Task

Iris Flower Classification

Iris flower has three species; setosa, versicolor, and virginica, which differs according to their measurements. Now assume that you have the measurements of the iris flowers according to their species, and here your task is to train a machine learning model that can learn from the measurements of the iris species and classify them Although the Scikit-learn library provides a dataset for iris flower classification.

Import Librarys

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

Load dataset

```
df = pd.read_csv("Iris.csv")
```

```
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Data Information

```
df.shape
```

```
(150, 5)
```

```
df.isnull().sum()
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
df.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
df.nunique()
```

```
Id               150
SepalLengthCm     35
SepalWidthCm      23
PetalLengthCm     43
PetalWidthCm      22
Species            3
dtype: int64
```

```
df['Species'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
dtype=object)
```

Drop 'Id' column as it's not needed

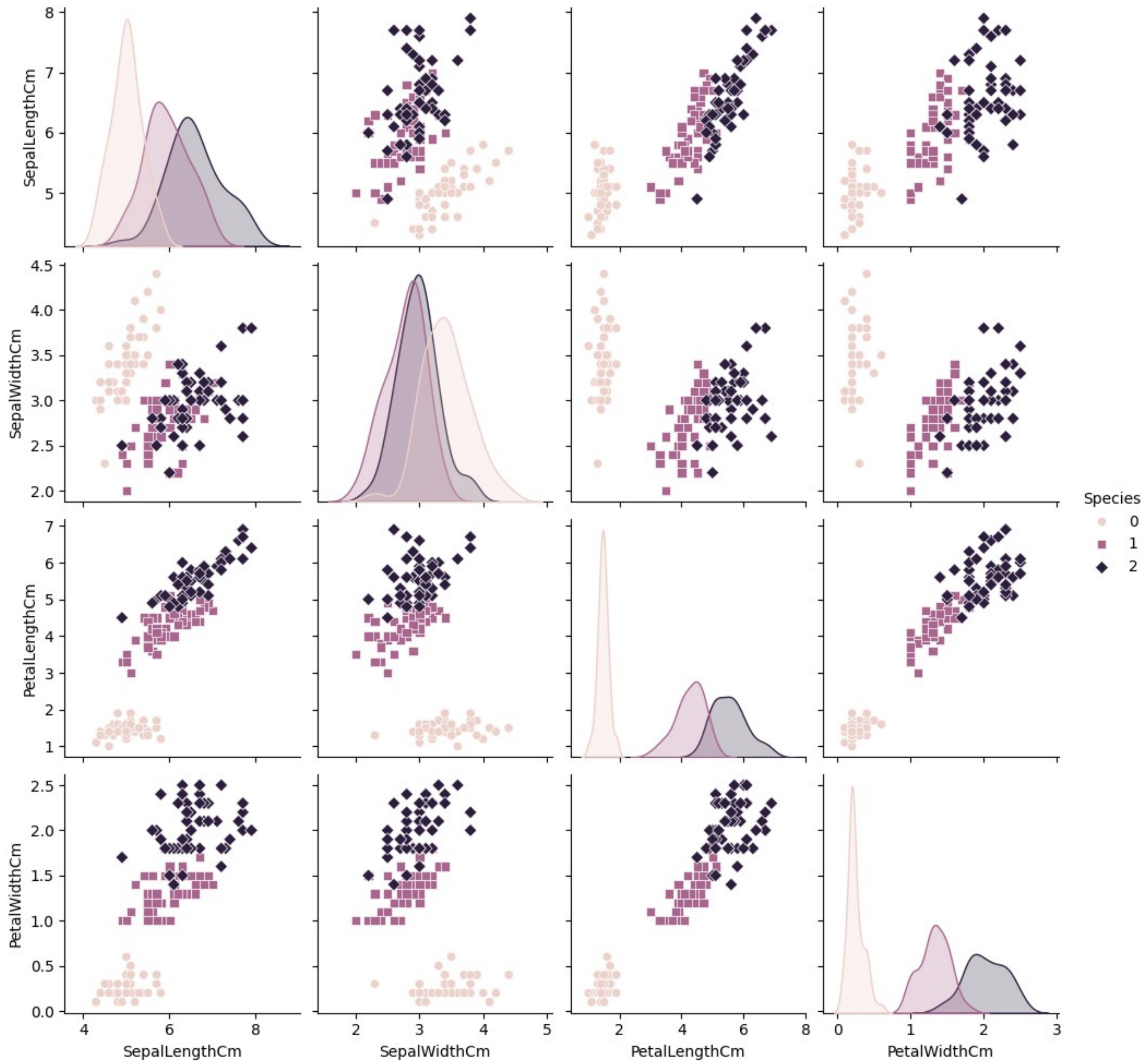
```
df.drop(columns=['Id'], inplace=True)

df.columns

Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
'PetalWidthCm',
       'Species'],
      dtype='object')
```

Encode species labels

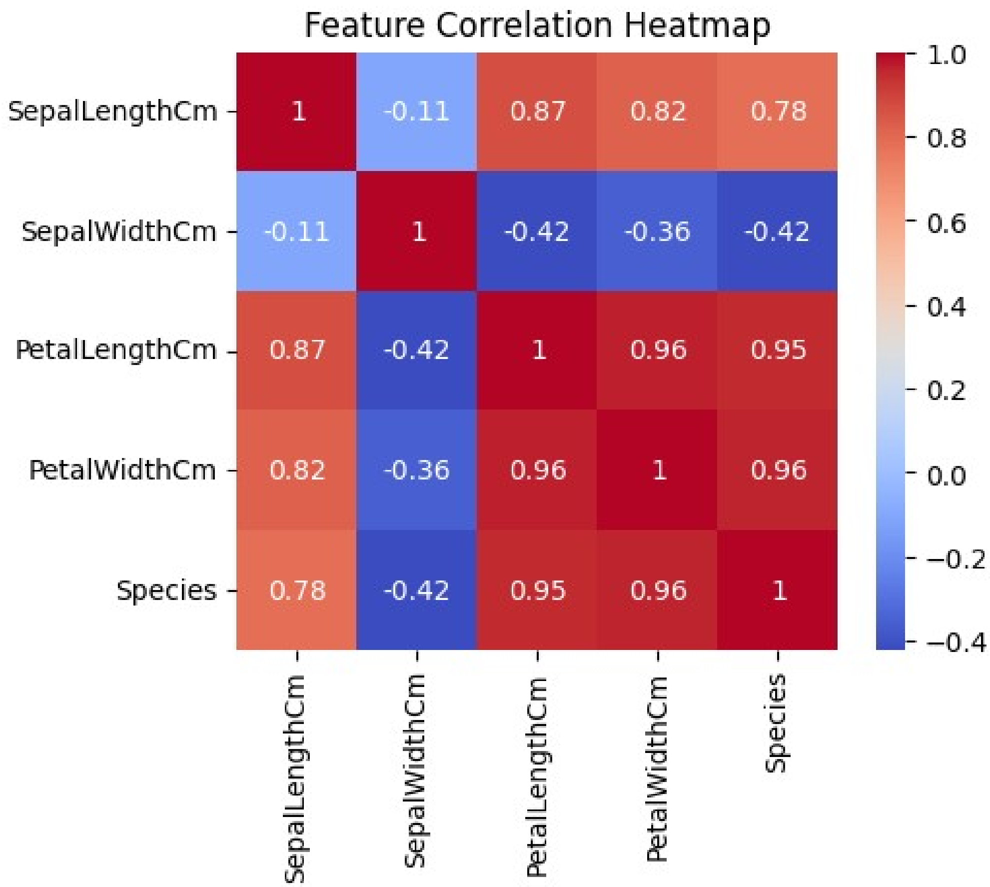
```
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

df.head()

   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0            5.1           3.5            1.4           0.2        0
1            4.9           3.0            1.4           0.2        0
2            4.7           3.2            1.3           0.2        0
3            4.6           3.1            1.5           0.2        0
4            5.0           3.6            1.4           0.2        0

df['Species'].unique()

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

--- Data Visualization ---

```
sns.pairplot(df, hue="Species", markers=["o", "s", "D"])
plt.show()
```




```
# Pairplot shows relationships between features for different species.

plt.figure(figsize=(5, 4))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```




```
# Heatmap shows correlations between sepal/petal measurements.
```

Split features and target

```
X = df.drop(columns=['Species'])
y = df['Species']
```

Split data into training and test sets (80-20 split)

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Train the model (Random Forest)

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

Make predictions

```
y_pred = model.predict(X_test)
```

```
y_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])
```

Evaluate the model

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.0
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

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Train Machine Learning Models (SVM, k-NN, Decision Tree)

```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

Train models

```
models = {
    "SVM": SVC(kernel='linear'),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "Decision Tree": DecisionTreeClassifier(random_state=42)
}
```

Evaluate models

```
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\n=== {name} Model Performance ===")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test,
y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
=== SVM Model Performance ===
Accuracy: 1.0
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

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

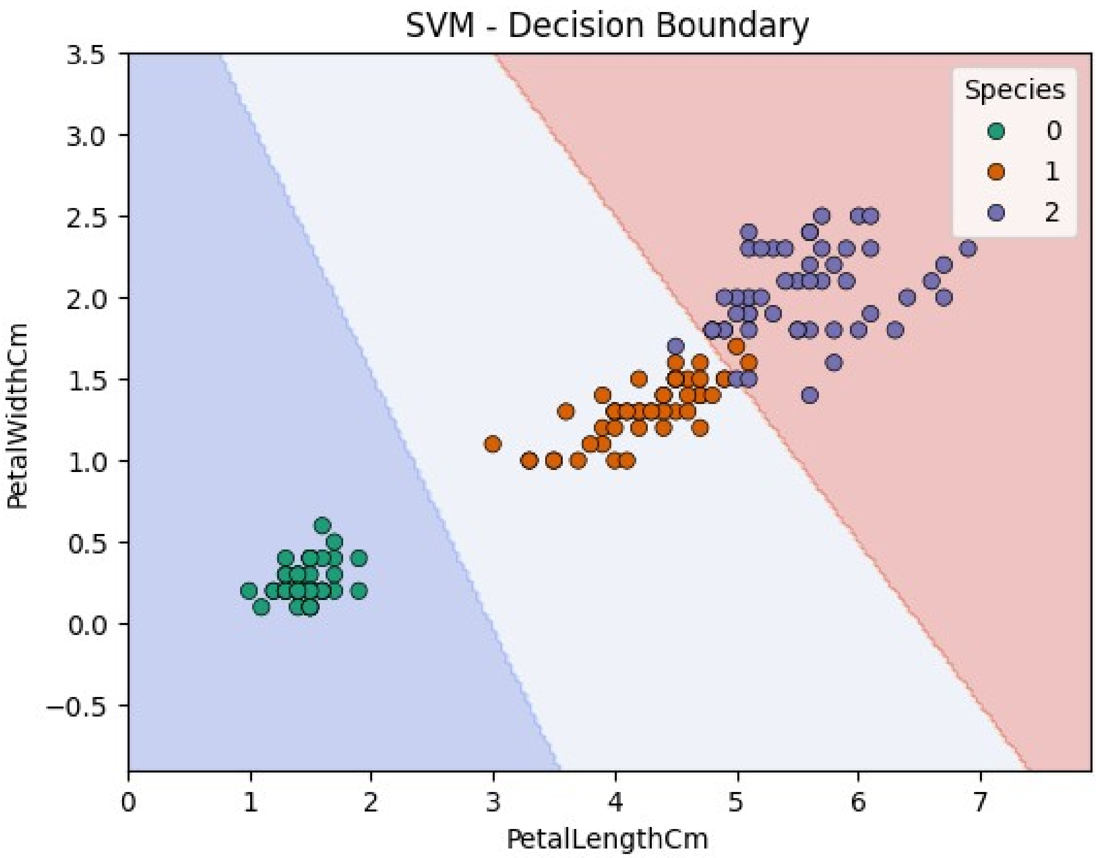
```
=== k-NN Model Performance ===
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           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

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

=== Decision Tree Model Performance ===
Accuracy: 1.0
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

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Select only two features for visualization

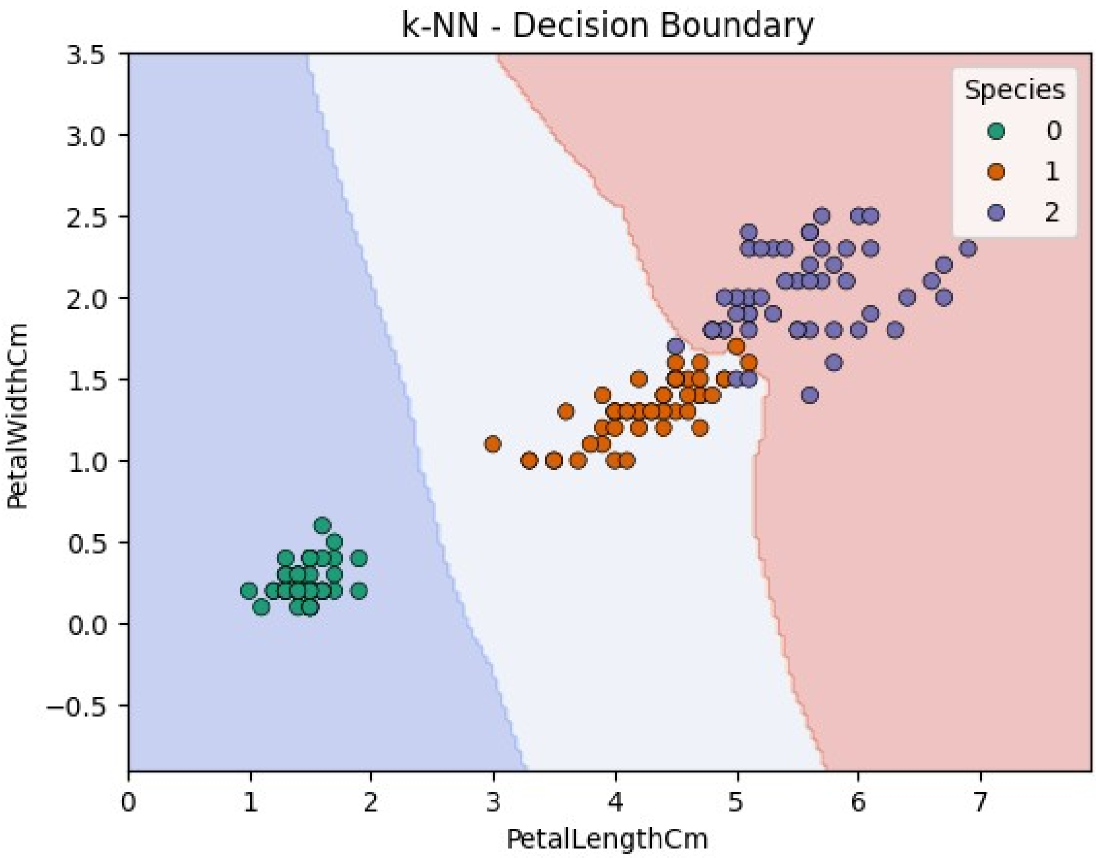
```
X = df[['PetalLengthCm', 'PetalWidthCm']]  # Use only PetalLength and PetalWidth
y = df["Species"]

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

# Define models and train them on the selected features
models = {
```

```python
    "SVM": SVC(kernel='linear'),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=10, 
random_state=42)
}

for name, model in models.items():
    model.fit(X_train, y_train)  # Train using only the two selected 
features

def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
    y_min, y_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), 
np.linspace(y_min, y_max, 200))

    # Predict labels for each point in mesh grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=y, 
palette="Dark2", edgecolor="k")

    plt.xlabel(X.columns[0])
    plt.ylabel(X.columns[1])
    plt.title(title)
    plt.show()

# Plot decision boundaries for each model
for name, model in models.items():
    plot_decision_boundary(model, X, y, f"{name} - Decision Boundary")
```

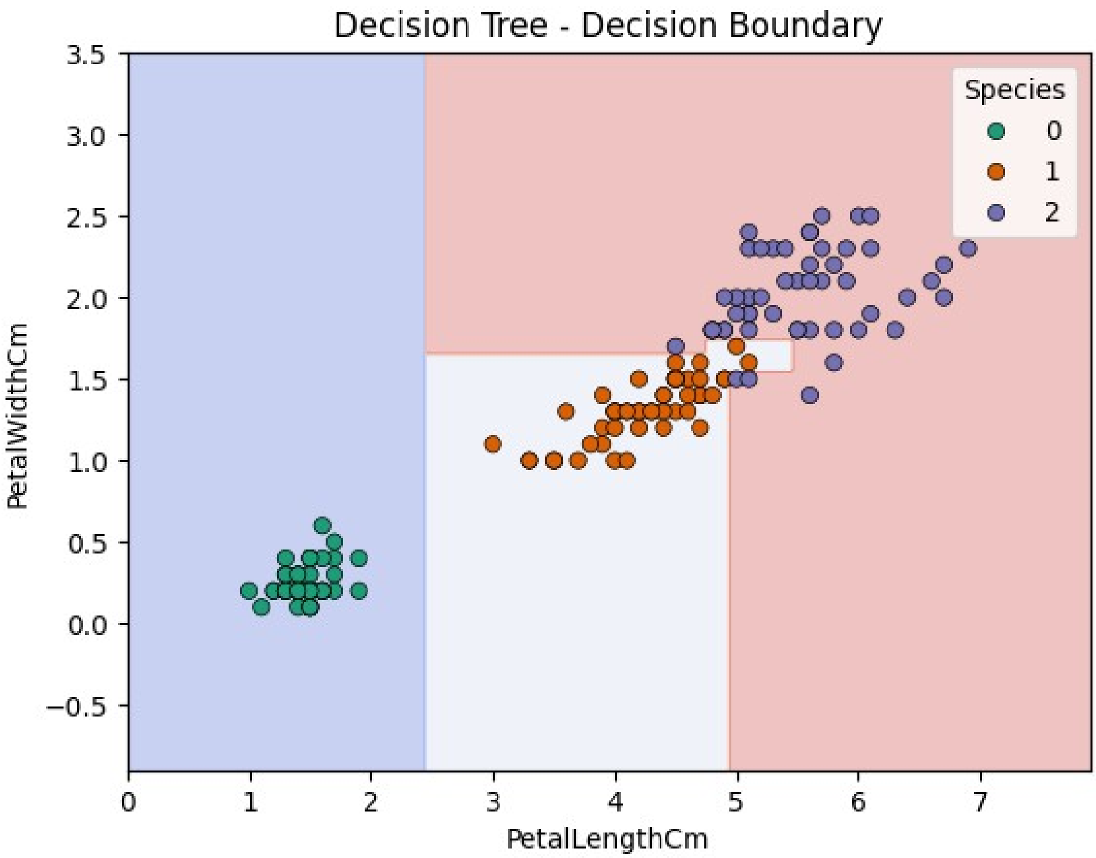
```
C:\Users\Admin\AppData\Local\Programs\Python\Python313\Lib\site-
packages\sklearn\utils\validation.py:2739: UserWarning: X does not 
have valid feature names, but SVC was fitted with feature names
  warnings.warn(
```



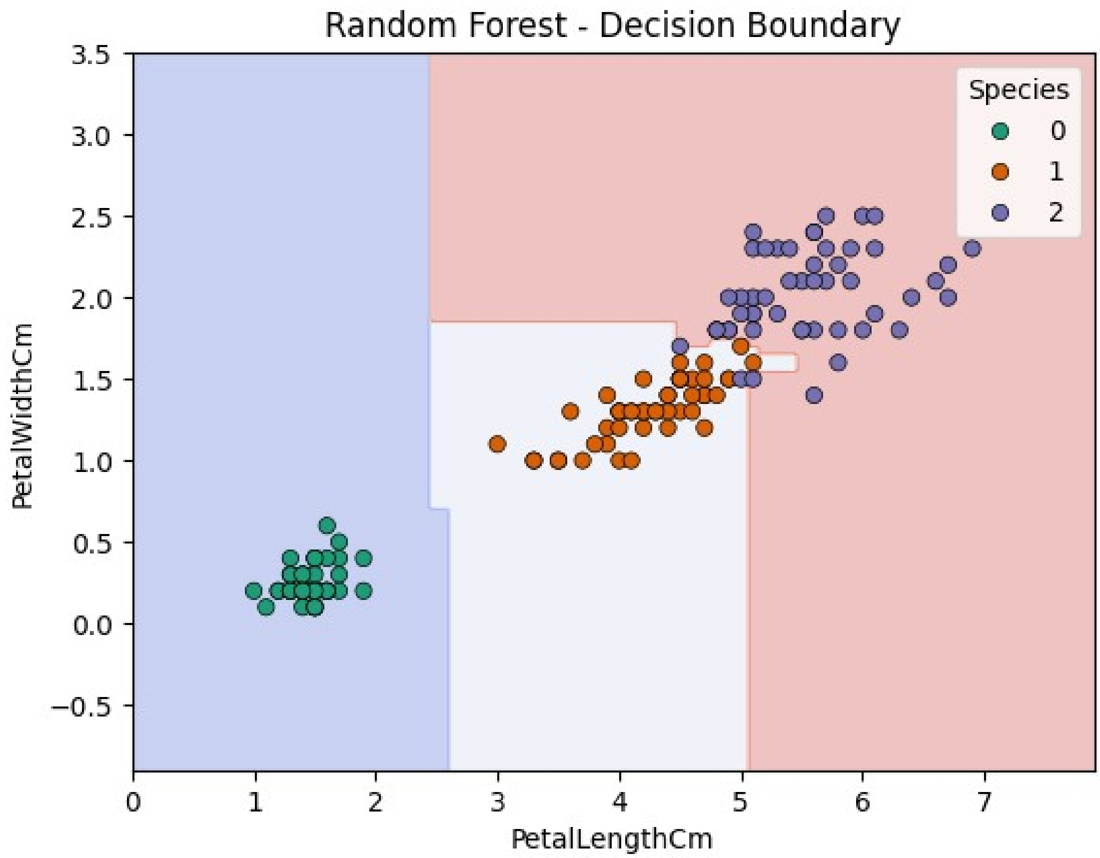

```
C:\Users\Admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```




```
C:\Users\Admin\AppData\Local\Programs\Python\Python313\Lib\site-
packages\sklearn\utils\validation.py:2739: UserWarning: X does not
have valid feature names, but DecisionTreeClassifier was fitted with
feature names
  warnings.warn(
```




```
C:\Users\Admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```



Conclusion

All four models (SVM, k-NN, Decision Tree, and Random Forest) achieved 100% accuracy on the test set. This suggests that the Iris dataset is well-separated and easily classified using these models.

SVM performed well, likely due to the clear decision boundaries between species. k-NN worked perfectly, benefiting from the small dataset size. Decision Tree & Random Forest both achieved perfect classification, showing that tree-based methods are effective.