In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn import metrics
```

Section 5 Final Notebook - Categorical Machine Learning Using Car Evaluation Dataset

In [ ]:

```
#Reading the dataset - Just making sure there are no unexpected errors here
df = pd.read_csv(r'C:\Users\yaoz\Desktop\dataeng_test\Section_5_Machine_Learning\data\car.data',header =
None,names = ['buying_price', 'maintenance_price','no_of_doors','no_of_person','lug_boot','safety','decis
ion']) #Change your file path here
df
```

	buying_price	maintenance_price	no_of_doors	no_of_person	lug_boot	safety	decision
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

Initial EDA - Explaratory Data Analysis: Here i just want to see the size and variables of the data and also to check there are no missing fields

- So, we have 1728 datapoints with 6 columns (features) and 1 class label(target) column.
- Data set contains categorical features.
- There are no missing value in the dataset.

In [ ]:

```
#lets explore the given dataset
print(df.shape)

print('\n',df.info())

#checking for any missing value in the given dataset
print('\n',df.isnull().sum())
```

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

 None

 buying_price         0
maintenance_price    0
no_of_doors          0
no_of_person         0
lug_boot             0
safety               0
decision             0
dtype: int64
```

In [ ]:

```
df.buying_price.value_counts()
```

```
med      432
low      432
vhigh    432
high     432
Name: buying_price, dtype: int64
```

We must take note that in this use case the target variable is swapped and is not the last column 'decision', rather in this example we use buying price as the target variable. At first glance it seems our tgt variable is evenly split amongst all categories - This is bad! This could mean the dataset cannot sufficiently account for variation in our target column.

- Note: In the intermediate notebook, i explore the original target var of 'decision' instead.

In [ ]:

```
#Segregating datasets into features and target datasets

#dropping first column(target label) and selecting rest, dropping no_of_person as well as this is not a f
actor in our final model
X=df.drop(['buying_price','no_of_person'],axis=1)

#selecting only target variable
Y=df['buying_price']

print(X.shape,Y.shape)
```

```
(1728, 5) (1728,)
```

Feature engineering

As the dataset has categorical features and hence they needs to be encoded in the appropriate form. There are two main method of encoding:

- One hot encoding
- Categorical encoding

As we have categorial features that are ordinal in nature i.e that can be ranked (ordered) hence label encoding will solve our purpose.Had there been nominal features we could have preferred one hot encoding.

In [ ]:

```
# importing necessary package for encoding our categorial features
import category_encoders as ce

encoder = ce.OrdinalEncoder(cols=['maintenance_price','no_of_doors','lug_boot','safety','decision'])
x= encoder.fit_transform(X)

print(x.head())
print(x.shape)
```

```
   maintenance_price  no_of_doors  lug_boot  safety  decision
0                  1            1         1       1         1
1                  1            1         1       2         1
2                  1            1         1       3         1
3                  1            1         2       1         1
4                  1            1         2       2         1
(1728, 5)
```

In [ ]:

```
print('Original dataset')
print('\n\n',X.head())
print('\n Encoded dataset')
print('\n',x.head())
```

```
Original dataset


   maintenance_price no_of_doors lug_boot safety decision
0              vhigh           2    small    low    unacc
1              vhigh           2    small    med    unacc
2              vhigh           2    small   high    unacc
3              vhigh           2      med    low    unacc
4              vhigh           2      med    med    unacc

 Encoded dataset

    maintenance_price  no_of_doors  lug_boot  safety  decision
0                  1            1         1       1         1
1                  1            1         1       2         1
2                  1            1         1       3         1
3                  1            1         2       1         1
4                  1            1         2       2         1
```

Comparing labels after encoding and before encoding in the above cell

Below we split our dataset using train_test_split into Training Dataset, Test and Cross Validation

In [ ]:

```
#importing necessary packages and modules
from sklearn.model_selection import train_test_split

x_1,xtest,y_1,ytest=train_test_split(x,Y,test_size=0.3,random_state=2)

xtrain,x_cv,ytrain,y_cv=train_test_split(x_1,y_1,test_size=0.3,random_state=2)


# Exploring class distribution under train ,crossvalidation and test dataset
print('Training Dataset',xtrain.shape,ytrain.shape)
print('\n Class label distribution in Training Set\n',ytrain.value_counts())
print('\n***********')
print("\n CrossValidation Dataset",x_cv.shape,y_cv.shape)
print('\nClass label distribution in Cross Validation Set\n',y_cv.value_counts())
print('\n***********')
print("\n Test Dataset",xtest.shape,ytest.shape)
print('\nClass label distribution in Test Set\n',ytest.value_counts())
```

```
Training Dataset (846, 5) (846,)

 Class label distribution in Training Set
 high     223
vhigh    216
low      204
med      203
Name: buying_price, dtype: int64

***********

 CrossValidation Dataset (363, 5) (363,)

Class label distribution in Cross Validation Set
 med      106
low       88
vhigh     88
high      81
Name: buying_price, dtype: int64

***********

 Test Dataset (519, 5) (519,)

Class label distribution in Test Set
 low      140
high     128
vhigh    128
med      123
Name: buying_price, dtype: int64
```

In [ ]:

```
# importing necessary packages
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
from sklearn import tree
```

Below we use grid search for optimizing parameters on our decision tree model and check for classification accuracy on cross validation dataset

In [ ]:

```
from sklearn.model_selection import GridSearchCV

parameters={'max_depth': list(range(1,30)),
            'min_samples_leaf' : list(range(5,200,20)),
            'min_samples_split': list(range(5,200,20))
            }
model=GridSearchCV(DecisionTreeClassifier(class_weight='balanced'),parameters,n_jobs=-1,cv=10,scoring='ac
curacy')
model.fit(xtrain,ytrain)
print(model.best_estimator_)
print("\n",model.best_params_)
print("\n",model.score(x_cv,y_cv))

ypredict=model.predict(x_cv)
accuracy=accuracy_score(y_cv,ypredict,normalize=True)*float(100)
print('\n\n classification report')
print(classification_report(y_cv,ypredict))
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=45, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')

 {'max_depth': 5, 'min_samples_leaf': 45, 'min_samples_split': 5}

 0.31955922865013775


 classification report
              precision    recall  f1-score   support

        high       0.30      0.20      0.24        81
         low       0.45      0.19      0.27        88
         med       0.36      0.39      0.37       106
       vhigh       0.27      0.48      0.34        88

    accuracy                           0.32       363
   macro avg       0.34      0.31      0.31       363
weighted avg       0.34      0.32      0.31       363
```

Testing model accuracy on Unseen Data (Test Dataset) Using optimal value of hyper parameters that we got via Grid search, predicting the class labels for the test dataset and getting its classification report

```
In [ ]:
clf=tree.DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=45, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
clf.fit(xtrain,ytrain)
ypredict=clf.predict(xtest)
accuracy=accuracy_score(ytest,ypredict,normalize=True)*float(100)
print('\n Accuracy score is',accuracy)
print('\n classification report')
print(classification_report(ytest,ypredict))
```

```
 Accuracy score is 29.09441233140655

 classification report
              precision    recall  f1-score   support

        high       0.23      0.11      0.15       128
         low       0.40      0.20      0.27       140
         med       0.27      0.38      0.32       123
       vhigh       0.29      0.48      0.36       128

    accuracy                           0.29       519
   macro avg       0.30      0.29      0.27       519
weighted avg       0.30      0.29      0.27       519
```

Accuracy Score

Model Accuracy on Training dataset= 32% , Model Accuracy on Test Dataset=29% Thus the present model has shown consistant results on both Training and Test Datasets, however because of the low accuracy of both, cannot be considered a generalized model

