Author: Abdel Halim Mansour

Task 6: Prediction Using Decision Tree Algorithm

Data Science & Business Analytics - TSFGRIP - September 2023

GitHub: https://github.com/HalimMansour/Data-Science-Business-Analytics-The-Sparks-Foundation

LinkedIn: https://www.linkedin.com/in/abdel-halim-mansour-87a255235/

Step 1: Import the necessary libraries and load the data

In [103…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image
import pydotplus

#scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import tree




# Load the Dataset
iris = pd.read_csv('D:/Education/data/Projects/Python/2- Prediction Using Decision Tree
```

Step 2: Overview of the dataset

In [104…
```
print(iris.shape)
```

```
(150, 6)
```

In [105…
```
# Display the first few rows of the dataset
iris.head()
```

Out[105]:

	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

In [106…
```
# Check basic information about the dataset
iris.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
```

In [107…
```
# Get basic statistics of numerical columns
iris.describe()
```

Out[107]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [108…
```
# Check for missing values
iris.isna().sum()
```

Out[108]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

Step 3: Visualization

Univariate Analysis

- **Histograms**
- **Box Plots**

Bivariate Analysis

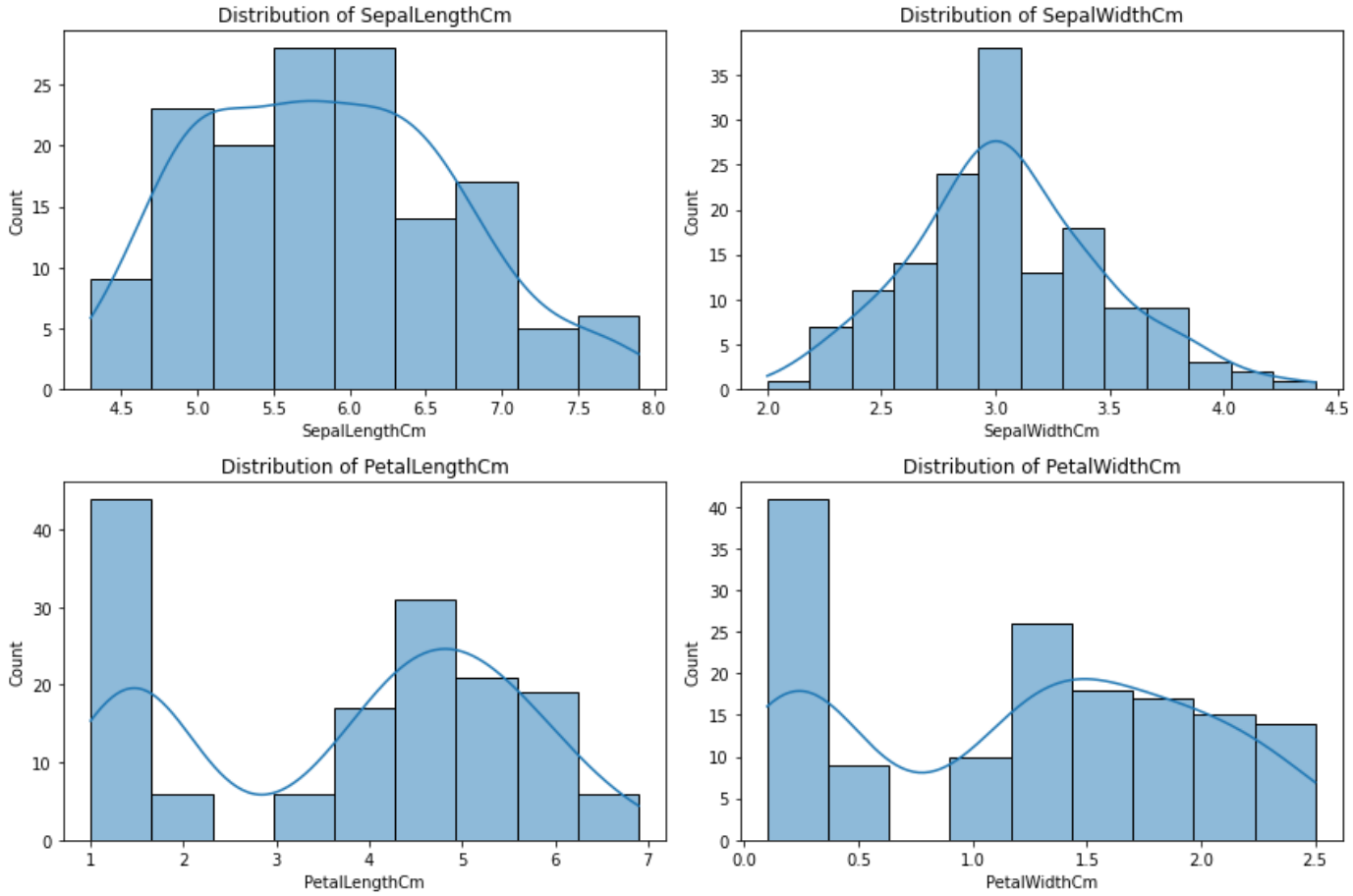
- **Pair Plot**
- **Correlation Heatmap**

1- Univariate Analysis

Histograms

In [109…

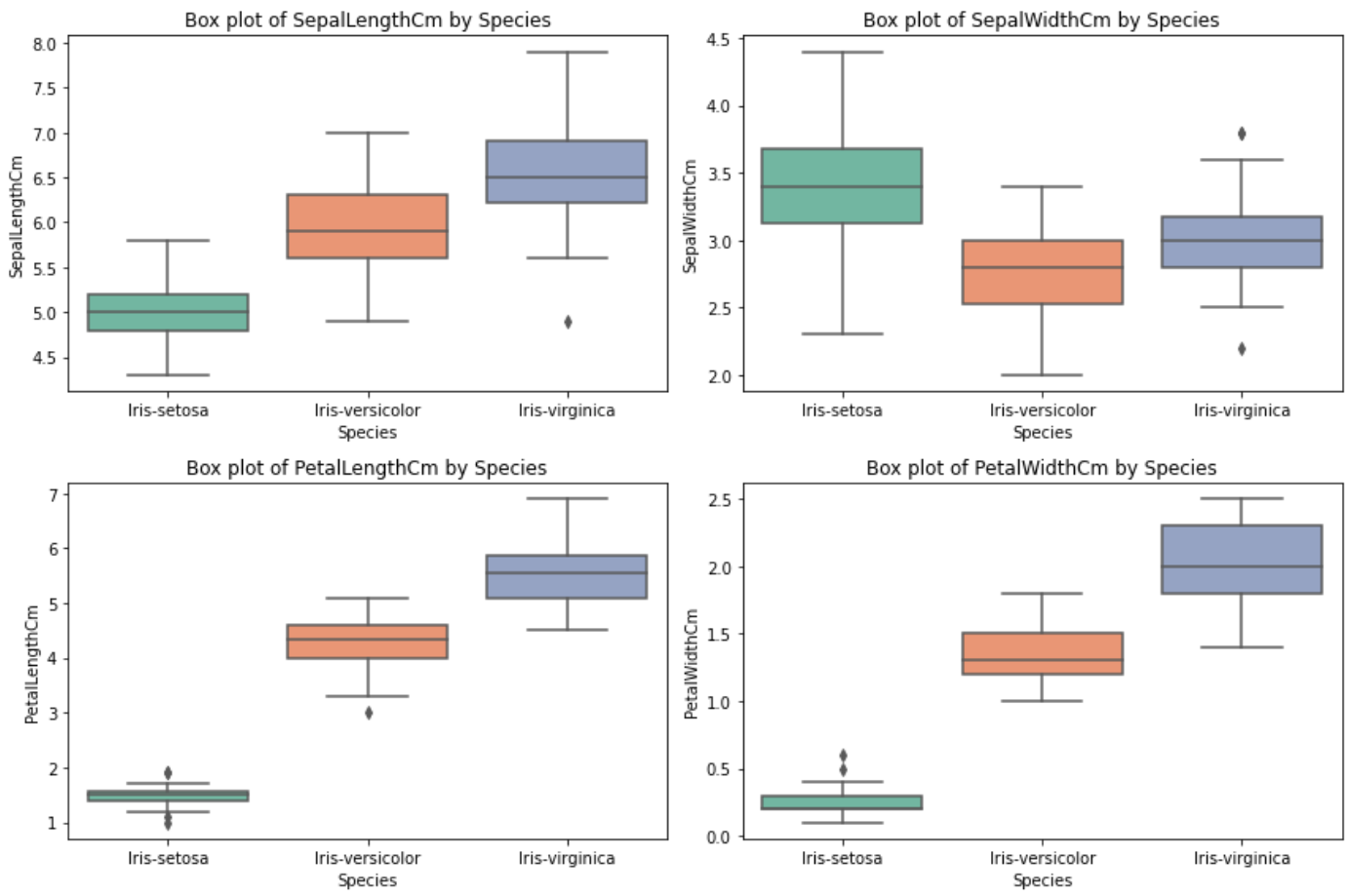
```
# Histograms and Distributions

