In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [3]:
```python
#reading a dataset
df=pd.read_csv('cars.csv')
df.head()
```

Out[3]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	h
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	

Steps for model building

1. read data ----> basic annalysis
2. Missing values and encoding
3. Build a baseline model----> without removing outliers,scaling,skweness
4. skewness,outliers,scaling
5. next model

Handling Missing vlues

In [4]:
```python
#setp 1:  replace '?' with NAN
df['normalized-losses'].replace('?',np.nan,inplace=True)
df['horsepower'].replace('?',np.nan,inplace=True)
```

In [5]:
```python
#step2:changing the datatype of mv columns
df['normalized-losses']=df['normalized-losses'].astype('float64')
df['horsepower']=df['horsepower'].astype('float64')
```

In [6]:
```python
from sklearn.impute import SimpleImputer
```

In [7]:
```python
si=SimpleImputer(missing_values=np.nan,strategy='mean')
```

In [8]:
```python
X=df.iloc[:,:-1]# All the cols except last col(features)
y=df.iloc[:,-1]# All the rows of last col(response)
```

In [9]:
```python
#fit nan with mean
X[['normalized-losses','horsepower']]=si.fit_transform(X[['normalized-losses','hors
```

Encoding

In [10]:
```python
#separating cat columns
cat_col=X.select_dtypes(object).columns
cat_col
```

Out[10]:
```
Index(['make', 'fuel-type', 'body-style', 'drive-wheels', 'engine-location',
       'engine-type'],
      dtype='object')
```

In [11]:
```python
#spliting trainig and testing data
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.3,random_state=1)
```

In [12]:
```python
#encoding
for col in cat_col:
    le=LabelEncoder() # le is an object
    xtrain[col]=le.fit_transform(xtrain[col])
    xtest[col]=le.transform(xtest[col])
```

In [13]:
```python
xtrain
```

Out[13]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	hors
124	3	122.0	14	1	2	2	0	66.3	50.2	3	156	
181	-1	122.0	19	1	4	2	0	66.5	54.1	0	161	
154	0	81.0	19	1	4	0	0	63.6	59.1	3	92	
53	1	113.0	8	1	3	1	0	64.2	54.1	3	91	
94	1	128.0	12	1	3	1	0	63.8	54.5	3	97	
...	...	...	...	...	...	...	...	...	...	...	...	
133	2	104.0	17	1	3	1	0	66.5	56.1	3	121	
137	2	104.0	17	1	3	1	0	66.5	56.1	0	121	
72	3	142.0	9	1	0	2	0	70.5	50.8	5	234	
140	2	83.0	18	1	2	0	0	63.8	55.7	4	108	
37	0	106.0	5	1	2	1	0	65.2	53.3	3	110	

143 rows × 14 columns

In [14]:
```python
lr = LinearRegression()
lr.fit(xtrain,ytrain)
```

Out[14]:
```
▾ LinearRegression
LinearRegression()
```

In [15]:
```python
#testing model on training data
from sklearn.metrics import r2_score
y_pred=lr.predict(xtrain)
r2_score(ytrain,y_pred)  # 0.85
```

Out[15]:
```
0.8504573774895473
```

In [16]:
```python
#testing model on testing data
from sklearn.metrics import r2_score
y_pred=lr.predict(xtest)
r2_score(ytest,y_pred)  # 0.79
```

Out[16]:
```
0.7965566780397378
```

In [17]:
```python
# as the r2_score for train data > r2_score test dat , this is an overfit model
```

In [18]:
```python
lr.score(xtrain,ytrain)
```

Out[18]:
```
0.8504573774895473
```

In [19]:
```python
lr.score(xtest,ytest) #short form of testing model
```

Out[19]:
```
0.7965566780397378
```

In [20]:
```python
#regularization
from sklearn.linear_model import Lasso, Ridge
```

In [21]:
```python
#ridge regularization
l2=Ridge(0.01)  # ridge is a class
l2.fit(xtrain,ytrain)  # training the model
```

Out[21]:
```
▾       Ridge
Ridge(alpha=0.01)
```

In [22]:
```python
#testing model
l2.score(xtest,ytest)
l2.coef_
```

Out[22]:
```
array([ 4.64993498e+01,  1.50176171e+00, -2.00054057e+02, -6.30175160e+02,
       -1.73592151e+02,  1.86936419e+03,  1.63210501e+04,  7.86314817e+02,
        3.64159605e+02,  2.85251823e+02,  9.83794562e+01, -1.05869814e+01,
        3.07288552e+02, -4.16061591e+02])
```

In [23]:
```python
#Including different values of error to L2 model
for alpha in range(1,5):
    l2=Ridge(alpha)
    l2.fit(xtrain,ytrain)
    test_score=l2.score(xtest,ytest)
    print('Alpha:', alpha)
    print('Test score:', test_score)
    print('-------------------------------')
```

```
Alpha: 1
Test score: 0.8074518758147275
-------------------------------
Alpha: 2
Test score: 0.8110292248150518
-------------------------------
Alpha: 3
Test score: 0.8126933383890036
-------------------------------
Alpha: 4
Test score: 0.8136148645029301
-------------------------------
```

In [ ]:
```python
'''
after an aipha value of 2 we can observe that there is a small change in the r2
score, therfore ridge with alpha=2 is a good model with an r2 score of 81%
'''
```

In [24]:
```python
# Lasso
for alpha in range(100,151,10):
    l1=Lasso(alpha)
    l1.fit(xtrain,ytrain)
    test_score=l1.score(xtest,ytest)
    print('Alpha:', alpha)
    print('Test score:', test_score)
    print('-------------------------------')
```

```
Alpha: 100
Test score: 0.8089989519118684
-------------------------------
Alpha: 110
Test score: 0.8098656626873879
-------------------------------
Alpha: 120
Test score: 0.8106487931098092
-------------------------------
Alpha: 130
Test score: 0.8113484125018898
-------------------------------
Alpha: 140
Test score: 0.8119644623062724
-------------------------------
Alpha: 150
Test score: 0.8124969899539803
-------------------------------
```

In [25]:
```python
l1=Lasso(130)
l1.fit(xtest,ytest)
l1.coef_
```

Out[25]:
```
array([  -0.        ,   -4.0813863 , -180.83436083,   -0.        ,
         -0.        , 2167.45227523,    0.        ,  664.88872076,
        171.05069745,   -0.        ,   71.49779216,   86.09086666,
         -0.        ,   88.22713844])
```

In [26]:
```python
l2=Ridge(2)
l2.fit(xtest,ytest)
l2.coef_
```

Out[26]:
```
array([ -93.33033533,   -4.79267756, -196.47591505, -786.64560775,
       -230.26519014, 2354.42511503,    0.        ,  605.82766781,
        216.01097772,   12.05853518,   70.09446003,   87.67942628,
       -150.50940981,  217.65784678])
```

In [27]:
```python
# Final Ridge model
l2=Ridge(2)
l2.fit(xtrain,ytrain)
l2.coef_
```

Out[27]:
```
array([ 1.66477241e+02, -8.84331252e-01, -1.94641841e+02, -1.13894088e+03,
       -4.80922921e+02,  1.88121378e+03,  7.76076971e+03,  5.06201386e+02,
        5.02070806e+02,  4.65928529e+02,  1.00154543e+02,  1.04124700e+01,
        2.44076984e+02, -3.27713737e+02])
```

In [28]:
```python
# Final Lasso model
l1=Lasso(130)
l1.fit(xtest,ytest)
l1.coef_
```

Out[28]:
```
array([  -0.        ,   -4.0813863 , -180.83436083,   -0.        ,
         -0.        , 2167.45227523,    0.        ,  664.88872076,
        171.05069745,   -0.        ,   71.49779216,   86.09086666,
         -0.        ,   88.22713844])
```

cross validation

In [29]:
```python
from sklearn.model_selection import cross_val_score
```

In [32]:
```python
#Encoding
catcol=X.select_dtypes(object).columns

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

In [34]:
```python
X.head()
```

Out[34]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsep
0	3	122.0	0	1	0	2	0	64.1	48.8	0	130	
1	3	122.0	0	1	0	2	0	64.1	48.8	0	130	
2	1	122.0	0	1	2	2	0	65.5	52.4	5	152	
3	2	164.0	1	1	3	1	0	66.2	54.3	3	109	
4	2	164.0	1	1	3	0	0	66.4	54.3	3	136	

In [36]:
```python
#k-fold validation on ridge model
#cross_val_score(model,feature,target,cv=value)
cross_val_score(l2,X,y,cv=4)
#returns r2 score for all 4 parts of data
```

Out[36]:
```
array([0.71176474, 0.86474228, 0.37640664, 0.47020196])
```

In [37]:
```python
#k-fold validation for lasso model
cv1=cross_val_score(l1,X,y,cv=4)
cv1
```

Out[37]:
```
array([0.74048997, 0.8346221 , 0.41264006, 0.47040374])
```