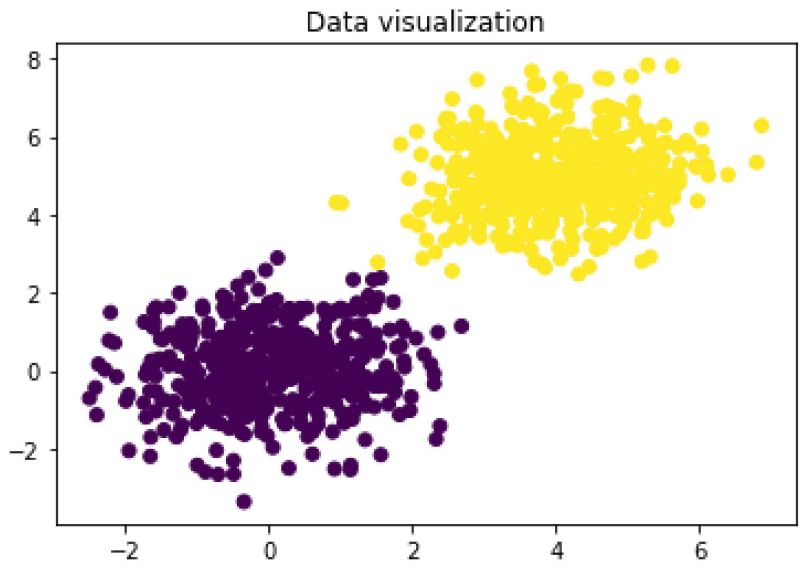
```
Initial data dimension= 784
Retained dimesion after PCA= 50
KNN Training accuracy = 90.66666666666666
(50, 28, 28)
KNN Testing accuracy = 86.0
```

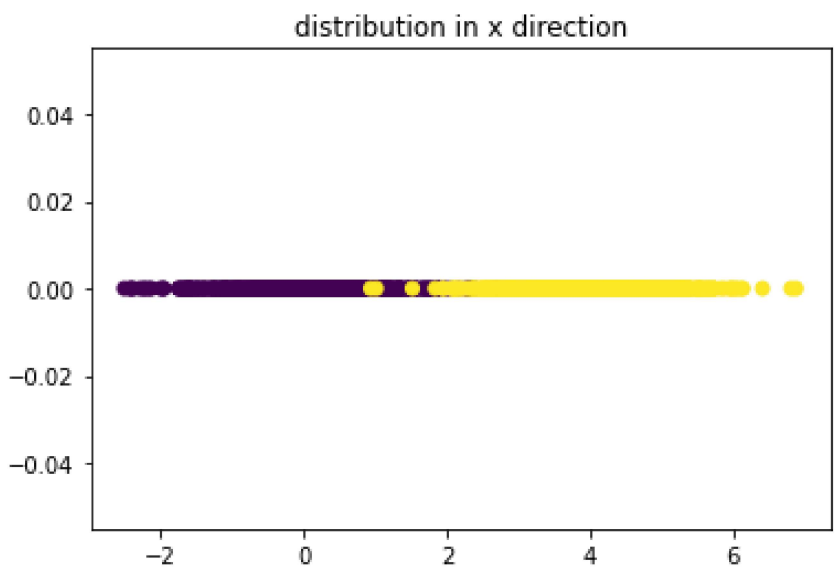
Task 2: LDA

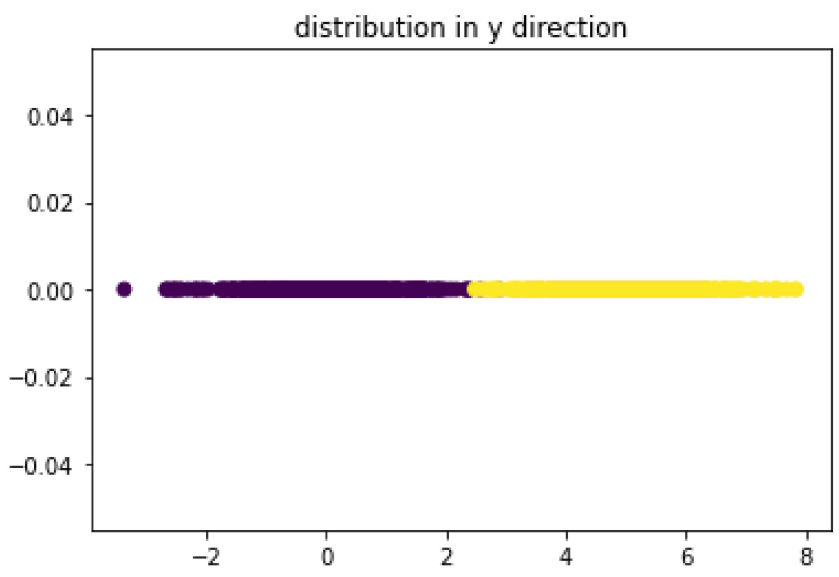
2.1 Data Generation

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.show()

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

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





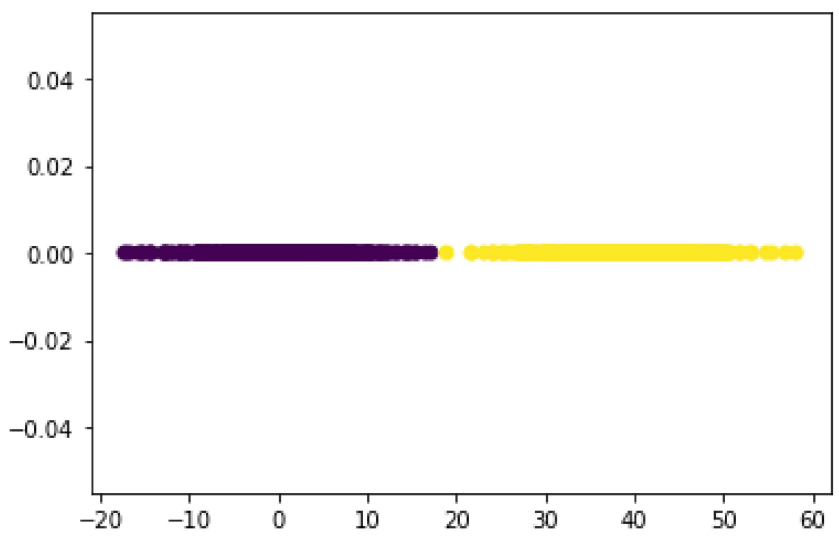


2.2 Perform 2-class and m-class LDA

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].shape[0]-1) ## Write your code here
    S_w=S_w+cov_l[i]


  S_w=S_w/len(data_l)

  if len(data_l)==2:
    S_b= (mean_l[1]-mean_l[0]).T @ (mean_l[1]-mean_l[0]) ## Write your code here
    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)
    S_b= S_t-S_w ## Write your code here
    u,_,_= np.linalg.svd(np.linalg.pinv(S_w) @ S_b) ## Write your code here
    w=u[:,:len(data_l)-1]

  return w
```

In [ ]:
```
# after LDA projection

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

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x23a8bcc1fa0>
```



2.3 Perform classification using LDA with K-NN

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

## Write your code here
from sklearn.neighbors import KNeighborsClassifier
```

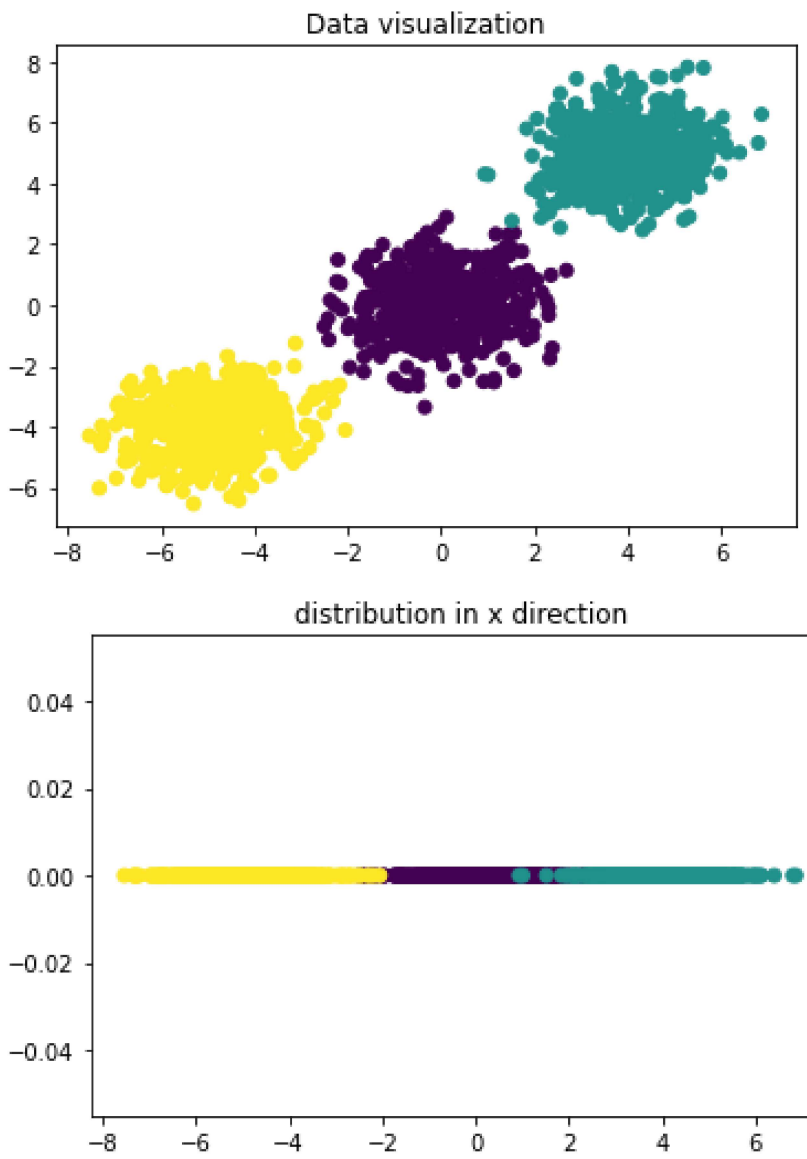
```
LDA_data = data @ w[:,np.newaxis]
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(LDA_data, label)

print('KNN Training accuracy =', knn.score(LDA_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(data@w[:,np.newaxis], tst_label)*100)
```

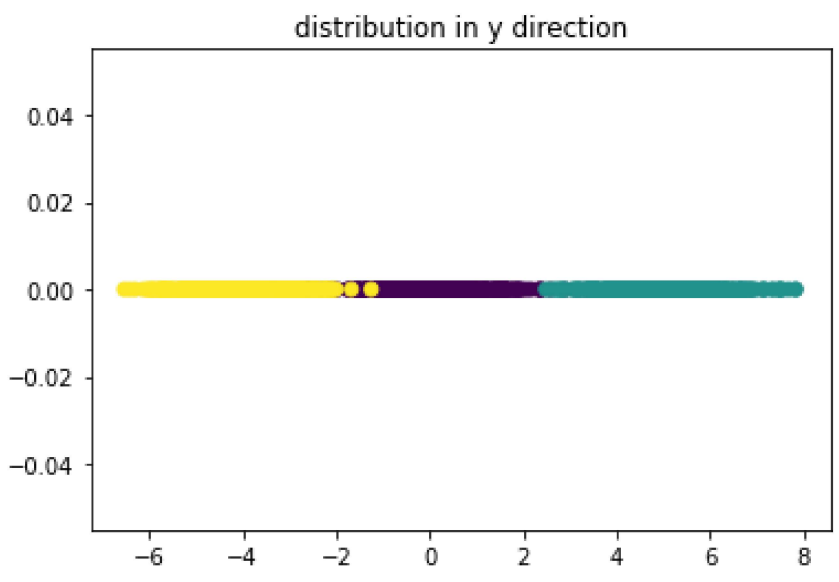
```
KNN Training accuracy = 100.0
KNN Testing accuracy = 100.0
```

2.4 LDA Multiclass

In [ ]:

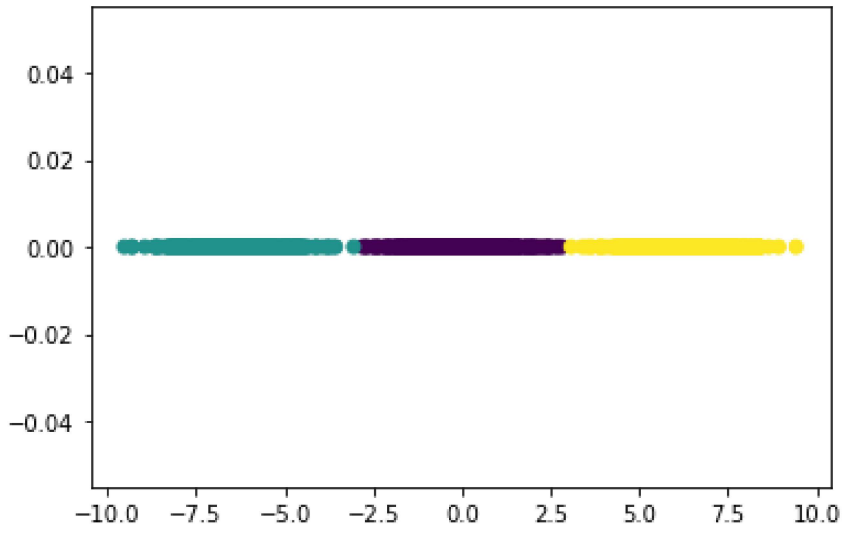
```
mean1=np.array([0,0])
mean2=np.array([4,5])
mean3=np.array([-5,-4])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data3=np.random.multivariate_normal(mean3,var,500)
data=np.concatenate((data1,data2,data3))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0]),np.ones(data3.shape[0])+1))

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

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

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





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 projection
```

Out[ ]:
```
(2, 2)
<matplotlib.collections.PathCollection at 0x23a8bd73220>
```



2.5 Testing with K-NN

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

## Write your code here
from sklearn.neighbors import KNeighborsClassifier

LDA_data = data @ w
k = 5
knn = KNeighborsClassifier(n_neighbors = k)
knn.fit(LDA_data, label)

print('KNN Training accuracy =',knn.score(LDA_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)
data3 = np.random.multivariate_normal(mean3,var,50)
data_tst = np.concatenate((data1,data2,data3))
tst_label = np.concatenate((np.zeros(data1.shape[0]), np.ones(data2.shape[0]), np.ones(data2.shape[0])+1))

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

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

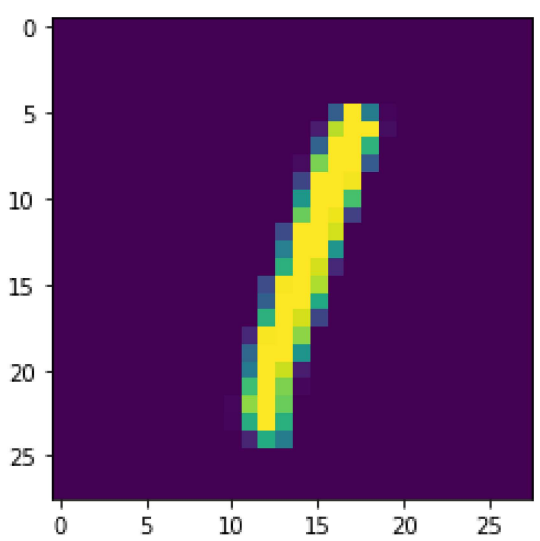
2.6 Perform LDA on MNIST and Classify using the data of any 3 classes

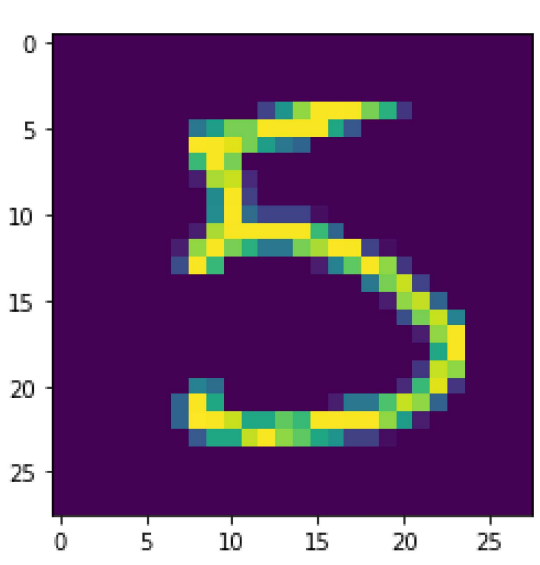
In [ ]:
```
## Write your code here
file1='t10k-images-idx3-ubyte' ## Change the path accordingly
file2='t10k-labels-idx1-ubyte' ## Change the path accordingly

import idx2numpy

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

cl=[1,5,3]
```

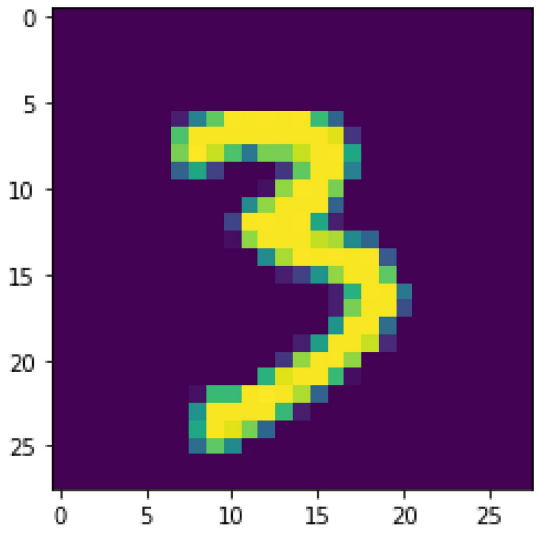
```
# for class 1
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[:50]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# 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 3
id_3=np.where(labels==cl[2])
id3=id_3[0]
id3=id3[:50]
Im_3=Images[id3,:,:]
lab_3=labels[id3]

plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_3[1,:,:])

print(Im_5.shape)
```

```
data=np.concatenate((Im_1,Im_5,Im_3))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
label=np.concatenate((lab_1,lab_5,lab_3))
print(G_lab.shape)

data = data.astype('float32')

data /= 255
```

```
(50, 28, 28)
(150, 784)
(150,)
```







In [ ]:
```
w = LDA(data,label)

from sklearn.neighbors import KNeighborsClassifier

LDA_data = data @ w
k = 5
knn = KNeighborsClassifier(n_neighbors = k)
knn.fit(LDA_data, label)

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

```
KNN Training accuracy = 100.0
```

In [ ]:
```
## 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 5
id_3=np.where(labels==cl[2])
id3=id_3[0]
id3=id3[100:150]
Im_3=Images[id3,:,:]
lab_3=labels[id3]

# plt.imshow(Im_1[1,:,:])
# plt.figure()
# plt.imshow(Im_5[1,:,:])

print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5,Im_3))
data_tst=np.reshape(data_tst,(data_tst.shape[0],data_tst.shape[1]*data_tst.shape[2]))


tst_label=np.concatenate((lab_1,lab_5,lab_3))


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

```
(50, 28, 28)
KNN Testing accuracy = 50.66666666666667
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.2, random_state=0)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

print('KNN Training accuracy =',knn.score(X_train, y_train)*100)
y_pred = knn.predict(X_test)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)
print('Accuracy' + str(accuracy_score(y_test, y_pred)*100))
```

```
KNN Training accuracy = 89.16666666666667
[[11  0  0]
 [ 1  0  5]
 [ 2  7  4]]
Accuracy50.0
```

In [ ]: