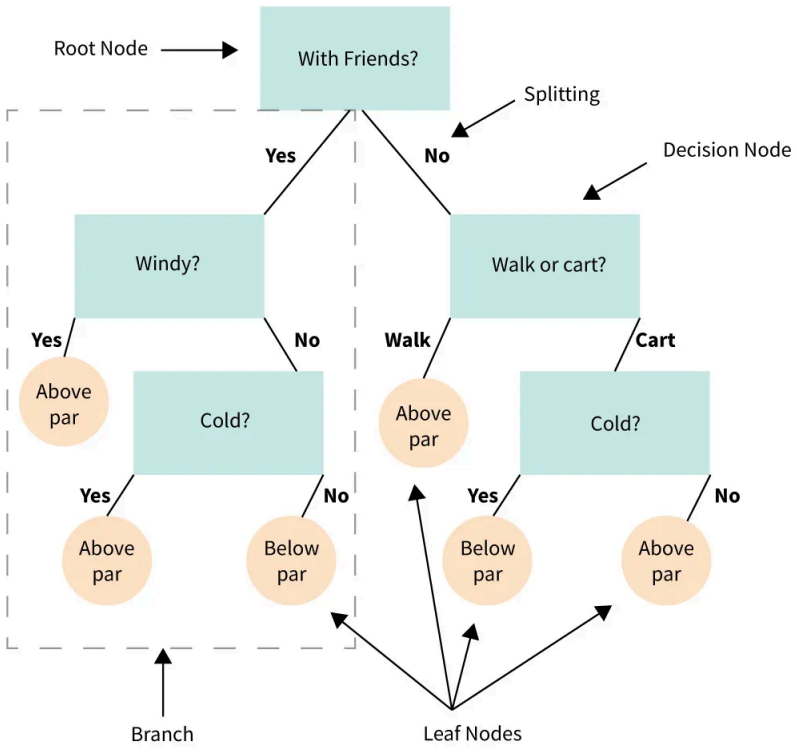
Decision Tree

```
rom sklearn. tree import DecisionTreeClassifier
```

- You can have **mixed data** i.e. **Numeric & Categorical**

> A **Decision Tree** is like a flowchart where each question (node) splits the data based on conditions until a final decision (leaf) is reached.



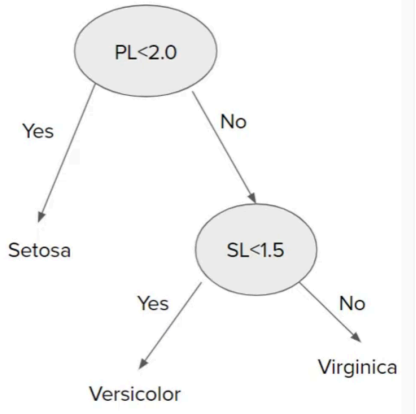
How It Works

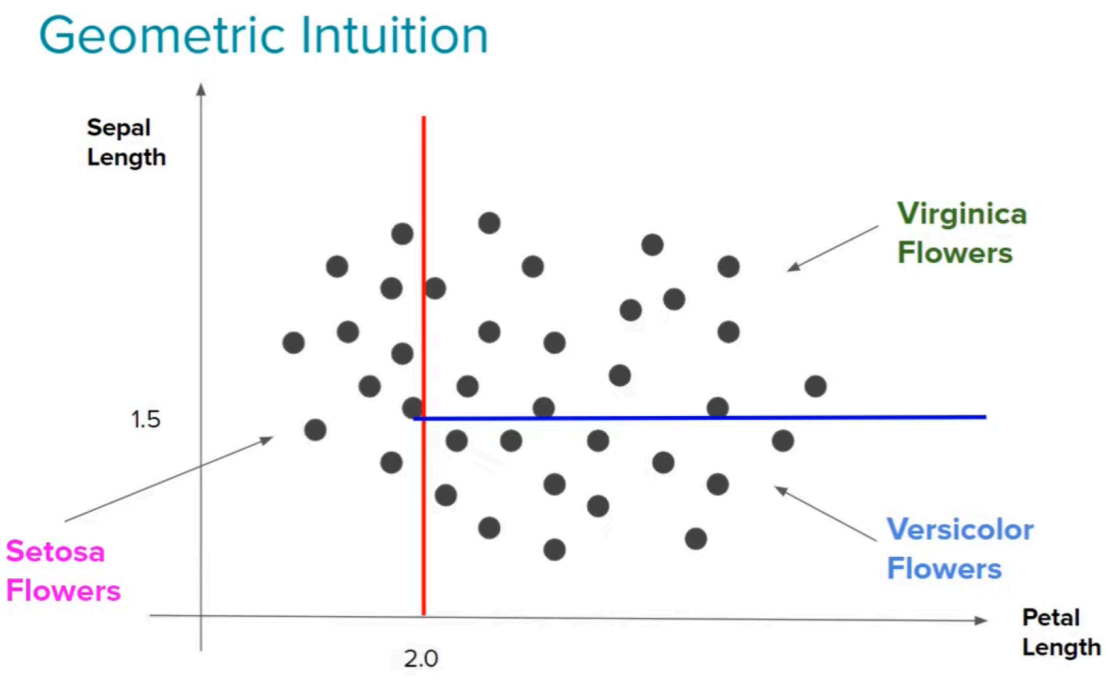
1. **Start with the full dataset.**
2. **Pick the best feature** (the one that best splits the data).
3. **Split the data** into smaller groups.
4. **Repeat** for each group until stopping conditions are met.
5. **Stop When You Reach a Leaf**:
    - The splitting stops when the data in a group is **pure (all the same)** or when a **stopping condition is met (e.g., maximum depth of the tree)**.
    - The final groups are called **leaf nodes**, and they represent the predictions.

For Numerical Data

What if we have numerical data?

Petal Length	Sepal Length	Type
1.34	0.34	Setosa
3.45	1.45	Versicolor
1.69	0.98	Setosa
2.56	1.79	Virginica
3.00	1.13	Versicolor
1.3	0.88	Setosa





Key Components

- **Root Node** → The first decision/split.
  - The topmost node where the tree starts.
- **Decision Node**:
  - A node where the data is split based on a question.
- **Internal Nodes** → More splits based on features.
- **Leaf Nodes** → The final decisions (**outputs**).
  - The final node that gives the **prediction** (e.g., "play outside" or "stay indoors").
- **Branch**:
  - The path from one node to another.
- **Splitting**:
  - Dividing the data into smaller groups based on a feature (e.g., "Is it raining?").

- **Impurity**:
  - A measure of how mixed the data is in a node. The goal is to reduce impurity as much as possible.

Splitting Criteria

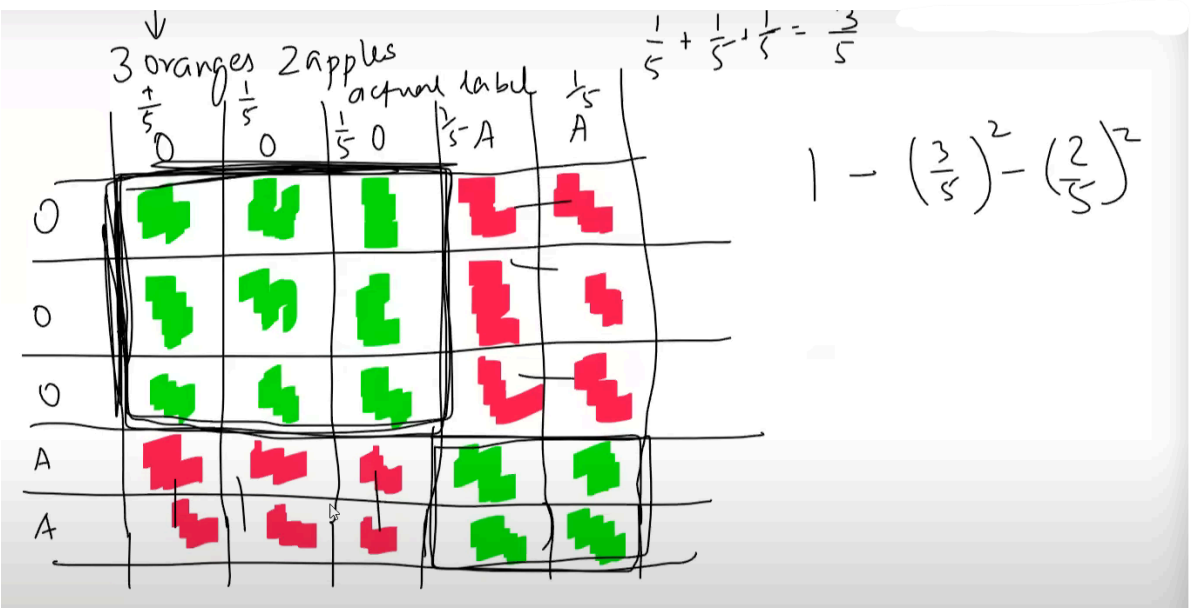
A decision tree chooses splits using:

1. **Gini Impurity**:
   - Measures how often a randomly chosen data point would be incorrectly labeled.
   - The **tree tries to minimize Gini impurity.**
   - It's another measure of "impurity" in the data. A Gini score of **0 means perfect purity** (all data points belong to the same class).

$$Gini(S) = 1 - \sum_{i=1}^{C} p_i^2$$

Where p_i is the probability of each class.

- Max value is **0.5 (in case of binary categories)**
- Faster
  - Preferred for big datasets.
  - Log in entropy takes more time.



- Gini entropy is the probability of red values👆.

2. **Entropy**:
   - Measures the **disorder** or **uncertainty** in the data.
   - The tree tries to **minimize** **entropy.**

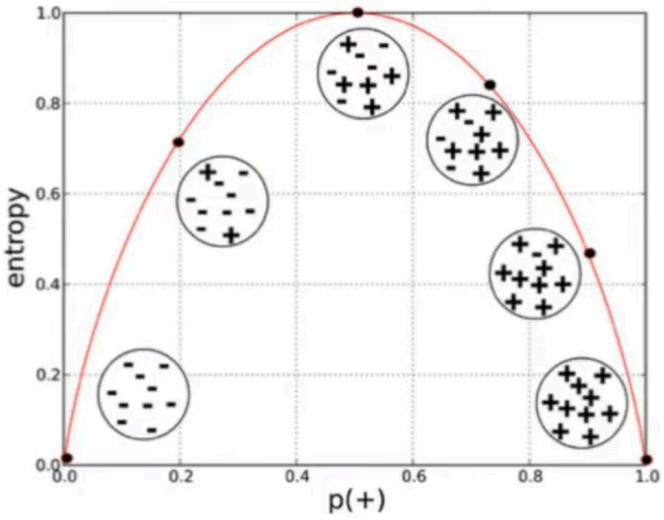
$$Entropy(S) = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \ldots - p_n \log_2(p_n)$$

Where p_i is the probability of each class in the dataset.

> ***More knowledge, less entropy.***

- For 2 class problem, the min entropy is 0 & **max entropy is 1**.
- For 2+ class problem, min entropy is 0 but max entropy can be greater than 1.
- Both log_2& log_ecan be used to calculate entropy.

Entropy Vs Probability



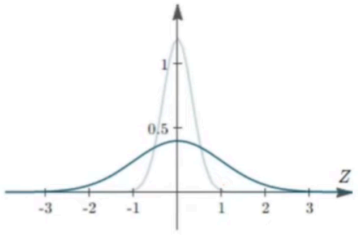
Entropy for continuous variables

Area	Built in	Price
1200	1999	3.5
1800	2011	5.6
1400	2000	7.3
...	...	...

Dataset 1

Area	Built in	Price
2200	1989	4.6
800	2018	6.5
1100	2005	12.8
...	...	...

Dataset 2

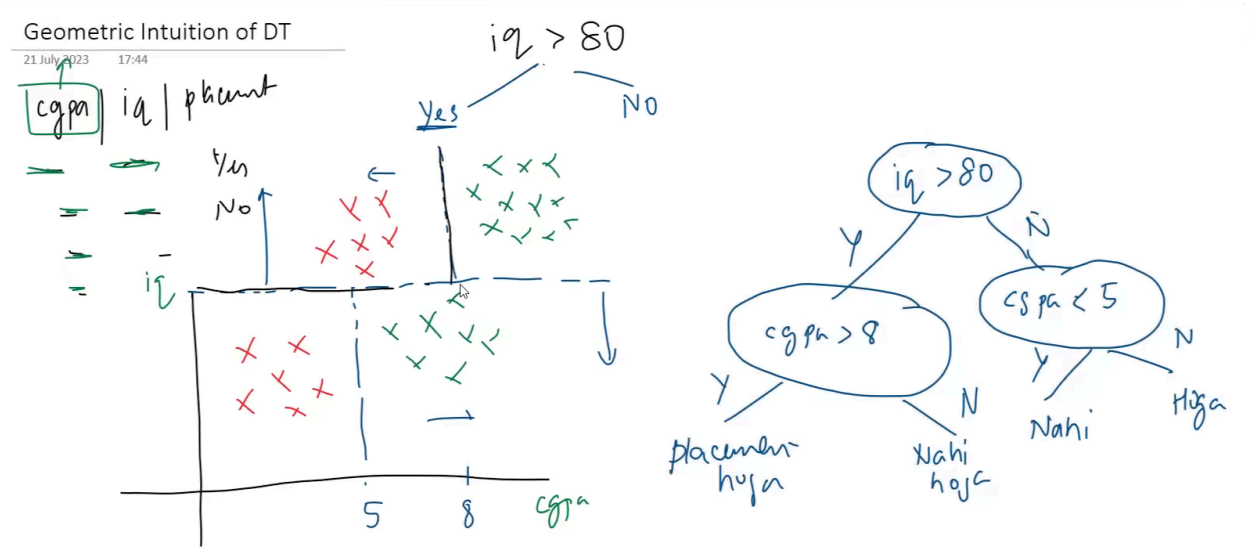


Quiz: Which of the above datasets have higher entropy?

Ans: Whichever is less peaked

3. **Information Gain:**
    - Measures how much a split reduces impurity.
    - The tree chooses the split that gives the **highest** **information gain.**

Geometric Intuition of DT



Types of Decision Trees

1. **Classification Tree** → Predicts categories (e.g., spam or not).
2. **Regression Tree** → Predicts numbers (e.g., house prices).

Python Code Example (Classification Tree)

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.
2, random_state=42)

# Create & train the model
tree = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
tree.fit(X_train, y_train)

# Make predictions
```

```
y_pred = tree.predict(X_test)

# Evaluate accuracy
from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

Hyperparameter Tuning for Decision Trees

- **criterion** → `"gini"` (default) or `"entropy"` (for information gain).
- `max_depth` → Limits tree depth to prevent overfitting. (**Default:** *`max_depth=None`*)
- `min_samples_split` → Minimum samples needed to split a node.
- `min_samples_leaf` → Minimum samples required in a leaf node.

> 💡 Upon removing `max_depth=3`, it took more time to run.

Advantages of Decision Trees:

- **Simple to Understand**: Decision trees are easy to interpret and visualize.
- **Non-linear**: They can capture complex relationships between features.
- **No Need for Feature Scaling**: They don't require normalization or standardization of data.
- **Handle Both Numerical and Categorical Data**: They can deal with both types of data.

Disadvantages of Decision Trees:

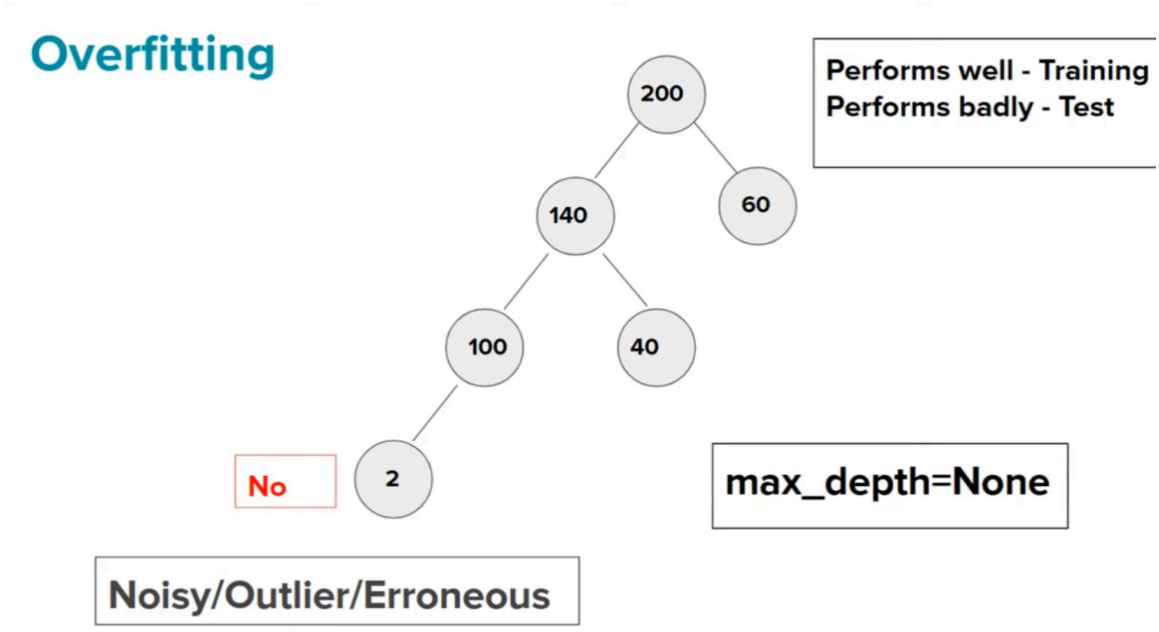
- **Overfitting**: Decision trees can easily overfit the data, especially if they are deep.

- **Prone to Imbalance Datasets**
- **Instability**: Small changes in the data can lead to large changes in the tree structure.
- **Bias toward Features with More Categories**: Decision trees tend to favor features with more levels (values).

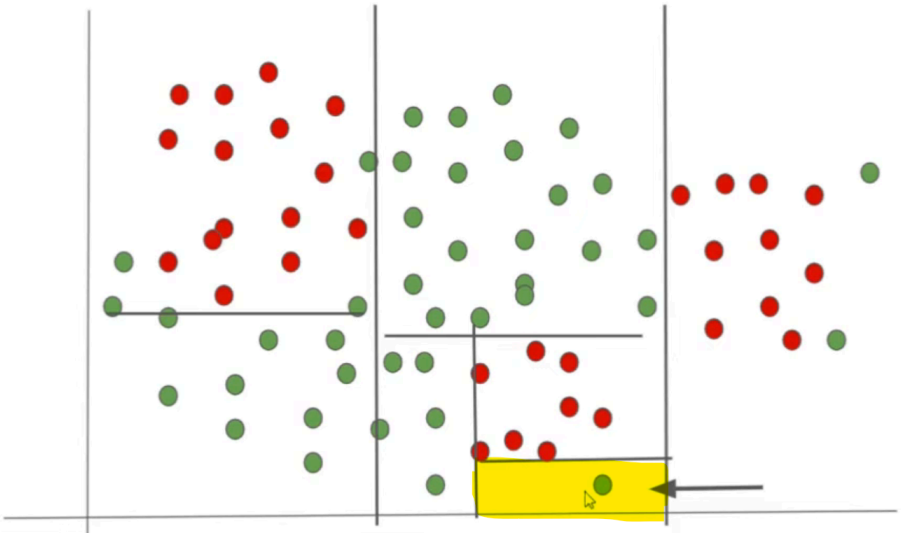
When to Use Decision Trees?

- When you want a **simple, interpretable model.**
- When your data has a **mix of numerical and categorical features**.
- When you want to **visualize the decision-making process.**

Underfitting vs Overfitting in Decision Tree

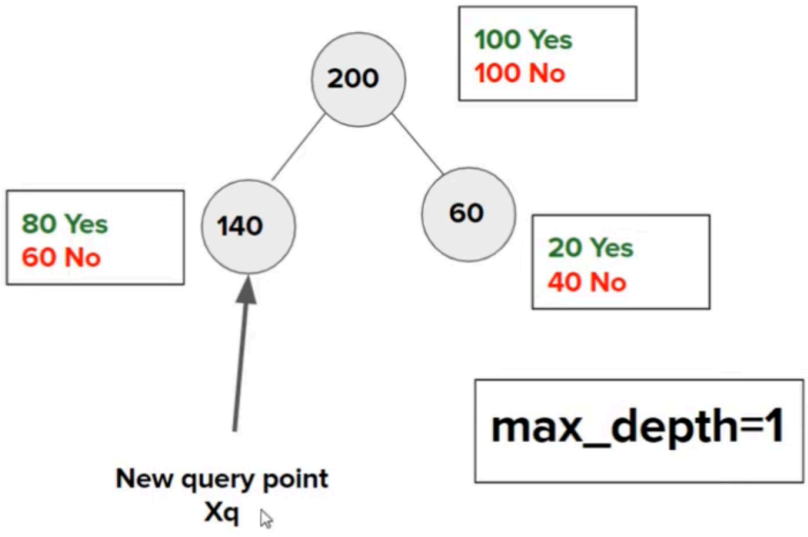


Geometric Intuition of Overfitting



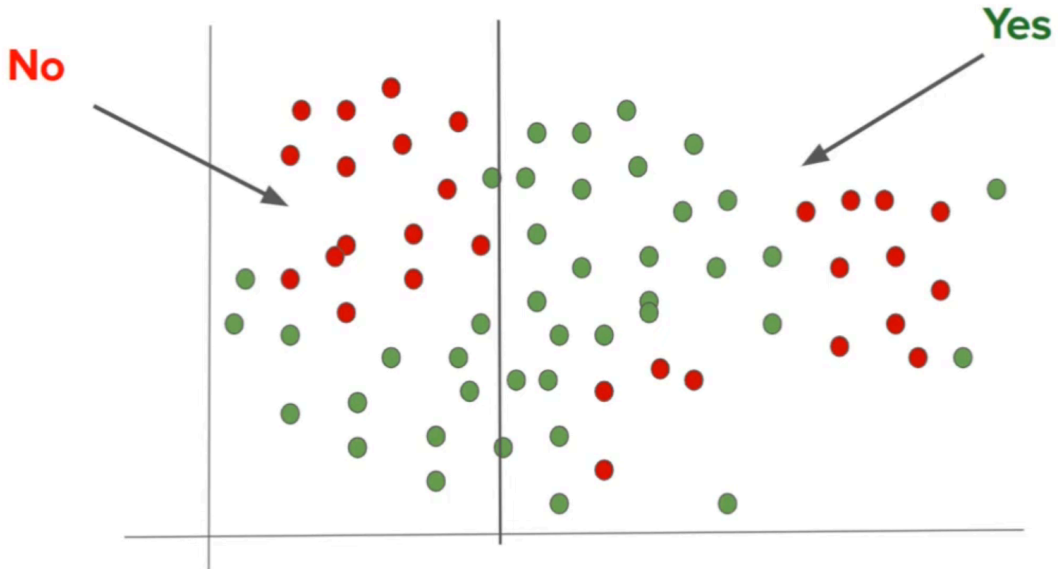
- The box should have been a red region, but due to overfitting, any point falling in it will be classified as green.

Underfitting



- If a point falls in the 140 region, it will be classified as Yes because number of Yes is more.

Geometric Intuition of Underfitting



- Tune `max_depth` to prevent these.

Hyperparameter Tuning for Decision Tree

- `splitter='best'` *(default)*
  - `"random"` → *Reduces overfitting*
  - Generally kept as default
- `max_depth = None` *(default)*
  - Low (1 or 2) : Underfitting
  - High : Overfitting
- `min_samples_split=2` *(default)*
  - The minimum number of samples required to split an internal node.
  - High : Underfitting
  - Low : Overfitting
- `min_samples_leaf=1` *(default)*

    - The minimum number of samples required to be at a leaf node.
    - High : Underfitting
    - Low : Overfitting
- `max_features=None` *(default)*
    - The number of features to consider when looking for the best split
- `min_impurity_decrease=0.0` *(default)*
    - A node will be split if this split induces a decrease of the impurity greater than or equal to this value
    - eg. 0.01

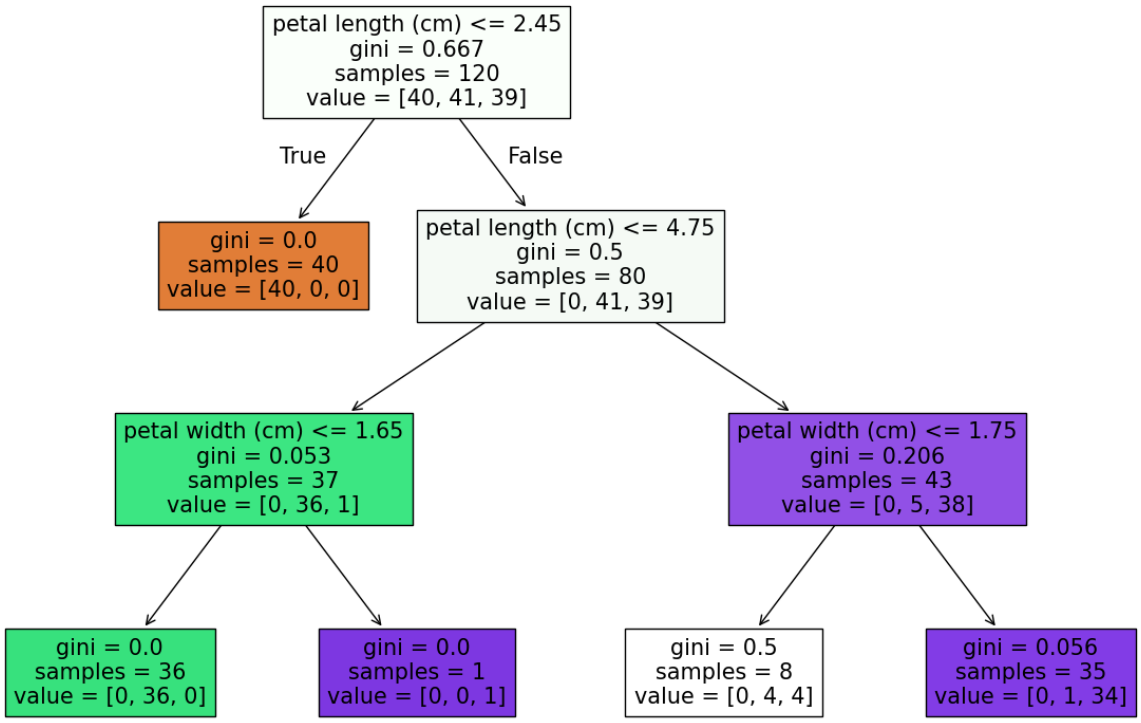
Visualize Decision Tree

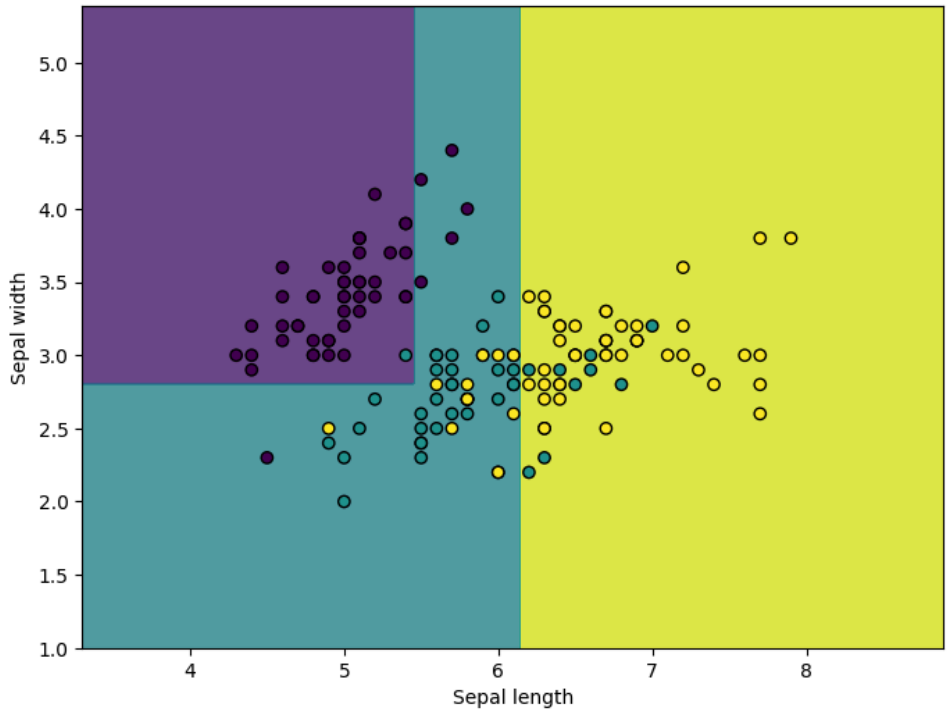
`from sklearn import tree`

`tree.plot_tree(tree1)`

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.
2, random_state=42)

# Create & train the model
tree1 = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=4
2)
tree1.fit(X_train, y_train)

# Make predictions
y_pred = tree1.predict(X_test)
```

```
# Evaluate accuracy
from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(y_test, y_pred))

from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(15,10))
tree.plot_tree(tree1, feature_names=iris.feature_names, filled=True)
```





Print the Logic

```
r = export_text(clf, feature_names=['sepal_length','sepal_width'])
print(r)
```

```
|--- sepal_length <= 5.45
|   |--- sepal_width <= 2.80
|   |   |--- class: 1
|   |--- sepal_width >  2.80
|   |   |--- class: 0
|--- sepal_length >  5.45
|   |--- sepal_length <= 6.15
|   |   |--- class: 1
|   |--- sepal_length >  6.15
|   |   |--- sepal_length <= 7.05
|   |   |   |--- class: 2
|   |   |--- sepal_length >  7.05
|   |   |   |--- class: 2
```

```
clf = DecisionTreeClassifier(max_depth=3,min_samples_split=40)
```