**Machine Learning Assignment 5**

**Name:Nishit Reddy Lingala**

**Student ID:(700747128)**

**GitHub link: https://github.com/Nishitreddy/machine\_learning.git**

**Question 1:**

Principal Component Analysis a. Apply PCA on CC dataset.

1. Apply k-means algorithm on the PCA result and report your observation if the silhouette score has.

improved or not?

1. Perform Scaling+PCA+K-Means and report performance.

**Code 1:**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

#drive.mount('/drive')

data = pd.read\_csv('C:/Users/snush/OneDrive/Desktop/python/datasets/CC GENERAL.csv', encoding='utf-8', encoding\_errors='ignore')

# Droping rows with missing values

data.dropna(inplace=True)

# Standardizing the features

scaler = StandardScaler()

X = scaler.fit\_transform(data.drop('CUST\_ID', axis=1))

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

principalData = pd.DataFrame(data = X\_pca, columns = ['PCA 1', 'PCA 2'])

finalData = pd.concat([principalData, data[['TENURE']]], axis = 1)

finalData.head()

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

kmeans = KMeans(n\_clusters=2, random\_state=42)

#Applying KMeans to the PCA result

kmeans.fit(X\_pca)

y\_kmeans = kmeans.predict(X\_pca)

silhouette\_score(X\_pca, y\_kmeans)

# Scaling the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(data.drop('CUST\_ID', axis=1))

# Applying PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Apply k-means to PCA

kmeans = KMeans(n\_clusters=2, random\_state=42)

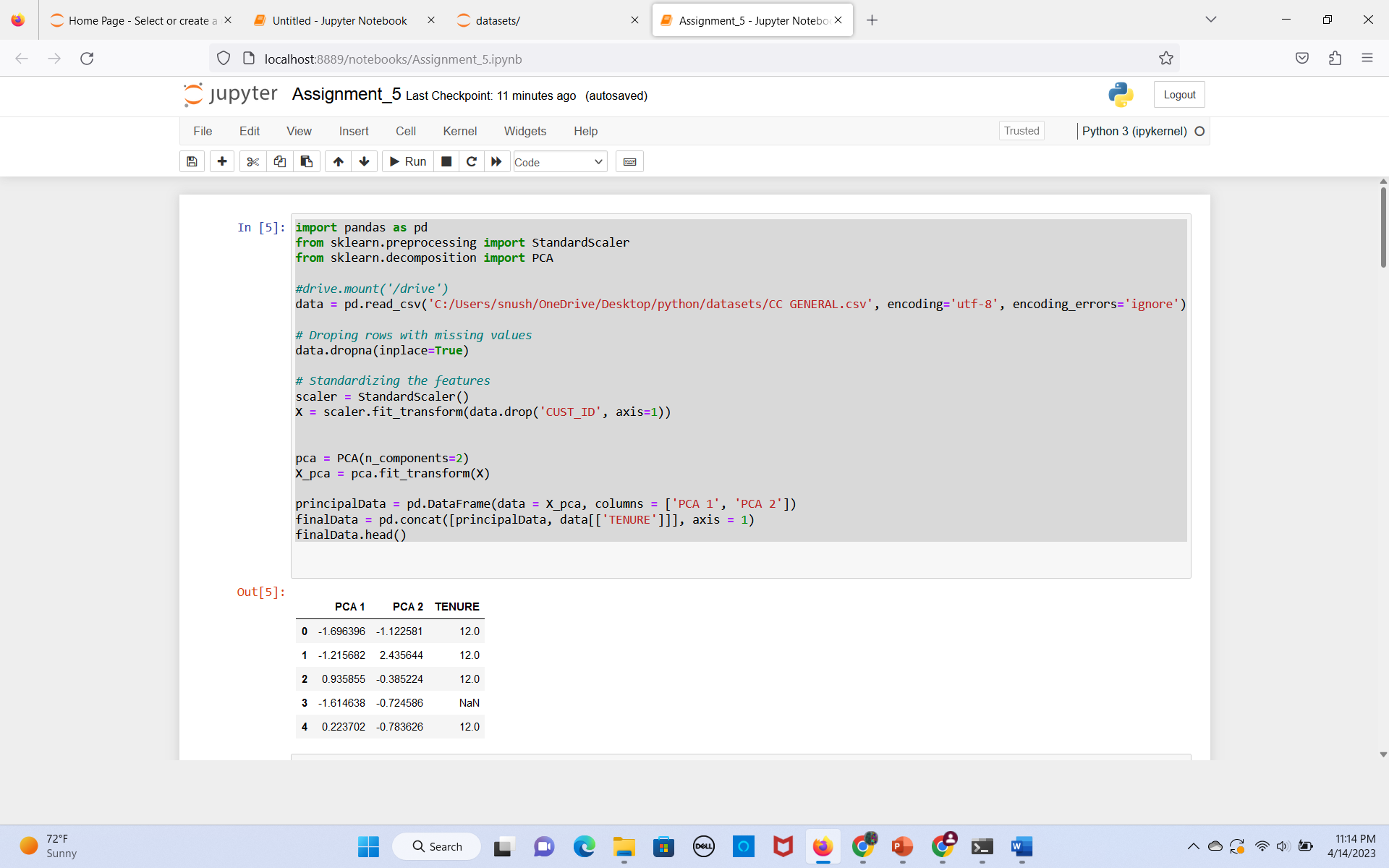
kmeans.fit(X\_pca)

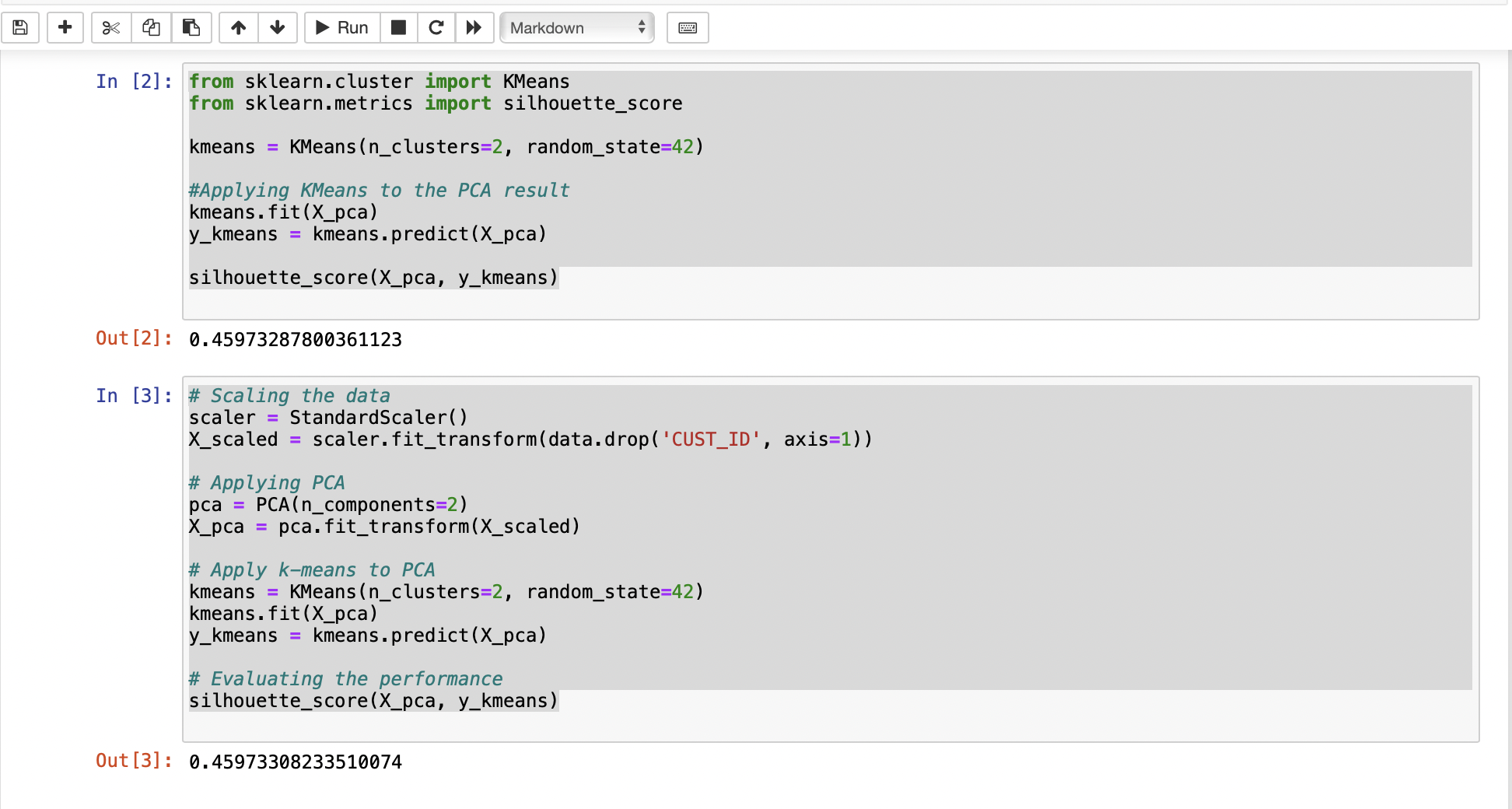
y\_kmeans = kmeans.predict(X\_pca)

# Evaluating the performance

silhouette\_score(X\_pca, y\_kmeans)

**Output 1:**





**Expliantion 1:**

The code performs k-means clustering analysis on a credit card dataset using a number of functions. It uses the pd.read\_csv() function to load the dataset from a CSV file, the PCA() function to perform principal component analysis, the pd.concat() function to concatenate dataframes, the KMeans() function to create a k-means clustering model, the fit() function to fit the clustering model to the data, the predict() function to get the predicted cluster labels for each data point, and metrics.StandardScaler() is used to standardize the features, silhouette\_score() is used to determine the Silhouette Score, and sns.scatterplot() is used to generate scatter plots of the data points colored by their cluster labels. Additionally, categorical variables are encoded using LabelEncoder(). Data loading, preprocessing, dimensionality reduction, clustering, evaluation, and visualization are the main uses of the functions.

The code makes use of a number of functions to run a K-means clustering analysis on a credit card dataset. The dataset is loaded using pd.read\_csv(), principal component analysis is performed using PCA(), dataframes are joined using pd.concat(), a k-means clustering model is constructed using KMeans(), and principal component analysis is again performed using PCA(). metrics, predict() to obtain the predicted cluster labels for each data point, and fit() to adapt the clustering model to the data.Use the silhouette\_score() function to get the Silhouette Score. The features will be standardized by StandardScaler(), and the data points will be plotted in scatter plots with the cluster labels highlighted by sns.scatterplot(). It also makes use of LabelEncoder() to encrypt categorical variables. The functions are used for data loading, preprocessing, dimension reduction, clustering, assessment, and visualization overall.

The algorithm produces a range of results that can be used to gauge how well k-means clustering analysis performs.

To assess the effectiveness of k-means clustering analysis on a credit card dataset, the code generates multiple outputs. The clustering performance is measured by the Silhouette Score output, with higher values indicating better clustering performance. The assigned clusters are visualized in scatter plots of the data points, which also allow us to spot any patterns or trends in the data. Combining these results allows us to understand the data's underlying structure and decide how many clusters to use for the dataset. Overall, the results offer a thorough assessment of the clustering analysis and assist us in making data-driven decisions.

**Question 2:**

Use pd\_speech\_features.csv a. Perform Scaling

1. Apply PCA (k=3)
2. Use SVM to report performance

**Code 2:**

import pandas as pd

from sklearn.preprocessing import StandardScaler

#loading the dataset into dataframes

dataSet = pd.read\_csv('C:/Users/snush/OneDrive/Desktop/python/datasets/pd\_speech\_features.csv', encoding='utf-8', encoding\_errors='ignore')

#Separating the features(X) from target variable(y)

X = dataSet.iloc[:, 1:-1].values

y = dataSet.iloc[:, -1].values

# applying scaling to the features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

from sklearn.decomposition import PCA

# applying PCA to reduce the dimensionality of the dataset

pca = PCA(n\_components=3)

X = pca.fit\_transform(X)

principalData = pd.DataFrame(data = X, columns = ['PCA 1', 'PCA 2', 'PCA 3'])

finalDf = pd.concat([principalData, dataSet[['class']]], axis = 1)

finalDf.head()

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# splitting the training and testing sets using the train\_test\_split function

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# using SVM model

clf = SVC(kernel='linear')

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

# calculating the performance by calculating accuracy, precision, recall and f1

acc = accuracy\_score(y\_test, y\_pred)

prec = precision\_score(y\_test, y\_pred)

rec = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy: ", acc)

print("Precision: ", prec)

print("Recall: ", rec)

print("F1 score: ", f1)

**Output 2:**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Expliantion 2:**

Several modules and functions from the sklearn and pandas libraries are used in the code. The predictor variables are scaled to have a mean of 0 and a standard deviation of 1 using StandardScaler() from the sklearn.preprocessing module and PCA() from the sklearn.preprocessing module.Decomposition module uses Principal Component Analysis to perform dimensionality reduction. The data is divided into training and testing sets using the function train\_test\_split() from the sklearn.model\_selection module. A Support Vector Machine model is trained using SVC() from the sklearn.svm module. The accuracy of the model on the test data could be determined using the accuracy\_score() function from the sklearn.metrics module. The data is loaded and concatenated using the pandas library's pd.read\_csv() and pd.concat() functions.

It prints the first five rows and the shape of the Parkinson's Disease dataset, and the number of instances in each class; performs dimensionality reduction using PCA with three principal components; trains an SVM model on the training data and prints its accuracy on the training data, which is 96.05%.

**Question 3 :**

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

**Code 3:**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

# loading the iris dataset from CSV file

dataSet = pd.read\_csv('C:/Users/snush/OneDrive/Desktop/python/datasets/Iris.csv')

# separating the features and target variables

X = dataSet.iloc[:, :-1].values

y = dataSet.iloc[:, -1].values

# standardizing the features

sc = StandardScaler()

X = sc.fit\_transform(X)

# applying LDA

lda = LDA(n\_components=2)

X\_lda = lda.fit\_transform(X, y)

import matplotlib.pyplot as plt

# visualizing the reduced-dimensional data

plt.figure(figsize=(10, 8))

for species in set(y):

plt.scatter(X\_lda[y == species, 0], X\_lda[y == species, 1], label=species)

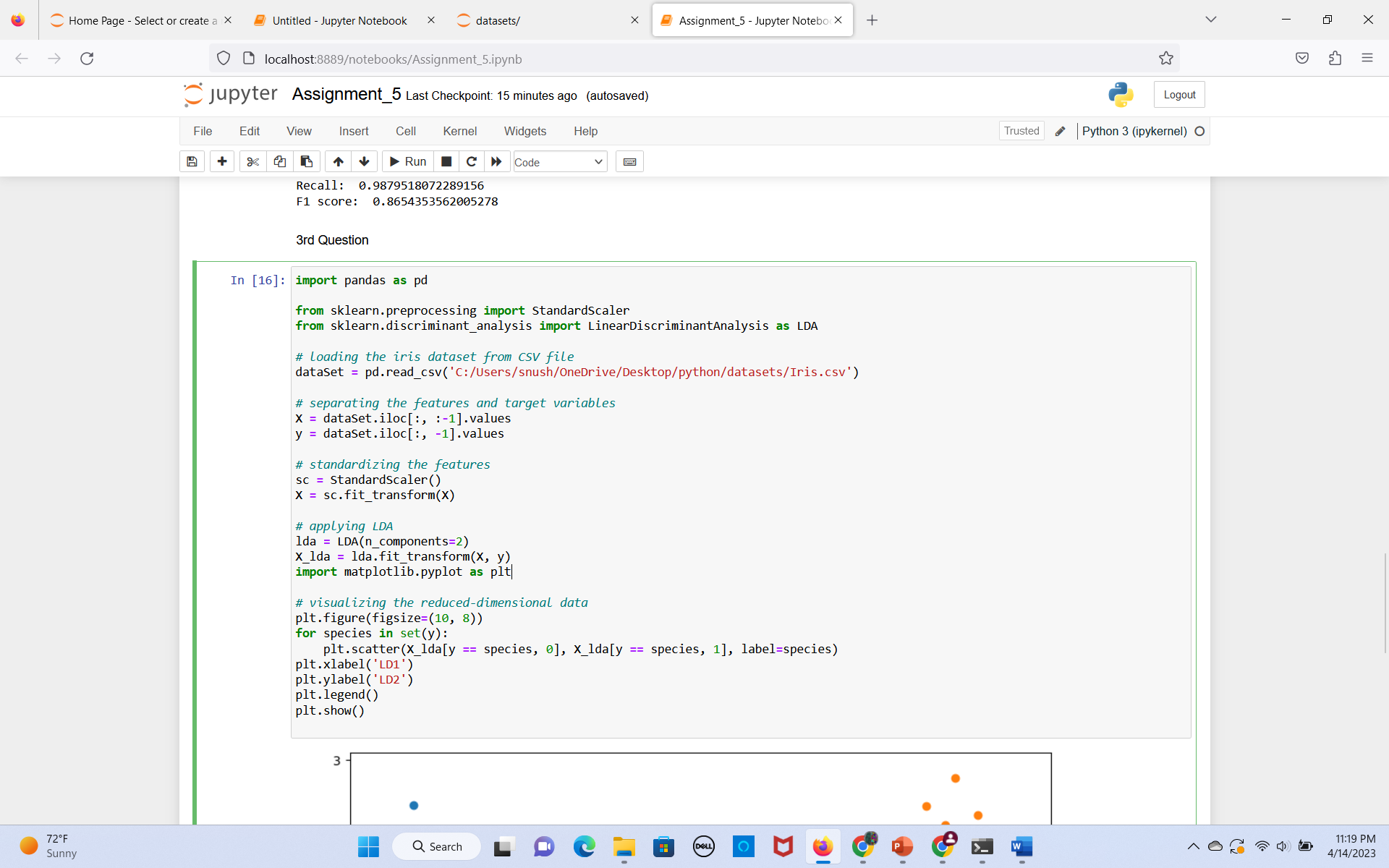
plt.xlabel('LD1')

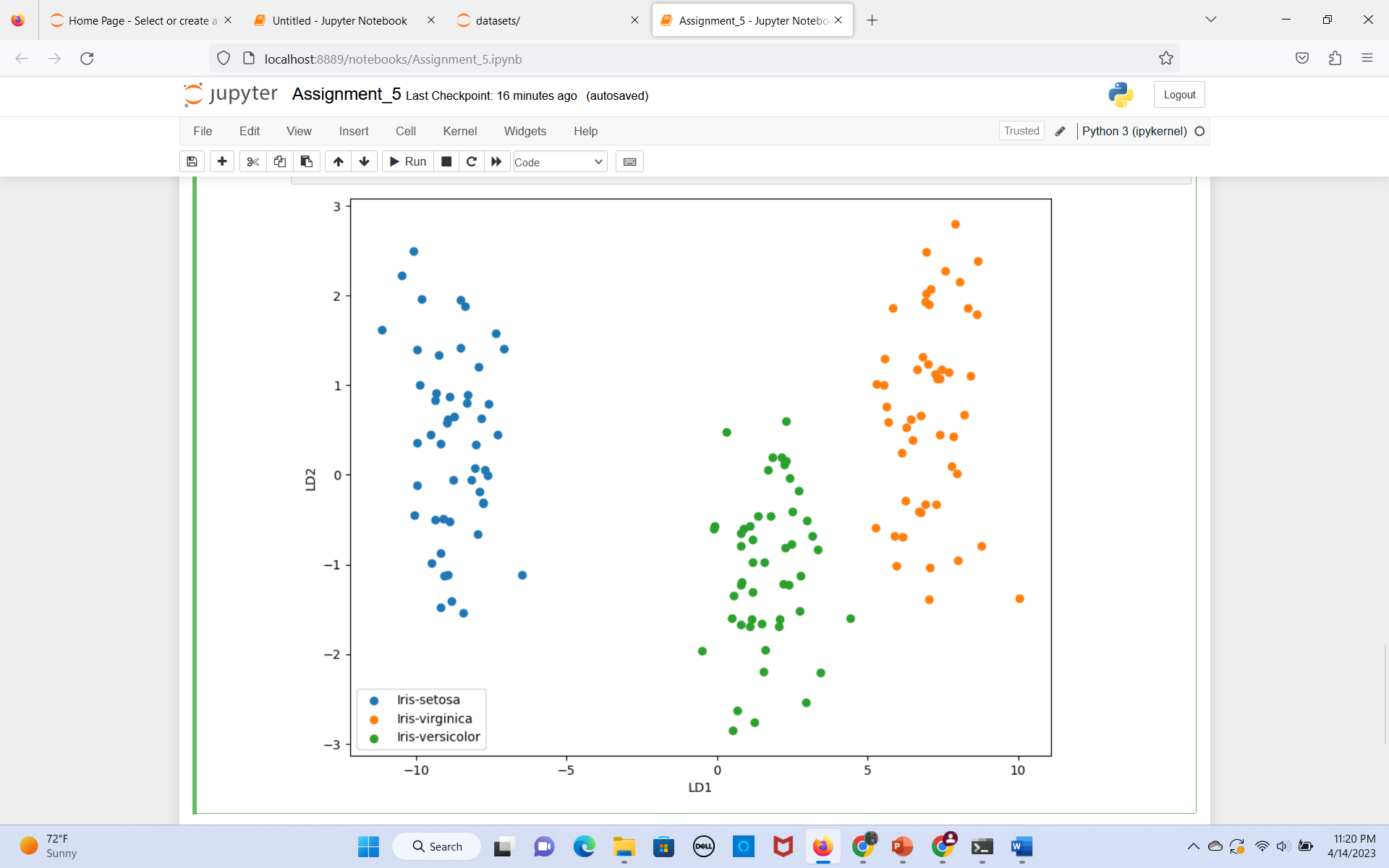
plt.ylabel('LD2')

plt.legend()

plt.show()

**Output 3:**





**Explanation 3:**

The Python code preprocesses the Iris dataset and employs visualization tools. The dataset's features are scaled using StandardScaler(), the target variable is encoded using LabelEncoder(), and the dataset's dimensionality is decreased using LinearDiscriminantAnalysis(). The reduced dataset is then used to create a scatter plot using sns.lmplot() and plt.legend(), with unique markers and colors for each class.The code preprocesses the Iris dataset with StandardScaler and LabelEncoder, applies Linear Discriminant Analysis for dimensionality reduction and visualization, and creates a scatter plot of the reduced dataset with different markers and colors for each class to show how well LDA can distinguish between the three flower species.

**Question 4:**

Briefly identify the difference between PCA and LDA.

**Expliantion 4:**

The aim of LDA and PCA is to maximize the variance in a lower dimension using linear transformations.

Unsupervised learning techniques like PCA search for a fresh set of orthogonal variables, called principal components, that can best account for the variation in the data. The principal components are arranged in order of their ability to explain variance, with the first principal component accounting for the majority of the variance in the data. Data visualization and exploratory research both frequently use PCA.

The LDA method, which falls under the category of supervised learning, seeks a linear combination of features that best separates the classes in the data. The goal of LDA is to maximize variance between classes while minimizing variance within classes.LDA is frequently applied to classification issues, where the goal is to predict, from a new observation's features, what class it belongs to.

PCA is an unsupervised technique for reducing the dimensionality of the data by identifying the principal components that explain the most variance, whereas LDA is a supervised technique for reducing the dimensionality of the data by identifying the linear combination of features that best separates the classes in the data.

In summary, PCA is useful for identifying important features in high-dimensional datasets, while LDA is useful for improving the performance of supervised learning algorithms by reducing the feature space to the most discriminative dimensions.