In [1]:

```
import os
print(os.getcwd())
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
%matplotlib inline
```

```
C:\Users\divya
```

In [2]:

```
df = pd.read_csv(r'C:\Users\divya\Downloads\car_evaluation.csv', header = None)
```

In [3]:

```
df.head()
```

Out[3]:

| | 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 |

In [4]:

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
df.columns = col_names
col_names
```

Out[4]:

```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

In [5]:

```
df.head()
```

Out[5]:

| | buying | maint | doors | persons | lug_boot | safety | class |
|---|---|---|---|---|---|---|---|
| 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 |

In [6]:

```
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   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [7]:

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

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

In [8]:

```
df.shape
```

Out[8]:

```
(1728, 7)
```

In [9]:

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

In [10]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

In [11]:

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
X_train = enc.fit_transform(X_train)
X_test = enc.transform((X_test))
```

## Gini index as criterion

In [12]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [13]:

```
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
clf_gini.fit(X_train, y_train)
```

Out[13]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, 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='deprecated',
                       random_state=42, splitter='best')
```

In [14]:

```
y_pred = clf_gini.predict(X_test)
```

In [15]:

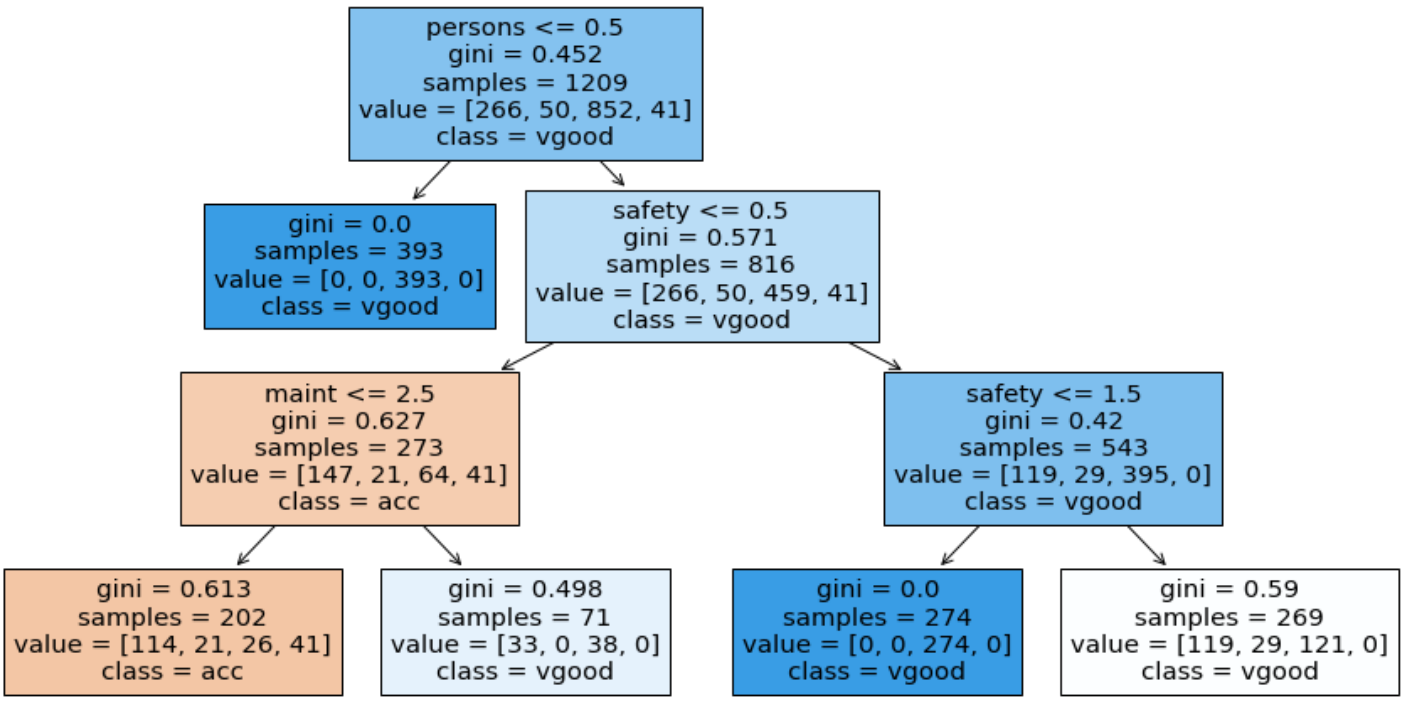
```
from sklearn.metrics import accuracy_score

print(f'Model with gini index gives an accuracy of: {accuracy_score(y_true=y_test, y_pred
=y_pred)}')
```

