g26

February 16, 2024

1 ECE 657A: Data and Knowledge Modeling and Analysis

Assignment 1: Data Cleaning and Dimensionality ReducEon

Submitted By Group 26:

1. Neeraj Nagar (ID: 21107021)
2. Seturaj Matroja (ID: 21064444)
3. Akshat Baheti (ID: 21100660)

[168]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from scipy import stats
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

2 Question I. Data Cleaning and Preprocessing (for dataset A)

1. Detect any problems that need to be fixed in dataset A. Report such problems.

Based on the analysis of dataset A, the following problems were identified:

i. First Column unnamed is not required.

- The first column does not have a meaningfull name and seems redundant as pandas automatically provides an index.
- This column could be removed from the dataset

ii. Feature 34, 35, and 36 columns have only 1 non-null value.

- This suggests that these columns may not contain sufficient information to be meaningful for analysis.
- These columns could be candidates for removal from the dataset

iii. The last 773 rows of the data set values are missing 18,227 to 18,999.

- It seems there is a significant chunk of missing data in the last portion of the dataset.

iv. Missing values or Null values present in the data set.

- Identify the presence of missing or null values in the dataset

v. Outliers present in the data.

- Perform outlier detection to identify extreme values in the dataset using box plots that might adversely affect analysis or modeling.

[169]:
```
#Load the dataset
DataA = pd.read_csv('DataA.csv',encoding='latin-1')
DataA.head()
```

[169]:
```
   Unnamed: 0  fea.1  fea.2  fea.3  fea.4   fea.5  fea.6  fea.7  fea.8  fea.9  \
0           1 -153.0  414.0  939.0 -161.0  1007.0   99.0 -210.0  948.0  333.0
1           2 -150.0  420.0  939.0 -177.0  1008.0  103.0 -207.0  939.0  316.0
2           3 -160.0  432.0  941.0 -162.0   982.0   98.0 -198.0  936.0  315.0
3           4 -171.0  432.0  911.0 -174.0   999.0  115.0 -187.0  918.0  338.0
4           5 -171.0    NaN  929.0 -189.0  1004.0  104.0 -198.0  939.0  350.0

   …  fea.72  fea.73  fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  \
0  …   655.0  -316.0  -302.0  -617.0  -955.0  -264.0    23.0   -29.0
1  …   655.0  -309.0  -304.0  -619.0  -955.0  -265.0    19.0   -31.0
2  …   655.0  -302.0  -308.0  -621.0  -966.0  -270.0    10.0   -38.0
3  …   655.0  -293.0  -312.0  -622.0  -964.0  -269.0    14.0   -51.0
4  …   655.0  -284.0  -318.0  -624.0  -966.0  -262.0    24.0   -40.0

   fea.80  fea.81
0    36.0    24.0
1    47.0     3.0
2    20.0     0.0
3    33.0    -1.0
4     1.0     4.0

[5 rows x 82 columns]
```

[170]:
```
#Print the columns
print(DataA.columns)
```

```
Index(['Unnamed: 0', 'fea.1', 'fea.2', 'fea.3', 'fea.4', 'fea.5', 'fea.6',
       'fea.7', 'fea.8', 'fea.9', 'fea.10', 'fea.11', 'fea.12', 'fea.13',
       'fea.14', 'fea.15', 'fea.16', 'fea.17', 'fea.18', 'fea.19', 'fea.20',
       'fea.21', 'fea.22', 'fea.23', 'fea.24', 'fea.25', 'fea.26', 'fea.27',
       'fea.28', 'fea.29', 'fea.30', 'fea.31', 'fea.32', 'fea.33', 'fea.34',
       'fea.35', 'fea.36', 'fea.37', 'fea.38', 'fea.39', 'fea.40', 'fea.41',
       'fea.42', 'fea.43', 'fea.44', 'fea.45', 'fea.46', 'fea.47', 'fea.48',
```

```
       'fea.49', 'fea.50', 'fea.51', 'fea.52', 'fea.53', 'fea.54', 'fea.55',
       'fea.56', 'fea.57', 'fea.58', 'fea.59', 'fea.60', 'fea.61', 'fea.62',
       'fea.63', 'fea.64', 'fea.65', 'fea.66', 'fea.67', 'fea.68', 'fea.69',
       'fea.70', 'fea.71', 'fea.72', 'fea.73', 'fea.74', 'fea.75', 'fea.76',
       'fea.77', 'fea.78', 'fea.79', 'fea.80', 'fea.81'],
      dtype='object')
```

[171]:
```
#Get the shape of the dataset
DataA.shape
```

[171]:
```
(19000, 82)
```

[172]:
```
#Get top 5 rows of the dataset
DataA.head()
```

[172]:
```
   Unnamed: 0  fea.1  fea.2  fea.3  fea.4   fea.5  fea.6  fea.7  fea.8  fea.9  \
0           1 -153.0  414.0  939.0 -161.0  1007.0   99.0 -210.0  948.0  333.0
1           2 -150.0  420.0  939.0 -177.0  1008.0  103.0 -207.0  939.0  316.0
2           3 -160.0  432.0  941.0 -162.0   982.0   98.0 -198.0  936.0  315.0
3           4 -171.0  432.0  911.0 -174.0   999.0  115.0 -187.0  918.0  338.0
4           5 -171.0    NaN  929.0 -189.0  1004.0  104.0 -198.0  939.0  350.0

   …  fea.72  fea.73  fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  \
0  …   655.0  -316.0  -302.0  -617.0  -955.0  -264.0    23.0   -29.0
1  …   655.0  -309.0  -304.0  -619.0  -955.0  -265.0    19.0   -31.0
2  …   655.0  -302.0  -308.0  -621.0  -966.0  -270.0    10.0   -38.0
3  …   655.0  -293.0  -312.0  -622.0  -964.0  -269.0    14.0   -51.0
4  …   655.0  -284.0  -318.0  -624.0  -966.0  -262.0    24.0   -40.0

   fea.80  fea.81
0    36.0    24.0
1    47.0     3.0
2    20.0     0.0
3    33.0    -1.0
4     1.0     4.0

[5 rows x 82 columns]
```

[173]:
```
# Dropping the 'Unnamed: 0' column and reset the index for the columns
DataA = DataA.drop('Unnamed: 0',axis=1)
DataA = DataA.reset_index(drop=True)
```

[174]:
```
# Now there are 81 columns
DataA.head()
```

[174]:
```
   fea.1  fea.2  fea.3  fea.4   fea.5  fea.6  fea.7  fea.8  fea.9  fea.10  \
0 -153.0  414.0  939.0 -161.0  1007.0   99.0 -210.0  948.0  333.0   -19.0
```

```
1 -150.0  420.0  939.0 -177.0  1008.0  103.0 -207.0  939.0  316.0     9.0
2 -160.0  432.0  941.0 -162.0   982.0   98.0 -198.0  936.0  315.0   -10.0
3 -171.0  432.0  911.0 -174.0   999.0  115.0 -187.0  918.0  338.0    34.0
4 -171.0    NaN  929.0 -189.0  1004.0  104.0 -198.0  939.0  350.0    60.0

   …  fea.72  fea.73  fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  \
0  …   655.0  -316.0  -302.0  -617.0  -955.0  -264.0    23.0   -29.0
1  …   655.0  -309.0  -304.0  -619.0  -955.0  -265.0    19.0   -31.0
2  …   655.0  -302.0  -308.0  -621.0  -966.0  -270.0    10.0   -38.0
3  …   655.0  -293.0  -312.0  -622.0  -964.0  -269.0    14.0   -51.0
4  …   655.0  -284.0  -318.0  -624.0  -966.0  -262.0    24.0   -40.0

   fea.80  fea.81
0    36.0    24.0
1    47.0     3.0
2    20.0     0.0
3    33.0    -1.0
4     1.0     4.0

[5 rows x 81 columns]
```

[175]:
```python
DataA.shape
```

[175]:
```
(19000, 81)
```

[176]:
```python
# Knowing the dataset
# Checking the non-null and null values for each columns
DataA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19000 entries, 0 to 18999
Data columns (total 81 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   fea.1   17813 non-null  float64
 1   fea.2   17812 non-null  float64
 2   fea.3   17813 non-null  float64
 3   fea.4   18200 non-null  float64
 4   fea.5   18200 non-null  float64
 5   fea.6   18200 non-null  float64
 6   fea.7   18099 non-null  float64
 7   fea.8   18099 non-null  float64
 8   fea.9   18099 non-null  float64
 9   fea.10  18043 non-null  float64
 10  fea.11  18044 non-null  float64
 11  fea.12  18044 non-null  float64
 12  fea.13  17950 non-null  float64
```

```
13  fea.14  17950 non-null  float64
14  fea.15  17950 non-null  float64
15  fea.16  18202 non-null  float64
16  fea.17  18202 non-null  float64
17  fea.18  18202 non-null  float64
18  fea.19  17964 non-null  float64
19  fea.20  17964 non-null  float64
20  fea.21  17964 non-null  float64
21  fea.22  17677 non-null  float64
22  fea.23  17677 non-null  float64
23  fea.24  17677 non-null  float64
24  fea.25  18106 non-null  float64
25  fea.26  18106 non-null  float64
26  fea.27  18106 non-null  float64
27  fea.28  18106 non-null  float64
28  fea.29  18106 non-null  float64
29  fea.30  18106 non-null  float64
30  fea.31  18083 non-null  float64
31  fea.32  18083 non-null  float64
32  fea.33  18083 non-null  float64
33  fea.34  1 non-null      float64
34  fea.35  1 non-null      float64
35  fea.36  1 non-null      float64
36  fea.37  18227 non-null  float64
37  fea.38  18227 non-null  float64
38  fea.39  18227 non-null  float64
39  fea.40  18227 non-null  float64
40  fea.41  18227 non-null  float64
41  fea.42  18227 non-null  float64
42  fea.43  18227 non-null  float64
43  fea.44  18227 non-null  float64
44  fea.45  18227 non-null  float64
45  fea.46  18227 non-null  float64
46  fea.47  18227 non-null  float64
47  fea.48  18227 non-null  float64
48  fea.49  18227 non-null  float64
49  fea.50  18227 non-null  float64
50  fea.51  18227 non-null  float64
51  fea.52  18227 non-null  float64
52  fea.53  18227 non-null  float64
53  fea.54  18227 non-null  float64
54  fea.55  18227 non-null  float64
55  fea.56  18227 non-null  float64
56  fea.57  18227 non-null  float64
57  fea.58  18227 non-null  float64
58  fea.59  18227 non-null  float64
59  fea.60  18227 non-null  float64
60  fea.61  18227 non-null  float64
```

```
 61   fea.62  18227 non-null  float64
 62   fea.63  18227 non-null  float64
 63   fea.64  18227 non-null  float64
 64   fea.65  18227 non-null  float64
 65   fea.66  18227 non-null  float64
 66   fea.67  18227 non-null  float64
 67   fea.68  18227 non-null  float64
 68   fea.69  18227 non-null  float64
 69   fea.70  18226 non-null  float64
 70   fea.71  18227 non-null  float64
 71   fea.72  18227 non-null  float64
 72   fea.73  18227 non-null  float64
 73   fea.74  18227 non-null  float64
 74   fea.75  18227 non-null  float64
 75   fea.76  18227 non-null  float64
 76   fea.77  18227 non-null  float64
 77   fea.78  18227 non-null  float64
 78   fea.79  18227 non-null  float64
 79   fea.80  18227 non-null  float64
 80   fea.81  18227 non-null  float64
dtypes: float64(81)
memory usage: 11.7 MB
```

[177]:
```
# Checking total null values in columns
DataA.isnull().sum()
```

[177]:
```
fea.1     1187
fea.2     1188
fea.3     1187
fea.4      800
fea.5      800
          …
fea.77     773
fea.78     773
fea.79     773
fea.80     773
fea.81     773
Length: 81, dtype: int64
```

[178]:
```
# last 773 column has Null values
DataA.tail(773)
```

[178]:
```
       fea.1  fea.2  fea.3  fea.4  fea.5  fea.6  fea.7  fea.8  fea.9  fea.10  \
18227    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
18228    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
18229    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
18230    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
```

```
18231     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN      NaN
…        …     …     …     …     …     …     …     …     …      …
18995     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN      NaN
18996     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN      NaN
18997     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN      NaN
18998     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN      NaN
18999     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN      NaN

       …  fea.72  fea.73  fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  \
18227  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18228  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18229  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18230  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18231  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
…     …       …       …       …       …       …       …       …       …
18995  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18996  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18997  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18998  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
18999  …     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN

       fea.80  fea.81
18227     NaN     NaN
18228     NaN     NaN
18229     NaN     NaN
18230     NaN     NaN
18231     NaN     NaN
…          …       …
18995     NaN     NaN
18996     NaN     NaN
18997     NaN     NaN
18998     NaN     NaN
18999     NaN     NaN

[773 rows x 81 columns]
```

[179]:
```python
# Last 773 rows of the dataset are missing
df_na = DataA.any(skipna=True, axis=1)
df_na[df_na==False].shape[0]
```

[179]:
```
773
```

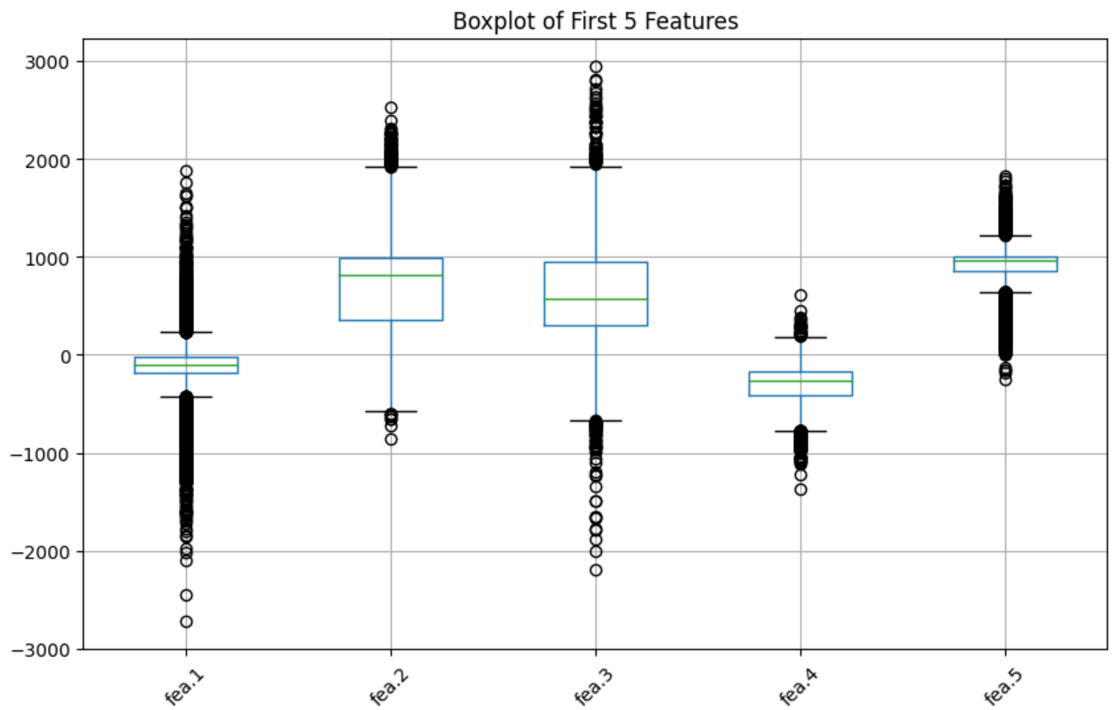
[180]:
```python
DataA.describe()
```

[180]:
```
              fea.1         fea.2         fea.3         fea.4         fea.5  \
count  17813.000000  17812.000000  17813.000000  18200.000000  18200.000000
mean    -132.812384    698.264485    597.541402   -307.128462    909.548077
```

```
std       284.183187    375.672475    396.654659    183.151634    193.963300
min     -2724.000000   -855.000000  -2196.000000  -1365.000000   -245.000000
25%      -179.000000    360.000000    304.000000   -409.000000    860.000000
50%      -100.000000    811.000000    574.000000   -266.000000    969.500000
75%       -15.000000    984.000000    955.000000   -167.000000   1006.000000
max      1887.000000   2531.000000   2941.000000    609.000000   1833.000000

               fea.6         fea.7         fea.8         fea.9        fea.10  \
count   18200.000000  18099.000000  18099.000000  18099.000000  18043.000000
mean      -32.760824     61.974363    899.313498     81.650478    356.638752
std       254.001018    317.393784    196.829252    327.904371    343.131382
min      -920.000000  -1580.000000   -149.000000  -1624.000000  -1792.000000
25%      -144.000000   -131.000000    854.000000   -155.000000    158.000000
50%       -39.000000     70.000000    946.000000     41.000000    380.000000
75%        45.000000    251.000000    997.000000    315.000000    583.000000
max      1215.000000   1490.000000   1682.000000   1096.000000   2202.000000

       …        fea.72        fea.73        fea.74        fea.75  \
count  …  18227.000000  18227.000000  18227.000000  18227.000000
mean   …   -124.658035    -37.973391    137.400176    374.762934
std    …    481.492994    355.841529    352.788441    583.792739
min    …   -953.000000   -853.000000   -771.000000   -984.000000
25%    …   -487.000000   -323.000000   -173.000000     29.000000
50%    …   -223.000000     32.000000    251.000000    698.000000
75%    …    174.000000    179.000000    413.000000    823.000000
max    …    949.000000    775.000000    759.000000    999.000000

              fea.76        fea.77        fea.78        fea.79        fea.80  \
count   18227.000000  18227.000000  18227.000000  18227.000000  18227.000000
mean     -880.583804    -47.607780    137.641192    -18.099523      4.671257
std       217.634117    373.064609    248.988603    778.015520    480.779966
min     -2562.000000  -5424.000000  -3133.000000  -7189.000000  -5861.000000
25%      -983.000000   -276.000000     31.000000   -246.500000   -118.000000
50%      -940.000000      0.000000    132.000000    -29.000000      4.000000
75%      -840.000000    225.000000    276.000000    195.000000    115.000000
max       613.000000   4877.000000   3742.000000   7497.000000   8675.000000

              fea.81
count   18227.000000
mean       20.726834
std       455.160604
min     -3051.000000
25%      -115.000000
50%        19.000000
75%       169.000000
max      5821.000000
```

[8 rows x 81 columns]

[181]:
```
# Select the first 5 features
first_5_features = DataA.iloc[:, :5]

# Create box plots for the first 5 features
first_5_features.boxplot(figsize=(10, 6))
plt.title('Boxplot of First 5 Features')
plt.xticks(rotation=45)
plt.show()
```



2. Fix the detected problems using some of the methods discussed in class.

i. First column unnamed does not provide any information.

Solution: Since unnamed column is just an index, dropping it is a reasonable solution. This can be done using the drop method.

ii. Feature 34, 35, and 36 columns have only 1 non-null value.

Solution: Since these columns have only one non-null value, they are unlikely to contribute meaningful information to our analysis. Therefore, dropping them is a reasonable solution. This can be done using the drop method.

iii. The last 773 rows of the data set values are missing 18,227 to 18,999

Solution: As these last 773 rows contain incomplete data and may not contribute meaningfully to the analysis, consider dropping these rows. By dropping these rows, we ensure that this dataset is more consistent and reliable for further analysis or modeling.

iv. Missing values or Null values present in the data set.

Solution: Used DataA.fillna(DataA.median(), inplace=True) to replace missing values with the median of each column.

v. Outliers present in the data.

Solution: In the dataset, the Interquartile Range (IQR) method is being utilized to detect and smooth outliers using a custom function. Through iteration over each numerical column, extreme values beyond a specified IQR range are replaced with the nearest boundary.

[182]:
```
DataA.shape
```

[182]:
```
(19000, 81)
```

[183]:
```
# Remove columns 'fea.34', 'fea.35', and 'fea.36' from the DataFrame DataA
DataA=DataA.drop(columns=['fea.34','fea.35','fea.36'])
# Display information about the DataFrame to show the non-null values present␣
↪in each column
DataA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19000 entries, 0 to 18999
Data columns (total 78 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   fea.1   17813 non-null  float64
 1   fea.2   17812 non-null  float64
 2   fea.3   17813 non-null  float64
 3   fea.4   18200 non-null  float64
 4   fea.5   18200 non-null  float64
 5   fea.6   18200 non-null  float64
 6   fea.7   18099 non-null  float64
 7   fea.8   18099 non-null  float64
 8   fea.9   18099 non-null  float64
 9   fea.10  18043 non-null  float64
 10  fea.11  18044 non-null  float64
 11  fea.12  18044 non-null  float64
 12  fea.13  17950 non-null  float64
 13  fea.14  17950 non-null  float64
 14  fea.15  17950 non-null  float64
 15  fea.16  18202 non-null  float64
 16  fea.17  18202 non-null  float64
 17  fea.18  18202 non-null  float64
 18  fea.19  17964 non-null  float64
 19  fea.20  17964 non-null  float64
 20  fea.21  17964 non-null  float64
```

```
21  fea.22  17677 non-null  float64
22  fea.23  17677 non-null  float64
23  fea.24  17677 non-null  float64
24  fea.25  18106 non-null  float64
25  fea.26  18106 non-null  float64
26  fea.27  18106 non-null  float64
27  fea.28  18106 non-null  float64
28  fea.29  18106 non-null  float64
29  fea.30  18106 non-null  float64
30  fea.31  18083 non-null  float64
31  fea.32  18083 non-null  float64
32  fea.33  18083 non-null  float64
33  fea.37  18227 non-null  float64
34  fea.38  18227 non-null  float64
35  fea.39  18227 non-null  float64
36  fea.40  18227 non-null  float64
37  fea.41  18227 non-null  float64
38  fea.42  18227 non-null  float64
39  fea.43  18227 non-null  float64
40  fea.44  18227 non-null  float64
41  fea.45  18227 non-null  float64
42  fea.46  18227 non-null  float64
43  fea.47  18227 non-null  float64
44  fea.48  18227 non-null  float64
45  fea.49  18227 non-null  float64
46  fea.50  18227 non-null  float64
47  fea.51  18227 non-null  float64
48  fea.52  18227 non-null  float64
49  fea.53  18227 non-null  float64
50  fea.54  18227 non-null  float64
51  fea.55  18227 non-null  float64
52  fea.56  18227 non-null  float64
53  fea.57  18227 non-null  float64
54  fea.58  18227 non-null  float64
55  fea.59  18227 non-null  float64
56  fea.60  18227 non-null  float64
57  fea.61  18227 non-null  float64
58  fea.62  18227 non-null  float64
59  fea.63  18227 non-null  float64
60  fea.64  18227 non-null  float64
61  fea.65  18227 non-null  float64
62  fea.66  18227 non-null  float64
63  fea.67  18227 non-null  float64
64  fea.68  18227 non-null  float64
65  fea.69  18227 non-null  float64
66  fea.70  18226 non-null  float64
67  fea.71  18227 non-null  float64
68  fea.72  18227 non-null  float64
```

```
 69   fea.73  18227 non-null  float64
 70   fea.74  18227 non-null  float64
 71   fea.75  18227 non-null  float64
 72   fea.76  18227 non-null  float64
 73   fea.77  18227 non-null  float64
 74   fea.78  18227 non-null  float64
 75   fea.79  18227 non-null  float64
 76   fea.80  18227 non-null  float64
 77   fea.81  18227 non-null  float64
dtypes: float64(78)
memory usage: 11.3 MB
```

[184]:
```python
# Drop the last 773 rows
DataA = DataA.iloc[:-773]
DataA.shape
```

[184]:
```
(18227, 78)
```

[185]:
```python
# Fill null values with median
DataA.fillna(DataA.median(), inplace=True)
DataA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18227 entries, 0 to 18226
Data columns (total 78 columns):
 #    Column  Non-Null Count  Dtype
---   ------  --------------  -----
 0    fea.1   18227 non-null  float64
 1    fea.2   18227 non-null  float64
 2    fea.3   18227 non-null  float64
 3    fea.4   18227 non-null  float64
 4    fea.5   18227 non-null  float64
 5    fea.6   18227 non-null  float64
 6    fea.7   18227 non-null  float64
 7    fea.8   18227 non-null  float64
 8    fea.9   18227 non-null  float64
 9    fea.10  18227 non-null  float64
 10   fea.11  18227 non-null  float64
 11   fea.12  18227 non-null  float64
 12   fea.13  18227 non-null  float64
 13   fea.14  18227 non-null  float64
 14   fea.15  18227 non-null  float64
 15   fea.16  18227 non-null  float64
 16   fea.17  18227 non-null  float64
 17   fea.18  18227 non-null  float64
 18   fea.19  18227 non-null  float64
 19   fea.20  18227 non-null  float64
 20   fea.21  18227 non-null  float64
```

```
21   fea.22  18227 non-null  float64
22   fea.23  18227 non-null  float64
23   fea.24  18227 non-null  float64
24   fea.25  18227 non-null  float64
25   fea.26  18227 non-null  float64
26   fea.27  18227 non-null  float64
27   fea.28  18227 non-null  float64
28   fea.29  18227 non-null  float64
29   fea.30  18227 non-null  float64
30   fea.31  18227 non-null  float64
31   fea.32  18227 non-null  float64
32   fea.33  18227 non-null  float64
33   fea.37  18227 non-null  float64
34   fea.38  18227 non-null  float64
35   fea.39  18227 non-null  float64
36   fea.40  18227 non-null  float64
37   fea.41  18227 non-null  float64
38   fea.42  18227 non-null  float64
39   fea.43  18227 non-null  float64
40   fea.44  18227 non-null  float64
41   fea.45  18227 non-null  float64
42   fea.46  18227 non-null  float64
43   fea.47  18227 non-null  float64
44   fea.48  18227 non-null  float64
45   fea.49  18227 non-null  float64
46   fea.50  18227 non-null  float64
47   fea.51  18227 non-null  float64
48   fea.52  18227 non-null  float64
49   fea.53  18227 non-null  float64
50   fea.54  18227 non-null  float64
51   fea.55  18227 non-null  float64
52   fea.56  18227 non-null  float64
53   fea.57  18227 non-null  float64
54   fea.58  18227 non-null  float64
55   fea.59  18227 non-null  float64
56   fea.60  18227 non-null  float64
57   fea.61  18227 non-null  float64
58   fea.62  18227 non-null  float64
59   fea.63  18227 non-null  float64
60   fea.64  18227 non-null  float64
61   fea.65  18227 non-null  float64
62   fea.66  18227 non-null  float64
63   fea.67  18227 non-null  float64
64   fea.68  18227 non-null  float64
65   fea.69  18227 non-null  float64
66   fea.70  18227 non-null  float64
67   fea.71  18227 non-null  float64
68   fea.72  18227 non-null  float64
```

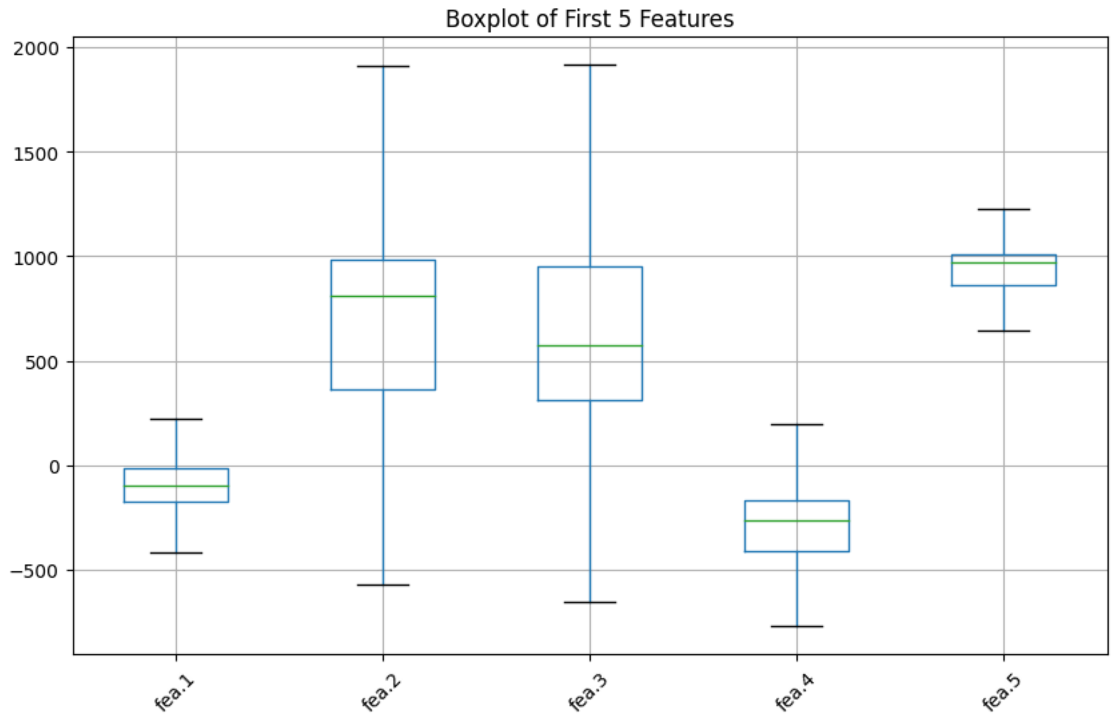
```
 69   fea.73  18227 non-null  float64
 70   fea.74  18227 non-null  float64
 71   fea.75  18227 non-null  float64
 72   fea.76  18227 non-null  float64
 73   fea.77  18227 non-null  float64
 74   fea.78  18227 non-null  float64
 75   fea.79  18227 non-null  float64
 76   fea.80  18227 non-null  float64
 77   fea.81  18227 non-null  float64
dtypes: float64(78)
memory usage: 10.8 MB
```

[186]:

```
# Function to smooth outliers using IQR
def smooth_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    column[column < lower_bound] = lower_bound
    column[column > upper_bound] = upper_bound
    return column

# Apply the function to each numerical column in the DataFrame
numerical_columns = DataA.select_dtypes(include='number').columns
for column in numerical_columns:
    DataA[column] = smooth_outliers_iqr(DataA[column])
```

[187]:

```
# Select the first 5 features
first_5_features = DataA.iloc[:, :5]

# Create box plots for the first 5 features
first_5_features.boxplot(figsize=(10, 6))
plt.title('Boxplot of First 5 Features')
plt.xticks(rotation=45)
plt.show()
```



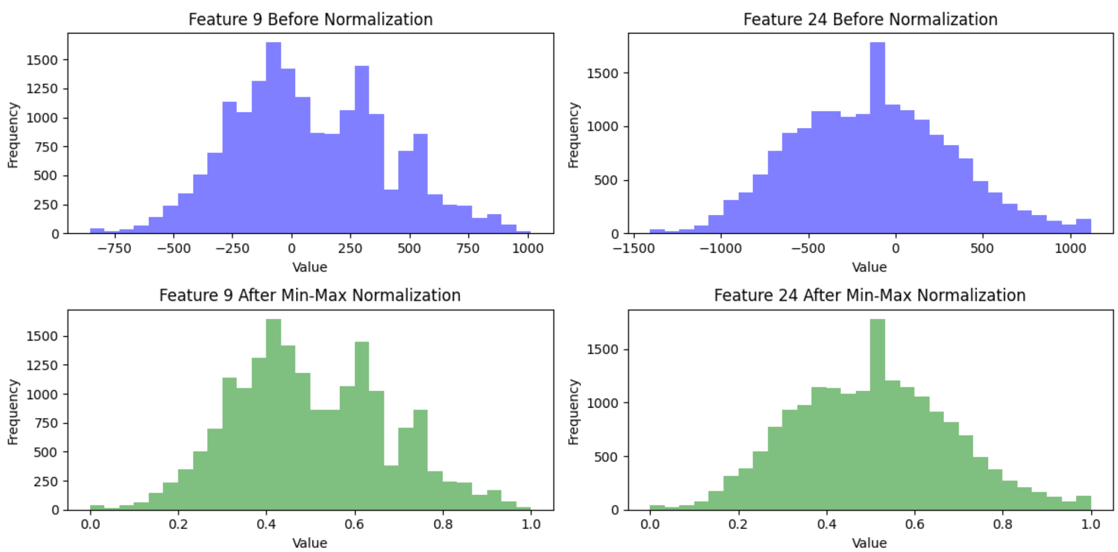
3. Normalize the data using min-max and z-score normalization. Plot histograms of feature 9 and 24; compare and comment on the differences before and after normalization.

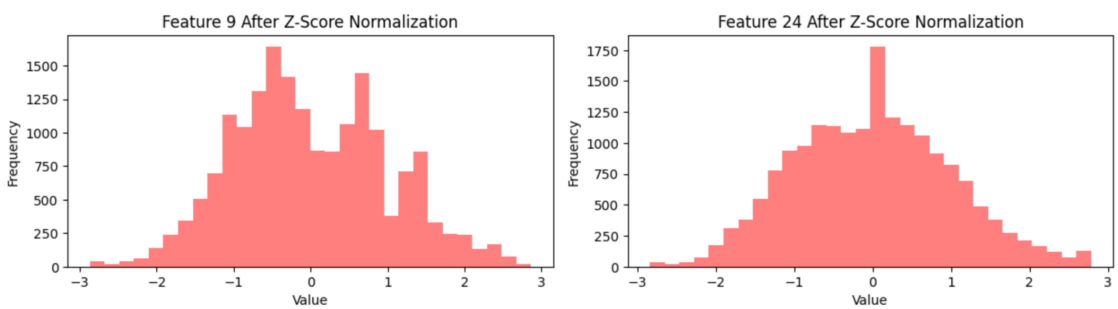
[188]:
```python
# We have compare the feature 9 and 24 before and after normalization using 2␣
↪types plot diagram.
# Extract features 9 and 24
feature_9 = DataA['fea.9']
feature_24 = DataA['fea.24']

# Initialize MinMaxScaler and StandardScaler objects
min_max_scaler = MinMaxScaler()
standard_scaler = StandardScaler()

# Min-max normalization
feature_9_minmax = min_max_scaler.fit_transform(feature_9.values.reshape(-1, 1))
feature_24_minmax = min_max_scaler.fit_transform(feature_24.values.reshape(-1,␣
↪1))

# Z-score normalization
feature_9_zscore = standard_scaler.fit_transform(feature_9.values.reshape(-1,␣
↪1))
feature_24_zscore = standard_scaler.fit_transform(feature_24.values.reshape(-1,␣
↪1))
```

[189]:

```python
# Plot-1 diagram

# Plot histograms before and after normalization
plt.figure(figsize=(12, 6))

# Before normalization, the histograms of features 9 and 24 show the␣
↪distribution of their values in their original scales.
plt.subplot(2, 2, 1)
plt.hist(feature_9, bins=30, color='blue', alpha=0.5)
plt.title('Feature 9 Before Normalization')
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
plt.hist(feature_24, bins=30, color='blue', alpha=0.5)
plt.title('Feature 24 Before Normalization')
plt.xlabel('Value')
plt.ylabel('Frequency')

# After Min-Max normalization, both features have values scaled between 0 and 1.
# The shape of the distribution remains similar, but the scale is adjusted.
plt.subplot(2, 2, 3)
plt.hist(feature_9_minmax, bins=30, color='green', alpha=0.5)
plt.title('Feature 9 After Min-Max Normalization')
plt.xlabel('Value')
plt.ylabel('Frequency')


plt.subplot(2, 2, 4)
plt.hist(feature_24_minmax, bins=30, color='green', alpha=0.5)
plt.title('Feature 24 After Min-Max Normalization')
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

# After z-score normalization, the histograms show distributions centered␣
↪around 0 with a standard deviation of 1.
# The original shape of the distribution is preserved, but the scale is␣
↪standardized
plt.subplot(2, 2, 1)
plt.hist(feature_9_zscore, bins=30, color='red', alpha=0.5)
plt.title('Feature 9 After Z-Score Normalization')
plt.xlabel('Value')
```

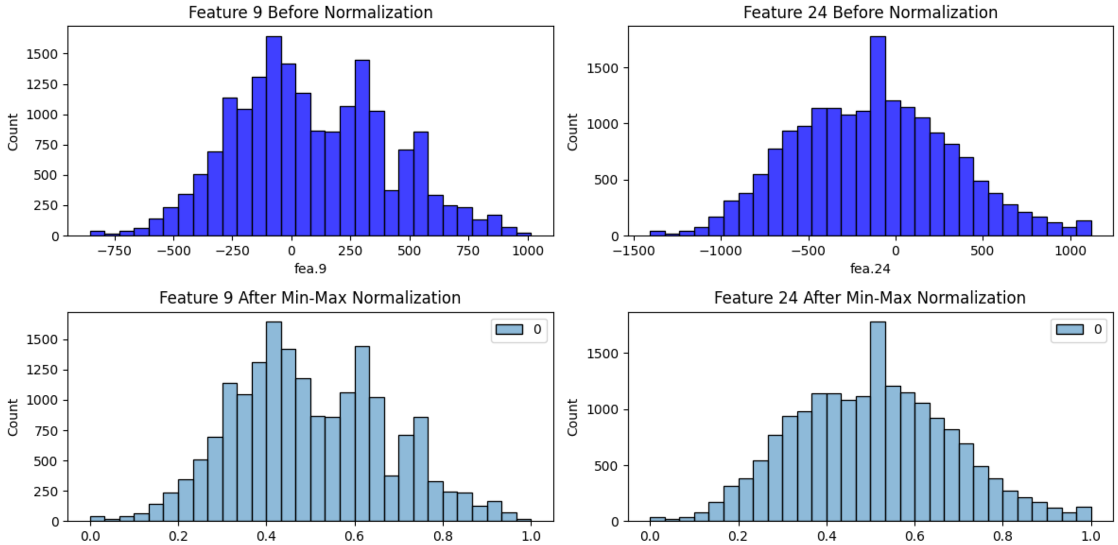
```
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
plt.hist(feature_24_zscore, bins=30, color='red', alpha=0.5)
plt.title('Feature 24 After Z-Score Normalization')
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

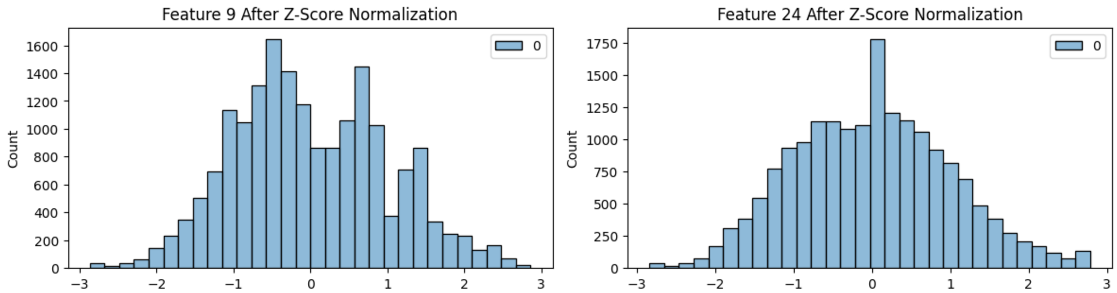




[190]:
```
# Plot-2 diagram

# Plot histograms before and after normalization
plt.figure(figsize=(12, 6))
```

```python
# Before normalization, the histograms of features 9 and 24 show the␣
↪distribution of their values in their original scales.
plt.subplot(2, 2, 1)
sns.histplot(feature_9, bins=30, color='blue')
plt.title('Feature 9 Before Normalization')

plt.subplot(2, 2, 2)
sns.histplot(feature_24, bins=30, color='blue')
plt.title('Feature 24 Before Normalization')

# After Min-Max normalization, both features have values scaled between 0 and 1.
# The shape of the distribution remains similar, but the scale is adjusted.
plt.subplot(2, 2, 3)
sns.histplot(feature_9_minmax, bins=30, color='green')
plt.title('Feature 9 After Min-Max Normalization')

plt.subplot(2, 2, 4)
sns.histplot(feature_24_minmax, bins=30, color='green')
plt.title('Feature 24 After Min-Max Normalization')

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

# After z-score normalization, the histograms show distributions centered␣
↪around 0 with a standard deviation of 1.
# The original shape of the distribution is preserved, but the scale is␣
↪standardized
plt.subplot(2, 2, 1)
sns.histplot(feature_9_zscore, bins=30, color='red')
plt.title('Feature 9 After Z-Score Normalization')

plt.subplot(2, 2, 2)
sns.histplot(feature_24_zscore, bins=30, color='red')
plt.title('Feature 24 After Z-Score Normalization')

plt.tight_layout()
plt.show()
```





Analysis

Data normalization using min-max and z-score methods was conducted, followed by a comparison of histograms for features 9 and 24 before and after normalization.

Before normalization, the histograms show the raw distributions of features 9 and 24, which may have varying scales and spreads. After normalization:

i. Min-Max normalization (green histograms) scales both features to a range between 0 and 1, preserving the original distribution shape but standardizing the scale.

ii. Z-score normalization (red histograms) centers the data around a mean of 0 and scales it according to the standard deviation, resulting in standardized distributions with mean values close to 0 and a standard deviation of 1.

[190]:

3 Question II. Feature Extraction (for dataset B)

1. Use PCA as a dimensionality reduction technique to the data, compute the eigenvectors and eigenvalues.

[191]:
```
#Load the dataset
DataB = pd.read_csv('DataB.csv',encoding='latin-1')
# Seperating the classes and features
DataB_X = DataB.iloc[:,:-1]
DataB_Y = DataB.iloc[:,-1]
DataB.head()
```

[191]:
```
   Unnamed: 0  fea.1  fea.2  fea.3  fea.4  fea.5  fea.6  fea.7  fea.8  fea.9  \
0           1      4      4      3      0      0      4      2      1      4
1           2      5      1      4      3      1      3      5      1      4
2           3      1      3      0      3      1      1      0      1      0
3           4      5      3      2      3      5      2      2      0      4
4           5      3      5      3      3      0      4      1      1      4

   …  fea.776  fea.777  fea.778  fea.779  fea.780  fea.781  fea.782  \
0  …        1        3        0        4        2        1        1
1  …        1        1        3        3        1        3        3
2  …        3        0        2        4        2        2        1
3  …        5        4        5        1        4        4        2
4  …        1        3        3        3        1        2        4

   fea.783  fea.784  gnd
0        4        5    0
1        5        4    0
2        2        4    0
3        4        4    0
4        1        1    0

[5 rows x 786 columns]
```

[192]:
```
DataB.shape
```

[192]:
```
(2066, 786)
```

[193]:
```
print(DataB.columns)
```

```
Index(['Unnamed: 0', 'fea.1', 'fea.2', 'fea.3', 'fea.4', 'fea.5', 'fea.6',
       'fea.7', 'fea.8', 'fea.9',
       …
       'fea.776', 'fea.777', 'fea.778', 'fea.779', 'fea.780', 'fea.781',
       'fea.782', 'fea.783', 'fea.784', 'gnd'],
      dtype='object', length=786)
```

[194]:
```
# Drop the 'Unnamed: 0' column
DataB = DataB.drop('Unnamed: 0', axis=1)
DataB = DataB.drop('gnd', axis=1)
# Reset the index
DataB = DataB.reset_index(drop=True)
```

[195]:
```
DataB.shape
```

[195]:
```
(2066, 784)
```

[196]:
```
DataB_centered = DataB - DataB.mean()
DataB_centered.head()
```

[196]:
```
      fea.1     fea.2     fea.3     fea.4     fea.5     fea.6     fea.7  \
0  1.491772  1.452565  0.539206 -2.496612 -2.472894  1.509681 -0.486447
1  2.491772 -1.547435  1.539206  0.503388 -1.472894  0.509681  2.513553
2 -1.508228  0.452565 -2.460794  0.503388 -1.472894 -1.490319 -2.486447
3  2.491772  0.452565 -0.460794  0.503388  2.527106 -0.490319 -0.486447
4  0.491772  2.452565  0.539206  0.503388 -2.472894  1.509681 -1.486447

      fea.8     fea.9    fea.10  …   fea.775   fea.776   fea.777   fea.778  \
0 -1.512585  1.477735 -1.482091  … -1.517909 -1.469506  0.477251 -2.486447
1 -1.512585  1.477735  1.517909  …  0.482091 -1.469506 -1.522749  0.513553
2 -1.512585 -2.522265 -0.482091  …  1.482091  0.530494 -2.522749 -0.486447
3 -2.512585  1.477735  2.517909  …  1.482091  2.530494  1.477251  2.513553
4 -1.512585  1.477735  0.517909  … -1.517909 -1.469506  0.477251  0.513553

    fea.779   fea.780   fea.781  fea.782  fea.783   fea.784
0  1.550339 -0.498064 -1.525653 -1.54211  1.59971  2.480639
1  0.550339 -1.498064  0.474347  0.45789  2.59971  1.480639
2  1.550339 -0.498064 -0.525653 -1.54211 -0.40029  1.480639
3 -1.449661  1.501936  1.474347 -0.54211  1.59971  1.480639
4  0.550339 -1.498064 -0.525653  1.45789 -1.40029 -1.519361

[5 rows x 784 columns]
```

[197]:
```
X = np.asmatrix(DataB_centered)
cov_mat = np.cov(X.T)
eigenvalues, eigenvectors = np.linalg.eig(cov_mat)
```

[198]:
```
print("EigenValues :")
print(eigenvalues)
print("EigenVectors :")
print(eigenvectors)
```

```
EigenValues :
[4.67242207e+05 2.78894146e+05 2.13480284e+05 2.05514154e+05
```

1.71638869e+05	1.29473256e+05	1.13282522e+05	9.13665833e+04
8.81948304e+04	7.26695964e+04	6.47973043e+04	5.91614589e+04
5.71810362e+04	5.15388208e+04	4.71162983e+04	4.30116981e+04
4.01681360e+04	3.92327232e+04	3.81662137e+04	3.44883896e+04
3.25474987e+04	3.08116460e+04	2.87269206e+04	2.77117300e+04
2.66864459e+04	2.59429468e+04	2.44575328e+04	2.37064782e+04
2.32238894e+04	2.19475845e+04	2.14949943e+04	1.99553743e+04
1.95307071e+04	1.77691867e+04	1.67857005e+04	1.61692889e+04
1.63009352e+04	1.55764739e+04	1.45129452e+04	1.41356650e+04
1.37490819e+04	1.31545707e+04	1.23464386e+04	1.19231480e+04
1.17660533e+04	1.16006832e+04	1.13650854e+04	1.12916028e+04
1.05749096e+04	1.00068073e+04	9.88224042e+03	9.42109813e+03
9.06882290e+03	8.99328094e+03	8.78820190e+03	8.58885957e+03
8.27499662e+03	7.72571780e+03	7.83097385e+03	7.42459108e+03
7.14859947e+03	7.04846654e+03	6.86439480e+03	6.72662421e+03
6.47571542e+03	6.47925363e+03	6.30326225e+03	6.11098144e+03
5.92944061e+03	5.88810104e+03	5.59999789e+03	5.55085964e+03
5.33254963e+03	5.28563156e+03	5.07690050e+03	5.00405724e+03
4.86324361e+03	4.75034496e+03	4.56436941e+03	4.64618273e+03
4.40537267e+03	4.37834758e+03	4.24648180e+03	4.11644413e+03
4.01634352e+03	3.94486622e+03	3.82564914e+03	3.73794488e+03
3.70756420e+03	3.66946405e+03	3.58233227e+03	3.50805016e+03
3.44314126e+03	3.36091271e+03	3.32593381e+03	3.24766347e+03
3.18206524e+03	3.12525936e+03	3.04518281e+03	3.00134716e+03
2.93050507e+03	2.89831283e+03	2.86512018e+03	2.81358559e+03
2.80286214e+03	2.72572152e+03	2.65959932e+03	2.64001728e+03
2.59943188e+03	2.55084805e+03	2.45327569e+03	2.50162453e+03
2.40185642e+03	2.33619179e+03	2.30758070e+03	2.22694272e+03
2.28617001e+03	2.27407565e+03	2.17603201e+03	2.12254245e+03
2.13073914e+03	2.09810587e+03	2.06215984e+03	2.03592893e+03
2.01720784e+03	1.96864324e+03	1.94984425e+03	1.93831264e+03
1.90197437e+03	1.87576506e+03	1.84149958e+03	1.82530820e+03
1.80227395e+03	1.76016048e+03	1.74197905e+03	1.69962356e+03
1.72346098e+03	1.63916043e+03	1.64968360e+03	1.61003160e+03
1.59119934e+03	1.58094362e+03	1.55651230e+03	1.53975040e+03
1.52124346e+03	1.48789670e+03	1.47952440e+03	1.38044144e+03
1.44706618e+03	1.41106473e+03	1.40920923e+03	1.43994383e+03
1.43824759e+03	1.34891484e+03	1.33925441e+03	1.33459516e+03
1.31120537e+03	1.29509564e+03	1.29279233e+03	1.26709812e+03
1.26063023e+03	1.24661825e+03	1.22122636e+03	1.21078048e+03
1.19429232e+03	1.18551512e+03	1.17023828e+03	1.16263550e+03
1.15266053e+03	1.12494728e+03	1.11423946e+03	1.10360267e+03
1.09749067e+03	1.09437304e+03	1.07308311e+03	1.06780345e+03
1.05036693e+03	1.00753852e+03	1.03738611e+03	1.02605934e+03
1.01675570e+03	8.74853070e+02	8.84325346e+02	9.89611727e+02
9.76954606e+02	9.23105231e+02	8.87735796e+02	9.51102064e+02
9.57522045e+02	9.62990548e+02	9.12623663e+02	8.95916572e+02
9.34693258e+02	9.83897878e+02	9.19485680e+02	8.76639769e+02

8.62457391e+02	8.56604270e+02	8.24949902e+02	8.32045927e+02
8.38805737e+02	8.46060912e+02	8.07933774e+02	8.01549193e+02
7.97799519e+02	7.93570117e+02	7.87544758e+02	7.76804385e+02
7.63357048e+02	7.53466944e+02	7.44517086e+02	7.56673138e+02
7.39389969e+02	7.26074502e+02	7.24303882e+02	7.15204818e+02
7.10375008e+02	7.02262036e+02	6.94205778e+02	6.86455992e+02
6.80618411e+02	6.72978762e+02	6.66535340e+02	6.67086757e+02
6.59709169e+02	6.54263731e+02	6.46890630e+02	6.37272169e+02
6.35949394e+02	6.18731045e+02	5.26344782e+02	6.26509491e+02
6.09940787e+02	5.45702583e+02	5.33386077e+02	5.31585412e+02
5.35965265e+02	5.77767921e+02	6.02922881e+02	5.92660156e+02
5.75368549e+02	5.57577848e+02	6.25617741e+02	5.67920798e+02
5.61499826e+02	5.50010484e+02	5.99214050e+02	5.98302440e+02
5.37107871e+02	5.89219987e+02	5.63164653e+02	5.05057182e+02
5.08196864e+02	5.18607186e+02	5.17003665e+02	5.17397267e+02
4.93798322e+02	4.97737618e+02	4.89358947e+02	4.85668210e+02
4.81342460e+02	4.75281512e+02	4.68517988e+02	4.65527419e+02
4.61875882e+02	3.73885637e+02	3.77859762e+02	4.59706623e+02
3.85239340e+02	3.82642405e+02	3.88614730e+02	4.42185915e+02
4.53759112e+02	4.48147488e+02	4.52709208e+02	4.34146592e+02
3.95139452e+02	3.99532803e+02	4.15655739e+02	3.92792403e+02
4.27020437e+02	4.11813803e+02	3.83098607e+02	4.06650400e+02
4.49112092e+02	4.08560714e+02	4.23747584e+02	4.22599052e+02
4.30133768e+02	4.02727820e+02	4.31587094e+02	3.71381755e+02
3.67487069e+02	3.64126767e+02	3.74840614e+02	2.83173275e+02
3.57015638e+02	3.55453083e+02	3.51696262e+02	2.85844230e+02
3.47162307e+02	2.89403878e+02	2.90084215e+02	3.35298686e+02
3.41270916e+02	3.30920190e+02	2.91720997e+02	3.26842218e+02
3.43207178e+02	2.95912146e+02	3.19549792e+02	3.11755811e+02
3.09540628e+02	3.23366968e+02	3.15617740e+02	3.06492789e+02
2.99279340e+02	3.01334601e+02	3.05669319e+02	3.42583728e+02
3.49953245e+02	3.00524535e+02	3.22994230e+02	2.82081415e+02
2.77592134e+02	2.76014692e+02	2.79706379e+02	2.71084906e+02
2.69322223e+02	2.67289655e+02	2.63813067e+02	2.60634013e+02
2.14862562e+02	2.16817446e+02	2.57881835e+02	2.20371461e+02
2.22091834e+02	2.52107192e+02	2.54768953e+02	2.54004619e+02
2.49641712e+02	2.24678716e+02	2.46847798e+02	2.40649190e+02
2.43604230e+02	2.45685763e+02	2.35696116e+02	2.34712315e+02
2.26632079e+02	2.30480811e+02	2.37284545e+02	2.28642243e+02
2.31036581e+02	2.38796847e+02	2.13851839e+02	2.12099016e+02
2.03684826e+02	2.08805420e+02	2.06608687e+02	2.01486779e+02
2.00324187e+02	2.00685328e+02	1.96864112e+02	1.95743272e+02
1.91564370e+02	1.88703297e+02	1.90625273e+02	1.89690726e+02
1.85582676e+02	1.81343658e+02	1.78721687e+02	1.40075782e+02
1.75940415e+02	1.82743038e+02	1.70467898e+02	1.73331016e+02
1.43722864e+02	1.44585328e+02	1.54969548e+02	1.55871174e+02
1.58565442e+02	1.66423564e+02	1.42948192e+02	1.62527701e+02
1.72831320e+02	1.50333953e+02	1.49463412e+02	1.60443325e+02

1.63139562e+02	1.50584404e+02	1.66919626e+02	1.77181047e+02
1.37722868e+02	1.37463867e+02	1.36119777e+02	1.34778065e+02
1.30081961e+02	1.31522088e+02	1.23933488e+02	1.26545455e+02
1.26727965e+02	1.21748826e+02	1.20516826e+02	1.16321769e+02
1.10762661e+02	1.13241971e+02	1.12522848e+02	1.15732168e+02
1.09986558e+02	1.07695045e+02	1.06340126e+02	1.04893795e+02
1.02741746e+02	1.02216908e+02	1.00479284e+02	9.95391769e+01
9.89531265e+01	9.58979786e+01	9.66832412e+01	5.62900674e+01
5.66792106e+01	9.40099561e+01	5.75024956e+01	5.75836583e+01
5.84543472e+01	9.11919870e+01	5.94467528e+01	6.05743963e+01
6.26052655e+01	6.30789169e+01	8.84643635e+01	8.66292233e+01
8.92254411e+01	8.70283903e+01	6.60716551e+01	6.99329228e+01
6.89367091e+01	8.31134660e+01	8.06154572e+01	7.78279367e+01
7.29420610e+01	7.42339949e+01	6.52693353e+01	7.86787615e+01
8.18581996e+01	6.73454368e+01	7.37842037e+01	8.27267817e+01
7.17529938e+01	4.34006423e+01	4.38030692e+01	4.49652979e+01
4.59313848e+01	4.66073355e+01	5.35396968e+01	5.27636065e+01
4.94270949e+01	5.13761639e+01	5.04655922e+01	4.90722790e+01
5.47373858e+01	5.01560415e+01	4.29415186e+01	4.16149272e+01
4.17966235e+01	4.03631707e+01	3.91944917e+01	3.79291821e+01
3.87353274e+01	3.84117729e+01	3.62137984e+01	3.72068855e+01
3.53388664e+01	3.45169533e+01	2.87037195e+01	3.00595304e+01
3.24415724e+01	3.31182098e+01	3.37388082e+01	2.74954413e+01
2.60721957e+01	3.16184386e+01	2.91209759e+01	2.69698757e+01
2.45278398e+01	2.50406847e+01	3.18384593e+01	3.06373320e+01
2.63719330e+01	2.30084254e+01	2.34664354e+01	2.38327124e+01
2.24227615e+01	2.11883641e+01	2.13713292e+01	2.07262033e+01
1.97964286e+01	1.95083926e+01	1.94120298e+01	1.81634260e+01
1.85854333e+01	1.83469325e+01	1.75243837e+01	1.67898609e+01
1.59805568e+01	1.57995421e+01	1.56003391e+01	1.49609822e+01
1.44756352e+01	1.40441271e+01	1.38002918e+01	1.32554970e+01
1.28638760e+01	1.13625233e+01	1.25494154e+01	1.23477699e+01
1.24390944e+01	1.20611669e+01	1.01717712e+01	9.57864320e+00
9.69439484e+00	9.40876917e+00	8.79351581e+00	8.29323203e+00
8.10287952e+00	7.91950500e+00	7.76510708e+00	7.52042303e+00
7.27536971e+00	6.69502075e+00	6.36173086e+00	6.02513388e+00
5.79981772e+00	5.46140852e+00	5.20034129e+00	5.08894921e+00
4.99553493e+00	4.70416634e+00	4.45972062e+00	4.35378292e+00
4.29347527e+00	4.00364919e+00	3.84760105e+00	3.90008090e+00
3.90950099e+00	3.60918634e+00	3.51338753e+00	3.40803094e+00
3.31125363e+00	3.25199259e+00	3.22338033e+00	3.19405896e+00
3.14198582e+00	3.07564645e+00	3.06596852e+00	3.01990548e+00
2.99770494e+00	2.98083425e+00	2.92935909e+00	2.90450656e+00
6.11666181e-01	2.85039518e+00	2.85708208e+00	6.33176360e-01
6.49974412e-01	6.55059466e-01	2.81946772e+00	6.70248559e-01
2.78835450e+00	2.80317057e+00	2.75192835e+00	6.93880896e-01
2.72439268e+00	7.25524005e-01	7.08481656e-01	7.08864551e-01
7.31384668e-01	2.71097709e+00	2.70825269e+00	2.68589901e+00

2.67369519e+00	7.43979885e-01	7.37200370e-01	2.64436017e+00
2.64055968e+00	2.61989901e+00	2.59903020e+00	2.57715507e+00
2.56095208e+00	2.54461286e+00	2.52637725e+00	2.49880335e+00
2.48999888e+00	2.48122893e+00	2.49394850e+00	2.46473686e+00
2.44163703e+00	2.43041895e+00	2.41509411e+00	2.40106290e+00
2.40455514e+00	2.38230545e+00	2.37012050e+00	2.34446524e+00
2.33130546e+00	2.31452922e+00	2.31017663e+00	2.30330119e+00
2.28221304e+00	2.26949722e+00	2.25130361e+00	2.24810843e+00
2.23959738e+00	2.22864543e+00	2.20951539e+00	2.19975359e+00
2.18382383e+00	2.16480233e+00	2.14889923e+00	1.98478194e+00
2.00166112e+00	2.02102605e+00	2.02660131e+00	2.11899347e+00
1.99414634e+00	2.04975486e+00	2.09961679e+00	2.13153253e+00
2.07520609e+00	2.11594185e+00	2.06673951e+00	2.08622461e+00
2.10708690e+00	2.05617500e+00	2.07116282e+00	7.53905429e-01
7.64478756e-01	7.70673626e-01	7.78175557e-01	7.90567365e-01
7.94456975e-01	8.04061987e-01	8.17635832e-01	8.20627239e-01
8.24697760e-01	8.48161263e-01	8.56931834e-01	8.39795795e-01
8.63590387e-01	8.66674340e-01	8.76211709e-01	8.92888819e-01
8.85215704e-01	1.96367946e+00	1.95319261e+00	1.94956745e+00
1.93312686e+00	1.93633798e+00	1.91440579e+00	1.90679896e+00
1.90095201e+00	1.89144619e+00	1.87128530e+00	1.86058121e+00
1.88244345e+00	1.85137616e+00	1.82377541e+00	1.84614237e+00
1.80948092e+00	1.79571402e+00	1.83735921e+00	1.78083713e+00
1.80187251e+00	1.75950870e+00	1.74860339e+00	1.77684557e+00
1.77034443e+00	8.99493564e-01	9.10834873e-01	9.04520350e-01
9.18373473e-01	9.29451492e-01	1.74597160e+00	1.72878691e+00
1.72676629e+00	1.71971906e+00	9.43639507e-01	9.48091439e-01
9.50504154e-01	1.70914995e+00	1.70027374e+00	1.69204946e+00
1.68675176e+00	9.66452459e-01	1.66818052e+00	1.66307172e+00
1.65156148e+00	1.64865607e+00	1.62543118e+00	1.61737997e+00
9.74104017e-01	9.73258752e-01	9.83108691e-01	1.63990303e+00
9.85722842e-01	1.59890130e+00	1.60768603e+00	1.58505280e+00
1.00074915e+00	9.98077574e-01	1.00897347e+00	1.59301872e+00
1.56786186e+00	1.05043893e+00	1.06122329e+00	1.56287239e+00
1.55181968e+00	1.54567259e+00	1.54174404e+00	1.51957639e+00
1.52867202e+00	1.03410813e+00	1.02962165e+00	1.02420200e+00
1.01855309e+00	1.52361374e+00	1.04729258e+00	1.06360010e+00
1.07408269e+00	1.07995491e+00	1.49384463e+00	1.09024996e+00
1.49210173e+00	1.48763002e+00	1.48062235e+00	1.09914463e+00
1.10744305e+00	1.46848098e+00	1.11170403e+00	1.12570645e+00
1.11796125e+00	1.46325255e+00	1.45463378e+00	1.44786655e+00
1.13699912e+00	1.14310959e+00	1.43711771e+00	1.15137945e+00
1.42580775e+00	1.15733885e+00	1.41691371e+00	1.16505204e+00
1.18043758e+00	1.16964459e+00	1.40385162e+00	1.40178591e+00
1.39899611e+00	1.18643220e+00	1.19340583e+00	1.19973423e+00
1.37304050e+00	1.21044276e+00	1.33605998e+00	1.32730707e+00
1.31239514e+00	1.35076834e+00	1.30713650e+00	1.36439423e+00
1.29574466e+00	1.22003708e+00	1.34416954e+00	1.23972447e+00

```
 1.28724280e+00 1.24524184e+00 1.27455593e+00 1.21574345e+00
 1.28460163e+00 1.36215169e+00 1.37905864e+00 1.37747057e+00
 1.26326638e+00 1.25175309e+00 1.25762102e+00 1.22919039e+00]
EigenVectors :
[[ 3.61067274e-05 -4.10014533e-05 -4.90102245e-05 … -3.97013571e-02
   9.22625091e-03  4.15737480e-04]
 [ 1.88873948e-05 -2.61454773e-05 -1.02341601e-05 … -4.09562800e-02
  -5.85006343e-02  1.05865531e-01]
 [-1.22686065e-05  7.18534632e-05  5.14355609e-05 …  1.94223490e-02
  -4.19090873e-02 -4.99932146e-02]
 …
 [ 1.13194055e-05 -2.51662698e-06 -4.48272573e-05 … -1.42584625e-01
  -2.84058065e-03  2.71830211e-02]
 [-3.02459245e-05 -9.63962699e-05  5.97317182e-05 …  6.53075840e-02
  -9.43061662e-02 -1.24043606e-01]
 [ 8.85748448e-05 -1.03832848e-04 -1.40085757e-05 … -1.79766040e-02
  -6.25206780e-02 -6.37797732e-03]]
```

1.

i. The unnamed column is removed as pandas automatically provides an index so it seems redundant

ii. The dataset is split into classes and features and stored in different dataframes. It will be used in the further steps

iii. In PCA the dataset should be normalized i.e. mean = 0 and sd = 1. The dataset has been centered; each feature now has the mean of that feature subtracted resulting in a dataset where each feature has a mean of zero. This centered dataset is ready for covariance matrix computation, which is the part of the PCA process. This process is important because PCA is sensitive to the variances of the initial variances.

iv. The covariance matrix is calculated. It expresses how the variables of the dataset vary from the mean with respect to each other. It expresses the correlation between the different features in the dataset.

v. Eigenvectors and eigenvalues are crucial to PCA: eigenvectors determine the directions of the new feature space and eigenvalues determine the magnitude. The eigenvectors point in the direction of the largest variances, it is necessary as the principal components must be orthogonal. They are calculated by covariance matrix.

[198]:

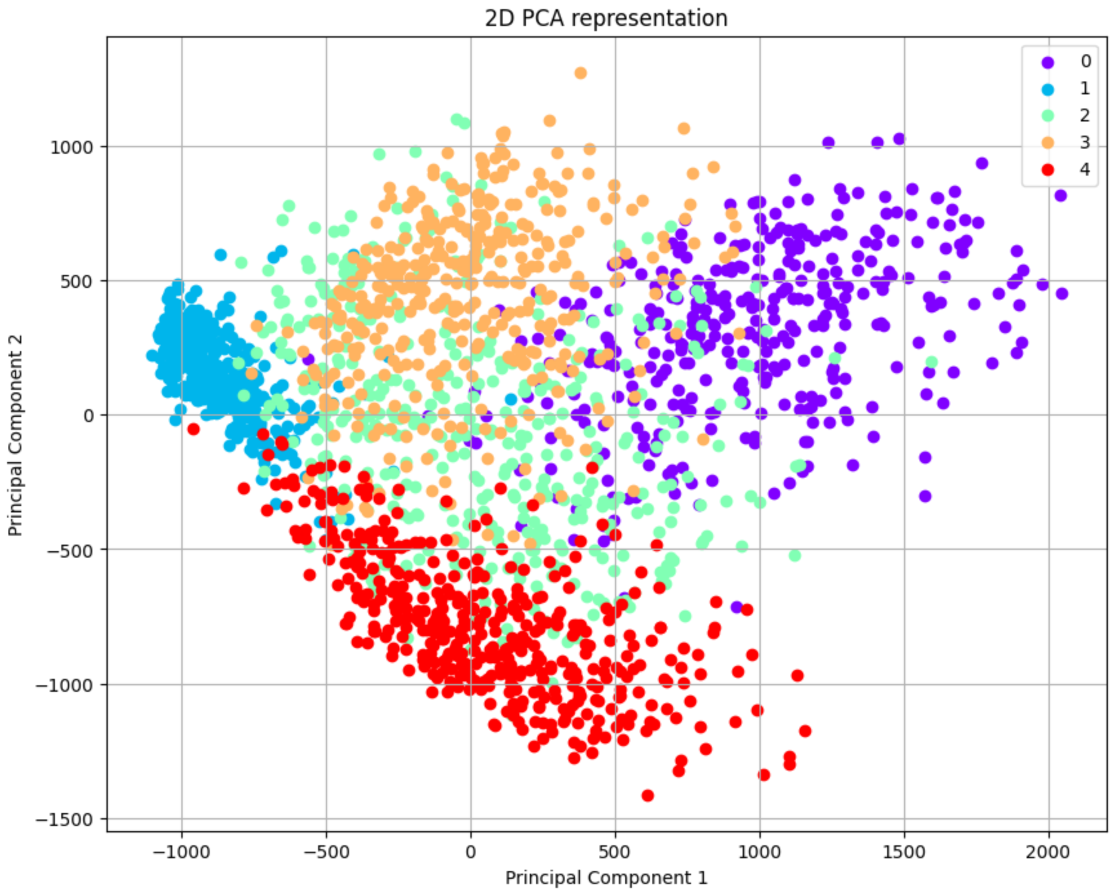
```

```

2. Plot a 2-dimensional representation of the data points based on the first and second principal components. Explain the results versus the known classes (display data points of each class with a different color)

[199]:

```
DatanewB = pd.read_csv('DataB.csv',encoding='latin-1')
eigen_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:,i]) for i in␣
 ↪range(len(eigenvalues))]
eigen_pairs.sort(key=lambda k: k[0], reverse=True)
```

```
# Extract the eigenvectors for the two largest eigenvalues
w = np.hstack((eigen_pairs[0][1].reshape(-1,1), eigen_pairs[1][1].
 ↪reshape(-1,1)))

# Transform the data onto the two principal components
X_pca = np.dot(DataB_centered, w)

# Add the principal components to the DataFrame
DatanewB['PC1'] = X_pca[:, 0]
DatanewB['PC2'] = X_pca[:, 1]

# Plot the data points with colors based on their class
unique_classes = np.unique(DatanewB['gnd'])
colors = plt.cm.rainbow(np.linspace(0, 1, len(unique_classes)))
#colors = ["#3778bf", "#ffc107", "#6c757d", "#89a894", "#6a5a8c"]
plt.figure(figsize=(10, 8))

for class_label, color in zip(unique_classes, colors):
    plt.scatter(DatanewB.loc[DatanewB['gnd'] == class_label, 'PC1'],
                DatanewB.loc[DatanewB['gnd'] == class_label, 'PC2'],
                label=class_label,
                c= [color] )

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2D PCA representation')
plt.legend()
plt.grid(True)
plt.show()
```



2.

i. The eigenvectors are then sorted by the decreasing Eigenvalues as the eigenvectors with the highest eigenvalues carry the most information about the distribution of the data.

ii. The eigenvector with the highest eigenvalue is considered the first principal component since it accounts for the most variance in the data. The second highest eigenvector is the second principal component, and so on.

iii. After sorting, we choose the top 2 eigenvectors, which allows us to reduce the dimensionality of the dataset while retaining the most significant variance.

iv. The selected eigenvectors form a new axis, and we can project the original dataset onto this new subspace. This is done by multiplying the original standardized data (mean =0 sd = 1) by the matrix composed of the selected eigenvectors

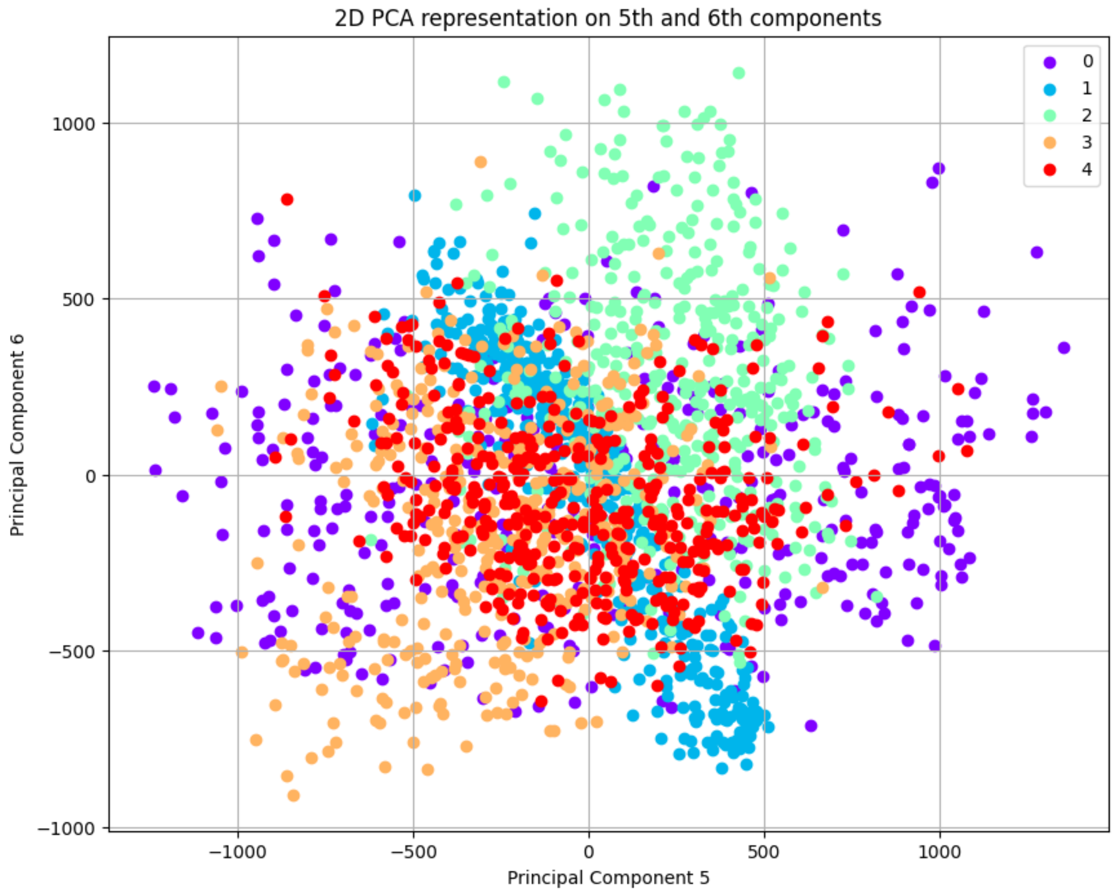
v. Now we can plot the points of the transformed data. Each point is plotted according to its values for the first and second principal components. To differentiate between known classes in the dataset, we use different colors for each class's data points.

[199]:

```
```

3. Repeat step 2 for the 5th and 6st components. Comment on the result.

[200]:
```
eigen_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:,i]) for i in␣
 ↪range(len(eigenvalues))]
eigen_pairs.sort(key=lambda k: k[0], reverse=True)

# Extract the eigenvectors for the 5th and 6th largest eigenvalues
# Note that Python uses 0-indexing, so the 5th component is at index 4 and the␣
 ↪6th at index 5.
w_56 = np.hstack((eigen_pairs[4][1].reshape(-1,1), eigen_pairs[5][1].
 ↪reshape(-1,1)))

# Transform the data onto the 5th and 6th principal components
X_pca_56 = np.dot(DataB_centered, w_56)

# Add the 5th and 6th principal components to the DataFrame
DatanewB['PC5'] = X_pca_56[:, 0]
DatanewB['PC6'] = X_pca_56[:, 1]

# Plot the data points with colors based on their class
unique_classes = np.unique(DatanewB['gnd'])
colors = plt.cm.rainbow(np.linspace(0, 1, len(unique_classes)))
#colors = ["#3778bf", "#ffc107", "#6c757d", "#89a894", "#6a5a8c"]
plt.figure(figsize=(10, 8))

for class_label, color in zip(unique_classes, colors):
    plt.scatter(DatanewB.loc[DatanewB['gnd'] == class_label, 'PC5'],
                DatanewB.loc[DatanewB['gnd'] == class_label, 'PC6'],
                label=class_label,
                c=[color])

plt.xlabel('Principal Component 5')
plt.ylabel('Principal Component 6')
plt.title('2D PCA representation on 5th and 6th components')
plt.legend()
plt.grid(True)
# plt.show()
```



3.

i. After sorting, we choose the 5th and 6th eigenvectors as given in the question to get the 5th and 6st principal component.

ii. The selected eigenvectors form a new axis, and we can project the original dataset onto this new subspace. This is done by multiplying the original standardized data (mean =0 sd = 1) by the matrix composed of the selected eigenvectors

iii. Now we can plot the points of the transformed data. Each point is plotted according to its values for the fifth and sixth principal components. To differentiate between known classes in the dataset, we use different colors for each class's data points.

[200]:

```

```

Analysis based on the above two plots.

i. From the above two plots of the data on first and second principal components and fifth and sixth components, it can be visualized that the range of the variance described by the first two components is greater than that of 5th and 6th components.

ii. Classes have maximum variance in the first two components

iii The variance decreases as the 5th and 6th components are plotted; understood from the plot.

iv The Classes look more separated when projected on Principal Components 1 and 2 compared to the projection on Principal Components 5 and 6.

[200]:

```

```

4. Use the Naive Bayes classifier to classify 8 sets of dimensionality reduced data (using the first 2, 4, 10, 30, 60, 200, 500, and all 784 PCA components). Plot the classification error for the 8 sets against the retained variance of each case.

[201]:

```
# Function to perform classification
def NaiveBayesClass(n, X, Digit_actual, eig_val, eig_vec):
    # Selecting n Principal components
    U = np.asmatrix(eig_vec[:,:n])
    # Projecting original dataset over n-components:
    Y = np.dot(U.T,X.T)
    Ret_Variance = sum(eig_val[:n])/sum(eig_val)

    df_PCA = pd.DataFrame(Y.T)
    # Split dataset into training set and test set
    X_train, X_test, y_train, y_test = train_test_split(df_PCA,
                                                        Digit_actual,␣
 ↪test_size=0.3,
                                                        random_state=109)
    # Create a Gaussian Classifier
    gnb = GaussianNB()
    # Train the model using the training sets
    gnb.fit(X_train, y_train)

    # Predict the response for train dataset
    y_train_pred = gnb.predict(X_train)

    # Predict the response for train dataset
    y_pred = gnb.predict(X_test)

    # Model Accuracy, how often is the classifier
    # correct on train set (for overfitting)?
    train_error = (1 - accuracy_score(y_train, y_train_pred))

    # Model Accuracy, how often is the classifier correct?
    error = (1 - accuracy_score(y_test, y_pred))

    return Ret_Variance, error, train_error
```

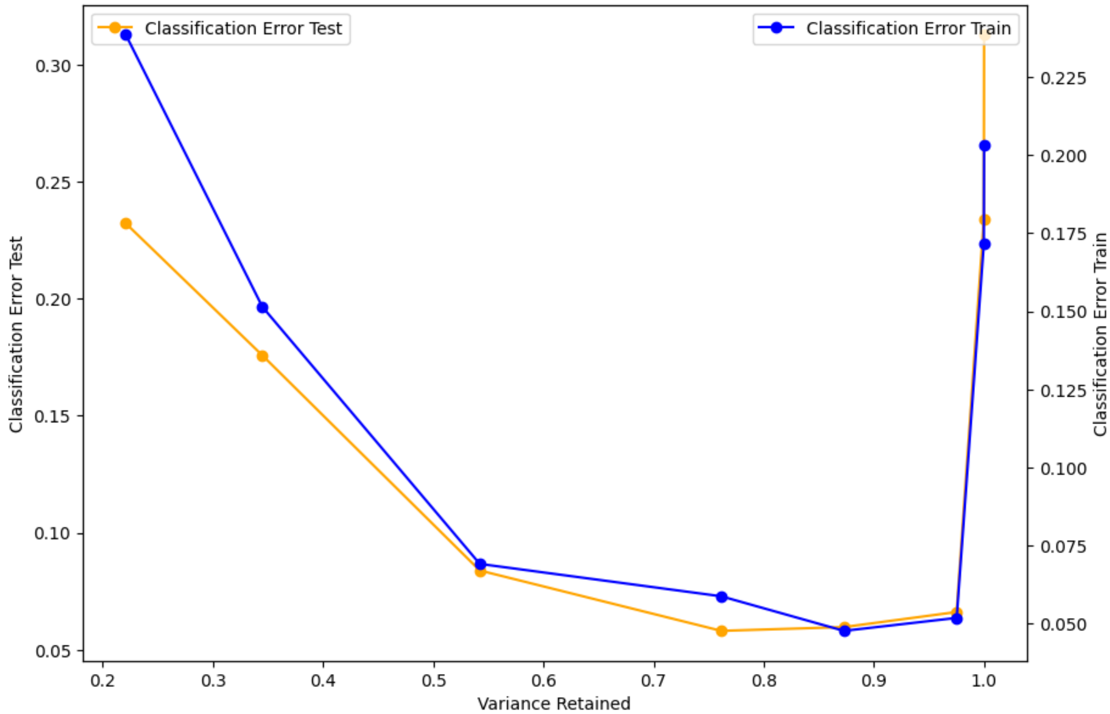
[202]:

```
def plotvar(final_metrics):
    df_final = pd.DataFrame(final_metrics,columns=['Retained Variance',
```

```
                'Classification Error Test','Classification Error Train'])

    fig, ax = plt.subplots(figsize=(10, 7))
    ax.plot(df_final['Retained Variance'], df_final['Classification Error␣
 ↪Test'],
            'o-', label = 'Classification Error Test',color = 'orange')
    ax2 = plt.twinx()
    ax2.plot(df_final['Retained Variance'], df_final['Classification Error␣
 ↪Train'],
             'o-', label = 'Classification Error Train', color = 'blue')
    ax.legend(loc=0)
    ax2.legend(loc=1)
    ax.set_xlabel("Variance Retained")
    ax.set_ylabel("Classification Error Test")
    ax2.set_ylabel("Classification Error Train")
    plt.show()
```

[203]:
```
# Initialize a list to store metrics
final_metrics = []

# Loop over the component list and perform the tasks
#component_list = [2, 4, 10, 30, 60, 200, 500, 784]
component_list = [2,4,10,30,60,200,500,784]
for components in component_list:
    Ret_Variance, error_test, error_train = NaiveBayesClass(components,␣
 ↪DataB_centered, DataB_Y, eigenvalues, eigenvectors)
    final_metrics.append({
        'Retained Variance': Ret_Variance,
        'Classification Error Test': error_test,
        'Classification Error Train': error_train
    })
```

[204]:
```
plotvar(final_metrics)
```



4.

i. As the components are increased the retained variance is increased as it is accumulated and the error is decreasing.

ii. But after some point the variance is so high that it is overfitting.

[204]:

```

```

5. As the class labels are already known, you can use the Linear Discriminant Analysis (LDA) to reduce the dimensionality, plot the data points using the first 2 LDA components (display data points of each class with a different color). Explain the results obtained in terms of the known classes. Compare with the results obtained by using PCA.

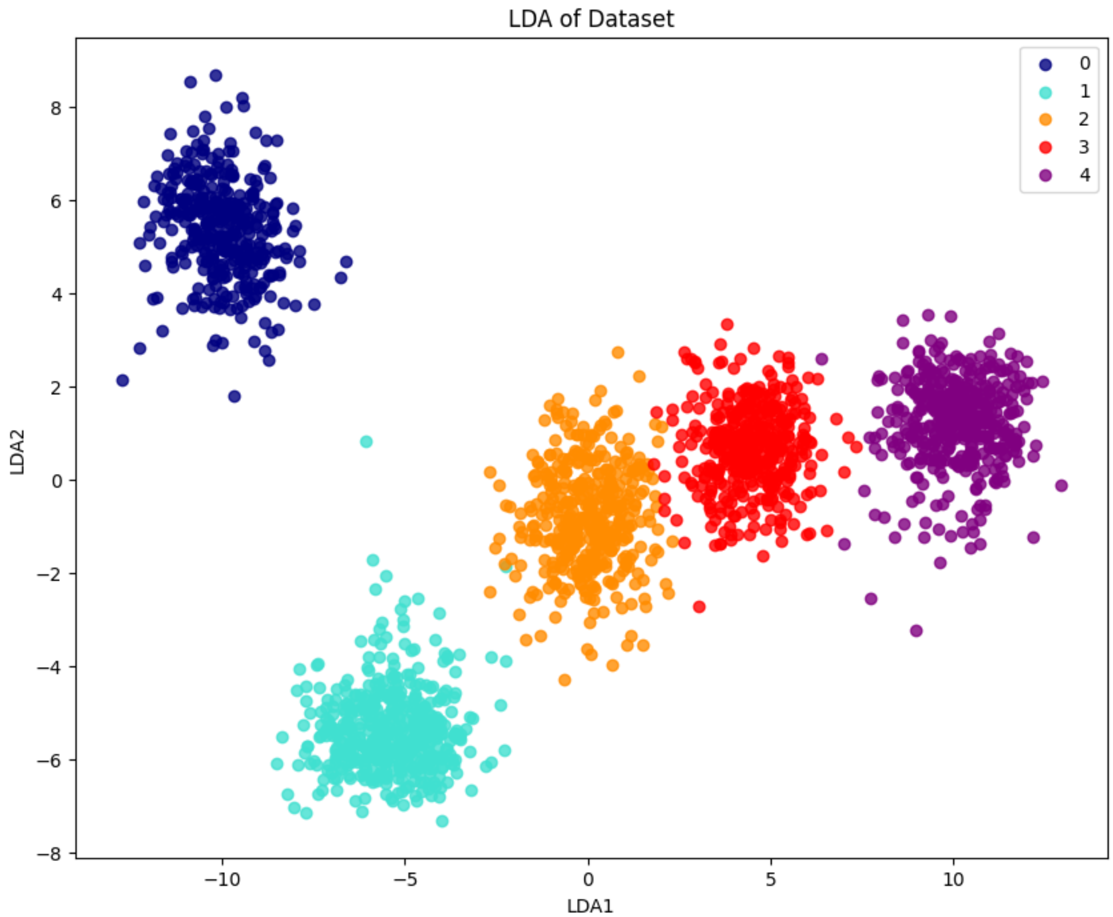
[205]:

```
lda = LDA(n_components=2)

# Fit the LDA model
#Non normalized data is passed here
scaler = StandardScaler()
X_std = scaler.fit_transform(DataB_X)
X_lda = lda.fit_transform(X_std, DataB_Y)

# Create a DataFrame for the LDA components
lda_df = pd.DataFrame(X_lda, columns=['LDA1', 'LDA2'])
```

```
# Add the class labels to the DataFrame
lda_df['class'] = DataB_Y
```

[206]:
```
colors = ['navy', 'turquoise', 'darkorange', 'red', 'purple']

# Start the plot, set figure size and title
plt.figure(figsize=(10, 8))
plt.title('LDA of Dataset')

# Plot each class
for color, i, label in zip(colors, [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]):
    plt.scatter(X_lda[DataB_Y == i, 0], X_lda[DataB_Y == i, 1], alpha=.8,␣
 ↪color=color, label=label)

plt.xlabel('LDA1')
plt.ylabel('LDA2')
plt.legend(loc='best', shadow=False, scatterpoints=1)
#plt.title('LDA of Dataset (first two components)')
```

[206]:
```
<matplotlib.legend.Legend at 0x7820d67dc550>
```



In the above LDA the data passed is not centered.

[206]:

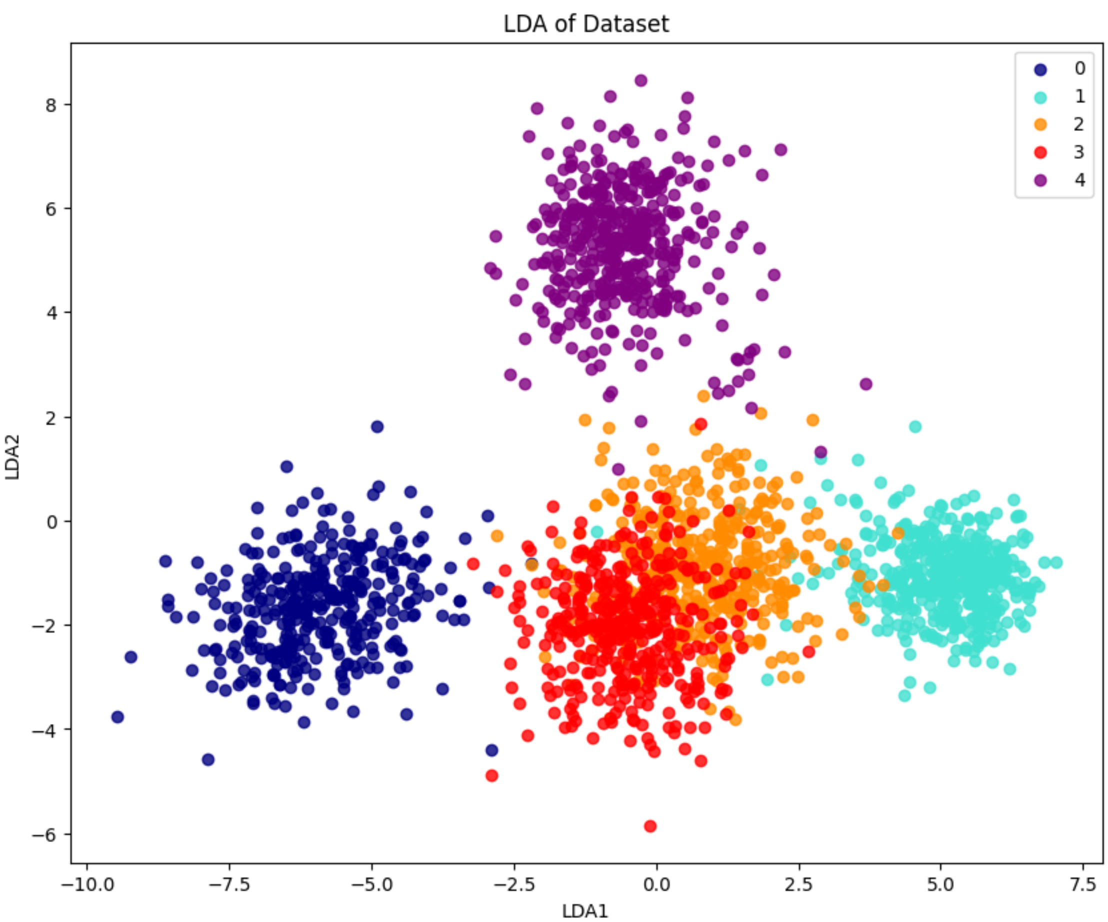
```

```

[207]:

```
lda = LDA(n_components=2)

# Fit the LDA model
#Normalized data is passed here
scaler = StandardScaler()
X_std = scaler.fit_transform(DataB_centered)
X_lda = lda.fit_transform(X_std, DataB_Y)

# Create a DataFrame for the LDA components
lda_df = pd.DataFrame(X_lda, columns=['LDA1', 'LDA2'])

# Add the class labels to the DataFrame
lda_df['class'] = DataB_Y
```

[208]:
```python
colors = ['navy', 'turquoise', 'darkorange', 'red', 'purple']

# Start the plot, set figure size and title
plt.figure(figsize=(10, 8))
plt.title('LDA of Dataset')

# Plot each class
for color, i, label in zip(colors, [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]):
    plt.scatter(X_lda[DataB_Y == i, 0], X_lda[DataB_Y == i, 1], alpha=.8,␣
 ↪color=color, label=label)

plt.xlabel('LDA1')
plt.ylabel('LDA2')
plt.legend(loc='best', shadow=False, scatterpoints=1)
#plt.title('LDA of Dataset (first two components)')
```

[208]: <matplotlib.legend.Legend at 0x7820d6882800>



In the Above the centered data is passed which was used for the PCA.

[208]:

```

```

Analysis

i. In the first LDA plot the data was not normalized which was required in PCA.

ii. The second LDA plot the normalized data was used which was used in PCA.

iii. The first plot classifies the classes better than the second one in which normalized data was passed.

iv. In comparison to PCA, LDA classified the classes better. Comparing the results of LDA and PCA, 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. So LDA outperforms PCA in lower dimensions.

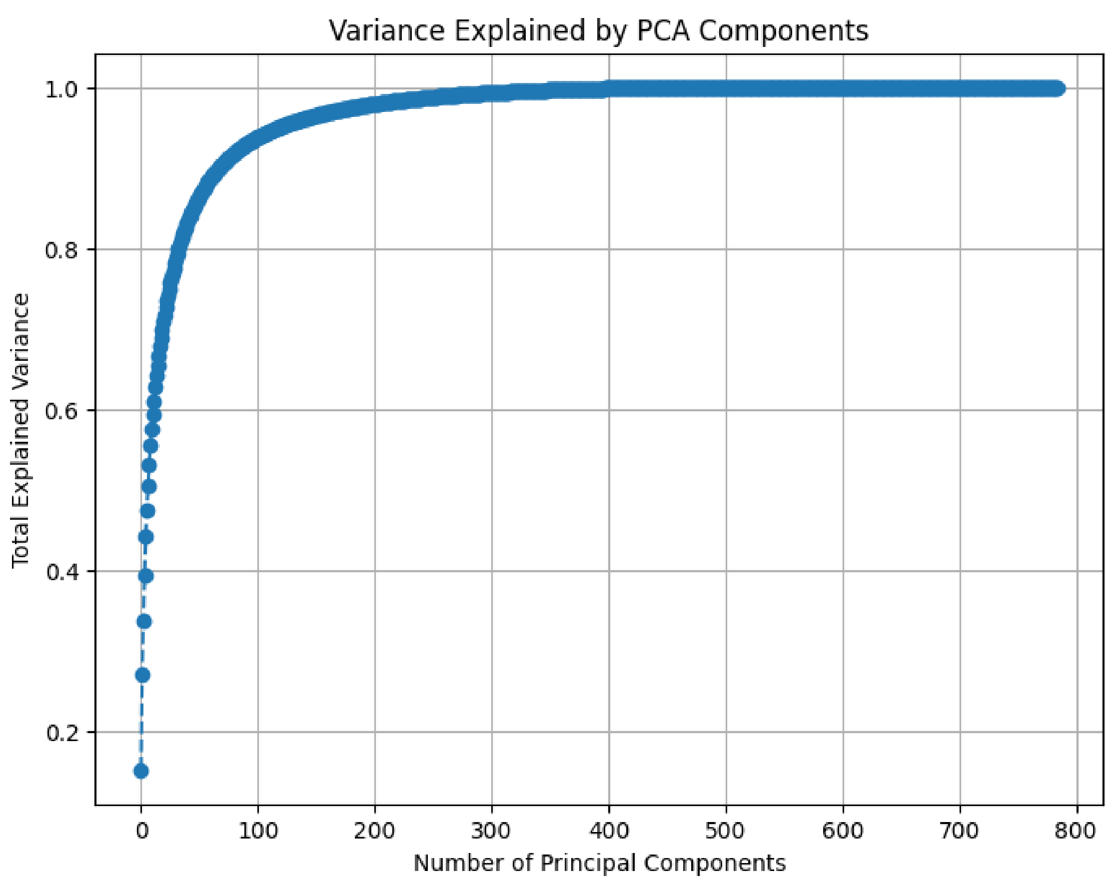
6. Prove that the PCA is the best linear method for transformation (with orthonormal bases)

[209]:

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
train_features, test_features, train_labels, test_labels = train_test_split(
    DataB_X, DataB_Y, test_size=0.3, random_state=109)

# Apply Principal Component Analysis
pca_transformer = PCA()
pca_train_features = pca_transformer.fit_transform(train_features)
pca_test_features = pca_transformer.transform(test_features)

# Apply Linear Discriminant Analysis
lda_transformer = LDA()
lda_train_features = lda_transformer.fit_transform(train_features, train_labels)
lda_test_features = lda_transformer.transform(test_features)

# Classifier using PCA features
classifier_pca = KNeighborsClassifier(n_neighbors=3)
classifier_pca.fit(pca_train_features, train_labels)
predictions_pca = classifier_pca.predict(pca_test_features)

# Classifier using LDA features
classifier_lda = KNeighborsClassifier(n_neighbors=3)
classifier_lda.fit(lda_train_features, train_labels)
predictions_lda = classifier_lda.predict(lda_test_features)

# Calculate and print the accuracies
accuracy_pca = accuracy_score(test_labels, predictions_pca)
accuracy_lda = accuracy_score(test_labels, predictions_lda)
print(f"Accuracy after PCA: {accuracy_pca*100}")
```

```python
print(f"Accuracy after LDA: {accuracy_lda*100}")

# Visualize the variance explained by PCA components
plt.figure(figsize=(8, 6))
plt.plot(range(len(pca_transformer.explained_variance_ratio_)),
         np.cumsum(pca_transformer.explained_variance_ratio_), marker='o',␣
 ↪linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Total Explained Variance')
plt.title('Variance Explained by PCA Components')
plt.grid(True)
plt.show()
```

```
Accuracy after PCA: 97.74193548387096
Accuracy after LDA: 93.38709677419355
```



Analysis:

The accuracy of PCA is higher than that of LDA. So it can be said that PCA is best linear method for transformation (with orthonormal bases)

[209]:

```

```