Name: S U Swakath

Roll: 180020036

google colab link : https://colab.research.google.com/drive/1LkQTE-B7WmqvfKMHu8Mmp3CGHUop3O03?usp=sharing

Classification:

1. K nearest neighbor (K-NN)

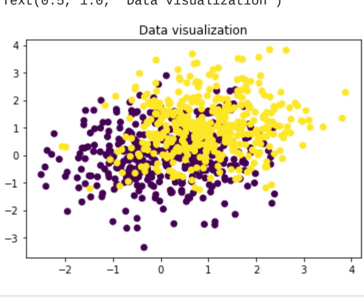
Dimensionality Reduction techniques:

1. Principal component analysis (PCA)
2. Linear discriminative analysis (LDA)

K-NN

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt

mean1=np.array([0,0])
mean2=np.array([1,1])
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_train=np.concatenate((data1[:-100,],data2[:-100]))
label=np.concatenate((np.zeros(data1.shape[0]-100),np.ones(data2.shape[0]-100)))

label=label.astype(np.int32)

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

Out[1]: `Text(0.5, 1.0, 'Data visualization')`



In [2]:
```python
def euclidean_distance(row1, row2):
  return np.linalg.norm(row1-row2)
```

In [3]:
```python
def get_neighbors(train,label_train, test_row, nk):
  # insert your code here
  ng = []
  for i in range(train.shape[0]) :
    ng.append([euclidean_distance(train[i], test_row), label_train[i]])
  ng = np.array(ng)
  ng = ng[ng[:,0].argsort()]
  ng = ng[:nk, 1]
  return ng
```

In [4]:
```python
def predict_classification(ng):
  ng = list(ng)
  if ng.count(1) >= ng.count(0):
    return 1;
  return 0;
```

In [5]:
```python
# test data generation
data_test=np.concatenate((data1[-100:],data2[-100:]))
label_test=np.concatenate((np.zeros(100),np.ones(100)))
label_test = label_test.astype(np.int32)
```

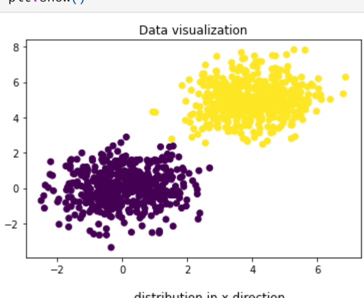
In [6]:
```python
K=2

pred_label=np.zeros(data_test.shape[0])
for i in range(data_test.shape[0]):
  neig=get_neighbors(data_train,label, data_test[i,:], K)
  pred_label[i]=predict_classification(neig)

pred_label = pred_label.astype(np.int32)

accuracy=(len(np.where(pred_label==label_test)[0])/len(label_test))*100
print('Testing Accuracy=',accuracy,'%')
```

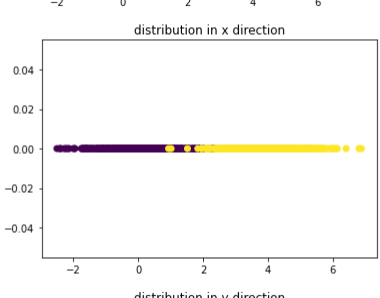
```
Testing Accuracy= 65.5 %
```

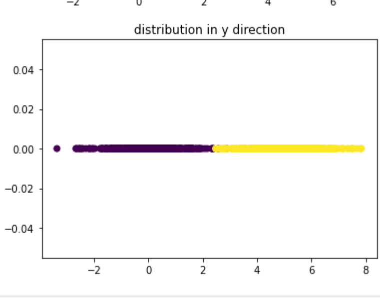
Principal component analysis (PCA)

1. Generate 2D data of 1000 points
2.

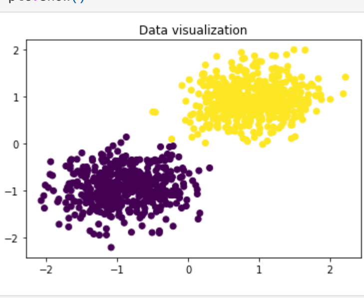
In [34]:
```python
import numpy as np
import matplotlib.pyplot as plt

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()
```





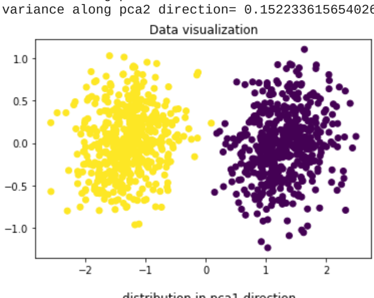


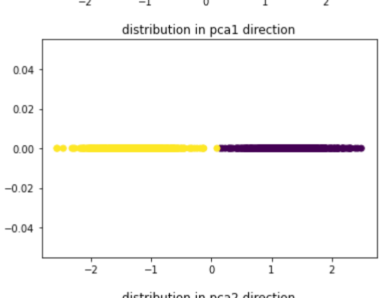
In [35]:
```python
#Data normalization
# print(data[:5])
# perform data normalization here using mean substraction and std division
data_mean = np.mean(data, axis=0)
data_var = np.std(data, axis=0)
# print(data_mean.shape, data_var.shape)
data_norm = ( data - data_mean ) / data_var

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

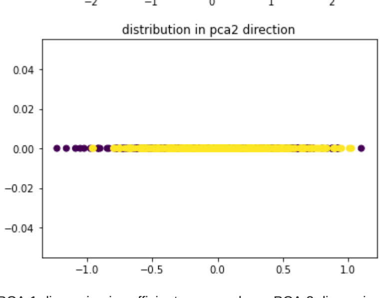


In [37]:
```python
# PCA

# coverance matrix
cov= data_norm.T @ data_norm


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

trans_data = data_norm @ u
print(trans_data.shape)

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()
```

```
(1000, 2)
variance along pca1 direction= 1.8477663843459717
variance along pca2 direction= 0.15223361565402696
```



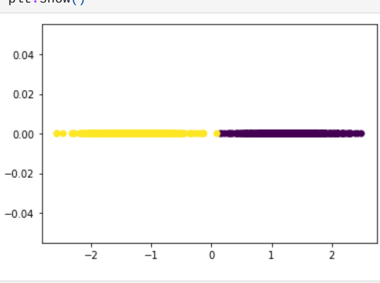




PCA 1 dimension is sufficient, we can droup PCA 2 dimension

In [26]:
```python
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)
    if retain_dim == None:
      self.retain_dim = self.ret_dim(self.data)
    else :
      self.retain_dim = retain_dim
    # compute pca transform value
  def pca_comp(self,data):
    data=self.pre_process(data)
    cov = data.T @ data
    u,_,_=np.linalg.svd(cov) # singular value decomposition
    u_req = u[:, 0:self.retain_dim]
    # print("ureq_shape : ", u_req.shape)
    # print("ret_dim : ", self.retain_dim)
    trans_data= data @ u_req
    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)
    cur_sum = 0
    total_sum = np.sum(s)
    # print(s.shape)
    ind = -1
    for i in range(len(s)):
      cur_sum += s[i];
      if cur_sum / total_sum > 0.9:
        ind = i
        break
    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 [38]:
```python
# 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)
plt.show()
```



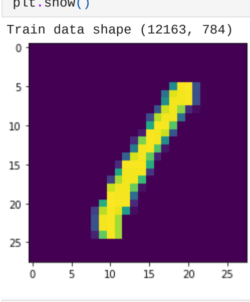
In [39]:
```python
#classification using pca
#use k-nearest neighbour classifier after dimensionality reduction

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
```

In [ ]:
```python
from google.colab import drive
drive.mount('/gdrive')
```

In [40]:
```python
# MNIST data
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

(X_train, y_train), (X_test, y_test) = mnist.load_data()

train_data = []
test_data = []
data_plot = []
train_label =[]
test_label =[]
for i in range(len(X_train)):
  if y_train[i] == 1 or y_train[i] == 5:
    data_plot.append(X_train[i])
    train_data.append(X_train[i].flatten()/255)
    train_label.append(y_train[i])
train_data = np.array(train_data)
train_label = np.array(train_label)
data_plot = np.array(data_plot)

print("Train data shape",train_data.shape)

for i in range(len(X_test)):
  if y_test[i] == 1 or y_test[i] == 5:
    test_data.append(X_test[i].flatten()/255)
    test_label.append(y_test[i])
test_data = np.array(test_data)
test_label = np.array(test_label)

plt.imshow(data_plot[1])
plt.show()
```

```
Train data shape (12163, 784)
```



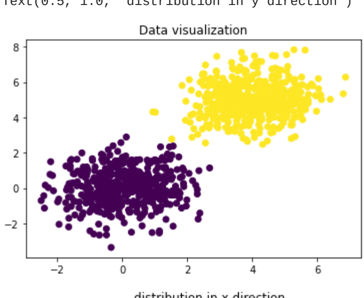
In [41]:
```python
print('Initial data dimension=',train_data.shape[1])
PCA=pca(data=train_data)

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

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

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

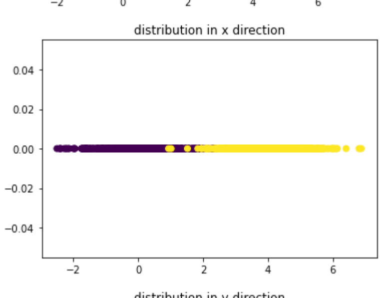
```
Initial data dimension= 784
Retained dimesion after PCA= 173
KNN Training accuracy = 99.78623694812136
KNN Testing accuracy = 99.80266403552048
```

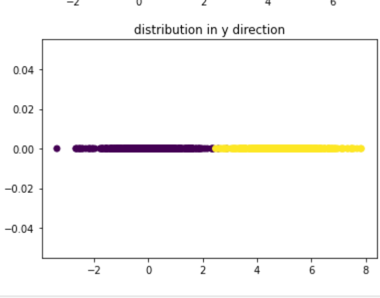
Linear Discriminate Analysis (LDA)

In [85]:
```python
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[85]: `Text(0.5, 1.0, 'distribution in y direction')`

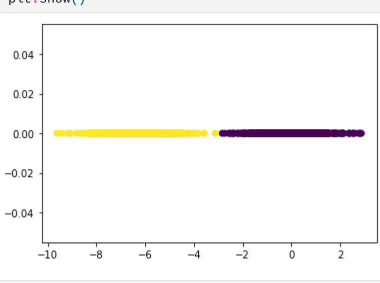






In [87]:
```python
# perform 2-class and m-class LDA
def LDA(data,label):
  id={}
  data_l={}
  mean_l={}
  cov_l={}
  S_w=np.zeros((data.shape[1],data.shape[1]))
  cls=np.unique(label)

  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)
    temp = data_l[i]-mean_l[i];
    cov_l[i] = temp.T@temp
    S_w=S_w+cov_l[i]

  S_w = S_w/len(data_l)
  if len(data_l)==2:
    temp = mean_l[0]-mean_l[1]
    temp = np.array([temp])
    S_b = temp.T @ temp;
    temp = np.linalg.pinv(S_w)@S_b
    u,_,_=np.linalg.svd(temp)
    w = u[:,:len(data_l)-1]
  else:
    S_t = np.cov(data,rowvar=False)
    S_b = S_t-S_w
    temp = np.linalg.pinv(S_w)@S_b
    u,_,_ = np.linalg.svd(temp)
    w = u[:,:len(data_l)-1]
  return w
a = LDA(data,label)
```

In [88]:
```python
# after LDA projection
w=LDA(data,label)
plt.figure()
plt.scatter(data @ w,np.zeros(data.shape[0]),c=label)
plt.show()
```



In [75]:
```python
#classification using pca
#use k-nearest neighbour classifier after dimensionality reduction
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)
data_tst=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_tst@ w,tst_label)*100)
```

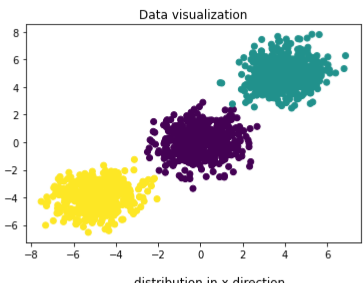
```
KNN Training accuracy = 100.0
KNN Testing accuracy = 100.0
```

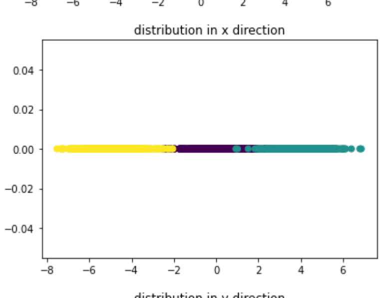
LDA multiclass

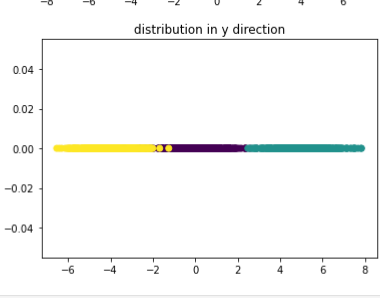
1. 3 class Sythetic data
2. Homework: Mnist 3 class and 10 class

In [79]:
```python
import numpy as np
import matplotlib.pyplot as plt

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



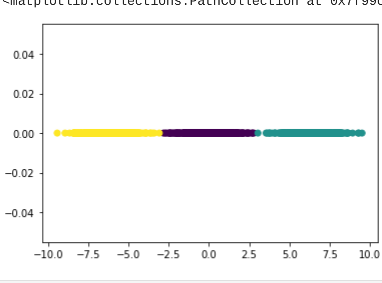




In [82]:
```python
# after projection
w=LDA(data,label)
print(w.shape)
plt.figure()
plt.scatter(data @ w[:,1],np.zeros(data.shape[0]),c=label) # by performing 1D projection
```

```
(2, 2)
```

Out[82]: `<matplotlib.collections.PathCollection at 0x7f99c1f6aa58>`



In [83]:
```python
# testing (using KNN)

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
```