



Question 1:

 jupyter assignment5 Last Checkpoint: an hour ago (autosaved)  Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

```
In [2]: dataset = pd.read_csv('cc.csv') #importing data
```

```
In [3]: dataset.head()
```

```
Out[3]:
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083

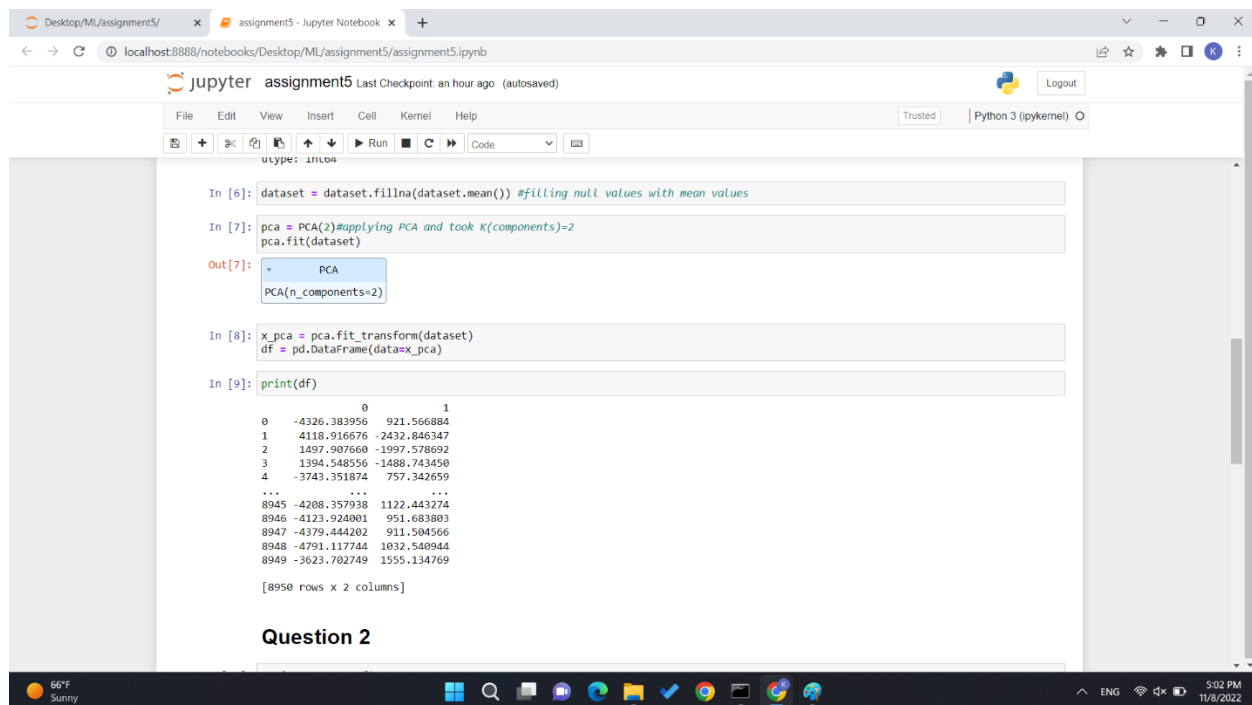
```
In [4]: #dropping the unwanted data
dataset = dataset.drop("CUST_ID", axis=1)
```

```
In [5]: #checking for Null values
dataset.isnull().sum()
```

```
Out[5]:
```

BALANCE	0
BALANCE_FREQUENCY	0
PURCHASES	0
ONEOFF_PURCHASES	0
INSTALLMENTS_PURCHASES	0

First, I have imported the required libraries then imported the dataset "cc.csv". After that displayed the first few rows of the dataset. I have dropped the CUST_ID column from the dataset. And then checked if there are any null values in the dataset.



```
In [6]: dataset = dataset.fillna(dataset.mean()) #filling null values with mean values

In [7]: pca = PCA(2)#applying PCA and took K(components)=2
pca.fit(dataset)

Out[7]:
PCA(n_components=2)

In [8]: x_pca = pca.fit_transform(dataset)
df = pd.DataFrame(data=x_pca)

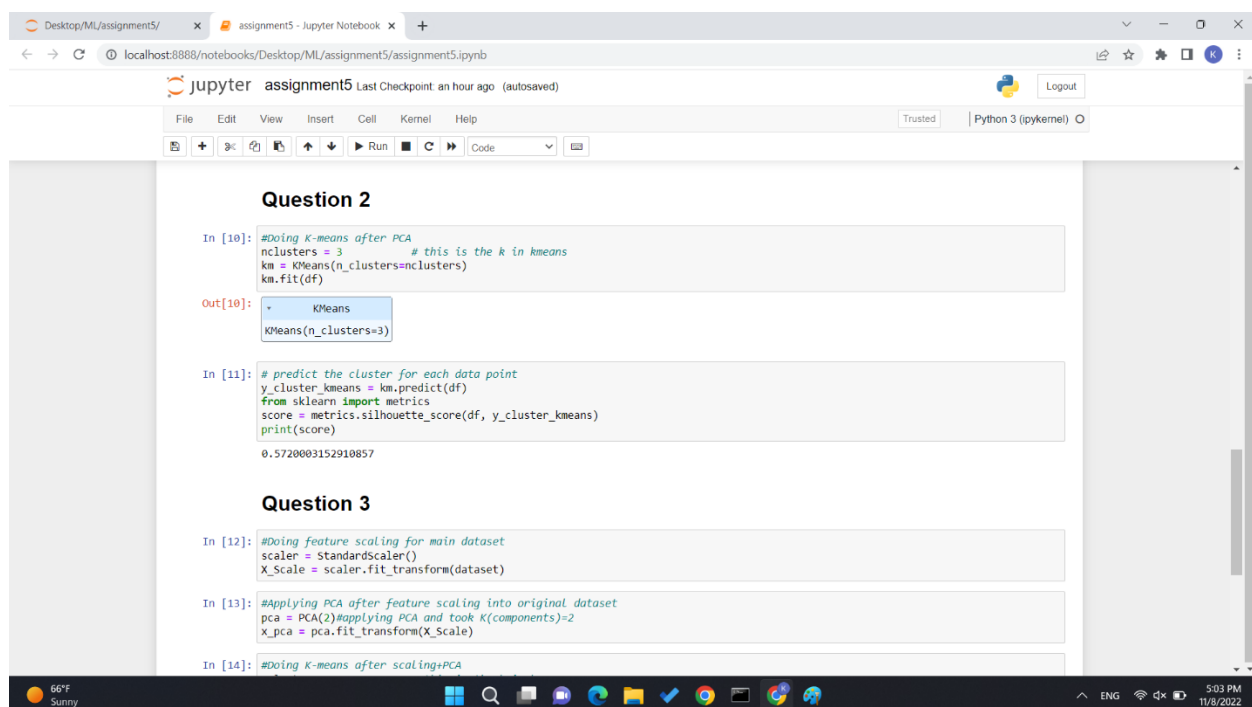
In [9]: print(df)

      0      1
0  -4326.383956   921.566884
1   4118.916676  -2432.846347
2   1497.907660  -1997.578692
3   1394.548556  -1488.743450
4   -3743.351874   757.342659
...
8945  -4208.357938   1122.443274
8946  -4123.924001   951.683803
8947  -4379.444202   911.594566
8948  -4791.117744   1032.540944
8949  -3623.702749   1555.134769

[8950 rows x 2 columns]
```

Question 2

In above screenshot I have filled the null values with the mean values of the column. After that applied the PCA for components =2 and printed the output. This is the question 1.



```
Question 2

In [10]: #Doing K-means after PCA
nclusters = 3 # this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(df)

Out[10]:
KMeans(n_clusters=3)

In [11]: # predict the cluster for each data point
y_cluster_kmeans = km.predict(df)
from sklearn import metrics
score = metrics.silhouette_score(df, y_cluster_kmeans)
print(score)

0.5720003152910857

Question 3

In [12]: #Doing feature scaling for main dataset
scaler = StandardScaler()
X_Scale = scaler.fit_transform(dataset)

In [13]: #Applying PCA after feature scaling into original dataset
pca = PCA(2)#applying PCA and took K(components)=2
x_pca = pca.fit_transform(X_Scale)

In [14]: #Doing K-means after scaling+PCA
```

For question 1.2 first I have applied the k-means (clusters = 3) on the output from question 1. Then predicted the number of clusters for each datapoints and computed the silhouette score. Which is 0.572000.

