

Importing Modules

```
In [1]: import pandas as pd
import numpy as np
from sklearn import datasets
from collections import Counter
```

Loading Data

```
In [2]: iris = datasets.load_iris()
species = iris.target
data = pd.DataFrame(np.c_[iris.data, species.reshape((Species.shape[0],1))], columns=
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]: data['Species'].value_counts()
```

```
Out[3]: 2.0    50
1.0    50
0.0    50
Name: Species, dtype: int64
```

Splitting into train and test

```
In [4]: from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size = 0.2, random_state = 123)
```

Making Naive Bayes function

```
In [5]: class NB():
    def __init__(self,train):
        self.train = train
        self.X_train = train.drop('Species', axis = 1)
        self.Y_train = train['Species']
        self.s = {}

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

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

        for i in self.result:
            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:
    prob_index = {}
    #enters into a loop for every target value
    for target in self.result:
        prob = self.result[target]
        #loop where conditional probability for each column value is multipli
        for col in self.X_train:
            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)
        #adds value of P(condition/target) to a list
        prob_index[target] = prob

    probability = 0
    #Looks for the outcome for highest probability for particular row
    for target in prob_index:
        if prob_index[target] > probability:
            pred = target
            probability = prob_index[target]
    #adds prediction to a list
    prediction.append(pred)

return prediction

```

```

In [6]: clf = NB(train)
        clf.fit()

```

Predictions

```

In [7]: Y_test = test['Species']
        X_test = test.drop('Species', axis = 1)
        predictions = clf.predict(X_test)

```

Accuracy

```

In [8]: from sklearn.metrics import accuracy_score
        accuracy_score(Y_test, predictions)

```

```

Out[8]: 0.9666666666666667

```

```

In [9]: from sklearn.naive_bayes import GaussianNB
        gnb = GaussianNB()
        mod = gnb.fit(data.iloc[:,4], data.iloc[:,4])
        predictions1 = clf.predict(data.iloc[:,4])
        accuracy_score(data.iloc[:,4], predictions1)

```

```

Out[9]: 0.9666666666666667

```