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

Sihouette 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

```python
# importing required libraries
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
# Importing PCA from scikit learn
from sklearn.decomposition import PCA
# Importing K-means from scikit learn
from sklearn.cluster import KMeans
sns.set(style="white", color_codes=True)
import warnings
warnings.filterwarnings("ignore")
```

```python
#importing cc dataset and reading all the data from dataset
dataset_CC = pd.read_csv('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
```

```python
dataset_CC.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUEI
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.00
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.00

2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333

[5]:
```
#checking if datset values has null values or not
dataset_CC.isnull().any()
```

[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
```

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

[6]:
```
CUST_ID                           False
,BALANCE                          False
,BALANCE_FREQUENCY                False
```

[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
```

[7]:
```
s = dataset_CC.iloc[:,1:-1]
t = dataset_CC.iloc[:,-1]
print(s.shape,t.shape)
```

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

[9]:
```
#1.a Applying PCA on CC Dataset
pca = PCA(3)
s_pca = pca.fit_transform(s)
principalDf = pd.DataFrame(data = s_pca, columns = ['principal component 1', 'principal component 2', 'principal component 3'])
finalDf = pd.concat([principalDf, dataset_CC.iloc[:,-1]], axis = 1)
finalDf.head()
```

[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

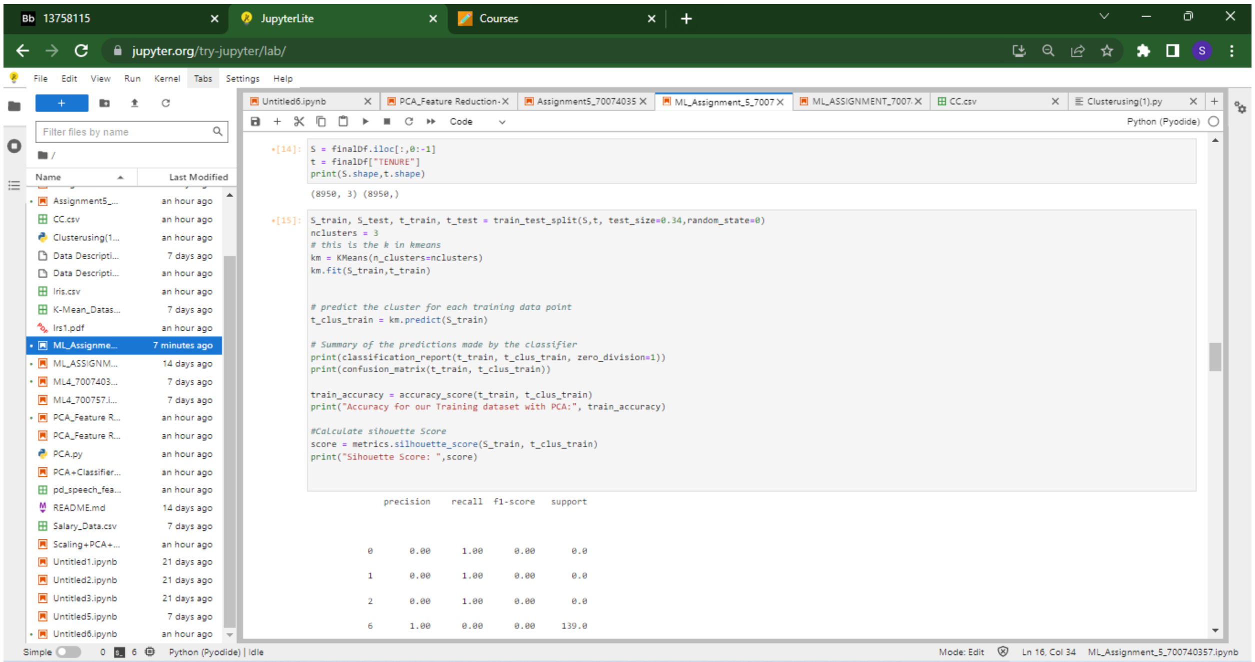
3	1394.548536	-1488.743453	-2431.799649	12
4	-3743.351896	757.342657	512.476492	12

[10]:
```
#1.b Applying K Means on PCA Result
S = finalDf.iloc[:,0:-1]
t = finalDf.iloc[:,-1]
```

[11]:
```
nclusters = 3 # this is the k in kmeans i.e.., 3
km = KMeans(n_clusters=nclusters)
km.fit(S)

