Complete understanding of Decision Tree with GridSearchCV

Prashant Sundge

LinkedIn GitHub

👍 If you found this kernel helpful, please consider giving it an upvote! Your support motivates me to create more valuable content.

💬 I'd also love to hear your feedback and comments. Let me know if you have any questions, suggestions, or insights. Your input is highly appreciated!

Happy coding! 😊

Table of Contents

Introduction

Definition of a Decision Tree:

A Decision Tree is a hierarchical and tree-like structure used in machine learning for both classification and regression tasks. It systematically divides data into subsets based on the values of input features, ultimately leading to decisions or predictions. It consists of nodes, branches, and leaves, where nodes represent feature attributes, branches represent decision rules, and leaves represent the final outcomes or predictions.

Function of a Decision Tree:

The primary function of a Decision Tree is to make decisions or predictions based on input data. It works by recursively splitting the dataset into subsets based on the most informative features, using a set of predefined criteria such as Gini impurity or entropy for classification and mean squared error for regression. The tree structure guides the decision-making process, where each internal node represents a decision point, and each leaf node represents a class label (in classification) or a predicted value (in regression).

In classification, a Decision Tree helps classify data into different categories or classes, while in regression, it predicts numerical values based on the input features. The simplicity and interpretability of Decision Trees make them valuable tools in machine learning, allowing users to understand and visualize decision-making processes.

Import Libraries

In [1]:
```
import pandas as pd
```

Read Dataset

In [2]:
```
example=pd.read_excel("Example1.xlsx")
```

In [3]:
```
example
```

Out[3]:

	Gender	Occupation	Suggestion
0	F	Student	ENGINEER
1	F	Programmer	JAVA
2	M	Programmer	PYTHON
3	F	Programmer	JAVA
4	M	Student	ENGINEER
5	M	Student	ENGINEER

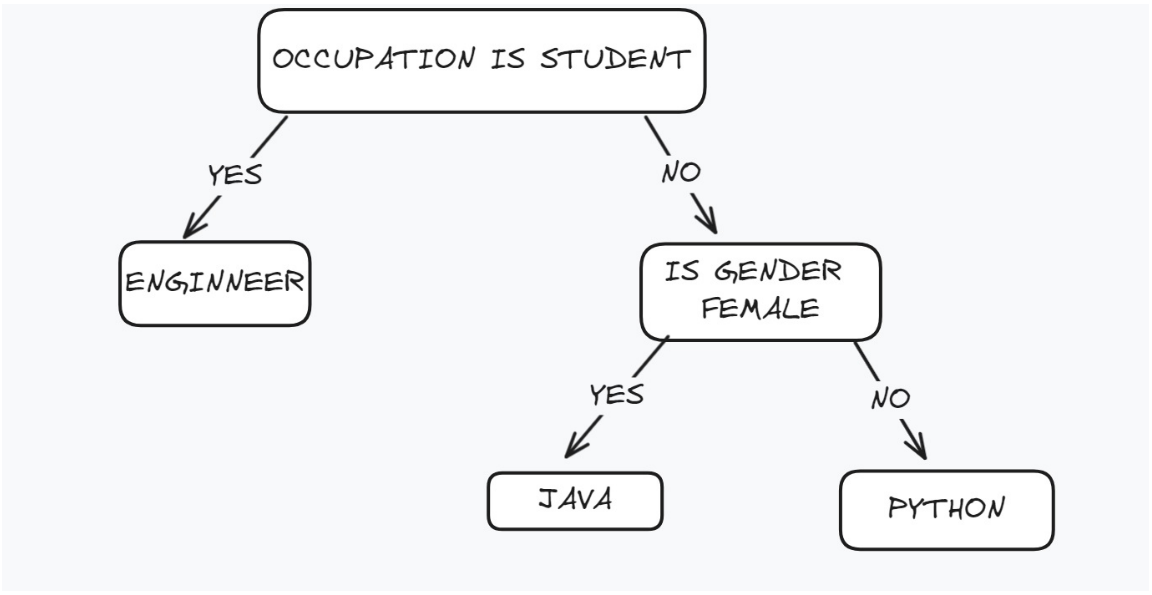
IF ELSE Representation of Decision Tree

In [4]:
```
def decision(row):
    if row['Occupation'] == 'Student':
        print("ENGINEER")
    elif row['Gender'] == 'F':
        print("JAVA")
    else:
        print("PYHTON")
```

```
In [5]: example['Suggestion_pred'] = example.apply(decision, axis=1)
```

```
ENGINEER
JAVA
PYHTON
JAVA
ENGINEER
ENGINEER
```

Tree Representation of Dataset

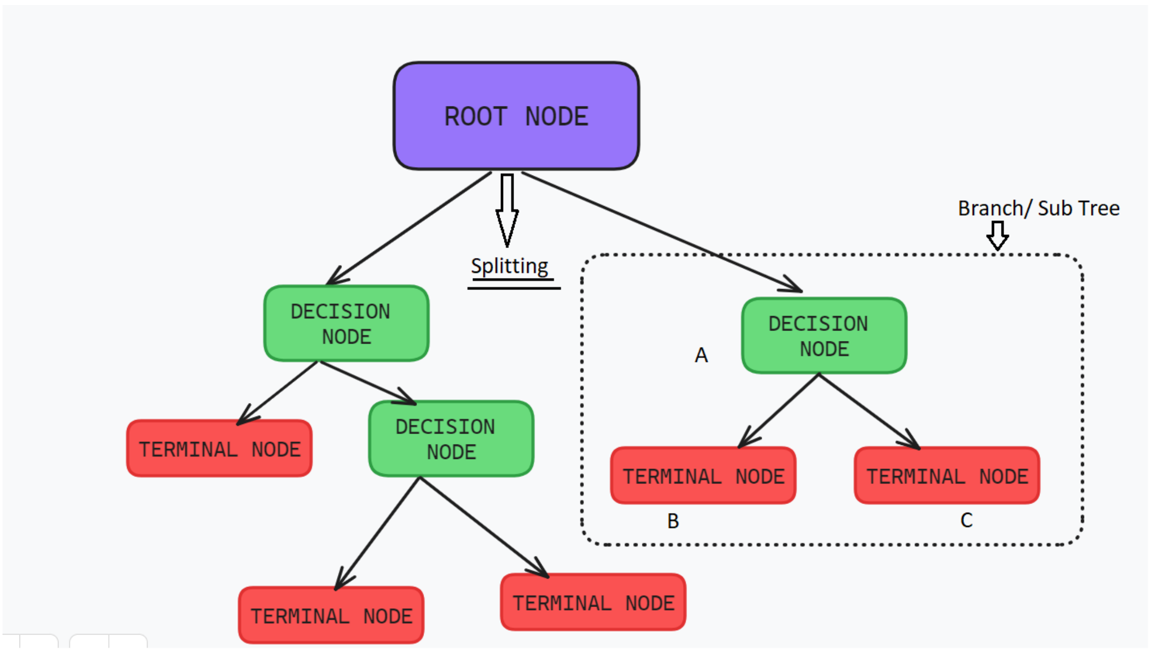


2. Decision Tree Basics

Here, we'll dive into the fundamental concepts of decision trees...

Decision Tree Terminologies

- **Root Node:** The initial node at the beginning of a decision tree, where the entire population or dataset starts dividing based on various features or conditions.
- **Decision Nodes:** Nodes resulting from the splitting of root nodes are known as decision nodes. These nodes represent intermediate decisions or conditions within the tree.
- **Leaf Nodes:** Nodes where further splitting is not possible, often indicating the final classification or outcome. Leaf nodes are also referred to as terminal nodes.
- **Sub-Tree:** Similar to a subsection of a graph being called a sub-graph, a sub-section of a decision tree is referred to as a sub-tree. It represents a specific portion of the decision tree.
- **Pruning:** The process of removing or cutting down specific nodes in a decision tree to prevent overfitting and simplify the model.
- **Branch / Sub-Tree:** A subsection of the entire decision tree is referred to as a branch or sub-tree. It represents a specific path of decisions and outcomes within the tree.
- **Parent and Child Node:** In a decision tree, a node that is divided into sub-nodes is known as a parent node, and the sub-nodes emerging from it are referred to as child nodes. The parent node represents a decision or condition, while the child nodes represent the potential outcomes or further decisions based on that condition.



Decision Tree Terminalogy Naming conventions

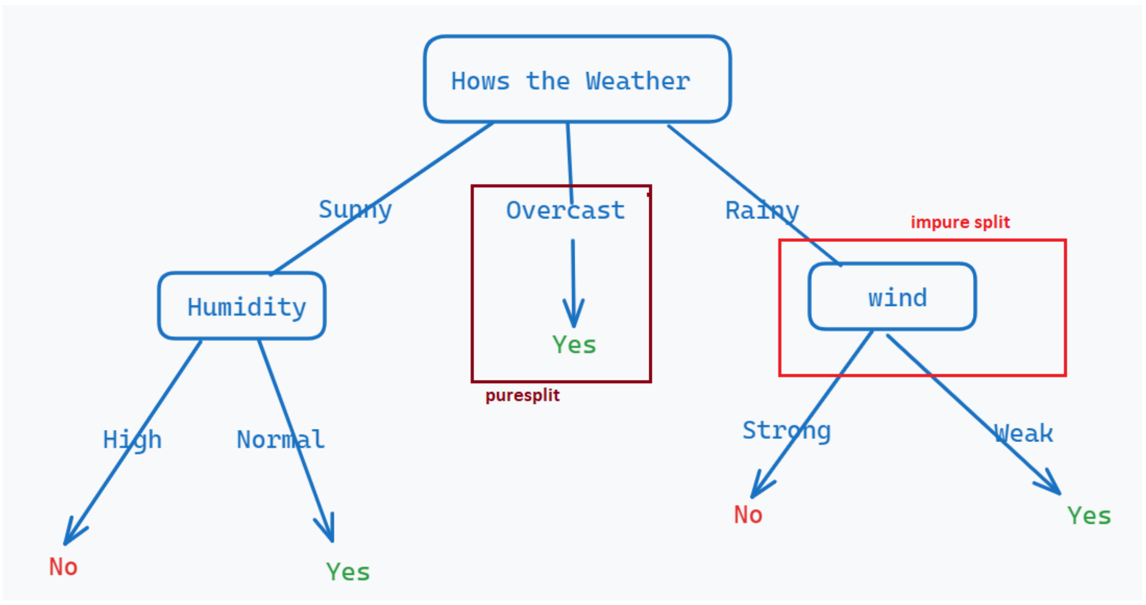
Example of Decision Tree

```
In [6]: play_tennis=pd.read_csv('play_tennis.csv')
```

```
In [7]: play_tennis
```

Out[7]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Cool	Normal	Strong	Yes
12	D13	Sunny	Mild	Normal	Strong	Yes
13	D14	Rain	Mild	High	Strong	No



ROOT NODE IF ELSE NODE

Root Node: Outlook

- If Outlook is Sunny:
    - Subnode: Humidity
        - If Humidity is High: Don't Play Tennis
        - If Humidity is Normal: Play Tennis
- If Outlook is Overcast: Play Tennis
- If Outlook is Rainy:
    - Subnode: Wind
        - If Wind is Weak: Play Tennis
        - If Wind is Strong: Don't Play Tennis

A decision tree is a machine learning algorithm that makes decisions based on the values of attributes (features) in a dataset. It works as follows:

Building the Tree

Root Node: The entire dataset is considered initially, and a root node is created.

Splitting: To create child nodes, the algorithm selects an attribute and splits the data based on its values. The attribute selection is based on **criteria like Gini Impurity and Entropy.**

Entropy:

- Entropy measures the average information content in a dataset. For a node, it's calculated as:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \, log_2(p_i)$$

$Entropy(S)$ is the entropy of node S. c is the number of classes. p_i is the probability of a randomly chosen data point belonging to class i.

Entropy is minimized when all data points in the node belong to a single class, and it is a measure of the disorder in the data.

In [8]:
```
play_tennis
```

Out[8]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Cool	Normal	Strong	Yes
12	D13	Sunny	Mild	Normal	Strong	Yes
13	D14	Rain	Mild	High	Strong	No

$$Entropy(S) = -(P_{\oplus} log_2 P_{\oplus} + P_{\ominus} log_2 P_{\ominus}) \qquad (1.1)$$

Where $P_{\oplus}$ is the portion of positive examples and $P_{\ominus}$ is the portion of negative examples in S.

Example

- To illustrate the equation, we will do an example that calculates the entropy of our dataset

$$Entropy([9+, 5-]) = -(\frac{9}{14} log_2 \frac{9}{14} + \frac{5}{14} log_2 \frac{5}{14}) = 0.940 \qquad (1.2)$$

Which concludes, the dataset is 94% impure or 94% non-homogeneous.

Let's do some more calculations and try to understand the nature of *Entropy*. What could be the Entropy of [7+,7-] & [14+,0-]?

$$Entropy[7+, 7-] = -(\frac{7}{14} log_2 \frac{7}{14} + \frac{7}{14} log_2 \frac{7}{14}) = 1 \qquad (1.3)$$

And,

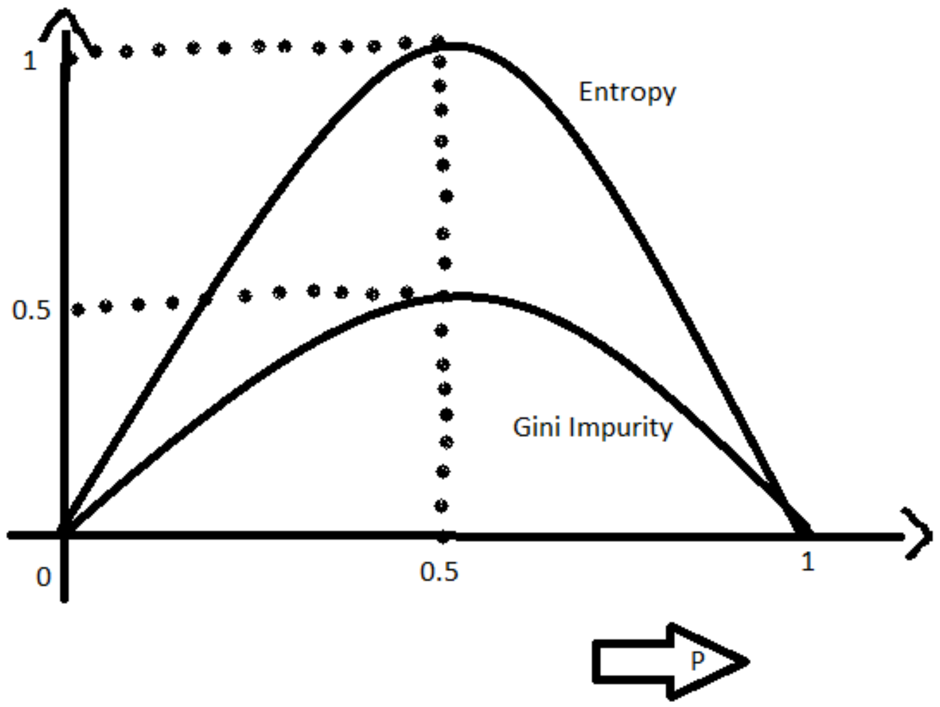
$$Entropy[14+, 0-] = -(\frac{14}{14} log_2 \frac{14}{14} + \frac{0}{14} log_2 \frac{0}{14}) = 0 \qquad (1.4)$$

When a dataset is completely homogeneous (all data points belong to a single class), it has zero impurity, and the entropy is zero (Equation 1.4).

In contrast, if the dataset can be equally divided into two classes, it is entirely non-homogeneous, resulting in maximum impurity of 100%, and the entropy is one (Equation 1.3).

Impurity and entropy are measures used to quantify the level of disorder or uncertainty in a dataset, with higher values indicating greater impurity and uncertainty, and lower values indicating greater homogeneity and certainty.

Entropy and GiNi Impurity Ranges



$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

Information Gain:

- Information Gain is a measure used to assess the effectiveness of an attribute in classifying a training dataset. It quantifies the expected reduction in entropy achieved by partitioning the dataset based on this attribute.

- Information Gain, denoted as **Gain(S, A)**, is a function of an attribute **A** relative to a collection of data **S**.

- The formula for Information Gain is as follows:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \qquad (1.5)$$

Where, $Values(A)$ is the all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

Where:

- **Gain(S, A)** is the Information Gain for attribute **A** in dataset **S**.
- **Entropy(S)** is the entropy of the entire dataset **S**.
- **S_v** represents subsets of the dataset **S** created by partitioning it based on the values of attribute **A**.
- The summation iterates over all possible values of attribute **A**.

- The Information Gain measures how much uncertainty or impurity is reduced when you split the dataset based on attribute **A**. A higher Information Gain indicates that attribute **A** is more effective in making distinctions within the dataset.

- Decision tree algorithms use Information Gain (or similar criteria) to determine the best attribute for splitting the data at each node, aiming to create a tree structure that maximizes the reduction in impurity as it grows.

To become more clear, let's use this equation and measure the **information gain** of attribute **Wind** from the dataset of Figure 1. The dataset has 14 instances, so the sample space is 14 where the sample has 9 positive and 5 negative instances. The Attribute **Wind** can have the values **Weak or Strong.** Therefore,

Values(Wind) = Weak, Strong

$S = [9+, 5-]$
$S_{\text{weak}} = [6+, 2-]$
$S_{\text{strong}} = [3+, 3-]$
$Entropy(S) = 0.940$ from eqtn 1.2

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Wind) = Entropy(S) - (\frac{8}{14} Entropy(S_{\text{weak}}) + \frac{6}{14} Entropy(S_{\text{strong}})) \tag{1.6}$$

$$Entropy(S_{\text{weak}}) = -(\frac{6}{8} log_2 \frac{6}{8} + \frac{2}{8} log_2 \frac{2}{8}) = 0.811 \tag{1.7}$$

$$Entropy(S_{\text{strong}}) = -(\frac{3}{3} log_2 \frac{3}{3} + \frac{3}{3} log_2 \frac{3}{3}) = 1.00 \tag{1.8}$$

Put the values of $Entropy(S_{\text{weak}})$ and $Entropy(S_{\text{strong}})$ in eqtn 1.6

$$\begin{aligned} Gain(S, Wind) &= Entropy(S) - (\frac{8}{14} 0.811 + \frac{6}{14} 1.00) \\ &= 0.940 - (0.463 + 0.429) \\ &= 0.048 \end{aligned} \tag{1.9}$$

So, the **information gain** by the **Wind** attribute is **0.048.**

Let's calculate the **information gain** by the **Outlook** attribute.

$Values(Outlook) = Sunny, Overcast, Rain$
$S = [9+, 5-]$
$S_{\text{sunny}} = [2+, 3-]$
$S_{\text{overcast}} = [4+, 0-]$
$S_{\text{rain}} = [3+, 2-]$

$$G(S, Outlook) = Entropy(S) - (\frac{5}{14} Entropy(S_{\text{sunny}}) + \frac{4}{14} Entropy(S_{\text{overcast}}) + \frac{5}{14} Entropy(S_{\text{rain}})) \tag{1.10}$$

$$Entropy(S_{\text{sunny}}) = -(\frac{2}{5} log_2 \frac{2}{5} + \frac{3}{5} log_2 \frac{3}{5}) = 0.971 \tag{1.11}$$

$$Entropy(S_{\text{overcast}}) = -(\frac{4}{4} log_2 \frac{4}{4} + \frac{0}{4} log_2 \frac{0}{4}) = 0 \tag{1.12}$$

$$Entropy(S_{\text{rain}}) = -(\frac{3}{5} log_2 \frac{3}{5} + \frac{2}{5} log_2 \frac{2}{5}) = 0.971 \tag{1.13}$$

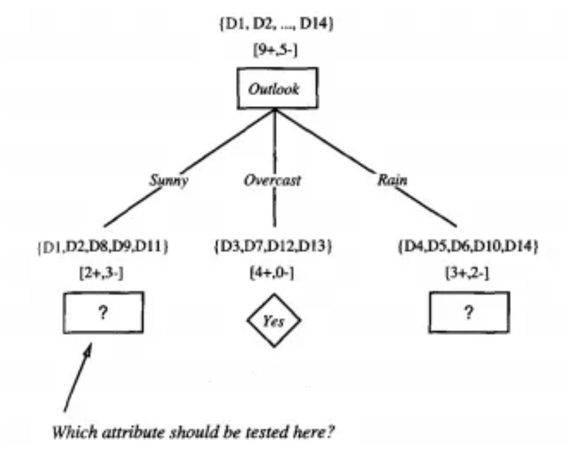
Put the values of eqtn 1.11, 1.12, 1.13 in 1.10.

These two examples should make us clear that how we can calculate **information gain.** The information gain of the 4 attributes of Figure 1 dataset are:

$Gain(S, Outlook) = 0.246$
$Gain(S, Humidity) = 0.151$
$Gain(S, Wind) = 0.048$
$Gain(S, Temperature) = 0.029$

It's crucial to keep in mind that the primary objective of assessing information gain is to pinpoint the attribute that is most valuable for classifying the training set. Our ID3 algorithm will employ this selected attribute as the root from which to construct the decision tree. Subsequently, it will once more compute information gain to determine the attribute for the next node.

Based on our calculations, it is evident that the attribute providing the most substantial information gain is "Outlook." This attribute will serve as the foundational root of our decision tree.



In Figure 3, we present a visual representation of the decision tree constructed during the initial stage of the ID3 algorithm. Here's a breakdown of the process:

- The training examples are effectively sorted into their respective descendant nodes within the tree structure.
- One of the descendant nodes, labeled as "Overcast," contains only positive instances and, as a result, is transformed into a leaf node with the classification "Yes."
- For the remaining two nodes, a critical question emerges: Which attribute should be chosen for further testing? To address this, we extend these nodes by selecting attributes that offer the highest information gain concerning the new subset of examples.
- The subsequent step involves identifying the attribute that is most suitable for testing within the "Sunny" descendant node.

The Dataset in Figure 1 has the value Sunny on Day1, Day2, Day8, Day9, Day11. So the Sample Space S=5 here.

$$S_{\text{suuny}} = 5 = S$$
$$Humidity = High, Normal$$
$$Humidity_{\text{high}} = [0+, 3-]$$
$$Humidity_{\text{normal}} = [2+, 0-]$$
$$Gain(S, Humidity) = ?$$

$$Gain(S_{\text{sunny}}, Humidity) = Entropy(S) - (\frac{3}{5}Entropy(Humidity_{\text{high}}) + \frac{2}{5}Humidity_{\text{normal}}) \tag{1.15}$$

$$Entropy(Humidity_{\text{high}}) = -(\frac{0}{3}log_2\frac{0}{3} + \frac{3}{3}log_2\frac{3}{3}) = 0 \tag{1.16}$$

$$Entropy(Humidity_{\text{normal}}) = -(\frac{2}{2}log_2\frac{2}{2} + \frac{0}{2}log_2\frac{0}{2}) = 0 \tag{1.17}$$

Put the values in eqtn 1.15

$$\begin{aligned} Gain(S_{\text{sunny}}, Humidity) &= Entropy(S) - (\frac{3}{5}0 + \frac{2}{5}0) \\ &= 0.970 - 0 \\ &= 0.970 \end{aligned} \tag{1.18}$$

We can now measure the information gain of Temperature and Wind by following the same way we measured **Gain(S, Humidity).** Finally, we will get:

$$Gain(S, Humidity) = 0.970$$
$$Gain(S, Temperature) = 0.570$$
$$Gain(S, Wind) = 0.019$$

At this stage, Humidity emerges as the attribute that yields the highest information gain. Therefore, in the "Sunny" descendant node following "Outlook," the attribute chosen is "Humidity."

- The "High" descendant node exclusively contains negative examples, while the "Normal" descendant node exclusively contains positive examples. Consequently, both of these nodes transition into leaf nodes and cannot be expanded further.
- If we apply the same procedure to extend the "Rain" descendant, we find that the attribute "Wind" provides the most information. I'll leave this part for readers to perform the calculations themselves.

As a result, our final decision tree structure aligns with Figure 4:

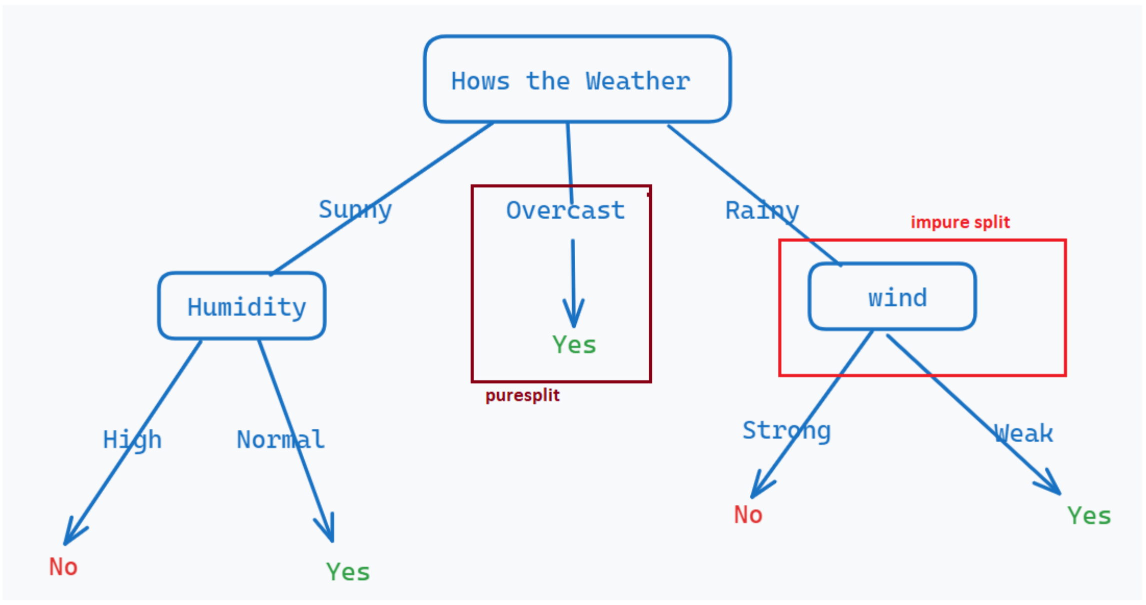


Figure 4:

```
In [9]:  import pandas as pd
         from sklearn.datasets import load_iris
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score
         from matplotlib import pyplot as plt
         from sklearn import tree
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.tree import plot_tree
         import matplotlib.pyplot as plt
```

Iris Dataset With Plane Decision Tree

```
In [10]: iris=load_iris()
```

```
In [11]: x=iris.data
         y=iris.target
```

```
In [12]: y
```

```
Out[12]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
In [13]: X_train, X_test, y_train, y_test =train_test_split(x,y, test_size=0.2, random_state=42)
```

```
In [14]: clf=DecisionTreeClassifier()
         clf.fit(X_train, y_train)
```

```
Out[14]: ▾ DecisionTreeClassifier

         DecisionTreeClassifier()
```

```
In [15]: y_pred=clf.predict(X_test)
```

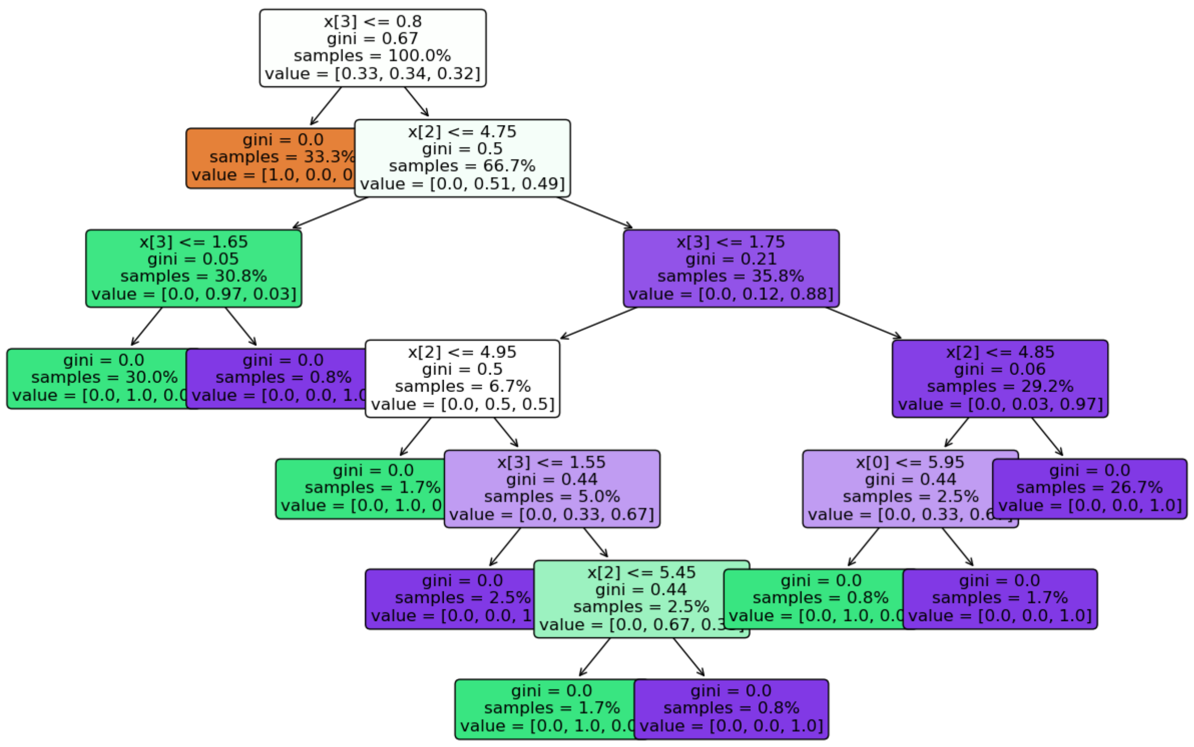
```
In [16]: accuracy = accuracy_score(y_test, y_pred)
         print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

Tree Function Definations

```
In [17]: def display_tree(model):
             # Create a figure with a larger size and set the background color
             plt.figure(figsize=(15, 10))
             plt.rcParams['axes.facecolor'] = 'lightgray'

             # Plot the decision tree with more cosmetics
             plot_tree(
                 model,
                 filled=True,

                 rounded=True,
                 proportion=True,
                 precision=2,
                 fontsize=12,
             )

             # Show the plot
             plt.show()
```

```
In [18]: display_tree(clf)
```



Model Evaluations

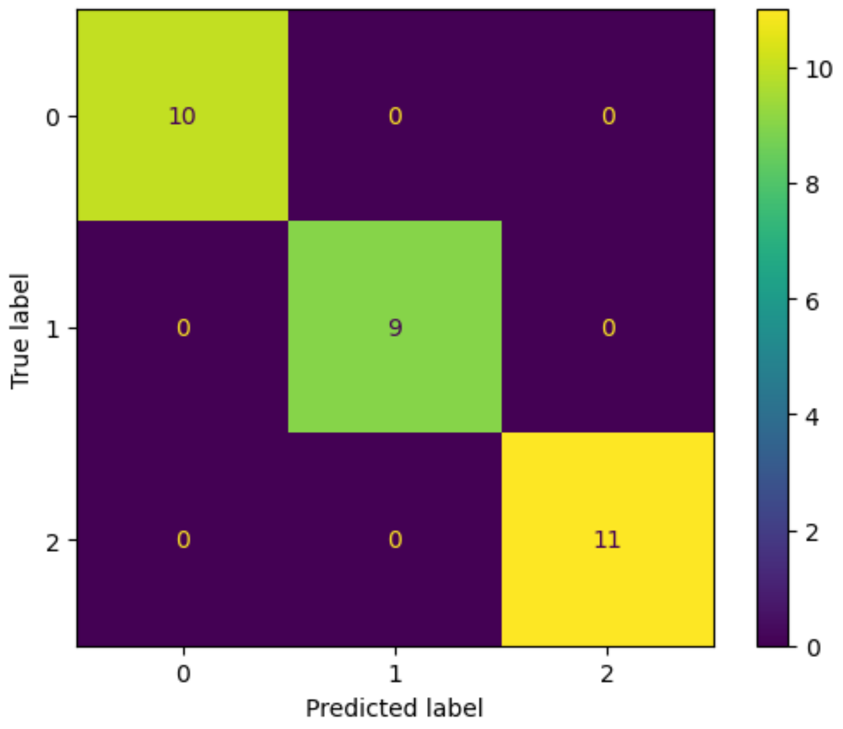
```
In [19]: from sklearn.metrics import confusion_matrix,classification_report, accuracy_score, ConfusionMat
```

Confusion Matrix Function defination

```
In [20]: def confusion_matrix_fun(y_test, y_pred):
             cm=confusion_matrix(y_test, y_pred)
             cm_display=ConfusionMatrixDisplay(confusion_matrix = cm)
             print(classification_report(y_test, y_pred))
             cm_display.plot()
             plt.show()
```

```
In [21]: confusion_matrix_fun(y_test, y_pred)
```

```
              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
```



In [22]:
```python
from sklearn import datasets
dataset_list=[name for name in dir(datasets) if name.startswith('load_')]
for dataset in dataset_list:
    print(dataset)
```

```
load_breast_cancer
load_diabetes
load_digits
load_files
load_iris
load_linnerud
load_sample_image
load_sample_images
load_svmlight_file
load_svmlight_files
load_wine
```

Cancer Dataset with Entropy in Decision tree

In [23]:
```python
from sklearn.datasets import load_breast_cancer
```

In [24]:
```python
cancer=load_breast_cancer()
```

In [25]:
```python
x=cancer.data
y=cancer.target
```

In [26]:
```python
y
```

```
Out[26]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
                1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
                0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
                1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
                0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
                0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
                1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
                1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
In [27]: X_train, X_test, y_train, y_test=train_test_split(x,y, test_size=0.2, random_state=123)
```

```
In [28]: cancer_model=DecisionTreeClassifier(criterion='entropy', splitter='best' )
```

```
In [29]: cancer_model.fit(X_train, y_train)
```

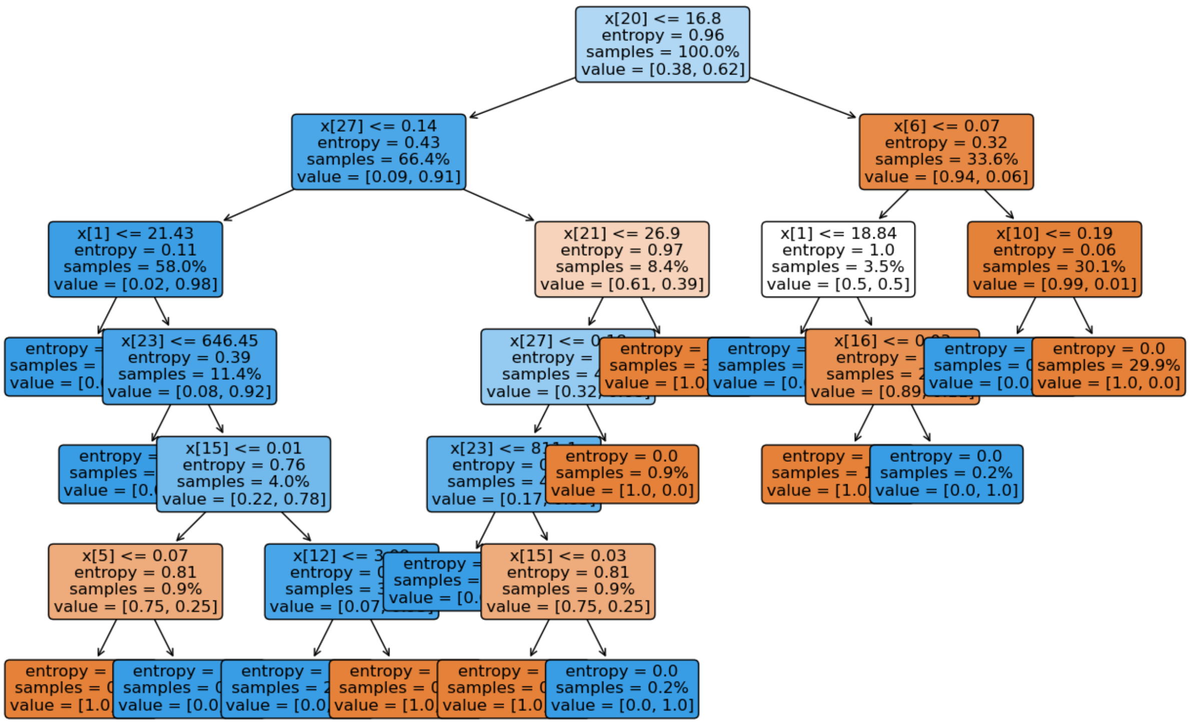
```
Out[29]: ▾          DecisionTreeClassifier

         DecisionTreeClassifier(criterion='entropy')
```

```
In [30]: y_cancer_predict=cancer_model.predict(X_test)
```

Accurecy and Model Evaluations

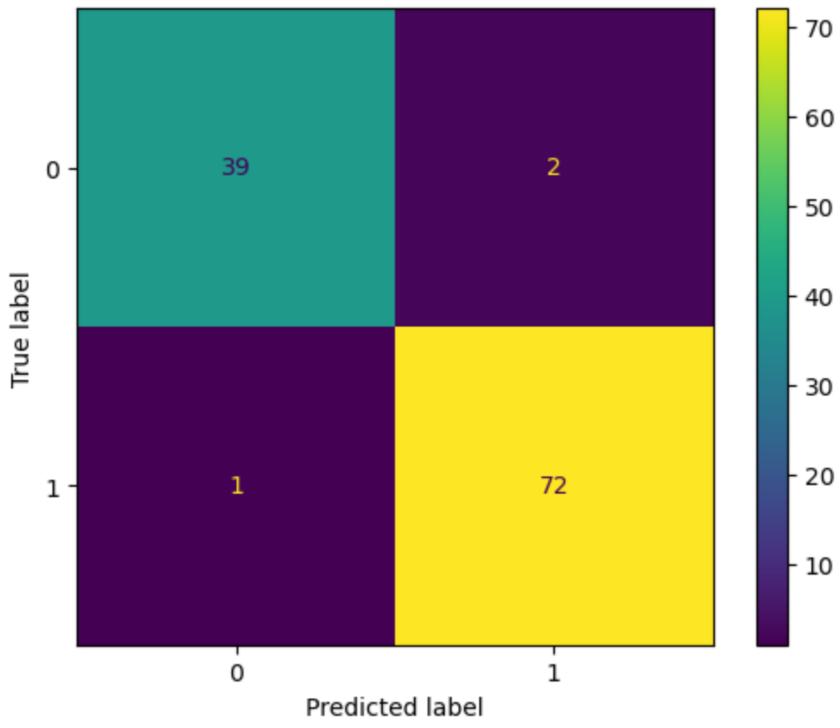
```
In [31]: cancer_accuracy = accuracy_score(y_test, y_cancer_predict)
         print(f"Accuracy: {cancer_accuracy:.2f}")

         Accuracy: 0.97
```

```
In [32]: display_tree(cancer_model)
```



```
In [33]: confusion_matrix_fun(y_test, y_cancer_predict)
```

```
              precision    recall  f1-score   support

           0       0.97      0.95      0.96        41
           1       0.97      0.99      0.98        73

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```



Gini Impurity

Que- How can we determine the optimal feature for partitioning the dataset and what criteria should we use to evaluate the quality of these partitions when constructing a decision tree?

Gini Impurity Formula:

If we have C total classes and p(i) is the probability of picking a datapoint with class I, then the Gini Impurity is calculated as:

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

Now, we have the formula, let's calculate the Gini Impurity for our dataset;

We have 2 classes (0 and 1);

Gini Impurity(df) = 1- $p^2(0)$-$p^2(1)$ = 1–$(5/14)^2$-$(9/14)^2$ = 0.459.

The next step is to calculate the Gini Impurity for the 4 features (outlook, temp, humidity, windy), and decide which feature will be the root node.

Let's calculate the Gini Impurity for Outlook; as you may notice, the outlook feature is a categorical variable, we have three possible values (sunny, overcast, and rainy).

When outlook = sunny, sunny (2yes/3no), for outlook = overcast(4yes/0no) and, finally for outlook = rainy(3yes/2no).

Gini Impurity(outlook = sunny) = 1-$(2/5)^2$-$(3/5)^2$ = 0.48

Gini Impurity(outlook = overcast) = 1-$(4/4)^2$ = 0

Gini Impurity(outlook = sunny) = 1-$(3/5)^2$-$(2/5)^2$ = 0.48

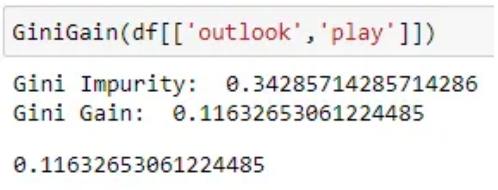
We'll calculate the Gini Impurity of outlook by weighting the impurity of each branch and how many elements it has,

Gini Impurity(outlook) = 5/14 *0.48 + 4/14* 0 + 5/14 * 0.48 = 0.34

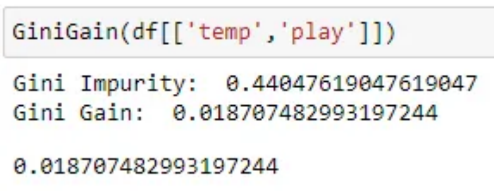
Congratulation! you have just calculated the Gini Impurity for the first feature, to calculate the Gini Gain, which is calculated by subtracting the weighted impurities of the branches from the original impurity.

Gini Gain(outlook) = Gini Impurity(df) — GiniImpurity(outlook)

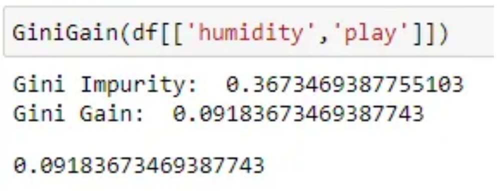
Gini Gain(outlook) = 0.459–0.34 = 0.119



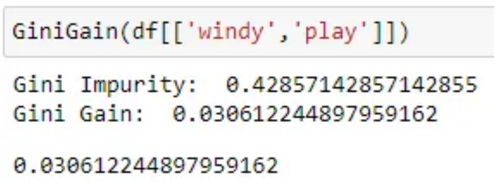
```
GiniGain(df[['outlook','play']])

Gini Impurity:  0.34285714285714286
Gini Gain:  0.11632653061224485

0.11632653061224485
```



```
GiniGain(df[['temp','play']])

Gini Impurity:  0.44047619047619047
Gini Gain:  0.018707482993197244

0.018707482993197244
```



```
GiniGain(df[['humidity','play']])

Gini Impurity:  0.3673469387755103
Gini Gain:  0.09183673469387743

0.09183673469387743
```



```
GiniGain(df[['windy','play']])

Gini Impurity:  0.42857142857142855
Gini Gain:  0.030612244897959162

0.030612244897959162
```

Which feature should I use as a decision node(root node)?

The best split is chosen by maximizing the Gini Gain or by minimizing the Gini Impurity.

In our example, the outlook has the minimum Gini Impurity value and the maximum Gini Gain value, so, It

In our example, the outlook has the minimum Gini impurity value and the maximum Gini Gain value, so, it will be chosen as the root decision to split our data.

Hyperparameters and GridSearchCV

As a data scientist, you're often tasked with building machine learning models to solve complex problems. To make these models perform at their best, you need to understand the concept of hyperparameters and how to optimize them. In this article, we'll break it down in simple terms and introduce you to the power of GridSearchCV, a handy tool to find the best hyperparameters for your models.

What are Hyperparameters?

Hyperparameters are like the dials and switches on a machine learning model, controlling how it learns from data. Unlike the model's parameters (like weights and biases), which it learns from the training data, hyperparameters are set by you before the training begins. Let's explore them in simple language:

- **Criterion:** Think of this as the decision-making principle for your model. It can be either "gini" or "entropy," determining how your model chooses which questions to ask during training.
- **Splitter:** This hyperparameter is all about how the model makes choices. It can "split" by selecting the "best" feature or do it "randomly." Like flipping a coin to decide.
- **Max Depth:** Imagine this as a tree in your backyard. The "max depth" is like deciding how tall this tree can grow. You can set it to a number (like 10), or you can let it grow as tall as it wants (None).
- **Min Samples Split:** This hyperparameter tells the model how many samples need to be at a branch before it splits. It's like saying, "Hey, only split if there are at least 5 apples on this branch."
- **Min Samples Leaf:** Now, think of this as a rule for when to stop growing a branch. It tells the model not to make a new branch if there are fewer than a certain number of samples left.

The Power of GridSearchCV

Finding the right hyperparameters can be like searching for a needle in a haystack. You could guess, but why not use a smarter approach? That's where GridSearchCV comes in:

- **Grid Search:** It's like having a map of the entire haystack, marking specific spots where you think the needle might be. In our case, these spots are different combinations of hyperparameter values.
- **Random Search:** Imagine instead of a map, you randomly drop pins into the haystack. Grid Search checks every marked spot, while Random Search explores some of them, hoping to find the needle faster.
- **Manual Search:** Sometimes, you don't need a map; you know the haystack well. You manually choose where to look for the needle. This is like setting hyperparameters based on your intuition.
- **Bayesian Optimization:** This is like having a detective that learns from previous attempts. It doesn't waste time revisiting spots where the needle isn't. It adapts and focuses on promising areas.
- **Genetic Algorithms:** Think of this as evolution. It starts with a population of possibilities and creates new ones by mixing and mutating them. Over time, it gets closer to finding the best hyperparameters.

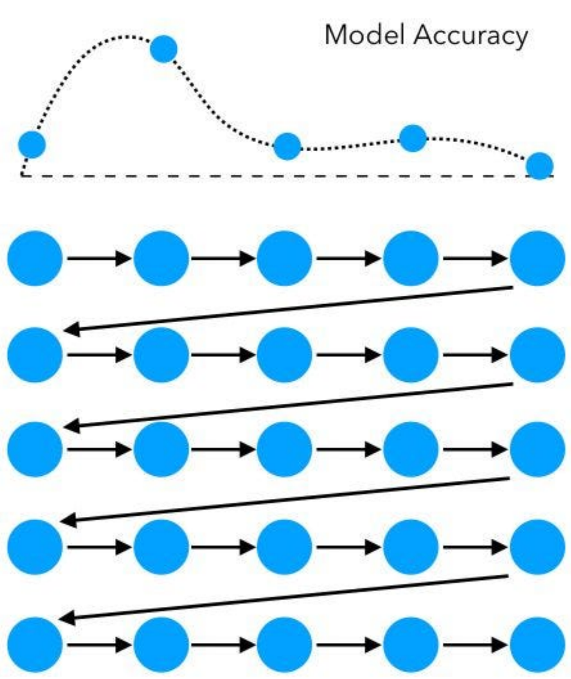
Putting It All Together

Here's how it all works together:

- You define a grid of hyperparameters, setting the values you want to explore.
- GridSearchCV, or one of the other methods, goes through each combination of hyperparameters and trains your model with them.
- It evaluates the model's performance using cross-validation and selects the best set of hyperparameters based on an evaluation metric like accuracy or error.
- Finally, you train your model using the best hyperparameters, making it perform at its peak on your specific data.

Hyperparameter tuning is like finding the best settings for your machine learning model. It's a bit like tuning a musical instrument - finding just the right notes to play. With techniques like GridSearchCV, you can make your models sing beautifully on your data, and that's what makes you a powerful data scientist.

Remember, finding the right hyperparameters is not a one-time thing. It's an iterative process that requires experimentation and fine-tuning. So, keep exploring, keep learning, and keep improving your models to



Play_tennis Dataset with Gini Impurity and grid search

```
In [35]: tennis=play_tennis.drop('day', axis=1)
         tennis
```

Out[35]:

	outlook	temp	humidity	wind	play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Cool	Normal	Strong	Yes
12	Sunny	Mild	Normal	Strong	Yes
13	Rain	Mild	High	Strong	No

Lebel encoder

In [36]:
```
from sklearn.preprocessing import LabelEncoder
```

In [37]:
```
le=LabelEncoder()
```

In [38]:
```
for col in tennis:
    tennis[col]=le.fit_transform(tennis[col])
```

In [39]:
```
tennis
```

Out[39]:

	outlook	temp	humidity	wind	play
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	0	1	0	1
12	2	2	1	0	1
13	1	2	0	0	0

In [40]:
```
y=tennis['play']
```

In [41]:
```
x=tennis.drop('play', axis=1)
```

In [42]:
```
y
```

Out[42]:
```
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     1
10    1
11    1
12    1
13    0
Name: play, dtype: int32
```

In [43]:
```
x
```

Out[43]:

	outlook	temp	humidity	wind
0	2	1	0	1
1	2	1	0	0
2	0	1	0	1
3	1	2	0	1
4	1	0	1	1
5	1	0	1	0
6	0	0	1	0
7	2	2	0	1
8	2	0	1	1
9	1	2	1	1
10	2	2	1	0
11	0	0	1	0
12	2	2	1	0
13	1	2	0	0

In [44]:
```
X_train, X_test, y_train, y_test=train_test_split(x,y, test_size=0.2, random_state=123)
```

In [45]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

In [46]:
```
# Define the parameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

In [47]:
```
tennis_model=DecisionTreeClassifier()
```

```
In [48]: # Use GridSearchCV to find the best hyperparameters
         grid_search = GridSearchCV(tennis_model, param_grid, cv=5)
         grid_search.fit(x, y)

         # Get the best parameters and the best estimator
         best_params = grid_search.best_params_
         best_estimator = grid_search.best_estimator_

         print("Best Parameters:", best_params)

         # Train the model with the best hyperparameters
         best_estimator.fit(x, y)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_spli
t': 5, 'splitter': 'random'}
```

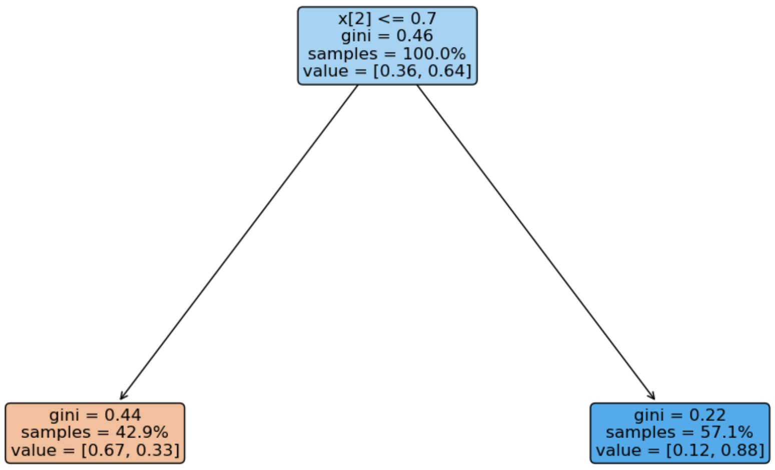
```
Out[48]: ▾                    DecisionTreeClassifier

DecisionTreeClassifier(max_depth=5, min_samples_leaf=4, min_samples_split=5,
                       splitter='random')
```

```
In [49]: tennis_pred_y=best_estimator.predict(X_test)
```

```
In [50]: tennis_accuracy = accuracy_score(y_test, tennis_pred_y)
         print(f"Accuracy: {cancer_accuracy:.2f}")
```

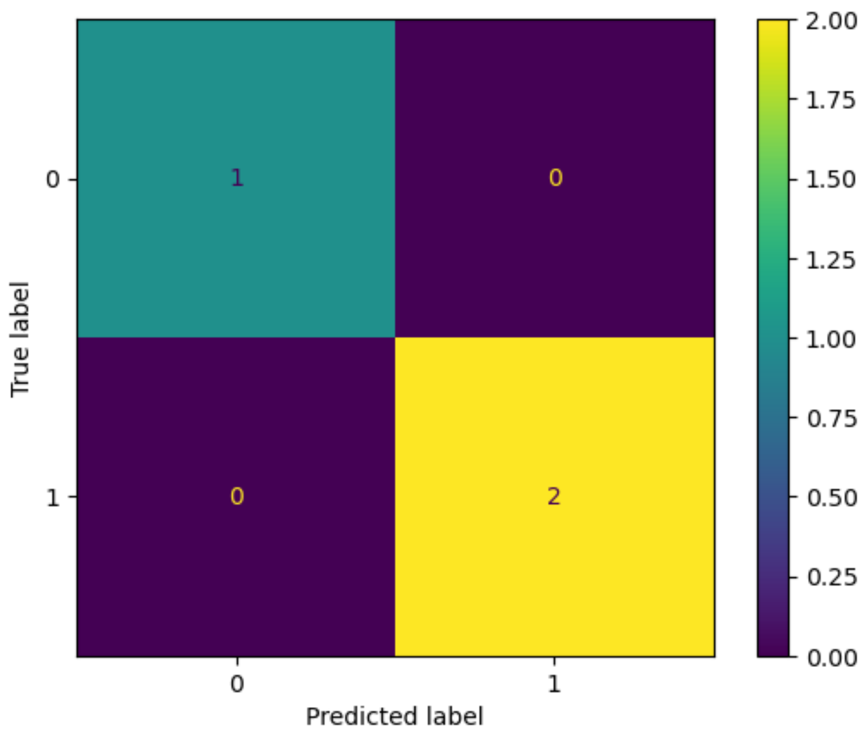
```
Accuracy: 0.97
```

```
In [51]: display_tree(best_estimator)
```



```
In [52]: confusion_matrix_fun(y_test, tennis_pred_y)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```


True label
Predicted label
0
1
0
1
1
0
0
2
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

What Are Regression Trees ?

A regression tree is a machine learning model that is used for regression tasks, where the goal is to predict continuous, numerical values (outputs) rather than discrete categories. It functions similarly to a decision tree, but instead of making categorical decisions, it makes splits and decisions to estimate and predict numeric outcomes.

Mean Square Error

- In Decision Trees for Classification, we learned that the tree makes decisions by asking the right questions at each node to classify data accurately.
- To do this, it uses measures like Entropy and Information Gain. However, when we're predicting continuous values, we can't use the same approach.
- We need a different way to measure how much our predictions differ from the actual target, and that's where the mean square error comes in.

It helps us understand how far off our predictions are from the real values we want to predict.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

In the Regression Tree algorithm:

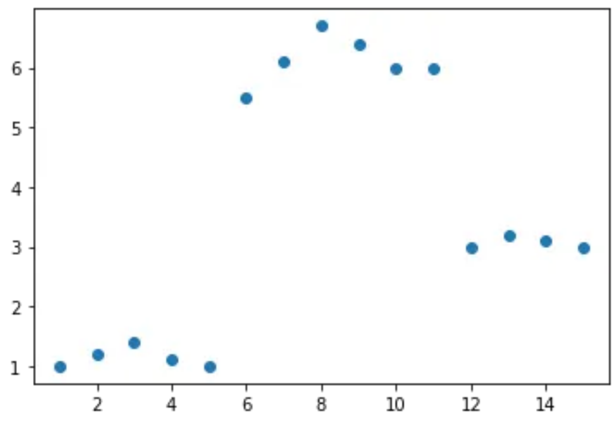
- We have the actual value Y and our prediction Y_hat.
- What we care about is how much our prediction varies from the target, regardless of the direction.
- So, we square the difference between Y and Y_hat.
- Then, we add up all these squared differences for all data points.
- Finally, we divide this sum by the total number of data points to calculate the Mean Square Error (MSE).

In Regression Trees, just like in Classification trees:

- We aim to reduce the Mean Square Error.
- But instead of reducing entropy as in Classification trees, we focus on minimizing the MSE at each child node to improve our predictions for continuous values.

Building a Regression Tree

Let's consider a dataset where we have 2 variables, as shown below



X	Y
1	1
2	1.2
3	1.4
4	1.1
5	1
6	5.5
7	6.1
8	6.7
9	6.4
10	6
11	6
12	3
13	3.2
14	3.1

Source:- https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047

Step 1: Initial Split and Calculation of Predicted Outputs and Mean Square Error

- Sort the data based on X (already sorted in this case).
- Calculate the average of the first 2 rows in variable X, which is (1+2)/2 = 1.5 according to the given dataset.
- Divide the dataset into two parts (Part A and Part B) based on the condition: X < 1.5 and X ≥ 1.5.
- Part A consists of only one point, which is the first row (1,1), and all the other points are in Part B.
- Calculate the average of all Y values in Part A and Part B separately. These two values are the predicted output of the decision tree for X < 1.5 and X ≥ 1.5, respectively.
- Using the predicted and original values, calculate the Mean Square Error (MSE) and note it down.

Step 2: Repeated Split and Mean Square Error Calculation

- Repeat the process for different pairs of rows in sorted X:
- Calculate the average for the second 2 numbers of sorted X, which is (2+3)/2 = 2.5.
- Split the dataset again based on X < 2.5 and X ≥ 2.5 into Part A and Part B.
- Predict outputs and find the Mean Square Error as shown in step 1.
- This process is repeated for the third 2 numbers, the fourth 2 numbers, the 5th, 6th, 7th, and so on until the (n-1)th 2 numbers, where n is the number of records or rows in the dataset.

Step 3: Choosing the Split Point

- Now that we have (n-1) mean squared errors calculated, we need to choose the point at which we are going to split the dataset.
- Choose the point that resulted in the lowest mean squared error upon splitting. In this case, the point is X = 5.5.
- Hence, the tree will be split into two parts: X < 5.5 and X ≥ 5.5.
- The root node is selected this way, and the data points that go towards the left child and right child of the root node are further recursively exposed to the same algorithm for further splitting.

This process creates a decision tree for regression by iteratively finding the best split points based on the lowest Mean Square Error, which helps in making accurate predictions for continuous variables.

Regression Dataset Model Predictions

```
In [53]: df_reg=pd.read_csv('play_tennis_reg.csv')
```

```
In [54]: df_reg=df_reg.drop('day', axis=1)
```

```
In [55]: y=df_reg['Hours played']
```

```
In [56]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()

         for col in df_reg:
             df_reg[col]=le.fit_transform(df_reg[col])
```

```
In [57]: x=df_reg
```

```
In [58]: X_train, X_test, y_train, y_test=train_test_split(x,y, test_size=0.2, random_state=123)
In [59]: reg_model=DecisionTreeClassifier()
```

```
In [60]: reg_model.fit(X_train, y_train)
```

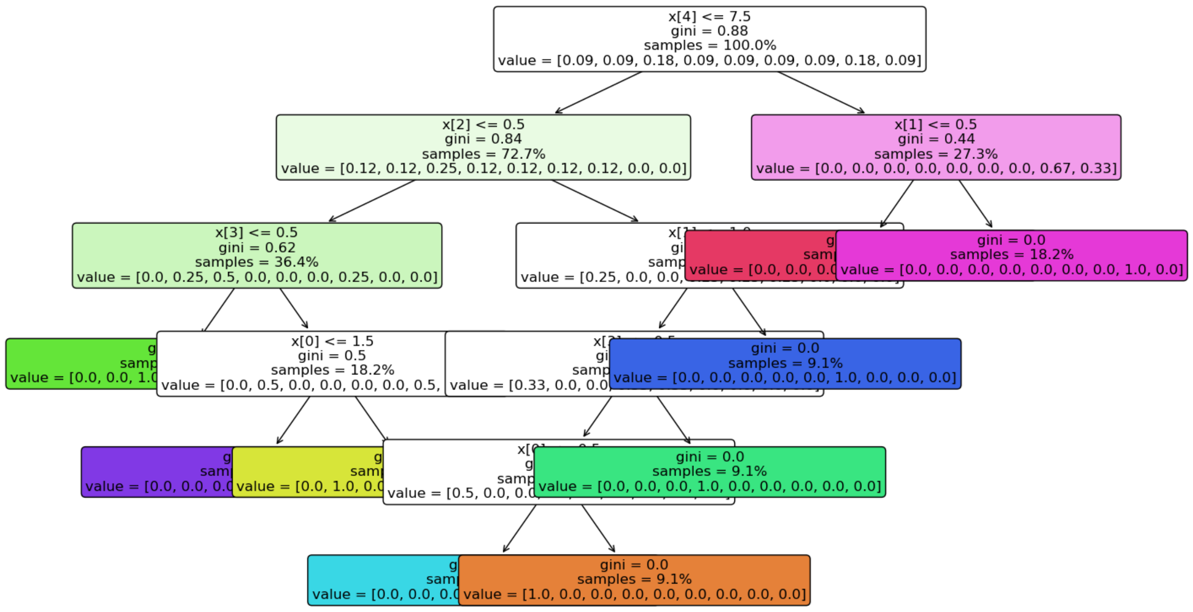
```
Out[60]: ▾ DecisionTreeClassifier
         DecisionTreeClassifier()
```

```
In [61]: y_pred_reg=reg_model.predict(X_test)
```

```
In [62]: cancer_accuracy = accuracy_score(y_test, y_pred_reg)
         print(f"Accuracy: {cancer_accuracy:.2f}")

         Accuracy: 0.67
```

Regression Tree plotted

```
In [63]: display_tree(reg_model)
```

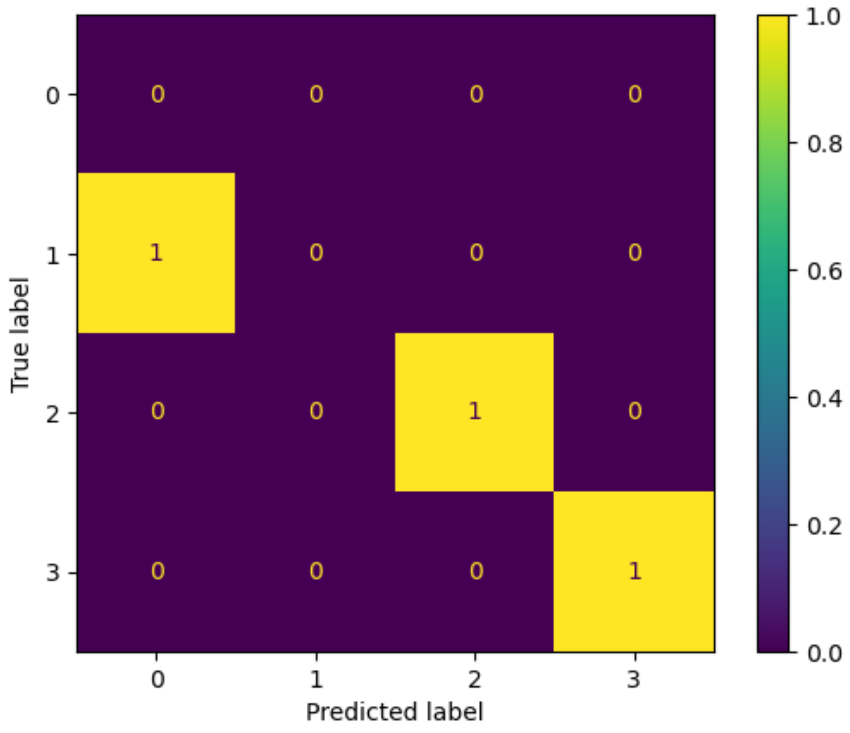


x[4] <= 7.5
gini = 0.88
samples = 100.0%
value = [0.09, 0.09, 0.18, 0.09, 0.09, 0.09, 0.09, 0.18, 0.09]
x[2] <= 0.5
gini = 0.84
samples = 72.7%
value = [0.12, 0.12, 0.25, 0.12, 0.12, 0.12, 0.12, 0.0, 0.0]
x[1] <= 0.5
gini = 0.44
samples = 27.3%
value = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.67, 0.33]
x[3] <= 0.5
gini = 0.62
samples = 36.4%
value = [0.0, 0.25, 0.5, 0.0, 0.0, 0.0, 0.25, 0.0, 0.0]
value = [0.25, 0.0, 0.0,
value = [0.0, 0.0, 0.0
gini = 0.0
samples = 18.2%
value = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
value = [0.0, 0.0, 1.0
x[0] <= 1.5
gini = 0.5
samples = 18.2%
value = [0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5,
value = [0.33, 0.0, 0.0
gini = 0.0
samples = 9.1%
value = [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
value = [0.0, 0.0, 0.0
value = [0.0, 1.0, 0.0
value = [0.5, 0.0, 0.0
gini = 0.0
samples = 9.1%
value = [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
value = [0.0, 0.0, 0.0
gini = 0.0
samples = 9.1%
value = [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Regression confusion matrix plotted

```
In [64]: confusion_matrix_fun(y_test, y_pred_reg)
```

```
C:\Users\prashantkumar.sundge\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:14
69: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prashantkumar.sundge\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:14
69: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels w
ith no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prashantkumar.sundge\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:14
69: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prashantkumar.sundge\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:14
69: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels w
ith no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prashantkumar.sundge\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:14
69: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prashantkumar.sundge\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:14
69: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels w
ith no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

          26       0.00      0.00      0.00         0
          35       0.00      0.00      0.00         1
          48       1.00      1.00      1.00         1
          52       1.00      1.00      1.00         1

    accuracy                           0.67         3
   macro avg       0.50      0.50      0.50         3
weighted avg       0.67      0.67      0.67         3
```



The logic behind the algorithm itself is not rocket science. All we are doing is splitting the data-set by selecting certain points that best splits the data-set and minimises the mean square error. And the way we are selecting these points is by going through an iterative process of calculating mean square error for all the splits and choosing the split that has the least value for the mse. So, It only natural this works.

What happens when there are multiple independent variables ?

- Let us consider that there are 3 variables similar to the independent variable X from fig 2.2.
- At each node, All the 3 variables would go through the same process as what X went through in the above example. The data would be sorted based on the 3 variables separately.
- The points that minimises the mse are calculated for all the 3 variables. out of the 3 variables and the points calculated for them, the one that has the least mse would be chosen.

Referance

www.analyticsvidhya.com

https://medium.com/@jairiidriss

https://scikit-learn.org

https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047