# PRML PA-4

**LDA | Naive Bayes Classifier**

Colab Link

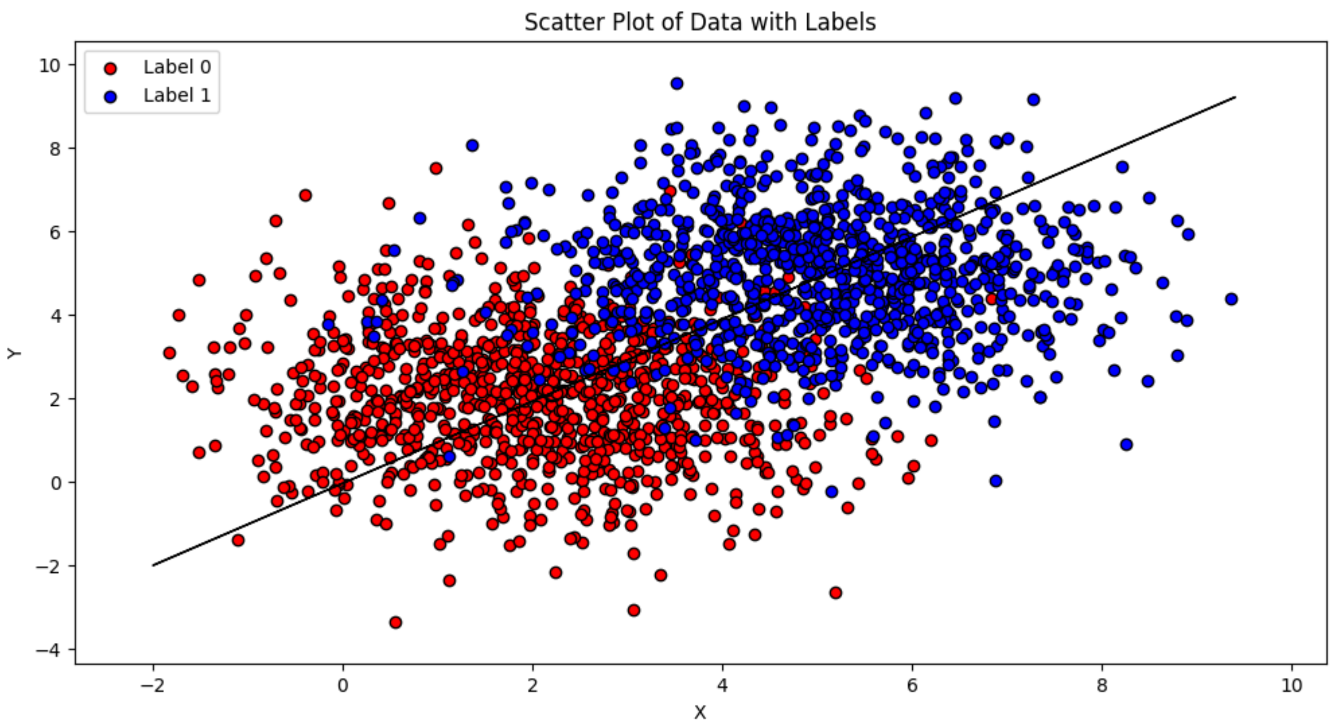
**Rhythm Patni (B22CS043)**

## Question 1 : LDA

Given a 2D data consisting of X-Coordinates, Y-coordinates and the labels of the points, The goal is to project the data onto a single dimension using Linear Discriminant Analysis.

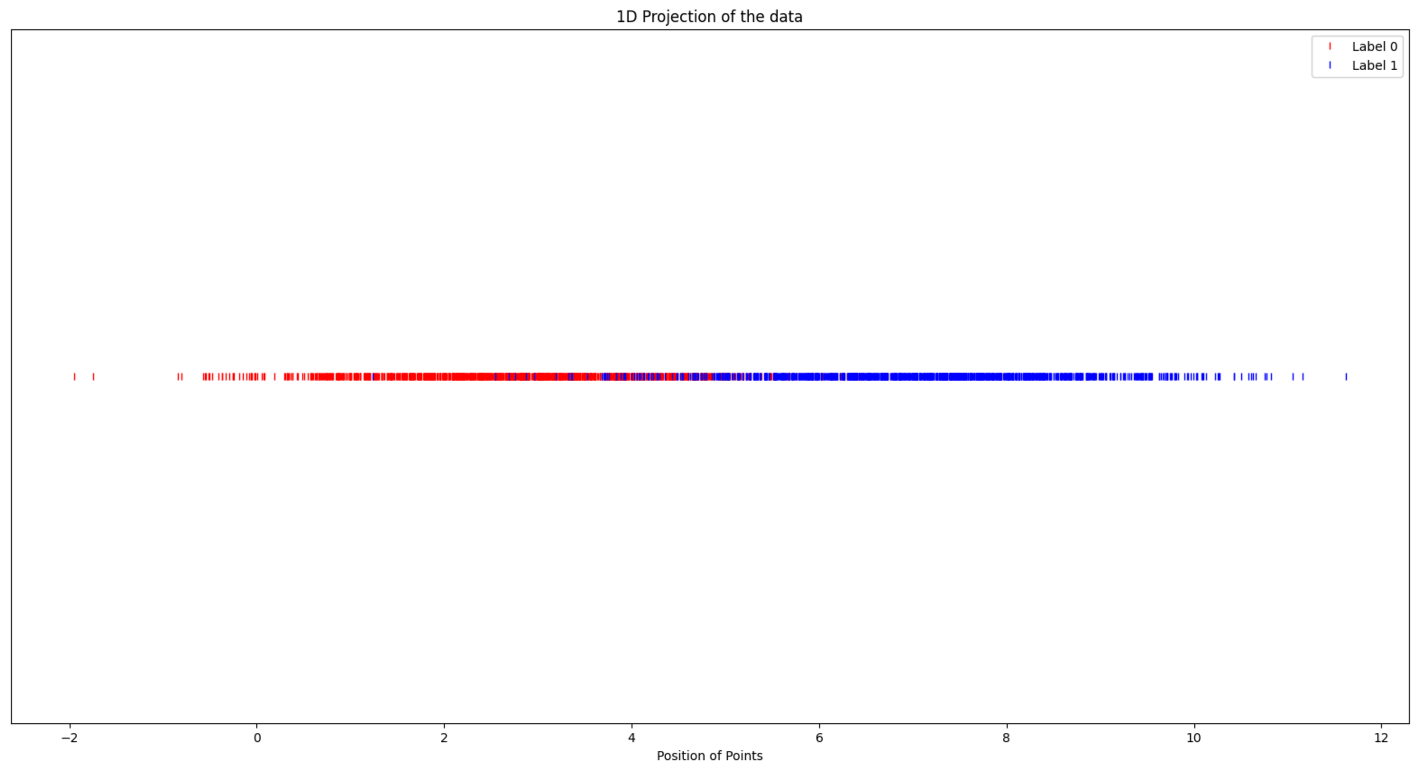
### Task 2 : Data Visualization

The data is first plotted on a 2D plane, along with the LDA projection vector. The plot is shown below.



The LDA projection vector has a slope close to unity and it almost completely divides the data into 2 similar halves.

After Getting the 1D projection of all the points, a new, transformed dataset is created whose plot is shown below.



As we can clearly see, the separation of the data is maximized in one dimension.

## Task 3 : Comparison

The performance of the 1-Nearest Neighbor model was used to compare the two datasets. Here is a table containing the results obtained. At all times, the 80-20 train - test split was used.

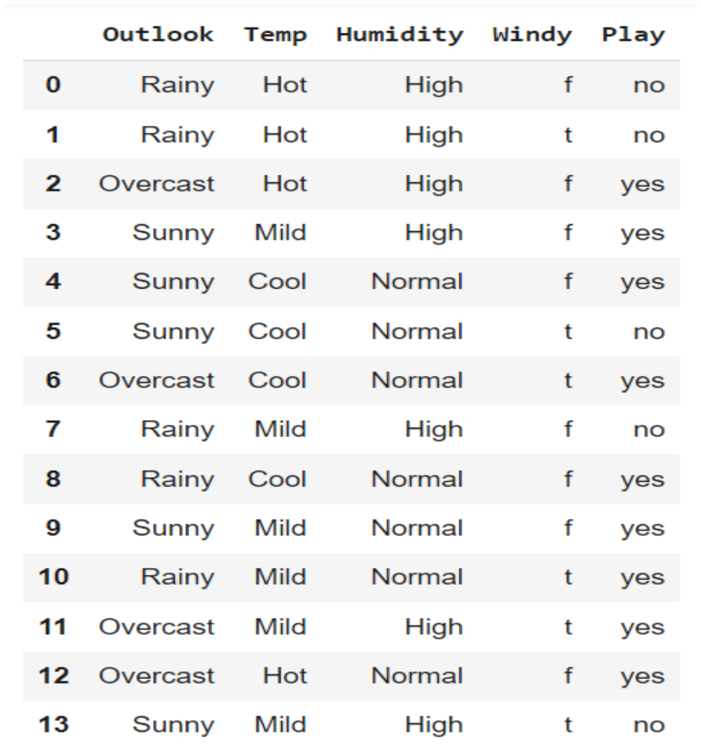
| Dataset | Accuracy | Random State of Split |
|---|---|---|
| Raw Dataset | Test Accuracy : 91.5 % | 7 |
| Transformed Dataset | Test Accuracy : 89.5% | 7 |

The accuracy of the model has decreased only slightly. Hence, in the transformation, the amount of data loss is insignificant. Here are the reasons for the same.

# Question 2 : Naive Bayes Classification

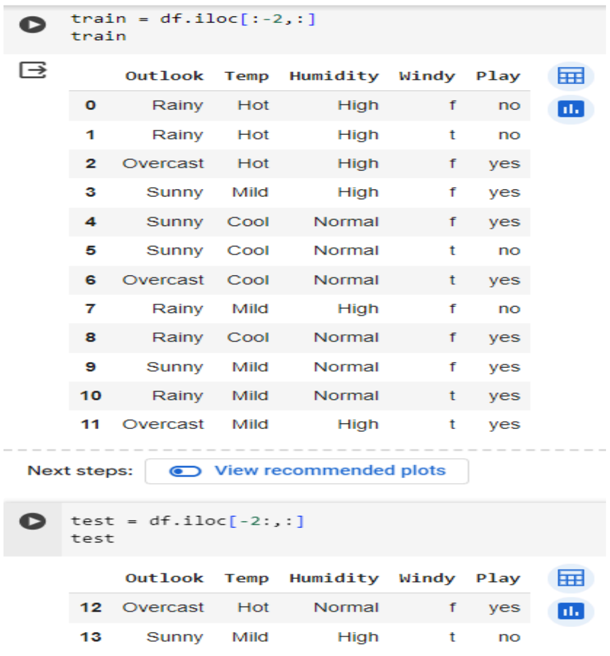
## Task 0:

Shown below is the raw dataset.



| | Outlook | Temp | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 0 | Rainy | Hot | High | f | no |
| 1 | Rainy | Hot | High | t | no |
| 2 | Overcast | Hot | High | f | yes |
| 3 | Sunny | Mild | High | f | yes |
| 4 | Sunny | Cool | Normal | f | yes |
| 5 | Sunny | Cool | Normal | t | no |
| 6 | Overcast | Cool | Normal | t | yes |
| 7 | Rainy | Mild | High | f | no |
| 8 | Rainy | Cool | Normal | f | yes |
| 9 | Sunny | Mild | Normal | f | yes |
| 10 | Rainy | Mild | Normal | t | yes |
| 11 | Overcast | Mild | High | t | yes |
| 12 | Overcast | Hot | Normal | f | yes |
| 13 | Sunny | Mild | High | t | no |

Here is the training data and the testing data.



```
train = df.iloc[:-2,:]
train
```

| | Outlook | Temp | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 0 | Rainy | Hot | High | f | no |
| 1 | Rainy | Hot | High | t | no |
| 2 | Overcast | Hot | High | f | yes |
| 3 | Sunny | Mild | High | f | yes |
| 4 | Sunny | Cool | Normal | f | yes |
| 5 | Sunny | Cool | Normal | t | no |
| 6 | Overcast | Cool | Normal | t | yes |
| 7 | Rainy | Mild | High | f | no |
| 8 | Rainy | Cool | Normal | f | yes |
| 9 | Sunny | Mild | Normal | f | yes |
| 10 | Rainy | Mild | Normal | t | yes |
| 11 | Overcast | Mild | High | t | yes |

Next steps: View recommended plots

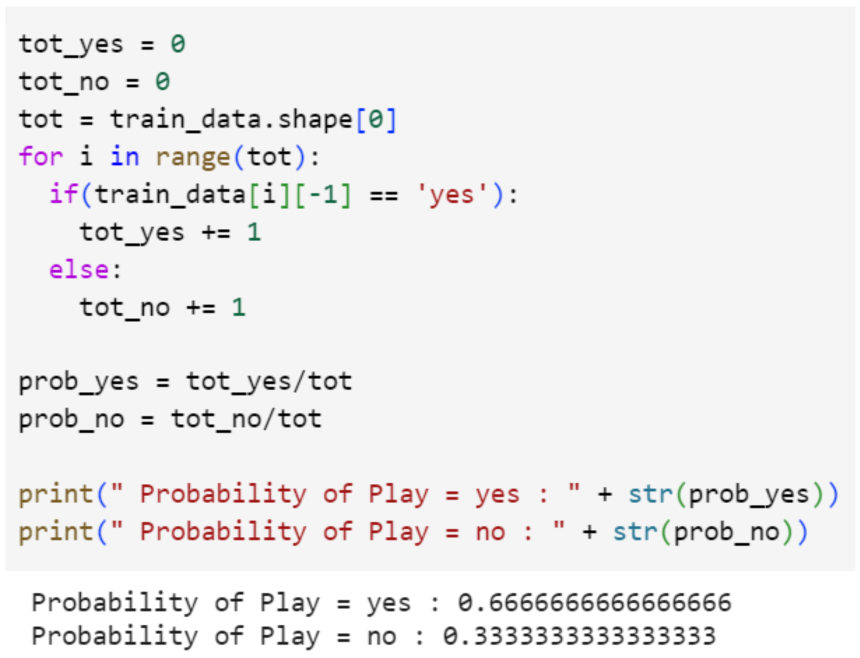
```
test = df.iloc[-2:,:]
test
```

| | Outlook | Temp | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 12 | Overcast | Hot | Normal | f | yes |
| 13 | Sunny | Mild | High | t | no |

The Split was made such that the first 12 data points are the training set and the last 2 samples are the testing set.

## Task 1 : Calculation of the Prior Probabilities

On the training dataset, the Prior Probabilities are the probabilities of playing and the probabilities of not playing.



```
tot_yes = 0
tot_no = 0
tot = train_data.shape[0]
for i in range(tot):
  if(train_data[i][-1] == 'yes'):
    tot_yes += 1
  else:
    tot_no += 1

prob_yes = tot_yes/tot
prob_no = tot_no/tot

print(" Probability of Play = yes : " + str(prob_yes))
print(" Probability of Play = no : " + str(prob_no))
```

```
 Probability of Play = yes : 0.6666666666666666
 Probability of Play = no : 0.3333333333333333
```

The prior Probabilities are as follows:

P(Play = yes) = 2/3

P(Play = no) = 1/3

## Task 2: Calculation of the Likelihood Probabilities

The likelihood probabilities are the conditional probability of observing each feature given the class label.

For example P(Outlook = Sunny|Play = yes), P(Temperature =Mild|Play = yes), and so on.

This is implemented using a dictionary of dictionaries in python. The keys of the outer dictionary are the features in the dataset (Outlook,Temperature,Humidity,Windy). Each of these keys point to another dictionary whose keys are the unique values of the features. Eg. Outlook points to a dictionary whose keys are rainy,sunny and overcast. Each of this inner dictionary points to a list containing 2 elements, The Probability of play = yes and the probability of play = no. The dictionary is being generated by a function: getLikelihoodProbabilities()

Here is an example of how to extract values from the dictionary.

P(Outlook = Overcast | Play = Yes) = 0.375

This above value is to be accessed using the dictionary as follows:

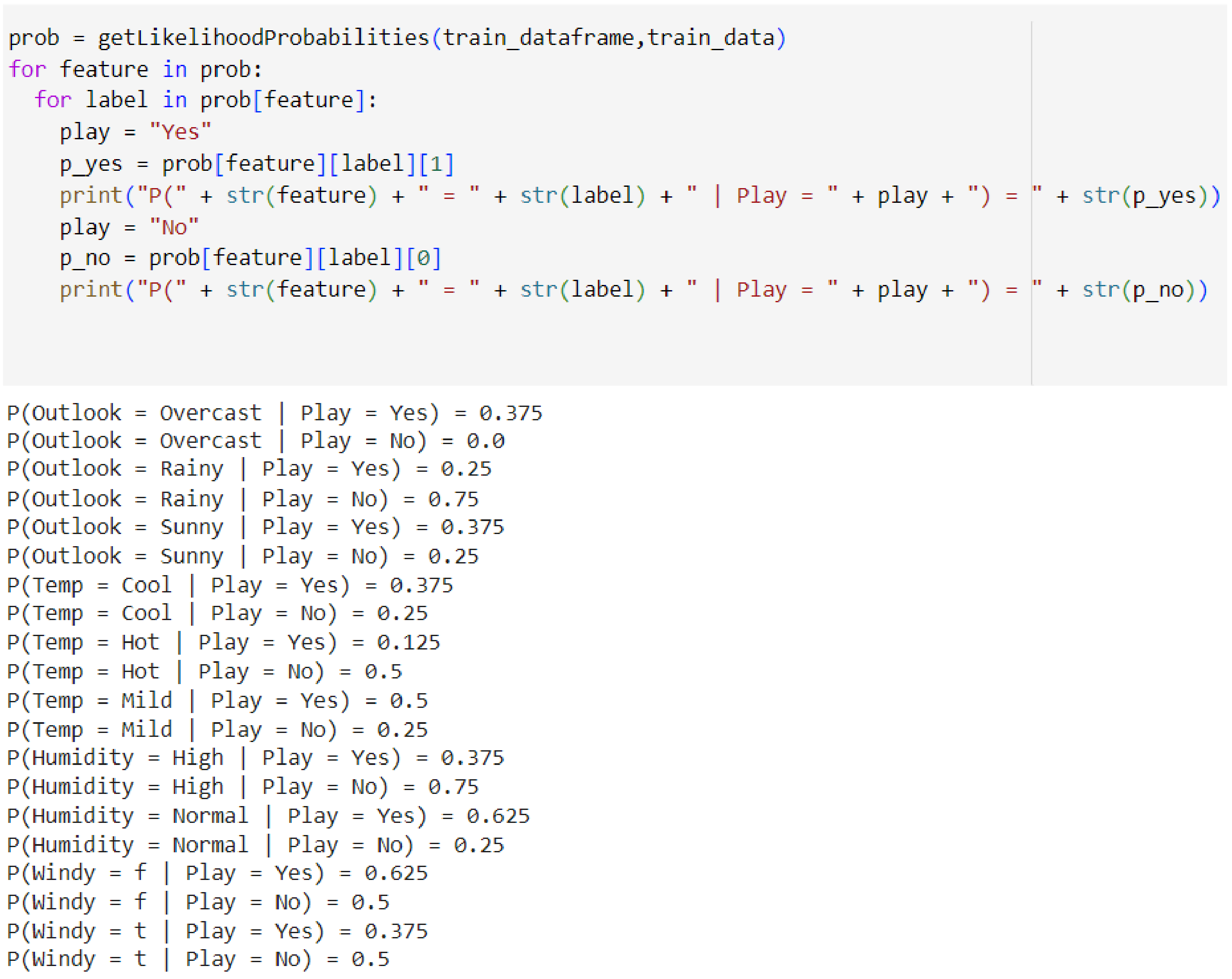
prob[‘Outlook’][‘Overcast’][1] = 0.375

P(Humidity = Normal | Play = No) = 0.25

This above value is to be accessed using the dictionary as follows:

prob[Humidity][Normal][0] = 0.25

Here is the dictionary printed.

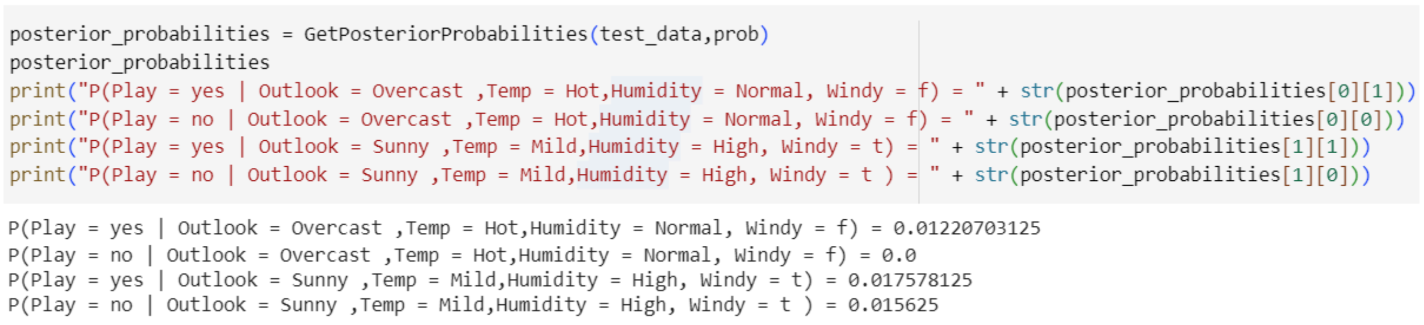


```python
prob = getLikelihoodProbabilities(train_dataframe,train_data)
for feature in prob:
  for label in prob[feature]:
    play = "Yes"
    p_yes = prob[feature][label][1]
    print("P(" + str(feature) + " = " + str(label) + " | Play = " + play + ") = " + str(p_yes))
    play = "No"
    p_no = prob[feature][label][0]
    print("P(" + str(feature) + " = " + str(label) + " | Play = " + play + ") = " + str(p_no))
```

```
P(Outlook = Overcast | Play = Yes) = 0.375
P(Outlook = Overcast | Play = No) = 0.0
P(Outlook = Rainy | Play = Yes) = 0.25
P(Outlook = Rainy | Play = No) = 0.75
P(Outlook = Sunny | Play = Yes) = 0.375
P(Outlook = Sunny | Play = No) = 0.25
P(Temp = Cool | Play = Yes) = 0.375
P(Temp = Cool | Play = No) = 0.25
P(Temp = Hot | Play = Yes) = 0.125
P(Temp = Hot | Play = No) = 0.5
P(Temp = Mild | Play = Yes) = 0.5
P(Temp = Mild | Play = No) = 0.25
P(Humidity = High | Play = Yes) = 0.375
P(Humidity = High | Play = No) = 0.75
P(Humidity = Normal | Play = Yes) = 0.625
P(Humidity = Normal | Play = No) = 0.25
P(Windy = f | Play = Yes) = 0.625
P(Windy = f | Play = No) = 0.5
P(Windy = t | Play = Yes) = 0.375
P(Windy = t | Play = No) = 0.5
```

## Task 3: Calculation of the Posterior Probabilities

For making predictions on the test set, the probabilities of both play = yes and play = no are calculated,using the Likelihood probabilities calculated before. Below is the implementation of the same.



```
posterior_probabilities = GetPosteriorProbabilities(test_data,prob)
posterior_probabilities
print("P(Play = yes | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = " + str(posterior_probabilities[0][1]))
print("P(Play = no | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = " + str(posterior_probabilities[0][0]))
print("P(Play = yes | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t) = " + str(posterior_probabilities[1][1]))
print("P(Play = no | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t ) = " + str(posterior_probabilities[1][0]))
```

```
P(Play = yes | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = 0.01220703125
P(Play = no | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = 0.0
P(Play = yes | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t) = 0.017578125
P(Play = no | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t ) = 0.015625
```

## Task 4: Making Predictions

Based on the calculated values of posterior probabilities, the following predictions are made.

- When Outlook is Overcast and Temperature is Hot and the Humidity is Normal and the Windy is f  Then Play = yes  -> Correct Prediction
- When Outlook is Sunny and Temperature is Mild and the Humidity is High and the Windy is t Then Play = yes   -> Incorrect Prediction

On the Other hand, the actual data is as follows.

- Actually when Outlook is Overcast and Temperature is Hot and the Humidity is Normal and the Windy is f Then Play = yes
- Actually when Outlook is Sunny and Temperature is Mild and the Humidity is High and the Windy is t  Then Play = no