# predict the cluster for each data point
t_cluster_kmeans = km.predict(S)


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


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


#Calculating sihouette Score
score = metrics.silhouette_score(S, t_cluster_kmeans)
print("Sihouette Score: ",score)
```

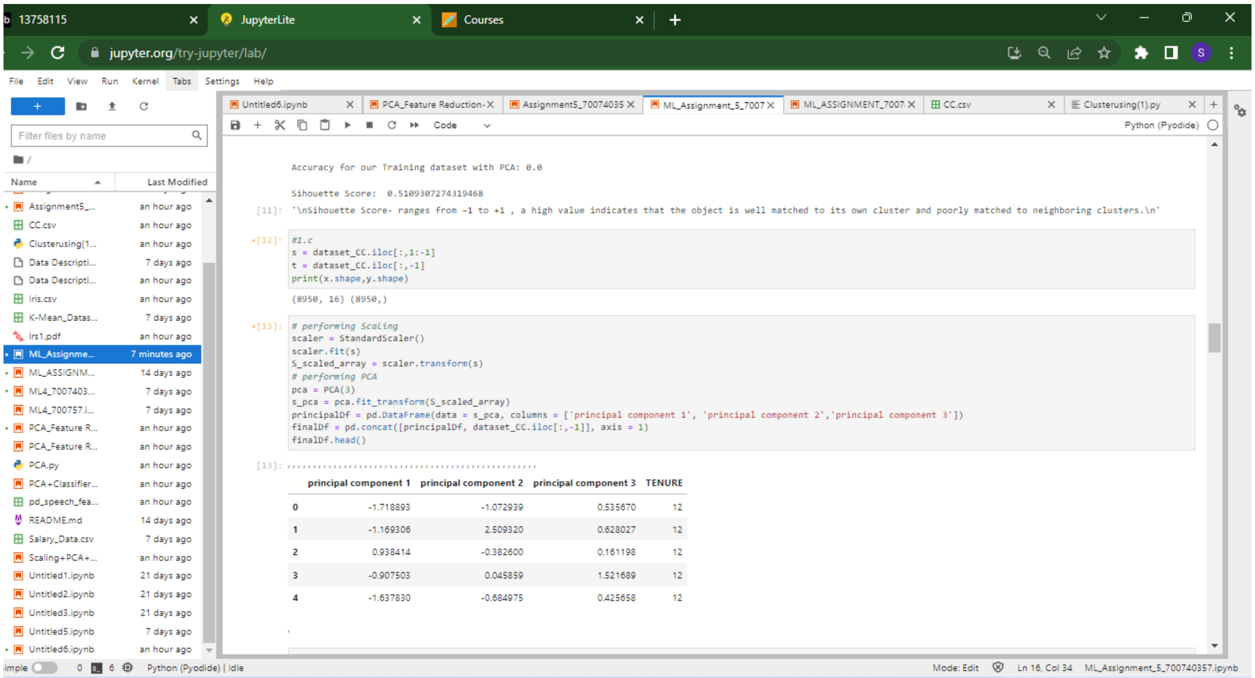
```
              precision    recall  f1-score   support

           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

           9       1.00      0.00      0.00     175.0

          10       1.00      0.00      0.00     236.0

          11       1.00      0.00      0.00     365.0

          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    1   28    0    0    0    0    0    0    0]

 [ 173    2   15    0    0    0    0    0    0    0]

 [ 169    0   27    0    0    0    0    0    0    0]

 [ 149    0   26    0    0    0    0    0    0    0]
```



```
[14]: S = finalDf.iloc[:,0:-1]
      t = finalDf["TENURE"]
      print(S.shape,t.shape)

      (8950, 3) (8950,)

[15]: S_train, S_test, t_train, t_test = train_test_split(S,t, test_size=0.34,random_state=0)
      nclusters = 3
      # this is the k in kmeans
      km = KMeans(n_clusters=nclusters)
      km.fit(S_train,t_train)


      # predict the cluster for each training data point
      t_clus_train = km.predict(S_train)

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

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

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

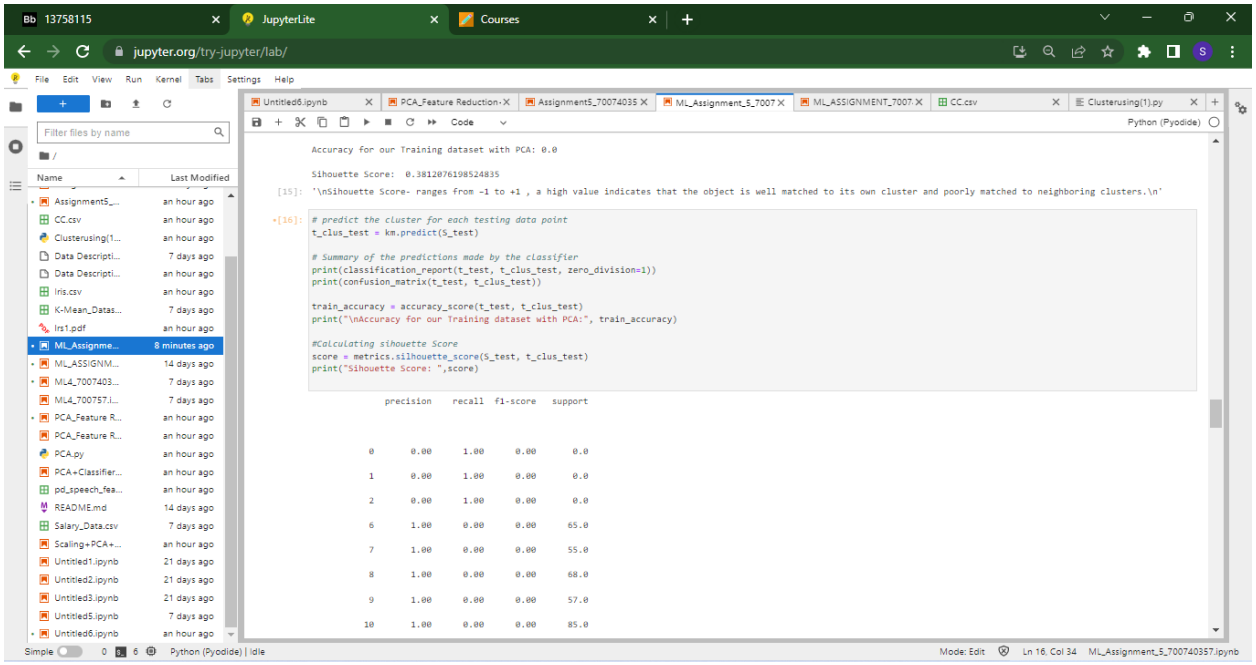
```
              precision    recall  f1-score   support


           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
```

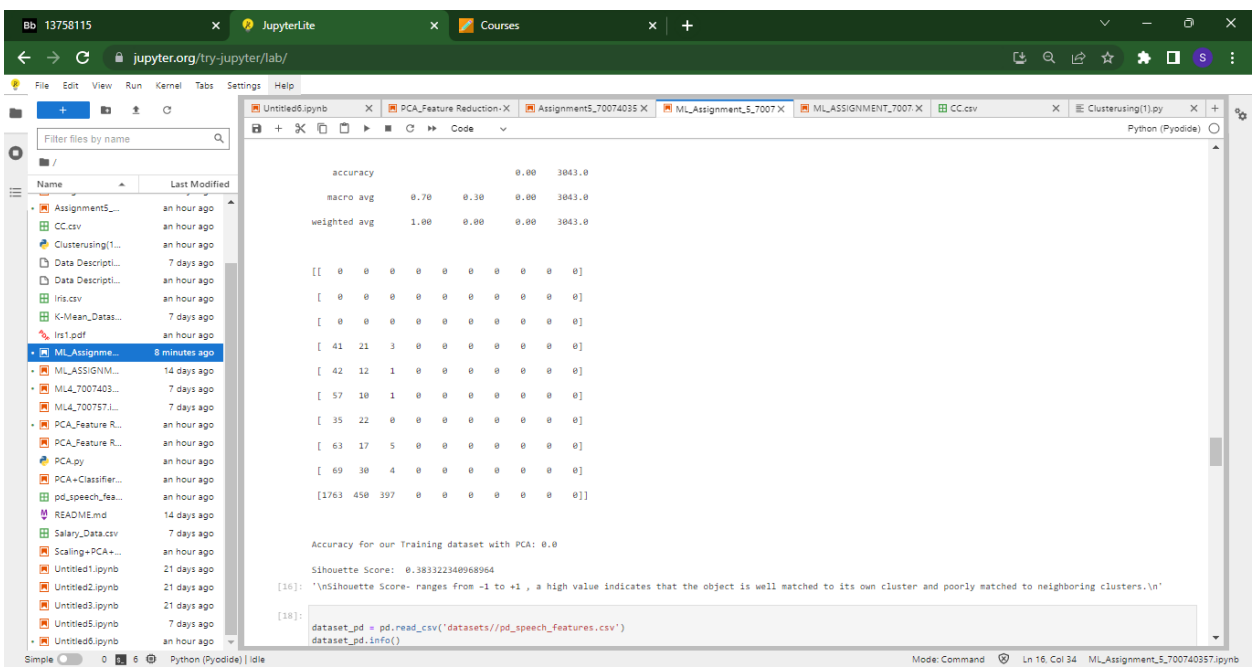


```
      Accuracy for our Training dataset with PCA: 0.0

      Sihouette Score:  0.5109307274319468
[11]: '\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'

[12]: #1.c
      s = dataset_CC.iloc[:,1:-1]
      t = dataset_CC.iloc[:,-1]
      print(x.shape,y.shape)

      (8950, 16) (8950,)

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

[13]:
```

	principal component 1	principal component 2	principal component 3	TENURE
0	-1.718893	-1.072939	0.535670	12
1	-1.169306	2.509320	0.628027	12
2	0.938414	-0.382600	0.161198	12
3	-0.907503	0.045859	1.521689	12
4	-1.637830	-0.684975	0.425658	12

```
           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]
 [ 105   30    4    0    0    0    0    0    0    0]
 [ 108   26    1    0    0    0    0    0    0    0]
 [  96   28    4    0    0    0    0    0    0    0]
 [  89   27    2    0    0    0    0    0    0    0]
 [ 107   38    6    0    0    0    0    0    0    0]
 [ 185   66   11    0    0    0    0    0    0    0]
```





2. Use pd_speech_features.csv

a. Perform Scaling

b. Apply PCA (k=3)

c. Use SVM to report performance

```
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]
 [  41   21    3    0    0    0    0    0    0    0]
 [  42   12    1    0    0    0    0    0    0    0]
 [  57   10    1    0    0    0    0    0    0    0]
 [  35   22    0    0    0    0    0    0    0    0]
 [  63   17    5    0    0    0    0    0    0    0]
 [  69   30    4    0    0    0    0    0    0    0]
 [1763  450  397    0    0    0    0    0    0    0]]


Accuracy for our Training dataset with PCA: 0.0

Sihouette Score:  0.383322340968964
```

[16]: '\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'

[18]:
```
dataset_pd = pd.read_csv('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
```

[19]:
```
dataset_pd.head()
```

[19]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisValue_dec_28	tqwt_kurtosisValue_dec_29	tqv
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.5620	2.6445	
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.5589	3.6107	
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	1.5643	2.3308	
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...	3.7805	3.5664	
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...	6.1727	5.8416	

5 rows × 755 columns

[20]:
```
dataset_pd.isnull().any()
```

[20]:
```
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
```

*[21]:
```
S = dataset_pd.drop('class',axis=1).values
t = dataset_pd['class'].values
```

*[22]:
```
#Scaling Dataset
scaler = StandardScaler()
S_Scale = scaler.fit_transform(S)
```

*[23]:
```
# Applying PCA with k =3
pca3 = PCA(n_components=3)
principalComponents = pca3.fit_transform(S_Scale)

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

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

[23]:

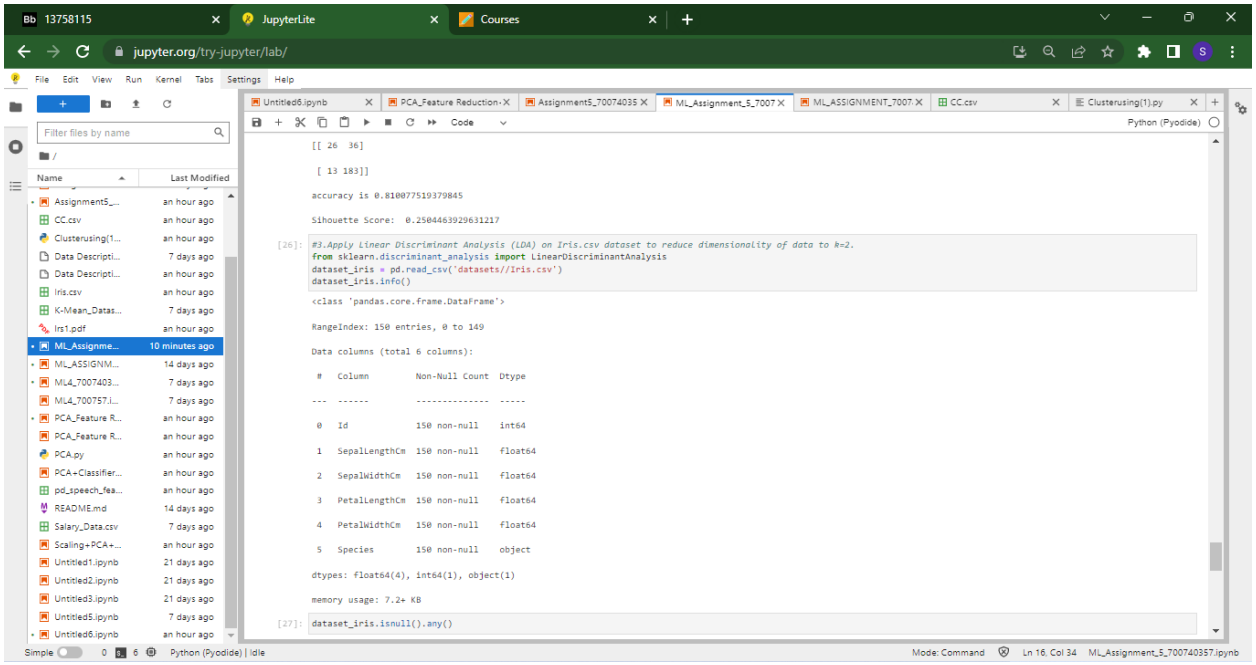
	principal component 1	principal component 2	Principal Component 3	class
0	-10.047372	1.471076	-6.846402	1
1	-10.637725	1.583749	-6.830976	1
2	-13.516185	-1.253542	-6.818696	1
3	-9.155084	8.833601	15.290906	1
4	-6.764470	4.611468	15.637121	1

*[24]:
```
S = finalDf.drop('class',axis=1).values
```

```
[24]: S = finalDf.drop('class',axis=1).values
      t = finalDf['class'].values
      S_train, S_test, t_train, t_test = train_test_split(S,t, test_size=0.34,random_state=0)

[25]: #2.c Support Vector Machine's

      from sklearn.svm import SVC

      svmClassifier = SVC()
      svmClassifier.fit(S_train, t_train)

      t_pred = svmClassifier.predict(S_test)

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

      #Calculate sihouette Score
      score = metrics.silhouette_score(S_test, t_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:  0.2504463929631217

[26]: #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//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

[27]: dataset_iris.isnull().any()
```

3. Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data to k=2.

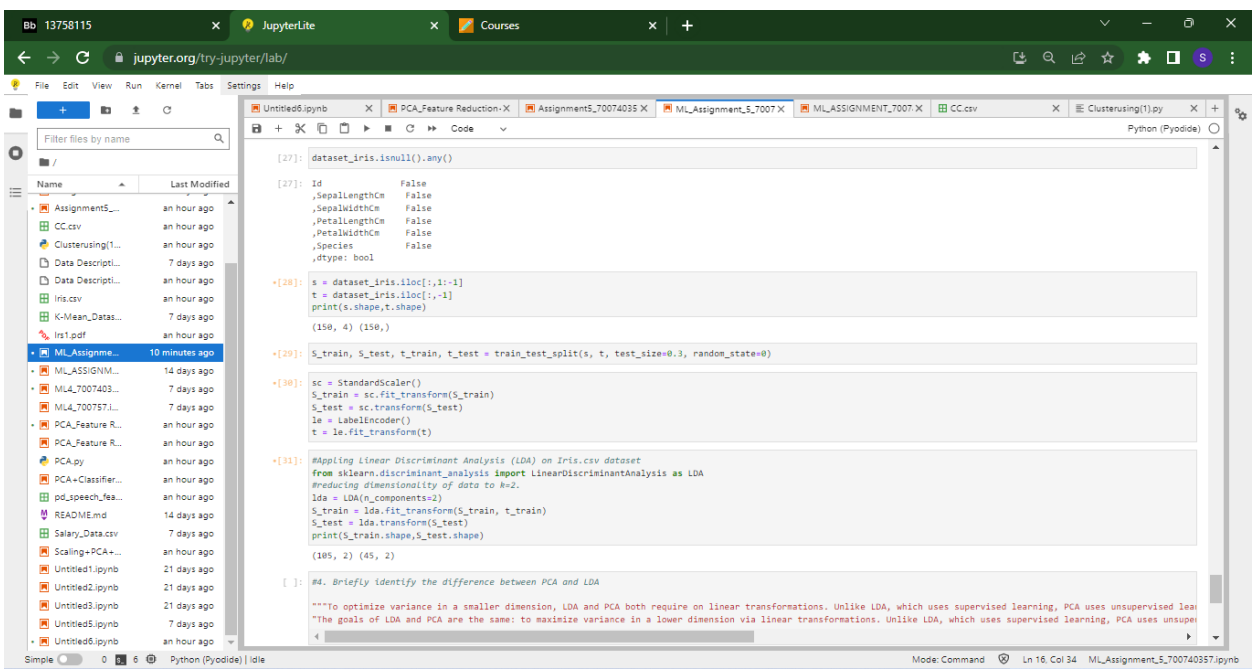


```
[[ 26  36]
 [ 13 183]]

accuracy is 0.810077519379845

Sihouette Score:  0.2504463929631217
```

```
[26]: #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//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
```

```
[27]: dataset_iris.isnull().any()
```



```
[27]: dataset_iris.isnull().any()
```

```
[27]: Id             False
,SepalLengthCm    False
,SepalWidthCm     False
,PetalLengthCm    False
,PetalWidthCm     False
,Species          False
,dtype: bool
```

```
[28]: s = dataset_iris.iloc[:,1:-1]
t = dataset_iris.iloc[:,-1]
print(s.shape,t.shape)
```

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

```
[29]: S_train, S_test, t_train, t_test = train_test_split(s, t, test_size=0.3, random_state=0)
```

```
[30]: sc = StandardScaler()
S_train = sc.fit_transform(S_train)
S_test = sc.transform(S_test)
le = LabelEncoder()
t = le.fit_transform(t)
```

```
[31]: #Appling Linear Discriminant Analysis (LDA) on Iris.csv dataset
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
#reducing dimensionality of data to k=2.
lda = LDA(n_components=2)
S_train = lda.fit_transform(S_train, t_train)
S_test = lda.transform(S_test)
print(S_train.shape,S_test.shape)
```

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

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

"""To optimize variance in a smaller dimension, LDA and PCA both require on linear transformations. Unlike LDA, which uses supervised learning, PCA uses unsupervised lea
"The goals of LDA and PCA are the same: to maximize variance in a lower dimension via linear transformations. Unlike LDA, which uses supervised learning, PCA uses unsuper
```

4. Briefly identify the difference between PCA and LDA?

"""To optimize variance in a smaller dimension, LDA and PCA both require on linear transformations. Unlike LDA, which uses supervised learning, PCA uses unsupervised learning. Accordingly, LDA discovers directions of maximal class separability while PCA discovers directions of maximum variance irrespective of class labels."
"The goals of LDA and PCA are the same: to maximize variance in a lower dimension via linear transformations. Unlike LDA, which uses supervised learning, PCA uses unsupervised learning.

This indicates that although LDA identifies pathways of maximum class separability, PCA finds directions of maximum variance regardless of class labels.