Based on the question on section 5, I will treat this as a classification problem even though it is framed and looked like a regression problem as the goal is to predict the buyer price. On the first look, the buyer price looks like continuous variable but as I look at the data dictionary, I realized that it is a discrete variable. Thus, I will approach this as classification problem

In [17]:
```
# import necesssary lib

import math
import pandas as pd
import numpy as np
import re
from datetime import datetime

# Libraries for ML
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.ensemble import GradientBoostingClassifier


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

In [2]:
```
# read the data into dataframe
df = pd.read_csv('data\car.data', names=['buying','maint','doors','persons','lug_boot','sa
```

In [3]:
```
# take a quick look at the dataframe
df.head()
```

Out[3]:

	buying	maint	doors	persons	lug_boot	safety	acceptability
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

I have the data dictionary which tells me all the unique values:

Class Values:

- unacc, acc, good, vgood

Attributes:

- buying: vhigh, high, med, low.
- maint: vhigh, high, med, low.
- doors: 2, 3, 4, 5more.
- persons: 2, 4, more.
- lug_boot: small, med, big.

- safety: low, med, high.

In [4]:
```
# check frequency of the dependent class

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

Out[4]:
```
vhigh    432
high     432
med      432
low      432
Name: buying, dtype: int64
```

Each class has exactly 432 data points. There is no need to perform any sort of methods to counter data imbalance issues

Encoding is need in order to fit into the ML model.

I decided to use ordinal encoder to encode the variables instead of label encoder. This is because these are ordinal variables and it make sense to have label them in a sequential way. The numbers used to encode these labels have meaningful sequence.

In [5]:
```
# perform ordinal encoding
buying_mapping = {
    'vhigh': 1,
    'high': 2,
    'med': 3,
    'low': 4
}
maint_mapping = {
    'vhigh': 1,
    'high': 2,
    'med': 3,
    'low': 4
}
doors_mapping = {
    '2': 1,
    '3': 2,
    '4': 3,
    '5more': 4
}
persons_mapping = {
    '2': 1,
    '4': 2,
    'more': 3
}
lug_boot_mapping = {
    'small': 1,
    'med': 2,
    'big': 3
}
safety_mapping = {
    'low': 1,
    'med': 2,
    'high': 3
}
acceptability_mapping = {
    'unacc': 1,
    'acc': 2,
    'good': 2,
    'vgood': 3
}
```

In [6]:
```
df['buying'] = df['buying'].map(buying_mapping)
df['maint'] = df['maint'].map(maint_mapping)
df['doors'] = df['doors'].map(doors_mapping)
df['persons'] = df['persons'].map(persons_mapping)
df['lug_boot'] = df['lug_boot'].map(lug_boot_mapping)
df['safety'] = df['safety'].map(safety_mapping)
df['acceptability'] = df['acceptability'].map(acceptability_mapping)
```

In [7]:
```
# Check to ensure all the data are encoded properly
# If they are encoded properly, the data types for each column should be integer
df.dtypes
```

Out[7]:
```
buying           int64
maint            int64
doors            int64
persons          int64
lug_boot         int64
safety           int64
acceptability    int64
dtype: object
```

In [8]:
```
# Assign the variables
# Do not include the persons column since it is not needed
X = df[['maint','doors','lug_boot','safety','acceptability']]
y = df[['buying']]
```

In [9]:
```
# Split the dataset to train and test
# Perform Stratified Splitting to ensure that both train and test data have the equal prop
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0,
```

In [52]:
```
# create a gradient boosting classifier model
gb = GradientBoostingClassifier(random_state = 0, max_depth = 5, n_estimators=100, learnir

# fit the training data into the model
gb.fit(X_train.values, y_train.values)

# do a prediction on the test data
y_pred = gb.predict(X_test.values)
```

In [53]:
```
# get confusion matrix of the predictions on the test data
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[30 18 15 24]
 [38 15 22 12]
 [25 13 23 25]
 [24 10 40 12]]
```

Now, lets predict with the following parameters:

- Maintenance = High
- Number of doors = 4
- Lug Boot Size = Big
- Safety = High
- Class Value = Good

In [62]:
```
# predict with the parameters and get the respective encoded values from the mapping files
```

```
to_predict = np.array([[maint_mapping.get('high'),doors_mapping.get('4'),lug_boot_mapping.
                        safety_mapping.get('high'),acceptability_mapping.get('good'),]])
```

In [63]:

```
# predicts and get the respective value
key_list = list(buying_mapping.keys())
val_list = list(buying_mapping.values())

y_pred = gb.predict(a)
print("The buying price is", key_list[val_list.index(y_pred[0])])
```

```
The buying price is high
```