```
#Doing feature scaling for main dataset
scaler = StandardScaler()
X_scale = scaler.fit_transform(dataset)

#Applying PCA after feature scaling into original dataset
pca = PCA(2)#applying PCA and took K(components)=2
x_pca = pca.fit_transform(X_scale)

#Doing K-means after scaling+PCA
nclusters = 3 # this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(x_pca)

KMeans
KMeans(n_clusters=3)

# predict the cluster for each data point
y_cluster_kmeans = km.predict(x_pca)
from sklearn import metrics
score = metrics.silhouette_score(x_pca, y_cluster_kmeans)
print(score)
0.4523238982312997

# silhouette score is decreasing after doing feature scaling
```

Question 1.3: first I have applied the feature scaling using `standardscaler()` on the original dataset. Then again applied the PCA and k-means (clusters = 3). And calculated silhouette score which is 0.45232.

Silhouette score after feature scaling is less compared to first silhouette score.

Question 2:

```
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('pd_speech_features.csv')#importing dataset

dataset.head()

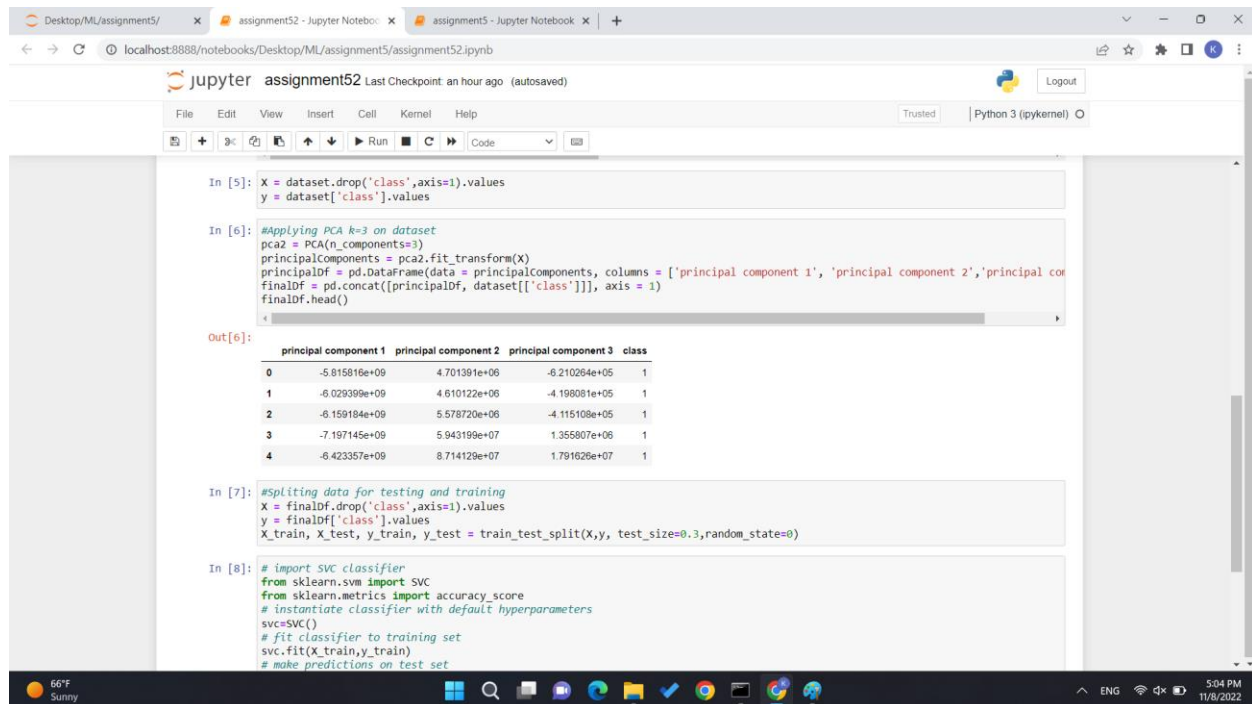
id gender PPE DFA RPDE numPulses numPeriodsPulses meanPeriodPulses stdDevPeriodPulses locPctLitter ... tqwt_kurtosisValue_dec_28 tq
0 0 1 0.85247 0.71826 0.57227 240 239 0.008054 0.000087 0.00218 ... 1.5620
1 0 1 0.76696 0.69481 0.53966 234 233 0.008258 0.000073 0.00195 ... 1.5589
2 0 1 0.85083 0.67604 0.58982 232 231 0.008340 0.000090 0.00176 ... 1.5643
3 1 0 0.41121 0.79672 0.59257 178 177 0.010658 0.000193 0.00419 ... 3.7805
4 1 0 0.32790 0.79782 0.53028 236 235 0.008192 0.002669 0.00535 ... 6.1727

5 rows x 755 columns

#Performing feature scaling on dataset
scaler = preprocessing.StandardScaler()
scaler.fit(dataset)
dataset.head()

id gender PPE DFA RPDE numPulses numPeriodsPulses meanPeriodPulses stdDevPeriodPulses locPctLitter ... tqwt_kurtosisValue_dec_28 tq
0 0 1 0.85247 0.71826 0.57227 240 239 0.008054 0.000087 0.00218 ... 1.5620
1 0 1 0.76696 0.69481 0.53966 234 233 0.008258 0.000073 0.00195 ... 1.5589
2 0 1 0.85083 0.67604 0.58982 232 231 0.008340 0.000090 0.00176 ... 1.5643
3 1 0 0.41121 0.79672 0.59257 178 177 0.010658 0.000193 0.00419 ... 3.7805
```

For question 2 first I have imported the required libraries. Imported the dataset “pd_speech_feature.csv” then displayed the first few rows of the dataset. Then applied the feature scaling on the dataset and displayed the first few rows of the dataset.



The screenshot shows a Jupyter Notebook titled "assignment52" with the following code and output:

```
In [5]: X = dataset.drop('class',axis=1).values
y = dataset['class'].values

In [6]: #Applying PCA k=3 on dataset
pca2 = PCA(n_components=3)
principalComponents = pca2.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2', 'principal component 3'])
finalDf = pd.concat([principalDf, dataset[['class']]], axis = 1)
finalDf.head()
```

Out[6]:

	principal component 1	principal component 2	principal component 3	class
0	-5.815816e+09	4.701391e+06	-6.210264e+05	1
1	-6.029399e+09	4.610122e+06	-4.198081e+05	1
2	-6.159184e+09	5.578720e+06	-4.115108e+05	1
3	-7.197145e+09	5.943199e+07	1.355807e+06	1
4	-6.423357e+09	8.714129e+07	1.791626e+07	1

```
In [7]: #splitting data for testing and training
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=0.3,random_state=0)

In [8]: # import SVC classifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# instantiate classifier with default hyperparameters
svc=SVC()
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
```

First all the columns but not “class” has been assigned to “X” and column “class” has been assign to “y”. PCA has been applied to “X” where k = 3 which means dataset has been reduced to 3 and displayed the new dataset named as “finalDf” with the class column.

The screenshot shows a Jupyter Notebook titled "assignment52" running on a local host. The notebook contains three code cells. The first cell displays a dataset with four columns and four rows of data. The second cell contains code to split the data into training and testing sets using `train_test_split`. The third cell contains code to train an SVC classifier and calculate its accuracy. The output of the third cell shows an accuracy of 75.77092511013215.

2	-6.159184e+09	5.578720e+06	-4.115108e+05	1
3	-7.197145e+09	5.943199e+07	1.355807e+06	1
4	-6.423357e+09	8.714129e+07	1.791826e+07	1

```
In [7]: #splitting data for testing and training
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=0.3,random_state=0)

In [8]: # import svc classifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# instantiate classifier with default hyperparameters
svc=SVC()
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)

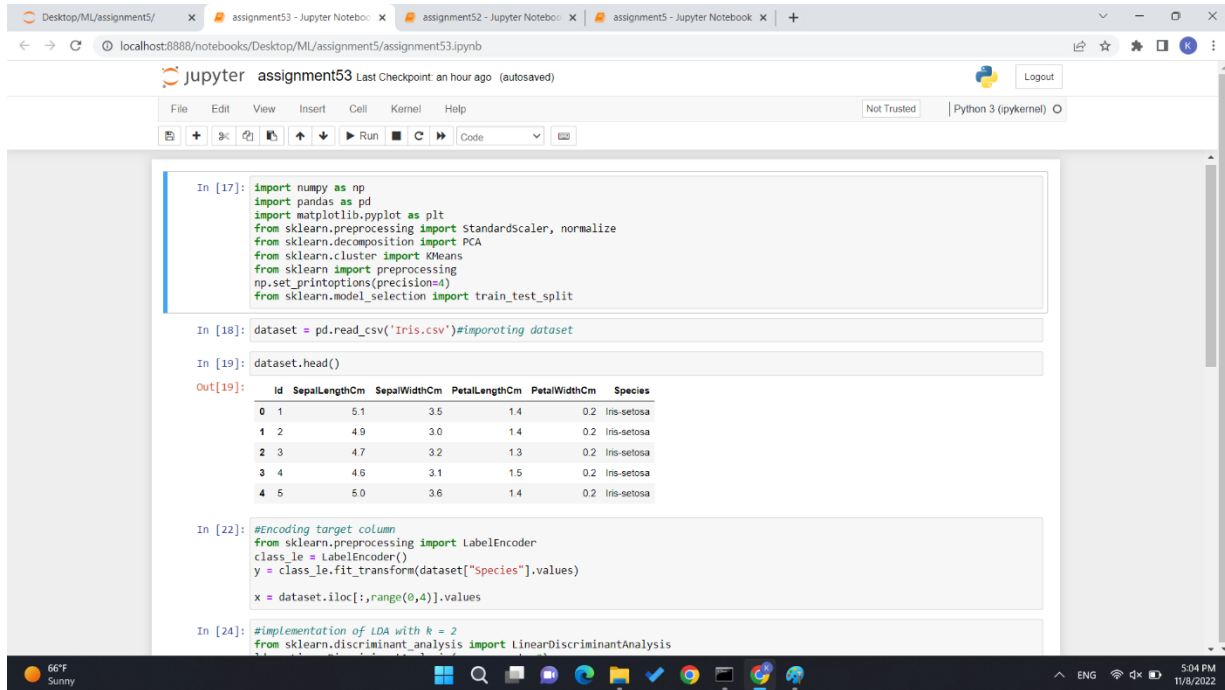
In [9]: # computing and print accuracy score
accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy of SVM:",accuracy)

accuracy of SVM: 75.77092511013215

In [ ]:
```

To calculate SVC first new dataset named “finaldf” is assigned to X and y. where column class has been assigned to y and other columns has been assigned to x. Train_test_split has been used to split the data for training and testing. Then SVC has been applied on x_train and y_train and prediction has been done on x_test. After that accuracy has been calculated which is 75.77092.

Question 3:



```
In [17]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn import preprocessing
np.set_printoptions(precision=4)
from sklearn.model_selection import train_test_split

In [18]: dataset = pd.read_csv('Iris.csv')#importing dataset

In [19]: dataset.head()

Out[19]:
```

	Id	Sepal.LengthCm	Sepal.WidthCm	Petal.LengthCm	Petal.WidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
In [22]: #Encoding target column
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(dataset["Species"].values)
x = dataset.iloc[:,range(0,4)].values

In [24]: #implementation of LDA with k = 2
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

First all the libraries and dataset “iris.csv” has been imported and displayed the first few rows of the dataset.

The screenshot shows a Jupyter Notebook titled 'assignment53' running on a local host. The notebook contains several code cells. The first cell (In [22]) encodes the target column 'Species' using a LabelEncoder. The second cell (In [24]) implements LDA with k=2. The third cell (In [27]) displays the result of LDA. The output of the LDA cell is a table with columns 'LD1', 'LD2', and 'class'. The fourth cell (In [29]) implements PCA with k=2.

```
In [22]: #Encoding target column
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(dataset["Species"].values)
x = dataset.iloc[:,range(0,4)].values

In [24]: #Implementation of LDA with k = 2
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)
X_train_lda = lda.fit_transform(x, y)

In [27]: #Displaying result of LDA with k=2
data=pd.DataFrame(X_train_lda)
data["class"]=y
data.columns=["LD1","LD2","class"]
data.head()

Out[27]:
```

	LD1	LD2	class
0	9.423452	-0.513978	0
1	8.751900	-1.591678	0
2	8.973004	-1.068204	0
3	8.170186	-1.435135	0
4	9.249789	-0.136869	0

```
In [29]: #Implementation of PCA with k = 2
pca = PCA(n_components=2)
principalcomponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalcomponents, columns = ['principal component 1', 'principal component 2'])
```

To changed the values of target column labelEncoder() has been used. So it is easy to process the dataset. Values of all columns but the target has been assigned to X and target column has been assigned to y. implementation of LDA has been done and dataset has been reduced to k=2. And result of the new dataset has been displayed.

The screenshot shows a Jupyter Notebook titled 'assignment53' running on a local host. The notebook contains a code cell with the following Python code:

```
In [29]: #implementation of PCA with k = 2
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)

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

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

The output of the code cell is a table with 5 rows and 3 columns:

	principal component 1	principal component 2	class
0	74.541053	0.408042	0
1	73.542545	0.274103	0
2	72.550398	0.043474	0
3	71.544942	0.125560	0
4	70.545790	0.203177	0

Below the output, there is a text cell with the following text:

```
In [ ]: #difference between LDA and PCA
#LDA is useful when we have labels for data points and want to predict which label new points will have based on their
#values. while if we don't have labels then PCA will be useful.
#LDA is used for classification and PCA is used to get low dimensional data.
```

implementation of PCA has been done where $k=2$ to compare result of LDA and PCA.

difference between LDA and PCA:

LDA is useful when we have labels for data points and want to predict which label new points will have based on their values. while if we don't have labels then PCA will be useful. LDA is used for classification and PCA is used to get low dimensional data.