In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```python
df = pd.read_csv("Cars.csv")
df.head()
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	13495
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	16500
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154	19	26	16500
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102	24	30	13950
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115	18	22	17450

In [3]:

```python
df.shape
```

Out[3]:

```
(205, 15)
```

In [4]:

```python
df.isnull().sum()
```

Out[4]:

```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
body-style           0
drive-wheels         0
engine-location      0
width                0
height               0
engine-type          0
engine-size          0
horsepower           0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

In [5]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 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   body-style         205 non-null    object
 5   drive-wheels       205 non-null    object
 6   engine-location    205 non-null    object
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object
 10  engine-size        205 non-null    int64
 11  horsepower         205 non-null    object
 12  city-mpg           205 non-null    int64
 13  highway-mpg        205 non-null    int64
 14  price              205 non-null    int64
dtypes: float64(2), int64(5), object(8)
memory usage: 24.1+ KB
```

In [6]:

```python
df["normalized-losses"].value_counts()
```

Out[6]:

```
?      41
161    11
91      8
150     7
128     6
104     6
134     6
85      5
102     5
95      5
103     5
74      5
168     5
94      5
65      5
148     4
93      4
106     4
118     4
122     4
125     3
101     3
137     3
83      3
115     3
154     3
192     2
129     2
108     2
158     2
87      2
113     2
188     2
164     2
119     2
81      2
197     2
110     2
145     2
89      2
153     2
194     2
256     1
121     1
77      1
186     1
231     1
98      1
90      1
142     1
107     1
78      1
Name: normalized-losses, dtype: int64
```

In [7]:

```python
df["horsepower"].value_counts()
```

Out[7]:

```
68     19
70     11
69     10
116     9
110     8
95      7
114     6
88      6
62      6
160     6
101     6
97      5
82      5
76      5
145     5
102     5
84      5
92      4
86      4
123     4
111     4
152     3
90      3
73      3
182     3
85      3
207     3
121     3
184     2
100     2
162     2
112     2
161     2
?       2
155     2
156     2
56      2
176     2
94      2
52      2
78      1
200     1
120     1
64      1
175     1
72      1
60      1
143     1
58      1
106     1
135     1
288     1
142     1
55      1
154     1
48      1
140     1
134     1
115     1
262     1
Name: horsepower, dtype: int64
```

In [8]:

```python
df["normalized-losses"].replace("?",np.nan,inplace=True)
df["horsepower"].replace("?",np.nan,inplace=True)
```

In [9]:

```python
df["normalized-losses"]=df["normalized-losses"].astype("float64")
df["horsepower"]=df["horsepower"].astype("float64")
```

In [10]:

```python
nmean=df["normalized-losses"].mean()
hmean=df["horsepower"].mean()
```

In [11]:

```python
df["normalized-losses"].fillna(nmean,inplace=True)
df["horsepower"].fillna(hmean,inplace=True)
```

In [12]:

```python
df.head()
```

Out[12]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111.0	21	27	13495
1	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111.0	21	27	16500
2	1	122.0	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154.0	19	26	16500
3	2	164.0	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102.0	24	30	13950
4	2	164.0	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115.0	18	22	17450

In [13]:

```python
df["normalized-losses"].value_counts()
```

Out[13]:

```
122.0    45
161.0    11
91.0      8
150.0     7
134.0     6
104.0     6
128.0     6
103.0     5
102.0     5
74.0      5
65.0      5
168.0     5
85.0      5
95.0      5
94.0      5
118.0     4
93.0      4
106.0     4
148.0     4
101.0     3
115.0     3
125.0     3
83.0      3
154.0     3
137.0     3
108.0     2
145.0     2
188.0     2
81.0      2
158.0     2
129.0     2
197.0     2
153.0     2
192.0     2
164.0     2
194.0     2
110.0     2
113.0     2
89.0      2
119.0     2
87.0      2
256.0     1
90.0      1
142.0     1
78.0      1
186.0     1
231.0     1
107.0     1
98.0      1
121.0     1
77.0      1
Name: normalized-losses, dtype: int64
```

In [14]:

```python
df["horsepower"].value_counts()
```

Out[14]:

```
68.000000     19
70.000000     11
69.000000     10
116.000000     9
110.000000     8
95.000000      7
88.000000      6
114.000000     6
62.000000      6
160.000000     6
101.000000     6
97.000000      5
102.000000     5
82.000000      5
76.000000      5
84.000000      5
145.000000     5
86.000000      4
111.000000     4
123.000000     4
92.000000      4
207.000000     3
85.000000      3
152.000000     3
73.000000      3
182.000000     3
121.000000     3
90.000000      3
56.000000      2
155.000000     2
162.000000     2
94.000000      2
52.000000      2
104.256158     2
176.000000     2
112.000000     2
100.000000     2
161.000000     2
156.000000     2
184.000000     2
288.000000     1
140.000000     1
175.000000     1
78.000000      1
48.000000      1
134.000000     1
120.000000     1
60.000000      1
106.000000     1
142.000000     1
58.000000      1
72.000000      1
64.000000      1
135.000000     1
262.000000     1
154.000000     1
143.000000     1
55.000000      1
200.000000     1
115.000000     1
Name: horsepower, dtype: int64
```

In [15]:

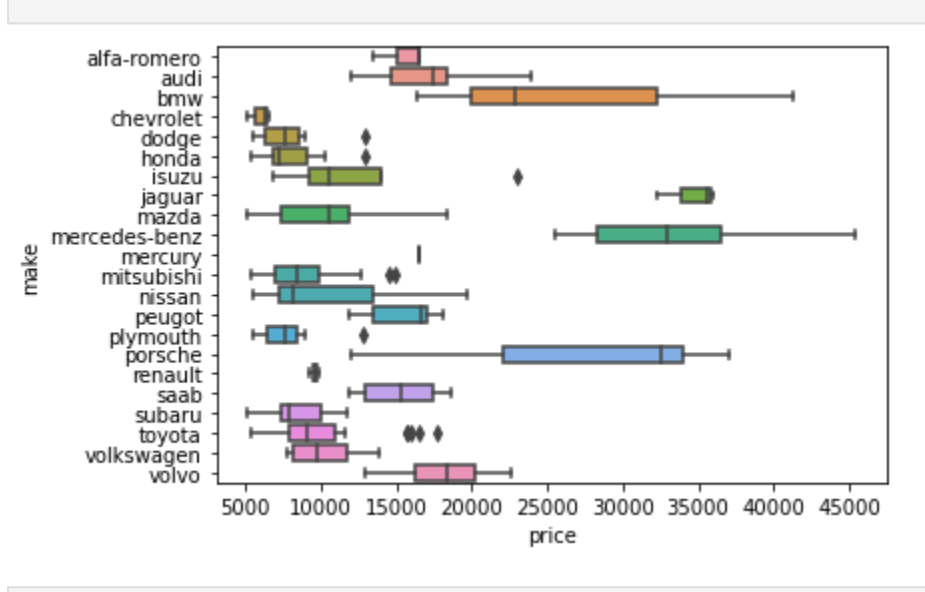
```python
sns.boxplot(df["price"])
```

Out[15]:

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

In [16]:

```python
sns.boxplot(data=df,x="price",y="make")
plt.show()
```



In [17]:

```python
df.head()
```

Out[17]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111.0	21	27	13495
1	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111.0	21	27	16500
2	1	122.0	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154.0	19	26	16500
3	2	164.0	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102.0	24	30	13950
4	2	164.0	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115.0	18	22	17450

In [18]:

```python
df_cat=df.select_dtypes("object")
df_num=df.select_dtypes(["int64","float64"])
```

In [19]:

```python
df_num
```

Out[19]:

	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	64.1	48.8	130	111.0	21	27	13495
1	3	122.0	64.1	48.8	130	111.0	21	27	16500
2	1	122.0	65.5	52.4	152	154.0	19	26	16500
3	2	164.0	66.2	54.3	109	102.0	24	30	13950
4	2	164.0	66.4	54.3	136	115.0	18	22	17450
...	...	...	...	...	...	...	...	...	...
200	-1	95.0	68.9	55.5	141	114.0	23	28	16845
201	-1	95.0	68.8	55.5	141	160.0	19	25	19045
202	-1	95.0	68.9	55.5	173	134.0	18	23	21485
203	-1	95.0	68.9	55.5	145	106.0	26	27	22470
204	-1	95.0	68.9	55.5	141	114.0	19	25	22625

205 rows × 9 columns

In [20]:

```python
from sklearn.preprocessing import LabelEncoder

for col in df_cat:
    le = LabelEncoder()
    df_cat[col]=le.fit_transform(df_cat[col])
```

In [21]:

```python
df = pd.concat([df_cat,df_num],axis=1)
```

In [22]:

```python
df.head()
```

Out[22]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
0	0	1	0	2	0	0	3	122.0	64.1	48.8	130	111.0	21	27	13495
1	0	1	0	2	0	0	3	122.0	64.1	48.8	130	111.0	21	27	16500
2	0	1	2	2	0	5	1	122.0	65.5	52.4	152	154.0	19	26	16500
3	1	1	3	1	0	3	2	164.0	66.2	54.3	109	102.0	24	30	13950
4	1	1	3	0	0	3	2	164.0	66.4	54.3	136	115.0	18	22	17450

In [23]:

```python
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [24]:

```python
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=1)
```

In [25]:

```python
from sklearn.linear_model import LinearRegression
linreg=LinearRegression()
linreg.fit(xtrain,ytrain)
ypred = linreg.predict(xtest)
```

In [26]:

```python
from sklearn.metrics import mean_absolute_error as mae , mean_squared_error as mse , r2_score
```

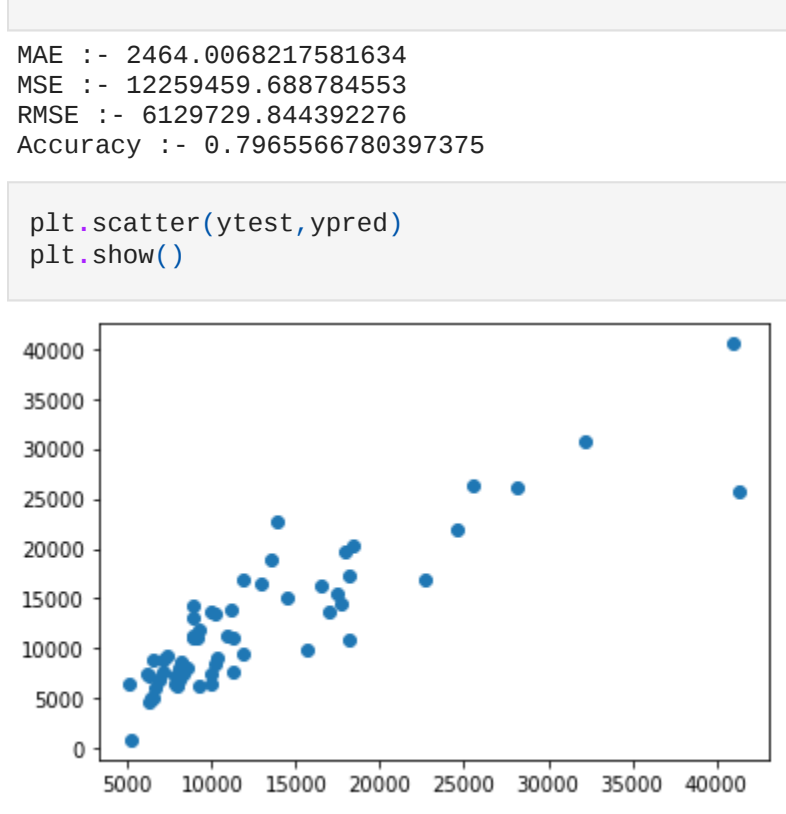
In [27]:

```python
print(f"MAE :- {mae(ytest,ypred)}")
print(f"MSE :- {mse(ytest,ypred)}")
print(f"RMSE :- {mse(ytest,ypred)*0.5}")
print(f"Accuracy :- {r2_score(ytest,ypred)}")
```

```
MAE :- 2464.0068217581634
MSE :- 12259459.688784553
RMSE :- 6129729.844392276
Accuracy :- 0.7965566780397375
```

In [28]:

```python
plt.scatter(ytest,ypred)
plt.show()
```



In [29]:

```python
#TRANING AND TESTING ACCURACY:-
print(f"Traning Accuracy :- {linreg.score(xtrain,ytrain)}")
print(f"Testing Accuracy :- {linreg.score(xtest,ytest)}")
```

```
Traning Accuracy :- 0.8504573774895474
Testing Accuracy :- 0.7965566780397375
```

In [30]:

```python
#COEFFICIENT AND INTERCEPT :-
print(f"COEFFICIENT :- {linreg.coef_}")
print(f"INTERCEPT :- {linreg.intercept_}")
```

```
COEFFICIENT :- [-2.00099087e+02 -6.22650015e+02 -1.70235175e+02  1.86860719e+03
  1.64133620e+04  2.83174279e+02  4.51384957e+01  1.53127607e+00
  7.89452171e+02  3.62663990e+02  9.83682875e+01 -1.08169245e+01
  3.08017854e+02 -4.17024371e+02]
INTERCEPT :- -65022.769859956396
```

- L1 AND L2 Regularization

In [31]:

```python
# RIDGE L2

from sklearn.linear_model import Ridge , Lasso

l2 = Ridge(alpha=11)
l2.fit(xtrain,ytrain)
ypred=l2.predict(xtest)
```

In [32]:

```python
print(f"L2 Accuracy :- {r2_score(ytest,ypred)}")
```

```
L2 Accuracy :- 0.815027724543179
```

In [33]:

```python
l1 = Lasso(alpha=200)
l1.fit(xtrain,ytrain)
ypred=l1.predict(xtest)
```

In [34]:

```python
print(f"L1 Accuracy :- {r2_score(ytest,ypred)}")
```

```
L1 Accuracy :- 0.8139201176339511
```

In [35]:

```python
for i in range(50):
    l2=Ridge(alpha=i)
    l2.fit(xtrain,ytrain)
    print(f"{i} :- {l2.score(xtest,ytest)}")
```

```
0 :- 0.796556678039738
1 :- 0.8074518758147272
2 :- 0.8110292248150515
3 :- 0.8126933383890032
4 :- 0.8136148645029299
5 :- 0.8141745853539418
6 :- 0.8145301242133356
7 :- 0.8147582608502815
8 :- 0.8149010602831954
9 :- 0.8149836949253052
10 :- 0.8150222867376529
11 :- 0.815027724543179
12 :- 0.8150076788279417
13 :- 0.814967738178826
14 :- 0.8149120868051175
15 :- 0.814843927825252
16 :- 0.8147657584947388
17 :- 0.8146795554128125
18 :- 0.8145869029046833
19 :- 0.8144890843369584
20 :- 0.8143871485445551
21 :- 0.8142819591129826
22 :- 0.8141742315788489
23 :- 0.8140645619421539
24 :- 0.8139534488166058
25 :- 0.8138413108452034
26 :- 0.8137285005403769
27 :- 0.8136153153884845
28 :- 0.813502006836264
29 :- 0.8133887876197212
30 :- 0.813275837783145
31 :- 0.8131633096537951
32 :- 0.8130513319772295
33 :- 0.8129400133729975
34 :- 0.8128294452363017
35 :- 0.8127197041851855
36 :- 0.8126108541327799
37 :- 0.8125029480485757
38 :- 0.8123960294605082
39 :- 0.8122901337400302
40 :- 0.8121852892047277
41 :- 0.8120815180669059
42 :- 0.8119788372516912
43 :- 0.8118772591041921
44 :- 0.8117767920020494
45 :- 0.8116774408870526
46 :- 0.8115792077273272
47 :- 0.8114820919198038
48 :- 0.8113860906412017
49 :- 0.8112911991545091
```

In [36]:

```python
for i in range(0,1000,50):
    l1=Lasso(alpha=i)
    l1.fit(xtrain,ytrain)
    print(f"{i} :- {l1.score(xtest,ytest)}")
```

```
0 :- 0.796556678039738
50 :- 0.8039357368934332
100 :- 0.8089989458401144
150 :- 0.8124970088065938
200 :- 0.8139201176339511
250 :- 0.8124236288710353
300 :- 0.8085081436827368
350 :- 0.8036070271922543
400 :- 0.7977246513279259
450 :- 0.7950447703402095
500 :- 0.7945982296092685
550 :- 0.7942417035818767
600 :- 0.7938601904983762
650 :- 0.7934534948615022
700 :- 0.7930217288952354
750 :- 0.7925635515879158
800 :- 0.7920809270031911
850 :- 0.7915731047691582
900 :- 0.7910716781565975
950 :- 0.7910214504771031
```

- CROSS VALIDATION

In [37]:

```python
from sklearn.model_selection import cross_val_score
csv = cross_val_score(l1,x,y,cv=4)
csv
```

Out[37]:

```
array([0.77085836, 0.81767385, 0.54379365, 0.447654  ])
```

In [38]:

```python
csv.mean()
```

Out[38]:

```
0.6449949624173605
```

In [39]:

```python
print(f"Avg Accuracy :- {csv.mean()}")
```

```
Avg Accuracy :- 0.6449949624173605
```

In [40]:

```python
csv1 = cross_val_score(l2,x,y,cv=4)
csv1
```

Out[40]:

```
array([0.76497804, 0.84463432, 0.45999473, 0.44701677])
```

In [41]:

```python
print(f"Avg Accuracy :- {csv1.mean()}")
```

```
Avg Accuracy :- 0.6291559652081944
```

In [ ]: