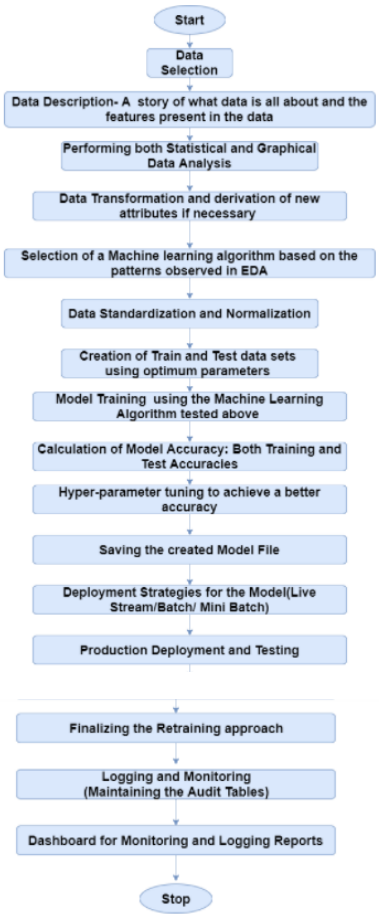
Application Flow

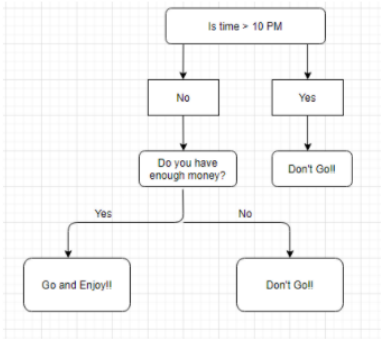
Decision Tree is one of the most fundamental algorithms for classification and regression in the Machine Learning world.

But before proceeding with the algorithm, let's first discuss the lifecycle of any machine learning model. This diagram explains the creation of a Machine Learning model from scratch and then taking the same model further with hyperparameter tuning to increase its accuracy, deciding the deployment strategies for that model and once deployed setting up the logging and monitoring frameworks to generate reports and dashboards based on the client requirements. A typical lifecycle diagram for a machine learning model looks like:



Decision Tree

Decision tree algorithm is one of the most versatile algorithms in machine learning which can perform both classification and regression analysis. It is very powerful and works great with complex datasets. Apart from that, it is very easy to understand and read. That makes it more popular to use. When coupled with ensemble techniques – which we will learn very soon- it performs even better. As the name suggests, this algorithm works by dividing the whole dataset into a tree-like structure based on some rules and conditions and then gives prediction based on those conditions. Let's understand the approach to decision tree with a basic scenario. Suppose it's Friday night and you are not able to decide if you should go out or stay at home. Let the decision tree decide it for you.



Although we may or may not use the decision tree for such decisions, this was a basic example to help you understand how a decision tree makes a decision. So how did it work?

So how did it work?

- It selects a root node based on a given condition, e.g. our root node was chosen as time >10 pm.
- Then, the root node was split into child notes based on the given condition. The right child node in the above figure fulfilled the condition, so no more questions were asked.
- The left child node didn't fulfil the condition, so again it was split based on a new condition.
- This process continues till all the conditions are met or if you have predefined the depth of your tree, e.g. the depth of our tree is 3, and it reached there when all the conditions were exhausted.

Tree Pruning

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Where, T is the subtree which is a subset of the full tree T0 And α is the non-negative tuning parameter which penalises the MSE with an increase in tree length. By using cross-validation, such values of α and T are selected for which our model gives the lowest test error rate. This is how the decision tree regression model works. Let's now see the working algorithm of doing classification using a decision tree. Greedy Algorithm As per Hands-on machine learning book "greedy algorithm greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution."

Post-pruning

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. Cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will make an improvement or not. If any improvement is there then we continue by expanding that node else if there is reduction in accuracy then the node not be expanded and should be converted in a leaf node.

Pre-pruning

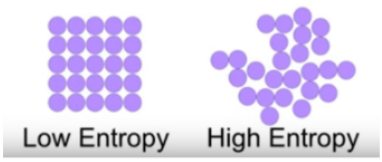
Pre-pruning, also known as forward pruning, stops the non-significant branches from generating. It uses a condition to decide when should it terminate splitting of some of the branches prematurely as the tree is generated.

Classification Trees

Regression trees are used for quantitative data. In the case of qualitative data or categorical data, we use classification trees. In regression trees, we split the nodes based on RSS criteria, but in classification, it is done using classification error rate, Gini impurity and entropy. Let's understand these terms in detail.

Entropy

Entropy is the measure of randomness in the data. In other words, it gives the impurity present in the dataset.



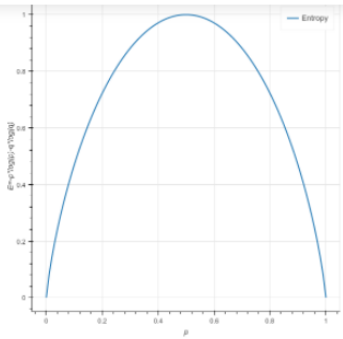
When we split our nodes into two regions and put different observations in both the regions, the main goal is to reduce the entropy i.e. reduce the randomness in the region and divide our data cleanly than it was in the previous node. If splitting the node doesn't lead into entropy reduction, we try to split based on a different condition, or we stop. A region is clean (low entropy) when it contains data with the same labels and random if there is a mixture of labels present (high entropy). Let's suppose there are 'm' observations and we need to classify them into categories 1 and 2. Let's say that category 1 has 'n' observations and category 2 has 'm-n' observations.

p= n/m and q = m-n/m = 1-p

then, entropy for the given set is:

entropy = Σ -p * log2(p)

When all the observations belong to category 1, then p = 1 and all observations belong to category 2, then p =0, int both cases E =0, as there is no randomness in the categories. If half of the observations are in category 1 and another half in category 2, then p =1/2 and q =1/2, and the entropy is maximum, E =1.



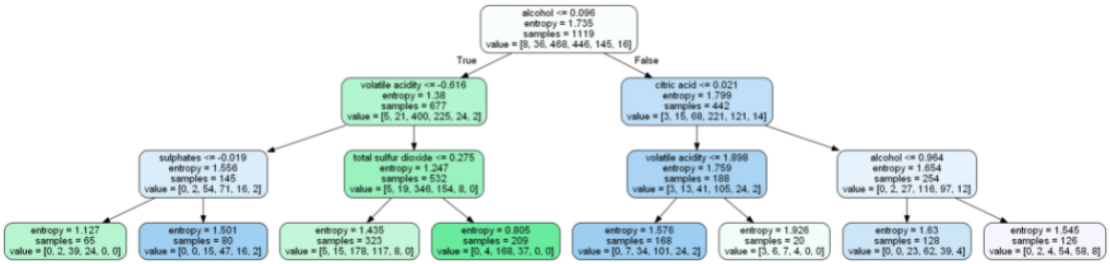
Information Gain

Information gain calculates the decrease in entropy after splitting a node. It is the difference between entropies before and after the split. The more the information gain, the more entropy is removed.

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

Where, T is the parent node before split and X is the split node from T.

A tree which is splitted on basis of entropy and information gain value looks like:

