In [1]:

```
import pandas as pd
import numpy as np
from sklearn import datasets
from collections import Counter
from sklearn.metrics import accuracy_score
```

In [2]:

```
iris = datasets.load_iris()
Species = iris.target
data = pd.DataFrame(np.c_[iris.data, Species.reshape((Species.shape[0],1))], columns = iris
data.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

In [3]:

```
X = data.drop(['Species'], axis = 1)
Y = data['Species']
```

Splitting to Train and Test set

In [4]:

```
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=10)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(120, 4) (30, 4) (120,) (30,)
```

Naive-Bayes Function

In [5]:

```
class NB():
    def __init__(self, X_train, Y_train):
        self.train = pd.DataFrame(np.hstack([X_train, np.array(Y_train).reshape(-1,1)]), co
        self.X_train = X_train
        self.Y_train = Y_train
        self.s = {}


    def fit(self):

        self.result = Counter(self.Y_train) #makes a dictionary of all possible targets

        for target in self.result.keys():
            for col in self.X_train.columns:                  #calls the add_to_dict functi
                self.s[target,col,"mean"] = self.train[self.train['Species'] == target].mea
                self.s[target,col,"std"] = self.train[self.train['Species'] == target].std(

        for i in self.result:   #changes the values from count of to probability
            self.result[i] = round(self.result[i]/len(self.X_train.index),8)

    def predict(self,X_test):
        count = 0
        prediction = []
        for i in X_test.index:                        #enters into a row-wise loop
            prob_index = {}
            for target in self.result:                  #enters into a loop for every value
                prob = self.result[target]
                for col in self.X_train:     #enters into a loop where it multiplies the co
                    a = 1/(((2*np.pi)**0.5)*self.s[target,col,"std"])
                    b = -((X_test[col][i] - self.s[target,col,"mean"])**2)
                    c = 2*(self.s[target,col,"std"]**2)
                    prob = prob * a * np.exp(b/c)
                prob_index[target] = prob           #adds value of P(condition/target) to a

            probability = 0
            for target in prob_index:             #this loop looks for the outcome for high
                if prob_index[target] > probability:
                    pred = target
                    probability = prob_index[target]
            prediction.append(pred)              #will add the prediction to a list

        return prediction
```

Training and Predicting

In [6]:

```
naive = NB(X_train, Y_train)
naive.fit()
```

In [7]:

```
Y_pred = naive.predict(X_test)
```

Sklearn

In [8]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

In [9]:

```
gnb.fit(X_train, Y_train)
```

Out[9]:

```
GaussianNB()
```

In [10]:

```
Y_pred_sk = gnb.predict(X_test)
```

Comparison

In [11]:

```
print(f'Accuracy using self-made function : {accuracy_score(Y_test, Y_pred)}')
print(f'Accuracy using sklearn : {accuracy_score(Y_test, Y_pred_sk)}')
```

```
Accuracy using self-made function : 1.0
Accuracy using sklearn : 1.0
```