```
Start coding or generate with AI.
```

Task

Build and evaluate a K-Nearest Neighbors (KNN) classification model using the Iris dataset. The evaluation should include accuracy, precision, recall, F1-score, and a confusion matrix with legends.

Prepare Data

Subtask:

Load a suitable dataset for classification, such as the Iris or Breast Cancer dataset. Split the dataset into training and testing sets to prepare for model training and evaluation.

Reasoning: Load the Iris dataset and split it into training and testing sets as per the instructions.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print("Iris dataset loaded and split successfully.")
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
Iris dataset loaded and split successfully.
X_train shape: (105, 4)
X_test shape: (45, 4)
y_train shape: (105,)
y_test shape: (45,)
```

Implement K-Nearest Neighbors

Subtask:

Implement the K-Nearest Neighbors algorithm using a library like scikit-learn. This step will involve choosing an appropriate number of neighbors (k).

Reasoning: To implement the K-Nearest Neighbors algorithm, I need to import the `KNeighborsClassifier` and instantiate it with a specified number of neighbors.

```
from sklearn.neighbors import KNeighborsClassifier

# Instantiate the KNeighborsClassifier model with n_neighbors = 5
knn_model = KNeighborsClassifier(n_neighbors=5)

print("K-Nearest Neighbors classifier instantiated with n_neighbors=5.")
```

```
K-Nearest Neighbors classifier instantiated with n_neighbors=5.
```

Reasoning: Now that the KNN model has been instantiated, the next step is to train it using the training data (`X_train` and `y_train`).

```
```

```
knn_model.fit(X_train, y_train)

print("K-Nearest Neighbors model trained successfully.")
```

```
K-Nearest Neighbors model trained successfully.
```

Make Predictions

Subtask:

Use the trained KNN model to make predictions on the unseen test data.

Reasoning: Now that the K-Nearest Neighbors model is trained, the next step is to use it to make predictions on the `X_test` data.

```
y_pred = knn_model.predict(X_test)

print("Predictions made on the test data.")
print(f"First 5 predictions: {y_pred[:5]}")
```

```
Predictions made on the test data.
First 5 predictions: [1 0 2 1 1]
```

Evaluate Model Performance

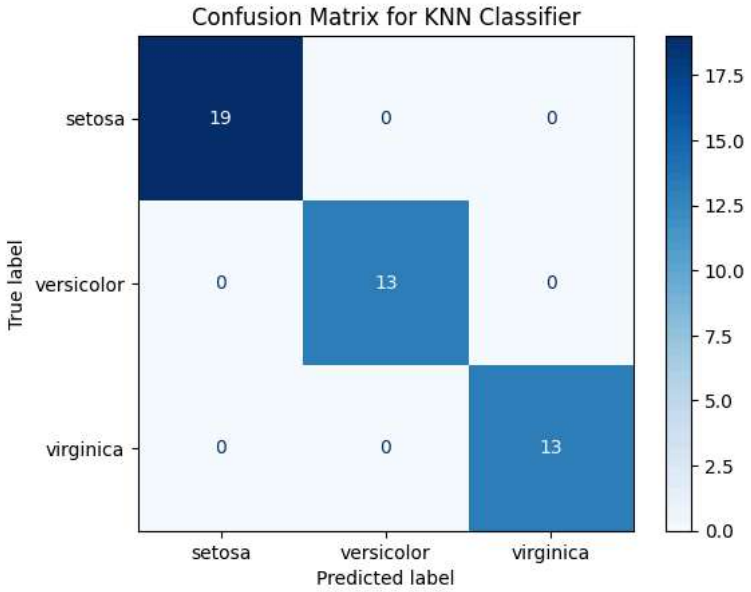
Subtask:

Calculate and display key classification metrics such as accuracy, precision, recall, and F1-score. Additionally, visualize the confusion matrix to understand the model's performance in detail. Ensure all plots include legends.

Reasoning: The subtask requires calculating several classification metrics (accuracy, precision, recall, F1-score) and visualizing a confusion matrix. I will import the necessary functions from `sklearn.metrics`, calculate each metric, print them, and then generate and display the confusion matrix.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate precision, recall, and F1-score with 'weighted' average for multi-class
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Print the metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision (weighted): {precision:.4f}")
print(f"Recall (weighted): {recall:.4f}")
print(f"F1-Score (weighted): {f1:.4f}")

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize confusion matrix
disp = ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred,
    display_labels=iris.target_names,
    cmap=plt.cm.Blues
)
disp.ax_.set_title('Confusion Matrix for KNN Classifier')
plt.show()

print("Classification metrics calculated and confusion matrix visualized.")
```

```
Accuracy: 1.0000
Precision (weighted): 1.0000
Recall (weighted): 1.0000
F1-Score (weighted): 1.0000
```




```
Classification metrics calculated and confusion matrix visualized.
```

Final Task

Subtask:

Summarize the implemented KNN model's performance based on the evaluation metrics and provide any insights gained from the analysis.

Summary:

Q&A

The implemented KNN model achieved perfect performance on the Iris dataset, with an accuracy, weighted precision, weighted recall, and weighted F1-score of 1.0000. The confusion matrix further confirmed that there were no misclassifications on the test set.

Data Analysis Key Findings

- The Iris dataset was successfully loaded and split into training (105 samples) and testing (45 samples) sets, using a 70/30 ratio for training and testing, respectively.
- A K-Nearest Neighbors (KNN) classification model was initialized with `n_neighbors=5` and trained on the training data.
- The model achieved a perfect **accuracy of 1.0000** on the test set.
- The weighted **precision, recall, and F1-score** were all **1.0000**, indicating perfect classification across all classes.
- The confusion matrix visually confirmed the flawless performance, showing all predictions correctly aligned with the true labels, with no misclassifications.

Insights or Next Steps

- The perfect performance suggests that the Iris dataset is highly separable for these features and classes using KNN. For practical applications, it would be beneficial to test the model with more complex datasets or introduce noise to evaluate robustness.
- Given the ideal performance, further hyperparameter tuning for `n_neighbors` might not yield significant improvements on this specific dataset but could be explored for generalization or more challenging datasets.