In [90]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data=pd.read_csv('Iris.csv')
data
```

Out[90]:

	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [91]:
```
data_maping = {'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica':2 }
data['Species']=data['Species'].map(data_maping)
```

In [92]:
```
data
```

Out[92]:

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

4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

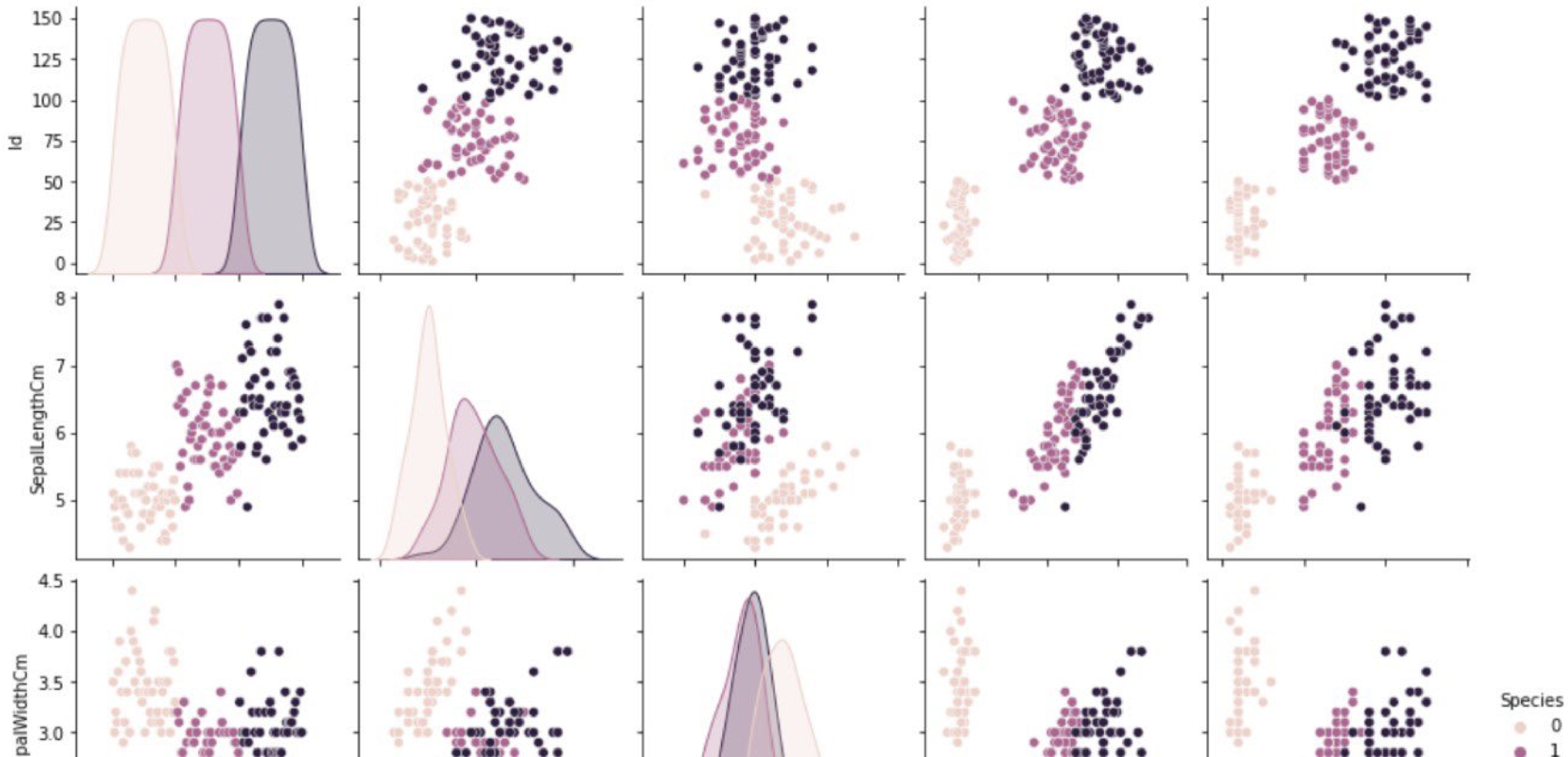
150 rows × 6 columns

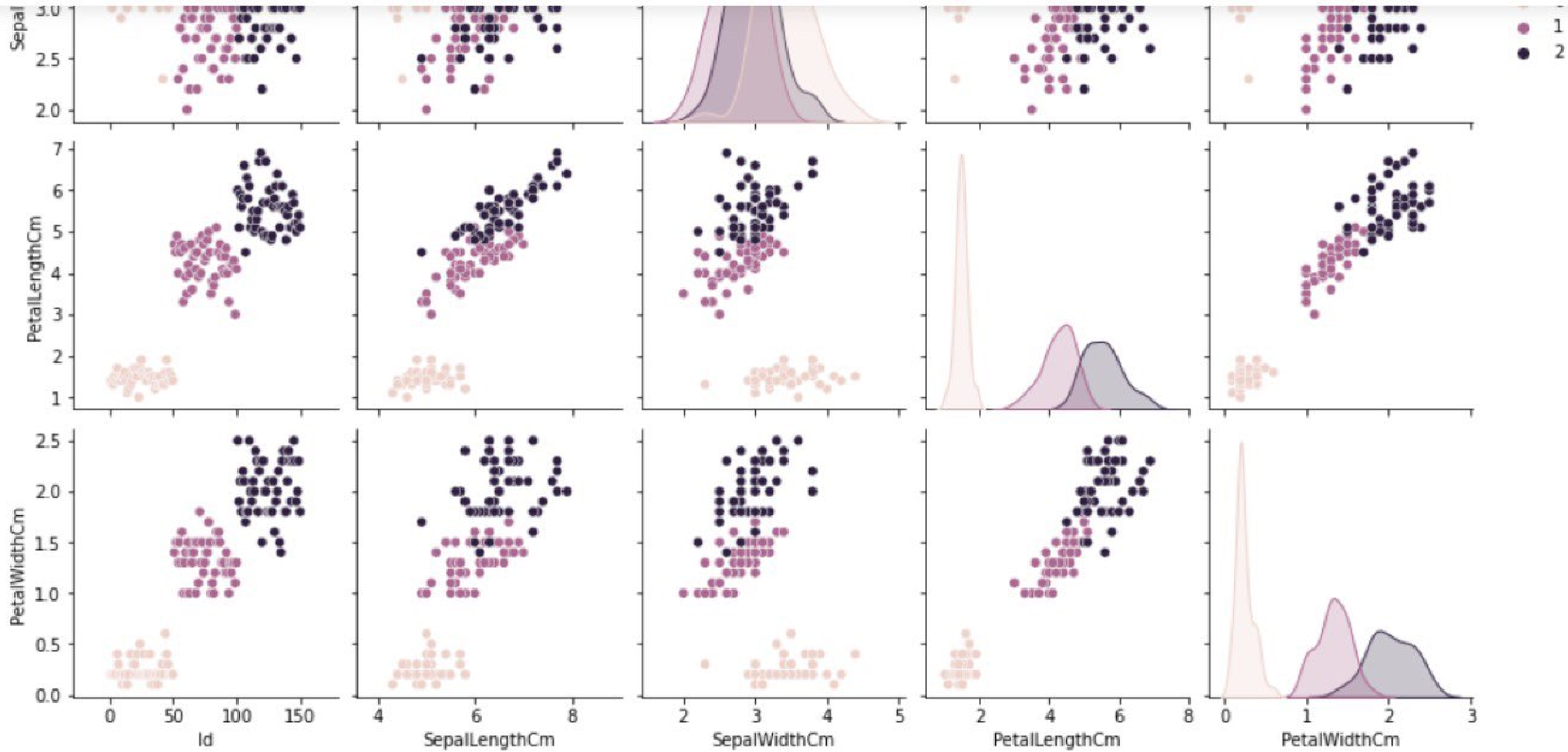
In [93]:

```
print(data.describe())


correlation_matrix = data.corr()
print(correlation_matrix)

sns.pairplot(data, hue='Species')
plt.show()
```

```
               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

          Species
count  150.000000
mean     1.000000
std      0.819232
min      0.000000
25%      0.000000
50%      1.000000
75%      2.000000
max      2.000000
```

```
max       2.000000
                     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  \
Id             1.000000       0.716676     -0.397729       0.882747
SepalLengthCm  0.716676       1.000000     -0.109369       0.871754
SepalWidthCm  -0.397729      -0.109369      1.000000      -0.420516
PetalLengthCm  0.882747       0.871754     -0.420516       1.000000
PetalWidthCm   0.899759       0.817954     -0.356544       0.962757
Species        0.942830       0.782561     -0.419446       0.949043

               PetalWidthCm   Species
Id                 0.899759  0.942830
SepalLengthCm      0.817954  0.782561
SepalWidthCm      -0.356544 -0.419446
PetalLengthCm      0.962757  0.949043
PetalWidthCm       1.000000  0.956464
Species            0.956464  1.000000
```





In [94]:
```
class DecisionTreeClassifier:
    def __init__(self, max_depth= None):
        self.max_depth= max_depth
        self.tree=None

    def fit(self, X, y):
        self.tree= self.build_tree(X,y)


    def build_tree(self, X, y, depth=0):
        classes= list(set(y))

        if self.max_depth is not None and depth >= self.max_depth or len(classes)== 1:
           return {'class': classes[0]}

        num_features = len(X[0])
```

```python
        best_gini = float('inf')
        best_feature = None
        best_threshold = None

        # En iyi bölme kriterini seçme
        for feature in range(num_features):
            thresholds = list(set(X[i][feature] for i in range(len(X))))
            for threshold in thresholds:
                left_indices = [i for i in range(len(X)) if X[i][feature] <= threshold]
                right_indices = [i for i in range(len(X)) if X[i][feature] > threshold]
                gini = self.gini_impurity(y, left_indices, right_indices)

                if gini < best_gini:
                    best_gini = gini
                    best_feature = feature
                    best_threshold = threshold

        # Düğümü oluşturma
        node = {
            'feature': best_feature,
            'threshold': best_threshold,
            'left': None,
            'right': None
        }

        # Sol ve sağ alt ağaçları oluşturma
        left_indices = [i for i in range(len(X)) if X[i][best_feature] <= best_threshold]
        right_indices = [i for i in range(len(X)) if X[i][best_feature] > best_threshold]
        node['left'] = self.build_tree([X[i] for i in left_indices], [y[i] for i in left_indices], depth + 1)
        node['right'] = self.build_tree([X[i] for i in right_indices], [y[i] for i in right_indices], depth + 1)

        return node


    def gini_impurity(self, y, left_indices, right_indices):
        left_classes = [y[i] for i in left_indices]
        right_classes = [y[i] for i in right_indices]

        left_counts = {}
        right_counts = {}

        for class_label in left_classes:
            if class_label in left_counts:
```

```
                left_counts[class_label] += 1
            else:
                left_counts[class_label] = 1

        for class_label in right_classes:
            if class_label in right_counts:
                right_counts[class_label] += 1
            else:
                right_counts[class_label] = 1

        left_probs = [count / len(left_classes) for count in left_counts.values()]
        right_probs = [count / len(right_classes) for count in right_counts.values()]

        left_gini = 1 - sum([prob**2 for prob in left_probs])
        right_gini = 1 - sum([prob**2 for prob in right_probs])

        gini = (sum(left_counts.values()) / len(y)) * left_gini + (sum(right_counts.values()) / len(y)) * right_gini
        return gini

    def traverse_tree(self, sample, node):
        if 'class' in node:
            return node['class']

        if sample[node['feature']] <= node['threshold']:
            return self.traverse_tree(sample, node['left'])
        else:
            return self.traverse_tree(sample, node['right'])

    def predict(self, X):
        predictions = []
        for sample in X:
            prediction = self.traverse_tree(sample, self.tree)
            predictions.append(prediction)
        return predictions

    def predict_proba(self, X):
        proba = []
        for sample in X:
            probabilities = self.traverse_proba(sample, self.tree)
            proba.append(probabilities)
        return np.array(proba)

    def traverse_proba(self, sample, node):
```

```python
        if 'class' in node:
            return {c: 1.0 if c == node['class'] else 0.0 for c in np.unique(y)}

        if sample[node['feature']] <= node['threshold']:
            return self.traverse_proba(sample, node['left'])
        else:
            return self.traverse_proba(sample, node['right'])
```

```python
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# datayı train ve test olarak bölme
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)

# modeli kurup eğitme
dt_classifier = DecisionTreeClassifier(max_depth=2)
dt_classifier.fit(X_train, y_train)

# test set üzerinden tahmin
y_pred = dt_classifier.predict(X_test)

# accuracy hesaplama
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

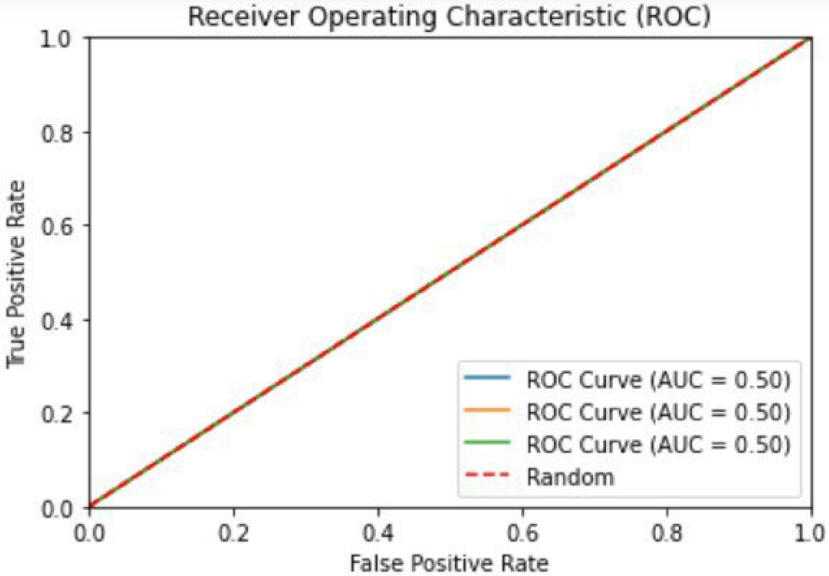
```python
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np


def convert_to_binary(y, target_class):
    binary_labels = np.zeros_like(y)
    binary_labels[y == target_class] = 1
    return binary_labels

# Confusion matrix
```

```
# Confusion matrix
train_cm = confusion_matrix(y_train, dt_classifier.predict(X_train))
test_cm = confusion_matrix(y_test, y_pred)

print("Training Confusion Matrix:")
print(train_cm)
print("\nTesting Confusion Matrix:")
print(test_cm)

# Classification report
train_report = classification_report(y_train, dt_classifier.predict(X_train))
test_report = classification_report(y_test, y_pred)

print("\nTraining Classification Report:")
print(train_report)
print("\nTesting Classification Report:")
print(test_report)

# Accuracy
train_accuracy = accuracy_score(y_train, dt_classifier.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred)

print("\nTraining Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)

# Precision
train_precision = train_cm[1, 1] / np.sum(train_cm[:, 1])
test_precision = test_cm[1, 1] / np.sum(test_cm[:, 1])

print("Training Precision:", train_precision)
print("Testing Precision:", test_precision)

# Recall
train_recall = train_cm[1, 1] / np.sum(train_cm[1, :])
test_recall = test_cm[1, 1] / np.sum(test_cm[1, :])

print("Training Recall:", train_recall)
print("Testing Recall:", test_recall)

# F1-Score
train_f1_score = 2 * (train_precision * train_recall) / (train_precision + train_recall)
test_f1_score = 2 * (test_precision * test_recall) / (test_precision + test_recall)
```

```python
print("Training F1-Score:", train_f1_score)
print("Testing F1-Score:", test_f1_score)

# Compute ROC curve and AUC for each class using OVR strategy
n_classes = len(np.unique(y_train))
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    binary_train_labels = convert_to_binary(y_train, i)
    binary_test_labels = convert_to_binary(y_test, i)
    binary_train_scores = convert_to_binary(dt_classifier.predict(X_train), i)
    binary_test_scores = convert_to_binary(y_pred, i)
    fpr[i], tpr[i], _ = roc_curve(binary_train_labels, binary_train_scores)
    roc_auc[i] = auc(fpr[i], tpr[i])

# her class için çizim
plt.figure()

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC Curve (AUC = %0.2f)' % roc_auc[i])

plt.plot([0, 1], [0, 1], 'r--', label='Random')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()

# Sonuclar
print("\nResults:")
print("The confusion matrix provides information about the model's performance across different classes.")
print("The accuracy, precision, recall, and F1-score metrics give an overall assessment of the model's performance.")
print("The ROC curve and AUC measure the model's ability to discriminate between the positive and negative classes.")
print("By examining these metrics, we can evaluate the effectiveness of the decision tree classifier.")
```

```
Training Confusion Matrix:
[[40  0  0]
 [ 0 41  0]
```

```
 [ 0  0 39]]

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

Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        41
           2       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120


Testing 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


Training Accuracy: 1.0
Testing Accuracy: 1.0
Training Precision: 1.0
Testing Precision: 1.0
Training Recall: 1.0
Testing Recall: 1.0
Training F1-Score: 1.0
Testing F1-Score: 1.0
```

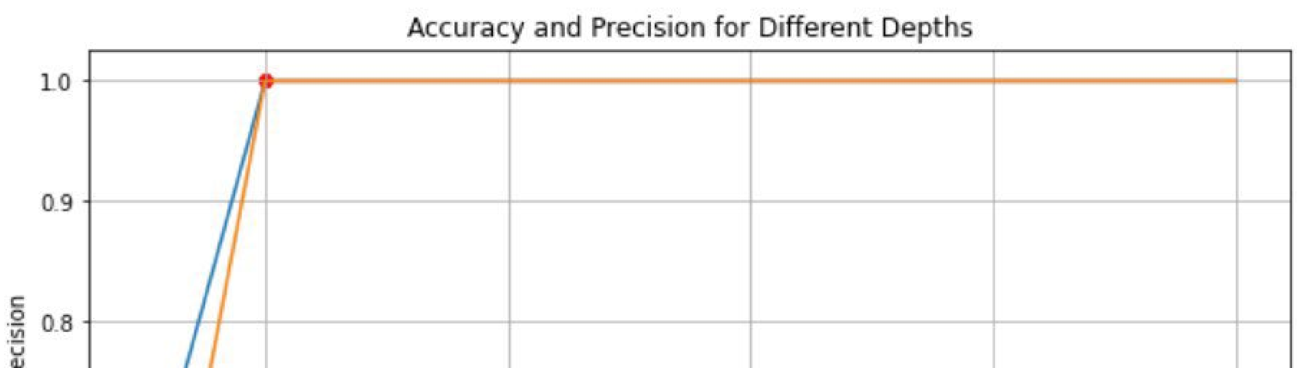


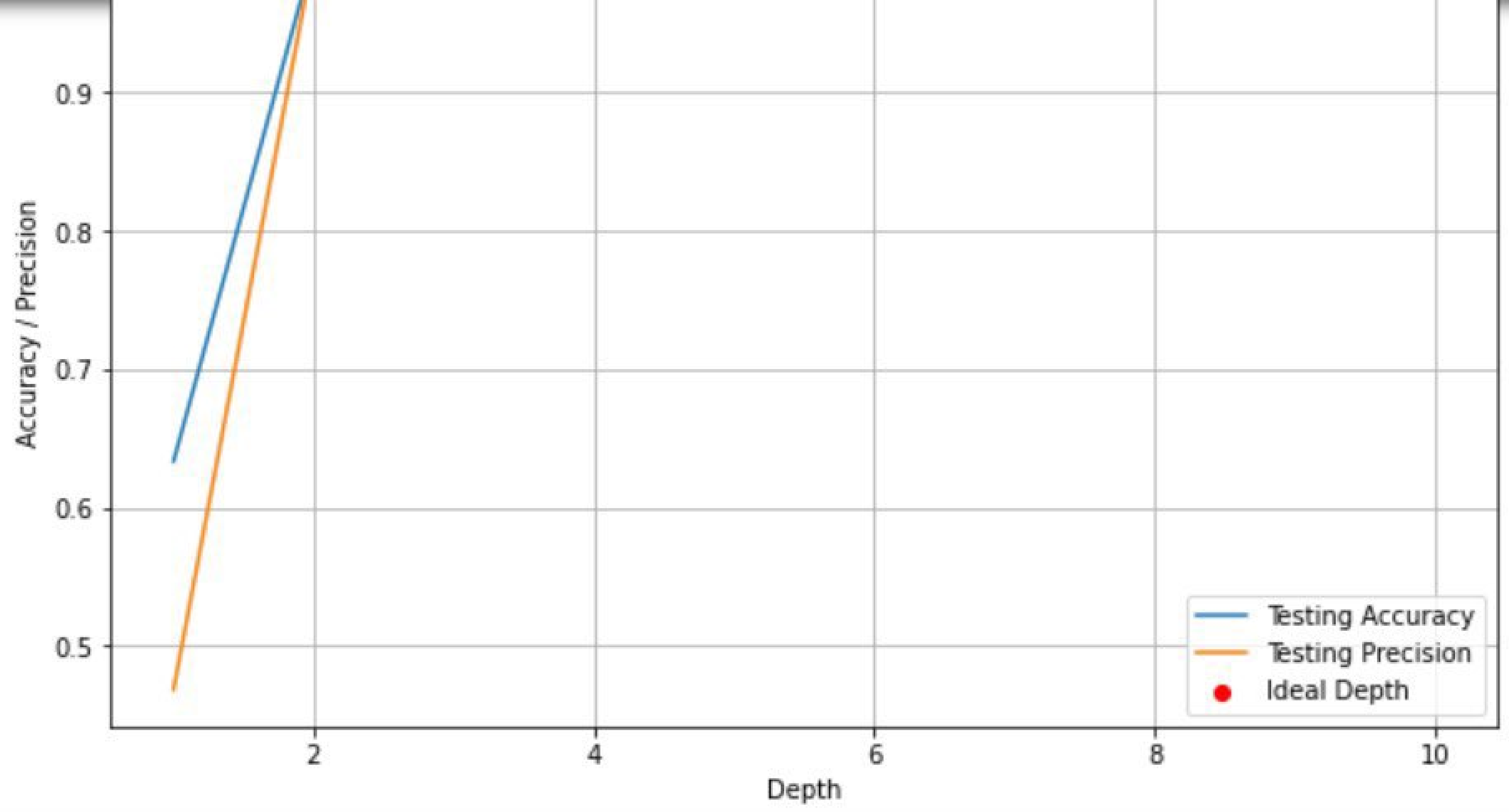

```
Results:
The confusion matrix provides information about the model's performance across different classes.
The accuracy, precision, recall, and F1-score metrics give an overall assessment of the model's performance.
The ROC curve and AUC measure the model's ability to discriminate between the positive and negative classes.
By examining these metrics, we can evaluate the effectiveness of the decision tree classifier.
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score
import matplotlib.pyplot as plt

L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
train_accuracies = []
test_accuracies = []
train_precisions = []
test_precisions = []

for depth in L:
    # Decision Tree Classifier
    dt_classifier = DecisionTreeClassifier(max_depth=depth)

    # Model eğitimi
    dt_classifier.fit(X_train, y_train)

    # Model evaluation on train set
    train_predictions = dt_classifier.predict(X_train)
```

```
    train_accuracy = accuracy_score(y_train, train_predictions)
    train_precision = precision_score(y_train, train_predictions, average='weighted')
    train_accuracies.append(train_accuracy)
    train_precisions.append(train_precision)

    # Model evaluation on test set
    test_predictions = dt_classifier.predict(X_test)
    test_accuracy = accuracy_score(y_test, test_predictions)
    test_precision = precision_score(y_test, test_predictions, average='weighted')
    test_accuracies.append(test_accuracy)
    test_precisions.append(test_precision)

# ideal depth değerini bulma
ideal_index = test_accuracies.index(max(test_accuracies))

# sonuçları çizdirme
plt.figure(figsize=(10, 6))
plt.plot(L, test_accuracies, label='Testing Accuracy')
plt.plot(L, test_precisions, label='Testing Precision')
plt.scatter(L[ideal_index], test_accuracies[ideal_index], color='red', label='Ideal Depth')
plt.xlabel('Depth')
plt.ylabel('Accuracy / Precision')
plt.title('Accuracy and Precision for Different Depths')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




0.9
0.8
0.7
0.6
0.5
Accuracy / Precision
2
4
6
8
10
Depth
Testing Accuracy
Testing Precision
Ideal Depth