

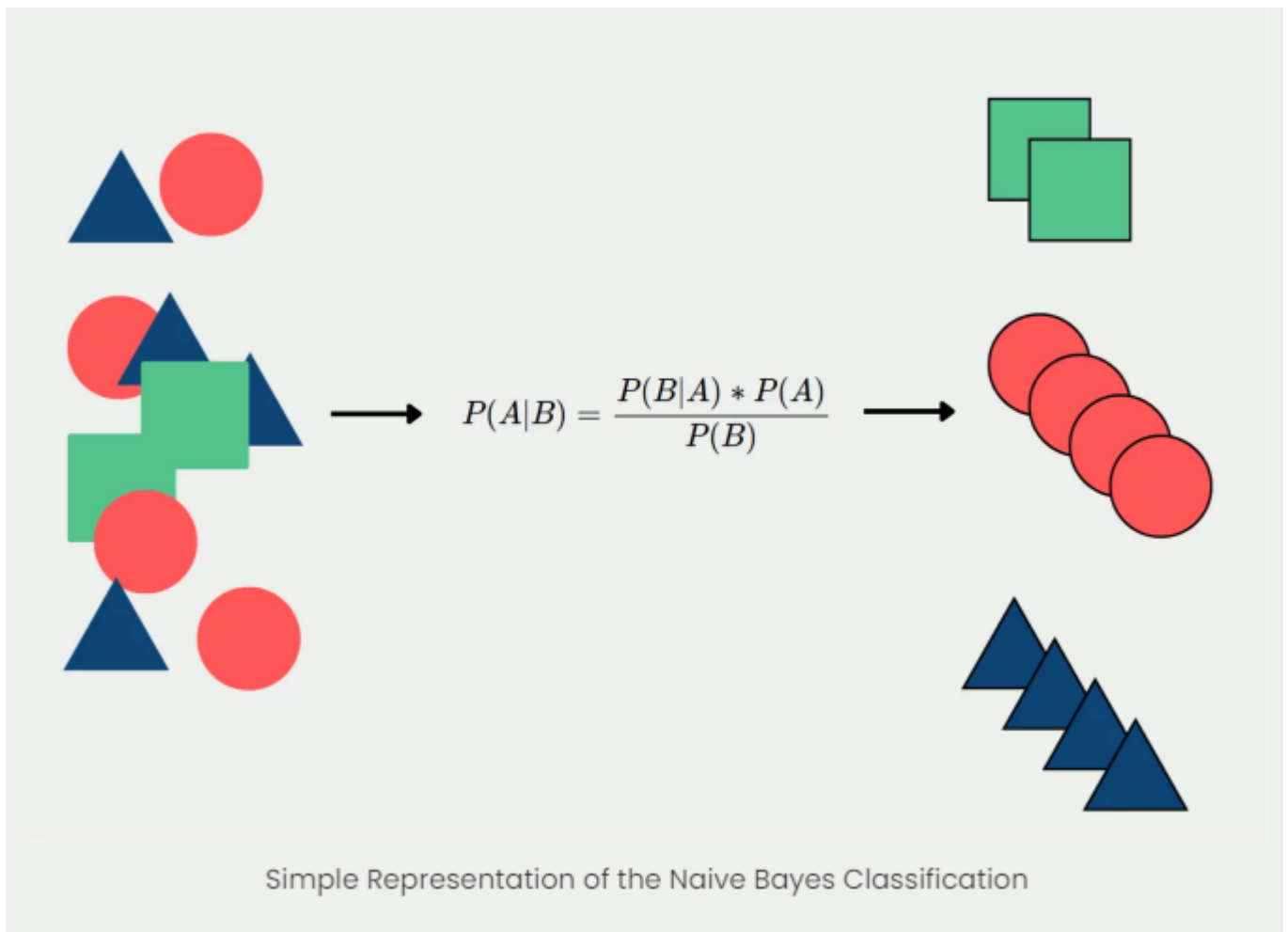
## ✓ Naive Bayes

### Agenda

- Introduction to Naive Bayes
- Naive Bayes Algorithm Intuition
  - Bayes' Theorem
  - Assumption of Feature Independence
- Example of Naive Bayes Algorithm
- Types of Naive Bayes Algorithms
  - Gaussian Naive Bayes
  - Multinomial Naive Bayes
  - Bernoulli Naive
- Use Case
  - Implementation of Naive Bayes
- Advantages of Naive Bayes
- Disadvantages of Naive Bayes

## ✓ Introduction to Naive Bayes

- Naive Bayes is a probabilistic machine learning algorithm based on Bayes' Theorem. It is primarily used for classification tasks and assumes that the features in a dataset are conditionally independent given the class label. Despite this strong and often unrealistic assumption, the algorithm performs well in many real-world scenarios, particularly in text classification, spam detection, and sentiment analysis.



## ✓ Bayes' Theorem Application

- Bayes' theorem is a fundamental tool in probability theory with a wide range of applications in various fields. Here are some real-world examples:
  - Medical Diagnosis
    - Disease Screening: Given a positive test result, what is the probability of actually having the disease? Bayes' theorem helps calculate the positive predictive value (PPV) of a test based on sensitivity, specificity, and prevalence of the disease.
    - Differential Diagnosis: When a patient presents with multiple possible diagnoses, Bayes' theorem can be used to update the probabilities of each diagnosis based on new symptoms or test results.
  - Spam Filtering
    - Email Classification: Bayes' theorem is used to classify emails as spam or ham based on the presence or absence of certain keywords or phrases. By calculating the probability of an email being spam or ham given its content, spam filters can effectively identify and filter unwanted messages.

- Natural Language Processing
  - Language Modeling: Bayes' theorem is used in language models to calculate the probability of a sequence of words occurring together. This is essential for tasks like machine translation, speech recognition, and text generation.
  - Sentiment Analysis: Bayes' theorem can be used to classify text as positive, negative, or neutral based on the presence of certain words or phrases.
- Recommendation Systems
  - Collaborative Filtering: Bayes' theorem can be used to calculate the probability that a user will like a particular item based on their past preferences and the preferences of similar users. This is a common approach in recommendation systems used by online retailers, streaming services, and social media platforms.

### Naive Bayes Variants:

- Naive Bayes models are often referred to as simple Bayes or independent Bayes, highlighting the core principles of the algorithm.
- These names underscore the application of Bayes' theorem in the classifier's decision-making process.
- Despite its simplicity, the Naive Bayes classifier harnesses the predictive power of Bayes' theorem effectively in machine learning applications.

### Strengths:

- Simplicity: Naive Bayes is a simple and easy-to-understand algorithm, making it accessible to practitioners with limited machine learning experience.
- Efficiency: It is computationally efficient, making it suitable for large datasets and real-time applications.
- Performance: Despite its simplicity, Naive Bayes often performs surprisingly well, especially in text classification tasks.
- Robustness: It can handle categorical and numerical features, making it versatile for various data types.
- Interpretability: The probabilities calculated by Naive Bayes can provide insights into the decision-making process, making it easier to understand the model's predictions.

## ✓ Naive Bayes Algorithm Intuition

### Predicting Class Memberships:

- The Naive Bayes Classifier employs Bayes' theorem to estimate the probabilities of membership for each class, determining the likelihood that a given data point belongs to a

particular class.

- It calculates the probability of each class and selects the class with the highest probability as the most probable class for the data point.
- This process is also known as Maximum A Posteriori (MAP) estimation.

### Understanding MAP:

For a hypothesis with events A and B, the MAP estimation for event A is computed as:

$$MAP(A) = \max(P(A|B)) = \max\left(\frac{P(B|A) \times P(A)}{P(B)}\right) = \max(P(B|A) \times P(A))$$

Here,  $P(B)$  represents the evidence probability, which is used to normalize the result. Since it remains constant, removing it does not affect the final result.

- At the heart of Naive Bayes lies Bayes' Theorem, which is a fundamental concept in probability theory.
- It describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

Mathematically, Bayes' Theorem can be expressed as:

### ✓ Bayes' Theorem

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Probability A Will Happen Given Evidence B Has Already Happened

Probability B Will Happen Given Evidence A Has Already Happened

Probability A Will Happen

Probability B Will Happen

Where:

- $P(A|B)$  is the probability of event A occurring given that event B has occurred.
- $P(B|A)$  is the probability of event B occurring given that event A has occurred.

- $P(A)$  and  $P(B)$  are the probabilities of events A and B occurring independently.

## ✓ Assumption of Feature Independence

- The Assumption of Feature Independence states that, given a class label, the presence (or absence) of one feature is independent of the presence (or absence) of any other feature.

Mathematically, this can be expressed as:

$$P(X_1, X_2, \dots, X_n | Y) = P(X_1 | Y) \cdot P(X_2 | Y) \cdot \dots \cdot P(X_n | Y)$$

- Where:
  - $X_1, X_2, \dots, X_n$  are the features.
  - $Y$  is the class label.
- Implications
  - Simplified Calculations:
    - The independence assumption allows for the simplification of the computation of joint probabilities, which can be extremely high-dimensional in real-world applications. By reducing the joint probability of features to the product of individual probabilities, Naive Bayes significantly cuts down on the complexity of computations.
  - Computational Efficiency:
    - Because the model calculates probabilities for each feature independently, it can efficiently handle large datasets with many features. This efficiency is particularly valuable in domains like text classification, where the number of unique features (words) can be very large.
  - Practical Performance:
    - Interestingly, Naive Bayes often performs surprisingly well even when the independence assumption is violated. In many real-world scenarios, especially in text classification, features (like the occurrence of certain words) may not be independent. Despite this, Naive Bayes tends to yield good classification results, which is partly attributed to the nature of the underlying data distributions.

### Simplification through Feature Independence:

- In real-world datasets, hypotheses are tested based on multiple pieces of evidence provided by various features.
- This can make calculations complex. To streamline the process, Naive Bayes employs the feature independence assumption, treating each feature as independent of others.

- This uncouples multiple pieces of evidence, simplifying the computation and making the classification more tractable.

## ✓ Example of Naive Bayes Algorithm

Consider a scenario where we aim to classify fruits based on their features. For instance, if a fruit exhibits characteristics such as being red, round, and approximately 3 inches wide, we might infer that it is likely an apple. Despite potential correlations between these attributes, each feature independently contributes to the classification decision, hence the term "Naive."

The Naive Bayes (NB) algorithm offers a straightforward approach to building classification models, particularly advantageous for handling large datasets. Despite its simplicity, Naive Bayes often outperforms more complex classification methods.

### Bayes' Theorem in Naive Bayes Algorithm:

Bayes' theorem provides a mathematical framework for computing the posterior probability  $P(c|x)$ , which represents the probability of a particular class  $c$  given the predictor  $x$ , based on prior knowledge and observed data. The theorem is expressed as follows:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

The diagram shows the formula  $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$  with four labels and arrows: 'Likelihood' points to  $P(B|A)$ , 'Prior' points to  $P(A)$ , 'Posterior' points to  $P(A|B)$ , and 'Evidence' points to  $P(B)$ .

Where:

- $P(A|B)$  is the posterior probability of the class  $A$  given the predictor  $B$  (target).
- $P(A)$  is the prior probability of the class.
- $P(B|A)$  is the likelihood, representing the probability of observing the predictor  $x$  given the class  $c$ .
- $P(B)$  is the prior probability of the predictor.

## ✓ Types of Naive Bayes Algorithms

Naive Bayes algorithm encompasses three main types, each suited for different types of data distributions and applications:

## ✓ Gaussian Naive Bayes

- Gaussian Naive Bayes is employed when dealing with continuous attribute values, assuming that the values associated with each class follow a Gaussian or Normal distribution.
- For example, suppose the training data contains a continuous attribute  $x$ .
- We first segment the data by the class, and then compute the mean and variance of  $x$  in each class. Let  $\mu_i$  be the mean of the values and let  $\sigma_i$  be the variance of the values associated with the  $i$ th class.
- Suppose we have some observation value  $x_i$ .

Then, the probability distribution of  $x_i$  given a class can be computed by the following equation -

$$p(x_i | y_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i - \mu_j)^2}{2\sigma_j^2}}$$

- Where:
  - $\mu$  is the mean of the feature values for class  $Y$ .
  - $\sigma$  is the standard deviation of the feature values for class  $Y$ .
- Use Case:
  - Gaussian Naive Bayes is commonly used in scenarios where features are continuous, such as predicting numeric outcomes in medical data or financial forecasting.

## ✓ Multinomial Naive Bayes

- With a Multinomial Naïve Bayes model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial  $(p_1, \dots, p_n)$  where  $p_i$  is the probability that event  $i$  occurs.

- Multinomial Naïve Bayes algorithm is preferred to use on data that is multinomially distributed.
- This algorithm is commonly used in text categorization tasks, where it models the frequency of occurrence of different terms or words within documents.
- It's particularly effective when dealing with text data with multiple occurrences of the same feature.
- Formula:
  - The probability of observing a certain word in a document given a class is calculated using the multinomial distribution:

$$P(X_i|Y) = \frac{n_{ij} + \alpha}{n_j + \alpha \cdot V}$$

- Where
- $n_{ij}$  is the number of times feature  $i$  appears in class  $j$ .
- $n_j$  is the total number of features in class  $j$ .
- $\alpha$  is a smoothing parameter (Laplace smoothing).
- $V$  is the total number of unique features (vocabulary size).
- Use Case:
  - Multinomial Naive Bayes is widely used in text classification, such as spam detection, sentiment analysis, and topic categorization.

## ✓ Bernoulli Naive Bayes

- Bernoulli Naive Bayes operates under the multivariate Bernoulli event model, where features are treated as independent boolean variables (binary variables).
- This model is typically used for document classification tasks, where the presence or absence of certain terms or features is considered, rather than their frequencies.
- It's particularly suitable for scenarios where binary term occurrence features are utilized instead of term frequencies.
- Key Characteristics
  - Binary Features:
    - Features are binary, indicating whether a specific attribute is present (1) or absent (0).



- This contrasts with other variants like Multinomial Naive Bayes, which can handle counts or frequencies of features.
- Feature Independence:
  - Like all Naive Bayes classifiers, Bernoulli Naive Bayes assumes that the features are conditionally independent given the class label. This simplifies the calculation of probabilities.
- Application Domain:
  - Commonly used in text classification, such as spam detection, sentiment analysis, and document categorization.

### Summary:

- **Gaussian Naive Bayes:** Ideal for continuous attribute values following a Gaussian distribution.
- **Multinomial Naive Bayes:** Effective for text categorization tasks, modeling the frequency of events generated by a multinomial distribution.
- **Bernoulli Naive Bayes:** Suited for document classification tasks, treating features as independent binary variables describing inputs.

These three types of Naive Bayes algorithms cater to different data distributions and are selected based on the nature of the dataset and the specific requirements of the classification task.

## ✓ Use Case

### ✓ Problem Statement: Gender Classification Based on Facial Features

#### Background:

- In today's digital age, automated systems for gender classification have become increasingly relevant, especially in areas like security, marketing, and human-computer interaction.
- Facial recognition technology plays a crucial role in developing such systems.
- By analyzing facial features, machines can predict the gender of individuals with reasonable accuracy.

#### Objective:

- The objective of this use case is to develop a machine learning model capable of accurately classifying gender based on facial features.

- The model will be trained on a dataset containing measurements of various facial attributes such as forehead width, forehead height, nose width, etc.

[Download Dataset From Here](#)

## ✓ Exploring Data

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

```
from google.colab import drive
```

```
# Mount Google Drive
drive.mount('/content/drive')
```

➡ Drive already mounted at /content/drive; to attempt to forcibly remount, call

```
data = pd.read_csv('/content/drive/MyDrive/gender_classification_v7.csv')
```

```
data.head()
```

➡

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips
0	1	11.8	6.1	1	0	
1	0	14.0	5.4	0	0	
2	0	11.8	6.3	1	1	
3	0	14.4	6.1	0	1	
4	1	13.5	5.9	0	0	

```
# information of data
data.info()
```

➡

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   long_hair                             5001 non-null   int64
1   forehead_width_cm                     5001 non-null   float64
2   forehead_height_cm                    5001 non-null   float64
3   nose_wide                             5001 non-null   int64
4   nose_long                             5001 non-null   int64
5   lips_thin                             5001 non-null   int64
6   distance_nose_to_lip_long             5001 non-null   int64
```

```
7 gender 5001 non-null object
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

## ✓ Dataset Description

The dataset contains facial feature measurements along with gender labels.

Here's a breakdown of the columns:

### 1. long\_hair: (Numeric - Integer)

- Indicates whether the individual has long hair or not.
- Data Type: Integer (0 for no, 1 for yes)

### 2. forehead\_width\_cm: (Numeric - Float)

- Represents the width of the forehead in centimeters.
- Data Type: Float (Continuous)

### 3. forehead\_height\_cm: (Numeric - Float)

- Denotes the height of the forehead in centimeters.
- Data Type: Float (Continuous)

### 4. nose\_wide: (Numeric - Integer)

- Indicates whether the individual has a wide nose or not.
- Data Type: Integer (0 for no, 1 for yes)

### 5. nose\_long: (Numeric - Integer)

- Indicates whether the individual has a long nose or not.
- Data Type: Integer (0 for no, 1 for yes)

### 6. lips\_thin: (Numeric - Integer)

- Indicates whether the individual has thin lips or not.
- Data Type: Integer (0 for no, 1 for yes)

### 7. distance\_nose\_to\_lip\_long: (Numeric - Integer)

- Represents the distance from the nose to the lips in centimeters.
- Data Type: Integer (Continuous)

### 8. gender: (Categorical - Object)


- Indicates the gender of the individual.
- Data Type: Object (String - "Male" or "Female")

## Summary:

- The dataset consists of 5001 entries.


- There are 7 features (predictor variables) and 1 target variable (gender).
- The features include both categorical (binary) and continuous variables.
- There are no missing values in the dataset.

```
# Data of values(Max, min, std...)
data.describe()
```



	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long
<b>count</b>	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000
<b>mean</b>	0.869626	13.181484	5.946311	0.493901	0.507898
<b>std</b>	0.336748	1.107128	0.541268	0.500013	0.499988
<b>min</b>	0.000000	11.400000	5.100000	0.000000	0.000000
<b>25%</b>	1.000000	12.200000	5.500000	0.000000	0.000000
<b>50%</b>	1.000000	13.100000	5.900000	0.000000	1.000000
<b>75%</b>	1.000000	14.000000	6.400000	1.000000	1.000000
<b>max</b>	1.000000	15.500000	7.100000	1.000000	1.000000

```
# Columns of data
data.columns
```




```
Index(['long_hair', 'forehead_width_cm', 'forehead_height_cm', 'nose_wide',
       'nose_long', 'lips_thin', 'distance_nose_to_lip_long', 'gender'],
      dtype='object')
```

## ✓ Data Preprocessing

## ✓ Handling Missing Values

```
# check missing values in categorical variables
```

```
data.isnull().sum()
```



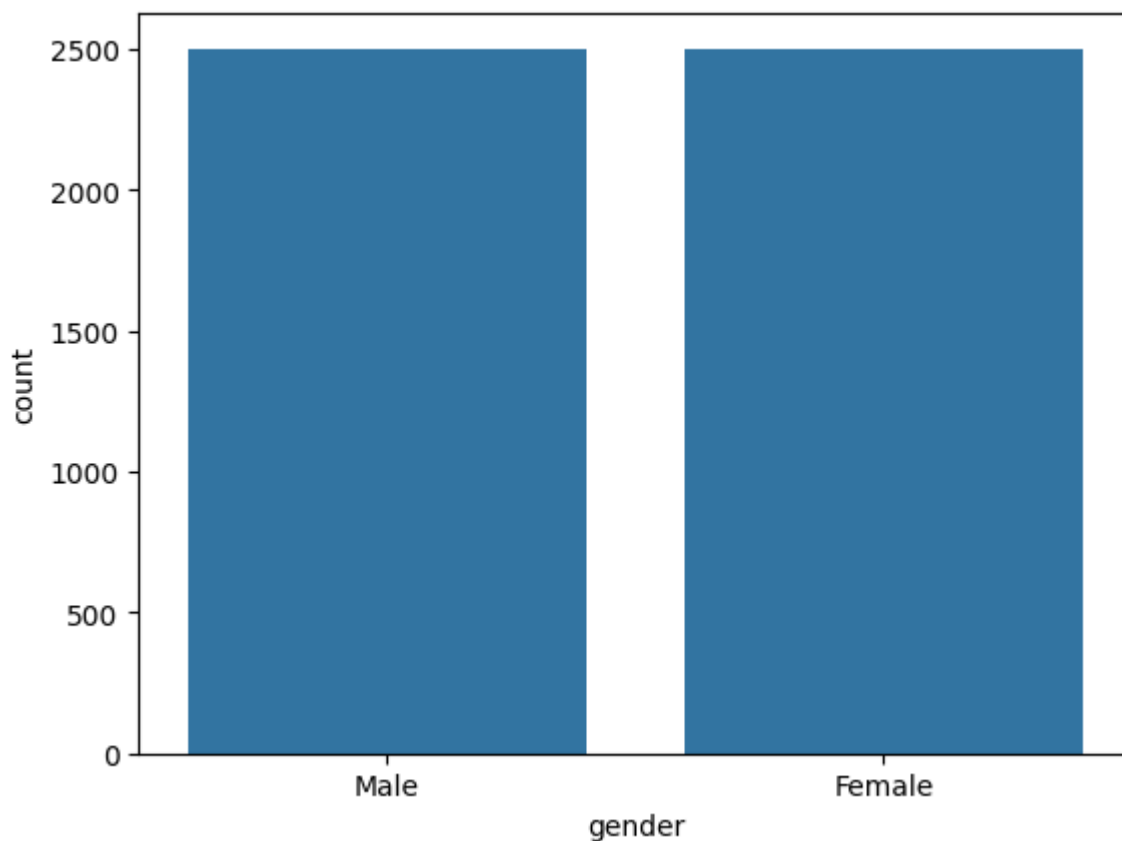
```
long_hair          0
forehead_width_cm  0
forehead_height_cm 0
nose_wide          0
nose_long          0
lips_thin          0
distance_nose_to_lip_long 0
gender            0
dtype: int64
```

There is no missing values, so we don't need to worry about this.

## ▼ Data Visualization

```
import seaborn as sns
# The number of gender in the dataset and its graphic.
sns.countplot(x = "gender", data = data)
data.loc[:, "gender"].value_counts()
```

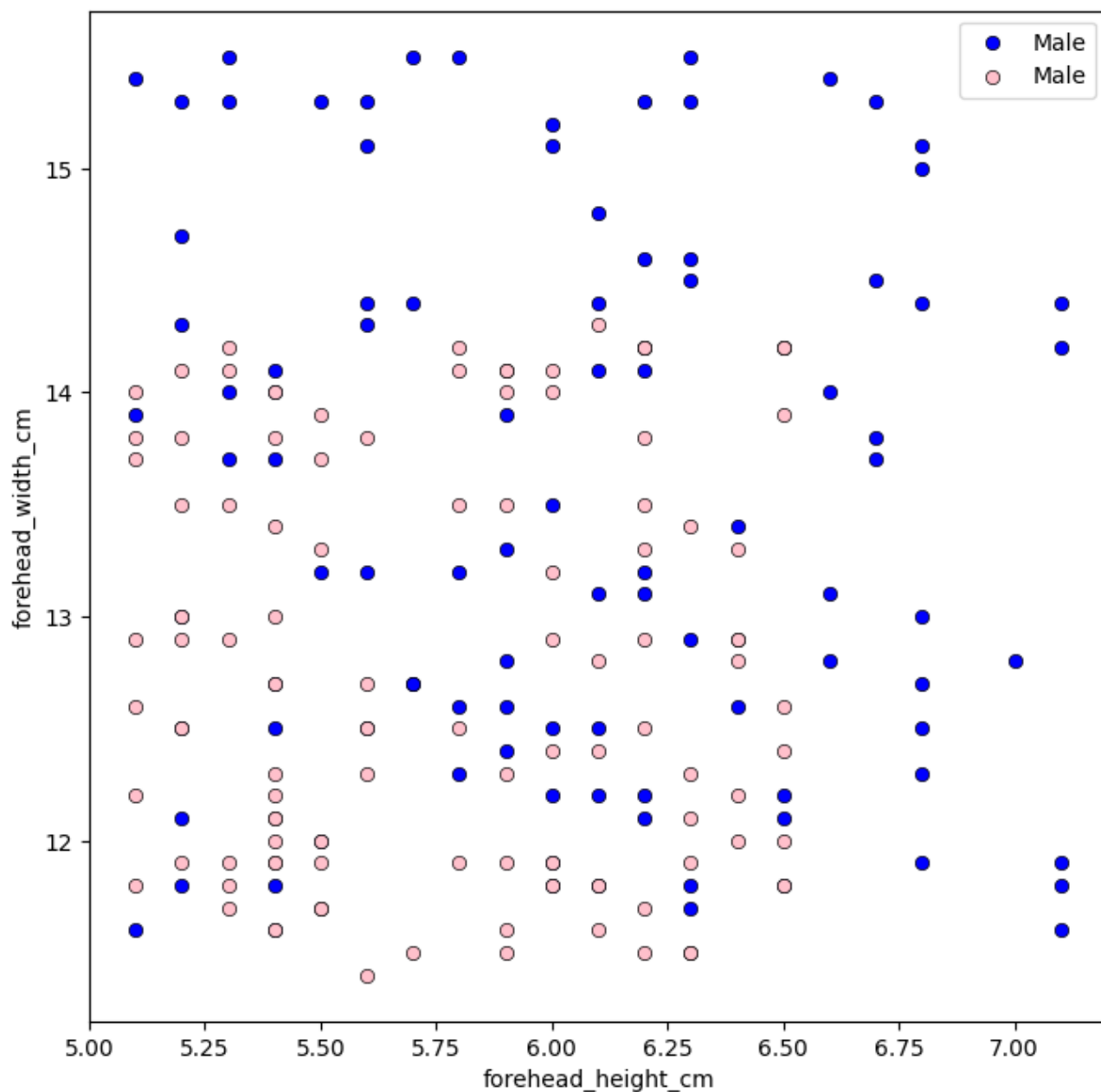
```
gender
Female    2501
Male      2500
Name: count, dtype: int64
```



No class Imbalancing here.

```
Male = data[data.gender == "Male"].iloc[:100,:]
Female = data[data.gender == "Female"].iloc[:100,:]

plt.figure(figsize = (8,8))
plt.scatter(Male.forehead_height_cm, Male.forehead_width_cm, color = "blue", label = "Male")
plt.scatter(Female.forehead_height_cm, Female.forehead_width_cm, color = "pink", label = "Female")
plt.xlabel("forehead_height_cm")
plt.ylabel("forehead_width_cm")
plt.legend()
plt.show()
```



### ✓ Categorical Variable

```
# find categorical variables
```

```
categorical = [var for var in data.columns if data[var].dtype=='O']
```

```
print('There are {} categorical variables\n'.format(len(categorical)))
```

```
print('The categorical variables are :\n\n', categorical)
```



```
There are 1 categorical variables
```

```
The categorical variables are :
```

```
['gender']
```

## ▼ Label Encoding

```
data.gender = [1 if i == "Male" else 0 for i in data.gender]
```

```
data.info()
```

```
>>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   long_hair                             5001 non-null   int64
1   forehead_width_cm                     5001 non-null   float64
2   forehead_height_cm                    5001 non-null   float64
3   nose_wide                             5001 non-null   int64
4   nose_long                             5001 non-null   int64
5   lips_thin                             5001 non-null   int64
6   distance_nose_to_lip_long             5001 non-null   int64
7   gender                                5001 non-null   int64
dtypes: float64(2), int64(6)
memory usage: 312.7 KB
```

```
data.head()
```

```
>>>
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips
0	1	11.8	6.1	1	0	
1	0	14.0	5.4	0	0	
2	0	11.8	6.3	1	1	
3	0	14.4	6.1	0	1	
4	1	13.5	5.9	0	0	

```
# x_data
x_data = data.drop(["gender"],axis = 1)
```

```
# y_data
y_data = data.gender.values
```

```
x_data.head()
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips
0	1	11.8	6.1	1	0	
1	0	14.0	5.4	0	0	
2	0	11.8	6.3	1	1	
3	0	14.4	6.1	0	1	
4	1	13.5	5.9	0	0	

y\_data

```
array([1, 0, 1, ..., 0, 0, 1])
```

```
# Train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0
```

## ✓ Training Model

### ✓ Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb = GaussianNB()
```

```
gnb.fit(x_train, y_train)
```

```
▼ GaussianNB
GaussianNB()
```

```
print("print Train for accuracy of NBC algo: ", gnb.score(x_train,y_train))
print("print Test for accuracy of NBC algo: ", gnb.score(x_test,y_test))
```

```
print Train for accuracy of NBC algo: 0.97
print Test for accuracy of NBC algo: 0.9713524317121919
```

```
y_pred = gnb.predict(x_test)
```

y\_pred

```
array([0, 1, 0, ..., 1, 0, 1])
```



```

from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))

⇒ Model accuracy score: 0.9714

y_pred_train = gnb.predict(x_train)

y_pred_train

⇒ array([0, 0, 1, ..., 0, 1, 1])

print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_p

⇒ Training-set accuracy score: 0.9700

# print the scores on training and test set

print('Training set score: {:.4f}'.format(gnb.score(x_train, y_train)))

print('Test set score: {:.4f}'.format(gnb.score(x_test, y_test)))

⇒ Training set score: 0.9700
   Test set score: 0.9714

```

- The training-set accuracy score is 0.97 while the test-set accuracy to be 0.9714.
- These two values are quite comparable. So, there is no sign of overfitting.

```

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

⇒ Confusion matrix

[[723  22]
 [ 21 735]]

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

```

```

⇒
True Positives(TP) = 723

True Negatives(TN) = 735

```

False Positives(FP) = 22

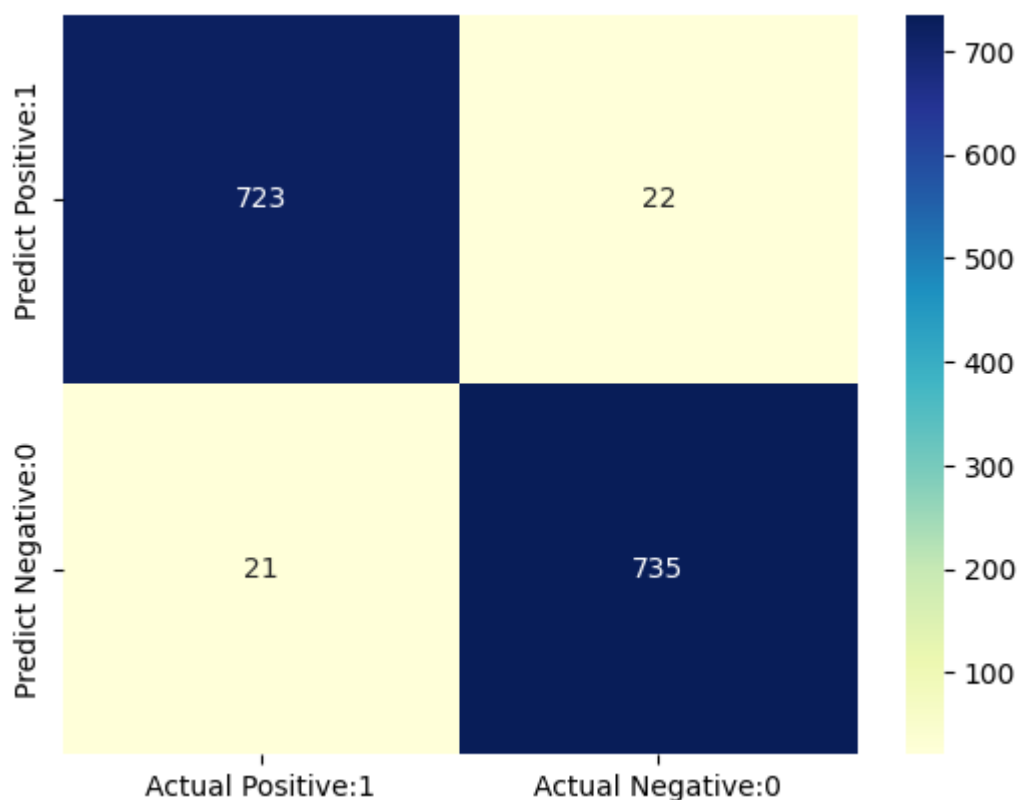
False Negatives(FN) = 21

```
# visualize confusion matrix with seaborn heatmap
import seaborn as sns
```

```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                          index=['Predict Positive:1', 'Predict Negative:0'])
```

```
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

 <Axes: >



```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.97	0.97	0.97	745
1	0.97	0.97	0.97	756
accuracy			0.97	1501
macro avg	0.97	0.97	0.97	1501
weighted avg	0.97	0.97	0.97	1501

## ✓ Multinomial Naive Bayes

```

from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()

mnb.fit(x_train, y_train)

print("print Train for accuracy of NBC algo: ", mnb.score(x_train,y_train))
print("print Test for accuracy of NBC algo: ", mnb.score(x_test,y_test))

⇒ print Train for accuracy of NBC algo: 0.9565714285714285
   print Test for accuracy of NBC algo: 0.960692871419054

y_pred_mnb = mnb.predict(x_test)

y_pred_mnb
⇒ array([0, 1, 0, ..., 1, 0, 1])

from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_mnb))
⇒ Model accuracy score: 0.9607

y_pred_mnb_train = mnb.predict(x_train)

y_pred_mnb_train
⇒ array([0, 1, 1, ..., 0, 1, 1])

print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_p
⇒ Training-set accuracy score: 0.9566

# print the scores on training and test set

print('Training set score: {:.4f}'.format(mnb.score(x_train, y_train)))

print('Test set score: {:.4f}'.format(mnb.score(x_test, y_test)))

⇒ Training set score: 0.9566
   Test set score: 0.9607

```

- The training-set accuracy score is 0.9566 while the test-set accuracy to be 0.9607.
- These two values are quite comparable. So, there is no sign of overfitting.

```

from sklearn.metrics import confusion_matrix

cm_mnb = confusion_matrix(y_test, y_pred_mnb)

```

```
print('Confusion matrix\n\n', cm_mnb)
```

⇒ Confusion matrix

```
[[692  53]
 [  6 750]]
```

```
print('\nTrue Positives(TP) = ', cm_mnb[0,0])
```

```
print('\nTrue Negatives(TN) = ', cm_mnb[1,1])
```

```
print('\nFalse Positives(FP) = ', cm_mnb[0,1])
```

```
print('\nFalse Negatives(FN) = ', cm_mnb[1,0])
```

⇒

```
True Positives(TP) = 692
```

```
True Negatives(TN) = 750
```

```
False Positives(FP) = 53
```

```
False Negatives(FN) = 6
```

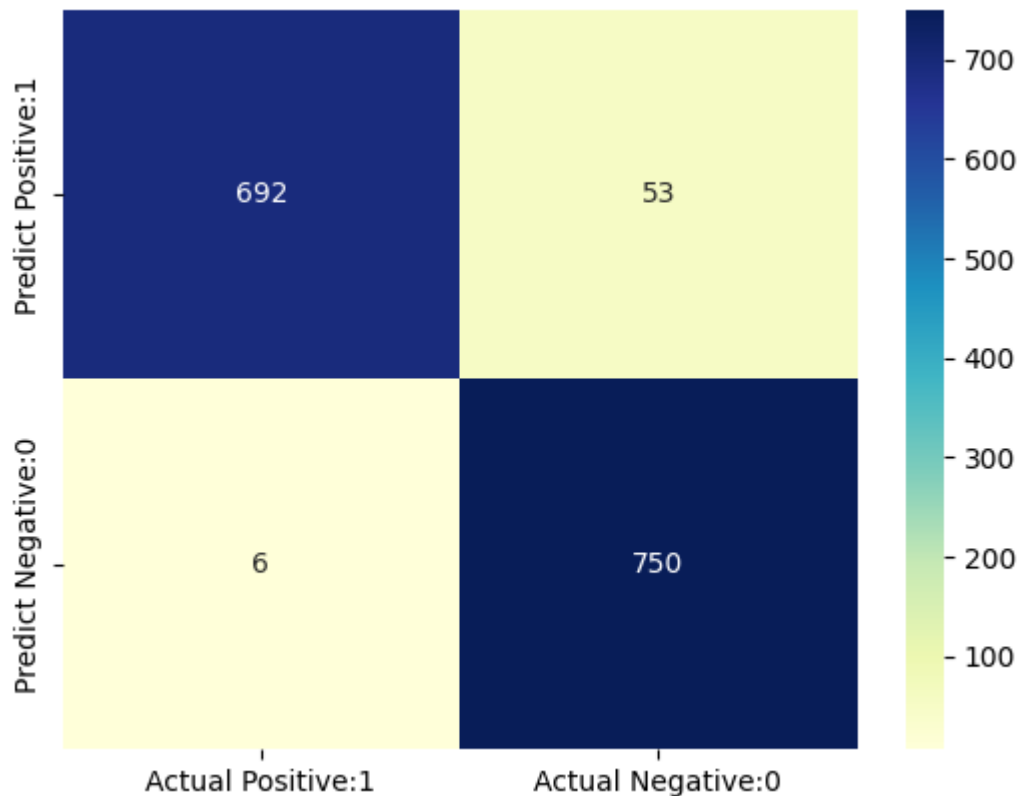
```
# visualize confusion matrix with seaborn heatmap
```

```
import seaborn as sns
```

```
cm_matrix = pd.DataFrame(data=cm_mnb, columns=['Actual Positive:1', 'Actual Negat
index=['Predict Positive:1', 'Predict Negative:0
```

```
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

↗ <Axes: >



```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_mnb))
```

↗

	precision	recall	f1-score	support
0	0.99	0.93	0.96	745
1	0.93	0.99	0.96	756
accuracy			0.96	1501
macro avg	0.96	0.96	0.96	1501
weighted avg	0.96	0.96	0.96	1501

## ✓ Bernoulli Naive Bayes

```
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB()
```

```
bnb.fit(x_train, y_train)
```

```
print("print Train for accuracy of NBC algo: ", bnb.score(x_train,y_train))
print("print Test for accuracy of NBC algo: ", bnb.score(x_test,y_test))
```

↗

```
print Train for accuracy of NBC algo: 0.9585714285714285
print Test for accuracy of NBC algo: 0.9640239840106596
```

```
y_pred_bnb = mnb.predict(x_test)
```

```
y_pred_bnb
```

```
→ array([0, 1, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import accuracy_score
```

```
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred_bnb))
```

```
→ Model accuracy score: 0.9607
```

```
y_pred_bnb_train = mnb.predict(x_train)
```

```
y_pred_bnb_train
```

```
→ array([0, 1, 1, ..., 0, 1, 1])
```

```
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_p
```

```
→ Training-set accuracy score: 0.9566
```

```
# print the scores on training and test set
```

```
print('Training set score: {:.4f}'.format(bnb.score(x_train, y_train)))
```

```
print('Test set score: {:.4f}'.format(bnb.score(x_test, y_test)))
```

```
→ Training set score: 0.9586  
Test set score: 0.9640
```

- The training-set accuracy score is 0.9586 while the test-set accuracy to be 0.9640.
- These two values are quite comparable. So, there is no sign of overfitting.

```
from sklearn.metrics import confusion_matrix
```

```
cm_bnb = confusion_matrix(y_test, y_pred_bnb)
```

```
print('Confusion matrix\n\n', cm_bnb)
```

```
→ Confusion matrix  
  
[[692  53]  
 [  6 750]]
```

```
print('\nTrue Positives(TP) = ', cm_bnb[0,0])

print('\nTrue Negatives(TN) = ', cm_bnb[1,1])

print('\nFalse Positives(FP) = ', cm_bnb[0,1])

print('\nFalse Negatives(FN) = ', cm_bnb[1,0])
```



```
True Positives(TP) = 692

True Negatives(TN) = 750

False Positives(FP) = 53

False Negatives(FN) = 6
```

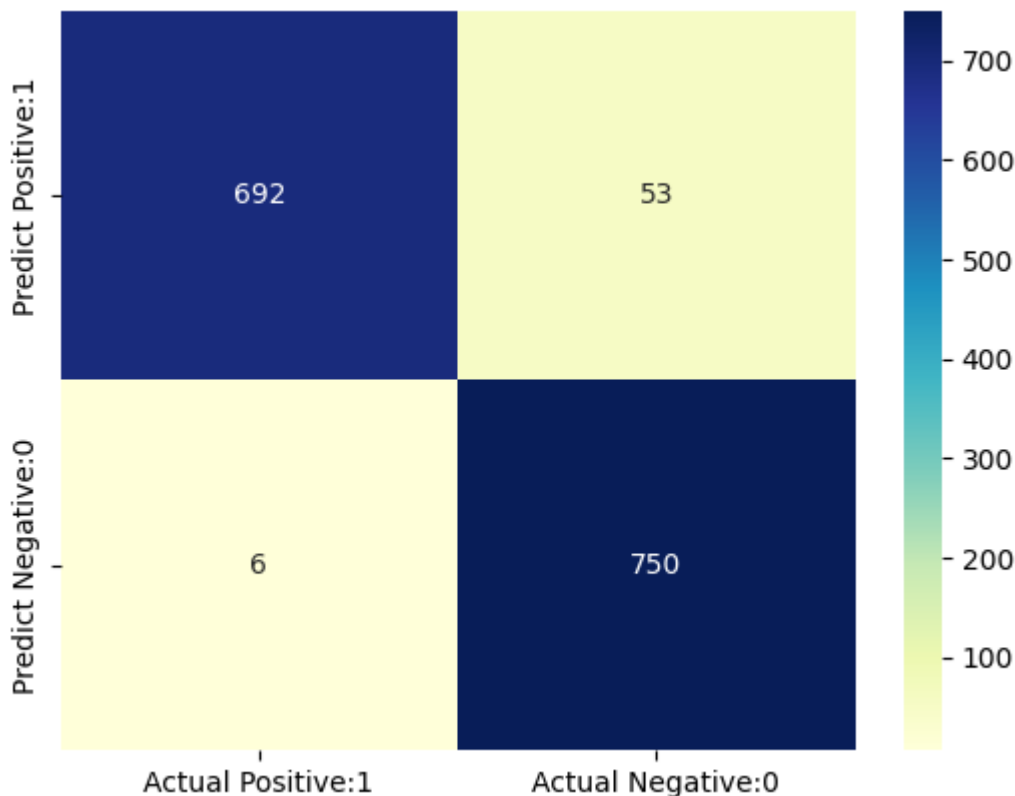
```
# visualize confusion matrix with seaborn heatmap
import seaborn as sns
```

```
cm_matrix = pd.DataFrame(data=cm_bnb, columns=['Actual Positive:1', 'Actual Negat
index=['Predict Positive:1', 'Predict Negative:0
```

```
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```



```
<Axes: >
```



```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_bnb))
```



```
precision    recall  f1-score   support
```

	0	0.99	0.93	0.96	745
	1	0.93	0.99	0.96	756
accuracy				0.96	1501
macro avg		0.96	0.96	0.96	1501
weighted avg		0.96	0.96	0.96	1501

✓ Advantages of Naive Bayes

- 1. **Efficiency and Speed:** Naive Bayes is known for its simplicity and speed in predicting the class of test data. It's particularly efficient when dealing with large datasets.