In [4]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from collections import Counter
```

In [5]:
```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["label"] = iris.target
df["label"] = df["label"].map({0: "setosa", 1: "versicolor", 2: "virginica"})
```

In [6]:
```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

In [7]:
```
def gini(data):
    labels = data["label"]
    counts = Counter(labels)
    total = len(labels)
    return 1 - sum((count/total)**2 for count in counts.values())
```

In [8]:
```
def best_split(data, attributes):
    best_attr, best_threshold, best_gini = None, None, 1

    for attr in attributes:
        values = sorted(data[attr].unique())
        for i in range(len(values) - 1):
            threshold = (values[i] + values[i+1]) / 2
            left_split = data[data[attr] <= threshold]
            right_split = data[data[attr] > threshold]

            if len(left_split) == 0 or len(right_split) == 0:
                continue

            weighted_gini = (len(left_split)/len(data))*gini(left_split) + \ (len(r

            if weighted_gini < best_gini:
                best_gini = weighted_gini
                best_attr = attr
                best_threshold = threshold

    return best_attr, best_threshold, best_gini
```

In [9]:
```
def cart(data, attributes, depth=0, max_depth=3):
    labels = data["label"]

    if len(set(labels)) == 1:
        return labels.iloc[0]
    if len(attributes) == 0 or depth == max_depth:
        return Counter(labels).most_common(1)[0][0]

    best_attr, best_threshold, best_gini = best_split(data, attributes)
    if best_attr is None:
        return Counter(labels).most_common(1)[0][0]
```

```
    left_split = data[data[best_attr] <= best_threshold]
    right_split = data[data[best_attr] > best_threshold]

    tree = {f"{best_attr} <= {best_threshold:.2f}": cart(left_split, attributes, de
            f"{best_attr} > {best_threshold:.2f}": cart(right_split, attributes, de

    return tree
```

In [10]:
```
def predict(tree, sample):
    if not isinstance(tree, dict):
        return tree

    node = list(tree.keys())[0]   # e.g. "sepal length (cm) <= 5.45"

    if "<=" in node:
        attr, threshold = node.split(" <= ")
        threshold = float(threshold)
        if sample[attr] <= threshold:
            return predict(tree[node], sample)
        else:
            other_key = f"{attr} > {threshold:.2f}"
            return predict(tree[other_key], sample)

    elif ">" in node:
        attr, threshold = node.split(" > ")
        threshold = float(threshold)
        if sample[attr] > threshold:
            return predict(tree[node], sample)
        else:
            other_key = f"{attr} <= {threshold:.2f}"
            return predict(tree[other_key], sample)

    else:
        return tree[node]
```

In [11]:
```
attributes = list(df.columns[:-1])
tree = cart(train_df, attributes, max_depth=3)

print("Decision Tree (CART):")
print(tree)
```

```
Decision Tree (CART):
{'petal length (cm) <= 2.45': 'setosa', 'petal length (cm) > 2.45': {'petal length
(cm) <= 4.75': {'petal width (cm) <= 1.65': 'versicolor', 'petal width (cm) > 1.65':
'virginica'}, 'petal length (cm) > 4.75': {'petal width (cm) <= 1.75': 'virginica',
'petal width (cm) > 1.75': 'virginica'}}}
```

In [12]:
```
correct = 0
for _, row in test_df.iterrows():
    pred = predict(tree, row)
    if pred == row["label"]:
        correct += 1

accuracy = correct / len(test_df)
print("\nTest Accuracy:", accuracy)
```

```
Test Accuracy: 0.9666666666666667
```

In [13]:
```python
sample = test_df.iloc[0]
print("\nTest Sample:", sample[:-1].to_dict())
print("True Label:", sample["label"])
print("Predicted:", predict(tree, sample))
```

```
Test Sample: {'sepal length (cm)': 6.1, 'sepal width (cm)': 2.8, 'petal length (c
m)': 4.7, 'petal width (cm)': 1.2}
True Label: versicolor
Predicted: versicolor
```

In [ ]:
```python

```