ECE657A Assignment 1 – Group17

I. Data Cleaning and Preprocessing (for dataset A)

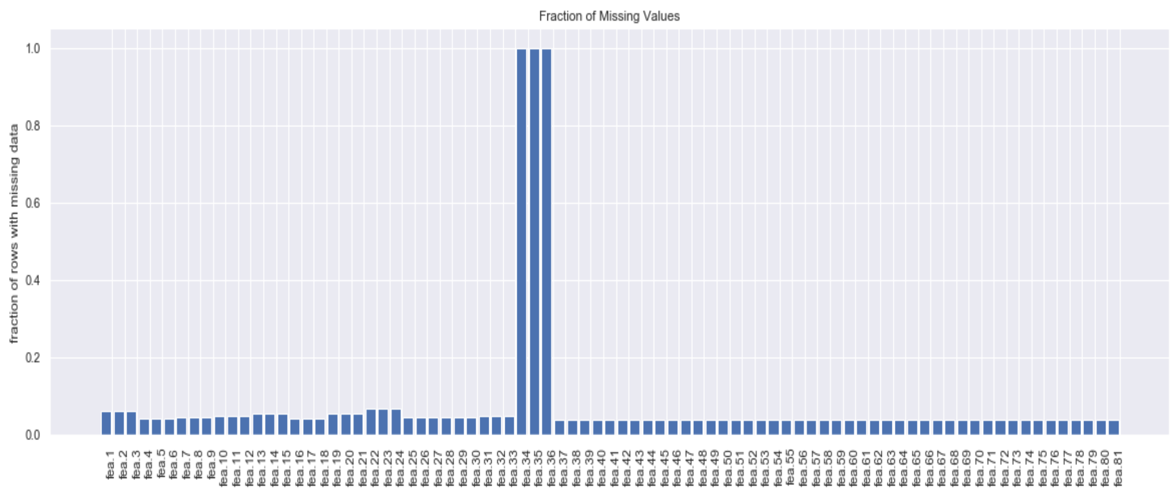
1) There are mainly four problems with the data available in data set DataA.

- To begin with there are three features (feature 34, 35, 36) within the data set which almost completely contain NaN/Null values.
- All of the remaining columns contain missing values.
- In addition, there are outliers present in the data and
- Lastly the data is not normalized.

All these problems need to be sorted and are handled gracefully in the proceeding parts. The histogram helped to visualize fractions of data missing in each features. The box plots helped identifying the presence of outliers and the info() function helped identifying the format of the data.

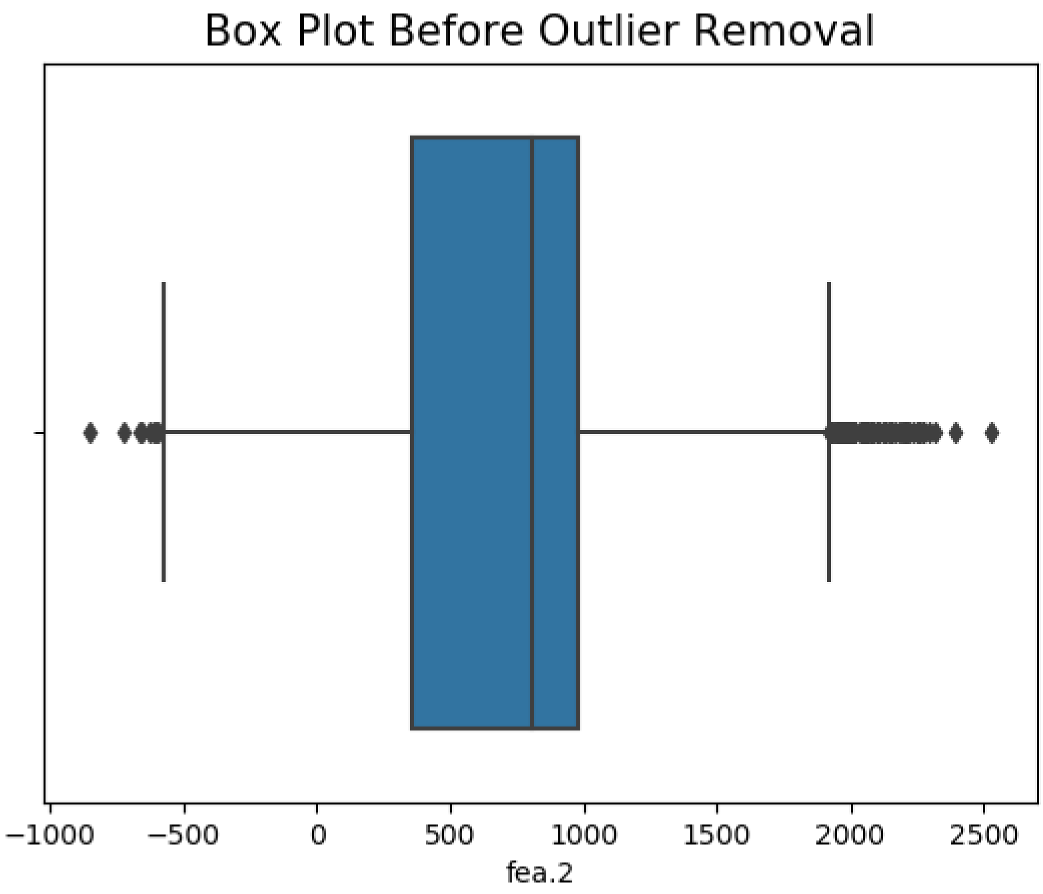
```
import pandas as pd
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df=pd.read_csv("C:/Users/HP/Downloads/DataA.csv",index_col=0)
print(df.info())
null_counts = df.isnull().sum()/len(df)
plt.figure(figsize=(20,6))
plt.xticks(np.arange(len(null_counts))+0.5,null_counts.index,rotation='vertical')
plt.ylabel('fraction of rows with missing data')
plt.bar(np.arange(len(null_counts)),null_counts)
plt.title("Fraction of Missing Values")
```



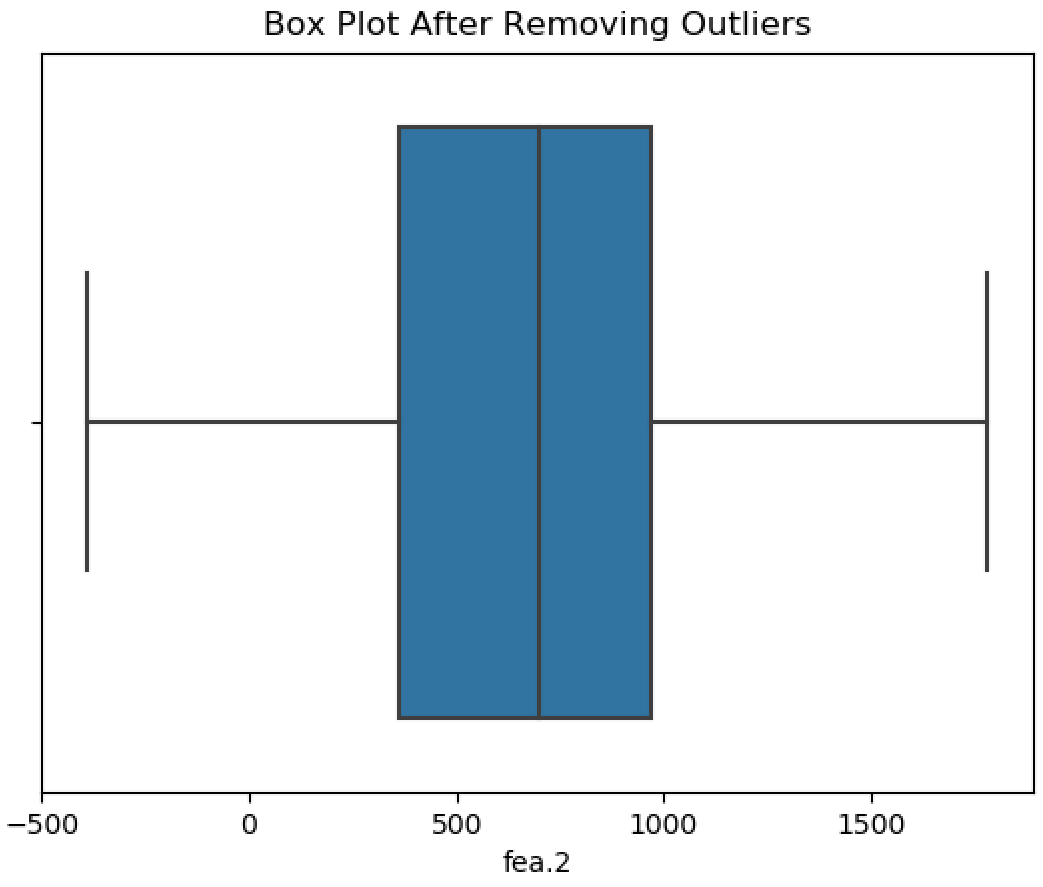
```
zz=sns.boxplot(x=df['fea.2'])
plt.title("Box Plot Before Outlier Removal",fontsize=15)
plt.show()
```

```
#three features namely:"fea.34", "fea.35", "fea.36", are almost filled with null.
So they doesn't have any information.
#missing values in every features
#Data is not normalized.
#Outliners in features such as in fea.2 indicated in box plot
```

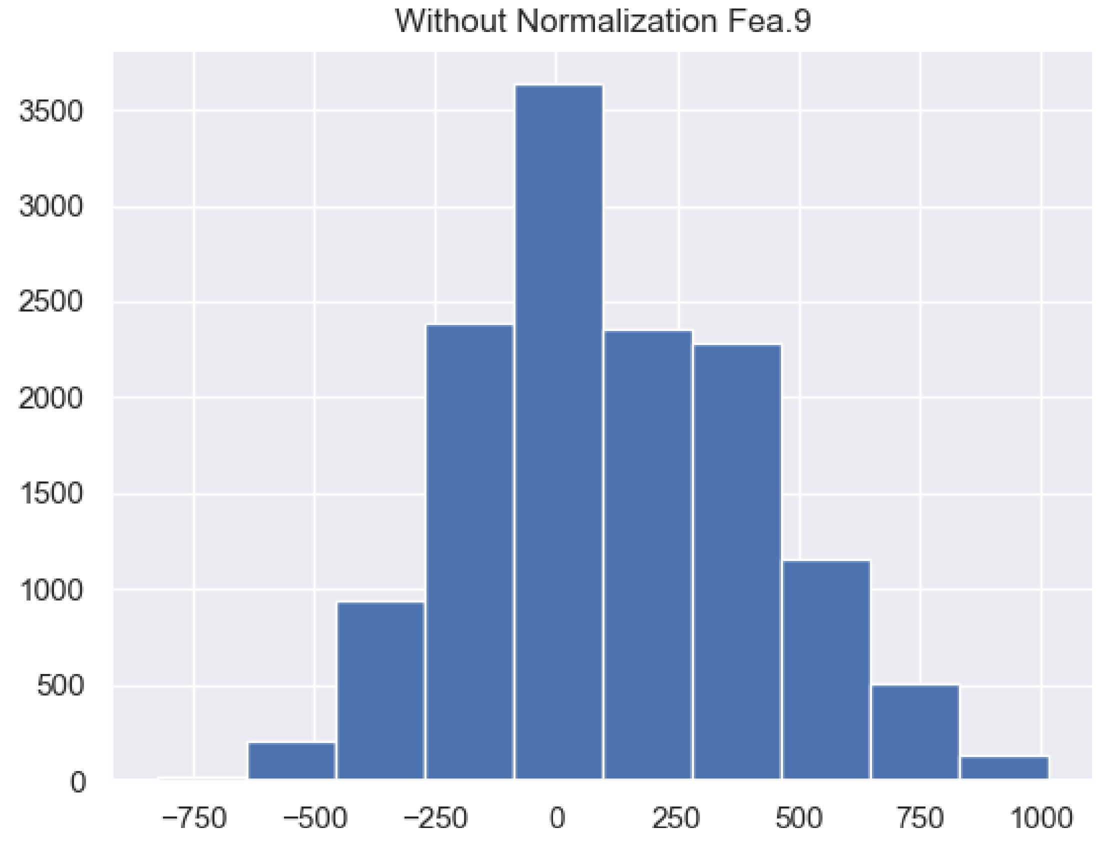


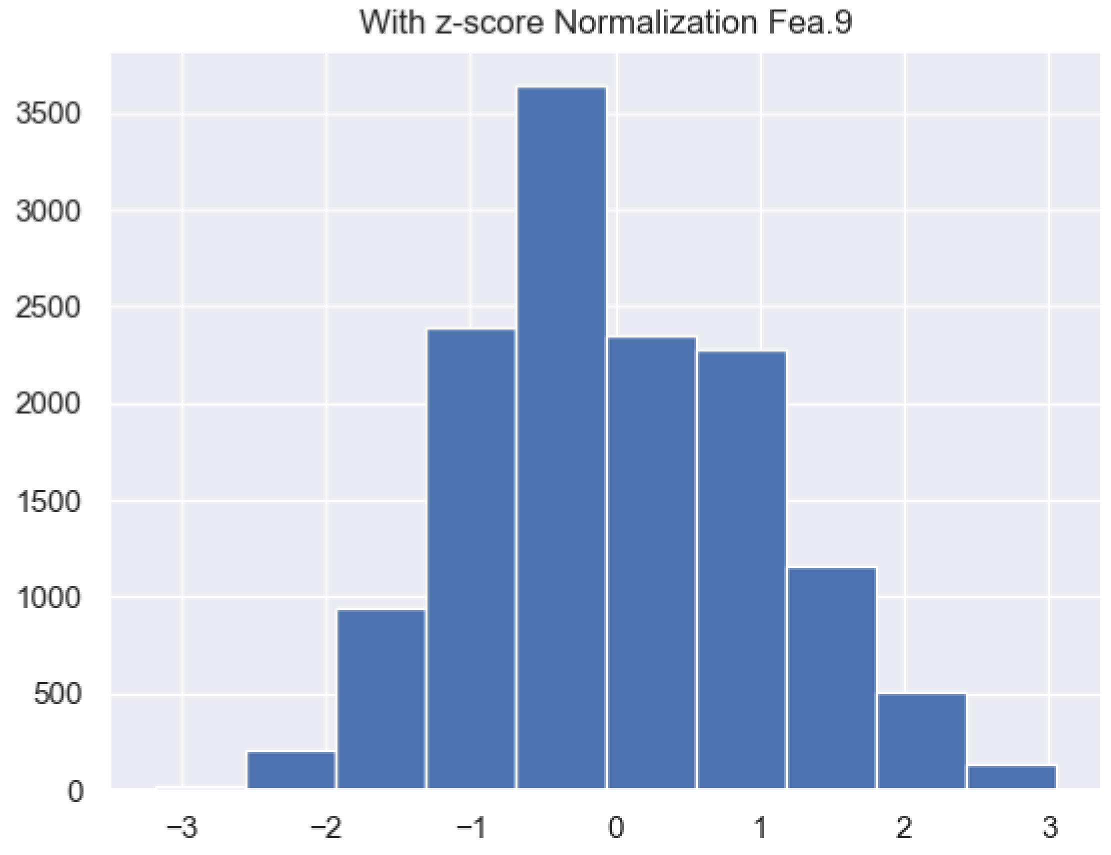
2) We solve the outlier problem by removing the values having z-score greater than 3. Following this we drop the columns which have values which are all empty. In addition, we replace the missing values in partially filled columns with the mean of that column and finally we normalize the entire data.

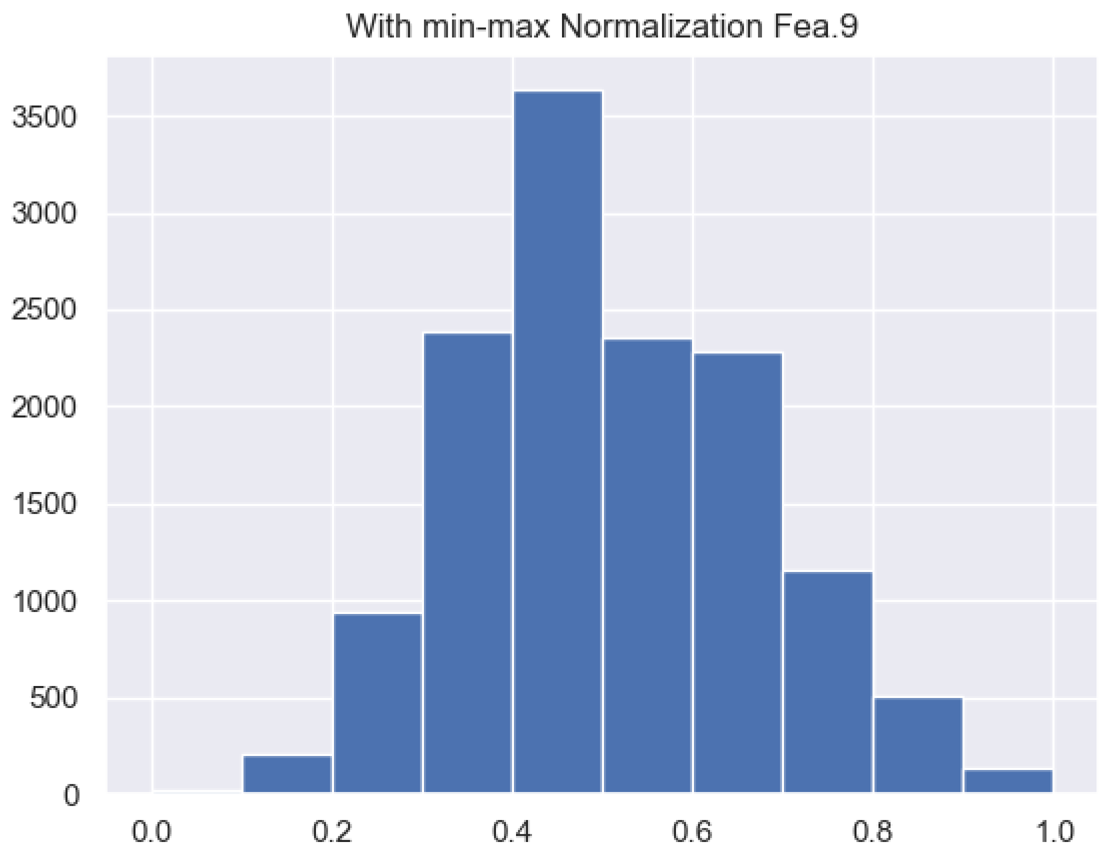
```
df = df.drop(["fea.34", "fea.36", "fea.35"], axis=1) # Remove rows/features
                                                      #with almost no information
df=df.fillna(df.mean()) #Replace NA values by mean
                        #of column



z = np.abs(stats.zscore(df))

df_o = df[(z < 3).all(axis=1)] # removing outliners
                               #with zscore>3

zzz=sns.boxplot(x=df_o['fea.2'])# outliners removed such as in fea.2
plt.title("Box Plot After Removing Outliers")
plt.show()
```



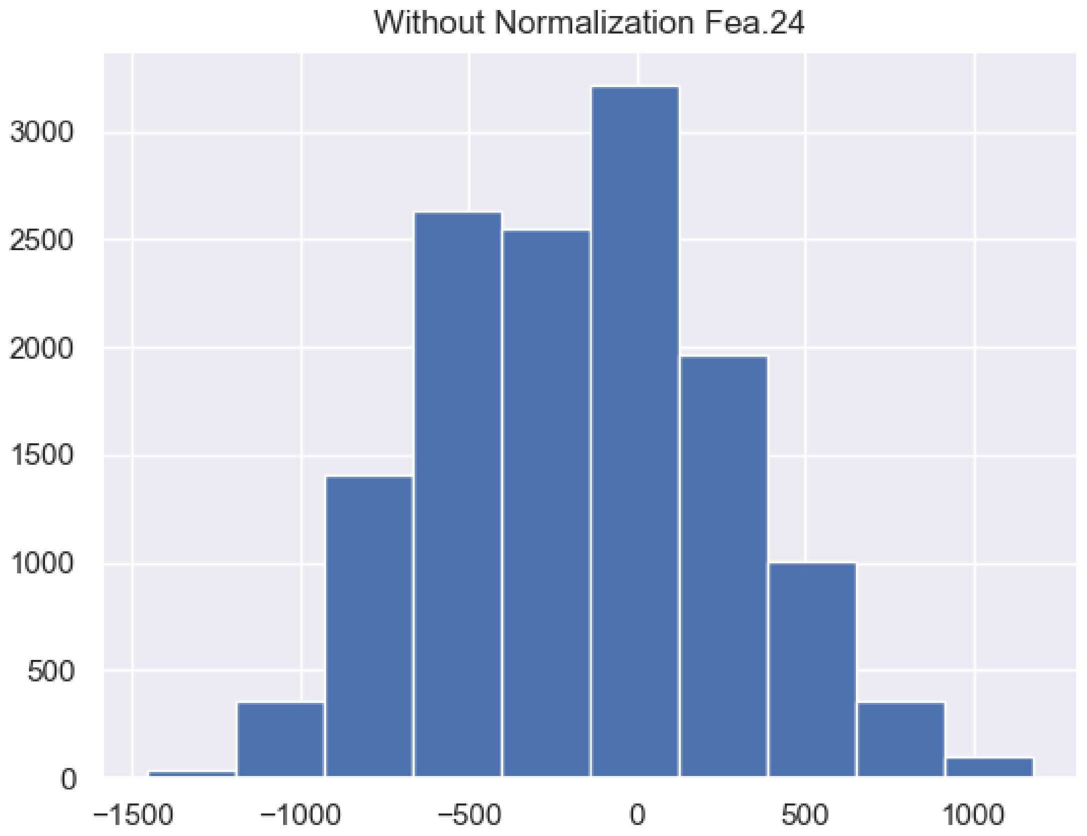
3) Before normalization the data values varies to greater extend but after normalization the values are now bounded for example between -3 and 3 in case of z-score and between 0 and 1 in case of min-max. The histograms for feature 9 and 24 before normalization and after z and min-max normalization are given below:


Without Normalization Fea.9
3500
3000
2500
2000
1500
1000
500
0
−750
−500
−250
0
250
500
750
1000

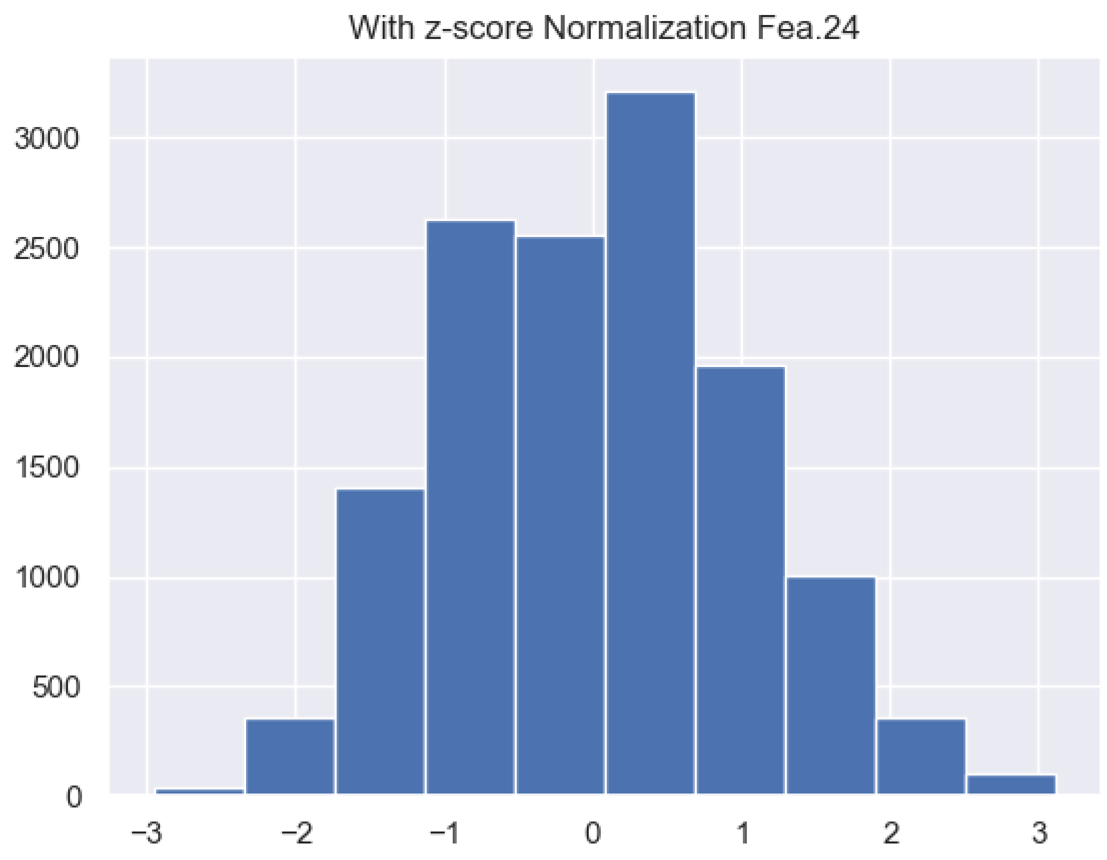

With z-score Normalization Fea.9
3500
3000
2500
2000
1500
1000
500
0
−3
−2
−1
0
1
2
3

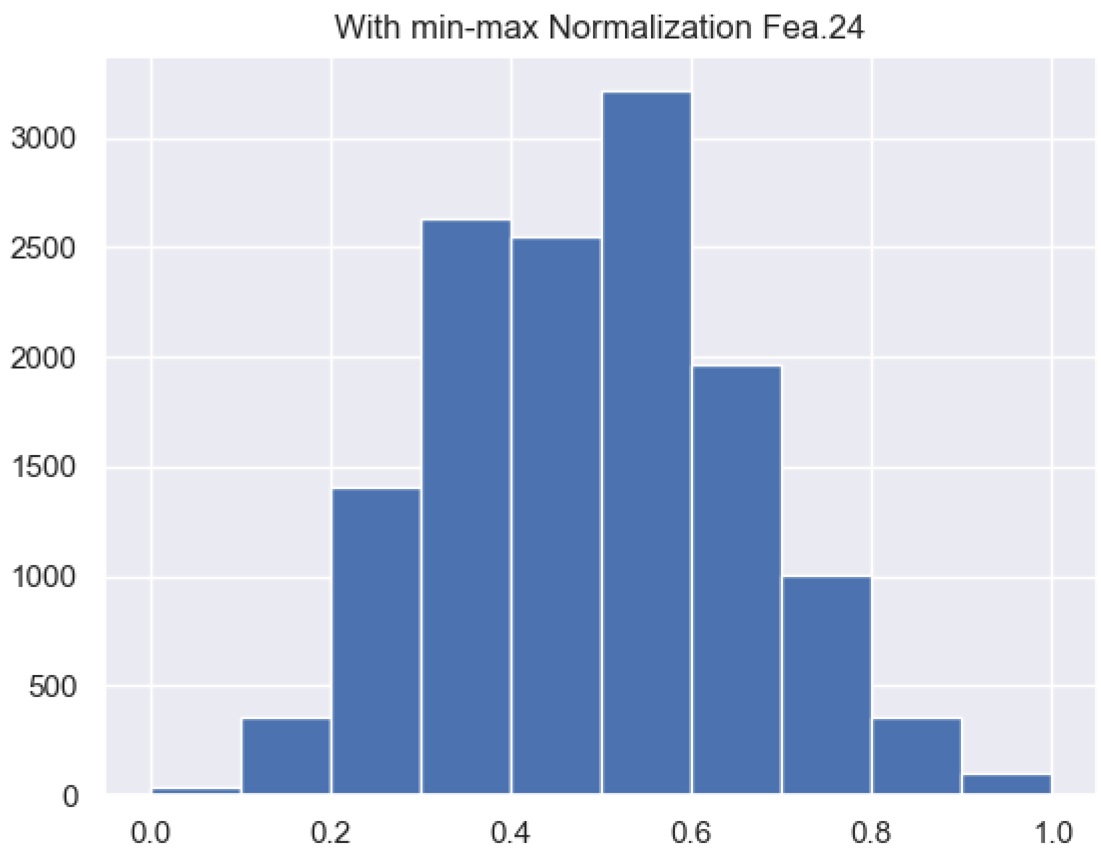


With min-max Normalization Fea.9
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0



Without Normalization Fea.24
3000
2500
2000
1500
1000
500
0
−1500
−1000
−500
0
500
1000


With z-score Normalization Fea.24
3000
2500
2000
1500
1000
500
0
−3
−2
−1
0
1
2
3



II. Feature Extraction (for dataset B)

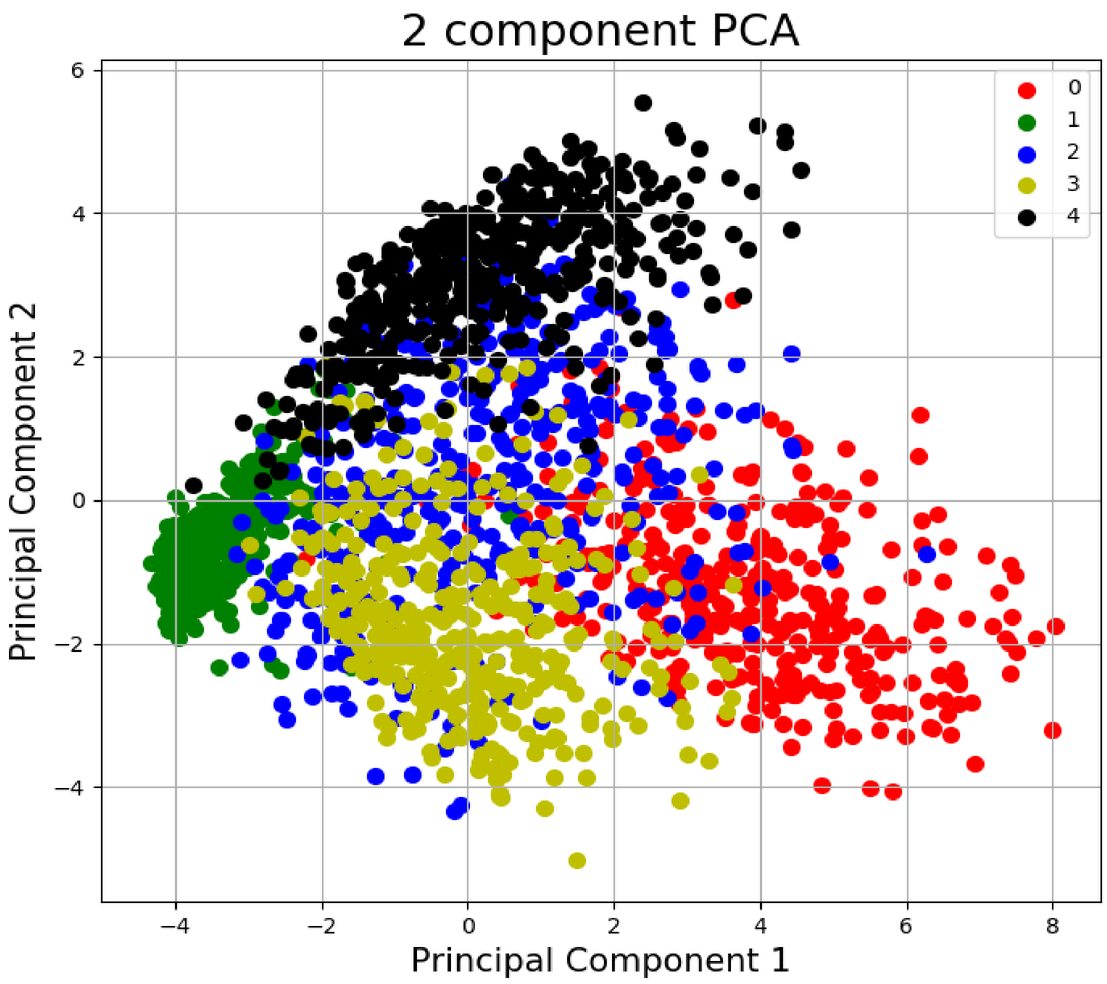
1) The dimensions of the data set “DataB” are 2066X785. The 785 dimension contains the labels which are separated to make the number of features 784 and number of examples 2066. After removing the labels, the data is centralized by removing the mean and then SVD(singular Value Decomposition) is applied to find the eigen values and the eigen vectors.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from scipy.linalg import svd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB,GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

data = pd.read_csv("C:/Users/HP/Downloads/DataB.csv")

data=data.drop(columns=[list(data)[0]]) #dropping the first column because it was
indices only
target = data[[list(data)[-1]]] #keeping the target seperate
features = data.drop(columns=[list(data)[-1]]) #removing the target from the data
to make features

X_sd = StandardScaler().fit_transform(features) # centralizing the data by removing
```

```
mean
eigen_vec,eigen_val,trans_eigen_vec = svd(X_sd) #computing eigen vectors and eigen
values
print("eigen_vectors:")
print(eigen_vec)
print("eigen_values :")
print(eigen_val)
```

2) In this part we use PCA to do dimensionality reduction and choose only first two principal components(PCs). The labels are plotted in a two dimensional grid of principal component 1 and 2. The first PC contains 10.6% of the variances of the data while the second PC contains 6.1% variance of the entire data. Even though there are overlap in the labels however we can see a great deal of distinction between different digits labelled with different colors.The plot corresponding can be seen below:

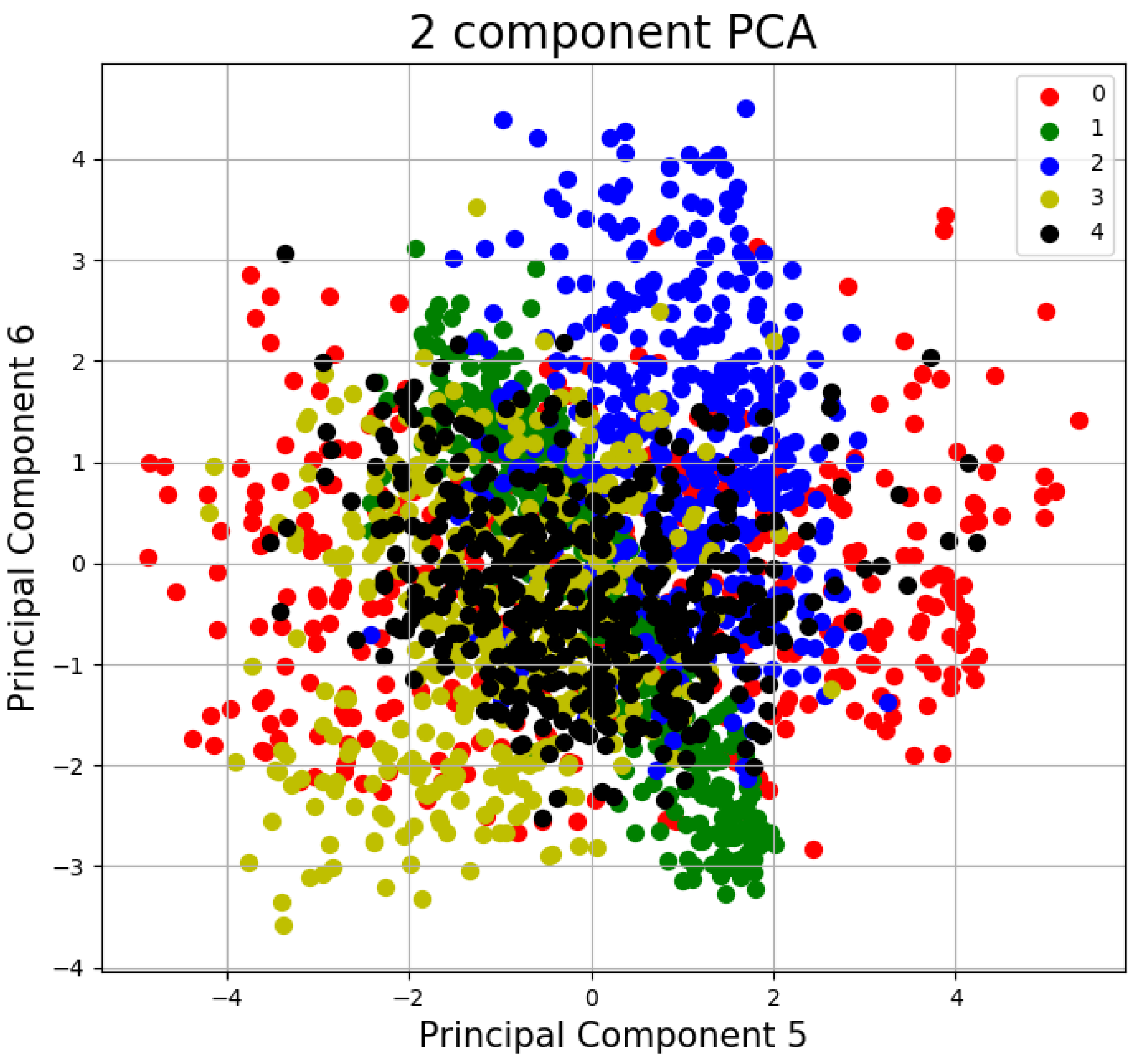
```
scaler = MinMaxScaler()
features = scaler.fit_transform(features)

#Applying PCA with two components
pca = PCA(n_components=2)
PCs12 = pca.fit(features)
print("PCA12 variances = ",pca.explained_variance_ratio_)
PCs12=pca.transform(features)
PCs12 = pd.DataFrame(data=PCs12, columns=['pc1','pc2'])

PCs12_target = pd.concat([PCs12,target],axis=1)

#Plotting the results
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

labels=[0,1,2,3,4]
colors =['r','g','b','y','k']

for label,color in zip(labels,colors):
    keep_index = PCs12_target['gnd']==label

ax.scatter(PCs12_target.loc[keep_index,'pc1'],PCs12_target.loc[keep_index,'pc2'],c=
color,s=50)
ax.legend(labels)
ax.grid()
plt.show()
```



3) In this part we are using only the fifth and sixth components of PCA to plot the labels with different colors. The fifth component contains 3.8% of the entire variance of the data while the sixth component contains 2.9% of the entire variance of the data. In the figure below in comparison to the figure in which PC 1 and PC 2 were used,the variance of the data is far less which can be seen from the x limit and y limit of the graph. Points of different colors are jumbled up and there is less distinction between the points in the 2D as compared to the figure with PC1 and PC2.

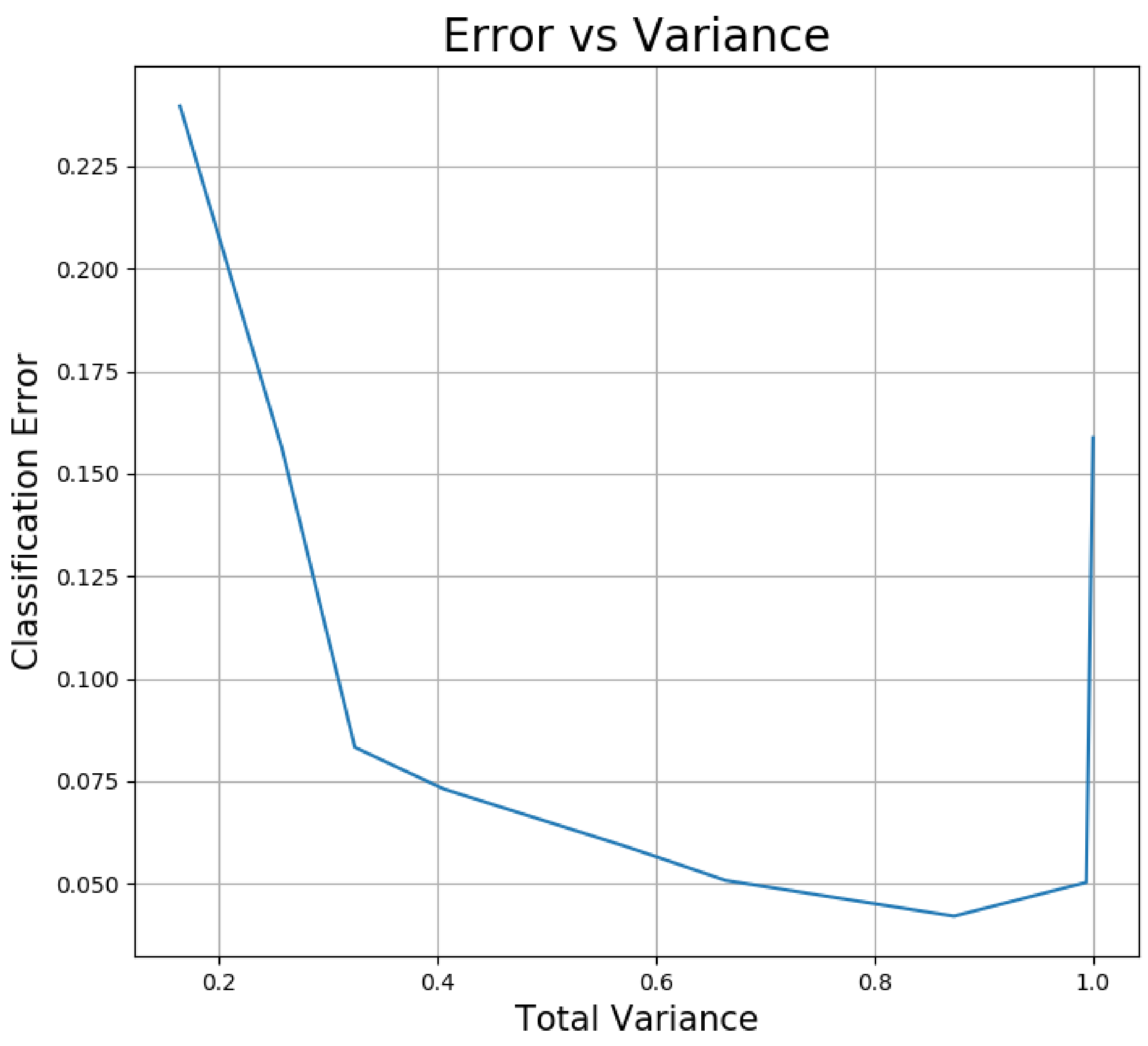
```
# Applying PCA with six components
pca = PCA(n_components=6)
PCs1to6 = pca.fit(features)
print("PCA16 variances = ",pca.explained_variance_ratio_)
PCs1to6 = pca.transform(features)
PCs1to6 = pd.DataFrame(data=PCs1to6,
columns=['pc1','pc2','pc3','pc4','pc5','pc6'])

PCs1to6_target = pd.concat([PCs1to6,target],axis=1)
PCs56_target=PCs1to6_target.drop(columns=['pc1','pc2','pc3','pc4'])


#Plotting the results
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 5', fontsize = 15)
ax.set_ylabel('Principal Component 6', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
```

```
labels=[0,1,2,3,4]
colors =['r','g','b','y','k']

for label,color in zip(labels,colors):
    keep_index = PCs56_target['gnd']==label

ax.scatter(PCs56_target.loc[keep_index,'pc5'],PCs56_target.loc[keep_index,'pc6']
,c=color,s=50)
ax.legend(labels)
ax.grid()
plt.show()
```



4) In order to train a Naïve Bayes classifier we first reduce the dimensionality of the data by applying PCA. After this we used the reduced dimensionality data to train the classifier and predict the labels. We try a variety of different dimensions and compute the classification across against the variance captured. From the figure below it can be easily seen that error increases as we increase the number of components beyond a certain point. As we move from using 200 components to 500 components, the error increases from 4.4% to 5.2% and then if we increase the components further to a maximum of 784 components the error increases to 15.87%. This means the optimal value of number of components lies between using 200 to 500 components.

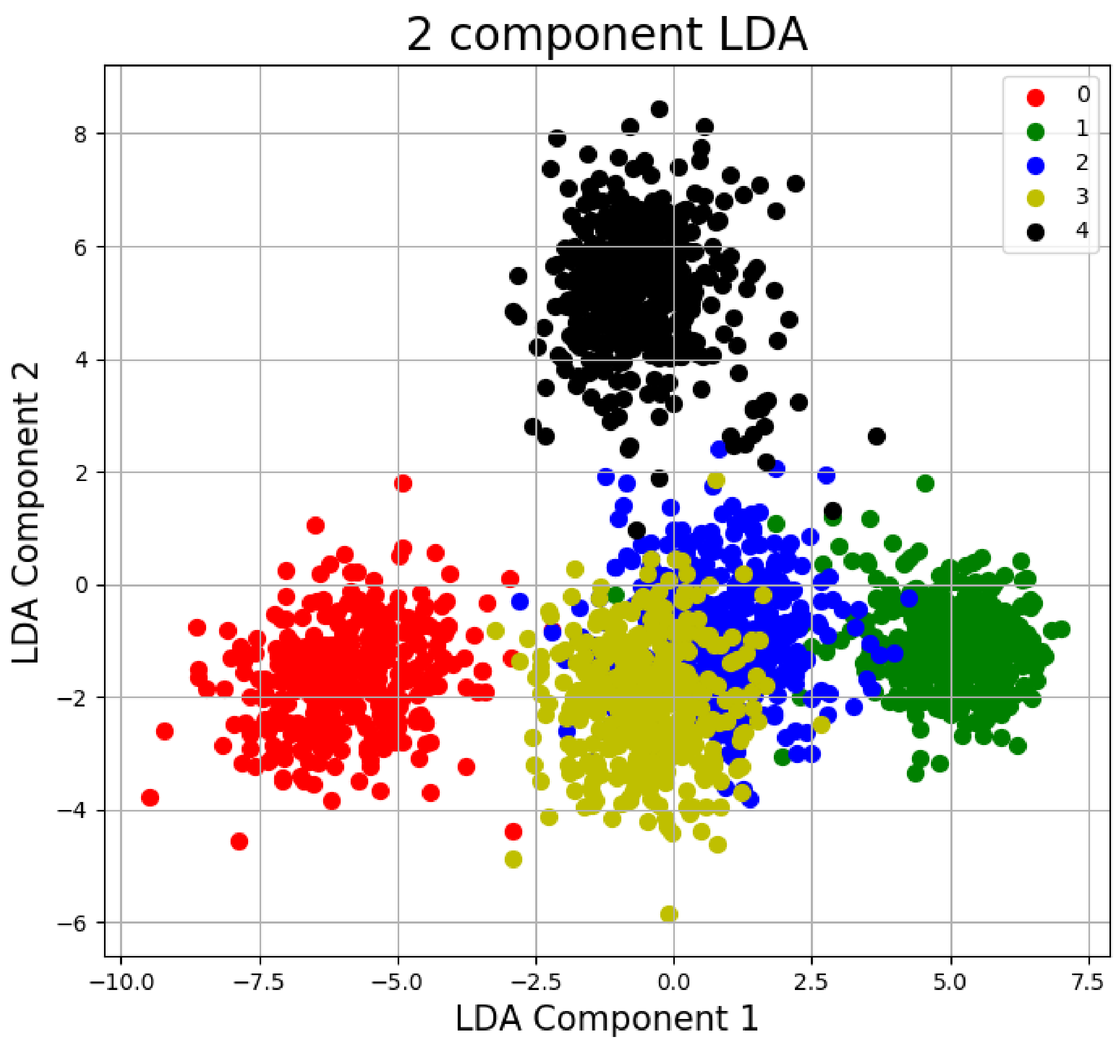
```
nc =[2,4,6,10,30,60,200,500,784]
list_accuracy=[]
list_var=[]
list_error=[]
#Training the Naive Bayes Classifier
for i in range(0,len(nc)):
    pca = PCA(n_components=nc[i])
    print(nc[i])
    pca.fit(features)
    list_var.append(np.sum(pca.explained_variance_ratio_))#computing the variances
combined
    x=pca.transform(features)
    nb=GaussianNB()
    nb.fit(x,target.values)
    y_pred=nb.predict(x)
    list_accuracy.append(accuracy_score(target.values, y_pred))#computing the
accuracy
    list_error.append(1-accuracy_score(target.values, y_pred))#computing the error

print("list_of_errors = ", list_error)
print("list_of_variances = ",list_var)
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Total Variance', fontsize = 15)
ax.set_ylabel('Classification Error', fontsize = 15)
ax.set_title('Error vs Variance', fontsize = 20)

ax.plot(list_var,list_error)
ax.grid()
plt.show()
```



5) In this part of the problem we apply LDA(Linear Discriminant Analysis) also known as FDA(Fischer's Discriminant Analysis) to the data set with reduced dimensions = 2. This is a supervised learning technique unlike PCA. The labels are used to minimize the within class variance and maximize the between class variance. This results in dimensionality reduction with complete distinction of the output classes from each other. The information regarding the labels is not lost along the way. This can be seen in the figure below where we can see separate clusters for separate classes.

```
#Applying LDA
lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(features,target)
lda_feats = lda.transform(features)
LDA_12 = pd.DataFrame(data=lda_feats, columns=['lda1','lda2'])
LDA_12_target = pd.concat([LDA_12,target],axis=1)
#Plotting results
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('LDA Component 1', fontsize = 15)
ax.set_ylabel('LDA Component 2', fontsize = 15)
ax.set_title('2 component LDA', fontsize = 20)

labels=[0,1,2,3,4]
```

```
colors =['r','g','b','y','k']

for label,color in zip(labels,colors):
    keep_index = LDA_12_target['gnd']==label

ax.scatter(LDA_12_target.loc[keep_index,'lda1'],LDA_12_target.loc[keep_index,'lda2'
],c=color,s=50)
ax.legend(labels)
ax.grid()
```



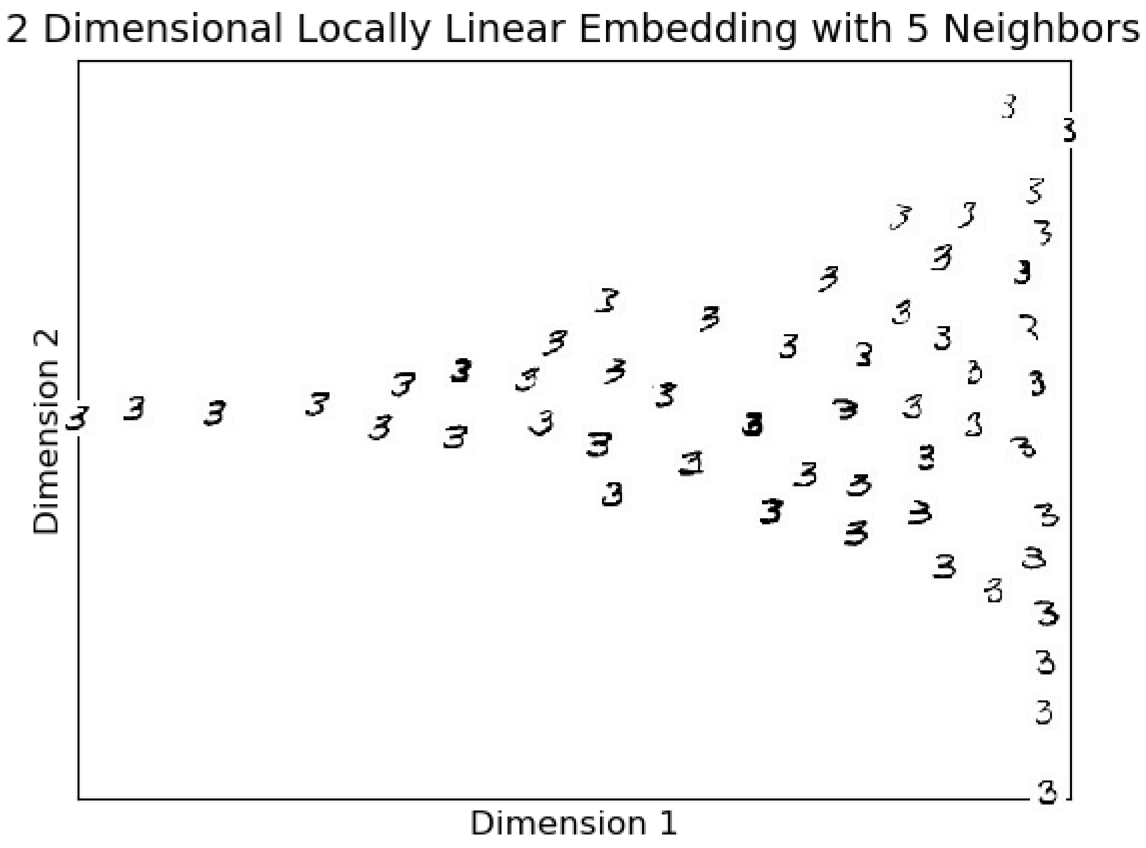
```
plw()
```

III. Nonlinear Dimensionality Reduction (for dataset B)

To begin with we separate out the data with the class labels equal to three and for part one and two, the analysis is done only for class label three only by using number of neighbors equal to five but number of dimensions equal to four and then choosing the first two dimensions for plotting the images of three in a 2D grid.

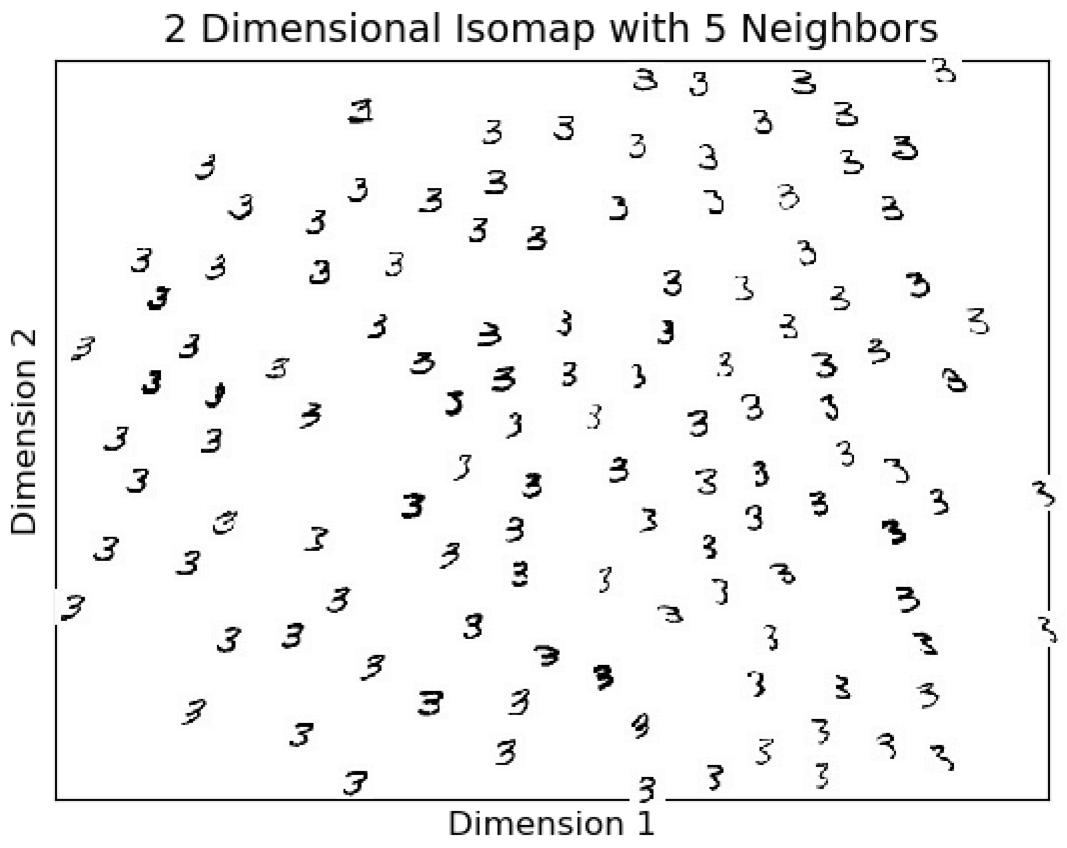
1) In this part we use LLE(Local Linear Embedding) which captures more of the local features as compared to global features. We start by first finding the nearest neighbors using KNN in the higher dimensional space. Following this we try to reconstruct data points by using the N nearest neighbors, in our case 5. In doing so we find the weight vector required to recreate the original data points. Finally, by using these weight vector we find a lower dimensional representation of the original data points. However, only local features are observed because we are using only the neighbor(local) information to reconstruct the points.

```
import numpy as np
import pandas as pd
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.preprocessing import MinMaxScaler
from matplotlib import offsetbox
import matplotlib.pyplot as plt
from matplotlib.offsetbox import AnnotationBbox, OffsetImage
from sklearn.decomposition import PCA
from numpy.linalg import norm
from sklearn.manifold import Isomap
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB,GaussianNB
from time import time
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

data = pd.read_csv("C:/Users/HP/Downloads/DataB.csv")


data=data.drop(columns=[list(data)[0]]) #dropping the first column because it was
indices only
data=pd.DataFrame(data[data['gnd']==3]) #removing the data with labels which are 3
print(data.head())

features = data.drop(columns=[list(data)[-1]]) #removing the target from the data
to make features
target = data[[list(data)[-1]]] #keeping the target seperate
image_data=features.values
n_samples, n_features=image_data.shape,image_data.shape

scaler = MinMaxScaler()
features = scaler.fit_transform(features)

def plot_embedding(X,image_data, title=None):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)

    plt.figure()
    ax = plt.subplot(111)
    if hasattr(offsetbox, 'AnnotationBbox'):
        # only print thumbnails with matplotlib > 1.0
        shown_images = np.array([[1., 1.]])  # just something big
        for i in range(X.shape[0]):
            dist = np.sum((X[i] - shown_images) ** 2, 1)
            if np.min(dist) < 3e-3:
                # don't show points that are too close
                continue
            shown_images = np.r_[shown_images, [X[i]]]
            imagebox = offsetbox.AnnotationBbox(
                offsetbox.OffsetImage(image_data[i,:].reshape(28,28),zoom=.45,
cmap=plt.cm.gray_r),
                X[i],frameon=False)
            ax.add_artist(imagebox)
    plt.xticks([]), plt.yticks([])
    plt.xlabel("Dimension 1",fontsize=12)
    plt.ylabel("Dimension 2",fontsize=12)
    if title is not None:
        plt.title(title,fontsize=15)


lle = LocallyLinearEmbedding(n_neighbors = 5,n_components=4)
lle_feats=lle.fit_transform(features)
LLE_14 = pd.DataFrame(data=lle_feats,columns=['lle1','lle2','lle3','lle4'])
LLE_12=LLE_14.drop(columns=['lle3','lle4']) #taking only the first dimensions for
plotting



plot_embedding(LLE_12.values,image_data,"2 Dimension Locally Linear Embedding with
5 Neighbors")
plt.show()
```



2) In this part of the problem we use Isomap with 5 neighbors which tries to capture global features of the higher dimension in a lower dimensional embedding. Similar to LLE it finds the nearest neighbors of the data point by using KNN but after that it forms a graph in which each point is connected to its nearest neighbors. The pair wise geodesic distances are then computed by finding the shortest path along the graph. Once these pair wise shortest distances are computed then finally MDS (Multidimensional Scaling) is applied on these pair wise distances to find a lower dimensional embedding. The computation of geodesic distances allows Isomap to capture global characteristics of the data. LLE on the other hand preserves the local properties of the data and assumes that the global manifold can be reconstructed by "local" or small connecting regions that are overlapped as if the neighborhoods are small and the manifolds are approximately linear. Hence, as Isomap also preserves features therefore it can be seen as the better of the two techniques.

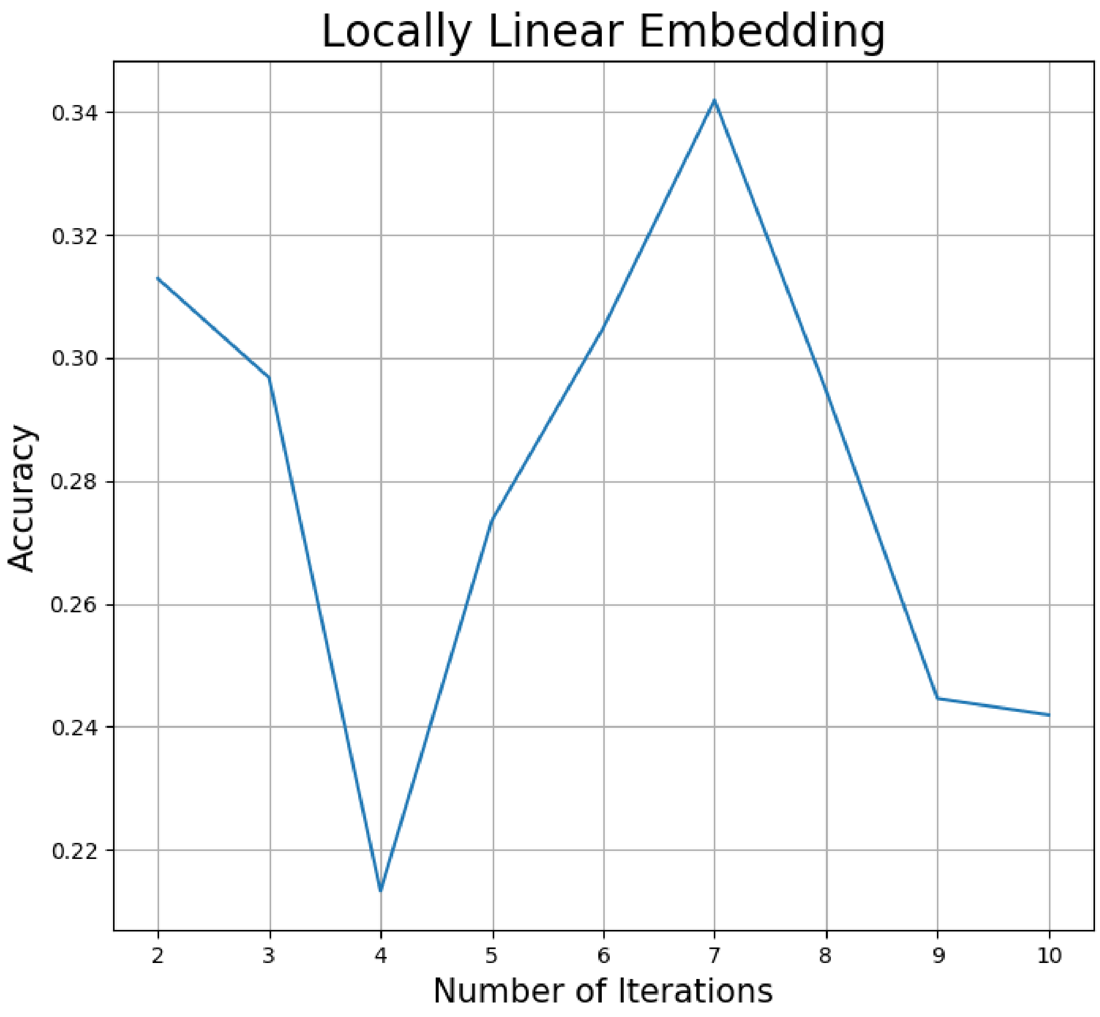
```
isomap = Isomap(n_neighbors=5,n_components = 4)
isomap_feats = isomap.fit_transform(features)
isomap_14 = pd.DataFrame(data=isomap_feats,columns=['iso1','iso2','iso3','iso4'])
isomap_12=isomap_14.drop(columns=['iso3','iso4'])


plot_embedding(isomap_12.values,image_data," 2 Dimensional Isomap with 5 Neighbors
" )
plt.show()
```

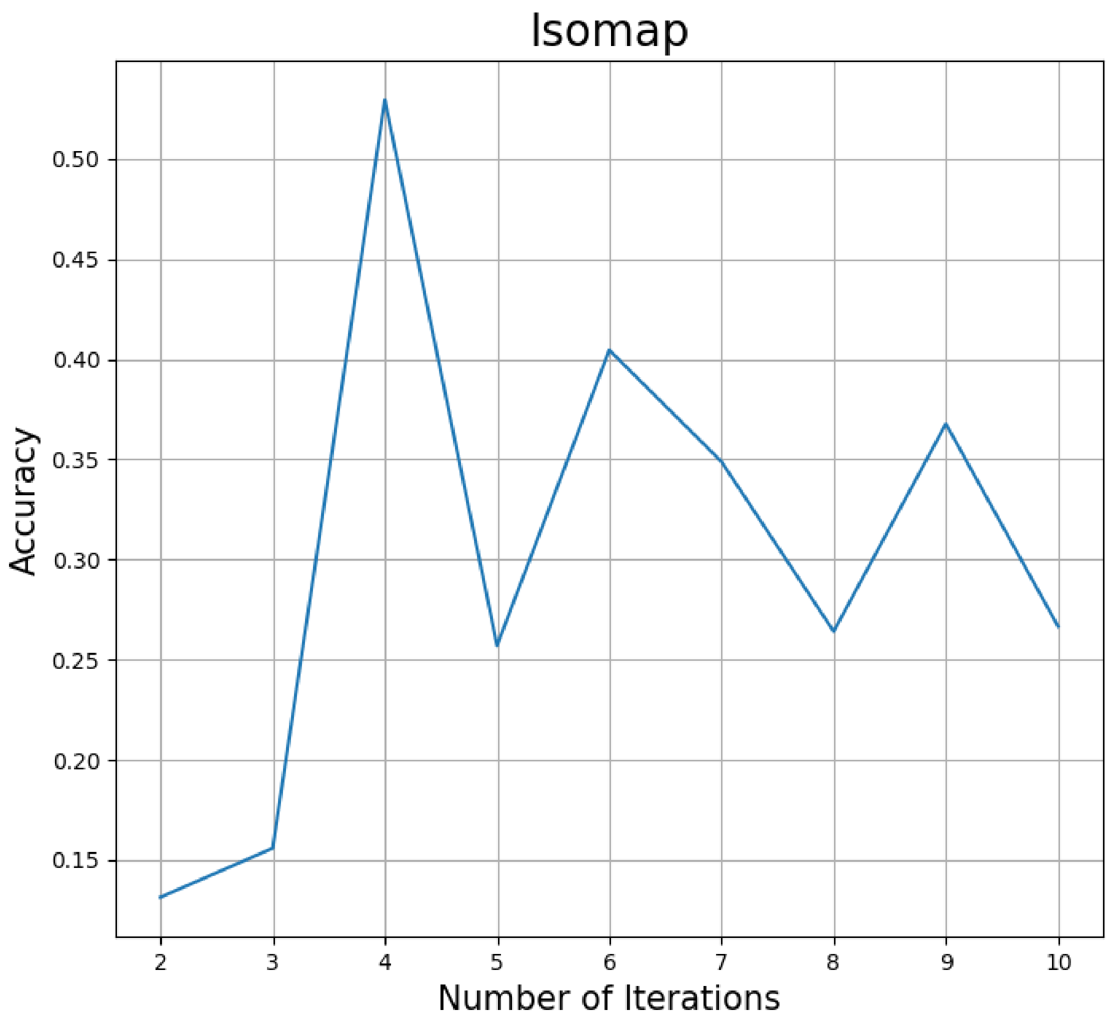


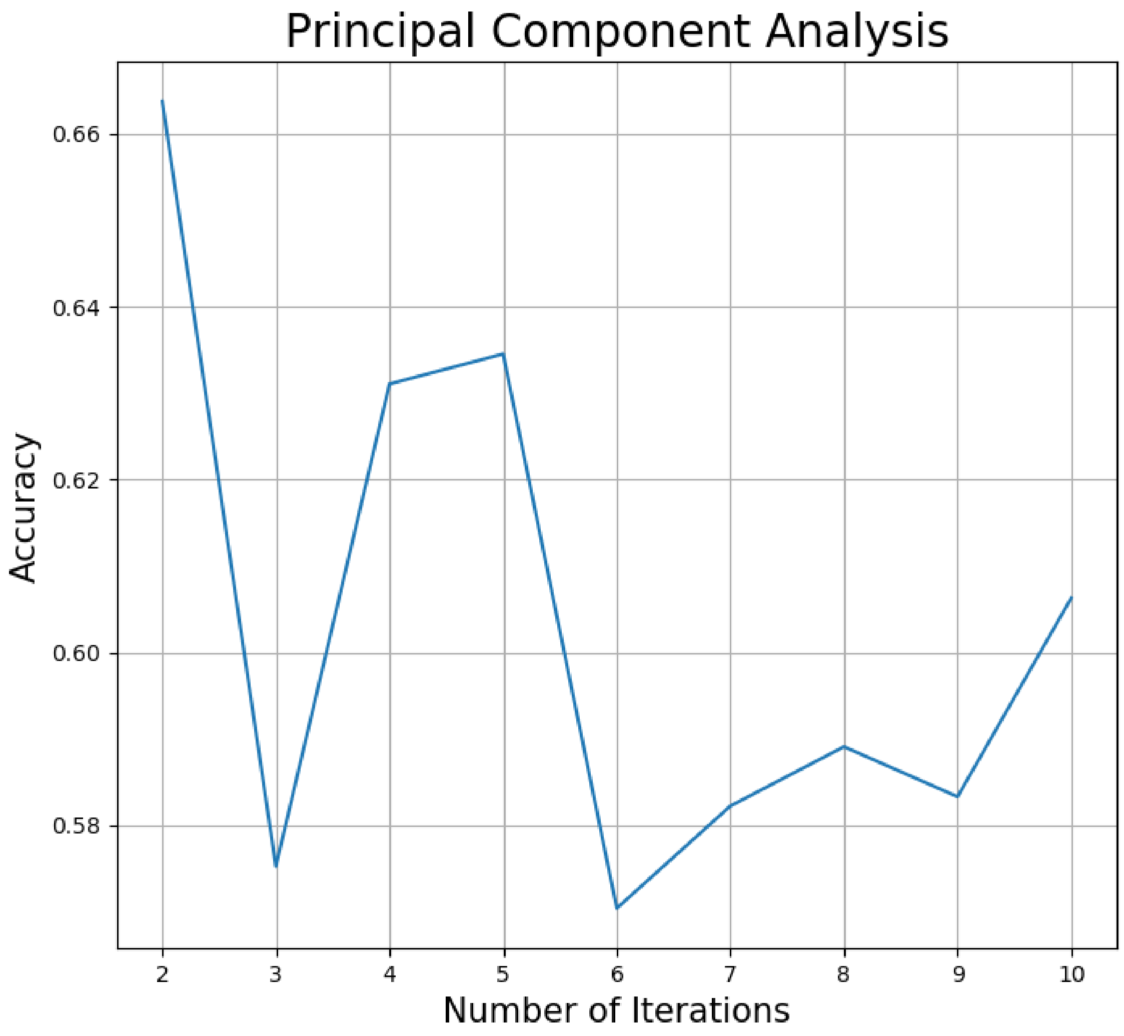
3) In this last part we are training Naïve Bayes algorithm using LLE and Isomap. We then by using train set of 70% and test set of 30% train the algorithm using multiple iterations such that the location of the training and test data sets vary inside the volume of the entire data keeping the percentages used for train and test set same as before i.e(70%,30%) correspondingly. This is pretty similar to K-fold cross validation. In each number of iterations an average score is computed and the results are plotted against the number of iterations to see how many number of iterations produce optimal results. The results for Isomap and LLE are then compared with PCA and LDA. All the dimensionality reduction algorithms are used with number components being equal to 4. Below are accuracy-iteration graphs for all the techniques. It can be seen easily that for Isomap and LLE , the optimal number of iterations are 4 and 7 respectively while PCA and LDA perform the best at 2 iterations. As a whole on average PCA and LDA perform much better than Isomap and LLE due to their higher average accuracy.

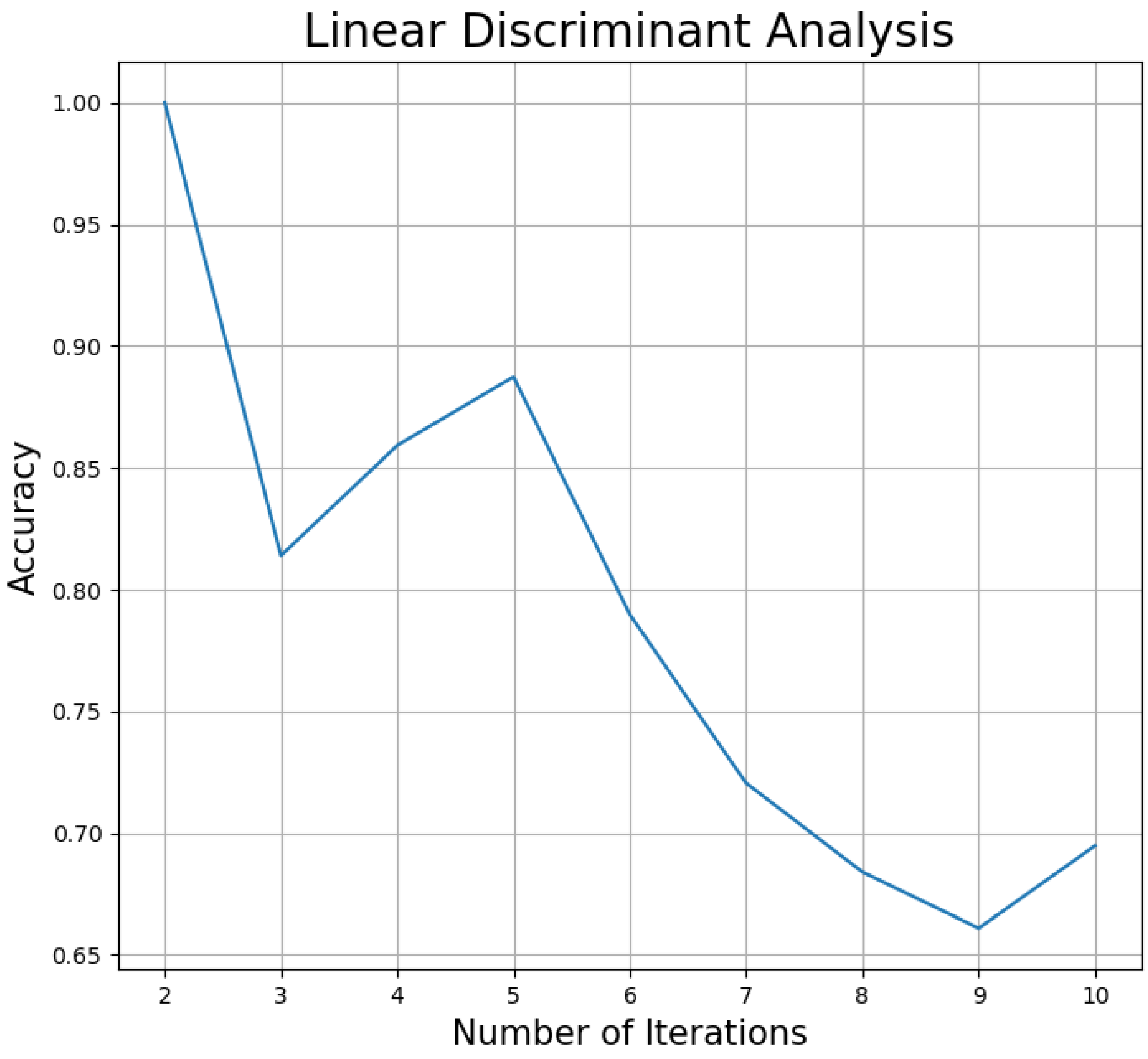
```
data = pd.read_csv("C:/Users/HP/Downloads/DataB.csv")


data=data.drop(columns=[list(data)[0]]) #dropping the first column because it was 
indices only

features = data.drop(columns=[list(data)[-1]]) #removing the target from the data 
to make features
target = data[[list(data)[-1]]] #keeping the target seperate
```

```python
scaler = MinMaxScaler()
features = scaler.fit_transform(features)


def score_lle(x_train,y_train,x_test,y_test):
    lle = LocallyLinearEmbedding(n_neighbors=5, n_components=4)
    x_train=lle.fit_transform(x_train)
    x_test=lle.fit_transform(x_test)
    nb=GaussianNB()
    nb.fit(x_train,y_train)
    y_pred=nb.predict(x_test)
    return accuracy_score(y_pred,y_test)


def score_isomap(x_train,y_train,x_test,y_test):
    iso = Isomap(n_neighbors=5, n_components=4)
    x_train=iso.fit_transform(x_train)
    x_test=iso.fit_transform(x_test)
    nb=GaussianNB()
    nb.fit(x_train, y_train)
    y_pred = nb.predict(x_test)
    return accuracy_score(y_pred, y_test)

def score_pca(x_train,y_train,x_test,y_test):
    pca = PCA(n_components=4)
    x_train=pca.fit_transform(x_train)
    x_test=pca.fit_transform(x_test)
    nb=GaussianNB()
    nb.fit(x_train, y_train)
    y_pred = nb.predict(x_test)
    return accuracy_score(y_pred, y_test)

def score_lda(x_train,y_train,x_test,y_test):
    lda1 = LinearDiscriminantAnalysis(n_components=4)
    lda2 = LinearDiscriminantAnalysis(n_components=4)
    lda1.fit(x_train,y_train)
    x_train=lda1.transform(x_train)
    lda2.fit(x_test,y_test)
    x_test=lda2.transform(x_test)
    nb=GaussianNB()
    nb.fit(x_train, y_train)
    y_pred = nb.predict(x_test)
    return accuracy_score(y_pred, y_test)

scores_lle=[]
scores_isomap=[]
scores_pca=[]
scores_lda=[]
iterations=[]
for i in range(2,11):
    print("here = ",i)
    iterations.append(i)
    cv = ShuffleSplit(n_splits=i, test_size=0.30, random_state=0)
    scores_1 = []
    scores_2 = []
    scores_3 = []
    scores_4 = []
    for train_index, test_index in cv.split(features):
        x_train = features[train_index]
        x_test = features[test_index]
        target = np.array(target)
        y_train = target[train_index]
        y_test = target[test_index]
        scores_1.append(score_lle(x_train,y_train,x_test,y_test))
        scores_2.append(score_isomap(x_train, y_train, x_test, y_test))
        scores_3.append(score_pca(x_train, y_train, x_test, y_test))
        scores_4.append(score_lda(x_train, y_train, x_test, y_test))
    scores_lle.append(np.mean(np.array(scores_1)))
    scores_isomap.append(np.mean(np.array(scores_2)))
    scores_pca.append(np.mean(np.array(scores_3)))
    scores_lda.append(np.mean(np.array(scores_4)))

print("lle = ",scores_lle)
print("isomap = ",scores_isomap)
print("pca = ",scores_pca)
print("lda = ",scores_lda)
print("iterations = ",iterations)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Number of Iterations', fontsize = 15)
ax.set_ylabel('Accuracy', fontsize = 15)
ax.set_title('Locally Linear Embedding', fontsize = 20)

ax.plot(iterations,scores_lle)
ax.grid()

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Number of Iterations', fontsize = 15)
ax.set_ylabel('Accuracy', fontsize = 15)
ax.set_title('Isomap', fontsize = 20)

ax.plot(iterations,scores_isomap)
ax.grid()

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Number of Iterations', fontsize = 15)
ax.set_ylabel('Accuracy', fontsize = 15)
ax.set_title('Principal Component Analysis', fontsize = 20)

ax.plot(iterations,scores_pca)
ax.grid()

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Number of Iterations', fontsize = 15)
ax.set_ylabel('Accuracy', fontsize = 15)
ax.set_title('Linear Discriminant Analysis', fontsize = 20)

ax.plot(iterations,scores_lda)
ax.grid()
plt.show()
```


Locally Linear Embedding
Accuracy
0.34
0.32
0.30
0.28
0.26
0.24
0.22
2
3
4
5
6
7
8
9
10
Number of Iterations


Isomap
Accuracy
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
2
3
4
5
6
7
8
9
10
Number of Iterations


Principal Component Analysis
Accuracy
0.66
0.64
0.62
0.60
0.58
2
3
4
5
6
7
8
9
10
Number of Iterations


Linear Discriminant Analysis
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
2
3
4
5
6
7
8
9
10
Number of Iterations