Linear Regression- Auto Dataset

Import Libraries

In [1]:

```

# common libraries
import numpy as np
import pandas as pd

# Visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Feature selection
from statsmodels.stats.outliers_influence import variance_inflation_factor

# model building
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearch
from sklearn.linear_model import LinearRegression, Ridge, Lasso

# Evaluaion Matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# To avoid warning

import warnings
warnings.filterwarnings("ignore")

# To handle outliers
from scipy.stats import boxcox

# To check Normality of Residual
from scipy.stats import shapiro, kstest, normaltest
import statsmodels.api as sm

# To create pickle file
import pickle

# To create json file
import json
```

Step 1: Problem Statement

```
To predict price of the cars by using supervised machine learning considering as a regression problem.
```

Step 2: Data Gathering

In [145]:

```
df = pd.read_csv("autos_dataset.csv")
df
```

Out[145]:

nake	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore
alfa-nero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47
alfa-nero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47
alfa-nero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68
audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19
audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19
...	...	...	...	...	...	...	...	...	...	...	...
/olvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78
/olvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78
/olvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58
/olvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01
/olvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78

Step 3: Exploratory Data Analysis

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       205 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64  
 14  engine-type        205 non-null    object 
 15  num-of-cylinders   205 non-null    object 
 16  engine-size        205 non-null    int64  
 17  fuel-system        205 non-null    object 
 18  bore               205 non-null    object 
 19  stroke             205 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object 
 22  peak-rpm           205 non-null    object 
 23  city-mpg           205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              205 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

In [4]:

```
df.describe()
```

Out[4]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000

In [5]:

```
# finding the no. of rows and columns

no_of_rows = df.shape[0]
print("No. of rows: ", no_of_rows)

no_of_columns = df.shape[1]
print("No. of columns: ", no_of_columns)
```

```
No. of rows:  205
No. of columns:  26
```

In [6]:

```
# Checking for missing values

df.isna().sum()                  # There is no missing values in this dataset
```

Out[6]:

```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

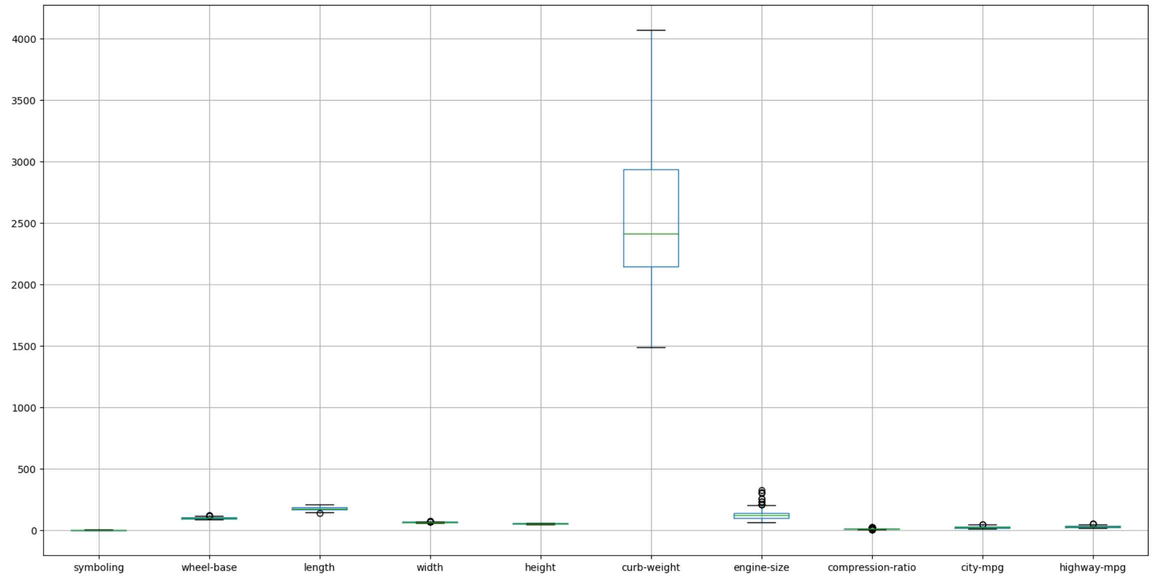
In [7]:

```
## Checking for outliers

plt.figure(figsize=(20,10))
df.boxplot()
```

Out[7]:

```
<AxesSubplot:>
```



From the boxplot we can see that there is outliers present in some of the columns

Step 4: Feature Engineering

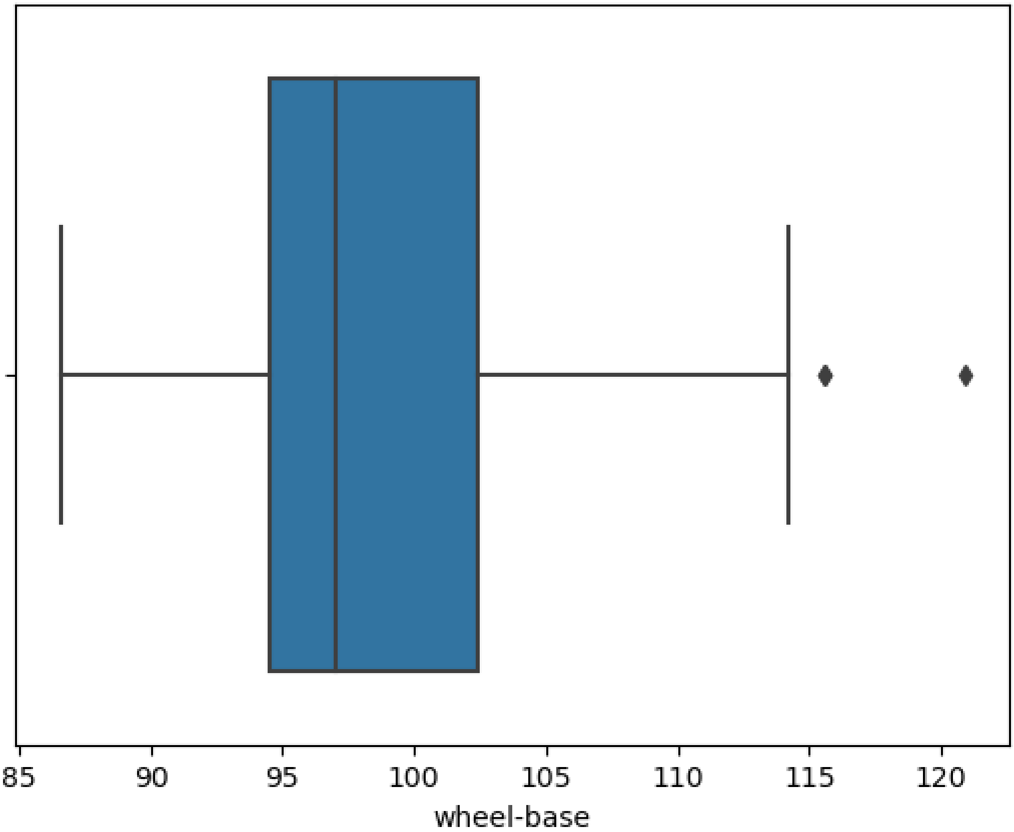
Outliers Handling

In [8]:

```
sns.boxplot(x = df['wheel-base'])
```

Out[8]:

```
<AxesSubplot:xlabel='wheel-base'>
```



In [9]:

```
# Ouliers finding by using IQR Method

q1=df['wheel-base'].quantile(0.25)
q2=df['wheel-base'].quantile(0.50)
q3=df['wheel-base'].quantile(0.75)
median=df['wheel-base'].median()
IQR=q3-q1
upper_tail = q3 + 1.5 * IQR
lower_tail = q1 - 1.5 * IQR
print("Q1 :", q1)
print("Q2 :", q2)
print("Q3 :", q3)
print("Median :",median)
print("upper_tail :", upper_tail)
print("lower_tail :", lower_tail)
```

```
Q1 : 94.5
Q2 : 97.0
Q3 : 102.4
Median : 97.0
upper_tail : 114.25000000000001
lower_tail : 82.64999999999999
```

In [10]:

```
# Detecting outliers in the column

df['wheel-base'].loc[(df['wheel-base'] > upper_tail)]
```

Out[10]:

```
70    115.6
71    115.6
73    120.9
Name: wheel-base, dtype: float64
```

In [11]:

```
# Calculating the median of good data of Item_Visibility column

median_wheel_base = df['wheel-base'].loc[(df['wheel-base'] <= upper_tail)].median()
median_wheel_base
```

Out[11]:

```
96.9
```

In [12]:

```
# Imputing outliers by median value

df['wheel-base'].loc[(df['wheel-base'] > upper_tail)] = median_wheel_base
```

In [13]:

```
# Checking outliers after imputation

df[['wheel-base']].loc[(df['wheel-base'] > upper_tail)]
```

Out[13]:

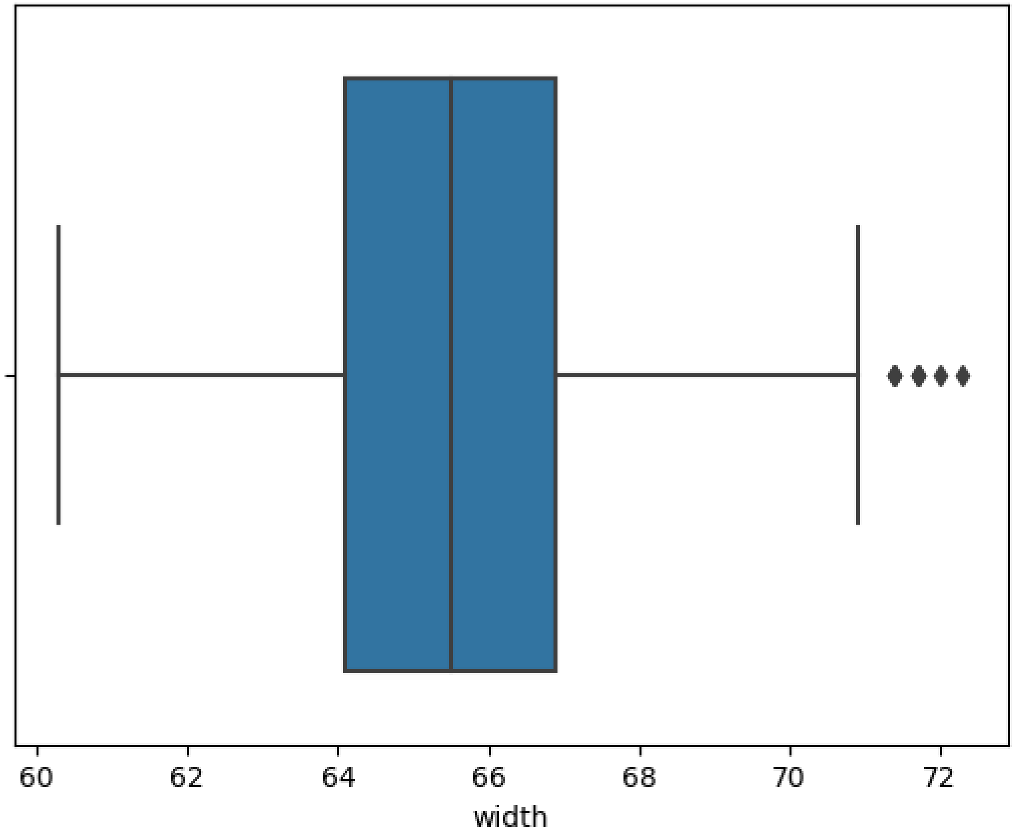
wheel-base

In [14]:

```
sns.boxplot(x = df['width'])
```

Out[14]:

```
<AxesSubplot:xlabel='width'>
```



In [15]:

```
# Ouliers finding by using IQR Method

q1= df['width'].quantile(0.25)
q2= df['width'].quantile(0.50)
q3= df['width'].quantile(0.75)
median= df['width'].median()
IQR=q3-q1
upper_tail = q3 + 1.5 * IQR
lower_tail = q1 - 1.5 * IQR
print("Q1 :", q1)
print("Q2 :", q2)
print("Q3 :", q3)
print("Median :",median)
print("upper_tail :", upper_tail)
print("lower_tail :", lower_tail)
```

```
Q1 : 64.1
Q2 : 65.5
Q3 : 66.9
Median : 65.5
upper_tail : 71.10000000000002
lower_tail : 59.89999999999998
```

In [16]:

```
# Detecting outliers in the column

df['width'].loc[(df['width'] > upper_tail)]
```

Out[16]:

```
6      71.4
7      71.4
8      71.4
70     71.7
71     71.7
73     71.7
74     72.0
129    72.3
Name: width, dtype: float64
```

In [17]:

```
# Calculating the median of good data of Item_Visibility column

median_width = df['width'].loc[(df['width'] <= upper_tail)].median()
median_width
```

Out[17]:

```
65.4
```

In [18]:

```
# Imputing outliers by median value

df['width'].loc[(df['width'] > upper_tail)] = median_width
```

In [19]:

```
# Checking outliers after imputation
df[['width']].loc[(df['width'] > upper_tail)]
```

Out[19]:

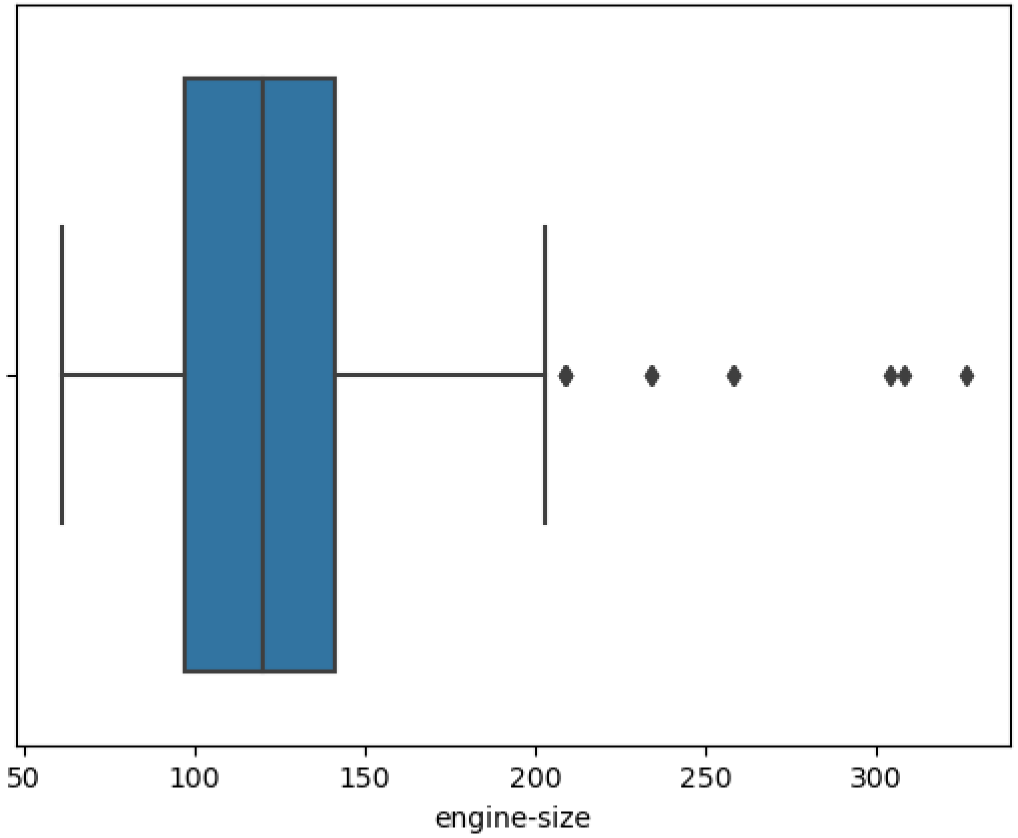
width

In [20]:

```
sns.boxplot(x = df['engine-size'])
```

Out[20]:

```
<AxesSubplot:xlabel='engine-size'>
```



In [21]:

```
# Ouliers finding by using IQR Method

q1= df['engine-size'].quantile(0.25)
q2= df['engine-size'].quantile(0.50)
q3= df['engine-size'].quantile(0.75)
median= df['engine-size'].median()
IQR=q3-q1
upper_tail = q3 + 1.5 * IQR
lower_tail = q1 - 1.5 * IQR
print("Q1 :", q1)
print("Q2 :", q2)
print("Q3 :", q3)
print("Median :",median)
print("upper_tail :", upper_tail)
print("lower_tail :", lower_tail)
```

```
Q1 : 97.0
Q2 : 120.0
Q3 : 141.0
Median : 120.0
upper_tail : 207.0
lower_tail : 31.0
```

In [22]:

```
# Detecting outliers in the column

df['engine-size'].loc[(df['engine-size'] > upper_tail)]
```

Out[22]:

```
15    209
16    209
17    209
47    258
48    258
49    326
71    234
72    234
73    308
74    304
Name: engine-size, dtype: int64
```

In [23]:

```
# Imputing outliers by upper_tail value

df['engine-size'].loc[(df['engine-size'] > upper_tail)] = upper_tail
```

In [24]:

```
# Checking outliers after imputation

df[['engine-size']].loc[(df['engine-size'] > upper_tail)]
```

Out[24]:

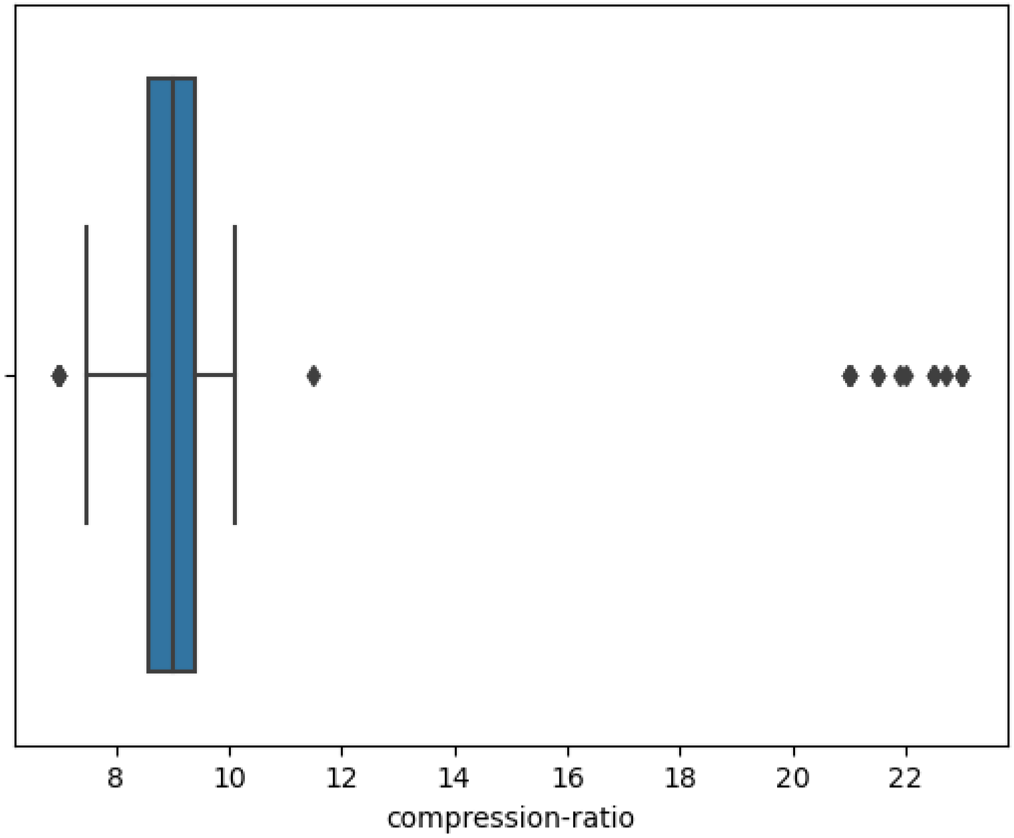
engine-size

In [25]:

```
sns.boxplot(x = df['compression-ratio'])
```

Out[25]:

```
<AxesSubplot:xlabel='compression-ratio'>
```



In [26]:

```
# Ouliers finding by using IQR Method

q1= df['compression-ratio'].quantile(0.25)
q2= df['compression-ratio'].quantile(0.50)
q3= df['compression-ratio'].quantile(0.75)
median= df['compression-ratio'].median()
IQR=q3-q1
upper_tail = q3 + 1.5 * IQR
lower_tail = q1 - 1.5 * IQR
print("Q1 :", q1)
print("Q2 :", q2)
print("Q3 :", q3)
print("Median :",median)
print("upper_tail :", upper_tail)
print("lower_tail :", lower_tail)
```

```
Q1 : 8.6
Q2 : 9.0
Q3 : 9.4
Median : 9.0
upper_tail : 10.600000000000001
lower_tail : 7.399999999999999
```

In [27]:

```
# Detecting outliers in the column

outliers = df['compression-ratio'].loc[(df['compression-ratio'] > upper_tail) | (df[
outliers
```

Out[27]:

```
9       7.0
29      7.0
49     11.5
63     22.7
66     22.0
67     21.5
68     21.5
69     21.5
70     21.5
82      7.0
83      7.0
84      7.0
90     21.9
108    21.0
110    21.0
112    21.0
114    21.0
116    21.0
117     7.0
124     7.0
158    22.5
159    22.5
174    22.5
182    23.0
184    23.0
187    23.0
192    23.0
203    23.0
Name: compression-ratio, dtype: float64
```

In [28]:

```
# Imputing outliers by upper_tail and lower_tail values

df['compression-ratio'].loc[(df['compression-ratio'] > upper_tail)] = upper_tail
df['compression-ratio'].loc[(df['compression-ratio'] < lower_tail)] = lower_tail
```

In [29]:

```
# Checking outliers after imputation

df[['compression-ratio']].loc[(df['compression-ratio'] > upper_tail) | (df['compress
```

Out[29]:

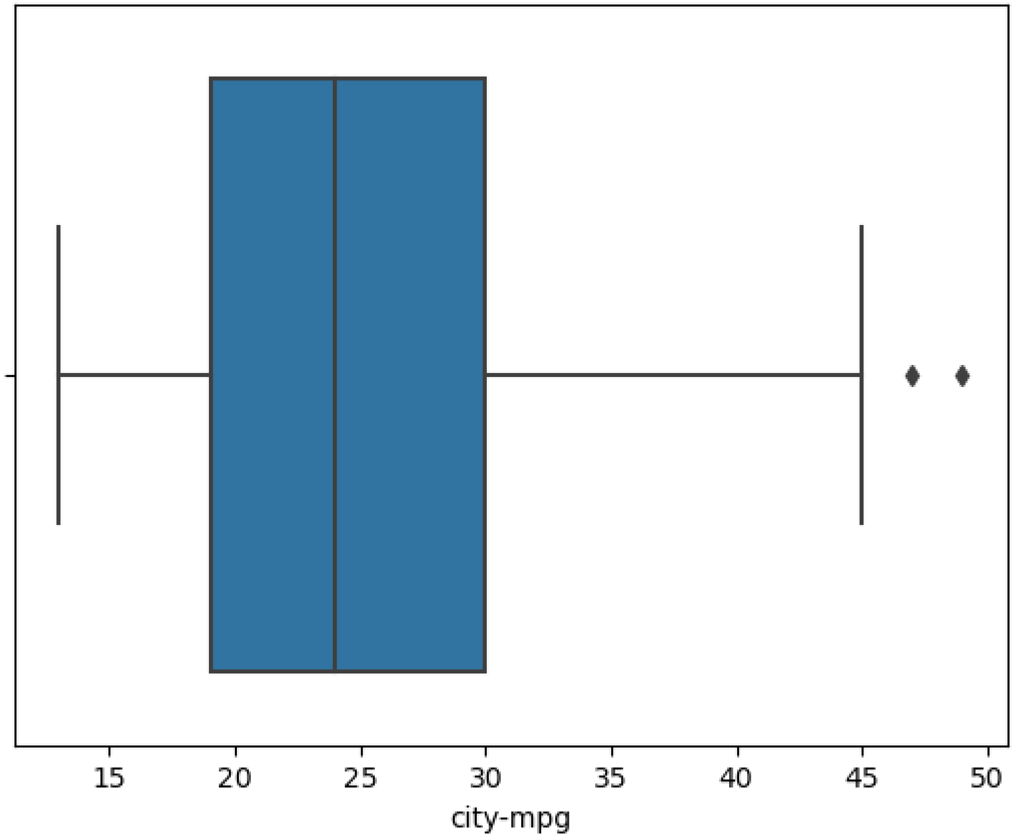
compression-ratio

In [30]:

```
sns.boxplot(x = df['city-mpg'])
```

Out[30]:

```
<AxesSubplot:xlabel='city-mpg'>
```



In [31]:

```
# Ouliers finding by using IQR Method

q1= df['city-mpg'].quantile(0.25)
q2= df['city-mpg'].quantile(0.50)
q3= df['city-mpg'].quantile(0.75)

IQR=q3-q1
upper_tail = q3 + 1.5 * IQR
lower_tail = q1 - 1.5 * IQR
print("Q1 :", q1)
print("Q2 :", q2)
print("Q3 :", q3)

print("upper_tail :", upper_tail)
print("lower_tail :", lower_tail)
```

```
Q1 : 19.0
Q2 : 24.0
Q3 : 30.0
upper_tail : 46.5
lower_tail : 2.5
```

In [32]:

```
# Detecting outliers in the column

df['city-mpg'].loc[(df['city-mpg'] > upper_tail)]

```

Out[32]:

```
18    47
30    49
Name: city-mpg, dtype: int64
```

In [33]:

```
# Imputation of outliers with upper_tail,value

df['city-mpg'].loc[(df['city-mpg'] > upper_tail)] = upper_tail
```

In [34]:

```
# Rechecking outliers after imputation

df[['city-mpg']].loc[(df['city-mpg'] > upper_tail)]
```

Out[34]:

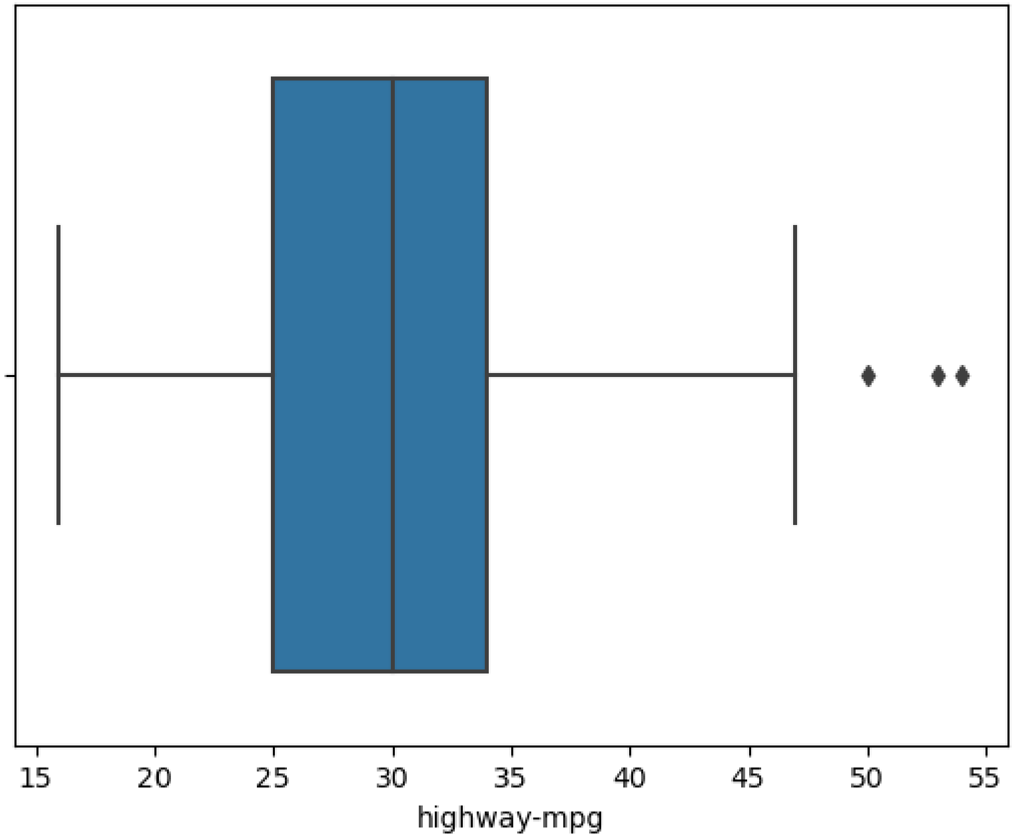
city-mpg

In [35]:

```
sns.boxplot(x = df['highway-mpg'])
```

Out[35]:

<AxesSubplot:xlabel='highway-mpg'>



In [36]:

```
# Ouliers finding by using IQR Method

q1= df['highway-mpg'].quantile(0.25)
q2= df['highway-mpg'].quantile(0.50)
q3= df['highway-mpg'].quantile(0.75)

IQR=q3-q1
upper_tail = q3 + 1.5 * IQR
lower_tail = q1 - 1.5 * IQR
print("Q1 :", q1)
print("Q2 :", q2)
print("Q3 :", q3)

print("upper_tail :", upper_tail)
print("lower_tail :", lower_tail)
```

```
Q1 : 25.0
Q2 : 30.0
Q3 : 34.0
upper_tail : 47.5
lower_tail : 11.5
```

In [37]:

```
# Detecting outliers in the column

df['highway-mpg'].loc[(df['highway-mpg'] > upper_tail)]
```

Out[37]:

```
18    53
30    54
90    50
Name: highway-mpg, dtype: int64
```

In [38]:

```
# Imputation of outliers with upper_tail,value

df['highway-mpg'].loc[(df['highway-mpg'] > upper_tail)] = upper_tail
```

In [39]:

```
# Rechecking the outliers after imputation

df[['highway-mpg']].loc[(df['highway-mpg'] > upper_tail)]
```

Out[39]:

highway-mpg

Converting Categorical Columns into Numerical column

In [40]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       205 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64  
 14  engine-type        205 non-null    object 
 15  num-of-cylinders   205 non-null    object 
 16  engine-size        205 non-null    int64  
 17  fuel-system        205 non-null    object 
 18  bore               205 non-null    object 
 19  stroke             205 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object 
 22  peak-rpm           205 non-null    object 
 23  city-mpg           205 non-null    float64
 24  highway-mpg        205 non-null    float64
 25  price              205 non-null    object 
dtypes: float64(7), int64(3), object(16)
memory usage: 41.8+ KB
```

1. normalized-losses

In [147]:

```
df['normalized-losses']
```

Out[147]:

```
0        ?
1        ?
2        ?
3      164
4      164
      ...
200     95
201     95
202     95
203     95
204     95
Name: normalized-losses, Length: 205, dtype: object
```

In [148]:

```
df['normalized-losses'].value_counts()
```

Out[148]:

```
?      41
161    11
91      8
150     7
134     6
128     6
104     6
85      5
94      5
65      5
102     5
74      5
168     5
103     5
95      5
106     4
93      4
118     4
```

In [149]:

```
df['normalized-losses'].replace({"?": np.nan},inplace=True)
```

In [150]:

```
df.replace({"?": np.nan},inplace=True)
```

In [151]:

```
# Checking for missing value
df['normalized-losses'].isna().sum()
```

Out[151]:

41

In [152]:

```
# converting into numeric data type

df['normalized-losses'] = df['normalized-losses'].astype(float)
```

```
# Outliers checking

sns.boxplot(x = df['normalized-losses'])
```

In [154]:

```
# imputaion of missing value with median value because outliers are present in the da

df['normalized-losses'].fillna(df['normalized-losses'].median(), inplace=True)
```

In [155]:

```
# Rechecking for missing value
df['normalized-losses'].isna().sum()
```

Out[155]:

0

2. make

In [156]:

```
df['make'].isna().sum()
```

Out[156]:

0

In [157]:

```
df['make'].nunique()
```

Out[157]:

22

In [158]:

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

Out[158]:

```
toyota           32
nissan           18
mazda            17
mitsubishi       13
honda            13
volkswagen       12
subaru           12
peugot           11
volvo            11
dodge             9
mercedes-benz     8
bmw               8
audi              7
plymouth          7
saab              6
porsche           5
isuzu             4
jaguar            3
```

In [159]:

```
# Converting categorical values into numeric value by using get_dummies

df = pd.get_dummies(df, columns = ['make'])
```

3. fuel-type

In [217]:

```
df['fuel-type'].value_counts()
```

Out[217]:

```
1    185
0     20
Name: fuel-type, dtype: int64
```

In [162]:

```
df['fuel-type'].replace({"gas":1, "diesel":0}, inplace=True)
```

In [163]:

```
fuel_type_dict = {"gas":1, "diesel":0}
```

In [164]:

```
df['fuel-type'].value_counts()
```

Out[164]:

```
1    185
0     20
Name: fuel-type, dtype: int64
```

4. aspiration

In [165]:

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

Out[165]:

```
std      168
turbo     37
Name: aspiration, dtype: int64
```

In [166]:

```
df['aspiration'].value_counts().to_dict()
```

Out[166]:

```
{'std': 168, 'turbo': 37}
```

In [167]:

```
df['aspiration'].replace({'std': 0, 'turbo': 1}, inplace=True)
```

In [168]:

```
df['aspiration'].value_counts().to_dict()
```

Out[168]:

```
{0: 168, 1: 37}
```

In [181]:

```
aspiration_dict = {'std': 0, 'turbo': 1}
```

5. num-of-doors

In [182]:

```
df['num-of-doors'].unique()
```

Out[182]:

```
array([2, 4], dtype=int64)
```

In [183]:

```
df['num-of-doors'].isna().sum()
```

Out[183]:

```
0
```

In [184]:

```
df['num-of-doors'].value_counts()
```

Out[184]:

```
4    116
2     89
Name: num-of-doors, dtype: int64
```

In [185]:

```
# Imputation of missing value with mode

df['num-of-doors'].fillna(df['num-of-doors'].mode()[0], inplace=True)
```

In [186]:

```
df['num-of-doors'].value_counts().to_dict()
```

Out[186]:

```
{4: 116, 2: 89}
```

In [187]:

```
df['num-of-doors'].replace({'four': 4, 'two': 2}, inplace=True)
```

In [188]:

```
df['num-of-doors'].value_counts().to_dict()
```

Out[188]:

```
{4: 116, 2: 89}
```

In [189]:

```
num_of_doors_dict = {'four': 4, 'two': 2}
```

6. body-style

In [178]:

```
df['body-style'].unique()
```

Out[178]:

```
array(['convertible', 'hatchback', 'sedan', 'wagon', 'hardtop'],
      dtype=object)
```

In [179]:

```
df['body-style'].value_counts()
```

Out[179]:

```
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: body-style, dtype: int64
```

In [180]:

```
df = pd.get_dummies(df, columns=['body-style'])
```

7. drive-wheels

In [212]:

```
df['drive-wheels'].unique()
```

Out[212]:

```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

In [213]:

```
df['drive-wheels'].value_counts()
```

Out[213]:

```
fwd    120
rwd     76
4wd      9
Name: drive-wheels, dtype: int64
```

In [214]:

```
df['drive-wheels'].replace({'fwd': 0, 'rwd': 1, '4wd': 2}, inplace = True)
```

In [215]:

```
drive_wheels_dict = {'fwd': 0, 'rwd': 1, '4wd': 2}
```

In [216]:

```
df['drive-wheels'].value_counts()
```

Out[216]:

```
0    120
1     76
2      9
Name: drive-wheels, dtype: int64
```

8. engine-location

In [208]:

```
df['engine-location'].unique()
```

Out[208]:

```
array(['front', 'rear'], dtype=object)
```

In [209]:

```
df['engine-location'].replace({'front': 0, 'rear': 1}, inplace=True)
```

In [301]:

```
engine_location_dict  = {'front': 0, 'rear': 1}
```

In [211]:

```
df['engine-location'].value_counts()
```

Out[211]:

```
0    202
1      3
Name: engine-location, dtype: int64
```

9. engine-type

In [227]:

```
df['engine-type'].unique()
```

Out[227]:

```
array(['dohc', 'ohcv', 'ohc', 'l', 'rotor', 'ohcf', 'dohcv'], dtype=objec
t)
```

In [228]:

```
df['engine-type'].value_counts()
```

Out[228]:

```
ohc      148
ohcf      15
ohcv      13
dohc      12
l         12
rotor      4
dohcv      1
Name: engine-type, dtype: int64
```

In [229]:

```
df = pd.get_dummies(df, columns=['engine-type'])
df
```

Out[229]:

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	drive-wheels	engine-location	wheel-base	length	width
0	3	115.0	1	0	2	1	0	88.6	168.8	64.1
1	3	115.0	1	0	2	1	0	88.6	168.8	64.1
2	1	115.0	1	0	2	1	0	94.5	171.2	65.5
3	2	164.0	1	0	4	0	0	99.8	176.6	66.2
4	2	164.0	1	0	4	2	0	99.4	176.6	66.4
...	...	...	...	...	...	...	...	...	...	...
200	-1	95.0	1	0	4	1	0	109.1	188.8	68.9
201	-1	95.0	1	1	4	1	0	109.1	188.8	68.8
202	-1	95.0	1	0	4	1	0	109.1	188.8	68.9
203	-1	95.0	0	1	4	1	0	109.1	188.8	68.9
204	-1	95.0	1	1	4	1	0	109.1	188.8	68.9

205 rows × 64 columns

10. num-of-cylinders

In [230]:

```
df['num-of-cylinders'].unique()
```

Out[230]:

```
array(['four', 'six', 'five', 'three', 'twelve', 'two', 'eight'],
      dtype=object)
```

In [231]:

```
df['num-of-cylinders'].value_counts().to_dict()
```

Out[231]:

```
{'four': 159,
 'six': 24,
 'five': 11,
 'eight': 5,
 'two': 4,
 'three': 1,
 'twelve': 1}
```

In [232]:

```
df['num-of-cylinders'].replace({'four': 4,
 'six': 6,
 'five': 5,
 'eight': 8,
 'two': 2,
 'three': 3,
 'twelve': 12},inplace=True)
```

In [302]:

```
num_of_cylinders_dict = {'four': 4,
 'six': 6,
 'five': 5,
 'eight': 8,
 'two': 2,
 'three': 3,
 'twelve': 12}
```

In [234]:

```
df['num-of-cylinders'].value_counts()
```

Out[234]:

```
4     159
6      24
5      11
8       5
2       4
3       1
12      1
Name: num-of-cylinders, dtype: int64
```

11. fuel-system

In [223]:

```
df['fuel-system'].unique()
```

Out[223]:

```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

In [224]:

```
df['fuel-system'].value_counts()
```

Out[224]:

```
mpfi     94
2bbl     66
idi      20
1bbl     11
spdi      9
4bbl      3
mfi       1
spfi      1
Name: fuel-system, dtype: int64
```

In [225]:

```
df = pd.get_dummies(df, columns=['fuel-system'])
df
```

Out[225]:

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	drive-wheels	engine-location	wheel-base	length	width
0	3	115.0	1	0	2	1	0	88.6	168.8	64.1
1	3	115.0	1	0	2	1	0	88.6	168.8	64.1
2	1	115.0	1	0	2	1	0	94.5	171.2	65.5
3	2	164.0	1	0	4	0	0	99.8	176.6	66.2
4	2	164.0	1	0	4	2	0	99.4	176.6	66.4
...	...	...	...	...	...	...	...	...	...	...
200	-1	95.0	1	0	4	1	0	109.1	188.8	68.9
201	-1	95.0	1	1	4	1	0	109.1	188.8	68.8
202	-1	95.0	1	0	4	1	0	109.1	188.8	68.9
203	-1	95.0	0	1	4	1	0	109.1	188.8	68.9
204	-1	95.0	1	1	4	1	0	109.1	188.8	68.9

205 rows × 58 columns

12. bore

In [219]:

```
df['bore'].unique()
```

Out[219]:

```
array(['3.47', '2.68', '3.19', '3.13', '3.5', '3.31', '3.62', '2.91',
       '3.03', '2.97', '3.34', '3.6', '2.92', '3.15', '3.43', '3.63',
       '3.54', '3.08', nan, '3.39', '3.76', '3.58', '3.46', '3.8', '3.78',
       '3.17', '3.35', '3.59', '2.99', '3.33', '3.7', '3.61', '3.94',
       '3.74', '2.54', '3.05', '3.27', '3.24', '3.01'], dtype=object)
```

In [220]:

```
df['bore'].isna().sum()
```

Out[220]:

```
4
```

In [221]:

```
# Imputation of missing value with median

df['bore'] = df['bore'].fillna(df['bore'].median()).astype(float)
```

In [222]:

```
df['bore'].isna().sum()
```

Out[222]:

```
0
```

13. stroke

In [190]:

```
df['stroke'].unique()
```

Out[190]:

```
array(['2.68', '3.47', '3.4', '2.8', '3.19', '3.39', '3.03', '3.11',
       '3.23', '3.46', '3.9', '3.41', '3.07', '3.58', '4.17', '2.76',
       '3.15', nan, '3.16', '3.64', '3.1', '3.35', '3.12', '3.86', '3.29',
       '3.27', '3.52', '2.19', '3.21', '2.9', '2.07', '2.36', '2.64',
       '3.08', '3.5', '3.54', '2.87'], dtype=object)
```

In [191]:

```
df['stroke'].isna().sum()
```

Out[191]:

```
4
```

In [192]:

```
# Imputation of missing value

df['stroke'] = df['stroke'].fillna(df['stroke'].median()).astype(float)
```

In [193]:

```
df['stroke'].isna().sum()
```

Out[193]:

```
0
```

14. horsepower

In [194]:

```
df['horsepower'].unique()
```

Out[194]:

```
array(['111', '154', '102', '115', '110', '140', '160', '101', '121',
       '182', '48', '70', '68', '88', '145', '58', '76', '60', '86',
       '100', '78', '90', '176', '262', '135', '84', '64', '120', '72',
       '123', '155', '184', '175', '116', '69', '55', '97', '152', '200',
       '95', '142', '143', '207', '288', nan, '73', '82', '94', '62',
       '56', '112', '92', '161', '156', '52', '85', '114', '162', '134',
       '106'], dtype=object)
```

In [195]:

```
df['horsepower'].isna().sum()
```

Out[195]:

```
2
```

In [196]:

```
# Imputation of missing value

df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median()).astype(float)
```

In [197]:

```
df['horsepower'].isna().sum()
```

Out[197]:

```
0
```

15. peak-rpm

In [198]:

```
df['peak-rpm'].unique()
```

Out[198]:

```
array(['5000', '5500', '5800', '4250', '5400', '5100', '4800', '6000',
       '4750', '4650', '4200', '4350', '4500', '5200', '4150', '5600',
       '5900', '5750', nan, '5250', '4900', '4400', '6600', '5300'],
      dtype=object)
```

In [199]:

```
df['peak-rpm'].isna().sum()
```

Out[199]:

```
2
```

In [200]:

```
df['peak-rpm'] = df['peak-rpm'].fillna(df['peak-rpm'].median()).astype(float)
```

In [201]:

```
df['peak-rpm'].isna().sum()
```

Out[201]:

```
0
```

16. price

In [202]:

```
df['price'].isna().sum()
```

Out[202]:

```
4
```

In [203]:

```
df['price'] = df['price'].fillna(df['price'].median()).astype(float)
```

In [204]:

```
df['price'].isna().sum()
```

Out[204]:

```
0
```

In [235]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 64 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   symboling               205 non-null    int64  
 1   normalized-losses       205 non-null    float64
 2   fuel-type               205 non-null    int64  
 3   aspiration              205 non-null    int64  
 4   num-of-doors            205 non-null    int64  
 5   drive-wheels            205 non-null    int64  
 6   engine-location         205 non-null    int64  
 7   wheel-base              205 non-null    float64
 8   length                  205 non-null    float64
 9   width                   205 non-null    float64
 10  height                  205 non-null    float64
 11  curb-weight             205 non-null    int64  
 12  num-of-cylinders        205 non-null    int64  
 13  engine-size             205 non-null    int64  
 14  bore                    205 non-null    float64
 15  stroke                  205 non-null    float64
 16  compression-ratio       205 non-null    float64
 17  horsepower              205 non-null    float64
 18  peak-rpm                205 non-null    float64
 19  city-mpg                205 non-null    int64  
 20  highway-mpg             205 non-null    int64  
 21  price                   205 non-null    float64
 22  make_alfa-romero        205 non-null    uint8  
 23  make_audi               205 non-null    uint8  
 24  make_bmw                205 non-null    uint8  
 25  make_chevrolet          205 non-null    uint8  
 26  make_dodge              205 non-null    uint8  
 27  make_honda              205 non-null    uint8  
 28  make_isuzu              205 non-null    uint8  
 29  make_jaguar             205 non-null    uint8  
 30  make_mazda              205 non-null    uint8  
 31  make_mercedes-benz      205 non-null    uint8  
 32  make_mercury            205 non-null    uint8  
 33  make_mitsubishi         205 non-null    uint8  
 34  make_nissan             205 non-null    uint8  
 35  make_peugot             205 non-null    uint8  
 36  make_plymouth           205 non-null    uint8  
 37  make_porsche            205 non-null    uint8  
 38  make_renault            205 non-null    uint8  
 39  make_saab               205 non-null    uint8  
 40  make_subaru             205 non-null    uint8  
 41  make_toyota             205 non-null    uint8  
 42  make_volkswagen         205 non-null    uint8  
 43  make_volvo              205 non-null    uint8  
 44  body-style_convertible  205 non-null    uint8  
 45  body-style_hardtop      205 non-null    uint8  
 46  body-style_hatchback    205 non-null    uint8  
 47  body-style_sedan        205 non-null    uint8  
 48  body-style_wagon         205 non-null    uint8  
 49  fuel-system_1bbl        205 non-null    uint8  
 50  fuel-system_2bbl        205 non-null    uint8  
 51  fuel-system_4bbl        205 non-null    uint8  
 52  fuel-system_idi         205 non-null    uint8  
 53  fuel-system_mfi         205 non-null    uint8  
 54  fuel-system_mpfi        205 non-null    uint8  
 55  fuel-system_spdi        205 non-null    uint8  
```

```
 56  fuel-system_spfi        205 non-null    uint8
 57  engine-type_dohc        205 non-null    uint8
 58  engine-type_dohcv       205 non-null    uint8
 59  engine-type_l           205 non-null    uint8
 60  engine-type_ohc         205 non-null    uint8
 61  engine-type_ohcf        205 non-null    uint8
 62  engine-type_ohcv        205 non-null    uint8
 63  engine-type_rotor       205 non-null    uint8
dtypes: float64(11), int64(11), uint8(42)
memory usage: 43.8 KB
```

Step 5: Feature Selection

In [236]:

```
df.corr()
```

Out[236]:

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	drive-wheels	engine-location	
symboling	1.000000	0.457484	0.194311	-0.059866	-0.663595	-0.111150	0.212471	-0
normalized-losses	0.457484	1.000000	0.104668	-0.011273	-0.348850	0.133824	-0.021510	-0
fuel-type	0.194311	0.104668	1.000000	-0.401397	-0.188496	-0.051874	0.040070	-0
aspiration	-0.059866	-0.011273	-0.401397	1.000000	0.052803	0.153897	-0.057191	0
num-of-doors	-0.663595	-0.348850	-0.188496	0.052803	1.000000	-0.003230	-0.139129	0
...	...	...	...	...	...	...	...	
engine-type_l	-0.133979	0.170806	-0.268163	0.207156	0.176489	0.197050	-0.030388	0
engine-type_ohc	-0.082855	-0.156069	-0.020584	-0.020162	0.027539	-0.429386	-0.196371	-0
engine-type_ohcf	0.037513	-0.210771	0.092384	-0.034450	0.019357	0.197807	0.433727	-0
engine-type_ohcv	-0.013597	0.130717	0.085556	-0.070070	-0.054764	0.139453	-0.031711	0
engine-type_rotor	0.245950	0.130721	0.046383	-0.066203	-0.161052	0.131758	-0.017192	-0

64 rows × 64 columns

In [237]:

```
# Checking for Multicollinearity

df1 = df.drop('price', axis=1)    # Dropping dependant variable from dataset

vif_list = []

for i in range(df1.shape[1]):
    vif = variance_inflation_factor(df1,i)
    vif_list.append(vif)

s1 = pd.Series(vif_list, index=df1.columns)
s1

# s1.sort_values().plot(kind = 'barh')
```

Out[237]:

```
symboling           6.062736
normalized-losses   3.112823
fuel-type                inf
aspiration          6.187690
num-of-doors        3.817695
                      ...
engine-type_l            inf
engine-type_ohc          inf
engine-type_ohcf         inf
engine-type_ohcv         inf
engine-type_rotor        inf
Length: 63, dtype: float64
```

Train-Test Split

In [238]:

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

In [251]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state

print("Training data counts", x_train.shape, y_train.shape)
print("Testing data counts", x_test.shape, y_test.shape)
```

```
Training data counts (164, 63) (164,)
Testing data counts (41, 63) (41,)
```

Model Training

In [258]:

```
model_linear = LinearRegression()

model_linear.fit(x_train, y_train)
```

Out[258]:

```
LinearRegression()
```

In [260]:

```
# Model Evaluation for Training Data

y_pred_train = model_linear.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("Mean Sqaured Error :",mse)

rmse = np.sqrt(mse)
print("Root Mean Sqaured Error :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("Mean Absolute Error :",mae)

r_squared_value = r2_score(y_train, y_pred_train)
print("R Squared Value :",r_squared_value)

adj_r2  = 1 - (((1 - r_squared_value) * (x_train.shape[0] - 1)) / (x_train.shape[0]
print("Adjusted R Squared Value :",adj_r2)
```

```
Mean Sqaured Error : 2391392.9332026816
Root Mean Sqaured Error : 1546.4129245459253
Mean Absolute Error : 1129.2985382288748
R Squared Value : 0.962437619451845
Adjusted R Squared Value : 0.9387733197065073
```

In [261]:

```
# Model Evaluation for Testing Data
y_pred = model_linear.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Sqaured Error :",mse)

rmse = np.sqrt(mse)
print("Root Mean Sqaured Error :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error :",mae)

r_squared_value = r2_score(y_test, y_pred)
print("R Squared Value :",r_squared_value)

adj_r2  = 1 - (((1 - r_squared_value) * (x_test.shape[0] - 1)) / (x_test.shape[0] - 
print("Adjusted R Squared Value :",adj_r2)
```

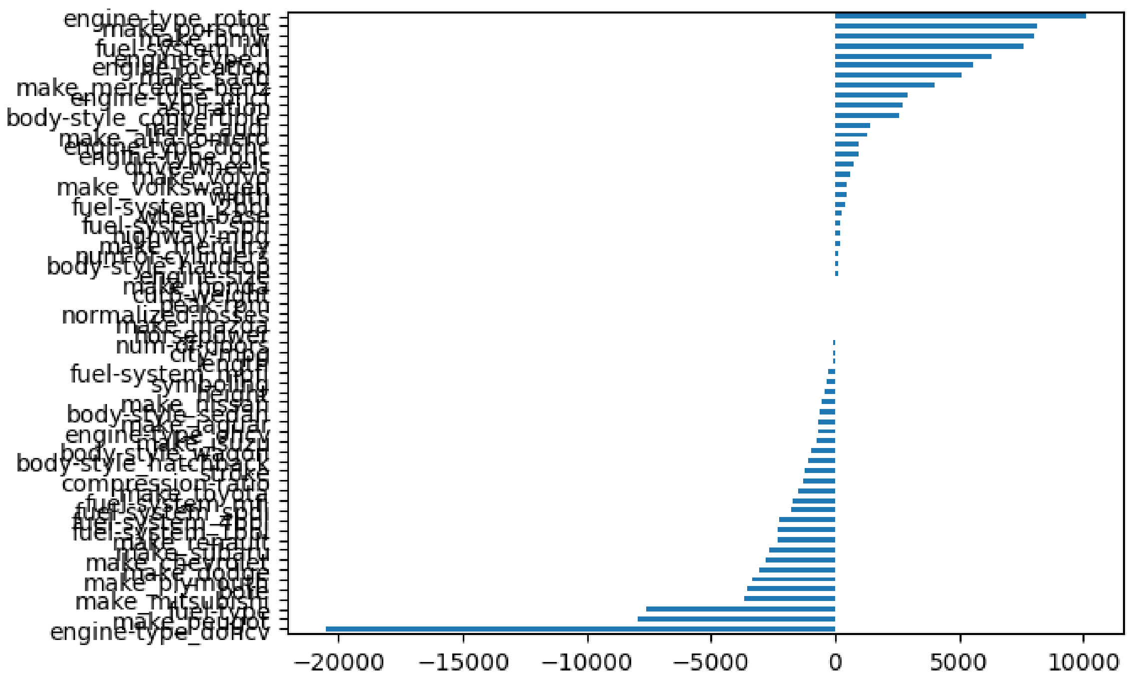
```
Mean Sqaured Error : 6792666.853525117
Root Mean Sqaured Error : 2606.2745161485036
Mean Absolute Error : 1592.968674573721
R Squared Value : 0.8701141652233484
Adjusted R Squared Value : 1.2258884083072201
```

In [262]:

```
s2 = pd.Series(model_linear.coef_, index=x.columns)
s2.sort_values().plot(kind = "barh")
```

Out[262]:

<AxesSubplot:>



```
By comparing the accuracy of training and testing data. It is observed that model is overfitted therefore we are checking the assumptions of linear regression.
```

In [263]:

```
Residual = y_train - y_pred_train
Residual
```

Out[263]:

```
111    1805.501237
93      636.621267
148   -1326.147280
21     -818.147122
28      771.501224
          ...
56     -888.691189
182     -22.879388
204    1489.372537
92      431.285687
126   -1131.585946
Name: price, Length: 164, dtype: float64
```

1. Assumption of Normality of Residual

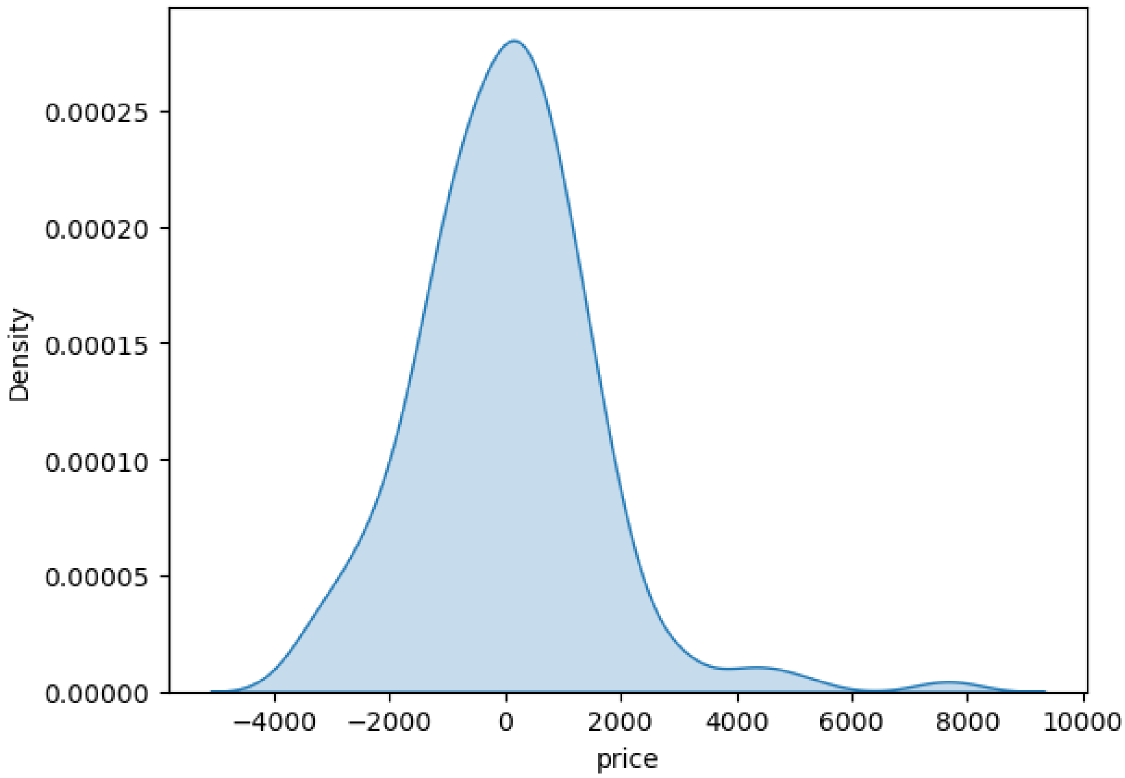
In [264]:

```
# KDE plot

sns.kdeplot(Residual, fill= True )
```

Out[264]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



In [265]:

```
# Hypothesis Testing

_ , p_val = shapiro(Residual)

print("P_Value:",p_val)

if p_val >= 0.05:
    print("Null Hypothesis is Accepted")
    print("--> Data is Normally Distributed")

else:
    print("Null Hypothesis is Rejected >> Alternate Hypothesis is Accepted")
    print("Data is NOT Normally Distributed")
```

```
P_Value: 7.378377631539479e-06
Null Hypothesis is Rejected >> Alternate Hypothesis is Accepted
Data is NOT Normally Distributed
```

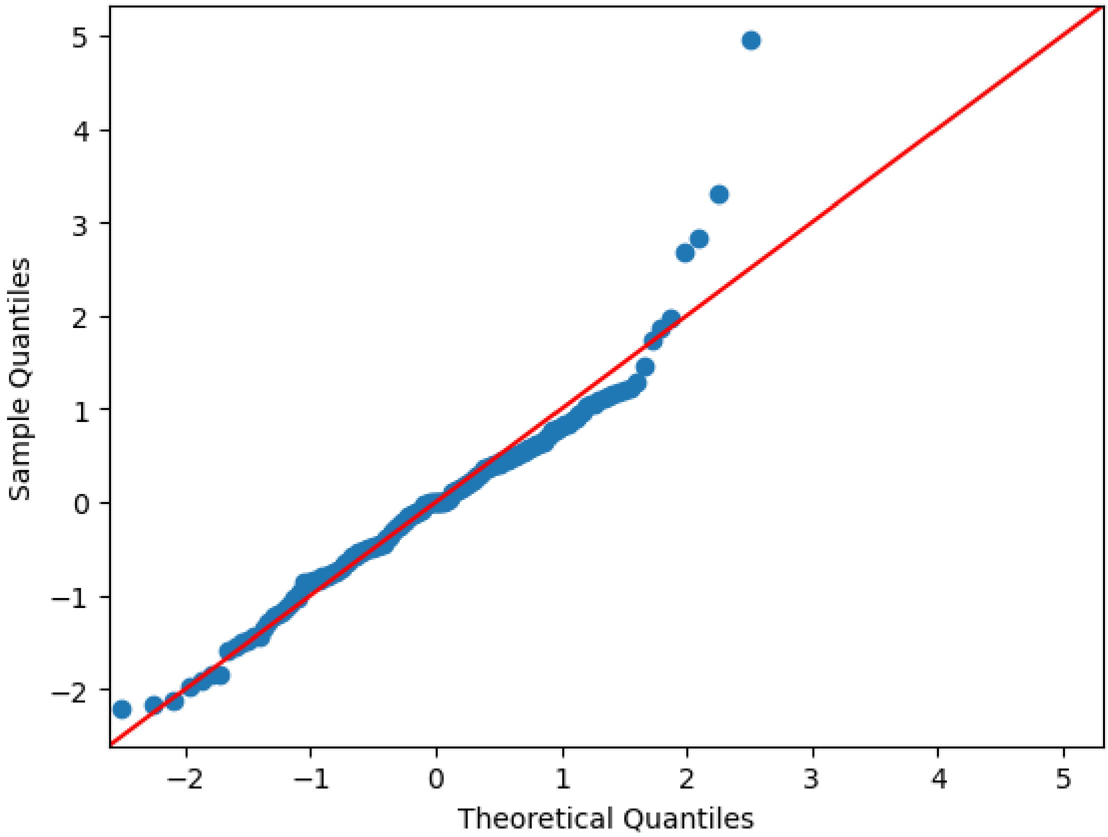
In [266]:

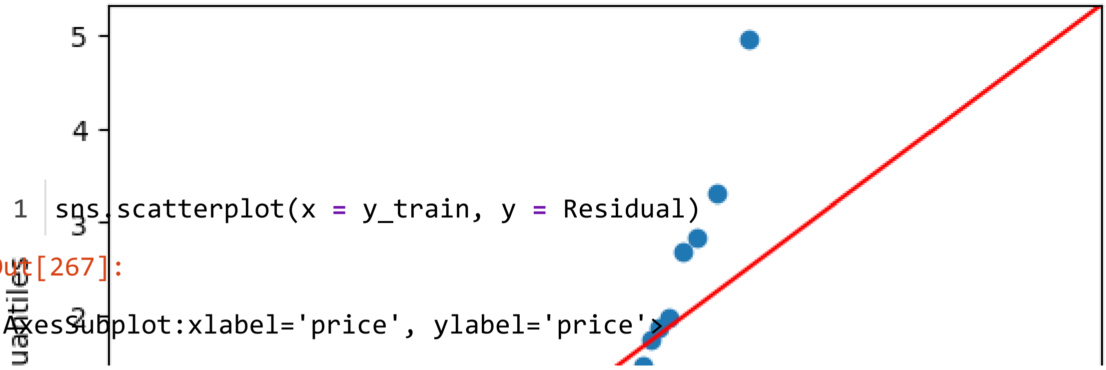
```
# By Using QQ plot (Visualization Method)

sm.qqplot(Residual, line = '45', fit=True)

# If 90% of datapoints are on the line --> data is Normally distributed
```

Out[266]:

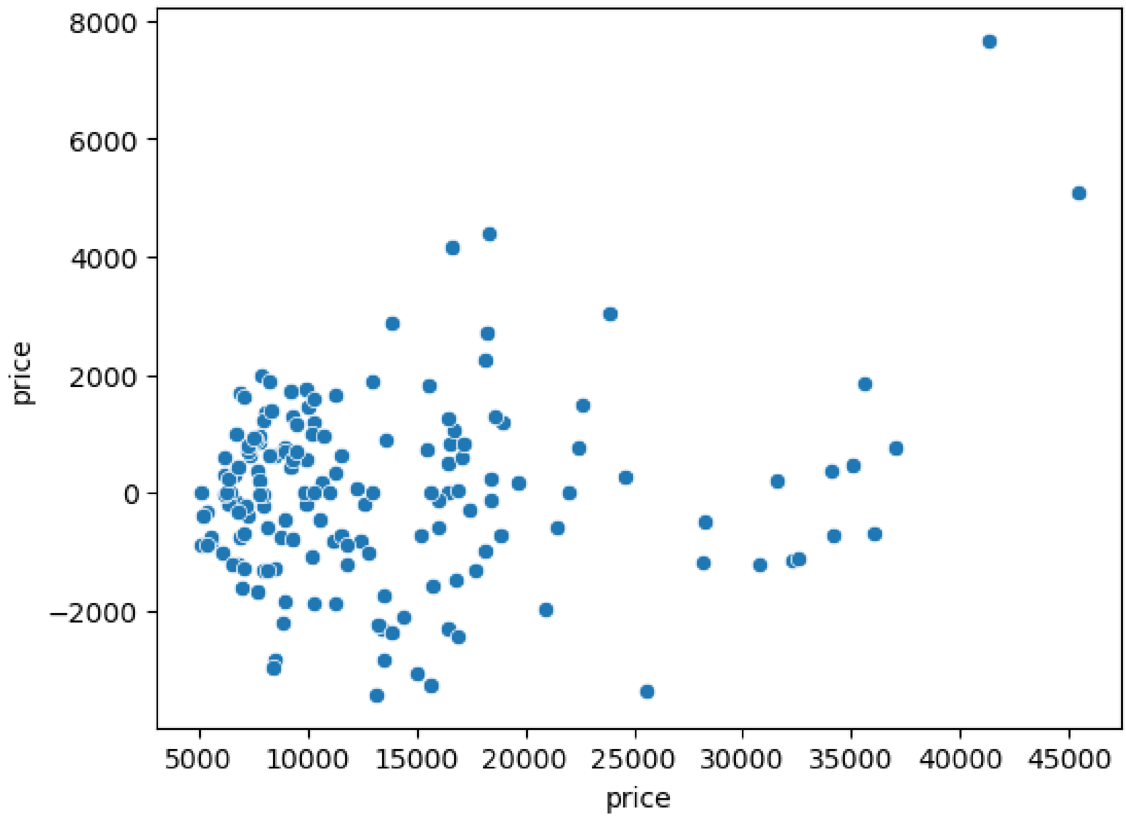





```
sns.scatterplot(x = y_train, y = Residual)
```

Out[267]:

<AxesSubplot:xlabel='price', ylabel='price'>



Model Building By using Regularization Techniques

1. Ridge Regression

In [268]:

```
ridge_reg_model = Ridge()        # alpha= 1.0 (bydefault)

ridge_reg_model.fit(x_train, y_train)
```

Out[268]:

```
Ridge()
```

In [269]:

```
# Model Evaluation on Training Dataset

y_pred_train = ridge_reg_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("MAE :",mae)

r2 = r2_score(y_train, y_pred_train)
print("R2 Scored :", r2)
```

```
MSE : 3382887.8923862255
RMSE : 1839.262866581671
MAE : 1294.086249500661
R2 Scored : 0.9468638881543489
```

In [270]:

```
# Model Evaluation on Testing Dataset

y_pred = ridge_reg_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE :",mae)

r2 = r2_score(y_test, y_pred)
print("R2 Scored :", r2)
```

```
MSE : 6025948.636963595
RMSE : 2454.7807716705775
MAE : 1678.821286187017
R2 Scored : 0.8847749512951515
```

In [271]:

```
s3 = pd.Series(ridge_reg_model.coef_, index=x.columns)
s3.sort_values()
```

Out[271]:

```
engine-type_dohcv    -6004.355103
bore                 -4197.134277
make_subaru          -2842.436517
make_mitsubishi      -2661.321138
make_plymouth        -2289.347167
                         ...
aspiration            3142.450446
engine-type_rotor     3330.204138
make_mercedes-benz    4130.231640
engine-location       5637.771822
make_bmw              6226.981653
Length: 63, dtype: float64
```

2. Lasso Regression

In [272]:

```
lasso_reg_model = Lasso()                # alpha= 1.0 (bydefault)

lasso_reg_model.fit(x_train, y_train)
```

Out[272]:

```
Lasso()
```

In [273]:

```
# Model Evaluation on Training Dataset

y_pred_train = lasso_reg_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("MAE :",mae)

r2 = r2_score(y_train, y_pred_train)
print("R2 Scored :", r2)
```

```
MSE : 2404575.9929006225
RMSE : 1550.6695305256444
MAE : 1134.8654267419054
R2 Scored : 0.9622305488787544
```

In [274]:

```
# Model Evaluation on Testing Dataset

y_pred = lasso_reg_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE :",mae)

r2 = r2_score(y_test, y_pred)
print("R2 Scored :", r2)
```

```
MSE : 6455877.75814986
RMSE : 2540.841938836389
MAE : 1550.8424925158415
R2 Scored : 0.876554070747905
```

In [275]:

```
s4 = pd.Series(lasso_reg_model.coef_, index=x.columns)
s4.sort_values()
```

Out[275]:

```
engine-type_dohcv   -20959.411383
make_peugot          -6490.384046
bore                 -3728.854131
make_mitsubishi      -3265.543817
make_plymouth        -2947.792748
                         ...
engine-location       7716.240019
engine-type_rotor     7920.855914
make_porsche          7934.902085
make_bmw              8380.363052
fuel-system_idi       9545.428879
Length: 63, dtype: float64
```

3. Hyperparameter Tuning

A. Ridge Regression

BY Using GridsearchCV

In [276]:

```
# Model instance
ridge_reg_model = Ridge()

# Defined param_grid
param_grid = {"alpha": np.arange(0.01,3,0.01)}

gscv_ridge_model = GridSearchCV(ridge_reg_model, param_grid, n_jobs=-1)

gscv_ridge_model.fit(x_train, y_train)

gscv_ridge_model.best_estimator_
```

Out[276]:

```
Ridge(alpha=0.01)
```

In [277]:

```
# Rebuild the model by using new alpha value

ridge_reg_model = Ridge(alpha = 0.01)

ridge_reg_model.fit(x_train, y_train)
```

Out[277]:

```
Ridge(alpha=0.01)
```

In [278]:

```
# Evaluation Matrix on Training Dataset

y_pred_train = ridge_reg_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("MAE :",mae)

r2 = r2_score(y_train, y_pred_train)
print("R2 Scored :", r2)
```

```
MSE : 2393827.1930172667
RMSE : 1547.199790918182
MAE : 1132.2192184501341
R2 Scored : 0.962399383747357
```

In [279]:

```
# Evaluation Matrix on Testing Dataset

y_pred = ridge_reg_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE :",mae)

r2 = r2_score(y_test, y_pred)
print("R2 Scored :", r2)
```

```
MSE : 6677560.767209952
RMSE : 2584.097669827894
MAE : 1581.9639298494208
R2 Scored : 0.8723151638048054
```

By using Randomized Search CV

In [280]:

```
# Model instance
ridge_reg_model = Ridge()

# Defined param_grid
param_grid = {"alpha": np.arange(0.01,3,0.01)}

rscv_ridge_model = RandomizedSearchCV(ridge_reg_model, param_grid, n_jobs=-1)

rscv_ridge_model.fit(x_train, y_train)

rscv_ridge_model.best_estimator_
```

Out[280]:

```
Ridge(alpha=0.09999999999999999)
```

In [281]:

```
# Rebuild the model by using new alpha value

ridge_reg_model = Ridge(alpha = 0.09999999999999999)

ridge_reg_model.fit(x_train, y_train)
```

Out[281]:

```
Ridge(alpha=0.09999999999999999)
```

In [282]:

```
# Model Evaluation on Training Dataset

y_pred_train = ridge_reg_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("MAE :",mae)

r2 = r2_score(y_train, y_pred_train)
print("R2 Scored :", r2)
```

```
MSE : 2488819.012457033
RMSE : 1577.5991292014055
MAE : 1151.6574760664
R2 Scored : 0.960907316583815
```

In [283]:

```
# Model Evaluation on Testing Dataset

y_pred = ridge_reg_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE :",mae)

r2 = r2_score(y_test, y_pred)
print("R2 Scored :", r2)
```

```
MSE : 6262444.761345223
RMSE : 2502.4877145243336
MAE : 1579.765486212505
R2 Scored : 0.880252795682469
```

B. Lasso Regression

By using GridSearchCV

In [285]:

```
# Model instance
lasso_reg_model = Lasso()

# Defined param_grid
param_grid = {"alpha": np.arange(0.01,3,0.01)}


gscv_lasso_model = GridSearchCV(lasso_reg_model, param_grid, n_jobs=-1, cv= 5)

gscv_lasso_model.fit(x_train, y_train)
gscv_lasso_model.best_estimator_
```

Out[285]:

```
Lasso(alpha=0.13)
```

In [286]:

```
# Rebuild model by using new value of alpha

lasso_reg_model = Lasso(alpha = 0.13)
lasso_reg_model.fit(x_train, y_train)
```

Out[286]:

```
Lasso(alpha=0.13)
```

In [287]:

```
# Model Evaluation on Training Dataset

y_pred_train = lasso_reg_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("MAE :",mae)

r2 = r2_score(y_train, y_pred_train)
print("R2 Scored :", r2)
```

```
MSE : 2391726.5346919987
RMSE : 1546.5207837892121
MAE : 1129.8624048067356
R2 Scored : 0.962432379465593
```

In [288]:

```
# Model Evaluation on Testing Dataset

y_pred = lasso_reg_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE :",mae)

r2 = r2_score(y_test, y_pred)
print("R2 Scored :", r2)
```

```
MSE : 6733725.918703463
RMSE : 2594.942372906085
MAE : 1585.3126714642851
R2 Scored : 0.8712412030550145
```

By using RandomizesSearchCV

In [289]:

```
# Model instance
lasso_model = Lasso()

# Defined param_grid
param_grid = {"alpha": np.arange(0.01,3,0.01)}


rscv_lasso_model = RandomizedSearchCV(lasso_model, param_grid, n_jobs=-1)

rscv_lasso_model.fit(x_train, y_train)

rscv_lasso_model.best_estimator_
```

Out[289]:

```
Lasso(alpha=1.47)
```

In [290]:

```
# Rebuild the model by using new value of alpha

lasso_reg_model = Lasso(alpha=1.47)
lasso_reg_model.fit(x_train, y_train)
```

Out[290]:

```
Lasso(alpha=1.47)
```

In [291]:

```
# Model Evaluation on Training Dataset

y_pred_train = lasso_reg_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred_train)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_train, y_pred_train)
print("MAE :",mae)

r2 = r2_score(y_train, y_pred_train)
print("R2 Scored :", r2)
```

```
MSE : 2418392.208075672
RMSE : 1555.118068853832
MAE : 1138.2441936562661
R2 Scored : 0.962013533128254
```

In [292]:

```
# Model Evalution on Testing Dataset

y_pred = lasso_reg_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :",mse)

rmse = np.sqrt(mse)
print("RMSE :",rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE :",mae)

r2 = r2_score(y_test, y_pred)
print("R2 Scored :", r2)
```

```
MSE : 6329397.040614621
RMSE : 2515.8292948081
MAE : 1534.5770906645973
R2 Scored : 0.8789725690983267
```

```
By comparing all model accuracy it is observed that we get good accuracy on ridge regression model. Therefore we are creating pickel file of ridge regression model.
```

Creating Pickle File

In [293]:

```
with open("Ridge Model.pkl", "wb") as f:
    pickle.dump(ridge_reg_model, f)
```

Creating JSON File

In [295]:

```
column_names = x.columns
```

In [303]:

```
json_data = {"fuel_type":fuel_type_dict, "aspiration": aspiration_dict, "num_of_door
            "drive_wheels": drive_wheels_dict, "engine_location" : engine_location_d
             "num_of_cylinders":num_of_cylinders_dict, "columns": list(column_names)
json_data
```

Out[303]:

```
{'fuel_type': {'gas': 1, 'diesel': 0},
 'aspiration': {'std': 0, 'turbo': 1},
 'num_of_doors': {'four': 4, 'two': 2},
 'drive_wheels': {'fwd': 0, 'rwd': 1, '4wd': 2},
 'engine_location': {'front': 0, 'rear': 1},
 'num_of_cylinders': {'four': 4,
  'six': 6,
  'five': 5,
  'eight': 8,
  'two': 2,
  'three': 3,
  'twelve': 12},
 'columns': ['symboling',
  'normalized-losses',
  'fuel-type',
  'aspiration',
  'num-of-doors',
  'drive-wheels',
  'engine-location',
  'wheel-base',
  'length',
  'width',
  'height',
  'curb-weight',
  'num-of-cylinders',
  'engine-size',
  'bore',
  'stroke',
  'compression-ratio',
  'horsepower',
  'peak-rpm',
  'city-mpg',
  'highway-mpg',
  'make_alfa-romero',
  'make_audi',
  'make_bmw',
  'make_chevrolet',
  'make_dodge',
  'make_honda',
  'make_isuzu',
  'make_jaguar',
  'make_mazda',
  'make_mercedes-benz',
  'make_mercury',
  'make_mitsubishi',
  'make_nissan',
  'make_peugot',
  'make_plymouth',
  'make_porsche',
  'make_renault',
  'make_saab',
  'make_subaru',
  'make_toyota',
  'make_volkswagen',
  'make_volvo',
  'body-style_convertible',
  'body-style_hardtop',
  'body-style_hatchback',
  'body-style_sedan',
  'body-style_wagon',
  'fuel-system_1bbl',
```

In [305]:

```
import json
with open ("project_data.json", 'w') as f:
    json.dump(json_data,f)
```

Testing for user input value

In [306]:

```
symboling = 3.00
normalized_losses = 118.00
fuel_type = "gas"
aspiration = "turbo"
num_of_doors = "four"
drive_wheels = "4wd"
engine_location = "rear"
wheel_base = 88.60
length = 170.80
width = 64.10
height = 50.80
curb_weight = 2600.00
num_of_cylinders = "five"
engine_size = 130.00
bore = 3.47
stroke = 2.68
compression_ratio = 9.00
horsepower = 111.00
peak_rpm = 6000.00
city_mpg = 21.00
highway_mpg = 27.00


# onehot encoded columns
engine_type = "ohc"
body_style = "sedan"
fuel_system = "mpfi"
make = "audi"
```

```
 'fuel-system_2bbl',
 'fuel-system_4bbl',
 'fuel-system_idi',
 'fuel-system_mfi',
 'fuel-system_mpfi',
 'fuel-system_spdi',
 'fuel-system_spfi',
 'engine-type_dohc',
 'engine-type_dohcv',
 'engine-type_l',
 'engine-type_ohc',
 'engine-type_ohcf',
 'engine-type_ohcv',
 'engine-type_rotor']
```

In [307]:

```python
array = np.zeros(len(x.columns), dtype = int)
array
```

Out[307]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [309]:

```python
body_style_col = "body-style_" + body_style
engine_type_col = "engine-type_" + engine_type
fuel_system_col = "fuel-system_" + fuel_system
make_col = "make_" + make
```

In [313]:

```python
# Find the index of this column -->

body_style_index = json_data['columns'].index(body_style_col)
engine_type_index = json_data['columns'].index(engine_type_col)
fuel_system_index = json_data['columns'].index(fuel_system_col)
make_index        = json_data['columns'].index(make_col)
```

In [316]:

```
array[0] = symboling
array[1] = normalized_losses
array[2] = fuel_type_dict[fuel_type]
array[3] = aspiration_dict[aspiration]
array[4] = num_of_doors_dict[num_of_doors]
array[5] = drive_wheels_dict[drive_wheels]
array[6] = engine_location_dict[engine_location]
array[7] = wheel_base
array[8] = length
array[9] = width
array[10] = height
array[11] = curb_weight
array[12] = num_of_cylinders_dict[num_of_cylinders]
array[13] = engine_size
array[14] = bore
array[15] = stroke
array[16] = compression_ratio
array[17] = horsepower
array[18] = peak_rpm
array[19] = city_mpg
array[20] = highway_mpg

array[body_style_index] = 1
array[engine_type_index] = 1
array[fuel_system_index] = 1
array[make_index]    = 1
array
```

Out[316]:

```
array([   3,  118,    1,    1,    4,    2,    1,   88,  170,   64,   50,
       2600,    5,  130,    3,    2,    9,  111, 6000,   21,   27,    0,
          1,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    1,    0,    0,    0,    0,    0,    0,    1,    0,
          0,    0,    0,    0,    1,    0,    0,    0])
```

In [317]:

```
prediction = ridge_reg_model.predict([array])[0]  # 2D array
print("Prediction of your car Price is : $", round(prediction, 2))
```

```
Prediction of your car Price is : $ 28388.7
```