Linear Discriminant Analysis (LDA), it not only finds the component axises, it also maximizes the separation between multiple classes. Both PCA and LDA are linear transformation techniques used for dimensionality reduction. PCA is unsupervised and LDA is supervised because of the relation to the dependent variable.

Importing Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
```

Importing Datasets

```
df = pd.read_csv('Wine.csv')
df.head()
x = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

Feature Scaling

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Applying PCA

```
lda = LDA(n_components = 2)
x_train = lda.fit_transform(x_train, y_train)
x_test = lda.transform(x_test)
```

Training the logistic regression model on training set

```
lr = LogisticRegression(random_state = 0)
lr.fit(x_train, y_train)
```

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

Making confusion matrix

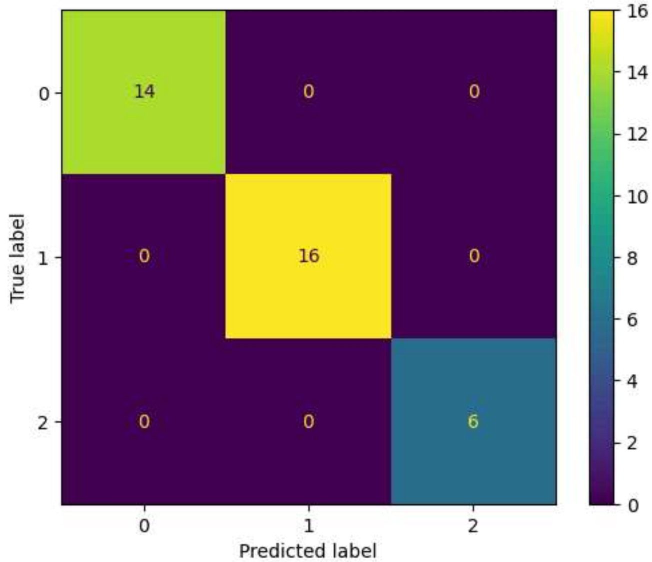
```
cm = confusion_matrix(y_test, lr.predict(x_test))
print(cm)
accuracy_score(y_test, lr.predict(x_test))
```

```
[[14  0  0]
 [ 0 16  0]
 [ 0  0  6]]
1.0
```

Visualizing confusion matrix

```
from sklearn import metrics
import matplotlib.pyplot as plt

# Assuming 'cm' is your confusion matrix
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm) # Remove display_labels
cm_display.plot()
plt.show()
```



Visualizing the training result

```
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, lr.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('black', 'yellow', 'orange')))
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(('white', 'red', 'black'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
<ipython-input-10-212a150b7978>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
  plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```

Visualizing the test result

```
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x[:, 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, lr.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
            alpha = 0.75, cmap = ListedColormap(('black', 'yellow', 'orange')))
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(('white', 'red', 'black'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
<ipython-input-11-8ba87aafc3a2>:9: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
  plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```