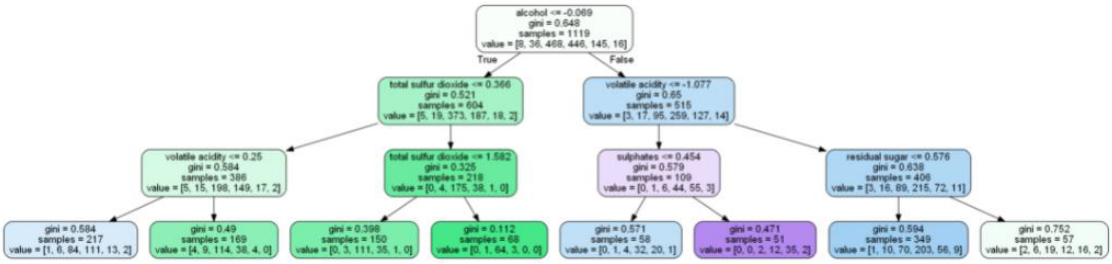


Ginni Impurity

According to wikipedia, 'Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset.' It is calculated by multiplying the probability that a given observation is classified into the correct class and sum of all the probabilities when that particular observation is classified into the wrong class.

Ginni impurity value lies between 0 and 1, 0 being no impurity and 1 denoting random distribution. The node for which the Ginni impurity is least is selected as the root node to split.

A tree which is splitted on basis of ginni impurity value looks like:



Entropy and Information Gain

X feature

- Total ones in 'X' = 3
- Count of Label 'A' when X equal to 1 => 2
- Count of Label 'B' when X equal to 1 => 1

- Total zeros in 'X' = 1
- Count of Label 'A' when X equal to 0 => 0
- Count of Label 'B' when X equal to 0 => 1

entropy = Σ -p * log2(p)

Y feature

- Total ones in 'Y' = 2
- Count of Label 'A' when Y equal to 1 => 2
- Count of Label 'B' when Y equal to 1 => 0

- Total zeros in 'Y' = 2
- Count of Label 'A' when Y equal to 0 => 0
- Count of Label 'B' when Y equal to 0 => 2

Z feature

- Total ones in 'Z' = 2
- Count of Label 'A' when Z equal to 1 => 1
- Count of Label 'B' when Z equal to 1 => 1

- Total zeros in 'Z' = 2
- Count of Label 'A' when Z equal to 0 => 1
- Count of Label 'B' when Z equal to 0 => 1

Finding Information Gain using entropy

Inf.Gain = 1 - Σ sv/s * E

- s = Total records
- sv = Category counts (1/0)
- E = entropy

- entropy_X_1 = 0.9182958340544896
- entropy_X_0 = nan
- entropy_Y_1 = nan
- entropy_Y_0 = nan
- entropy_Z_1 = 1.0
- entropy_Z_0 = 1.0

Gini Indexing

Gini Impurity = 1 - Σ p2

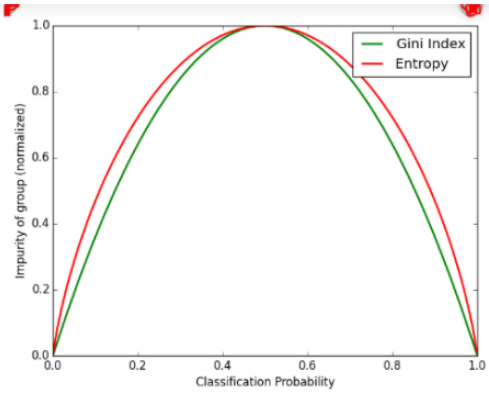
Different Algorithms for Decision Tree

- ID3 (Iterative Dichotomiser) : It is one of the algorithms used to construct decision tree for classification. It uses Information gain as the criteria for finding the root nodes and splitting them. It only accepts categorical attributes.
- C4.5 : It is an extension of ID3 algorithm, and better than ID3 as it deals both continuous and discreet values.It is also used for classfication purposes.

- Classfication and Regression Algorithm(CART) : It is the most popular algorithm used for constructing decison trees. It uses ginni impurity as the default calculation for selecting root nodes, however one can use "entropy" for criteria as well. This algorithm works on both regression as well as classfication problems. We will use this algorithm in our pyhton implementation.

Entropy and Ginni impurity can be used reversibly. It doesn't affects the result much. Although, ginni is easier to compute than entropy, since entropy has a log term calculation. That's why CART algorithm uses ginni as the default algorithm.

