Machine Learning: Assignment 5

Ayesha Farhana
700735341

GitHub link: https://github.com/AishaFar/MLAssignment5_Farhana_700735341

Programming elements:
Principal Component Analysis

In class programming:

1. Principal Component Analysis
   a. Apply PCA on CC dataset.
   b. Apply k-means algorithm on the PCA result and report your observation if the silhouette score has improved or not?
   c. Perform Scaling+PCA+K-Means and report performance.

```
In [1]:
# importing required libraries for assignment 5 here
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import preprocessing, metrics
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
sns.set(style="white", color_codes=True)
import warnings
warnings.filterwarnings("ignore")
```

- importing required libraries for assignment 5 here numpy, matplotlib.pyplot, pandas, seaborn, sklearn.preprocessing, StandardScaler, LabelEncoder, sklearn.model_selection, train_test_split, sklearn.metrics, accuracy_score, classification_report, confusion_matrix, sklearn.metrics, accuracy_score, classification_report, confusion_matrix, sklearn.decomposition,PCA , KMeans and importing warnings.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from sklearn import preprocessing, metrics from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

sns.set(style="white", color_codes=True)

import warnings

warnings.filterwarnings("ignore")

```
In [2]:  # Principal Component Analysis
         # a. Apply PCA on CC dataset.
         # b. Apply k-means algorithm on the PCA result and report your observation if the silhouette score
         # has improved or not?
         # c. Perform Scaling+PCA+K-Means and report performance.

In [3]:  dataset_CC = pd.read_csv('datasets(1)/datasets/CC.csv')
         dataset_CC.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 8950 entries, 0 to 8949
         Data columns (total 18 columns):
          #   Column                            Non-Null Count  Dtype
         ---  ------                            --------------  -----
          0   CUST_ID                           8950 non-null   object
          1   BALANCE                           8950 non-null   float64
          2   BALANCE_FREQUENCY                 8950 non-null   float64
          3   PURCHASES                         8950 non-null   float64
          4   ONEOFF_PURCHASES                  8950 non-null   float64
          5   INSTALLMENTS_PURCHASES            8950 non-null   float64
          6   CASH_ADVANCE                      8950 non-null   float64
          7   PURCHASES_FREQUENCY               8950 non-null   float64
          8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
          9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
          10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
          11  CASH_ADVANCE_TRX                  8950 non-null   int64
          12  PURCHASES_TRX                     8950 non-null   int64
          13  CREDIT_LIMIT                      8949 non-null   float64
          14  PAYMENTS                          8950 non-null   float64
          15  MINIMUM_PAYMENTS                  8637 non-null   float64
          16  PRC_FULL_PAYMENT                  8950 non-null   float64
          17  TENURE                            8950 non-null   int64
         dtypes: float64(14), int64(3), object(1)
         memory usage: 1.2+ MB
```

Here it is reading csv file datasets, the path is given as 'datasets(1)/datasets/CC.csv , datasets(1) -> datasets ->CC.csv file

dataset_CC = pd.read_csv('datasets(1)/datasets/CC.csv')

dataset_CC.info()

In [4]:
```
dataset_CC.head()
```

Out[4]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQ
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	

- dataset_CC.head() it gives first 5 records

In [4]:
```
dataset_CC.head()
```

Out[4]:

PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_
0.166667	0.000000	0.083333	0.000000	
0.000000	0.000000	0.000000	0.250000	
1.000000	1.000000	0.000000	0.000000	
0.083333	0.083333	0.000000	0.083333	
0.083333	0.083333	0.000000	0.000000	

In [4]:
```
dataset_CC.head()
```

Out[4]:

CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0.000000	0	2	1000.0	201.802084	139.509787	0.000000	12
0.250000	4	0	7000.0	4103.032597	1072.340217	0.222222	12
0.000000	0	12	7500.0	622.066742	627.284787	0.000000	12
0.083333	1	1	7500.0	0.000000	NaN	0.000000	12
0.000000	0	1	1200.0	678.334763	244.791237	0.000000	12

In [5]:
```
dataset_CC.isnull().any()
```

Out[5]:
```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                         True
PAYMENTS                            False
MINIMUM_PAYMENTS                     True
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

dataset_CC.isnull().any()

It checks with CC having any null values
the output is : CUST_ID False
BALANCE False
BALANCE_FREQUENCY False
PURCHASES False

ONEOFF_PURCHASES False
INSTALLMENTS_PURCHASES False
CASH_ADVANCE False
PURCHASES_FREQUENCY False
ONEOFF_PURCHASES_FREQUENCY False
PURCHASES_INSTALLMENTS_FREQUENCY False
CASH_ADVANCE_FREQUENCY False
CASH_ADVANCE_TRX False
PURCHASES_TRX False
CREDIT_LIMIT True
PAYMENTS False
MINIMUM_PAYMENTS True
PRC_FULL_PAYMENT False
TENURE False
dtype: bool

```
In [6]:  dataset_CC.fillna(dataset_CC.mean(), inplace=True)
         dataset_CC.isnull().any()

Out[6]:  CUST_ID                             False
         BALANCE                             False
         BALANCE_FREQUENCY                   False
         PURCHASES                           False
         ONEOFF_PURCHASES                    False
         INSTALLMENTS_PURCHASES              False
         CASH_ADVANCE                        False
         PURCHASES_FREQUENCY                 False
         ONEOFF_PURCHASES_FREQUENCY          False
         PURCHASES_INSTALLMENTS_FREQUENCY    False
         CASH_ADVANCE_FREQUENCY              False
         CASH_ADVANCE_TRX                    False
         PURCHASES_TRX                       False
         CREDIT_LIMIT                        False
         PAYMENTS                            False
         MINIMUM_PAYMENTS                    False
         PRC_FULL_PAYMENT                    False
         TENURE                              False
         dtype: bool
```

dataset_CC.fillna(dataset_CC.mean(), inplace=True)

dataset_CC.isnull().any()

Output: CUST_ID False
BALANCE False
BALANCE_FREQUENCY False
PURCHASES False
ONEOFF_PURCHASES False
INSTALLMENTS_PURCHASES False
CASH_ADVANCE False
PURCHASES_FREQUENCY False
ONEOFF_PURCHASES_FREQUENCY False
PURCHASES_INSTALLMENTS_FREQUENCY False
CASH_ADVANCE_FREQUENCY False
CASH_ADVANCE_TRX False
PURCHASES_TRX False
CREDIT_LIMIT False

PAYMENTS False
MINIMUM_PAYMENTS False
PRC_FULL_PAYMENT False
TENURE False
dtype: bool

In [7]:
```
x = dataset_CC.iloc[:,1:-1]
y = dataset_CC.iloc[:,-1]
print(x.shape,y.shape)
```
```
(8950, 16) (8950,)
```

x = dataset_CC.iloc[:,1:-1]

y = dataset_CC.iloc[:,-1]

print(x.shape,y.shape)

The output:
(8950, 16) (8950,)

#1.a Apply PCA on CC Dataset

In [9]:
```
pca = PCA(3)
x_pca = pca.fit_transform(x)
principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2', 'principal component 3
finalDf = pd.concat([principalDf, dataset_CC.iloc[:,-1]], axis = 1)
finalDf.head()
```

Out[9]:

	principal component 1	principal component 2	principal component 3	TENURE
0	-4326.383979	921.566882	183.708383	12
1	4118.916665	-2432.846346	2369.969289	12
2	1497.907641	-1997.578694	-2125.631328	12
3	1394.548536	-1488.743453	-2431.799649	12
4	-3743.351896	757.342657	512.476492	12

In [9]:
```
PCA(3)
= pca.fit_transform(x)
palDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2', 'principal component 3'])
F = pd.concat([principalDf, dataset_CC.iloc[:,-1]], axis = 1)
F.head()
```

Out[9]:

	principal component 1	principal component 2	principal component 3	TENURE
0	-4326.383979	921.566882	183.708383	12
1	4118.916665	-2432.846346	2369.969289	12
2	1497.907641	-1997.578694	-2125.631328	12
3	1394.548536	-1488.743453	-2431.799649	12
4	-3743.351896	757.342657	512.476492	12

- pca = PCA(3)
- x_pca = pca.fit_transform(x)
- principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2', 'principal component 3'])
- finalDf = pd.concat([principalDf, dataset_CC.iloc[:,-1]], axis = 1)

- finalDf.head()

And output has shown above

```
In [10]:  #1.b Apply K Means on PCA Result
          X = finalDf.iloc[:,0:-1]
          y = finalDf.iloc[:,-1]
```

#1.b Apply K Means on PCA Result
X = finalDf.iloc[:,0:-1]
y = finalDf.iloc[:,-1]

```
In [11]:  nclusters = 3 # this is the k in kmeans
          km = KMeans(n_clusters=nclusters)
          km.fit(X)

          # predict the cluster for each data point
          y_cluster_kmeans = km.predict(X)


          # Summary of the predictions made by the classifier
          print(classification_report(y, y_cluster_kmeans, zero_division=1))
          print(confusion_matrix(y, y_cluster_kmeans))


          train_accuracy = accuracy_score(y, y_cluster_kmeans)
          print("\nAccuracy for our Training dataset with PCA:", train_accuracy)


          #Calculate sihouette Score
          score = metrics.silhouette_score(X, y_cluster_kmeans)
          print("Sihouette Score: ",score)

          """
          Sihouette Score- ranges from −1 to +1 , a high value indicates that the object is well matched to its own cluster and poorly
          """
```

nclusters = 3 # this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(X)

# predict the cluster for each data point
y_cluster_kmeans = km.predict(X)

# Summary of the predictions made by the classifier
print(classification_report(y, y_cluster_kmeans, zero_division=1))
print(confusion_matrix(y, y_cluster_kmeans))

train_accuracy = accuracy_score(y, y_cluster_kmeans)
print("\nAccuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X, y_cluster_kmeans)
print("Sihouette Score: ",score)

"""

Sihouette Score- ranges from −1 to +1 , a high value indicates that the object is well matched to its own cluste r and poorly matched to neighboring clusters.

"""

Output is:

```
                precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00     204.0
           7       1.00      0.00      0.00     190.0
           8       1.00      0.00      0.00     196.0
```

click to scroll output; double click to hide

```
          12       1.00      0.00      0.00    7584.0

    accuracy                           0.00    8950.0
   macro avg       0.70      0.30      0.00    8950.0
weighted avg       1.00      0.00      0.00    8950.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [ 175   28    1    0    0    0    0    0    0    0]
 [ 173   15    2    0    0    0    0    0    0    0]
 [ 169   27    0    0    0    0    0    0    0    0]
 [ 149   26    0    0    0    0    0    0    0    0]
 [ 188   47    1    0    0    0    0    0    0    0]
 [ 284   78    3    0    0    0    0    0    0    0]
 [5390 2068  126    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.5109769750121258
```

```
11]: '\nSihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and poo
rly matched to neighboring clusters.\n'
```

```
In [12]: #1.c Scaling +PCA + KMeans
         x = dataset_CC.iloc[:,1:-1]
         y = dataset_CC.iloc[:,-1]
         print(x.shape,y.shape)

         (8950, 16) (8950,)
```

#1.c Scaling +PCA + KMeans
x = dataset_CC.iloc[:,1:-1]
y = dataset_CC.iloc[:,-1]
print(x.shape,y.shape)

Output kis
(8950, 16) (8950,)

```
In [13]: #Scaling
         scaler = StandardScaler()
         scaler.fit(x)
         X_scaled_array = scaler.transform(x)
         #PCA
         pca = PCA(3)
         x_pca = pca.fit_transform(X_scaled_array)
         principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2','principal component 3'
         finalDf = pd.concat([principalDf, dataset_CC.iloc[:,-1]], axis = 1)
         finalDf.head()
```

#Scaling
scaler = StandardScaler()
scaler.fit(x)
X_scaled_array = scaler.transform(x)
#PCA
pca = PCA(3)

x_pca = pca.fit_transform(X_scaled_array)
principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2','principal component 3'])
finalDf = pd.concat([principalDf, dataset_CC.iloc[:,-1]], axis = 1)
finalDf.head()

Output:

Out[13]:

	principal component 1	principal component 2	principal component 3	TENURE
0	-1.718893	-1.072940	0.535693	12
1	-1.169306	2.509311	0.627766	12
2	0.938414	-0.382589	0.161475	12
3	-0.907502	0.045856	1.521627	12
4	-1.637830	-0.684974	0.425735	12

In [14]:
```
X = finalDf.iloc[:,0:-1]
y = finalDf["TENURE"]
print(X.shape,y.shape)
```

```
(8950, 3) (8950,)
```

In [15]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.34,random_state=0)
nclusters = 3
# this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(X_train,y_train)


# predict the cluster for each training data point
y_clus_train = km.predict(X_train)

# Summary of the predictions made by the classifier
print(classification_report(y_train, y_clus_train, zero_division=1))
print(confusion_matrix(y_train, y_clus_train))

train_accuracy = accuracy_score(y_train, y_clus_train)
print("Accuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_train, y_clus_train)
print("Sihouette Score: ",score)

"""
Sihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and poorly
"""
```

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00     139.0
           7       1.00      0.00      0.00     135.0
           8       1.00      0.00      0.00     128.0
           9       1.00      0.00      0.00     118.0
          10       1.00      0.00      0.00     151.0
          11       1.00      0.00      0.00     262.0
          12       1.00      0.00      0.00    4974.0

    accuracy                           0.00    5907.0
   macro avg       0.70      0.30      0.00    5907.0
weighted avg       1.00      0.00      0.00    5907.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [  30    4  105    0    0    0    0    0    0    0]
 [  26    1  108    0    0    0    0    0    0    0]
 [  28    4   96    0    0    0    0    0    0    0]
 [  27    2   89    0    0    0    0    0    0    0]
 [  38    6  107    0    0    0    0    0    0    0]
 [  66   11  185    0    0    0    0    0    0    0]
 [ 842  735 3397    0    0    0    0    0    0    0]]
Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.3814042016392309
```

Out[15]:
```
'\nSihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.\n'
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.34,random_state=0)
nclusters = 3
# this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(X_train,y_train)


# predict the cluster for each training data point
y_clus_train = km.predict(X_train)

# Summary of the predictions made by the classifier
print(classification_report(y_train, y_clus_train, zero_division=1))
print(confusion_matrix(y_train, y_clus_train))

train_accuracy = accuracy_score(y_train, y_clus_train)
print("Accuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_train, y_clus_train)
print("Sihouette Score: ",score)

"""
Sihouette Score- ranges from −1 to +1 , a high value indicates that the object is well matched to its own cluste
r and poorly matched to neighboring clusters.
"""
```

In [16]:

```python
# predict the cluster for each testing data point
y_clus_test = km.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_clus_test, zero_division=1))
print(confusion_matrix(y_test, y_clus_test))

train_accuracy = accuracy_score(y_test, y_clus_test)
print("\nAccuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_test, y_clus_test)
print("Sihouette Score: ",score)

"""
Sihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and poorly
"""
```

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00      65.0
           7       1.00      0.00      0.00      55.0
           8       1.00      0.00      0.00      68.0
           9       1.00      0.00      0.00      57.0
          10       1.00      0.00      0.00      85.0
          11       1.00      0.00      0.00     103.0
          12       1.00      0.00      0.00    2610.0

    accuracy                           0.00    3043.0
   macro avg       0.70      0.30      0.00    3043.0
weighted avg       1.00      0.00      0.00    3043.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [  21    3   41    0    0    0    0    0    0    0]
 [  12    0   43    0    0    0    0    0    0    0]
 [  10    1   57    0    0    0    0    0    0    0]
 [  22    0   35    0    0    0    0    0    0    0]
 [  17    5   63    0    0    0    0    0    0    0]
 [  30    4   69    0    0    0    0    0    0    0]
 [ 450  395 1765    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.3836430123932328
```

Out[16]:

```
'\nSihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and poo
rly matched to neighboring clusters.\n'
```


# predict the cluster for each testing data point
y_clus_test = km.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_clus_test, zero_division=1))
print(confusion_matrix(y_test, y_clus_test))

train_accuracy = accuracy_score(y_test, y_clus_test)
print("\nAccuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_test, y_clus_test)
print("Sihouette Score: ",score)

"""
Sihouette Score- ranges from −1 to +1 , a high value indicates that the object is well matched to its own cluste
r and poorly matched to neighboring clusters.
"""

```
In [17]: # Use pd_speech_features.csv
         # a. Perform Scaling
         # b. Apply PCA (k=3)
         # c. Use SVM to report performance
```

```
In [19]: dataset_pd = pd.read_csv('datasets(1)/datasets/pd_speech_features.csv')
         dataset_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Columns: 755 entries, id to class
dtypes: float64(749), int64(6)
memory usage: 4.4 MB
```

```
In [20]: dataset_pd.head()
```

Out[20]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisValue_dec_2
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.562
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.558
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	1.564
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...	3.780
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...	6.172

5 rows × 755 columns

dataset_pd = pd.read_csv('datasets(1)/datasets/pd_speech_features.csv')
dataset_pd.info()
dataset_pd.head()

```
In [21]: dataset_pd.isnull().any()

Out[21]: id                            False
         gender                        False
         PPE                           False
         DFA                           False
         RPDE                          False
                                        ...
         tqwt_kurtosisValue_dec_33     False
         tqwt_kurtosisValue_dec_34     False
         tqwt_kurtosisValue_dec_35     False
         tqwt_kurtosisValue_dec_36     False
         class                         False
         Length: 755, dtype: bool
```

dataset_pd.isnull().any()

In [22]:
```
X = dataset_pd.drop('class',axis=1).values
y = dataset_pd['class'].values
```

In [23]:
```
#Scaling Data
scaler = StandardScaler()
X_Scale = scaler.fit_transform(X)
```

In [24]:
```
# Apply PCA with k =3
pca3 = PCA(n_components=3)
principalComponents = pca3.fit_transform(X_Scale)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2','Principa

finalDf = pd.concat([principalDf, dataset_pd[['class']]], axis = 1)
finalDf.head()
```

Out[24]:

	principal component 1	principal component 2	Principal Component 3	class
0	-10.047372	1.471074	-6.846412	1
1	-10.637725	1.583748	-6.830978	1
2	-13.516185	-1.253543	-6.818692	1
3	-9.155084	8.833590	15.290847	1
4	-6.764470	4.611457	15.637058	1

X = dataset_pd.drop('class',axis=1).values
y = dataset_pd['class'].values

#Scaling Data
scaler = StandardScaler()
X_Scale = scaler.fit_transform(X)

# Apply PCA with k =3
pca3 = PCA(n_components=3)
principalComponents = pca3.fit_transform(X_Scale)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal comp
onent 2','Principal Component 3'])

finalDf = pd.concat([principalDf, dataset_pd[['class']]], axis = 1)
finalDf.head()

```
In [25]: X = finalDf.drop('class',axis=1).values
         y = finalDf['class'].values
         X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.34,random_state=0)
```

```
In [26]: #2.c Support Vector Machine's

         from sklearn.svm import SVC

         svmClassifier = SVC()
         svmClassifier.fit(X_train, y_train)

         y_pred = svmClassifier.predict(X_test)

         # Summary of the predictions made by the classifier
         print(classification_report(y_test, y_pred, zero_division=1))
         print(confusion_matrix(y_test, y_pred))
         # Accuracy score
         glass_acc_svc = accuracy_score(y_pred,y_test)
         print('accuracy is',glass_acc_svc )

         #Calculate sihouette Score
         score = metrics.silhouette_score(X_test, y_pred)
         print("Sihouette Score: ",score)
```

```
              precision    recall  f1-score   support

           0       0.67      0.42      0.51        62
           1       0.84      0.93      0.88       196

    accuracy                           0.81       258
   macro avg       0.75      0.68      0.70       258
weighted avg       0.80      0.81      0.79       258

[[ 26  36]
 [ 13 183]]
accuracy is 0.810077519379845
Sihouette Score:  0.25044634287068235
```

```
X = finalDf.drop('class',axis=1).values
y = finalDf['class'].values
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.34,random_state=0)

#2.c Support Vector Machine's

from sklearn.svm import SVC

svmClassifier = SVC()
svmClassifier.fit(X_train, y_train)

y_pred = svmClassifier.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred, zero_division=1))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
glass_acc_svc = accuracy_score(y_pred,y_test)
print('accuracy is',glass_acc_svc )

#Calculate sihouette Score
score = metrics.silhouette_score(X_test, y_pred)
print("Sihouette Score: ",score)
```

```
In [27]: #3.Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data to k=2.
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         dataset_iris = pd.read_csv('datasets(1)/datasets/Iris.csv')
         dataset_iris.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 150 entries, 0 to 149
         Data columns (total 6 columns):
          #   Column         Non-Null Count  Dtype
         ---  ------         --------------  -----
          0   Id             150 non-null    int64
          1   SepalLengthCm  150 non-null    float64
          2   SepalWidthCm   150 non-null    float64
          3   PetalLengthCm  150 non-null    float64
          4   PetalWidthCm   150 non-null    float64
          5   Species        150 non-null    object
         dtypes: float64(4), int64(1), object(1)
         memory usage: 7.2+ KB
```

```
In [28]: dataset_iris.isnull().any()

Out[28]: Id               False
         SepalLengthCm    False
         SepalWidthCm     False
         PetalLengthCm    False
         PetalWidthCm     False
         Species          False
         dtype: bool
```

```
In [29]: x = dataset_iris.iloc[:,1:-1]
         y = dataset_iris.iloc[:,-1]
         print(x.shape,y.shape)

         (150, 4) (150,)
```

```
In [30]: X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
In [31]: sc = StandardScaler()
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
         le = LabelEncoder()
         y = le.fit_transform(y)
```

```
In [32]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
         lda = LDA(n_components=2)
         X_train = lda.fit_transform(X_train, y_train)
         X_test = lda.transform(X_test)
         print(X_train.shape,X_test.shape)

         (105, 2) (45, 2)
```


```
#3.Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data to k=2.
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
dataset_iris = pd.read_csv('datasets(1)/datasets/Iris.csv')
dataset_iris.info()

dataset_iris.isnull().any()

x = dataset_iris.iloc[:,1:-1]
y = dataset_iris.iloc[:,-1]
print(x.shape,y.shape)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
le = LabelEncoder()
y = le.fit_transform(y)
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
print(X_train.shape,X_test.shape)
```

```
In [33]: #4. Briefly identify the difference between PCA and LDA

         """Both LDA and PCA rely on linear transformations and aim to maximize the variance in a lower dimension. PCA is an unsupervi

Out[33]: 'Both LDA and PCA rely on linear transformations and aim to maximize the variance in a lower dimension. PCA is an unsupervis
         ed learning algorithm while LDA is a supervised learning algorithm. This means that PCA finds directions of maximum variance
         regardless of class labels while LDA finds directions of maximum class separability'
```

#4. Briefly identify the difference between PCA and LDA

"""Both LDA and PCA rely on linear transformations and aim to maximize the variance in a lower dimension. PCA is an unsupervised learning algorithm while LDA is a supervised learning algorithm. This means that PCA finds directions of maximum variance regardless of class labels while LDA finds directions of maximum class separability"""

```
In [34]: #PCA
         """It reduces the features into a smaller subset of orthogonal variables, called principal components - linear combinations c

Out[34]: 'It reduces the features into a smaller subset of orthogonal variables, called principal components - linear combinations of
         the original variables. The first component captures the largest variability of the data, while the second captures the seco
         nd largest, and so on.'
```

#PCA
"""It reduces the features into a smaller subset of orthogonal variables, called principal components – linear combinations of the original variables. The first component captures the largest variability of the data, while the second captures the second largest, and so on."""

Output:
'It reduces the features into a smaller subset of orthogonal variables, called principal components – linear combinations of the original variables. The first component captures the largest variability of the data, while the second captures the second largest, and so on.'

```
In [35]: #LDA
         """LDA finds the linear discriminants in order to maximize the variance between the different categories while minimizing the

Out[35]: 'LDA finds the linear discriminants in order to maximize the variance between the different categories while minimizing the
         variance within the class.'
```

#LDA
"""LDA finds the linear discriminants in order to maximize the variance between the different categories while minimizing the variance within the class."""

Output:
'LDA finds the linear discriminants in order to maximize the variance between the different categories while minimizing the variance within the class.'