.

**🌳 Decision Tree Classifier on the Iris Dataset**

**📘 Overview**

This project demonstrates the implementation of a **Decision Tree Classifier** using the classic **Iris dataset**. The goal is to classify iris flowers into three species based on four measurable features: sepal length, sepal width, petal length, and petal width.

**📂 Dataset Description**

* **Source**: sklearn.datasets.load\_iris()
* **Samples**: 150
* **Classes**:
  + 0: Setosa
  + 1: Versicolor
  + 2: Virginica
* **Features**:
  + sepal length (cm)
  + sepal width (cm)
  + petal length (cm)
  + petal width (cm)

**🧪 Project Objectives**

* Train a **Decision Tree Classifier** on the Iris dataset.
* Evaluate the model using classification metrics.
* Visualize both the **confusion matrix** and the **decision tree structure**.

**⚙️ Tools and Technologies**

* **Python**
* **Scikit-learn** (for ML models and dataset)
* **Matplotlib / Seaborn** (for visualization)
* **Pandas** (for data handling)

**🧾 Code Breakdown**

**1. Data Loading and Preparation**

python

CopyEdit

from sklearn.datasets import load\_iris

import pandas as pd

data = load\_iris()

X = pd.DataFrame(data.data, columns=data.feature\_names)

y = data.target

**2. Train-Test Split**

python

CopyEdit

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

**3. Training the Model**

python

CopyEdit

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=42)

model.fit(X\_train, y\_train)

**4. Evaluation**

python

CopyEdit

from sklearn.metrics import classification\_report, confusion\_matrix

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred, target\_names=data.target\_names))

**5. Confusion Matrix Visualization**

python

CopyEdit

import seaborn as sns

import matplotlib.pyplot as plt

conf\_mat = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_mat, annot=True, cmap='Greens',

xticklabels=data.target\_names,

yticklabels=data.target\_names)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix - Iris Classification")

plt.show()

**6. Decision Tree Plot**

python

CopyEdit

from sklearn.tree import plot\_tree

plt.figure(figsize=(12, 8))

plot\_tree(model, feature\_names=data.feature\_names,

class\_names=data.target\_names, filled=True, rounded=True)

plt.title("Decision Tree Visualization")

plt.show()

**📊 Results**

* **Model Accuracy**: High accuracy on test set (typically >90%)
* **Key Features**: Petal length and petal width are dominant decision points.
* **Interpretability**: The tree visualization helps explain model decisions in an intuitive way.

**✅ classification\_report**

This function from sklearn.metrics prints **precision**, **recall**, **f1-score**, and **support** for each class.

**A screenshot of a computer

AI-generated content may be incorrect.**

**🔍 Explanation of Each Metric:**

| **Metric** | **Meaning** |
| --- | --- |
| **Precision** | Out of the predicted instances for a class, how many were correct? |
| Precision = TP / (TP + FP) |  |
| **Recall** | Out of all actual instances of a class, how many were captured correctly? |
| Recall = TP / (TP + FN) |  |
| **F1-score** | Harmonic mean of precision and recall. A balanced score when both metrics matter. |
| F1 = 2 \* (precision \* recall) / (precision + recall) |  |
| **Support** | Number of **actual instances** for each class in the test set. |

**🔹 Macro vs Weighted Average:**

* **Macro Avg**: Arithmetic mean of the scores for each class (treats all classes equally).
* **Weighted Avg**: Accounts for class imbalance by weighting each score by its support.

🧠 **Use when**: You want a concise summary of model performance across all classes.

**🎯 What is a Confusion Matrix?**

A **confusion matrix** is a table used to describe the **performance of a classification model** on a test dataset where the **true values are known**.

It compares the **actual labels** with the **predicted labels** by the model.

**✅ Let’s Use a Real-Life Analogy**

Imagine you're building a model to **classify flowers** into 3 types:

* **Setosa**
* **Versicolor**
* **Virginica**

You trained your model, tested it on 30 samples, and now want to know how well it did.

**🧾 Output of confusion\_matrix (for example):**

python

CopyEdit

[[10, 0, 0],

[ 0, 9, 1],

[ 0, 0, 10]]

Let’s break this down into a table:

|  | **Predicted Setosa** | **Predicted Versicolor** | **Predicted Virginica** |
| --- | --- | --- | --- |
| **Actual Setosa** | 10 | 0 | 0 |
| **Actual Versicolor** | 0 | 9 | 1 |
| **Actual Virginica** | 0 | 0 | 10 |

**🧠 How to Read It**

* **Rows** → Actual classes
* **Columns** → Predicted classes

Now read each row:

* First row: There were 10 actual **Setosa** flowers. All 10 were predicted correctly → Perfect!
* Second row: There were 10 **Versicolor** flowers.
  + 9 predicted correctly as Versicolor
  + 1 predicted **wrongly as Virginica**
* Third row: 10 **Virginica** flowers → all predicted correctly.

**💡 Why is it called “Confusion” Matrix?**

Because it shows where your model is **getting confused** between classes.

In our example, the model **mistook 1 Versicolor as Virginica**, so there’s some confusion.

**✅ Ideal Confusion Matrix**

Looks like this (all correct, no confusion):

lua

CopyEdit

[[10, 0, 0],

[0, 10, 0],

[0, 0, 10]]

All values are on the diagonal (top-left to bottom-right), and all others are zero.

**🔍 Use Cases**

* Shows exactly **which classes are misclassified**
* Helps you **improve your model** by identifying weak spots
* Useful in **imbalanced datasets** where accuracy alone can be misleading

**📌 Conclusion**

This project highlights how a **Decision Tree Classifier** can be effectively used for **multi-class classification** problems. The Iris dataset serves as an ideal starting point due to its balanced size, clean features, and clearly defined classes. The decision tree structure makes the model easily interpretable — an important quality in many real-world applications.