1. **Write a Python program to load iris data set and apply Naïve-Bayes algorithm for classification of Iris flowers.**

Overview of Naive Bayes Classification:

Naive Bayes is one such algorithm in classification that can never be overlooked upon due to its special characteristic of being "naive". It makes the assumption that features of a measurement are independent of each other.

For example, an animal may be considered as a cat if it has cat eyes, whiskers and a long tail. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this animal is a cat and that is why it is known as 'Naive'.

According to Bayes Theorem, the various features are mutually independent. For two independent events, P(A,B) = P(A)P(B). This assumption of Bayes Theorem is probably never encountered in practice, hence it accounts for the "naive" part in Naive Bayes. Bayes' Theorem is stated as: P(a|b) = (P(b|a) * P(a)) / P(b). Where P(a|b) is the probability of a given b.

Let us understand this algorithm with a simple example. The Student will be a pass if he wears a "red" color dress on the exam day. We can solve it using above discussed method of posterior probability.

By Bayes Theorem, P(Pass| Red) = P(Red| Pass) * P(Pass) / P (Red).

From the values, let us assume P (Red|Pass) = 3/9 = 0.33, P(Red) = 5/14 = 0.36, P(Pass)= 9/14 = 0.64. Now, P (Pass| Red) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

In this way, Naive Bayes uses a similar method to predict the probability of different class based on various attributes.

Problem Analysis:

To implement the Naive Bayes Classification, we shall use a very famous Iris Flower Dataset that consists of 3 classes of flowers. In this, there are 4 independent variables namely the, sepal_length, sepal_width, petal_length and petal_width. The dependent variable is the species which we will predict using the four independent features of the flowers.

There are 3 classes of species namely setosa, versicolor and the virginica. This dataset was originally introduced in 1936 by Ronald Fisher. Using the various features of the flower (independent variables), we have to classify a given flower using Naive Bayes Classification model.

Step 1: Importing the Libraries

As always, the first step will always include importing the libraries which are the NumPy, Pandas and the Matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Step 2: Importing the dataset

In this step, we shall import the Iris Flower dataset which is stored in my github repository as IrisDataset.csv and save it to the variable dataset. After this, we assign the 4 independent variables to X and the dependent variable 'species' to Y. The first 5 rows of the dataset are displayed.

```
dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Classification/master/IrisDataset.csv')



X = dataset.iloc[:,:4].values
y = dataset['species'].values



dataset.head(5)



>>
sepal_length  sepal_width  petal_length  petal_width   species
5.1           3.5          1.4           0.2           setosa
4.9           3.0          1.4           0.2           setosa
4.7           3.2          1.3           0.2           setosa
4.6           3.1          1.5           0.2           setosa
5.0           3.6          1.4           0.2           setosa
```

Step 3: Splitting the dataset into the Training set and Test set

Once we have obtained our data set, we have to split the data into the training set and the test set. In this data set, there are 150 rows with 50 rows of each of the 3 classes. As each class is given in a continuous order, we need to randomly split the dataset. Here, we have the test_size=0.2, which means that 20% of the dataset will be used for testing purpose as the test set and the remaining 80% will be used as the training set for training the Naive Bayes classification model.

```
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)
```

Step 4: Feature Scaling

The dataset is scaled down to a smaller range using the Feature Scaling option. In this, both the X_train and X_test values are scaled down to smaller values to improve the speed of the program.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Step 5: Training the Naive Bayes Classification model on the Training Set

In this step, we introduce the class GaussianNB that is used from the sklearn.naive_bayes library. Here, we have used a Gaussian model, there are several other models such as Bernoulli, Categorical and Multinomial. Here, we assign the GaussianNB class to the variable classifier and fit the X_train and y_train values to it for training purpose.

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Step 6: Predicting the Test set results

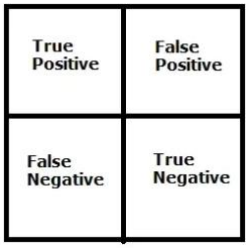
Once the model is trained, we use the the classifier.predict() to predict the values for the Test set and the values predicted are stored to the variable y_pred.

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

Step 7: Confusion Matrix and Accuracy

This is a step that is mostly used in classification techniques. In this, we see the Accuracy of the trained model and plot the confusion matrix.

The confusion matrix is a table that is used to show the number of correct and incorrect predictions on a classification problem when the real values of the Test Set are known. It is of the format.



The True values are the number of correct predictions made.

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

from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm

>>Accuracy :  0.9666666666666667

>>array([[14,  0,  0],
       [ 0,  7,  0],
       [ 0,  1,  8]])
```

From the above confusion matrix, we infer that, out of 30 test set data, 29 were correctly classified and only 1 was incorrectly classified. This gives us a high accuracy of 96.67%.

Step 8: Comparing the Real Values with Predicted Values

In this step, a Pandas DataFrame is created to compare the classified values of both the original Test set (y_test) and the predicted results (y_pred).

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})

df

>>

Real Values   Predicted Values
setosa        setosa
setosa        setosa
virginica     virginica
versicolor    versicolor
setosa        setosa
setosa        setosa
...  ...   ...  ...  ...
virginica     versicolor
virginica     virginica
setosa        setosa
```

```
setosa        setosa
versicolor    versicolor
versicolor    versicolor
```

This step is an additional step which is not much informative as the Confusion matrix and is mainly used in regression to check the accuracy of the predicted value.

As you can see, there is one incorrect prediction that has predicted versicolor instead of virginica.