Complete Guide

on

Dimensionality Reduction Techniques

(PCA & LDA)

With Step by Step Python Implementation

What is Dimensionality Reduction?

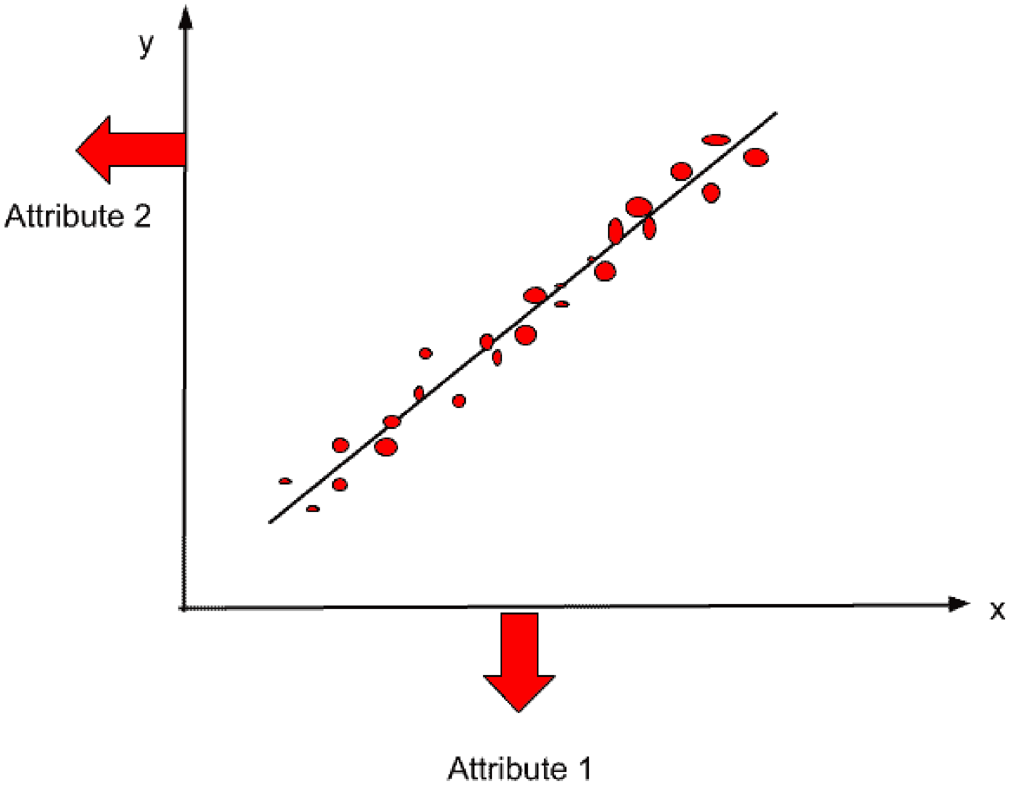
Dimensionality Reduction is a pre-processing step used in pattern classification and machine learning applications.

Let me simplify it,

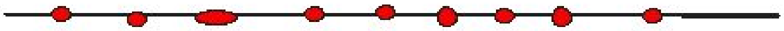
The data you collect for processing is big in size. So to process huge size data is complex. It requires more processing power and space. Therefore Dimensionality Reduction comes into the scene. It reduces the dimension of data.

So, What you mean by Reducing the dimensions?

Suppose, This is our dataset scattered in 2-dimensional space.



So, we can represent these data items in 1-dimensional space by applying dimensionality reduction. After applying dimensionality reduction data points will look something like that-



So, Dimensionality Reduction is a technique to reduce the number of dimensions. In this example, we reduced from 2- dimension to 1-dimension. I hope now you understood dimensionality reduction.

For dimensionality reduction, the two most popular techniques are-

1. PCA (Principal Component Analysis)
2. LDA(Linear Discriminant Analysis)

Now, let’s move to Principal Component Analysis-

What is Principal Component Analysis?

Principal Component Analysis(PCA) is one of the best-unsupervised algorithms. Also, it is the most popular dimensionality Reduction Algorithm.

PCA is used in various Operations. Such as-

1. Noise Filtering.
2. Visualization.
3. Feature Extraction.
4. Stock Market Prediction.
5. Gene Data Analysis.

The goal of PCA is to identify and detect the correlation between attributes. If there is a strong correlation and it is found. Then PCA reduces the dimensionality.

The main working of PCA is-

The Principal Component Analysis reduces the dimensions of a d-dimensional dataset by projecting it onto a k-dimensional subspace (where k<d).

Confused?

Don’t worry.

You will understand this definition at the end of this chapter. I will explain with the help of an example.

Now, let’s see the main purpose of PCA.

What is the main Purpose of Principal Component Analysis?

The first question is What is the need or purpose of Principal Component Analysis.

or

In which problem, The principal component analysis is used?

So the problem is **Overfitting**. **PCA** is used for the Overfitting problem.

Overfitting is the problem when you supply extra data at the training phase. When we train the model, we supply data to the model. This data is known as **Training Data.**

But, If we supply extra data, then the overfitting problem occurs.

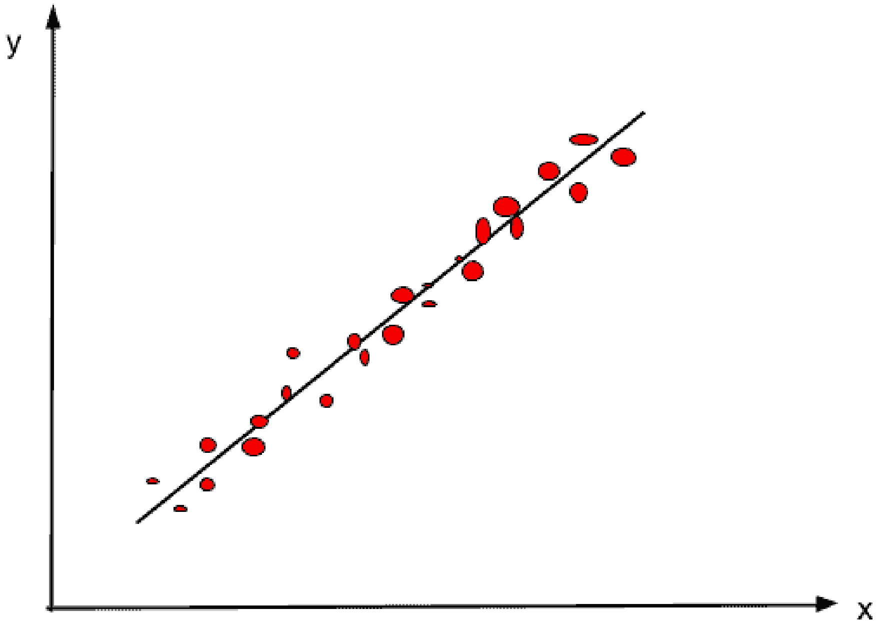
In simple words, you can consider **overfitting as overeating**. When you eat extra food, you face digestive problems. LOL.

Similarly, the same problem occurs in overfitting. When you supply extra data, you will face a problem.

I hope, now you better understand Overfitting. Right?

Now, let’s see how the **PCA** solves overfitting problems.

Suppose after the training phase, this hypothesis is generated.



So, What this hypothesis or model is doing?

This model is trying to reach each point. The single straight line is trying to touch each point. And that’s the overfitting problem.

So to solve this problem. **PCA tries to convert high dimensionality into a set of low dimensionality.**

PCA reduces the features and attributes into a low dimension, in order to solve overfitting.

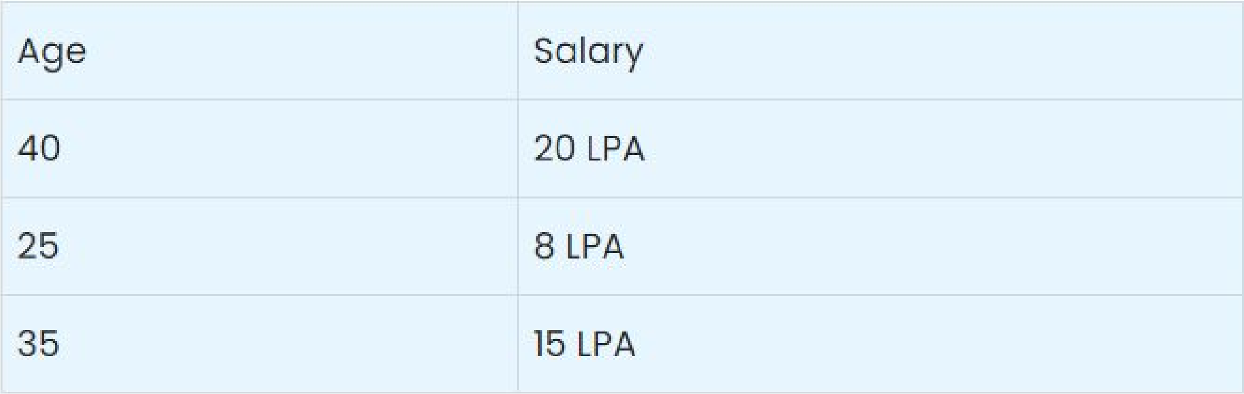
And that’s the main motive or **purpose of PCA.**

Now, let’s see how PCA works?

How Principal Component Analysis work?

In the previous section, you understand the whole concept of Overfitting.

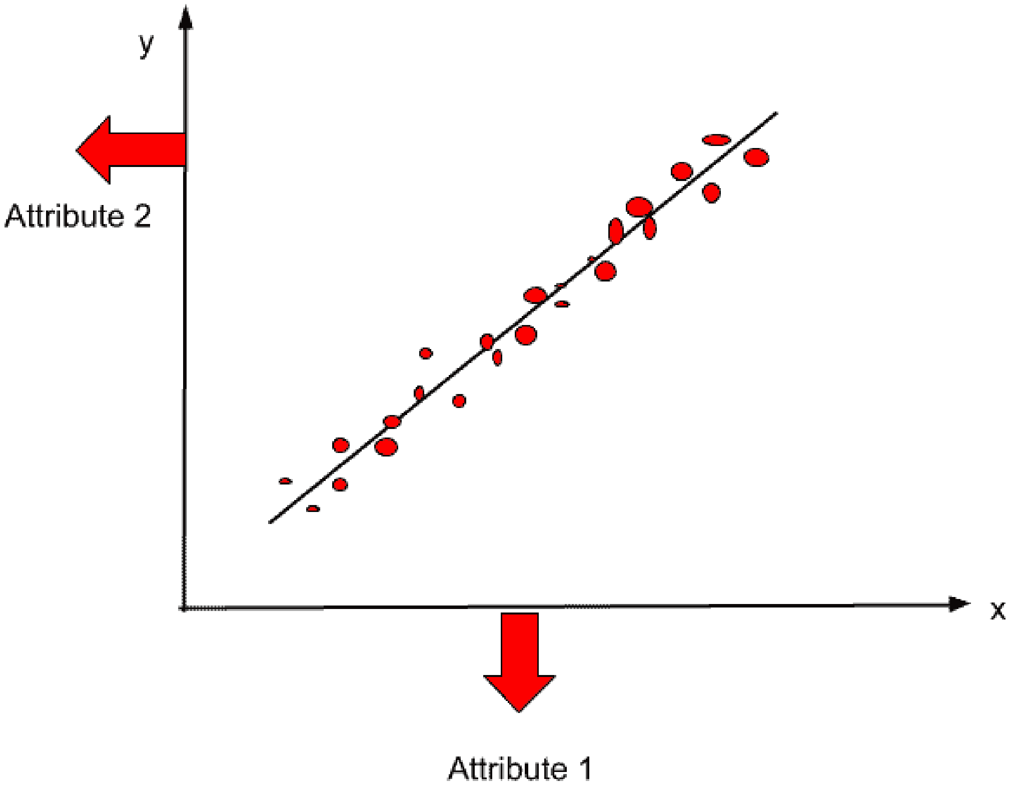
Assume we have given data in such a format for the training phase. It is just for your reference. Ok.



Age	Salary
40	20 LPA
25	8 LPA
35	15 LPA

The data has only two attributes the Age, and salary. We train our model on these two attributes. And after training, we get this hypothesis.

One attribute is on the x-axis, and the second one is on the y-axis.



Here, we found that our model is facing the overfitting problem. So to solve overfitting we use principal component analysis.

And for that work, we need to find a PC (Principal Components).

So, the next question is-

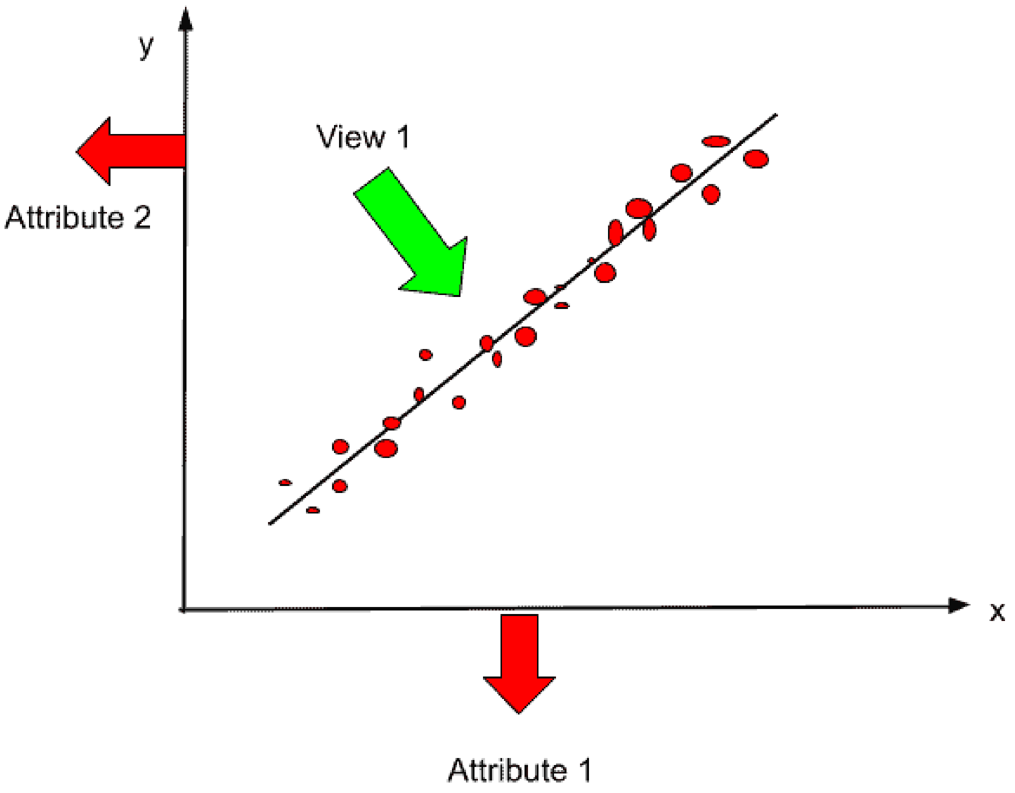
How to find PCs?

To find PCs, there is another term used, and that is **Views.**

Views are nothing but different angles from where you saw your data or hypothesis.

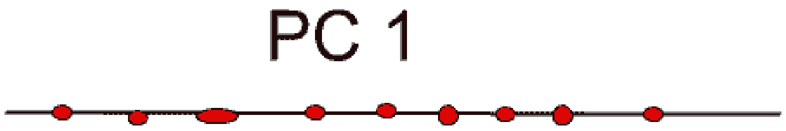
Suppose you saw your hypothesis from the top level, so how it looks?.

Let’ see the below image.



If I see my data points from that point, so I will see one line. And on that line, there will be some points mapped on that line. Just see the image below.

.



So what we have done here?.

Before we have two dimensions x and y. And now we have only one. So basically we reduce from two-dimension to one-dimension.

As I discussed the definition of PCA in the first section, that PCA reduces the **d-dimensional dataset into a k-dimensional subspace.**

So, here d-dimensional dataset is the two-dimensional dataset- ***age, and salary.*** **And PCA reduced it into a k-dimensional subspace that is a one-dimensional single line.**

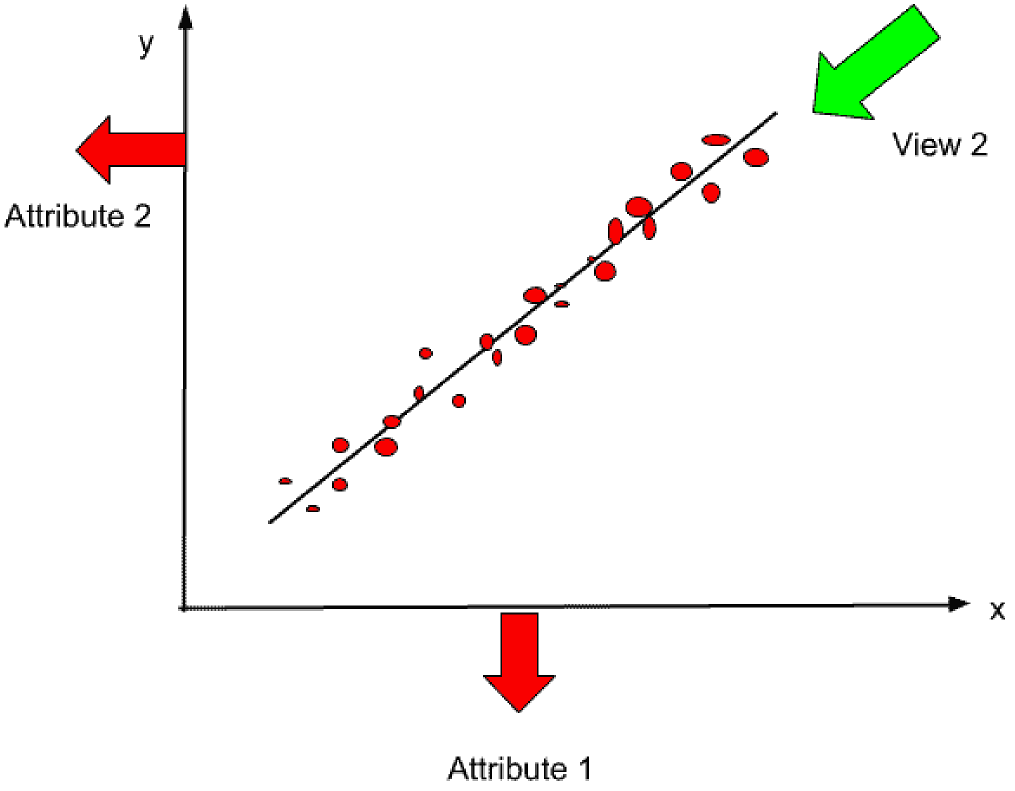
I hope, now you understand what PCA does.

In other words, we have reduced **dimensionality.**

You can call it PC1. Similarly, we can find more PCs by looking at different angles or views.

Let's see how to generate the Second PC?

If I see the hypothesis from that view, then let's see how it will look.



So from that view How it will look?.

Are you thinking that only one point will be seen?

If yes, then sorry you are thinking wrong. You will not see only a single dot point.

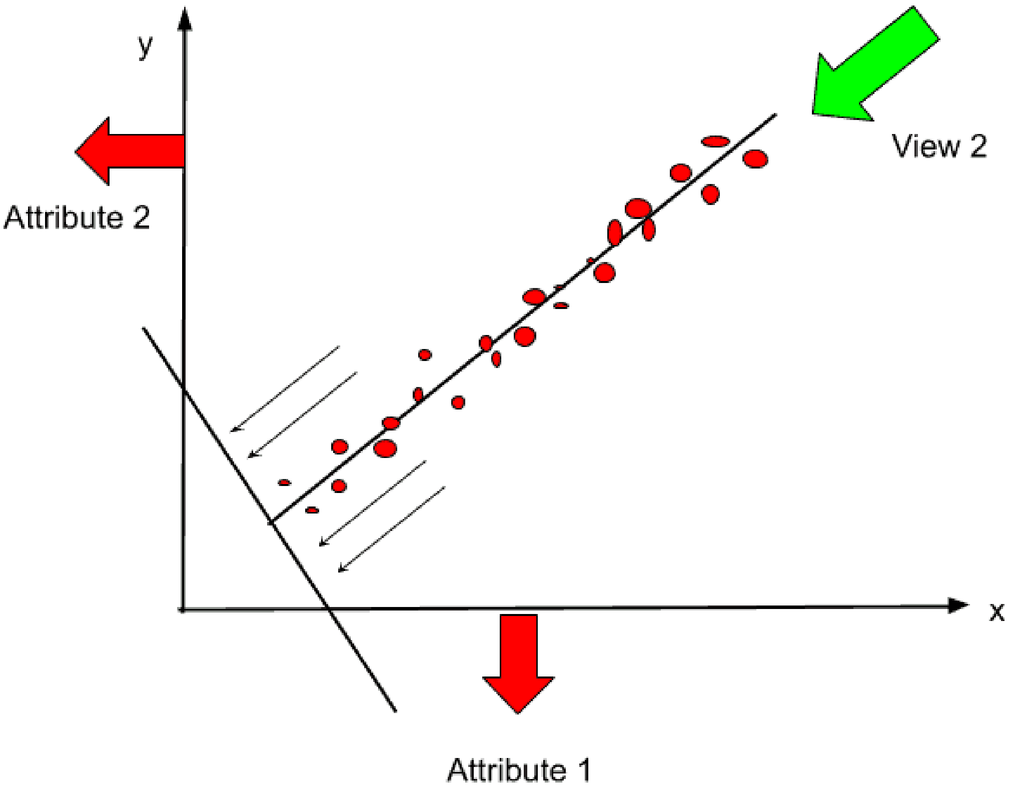
Now, you have a question Why we will not see only one point?

So to understand, we need to go into a little more detail.

We can imagine one line horizontal to this straight line, and all these points will be mapped on that line.

Confused?

Don’t worry. Just look at this image, and then you will understand.

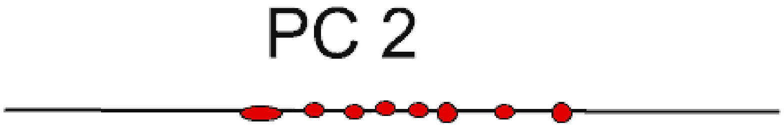


Now I hope you understand what I was saying.

Right?.

So, now when we plot this line, and all these points will be mapped on that line. So how it will look?.

Let’s see this image.



Very much similar to PC1. So as of now, we have generated two principal components PC1, and PC2.

The very important point you should keep in your mind is that the number of principal components can be less than or equal to the number of attributes.

That means your Principal Components (like PC1, and PC2) should be equal to or less than the attributes (in that example age, and salary).

Got it?.

Now, the next question is-

What to do when we have more than two attributes?

So in that case, we can also reduce the dimension in the same way.

Suppose we have generated 5 principal components PC1, PC2, PC3, PC4, and PC5. So whom to give more preference above 5?

The answer is PC1.

Always give preference to PC1 of any model. Then PC2, and so on.

Suppose in this example, we have generated two Principal Components PC1 and PC2. But we will give preference to PC1.

There is one more Property. And that is the **Orthogonal Property.**

So, What is Orthogonal Property?

There should be orthogonal property between PC1 and PC2. That means both PC1 and PC2 should be independent of each other. No one should be dependent on each other.

In simple words, PC1 should not dependent upon PC2 and vise versa.

All Principal Components should be totally independent.

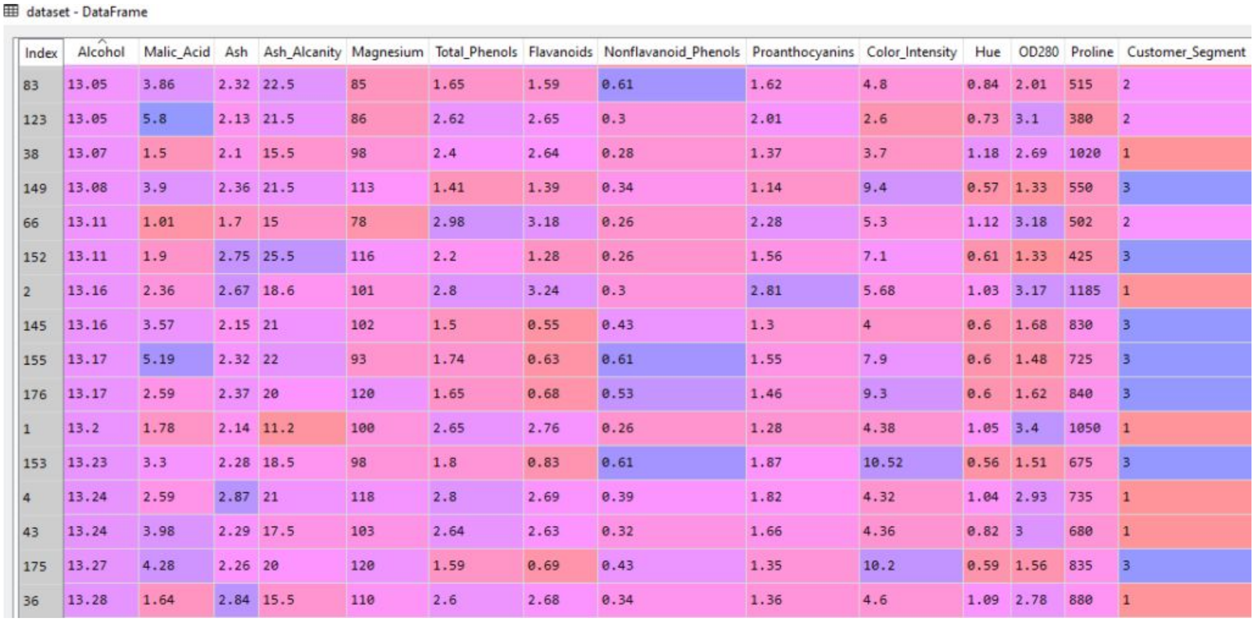
Now let's see how to implement PCA in Python-

Implementation of PCA in Python-

Before moving to the implementation part, I would like to tell you about the **Dataset and the Problem Statement.**

Dataset

This is our dataset-



dataset - DataFrame

Index	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	Hue	OD280	Proline	Customer_Segment
83	13.05	3.86	2.32	22.5	85	1.65	1.59	0.61	1.62	4.8	0.84	2.01	515	2
123	13.05	5.8	2.13	21.5	86	2.62	2.65	0.3	2.01	2.6	0.73	3.1	380	2
38	13.07	1.5	2.1	15.5	98	2.4	2.64	0.28	1.37	3.7	1.18	2.69	1020	1
149	13.08	3.9	2.36	21.5	113	1.41	1.39	0.34	1.14	9.4	0.57	1.33	550	3
66	13.11	1.01	1.7	15	78	2.98	3.18	0.26	2.28	5.3	1.12	3.18	502	2
152	13.11	1.9	2.75	25.5	116	2.2	1.28	0.26	1.56	7.1	0.61	1.33	425	3
2	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185	1
145	13.16	3.57	2.15	21	102	1.5	0.55	0.43	1.3	4	0.6	1.68	830	3
155	13.17	5.19	2.32	22	93	1.74	0.63	0.61	1.55	7.9	0.6	1.48	725	3
176	13.17	2.59	2.37	20	120	1.65	0.68	0.53	1.46	9.3	0.6	1.62	840	3
1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050	1
153	13.23	3.3	2.28	18.5	98	1.8	0.83	0.61	1.87	10.52	0.56	1.51	675	3
4	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735	1
43	13.24	3.98	2.29	17.5	103	2.64	2.63	0.32	1.66	4.36	0.82	3	680	1
175	13.27	4.28	2.26	20	120	1.59	0.69	0.43	1.35	10.2	0.59	1.56	835	3
36	13.28	1.64	2.84	15.5	110	2.6	2.68	0.34	1.36	4.6	1.09	2.78	880	1

This dataset has **13** features from **Alcohol to Proline.** Each feature is giving some information related to a certain wine. Each row corresponds to wine, and each wine has different features like **Alcohol level, malic acid level, etc.** So these are different wine characteristics given.

The last column, “**Customer_Segment**” represents different segments of wine. As you can see there are three segments 1, 2, and 3.

This dataset belongs to **UCI Machine Learning Repository.** You can download the dataset from **Kaggle.**

Problem Statement

Suppose this dataset belongs to a **Wine Merchant,** who has different bottles of wine to sell and has a large base of customers. And the wine merchant **hires you as a data scientist** and wants you to **reduce the complexity of this dataset**. And the owner would also want you to build a **predictive model**, trained on this data including the dependent variable column- “**Customer_Segment**“.

The objective of this problem is to, identify **which customer segment each wine belongs to.** So that you can recommend the wine to the right customer. In a nutshell, you have to build a **Wine Recommender system.**

This Recommender system will optimize the sale and increase the profit of Wine Shop.

I hope now you understand the dataset and the problem statement.

Now let’s move to the implementation part-

So, the first step is-

1. Import the Libraries

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

NumPy is an open-source Python library used to perform various **mathematical and scientific tasks.** NumPy is **used** for working with arrays. It also has functions for working in the domain of **linear algebra, Fourier transform, and matrices.**

Matplotlib is a plotting library, that is used for creating a **figure, plotting area in a figure, plot some lines in a plotting area, decorates the plot with labels, etc.**

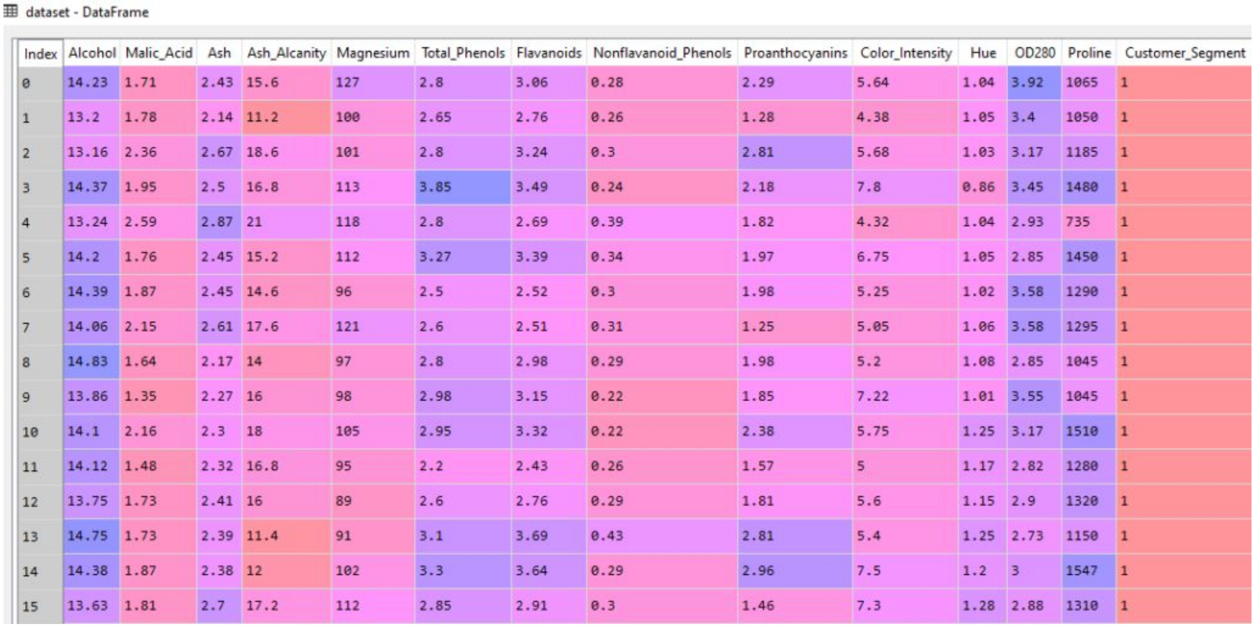
Pandas is a tool used for **data wrangling and analysis.**

So in step 1, we imported all required libraries. Now the next step is-

2. Load the Dataset

```
dataset = pd.read_csv('Wine.csv')
```

So, when you load the dataset after running this line of code, you will get your data something like this-



dataset - DataFrame

Index	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	Hue	OD280	Proline	Customer_Segment
0	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065	1
1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050	1
2	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185	1
3	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480	1
4	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735	1
5	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450	1
6	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290	1
7	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295	1
8	14.83	1.64	2.17	14	97	2.8	2.98	0.29	1.98	5.2	1.08	2.85	1045	1
9	13.86	1.35	2.27	16	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045	1
10	14.1	2.16	2.3	18	105	2.95	3.32	0.22	2.38	5.75	1.25	3.17	1510	1
11	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	5	1.17	2.82	1280	1
12	13.75	1.73	2.41	16	89	2.6	2.76	0.29	1.81	5.6	1.15	2.9	1320	1
13	14.75	1.73	2.39	11.4	91	3.1	3.69	0.43	2.81	5.4	1.25	2.73	1150	1
14	14.38	1.87	2.38	12	102	3.3	3.64	0.29	2.96	7.5	1.2	3	1547	1
15	13.63	1.81	2.7	17.2	112	2.85	2.91	0.3	1.46	7.3	1.28	2.88	1310	1

As you can see in the dataset, there are **13 independent variables** and **one dependent variable** present. But we have to separate independent variables and dependent variables.

So for this, we perform our next step-

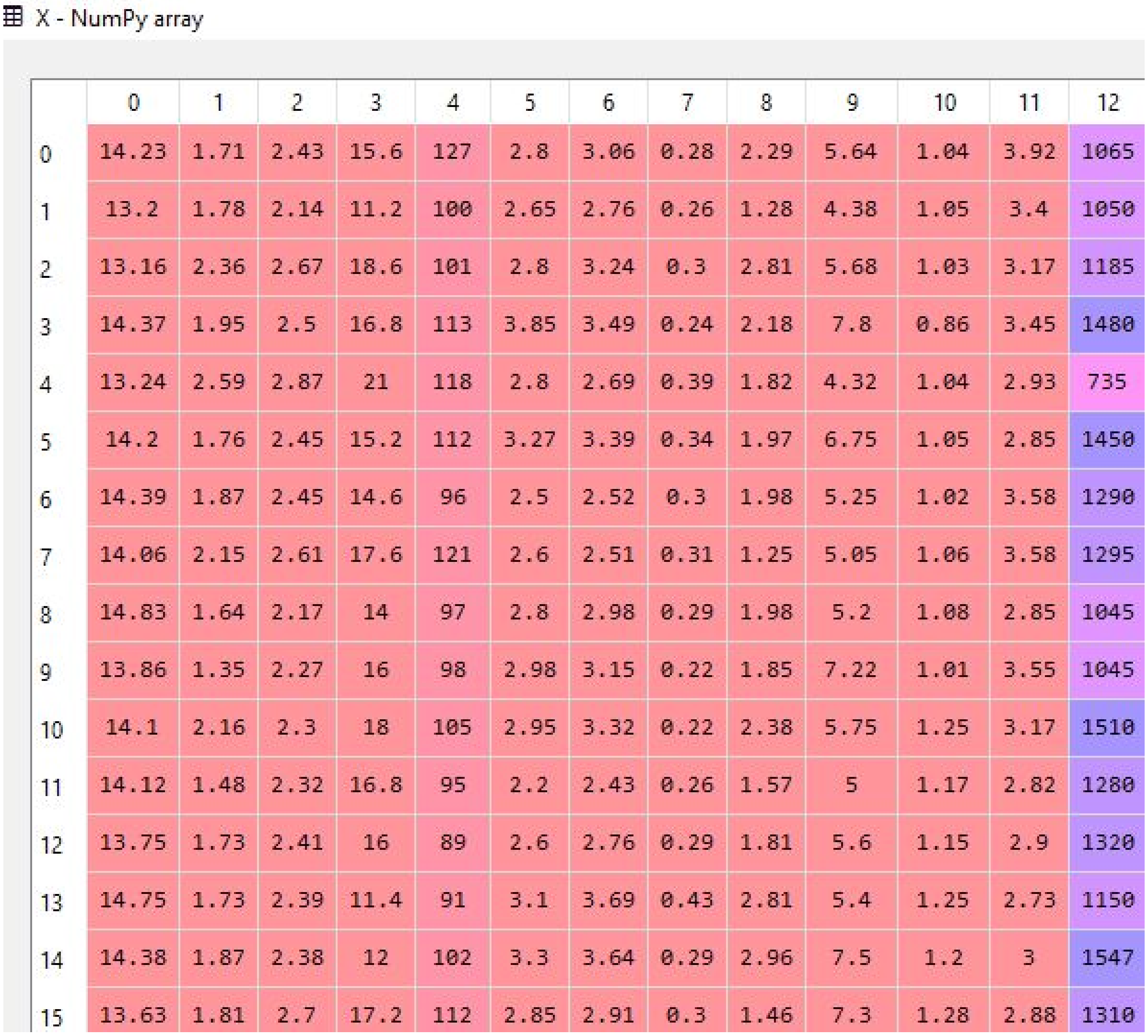
3. Split Dataset into X and Y

```
X = dataset.iloc[:, 0:13].values
```

```
y = dataset.iloc[:, 13].values
```

When you run these lines, you get two separate tables X and Y. Something like this-

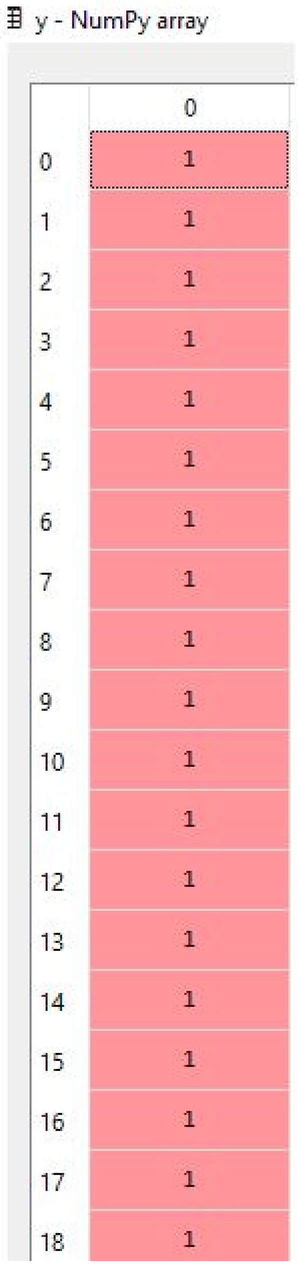
Independent Variables (X)-



X - NumPy array

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
2	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480
4	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735
5	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290
7	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295
8	14.83	1.64	2.17	14	97	2.8	2.98	0.29	1.98	5.2	1.08	2.85	1045
9	13.86	1.35	2.27	16	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045
10	14.1	2.16	2.3	18	105	2.95	3.32	0.22	2.38	5.75	1.25	3.17	1510
11	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	5	1.17	2.82	1280
12	13.75	1.73	2.41	16	89	2.6	2.76	0.29	1.81	5.6	1.15	2.9	1320
13	14.75	1.73	2.39	11.4	91	3.1	3.69	0.43	2.81	5.4	1.25	2.73	1150
14	14.38	1.87	2.38	12	102	3.3	3.64	0.29	2.96	7.5	1.2	3	1547
15	13.63	1.81	2.7	17.2	112	2.85	2.91	0.3	1.46	7.3	1.28	2.88	1310

Dependent Variable(Y)–



y - NumPy array

	0
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1

Now we have divided our dataset into **X and Y.** So the next step is-

4. Split the X and Y Dataset into the Training set and Test set

For building a machine learning model, we need to train our model on the training set. And for checking the performance of our model, we use a Test set. That’s why we have to split the X and Y datasets into the **Training set and Test set**.

```
from sklearn.model_selection import train_test_split
```

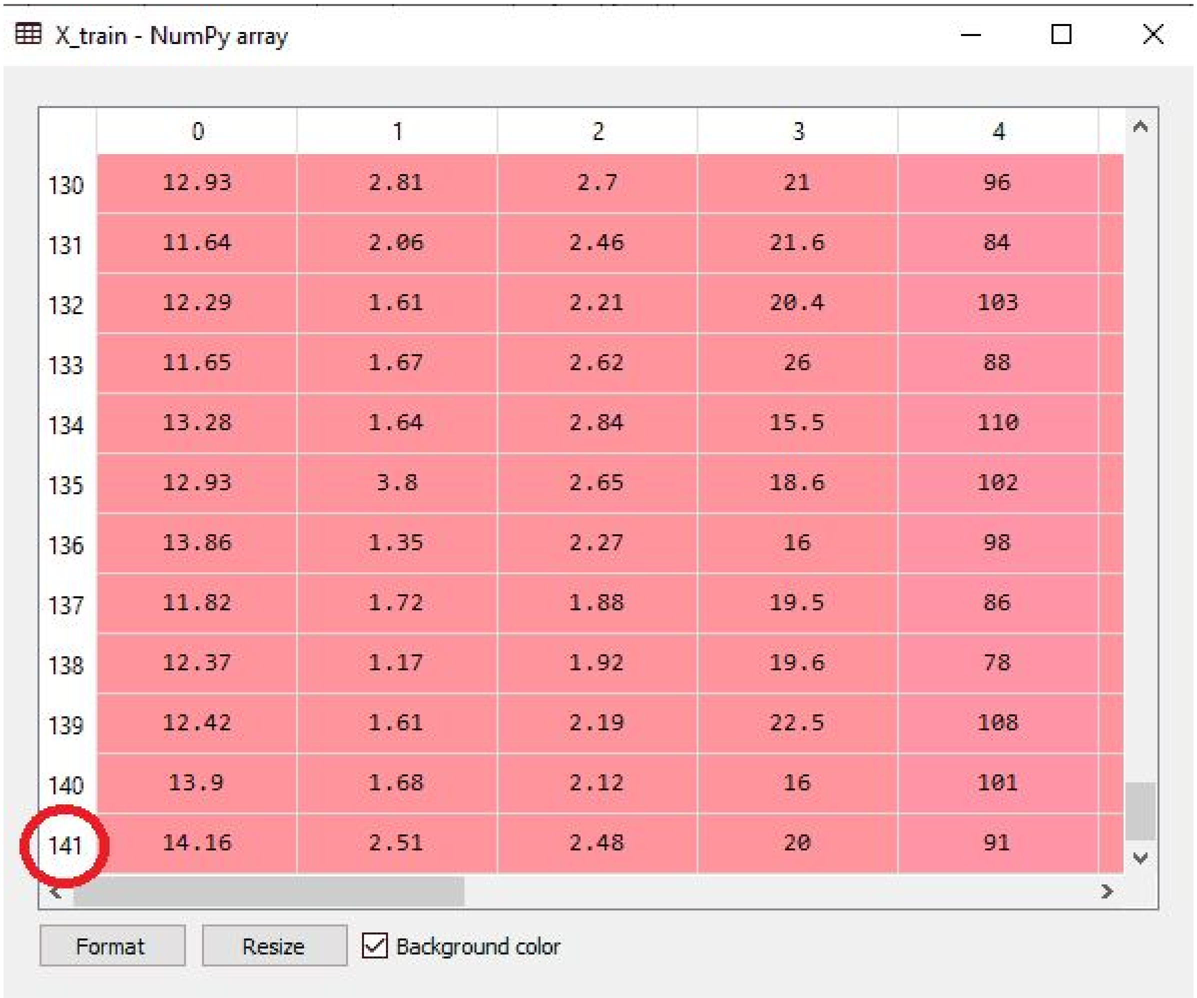
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 0)
```

While splitting into training and test set, you have to remember that, 80%-90% of your data should be in the training tests. And that’s why I write **test_size = 0.2.**

I hope now you understood.

So, when you run these lines, you get 4 different tables- **X_train, X_test, y-train, and y_test.**

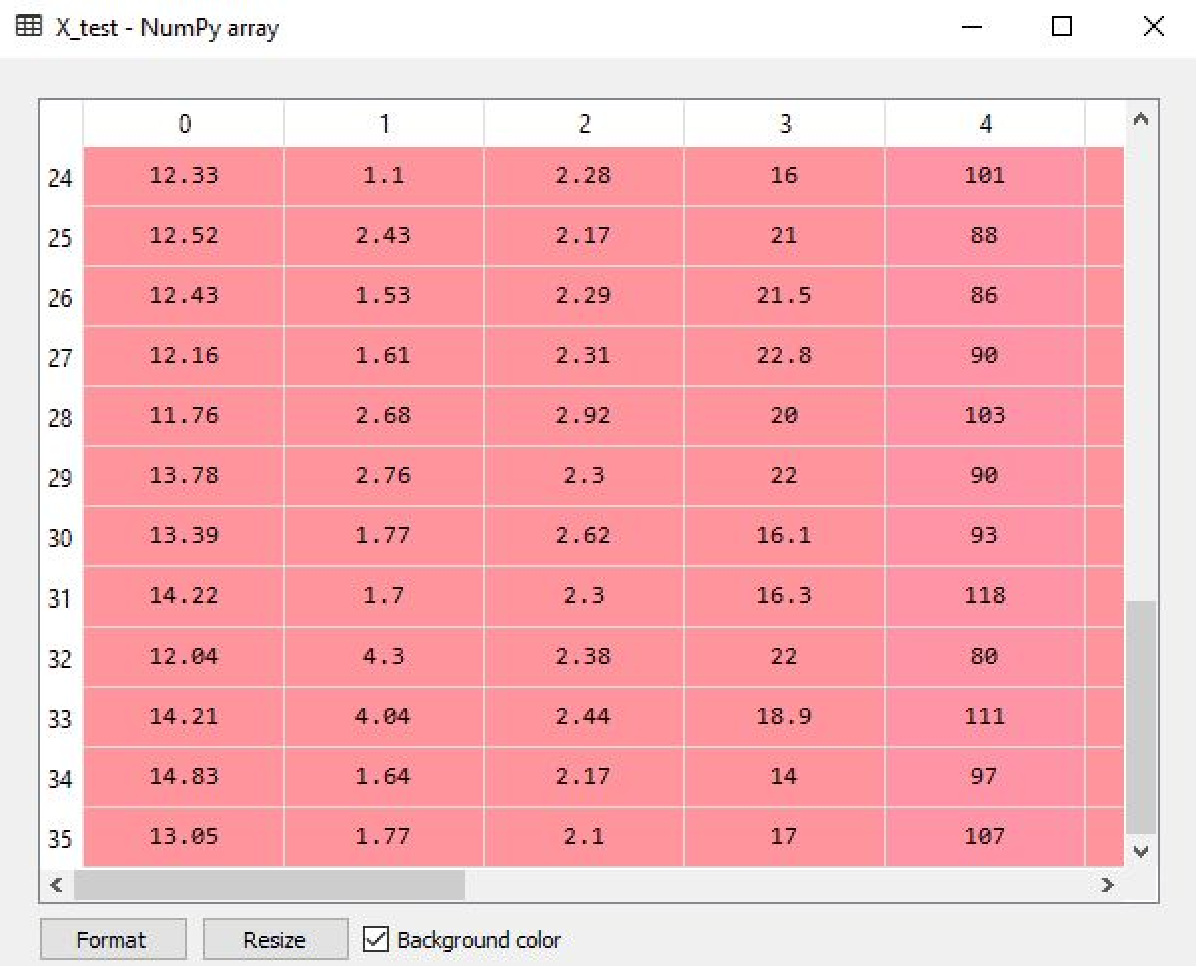
As you can see here, in **X_train we have 141 rows-**



X_train - NumPy array

	0	1	2	3	4
130	12.93	2.81	2.7	21	96
131	11.64	2.06	2.46	21.6	84
132	12.29	1.61	2.21	20.4	103
133	11.65	1.67	2.62	26	88
134	13.28	1.64	2.84	15.5	110
135	12.93	3.8	2.65	18.6	102
136	13.86	1.35	2.27	16	98
137	11.82	1.72	1.88	19.5	86
138	12.37	1.17	1.92	19.6	78
139	12.42	1.61	2.19	22.5	108
140	13.9	1.68	2.12	16	101
141	14.16	2.51	2.48	20	91

And in **X_test, we have only 35 rows-**



	0	1	2	3	4
24	12.33	1.1	2.28	16	101
25	12.52	2.43	2.17	21	88
26	12.43	1.53	2.29	21.5	86
27	12.16	1.61	2.31	22.8	90
28	11.76	2.68	2.92	20	103
29	13.78	2.76	2.3	22	90
30	13.39	1.77	2.62	16.1	93
31	14.22	1.7	2.3	16.3	118
32	12.04	4.3	2.38	22	80
33	14.21	4.04	2.44	18.9	111
34	14.83	1.64	2.17	14	97
35	13.05	1.77	2.1	17	107

Now, we have split our dataset into **X_train, X_test, y-train, and y_test.** The next step is-

Perform Feature Scaling

As you can see in the dataset, all values are not in the same range. And that requires a lot of time for calculation. So to overcome this problem, we perform **feature scaling.**

Feature scaling help us to **normalize the data within a particular range.**

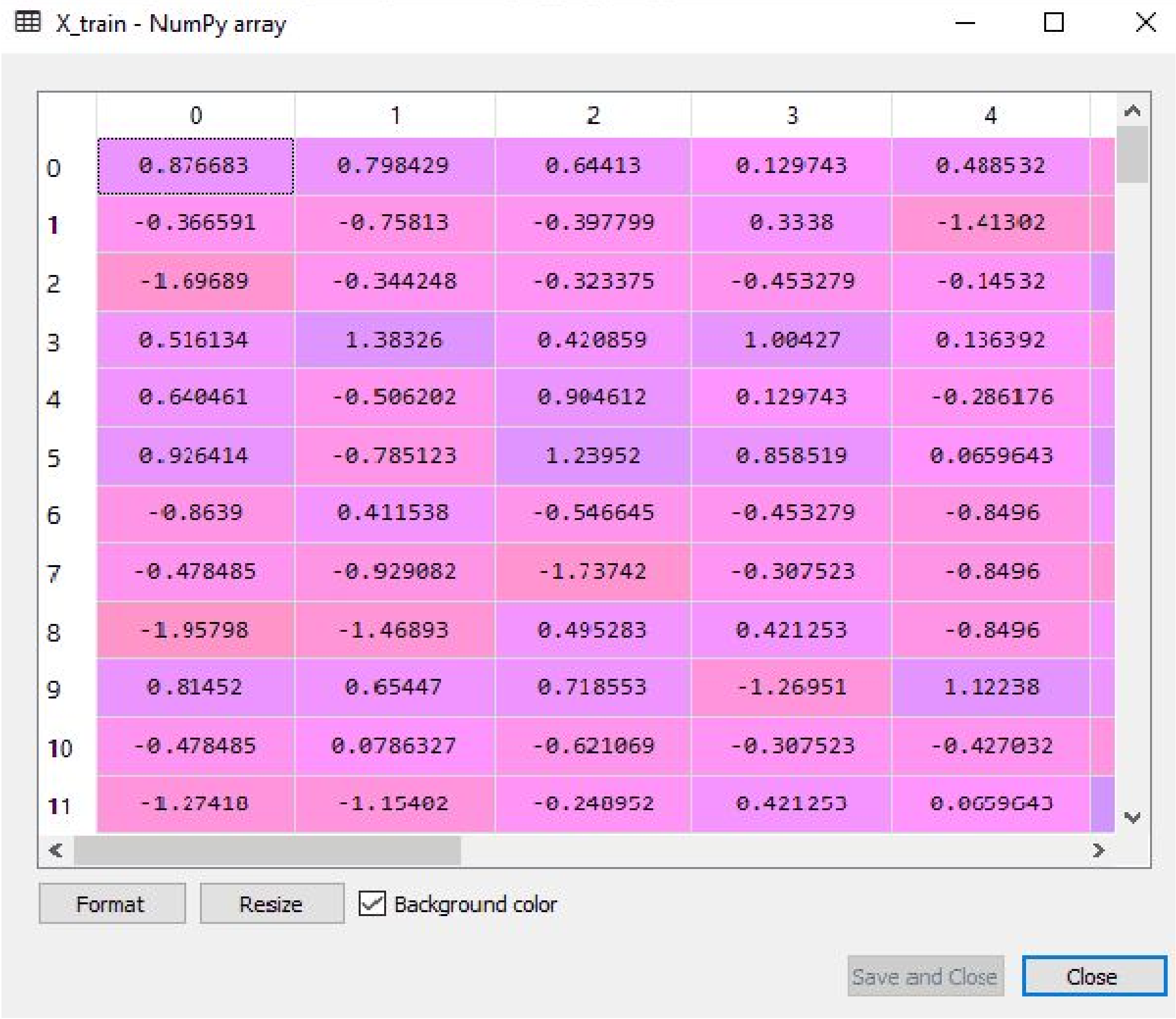
```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

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

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

After performing feature scaling, all values are normalized and looks something like this-



	0	1	2	3	4
0	0.876683	0.798429	0.64413	0.129743	0.488532
1	-0.366591	-0.75813	-0.397799	0.3338	-1.41302
2	-1.69689	-0.344248	-0.323375	-0.453279	-0.14532
3	0.516134	1.38326	0.420859	1.00427	0.136392
4	0.640461	-0.506202	0.904612	0.129743	-0.286176
5	0.926414	-0.785123	1.23952	0.858519	0.0659643
6	-0.8639	0.411538	-0.546645	-0.453279	-0.8496
7	-0.478485	-0.929082	-1.73742	-0.307523	-0.8496
8	-1.95798	-1.46893	0.495283	0.421253	-0.8496
9	0.81452	0.65447	0.718553	-1.26951	1.12238
10	-0.478485	0.0786327	-0.621069	-0.307523	-0.427032
11	-1.27418	-1.15402	-0.248952	0.421253	0.0659643

Now, we are done with data preprocessing steps. It’s time to apply PCA to our dataset-

5. Apply PCA

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 2)
```

```
X_train = pca.fit_transform(X_train)
```

```
X_test = pca.transform(X_test)
```

n_components is the number of extracted features or principal components. You can change this number based on the results. So let’s test **n_components as 2**. If we will get good results, then we will keep it up, otherwise, we will change and check the performance of our model.

We apply PCA to our training set as well as the test set. But did you notice, I didn’t write the **fit_transform method in X_test**?

Do you know why…?

Because to avoid **information leakage on the test set**. The test set should be new observations, on which we test our model performance. If we perform fit_transform on X_test, that means we are giving some hints to the test set.

I hope you understood the reason.

So after applying PCA, the next step is-

6. Fit Logistic Regression to the Training set

I am going to use **Logistic Regression**, but you can use any **other classification algorithms**. It’s up to you. Even for practice, just try to use other classification algorithms and compare the accuracies of different algorithms.

Always make sure, that **you have to apply PCA before fitting any machine learning algorithm.** Why…? Because we train our model on the low dimension dataset. That's why I applied PCA first.

So, let's fit logistic regression to the training set-

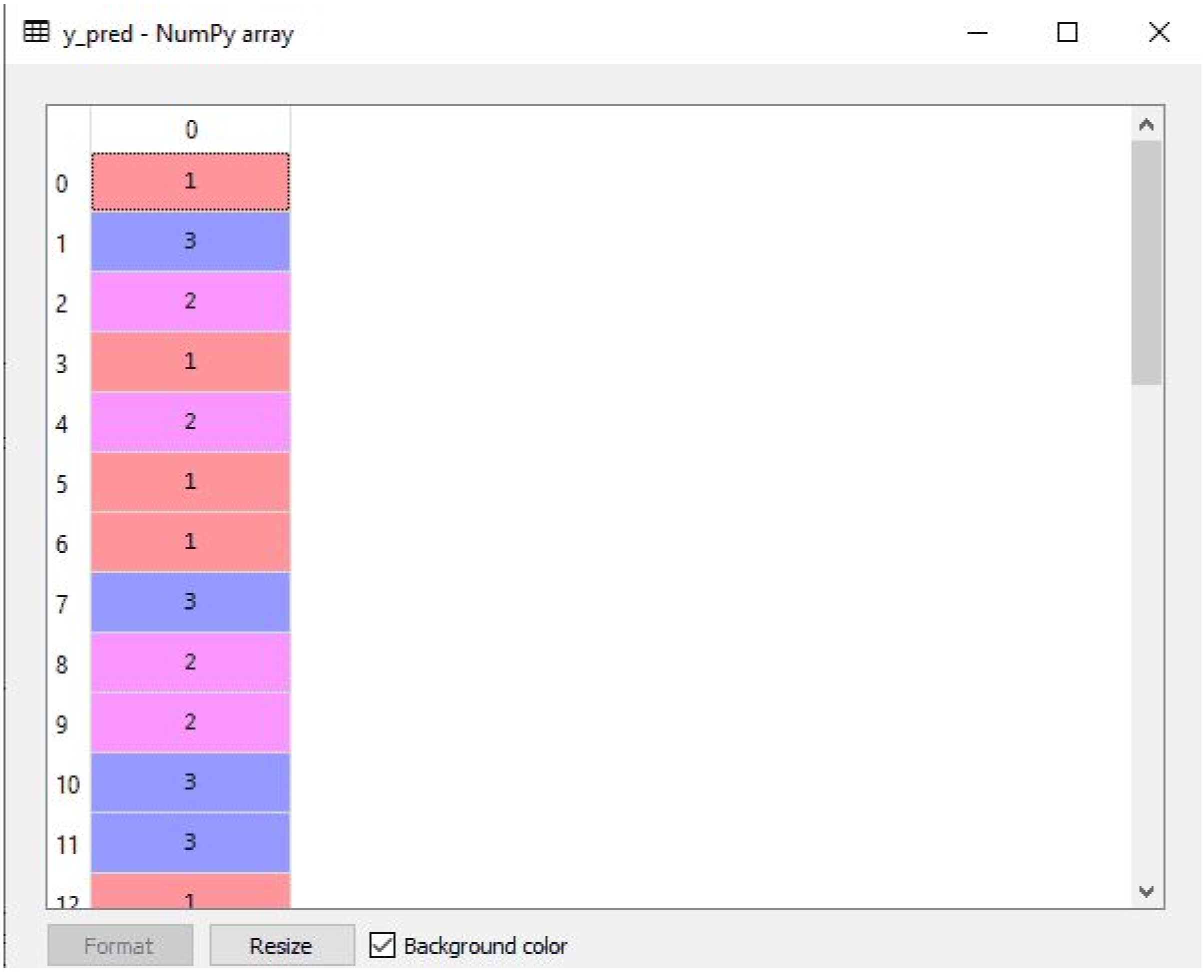
```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train)
```

Now, all done. It's time to predict the Test set. So the next step is-

7. Predict the Test Set Results

```
y_pred = classifier.predict(X_test)
```

When you run this line of code, you will get y_pred, something like this-



	0
0	1
1	3
2	2
3	1
4	2
5	1
6	1
7	3
8	2
9	2
10	3
11	3
12	1

☑ Background color

But can you explain by looking at these predicted values, how many values are predicted right, and how many values are predicted wrong?

For a small dataset, you can. But when we have a large dataset, it’s quite impossible. And that’s why we use a **confusion matrix**, to clear our confusion.

So, the next step is-

8. Make the Confusion Matrix

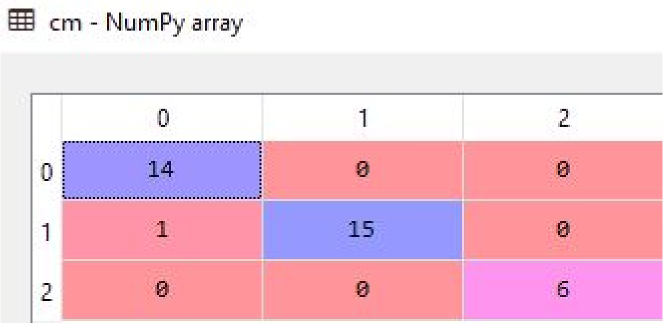
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)
```

```
accuracy_score(y_test,y_pred)
```

So, when you run these lines of code, you get confusion matrix, that looks something like this-



cm - NumPy array

	0	1	2
0	14	0	0
1	1	15	0
2	0	0	6

Here we got a confusion matrix of 3 rows and 3 columns because we have 3 classes or customer_segments.

Our model does only one wrong prediction. And that’s why our model accuracy is **0.9722222222222222 (97%).** Amazing…Right?.

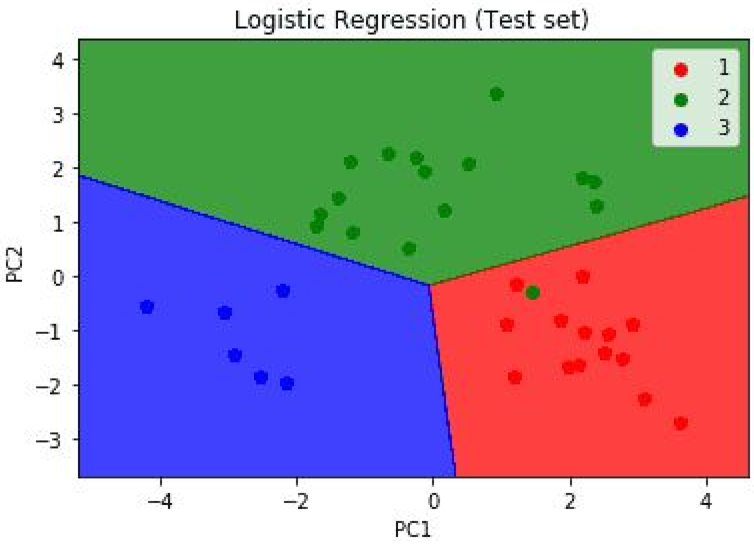
Now it’s time to showcase our findings in a visual form. So the next step is-

9. Visualise the Test set results

```
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(('red', 'green', 'blue')))

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 (Test set)')

plt.xlabel('PC1')

plt.ylabel('PC2')
plt.legend()
```

```
plt.show()
```

So, after running this code, you will get your visual results-



As you can see, there is only **one incorrect prediction.** The green dot showing in the red region. This is a great prediction.

And here we go…Congratulation! You build your **Wine Recommender System Using Principal Component Analysis with 97% accuracy.**

Now let's move to the next Dimensionality reduction technique- **Linear Discriminant Analysis**

What is a Linear Discriminant Analysis?

Linear Discriminant Analysis is a method of Dimensionality Reduction. The goal of LDA is to project a dataset onto a lower-dimensional space. It sounds similar to PCA. Right?

But LDA is different from PCA. Linear Discriminant Analysis finds the area that maximizes the separation between multiple classes. That is not done in PCA.

So, the definition of LDA is- **LDA project a feature space (N-dimensional data) onto a smaller subspace k(k<= n-1) while maintaining the class discrimination information.**

PCA is known as **Unsupervised** but LDA is **supervised** because of the relation to the dependent variable.

Now, let's see how LDA works-

How Linear Discriminant Analysis Works?

LDA works in the following steps-

Step 1-

Compute the d-dimensional mean vectors for the different classes from the dataset.

Step 2-

Compute **within class Scatter matrix (Sw).**

Suppose we have a 2-D dataset C1 and C2. So to calculate Sw for the 2-D dataset, the formula of Sw is-

$$Sw = S1+ S2$$

S1 is the covariance matrix for the class C1 and S2 is the covariance matrix for the class for C2.

Now, the formula of **covariance matrix** S1 is-

$$S1= \Sigma (x-u1).(x-u1)^T$$

Where u1 is the **mean** of class C1. Similarly, you can calculate S2 and C2.

Step 3-

Compute **between class Scatter Matrix (Sb)**

The formula for calculating Sb is-

$$Sb= (u1-u2).(u1-u2)^T$$

Step 4-

Compute the eigenvectors (e1,e2, e3,……ed) and corresponding eigenvalues (λ1, λ2,,…… λd) for the scatter matrix.

Step 5–

Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d X k dimensional matrix W.

Step 6-

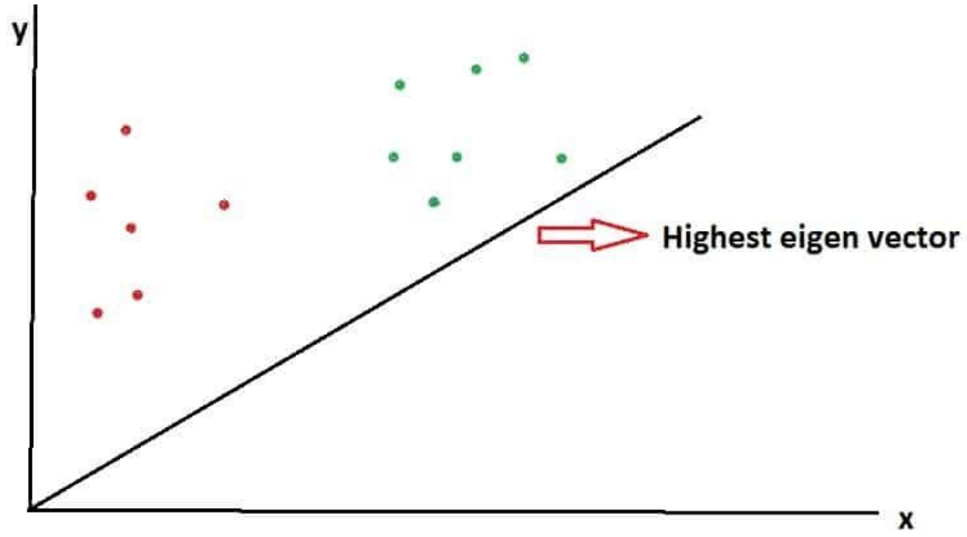
Reduce the Dimension

$$y= W^T. X$$

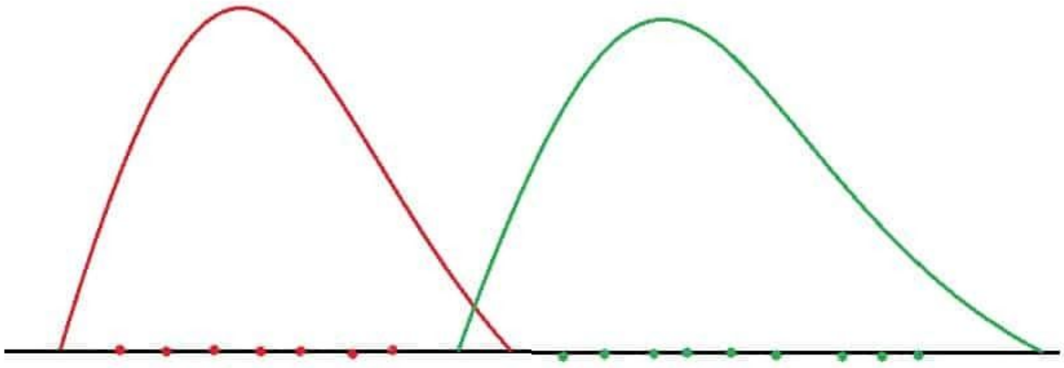
Where W^T is the projection vector and X is the input data sample. Here, the projection vector corresponds to the highest Eigenvalue.

So, let's visualize the whole working of LDA-

Suppose, this black line is the highest eigenvector, and red and green dots are two different classes.



When data points are projected onto this vector, so the dimensionality is reduced as well as the discrimination between the classes is also visualized.



In that image, Red represents one class and green represents the second class. So, by applying LDA, the dimension is reduced as well as the separation between two classes is also maximized.

I hope, now you understood the whole working of LDA. Now, let's see how to implement Linear Discriminant Analysis in Python.

Implementation of Linear Discriminant Analysis in Python

For this implementation, I am going to use the same Wine Dataset. You can download the dataset from here.

Our objective is to identify different customer segments based on several wine features available. So, the shop owner of Wine shop can recommend wine according to the customer segment.

Now, let's start with the first step-

1- Import important libraries

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd
```

2- Load the Dataset

```
dataset = pd.read_csv('Wine.csv')
```

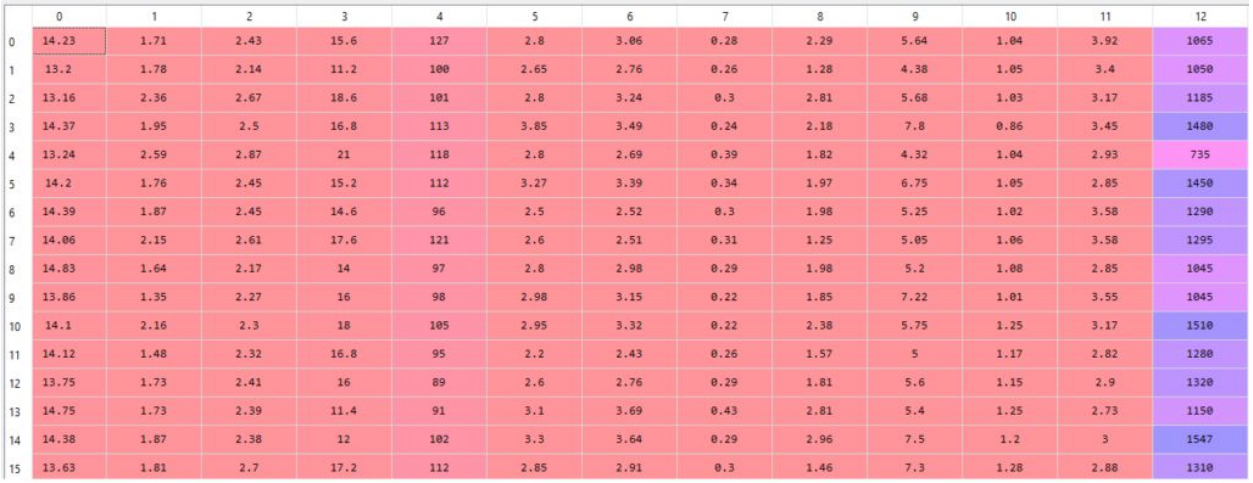
```
X = dataset.iloc[:, 0:13].values
```

```
y = dataset.iloc[:, 13].values
```

After splitting the dataset into X and Y, we will get something like that-

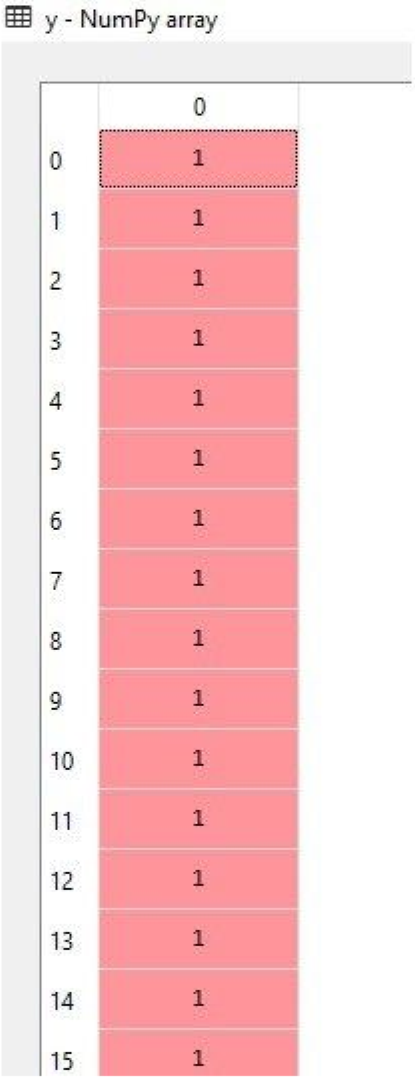
Here X is independent variables and Y is dependent variable. Y is dependent because the prediction of y depends upon X values.

X-



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
2	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480
4	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735
5	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290
7	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295
8	14.83	1.64	2.17	14	97	2.8	2.98	0.29	1.98	5.2	1.08	2.85	1045
9	13.86	1.35	2.27	16	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045
10	14.1	2.16	2.3	18	105	2.95	3.32	0.22	2.38	5.75	1.25	3.17	1510
11	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	5	1.17	2.82	1280
12	13.75	1.73	2.41	16	89	2.6	2.76	0.29	1.81	5.6	1.15	2.9	1320
13	14.75	1.73	2.39	11.4	91	3.1	3.69	0.43	2.81	5.4	1.25	2.73	1150
14	14.38	1.87	2.38	12	102	3.3	3.64	0.29	2.96	7.5	1.2	3	1547
15	13.63	1.81	2.7	17.2	112	2.85	2.91	0.3	1.46	7.3	1.28	2.88	1310

Y-



y - NumPy array

	0
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1

Now, the next step is-

3- Split the dataset into the Training set and Test 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)
```

Here, we are dividing the dataset into the Training set and Test set. That means, we use maximum data to train the model and separate some data for testing.

4- Apply Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

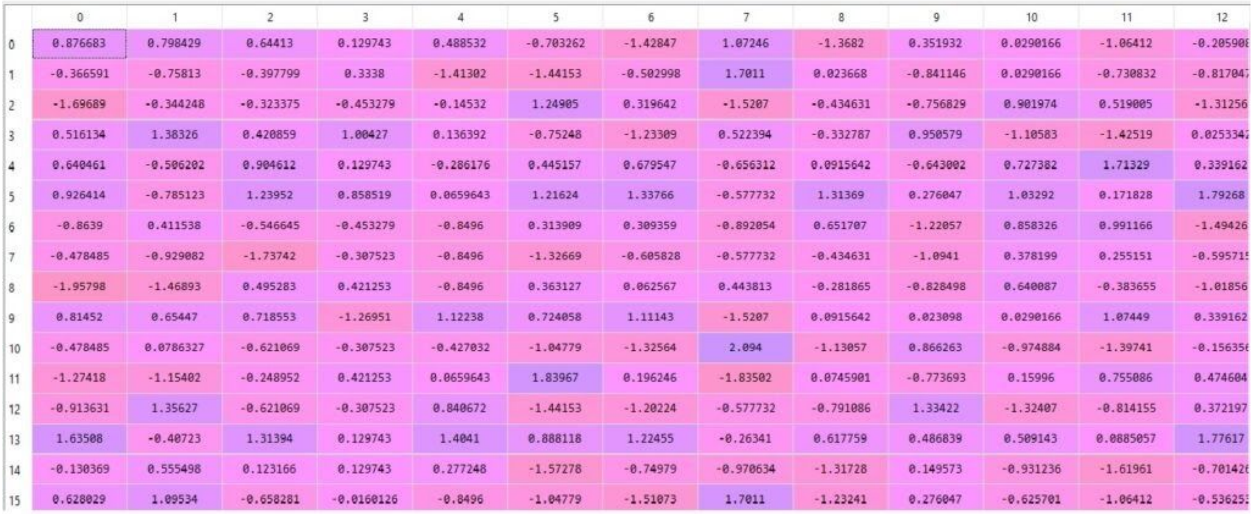
```
sc = StandardScaler()
```

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

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

Feature scaling is an important step to perform. After applying feature scaling, we will get our data in this form-

X train-



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.876683	0.798429	0.64413	0.129743	0.488532	-0.703262	-1.42847	1.07246	-1.3682	0.351932	0.0290166	-1.06412	-0.20590
1	-0.366591	-0.75813	-0.397799	0.3338	-1.41302	-1.44153	-0.502998	1.7011	0.023668	-0.841146	0.0290166	-0.730832	-0.81704
2	-1.69689	-0.344248	-0.323375	-0.453279	-0.14532	1.24905	0.319642	-1.5207	-0.434631	-0.756829	0.901974	0.519005	-1.31256
3	0.516134	1.38326	0.420859	1.00427	0.136392	-0.75248	-1.23309	0.522394	-0.332787	0.950579	-1.10583	-1.42519	0.025334
4	0.640461	-0.506202	0.904612	0.129743	-0.286176	0.445157	0.679547	-0.656312	0.0915642	-0.643002	0.727382	1.71329	0.339162
5	0.926414	-0.785123	1.23952	0.858519	0.0659643	1.21624	1.33766	-0.577732	1.31369	0.276047	1.03292	0.171828	1.79268
6	-0.8639	0.411538	-0.546645	-0.453279	-0.8496	0.313909	0.309359	-0.892054	0.651707	-1.22057	0.858326	0.991166	-1.49426
7	-0.478485	-0.929082	-1.73742	-0.307523	-0.8496	-1.32669	-0.605828	-0.577732	-0.434631	-1.0941	0.378199	0.255151	-0.59571
8	-1.95798	-1.46893	0.495283	0.421253	-0.8496	0.363127	0.062567	0.443813	-0.281865	-0.828498	0.640087	-0.383655	-1.01856
9	0.81452	0.65447	0.718553	-1.26951	1.12238	0.724058	1.11143	-1.5207	0.0915642	0.023098	0.0290166	1.07449	0.339162
10	-0.478485	0.0786327	-0.621069	-0.307523	-0.427032	-1.04779	-1.32564	2.094	-1.13057	0.866263	-0.974884	-1.39741	-0.15635
11	-1.27418	-1.15402	-0.248952	0.421253	0.0659643	1.83967	0.196246	-1.83502	0.0745901	-0.773693	0.15996	0.755086	0.474604
12	-0.913631	1.35627	-0.621069	-0.307523	0.840672	-1.44153	-1.20224	-0.577732	-0.791086	1.33422	-1.32407	-0.814155	0.372197
13	1.63508	-0.40723	1.31394	0.129743	1.4041	0.888118	1.22455	-0.26341	0.617759	0.486839	0.509143	0.0885057	1.77617
14	-0.130369	0.555498	0.123166	0.129743	0.277248	-1.57278	-0.74979	-0.970634	-1.31728	0.149573	-0.931236	-1.61961	-0.70142
15	0.628029	1.09534	-0.658281	-0.0160126	-0.8496	-1.04779	-1.51073	1.7011	-1.23241	0.276047	-0.625701	-1.06412	-0.53625

Here the values are scaled. After applying feature scaling, it’s time to apply Linear Discriminant Analysis (LDA).

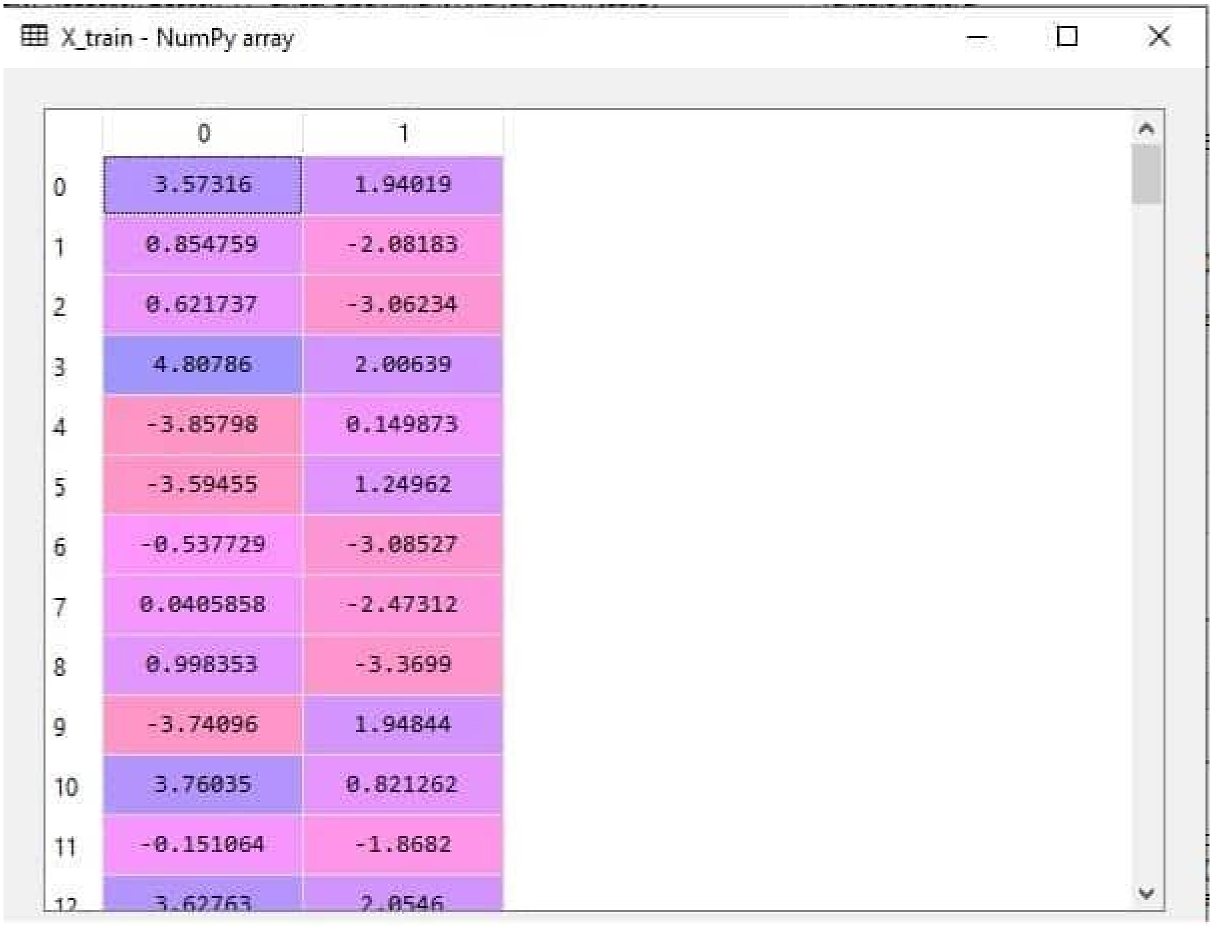
5. Apply LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA

lda = LDA(n_components = 2)

X_train = lda.fit_transform(X_train, y_train)

X_test = lda.transform(X_test)
```

Here, **n_components = 2** represents the number of extracted features. That means we are using only 2 features from all the features. And these two features will give best result.

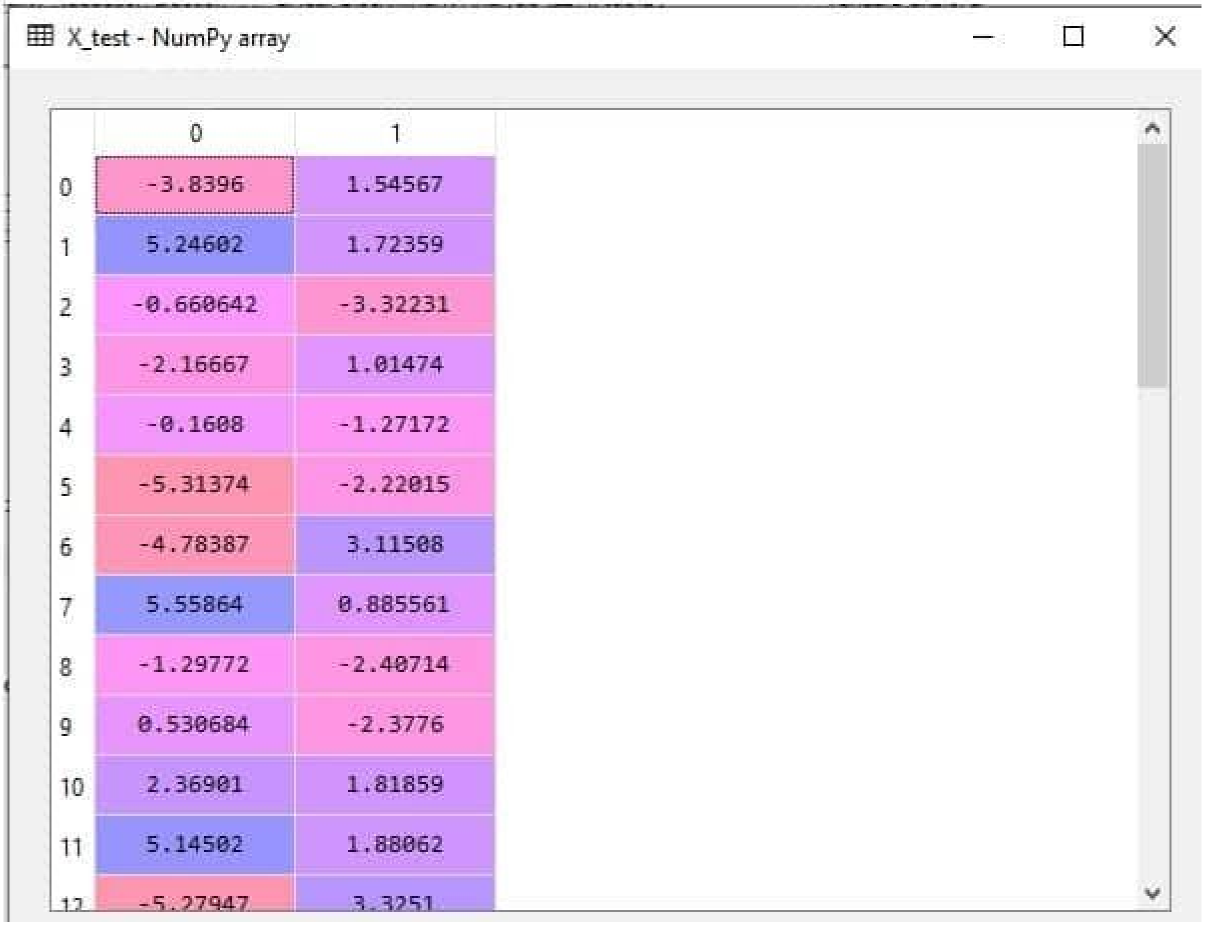
So, after applying LDA, we will get X_train and X_test something like that-

X_train-



	0	1
0	3.57316	1.94019
1	0.854759	-2.08183
2	0.621737	-3.06234
3	4.80786	2.00639
4	-3.85798	0.149873
5	-3.59455	1.24962
6	-0.537729	-3.08527
7	0.0405858	-2.47312
8	0.998353	-3.3699
9	-3.74096	1.94844
10	3.76035	0.821262
11	-0.151064	-1.8682
12	3.62763	2.0546

X_ Test-



	0	1
0	-3.8396	1.54567
1	5.24602	1.72359
2	-0.660642	-3.32231
3	-2.16667	1.01474
4	-0.1608	-1.27172
5	-5.31374	-2.22015
6	-4.78387	3.11508
7	5.55864	0.885561
8	-1.29772	-2.40714
9	0.530684	-2.3776
10	2.36901	1.81859
11	5.14502	1.88062
12	-5.27947	3.3251

After applying LDA, now it's time to apply any Classification algorithm. Here I am using Logistic Regression. But you can use any other classification algorithm and check the accuracy.

6. Fit Logistic Regression to the Training set

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

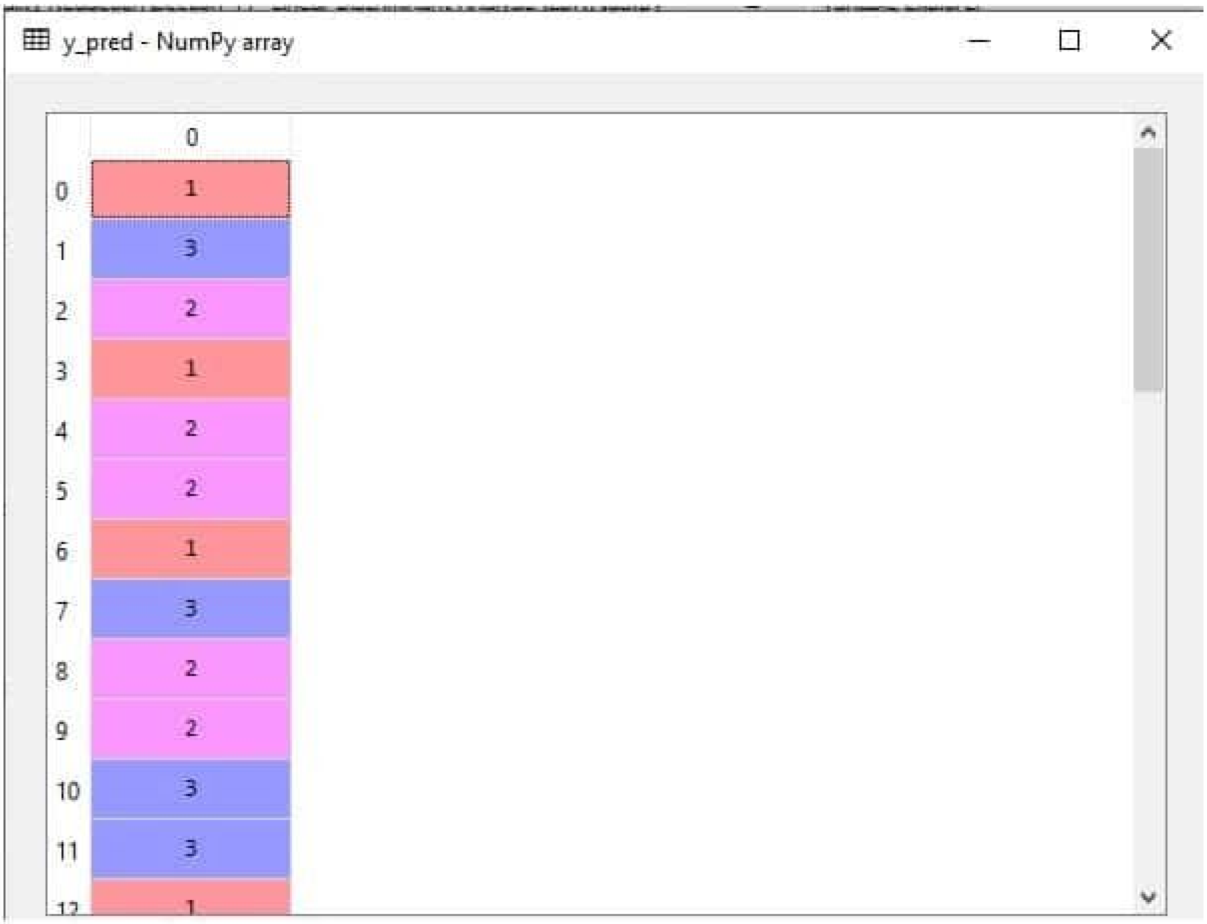
NOTE- Always apply LDA first before applying classification algorithm.

Now, it’s time to predict the result.

7. Predict the Test set results

```
y_pred = classifier.predict(X_test)
```

After running this code, we will get Y_Pred something like that-



	0
0	1
1	3
2	2
3	1
4	2
5	2
6	1
7	3
8	2
9	2
10	3
11	3
12	1

8. Check the accuracy by Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test,y_pred)
```

We got this confusion matrix and accuracy score, that is superb! We got **100%** accuracy.

```
[14 0 0]

[ 0 16 0]

[ 0 0 6]

Out: 1.0
```

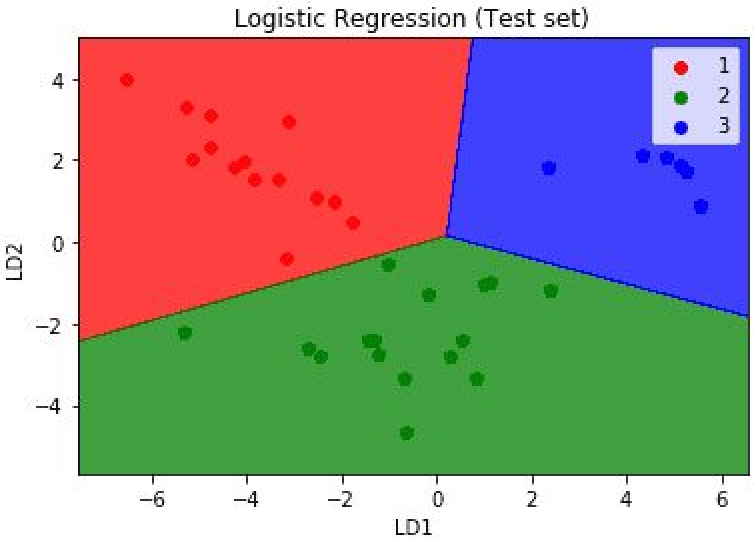
Now, let’s visualize the Test set result-

9. Visualize the Test set results

```
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(('red',
'green', 'blue')))

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 (Test set)')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.legend()
plt.show()
```

So, after running this code, we will get



Here, you can see all three classes are falling into the correct region. There is no incorrect result.

I hope, you understood the whole working procedure of LDA.

So in this PDF, you understood the 2 most popular dimensionality reduction techniques.

I hope you found this guide useful.

For more machine learning and data science-related tutorials-

Visit- https://www.mltut.com/

Happy Learning