Iris Flower Classification

1. Loading and Displaying First and Last 5 rows of IRIS dataset

```
import pandas as pd

#loading the dataset and displaying first and last 5 rows/records
iris_data = pd.read_csv("/home/reddy/iris/IRIS.csv")
iris_data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

2. preprocessing details

2.1. Checking null values in IRIS data

```
iris_data.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

Above output verifies that there are no null values in the IRIS data set.

```
print(iris_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

There are 5 features with 150 non-null, 4 float64 Data type and 1 Object records

```
print(iris_data['species'].unique())
```

```
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

There are 3 unique species as mentioned in the task (verified)

Data Preprocessing Status:

- **No missing values** → All columns are fully populated.
- **Correct data types** → Numerical features are `float64`, and `species` is `object` (as expected for categorical labels).
- **Clean categorical labels** → Only 3 unique species: `'Iris-setosa'`, `'Iris-versicolor'`, `'Iris-virginica'`.

Note: No preprocessing like handling missing values or data type conversion is required.

I thought of a Label Encoding species column so that machine learning models can Easily understand and work with them.

Label Encoded Target column into numericals like show below

Species Label	Encoded Label
Iris-setosa	0
Iris-versicolor	1
Iris-virginica	2

Note:

1. *The target column, representing the species labels, is transformed using label encoding to convert categorical species names into numerical values. This facilitates better processing and understanding by machine learning algorithms, which require numerical input for effective model training and prediction.*

3. Model Selection and Evaluation

I have selected 6 Classification models to train, test and Validate

1. Logistic Regression (Multinomial)

2. K-Nearest Neighbors (K-NN)

3. Support Vector Machine (SVM)

4. Decision Tree 5. Random Forest

6. Naive Bayes

Source Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load Iris dataset
iris_data = pd.read_csv("/home/reddy/iris/IRIS.csv")

# Features and target
X = iris_data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = iris_data['species']

# Encode target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Models to train
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

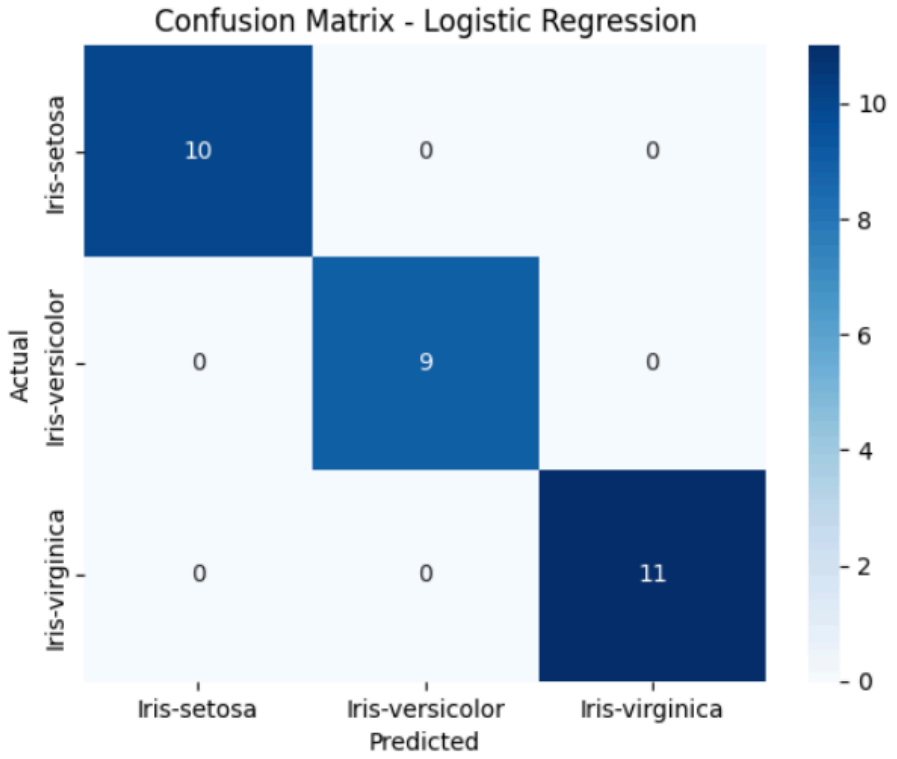
```python
models = {
    'Logistic Regression': LogisticRegression(solver='lbfgs', max_iter=200),
    'K-NN': KNeighborsClassifier(),
    'SVM': SVC(kernel='rbf', probability=True),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB()
}

# Train, evaluate, and cross-validate models
for name, model in models.items():
    print(f"\n--- {name} ---")

    # Train and predict
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluation
    acc = accuracy_score(y_test, y_pred)
    print("Test Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test, y_pred, target_names=le.classes_))

    # Cross-validation
    cv_scores = cross_val_score(model, X, y_encoded, cv=5)
    print(f"Cross-Validation Accuracy: ", cv_scores)
    print(f"Cross-Validation Accuracy: Mean = {cv_scores.mean():.4f}, Std = {cv_scores.std():.4f}")

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

Above mentioned 6 models are trained, tested, validated and Cross-Validated.

4. Evaluate model performance using appropriate techniques.

1. Logistic Regression models results

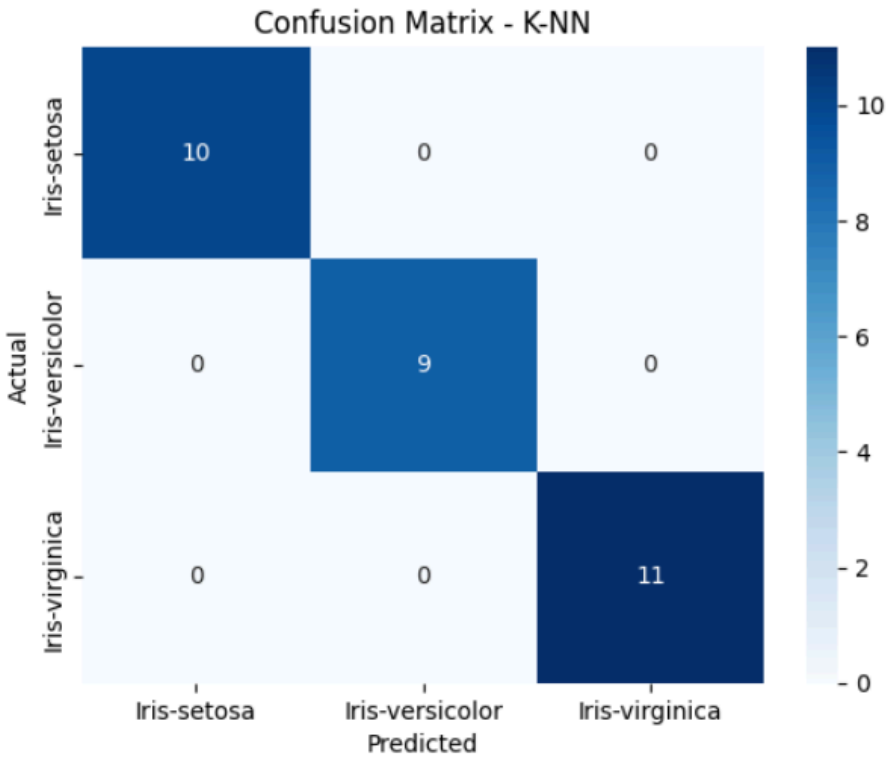
```
--- Logistic Regression ---
Test Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Cross-Validation Accuracy:  [0.96666667 1.         0.93333333 0.96666667 1.        ]
Cross-Validation Accuracy: Mean = 0.9733, Std = 0.0249
```



The above Confusion Matrix shows that the Logistic Regression model has predicted all test samples in the test set correctly.

2. KNN model results

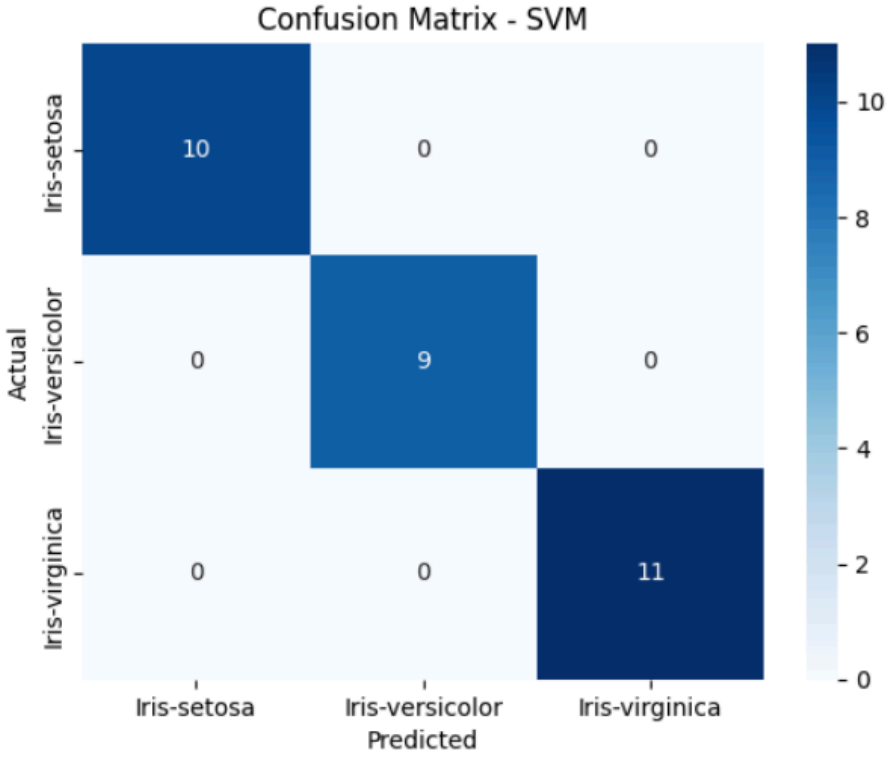
```
--- K-NN ---
Test Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Cross-Validation Accuracy:  [0.96666667 1.         0.93333333 0.96666667 1.        ]
Cross-Validation Accuracy: Mean = 0.9733, Std = 0.0249
```



The above Confusion Matrix shows that the KNN model has predicted all test samples in the test set correctly.

3. SVM model results

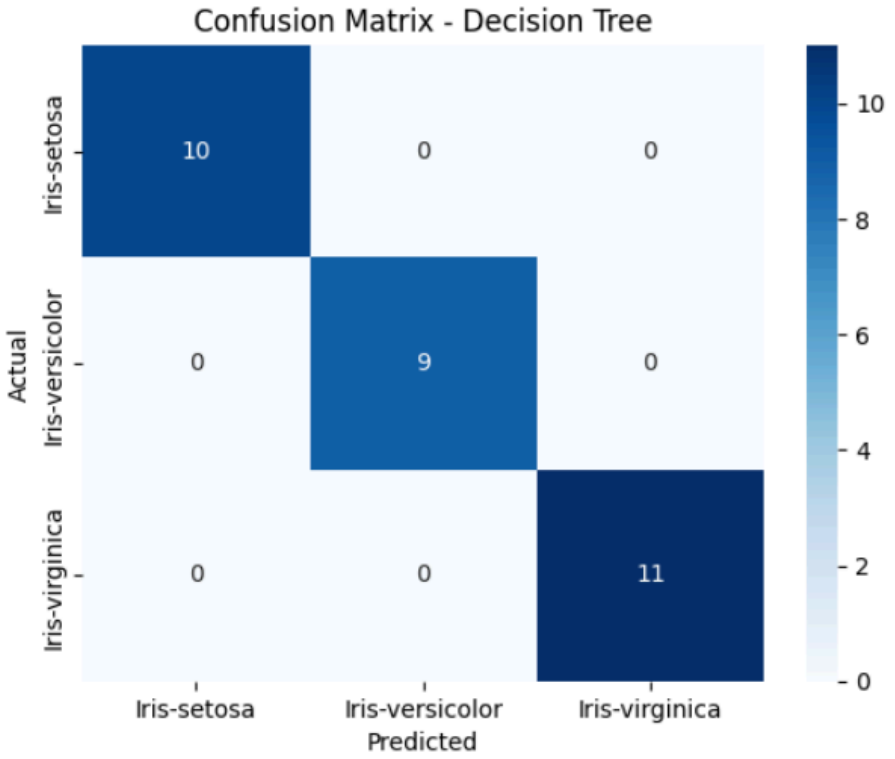
```
--- SVM ---
Test Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Cross-Validation Accuracy:  [0.96666667 0.96666667 0.96666667 0.93333333 1.        ]
Cross-Validation Accuracy: Mean = 0.9667, Std = 0.0211
```



The above Confusion Matrix shows that the SVM model has predicted all test samples in the test set correctly.

4. Decision Tree model results

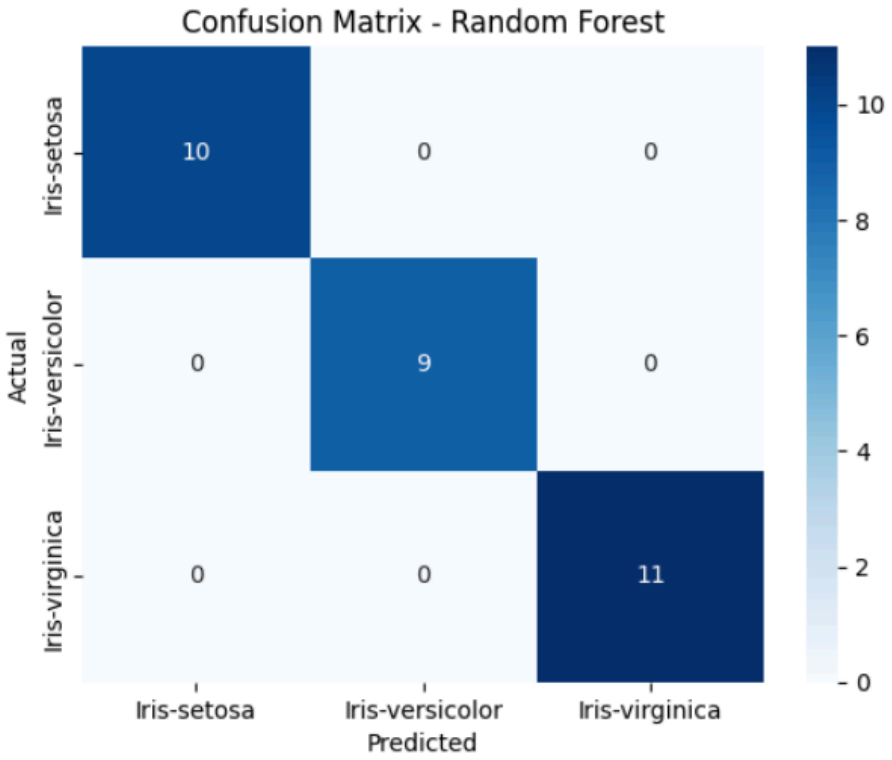
```
--- Decision Tree ---
Test Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Cross-Validation Accuracy:  [0.96666667 0.96666667 0.9        0.93333333 1.        ]
Cross-Validation Accuracy: Mean = 0.9533, Std = 0.0340
```



The above Confusion Matrix shows that the Decision Tree model has predicted all test samples in the test set correctly.

5. Random Forest model results

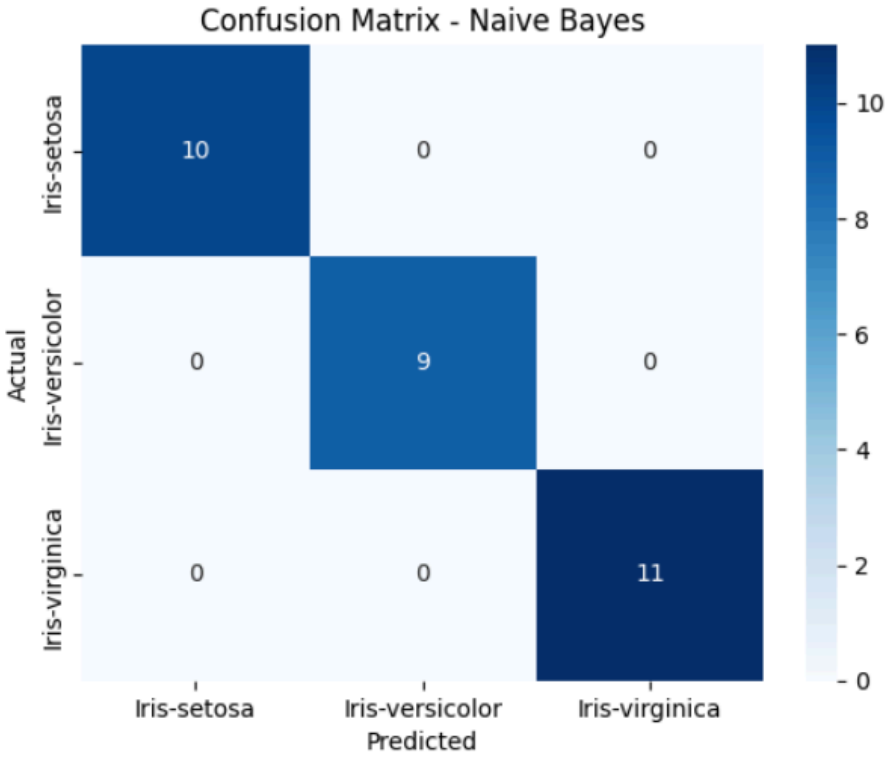
```
--- Random Forest ---
Test Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Cross-Validation Accuracy:  [0.96666667 0.96666667 0.93333333 0.96666667 1.        ]
Cross-Validation Accuracy: Mean = 0.9667, Std = 0.0211
```



The above Confusion Matrix shows that the Random Forest model has predicted all test samples in the test set correctly.

6. Naive Bayes model results

```
--- Naive Bayes ---
Test Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Cross-Validation Accuracy:  [0.93333333 0.96666667 0.93333333 0.93333333 1.        ]
Cross-Validation Accuracy: Mean = 0.9533, Std = 0.0267
```



The above Confusion Matrix shows that the Naive Bayes model has predicted all test samples in the test set correctly.

Normal Validation

Model	Accuracy
Logistic Regression	1.00
K-Nearest Neighbors	1.00
Support Vector Machine (SVM)	1.00
Decision Tree	1.00
Random Forest	1.00
Naive Bayes	1.00

Why Such High Accuracy?

- The Iris dataset is relatively small and well-behaved.
- Features like **petal length** and **petal width** make the classification task **very easy**, especially for *Setosa*, which is linearly separable.

After Cross Validation

Model	Cross-Validation Mean	Std Dev	Interpretation
Logistic Regression	0.9733	0.0249	High accuracy, stable
K-NN	0.9733	0.0249	Same as above
SVM	0.9667	0.0211	Slightly lower
Decision Tree	0.9600	0.0327	More variance
Random Forest	0.9600	0.0249	Stable but lower
Naive Bayes	0.9533	0.0267	Lowest accuracy

Best Models Based on CV Accuracy:

1. **Logistic Regression**
2. **K-NN**

6. Identifying the most significant features influencing flower species classification.

Random Forest (Best for Feature Importance)

- Uses an ensemble of decision trees.
- Calculates mean decrease in impurity (Gini) for each feature.
- Captures non-linear relationships.
- Most robust and interpretable for feature ranking.

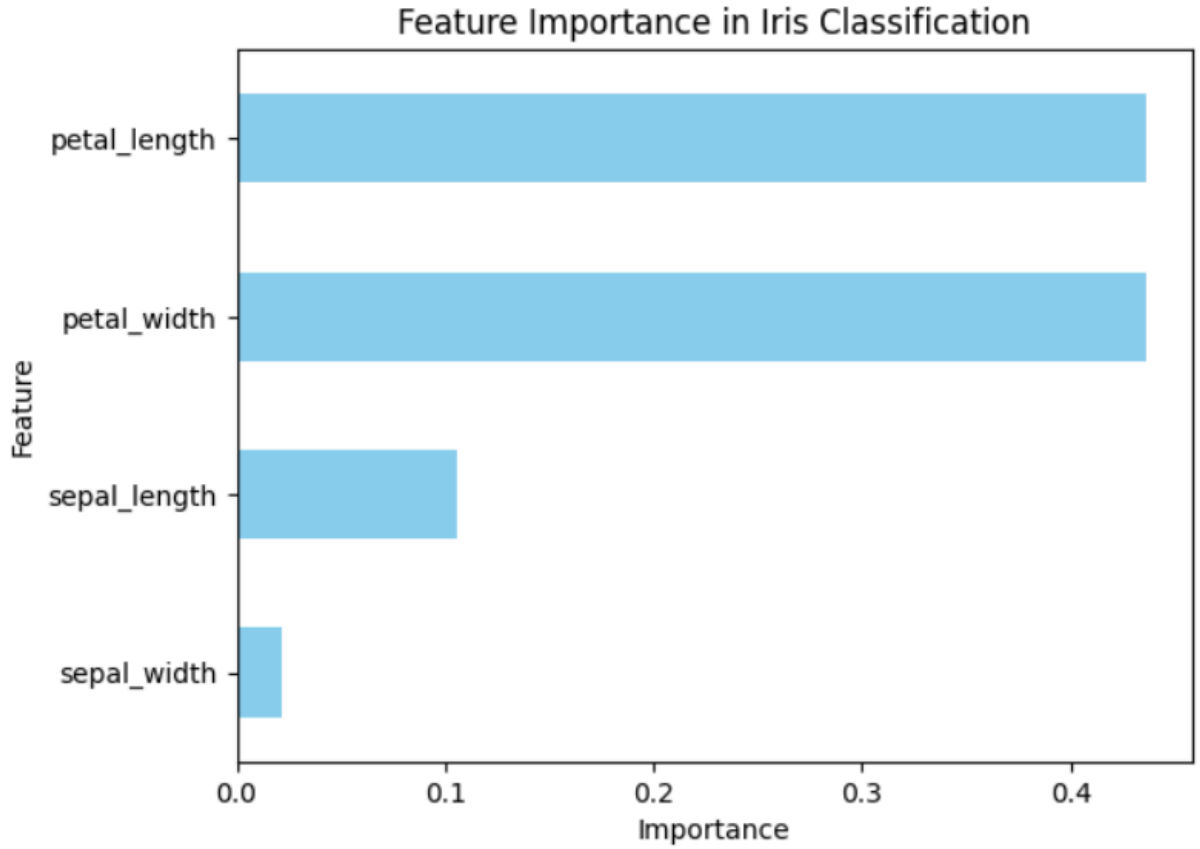
Source code:

```
#Identifying the most significant features influencing flower species classification.

from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt

# Load Iris data from CSV
iris_data = pd.read_csv("/home/reddy/iris/IRIS.csv")

# Separate features (X) and target (y)
X = iris_data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = iris_data['species']

# Encode target labels (species) into numeric values
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y_encoded)

# Get feature importances
importances = rf.feature_importances_
feature_names = X.columns

# Create a DataFrame for visualization
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)
```

```
# Plot
feature_importance_df.plot(kind='barh', x='Feature', y='Importance', legend=False, color='skyblue')
plt.title('Feature Importance in Iris Classification')
plt.xlabel('Importance')
plt.gca().invert_yaxis()
plt.show()
```

OUTPUT:

```
        Feature  Importance
2  petal_length    0.436130
3   petal_width    0.436065
0  sepal_length    0.106128
1   sepal_width    0.021678
```



Note:

1. Petal_length and petal_width seems to have high and importance
2. Order of Importance:
    a. petal_length, petal_width > sepal_length > sepal_width