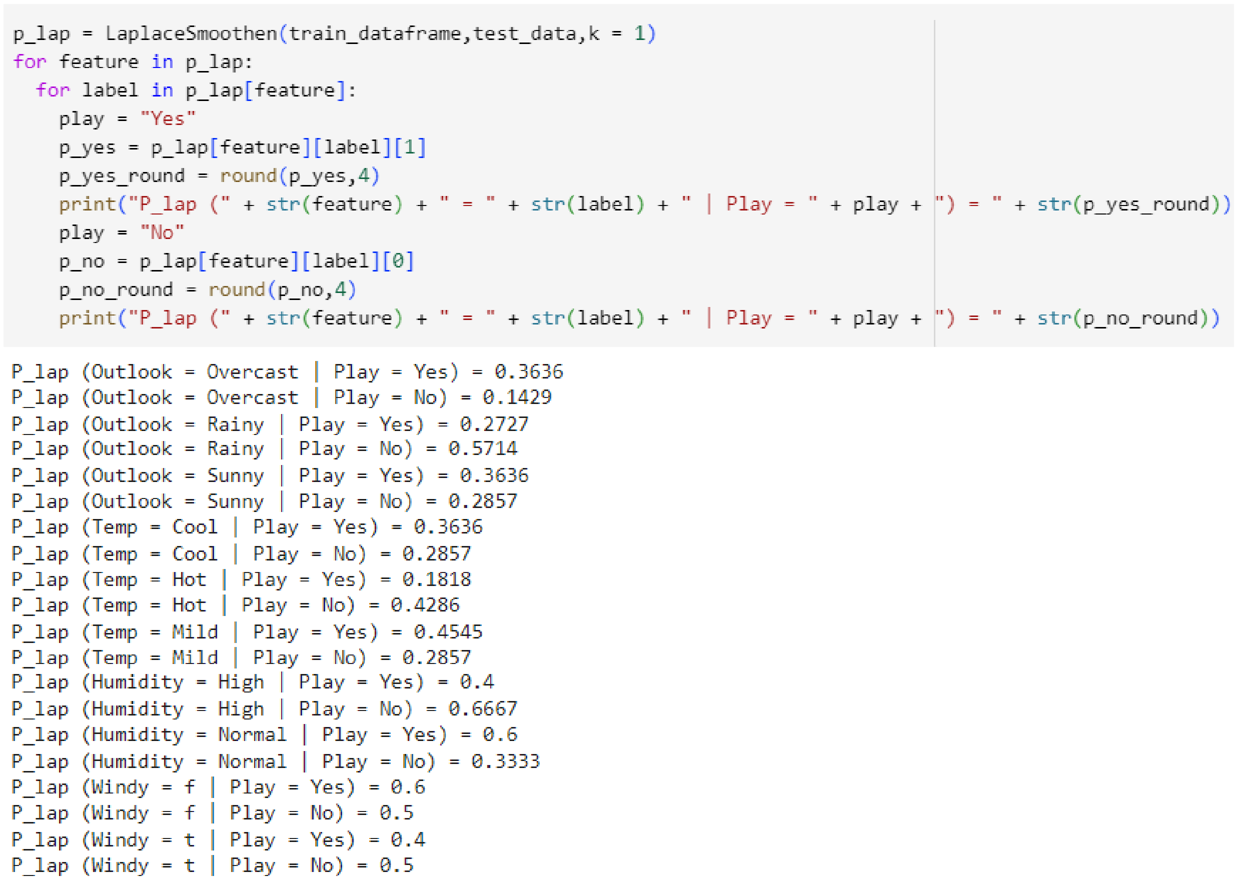
We can see that the second prediction is not correct as it corresponds to not playing category. We know that only 33% of the data corresponds to not playing class, Hence due to less availability of data on the ‘no’ class, the Naive Bayes classification fails.

## Task 5: Using Laplace Smoothing

Laplace smoothing is a method to adjust probabilities by adding a small number to each count. It helps avoid dividing by zero and ensures all outcomes have some chance of occurring, even if they haven't been seen in the training data. This is useful for making more reliable predictions, especially with limited data.

A very similar implementation is used to get the probability values using Laplace Smoothing.

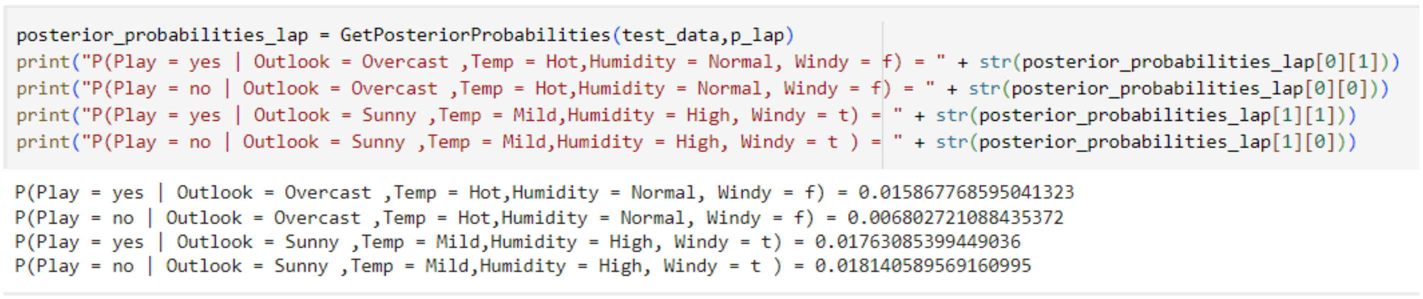
Here are the Likelihood probabilities obtained.



```
p_lap = LaplaceSmoothen(train_dataframe,test_data,k = 1)
for feature in p_lap:
  for label in p_lap[feature]:
    play = "Yes"
    p_yes = p_lap[feature][label][1]
    p_yes_round = round(p_yes,4)
    print("P_lap (" + str(feature) + " = " + str(label) + " | Play = " + play + ") = " + str(p_yes_round))
    play = "No"
    p_no = p_lap[feature][label][0]
    p_no_round = round(p_no,4)
    print("P_lap (" + str(feature) + " = " + str(label) + " | Play = " + play + ") = " + str(p_no_round))
```

```
P_lap (Outlook = Overcast | Play = Yes) = 0.3636
P_lap (Outlook = Overcast | Play = No) = 0.1429
P_lap (Outlook = Rainy | Play = Yes) = 0.2727
P_lap (Outlook = Rainy | Play = No) = 0.5714
P_lap (Outlook = Sunny | Play = Yes) = 0.3636
P_lap (Outlook = Sunny | Play = No) = 0.2857
P_lap (Temp = Cool | Play = Yes) = 0.3636
P_lap (Temp = Cool | Play = No) = 0.2857
P_lap (Temp = Hot | Play = Yes) = 0.1818
P_lap (Temp = Hot | Play = No) = 0.4286
P_lap (Temp = Mild | Play = Yes) = 0.4545
P_lap (Temp = Mild | Play = No) = 0.2857
P_lap (Humidity = High | Play = Yes) = 0.4
P_lap (Humidity = High | Play = No) = 0.6667
P_lap (Humidity = Normal | Play = Yes) = 0.6
P_lap (Humidity = Normal | Play = No) = 0.3333
P_lap (Windy = f | Play = Yes) = 0.6
P_lap (Windy = f | Play = No) = 0.5
P_lap (Windy = t | Play = Yes) = 0.4
P_lap (Windy = t | Play = No) = 0.5
```

Posterior probabilities were calculated , the values are as follows.



```
posterior_probabilities_lap = GetPosteriorProbabilities(test_data,p_lap)
print("P(Play = yes | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = " + str(posterior_probabilities_lap[0][1]))
print("P(Play = no | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = " + str(posterior_probabilities_lap[0][0]))
print("P(Play = yes | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t) = " + str(posterior_probabilities_lap[1][1]))
print("P(Play = no | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t ) = " + str(posterior_probabilities_lap[1][0]))
```

```
P(Play = yes | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = 0.015867768595041323
P(Play = no | Outlook = Overcast ,Temp = Hot,Humidity = Normal, Windy = f) = 0.006802721088435372
P(Play = yes | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t) = 0.01763085399449036
P(Play = no | Outlook = Sunny ,Temp = Mild,Humidity = High, Windy = t ) = 0.018140589569160995
```

Based on the values obtained, predictions were made again on the test data.

Here are the prediction results.

- When Outlook is Overcast and Temperature is Hot and the Humidity is Normal and the Windy is f Then Play = yes -> Correct Prediction
- When Outlook is Sunny and Temperature is Mild and the Humidity is High and the Windy is t Then Play = no -> Correct Prediction

Actual values are as follows :

- Actually when Outlook is Overcast and Temperature is Hot and the Humidity is Normal and the Windy is f Then Play = yes
- Actually when Outlook is Sunny and Temperature is Mild and the Humidity is High and the Windy is t  Then Play = no

As we can see, the predictions of the Naive Bayes Classifier improved significantly after Laplace Smoothing. Here are the reasons for the same.

- **Preventing Zero Probabilities:** The probabilities of features in the training data are computed by Naive Bayes classifiers. However, using Bayes' theorem directly will result in a zero probability if a feature hasn't been detected with a specific class label in the training set. Laplace Smoothing prevents zero probabilities by adding a little count to every feature occurrence, ensuring that each feature has a non-zero probability.
- **Minimizing Overfitting:** In the absence of Laplace Smoothing, the model may overfit the training set, particularly in cases involving short datasets or uncommon characteristics. Laplace Smoothing helps to regularize the model and lowers the chance of overfitting by adding a modest count to each feature, resulting in more broadly applicable predictions on unobserved data.
- **Better Generalization**: Laplace Smoothing yields more reliable probability estimates, especially for characteristics that are uncommon or not included in the training set. As a result, the Naive Bayes classifier may more effectively generalize to new instances, improving prediction accuracy on unobserved data.
- **Improved Handling of Sparse Data**: Laplace Smoothing is very helpful when working with sparse datasets, which may have extremely few or no instances of a given feature-label pair in the training set. Even with sparse data, Laplace Smoothing helps to smooth out the probability estimates and produces more stable forecasts by adding a little count to every feature occurrence.

Overall, by guaranteeing that all features have non-zero probabilities and by offering more reliable estimates, Laplace Smoothing helps to overcome the shortcomings of the Naive Bayes classifier and produces noticeably better prediction performance.