**Hello friends,

A decision tree classifier to predict automobile safety. Two models were built, one with the "gini index" criterion and the other with the "entropy" criterion. Implementing the decision tree classification with Python and Scikit-Learn.

Introduction to the decision tree algorithm

The decision tree algorithm is one of the best known machine learning algorithms. Use a tree structure and its possible combinations to solve particular problems. It belongs to the class of supervised learning algorithms where it can be used for both classification and regression purposes.

In machine learning, a decision tree is a predictive model that uses a tree-like structure to make decisions or predictions based on input features. It is a popular and intuitive algorithm for both classification and regression tasks.

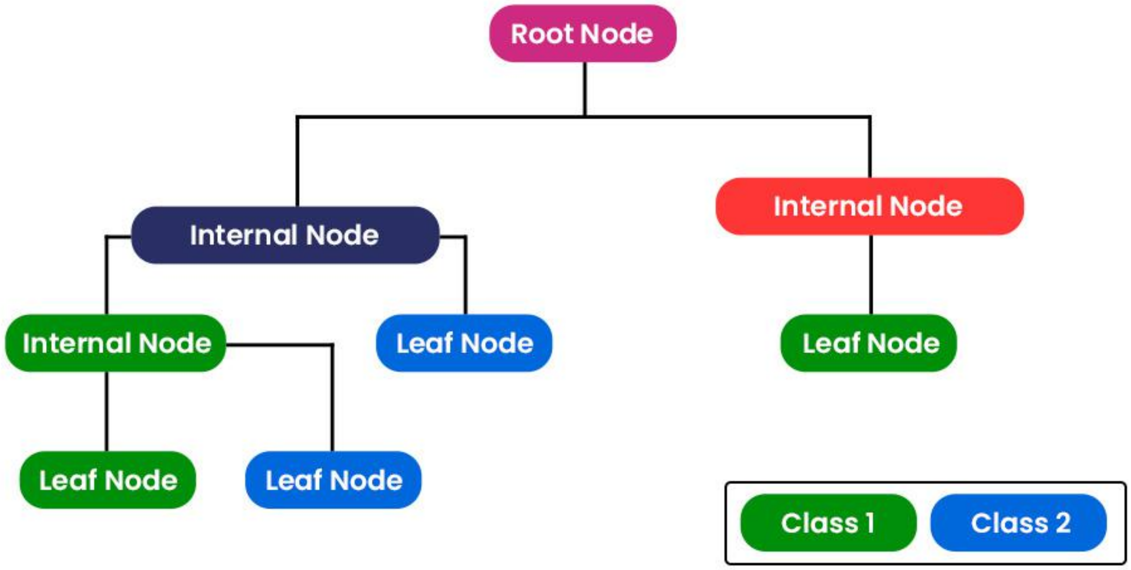
In a decision tree, each internal node represents a feature or attribute, and the branches emanating from the node represent the possible values or ranges of that feature. The leaves of the tree represent the predicted class or regression value. The goal is to learn the optimal decision tree that can accurately predict the target variable based on the input features.

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each inner node denotes a *test* on an attribute, each branch denotes the result of a test, and each leaf node contains a class label. The top node of the tree is the root node..

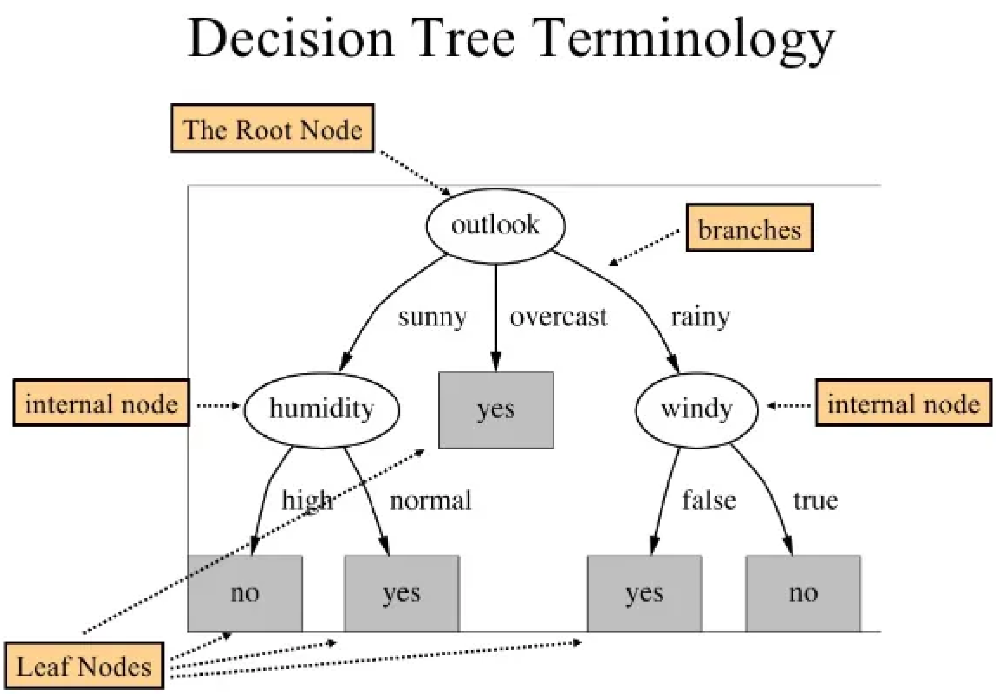
We make some assumptions while implementing the Decision-Tree algorithm which are as follows:

1. At first, the entire training set is considered as the root.
    B. Feature values must be categorical. If the values are continuous, they are discretized (divided into ranks) before building the model.
    C. Records are distributed recursively based on attribute values.
    D. The attributes as root or internal node of the tree, some statistical approach is used.

Classification and Regression Trees (CART)



The CART algorithm provides a foundation for other important algorithms such as packed decision trees, random forests, and boosted decision trees. In this core, I will solve a classification problem. Therefore, I will refer to the algorithm also as a Decision Tree Classification problem.



- In a decision tree algorithm, there is a tree-like structure in which each inner node represents a test on an attribute, each branch represents the result of the test, and each leaf node represents a class label. The paths from the root node to the leaf node represent classification rules.
- We can see that there is some terminology involved in the decision tree algorithm. The terms involved in the decision tree algorithm are as follows:

Root node

- Represents the entire population or sample. This is further divided into two or more homogeneous sets.

Separation

- It is a process of splitting a node into two or more sub-nodes.

Decision node

- When a subnode is split into more subnodes, it is called a decision node.

Leaf/Terminal node

- Nodes that do not split are called Leaf or Terminal nodes.

Pruning

- When we remove subnodes from a decision node, this process is called pruning. It is the opposite process of division.

Branch/Subtree

- A subsection of a complete tree is called a branch or subtree

Children of a Parent node.

- The node of parent and child node, which is divided into subnodes, is called a parent node of subnodes, where the subnodes are the children of a parent node.

When to use the decision tree algorithm

The decision tree algorithm is one of the most frequently used supervised machine learning algorithms and can be used for both classification and regression tasks. The intuition behind the Decision-Tree algorithm is very simple to understand.

The criteria for the use of the decision tree algorithm is the following:

1. For each attribute in the data set, the decision tree algorithm forms a node. The most important attribute is placed at the root node.
2. To evaluate the task at hand, we start at the root node and work our way down the tree following the corresponding node that meets our condition or decision.
3. This process continues until a leaf node is reached. Contains the prediction or result of the decision tree.

Attribute selection measures

The main challenge in implementing the decision tree is to identify the attributes that we consider to be the root node and each level. This process is known as **attribute selection**. There are different attribute selection measures to identify the attribute that can be considered as the root node at each level..

There are 2 popular measures of attribute selection. They are as follows:-

- **Information Gain**
- **Gini Index**

While using **information gain** as the criterion, we assume that the attributes are categorical and for the **Gini index** the attributes are assumed to be continuous. These attribute selection measures are described below

5.1 information gain

[Table of Contents]

Using information gain as a criterion, we try to estimate the information contained by each attribute. To understand the concept of Information Gain, we need to know another concept called **Entropy**.

entropy

Entropy measures the impurity in the given data set. In Physics and Mathematics, entropy refers to the randomness or uncertainty of a random variable X. In information theory, it refers to the impurity in a set of examples. **Information gain** is the decrease in entropy. Information gain calculates the difference between the entropy before the split and the average entropy after the split of the data set based on the given attribute values.

Entropy is represented by the following formula:

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

Here, **c** is the number of classes and **pi** is the probability associated with the its class.

The ID3 (Iterative Dichotomizer) decision tree algorithm uses entropy to calculate the information gini. So, by calculating the decrease in the **entropy measure** of each attribute, we can calculate its information gini. The attribute with the highest information gain is chosen as the split attribute at the node.

5.2 Gini index

[Table of Contents]

Another measure of attribute selection that **CART (Categorical and Regression Trees)** uses is the **Gini Index**. Use the Gini method to create division points.

The Gini index can be represented by the following diagram:-

Gini index

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

Here again **c** is the number of classes and **pi** is the probability associated with the its class.

The Gini index says that if we randomly select two elements from a population, they must be of the same class and the probability of this is 1 if the population is pure.

It works with the categorical objective variable "Success" or "Failure". It only performs binary divisions. The higher the Gini value, the greater the homogeneity. **CART (Classification and Regression Tree) uses the Gini method to create binary splits.**

Steps to calculate the Gini of a split

1. Calcule Gini para los subnodos, usando la fórmula suma del cuadrado de la probabilidad de éxito y fracaso (p^2+q^2).
2. Calculate the Gini for the split using the weighted Gini score of each node in that split.

In the case of a discrete value attribute, the subset that gives the minimum Gini index for the chosen one is selected as the split attribute. In the case of continuous value attributes, the strategy is to select each pair of adjacent values as a possible split point and choose the point with the smallest Gini index as the split point. The attribute with the minimum Gini index is chosen as the division attribute.

6. Overfitting in the decision tree algorithm

Table of Contents

Overfitting is a practical problem when building a decision tree model. The overfitting problem is considered when the algorithm continues to dig deeper and deeper to reduce the error of the training set but results in a larger error of the test set. So, the prediction accuracy of our model decreases. It usually happens when we build many branches due to outliers and irregularities in the data..

Two approaches that can be used to avoid overfitting are as follows:

- Pre-Pruning
- Post-Pruning

Pre-Pruning

In the pre-pruning, we stop the construction of the tree a little earlier. We prefer not to split a node if its goodness measure is below a threshold value. But it's hard to choose a suitable stopping point.

Post-Pruning

In post-pruning, we go deeper and deeper into the tree to build a complete tree. If the tree shows the overfitting problem, pruning is done as a post-pruning step. We use the cross-validation data to verify the effect of our pruning. Using cross-validation data, we test whether expanding a node will result in an improvement or not. If it shows improvement, then we can continue to expand that node. But if it shows a reduction in accuracy, then it should not be expanded. Therefore, the node must become a leaf node.

import libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

import dataset.

In [3]:
```
data=('car_evaluation.csv')
df=pd.read_csv(data,header=None)
```

EDA

In [4]:
```
df.shape
```

Out[4]: (1728, 7)

1. We can see that there are 1728 and 7 variables in the dataset.

Show the first 5 rows of the dataset

In [6]:
```
df.head()
# previwe of the data set
```

Out[6]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

We can see that the data set does not have proper column names. The columns are simply labeled 0,1,2.... and so on. We must give proper names to the columns. I will do it as follows:

Renaming the columns

In [7]:
```
col_names=['buying','maint','door','persons','lug_boot','sefety','class']
df.columns=col_names
col_names
```

Out[7]: ['buying', 'maint', 'door', 'persons', 'lug_boot', 'sefety', 'class']

after renaming the columns names

In [8]:
```
df.head(2)
```

Out[8]:

	buying	maint	door	persons	lug_boot	sefety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc

We can see that the column names are renamed. Now, the columns have meaningful names.

View dataset summary

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   door      1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   sefety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

frequency distribution of values in variable

now check the frequency count of the categorical variables

In [10]:
```
col_names=['buying','maint','door','persons','lug_boot','sefety','class']

for col in col_names :
    print(df[col].value_counts())
```

```
vhigh    432
high     432
med      432
low      432
Name: buying, dtype: int64
vhigh    432
high     432
med      432
low      432
Name: maint, dtype: int64
2        432
3        432
4        432
5more    432
Name: door, dtype: int64
2       576
4       576
more    576
Name: persons, dtype: int64
small    576
med      576
big      576
Name: lug_boot, dtype: int64
low     576
med     576
high    576
Name: sefety, dtype: int64
unacc    1210
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

We can see that doors and people are categorical in nature. So, I'll treat them as categorical variables

Summary of variables

There are 7 variables in the data set. All variables are of categorical data type

- These are given by `purchase`, `maintenance`, `doors`, `people`, `lug_boot`, `security` and `class`.
- `class` is the target variable

Explore variable `class`

In [11]:
```
df['class'].value_counts()
```

Out[11]:
```
unacc    1210
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

The target variable `class` is ordinal in nature.

Missing values in variables

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
buying      0
maint       0
door        0
persons     0
lug_boot    0
sefety      0
class       0
dtype: int64
```

We can see that there are no missing values in the data set. I have checked the frequency distribution of values previously. Also confirms that there are no missing values in the dataset

Declare feature vector and target variable

In [13]:
```
X=df.drop(['class'],axis=1)
y=df['class']
```

Split the data into separate training and test sets

In [14]:
```
# split X and y into training and test sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33, random_state=42)
```

In [15]:
```
# checking the shape of train and test
X_train.shape,X_test.shape
```

Out[15]:
```
((1157, 6), (571, 6))
```

feature Engineering

Feature Engineering is the processof transfroming raw data into useful features that help us better understanding our model and increase its predict power . I will perfrom the feature engineering on different type

first ,i'll preview the data types of variable again

In [16]:
```
# checking data type of X_train
X_train.dtypes
```

Out[16]:
```
buying      object
maint       object
door        object
persons     object
lug_boot    object
sefety      object
dtype: object
```

Encode categorical variables

Now, I'll code the categorical variables.

In [17]:
```
X_train.head()
```

Out[17]:

	buying	maint	door	persons	lug_boot	sefety
48	vhigh	vhigh	3	more	med	low
468	high	vhigh	3	4	small	low
155	vhigh	high	3	more	small	high
1721	low	low	5more	more	small	high
1208	med	low	2	more	small	high

We can see that all the variables are of ordinal categorical type.

In [18]:
```
# import category encoders

import category_encoders as ce
```

In [19]:
```
X_train.head()
```

Out[19]:

	buying	maint	door	persons	lug_boot	sefety
48	vhigh	vhigh	3	more	med	low
468	high	vhigh	3	4	small	low
155	vhigh	high	3	more	small	high
1721	low	low	5more	more	small	high
1208	med	low	2	more	small	high

In [20]:
```
# encode variables with  ordinal encoding
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'door', 'persons', 'lug_boot', 'sefety'])


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [21]:
```
X_train.head()
```

Out[21]:

	buying	maint	door	persons	lug_boot	sefety
48	1	1	1	1	1	1
468	2	1	1	2	2	1
155	1	2	1	1	2	2
1721	3	3	2	1	2	2
1208	4	3	3	1	2	2

In [22]:
```
X_test.head()
```

Out[22]:

	buying	maint	door	persons	lug_boot	sefety
599	2	2	4	3	1	2
1201	4	3	3	2	1	3
628	2	2	2	3	3	3
1498	3	2	2	2	1	3
1263	4	3	4	1	1	1

We now have training and test set ready for model building.

Decision Tree Classifier with Criterion Gini Index

In [23]:
```
# import decision tree classification
from sklearn.tree import DecisionTreeClassifier
```

In [24]:
```
# instantites the decisionTreeClssifer with criterion gini index
```

In [25]:
```
clf_gini=DecisionTreeClassifier(criterion='gini' ,max_depth=3,random_state=0)
clf_gini.fit(X_train,y_train)
```

Out[25]:
```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

Predict Test Set Results Using the Criterion Gini Index

In [26]:
```
y_pred_gini=clf_gini.predict(X_test)
```

In [27]:
```
from sklearn.metrics import accuracy_score
```

Check the accuracy score with the criteria gini index

In [28]:
```
print("Model accuracy score with criterion gini index {0:0.4f}".format(accuracy_score(y_test,y_
```

```
Model accuracy score with criterion gini index 0.8021
```

Here, **y_test** are the true class labels and **y_pred_gini** are the predicted class labels in the test set.

Compare the accuracy of the training set and the test set

Now I will compare the accuracy of the train set and the test set to check for overfitting.

In [29]:
```
y_pred_train_gini=clf_gini.predict(X_train)
y_pred_train_gini
```

Out[29]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

In [30]:
```
print('Training Set Accuracy Score {0:0.4f}'.format(accuracy_score(y_train,y_pred_train_gini)))
```

```
Training Set Accuracy Score 0.7865
```

Check for Overfitting and underfitting

In [31]:
```
# print the score on training and test set
print('training set score {:.4f}'.format(clf_gini.score(X_train,y_train)))
print('test set scorw {:.4f}'.format(clf_gini.score(X_test,y_test)))
```

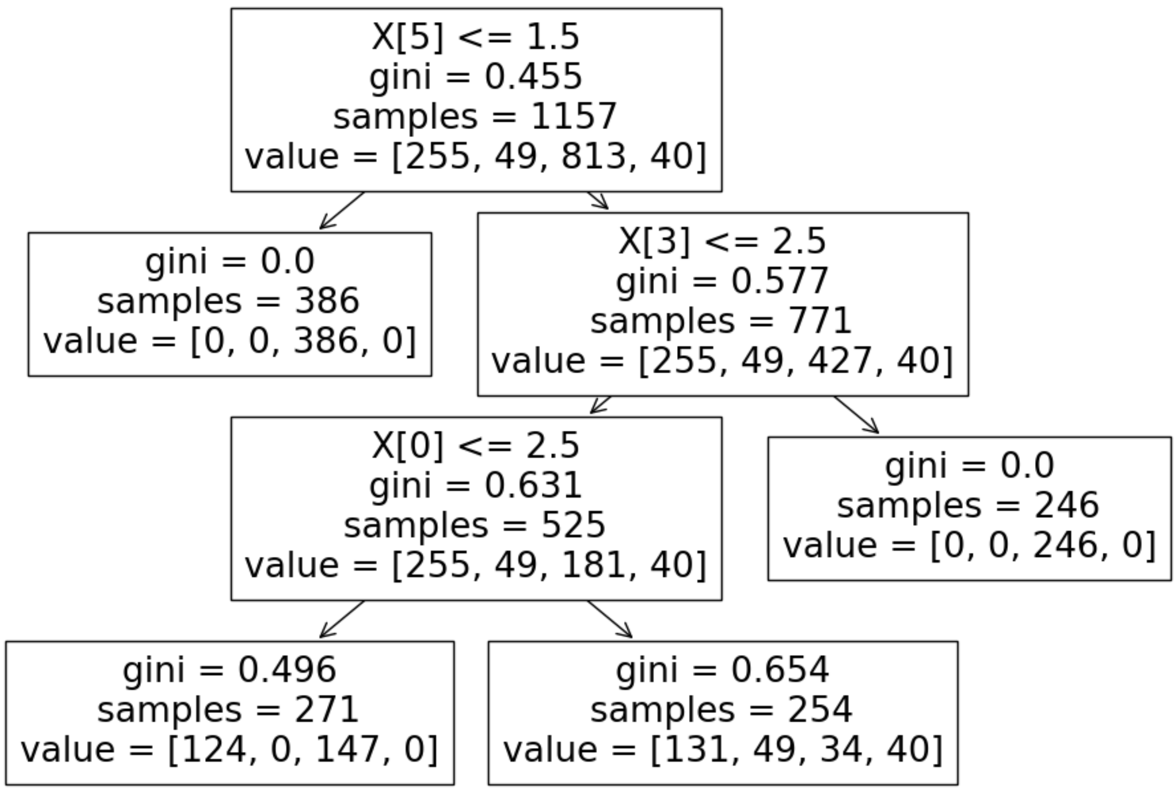
```
training set score 0.7865
test set scorw 0.8021
```

Here, the accuracy score of the training set is 0.7865, while the accuracy of the test set is 0.8021. These two values are quite comparable. Therefore, there are no signs of overfitting.

visualization decision tree

In [32]:
```
plt.figure(figsize=(12,8))
from sklearn import tree
tree.plot_tree(clf_gini.fit(X_train,y_train))
```

Out[32]:
```
[Text(0.4, 0.875, 'X[5] <= 1.5\ngini = 0.455\nsamples = 1157\nvalue = [255, 49, 813, 40]'),
 Text(0.2, 0.625, 'gini = 0.0\nsamples = 386\nvalue = [0, 0, 386, 0]'),
 Text(0.6, 0.625, 'X[3] <= 2.5\ngini = 0.577\nsamples = 771\nvalue = [255, 49, 427, 40]'),
 Text(0.4, 0.375, 'X[0] <= 2.5\ngini = 0.631\nsamples = 525\nvalue = [255, 49, 181, 40]'),
 Text(0.2, 0.125, 'gini = 0.496\nsamples = 271\nvalue = [124, 0, 147, 0]'),
 Text(0.6, 0.125, 'gini = 0.654\nsamples = 254\nvalue = [131, 49, 34, 40]'),
 Text(0.8, 0.375, 'gini = 0.0\nsamples = 246\nvalue = [0, 0, 246, 0]')]
```



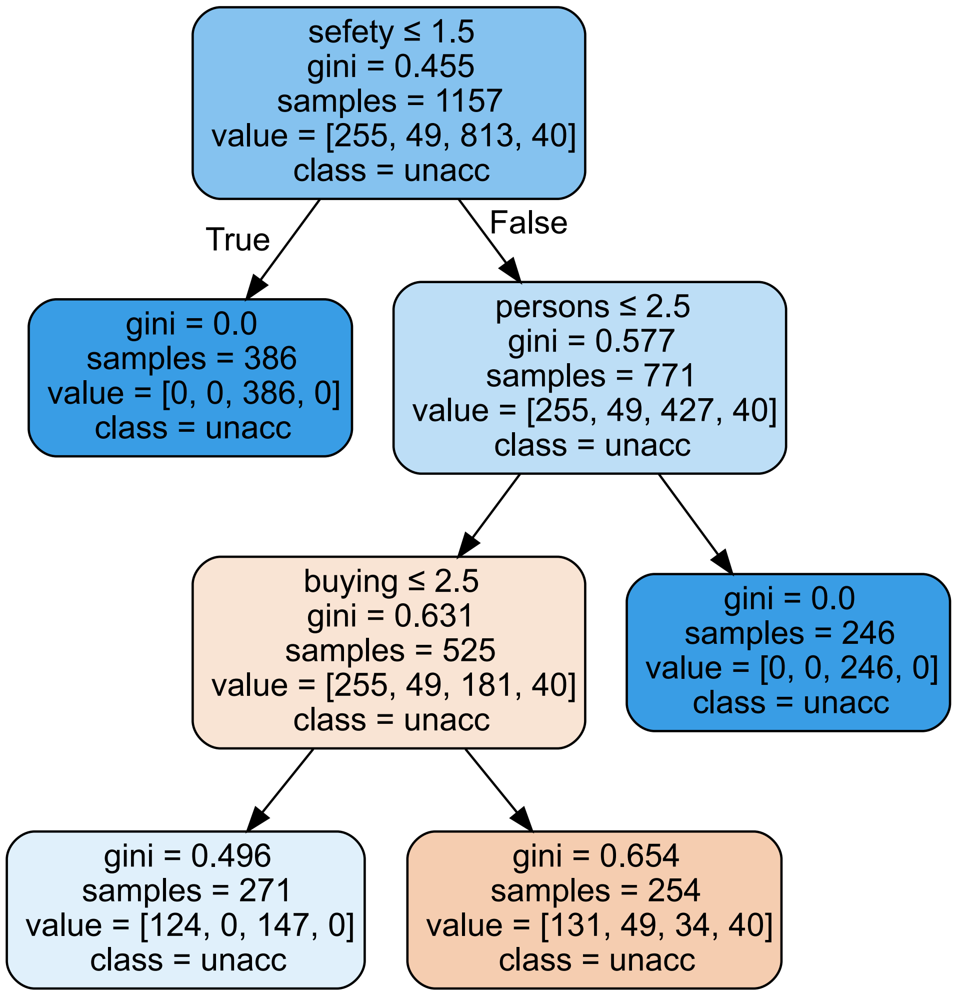
Visualize decision trees with graphviz

In [33]:
```
import graphviz
dot_data =tree.export_graphviz(clf_gini, out_file=None,
                              feature_names=X_train.columns,
                              class_names=y_train,
                              filled=True, rounded=True,
                              special_characters=True)

graph=graphviz.Source(dot_data)

graph
```

Out[33]:



Decision Tree Classifier with Criterion Entropy

In [34]:
```
# instantiate the DecisionTreeClassifier model with criterion entropy
clf_en=DecisionTreeClassifier(criterion='entropy',max_depth=3,random_state=0)

# fit the model
clf_en.fit(X_train,y_train)
```

Out[34]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

Predict Test Set Results with Criterion Entropy

In [35]:
```
y_pred_en = clf_en.predict(X_test)
```

Compare the accuracy of the train set and the test set

Now I will compare the accuracy of the **train set** and **test set** to check for overfitting.

In [36]:
```
y_pred_train_en=clf_en.predict(X_train)
y_pred_train_en
```

Out[36]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

In [37]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train_en))
```

```
Training-set accuracy score: 0.7865
```

Check for Overfit and Underfit

In [38]:
```
# print the score on training and test set
print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

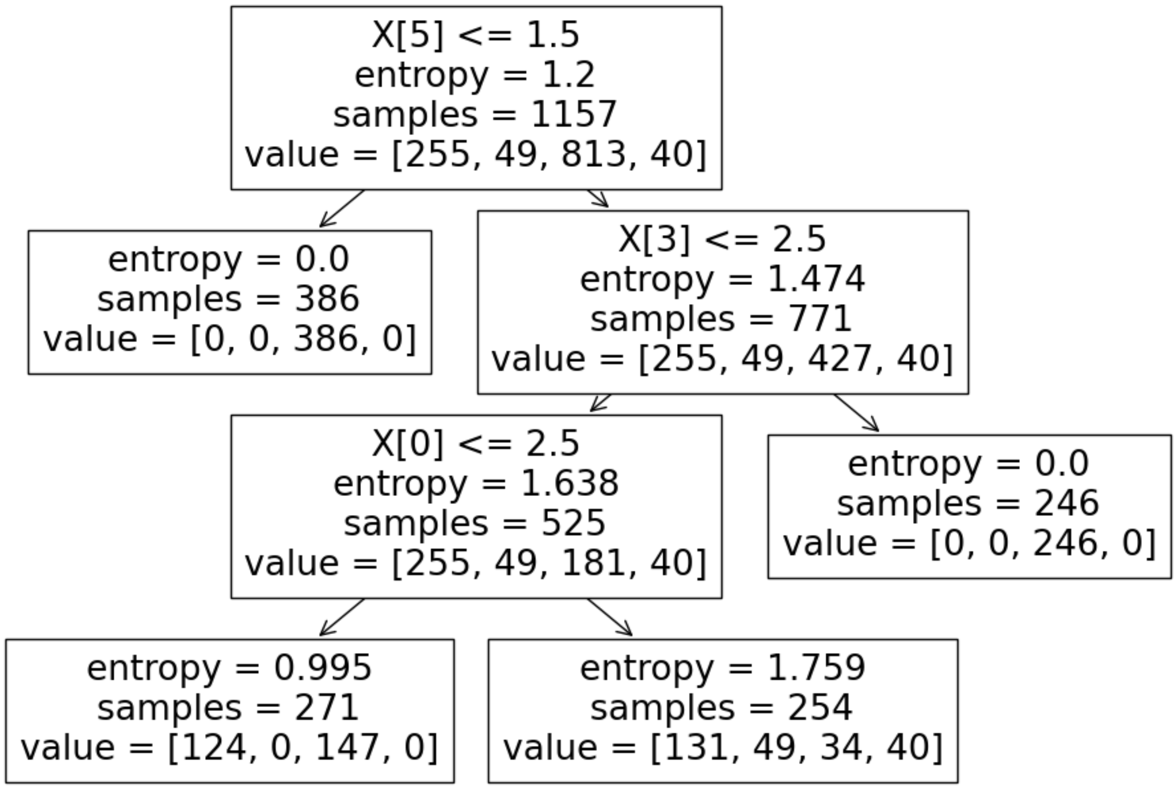
```
Training set score: 0.7865
Test set score: 0.8021
```

We can see that the training set score and the test set score are the same as above. The accuracy score of the training set is 0.7865, while the accuracy of the test set is 0.8021. These two values are quite comparable. Therefore, there are no signs of overfitting.

Visualize decision trees

In [39]:
```
plt.figure(figsize=(12,8))
from sklearn import tree
tree.plot_tree(clf_en.fit(X_train,y_train))
```

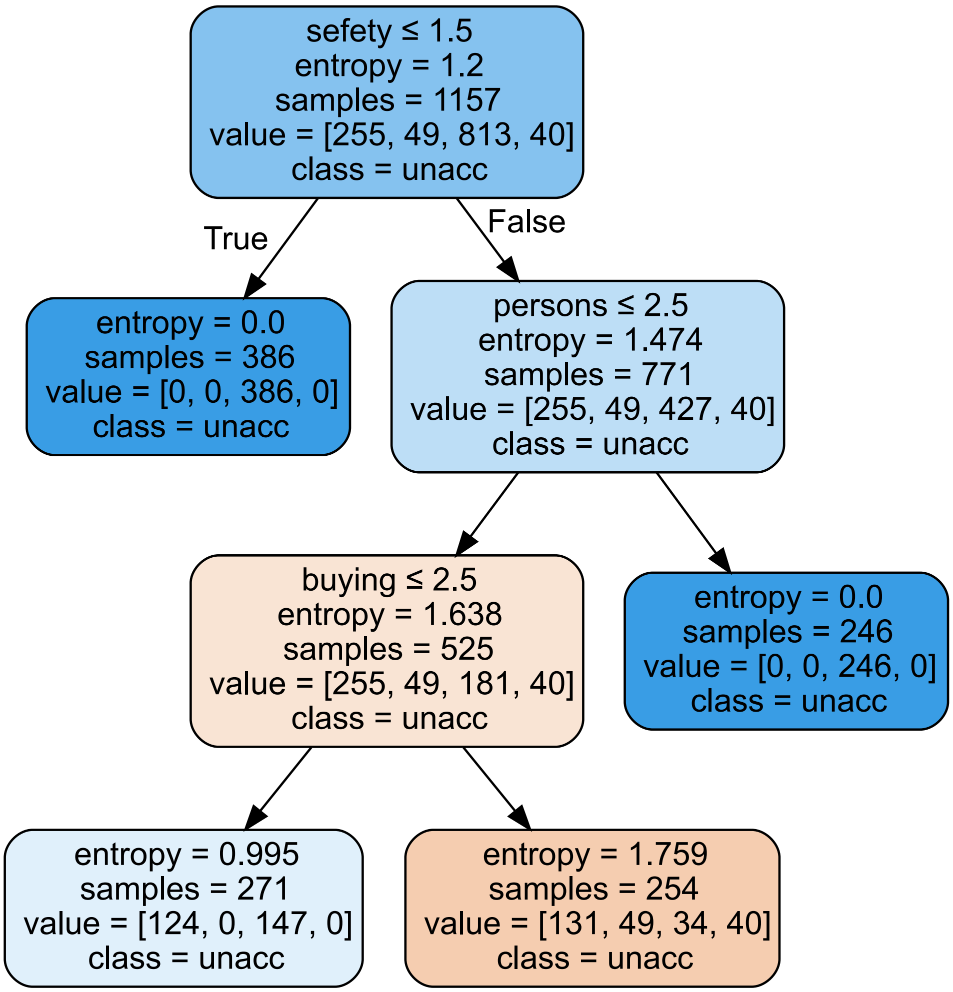
Out[39]:
```
[Text(0.4, 0.875, 'X[5] <= 1.5\nentropy = 1.2\nsamples = 1157\nvalue = [255, 49, 813, 40]'),
 Text(0.2, 0.625, 'entropy = 0.0\nsamples = 386\nvalue = [0, 0, 386, 0]'),
 Text(0.6, 0.625, 'X[3] <= 2.5\nentropy = 1.474\nsamples = 771\nvalue = [255, 49, 427, 40]'),
 Text(0.4, 0.375, 'X[0] <= 2.5\nentropy = 1.638\nsamples = 525\nvalue = [255, 49, 181, 40]'),
 Text(0.2, 0.125, 'entropy = 0.995\nsamples = 271\nvalue = [124, 0, 147, 0]'),
 Text(0.6, 0.125, 'entropy = 1.759\nsamples = 254\nvalue = [131, 49, 34, 40]'),
 Text(0.8, 0.375, 'entropy = 0.0\nsamples = 246\nvalue = [0, 0, 246, 0]')]
```



In [40]:
```
import graphviz
dot_data=tree.export_graphviz(clf_en,out_file=None,
                              feature_names=X_train.columns,
                              class_names=y_train,
                              filled=True,rounded=True,
                              special_characters=True)

graph=graphviz.Source(dot_data)
graph
```

Out[40]:



Now, based on the above analysis, we can conclude that the accuracy of our classification model is very good. Our model is doing a very good job in terms of predicting the class labels.

But, it does not give the underlying distribution of values. Also, it doesn't say anything about the kind of mistakes our classifier is making.

We have another tool called `Confusion Matrix` that comes to our rescue.

Confusion Matrix

A confusion metrix is tool for summarizing the performance of classification algoritham , A confusion matrix give as a clear picture classification model performance and the type of error prodeced by the model , It gives is summary od correct and incorrect prediction broken down for each category . The summuary is repreasent tabular format

four type of outcomes are possiable while evaluating a classifiacation model performance:-

True Positive:- True positives occurs when we predict an observation belong to a certain class and observation actually belongs to tht class .

True Negative:- The true nagative occurs when we predict an observation does not belong to a certion class and the obervastion dont belongs to the class

False Positive:- False positive occurs when we predict an observation belong to a certian class but observation actually does not belong to that class. This type of error is class **Type I error**

False Negative:- False negative occurs when we predict an observation does not belong to a certain class but observation actually belongs to that class .This is very serious error and it is called **Type II error**

In [41]:
```
#print the confusion matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred_en)
print('Confusion Metrix\n\n',cm)
```

```
Confusion Metrix

 [[ 73   0  56   0]
 [ 20   0   0   0]
 [ 12   0 385   0]
 [ 25   0   0   0]]
```

Classification Report

Classification Report is another way to evaluate the performance of the classification model. Displays the **precision**, **recall**, **f1**, and **support** scores for the model. I have described these terms below.

We can print a classification report as follows:-`

In [42]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_en))
```

```
              precision    recall  f1-score   support

         acc       0.56      0.57      0.56       129
        good       0.00      0.00      0.00        20
       unacc       0.87      0.97      0.92       397
       vgood       0.00      0.00      0.00        25

    accuracy                           0.80       571
   macro avg       0.36      0.38      0.37       571
weighted avg       0.73      0.80      0.77       571
```

Results and conclusion

1. In this project, I build a decision tree classifier model to predict car safety. I build two models, one with the `Gini index` criterion and the other with the `entropy` criterion. The model produces very good performance as indicated by the model's precision in both cases, which turned out to be **0.8021**.
2. In the model with the "Gini index" criterion, the accuracy score of the training set is **0.7865**, while the accuracy of the test set is **0.8021** . These two values are quite comparable. Therefore, there are no signs of overfitting.
3. Similarly, in the model with the `entropy` criterion, the accuracy score of the training set is **0.7865** while the accuracy of the test set is **0.8021**. We obtain the same values as in the case with the `gini` criterion. Therefore, there are no signs of overfitting.

4. In both cases, the accuracy score of the training set and the test set is the same. It can happen due to a small data set.
5. The confusion matrix and the classification report produce very good model performance

For More Machine Learning connect me on **linkedin** ([https://www.linkedin.com/in/akash-kumbhar-67540a22a/ (https://www.linkedin.com/in/akash-kumbhar-67540a22a/)](https://www.linkedin.com/in/akash-kumbhar-67540a22a/))