```
Model with gini index gives an accuracy of: 0.7572254335260116
```

In [16]:

```
from sklearn import tree
plt.figure(figsize=(15,8))
tree.plot_tree(clf_gini,
               feature_names=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety
'],
               class_names= list(set(y_train)),
               filled = True)
plt.show()
```



In [17]:

```
# Check for underfitting

print(f'Training set score: {clf_gini.score(X_train,y_train)}')
print(f'Test set score: {clf_gini.score(X_test,y_test)}')
```

```
Training set score: 0.7775020678246485
Test set score: 0.7572254335260116
```

## Entropy as criterion

In [18]:

```
clf_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
clf_entropy.fit(X_train, y_train)
```

Out[18]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, 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='deprecated',
                       random_state=42, splitter='best')
```

In [19]:

```
y_pred = clf_entropy.predict(X_test)
```

In [20]:

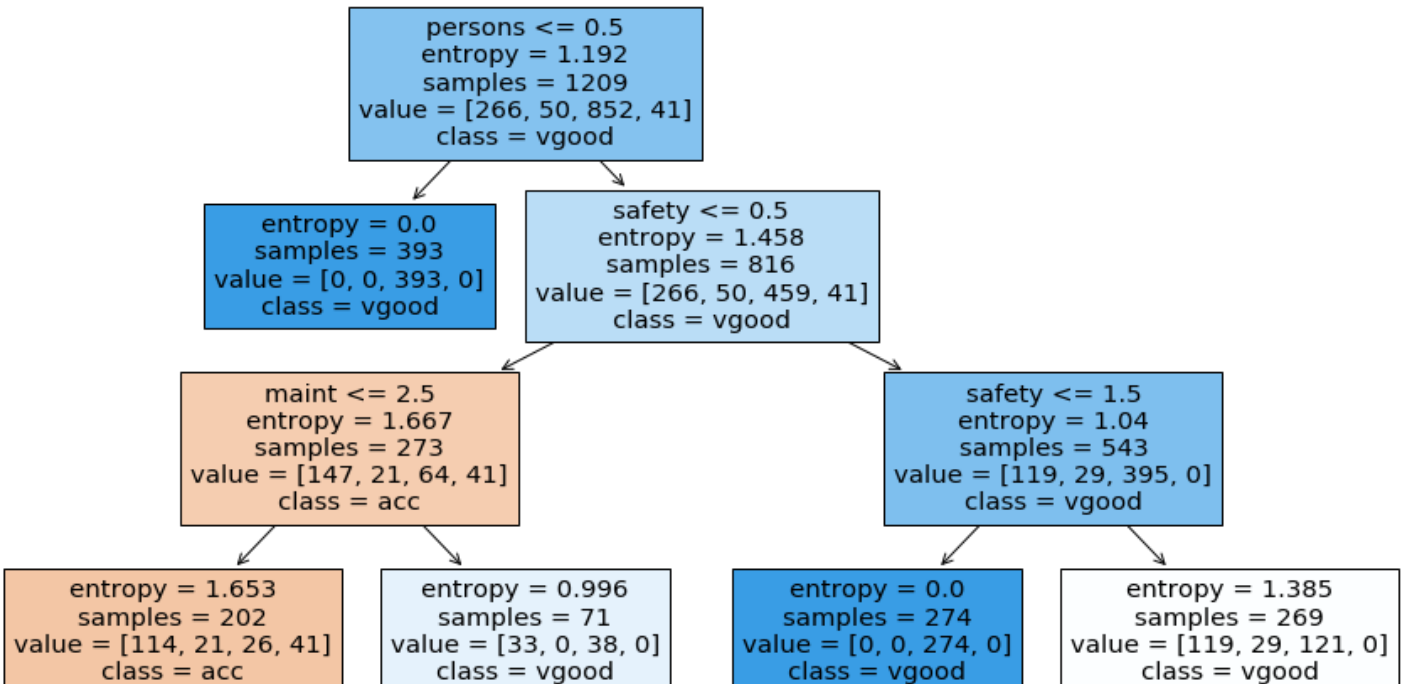
```
from sklearn.metrics import accuracy_score

print(f'Model with gini index gives an accuracy of: {accuracy_score(y_test, y_pred)}')
```

```
Model with gini index gives an accuracy of: 0.7572254335260116
```

In [21]:

```
plt.figure(figsize=(15,8))
tree.plot_tree(clf_entropy,
               feature_names=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety
'],
               class_names= list(set(y_train)),
               filled = True)
plt.show()
```



In [22]:

```
# Check for underfitting

print(f'Training set score: {clf_entropy.score(X_train,y_train)}')
print(f'Test set score: {clf_entropy.score(X_test,y_test)}')
```

```
Training set score: 0.7775020678246485
Test set score: 0.7572254335260116
```

In [23]:

```
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(y_test, y_pred)
```

In [24]:

```
print(cm)
```

```
[[ 44   0  74   0]
 [  9   0  10   0]
 [  9   0 349   0]
 [ 24   0   0   0]]
```

In [25]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         acc       0.51      0.37      0.43       118
        good       0.00      0.00      0.00        19
       unacc       0.81      0.97      0.88       358
       vgood       0.00      0.00      0.00        24

    accuracy                           0.76       519
   macro avg       0.33      0.34      0.33       519
weighted avg       0.67      0.76      0.71       519
```

```
C:\Users\divya\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1272: Undefi
nedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels wi
th no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

# Cross Validation

In [26]:

```
params_grid = {
    'criterion':['gini','entropy'],
    'max_depth':[3,4,5,6,7,8,9,10]
}
```

In [27]:

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train,y_train)
```

Out[27]:

```
DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                       random_state=None, splitter='best')
```

In [28]:

```
dt_validated = GridSearchCV(estimator=decision_tree, param_grid=params_grid,scoring='accuracy',cv=20)
```

In [29]:

```
%%time
dt_validated.fit(X_train,y_train)
```

```
Wall time: 1.41 s
```

Out[29]:

```
GridSearchCV(cv=20, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 4, 5, 6, 7, 8, 9, 10]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [30]:

```
print(f'Best parameters for decison tree classifier after CV -> {dt_validated.best_params_}')
print(f'Best score on decision tree classifier after CV -> {dt_validated.best_score_}')
```

```
Best parameters for decison tree classifier after CV -> {'criterion': 'entropy', 'max_depth': 10}
Best score on decision tree classifier after CV -> 0.9809699453551912
```

In [31]:

```
print(f'Score on train set of DT classifier before CV -> {decision_tree.score(X_train, y_train)}')
print(f'Score on test set of DT classifier before CV -> {decision_tree.score(X_test, y_test)}')
print(f'Score on train set of DT classfifier after CV -> {dt_validated.score(X_train, y_train)}')
print(f'Score on test set of DT classifier after CV -> {dt_validated.score(X_test, y_test)}')
```

```
Score on train set of DT classifier before CV -> 1.0
Score on test set of DT classifier before CV -> 0.9653179190751445
Score on train set of DT classfifier after CV -> 0.9925558312655087
Score on test set of DT classifier after CV -> 0.9595375722543352
```

In [32]:

```
print('Classification report on train set')
print(classification_report(y_true=y_train, y_pred=dt_validated.predict(X_train)))
```

```
Classification report on train set
              precision    recall  f1-score   support

         acc       0.98      0.99      0.99       266
        good       0.98      0.98      0.98        50
       unacc       1.00      0.99      1.00       852
       vgood       0.98      1.00      0.99        41
```

```
    accuracy                           0.99      1209
   macro avg       0.98      0.99      0.99      1209
weighted avg       0.99      0.99      0.99      1209
```

In [33]:

```
print('Classification report on test set')
print(classification_report(y_true=y_test, y_pred=dt_validated.predict(X_test)))
```

```
Classification report on test set
              precision    recall  f1-score   support

         acc       0.92      0.92      0.92       118
        good       0.71      0.89      0.79        19
       unacc       0.99      0.98      0.99       358
       vgood       0.88      0.88      0.88        24

    accuracy                           0.96       519
   macro avg       0.88      0.92      0.89       519
weighted avg       0.96      0.96      0.96       519
```

In [34]:

```
print('Confusion matrix on train set')
print(confusion_matrix(y_true=y_train, y_pred=dt_validated.predict(X_train)))
```

```
Confusion matrix on train set
[[263   1   2   0]
 [  0  49   0   1]
 [  5   0 847   0]
 [  0   0   0  41]]
```

In [35]:

```
print('Confusion matrix on test set')
print(confusion_matrix(y_true=y_test, y_pred=dt_validated.predict(X_test)))
```

```
Confusion matrix on test set
[[108   7   2   1]
 [  0  17   0   2]
 [  6   0 352   0]
 [  3   0   0  21]]
```

In [36]:

```
best_dt = DecisionTreeClassifier(criterion='entropy',max_depth=9)
```

In [37]:

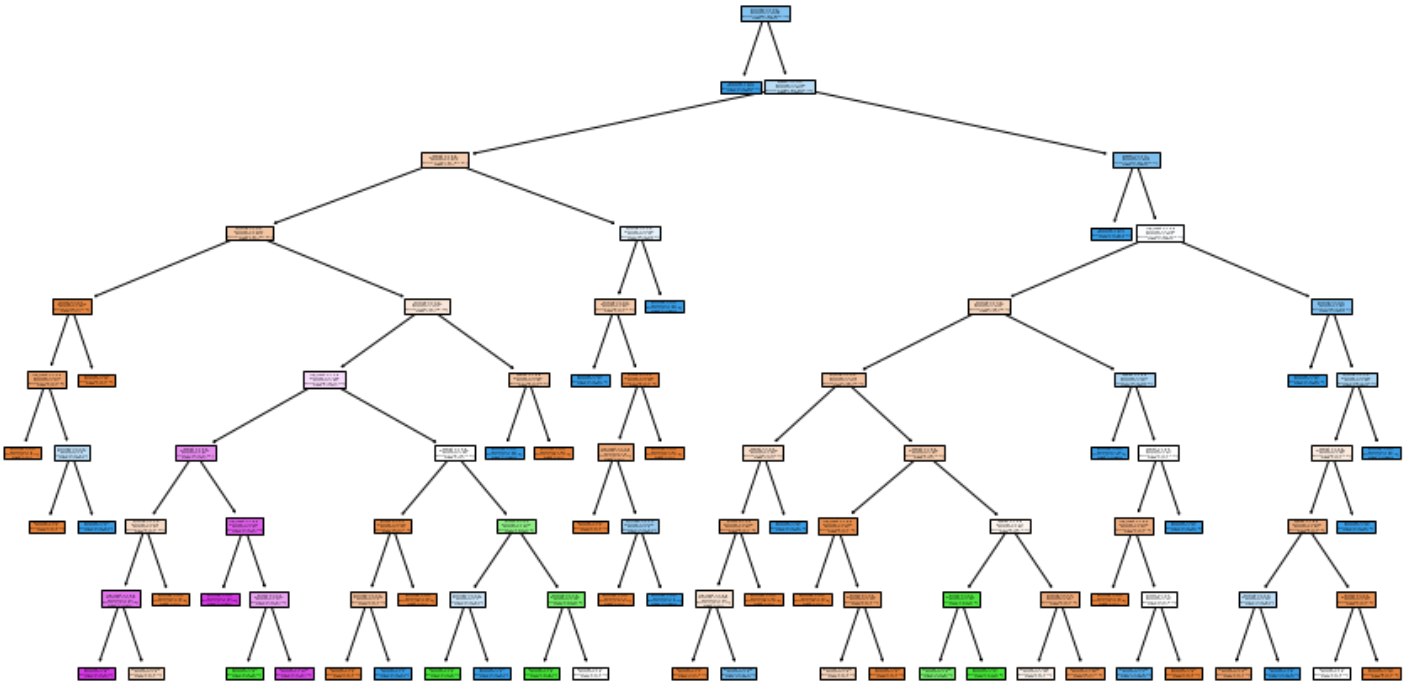
```
best_dt.fit(X_train,y_train)
```

Out[37]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=9, 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='deprecated',
                       random_state=None, splitter='best')
```

In [38]:

```
plt.figure(figsize=(15,8))
tree.plot_tree(best_dt,
               feature_names=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety
'],
               class_names= list(set(y_train)),
               filled = True)
plt.show()
```



In [ ]: