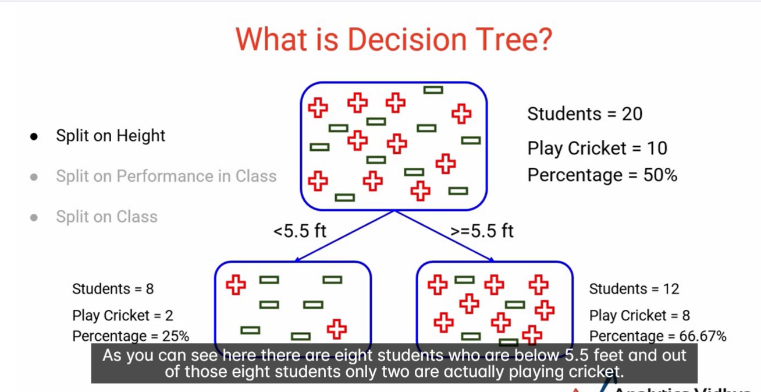
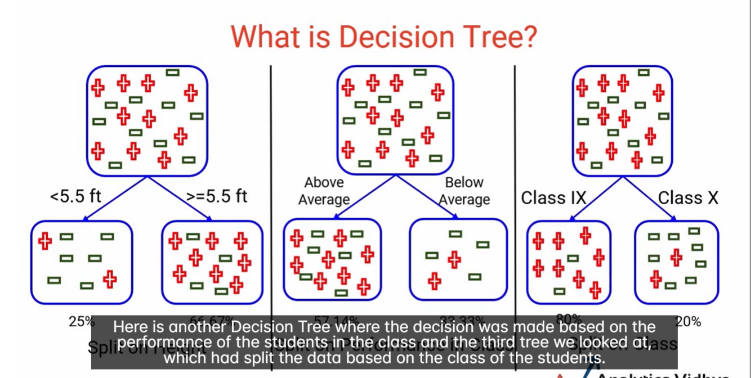
# Decision Trees and Random Forests Tutorials

* Split the group of data samples on values of various features/parameters:



* Every time, we split the data into two subsets based on certain conditions or decisions to get a Decision Tree.

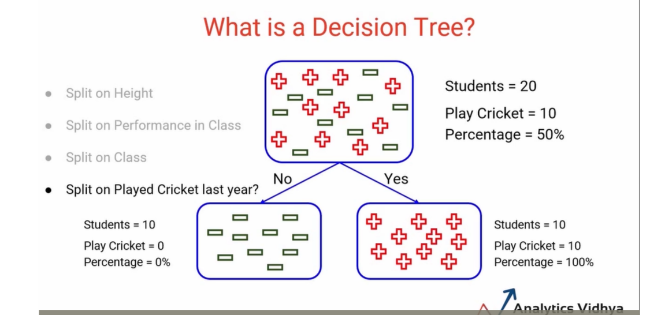


* Which decision tree gives more confidence to the teacher to predict whether students play cricket or not? We want a decision which can best separate the classes (all the positives on one side, and all the negatives in another node in this case of cricket or cricket). The 80%-20% split on the right-hand side here is the optimal split (by Class the students are in), as ‘it has produced almost pure nodes’.

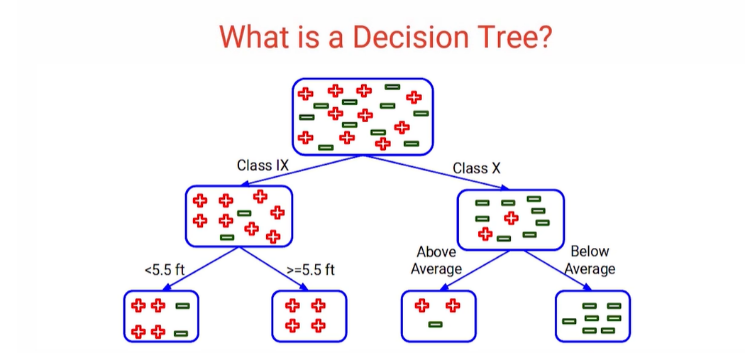
**Node Purity**

* An **ideal split** segregates the positive completely from the negatives (for the above example), which produces **pure nodes**.
* Each node with a decision results in two-subgroups/splits for ‘yes’ and ‘no’.

A perfect split:



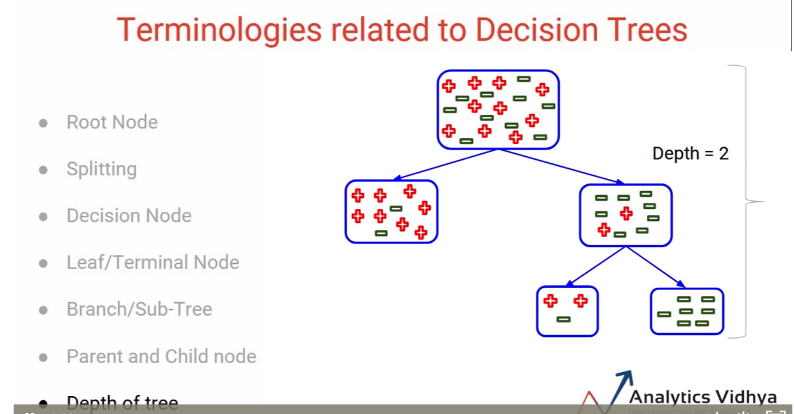
* The objective is to have **pure nodes**.
* After each split, we can split the nodes further and further. Can have multiple splits.



* We can have multiple splits/decisions in a tree.
* The question now: how to decide which split is better? What should be the first, second split etc? What is the sequence in which the splits should be made?
* **There are techniques to decide the Purity of Nodes we can use to determine the best splitting point**.
* E.g. why 5.5 and not 6 feet?
* An ‘ideal node’ has 100% of one class and 0 of other classes.

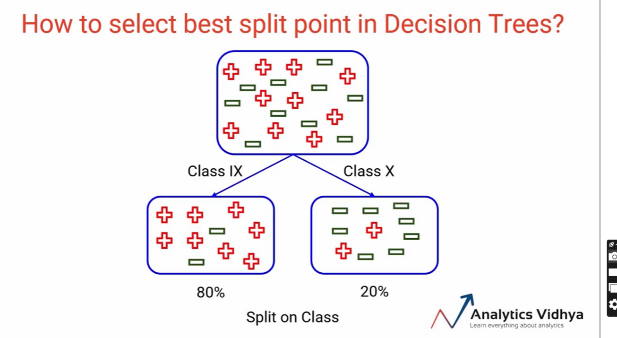
**Terminologies Related to Decision Trees**

* **Root node:** represents the **entire population**/all samples of the data
* Root nodes then get divided into two or more sets, by **splitting**: dividing a node into 2 or more sub-nodes.
* When a sub-node is further divided into even more sub-nodes, the initial sub-node is then called a **decision node**.
* **Leaf/terminal node:** nodes which do not split further.
* **Branch or sub-tree:** A subsection of the whole tree.
* Parent/child nodes: a node divided into sub-nodes is called a parent node, while the sub-nodes are children of the parent nodes.
* Depth of the tree: **length of the longest path from the Root Node to a Leaf Node**.



* With each split, resultant nodes should be as **homogenous as possible**.
* Non-continuous variables can also be used to split the decision trees.

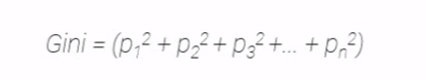
**How to Select the Best Split Point in Decision Trees**



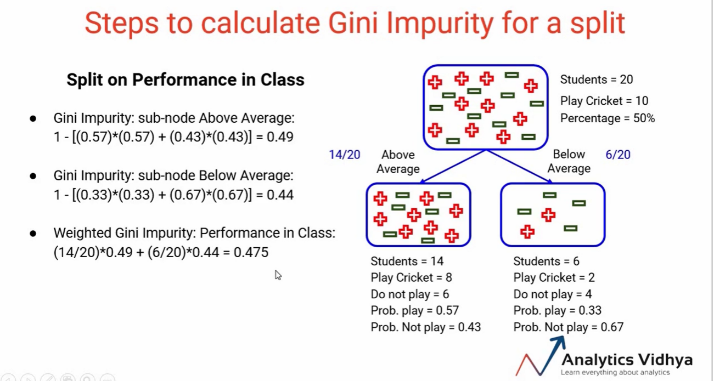
* E.g. in the cricket example, the class a student is in ‘does a better’ job than the other attributes (variables) like height or performance in class, as the resulting nodes/sub-groups are more homogenous.
* If you split on all the available features, then compare which split leads to more homogenous groups, to see which one is the best, is the core concept behind the algorithm:
  + Split the node on **all** available features/variables
  + Select the split which results in the most **homogenous sub-nodes**
  + Homogenous: having similar properties with respect to the problem that we have (e.g. album name)
* If the nodes are entirely pure, each node will only contain a single class 🡪 hence they will be homogenous.
* The greater the homogeneity of a node, the greater its purity.
* There are **multiple** decision tree algorithms to decide the best split for the problem.
* Most common and popular is the Gini Impurity.
* Gini Impurity measure measures the impurity of the nodes and is calculates as ***Gini Impurity = 1 – Gini****.* Gini is used to calculate the right split.
* **What is ‘Gini’?** if you pick two points from the group/population at random, then they **must** be from the same class. Probability that the same two points will be from the same class in a pure node will be 1.0, they’ll always be from same class. That’s what we want to achieve using Gini. Gini ranges from 0 to 1 as it is a probability.
* **Gini = probability two randomly picked samples from a node/sub-group will belong to the same class**.
* **Gini impurity** decides the best split.
* ***Gini Impurity = 1 – Gini***
* Gini = *the purity*
* **The lower the Gini impurity, the higher the homogeneity of the nodes.**
* Works only with **categorical targets** and does not work with regression/continuous targets! Gini is not right for house price predictions. It only performs binary splits, and splits node into two sub-nodes.

Steps to Calculate Gini Impurity

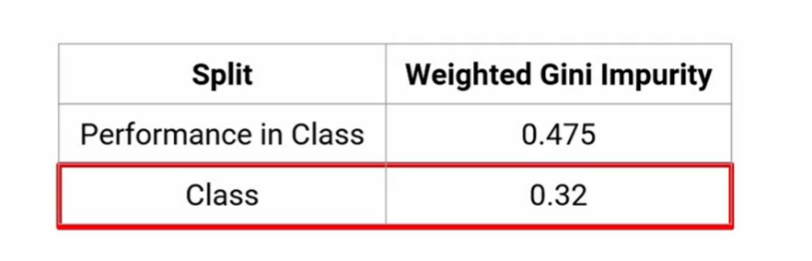
1. Calculate the Gini impurity for the sub-nodes: *Gini Impurity = 1 – Gini* (
2. Gini = Sum of square of probabilities for **each class or category**



1. Considering there are *n* classes.
2. Once you’ve calculated the Gini impurity for sub-nodes, calculate the Gini impurity for the split**, taking the weighted Gini impurity of both sub-nodes of that split**. Here the weight is decided by the number of observational samples in both the nodes.
3. Let’s look at these calculations with an example (split on the performance in a class):



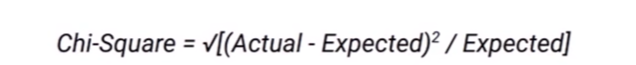
Comparing Splits by Gini Impurity:



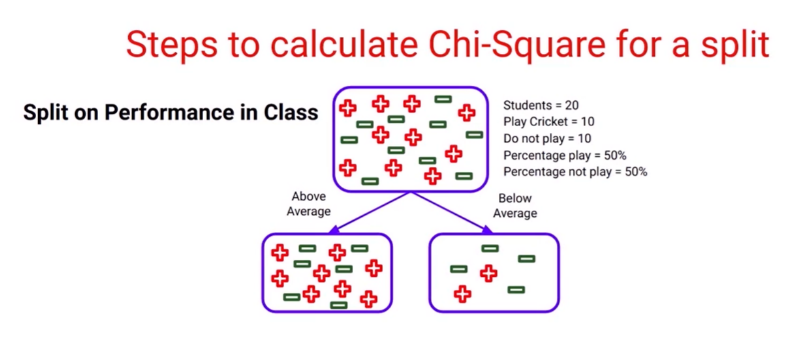
* Select split with minimum Gini impurity.
* Gini can also be used for multi-class classification
* Gini = 1 means nodes are homogenous
* Gini Impurity should be lowest possible for any split.

**Chi-Squared as Alternative Method for Choosing Best Splitting Point**

* How to decide the best split point in Decision Trees?
* Use Chi-Squared: *measures the statistical significance of differences between child nodes & parent nodes.*
* It’s measured as the **sum** of **squared standardized differences** between **actual & expected frequencies** of target variable for each node. It is calculated using the following formula:



* How to calculate ‘expected’ value?



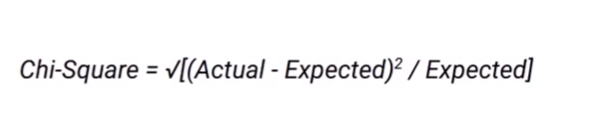
* We consider the ‘Above Average’ Node and count the nr of students, which is 14. The nr of students who play cricket based on the Parent Node is 50%. So we multiply 50% by 14 (nr of students in this child node) to get 7 as the **expected value**. In the Below Average node, there are 6 students. So 50% \* 6 = 3 will be the expected value. Then, if you look at the **actual value** for Above Average performance, it is 8. So now we have both exp and actual values.
* The expected value is based on the distribution of the parent node of the same class.
* Now we can calculate chi squared easily.
* If the actual and expected values are the same, chi-squared will be 0 as actual-expected = 0.
* Therefore, if the parent node perfectly predicts the number of values in the split, the chi-squared will be 0.
* From this, we can generalize to state that the distribution of the child node when chi-squared is 0 is the same as the distribution of the parent node, **thus we are not improving the purity of the nodes**.
* On the other hand, if chi-squared is high, this means the distribution in the child nodes is changing with respect to the parent node based on this split, and we are going in a direction to achieve purer nodes.
* **Thus, the higher the chi-squared value, the greater the purity of nodes after the split.**

**Properties of Chi-Squared Split Selecting Algorithm**

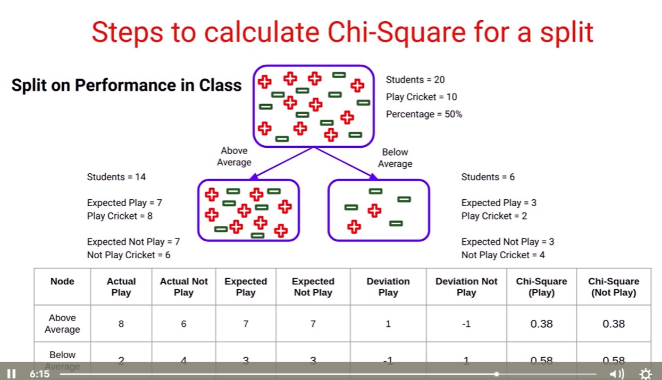
* Works only with categorical target variables.
* Cannot use it for regression.
* The higher the value, means sub-nodes are more different from parent nodes, hence **the homogeneity is greater** and the split performs better.
* You must consider these properties when choosing an algorithm!

**Steps for Calculating Chi-Squared for a Split**

1. Calculate expected values for each class for every child node
2. Calculate chi-squared for each child node using the formula below



1. Calculate chi-squared for split using sum of chi-squared for each child node for that split.



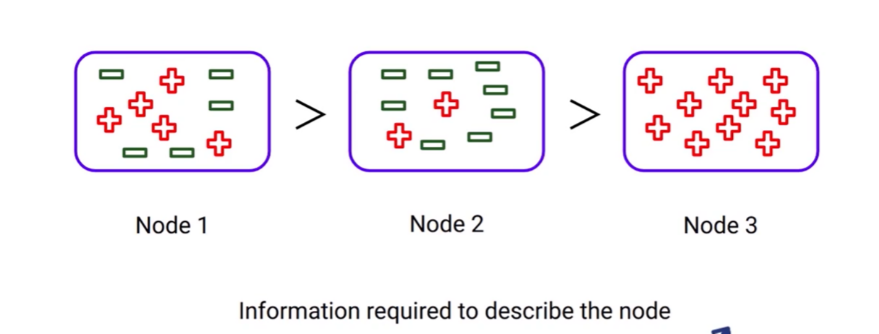
**Total (sum of all chi-squared values)**: 0.38 + 0.38 + 0.58 + 0.58 = 1.92

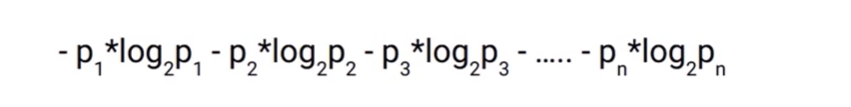
**Summary**

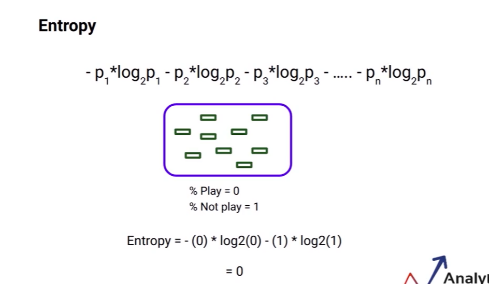
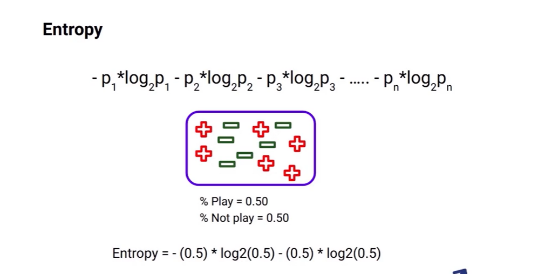
* A high chi-squared value means that
  + Node distribution is very different from the parent node
  + Node is more homogenous than parent
* Chi-squared can be used for multiple categorical labels

What to do next: compare the chi-squared values, and the population will split on the variable with the highest chi-squared score.

**Information Gain Algorithm for Deciding Splits**



* Class the nodes by amount of info you need to describe them
* More impure nodes require ‘more information’ to describe them.
* *Information Gain = 1 – Entropy*
* **Entropy**:
* Considering there are *n* classes.
* Each *p* represents % of that class in the node.

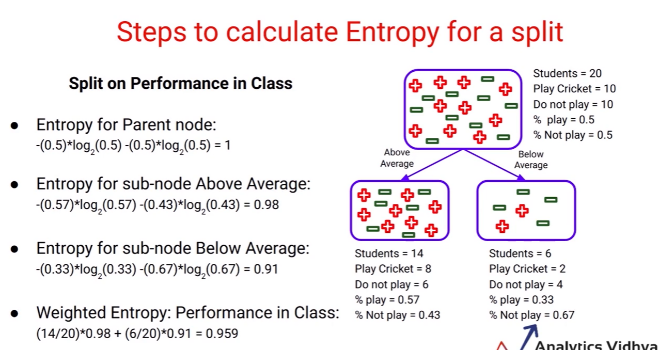


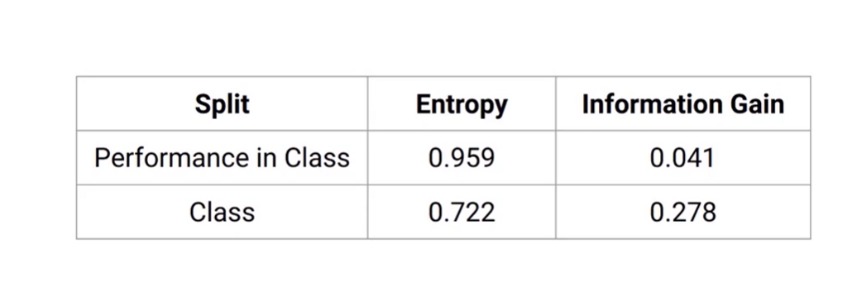
* Low entropy means purer nodes.
* High entropy means less pure nodes.
* Therefore, we want the lowest entropy possible.

**Properties of Entropy**

* Only works with categorical targets
* Lesser the entropy, higher the homogeneity of nodes/purity of nodes

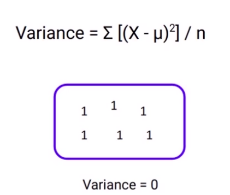
**Steps to Calculate Entropy for a Split**

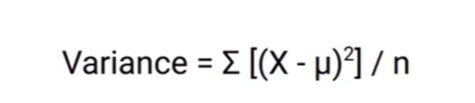
* Calculate entropy of parent node
* Calculate the entropy for each child node
* Calculate the weighted average entropy of the split 🡪 *the weight is the total nr of samples in the child node divided by the nr of samples in the parent node*
* If the weighted entropy of the child nodes is greater than the entropy of the parent node, we will *not* consider that split, as its returning impurer nodes than the parent node.
* The weighted entropy is lower than that of the parent nodes, so we can say it **is** returning purer nodes compared to the parent nodes.
* Then compare the entropy/info gain for 2 splits:



* Entropy is bad, information gain is good because leads to more homogeneity
* So, here, split on Class variable can produce purer nodes.

**Another Algorithm: Reduction in Variance**

* Gini, Chi-Squared and Entropy are used only for categorical variables
* In contract, this algorithm can be used to predict continuous variables



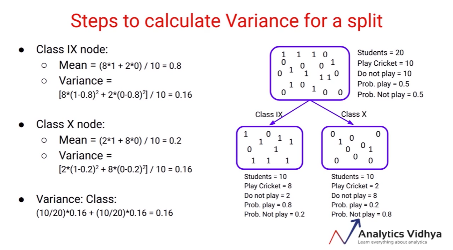
* Lower variance leads to more pure nodes, whereas higher variance leads to more impure nodes
* So, we choose the variable with lower variance for splitting

**Properties of Reduction in Variance**

* Used for continuous target variables
* We calculate the variance for all the splits, and the split with the lowest variance is selected.

**Steps to Calculate Variance for a Split**

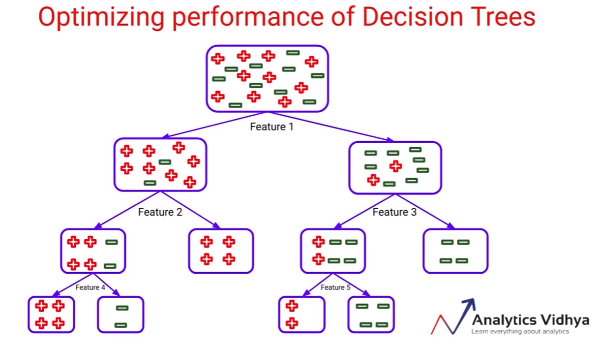
1. Calculate variance of each child node
2. Calculate the variance of the entire split as the weighted average variance of each child node



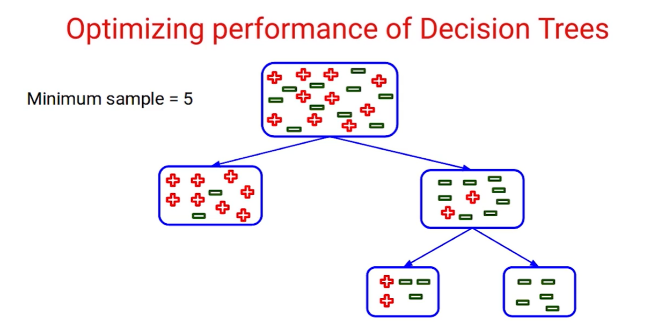
* Choose the split with the lesser variance

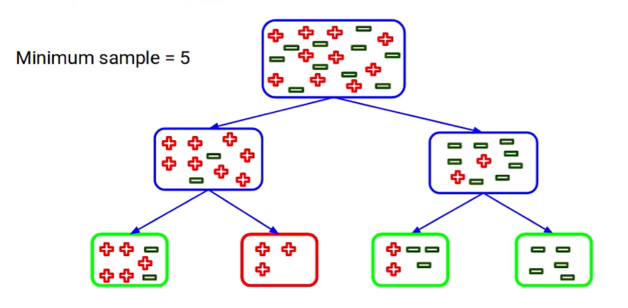
**Optimizing Performance of Decision Trees**

* Trees can keep on growing if there are no constraints on them, until the nodes are completely pure
* We will get 100% training accuracy, perfect score on training data but not very good score on test data.
* Can lead to overfitting!



* Solutions: set threshold to tree-depth, controlling the growth of the tree. 1 level of nodes 🡺 set level to 1. Model will be stopped from learning the patterns if the depth is set too shallow, as this will cause underfitting.
* **More generalizable results tips:**
  + Setting constraints on tree size can be done in many ways

1. **Constraint minimum samples threshold for a node split:** the node will only split into child nodes if it contains a certain threshold nr of samples. E.g. minimums sample for split = 10. If there are more than 10 samples, the node will split further. The node will stop splitting if it has less than 10 samples, so tree will not grow any further. This helps to control overfitting. Too high sample value for splitting will lead to underfitting, and the model won’t learn anything at all. You need to experiment with different values to find the correct one.
2. **Minimum samples for a terminal node:** a node considered as a terminal node if it has a specified minimum number of samples. E.g. if this minimum sample threshold is 5, this node can be considered a terminal node, but if it has less than 5, it cannot be considered as a leaf node, so its parent node will become the leaf node.



* Setting a high value for the node to be a leaf will control overfitting, but again, setting it too high could lead to underfitting problems.

1. Set the depth of the tree and stop splitting once the tree has grown to that defined depth. Higher depth can lead to overfitting, but lower depth will lead to underfitting (the model won’t learn many patterns).
2. Set maximum number of terminal nodes (leaves): if after splitting, we have more nodes than this number, the tree will not split or go beyond that point.

* We need to pick the right value as this is a trade-off between underfitting and overfitting.

**Which Parameters can be Tweaked to Avoid Overfitting?**

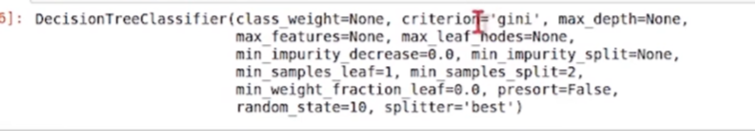
1. Minimum samples for node to split.
2. Minimum samples to become a leaf node.
3. Max depth.
4. Maximum number of leaf nodes allowed.

**What is the maximum number of terminal nodes in a decision tree if our training dataset has N samples?**

N leaf nodes.

**Implementing Decision Tree**

* Titanic data example
* Target labels: 0 as died and 1 as survived
* Other independent variables: in X
* Create train test split: set random state to get the same observations in train and validation sets. *stratify=y* means that you will get **same distribution of classes** in both the training and the test data set. 75% train data.
* **Building a decision tree model:**
  + from sklearn.tree import DecisionTreeClassifier
  + dt\_model = DecisionTreeClassifier(random\_state=12) # use random\_state for reproducible results
  + dt\_model.fit(X\_train, y\_train)
  + dt\_model.score(X\_train, y\_train) *# training score 🡪 returns accuracy of the model (should be high)*
  + dt\_model.score(X\_test, y\_test) *# validation score 🡪 returns accuracy on the validation set*
  + dt\_model.predict(X\_test) *# outputs y predictions*
  + **Can also predict probability of classes:** dt\_model.predit\_proba(X\_test). Outputs probabilities for each class per column. Default threshold is 0.5. Can increase threshold to determine which threshold for putting item into a class should be used.
  + Many parameters can be used to try to optimize the performance, e.g. max depth (length of longest path in the tree). Lets us select the best parameter. If max depth is too low, there will be underfitting. Lets us tune the optimum parameters. E.g. min\_samples\_leaf. Can do multiple for-loops and find the best parameters.
  + Training and validation accuracy should be more in sync. This means model is more generalized.
  + Can loop over different parameter and get the accuracy scores, then put into a DataFrame to compare.
  + criterion=’gini’ is the default but can change it to chi-squared. Also lets you set max\_depth etc. parameter



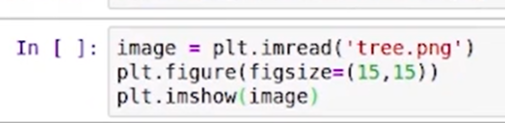
**Plotting a Decision Tree**

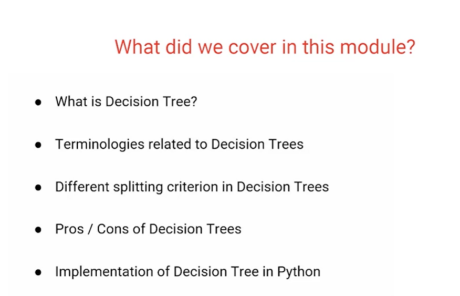
From sklearn import tree

!pip install graphviz

decision\_tree = ***tree.export\_graphviz***(dt\_model, out\_file=’tree.dot’, feature\_names=X\_train.columns, max\_depth=2 ***[of representation, not the real model]***, filled=True)

!dot -Tpng tree.dot -o tree.png *# Save img as readable format*



* Shows the splits
* This visualization makes decision trees easy to interpret
* But, they are prone to overfitting (lots of parameters)
* One way to combat overfitting is random forest

**Grid Search with Decision Trees**

<https://vitalflux.com/decision-tree-hyperparameter-tuning-grid-search-example/>

‘Passing all sets of hyperparameters manually through the model and checking the result might be a hectic work and may not be possible to do.’ - <https://www.projectpro.io/recipes/optimize-hyper-parameters-of-decisiontree-model-using-grid-search-in-python>