minimizing variance shared among all classes respectively. Thus a few components obtained using PCA and LDA(few plots using few components) might summarize the dataset well compared to same number of features used directly from dataset.

Q 2

2.1 Dataset

In [139]:

```
import numpy as np
import pandas as pd
import random
import seaborn as sns
sns.set(style="ticks", color_codes=True)
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
DataB = pd.read_csv("DataB.csv", sep=',')
DataB_features = DataB.drop(['gnd'], axis =1)
DataB_features.columns
DataB_features.rename(columns={'Unnamed: 0': "indexed"}, inplace = True)
DataB_features.indexed = DataB_features.indexed -1
DataB_features.set_index('indexed', inplace = True)
DataB_features.shape
X = DataB_features
DataB_class = DataB.gnd
DataB_class = DataB_class.to_frame()
DataB_class.rename_axis('indexed', inplace = True)
DataB_class.rename(columns={'gnd': "class"}, inplace = True)
```

2.2 Principal Component Analysis

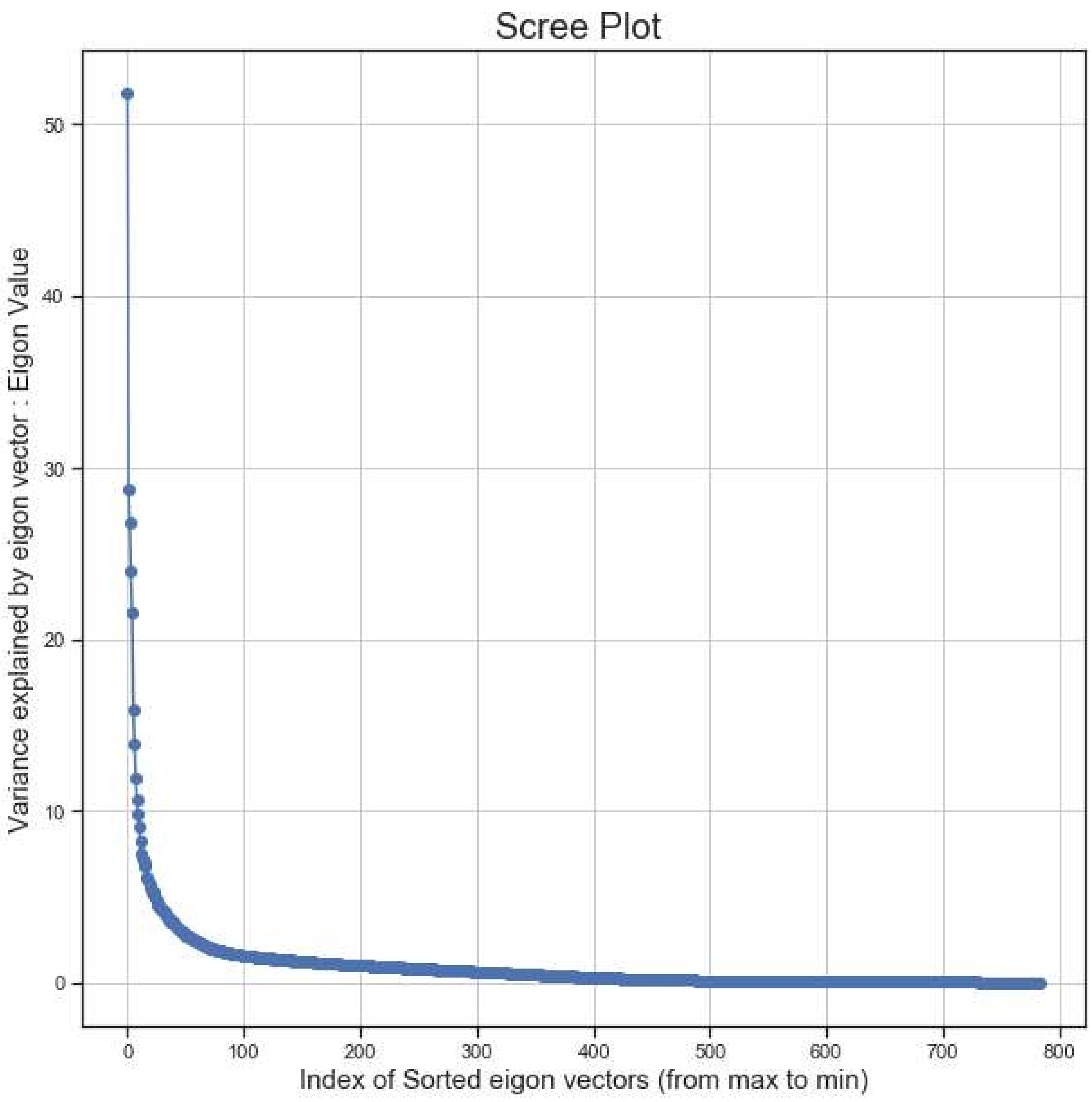
2.2.1 Practical Questions

1) In PCA, compute the eigenvectors and eigenvalues. Plot the scree plot and visually discuss which cut-off is good.

In [140]:
```
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
DataB_features_scaled = Scaler.fit_transform(DataB_features)
from sklearn.decomposition import PCA
pca = PCA(random_state =42)
X_pca = pca.fit_transform(DataB_features_scaled)
PCA_eigon_vector_data = pca.components_.T
col_names = ["Eigon Vector " + str(i) for i in range(1,785)]
PCA_eigon_vector_dataframe = pd.DataFrame(data = PCA_eigon_vector_data, columns=
eigon_values = pca.explained_variance_
print("eigon_values are : ", eigon_values)
print("eigon vectors are : ", PCA_eigon_vector_data)
```

```
 1.80894894e+00 1.80037769e+00 1.79143455e+00 1.77821823e+00
 1.75344083e+00 1.74420633e+00 1.73235499e+00 1.71703221e+00
 1.69560958e+00 1.67660264e+00 1.67204995e+00 1.66413703e+00
 1.65298469e+00 1.64607810e+00 1.62658205e+00 1.61023500e+00
 1.59724824e+00 1.59323756e+00 1.58168294e+00 1.56824196e+00
 1.55793648e+00 1.55649266e+00 1.55126745e+00 1.53478544e+00
 1.52250847e+00 1.51844179e+00 1.50526693e+00 1.50403086e+00
 1.49832877e+00 1.49084155e+00 1.48107619e+00 1.47692460e+00
 1.47144049e+00 1.46645002e+00 1.44965194e+00 1.44504737e+00
 1.44001722e+00 1.42705114e+00 1.42253474e+00 1.41615253e+00
 1.40832241e+00 1.40224124e+00 1.39570127e+00 1.38879684e+00
 1.37904784e+00 1.37790604e+00 1.36580346e+00 1.36372072e+00
 1.35608476e+00 1.34791805e+00 1.33889606e+00 1.33011761e+00
 1.32803706e+00 1.32266875e+00 1.31925957e+00 1.30663444e+00
 1.30378173e+00 1.29900900e+00 1.29700677e+00 1.29230361e+00
 1.28317401e+00 1.27869566e+00 1.27308404e+00 1.26556014e+00
 1.25805205e+00 1.25428149e+00 1.25068531e+00 1.24549638e+00
 1.24082952e+00 1.23076321e+00 1.22902220e+00 1.22246312e+00
 1.21690294e+00 1.21022711e+00 1.20439342e+00 1.19694226e+00
 1.19615017e+00 1.18957391e+00 1.18452868e+00 1.17925785e+00
```

In [141]:
```python
# lets plot scree plot for all components
eigon_value_df = pd.DataFrame(data = eigon_values, columns = ['Eigon Values'])
#plt.plot(eigon_value_df)
fig = plt.figure(figsize =(10,10))
ax = fig.add_subplot(1,1,1)
ax.plot(eigon_value_df, marker ='o')
ax.set_ylabel('Variance explained by eigon vector : Eigon Value', fontsize = 15)
ax.set_xlabel('Index of Sorted eigon vectors (from max to min)', fontsize = 15)
ax.set_title('Scree Plot ', fontsize = 20)
ax.grid()
```



Deciding how to choose cut off point

Looking at the above scree plot of the principal components and their corresponding eigen values, it can be observed that with increase in the index of the principal components their corresponding eigen value decreases. Moreover, decrease in the eigen values are very steep for the initial principal components while after certain value they seem to have approximately equal value which can be observed at approximately 370th principal component.

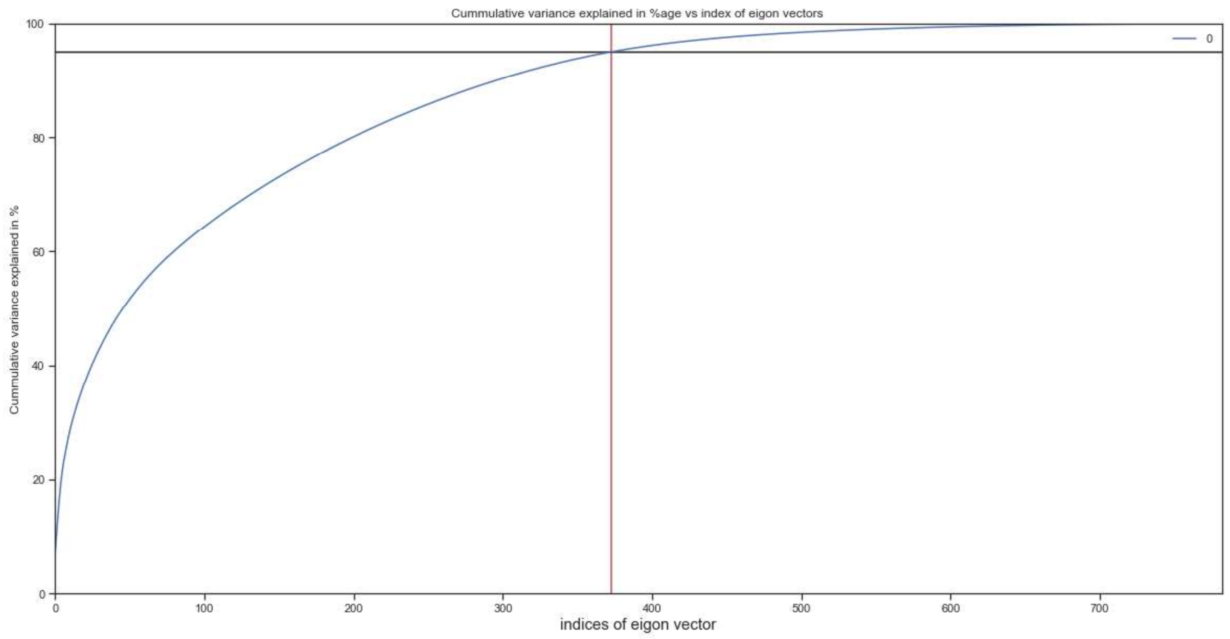
Moreover, similar threshold can also be obtained by plotting the cumulative variance explained by the principal component. Looking at the below plotted graph it can be depicted that the 95% variance of the total data can be explained using the first 373 principal components.

In [142]:
```
#plotting cummulative % variance plot
L = pca.explained_variance_ratio_.tolist()
for i in range(0,len(L)-1):
    L[i] = L[i]*100
Cum_L= []
Cum_L.append(L[0])
sum1 = L[0]
for i in range(1, len(L)-1):
    sum1 = sum1 + L[i]
    Cum_L.append(sum1)

for k in range(0, len(Cum_L)):
    if Cum_L[k] >= 95:
        break


Cum_L = pd.DataFrame(data=Cum_L)
ax = Cum_L.plot(figsize = (20,10), ylim =(0,100), title = "Cummulative variance 
ax.set_xlabel('indices of eigon vector', fontsize = 15)
ax.set_ylabel('Cummulative variance explained in %')
ax.vlines(x = k,ymin = 0, ymax =100 , color = 'r')
ax.hlines( y= 95, xmin =0, xmax =785, color = 'k' )
print("number of eigon vectors that explain 95 % variance ",k)
```

```
number of eigon vectors that explain 95 % variance  373
```

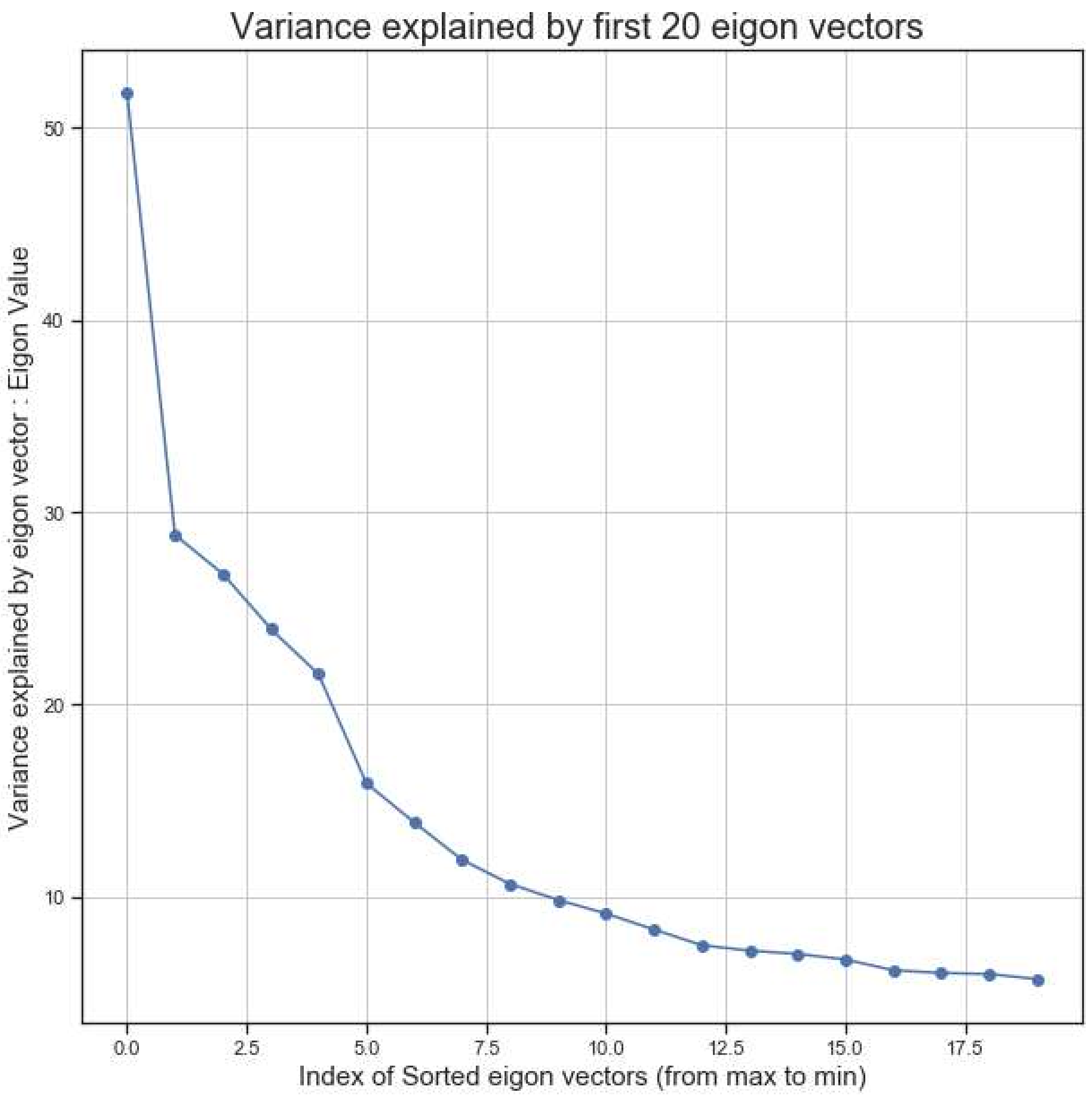


PCA results for first 20 eigon vectors :

⟹ Since its difficult to look at elbows in the scree plot plotted for 784 principal components, lets plot scree plot for first 20 principal components which can be used for the analysis below

In [143]:
```
# lets scale the features
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
DataB_features_scaled = Scaler.fit_transform(DataB_features)
# so we have data of features now
X1 = X
X = DataB_features_scaled
# lets do PCA on X
from sklearn.decomposition import PCA
pca = PCA(n_components= 20,random_state =42)
X_pca = pca.fit_transform(X)
PCA_eigon_vector_data = pca.components_.T
col_names = ["Eigon Vector " + str(i) for i in range(1,21)]
PCA_eigon_vector_dataframe = pd.DataFrame(data = PCA_eigon_vector_data, columns=
eigon_values = pca.explained_variance_
```

In [144]:
```
# lets plot scree plot for 20 components
eigon_value_df = pd.DataFrame(data = eigon_values, columns = ['Eigon Values'])
#plt.plot(eigon_value_df)
fig = plt.figure(figsize =(10,10))
ax = fig.add_subplot(1,1,1)
ax.plot(eigon_value_df, marker ='o')
ax.set_ylabel('Variance explained by eigon vector : Eigon Value', fontsize = 15)
ax.set_xlabel('Index of Sorted eigon vectors (from max to min)', fontsize = 15)
ax.set_title('Variance explained by first 20 eigon vectors ', fontsize = 20)
ax.grid()
```

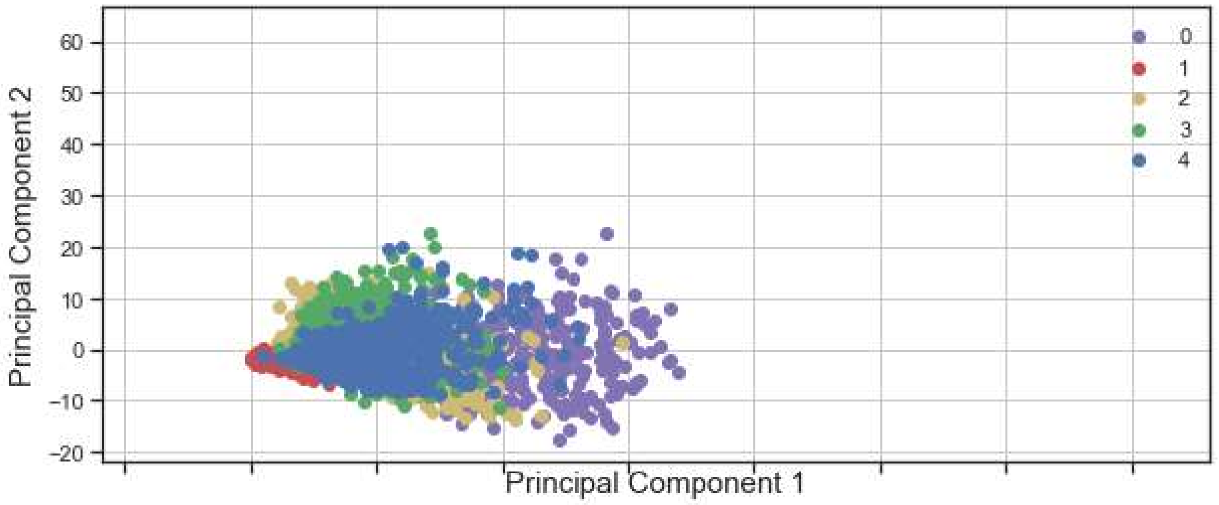


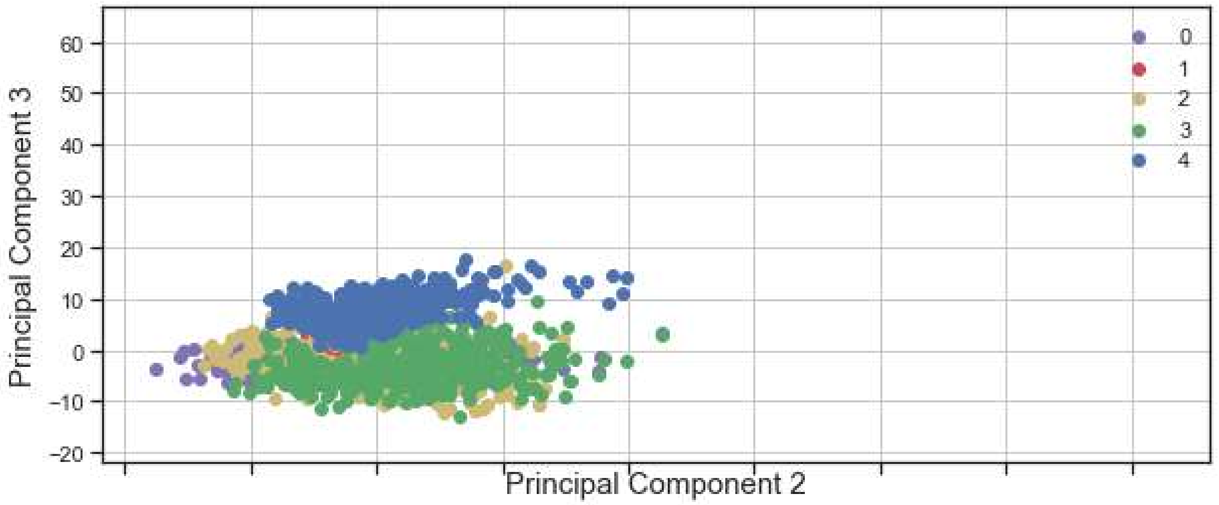
2) Using subplot in python matplotlib, plot the scatter plot of the projected data with the top 20 eigenvalues

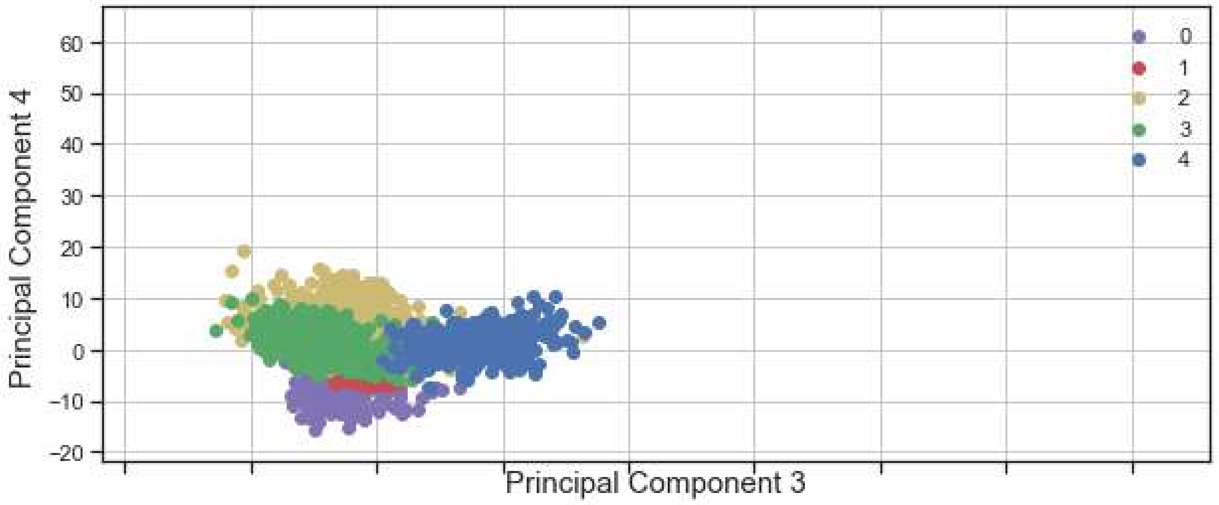
In [145]:
```
col_namess = ["Principal Component " + str(i) for i in range(1,21)]
X_pca_dataframe = pd.DataFrame(data = X_pca, columns= col_namess)
total_PCA_df_class = pd.DataFrame.join(X_pca_dataframe,DataB_class)
total_PCA_df_class.rename_axis('index')
total_PCA_df_class =total_PCA_df_class.assign(new_index = lambda z: z.index)
```

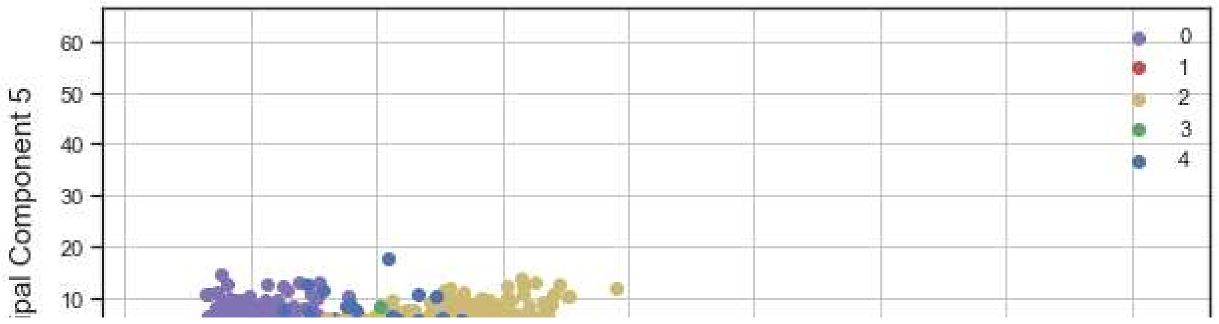
In [146]:
```
fig,axw = plt.subplots(20, sharex=True, sharey=True, figsize =(10,100))
fig.suptitle('Projection on various principal components')
for i in range(0,19):
    axw[i].set_ylabel('Principal Component '+ str(i+2), fontsize = 15)
    axw[i].set_xlabel('Principal Component '+ str(i+1), fontsize = 15)
    class_colors = [0,1,2,3,4]
    colors = ['m','r','y','g','b']
    for class_color, color in zip(class_colors,colors):
        indicesTokeep = total_PCA_df_class['class'] == class_color
        axw[i].scatter(x = total_PCA_df_class.loc[indicesTokeep,'Principal Compo
                      , c = color)
        axw[i].legend(class_colors)
        axw[i].grid()
```

Projection on various principal components

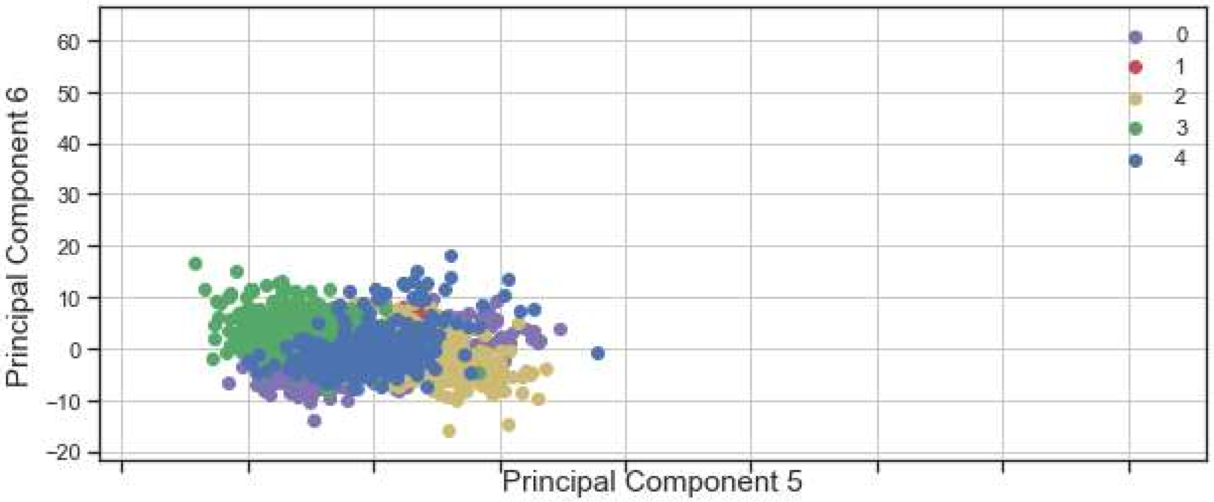

Principal Component 2
Principal Component 1
60
50
40
30
20
10
0
−10
−20
0
1
2
3
4


Principal Component 3
Principal Component 2
60
50
40
30
20
10
0
−10
−20
0
1
2
3
4

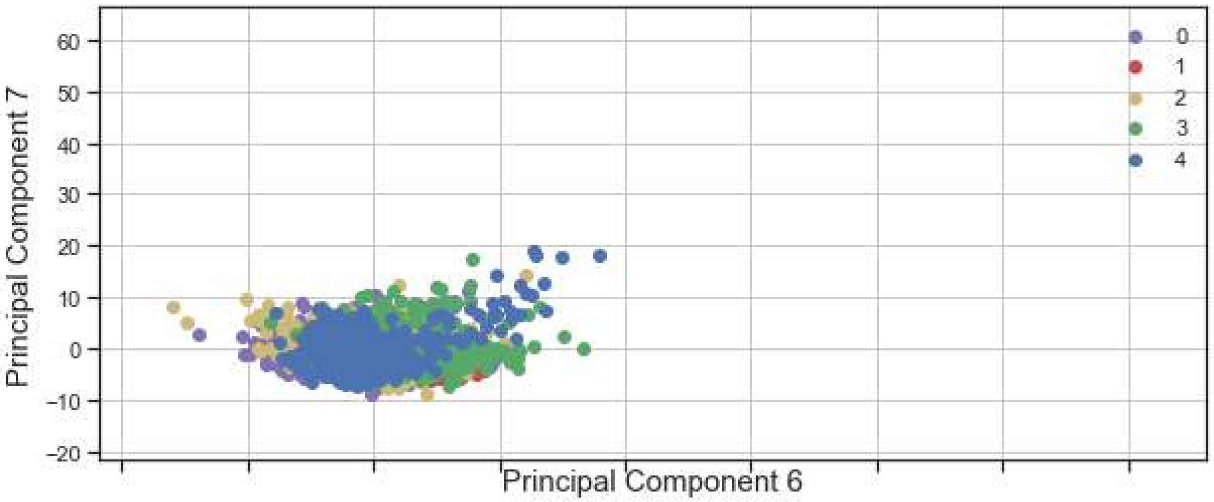

Principal Component 4
Principal Component 3
60
50
40
30
20
10
0
−10
−20
0
1
2
3
4


ipal Component 5
60
50
40
30
20
10
0
1
2
3
4

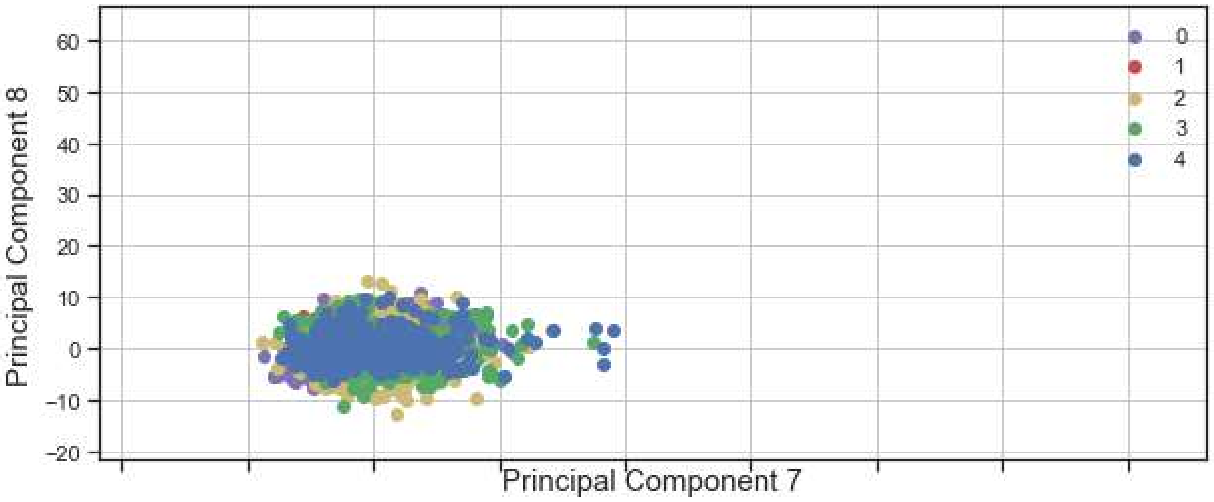
Princi
0
−10
−20
Principal Component 4



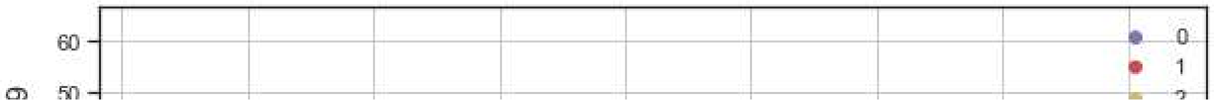
60
50
40
30
20
10
0
−10
−20
Principal Component 6
Principal Component 5
0
1
2
3
4



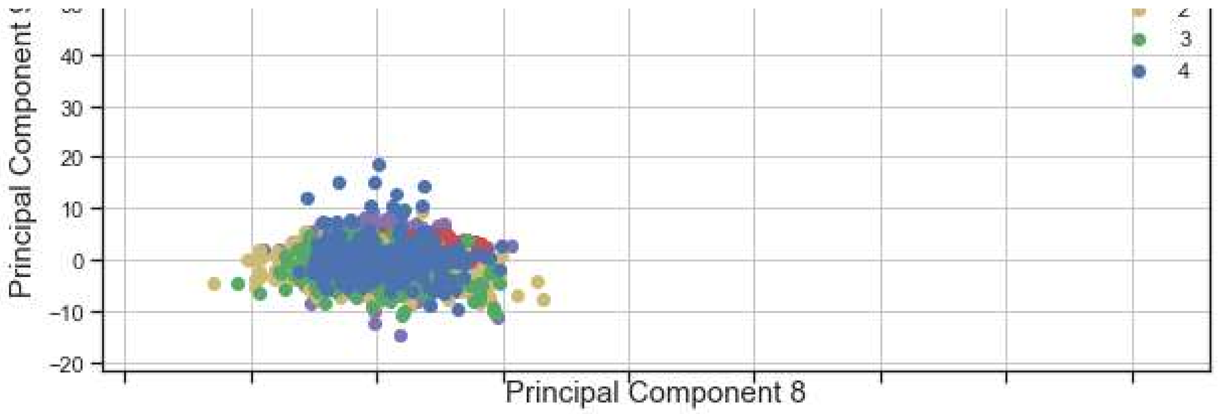
60
50
40
30
20
10
0
−10
−20
Principal Component 7
Principal Component 6
0
1
2
3
4

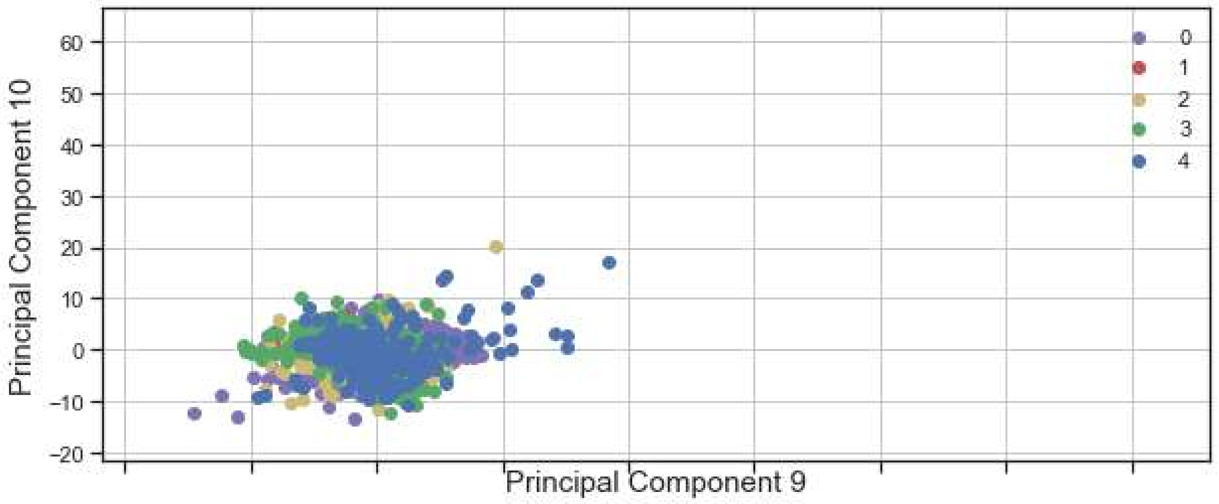


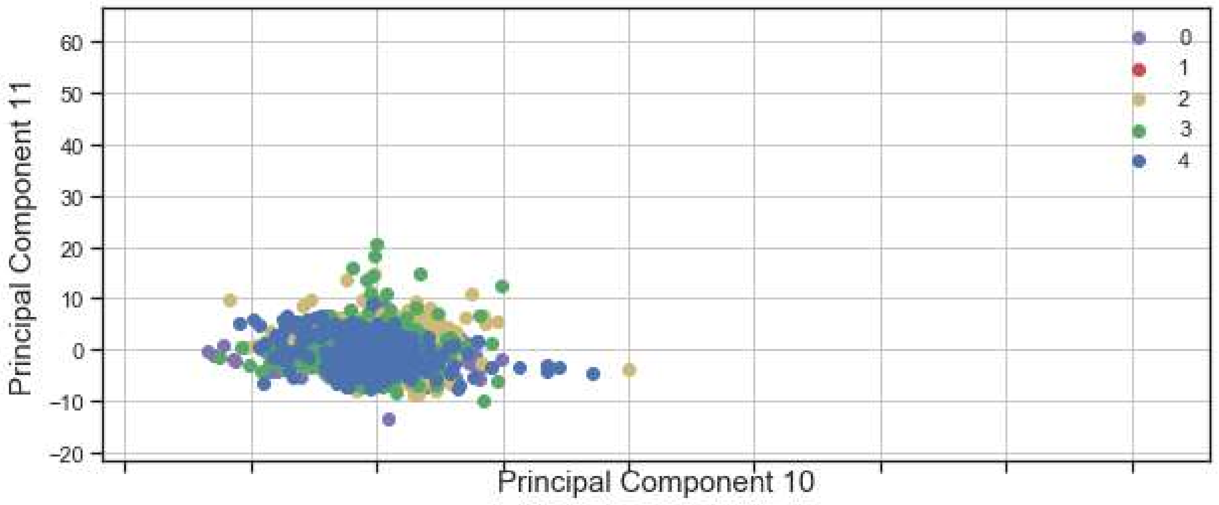
60
50
40
30
20
10
0
−10
−20
Principal Component 8
Principal Component 7
0
1
2
3
4

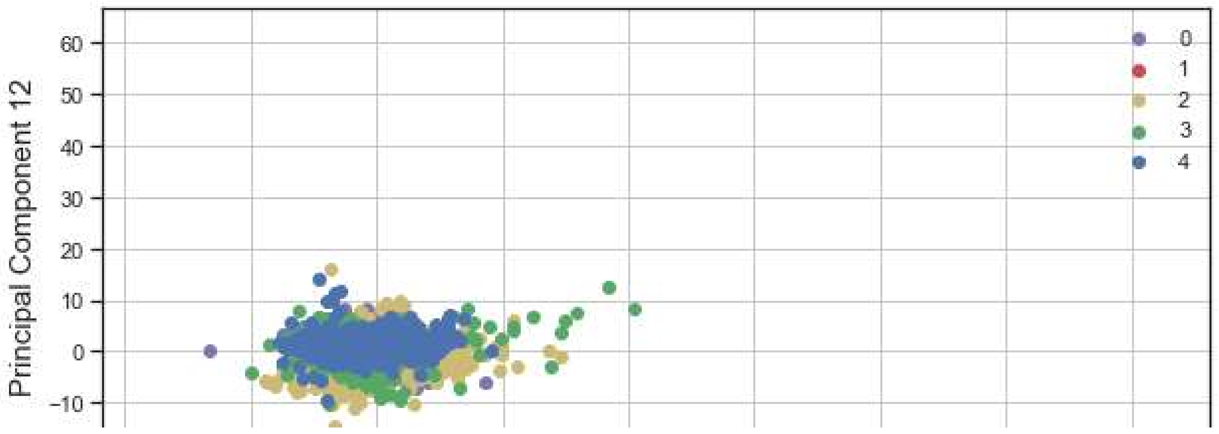


60
50
0
1

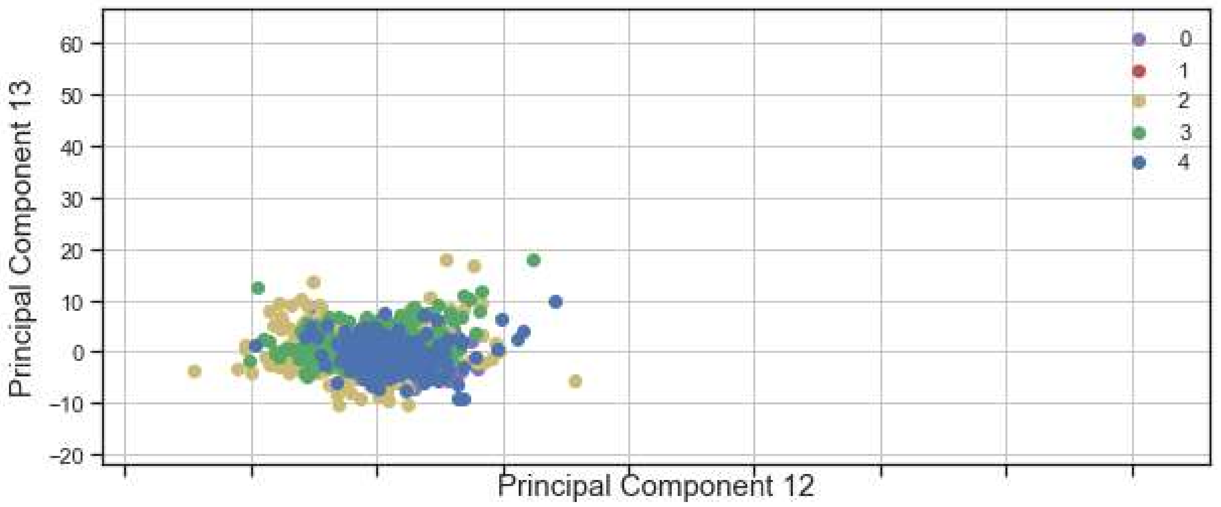

Principal Component 9
Principal Component 8
2
3
4
-20
-10
0
10
20
30
40

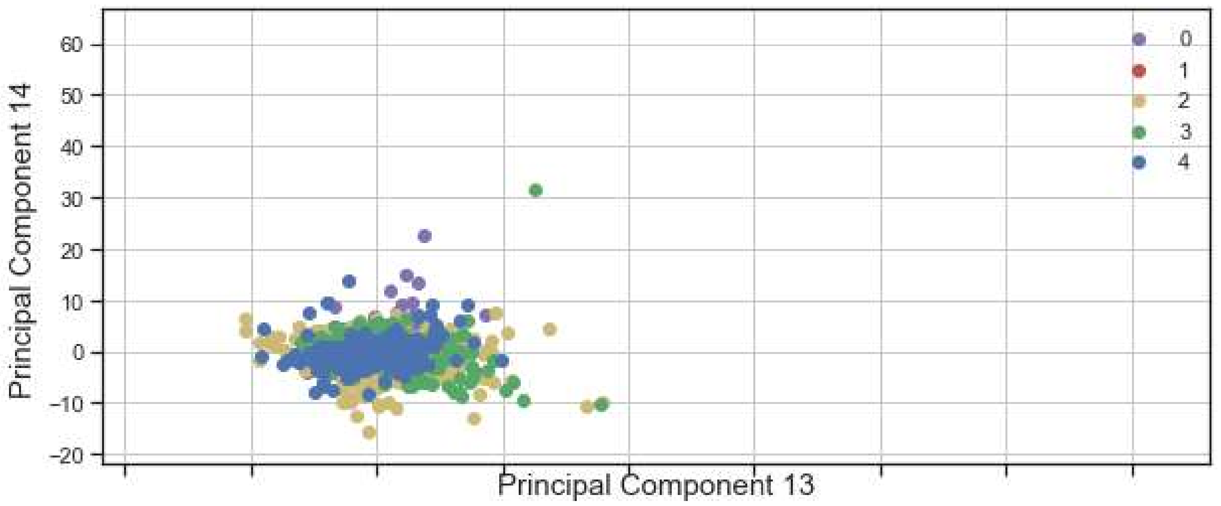

Principal Component 10
Principal Component 9
0
1
2
3
4
-20
-10
0
10
20
30
40
50
60

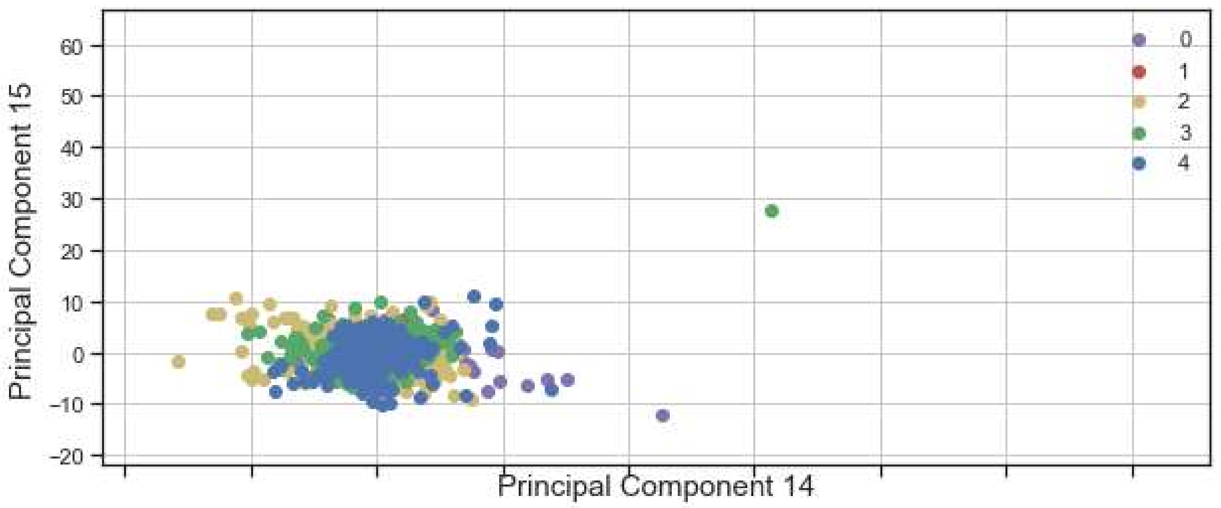

Principal Component 11
Principal Component 10
0
1
2
3
4
-20
-10
0
10
20
30
40
50
60

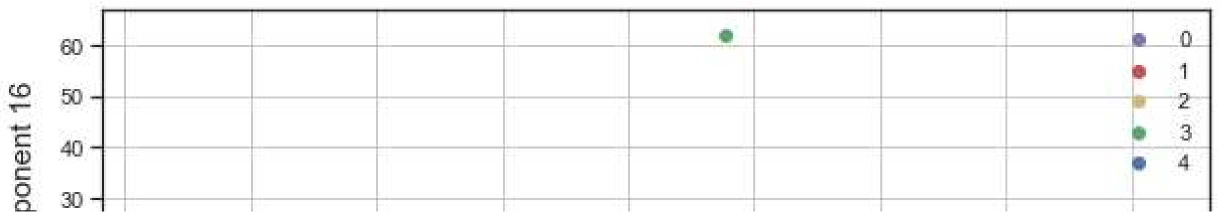

Principal Component 12
0
1
2
3
4
-10
0
10
20
30
40
50
60

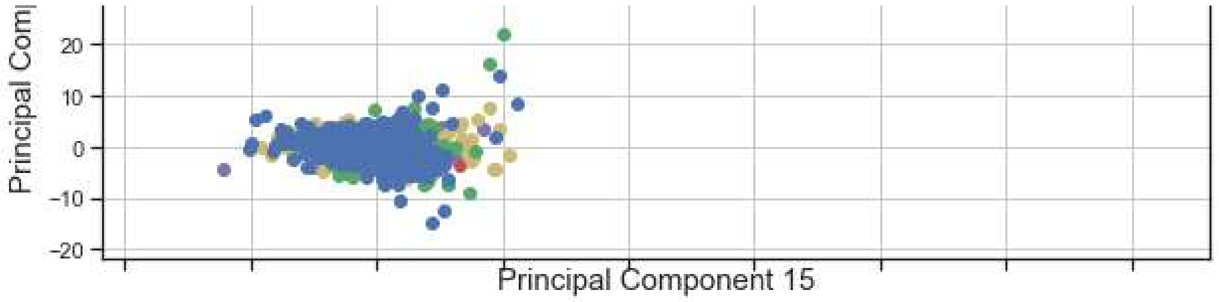
−20
Principal Component 11

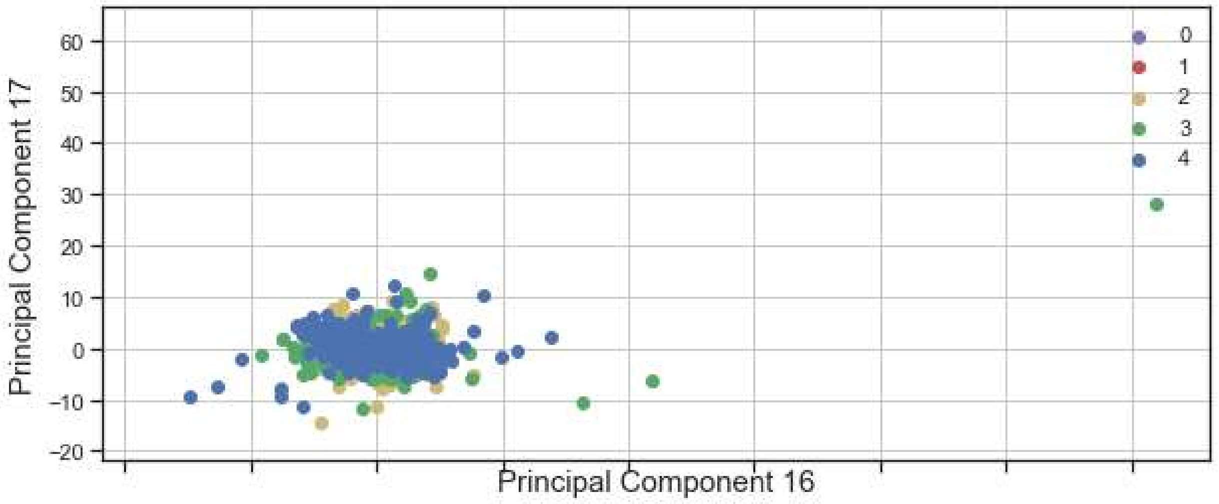

Principal Component 13
60
50
40
30
20
10
0
−10
−20
Principal Component 12
0
1
2
3
4

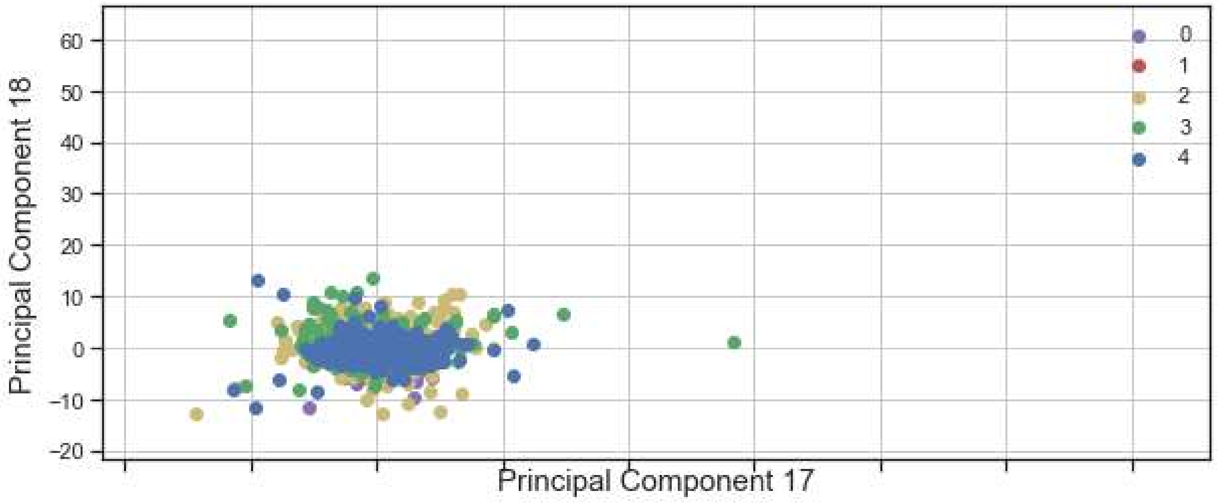

Principal Component 14
60
50
40
30
20
10
0
−10
−20
Principal Component 13
0
1
2
3
4

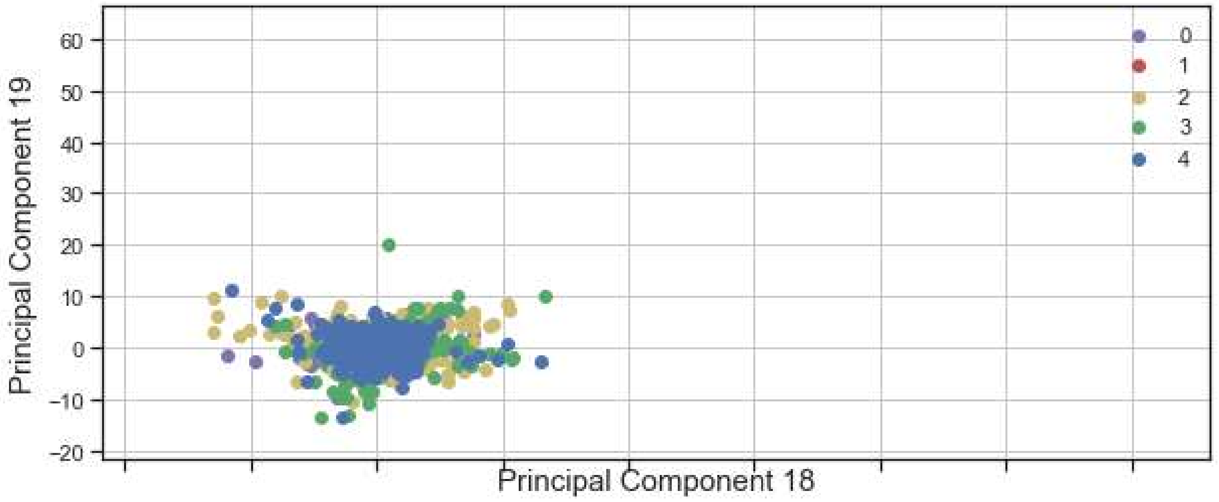

Principal Component 15
60
50
40
30
20
10
0
−10
−20
Principal Component 14
0
1
2
3
4

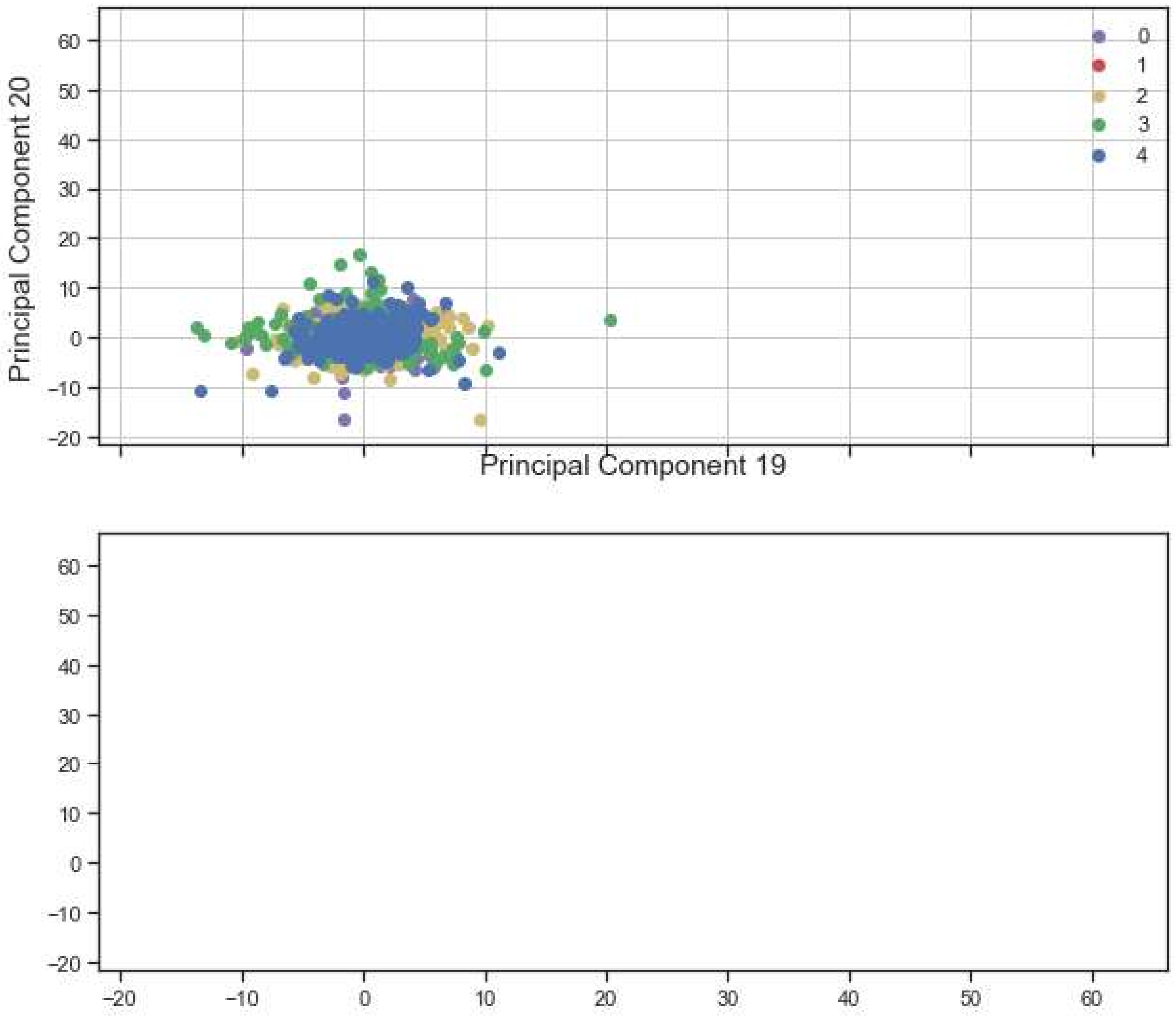

ponent 16
60
50
40
30
0
1
2
3
4


Principal Com
20
10
0
−10
−20
Principal Component 15


Principal Component 17
60
50
40
30
20
10
0
−10
−20
Principal Component 16
0
1
2
3
4


Principal Component 18
60
50
40
30
20
10
0
−10
−20
Principal Component 17
0
1
2
3
4


Principal Component 19
60
50
40
30
20
10
0
−10
−20
Principal Component 18
0
1
2
3
4

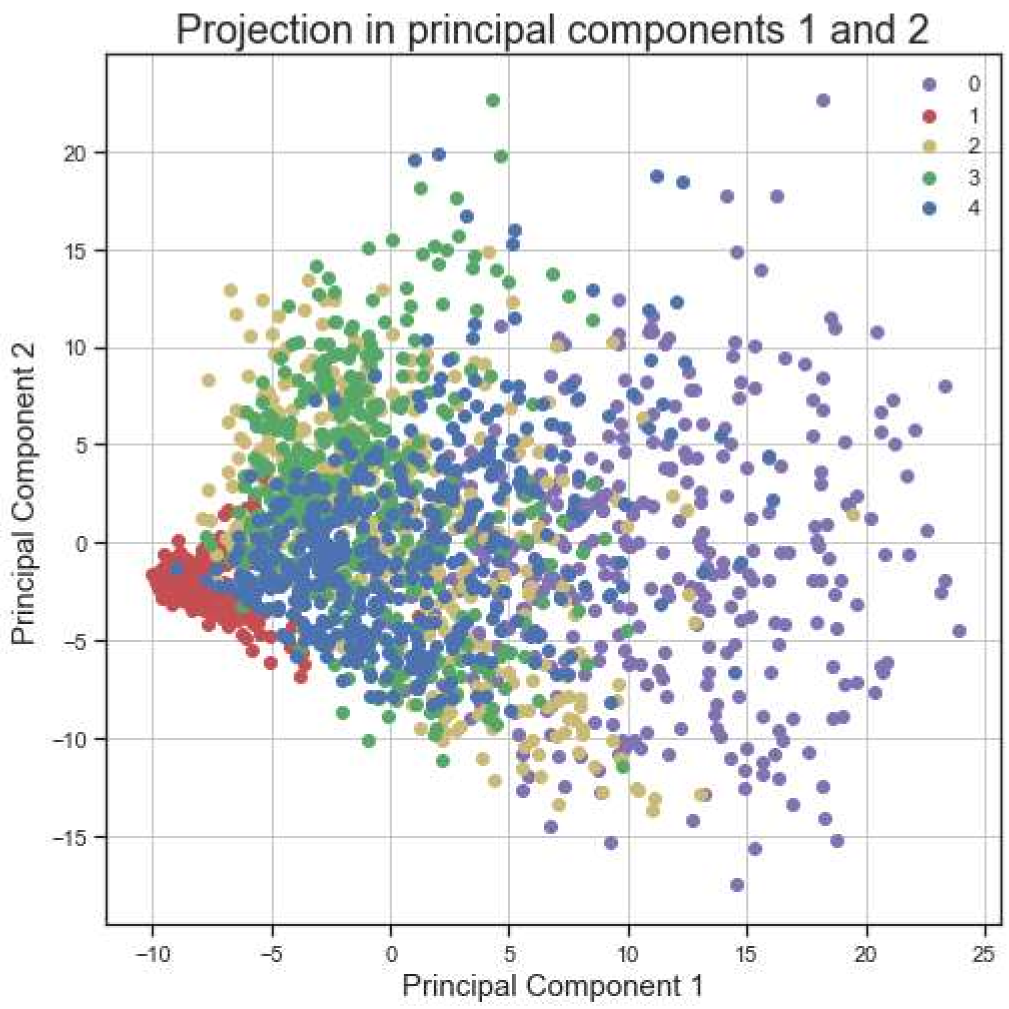


The above plots show the scatter plot of the transformed data on the top 20 principal components corresponding, it can be observed from the range of the variance(range of axis) described by each principal component that variance decreases with an increase in the number of principal components.

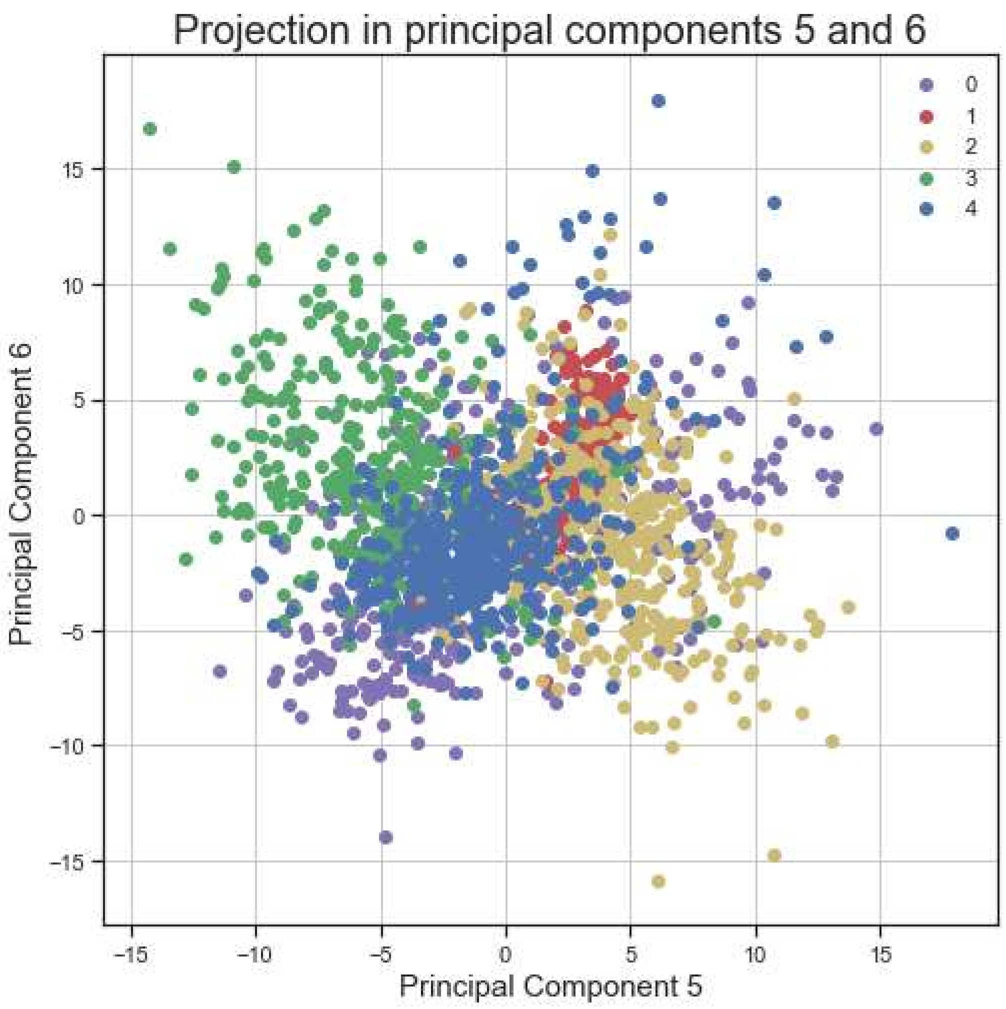
Looking at the range of variance explained by principal components it can be seen, the range of variance explained decreases as the eigenvalues corresponding to these principal components decrease. Also, it can be observed from the 5th plot that variance along principal components 13 and 14 drops drastically, this effect can also be seen from scree plotted for the first 20 eigenvectors.

3) Plot two 2-dimensional representations of the data points based on the first vs second principal components and 5th vs 6th displaying the data points of each class with a different color

In [147]:
```
figure = plt.figure(figsize = (8,8))
ax = figure.add_subplot(1,1,1)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_title('Projection in principal components 1 and 2', fontsize = 20)
class_colors = [0,1,2,3,4]
colors = ['m','r','y','g','b']
for class_color, color in zip(class_colors,colors):
    indicesTokeep = total_PCA_df_class['class'] == class_color
    ax.scatter(x = total_PCA_df_class.loc[indicesTokeep,'Principal Component 1']
             , c = color)
ax.legend(class_colors)
ax.grid()
```



In [148]:
```
figure = plt.figure(figsize = (8,8))
ax = figure.add_subplot(1,1,1)
ax.set_ylabel('Principal Component 6', fontsize = 15)
ax.set_xlabel('Principal Component 5', fontsize = 15)
ax.set_title('Projection in principal components 5 and 6', fontsize = 20)
class_colors = [0,1,2,3,4]
colors = ['m','r','y','g','b']
for class_color, color in zip(class_colors,colors):
    indicesTokeep = total_PCA_df_class['class'] == class_color
    ax.scatter(x = total_PCA_df_class.loc[indicesTokeep,'Principal Component 5']
              , c = color)
ax.legend(class_colors)
ax.grid()
```



1. From the above plot of the data on first and second principal components and fifth and sixth principal components, it can be visualized from the range of the different principal components that the variance described by the first two principal components is greater than the fifth and sixth components.
2. Classes of the MNIST digit dataset have maximum variance in the first principal components describing 6.6% of the total variance. While the second principal component explains 3.67% of

the total variance.

3. The explained variance corresponding principal component(individual) having low eigon-values, decrease as can be understood from the plot.
4. Also, the classes look more separated when projected on principal components 1 and 2 compared to the projection on principal components 5 and 6.

4) Implement (A) PCA and (B) dual PCA with singular value decomposition.

Here PCA will be implemented manually, no PCA or SVD library would be used

implementing PCA using svd

In [149]:
```
DataB_features = DataB_features_scaled
DataB_features_np = DataB_features.T
mean_array = np.zeros((784,1))
for i in range(0,784):
    mean_array[i,0] = np.mean(DataB_features_np[i,:])
scatter_matrix = np.zeros((784,784))
DataB_features_np_normalized = DataB_features_np - mean_array
DataB_features_np_normalized_T = DataB_features_np_normalized.transpose()
scatter_matrix = np.matmul(DataB_features_np_normalized,DataB_features_np_normal
np.shape(scatter_matrix)
```

Out[149]:
```
(784, 784)
```

In [150]:
```
eigon_value, eigon_vector = np.linalg.eig(scatter_matrix)
```

In [151]:
```
np.shape(eigon_value)
```

Out[151]:
```
(784,)
```

In [152]:
```
np.shape(eigon_vector)
```

Out[152]:
```
(784, 784)
```

In [153]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eigon_pairs = [(np.abs(eigon_value[i]), eigon_vector[:,i]) for i in range(len(ei

# Sort the (eigenvalue, eigenvector) tuples from high to low
eigon_pairs.sort(key=lambda x: x[0], reverse=True)
```

In [154]:
```
u_matrix = np.zeros((784,20))
for i in range(0,20):
    u_matrix[:,i]= eigon_pairs[i][1]
np.shape(u_matrix)
```

Out[154]:
```
(784, 20)
```

```
In [155]: u_matrix_transpose = u_matrix.transpose()
          np.shape(u_matrix_transpose)

Out[155]: (20, 784)
```

```
In [156]: # lets project data
          pca_transformed_data = np.matmul(u_matrix_transpose,DataB_features_np).transpose
          np.shape(pca_transformed_data)
          #pca_transformed_data

Out[156]: (2066, 20)
```

```
In [157]: #lets see how scree plot looks like here
          eigon_vector_list = []
          for i in range(0,20):
              eigon_vector_list.append(eigon_pairs[i][0])
          eigon_value_manual_df = pd.DataFrame(data = eigon_vector_list, columns= ['Eigon \
```

Implementing dual PCA using svd

```
In [158]: DataB_features_np_normalized = DataB_features_np - mean_array
          DataB_features_np_normalized_T = DataB_features_np_normalized.transpose()
          A_transpose_A = np.matmul(DataB_features_np_normalized_T, DataB_features_np_norma
          np.shape(A_transpose_A)

Out[158]: (2066, 2066)
```

```
In [159]: A_T_A_eigon_value, A_T_A_eigon_vector = np.linalg.eigh(A_transpose_A)
```

```
In [160]: # Make a list of (eigenvalue, eigenvector) tuples
          A_T_A_eigon_pairs = [(np.abs(A_T_A_eigon_value[i]), A_T_A_eigon_vector[:,i]) for
          # Sort the (eigenvalue, eigenvector) tuples from high to low
          A_T_A_eigon_pairs.sort(key=lambda x: x[0], reverse=True)
```

```
In [161]: S_matrix = np.zeros((20,2066))
          for i in range(0,20):
              S_matrix[i][i] = np.sqrt(A_T_A_eigon_pairs[i][0])
          V_matrix = np.zeros((2066,2066))
          for i in range(0,2066):
              V_matrix[:,i]= A_T_A_eigon_pairs[i][1]
          np.shape(V_matrix)

Out[161]: (2066, 2066)
```

```
In [162]: #calculating SV as Utranspose X
          dualpca_transformed_data = (np.matmul(S_matrix,V_matrix.transpose())).transpose(
          np.shape(dualpca_transformed_data)

Out[162]: (2066, 20)
```

lets compare time across both PCA and Dual PCA

In [163]:
```
#PCA
import time
start = time.time()
mean_array = np.zeros((784,1))
for i in range(0,784):
    mean_array[i,0] = np.mean(DataB_features_np[i,:])
scatter_matrix = np.zeros((784,784))
DataB_features_np_normalized = DataB_features_np - mean_array
DataB_features_np_normalized_T = DataB_features_np_normalized.transpose()
scatter_matrix = np.matmul(DataB_features_np_normalized,DataB_features_np_normal
eigon_value, eigon_vector = np.linalg.eig(scatter_matrix)
# Make a list of (eigenvalue, eigenvector) tuples
eigon_pairs = [(np.abs(eigon_value[i]), eigon_vector[:,i]) for i in range(len(ei
# Sort the (eigenvalue, eigenvector) tuples from high to low
eigon_pairs.sort(key=lambda x: x[0], reverse=True)
u_matrix = np.zeros((784,20))
for i in range(0,20):
    u_matrix[:,i]= eigon_pairs[i][1]
u_matrix_transpose = u_matrix.transpose()
pca_transformed_data = np.matmul(u_matrix_transpose,DataB_features_np).transpose
end = time.time()

print("Time taken for implementing PCA is ", end - start)

import time
start_dual = time.time()
mean_array = np.zeros((784,1))
DataB_features_np_normalized = DataB_features_np - mean_array
DataB_features_np_normalized_T = DataB_features_np_normalized.transpose()
A_transpose_A = np.matmul(DataB_features_np_normalized_T, DataB_features_np_norm
A_T_A_eigon_value, A_T_A_eigon_vector = np.linalg.eigh(A_transpose_A)
# Make a list of (eigenvalue, eigenvector) tuples
A_T_A_eigon_pairs = [(np.abs(A_T_A_eigon_value[i]), A_T_A_eigon_vector[:,i]) for
# Sort the (eigenvalue, eigenvector) tuples from high to low
A_T_A_eigon_pairs.sort(key=lambda x: x[0], reverse=True)
S_matrix = np.zeros((20,2066))
for i in range(0,20):
    S_matrix[i][i] = np.sqrt(A_T_A_eigon_pairs[i][0])
V_matrix = np.zeros((2066,2066))
for i in range(0,2066):
    V_matrix[:,i]= A_T_A_eigon_pairs[i][1]
dualpca_transformed_data = (np.matmul(S_matrix,V_matrix.transpose())).transpose(

end_dual = time.time()
print("Time taken for implementing dual PCA is ", end_dual - start_dual)
```

```
Time taken for implementing PCA is  0.3450784683227539
Time taken for implementing dual PCA is  1.0068280696868896
```

In [164]:
```
# lets see results of projected data for PCA
print("Results of PCA transformed data is :",pca_transformed_data)
```

```
Results of PCA transformed data is : [[ -9.97069222   6.18172201  -4.99286326
...  -0.26257488   1.42584762
   -1.16252257]
 [-11.41599978   6.94158705  -5.06302886 ...   0.96317397   1.11655238
    0.06708945]
 [ -3.69011918   4.69309729  -2.9086564  ...   2.65907012  -0.66109634
   -5.12371489]
 ...
 [  0.34942153   0.93368106   8.10744188 ...  -1.28086781   1.19700404
    1.08146006]
 [  3.11526327   2.09047425   6.27251911 ...  -1.30774666  -0.11716451
    1.59384718]
 [  5.64409375  -0.24616663   4.14018317 ...   2.8474039   -1.14882287
   -3.39490069]]
```

In [165]:
```
# lets see results of projeted data for dual PCA
print("Results of dual PCA transformed data is :",dualpca_transformed_data)
```

```
Results of dual PCA transformed data is : [[ 9.97069222 -6.18172201  4.99286326
...  0.26257488 -1.42584762
   1.16252257]
 [11.41599978 -6.94158705  5.06302886 ... -0.96317397 -1.11655238
  -0.06708945]
 [ 3.69011918 -4.69309729  2.9086564  ... -2.65907012  0.66109634
   5.12371489]
 ...
 [-0.34942153 -0.93368106 -8.10744188 ...  1.28086781 -1.19700404
  -1.08146006]
 [-3.11526327 -2.09047425 -6.27251911 ...  1.30774666  0.11716451
  -1.59384718]
 [-5.64409375  0.24616663 -4.14018317 ... -2.8474039   1.14882287
   3.39490069]]
```

For PCA, we decompose the matrix $X^T * X$ which has $d * d$ dimension where $d = 784$ for MNIST dataset whereas for Dual PCA, we decompose the matrix $X * X^T$ which has $n * n$ dimensions where $n = 2066$ for MNIST dataset. Decomposition of matrix with dimension $2066 * 2066$ takes longer time than decomposition of matrix with $784 * 784$ dimensions. That is why Dual PCA takes more time than PCA.

This is further being supported by the fact that the execution time of PCA is 0.35 seconds while dual PCA takes around 1.01 seconds.

2.2.2 Theoritical Questions

Prove that PCA is the best linear method for reconstruction (with orthonormal bases).

$\hat{X}$ is data point in original space and $UU^T\hat{X}$ is reconstruction of projected data on the principal components.

In order to reduce the recunstruction erroe we need to form an optimization problem and minimize it. The optimization problem is described as below:

$$\begin{array}{ll} \underset{U}{\text{minimize}} & ||\hat{X} - UU^T\hat{X}||_F^2 \\ \text{subject to} & U^TU = I. \end{array}$$

$$||\hat{X} - UU^T\hat{X}||_F^2$$

$$= tr((\hat{X} - UU^T\hat{X})^T(\hat{X} - UU^T\hat{X}))$$

$$= tr((\hat{X}^T - \hat{X}^TUU^T)(\hat{X} - UU^T\hat{X}))$$

$$= tr(\hat{X}^T\hat{X} - 2\hat{X}^TUU^T\hat{X} + \hat{X}^TU\underbrace{U^TU}_{\text{I}}U^T\hat{X}) = tr(\hat{X}^T\hat{X} - \hat{X}^TUU^T\hat{X})$$

$$= tr(\hat{X}^T\hat{X}) - tr(\hat{X}^TUU^T\hat{X})$$

$$= tr(\hat{X}^T\hat{X}) - tr(\hat{X}\hat{X}^TUU^T)$$

Using Lagrange multiplier, we have:
$\mathcal{L} = tr(\hat{X}^T\hat{X}) - tr(\hat{X}\hat{X}^TUU^T) - tr(\Lambda^T(U^TU - I)),$

where $\Lambda \in \mathbb{R}^{p\times p}$ is a diagonal matrix $diag([\lambda_1, \ldots, \lambda_p]^T)$ containing the Lagrange multipliers. Equating the derivative of Lagrangian to zero gives:

$$\mathbb{R}^{d\times p} \ni \frac{\partial\mathcal{L}}{\partial U} = 2\hat{X}\hat{X}^TU - 2U\Lambda = 0$$

$$\implies \hat{X}\hat{X}^TU = U\Lambda, \implies SU = U\Lambda$$

This is the eigen value problem for the covariance matrix S. We had same eigen value problem in PCA.

PCA subspace is the best linear projection in terms of reconstruction error as reconstruction error is minimized when maximum variance are captured along the data points. In other words, PCA has the least squared error in reconstruction.

2.3 Fisher Discriminant Analysis (FDA)

2.3.1 Practical Question

1) Applying LDA to reduce Dimensionality

In [166]:
```
#X_full_scaled_c
#yc_new
#lda = LinearDiscriminantAnalysis(n_components = 2)
#ldaComponents = lda.fit_transform(X_full_scaled_c, yc_new)
col_name_features = ["Feature " + str(i) for i in range(1,785)]

DataB_features_scaled_df = pd.DataFrame(data = DataB_features_scaled, columns = c
X_lda_df = DataB_features_scaled_df
y_lda_df = DataB_class
y_lda_np = pd.DataFrame.to_numpy(y_lda_df)
```

In [167]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components= 4)
ldaComponents =lda.fit_transform(X_lda_df, np.ravel(y_lda_np))
np.shape(ldaComponents)
```

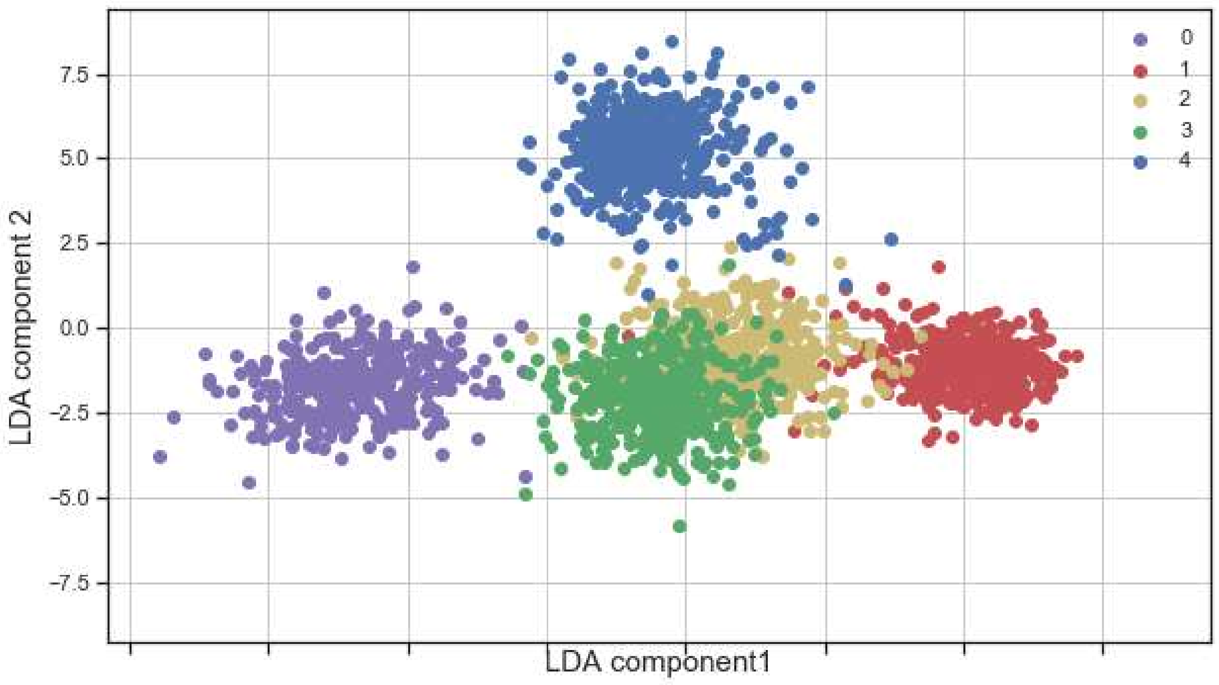
Out[167]:
```
(2066, 4)
```

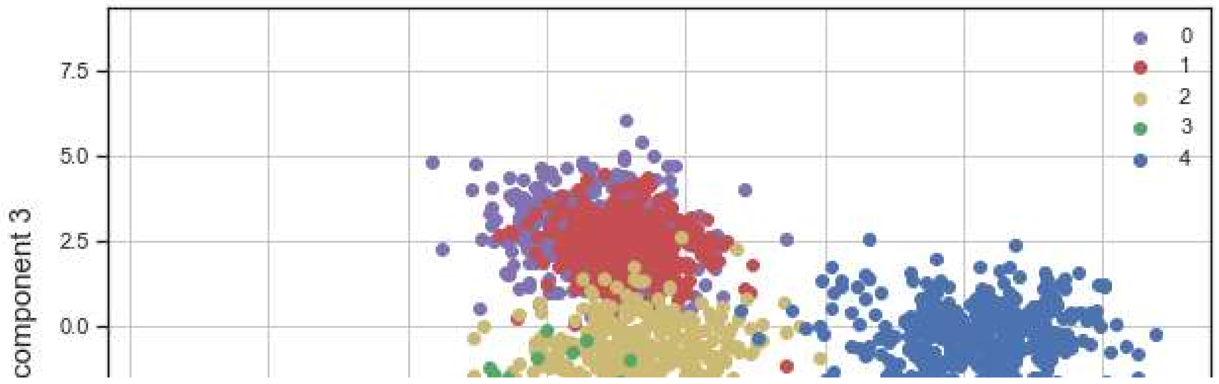
In [168]:
```
lda_1d_c = pd.DataFrame(data = ldaComponents, columns= ['LDA component 1','LDA co
lda_1d_final_c = pd.DataFrame.join(lda_1d_c, y_lda_df)
lda_1d_final_c.rename_axis('index')
lda_1d_final_c =lda_1d_final_c.assign(new_index = lambda z: z.index)
lda_1d_final_c.shape
```

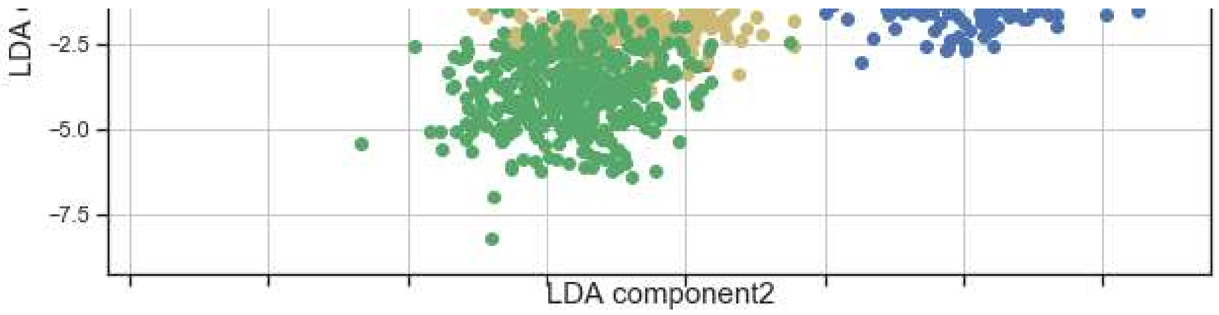
Out[168]:
```
(2066, 6)
```

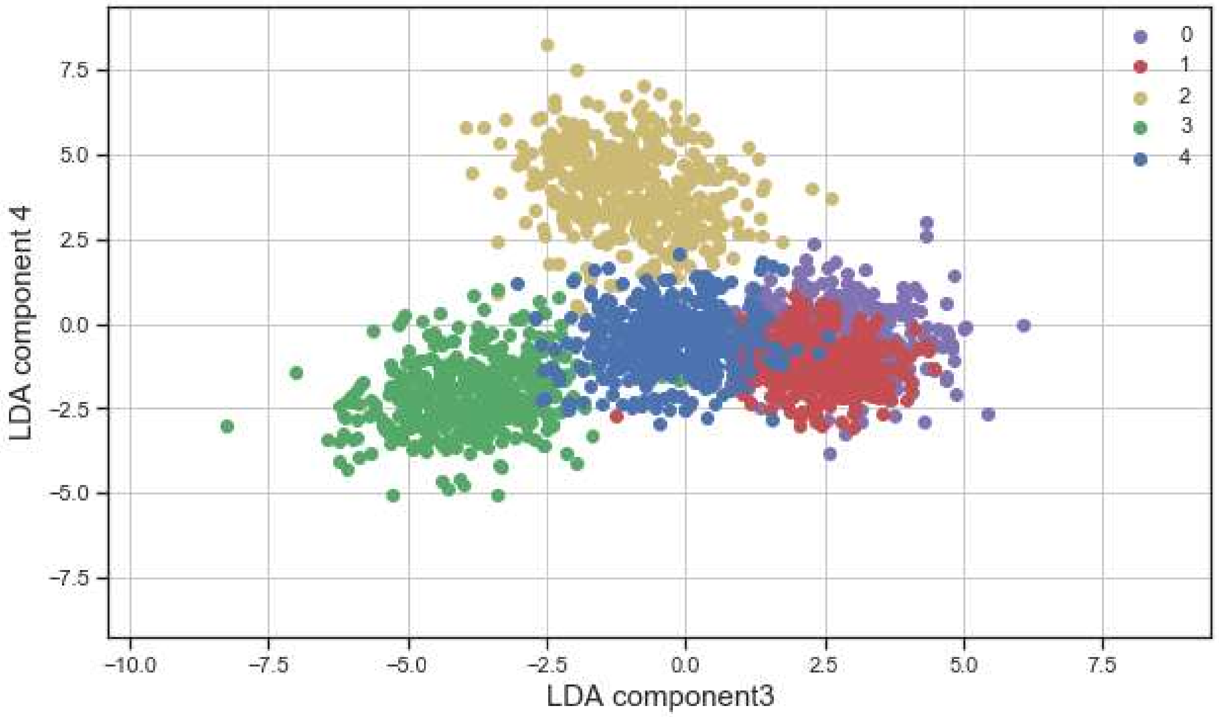
```
In [169]: fig,axs = plt.subplots(3, sharex=True, sharey=True, figsize =(10,20))
          fig.suptitle('Sharing both axes')
          for i in range(0,3):
              axs[int(i)].set_ylabel('LDA component '+ str(i+2), fontsize = 15)
              axs[int(i)].set_xlabel('LDA component' + str(i+1), fontsize = 15)
              class_colors = [0,1,2,3,4]
              colors = ['m','r','y','g','b']
              for class_color, color in zip(class_colors,colors):
                  indicesTokeep = lda_1d_final_c['class'] == class_color
                  axs[int(i)].scatter(x = lda_1d_final_c.loc[indicesTokeep,'LDA component
                            , c = color)
              axs[int(i)].legend(class_colors)
              axs[int(i)].grid()
```

Sharing both axes









It can be visualized from the above plots that different classes can be distinct in different directions. Following are the different directions responsible for different classes;
LDA component 1: This direction is responsible for the separation of the digit 0 and digits 1 as it can be observed from cluster spread along the x-axis of the first graph.
LDA component 2: if projected data on direction 2 than except digit 4 all the other classes are collapsed near to each other. So, the only digit 2 can be classified along direction 2.
LDA component 3: Direction 3 separates the MNIST digit 3 when projected on it
LDA component 4: when data are projected along direction 4 then digit 2 class seems to have separability from the rest of the class.

2) Compare results of LDA with results obatined using PCA

It can be visualized from the above plots, Projection on principal component 1 and principal component 2 provide good visualization of different classes, but LDA outperforms PCA when it comes to comprehending the separability of classes in lower dimensions. Comparing these results of PCA and LDA it can be said that LDA tries to attain maximum separability of classes across different directions while principal components in PCA are in the direction of the maximum variance.

2.3.2 Theoricitical Question

We can consider the total scatter as the summation of the within and between scatters: $\mathbf{S_T} = \mathbf{S_W} + \mathbf{S_B} \implies \mathbf{S_B} = \mathbf{S_T} - \mathbf{S_W}$. By substituting this into the Fisher criterion, the FDA optimization can be slightly modified to:

Here d is the dimensions of datapoints and p is the dimension of projection space The optimization equation is equivalent to:

$$\begin{array}{ll} \underset{U}{\text{maximize}} & tr(U^T S_T U) \\ \text{subject to} & U^T S_W U = I. \end{array}$$

Using Lagrange multiplier, we have:

$$\mathcal{L} = tr(U^T S_T U) - tr(\Lambda^T (U^T S_W U - I))$$

where $\Lambda \in \mathbb{R}^{d \times d}$ is a diagonal entries are the Lagrange multipliers. Equating the derivative of $\mathcal{L}$ to zero gives:

$$\mathbb{R}^{d \times p} \ni \frac{\partial L}{\partial U} = 2S_T U - 2S_W U \Lambda = 0$$

$$\implies 2S_T U = 2S_W U \Lambda$$

$$\implies S_T U = S_W U \Lambda$$

$$\implies S_W^{-1} S_T U = U \Lambda$$

Which is a generalized eigenvalue problem (S_T, S_W). The columns of U are the eigenvectors sorted by largest to smallest eigenvalues (because the optimization is maximization) and the diagonal entries of Λ are the corresponding eigenvalues. The columns of U are referred to as the Fisher directions or Fisher axes.

The FDA directions can be obtained by the generalized eigenvalue problem ($\mathbf{S_T}$,$\mathbf{S_W}$). By comparing the equations, it shows that PCA captures the orthonormal directions with the maximum variance of data. However, the FDA has the same goal but also it requires the manipulated directions to be orthonormal. This manipulation is done by the within scatter which makes sense because the within scatters make use of the class labels. This comparison gives a hint for the connection between PCA and FDA. From question 2 in a practical question, it is clearly seen that PCA intermingles the classes. There is not a cut point for the dimensions. LDA gives good clear cut dimensions since it considers labels in the data. Suppose there are two different clusters with opposite labels, but still they are placed very near to each other. Most of the data variation in the direction of these clusters. These clusters would be projected onto the direction of the greatest variety of data and it results in the formation of a single cluster of data. So PCA mixes up the clusters without considering the labels. FDA projects the data onto a direction that is orthogonal to the direction of the greatest variation of the data. This direction is in the least variation of the data. These two clusters would then be nearly perfectly separated from each other because of taking into account of their labels.

References :

1) B. Ghojogh, M. N. Samad, S. A. Mashhadi, T. Kapoor, W. Ali, F. Karray and M. Crowley, "Feature Selection and Feature Extraction in Pattern Analysis: A Literature Review", arXiv:1905.02845v1, 7 May 2019

2) B. Ghojogh,M. Crowley, "Unsupervised and Supervised Principal Component Analysis: Tutorial", arXiv:1906.03148v1, 1 Jun 2019

3) B. Ghojogh, F. Karray and M. Crowley, "Fisher and Kernel Fisher Discriminant Analysis: Tutorial", arXiv:1906.09436v1, 22 Jun 2019

Q 3 Nonlinear Dimensionality Reduction

3.1 Dataset

In [170]:
```
DataB_features =X1
import time
col_name_features = ["Feature "+str(i) for i in range(1,785)]
DataB_features_scaled_df = pd.DataFrame(data = DataB_features, columns = col_nam
X_KERNEL_PCA_df = DataB_features
y_KERNEL_PCA_df = DataB_class
```

3.2 Practical Questions

3.2.1 Different Embedding Marks

1) Kernel PCA