The Iris Dataset: Naive Bayes Classifier

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `numpy` and `pandas` for data manipulation, `sklearn` for building the Naive Bayes Classifer and evaluating its performance, and `matplotlib` and `seaborn` for creating a confusion matrix heatmap.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Loading the dataset

The Iris dataset available under the `sklearn` module in Python is known for its simplicity. It contains the measurements of 150 iris flowers from three different species: *Iris setosa*, *Iris versicolor*, and *Iris virginica*.

The `sklearn` module has a very straightforward set of data on these iris species, consisting of the following:

- Features in the Iris dataset:

1. sepal length (in cm)
2. sepal width (in cm)
3. petal length (in cm)
4. petal width (in cm)

- Target classes to predict:

1. *Iris setosa*
2. *Iris versicolor*
3. *Iris virginica*

```
iris = load_iris()
X_iris = iris.data
y_iris = iris.target
```

3. Splitting the data into training and testing sets

We now split the dataset into training and testing sets using the `train_test_split` function. We specify `test_size=0.2` to reserve 20% of the data for testing and `random_state=42` to ensure the data is split consistently every time we run the code.

```
X_iris_train, X_iris_test, y_iris_train, y_iris_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)
```

4. Creating and training the Naive Bayes Classifier

In this section, we create an instance of the Gaussian Naive Bayes model and train it. The model learns the relationship between the features (`X_iris_train`) and the target labels (`y_iris_train`).

```
gnb_iris = GaussianNB()
gnb_iris.fit(X_iris_train, y_iris_train)
```

```
GaussianNB
GaussianNB()
```

5. Making predictions

Once the model is trained, we use it to make predictions on the test data (`X_iris_test`). The model generates predicted labels (`y_iris_pred`) for the test set, which we will compare with the actual labels (`y_iris_test`) to evaluate its performance.

```
y_iris_pred = gnb_iris.predict(X_iris_test)

# Displaying the predicted labels
y_iris_pred
```

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

6. Evaluating the model

Now, we calculate the accuracy of the model and print a detailed classification report, which includes precision, recall, and F1-score for each class, providing insight into the model's performance.

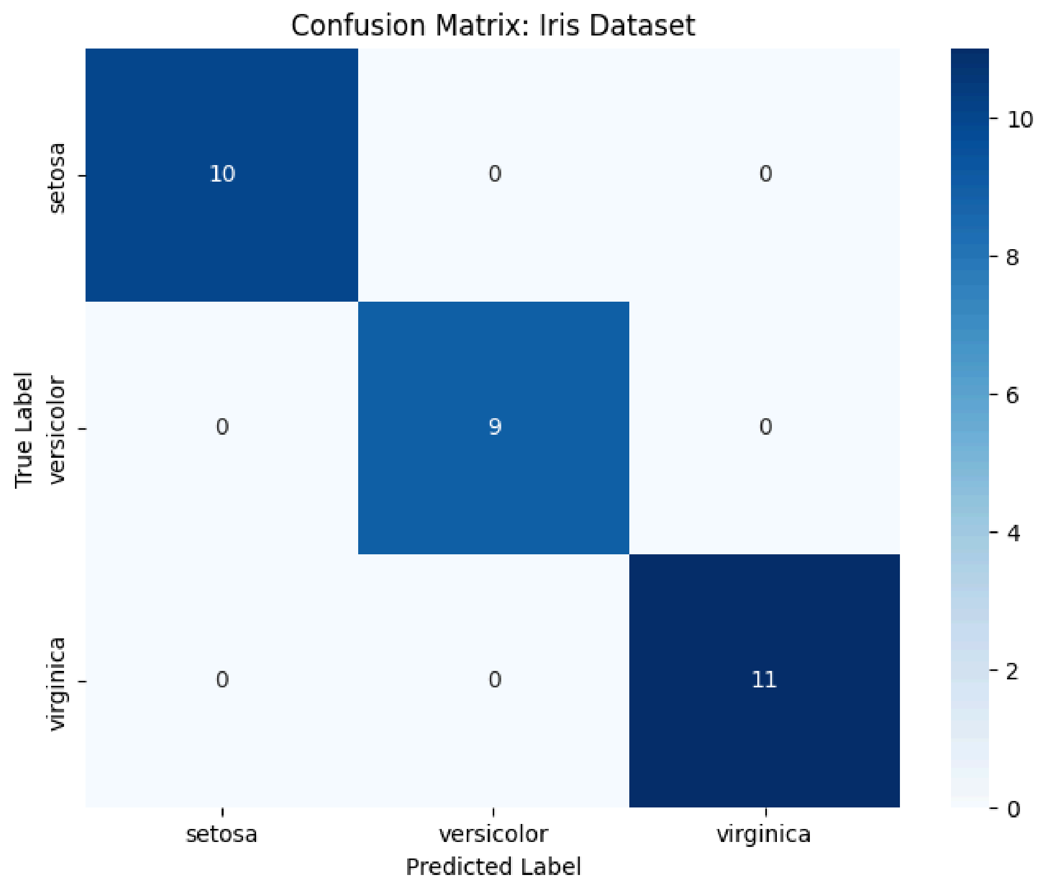
```
iris_accuracy = accuracy_score(y_iris_test, y_iris_pred)
iris_classification_report = classification_report(y_iris_test, y_iris_pred)

print(f'Accuracy: {iris_accuracy:.2f}')
print("\nClassification Report:\n", iris_classification_report)
```

```
Accuracy: 1.00

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

7. Generating the confusion matrix

Finally, we visualise the confusion matrix, which shows how many instances of each class were correctly or incorrectly classified. This helps us understand specific areas where the model may be confusing certain classes.

```
iris_conf_matrix = confusion_matrix(y_iris_test, y_iris_pred)

plt.figure(figsize=(8,6))
sns.heatmap(iris_conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.title('Confusion Matrix: Iris Dataset')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Note:

Getting an accuracy of 1 (or 100%) can be surprising and may indicate a few potential issues. The following are some of the possible reasons why we might encounter this result, especially when working with classifiers like Naive Bayes on small datasets:

- **Small sample size:** The dataset consists of 50 samples from each of three species of Iris. In such cases, the model may perform perfectly on the training and testing sets simply due to the limited number of examples.
- **Overfitting:** While Naive Bayes is generally simple, if the training data is small, it might memorise it fully, capturing noise rather than the underlying distribution.
- **Simplicity of the dataset:** Some datasets might be inherently simple or have clear boundaries, making it easy for the classifier to achieve perfect accuracy. If the features are very informative, the model might classify the training and test sets correctly.