Experiment No 6

Title: To implement PCA

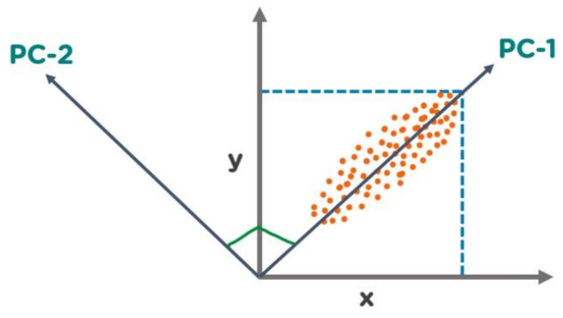
Lab Objective: PO2: To acquire in depth understanding of various supervised andunsupervised Algorithm.

Theory:

While working with high-dimensional data, machine learning models often seem to overfit, and this reduces the ability to generalize past the training set examples. Hence, it is important to perform dimensionality reduction techniques before creating a model. In this article, we'll learn the PCA in Machine Learning with a use case demonstration in Python.

What is Principal Component Analysis?

The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables.



In the above figure, we have several points plotted on a 2-D plane. There are two principal components. PC1 is the primary principal component that explains the maximum variance in the data. PC2 is another principal component that is orthogonal to PC1.

What is a Principal Component?

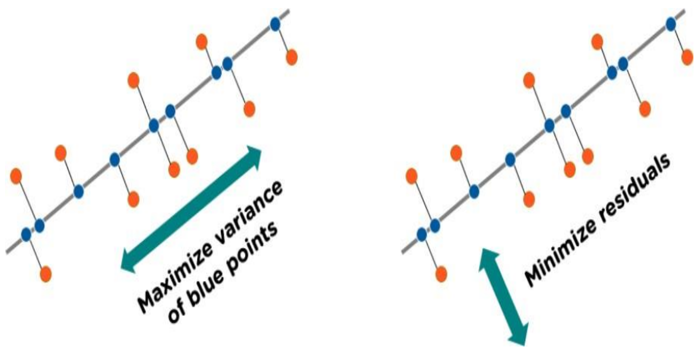
The Principal Components are a straight line that captures most of the variance of the data. They have a direction and magnitude. Principal components are orthogonal projections (perpendicular) of data onto lower-dimensional space.

Now that you have understood the basics of PCA, let’s look at the next topic on PCA in Machine Learning.

Applications of PCA in Machine Learning

- PCA is used to visualize multidimensional data.
- It is used to reduce the number of dimensions in healthcare data.
- PCA can help resize an image.
- It can be used in finance to analyze stock data and forecast returns.
- PCA helps to find patterns in the high-dimensional datasets.

How does Principal Component Analysis Work?



1. Normalize the data

Standardize the data before performing PCA. This will ensure that each feature has a mean = 0 and variance = 1.

$$Z = \frac{x - \mu}{\sigma}$$

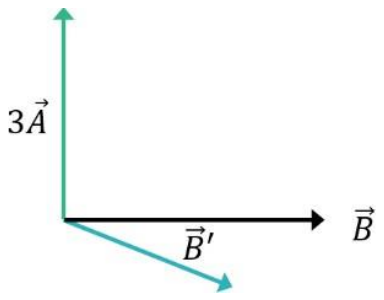
2. Build the covariance matrix

Construct a square matrix to express the correlation between two or more features in a multidimensional dataset.

$$\begin{bmatrix} \text{Var}(x) & \text{Cov}(x,y) & \cdots & \text{Cov}(x,m) \\ \text{Cov}(x,y) & \ddots & & \vdots \\ \text{Cov}(z,x) & & \ddots & \vdots \\ \vdots & & & \\ \text{Cov}(n,x) & \cdots & \cdots & \text{Var}(n) \end{bmatrix}$$

3. Find the Eigenvectors and Eigenvalues

Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.



4. Sort the eigenvectors in highest to lowest order and select the number of principal components.

Now that you have understood How PCA in Machine Learning works, let’s perform a hands-on demo on PCA with Python

Program:

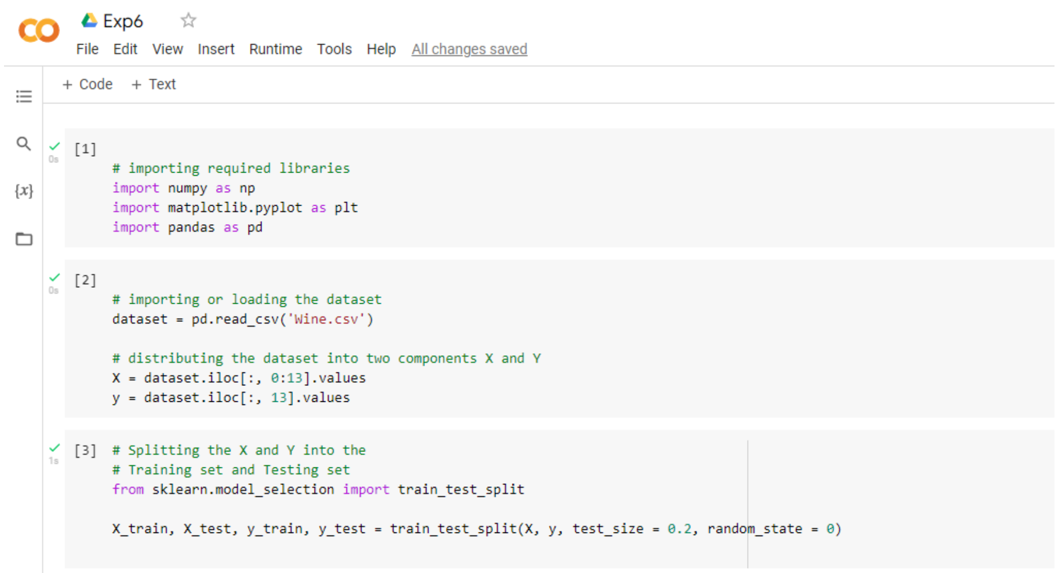
```
# importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# 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

# 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)

# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# 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_

# Fitting Logistic Regression To the training set
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
# Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)
# making confusion matrix between
# test set of Y and predicted value.
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

```
# 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()

# 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()

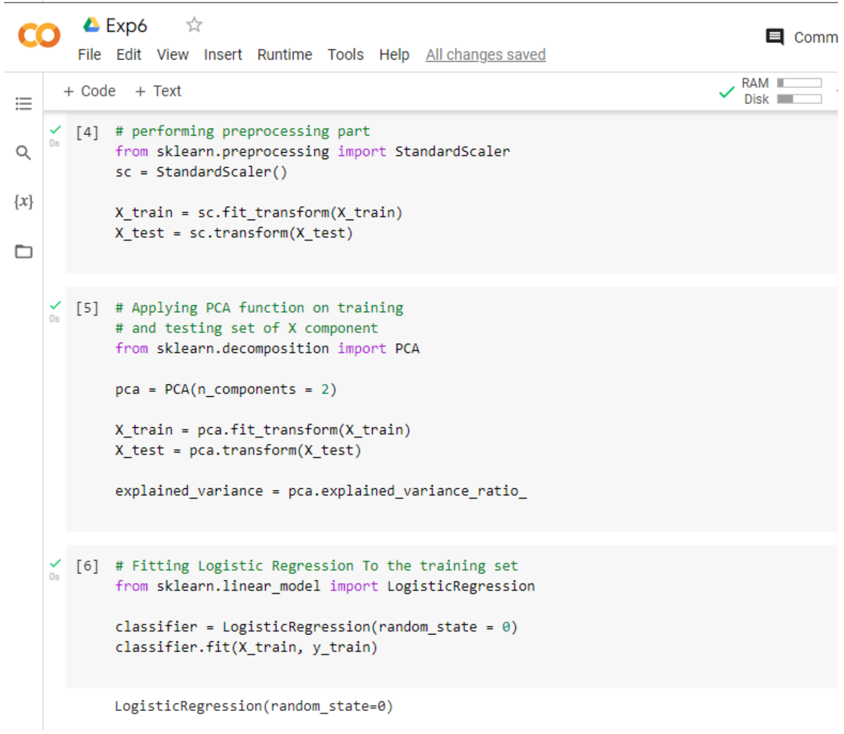
Implementation:



```
[1]
# importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
[2]
# 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
```

```
[3] # 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)
```

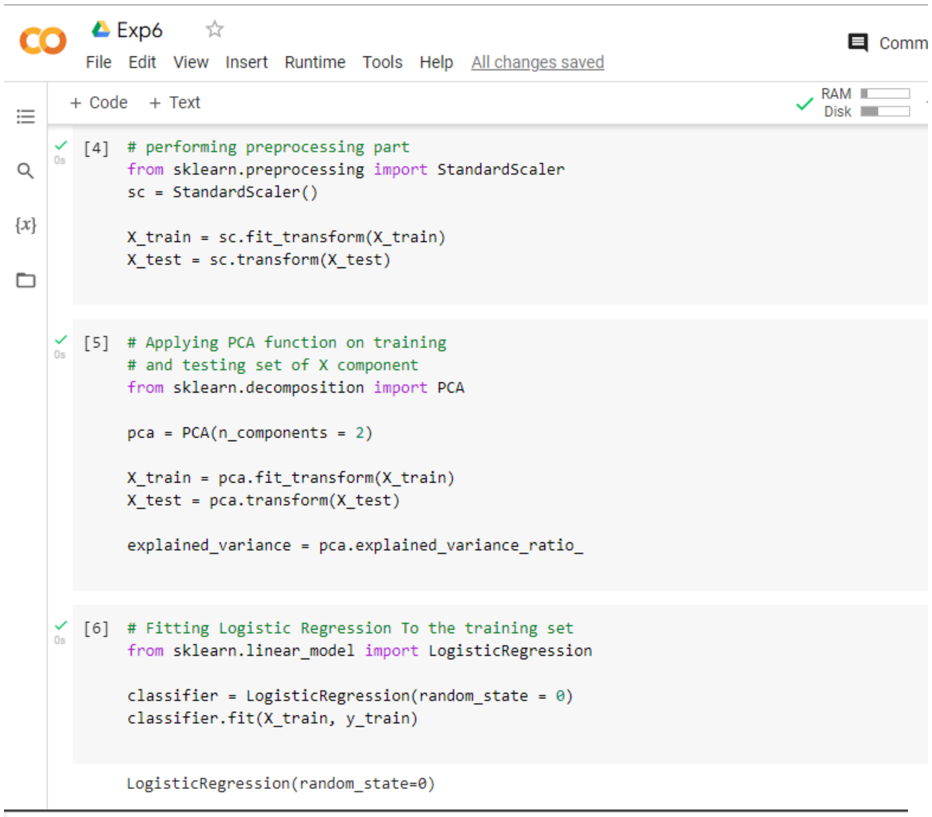


```
[4] # performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
[5] # 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_
```

```
[6] # Fitting Logistic Regression To the training set
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0)
```

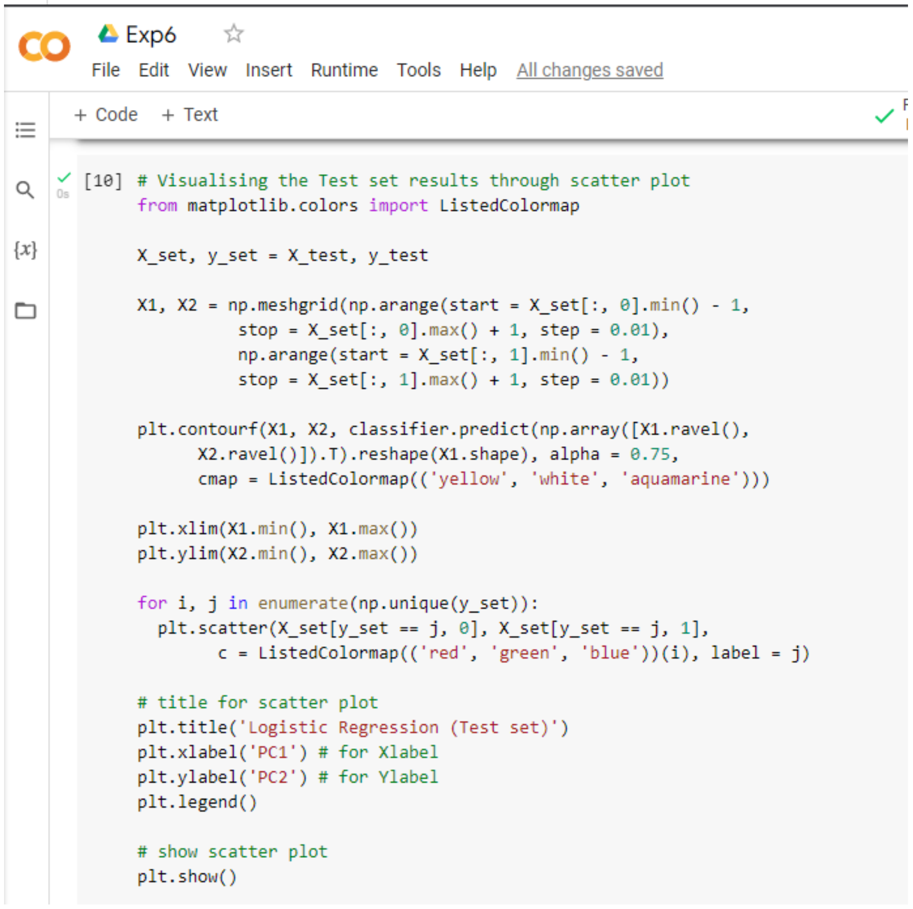


```
[4] # performing preprocessing part
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()

    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[5] # 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_
```

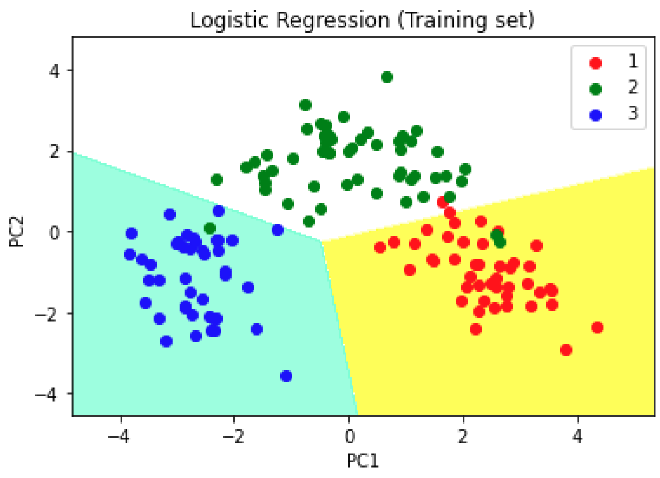
```
[6] # Fitting Logistic Regression To the training set
    from sklearn.linear_model import LogisticRegression

    classifier = LogisticRegression(random_state = 0)
    classifier.fit(X_train, y_train)
```

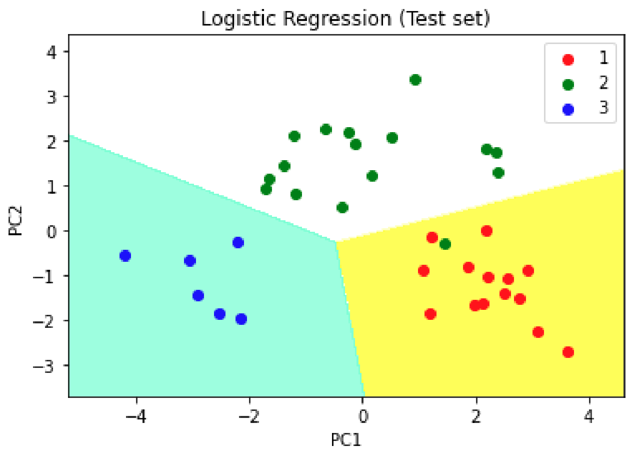
```
    LogisticRegression(random_state=0)
```



```
[10] # 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()
```

Output:





```
[ ]
```

Conclusion:

The principal component analysis is a widely used unsupervised learning method to perform dimensionality reduction. We hope that this article helped you understand what PCA is and the applications of PCA. You looked at the applications of PCA and how it works.