Naive Bayes Classifier: Iris Dataset Prediction

Objective:

The primary objective of this task is to build a Naive Bayes classifier using the Iris dataset to predict the species of iris flowers based on four features: sepal length, sepal width, petal length, and petal width.

Data Processing Steps:

1- **Data Loading:** The dataset is loaded from a file (iris.data) using pandas.read_csv(). The dataset consists of 150 samples with four features (sepal length, sepal width, petal length, and petal width) and a target variable representing the species (class).

2- **Feature and Target Extraction:**

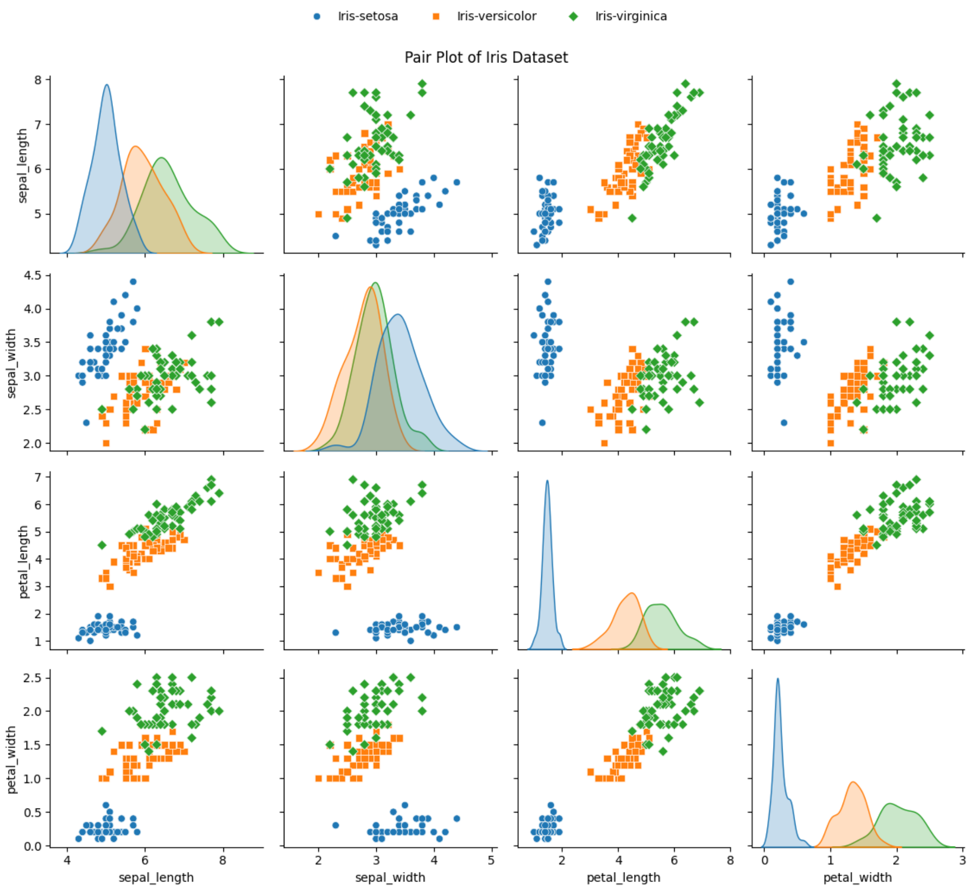
- **Features (X):** The features used for classification are the measurements of the iris flowers: **sepal_length**, **sepal_width**, **petal_length**, and **petal_width**.

- **Target (y):** The target variable is the class of the flower, which includes three species: **Iris-setosa**, **Iris-versicolor**, and **Iris-virginica.**

3- **Label Encoding:** The class labels are categorical, so they are converted into integer values using **LabelEncoder().**

4- **Splitting Data:** The dataset is divided into training (70%) and testing (30%) sets using train_test_split() to evaluate the model's performance on unseen data.

Visualize the distribution of each feature using histograms.



Model Choice: Naive Bayes Classifier:

The Naive Bayes classifier is chosen for this task because it is simple, fast, and effective for multi-class classification problems. Gaussian Naive Bayes is used here because the features are continuous, and this algorithm assumes that the continuous features follow a Gaussian distribution.

Model Training and Prediction:

- **Training**: The model is trained using the training data with the GaussianNB() class from sklearn.naive_bayes.
- **Prediction:** The model predicts the class labels for the test data.

Performance Evaluation:

- **Accuracy**: The accuracy of the model is calculated as the proportion of correctly classified samples out of the total samples in the test set.
- **Confusion Matrix**: This matrix shows the number of correct and incorrect predictions for each class, providing insights into where the model is making errors.

```
Accuracy: 0.9777777777777777
Confusion Matrix:
 [[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.93      1.00      0.96        13

       accuracy                           0.98        45
      macro avg       0.98      0.97      0.97        45
   weighted avg       0.98      0.98      0.98        45
```

- **Classification Report**: This report includes precision, recall, and F1-score for each class, offering a detailed evaluation of the model's performance.

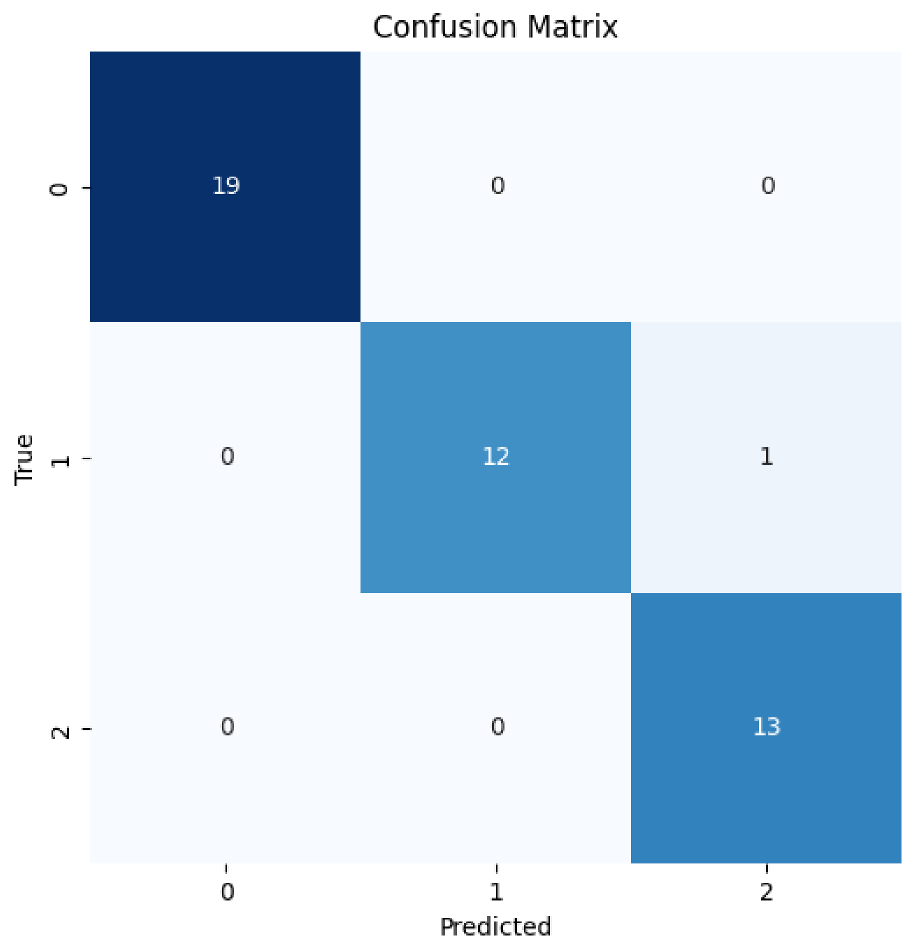
```
class_report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_)
```

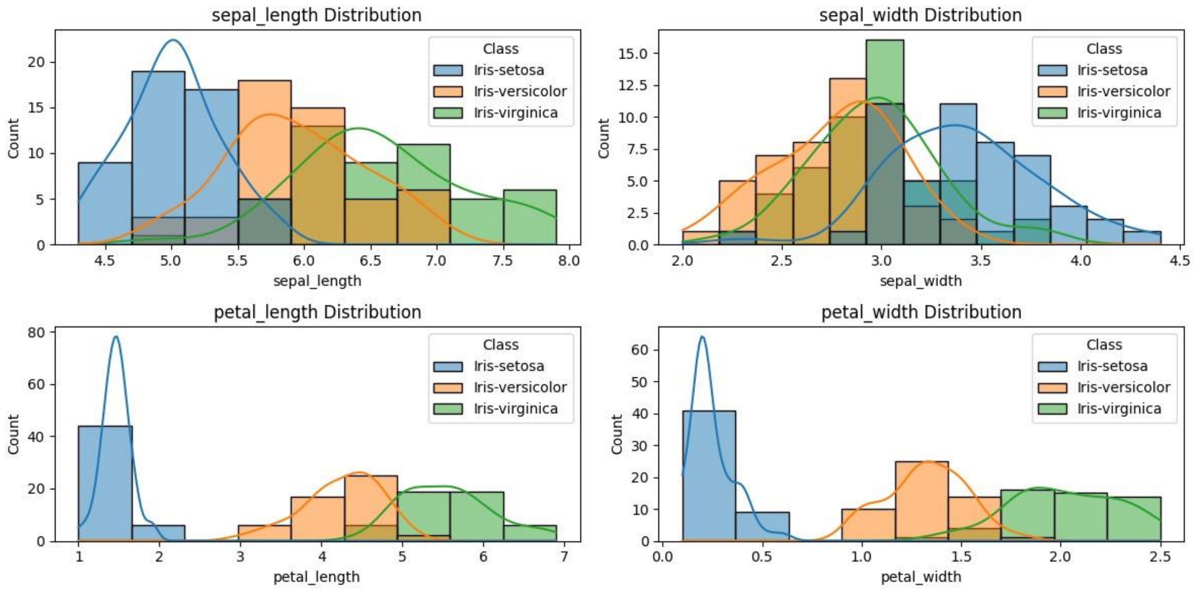
All Python code implementing the Naive Bayes classifier and output:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
file_path = '/content/drive/MyDrive/Colab Notebooks/iris.data'
columns = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width', 'Class']
iris = pd.read_csv(file_path, names=columns)

print(iris.head(10))
# Divide the data set into features (X) and target variable
(y)
X = iris.iloc[:, 0:4].values
y = iris.iloc[:, 4].values
```

```python
# Encode the target variable
le = LabelEncoder()
y = le.fit_transform(y)
ax = sns.pairplot(iris, hue='Class', markers=["o", "s", "D"])
plt.suptitle("Pair Plot of Iris Dataset")
sns.move_legend(
    ax, "lower center",
    bbox_to_anchor=(.5, 1), ncol=3, title=None, frameon=False)
plt.tight_layout()
plt.show()
# Visualize the distribution of each feature using histograms.
plt.figure(figsize=(12, 6))
for i, feature in enumerate(columns[:-1]):
    plt.subplot(2, 2, i + 1)
    sns.histplot(data=iris, x=feature, hue='Class', kde=True)
    plt.title(f'{feature} Distribution')

plt.tight_layout()
plt.show()
correlation_matrix = iris.corr(numeric_only = True)
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
X_train, X_test, y_train, y_test = train_test_split(X , y,
test_size=0.3, random_state=42)
gnb = GaussianNB()

gnb.fit(X_train, y_train)

#Predict the target for the test data
y_pred = gnb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_)

# Display results
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:\n",conf_matrix)
print("Classification Report:\n",  class_report)
accuracy = accuracy_score(y_test, y_pred)
conf_m = confusion_matrix(y_test, y_pred)
```

```
#Display the accuracy
print(f'Accuracy: {accuracy:.2f}')

#Display the confusion matrix as a heatmap
plt.figure(figsize=(6, 6))
sns.heatmap(conf_m, annot=True, fmt="d", cmap="Blues",
cbar=False, square=True)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```




sepal_length Distribution
Count
sepal_length
Class
Iris-setosa
Iris-versicolor
Iris-virginica
sepal_width Distribution
Count
sepal_width
Class
Iris-setosa
Iris-versicolor
Iris-virginica
petal_length Distribution
Count
petal_length
Class
Iris-setosa
Iris-versicolor
Iris-virginica
petal_width Distribution
Count
petal_width
Class
Iris-setosa
Iris-versicolor
Iris-virginica