# Create histograms for each feature (Column)
plt.figure(figsize=(12, 8))
for feature in iris.columns[1:-1]:
    plt.subplot(2, 2, iris.columns.get_loc(feature))
    sns.histplot(iris[feature], kde=True)
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()
```



Box Plots

In [110…

```
# Create box plots to identify outliers
plt.figure(figsize=(12, 8))
for feature in iris.columns[1:-1]:
    plt.subplot(2, 2, iris.columns.get_loc(feature))
    sns.boxplot(x='Species', y=feature, data=iris , palette='Set2')
    plt.title(f'Box plot of {feature} by Species')
plt.tight_layout()
plt.show()
```

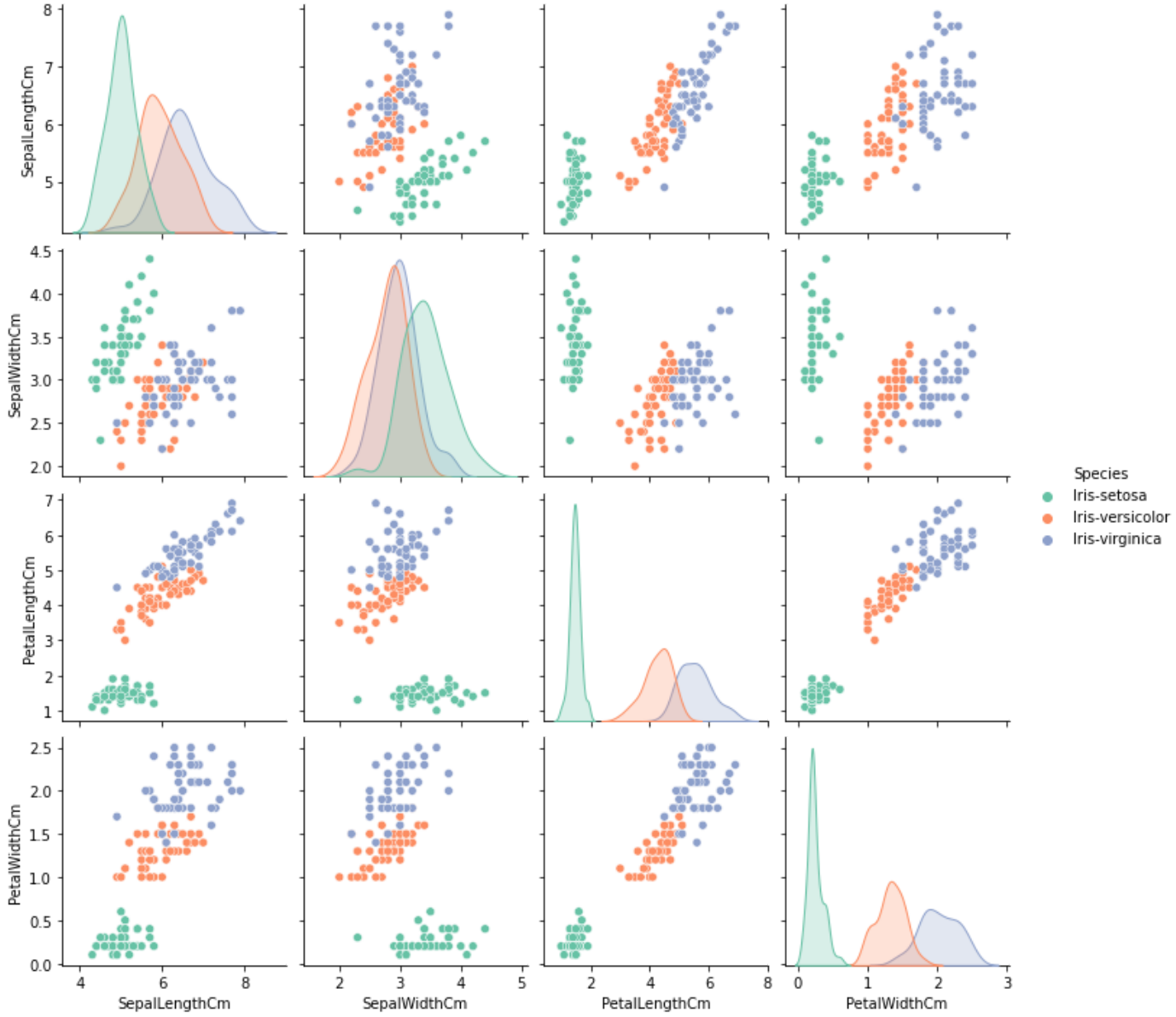


2- Bivariate Analysis

****Pair Plot****

In [111…

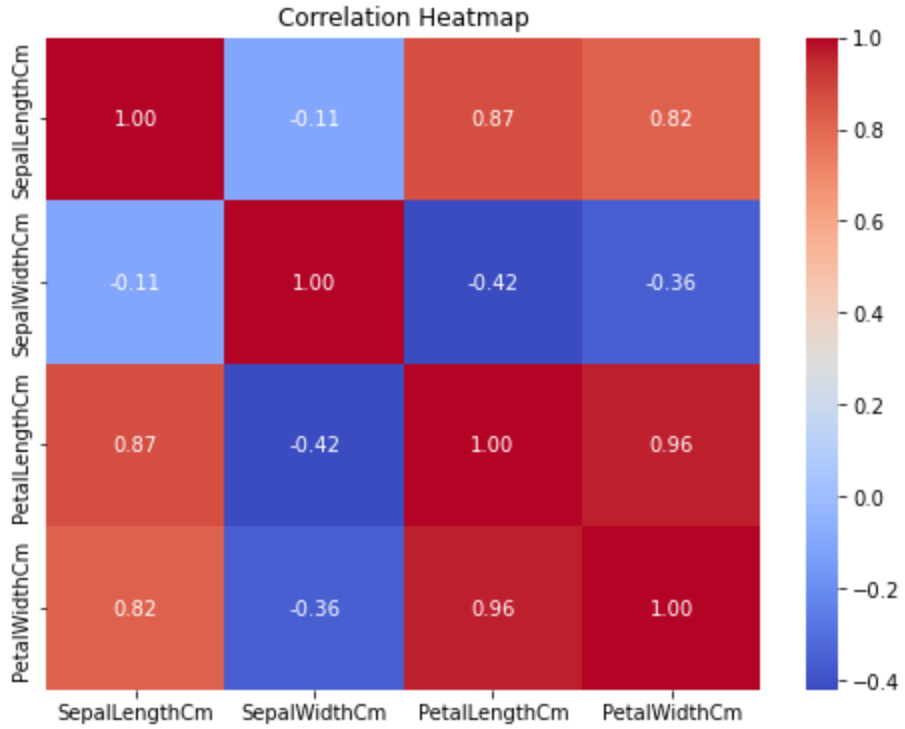
```
# Create a pair plot to visualize relationships between features
iris_no_id = iris.drop('Id', axis=1)  # Replace 'id' with the actual column name
sns.pairplot(iris_no_id, hue='Species', palette='Set2')
plt.show()
```



Correlation Heatmap

In [112…

```
# Calculate and visualize the correlation between features
correlation_matrix = iris_no_id.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



In [113…
```
iris_no_id.head()
```

Out[113]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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

Step 4: Building, Training and Evaluating Model

Note: We will select the **Entropy** criterion as the splitting criterion

Entropy(S) = - p1 Log2(p1) - p2 Log2(p2) - etc

In [275…
```
# <mark>**Gini(p) = 1 - Σ(p_i^2)**</mark>
```

In [276…
```
#Seperatingthe Target variable
#X = iris_no_id.values[:, :4]
#Y = iris_no_id.values[:,-1]

#SplitingDataset into Test and Train
# 70% for training
# 30% for testing

#X_train, X_test, y_train, y_test= train_test_split( X, Y, test_size= 0.2, random_state=

#Function to perform training with Entropy
#clf_entropy= DecisionTreeClassifier(criterion = "gini", random_state= 10)
#clf_entropy.fit(X_train, y_train)
```

In [290…
```
# Assuming X contains your feature data and Y contains your labels
```

```
# Replace this with your actual data
X = iris_no_id.values[:, :4]
Y = iris_no_id.values[:, -1]

# Identify the unique classes in your dataset
unique_classes = np.unique(Y)

# Initialize empty lists to store train and test data
X_train, X_test, y_train, y_test = [], [], [], []

# Loop through each unique class and create a custom train-test split
for class_label in unique_classes:
    # Find the indices of data points belonging to the current class
    class_indices = np.where(Y == class_label)[0]

    # Calculate the number of samples for training (80%) and testing (20%)
    num_samples = len(class_indices)
    num_train_samples = int(0.8 * num_samples)
    num_test_samples = num_samples - num_train_samples

    # Randomly shuffle the indices for this class
    np.random.shuffle(class_indices)

    # Split the indices into training and testing indices
    train_indices = class_indices[:num_train_samples]
    test_indices = class_indices[num_train_samples:]

    # Use the training and testing indices to select data points for this class
    X_train.extend(X[train_indices])
    X_test.extend(X[test_indices])
    y_train.extend(Y[train_indices])
    y_test.extend(Y[test_indices])

# Convert lists to NumPy arrays
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

# Now, you can proceed with training your classifier as before
clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=10)
clf_entropy.fit(X_train, y_train)
```

Out[290]:
```
DecisionTreeClassifier(criterion='entropy', random_state=10)
```

In [291…
```
#FunctiontomakePredictions
y_pred_en= clf_entropy.predict(X_test)
y_pred_en
```

Out[291]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica'], dtype='<U15')
```

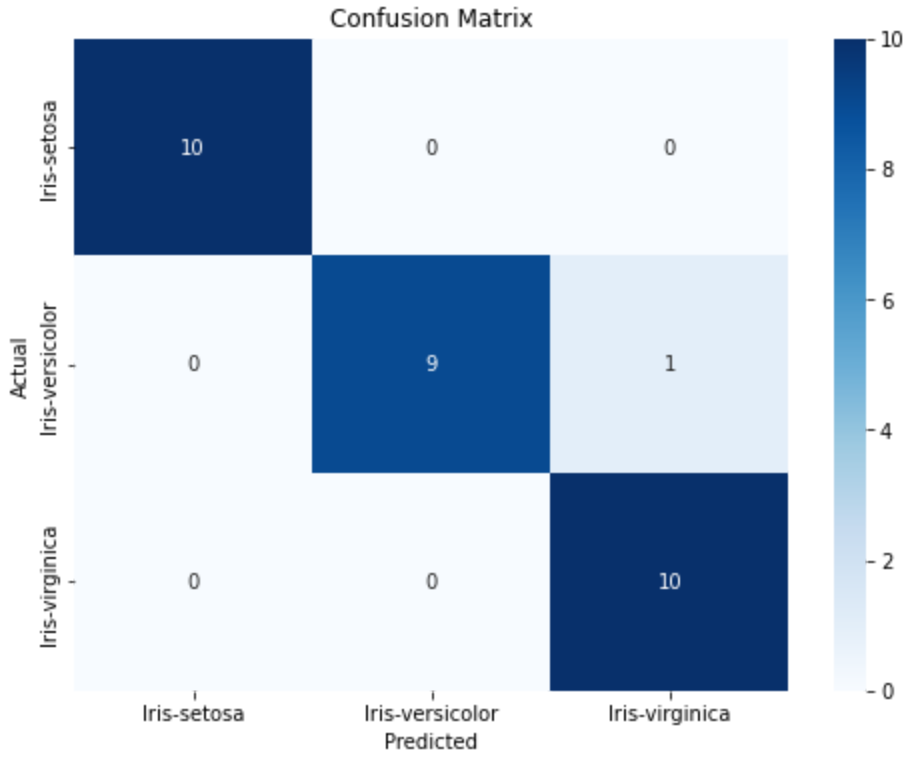
In [292…
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_en)
print(f'Accuracy: {accuracy:.5f}')
```

```
Accuracy: 0.96667
```

In [293…
```
# Generate a classification report
```

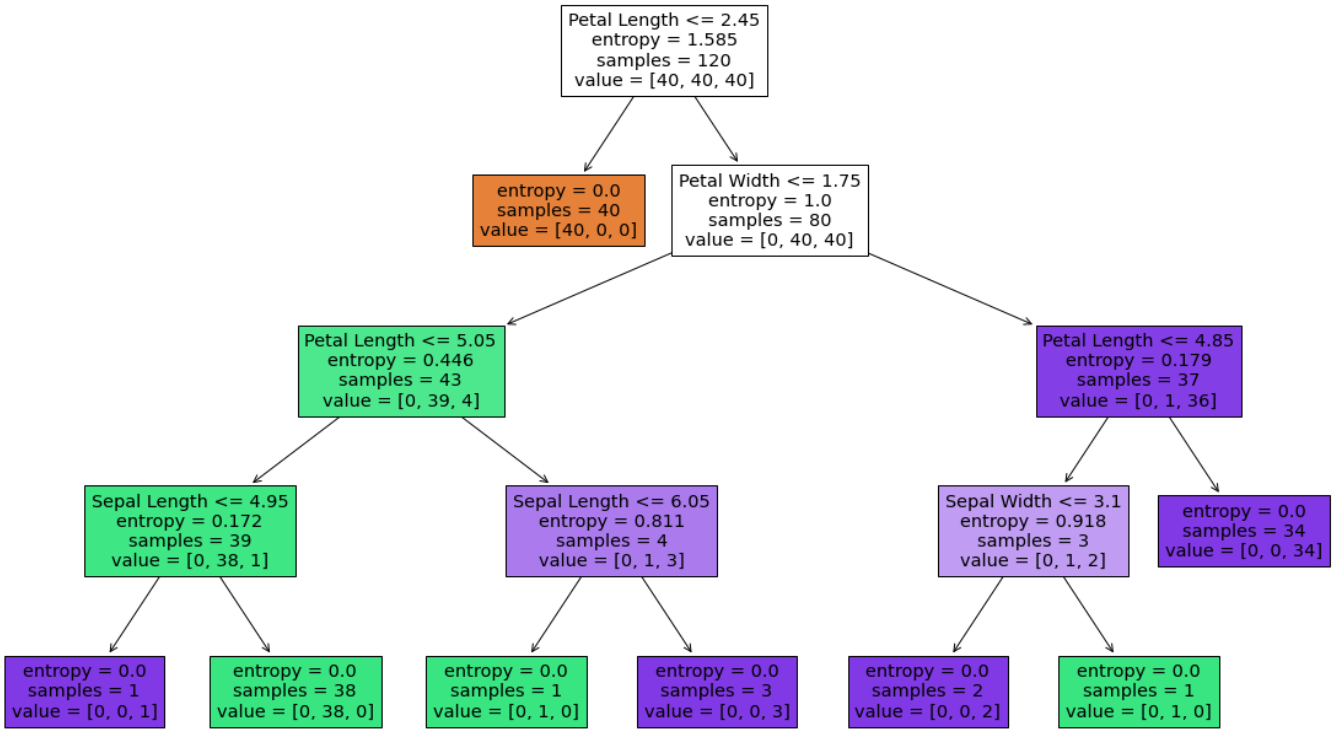
```
class_report = classification_report(y_test, y_pred_en)
print('Classification Report:\n', class_report)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_en)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
            yticklabels=['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
                  precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        10
 Iris-versicolor       1.00      0.90      0.95        10
  Iris-virginica       0.91      1.00      0.95        10

        accuracy                           0.97        30
       macro avg       0.97      0.97      0.97        30
    weighted avg       0.97      0.97      0.97        30
```



In [281…

```
plt.figure(figsize=(20, 12))
tree.plot_tree(clf_entropy, filled=True, feature_names=['Sepal Length', 'Sepal Width', '
plt.show()
```


Petal Length <= 2.45
entropy = 1.585
samples = 120
value = [40, 40, 40]
entropy = 0.0
samples = 40
value = [40, 0, 0]
Petal Width <= 1.75
entropy = 1.0
samples = 80
value = [0, 40, 40]
Petal Length <= 5.05
entropy = 0.446
samples = 43
value = [0, 39, 4]
Petal Length <= 4.85
entropy = 0.179
samples = 37
value = [0, 1, 36]
Sepal Length <= 4.95
entropy = 0.172
samples = 39
value = [0, 38, 1]
Sepal Length <= 6.05
entropy = 0.811
samples = 4
value = [0, 1, 3]
Sepal Width <= 3.1
entropy = 0.918
samples = 3
value = [0, 1, 2]
entropy = 0.0
samples = 34
value = [0, 0, 34]
entropy = 0.0
samples = 1
value = [0, 0, 1]
entropy = 0.0
samples = 38
value = [0, 38, 0]
entropy = 0.0
samples = 1
value = [0, 1, 0]
entropy = 0.0
samples = 3
value = [0, 0, 3]
entropy = 0.0
samples = 2
value = [0, 0, 2]
entropy = 0.0
samples = 1
value = [0, 1, 0]

In [ ]:

In [ ]: