CHENNAM RUSHWANTH - PRODIGY TASK 3- DATA SCIENCE INTERNSHIP

In [6]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings

warnings.filterwarnings('ignore')
data = "C:\\Users\\DELL\\Downloads\\car_evaluation.csv"
df = pd.read_csv(data, header=None)
```

In [8]:

```
df
```

Out[8]:

```
         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
 ...    ...    ...    ...   ...    ...   ...    ...
1723    low    low  5more  more    med   med   good
1724    low    low  5more  more    med  high  vgood
1725    low    low  5more  more    big   low  unacc
1726    low    low  5more  more    big   med   good
1727    low    low  5more  more    big  high  vgood
```

1728 rows × 7 columns

In [10]:

```
df.shape
```

Out[10]:

```
(1728, 7)
```

In [12]:

```
df.head()
```

Out[12]:

```
      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 [14]:

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


df.columns = col_names

col_names
```

Out[14]:

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

In [16]:

```
df.head()
```

Out[16]:

	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 [18]:

```
df.info
```

Out[18]:

```
<bound method DataFrame.info of      buying  maint  doors persons lug_boot safety  cl
ass
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
...     ...    ...   ...     ...      ...    ...    ...
1723    low    low 5more    more      med    med   good
1724    low    low 5more    more      med   high  vgood
1725    low    low 5more    more      big    low  unacc
1726    low    low 5more    more      big    med   good
1727    low    low 5more    more      big   high  vgood

[1728 rows x 7 columns]>
```

In [20]:

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
for col in col_names:

    print(df[col].value_counts())
```

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

In [22]:

```
df['class'].value_counts()
```

Out[22]:

```
class
unacc    1210
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

In [24]:

```
df.isnull().sum()
```

Out[24]:

```
buying      0
maint       0
doors       0
persons     0
lug_boot    0
safety      0
class       0
dtype: int64
```

In [26]:

```
# Declare feature vector and target variable
X = df.drop(['class'], axis=1)

y = df['class']
```

In [28]:

```
# split X and y into training and testing 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_sta
```

In [30]:

```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[30]:

```
((1157, 6), (571, 6))
```

In [32]:

```
# check data types in X_train

X_train.dtypes
```

Out[32]:

```
buying      object
maint       object
doors       object
persons     object
lug_boot    object
safety      object
dtype: object
```

In [34]:

```
X_train.head()
```

Out[34]:

	buying	maint	doors	persons	lug_boot	safety
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 [36]:

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\dell\anaconda3\lib\site-
packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in c:\users\dell\anaconda3\lib\site-pack
ages (from category_encoders) (1.24.3)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\dell\anaconda3\lib\si
te-packages (from category_encoders) (1.3.0)
Requirement already satisfied: scipy>=1.0.0 in c:\users\dell\anaconda3\lib\site-packa
ges (from category_encoders) (1.11.1)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\dell\anaconda3\lib\site
-packages (from category_encoders) (0.14.0)
Requirement already satisfied: pandas>=1.0.5 in c:\users\dell\anaconda3\lib\site-pack
ages (from category_encoders) (2.0.3)
Requirement already satisfied: patsy>=0.5.1 in c:\users\dell\anaconda3\lib\site-packa
ges (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dell\anaconda3\lib
\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-packa
ges (from pandas>=1.0.5->category_encoders) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\dell\anaconda3\lib\site-pac
kages (from pandas>=1.0.5->category_encoders) (2023.3)
Requirement already satisfied: six in c:\users\dell\anaconda3\lib\site-packages (from
patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\dell\anaconda3\lib\site-pack
ages (from scikit-learn>=0.20.0->category_encoders) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\anaconda3\lib\si
te-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in c:\users\dell\anaconda3\lib\site-pa
ckages (from statsmodels>=0.9.0->category_encoders) (23.1)
```

In [38]:

```
# import category encoders

import category_encoders as ce
```

In [40]:

```
# encode variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', '


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
X_train.head()
```

Out[40]:

	buying	maint	doors	persons	lug_boot	safety
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 [42]:

```
X_test.head()
```

Out[42]:

	buying	maint	doors	persons	lug_boot	safety
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

In [44]:

```
#Decision Tree Classifier with criterion gini index
```

In [46]:

```
# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier
```

In [48]:

```
# instantiate the DecisionTreeClassifier model with criterion gini index

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)


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

Out[48]:

```
▾                DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [50]:

```
#Predict the Test set results with criterion gini index
```

In [52]:

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

In [54]:

```
#Check accuracy score with criterion gini index
```

In [56]:

```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_scor
```

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

In [58]:

```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[58]:

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

In [60]:

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_t
#Check for overfitting and underfitting
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

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

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

In [62]:

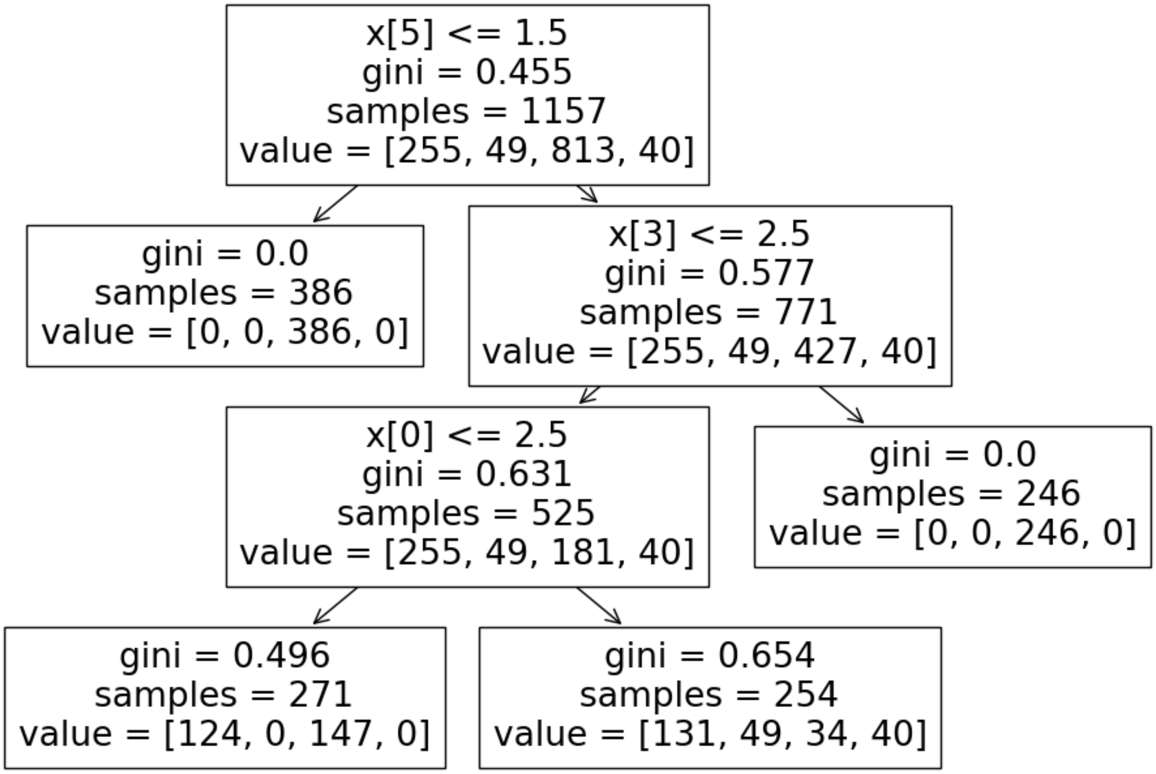
```
#Visualize decision-trees
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_gini.fit(X_train, y_train))
```

Out[62]:

```
[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, 4
0]'),
 Text(0.4, 0.375, 'x[0] <= 2.5\ngini = 0.631\nsamples = 525\nvalue = [255, 49, 181, 4
0]'),
 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]')]
```



In [64]:

```
# Decision Tree Classifier with criterion entropy
# 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[64]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

In [66]:

```
#Predict the Test set results with criterion entropy
y_pred_en = clf_en.predict(X_test)
#Check accuracy score with criterion entropy
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy_score(y
```

```
Model accuracy score with criterion entropy: 0.8021
```

In [68]:

```
#Compare the train-set and test-set accuracy
y_pred_train_en = clf_en.predict(X_train)
```

In [70]:

```
y_pred_train_en
```

Out[70]:

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

In [72]:

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_t
```

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

In [74]:

```
#Check for overfitting and underfitting
```

In [76]:

```
# print the scores 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 test-set score is same as above. The training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting.

In [81]:

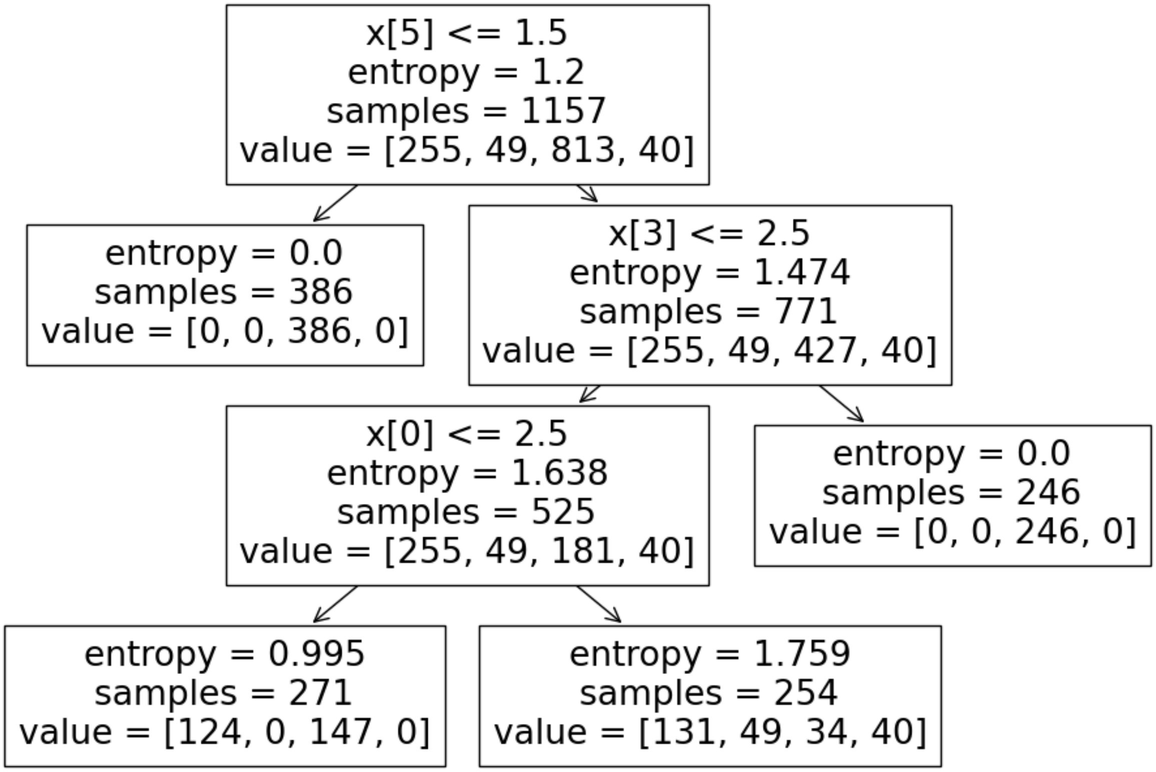
```
#Visualize decision-tres
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[81]:

```
[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, 42
7, 40]'),
 Text(0.4, 0.375, 'x[0] <= 2.5\nentropy = 1.638\nsamples = 525\nvalue = [255, 49, 18
1, 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]')]
```



Now, based on the above analysis we can conclude that our classification model accuracy 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 does not tell anything about the type of errors our classifer is making.

We have another tool called Confusion matrix that comes to our rescue

In [84]:
```
# Confusion matrix
```

In [86]:
```
# Print the Confusion Matrix and slice it into four pieces
```

In [88]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_en)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

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

In [90]:
```
#Classification Report
```

In [92]:
```
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 the safety of the car. I build two models, one with criterion gini index and another one with criterion entropy. The model yields a very good performance as indicated by the model accuracy in both the cases which was found to be 0.8021.

1. In the model with criterion gini index, the training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting. 3.Similarly, in the model with criterion entropy, the training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021.We get the same values as in the case with criterion gini. So, there is no sign of overfitting. 4.In both the cases, the training-set and test-set accuracy score is the same. It may happen because of small dataset. 5.The confusion matrix and classification report yields very good model performance.

```
In [ ]:
```