1. Importing Required Libraries

- **Head:** `import numpy as np`
- **Tail:** `from sklearn.preprocessing import LabelEncoder`
- **Explanation:**
  This section imports necessary libraries for data handling (`numpy`, `pandas`), machine learning (`train_test_split`, `GaussianNB`, `LabelEncoder`), performance evaluation (`confusion_matrix`, `accuracy_score`, etc.), and visualization (`matplotlib`, `seaborn`).

🔸 2. Load and Inspect Dataset

- **Head:** `data = pd.read_csv('Iris.csv')`
- **Tail:** `data.isnull().sum()`
- **Explanation:**
  Loads the Iris dataset from a CSV file, shows the first 5 rows, describes all columns, prints dataset info, checks shape, class distribution in the target column, and null values. Ensures the data is clean before model training.

🔸 3. Feature and Label Separation

- **Head:** `x = data.iloc[:,1:5]`
- **Tail:** `y = data['Species']`
- **Explanation:**
  Extracts features (SepalLength, SepalWidth, PetalLength, PetalWidth) and labels (Species). Note: `iloc[:,1:5]` skips the first column (ID).

🔸 4. Label Encoding

- **Head:** `encode = LabelEncoder()`
- **Tail:** `y = encode.fit_transform(y)`
- **Explanation:**
  Converts categorical labels (e.g., 'Iris-setosa') into numerical format (e.g., 0, 1, 2) using `LabelEncoder` so they can be used in training.

🔸 5. Train-Test Split

- **Head:** `x_train,x_test,y_train,y_test = train_test_split(...)`
- **Tail:** `random_state = 0`
- **Explanation:**
  Splits the dataset into training (70%) and testing (30%) subsets to evaluate model performance on unseen data.

🔸 6. Model Training

- **Head:** `naive_bayes = GaussianNB()`
- **Tail:** `naive_bayes.fit(x_train,y_train)`
- **Explanation:**
  Initializes the Naïve Bayes classifier (GaussianNB for continuous features) and trains it on the training data.

🔸 7. Prediction

- **Head:** `pred = naive_bayes.predict(x_test)`
- **Tail:** `y_test`
- **Explanation:**
  The trained model makes predictions on the test data, which are then compared to the actual test labels.

🔸 8. Confusion Matrix Calculation

- **Head:** `matrix = confusion_matrix(...)`
- **Tail:** `print(matrix)`
- **Explanation:**
  Computes a confusion matrix comparing actual vs predicted values, which helps analyze the model's classification ability across different classes.

🔸 9. Binary Matrix Reshaping

- **Head:** `tp, fn, fp, tn = confusion_matrix(...)`
- **Tail:** `.reshape(-1)`
- **Explanation:**
  Extracts TP, TN, FP, FN values only for binary evaluation (between class 1 and 0). This is less accurate for multi-class problems like Iris.

🔸 10. Confusion Matrix Visualization

- **Head:** `conf_matrix = ConfusionMatrixDisplay(...)`

- **Tail:** `plt.show()`

- **Explanation:**
  Uses `ConfusionMatrixDisplay` to visualize the confusion matrix using a color map (`YlGn`) for better understanding of class-wise performance.

🔸 11. Classification Report

- **Head:** `print(classification_report(...))`

- **Tail:** `f1_score` is included in the output

- **Explanation:**
  Displays precision, recall, F1-score, and support for each class. Useful for multi-class performance evaluation.

🔸 12. Performance Metrics

- **Head:** `print('\nAccuracy: {:.2f}'.format(...))`

- **Tail:** `print('False Positive Rate :',...)`

- **Explanation:**
  Calculates and prints accuracy, error rate, precision, recall, specificity, and false positive rate for evaluating the model's performance using binary metrics.