Used Cars Price Prediction

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt

import numpy as np

%matplotlib inline


# preprocessing
from sklearn.model_selection import train_test_split,GridSearchCV

# metrics and models
from sklearn.metrics import r2_score,mean_squared_error
import xgboost as xgb
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\user\anaconda3\lib\site-p
ackages (1.6.1)
Requirement already satisfied: scipy in c:\users\user\anaconda3\lib\site-pac
kages (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\user\anaconda3\lib\site-pac
kages (from xgboost) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

Read datasets

In [3]:

```
df=pd.read_csv(r'vehicles_data_students.csv')
df.head(5)
```

Out[3]:

	Unnamed: 0	id	price	year	manufacturer	model	condition	cylinders	fuel	o
0	55483	7315914053	0	2018.0	ram	promaster 2500	excellent	NaN	gas	
1	162368	7310885048	13995	2017.0	mazda	cx-3	NaN	4 cylinders	gas	
2	234393	7308243856	19990	2019.0	mitsubishi	eclipse cross sp	good	NaN	gas	
3	276110	7315817729	0	2019.0	honda	cr-v	NaN	NaN	gas	
4	349033	7301620999	42900	2015.0	chevrolet	corvette	excellent	8 cylinders	gas	

In [4]:

```
drop_columns=['Unnamed: 0','id','title_status','size','lat','long','county']
df=df.drop(columns=drop_columns,axis=1)
```

In [5]:

```
df.shape
```

Out[5]:

```
(64032, 13)
```

In [6]:

```
df.isna().sum()
```

Out[6]:

```
price              0
year             158
manufacturer    2569
model            802
condition      26097
cylinders      26511
fuel             424
odometer         669
transmission     353
drive          19471
type           13785
paint_color    19505
state              0
dtype: int64
```

In [7]:

```
df=df.dropna()
df.head(5)
```

Out[7]:

	price	year	manufacturer	model	condition	cylinders	fuel	odometer	transmission
5	0	2006.0	chrysler	300	like new	8 cylinders	gas	149000.0	automatic
9	20995	2011.0	chevrolet	silverado 1500	excellent	8 cylinders	gas	92001.0	automatic
15	50995	2017.0	gmc	yukon denali	like new	8 cylinders	gas	70227.0	automatic
22	13500	2014.0	chevrolet	tahoe	good	8 cylinders	gas	96007.0	automatic
29	34990	2016.0	gmc	canyon crew cab sle pickup	good	6 cylinders	gas	34425.0	other

In [8]:

```
df.shape
```

Out[8]:

```
(17491, 13)
```

In [9]:

```
df.describe()
```

Out[9]:

	price	year	odometer
count	1.749100e+04	17491.000000	1.749100e+04
mean	1.632545e+04	2009.535247	1.122616e+05
std	1.294429e+05	9.596493	2.155149e+05
min	0.000000e+00	1918.000000	0.000000e+00
25%	5.600000e+03	2006.000000	5.578700e+04
50%	1.095000e+04	2012.000000	1.025670e+05
75%	2.250000e+04	2015.000000	1.480000e+05
max	1.700000e+07	2022.000000	1.000000e+07

Check if there are any duplicates. Remove if there any duplicates

In [10]:

```
df.drop_duplicates(inplace=True)
```

In [11]:

```
df.shape
```

Out[11]:

```
(16399, 13)
```

Filter Categorical features

In [12]:

```
numerics=['int8','int16','int32','int64','float16','float32','float64']
categorical_columns=[]
features=df.columns.values.tolist()
for col in features:
    if df[col].dtype in numerics:
        continue
    categorical_columns.append(col)
```

In [13]:

```
categorical_columns
```

Out[13]:

```
['manufacturer',
 'model',
 'condition',
 'cylinders',
 'fuel',
 'transmission',
 'drive',
 'type',
 'paint_color',
 'state']
```

Encoding categorical columns using get_dummies.

Note - please visit encoder and imputer live class recordings for more information on get_dummmies

In [14]:

```
df_dummies=pd.get_dummies(df[categorical_columns],drop_first=True)
```

In [15]:

```
df_dummies.head()
```

Out[15]:

	manufacturer_alfa-romeo	manufacturer_audi	manufacturer_bmw	manufacturer_buick	manufacture
5	0	0	0	0	
9	0	0	0	0	
15	0	0	0	0	
22	0	0	0	0	
29	0	0	0	0	

5 rows × 4317 columns

In [16]:

```
df_dummies.shape
```

Out[16]:

```
(16399, 4317)
```

In [17]:

```
df=df.join(df_dummies)
```

In [18]:

```
df.shape
```

Out[18]:

```
(16399, 4330)
```

In [19]:

```
df.head()
```

Out[19]:

	price	year	manufacturer	model	condition	cylinders	fuel	odometer	transmission
5	0	2006.0	chrysler	300	like new	8 cylinders	gas	149000.0	automatic
9	20995	2011.0	chevrolet	silverado 1500	excellent	8 cylinders	gas	92001.0	automatic
15	50995	2017.0	gmc	yukon denali	like new	8 cylinders	gas	70227.0	automatic
22	13500	2014.0	chevrolet	tahoe	good	8 cylinders	gas	96007.0	automatic
29	34990	2016.0	gmc	canyon crew cab sle pickup	good	6 cylinders	gas	34425.0	other

5 rows × 4330 columns

In [20]:

```
df.drop(columns=categorical_columns,axis=1,inplace = True)
```

In [21]:

```
df.head(2)
```

Out[21]:

	price	year	odometer	manufacturer_alfa-romeo	manufacturer_audi	manufacturer_bmw	manufac
5	0	2006.0	149000.0	0	0	0	
9	20995	2011.0	92001.0	0	0	0	

2 rows × 4320 columns

while participating try other encoding techniques as well and compare the final accuracy

selecting realistic data.Here domain knowledge will help a lot to decide wha could be the higher and lower price

Let's consider below prices in this examples

In [22]:

```
df=df[df['price']> 1000]
df=df[df['price']< 40000]
```

In [23]:

```
df.shape
```

Out[23]:

```
(14742, 4320)
```

Divide dataset into features and label

In [24]:

```
y=df['price']
x=df.drop(['price'], axis=1)
```

In [25]:

```
# Data split into train test
train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=0.25,random_state=10)
```

XGB

In [26]:

```
import xgboost as xgb
xgb=xgb.XGBRegressor()
```

In [27]:

```
xgb.fit(train_x,train_y)
```

Out[27]:

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=
1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_job
s=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alph
a=0,
             reg_lambda=1, ...)
```

In [28]:

```
y_pred=xgb.predict(test_x)
```

In [29]:

```
r2_score(test_y,y_pred)
```

Out[29]:

```
0.8494808147571992
```

In [ ]:

```

```