If we plot ginni vs entropy graph, we can see there is not much difference between them:



Advantages of Decision Tree:

- It can be used for both Regression and Classification problems.
- Decision Trees are very easy to grasp as the rules of splitting is clearly mentioned.
- Complex decision tree models are very simple when visualized. It can be understood just by visualising.
- Scaling and normalization are not needed.

Disadvantages of Decision Tree:

- A small change in data can cause instability in the model because of the greedy approach.
- Probability of overfitting is very high for Decision Trees.
- It takes more time to train a decision tree model than other classification algorithms.

Implementation in Python

we will use Sklearn module to implement decision tree algorithm. Sklearn uses CART (classification and Regression trees) algorithm and by default it uses Gini impurity as a criteria to split the nodes.

What are hyper parameters?

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

We can see above the decision tree classifier algorithm takes all those parameters which are also known as hyperparameters.

Let's see the most important ones of the parameters(as per sklearn documentation) :

Parameters

- criterion : string, optional (default="gini")

  ```
  The function to measure the quality of a split. Supported criteria are
  "gini" for the Gini impurity and "entropy" for the information gain.
  ```

- splitter : string, optional (default="best") The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
- max_depth : int or None, optional (default=None) The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- min_samples_split : int, float, optional (default=2) The minimum number of samples required to split an internal node:
  - If int, then consider `min_samples_split` as the minimum number.
  - If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

  .. versionchanged:: 0.18 Added float values for fractions.
- min_samples_leaf : int, float, optional (default=1) The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
  - If int, then consider `min_samples_leaf` as the minimum number.
  - If float, then `min_samples_leaf` is a fraction and `ceil(min_samples_leaf * n_samples)` are the minimum number of samples for each node.
- max_features : int, float, string or None, optional (default=None) The number of features to consider when looking for the best split:

Implementation in Python

In [51]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier,export_graphviz
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,roc_curve,roc_auc_score

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
data=pd.read_csv("https://raw.githubusercontent.com/training-ml/Files/main/wine.csv")
data.head()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

In [4]:

```
# Any missing values?

data.isna().sum()
```

Out[4]:

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
Alcohol_content         0
quality                 0
dtype: int64
```

In [19]:

```
data.shape
```

Out[19]:

```
(1599, 13)
```

We have categorical column (Alcohol_Content), we need to convert into numeric data using encoding method

content is following some order like Low,Med,High. We need to give weight according to the order

In [5]:

```
from sklearn.preprocessing import OrdinalEncoder
```

In [6]:

```
ord_encoder=OrdinalEncoder(categories=[['Low','Medium','High']])
df1=ord_encoder.fit_transform(data[['Alcohol_content']])
df1
```

Out[6]:

```
array([[0.],
       [1.],
       [1.],
       ...,
       [2.],
       [1.],
       [2.]])
```

override alcohol content column with codes

In [7]:

```
data['Alcohol_content']=df1
data.head()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

Plotting Heatmap (Correlation matrix)

Let's try to see if we can reduce the features using different techniques.

Let's plot heatmap to visualize and find the coefficient of multicollinearity

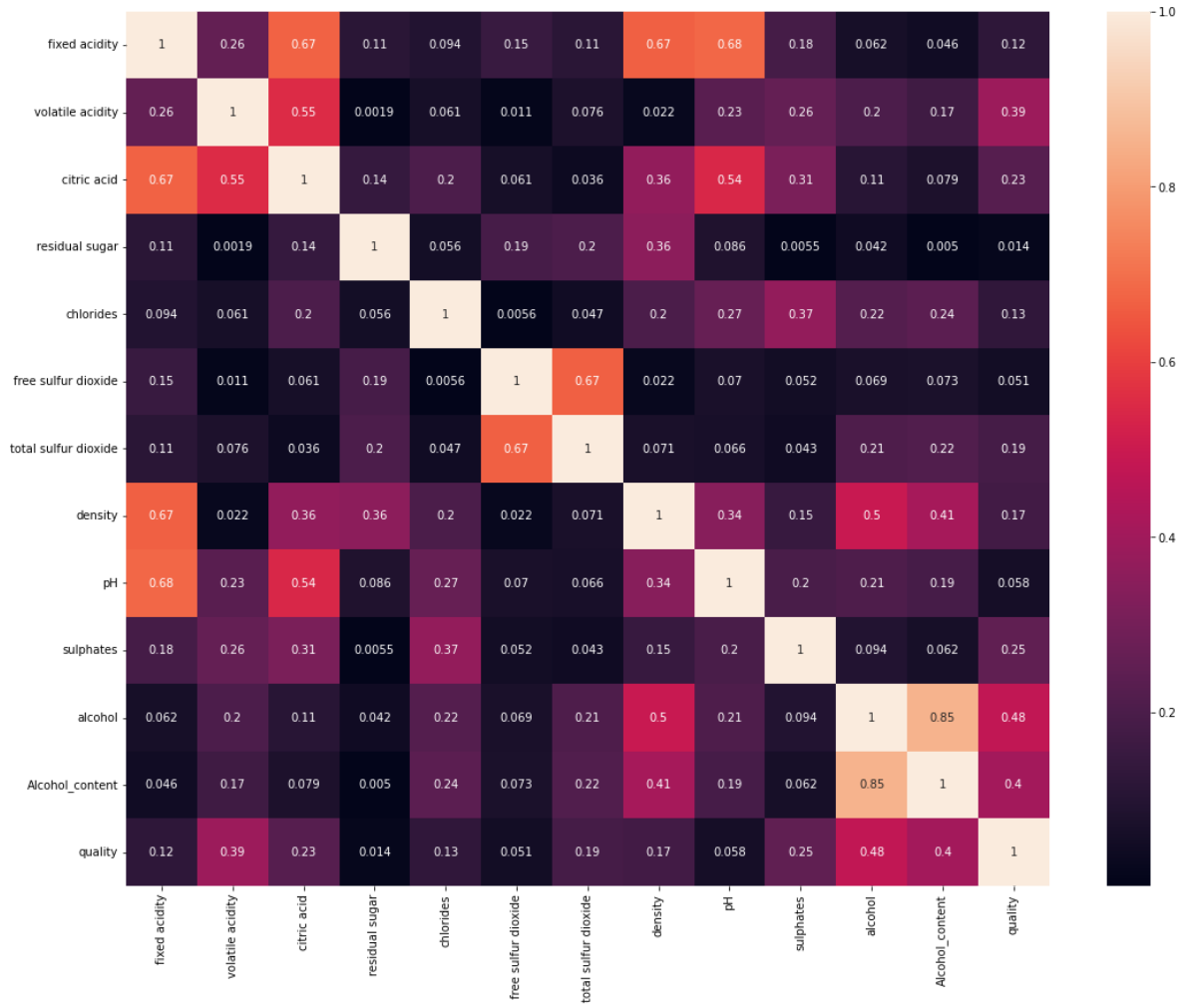
In [10]:

```
df_corr=data.corr().abs()  # This code will get the coefficient of one variable vs all othe

plt.figure(figsize=(18,14))
sns.heatmap(df_corr,annot=True, annot_kws={'size': 10})
```

Out[10]:

<AxesSubplot:>

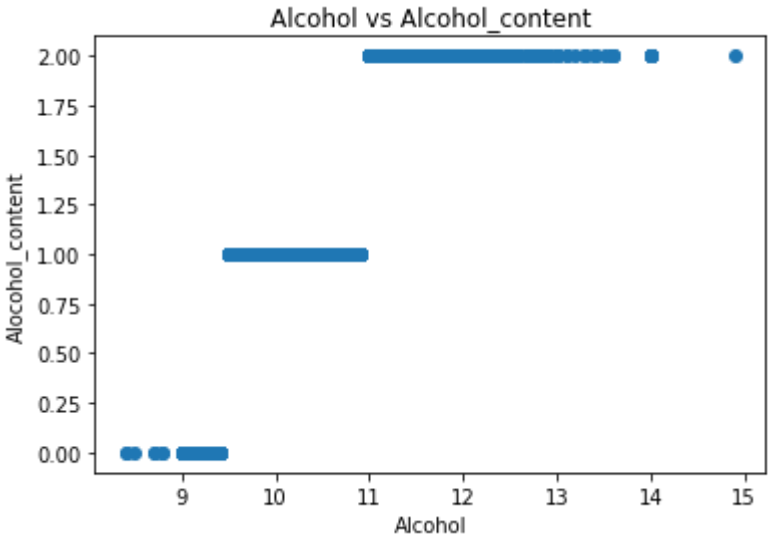


seems like Alcohol_content and alcohol are correlated as per above heat map.But we need more proof

Make sure they are really correlated (It should follow some trend)

In [11]:

```
plt.scatter(data.alcohol,data.Alcohol_content)
plt.xlabel('Alcohol')
plt.ylabel('Alocohol_content')
plt.title('Alcohol vs Alcohol_content')
plt.show()
```



we see clear trend. As and when Alcohol level increases its content also increasing. so we can delete one of them.

In [12]:

```
x= data.drop(columns=['quality','Alcohol_content'])
y= data['quality']
```

In [13]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25, random_state=41)
```

In [14]:

```
# Let's first visualize the tree on the data without doing any pre processing

clf=DecisionTreeClassifier()
clf.fit(x_train,y_train)
```

Out[14]:

```
DecisionTreeClassifier()
```

Let's see hoe the tree looks like (This is nothing to do with algorithm/accuracy). It's just for visualization purpose

In [15]:

```
feature_name=list(x.columns)
class_name=list(y_train.unique())
feature_name
```

Out[15]:

```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

In [18]:

```
import graphviz
from sklearn.tree import export_graphviz
from sklearn import tree
from Ipython.display import image
import pydotplus

#create a dot_file which stores the tree structure
dot_data=export_graphviz(clf,feature_names=feature_name,rounded= True, filled= True)

# draw graph
graph=pydotplus.graph_from_dot_data(dot_data)
graph.write_png("myTree.png")

# show graph
Image(graph.create_png())
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8744/2769373525.py in <module>
----> 1 import graphviz
      2 from sklearn.tree import export_graphviz
      3 from sklearn import tree
      4 from Ipython.display import image
      5 import pydotplus

ModuleNotFoundError: No module named 'graphviz'
```

In [20]:

```
clf.score(x_train,y_train)  # This is Training score
```

Out[20]:

```
1.0
```

In [21]:

```
y_pred=clf.predict(x_test)

# check the accuracy
accuracy_score(y_test,y_pred)
```

Out[21]:

```
0.62
```

Let's now try to tune some hyperparameters using the GridSearchCV algorithm

In [27]:

```
# we are tuning four Important hyperparameters right now, we are passing the different valu

grid_param={
     'criterion':['gini','entropy'],
     'max_depth': range(10,15),                # The maximum depth of the tree
     'min_samples_leaf' : range(2,10),          # The minimumn number of samples required to
     'min_samples_split': range(3,10),         # The minimum number of sample required to sp
     'max_leaf_nodes': range(2,4)              # Best nodes are defined as relative reductio
 }
```

In [28]:

```
grid_search=GridSearchCV(estimator=clf,
                     param_grid=grid_param,

                         cv=5,
                         n_jobs= -1)   # use all the cores in your system. For performance i
```

In [29]:

```
grid_search.fit(x_train,y_train)
```

Out[29]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(10, 15),
                         'max_leaf_nodes': range(2, 4),
                         'min_samples_leaf': range(2, 10),
                         'min_samples_split': range(3, 10)})
```

In [30]:

```
best_parameters=grid_search.best_params_
print(best_parameters)
```

```
{'criterion': 'gini', 'max_depth': 10, 'max_leaf_nodes': 2, 'min_samples_lea
f': 2, 'min_samples_split': 3}
```

In [39]:

```
clf=DecisionTreeClassifier(criterion= 'gini',min_samples_split=3, max_depth=10,min_samples_
clf.fit(x_train,y_train)
```

Out[39]:

```
DecisionTreeClassifier(max_depth=10, max_leaf_nodes=2, min_samples_leaf=2,
                       min_samples_split=3)
```

In [40]:

```
y_pred=clf.predict(x_test)
# Check the accuracy
accuracy_score(y_test,y_pred)
```

Out[40]:

```
0.505
```

Please Note - when you tune all the parameters there is a very good chance that, you can improve the accuracy

since other parameters are default , this combination is not giving better result. you can play around with other numbers to improve the accuracy. Let's visualize the final tree:

In [54]:

```
pip install python-graphviz
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Could not find a version that satisfies the requirement python-graphv
iz (from versions: none)
ERROR: No matching distribution found for python-graphviz
```

In [49]:

```
pip install pydotplus
```

```
Note: you may need to restart the kernel to use updated packages.Collecting
pydotplus

  Downloading pydotplus-2.0.2.tar.gz (278 kB)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\user\anaconda3\l
ib\site-packages (from pydotplus) (3.0.4)
Building wheels for collected packages: pydotplus
  Building wheel for pydotplus (setup.py): started
  Building wheel for pydotplus (setup.py): finished with status 'done'
  Created wheel for pydotplus: filename=pydotplus-2.0.2-py3-none-any.whl siz
e=24575 sha256=afc3e36b15b6aad2b94ed4bd6e256b1dac0497e358022ee19c614f9c98bc5
4d3
  Stored in directory: c:\users\user\appdata\local\pip\cache\wheels\89\e5\de
\6966007cf223872eedfbebbe0e074534e72e9128c8fd4b55eb
Successfully built pydotplus
Installing collected packages: pydotplus
Successfully installed pydotplus-2.0.2
```

In [52]:

```
#Import necessary libraries for graph viz
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import graphviz
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8744/3230783636.py in <module>
      4 from sklearn.tree import export_graphviz
      5 import pydotplus
----> 6 import graphviz

ModuleNotFoundError: No module named 'graphviz'
```

In [53]:

```
feature_name=list(x.columns)
class_name=list(y_train.unique())

# create a dot_file which stores the tree structure
dot_data=export_graphviz(clf,rounded=True,filled=True)

#Draw graph
graph=pydotplus.graph_from_dot_data(dot_data)
graph.write_png("tree_hype.png")
#show graph
Image(graph.create_png())
```

```
---------------------------------------------------------------------------
InvocationException                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8744/627419855.py in <module>
      7 #Draw graph
      8 graph=pydotplus.graph_from_dot_data(dot_data)
----> 9 graph.write_png("tree_hype.png")
     10 #show graph
     11 Image(graph.create_png())

~\anaconda3\lib\site-packages\pydotplus\graphviz.py in <lambda>(path, f, pro
g)
   1808                 lambda path,
   1809                 f=frmt,
-> 1810                 prog=self.prog: self.write(path, format=f, prog=prog
)
   1811             )
   1812

~\anaconda3\lib\site-packages\pydotplus\graphviz.py in write(self, path, pro
g, format)
   1916
   1917             else:
-> 1918                 fobj.write(self.create(prog, format))
   1919         finally:
   1920             if close:

~\anaconda3\lib\site-packages\pydotplus\graphviz.py in create(self, prog, fo
rmat)
   1957             self.progs = find_graphviz()
   1958             if self.progs is None:
-> 1959                 raise InvocationException(
   1960                     'GraphViz\'s executables not found')
   1961

InvocationException: GraphViz's executables not found
```

In [ ]: