Now, Let’s understand Principal Component Analysis with Python.  
To get the dataset used in the implementation, click [here](https://media.geeksforgeeks.org/wp-content/uploads/Wine.csv).  
**Step 1:** Importing the libraries

* Python

|  |
| --- |
| # importing required libraries  **import** numpy as np  **import** matplotlib.pyplot as plt  **import** pandas as pd |

**Step 2:** Importing the data set

Import the dataset and distributing the dataset into X and y components for data analysis.

* Python

|  |
| --- |
| # importing or loading the dataset  dataset **=** pd.read\_csv('wine.csv')    # distributing the dataset into two components X and Y  X **=** dataset.iloc[:, 0:13].values  y **=** dataset.iloc[:, 13].values |

**Step 3:** Splitting the dataset into the Training set and Test set

* Python

|  |
| --- |
| # Splitting the X and Y into the  # Training set and Testing set  **from** sklearn.model\_selection **import** train\_test\_split    X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.2, random\_state **=** 0) |

**Step 4:** Feature Scaling  
Doing the pre-processing part on training and testing set such as fitting the Standard scale.

* Python

|  |
| --- |
| # performing preprocessing part  **from** sklearn.preprocessing **import** StandardScaler  sc **=** StandardScaler()    X\_train **=** sc.fit\_transform(X\_train)  X\_test **=** sc.transform(X\_test) |

**Step 5:**Applying PCA function  
Applying the PCA function into the training and testing set for analysis.

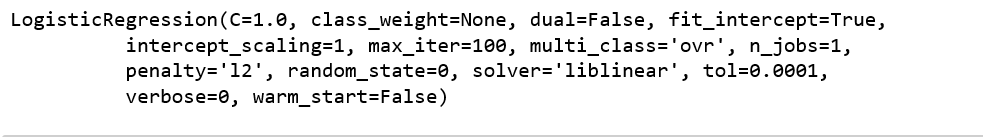
* Python

|  |
| --- |
| # Applying PCA function on training  # and testing set of X component  **from** sklearn.decomposition **import** PCA    pca **=** PCA(n\_components **=** 2)    X\_train **=** pca.fit\_transform(X\_train)  X\_test **=** pca.transform(X\_test)    explained\_variance **=** pca.explained\_variance\_ratio\_ |

**Step 6:**Fitting Logistic Regression To the training set

* Python

|  |
| --- |
| # Fitting Logistic Regression To the training set  **from** sklearn.linear\_model **import** LogisticRegression    classifier **=** LogisticRegression(random\_state **=** 0)  classifier.fit(X\_train, y\_train) |



**Step 7:**Predicting the test set result

* Python

|  |
| --- |
| # Predicting the test set result using  # predict function under LogisticRegression  y\_pred **=** classifier.predict(X\_test) |

**Step 8:**Making the confusion matrix

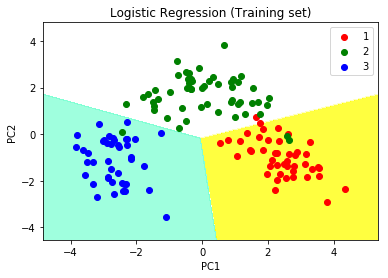
* Python

|  |
| --- |
| # making confusion matrix between  #  test set of Y and predicted value.  **from** sklearn.metrics **import** confusion\_matrix    cm **=** confusion\_matrix(y\_test, y\_pred) |

**Step 9:** Predicting the training set result

* Python

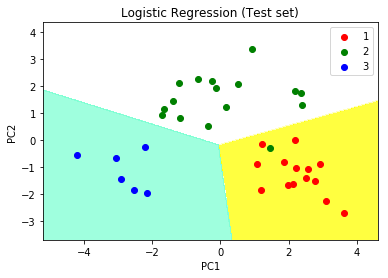
|  |
| --- |
| # Predicting the training set  # result through scatter plot  **from** matplotlib.colors **import** ListedColormap    X\_set, y\_set **=** X\_train, y\_train  X1, X2 **=** np.meshgrid(np.arange(start **=** X\_set[:, 0].min() **-** 1,                       stop **=** X\_set[:, 0].max() **+** 1, step **=** 0.01),                       np.arange(start **=** X\_set[:, 1].min() **-** 1,                       stop **=** X\_set[:, 1].max() **+** 1, step **=** 0.01))    plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),               X2.ravel()]).T).reshape(X1.shape), alpha **=** 0.75,               cmap **=** ListedColormap(('yellow', 'white', 'aquamarine')))    plt.xlim(X1.min(), X1.max())  plt.ylim(X2.min(), X2.max())    **for** i, j **in** enumerate(np.unique(y\_set)):      plt.scatter(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],                  c **=** ListedColormap(('red', 'green', 'blue'))(i), label **=** j)    plt.title('Logistic Regression (Training set)')  plt.xlabel('PC1') # for Xlabel  plt.ylabel('PC2') # for Ylabel  plt.legend() # to show legend    # show scatter plot  plt.show() |



**Step 10:** Visualizing the Test set results

* Python

|  |
| --- |
| # Visualising the Test set results through scatter plot  **from** matplotlib.colors **import** ListedColormap    X\_set, y\_set **=** X\_test, y\_test    X1, X2 **=** np.meshgrid(np.arange(start **=** X\_set[:, 0].min() **-** 1,                       stop **=** X\_set[:, 0].max() **+** 1, step **=** 0.01),                       np.arange(start **=** X\_set[:, 1].min() **-** 1,                       stop **=** X\_set[:, 1].max() **+** 1, step **=** 0.01))    plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),               X2.ravel()]).T).reshape(X1.shape), alpha **=** 0.75,               cmap **=** ListedColormap(('yellow', 'white', 'aquamarine')))    plt.xlim(X1.min(), X1.max())  plt.ylim(X2.min(), X2.max())    **for** i, j **in** enumerate(np.unique(y\_set)):      plt.scatter(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],                  c **=** ListedColormap(('red', 'green', 'blue'))(i), label **=** j)    # title for scatter plot  plt.title('Logistic Regression (Test set)')  plt.xlabel('PC1') # for Xlabel  plt.ylabel('PC2') # for Ylabel  plt.legend()    # show scatter plot  plt.show() |



**we can visualize the data in the new principal component space:**

* Python3

|  |
| --- |
| # plot the first two principal components with labels  y **=** df.iloc[:, **-**1].values  colors **=** ["r", "g"]  labels **=** ["Class 1", "Class 2"]  **for** i, color, label **in** zip(np.unique(y), colors, labels):      plt.scatter(X\_pca[y **==** i, 0], X\_pca[y **==** i, 1], color**=**color, label**=**label)  plt.xlabel("Principal Component 1")  plt.ylabel("Principal Component 2")  plt.legend()  plt.show()  7 |

This is a simple example of how to perform PCA using Python. The output of this code will be a scatter plot of the first two principal components and their explained variance ratio. By selecting the appropriate number of principal components, we can reduce the dimensionality of the dataset and improve our understanding of the data.