Problem statement

An American automobile Agency ABC Auto LTD. to enter indian market by setting up their firm, to give competition to their indian counterparts.

They want to understand the factors on which the pricing of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the AIndian market, since those may be very different from the American market. The company wants to know:

Which variables are significant in predicting the price of a car How well those variables describe the price of a car Based on various market surveys, the firm has gathered a large data set of different types of cars across the market.

Goal

They are required A model for the price of cars with the available independent variables. It will be used by the management to understand how exactly the prices vary with the Features. They can accordingly set the price of the cars, the business strategy etc.

To meet requirements they hire you as data scientist to make a machine learning model. The model will be a good way for management to understand the pricing dynamics of a new market.

importing libraries

```
import numpy as np  #importing numpy
import pandas as pd     #importing pandas
import matplotlib.pyplot as plt  #importing matplotlib.pyplot
import seaborn as sns   #importing seaborn
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Loading dataset

```
data=pd.read_csv("/content/car.csv")
```

- **Shape of data**

```
print("shape of data is: ",data.shape)
```

```
shape of data is:  (7632, 11)
```

- **Head of data**

```
data.head()
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
0	12000	merc	GLS Class	2017	Automatic	12046	Diesel	150	37.2	3.0	38000
1	12001	vw	Amarok	2017	Automatic	37683	Diesel	260	36.2	3.0	23495
2	12002	hyundi	Santa Fe	2017	Semi-Auto	32467	Diesel	235	42.8	2.2	18991
3	12003	vw	Arteon	2019	Automatic	1555	Petrol	145	40.4	1.5	22500
4	12004	merc	GLS Class	2019	Automatic	10000	Diesel	145	34.0	3.0	59999

Data cleaning

- Removing carID from dataset as it is not important for analysis.

```
data.drop(["carID"],axis=1,inplace=True)
```

Exploratory data analysis

- **Information of data**

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7632 entries, 0 to 7631
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   brand         7632 non-null   object 
 1   model         7632 non-null   object 
 2   year          7632 non-null   int64  
 3   transmission  7632 non-null   object 
 4   mileage       7632 non-null   int64  
 5   fuelType      7632 non-null   object 
 6   tax           7632 non-null   int64  
 7   mpg           7632 non-null   float64
 8   engineSize    7632 non-null   float64
 9   price         7632 non-null   int64  
dtypes: float64(2), int64(4), object(4)
memory usage: 596.4+ KB
```

Observations:-

- Total 7632 rows and 10 columns.
- Datatypes of 4 columns is int
- Datatypes of 4 columns is string
- Datatypes of 2 columns is float
- No null values in any columns

- **Columns name**

```
data.columns
```

```
Index(['brand', 'model', 'year', 'transmission', 'mileage', 'fuelType', 'tax',
       'mpg', 'engineSize', 'price'],
      dtype='object')
```

Continuous and categorical distribution

```
categorical_col,continuous_col=[],[]    #list for categorical and continuous column
def categorical_continuous_col(data):   #declaring a function
    for col in data.columns:    #for loop to check column type
        if data.dtypes[col]=="object":
            categorical_col.append(col) #appending categorical col
        else:
            continuous_col.append(col)  # appending continuous columns
    return categorical_col,continuous_col
```

```
categorical_col,continuous_col=categorical_continuous_col(data)
```

```
#categorical columns
categorical_col
```

```
['brand', 'model', 'transmission', 'fuelType']
```

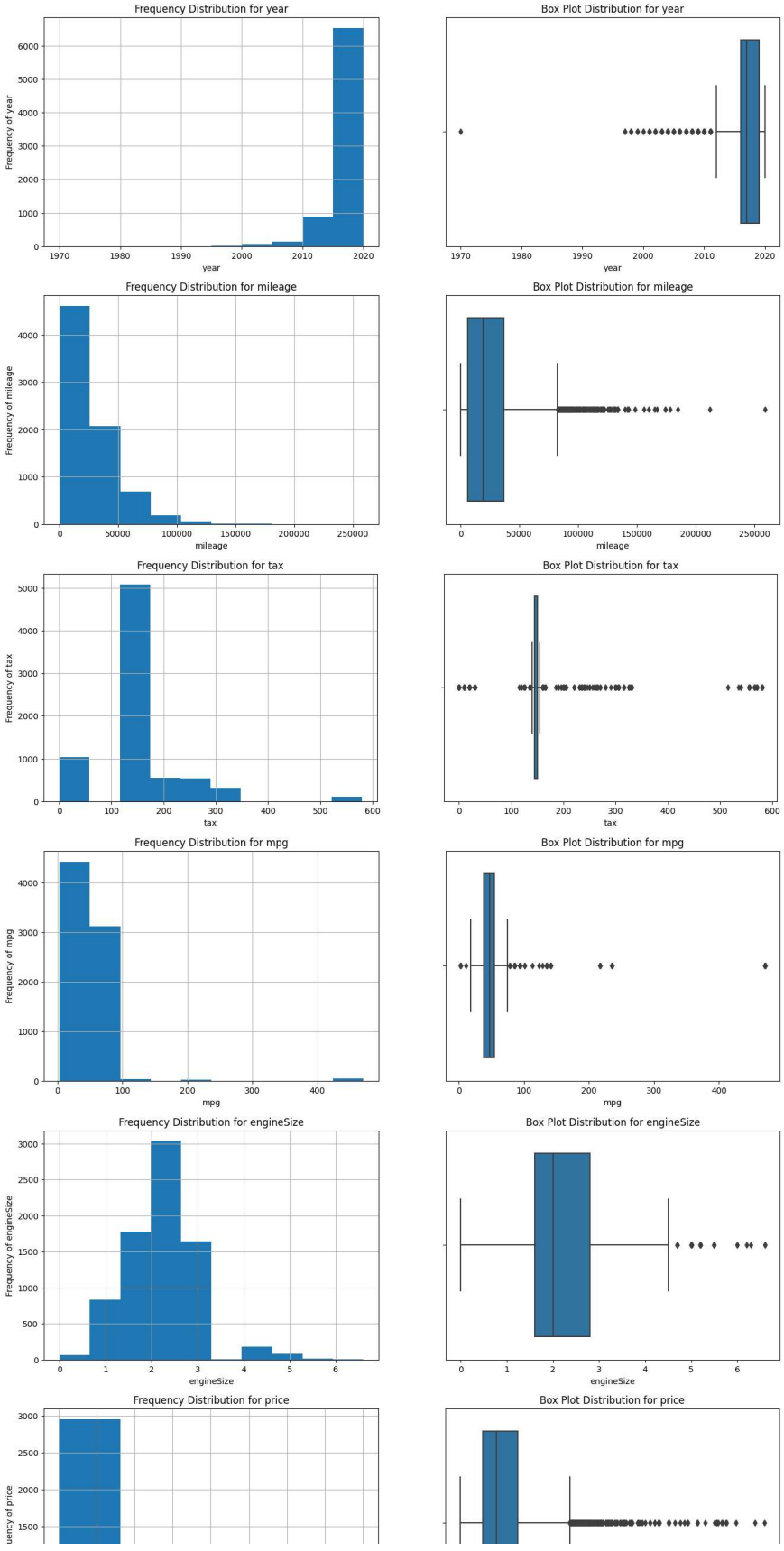
```
#continuous columns
continuous_col
```

```
['year', 'mileage', 'tax', 'mpg', 'engineSize', 'price']
```

Univariate analysis on Continuous column

```
for col in continuous_col:
    plt.figure(figsize = [16,5])    #figure size
    plt.subplot(1,2,1)  #subplot 1
    # Plotting the freq distibution for 'col'
    plt.hist(data[col])
    # To bring the grid structure out of the data
    plt.grid();
```

```
        # Label the x axis
        plt.xlabel(col)
        # Label the y axis
        plt.ylabel("Frequency of {}".format(col))
        # Giving a title on the top
        plt.title("Frequency Distribution for {}".format(col))
        plt.subplot(1,2,2)
        sns.boxplot(data = data, x =col)
        plt.title("Box Plot Distribution for {}".format(col))
        # to remove the array data
        plt.show();
```



Observations:-

- Most of the cars are between 2015 to 2020.There are some outliers as some cars manufactured before 2012.
- Histogram for price is right skewed,most of the cars price is between 20k to 35k USD.
- A few outliers in milage column, some cars run 200-400 mpg(mile-per-gallon).

categorical column

```
#To check values in each categorical column
for col in categorical_col:
```

```
print("*"*50)
print("value counts for {} is ".format(col))
print(data[col].value_counts())
```

```
**************************************************
value counts for brand is
merc        1219
ford        1196
vw          1159
bmw          978
hyundi       817
toyota       717
skoda        638
audi         523
vauxhall     385
Name: brand, dtype: int64
**************************************************
value counts for model is
 Arteon         248
 Grand C-MAX    247
 Santa Fe       245
 Scirocco       242
 CLS Class      237
               ...
 Z3               7
 CLK              7
 Eos              7
 Caddy            6
 Getz             6
Name: model, Length: 90, dtype: int64
**************************************************
value counts for transmission is
Manual       2937
Automatic    2588
Semi-Auto    2105
Other           2
Name: transmission, dtype: int64
**************************************************
value counts for fuelType is
Diesel      4651
Petrol      2655
Hybrid       287
Other         36
Electric       3
Name: fuelType, dtype: int64
```

```
#cateogorical columns
categorical_col
```

```
['brand', 'model', 'transmission', 'fuelType']
```

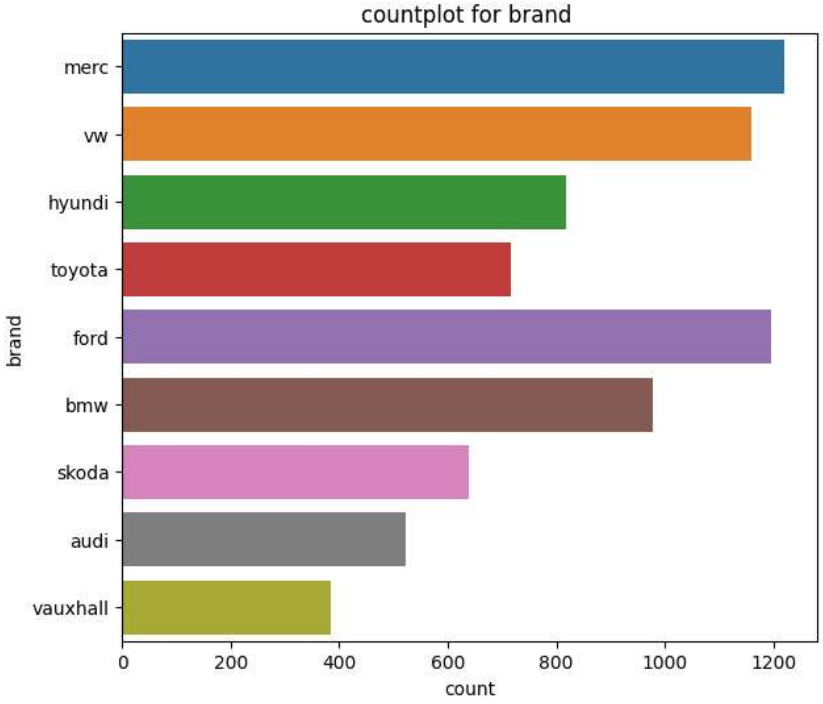
```
#after remove model from categorical column to better visualisation
categorical_col_1=['brand', 'transmission', 'fuelType']
```

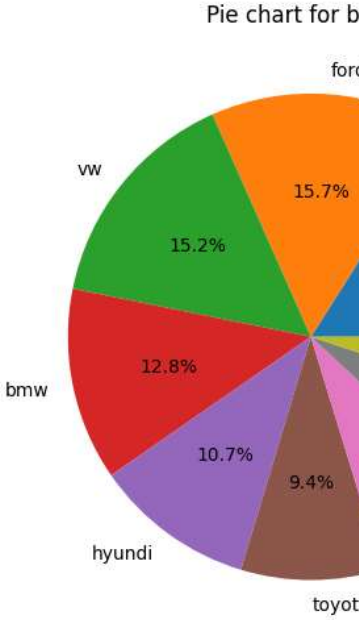
Univariate analysis for categorical columns

- using pie chart and countplot

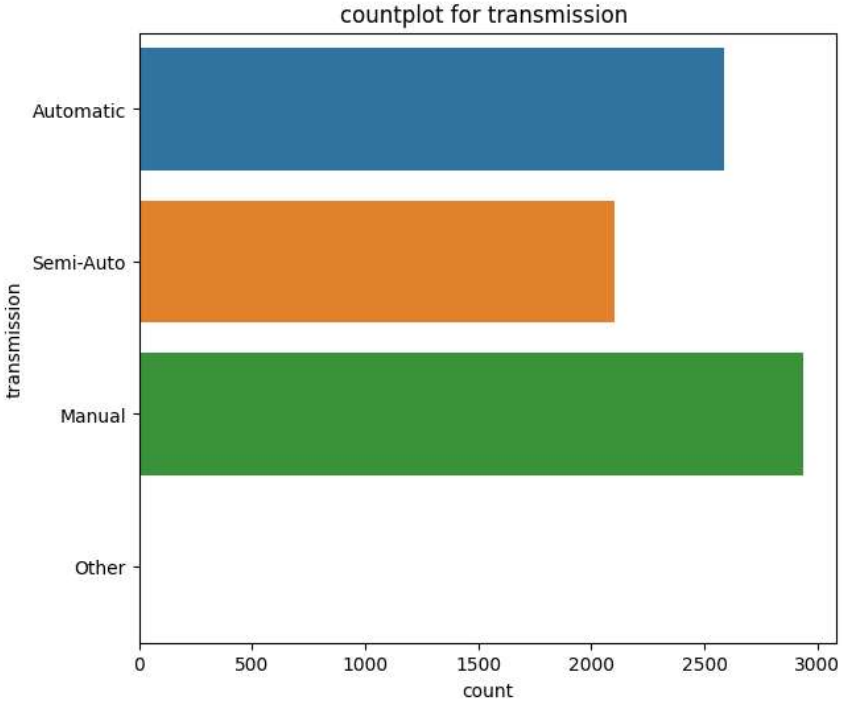
```
for col in categorical_col_1:   #looping for each column in categorical column
    print("*"*50)
    print("distributions for category {} is \n" .format(col),data[col].value_counts())  #value distribution for each category
    plt.figure(figsize=[15,6])  #figure size
    plt.subplot(1,2,1)  #subplot 1
    sns.countplot(data,y=col)   #countplot for all columns
    plt.title("countplot for {} ".format(col))

    plt.subplot(1,2,2)  #subplot 2
    mylabels=data[col].value_counts().index #labeles for pie chart
    y=data[col].value_counts().values   #values for each label in pie chart
    plt.pie(y,labels=mylabels,autopct="%1.1f%%")    #ploting pie chart
    plt.title("Pie chart for {} ".format(col))
    plt.show(); #to remove arrays in output
```

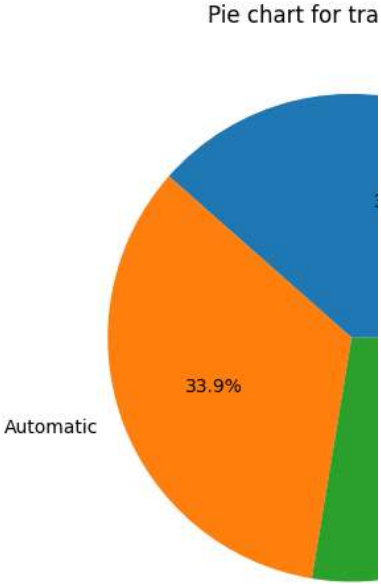
```
**************************************************
distributions for category brand is
 merc        1219
ford         1196
vw           1159
bmw           978
hyundi        817
toyota        717
skoda         638
audi          523
vauxhall      385
Name: brand, dtype: int64
```





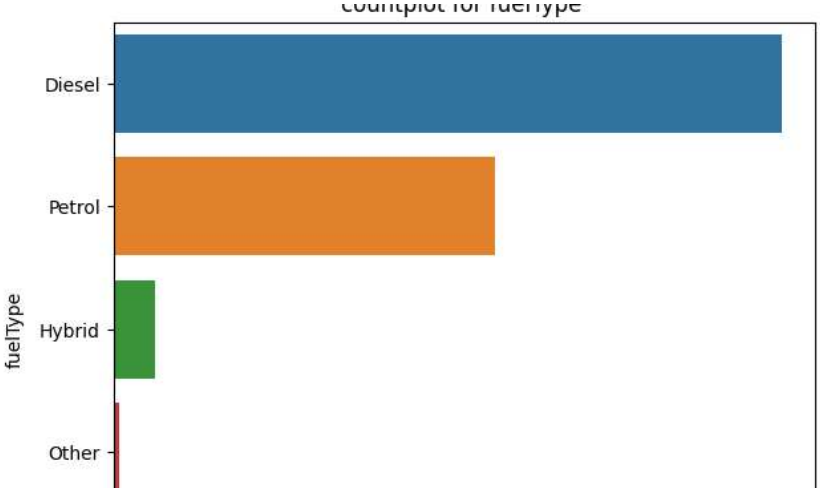

```
**************************************************
distributions for category transmission is
 Manual       2937
Automatic    2588
Semi-Auto    2105
Other           2
Name: transmission, dtype: int64
```

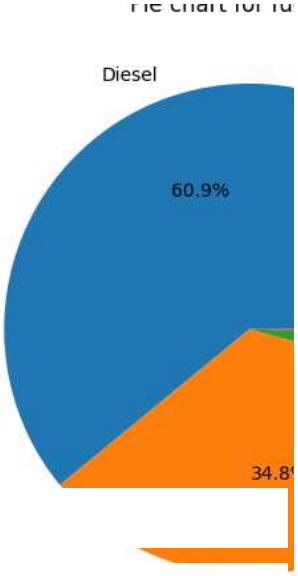





```
**************************************************
distributions for category fuelType is
 Diesel      4651
Petrol      2655
Hybrid       287
Other         36
Electric       3
Name: fuelType, dtype: int64
```

countplot for fuelType Pie chart for fu

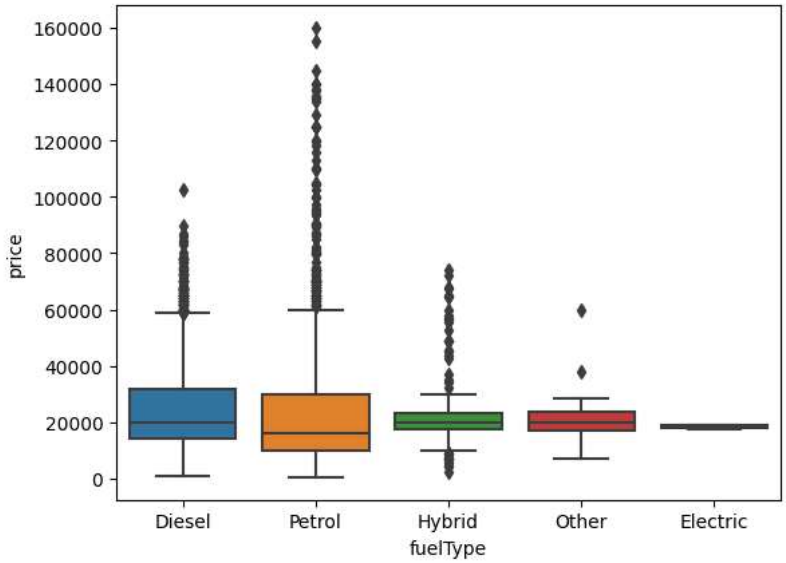




- **Price vs fuel type**

```
#price comparision with each fuel type models
sns.boxplot(x ="fuelType", y ="price",data =data)
```

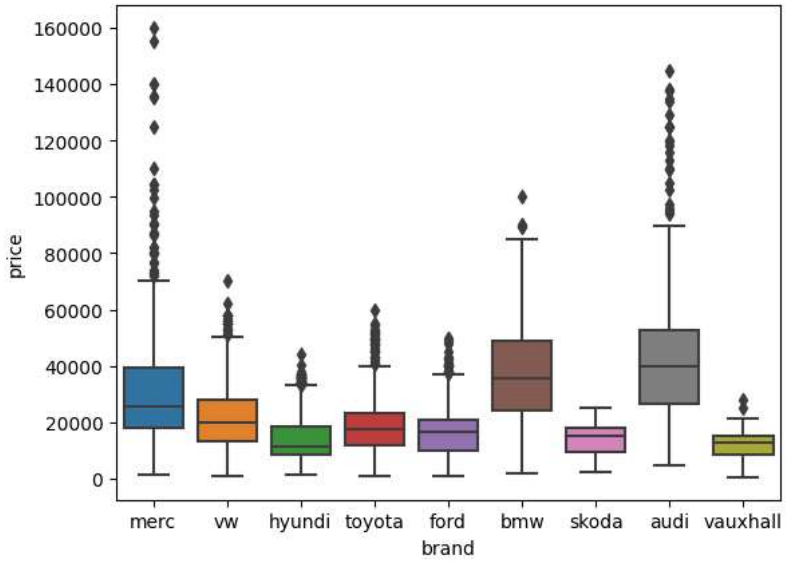
```
<Axes: xlabel='fuelType', ylabel='price'>
```



- **Price vs Brand**

```
sns.boxplot(x ="brand", y ="price",data =data)
```

```
<Axes: xlabel='brand', ylabel='price'>
```



Observation:-

- Most of the cars, almost 60% are operated by Diesel.
- Price of Electric vehicle is much less as compared to others.
- Price of diesel and petrol car is much higher than others,outliers are also detecting.
- Audi is most expensive with some outliers and vauxhell is most affordable brand.

Model

Feature encoding

- Machine only aknowledge continuous data, we need to encode object type data into continuous

```
data.dtypes
```

```
brand            object
model            object
year              int64
transmission     object
mileage           int64
fuelType         object
tax               int64
mpg             float64
engineSize      float64
price             int64
dtype: object
```

- **Label encoding**
- We will use label encoder to encode categorical column into continuous

```
#importing label encoder
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
```

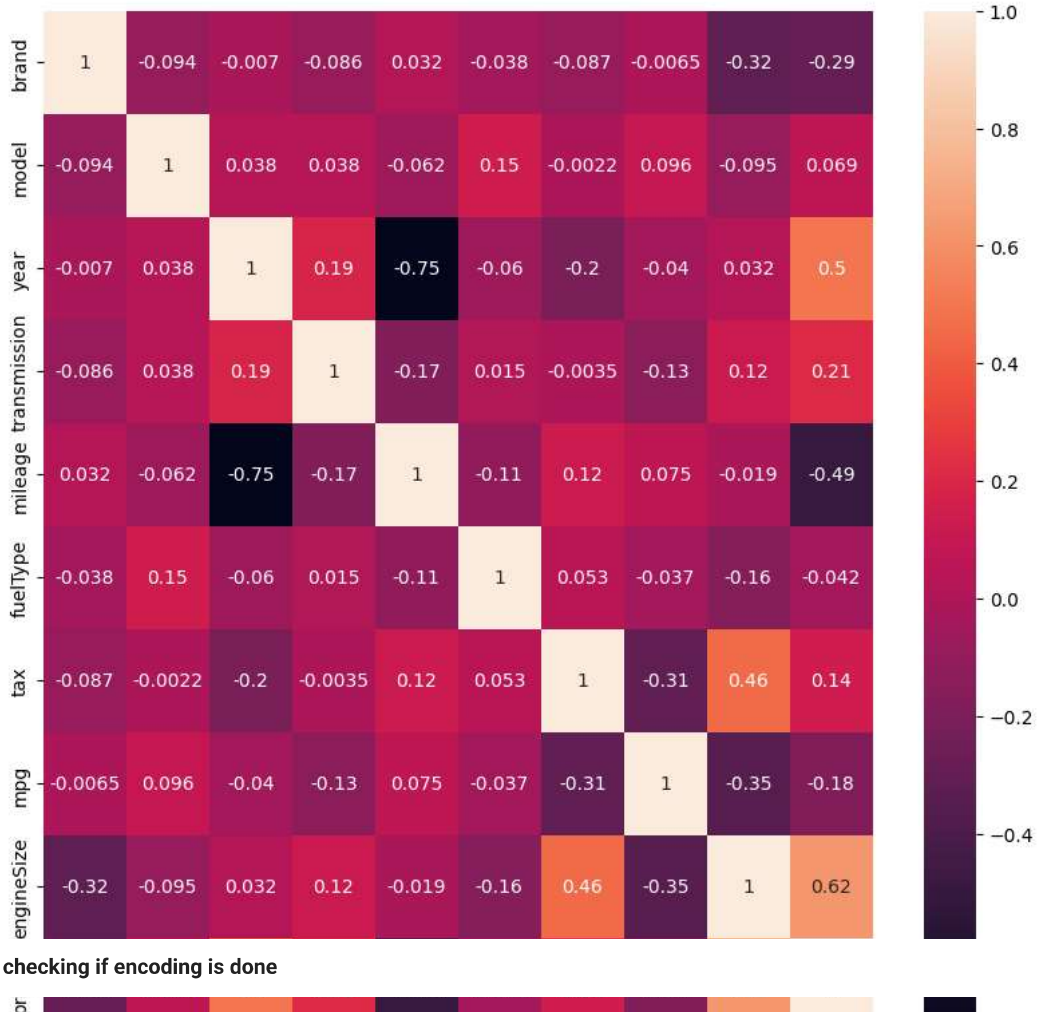
```
for col in categorical_col:
    data[col]=encoder.fit_transform(data[col])
```

```
data.corr()
```

	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
brand	1.000000	-0.093615	-0.006976	-0.086250	0.032212	-0.037775	-0.086550	-0.006525	-0.321437	-0.291048
model	-0.093615	1.000000	0.038252	0.038095	-0.061853	0.145478	-0.002209	0.096490	-0.094664	0.069008
year	-0.006976	0.038252	1.000000	0.187686	-0.753325	-0.060241	-0.200159	-0.040006	0.032041	0.498388
transmission	-0.086250	0.038095	0.187686	1.000000	-0.173706	0.014936	-0.003497	-0.133905	0.118333	0.210677
mileage	0.032212	-0.061853	-0.753325	-0.173706	1.000000	-0.110738	0.124939	0.075106	-0.018777	-0.486950
fuelType	-0.037775	0.145478	-0.060241	0.014936	-0.110738	1.000000	0.053483	-0.036772	-0.159256	-0.042186
tax	-0.086550	-0.002209	-0.200159	-0.003497	0.124939	0.053483	1.000000	-0.311065	0.456532	0.142273
mpg	-0.006525	0.096490	-0.040006	-0.133905	0.075106	-0.036772	-0.311065	1.000000	-0.353857	-0.178982
engineSize	-0.321437	-0.094664	0.032041	0.118333	-0.018777	-0.159256	0.456532	-0.353857	1.000000	0.623995
price	-0.291048	0.069008	0.498388	0.210677	-0.486950	-0.042186	0.142273	-0.178982	0.623995	1.000000

```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True)
```

```
<Axes: >
```



- **checking if encoding is done**

```
data
```

	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
0	4	28	2017	0	12046	0	150	37.2	3.0	38000
1	8	6	2017	0	37683	0	260	36.2	3.0	23495
2	3	69	2017	3	32467	0	235	42.8	2.2	18991
3	8	8	2019	0	1555	4	145	40.4	1.5	22500
4	4	28	2019	0	10000	0	145	34.0	3.0	59999
...	...	...	...	...	...	...	...	...	...	...
7627	6	29	2014	1	18389	4	260	36.2	2.0	14999
7628	5	61	2019	1	12850	4	145	58.9	1.0	9545
7629	2	22	2017	1	9975	0	145	48.7	2.0	21333
7630	4	12	2016	0	25726	4	200	41.5	2.0	18700
7631	7	7	2007	0	99500	0	565	32.8	2.0	2999

7632 rows × 10 columns

```
data2=data[["year",
"engineSize",
"transmission",
"mileage"
,"mpg","price"]]
```

```
data2
```

	year	engineSize	transmission	mileage	mpg	price
0	2017	3.0	0	12046	37.2	38000
1	2017	3.0	0	37683	36.2	23495
2	2017	2.2	3	32467	42.8	18991
3	2019	1.5	0	1555	40.4	22500
4	2019	3.0	0	10000	34.0	59999
...	...	...	...	...	...	...
7627	2014	2.0	1	18389	36.2	14999
7628	2019	1.0	1	12850	58.9	9545

- **Preparing train and test data**

```
from sklearn.model_selection import train_test_split    #importing train test split
x=data2.drop(["price"],axis=1)   # Except for price all the other data is my features. Hence, we need to drop it.
y=data2["price"]                         # price is the target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)     #telling our model that 80% is training data and 20%
```

```
data2.shape
```

```
(7632, 6)
```

```
x_train.shape
```

```
(6105, 5)
```

```
x_test.shape
```

```
(1527, 5)
```

```
x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=0.2)
```

```
x_train.shape,x_val.shape,x_test.shape,y_train.shape,y_val.shape,y_test.shape
```

- **Normalization of features by standard scaler**

```
from sklearn.preprocessing import StandardScaler
scale=StandardScaler()
x_train=scale.fit_transform(x_train)
x_val=scale.transform(x_val)
x_test=scale.transform(x_test)
```

Importing all regression model

```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.2-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 8.6 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.23.5)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.11.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2023.3.post1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.1.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (8.2.3)
Installing collected packages: catboost
Successfully installed catboost-1.2.2
```

```
from sklearn.linear_model import LinearRegression   #importing linearregression model
from sklearn.linear_model import Ridge  #importing Ridge
from sklearn.linear_model import Lasso  #importing Lasso
from sklearn.linear_model import ElasticNet #importing Elasticnet
from sklearn.svm import SVR #importing svr
from sklearn.ensemble import RandomForestRegressor #importing Random_forest
from sklearn.tree import DecisionTreeRegressor #importing Decision tree
from sklearn.neighbors import KNeighborsRegressor
from catboost import CatBoostRegressor
```

- **KPI**

```
#importing performance metrics
from sklearn.metrics import r2_score,mean_squared_error


r2_score_list_test=[]    #list to store r2 score of test data for all model
r2_score_list_val=[]    #list to store r2 score of val data for all model
r2_score_list_train=[]  #list to store r2 score of train data for all model
mean_sqr_error_test=[]   #list to store mean squared error of test data for all model
mean_sqr_error_val=[]   #list to store mean squared error of val data for all model
mean_sqr_error_train=[] #list to store mean squared error of train data for all model
r_mean_sqr_error_train=[]
r_mean_sqr_error_val=[]
r_mean_sqr_error_test=[]

#model list
model_list=[LinearRegression(),
               Ridge(),
               Lasso(),
               ElasticNet(),
            SVR(kernel='linear'),
            RandomForestRegressor(),
            DecisionTreeRegressor(),
            KNeighborsRegressor(),
            CatBoostRegressor()
               ]
for model in model_list:    #looping each model
    model.fit(x_train,y_train)  #training to each model with training data
    test_predtiction=model.predict(x_test)  #prediction on test data
    val_predtiction=model.predict(x_val)  #prediction on val data
    train_prediction=model.predict(x_train) #prediction on train data
    r2_score_list_test.append(r2_score(y_test,test_predtiction))    #r2 score for test data
    r2_score_list_val.append(r2_score(y_val,val_predtiction))    #r2 score for val data
    r2_score_list_train.append(r2_score(y_train,train_prediction))   #r2 score for train data
    mean_sqr_error_test.append(mean_squared_error(y_test,test_predtiction))    #MSA on test data
    mean_sqr_error_val.append(mean_squared_error(y_val,val_predtiction))    #MSA on test data
    mean_sqr_error_train.append(mean_squared_error(y_train,train_prediction))   #MSA on train data

    r_mean_sqr_error_test.append(sqrt(mean_squared_error(y_test,test_predtiction)))
    r_mean_sqr_error_val.append(sqrt(mean_squared_error(y_val,val_predtiction)))
    r_mean_sqr_error_train.append(sqrt(mean_squared_error(y_train,train_prediction)))
```

```
Learning rate set to 0.052602
0:      learn: 15642.4602215    total: 48.4ms   remaining: 48.3s
1:      learn: 15061.4014881    total: 52.7ms   remaining: 26.3s
2:      learn: 14522.2775202    total: 56.4ms   remaining: 18.7s
3:      learn: 14003.3969783    total: 60ms     remaining: 14.9s
4:      learn: 13550.4791754    total: 63.5ms   remaining: 12.6s
5:      learn: 13106.1268937    total: 68ms     remaining: 11.3s
6:      learn: 12670.5516152    total: 71.4ms   remaining: 10.1s
7:      learn: 12278.9347931    total: 75ms     remaining: 9.3s
8:      learn: 11910.4669673    total: 78.7ms   remaining: 8.66s
9:      learn: 11546.2139871    total: 82.2ms   remaining: 8.13s
10:     learn: 11224.6088713    total: 85.7ms   remaining: 7.71s
11:     learn: 10910.6458948    total: 89.3ms   remaining: 7.35s
12:     learn: 10613.3459015    total: 93ms     remaining: 7.06s
13:     learn: 10338.2365288    total: 96.5ms   remaining: 6.8s
14:     learn: 10089.8251242    total: 100ms    remaining: 6.57s
15:     learn: 9837.4220797     total: 104ms    remaining: 6.38s
16:     learn: 9605.9993764     total: 107ms    remaining: 6.2s
17:     learn: 9390.9109721     total: 111ms    remaining: 6.05s
18:     learn: 9192.3568451     total: 114ms    remaining: 5.91s
19:     learn: 8999.7005502     total: 118ms    remaining: 5.77s
20:     learn: 8810.8082390     total: 122ms    remaining: 5.67s
21:     learn: 8646.6624387     total: 125ms    remaining: 5.56s
22:     learn: 8483.8584567     total: 129ms    remaining: 5.46s
23:     learn: 8338.1528571     total: 132ms    remaining: 5.37s
24:     learn: 8205.6519692     total: 136ms    remaining: 5.29s
25:     learn: 8068.1117265     total: 139ms    remaining: 5.21s
26:     learn: 7963.1506912     total: 143ms    remaining: 5.14s
27:     learn: 7847.0185253     total: 146ms    remaining: 5.07s
28:     learn: 7733.9861606     total: 150ms    remaining: 5.01s
```

```
29:     learn: 7643.8427277     total: 153ms    remaining: 4.95s
30:     learn: 7537.5740984     total: 157ms    remaining: 4.9s
31:     learn: 7442.7279347     total: 161ms    remaining: 4.86s
32:     learn: 7350.5309973     total: 165ms    remaining: 4.84s
33:     learn: 7268.6886164     total: 168ms    remaining: 4.77s
34:     learn: 7179.1214042     total: 171ms    remaining: 4.73s
35:     learn: 7105.8264794     total: 175ms    remaining: 4.69s
36:     learn: 7024.4649387     total: 185ms    remaining: 4.81s
37:     learn: 6952.7964563     total: 190ms    remaining: 4.81s
38:     learn: 6888.6665757     total: 192ms    remaining: 4.73s
39:     learn: 6838.0341979     total: 196ms    remaining: 4.71s
40:     learn: 6779.0549879     total: 200ms    remaining: 4.67s
41:     learn: 6724.7571368     total: 203ms    remaining: 4.63s
42:     learn: 6670.9973638     total: 208ms    remaining: 4.62s
43:     learn: 6627.1621276     total: 211ms    remaining: 4.59s
44:     learn: 6588.3484253     total: 215ms    remaining: 4.57s
45:     learn: 6545.4354059     total: 219ms    remaining: 4.54s
46:     learn: 6499.5649135     total: 223ms    remaining: 4.51s
47:     learn: 6456.3703877     total: 226ms    remaining: 4.48s
48:     learn: 6413.7491732     total: 229ms    remaining: 4.45s
49:     learn: 6372.8547978     total: 233ms    remaining: 4.43s
50:     learn: 6329.1267203     total: 240ms    remaining: 4.47s
51:     learn: 6293.1962410     total: 244ms    remaining: 4.44s
52:     learn: 6258.3301247     total: 250ms    remaining: 4.46s
53:     learn: 6227.9066807     total: 252ms    remaining: 4.42s
54:     learn: 6196.5732728     total: 258ms    remaining: 4.43s
55:     learn: 6162.5903525     total: 263ms    remaining: 4.43s
56:     learn: 6135.6886765     total: 265ms    remaining: 4.38s
```

```
import math
from math import sqrt


#DataFrame to store r2_score of all the models
r2_score_df=pd.DataFrame({"model":model_list,
                              "r2_score_train":r2_score_list_train,
                          "r2_score_val":r2_score_list_val,
                              "r2_score_test":r2_score_list_test})


#DataFrame to store MSA of all the models
mse_score_df=pd.DataFrame({"model":model_list,
                              "mean_squared_error_train":mean_sqr_error_train,
                              "mean_squared_error_val":mean_sqr_error_val,
                              "mean_squared_error_test":mean_sqr_error_test})


#DataFrame to store MSA of all the models
r_mse_score_df=pd.DataFrame({"model":model_list,
                              "r_mean_squared_error_train":r_mean_sqr_error_train,
                              "r_mean_squared_error_val":r_mean_sqr_error_val,
                              "r_mean_squared_error_test":r_mean_sqr_error_test})


r2_score_df
```

	model	r2_score_train	r2_score_val	r2_score_test
0	LinearRegression()	0.669922	0.675631	0.677090
1	Ridge()	0.669922	0.675649	0.677083
2	Lasso()	0.669922	0.675648	0.677079
3	ElasticNet()	0.611637	0.631577	0.608078
4	SVR(kernel='linear')	0.287931	0.304732	0.279274
5	(DecisionTreeRegressor(max_features=1.0, rando...	0.993703	0.950772	0.948878
6	DecisionTreeRegressor()	0.999864	0.924058	0.921804
7	KNeighborsRegressor()	0.944935	0.915256	0.908207
8	<catboost.core.CatBoostRegressor object at 0x7...	0.985439	0.962919	0.965815

```
mse_score_df
```

	model	mean_squared_error_train	mean_squared_error_val	mean_squared_error_test
0	LinearRegression()	8.813018e+07	8.472499e+07	9.476004e+07
1	Ridge()	8.813019e+07	8.472044e+07	9.476200e+07
2	Lasso()	8.813019e+07	8.472059e+07	9.476345e+07

```
r_mse_score_df
```

	model	r_mean_squared_error_train	r_mean_squared_error_val	r_mean_squared_error_test
0	LinearRegression()	9387.767710	9204.617791	9734.476914
1	Ridge()	9387.768176	9204.370450	9734.577687
2	Lasso()	9387.768217	9204.378970	9734.651960
3	ElasticNet()	10182.939411	9809.787186	10724.375202
4	SVR(kernel='linear')	13788.440854	13476.047956	14543.103809
5	(DecisionTreeRegressor(max_features=1.0, rando...	1296.617206	3585.834988	3873.248070
6	DecisionTreeRegressor()	190.849070	4453.751879	4790.328270
7	KNeighborsRegressor()	3834.362463	4704.808805	5190.125673
8	<catboost.core.CatBoostRegressor object at 0x7...	1971.711032	3112.174318	3167.285383