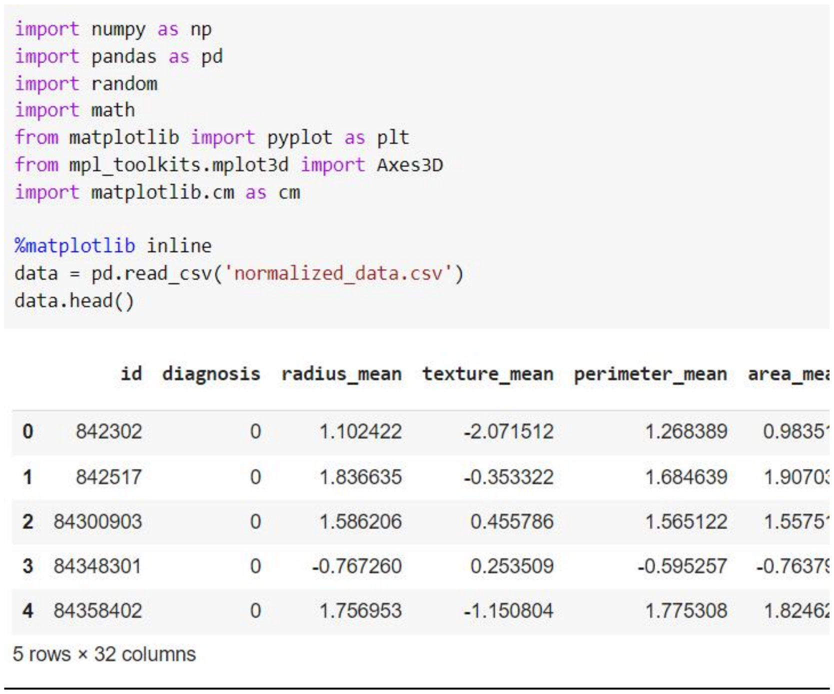
Part B : Fisher's linear discriminant model (FLDM1)



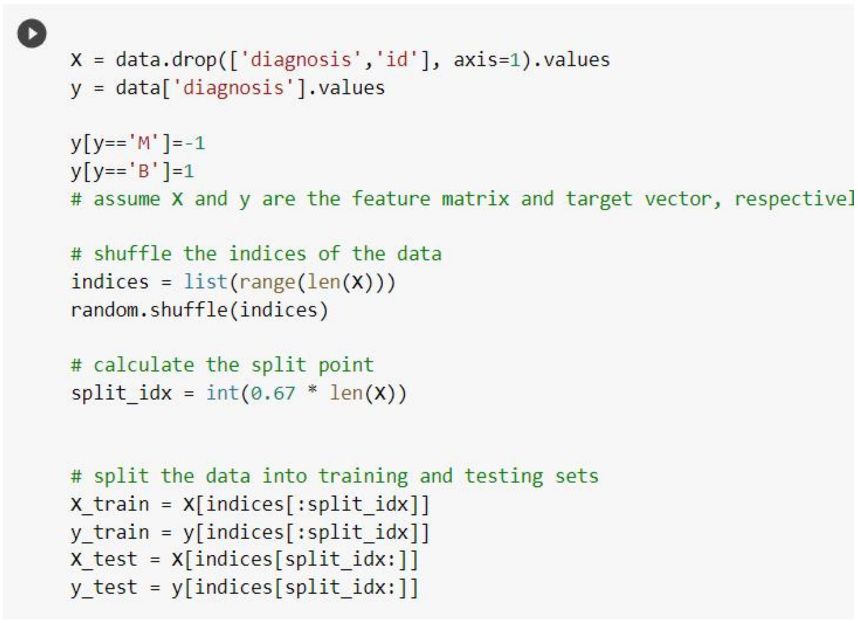
```python
import numpy as np
import pandas as pd
import random
import math
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm

%matplotlib inline
data = pd.read_csv('normalized_data.csv')
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mea
0	842302	0	1.102422	-2.071512	1.268389	0.9835
1	842517	0	1.836635	-0.353322	1.684639	1.9070
2	84300903	0	1.586206	0.455786	1.565122	1.5575
3	84348301	0	-0.767260	0.253509	-0.595257	-0.7637
4	84358402	0	1.756953	-1.150804	1.775308	1.8246

5 rows × 32 columns

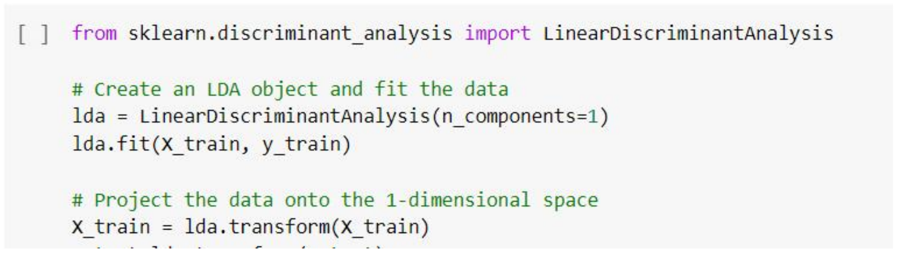
____ Here is a step by step implementation of FLDM algorithm:

First we read the data using pd.read csv function offered by the pandas library.



```python
X = data.drop(['diagnosis','id'], axis=1).values
y = data['diagnosis'].values

y[y=='M']=-1
y[y=='B']=1
# assume X and y are the feature matrix and target vector, respectivel

# shuffle the indices of the data
indices = list(range(len(X)))
random.shuffle(indices)

# calculate the split point
split_idx = int(0.67 * len(X))


# split the data into training and testing sets
X_train = X[indices[:split_idx]]
y_train = y[indices[:split_idx]]
X_test = X[indices[split_idx:]]
y_test = y[indices[split_idx:]]
```

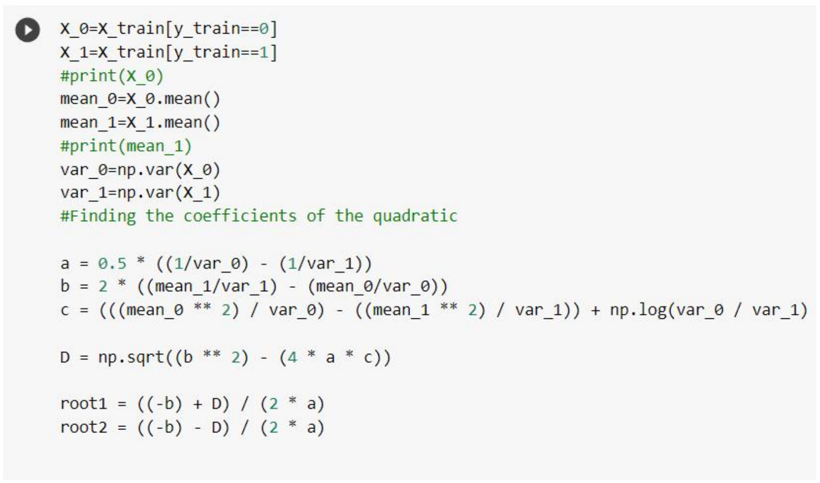
1. Shuffle the indices of the data.
2. Calculate the split point (index) for splitting the data into training and testing sets using a 67:33 ratio.
3. Split the data into training and testing sets using the calculated split point.



```
[ ]  from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

     # Create an LDA object and fit the data
     lda = LinearDiscriminantAnalysis(n_components=1)
     lda.fit(X_train, y_train)

     # Project the data onto the 1-dimensional space
     X_train = lda.transform(X_train)
```

4. Train the LDA model on the training dataset X_train and y_train.

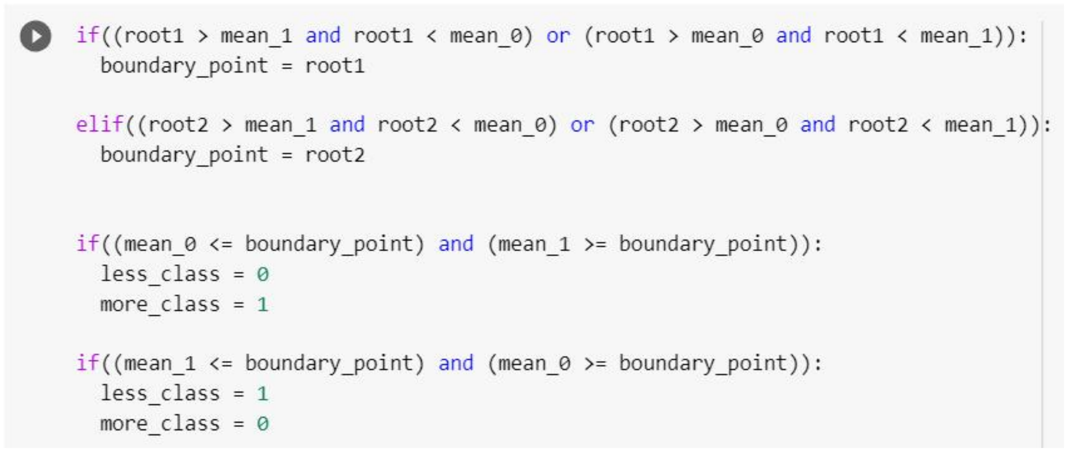
FITTING A NORMAL DISTRIBUTION



```
X_0=X_train[y_train==0]
X_1=X_train[y_train==1]
#print(X_0)
mean_0=X_0.mean()
mean_1=X_1.mean()
#print(mean_1)
var_0=np.var(X_0)
var_1=np.var(X_1)
#Finding the coefficients of the quadratic

a = 0.5 * ((1/var_0) - (1/var_1))
b = 2 * ((mean_1/var_1) - (mean_0/var_0))
c = (((mean_0 ** 2) / var_0) - ((mean_1 ** 2) / var_1)) + np.log(var_0 / var_1)

D = np.sqrt((b ** 2) - (4 * a * c))

root1 = ((-b) + D) / (2 * a)
root2 = ((-b) - D) / (2 * a)
```

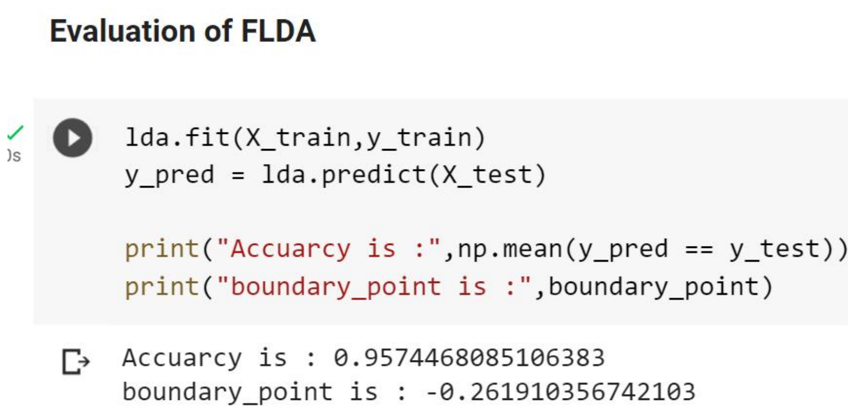
5. Separate the training dataset into two classes malignant and benign accordingly and fit them into a normal distribution and calculate the coefficients of the quadratic equation thus formed by equating the probabilities and the roots obtained from the same quadratic.

Updating The boundary_point Value



```
if((root1 > mean_1 and root1 < mean_0) or (root1 > mean_0 and root1 < mean_1)):
  boundary_point = root1

elif((root2 > mean_1 and root2 < mean_0) or (root2 > mean_0 and root2 < mean_1)):
  boundary_point = root2


if((mean_0 <= boundary_point) and (mean_1 >= boundary_point)):
  less_class = 0
  more_class = 1

if((mean_1 <= boundary_point) and (mean_0 >= boundary_point)):
  less_class = 1
  more_class = 0
```

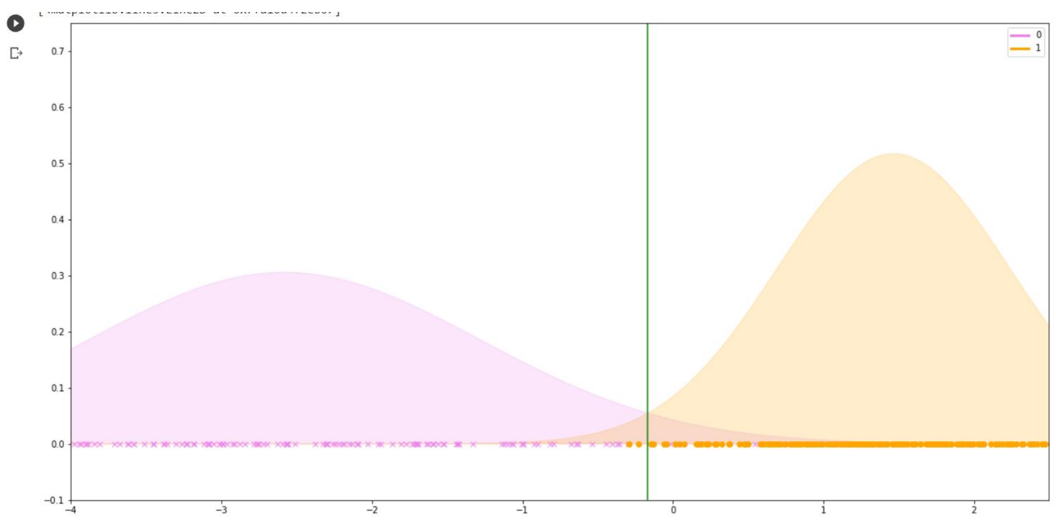
5. We find the point of intersection of the two distributions and name it as the threshold.


Evaluation of FLDA

```
lda.fit(X_train,y_train)
y_pred = lda.predict(X_test)

print("Accuarcy is :",np.mean(y_pred == y_test))
print("boundary_point is :",boundary_point)
```

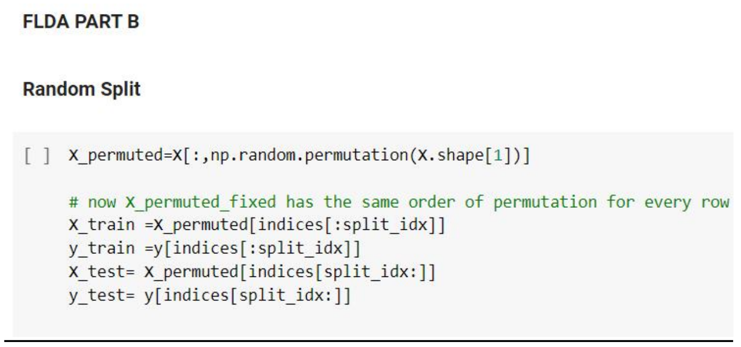
```
Accuarcy is : 0.9574468085106383
boundary_point is : -0.261910356742103
```

6. We find the evaluation parameters such as the accuracy and threshold accordingly.



7. Finally we plot the distribution of the points when projected into a 1D space.

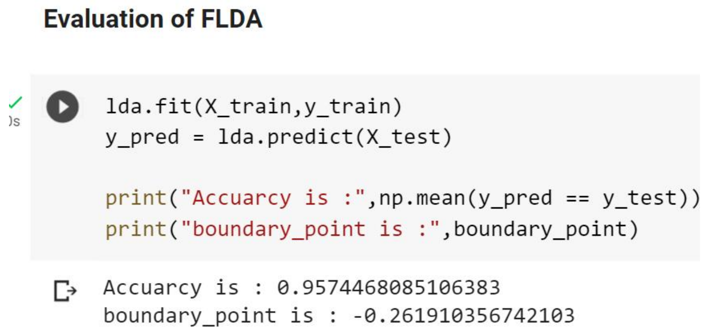
<u>TASK 2 : When we randomly shuffle the columns of the dataset</u>


FLDA PART B

Random Split

```
[ ]  X_permuted=X[:,np.random.permutation(X.shape[1])]

     # now X_permuted_fixed has the same order of permutation for every row
     X_train =X_permuted[indices[:split_idx]]
     y_train =y[indices[:split_idx]]
     X_test= X_permuted[indices[split_idx:]]
     y_test= y[indices[split_idx:]]
```

1. We randomly shuffle the columns of the dataset and implement the same code as done in part A.



Evaluation of FLDA

```
lda.fit(X_train,y_train)
y_pred = lda.predict(X_test)

print("Accuarcy is :",np.mean(y_pred == y_test))
print("boundary_point is :",boundary_point)
```

```
Accuarcy is : 0.9574468085106383
boundary_point is : -0.261910356742103
```

2. We observe that we get the same evaluation parameters as obtained in part A.

Outline the difference between the models – FLDM1 and FLDM2 - and their respective performances?

FLDM1 and FLDM2 have the same accuracy even after changing the order of feature tuple, as the dot product at each instances not affected by changing the order of the tuple.

For example, (x1, x2, x3, x4).(w1, w2, w3, w4) = (x2, x1, x3, x4).(w2, w1, w3, w4)