Classification Report

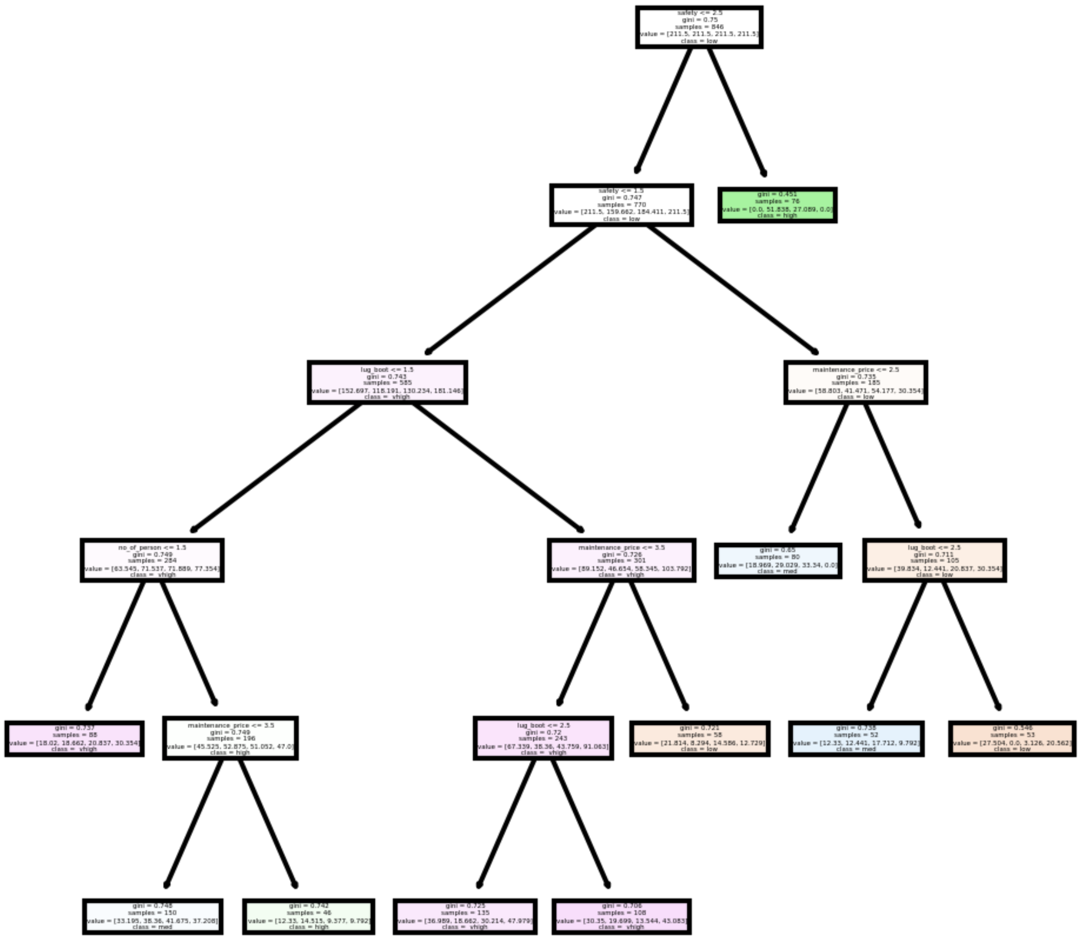
It is advisable to not just rely on accuracy score as accuracy score sometimes doesnot tell the true picture related to the model, especially in the case of imbalanced dataset. Classification Report is an efective metric that makes us doubly sure regarding the performance of our model. The classification report above should be compared to the intermediate example to compare the difference in accuracy. Personally i believe it is possibly due to the even split of target labels which suggest the wrong target variable is used as the dataset does not account for variation in buying price.

In [ ]:

```
# Visualising Decision Tree
cols=['maintenance_price','no_of_doors','no_of_person','lug_boot','safety','decision']
trgt=['low','high','med',' vhigh']

fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,feature_names = cols, class_names=trgt,filled = True)
```

```
[Text(597.8571428571429, 830.5, 'safety <= 2.5\ngini = 0.75\nsamples = 846\nvalue = [211.5, 211.5, 211.5, 211.5]\nclass = low'),
 Text(531.4285714285714, 679.5, 'safety <= 1.5\ngini = 0.747\nsamples = 770\nvalue = [211.5, 159.662, 184.411, 211.5]\nclass = low'),
 Text(332.14285714285717, 528.5, 'lug_boot <= 1.5\ngini = 0.743\nsamples = 585\nvalue = [152.697, 118.191, 130.234, 181.146]\nclass =  vhi
gh'),
 Text(132.85714285714286, 377.5, 'no_of_person <= 1.5\ngini = 0.749\nsamples = 284\nvalue = [63.545, 71.537, 71.889, 77.354]\nclass =  vhi
gh'),
 Text(66.42857142857143, 226.5, 'gini = 0.737\nsamples = 88\nvalue = [18.02, 18.662, 20.837, 30.354]\nclass =  vhigh'),
 Text(199.28571428571428, 226.5, 'maintenance_price <= 3.5\ngini = 0.749\nsamples = 196\nvalue = [45.525, 52.875, 51.052, 47.0]\nclass = h
igh'),
 Text(132.85714285714286, 75.5, 'gini = 0.748\nsamples = 150\nvalue = [33.195, 38.36, 41.675, 37.208]\nclass = med'),
 Text(265.7142857142857, 75.5, 'gini = 0.742\nsamples = 46\nvalue = [12.33, 14.515, 9.377, 9.792]\nclass = high'),
 Text(531.4285714285714, 377.5, 'maintenance_price <= 3.5\ngini = 0.726\nsamples = 301\nvalue = [89.152, 46.654, 58.345, 103.792]\nclass =
vhigh'),
 Text(465.0, 226.5, 'lug_boot <= 2.5\ngini = 0.72\nsamples = 243\nvalue = [67.339, 38.36, 43.759, 91.063]\nclass =  vhigh'),
 Text(398.57142857142856, 75.5, 'gini = 0.725\nsamples = 135\nvalue = [36.989, 18.662, 30.214, 47.979]\nclass =  vhigh'),
 Text(531.4285714285714, 75.5, 'gini = 0.706\nsamples = 108\nvalue = [30.35, 19.699, 13.544, 43.083]\nclass =  vhigh'),
 Text(597.8571428571429, 226.5, 'gini = 0.721\nsamples = 58\nvalue = [21.814, 8.294, 14.586, 12.729]\nclass = low'),
 Text(730.7142857142858, 528.5, 'maintenance_price <= 2.5\ngini = 0.735\nsamples = 185\nvalue = [58.803, 41.471, 54.177, 30.354]\nclass =
low'),
 Text(664.2857142857143, 377.5, 'gini = 0.65\nsamples = 80\nvalue = [18.969, 29.029, 33.34, 0.0]\nclass = med'),
 Text(797.1428571428571, 377.5, 'lug_boot <= 2.5\ngini = 0.711\nsamples = 105\nvalue = [39.834, 12.441, 20.837, 30.354]\nclass = low'),
 Text(730.7142857142858, 226.5, 'gini = 0.738\nsamples = 52\nvalue = [12.33, 12.441, 17.712, 9.792]\nclass = med'),
 Text(863.5714285714286, 226.5, 'gini = 0.546\nsamples = 53\nvalue = [27.504, 0.0, 3.126, 20.562]\nclass = low'),
 Text(664.2857142857143, 679.5, 'gini = 0.451\nsamples = 76\nvalue = [0.0, 51.838, 27.089, 0.0]\nclass = high')]
```


safety <= 2.5
gini = 0.75
samples = 846
value = [211.5, 211.5, 211.5, 211.5]
class = low
safety <= 1.5
gini = 0.747
samples = 770
value = [211.5, 159.662, 184.411, 211.5]
class = low
gini = 0.451
samples = 76
value = [0.0, 51.838, 27.089, 0.0]
class = high
lug_boot <= 1.5
gini = 0.743
samples = 585
value = [152.697, 118.191, 130.234, 181.146]
class = vhigh
maintenance_price <= 2.5
gini = 0.735
samples = 185
value = [58.803, 41.471, 54.177, 30.354]
class = low
no_of_person <= 1.5
gini = 0.749
samples = 284
value = [63.545, 71.537, 71.889, 77.354]
class = vhigh
maintenance_price <= 3.5
gini = 0.726
samples = 301
value = [89.152, 46.654, 58.345, 103.792]
class = vhigh
gini = 0.65
samples = 80
value = [18.969, 29.029, 33.34, 0.0]
class = med
lug_boot <= 2.5
gini = 0.711
samples = 105
value = [39.834, 12.441, 20.837, 30.354]
class = low
gini = 0.737
samples = 88
value = [18.02, 18.662, 20.837, 30.354]
class = vhigh
maintenance_price <= 3.5
gini = 0.749
samples = 196
value = [45.525, 52.875, 51.052, 47.0]
class = high
lug_boot <= 2.5
gini = 0.72
samples = 243
value = [67.339, 38.36, 43.759, 91.063]
class = vhigh
gini = 0.721
samples = 58
value = [21.814, 8.294, 14.586, 12.729]
class = low
gini = 0.738
samples = 52
value = [12.33, 12.441, 17.712, 9.792]
class = med
gini = 0.546
samples = 53
value = [27.504, 0.0, 3.126, 20.562]
class = low
gini = 0.748
samples = 150
value = [33.195, 38.36, 41.675, 37.208]
class = med
gini = 0.742
samples = 46
value = [12.33, 14.515, 9.377, 9.792]
class = high
gini = 0.725
samples = 135
value = [36.989, 18.662, 30.214, 47.979]
class = vhigh
gini = 0.706
samples = 108
value = [30.35, 19.699, 13.544, 43.083]
class = vhigh

In the above we visualize the decision tree plot

Prediction - This Model was trained on Buying_price as target variablee: Now lets test on the filtered sample specifications!

In [ ]:

```
filter = df['maintenance_price'] == 'high'
filter1 = df['no_of_doors'] == '4'
filter2 = df['lug_boot'] == 'big'
filter3= df['safety'] == 'high'
filter4 = df['decision'] == 'good'
filter5 = df['decision'] == 'vgood'


#We must exclude the last feature as the last variable is a missing factor

filtereddata = df.where(filter& filter1 & filter2&filter3&filter5)
filtereddata.dropna(inplace = True)

filterencoder = ce.OrdinalEncoder(cols = ['buying_price','maintenance_price','no_of_doors','no_of_person'
,'lug_boot','safety','decision'])
unseen = filterencoder.fit_transform(filtereddata)

print(filtereddata['buying_price'])
```

```
1475    low
1484    low
Name: buying_price, dtype: object
```

In [ ]:

```
unseen.drop(['buying_price','no_of_doors'],axis =1,inplace=True)

print(clf.predict(unseen))
```

```
['vhigh' 'vhigh']
```

In [ ]:

```
print("Accuracy:",metrics.accuracy_score(ytest, ypredict))
```

```
Accuracy: 0.2909441233140655
```

Conclusion: This model was not trained very well as the model predicted a buying price of vhigh and vhigh when the correct labels were both low , when given the same input as well. This can be observed in the low accuracy as well from the earlier analysis. We can see after filtering the dataset to the specifications with some modifcations, the correct label is 'low' for both however our model has incorrectly predicted 'vhigh' for both.