In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
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	eng
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc	
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc	

In [3]:
```
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 [4]:
```
df["normalized-losses"].replace("?", np.nan, inplace=True)
df["horsepower"].replace("?", np.nan, inplace=True)

df["normalized-losses"] = df["normalized-losses"].astype("float")
df["horsepower"] = df["horsepower"].astype("float")

nlmean = df["normalized-losses"].mean()
hpmean = df["horsepower"].mean()

df["normalized-losses"].fillna(nlmean, inplace=True)
df["horsepower"].fillna(hpmean, inplace=True)
```

In [ ]:
```
df.info()
```

In [5]:
```
df_num = df.select_dtypes(["int64", "float64"])
df_cat = df.select_dtypes(object)
```

In [6]:
```
df_cat
```

Out[6]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type
0	alfa-romero	gas	convertible	rwd	front	dohc
1	alfa-romero	gas	convertible	rwd	front	dohc
2	alfa-romero	gas	hatchback	rwd	front	ohcv
3	audi	gas	sedan	fwd	front	ohc
4	audi	gas	sedan	4wd	front	ohc
...	...	...	...	...	...	...
200	volvo	gas	sedan	rwd	front	ohc
201	volvo	gas	sedan	rwd	front	ohc
202	volvo	gas	sedan	rwd	front	ohcv
203	volvo	diesel	sedan	rwd	front	ohc
204	volvo	gas	sedan	rwd	front	ohc

205 rows × 6 columns

In [7]:
```
from sklearn.preprocessing import LabelEncoder
```

In [8]:
```
for col in df_cat:
    le = LabelEncoder()
    df_cat[col] = le.fit_transform(df_cat[col])
```

In [9]:
```
df_cat
```

Out[9]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type
0	0	1	0	2	0	0
1	0	1	0	2	0	0
2	0	1	2	2	0	5
3	1	1	3	1	0	3
4	1	1	3	0	0	3
...	...	...	...	...	...	...
200	21	1	3	2	0	3
201	21	1	3	2	0	3
202	21	1	3	2	0	5
203	21	0	3	2	0	3
204	21	1	3	2	0	3

205 rows × 6 columns

In [10]:
```
df = pd.concat([df_cat, df_num], axis=1)
```

In [11]:
```
df.head()
```

Out[11]:

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

In [12]:
```
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [13]:
```
x
```

Out[13]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type	symboling	normalized-losses	width	height	engine-size
0	0	1	0	2	0	0	3	122.0	64.1	48.8	130
1	0	1	0	2	0	0	3	122.0	64.1	48.8	130
2	0	1	2	2	0	5	1	122.0	65.5	52.4	152
3	1	1	3	1	0	3	2	164.0	66.2	54.3	109
4	1	1	3	0	0	3	2	164.0	66.4	54.3	136
...	...	...	...	...	...	...	...	...	...	...	...
200	21	1	3	2	0	3	-1	95.0	68.9	55.5	141
201	21	1	3	2	0	3	-1	95.0	68.8	55.5	141
202	21	1	3	2	0	5	-1	95.0	68.9	55.5	173
203	21	0	3	2	0	3	-1	95.0	68.9	55.5	145
204	21	1	3	2	0	3	-1	95.0	68.9	55.5	141

205 rows × 14 columns

In [14]:
```
y
```

Out[14]:
```
0      13495
1      16500
2      16500
3      13950
4      17450
       ...
200    16845
201    19045
202    21485
203    22470
204    22625
Name: price, Length: 205, dtype: int64
```

In [15]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size=0.3, random_state=
```

In [16]:
```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(xtrain, ytrain)
```

Out[16]:
```
LinearRegression()
```

In [17]:
```
train = linreg.score(xtrain, ytrain) # training acc
test = linreg.score(xtest, ytest) # testing acc
print(f"Traning Result -: {train}")
print(f"Test Result -: {test}")
```

```
Traning Result -: 0.8504573774895473
Test Result -: 0.7965566780397362
```

In [ ]:
```
#High training acc and low testing acc.
#low training error -: low bias
#high test error -: high variance
#overfitting
```

In [19]:
```
from sklearn.linear_model import Ridge, Lasso
```

In [ ]:
```
#L2 regularization
```

In [20]:
```
l2=Ridge(alpha=5)
l2.fit(xtrain,ytrain)
```

Out[20]:
```
Ridge(alpha=5)
```

In [21]:
```
train = l2.score(xtrain, ytrain) # training acc
test = l2.score(xtest, ytest) # testing acc
print(f"Traning Result -: {train}")
print(f"Test Result -: {test}")
```

```
Traning Result -: 0.8214255248858175
Test Result -: 0.8141745853539419
```

In [22]:
```
#hypertuning lambda/ alpha value
for i in range(1,30):
    l2=Ridge(alpha=i)
    l2.fit(xtrain,ytrain)
    test = l2.score(xtest, ytest)
    print(f"value of lambda {i} test score {test}")
```

...

In [ ]:
```
#final l2 model with best lambda value
```

In [23]:
```
l2=Ridge(alpha=11)
l2.fit(xtrain,ytrain)
```

Out[23]:
```
Ridge(alpha=11)
```

In [24]:
```
train = l2.score(xtrain, ytrain) # training acc
test = l2.score(xtest, ytest) # testing acc
print(f"Traning Result -: {train}")
print(f"Test Result -: {test}")
```

```
Traning Result -: 0.8096241695970485
Test Result -: 0.815027724543179
```

In [ ]:
```
#l1 regularization (LASSO - Least absolute and selection operator)
```

In [29]:
```
for i in range(200,250):
    l1=Lasso(alpha=i)
    l1.fit(xtrain,ytrain)
    test = l1.score(xtest, ytest)
    print(f"value of lambda {i} test score {test}")
```

...

In [32]:
```
#final Lasso model with best value of lambda
l1=Lasso(alpha=210)
l1.fit(xtrain,ytrain)
train = l1.score(xtrain, ytrain)
test = l1.score(xtest, ytest) # testing acc
print(f"Training Result -: {train}")
print(f"Test Result -: {test}")
```

```
Training Result -: 0.7934746040859662
Test Result -: 0.8139617970312925
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
l1.coef_
```

Cross Validation

In [ ]:
```
from sklearn.model_selection import KFold
```

In [ ]:
```
kf = KFold(n_splits=5, shuffle=False).split(range(25))
```

In [ ]:
```
print("{} {:^61} {}".format("Iteration", "Training Set Observation", "Testing Set

for iteration, data in enumerate(kf, start=1):
    print("{:^9} {} {:^25}".format(iteration, data[0], str(data[1])))
```

In [33]:
```
from sklearn.model_selection import cross_val_score
```

In [49]:
```
cvs = cross_val_score(l1, x, y, cv=4)
```

In [50]:
```
cvs
```

Out[50]:
```
array([0.76793604, 0.8173824 , 0.43616909, 0.44896261])
```

In [51]:
```
cvs.mean()
```

Out[51]:
```
0.617612537467179
```

In [ ]:

In [ ]: