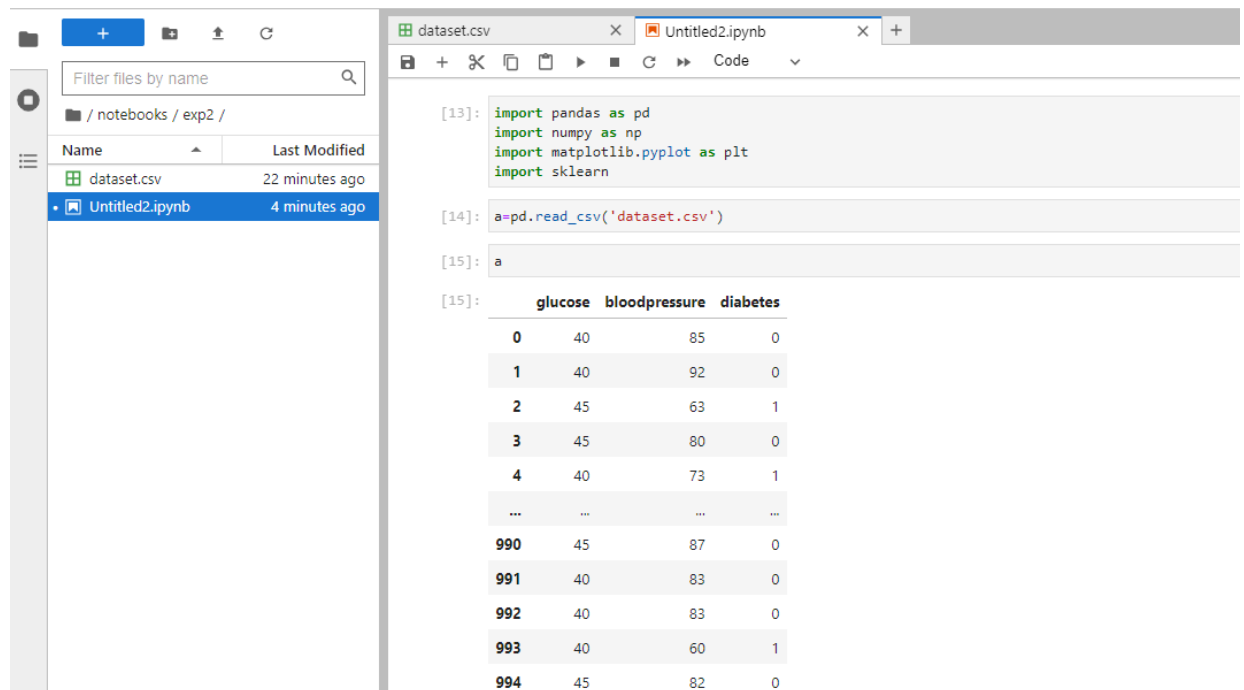


**Batch: 2****Experiment Number: 2****Roll Number: 16010421073****Name: Keyur Patel****Aim of the Experiment:** Implementation of Naïve Bayesian algorithm for classification

---

**Code and output:**

The screenshot shows a Jupyter Notebook environment with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with 'dataset.csv' and 'Untitled2.ipynb'. The code editor shows the following code:

```
[13]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

[14]: a=pd.read_csv('dataset.csv')

[15]: a
```

The output of the code is a table with 4 columns: 'glucose', 'bloodpressure', and 'diabetes'. The table contains 10 rows of data, with the first row being the header and the subsequent rows being data points. The 'diabetes' column has values 0 or 1, indicating the presence or absence of diabetes.

	glucose	bloodpressure	diabetes
0	40	85	0
1	40	92	0
2	45	63	1
3	45	80	0
4	40	73	1
...	...	...	...
990	45	87	0
991	40	83	0
992	40	83	0
993	40	60	1
994	45	82	0

```
[16]: x=a.iloc[:,[1,2]].values
      y=a.iloc[:, -1].values
```

```
[17]: x
```

```
[17]: array([[85,  0],
            [92,  0],
            [63,  1],
            ...,
            [83,  0],
            [60,  1],
            [82,  0]], dtype=int64)
```

```
[18]: y
```

```
[18]: array([0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0,
            1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0,
            0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
            1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0,
            0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
            1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1,
            0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1,
            1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
            1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
            1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
            1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
            1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
            1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
            1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
            0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
            0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1
```

```
[19]: from sklearn.model_selection import train_test_split
```

```
[20]: X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 0)
```

```
[21]: from sklearn.preprocessing import StandardScaler
      sc = StandardScaler()
      X_train = sc.fit_transform(X_train)
      X_test = sc.transform(X_test)
```

```
[22]: X_train
```

```
[22]: array([[ -0.68680808,  1.02803091],
            [-2.07708392,  1.02803091],
            [ 0.81041204, -0.97273334 ],
            ...,
            [ 0.38263486, -0.97273334 ],
            [ 1.13124493, -0.97273334 ],
            [-0.47291949,  1.02803091]])
```

```
[23]: X_test
```

```
[23]: array([[ -0.68680808,  1.02803091],
 [ -0.68680808,  1.02803091],
 [ -1.22152956,  1.02803091],
 [  0.27569057,  1.02803091],
 [  0.59652345, -0.9727334 ],
 [  1.6659664 , -0.9727334 ],
 [ -1.00764097,  1.02803091],
 [ -1.32847385,  1.02803091],
 [  0.27569057, -0.9727334 ],
 [ -1.22152956,  1.02803091],
 [  1.6659664 , -0.9727334 ],
 [  0.81041204, -0.9727334 ],
 [ -1.00764097,  1.02803091],
 [ -1.75625103,  1.02803091],
 [ -1.32847385,  1.02803091],
 [ -0.2590309 ,  1.02803091],
 [ -0.47291949,  1.02803091],
 [  1.34513352, -0.9727334 ],
 [ -1.00764097,  1.02803091],
 [  0.06180198,  1.02803091],
 [  0.91735634, -0.9727334 ],
 [  0.91735634, -0.9727334 ],
 [  0.06180198,  1.02803091],
 [  0.38263486, -0.9727334 ],
 [ -1.32847385,  1.02803091],
 [ -1.32847385,  1.02803091],
 [ -0.68680808, -0.9727334 ],
```

```
[24]: from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
[24]: GaussianNB
GaussianNB()
```

```
[25]: y_pred = classifier.predict(X_test)
```

```
[26]: y_pred
```

```
[26]: array([1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1,
 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
 1], dtype=int64)
```

```
[27]: from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
```

```
[28]: cm
```

```
[28]: array([[ 88,   0],
 [   0, 111]], dtype=int64)
```

```
[29]: ac
```

```
[29]: 1.0
```

### Applying Naïve Bayesian algorithm on unknown tuple

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=100, shuffle=True)
print(model.predict([[0,1]]))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=50, shuffle=True)
print(model.predict([[0,1]]))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=100, shuffle=True)
print(model.predict([[45,100]]))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=50, shuffle=True)
print(model.predict([[45,100]]))
```

```
[37] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100, shuffle=True)
print(model.predict([[0,1]]))

[0]
```

```
[39] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=50, shuffle=True)
print(model.predict([[0,1]]))

[0]
```

```
[42] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100, shuffle=True)
print(model.predict([[45,100]]))

[1]
```

```
[41] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=50, shuffle=True)
print(model.predict([[45,100]]))

[1]
```

---

### Post Lab Question-Answers:

#### 1. What are advantages and disadvantages of Bayesian Classification?

##### Ans: Advantages

- This algorithm works quickly and can save a lot of time.
- Naive Bayes is suitable for solving multi-class prediction problems.
- If its assumption of the independence of features holds true, it can perform better than other models and requires much less training data.
- Naive Bayes is better suited for categorical input variables than numerical variables.

**Disadvantages**

- Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.
- This algorithm faces the 'zero-frequency problem' where it assigns zero probability to a categorical variable whose category in the test data set wasn't available in the training dataset. It would be best if you used a smoothing technique to overcome this issue.
- Its estimations can be wrong in some cases, so you shouldn't take its probability outputs very seriously.

**2. Comment on Laplacian correction.**

**Ans:** Laplacian correction, also known as Laplace smoothing or add-k smoothing, is a technique used to handle the issue of zero probabilities in Bayesian classifiers. It addresses the problem that if a feature value is not present in the training data for a certain class, the probability estimate for that class will be zero, causing problems during classification.

Laplacian correction involves adding a small constant (often denoted as "k") to both the numerator and the denominator of the probability estimation formula. This ensures that even if a feature hasn't been observed with a certain class in the training data, there is still a non-zero probability assigned to it.

Mathematically, the formula for Laplacian-corrected probability estimation becomes:

$$P(\text{feature} = x \mid \text{class}) = (\text{count}(\text{feature} = x \text{ in class}) + k) / (\text{count}(\text{class}) + k * \text{number\_of\_possible\_values\_for\_feature})$$

The value of "k" is a hyperparameter that needs to be chosen carefully. A smaller "k" value puts more weight on the data, while a larger "k" value places more weight on the prior probability.

Laplacian correction can help to prevent overfitting and improve the generalization of the model. However, while Laplacian correction can be useful in some cases, it's not a one-size-fits-all solution. The choice of "k" and whether to apply Laplacian correction at all depends on the specific dataset and problem at hand.

**Outcomes:**

CO1: Comprehend basics of machine learning

**Conclusion (based on the Results and outcomes achieved):**

Through this experiment we learnt how to divide data into training and testing and implemented Naïve Bayesian algorithm for classification

**References:**

Books/ Journals/ Websites:

1. Han, Kamber, "Data Mining Concepts and Techniques", Morgan Kaufmann  
3 rd Edition