Import Libraries

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
```

ETL - Explore Transform Load

```
auto_df = pd.read_csv('automobiles.csv',index_col=0)
```

```
auto_df.head()
```

```
           normalized-losses         make fuel-type aspiration  num-
of-doors  \
symboling

3                      118.0  alfa-romero       gas        std
2.0
3                      118.0  alfa-romero       gas        std
2.0
1                      118.0  alfa-romero       gas        std
2.0
2                      164.0         audi       gas        std
4.0
2                      164.0         audi       gas        std
4.0

            body-style drive-wheels engine-location  wheel-base
length  ...  \
symboling
...
3          convertible          rwd           front        88.6
168.8  ...
3          convertible          rwd           front        88.6
168.8  ...
1            hatchback          rwd           front        94.5
171.2  ...
2                sedan          fwd           front        99.8
176.6  ...
2                sedan          4wd           front        99.4
176.6  ...

           engine-size  fuel-system  bore stroke  compression-ratio  \
symboling
```

```
3                  130         mpfi  3.47   2.68                9.0
3                  130         mpfi  3.47   2.68                9.0
1                  152         mpfi  2.68   3.47                9.0
2                  109         mpfi  3.19   3.40               10.0
2                  136         mpfi  3.19   3.40                8.0

           horsepower peak-rpm  city-mpg  highway-mpg  price
symboling
3               111.0   5000.0        21           27  13495
3               111.0   5000.0        21           27  16500
1               154.0   5000.0        19           26  16500
2               102.0   5500.0        24           30  13950
2               115.0   5500.0        18           22  17450

[5 rows x 25 columns]
```

The data we currently have , have already gone through the process of

- Handling the Missing values &
- EDA - Exploratory Data Analysis

from the ipynb file 'Analysing data with python (automobile csv Data Wrangling).ipynb'

Data Exploration

```
auto_df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 201 entries, 3 to -1
Data columns (total 25 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   normalized-losses  201 non-null    float64
 1   make               201 non-null    object
 2   fuel-type          201 non-null    object
 3   aspiration         201 non-null    object
 4   num-of-doors       201 non-null    float64
 5   body-style         201 non-null    object
 6   drive-wheels       201 non-null    object
 7   engine-location    201 non-null    object
 8   wheel-base         201 non-null    float64
 9   length             201 non-null    float64
 10  width              201 non-null    float64
 11  height             201 non-null    float64
 12  curb-weight        201 non-null    int64
 13  engine-type        201 non-null    object
 14  num-of-cylinders   201 non-null    int64
 15  engine-size        201 non-null    int64
```

```
 16   fuel-system         201 non-null    object
 17   bore                201 non-null    float64
 18   stroke              201 non-null    float64
 19   compression-ratio   201 non-null    float64
 20   horsepower          201 non-null    float64
 21   peak-rpm            201 non-null    float64
 22   city-mpg            201 non-null    int64
 23   highway-mpg         201 non-null    int64
 24   price               201 non-null    int64
dtypes: float64(11), int64(6), object(8)
memory usage: 40.8+ KB
```

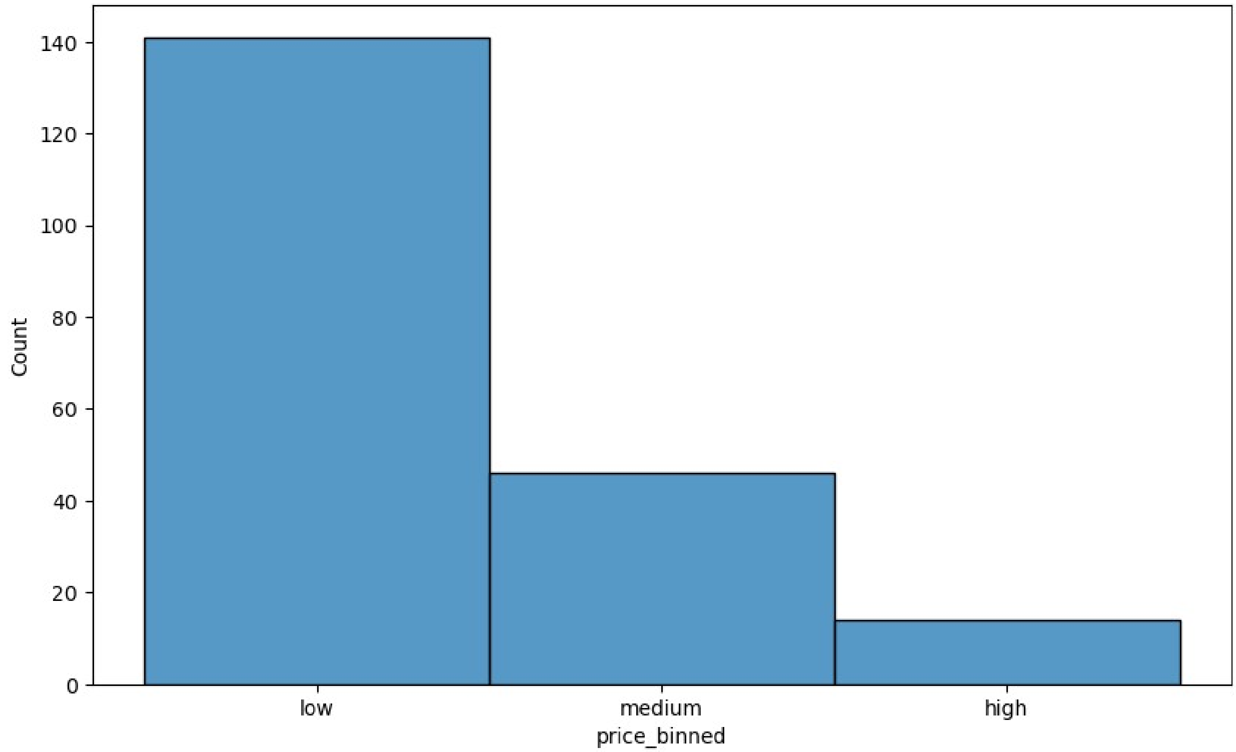
Binning

- Lets Bin the Price column into low, medium, high

```
bins = np.linspace(min(auto_df['price']),max(auto_df['price']) , 4)

price_groups_ = ['low','medium','high']

auto_df['price_binned'] = pd.cut( auto_df['price'],
                                bins,
                                labels = price_groups_,
                                include_lowest=True)

auto_df

           normalized-losses         make fuel-type aspiration  num-
of-doors  \
symboling

 3                     118.0  alfa-romero       gas        std
2.0
 3                     118.0  alfa-romero       gas        std
2.0
 1                     118.0  alfa-romero       gas        std
2.0
 2                     164.0         audi       gas        std
4.0
 2                     164.0         audi       gas        std
4.0
...                      ...          ...       ...        ...
...
-1                      95.0        volvo       gas        std
4.0
-1                      95.0        volvo       gas      turbo
4.0
-1                      95.0        volvo       gas        std
4.0
-1                      95.0        volvo    diesel      turbo
4.0
```

```
-1                      95.0        volvo       gas      turbo
4.0

            body-style drive-wheels engine-location  wheel-base
length  ...  \
symboling
...
 3         convertible          rwd           front        88.6
168.8  ...
 3         convertible          rwd           front        88.6
168.8  ...
 1           hatchback          rwd           front        94.5
171.2  ...
 2               sedan          fwd           front        99.8
176.6  ...
 2               sedan          4wd           front        99.4
176.6  ...
...                ...          ...             ...         ...      ..
.  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...

           fuel-system  bore  stroke compression-ratio  horsepower
peak-rpm  \
symboling

 3                mpfi  3.47    2.68               9.0       111.0
5000.0
 3                mpfi  3.47    2.68               9.0       111.0
5000.0
 1                mpfi  2.68    3.47               9.0       154.0
5000.0
 2                mpfi  3.19    3.40              10.0       102.0
5500.0
 2                mpfi  3.19    3.40               8.0       115.0
5500.0
...                ...   ...     ...               ...         ...
...
-1                mpfi  3.78    3.15               9.5       114.0
5400.0
-1                mpfi  3.78    3.15               8.7       160.0
5300.0
```

```
-1                  mpfi  3.58    2.87              8.8        134.0
5500.0
-1                   idi  3.01    3.40             23.0        106.0
4800.0
-1                  mpfi  3.78    3.15              9.5        114.0
5400.0

          city-mpg  highway-mpg  price  price_binned
symboling
 3              21           27  13495           low
 3              21           27  16500        medium
 1              19           26  16500        medium
 2              24           30  13950           low
 2              18           22  17450        medium
...            ...          ...    ...           ...
-1              23           28  16845        medium
-1              19           25  19045        medium
-1              18           23  21485        medium
-1              26           27  22470        medium
-1              19           25  22625        medium

[201 rows x 26 columns]

# an histogram of auto_df['price_binned']

plt.figure(figsize = (10,6))

sns.histplot(auto_df['price_binned'])

plt.show()
```



- Let focus on the fuel column

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

fuel-type
gas       181
diesel     19
109         1
Name: count, dtype: int64

fuel_types = pd.get_dummies(auto_df['fuel-type'],prefix='_',dtype =
'int')

auto_df = pd.concat([auto_df,fuel_types],axis=1)

auto_df

           normalized-losses         make fuel-type aspiration  num-
of-doors  \
symboling

 3                      118.0  alfa-romero       gas        std
2.0
 3                      118.0  alfa-romero       gas        std
2.0
 1                      118.0  alfa-romero       gas        std
2.0
 2                      164.0         audi       gas        std
```

```
4.0
 2                      164.0          audi       gas         std
4.0
...                       ...           ...       ...         ...
...
-1                       95.0         volvo       gas         std
4.0
-1                       95.0         volvo       gas       turbo
4.0
-1                       95.0         volvo       gas         std
4.0
-1                       95.0         volvo    diesel       turbo
4.0
-1                       95.0         volvo       gas       turbo
4.0

            body-style drive-wheels engine-location  wheel-base
length  ...  \
symboling
...
 3          convertible          rwd           front        88.6
168.8  ...
 3          convertible          rwd           front        88.6
168.8  ...
 1            hatchback          rwd           front        94.5
171.2  ...
 2                sedan          fwd           front        99.8
176.6  ...
 2                sedan          4wd           front        99.4
176.6  ...
...                 ...          ...             ...         ...      ..
.  ...
-1                sedan          rwd           front       109.1
188.8  ...
-1                sedan          rwd           front       109.1
188.8  ...
-1                sedan          rwd           front       109.1
188.8  ...
-1                sedan          rwd           front       109.1
188.8  ...
-1                sedan          rwd           front       109.1
188.8  ...

           compression-ratio  horsepower  peak-rpm city-mpg  highway-
mpg  \
symboling

 3                       9.0       111.0    5000.0       21
27
 3                       9.0       111.0    5000.0       21
```

```
27
 1                        9.0        154.0     5000.0        19
26
 2                       10.0        102.0     5500.0        24
30
 2                        8.0        115.0     5500.0        18
22
...                       ...          ...        ...       ...            .
..
-1                        9.5        114.0     5400.0        23
28
-1                        8.7        160.0     5300.0        19
25
-1                        8.8        134.0     5500.0        18
23
-1                       23.0        106.0     4800.0        26
27
-1                        9.5        114.0     5400.0        19
25

           price price_binned  __109  __diesel  __gas
symboling
 3         13495          low      0         0      1
 3         16500       medium      0         0      1
 1         16500       medium      0         0      1
 2         13950          low      0         0      1
 2         17450       medium      0         0      1
...          ...          ...    ...       ...    ...
-1         16845       medium      0         0      1
-1         19045       medium      0         0      1
-1         21485       medium      0         0      1
-1         22470       medium      0         1      0
-1         22625       medium      0         0      1

[201 rows x 29 columns]
```

lets convert the mpg to KmL

- the formula for converting the mpg to kml is to multiply it with 0.425144

```
auto_df['city-mpg'] = auto_df['city-mpg'] * 0.425144

auto_df['highway-mpg'] = auto_df['highway-mpg'] * 0.425144

auto_df.rename(columns = {'city-mpg':'city_kml','highway-mpg':'hiwy
kml'},inplace=True)

auto_df

           normalized-losses         make fuel-type aspiration  num-
of-doors  \
```

```
symboling

 3                    118.0  alfa-romero       gas        std
2.0
 3                    118.0  alfa-romero       gas        std
2.0
 1                    118.0  alfa-romero       gas        std
2.0
 2                    164.0         audi       gas        std
4.0
 2                    164.0         audi       gas        std
4.0
...                     ...          ...       ...        ...
...
-1                     95.0        volvo       gas        std
4.0
-1                     95.0        volvo       gas      turbo
4.0
-1                     95.0        volvo       gas        std
4.0
-1                     95.0        volvo    diesel      turbo
4.0
-1                     95.0        volvo       gas      turbo
4.0

            body-style drive-wheels engine-location  wheel-base
length  ...  \
symboling
...
 3         convertible          rwd           front        88.6
168.8  ...
 3         convertible          rwd           front        88.6
168.8  ...
 1           hatchback          rwd           front        94.5
171.2  ...
 2               sedan          fwd           front        99.8
176.6  ...
 2               sedan          4wd           front        99.4
176.6  ...
...                ...          ...             ...         ...      ..
.  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
```

```
-1                sedan          rwd           front       109.1
188.8  ...

           compression-ratio  horsepower  peak-rpm   city_kml   hiwy
kml  \
symboling

 3                       9.0       111.0    5000.0   8.928024
11.478888
 3                       9.0       111.0    5000.0   8.928024
11.478888
 1                       9.0       154.0    5000.0   8.077736
11.053744
 2                      10.0       102.0    5500.0  10.203456
12.754320
 2                       8.0       115.0    5500.0   7.652592
9.353168
...                      ...         ...       ...        ...         .
..
-1                       9.5       114.0    5400.0   9.778312
11.904032
-1                       8.7       160.0    5300.0   8.077736
10.628600
-1                       8.8       134.0    5500.0   7.652592
9.778312
-1                      23.0       106.0    4800.0  11.053744
11.478888
-1                       9.5       114.0    5400.0   8.077736
10.628600

           price price_binned  __109  __diesel  __gas
symboling
 3         13495          low      0         0      1
 3         16500       medium      0         0      1
 1         16500       medium      0         0      1
 2         13950          low      0         0      1
 2         17450       medium      0         0      1
...          ...          ...    ...       ...    ...
-1         16845       medium      0         0      1
-1         19045       medium      0         0      1
-1         21485       medium      0         0      1
-1         22470       medium      0         1      0
-1         22625       medium      0         0      1

[201 rows x 29 columns]

auto_df.columns

Index(['normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-
of-doors',
```

```
       'body-style', 'drive-wheels', 'engine-location', 'wheel-base',
'length',
       'width', 'height', 'curb-weight', 'engine-type', 'num-of-
cylinders',
       'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-
ratio',
       'horsepower', 'peak-rpm', 'city_kml', 'hiwy kml', 'price',
       'price_binned', '__109', '__diesel', '__gas'],
      dtype='object')
```

```
auto_df[['length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore']]
```

```
           length  width  height  curb-weight engine-type  num-of-
cylinders  \
symboling

 3          168.8   64.1    48.8         2548        dohc
4
 3          168.8   64.1    48.8         2548        dohc
4
 1          171.2   65.5    52.4         2823        ohcv
6
 2          176.6   66.2    54.3         2337         ohc
4
 2          176.6   66.4    54.3         2824         ohc
5
...           ...    ...     ...          ...         ...
...
-1          188.8   68.9    55.5         2952         ohc
4
-1          188.8   68.8    55.5         3049         ohc
4
-1          188.8   68.9    55.5         3012        ohcv
6
-1          188.8   68.9    55.5         3217         ohc
6
-1          188.8   68.9    55.5         3062         ohc
4

           engine-size fuel-system  bore
symboling
 3                 130        mpfi  3.47
 3                 130        mpfi  3.47
 1                 152        mpfi  2.68
 2                 109        mpfi  3.19
 2                 136        mpfi  3.19
...                ...         ...   ...
-1                 141        mpfi  3.78
-1                 141        mpfi  3.78
```

```
-1                 173        mpfi  3.58
-1                 145         idi  3.01
-1                 141        mpfi  3.78

[201 rows x 9 columns]
```

```
auto_df.rename(columns={"normalized-losses":"nrml_loss",
                      "num-of-doors":"nof_doors",
                       "body-style":"style",
                       "num-of-cylinders":"nof_cylndrs",
                       "compression-ratio":"cmprsn_RT",
                       "horsepower":"HP"
                      },inplace=True)
```

```
auto_df
```

```
           nrml_loss        make fuel-type aspiration  nof_doors  \
symboling
 3             118.0  alfa-romero       gas        std        2.0
 3             118.0  alfa-romero       gas        std        2.0
 1             118.0  alfa-romero       gas        std        2.0
 2             164.0         audi       gas        std        4.0
 2             164.0         audi       gas        std        4.0
...              ...          ...       ...        ...        ...
-1              95.0        volvo       gas        std        4.0
-1              95.0        volvo       gas      turbo        4.0
-1              95.0        volvo       gas        std        4.0
-1              95.0        volvo    diesel      turbo        4.0
-1              95.0        volvo       gas      turbo        4.0

                 style drive-wheels engine-location  wheel-base
length  ...  \
symboling
...
 3         convertible          rwd           front        88.6
168.8  ...
 3         convertible          rwd           front        88.6
168.8  ...
 1           hatchback          rwd           front        94.5
171.2  ...
 2               sedan          fwd           front        99.8
176.6  ...
 2               sedan          4wd           front        99.4
176.6  ...
...                ...          ...             ...         ...      ..
.  ...
-1               sedan          rwd           front       109.1
188.8  ...
-1               sedan          rwd           front       109.1
188.8  ...
```

```
-1                 sedan          rwd           front       109.1
188.8  ...
-1                 sedan          rwd           front       109.1
188.8  ...
-1                 sedan          rwd           front       109.1
188.8  ...

           cmprsn_RT     HP  peak-rpm   city_kml   hiwy kml  price  \
symboling
 3               9.0  111.0    5000.0   8.928024  11.478888  13495
 3               9.0  111.0    5000.0   8.928024  11.478888  16500
 1               9.0  154.0    5000.0   8.077736  11.053744  16500
 2              10.0  102.0    5500.0  10.203456  12.754320  13950
 2               8.0  115.0    5500.0   7.652592   9.353168  17450
...              ...    ...       ...        ...        ...    ...
-1               9.5  114.0    5400.0   9.778312  11.904032  16845
-1               8.7  160.0    5300.0   8.077736  10.628600  19045
-1               8.8  134.0    5500.0   7.652592   9.778312  21485
-1              23.0  106.0    4800.0  11.053744  11.478888  22470
-1               9.5  114.0    5400.0   8.077736  10.628600  22625

          price_binned  __109  __diesel  __gas
symboling
 3                 low      0         0      1
 3              medium      0         0      1
 1              medium      0         0      1
 2                 low      0         0      1
 2              medium      0         0      1
...                ...    ...       ...    ...
-1              medium      0         0      1
-1              medium      0         0      1
-1              medium      0         0      1
-1              medium      0         1      0
-1              medium      0         0      1

[201 rows x 29 columns]
```

```
auto_df.duplicated().sum()
```

```
0
```

Using Groupby()

```
auto_df_grp = auto_df[['drive-wheels','style','price']]

auto_test_grp_df = auto_df_grp.groupby(['drive-wheels','style'],
as_index = False).mean()
auto_test_grp_df
```

```
   drive-wheels        style         price
0           109          109    109.000000
1           4wd    hatchback   7603.000000
2           4wd        sedan  12647.333333
3           4wd        wagon   9095.750000
4           fwd  convertible  11595.000000
5           fwd      hardtop   8249.000000
6           fwd    hatchback   8396.387755
7           fwd        sedan   9811.800000
8           fwd        wagon   9997.333333
9           rwd  convertible  23949.600000
10          rwd      hardtop  24202.714286
11          rwd    hatchback  14337.777778
12          rwd        sedan  21808.057143
13          rwd        wagon  16994.222222
```

Using Pivot

```
new_df = auto_test_grp_df.pivot_table(index='drive-
wheels',columns='style', values='price')
new_df

style           109  convertible      hardtop     hatchback
sedan  \
drive-wheels

109           109.0          NaN          NaN           NaN
NaN
4wd             NaN          NaN          NaN   7603.000000
12647.333333
fwd             NaN      11595.0  8249.000000   8396.387755
9811.800000
rwd             NaN      23949.6 24202.714286  14337.777778
21808.057143

style                wagon
drive-wheels
109                    NaN
4wd            9095.750000
fwd            9997.333333
rwd           16994.222222
```

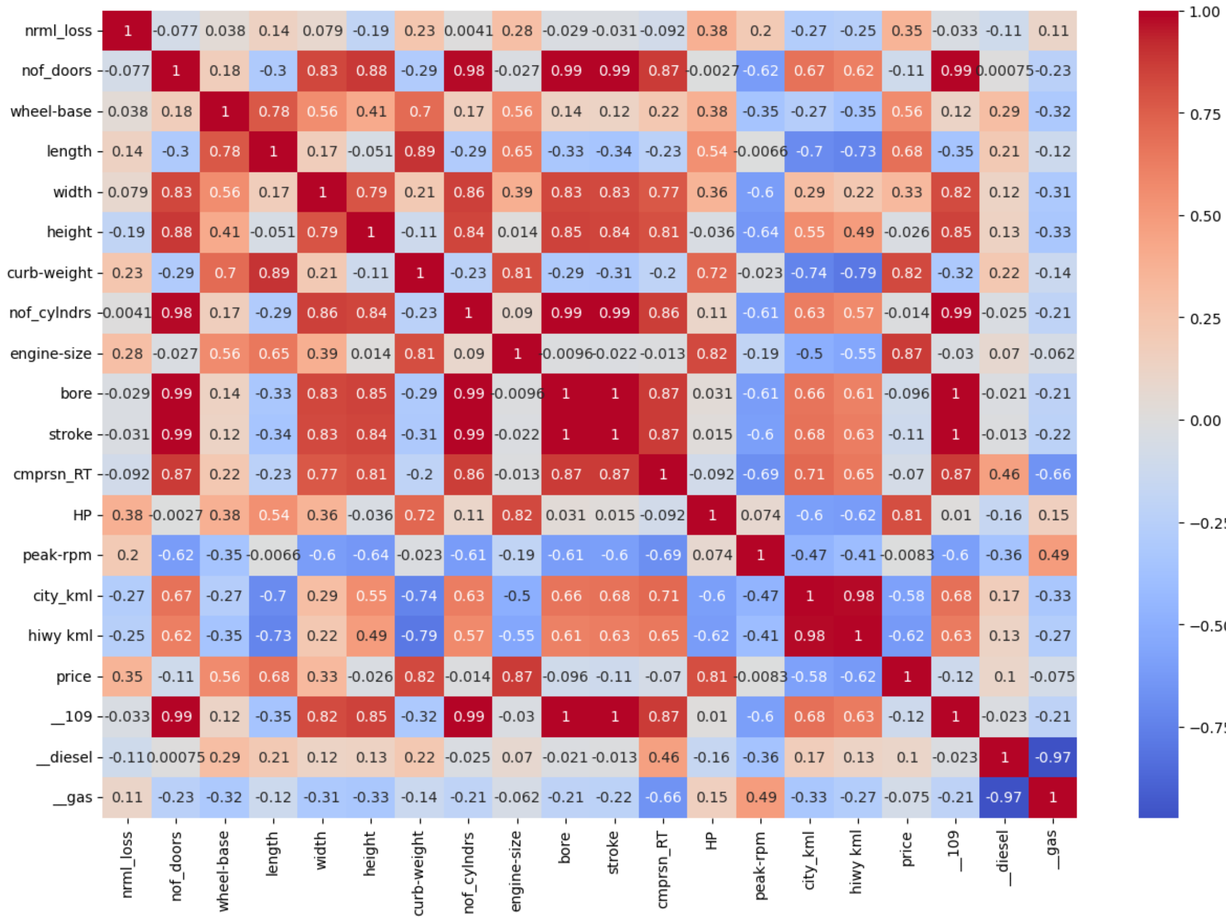
- Let us now separate the numerical columns to a new dataframe

```
auto_df_num = auto_df.select_dtypes(include=['int','float'])

auto_df_num.columns

Index(['nrml_loss', 'nof_doors', 'wheel-base', 'length', 'width',
'height',
```

```
       'curb-weight', 'nof_cylndrs', 'engine-size', 'bore', 'stroke',
       'cmprsn_RT', 'HP', 'peak-rpm', 'city_kml', 'hiwy kml', 'price',
'__109',
       '__diesel', '__gas'],
      dtype='object')
```

- Let us see more with a pairplot what this numerical data means

```
(auto_df_num==None).sum()

nrml_loss      0
nof_doors      0
wheel-base     0
length         0
width          0
height         0
curb-weight    0
nof_cylndrs    0
engine-size    0
bore           0
stroke         0
cmprsn_RT      0
HP             0
peak-rpm       0
city_kml       0
hiwy kml       0
price          0
__109          0
__diesel       0
__gas          0
dtype: int64

corr = auto_df_num.corr()

# Create a mask for the negative values
mask = corr < -1


def annot_func(data, mask):
    annot = data.copy()
    annot[mask] = np.nan
    return annot

plt.figure(figsize=(15,10))

# Plot the heatmap
sns.heatmap(corr, annot=annot_func(corr, mask), mask=mask,
cmap='coolwarm', center=0)

plt.show()
```



- From the heatmap above we understood that the price variable is affected by these columns only so we will create a new variable called 'initiator' which would have all the columns which has the positive correlation towards the price that are

['nrml_loss', 'nof_doors', 'wheel-base', 'length', 'width', 'height', 'curb-weight', 'nof_cylndrs', 'engine-size', 'bore', 'stroke', 'cmprsn_RT', 'HP', 'peak-rpm'] .

```
# initiator = auto_df_num[[ 'nrml_loss', 'nof_doors', 'wheel-base',
'length', 'width', 'height', 'curb-weight', 'nof_cylndrs', 'engine-
size', 'bore', 'stroke', 'cmprsn_RT', 'HP','price'] ]
# price = auto_df_num['price']
```

- Let us see with the help of boxplot values for all these numerical columns

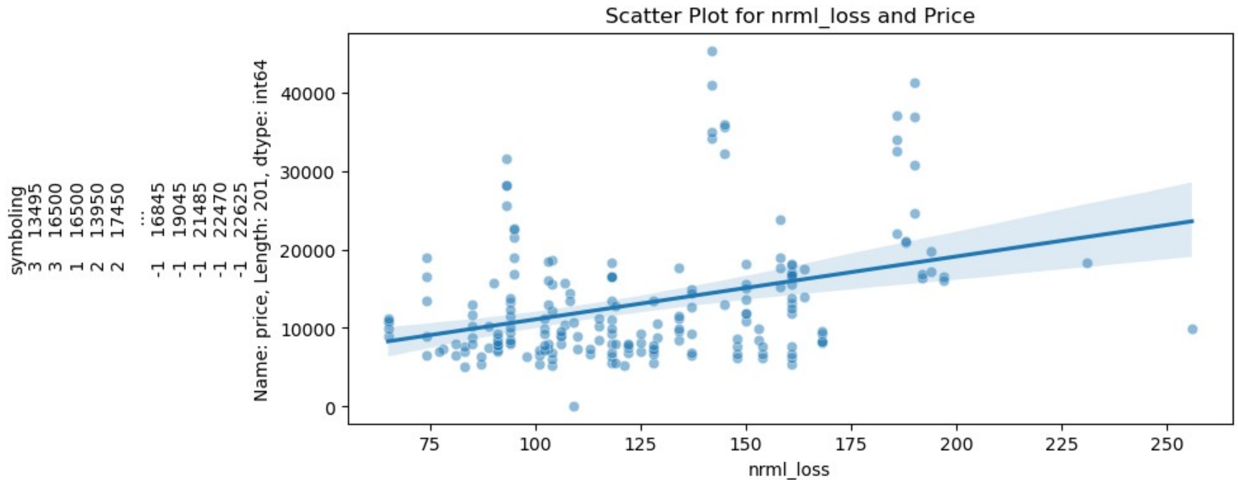
```
auto_df_num.head(5)

# print()
# print(price.head(5))
```

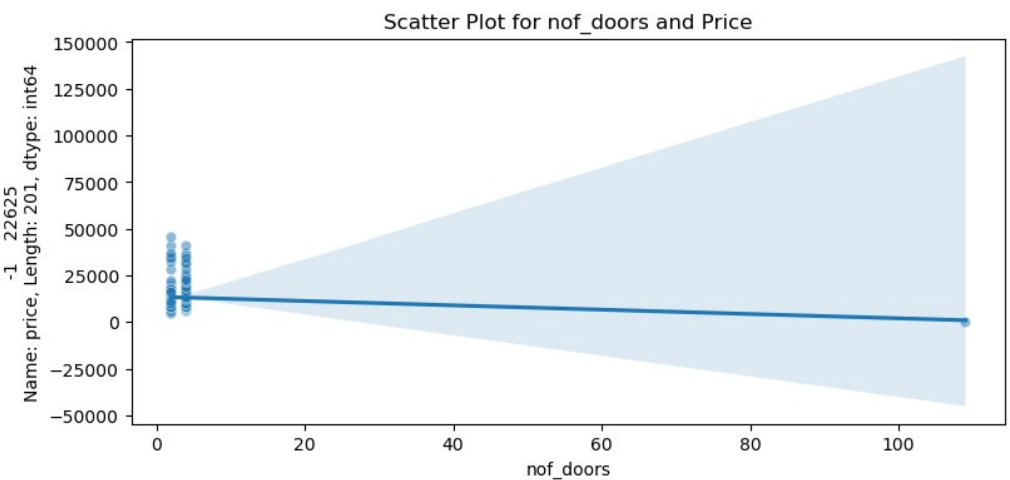
```
           nrml_loss  nof_doors  wheel-base  length  width  height  \
symboling
```

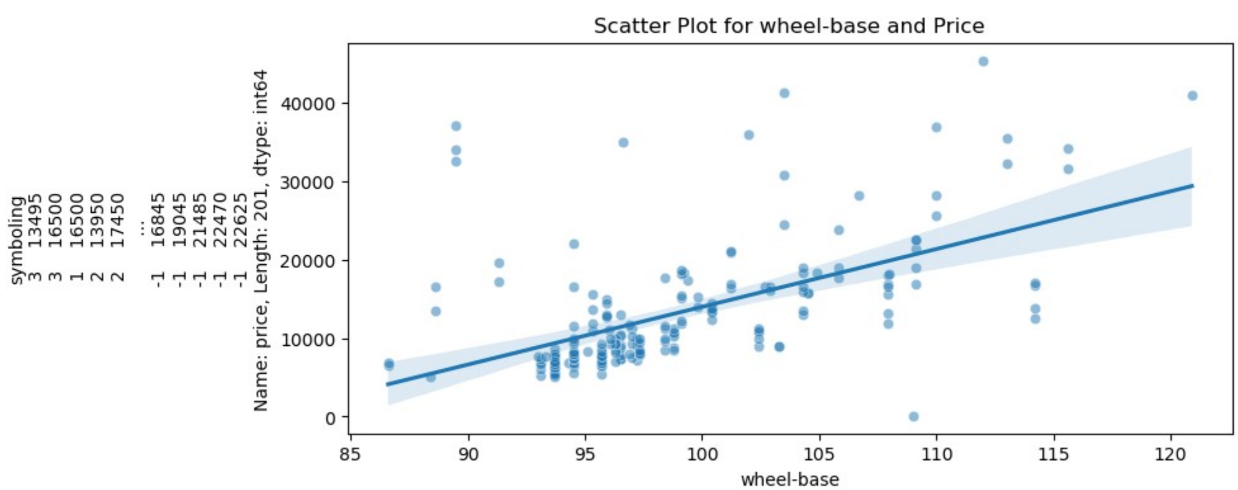
```
3                118.0        2.0         88.6   168.8   64.1    48.8
3                118.0        2.0         88.6   168.8   64.1    48.8
1                118.0        2.0         94.5   171.2   65.5    52.4
2                164.0        4.0         99.8   176.6   66.2    54.3
2                164.0        4.0         99.4   176.6   66.4    54.3

           curb-weight  nof_cylndrs  engine-size  bore  stroke
cmprsn_RT  \
symboling

3                 2548            4          130  3.47    2.68
9.0
3                 2548            4          130  3.47    2.68
9.0
1                 2823            6          152  2.68    3.47
9.0
2                 2337            4          109  3.19    3.40
10.0
2                 2824            5          136  3.19    3.40
8.0

              HP  peak-rpm   city_kml   hiwy kml  price  __109
__diesel  \
symboling

3          111.0    5000.0   8.928024  11.478888  13495      0
0
3          111.0    5000.0   8.928024  11.478888  16500      0
0
1          154.0    5000.0   8.077736  11.053744  16500      0
0
2          102.0    5500.0  10.203456  12.754320  13950      0
0
2          115.0    5500.0   7.652592   9.353168  17450      0
0

           __gas
symboling
3              1
3              1
1              1
2              1
2              1
```

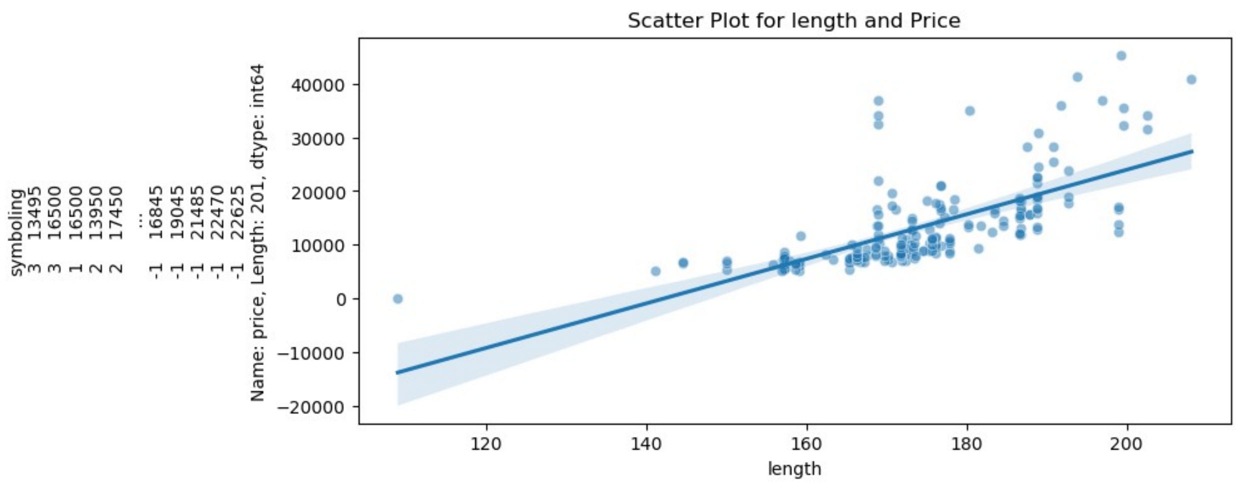
```python
for i in auto_df_num.columns:
    plt.figure(figsize=(9,4))

    sns.scatterplot(data =auto_df_num ,x = i , y = 'price', alpha=.5)
    sns.regplot(data = auto_df_num ,x = i , y ='price',scatter=False)
```

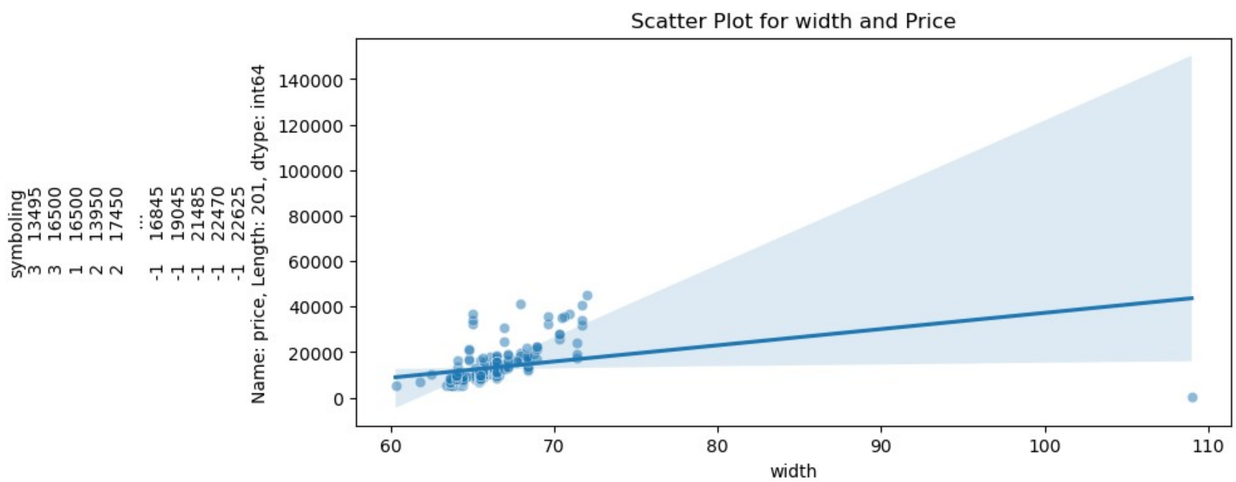
```
    plt.xlabel(f"{i}")
    plt.ylabel(f"{auto_df_num['price']}")
    plt.title(f"Scatter Plot for {i} and Price")
    plt.show()
```

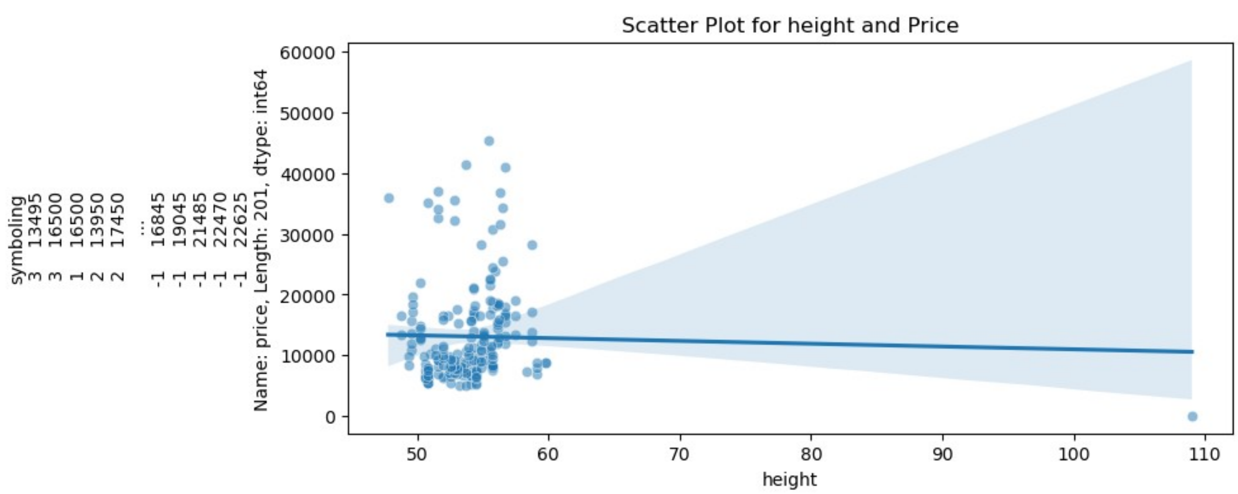


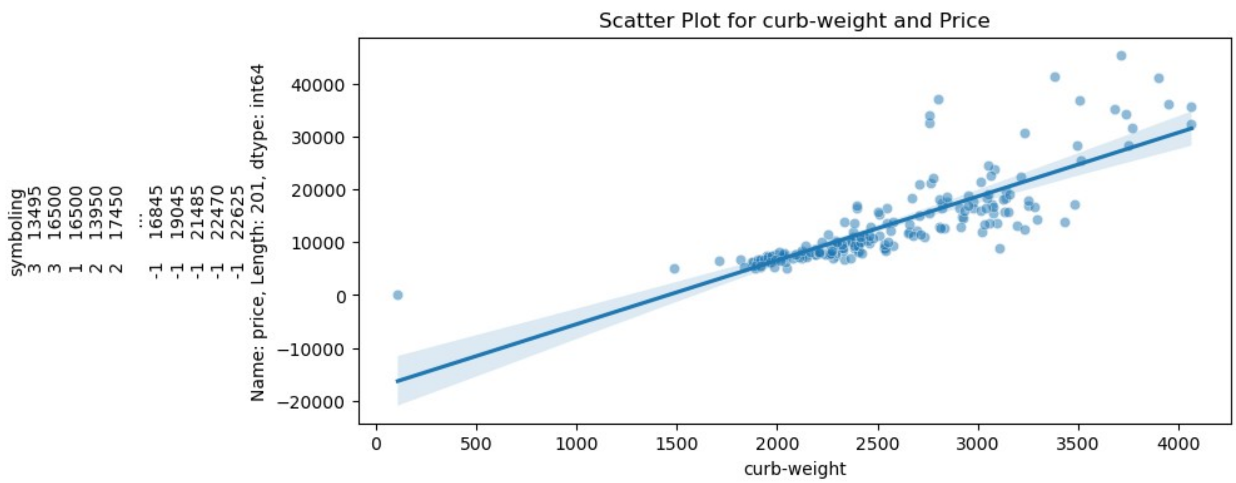


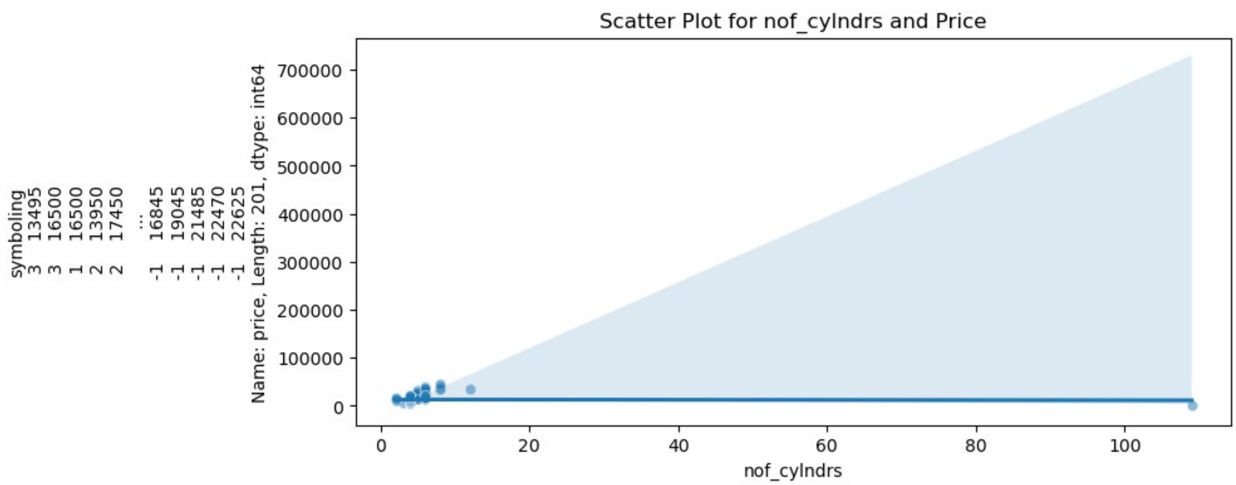


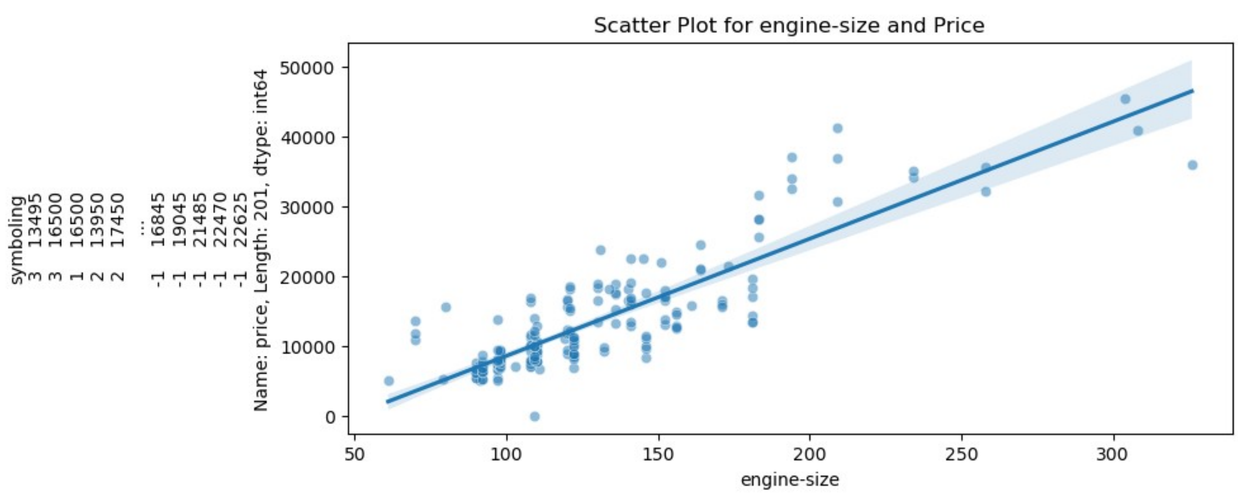

Scatter Plot for length and Price
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64
length

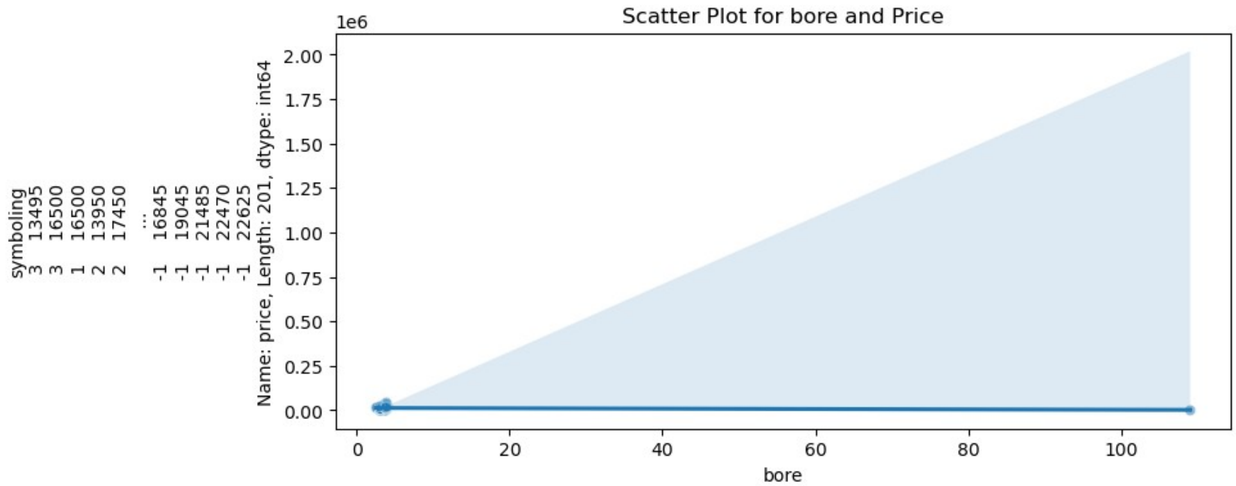

Scatter Plot for width and Price
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64
width

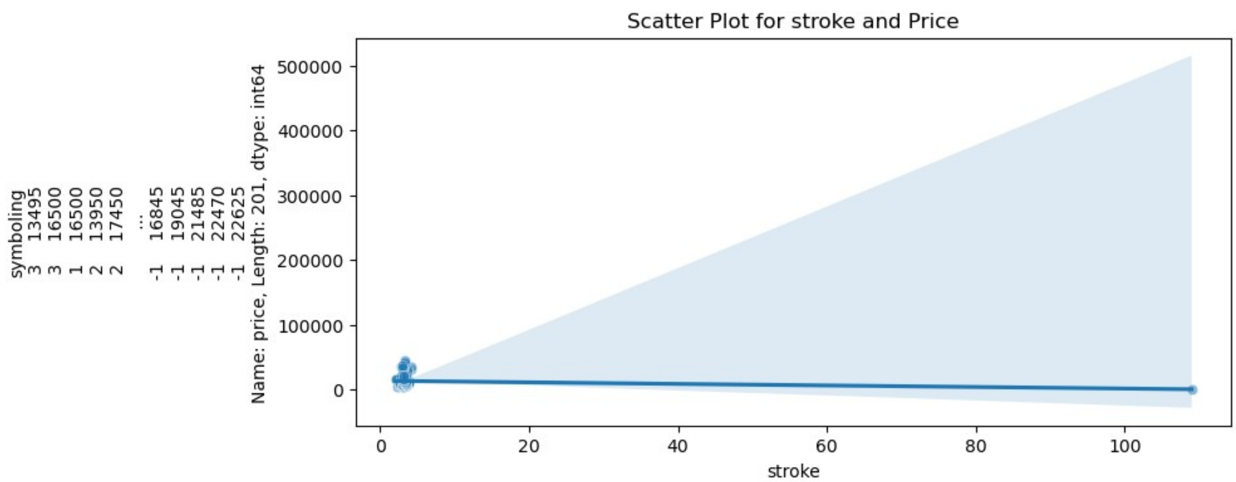

Scatter Plot for height and Price
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64
height

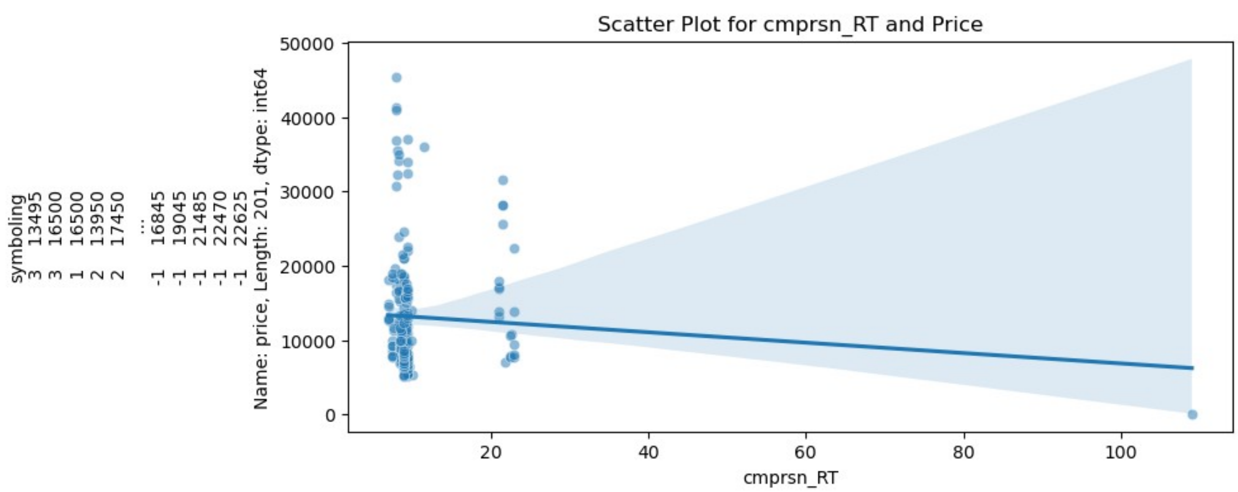

Scatter Plot for curb-weight and Price
curb-weight
Name: price, Length: 201, dtype: int64
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625

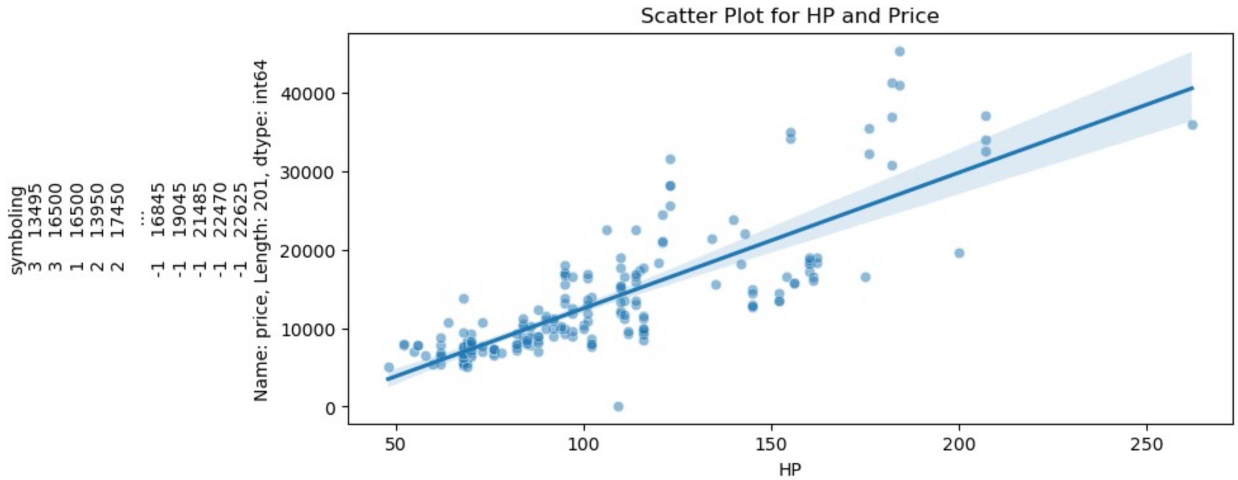

Scatter Plot for nof_cylndrs and Price
nof_cylndrs
Name: price, Length: 201, dtype: int64
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625

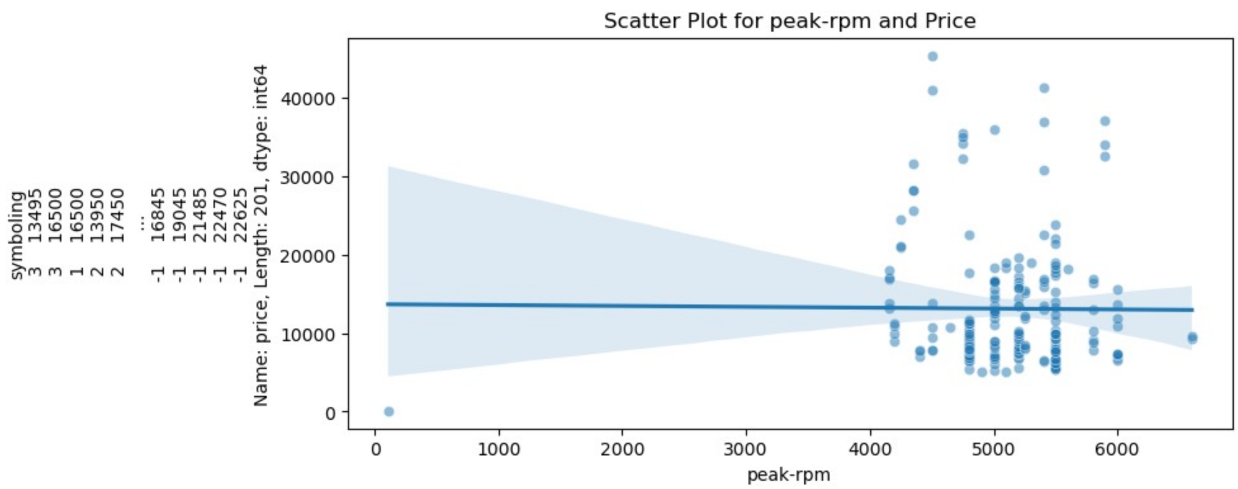

Scatter Plot for engine-size and Price
engine-size
Name: price, Length: 201, dtype: int64
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625

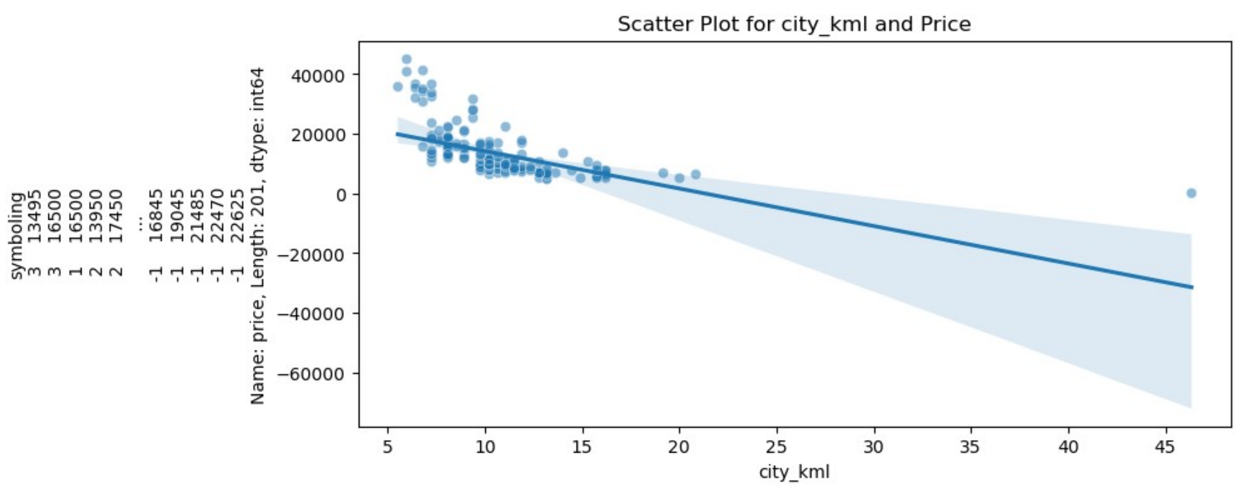

Scatter Plot for bore and Price
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0 20 40 60 80 100
bore
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64

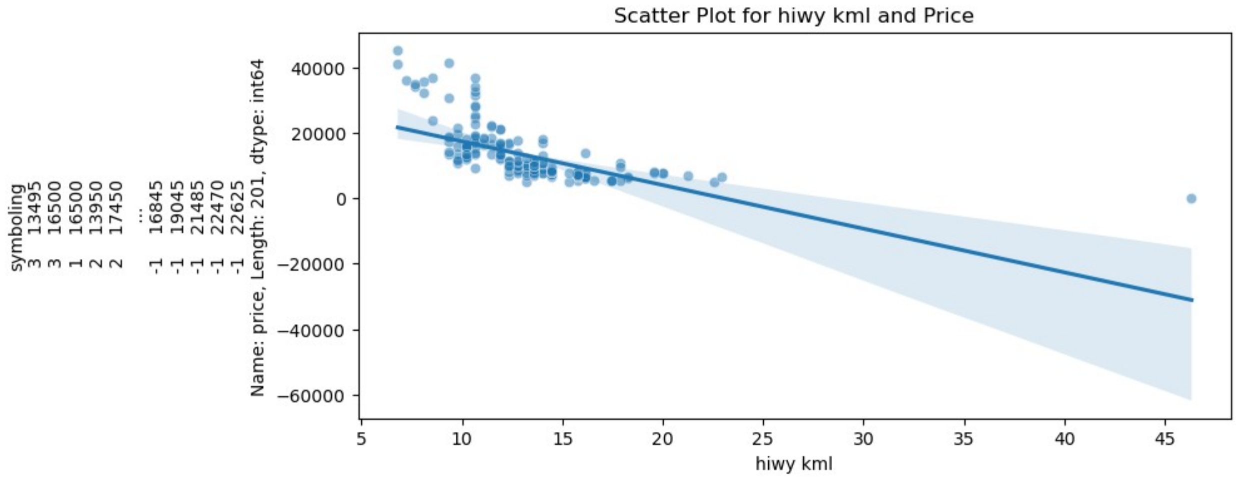

Scatter Plot for stroke and Price
500000
400000
300000
200000
100000
0
0 20 40 60 80 100
stroke
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64

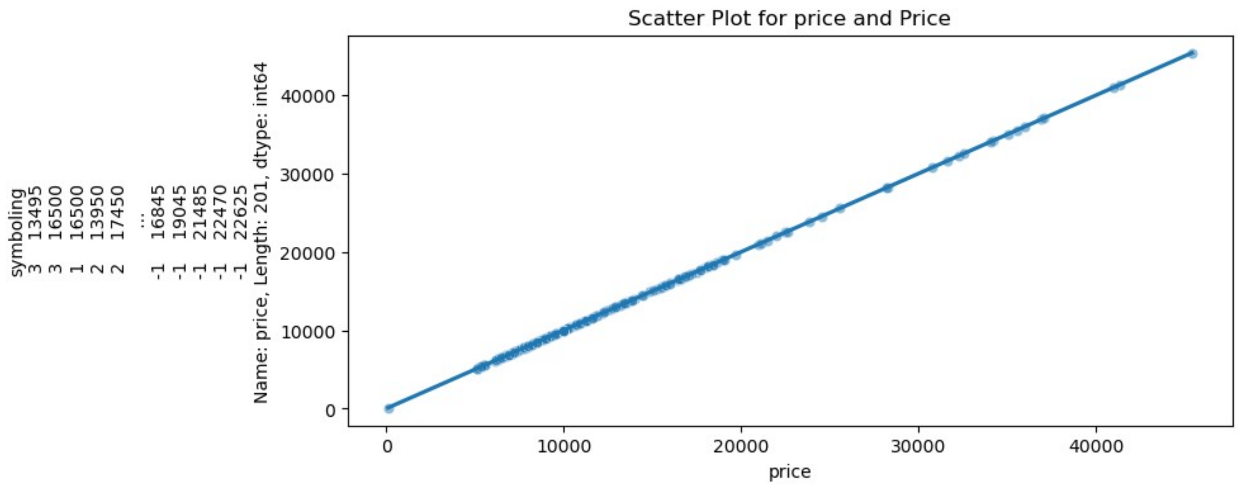

Scatter Plot for cmprsn_RT and Price
50000
40000
30000
20000
10000
0
20 40 60 80 100
cmprsn_RT
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64

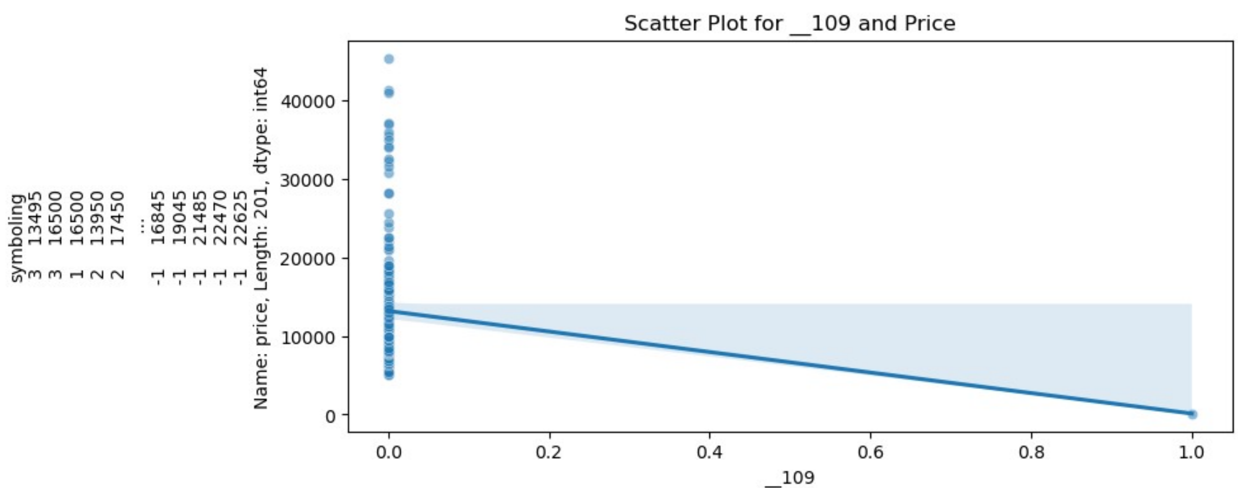

Scatter Plot for HP and Price
HP
Name: price, Length: 201, dtype: int64
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625

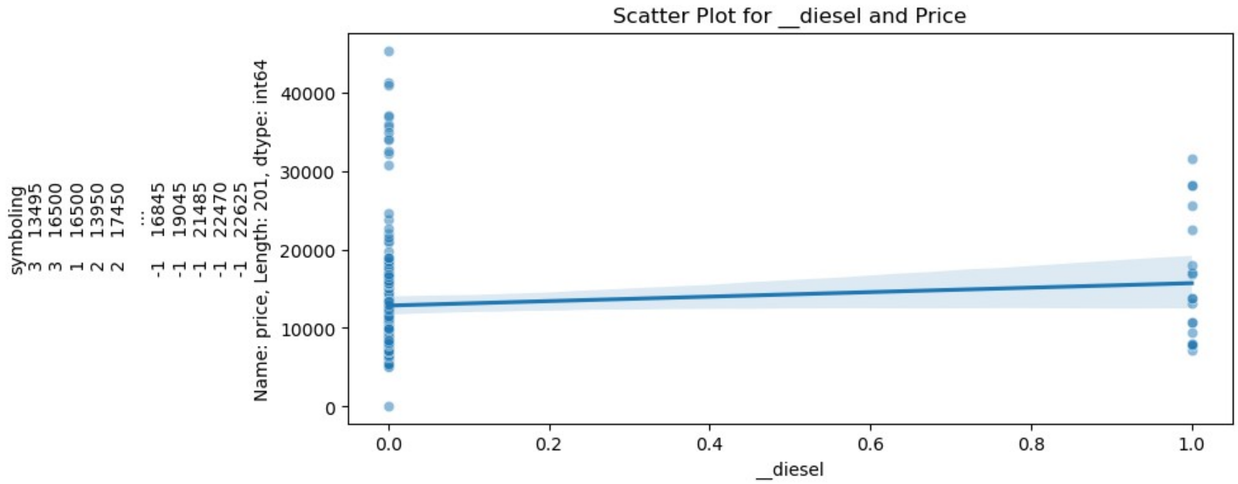

Scatter Plot for peak-rpm and Price
peak-rpm
Name: price, Length: 201, dtype: int64
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625

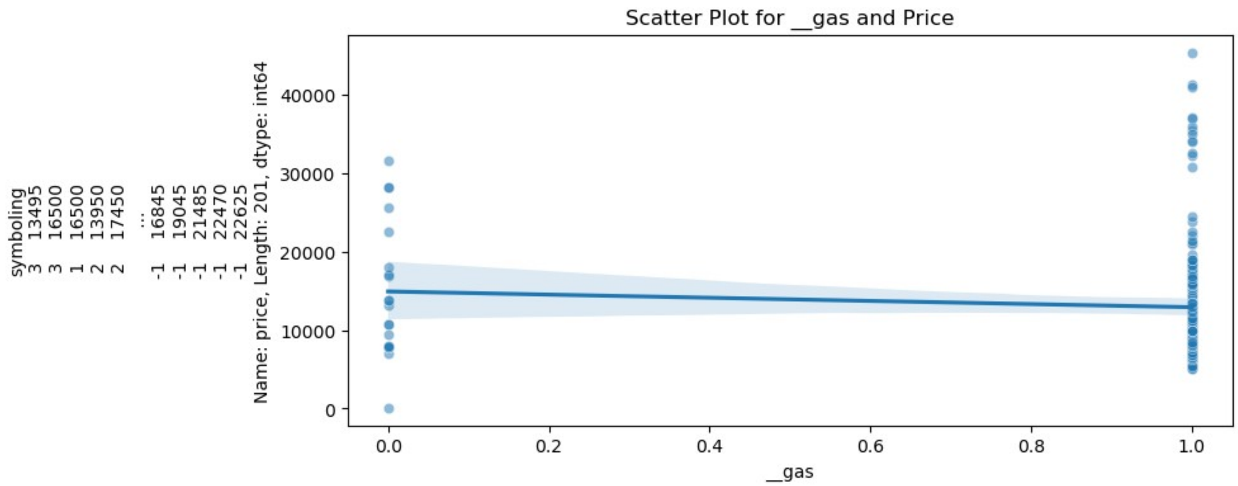

Scatter Plot for city_kml and Price
city_kml
Name: price, Length: 201, dtype: int64
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625


Scatter Plot for hiwy kml and Price
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64
hiwy kml


Scatter Plot for price and Price
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64
price


Scatter Plot for __109 and Price
symboling
3 13495
3 16500
1 16500
2 13950
2 17450
...
-1 16845
-1 19045
-1 21485
-1 22470
-1 22625
Name: price, Length: 201, dtype: int64
__109





- From the above graph we can understand that from the above columns that only some columns are having a slight linear regression feature to the graph because those can actually affect the price
- So our new columns include = { "wheel-base", "HP", "length" , "width", "curb-weight" ,"bore", "engine-size" , "nof_cylndrs" }

Data Transformation & Feature Engineering

```
initiator = auto_df_num[["nrml_loss", "wheel-base", "length", "width", "height","curb-weight", "nof_cylndrs",
                         "engine-size", "bore", "HP", "peak-rpm" , "city_kml", "hiwy kml" , "price"]]
```

Here we will perform Pearson Correlation

Pearson Correlation Analysis

P-Value for the correlation :-

- 0.0001 = Strong
- 0.05 = Moderate
- 0.1 = Weak
- 0 = No

P-corr are values that are :

- -1 => large negative
- +1 => large positive
- 0 => no correlation

```
from scipy import stats
```

```
initiator.head()
```

```
           nrml_loss  wheel-base  length  width  height  curb-weight  \
symboling
3              118.0        88.6   168.8   64.1    48.8         2548
3              118.0        88.6   168.8   64.1    48.8         2548
1              118.0        94.5   171.2   65.5    52.4         2823
2              164.0        99.8   176.6   66.2    54.3         2337
2              164.0        99.4   176.6   66.4    54.3         2824

           nof_cylndrs  engine-size  bore     HP  peak-rpm   city_kml  \
symboling
3                    4          130  3.47  111.0    5000.0   8.928024
3                    4          130  3.47  111.0    5000.0   8.928024
1                    6          152  2.68  154.0    5000.0   8.077736
2                    4          109  3.19  102.0    5500.0  10.203456
2                    5          136  3.19  115.0    5500.0   7.652592
```

```
             hiwy kml  price
symboling
3          11.478888  13495
3          11.478888  16500
1          11.053744  16500
2          12.754320  13950
2           9.353168  17450

auto_df_num.head(3)

           nrml_loss  nof_doors  wheel-base  length  width  height  \
symboling
3              118.0        2.0        88.6   168.8   64.1    48.8
3              118.0        2.0        88.6   168.8   64.1    48.8
1              118.0        2.0        94.5   171.2   65.5    52.4

           curb-weight  nof_cylndrs  engine-size  bore  stroke
cmprsn_RT  \
symboling

3                 2548            4          130  3.47    2.68
9.0
3                 2548            4          130  3.47    2.68
9.0
1                 2823            6          152  2.68    3.47
9.0

              HP  peak-rpm  city_kml   hiwy kml  price  __109
__diesel  __gas
symboling

3          111.0    5000.0  8.928024  11.478888  13495      0
0      1
3          111.0    5000.0  8.928024  11.478888  16500      0
0      1
1          154.0    5000.0  8.077736  11.053744  16500      0
0      1

# the values for the independent variables

pearson_X_comparison_val = initiator.iloc[:,:13]
pearson_Y_val = initiator['price']

pearson_X_comparison_val.head(2)

           nrml_loss  wheel-base  length  width  height  curb-
weight  \
symboling

3              118.0        88.6   168.8   64.1    48.8         2548
```

```
3                118.0        88.6   168.8   64.1    48.8         2548


           nof_cylndrs  engine-size  bore     HP  peak-rpm
city_kml  \
symboling

3                    4          130  3.47  111.0    5000.0  8.928024

3                    4          130  3.47  111.0    5000.0  8.928024


            hiwy kml
symboling
3          11.478888
3          11.478888
```

```
pearson_Y_val[:5]
```

```
symboling
3    13495
3    16500
1    16500
2    13950
2    17450
Name: price, dtype: int64
```

```
# the correlation coefficient for the dataframe
pearson_X_comparison_val is !

pearson_coef_val, p_val =
stats.pearsonr(pearson_X_comparison_val['hiwy kml'],pearson_Y_val)

print(f"pearson_coefficient : {pearson_coef_val} , p value : {p_val}")
```

```
pearson_coefficient : -0.6213854514083842 , p value :
7.405451907278584e-23
```

```
# a for loop for displaying all the correlations which has strong or
moderate and print and save it in
strong_cols = []
for col in auto_df_num.columns :
    prsn_corr, p_val  = stats.pearsonr(auto_df_num[col] ,
pearson_Y_val)
    print(f"pearson correlation for the column {col} is  : \
ncorrelation : {prsn_corr} ,  P Value is : {p_val}")
    if p_val <= 0.0001 :
        print(f"The column {col} has strong correlation ")
        strong_cols.append(col)
    elif 0.0001 < p_val <= 0.05:
        print(f" the column {col} has moderate correlation")
```

```
        strong_cols.append(col)
    else:
        print("No correlation")
    print()

print(strong_cols)
```

```
pearson correlation for the column nrml_loss is  :
correlation : 0.3537559925696345 ,  P Value is : 2.5822806249315043e-
07
The column nrml_loss has strong correlation

pearson correlation for the column nof_doors is  :
correlation : -0.1091798576850524 ,  P Value is : 0.12286630542221234
No correlation

pearson correlation for the column wheel-base is  :
correlation : 0.5618288025498396 ,  P Value is : 4.054012124464529e-18
The column wheel-base has strong correlation

pearson correlation for the column length is  :
correlation : 0.6835418013965626 ,  P Value is : 5.076076468934917e-29
The column length has strong correlation

pearson correlation for the column width is  :
correlation : 0.32961122732907466 ,  P Value is : 1.7695521820954646e-
06
The column width has strong correlation

pearson correlation for the column height is  :
correlation : -0.02646469471658685 ,  P Value is : 0.7092027107308819
No correlation

pearson correlation for the column curb-weight is  :
correlation : 0.8228852445855467 ,  P Value is : 9.605162325238672e-51
The column curb-weight has strong correlation

pearson correlation for the column nof_cylndrs is  :
correlation : -0.013917228892085722 ,  P Value is : 0.8445399677266382
No correlation

pearson correlation for the column engine-size is  :
correlation : 0.8700268715140339 ,  P Value is : 4.8869227245423e-63
The column engine-size has strong correlation

pearson correlation for the column bore is  :
correlation : -0.09592619767283761 ,  P Value is : 0.17553464008104455
No correlation

pearson correlation for the column stroke is  :
```

```
correlation : -0.11197313210815273 ,  P Value is : 0.11351877694621273
No correlation

pearson correlation for the column cmprsn_RT is  :
correlation : -0.06998149184587987 ,  P Value is : 0.32355347520411243
No correlation

pearson correlation for the column HP is  :
correlation : 0.8088198477340562 ,  P Value is : 9.065130138389822e-48
The column HP has strong correlation

pearson correlation for the column peak-rpm is  :
correlation : -0.008252925590469678 ,  P Value is : 0.9074324925941989
No correlation

pearson correlation for the column city_kml is  :
correlation : -0.583182618266924 ,  P Value is : 1.0471566274373468e-
19
The column city_kml has strong correlation

pearson correlation for the column hiwy kml is  :
correlation : -0.6213854514083842 ,  P Value is : 7.405451907278584e-
23
The column hiwy kml has strong correlation

pearson correlation for the column price is  :
correlation : 1.0 ,  P Value is : 0.0
The column price has strong correlation

pearson correlation for the column __109 is  :
correlation : -0.1153715093887452 ,  P Value is : 0.10290662310777886
No correlation

pearson correlation for the column __diesel is  :
correlation : 0.10496376494819956 ,  P Value is : 0.13809126398790697
No correlation

pearson correlation for the column __gas is  :
correlation : -0.07547014528852834 ,  P Value is : 0.2869573137102052
No correlation

['nrml_loss', 'wheel-base', 'length', 'width', 'curb-weight', 'engine-
size', 'HP', 'city_kml', 'hiwy kml', 'price']

-> So the correlated columns are as follows :
```

- 'nrml_loss', 'wheel-base', 'length', 'width', 'curb-weight', 'nof_cylndrs', 'engine-size', 'bore', 'HP', 'city_kml', 'hiwy kml', 'price'

```
initiator.drop(columns={'height','peak-rpm'},inplace=True)
```

```
C:\Users\preda\AppData\Local\Temp\ipykernel_24516\2780529288.py:1: 
SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: 
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  initiator.drop(columns={'height','peak-rpm'},inplace=True)

initiator_cols = initiator.columns
```

Data Normalization or Feature Scaling

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

Normalization technique

```
mms =  MinMaxScaler()

X1 = mms.fit_transform(initiator)

X1 = pd.DataFrame(X1,columns=initiator_cols)

X1.describe()

         nrml_loss  wheel-base      length       width  curb-weight  \
count  201.000000  201.000000  201.000000  201.000000   201.000000   
mean     0.315487    0.356193    0.654618    0.119148     0.615056   
std      0.184681    0.177662    0.132716    0.075900     0.137784   
min      0.000000    0.000000    0.000000    0.000000     0.000000   
25%      0.157068    0.230321    0.578204    0.078029     0.514531   
50%      0.277487    0.303207    0.647830    0.106776     0.582512   
75%      0.445026    0.460641    0.751766    0.135524     0.711903   
max      1.000000    1.000000    1.000000    1.000000     1.000000   

       nof_cylndrs  engine-size        bore          HP    city_kml  \
count   201.000000   201.000000  201.000000  201.000000  201.000000   
mean      0.026968     0.248118    0.012357    0.259334    0.130908   
std       0.069682     0.156841    0.070057    0.174351    0.090850   
min       0.000000     0.000000    0.000000    0.000000    0.000000   
25%       0.018692     0.139623    0.005730    0.102804    0.062500   
50%       0.018692     0.218868    0.007233    0.219626    0.114583   
75%       0.018692     0.301887    0.009863    0.317757    0.177083   
max       1.000000     1.000000    1.000000    1.000000    1.000000   

         hiwy_kml       price  
count  201.000000  201.000000  
mean     0.161665    0.287196  
```

```
std      0.094136    0.176461
min      0.000000    0.000000
25%      0.096774    0.169261
50%      0.150538    0.223797
75%      0.193548    0.361904
max      1.000000    1.000000
```

Standardization technique

```
sS = StandardScaler()

X2 = sS.fit_transform(initiator)

X2 = pd.DataFrame(X2,columns=initiator_cols)

X2.describe()
```

```
          nrml_loss    wheel-base        length         width   curb-
weight  \
count  2.010000e+02  2.010000e+02  2.010000e+02  2.010000e+02
2.010000e+02
mean   3.535038e-17 -3.305261e-15 -2.474527e-16  3.402475e-16
2.474527e-16
std    1.002497e+00  1.002497e+00  1.002497e+00  1.002497e+00
1.002497e+00
min   -1.712543e+00 -2.009892e+00 -4.944772e+00 -1.573711e+00 -
4.475057e+00
25%   -8.599366e-01 -7.102594e-01 -5.772073e-01 -5.431011e-01 -
7.314029e-01
50%   -2.062718e-01 -2.989834e-01 -5.127024e-02 -1.634026e-01 -
2.367845e-01
75%    7.031747e-01  5.893729e-01  7.338243e-01  2.162959e-01
7.046453e-01
max    3.715717e+00  3.632816e+00  2.608904e+00  1.163437e+01
2.800798e+00

        nof_cylndrs   engine-size          bore            HP
city_kml  \
count  2.010000e+02  2.010000e+02  2.010000e+02  2.010000e+02
2.010000e+02
mean   2.651279e-17 -5.081618e-17 -3.535038e-17  1.502391e-16
2.960595e-16
std    1.002497e+00  1.002497e+00  1.002497e+00  1.002497e+00
1.002497e+00
min   -3.879801e-01 -1.585923e+00 -1.768260e-01 -1.491144e+00 -
1.444518e+00
25%   -1.190698e-01 -6.934812e-01 -9.483284e-02 -9.000334e-01 -
7.548551e-01
50%   -1.190698e-01 -1.869603e-01 -7.332643e-02 -2.283168e-01 -
1.801359e-01
```

```
75%   -1.190698e-01  3.436806e-01 -3.569021e-02  3.359251e-01
5.095272e-01
max    1.399872e+01  4.805888e+00  1.413300e+01  4.258750e+00
9.590090e+00

           hiwy kml         price
count  2.010000e+02  2.010000e+02
mean  -6.628197e-16 -2.651279e-17
std    1.002497e+00  1.002497e+00
min   -1.721642e+00 -1.631600e+00
25%   -6.910497e-01 -6.700058e-01
50%   -1.184982e-01 -3.601784e-01
75%    3.395430e-01  4.244249e-01
max    8.927816e+00  4.049530e+00
```

- -> We will move forward with the normalization technique where the min is 0 and the maximum value is 1

```
# From here on we will be working with normalized data instead of
standardized data


# So lets split the data into x and y

x = X1[["nrml_loss", "wheel-base", "length", "width","curb-weight",
"nof_cylndrs",
                       "engine-size", "bore", "HP", "city_kml", "hiwy
kml"]]
y = X1['price']
```

MODEL SELECTION

Machine Learning Algorithms Applications

Linear Regression

```
# importing the MSE library from the metrics
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression , Ridge
from sklearn.model_selection import *
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
```

For Single Linear Regression we will be using two columns:

that are "engine-size" column which is a (independent variable) and "price" column as(dependent variable)

```
A = initiator['engine-size']
B = initiator['price']
print(A.head(5))
print(B.head(5))
```

```
symboling
3    130
3    130
1    152
2    109
2    136
Name: engine-size, dtype: int64
symboling
3    13495
3    16500
1    16500
2    13950
2    17450
Name: price, dtype: int64
```

Here we create an object for the Linear Regression called 'sLr'

```
A = A.values.reshape(-1,1)
B = B.values.reshape(-1,1)
print(A)
print(B)
```

```
[[130]
 [130]
 [152]
 [109]
 [136]
 [136]
 [136]
 [136]
 [131]
 [108]
 [108]
 [164]
 [164]
 [164]
 [209]
 [209]
```

```
[209]
[ 61]
[ 90]
[ 90]
[ 90]
[ 90]
[ 98]
[ 90]
[ 90]
[ 90]
[ 98]
[122]
[156]
[ 92]
[ 92]
[ 79]
[ 92]
[ 92]
[ 92]
[ 92]
[110]
[110]
[110]
[110]
[110]
[110]
[111]
[119]
[258]
[258]
[326]
[ 91]
[ 91]
[ 91]
[ 91]
[ 91]
[ 70]
[ 70]
[ 70]
[ 80]
[122]
[122]
[122]
[122]
[122]
[122]
[140]
[109]
[183]
```

```
[183]
[183]
[183]
[234]
[234]
[308]
[304]
[140]
[ 92]
[ 92]
[ 92]
[ 98]
[110]
[122]
[156]
[156]
[156]
[122]
[122]
[110]
[110]
[ 97]
[103]
[ 97]
[ 97]
[ 97]
[ 97]
[ 97]
[ 97]
[ 97]
[ 97]
[120]
[120]
[181]
[181]
[181]
[181]
[181]
[181]
[120]
[152]
[120]
[152]
[120]
[152]
[120]
[152]
[120]
[152]
```

```
[134]
[ 90]
[ 98]
[ 90]
[ 90]
[ 98]
[122]
[156]
[151]
[194]
[194]
[194]
[132]
[132]
[121]
[121]
[121]
[121]
[121]
[121]
[ 97]
[108]
[108]
[108]
[108]
[108]
[108]
[108]
[108]
[108]
[108]
[108]
[ 92]
[ 92]
[ 92]
[ 92]
[ 92]
[ 92]
[ 98]
[ 98]
[110]
[110]
[ 98]
[ 98]
[ 98]
[ 98]
[ 98]
[ 98]
[ 98]
```

```
 [146]
 [146]
 [146]
 [146]
 [146]
 [146]
 [122]
 [110]
 [122]
 [122]
 [122]
 [171]
 [171]
 [171]
 [161]
 [ 97]
 [109]
 [ 97]
 [109]
 [109]
 [ 97]
 [109]
 [109]
 [109]
 [136]
 [ 97]
 [109]
 [141]
 [141]
 [141]
 [141]
 [130]
 [130]
 [141]
 [141]
 [173]
 [145]
 [141]]
[[13495]
 [16500]
 [16500]
 [13950]
 [17450]
 [15250]
 [17710]
 [18920]
 [23875]
 [16430]
 [16925]
```

```
[20970]
[21105]
[24565]
[30760]
[41315]
[36880]
[ 5151]
[ 6295]
[ 6575]
[ 5572]
[ 6377]
[ 7957]
[ 6229]
[ 6692]
[ 7609]
[ 8558]
[ 8921]
[12964]
[ 6479]
[ 6855]
[ 5399]
[ 6529]
[ 7129]
[ 7295]
[ 7295]
[ 7895]
[ 9095]
[ 8845]
[10295]
[12945]
[10345]
[ 6785]
[11048]
[32250]
[35550]
[36000]
[ 5195]
[ 6095]
[ 6795]
[ 6695]
[ 7395]
[10945]
[11845]
[13645]
[15645]
[ 8845]
[ 8495]
[10595]
[10245]
```

```
 [10795]
 [11245]
 [18280]
 [  109]
 [25552]
 [28248]
 [28176]
 [31600]
 [34184]
 [35056]
 [40960]
 [45400]
 [16503]
 [ 5389]
 [ 6189]
 [ 6669]
 [ 7689]
 [ 9959]
 [ 8499]
 [12629]
 [14869]
 [14489]
 [ 6989]
 [ 8189]
 [ 9279]
 [ 9279]
 [ 5499]
 [ 7099]
 [ 6649]
 [ 6849]
 [ 7349]
 [ 7299]
 [ 7799]
 [ 7499]
 [ 7999]
 [ 8249]
 [ 8949]
 [ 9549]
 [13499]
 [14399]
 [13499]
 [17199]
 [19699]
 [18399]
 [11900]
 [13200]
 [12440]
 [13860]
 [15580]
```

```
[16900]
[16695]
[17075]
[16630]
[17950]
[18150]
[ 5572]
[ 7957]
[ 6229]
[ 6692]
[ 7609]
[ 8921]
[12764]
[22018]
[32528]
[34028]
[37028]
[ 9295]
[ 9895]
[11850]
[12170]
[15040]
[15510]
[18150]
[18620]
[ 5118]
[ 7053]
[ 7603]
[ 7126]
[ 7775]
[ 9960]
[ 9233]
[11259]
[ 7463]
[10198]
[ 8013]
[11694]
[ 5348]
[ 6338]
[ 6488]
[ 6918]
[ 7898]
[ 8778]
[ 6938]
[ 7198]
[ 7898]
[ 7788]
[ 7738]
[ 8358]
```

```
 [ 9258]
 [ 8058]
 [ 8238]
 [ 9298]
 [ 9538]
 [ 8449]
 [ 9639]
 [ 9989]
 [11199]
 [11549]
 [17669]
 [ 8948]
 [10698]
 [ 9988]
 [10898]
 [11248]
 [16558]
 [15998]
 [15690]
 [15750]
 [ 7775]
 [ 7975]
 [ 7995]
 [ 8195]
 [ 8495]
 [ 9495]
 [ 9995]
 [11595]
 [ 9980]
 [13295]
 [13845]
 [12290]
 [12940]
 [13415]
 [15985]
 [16515]
 [18420]
 [18950]
 [16845]
 [19045]
 [21485]
 [22470]
 [22625]]
```

Normalized Single Linear Regression values

```
# here we define and assign object of a linear regression to a
variable named sLr
```

```
sLr = LinearRegression()

# we will try to normalize the data here and see what actually happens
and lets see if we can denormalize it later
a = MinMaxScaler()
b = MinMaxScaler()

# normalizing the value
a_norml = a.fit_transform(A)
b_norml = b.fit_transform(B)

# then we fit the model first that is A,y

sLr.fit(a_norml,b_norml)

LinearRegression()

# lets the check the score of the data here and see how much accuracy
we will get

sLr.score(a_norml,b_norml)

0.7569467571564972
```

As you can see that we have got the accuracy of 76 % and we will now move forward with the data and see what happens now

```
predicted_val = sLr.predict(b_norml)

predicted_val

array([[0.33363112],
       [0.39857743],
       [0.39857743],
       [0.34346492],
       [0.41910954],
       [0.37156149],
       [0.42472885],
       [0.45088027],
       [0.55797143],
       [0.39706453],
       [0.40776284],
       [0.4951864 ],
       [0.49810412],
       [0.57288423],
       [0.70677519],
       [0.93489773],
       [0.8390452 ],
       [0.15329437],
       [0.17801935],
```

```
       [0.18407092],
       [0.16239333],
       [0.17979159],
       [0.21393973],
       [0.17659291],
       [0.18659961],
       [0.2064185 ],
       [0.22692899],
       [0.23477442],
       [0.32215476],
       [0.18199609],
       [0.19012249],
       [0.15865433],
       [0.18307673],
       [0.19604438],
       [0.1996321 ],
       [0.1996321 ],
       [0.21259974],
       [0.23853504],
       [0.23313185],
       [0.26447033],
       [0.32174411],
       [0.26555097],
       [0.18860959],
       [0.28074473],
       [0.73897818],
       [0.81030025],
       [0.82002598],
       [0.15424533],
       [0.1736968 ],
       [0.18882572],
       [0.18666445],
       [0.20179337],
       [0.27851862],
       [0.29797009],
       [0.33687304],
       [0.38009853],
       [0.23313185],
       [0.22556739],
       [0.27095416],
       [0.2633897 ],
       [0.27527671],
       [0.28500244],
       [0.43704812],
       [0.0443229 ],
       [0.59421601],
       [0.65248397],
       [0.65092785],
       [0.7249299 ],
```

```
        [0.78077724],
        [0.79962355],
        [0.92722521],
        [1.0231858 ],
        [0.39864226],
        [0.1584382 ],
        [0.1757284 ],
        [0.18610252],
        [0.20814752],
        [0.25720845],
        [0.22565384],
        [0.31491448],
        [0.36332704],
        [0.35511419],
        [0.19301859],
        [0.21895389],
        [0.24251178],
        [0.24251178],
        [0.1608156 ],
        [0.195396  ],
        [0.18567026],
        [0.18999281],
        [0.20079918],
        [0.19971855],
        [0.21052492],
        [0.2040411 ],
        [0.21484747],
        [0.22025066],
        [0.23537958],
        [0.24834723],
        [0.33371757],
        [0.35316905],
        [0.33371757],
        [0.41368474],
        [0.4677166 ],
        [0.43962003],
        [0.29915879],
        [0.32725536],
        [0.31082968],
        [0.34151978],
        [0.3786937 ],
        [0.40722253],
        [0.40279191],
        [0.41100476],
        [0.40138708],
        [0.42991591],
        [0.43423846],
        [0.16239333],
        [0.21393973],
```

```
       [0.17659291],
       [0.18659961],
       [0.2064185 ],
       [0.23477442],
       [0.31783221],
       [0.51783656],
       [0.74498653],
       [0.77740565],
       [0.84224389],
       [0.24285759],
       [0.25582524],
       [0.29807816],
       [0.30499423],
       [0.36702282],
       [0.37718081],
       [0.43423846],
       [0.44439645],
       [0.15258115],
       [0.19440181],
       [0.20628882],
       [0.19597954],
       [0.21000621],
       [0.25723006],
       [0.2415176 ],
       [0.28530502],
       [0.20326304],
       [0.2623739 ],
       [0.21515005],
       [0.29470657],
       [0.15755208],
       [0.1789487 ],
       [0.18219061],
       [0.19148409],
       [0.21266458],
       [0.2316838 ],
       [0.19191634],
       [0.19753566],
       [0.21266458],
       [0.21028718],
       [0.20920654],
       [0.22260644],
       [0.24205792],
       [0.21612262],
       [0.22001292],
       [0.24292243],
       [0.24810949],
       [0.2245732 ],
       [0.25029237],
       [0.25785683],
```

```
       [0.28400826],
       [0.29157272],
       [0.42384273],
       [0.23535797],
       [0.27318027],
       [0.25783522],
       [0.27750282],
       [0.28506728],
       [0.39983097],
       [0.38772783],
       [0.3810711 ],
       [0.38236787],
       [0.21000621],
       [0.21432876],
       [0.21476102],
       [0.21908357],
       [0.22556739],
       [0.24718014],
       [0.25798651],
       [0.29256691],
       [0.25766232],
       [0.32930857],
       [0.34119558],
       [0.30758776],
       [0.32163605],
       [0.3319021 ],
       [0.38744686],
       [0.39890162],
       [0.4400739 ],
       [0.45152866],
       [0.40603382],
       [0.45358187],
       [0.50631697],
       [0.52760552],
       [0.5309555 ]])
```

```
new_pred_price = b.inverse_transform(predicted_val)
new_pred_price
```

```
array([[15219.48721124],
       [18160.9702275 ],
       [18160.9702275 ],
       [15664.86983101],
       [19090.88998306],
       [16937.39160177],
       [19345.39433721],
       [20529.81844692],
       [25380.08411932],
       [18092.44982446],
       [18576.98696025],
```

```
       [22536.48739312],
       [22668.6338847 ],
       [26055.49952073],
       [32119.55518986],
       [42451.45310555],
       [38110.19614145],
       [ 7051.85516875],
       [ 8171.67432702],
       [ 8445.75593918],
       [ 7463.9564499 ],
       [ 8251.94108487],
       [ 9798.54446779],
       [ 8107.06937558],
       [ 8560.28289855],
       [ 9457.90017839],
       [10386.84107105],
       [10742.16830396],
       [14699.71101103],
       [ 8351.78510073],
       [ 8719.83755135],
       [ 7294.61316809],
       [ 8400.72824576],
       [ 8988.04598611],
       [ 9150.53722761],
       [ 9150.53722761],
       [ 9737.85496796],
       [10912.49044866],
       [10667.77472351],
       [12087.12592936],
       [14681.11261592],
       [12136.06907439],
       [ 8651.31714831],
       [12824.20969351],
       [33578.06091173],
       [36808.30848367],
       [37248.79678893],
       [ 7094.92513637],
       [ 7975.9017469 ],
       [ 8661.10577731],
       [ 8563.21948725],
       [ 9248.42351766],
       [12723.38681474],
       [13604.36342527],
       [15366.31664633],
       [17324.0424475 ],
       [10667.77472351],
       [10325.17270831],
       [12380.78479954],
       [12038.18278433],
```

```
       [12576.55737966],
       [13017.04568492],
       [19903.34619054],
       [ 2116.42842399],
       [27021.63720361],
       [29660.65158359],
       [29590.17345475],
       [32941.80002635],
       [35471.18176147],
       [36324.75021078],
       [42103.95677584],
       [46450.10805444],
       [18163.9068162 ],
       [ 7284.82453909],
       [ 8067.91485956],
       [ 8537.76905184],
       [ 9536.20921044],
       [11758.22799477],
       [10329.08815991],
       [14371.79193933],
       [16564.44483665],
       [16192.47693442],
       [ 8851.00518003],
       [10025.64066073],
       [11092.60122237],
       [11092.60122237],
       [ 7392.49945815],
       [ 8958.68009909],
       [ 8518.19179383],
       [ 8713.96437394],
       [ 9203.39582424],
       [ 9154.45267921],
       [ 9643.8841295 ],
       [ 9350.22525932],
       [ 9839.65670962],
       [10084.37243476],
       [10769.57646517],
       [11356.89420553],
       [15223.40266284],
       [16104.37927337],
       [15223.40266284],
       [18845.19539501],
       [21292.35264648],
       [20019.83087571],
       [13658.2008848 ],
       [14930.72265557],
       [14186.78685112],
       [15576.77216995],
       [17260.41635896],
```

```
       [18552.51538774],
       [18351.84849312],
       [18723.81639534],
       [18288.22240458],
       [19580.32143335],
       [19776.09401347],
       [ 7463.9564499 ],
       [ 9798.54446779],
       [ 8107.06937558],
       [ 8560.28289855],
       [ 9457.90017839],
       [10742.16830396],
       [14503.93843091],
       [23562.33571294],
       [33850.1847981 ],
       [35318.47914898],
       [38255.06785074],
       [11108.26302878],
       [11695.58076913],
       [13609.25773978],
       [13922.49386796],
       [16731.83039265],
       [17191.89595592],
       [19776.09401347],
       [20236.15957674],
       [ 7019.55269303],
       [ 8913.65240566],
       [ 9452.02700099],
       [ 8985.10939741],
       [ 9620.39141989],
       [11759.20685767],
       [11047.57352894],
       [13030.74976553],
       [ 9314.9861949 ],
       [11992.17622801],
       [ 9853.36079023],
       [13456.55512728],
       [ 7244.69116016],
       [ 8213.76543174],
       [ 8360.59486683],
       [ 8781.50591408],
       [ 9740.79155666],
       [10602.19090917],
       [ 8801.0831721 ],
       [ 9055.58752625],
       [ 9740.79155666],
       [ 9633.11663759],
       [ 9584.17349256],
       [10191.06849093],
```

```
       [11072.04510146],
       [ 9897.40962075],
       [10073.60494286],
       [11111.19961748],
       [11346.12671362],
       [10280.14501488],
       [11444.99186658],
       [11787.59388178],
       [12972.01799149],
       [13314.6200067 ],
       [19305.26095829],
       [10768.59760227],
       [12481.6076783 ],
       [11786.61501888],
       [12677.38025842],
       [13019.98227362],
       [18217.74427574],
       [17669.58105141],
       [17368.09127803],
       [17426.82305206],
       [ 9620.39141989],
       [ 9816.164     ],
       [ 9835.74125802],
       [10031.51383813],
       [10325.17270831],
       [11304.03560889],
       [11793.46705919],
       [13359.64770013],
       [11778.78411568],
       [15023.71463112],
       [15562.08922644],
       [14039.95741603],
       [14676.21830141],
       [15141.17817919],
       [17656.8558337 ],
       [18175.65317101],
       [20040.38699663],
       [20559.18433394],
       [18498.6779282 ],
       [20652.17630949],
       [23040.60178692],
       [24004.781744  ],
       [24156.50549359]])
```

Now we will work to obtain the intercept and slope of the linear regression

```
# Intercept and Slope

intercept = np.array(sLr.intercept_).reshape(-1,1)
```

```
print(f"Normalized Intercept Value : {intercept}")

slope = np.array(sLr.coef_).reshape(-1,1)
print(f"Normalized Slope Value : {slope}")

Normalized Intercept Value : [[0.0443229]]
Normalized Slope Value : [[0.9788629]]
```

Denormalized Single Linear Regression

```
denorm_intercept_val = b.inverse_transform(intercept)
print("Denormalized value for intercept : ",denorm_intercept_val)

denorm_slope_val = b.inverse_transform(slope)
print("Denormalized value for slope : ",denorm_slope_val)

Denormalized value for intercept :  [[2116.42842399]]
Denormalized value for slope :  [[44442.67963045]]

    =>  yhat = intercept + slope * interdependent_value
```

Multiple Linear Regression

Normalized values

```
x.head(5)

   nrml_loss  wheel-base    length     width  curb-weight  nof_cylndrs  \
0   0.277487    0.058309  0.603431  0.078029     0.616376     0.018692

1   0.277487    0.058309  0.603431  0.078029     0.616376     0.018692

2   0.277487    0.230321  0.627649  0.106776     0.685873     0.037383

3   0.518325    0.384840  0.682139  0.121150     0.563053     0.018692

4   0.518325    0.373178  0.682139  0.125257     0.686126     0.028037


   engine-size      bore        HP  city_kml  hiwy_kml
0     0.260377  0.008736  0.294393  0.083333  0.118280
1     0.260377  0.008736  0.294393  0.083333  0.118280
2     0.343396  0.001315  0.495327  0.062500  0.107527
3     0.181132  0.006106  0.252336  0.114583  0.150538
4     0.283019  0.006106  0.313084  0.052083  0.064516

y[:5]
```

```
0    0.295555
1    0.361904
2    0.361904
3    0.305602
4    0.382880
Name: price, dtype: float64
```

```
low_Lr = LinearRegression()
```

```
low_Lr.fit(x,y)
```

```
LinearRegression()
```

```
x_nrml_mMs = MinMaxScaler()
y_nrml_mMs = MinMaxScaler()
```

Denormalized Values for 'x' & 'y'

```
print(x.shape)
# denorm_y = mms.inverse_transform(y)
print(y.shape)
```

```
(201, 11)
(201,)
```

```
x.ndim
```

```
2
```

```
y
```

```
0      0.295555
1      0.361904
2      0.361904
3      0.305602
4      0.382880
         ...
196    0.369522
197    0.418096
198    0.471970
199    0.493718
200    0.497141
Name: price, Length: 201, dtype: float64
```

```
y_trial = np.array(y)
```

```
x
```

```
     nrml_loss  wheel-base    length     width  curb-weight
nof_cylndrs  \
```

```
0     0.277487    0.058309  0.603431  0.078029     0.616376
0.018692
1     0.277487    0.058309  0.603431  0.078029     0.616376
0.018692
2     0.277487    0.230321  0.627649  0.106776     0.685873
0.037383
3     0.518325    0.384840  0.682139  0.121150     0.563053
0.018692
4     0.518325    0.373178  0.682139  0.125257     0.686126
0.028037
..         ...         ...       ...       ...          ...          .
..
196   0.157068    0.655977  0.805247  0.176591     0.718474
0.018692
197   0.157068    0.655977  0.805247  0.174538     0.742987
0.018692
198   0.157068    0.655977  0.805247  0.176591     0.733637
0.037383
199   0.157068    0.655977  0.805247  0.176591     0.785444
0.037383
200   0.157068    0.655977  0.805247  0.176591     0.746272
0.018692

     engine-size      bore        HP  city_kml  hiwy kml
0       0.260377  0.008736  0.294393  0.083333  0.118280
1       0.260377  0.008736  0.294393  0.083333  0.118280
2       0.343396  0.001315  0.495327  0.062500  0.107527
3       0.181132  0.006106  0.252336  0.114583  0.150538
4       0.283019  0.006106  0.313084  0.052083  0.064516
..           ...       ...       ...       ...       ...
196     0.301887  0.011648  0.308411  0.104167  0.129032
197     0.301887  0.011648  0.523364  0.062500  0.096774
198     0.422642  0.009769  0.401869  0.052083  0.075269
199     0.316981  0.004415  0.271028  0.135417  0.118280
200     0.301887  0.011648  0.308411  0.062500  0.096774

[201 rows x 11 columns]

y_trial

array([0.29555541, 0.36190413, 0.36190413, 0.30560155, 0.3828796 ,
       0.33430483, 0.38862026, 0.41533638, 0.52474001, 0.36035857,
       0.37128789, 0.46059924, 0.46357996, 0.53997483, 0.67675697,
       0.90980548, 0.81188316, 0.11132455, 0.13658343, 0.14276567,
       0.12061999, 0.13839394, 0.17327946, 0.13512618, 0.14534897,
       0.16559581, 0.1865492 , 0.19456404, 0.28383122, 0.14064604,
       0.14894791, 0.11680025, 0.14175002, 0.15499768, 0.15866287,
       0.15866287, 0.17191053, 0.19840586, 0.192886  , 0.22490119,
       0.28341172, 0.22600517, 0.14740235, 0.24152701, 0.70965534,
       0.7825175 , 0.79245325, 0.11229604, 0.13216754, 0.14762315,
```

```
       0.1454152 , 0.16087081, 0.23925283, 0.25912433, 0.29886732,
       0.34302621, 0.192886  , 0.1851582 , 0.23152503, 0.22379722,
       0.23594092, 0.24587666, 0.40120554, 0.        , 0.56176724,
       0.62129341, 0.61970369, 0.6953037 , 0.75235698, 0.77161025,
       0.90196728, 1.        , 0.36197037, 0.11657945, 0.13424301,
       0.14484114, 0.16736217, 0.2174825 , 0.18524652, 0.27643461,
       0.32589256, 0.31750237, 0.15190656, 0.17840189, 0.20246848,
       0.20246848, 0.11900819, 0.1543353 , 0.14439955, 0.14881544,
       0.15985516, 0.15875119, 0.16979091, 0.16316708, 0.1742068 ,
       0.17972666, 0.19518227, 0.20842993, 0.29564373, 0.31551522,
       0.29564373, 0.37733766, 0.43253627, 0.40383299, 0.2603387 ,
       0.28904197, 0.2722616 , 0.3036144 , 0.34159104, 0.37073591,
       0.36620962, 0.37459981, 0.36477446, 0.39391932, 0.39833521,
       0.12061999, 0.17327946, 0.13512618, 0.14534897, 0.16559581,
       0.19456404, 0.27941534, 0.48373849, 0.71579342, 0.74891259,
       0.81515091, 0.20282175, 0.21606942, 0.25923473, 0.26630015,
       0.32966815, 0.34004548, 0.39833521, 0.40871255, 0.11059592,
       0.15331964, 0.16546334, 0.15493144, 0.169261  , 0.21750458,
       0.20145283, 0.24618578, 0.16237222, 0.22275949, 0.17451591,
       0.25579033, 0.1156742 , 0.13753284, 0.14084476, 0.15033892,
       0.17197677, 0.19140668, 0.15078051, 0.15652116, 0.17197677,
       0.16954803, 0.16844406, 0.18213332, 0.20200481, 0.17550948,
       0.17948378, 0.20288799, 0.20818706, 0.18414254, 0.21041708,
       0.21814489, 0.24486101, 0.25258881, 0.387715  , 0.19516019,
       0.23379921, 0.21812281, 0.2382151 , 0.2459429 , 0.36318474,
       0.35082025, 0.34401978, 0.34534455, 0.169261  , 0.17367689,
       0.17411848, 0.17853437, 0.1851582 , 0.20723764, 0.21827736,
       0.25360447, 0.21794617, 0.29113952, 0.30328321, 0.26894968,
       0.28330132, 0.29378905, 0.35053322, 0.36223532, 0.40429666,
       0.41599876, 0.36952154, 0.41809631, 0.47197015, 0.4937184 ,
       0.49714071])
```

```python
x['price'] = y
```

```python
x.head(3)
```

```
   nrml_loss  wheel-base    length     width  curb-weight  nof_cylndrs  \
0   0.277487    0.058309  0.603431  0.078029     0.616376     0.018692   
1   0.277487    0.058309  0.603431  0.078029     0.616376     0.018692   
2   0.277487    0.230321  0.627649  0.106776     0.685873     0.037383   

   engine-size      bore        HP  city_kml  hiwy_kml     price  
0     0.260377  0.008736  0.294393  0.083333  0.118280  0.295555  
1     0.260377  0.008736  0.294393  0.083333  0.118280  0.361904  
2     0.343396  0.001315  0.495327  0.062500  0.107527  0.361904  
```

```python
x.shape
```

```
(201, 12)

denormalized_x_y = mms.inverse_transform(x)

columns_denorm = ["nrml_loss", "wheel-base", "length", "width","curb-weight", "nof_cylndrs",
                       "engine-size", "bore", "HP", "city_kml", "hiwy kml","price"]

denorm_x_n_y = pd.DataFrame(denormalized_x_y, columns = columns_denorm)

denorm_x_n_y.head(5)

   nrml_loss  wheel-base  length  width  curb-weight  nof_cylndrs  \
0      118.0        88.6   168.8   64.1       2548.0          4.0
1      118.0        88.6   168.8   64.1       2548.0          4.0
2      118.0        94.5   171.2   65.5       2823.0          6.0
3      164.0        99.8   176.6   66.2       2337.0          4.0
4      164.0        99.4   176.6   66.4       2824.0          5.0

   engine-size  bore     HP   city_kml   hiwy kml    price
0        130.0  3.47  111.0   8.928024  11.478888  13495.0
1        130.0  3.47  111.0   8.928024  11.478888  16500.0
2        152.0  2.68  154.0   8.077736  11.053744  16500.0
3        109.0  3.19  102.0  10.203456  12.754320  13950.0
4        136.0  3.19  115.0   7.652592   9.353168  17450.0
```

Proceeding with 'initiator' dataset

Unfortunately we will not be working with feature scaled values Hence we will work with the initiator values for the multiple linear regression

```
df_x = initiator.iloc[:,:11]
df_y = initiator['price']

df_x.head(5)

           nrml_loss  wheel-base  length  width  curb-weight
nof_cylndrs  \
symboling

3              118.0        88.6   168.8   64.1         2548
4
3              118.0        88.6   168.8   64.1         2548
4
1              118.0        94.5   171.2   65.5         2823
6
2              164.0        99.8   176.6   66.2         2337
4
```

```
2                164.0         99.4   176.6   66.4          2824
5

           engine-size  bore     HP   city_kml   hiwy kml
symboling
3                  130  3.47  111.0   8.928024  11.478888
3                  130  3.47  111.0   8.928024  11.478888
1                  152  2.68  154.0   8.077736  11.053744
2                  109  3.19  102.0  10.203456  12.754320
2                  136  3.19  115.0   7.652592   9.353168
```

```
df_y.head(5)
```

```
symboling
3    13495
3    16500
1    16500
2    13950
2    17450
Name: price, dtype: int64
```

Creating an object of Linear Regression named as mLr

```
mLr = LinearRegression()
```

Fitting the data into the model

```
mLr.fit(df_x , df_y)
```

```
LinearRegression()
```

```
yhat_multi_predict = mLr.predict(df_x)
```

```
yhat_multi_predict
```

```
array([11625.92072994, 11625.92072994, 18542.92492051, 12085.61392929,
       16024.39897627, 14888.82950945, 18818.15941615, 19127.96014257,
       20559.97010942, 11681.63997946, 11681.63997946, 18120.81037367,
       18275.71073688, 19814.36890315, 26866.42932701, 27660.33363114,
       30254.26186079,   108.82877243,  5756.01679089,  5395.56723706,
        5885.15417615,  5970.33877651,  8872.40278163,  6764.02091623,
        6825.98106151,  6825.98106151,  9587.22655462, 11186.99027953,
       18338.80591976,  5856.62067852,  6811.95202695,  4793.45446859,
        6693.13688237,  6738.19880622,  6618.57369659,  6352.54270887,
        9555.3801867 ,  9704.64780943,  8949.83899834,  7447.98476922,
        9991.17565766, 10892.5864184 ,  6877.74834799, 11402.54141082,
       33472.3794226 , 33472.3794226 , 43840.64026229,  5321.54255815,
        5923.12785979,  5937.209711  ,  5800.85778482,  5789.12737823,
        8138.25308624,  8138.25308624,  8152.33493744, 11210.73967563,
       11480.87398327, 11300.50009694, 11480.87398327, 11300.50009694,
```

```
       10447.04346676, 11342.74565054, 15737.21363548,   109.49630098,
       23131.24561975, 23793.09262619, 22924.91273317, 24666.07488419,
       31223.27126398, 29597.23172085, 38683.18043999, 37642.6178776 ,
       20080.97359893,  7294.38680921,  7452.79703581,  7621.77925022,
        9620.30101109, 12266.53637117, 11375.06286219, 18267.7863154 ,
       18510.46444498, 18524.54629618, 11009.66781953, 11122.32262914,
       11889.87584756, 12104.8328267 ,  6219.59874026,  6635.21191094,
        6301.27347722,  6250.12239246,  5890.92974725,  6394.21369515,
        6595.09143208,  6343.06261038,  5927.54256038,  7485.97810768,
       10910.43509862, 10848.47495334, 21496.52668922, 21549.85535047,
       21037.6029632 , 23774.11550799, 26201.53795319, 25005.59825481,
       15731.99062954, 18580.78903796, 16305.64549541, 18674.1936869 ,
       15771.99512624, 18735.68940117, 16345.64999211, 18829.0940501 ,
       15886.89099275, 18735.68940117, 19566.87876395,  6021.35480783,
        8890.31586322,  6871.4994058 ,  6400.70049895,  7547.12211344,
       10536.95689054, 17897.94284152, 19943.09231804, 24437.40132291,
       24437.40132291, 24561.32161347, 12767.62682503, 12745.59283226,
       13466.59667923, 12746.80062473, 14120.84397683, 12924.23194986,
       16699.44081667, 15985.27750265,  5650.27865779,  6788.43775748,
        7243.66373878,  8631.18535328,  8633.90589146,  9774.13149333,
        8677.40094664, 11057.04900344,  8492.28676218,  9734.1568151 ,
        8797.1860261 , 10568.52367714,  5333.8886125 ,  5635.98163738,
        5332.70232063,  5456.34179362,  5341.73877168,  7830.29318458,
        6709.38798754,  6788.24635426,  6948.4668572 ,  7705.30347651,
        7174.8389021 ,  6672.44817797,  6723.1428423 ,  7625.57870708,
        7724.15166549, 10009.13324087, 10107.70619928, 14877.75770665,
       14866.49222569, 14908.73777929, 15269.23317004, 15367.80612844,
       16102.87876113, 11150.21767028,  9497.00369716, 11336.05019011,
       11336.05019011, 11459.97048068, 23185.30495567, 23357.35866271,
       20596.76327083, 20292.31950872,  7718.90119661,  9750.73368969,
        7225.78402266,  9257.61651574,  9435.04784087,  7938.23960555,
       10245.76913694,  8953.65166861, 11296.23025314, 14367.72203891,
        9282.7511623 , 10847.8665722 , 15614.53821119, 15438.65601423,
       15619.91582927, 15401.78807871, 17849.10840463, 17645.06250526,
       17131.74601962, 19951.20835679, 21013.59728253, 18141.78943139,
       17407.93355152])
```

```
mLr_intercept_val = mLr.intercept_
mLr_intercept_val
```

```
-53538.17329990071
```

```
mLr_slope = mLr.coef_.round(3)
mLr_slope
```

```
array([  17.913,  100.347,  -53.276,  627.173,    2.816,  152.811,
         72.189, -516.245,   57.448, -139.715,  212.641])
```

Model Developed

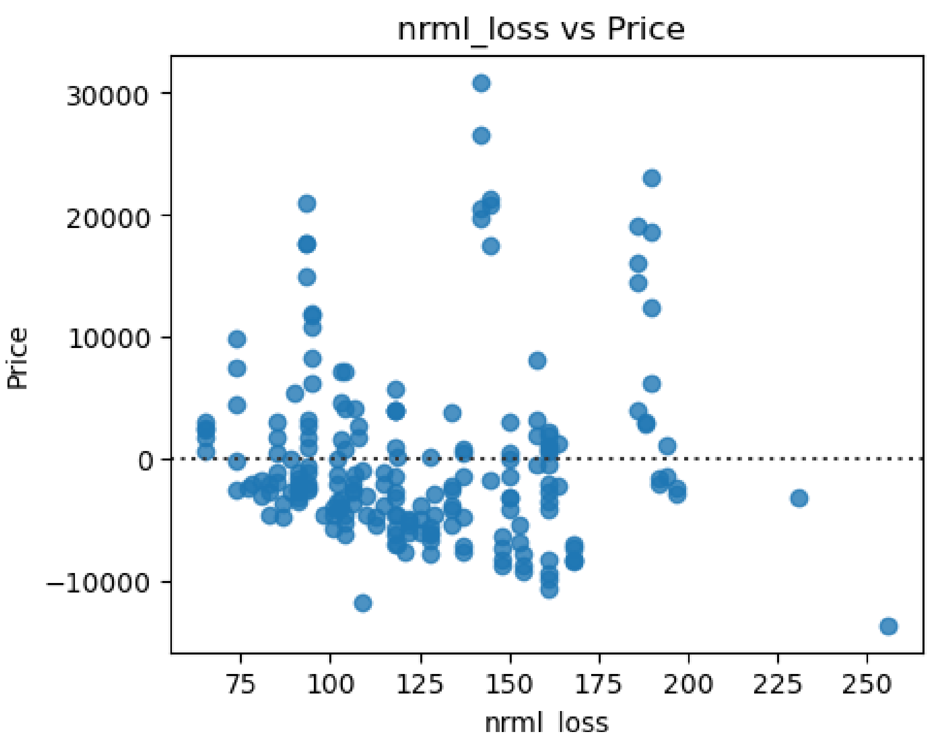
```
def lnr_model_predict_val():
    print(f"Input the following for new price prediction :-")

    nrml_loss_val = float(input("Normalized Loss Value : "))
    whl_base = float(input("Wheel Base : "))
    length = float(input("Length : "))
    width = float(input("Width : "))
    cb_wght = float(input("Curb Weight : "))
    cyln = float(input("No Of Cylinders : "))
    engn_sz = float(input("Engine Size : "))
    br = float(input("Bore : "))
    hp = float(input("HorsePower : "))
    city = float(input("City Km per Liter : "))
    hgwy = float(input("Highway Km per Liter"))

    predicted_price_value = mLr_intercept_val +
((mLr_slope[0]*nrml_loss_val) + (mLr_slope[1]*whl_base) +
                    (mLr_slope[2] * length) + (mLr_slope[3] * width) *
(mLr_slope[4] * cb_wght) +
                    (mLr_slope[5] * cyln) + (mLr_slope[6] * engn_sz) +
(mLr_slope[7] * br) +
                    (mLr_slope[8] * hp) + (mLr_slope[9] * city) +
(mLr_slope[10] * hgwy) )

    return "Predicted Price : ",round(predicted_price_value,3)

# eq_model_ = intercept + (b1x1) + b2x2  ...

# nrml_loss      wheel-base length     width curb-weight
# nof_cylndrs    engine-size      bore  HP    city_kml   hiwy kml

mLr_slope = list(mLr_slope)

# A loop to access and predict the price for the new values

for i in mLr_slope:
    print(i)
```

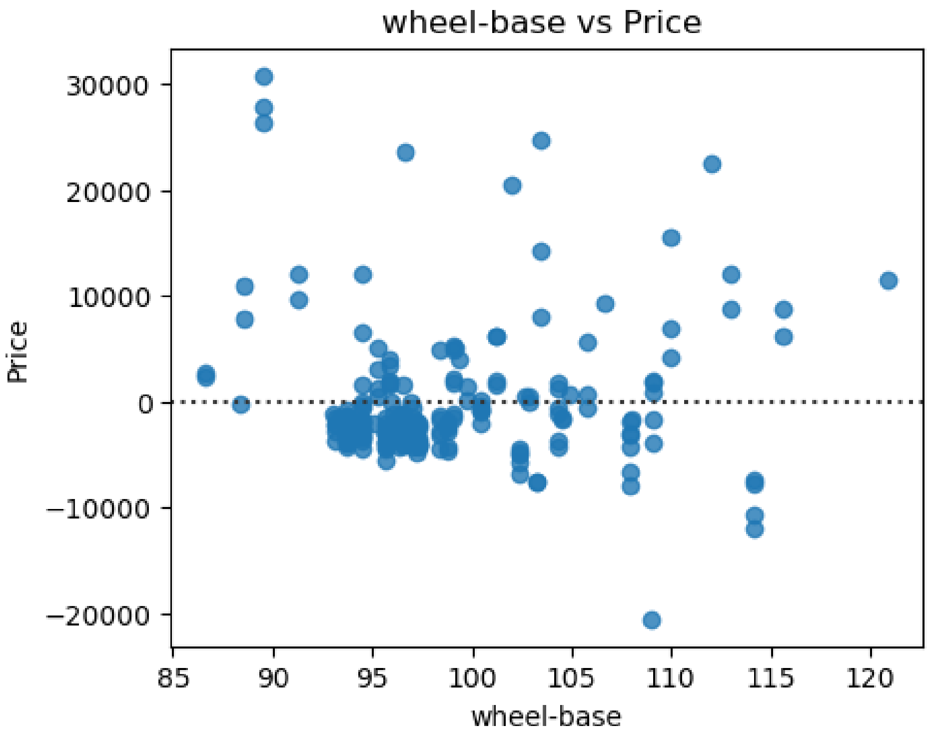
```
17.913
100.347
-53.276
627.173
2.816
152.811
72.189
-516.245
57.448
-139.715
212.641
```

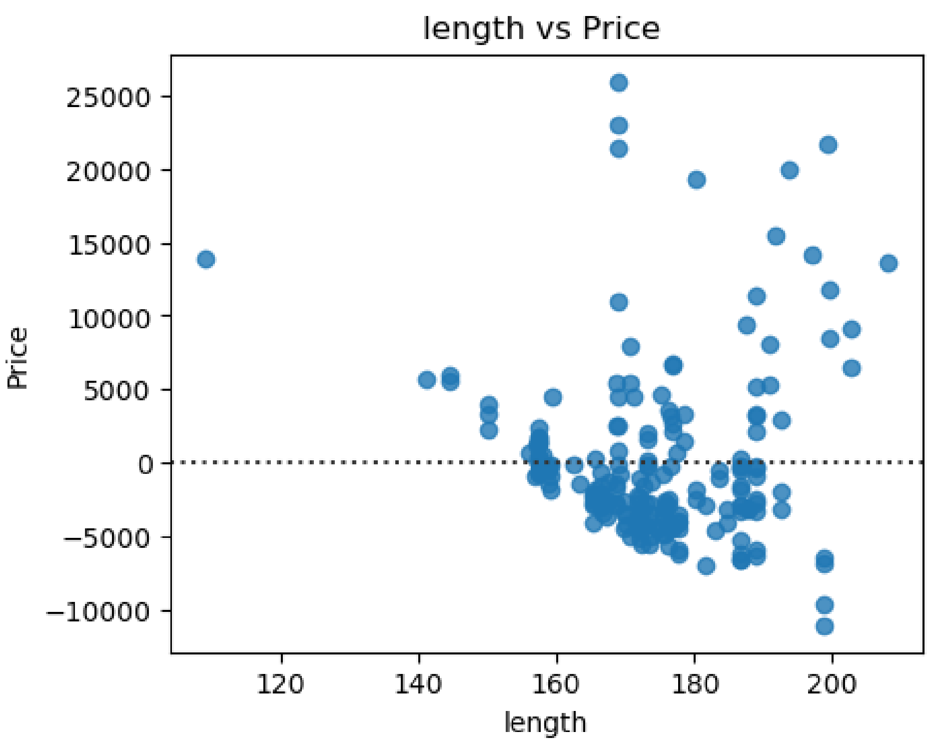
```
# lnr_model_predict_val()
```

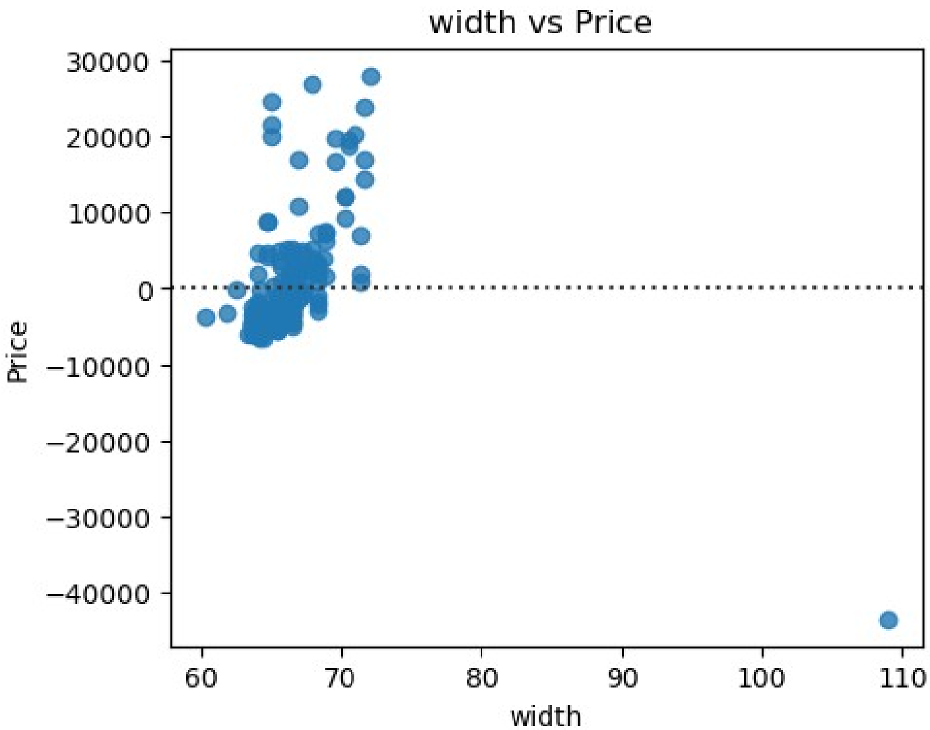
Residual Plot

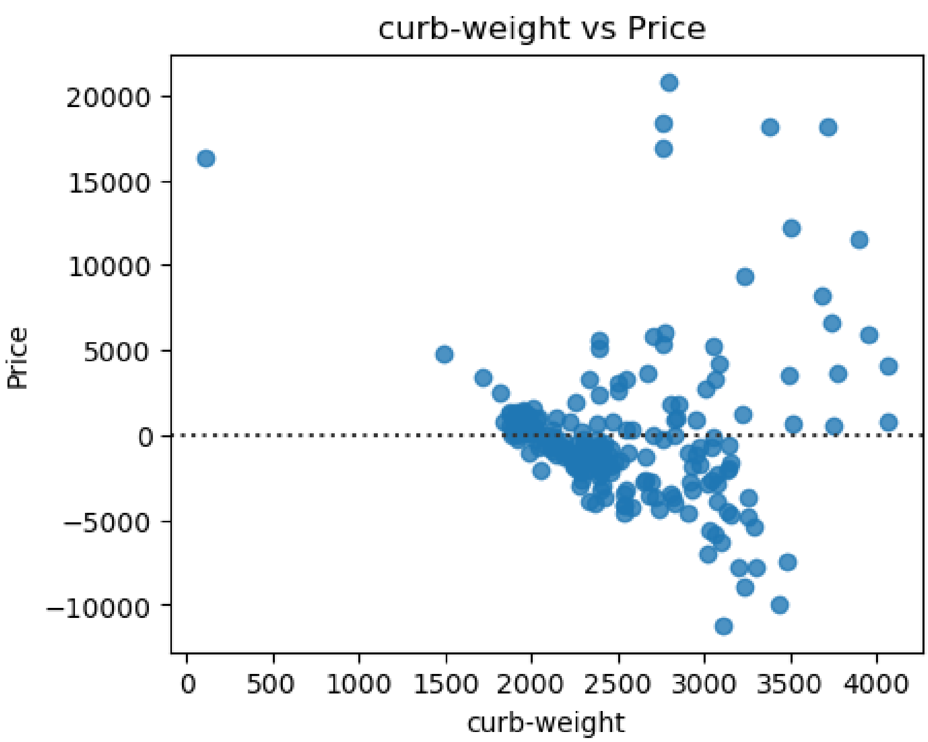
```
for i in initiator_cols:
    plt.figure(figsize=(5,4))

    # assigning the x and y axis with data
    init_y = initiator['price']
    init_x = initiator[i]


    # making a residual plot
    sns.residplot(x=init_x , y=init_y , data = initiator)

    # labelling the plot
    plt.xlabel(f"{i}")
    plt.ylabel(f"Price")
    plt.title(f"{i} vs Price ")
    plt.show()
```

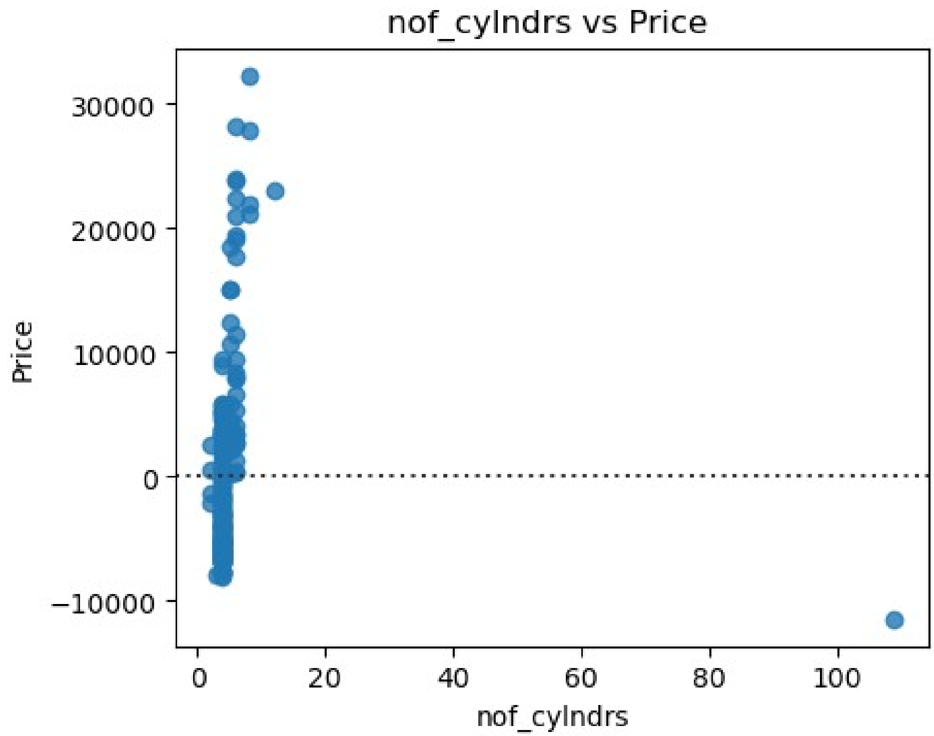


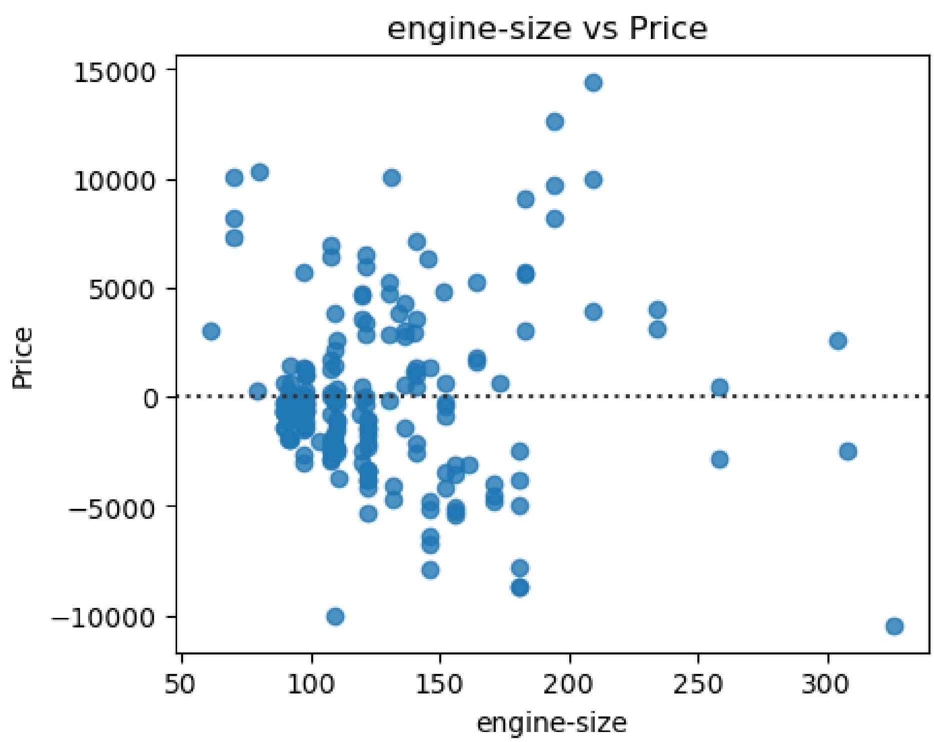

wheel-base vs Price
Price
30000
20000
10000
0
−10000
−20000
85
90
95
100
105
110
115
120
wheel-base

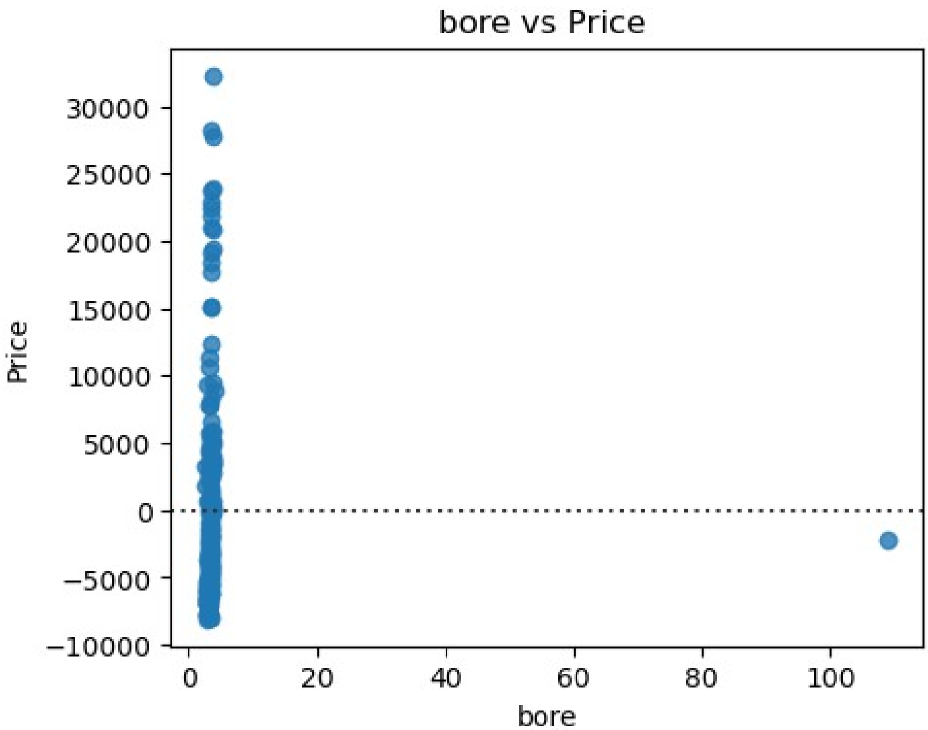

length vs Price
Price
25000
20000
15000
10000
5000
0
−5000
−10000
120
140
160
180
200
length

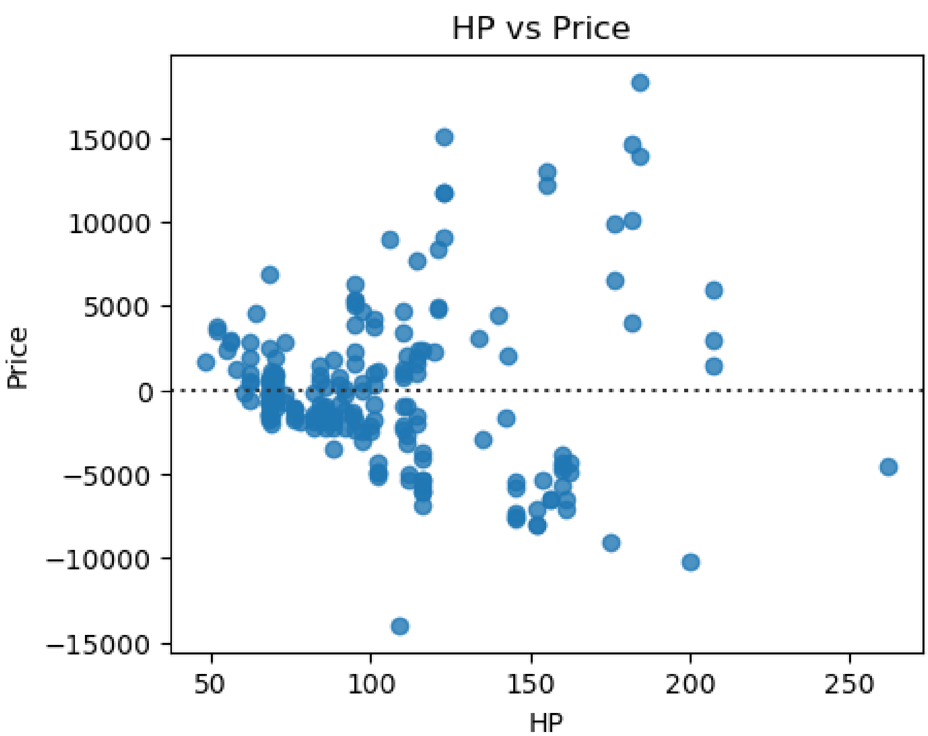

width vs Price
Price
width
30000
20000
10000
0
−10000
−20000
−30000
−40000
60
70
80
90
100
110

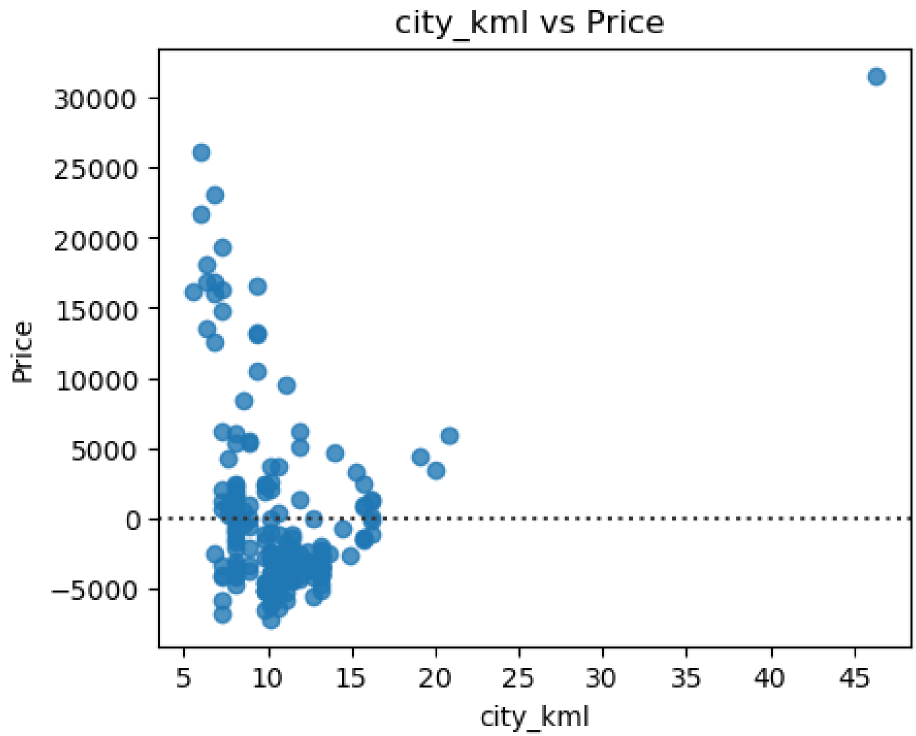

curb-weight vs Price
Price
curb-weight
20000
15000
10000
5000
0
−5000
−10000
0
500
1000
1500
2000
2500
3000
3500
4000

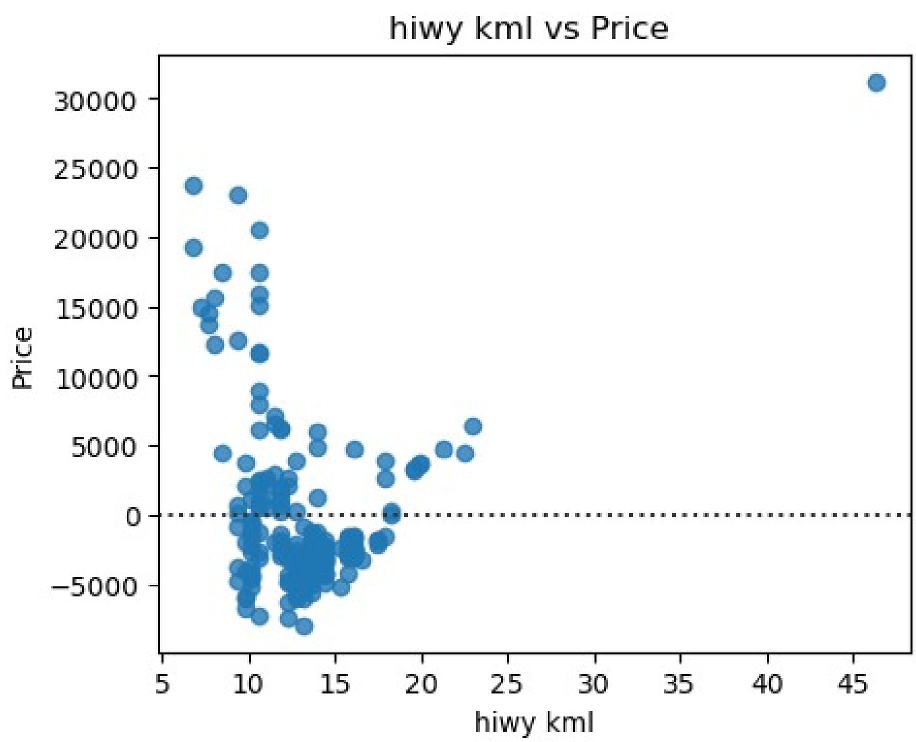

nof_cylndrs vs Price
Price
30000
20000
10000
0
−10000
0
20
40
60
80
100
nof_cylndrs

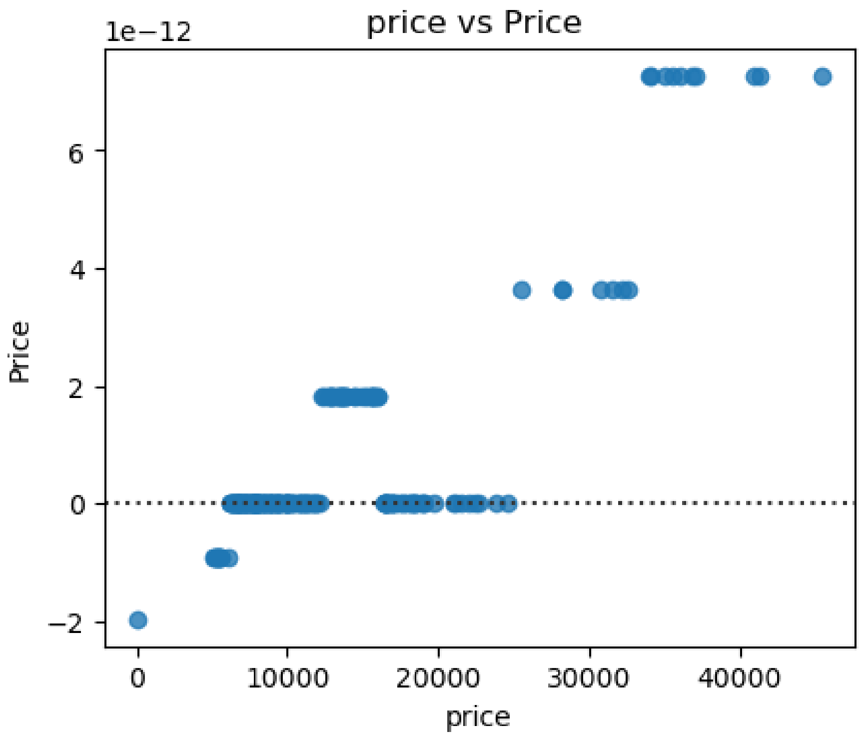

engine-size vs Price
Price
15000
10000
5000
0
−5000
−10000
50
100
150
200
250
300
engine-size


bore vs Price
Price
30000
25000
20000
15000
10000
5000
0
−5000
−10000
0
20
40
60
80
100
bore


HP vs Price
Price
15000
10000
5000
0
−5000
−10000
−15000
50
100
150
200
250
HP


city_kml vs Price
Price
city_kml
30000
25000
20000
15000
10000
5000
0
−5000
5
10
15
20
25
30
35
40
45

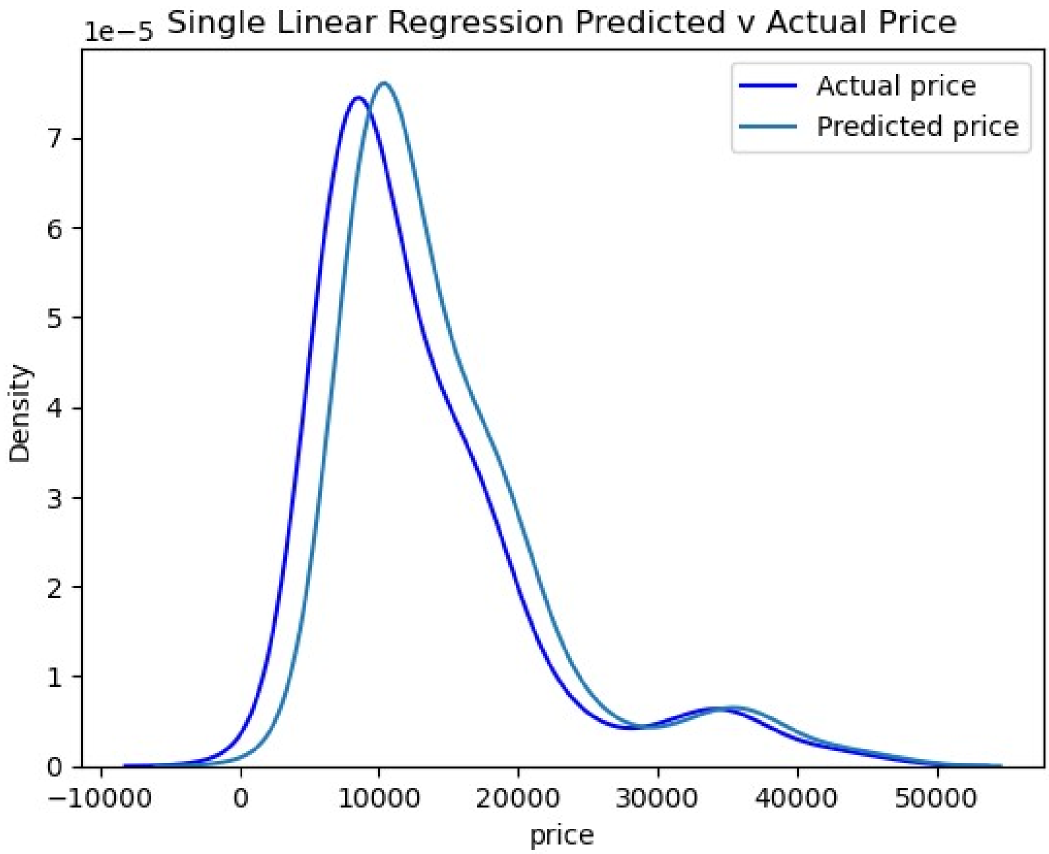

hiwy kml vs Price
Price
hiwy kml
30000
25000
20000
15000
10000
5000
0
−5000
5
10
15
20
25
30
35
40
45



Distribution Plots

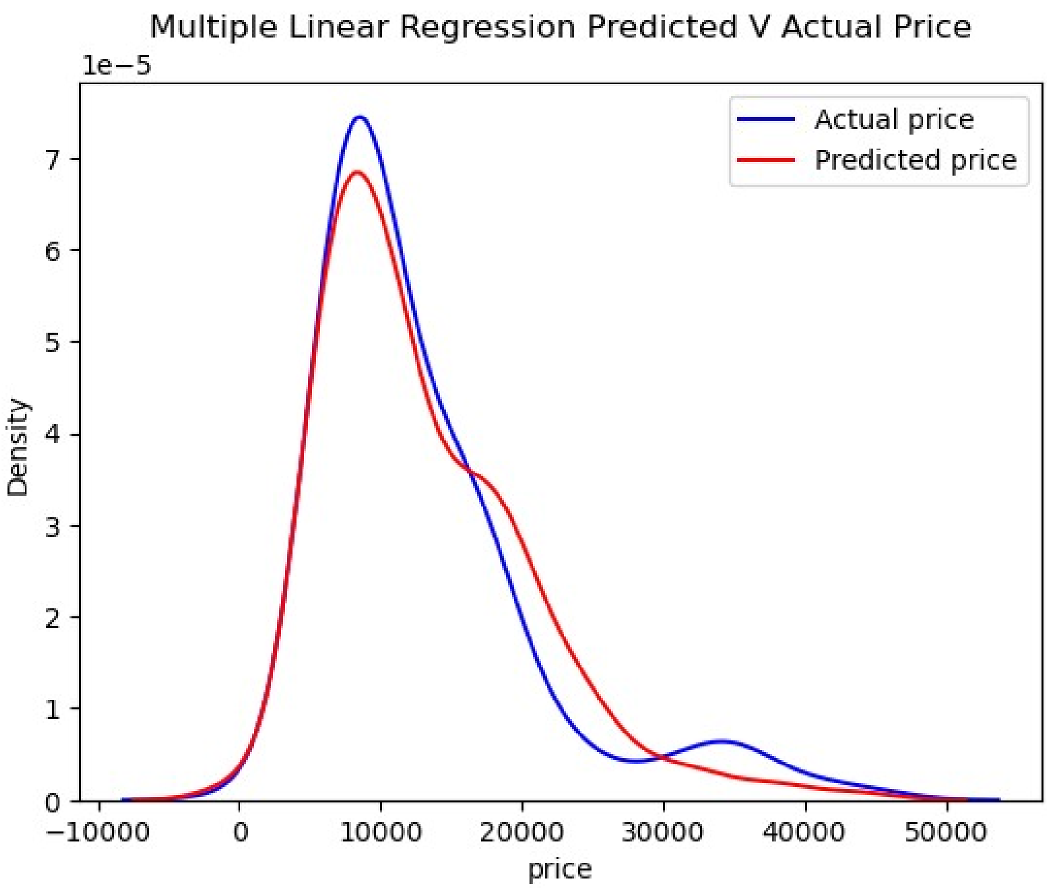
Distribution Plot for Single Linear Regression

```
ax = sns.kdeplot(df_y, color='blue', label="Actual price")
sns.kdeplot(new_pred_price, color='red', label="Predicted price")
plt.title("Single Linear Regression Predicted v Actual Price")
plt.legend()
plt.show()
```



Dirstribution Plot for Multiple Linear Regression

```
ax = sns.kdeplot(df_y, color='blue', label="Actual price")
sns.kdeplot(yhat_multi_predict, color='red', label="Predicted price")
plt.title("Multiple Linear Regression Predicted V Actual Price")
plt.legend()
plt.show()
```



Polynomial Regression

```
* For polynomial regression to occur we need pipelines (algorithm
methods used (customized))

* So while giving input to the pipeline we have to specify how we want
our data to be handled and in what all way should we proceed
```

- We know that we have been following the linear regression methods which is more suitable for the data and we will also tone the data to the standard scaler format

We use pipeline so that most of the data processing or modelling process goes through the pipeline only and therefore it follows the standards that we assign accordingly without sacrifising any data

```
sS

StandardScaler()

X2
```

```
     nrml_loss  wheel-base    length     width  curb-weight  
nof_cylndrs  \
0    -0.206272   -1.680871 -0.386650 -0.543101     0.009605    -
0.119070   
1    -0.206272   -1.680871 -0.386650 -0.543101     0.009605    -
0.119070   
2    -0.206272   -0.710259 -0.203716 -0.163403     0.515256     
0.149841   
3     1.101058    0.161646  0.207887  0.026447    -0.378367    -
0.119070   
4     1.101058    0.095842  0.207887  0.080689     0.517095     
0.015385   
..         ...         ...       ...       ...          ...          .
..   
196  -0.859937    1.691593  1.137805  0.758722     0.752452    -
0.119070   
197  -0.859937    1.691593  1.137805  0.731601     0.930809    -
0.119070   
198  -0.859937    1.691593  1.137805  0.758722     0.862776     
0.149841   
199  -0.859937    1.691593  1.137805  0.758722     1.239716     
0.149841   
200  -0.859937    1.691593  1.137805  0.758722     0.954713    -
0.119070   

     engine-size      bore        HP  city_kml  hiwy kml     price  
0       0.078360 -0.051820  0.201582 -0.524967 -0.462029  0.047489  
1       0.078360 -0.051820  0.201582 -0.524967 -0.462029  0.424425  
2       0.609001 -0.158008  1.356934 -0.754855 -0.576539  0.424425  
3      -0.428161 -0.089456 -0.040236 -0.180136 -0.118498  0.104563  
4       0.223080 -0.089456  0.309056 -0.869799 -1.034581  0.543589  
..           ...       ...       ...       ...       ...       ...  
196     0.343681 -0.010151  0.282188 -0.295080 -0.347519  0.467700  
197     0.343681 -0.010151  1.518146 -0.754855 -0.691050  0.743660  
198     1.115522 -0.037034  0.819561 -0.869799 -0.920070  1.049724  
199     0.440161 -0.113651  0.067238  0.049752 -0.462029  1.173279  
200     0.343681 -0.010151  0.282188 -0.754855 -0.691050  1.192721  

[201 rows x 12 columns]
```

We know that using the X2 standardized data can lead to some errors so we can actually proceed with another standard scaler operations with the initiator dataframe

```
initiator.head(5)

           nrml_loss  wheel-base  length  width  curb-weight  
nof_cylndrs  \
symboling
```

```
3              118.0        88.6   168.8   64.1          2548
4
3              118.0        88.6   168.8   64.1          2548
4
1              118.0        94.5   171.2   65.5          2823
6
2              164.0        99.8   176.6   66.2          2337
4
2              164.0        99.4   176.6   66.4          2824
5

           engine-size  bore     HP   city_kml   hiwy kml  price
symboling
3                  130  3.47  111.0   8.928024  11.478888  13495
3                  130  3.47  111.0   8.928024  11.478888  16500
1                  152  2.68  154.0   8.077736  11.053744  16500
2                  109  3.19  102.0  10.203456  12.754320  13950
2                  136  3.19  115.0   7.652592   9.353168  17450
```

```
init_df_noprice = initiator.drop(columns=['price'])

init_df_noprice.head(3)
```

```
           nrml_loss  wheel-base  length  width  curb-weight
nof_cylndrs  \
symboling

3              118.0        88.6   168.8   64.1          2548
4
3              118.0        88.6   168.8   64.1          2548
4
1              118.0        94.5   171.2   65.5          2823
6

           engine-size  bore     HP  city_kml   hiwy kml
symboling
3                  130  3.47  111.0  8.928024  11.478888
3                  130  3.47  111.0  8.928024  11.478888
1                  152  2.68  154.0  8.077736  11.053744
```

Polynomial Regression using numpy

```
pn_S_x = initiator['engine-size']
pn_S_y = initiator['price']

print(pn_S_x.ndim)
print(pn_S_y.ndim)

1
1
```

```python
# print(pn_S_x.shape)
# print(pn_S_x.ndim)
# pn_S_x = np.array(pn_S_x).reshape(-1,1)
# print(pn_S_x.shape)
# print(pn_S_x.ndim)

# print(pn_S_y.shape)
# print(pn_S_y.ndim)
# pn_S_y = np.array(pn_S_y).reshape(-1,1)
# print(pn_S_y.shape)
# print(pn_S_y.ndim)

# After converting both into a 2 dimensional we will now flatten and
see if the ndim still occurs to be 1D

# pn_S_x_flattened = pn_S_x.values.flatten()
# pn_S_y_flattened = pn_S_y.flatten()

# print(pn_S_y_flattened.ndim)
# print(pn_S_x_flattened.ndim)

# pn_S_y_new = pn_S_y.flatten()
# print(pn_S_y_new.ndim)

# now lets check the ndim of these new variables

print(pn_S_x.ndim)
print(pn_S_y.ndim)
```

```
1
1
```

```python
# fit the polynomial feature into the numpy library polyfit

pn_S = np.polyfit(pn_S_x , pn_S_y , 3)

# pn_S_new = pn_S.flatten()
# pn_S_new.ndim

pn_S_eq = np.poly1d(pn_S)

print(F"Polynomial Regression (Equation) Numpy :\n\n {pn_S_eq}")
```

```
Polynomial Regression (Equation) Numpy :

            3        2
-0.006839 x + 3.76 x - 452.9 x + 2.301e+04
```

Pipelines

Polynomial Regression using SKlearn

Before we move forward we will now use the pipeline to ease our work instead of doing everything step by step on our own

```
input_pipe1 = [('scale',StandardScaler()) ,
('polynomial',PolynomialFeatures(degree=2)) ,
('model',LinearRegression())]

pipeline1 = Pipeline(input_pipe1)
```

For Single Column polynomial regression

here we are actually fitting the column 'engine-size' and the 'price' column

```
x_plnrg = pn_S_x.values.reshape(-1,1)
print(x_plnrg.ndim)
y_plnrg = pn_S_y.values.reshape(-1,1)
print(y_plnrg.ndim)

2
2

single_col_pipe_result = pipeline1.fit(x_plnrg , y_plnrg)

single_col_pipe_result

Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures()),
                ('model', LinearRegression())])

# lets check the accuracy of the model that we have for this pipeline
with single column as its predicter

single_col_pipe_result.score(x_plnrg , y_plnrg)

0.7569936776403755

y_plnrg_hat_1_col = single_col_pipe_result.predict(x_plnrg)
print(y_plnrg_hat_1_col)

[[13693.62820877]
 [13693.62820877]
 [17394.15324729]
 [10145.88480778]
 [14704.50207588]
 [14704.50207588]
 [14704.50207588]
 [14704.50207588]
```

```
[13862.19259109]
[ 9976.56886619]
[ 9976.56886619]
[19405.65244593]
[19405.65244593]
[19405.65244593]
[26904.96195125]
[26904.96195125]
[26904.96195125]
[ 1980.18511772]
[ 6923.0402523 ]
[ 6923.0402523 ]
[ 6923.0402523 ]
[ 6923.0402523 ]
[ 8281.53055211]
[ 6923.0402523 ]
[ 6923.0402523 ]
[ 6923.0402523 ]
[ 8281.53055211]
[12343.88332599]
[18065.19956874]
[ 7262.86779797]
[ 7262.86779797]
[ 5051.54618351]
[ 7262.86779797]
[ 7262.86779797]
[ 7262.86779797]
[ 7262.86779797]
[10315.16658759]
[10315.16658759]
[10315.16658759]
[10315.16658759]
[10315.16658759]
[10315.16658759]
[10484.41420561]
[11837.16532549]
[34992.2021545 ]
[34992.2021545 ]
[46079.41542628]
[ 7092.97110603]
[ 7092.97110603]
[ 7092.97110603]
[ 7092.97110603]
[ 7092.97110603]
[ 3517.24920292]
[ 3517.24920292]
[ 3517.24920292]
[ 5221.85281687]
[12343.88332599]
```

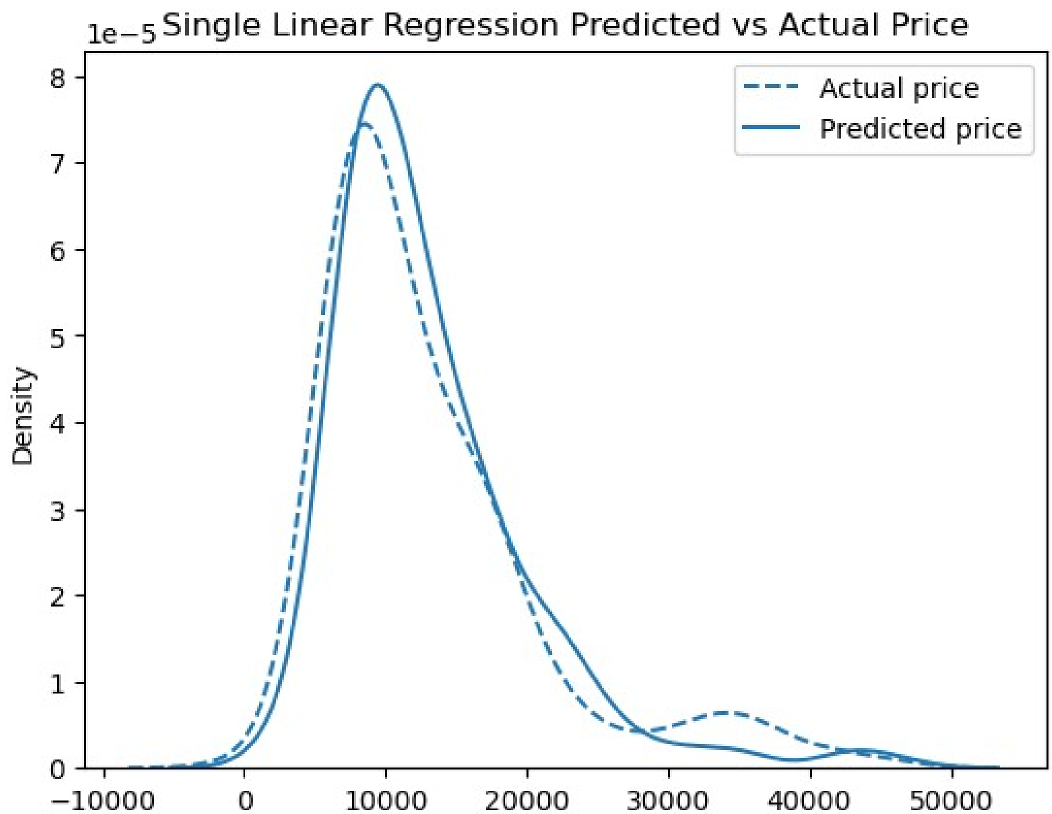
```
 [12343.88332599]
 [12343.88332599]
 [12343.88332599]
 [12343.88332599]
 [12343.88332599]
 [15377.73475158]
 [10145.88480778]
 [22580.46553136]
 [22580.46553136]
 [22580.46553136]
 [22580.46553136]
 [31041.35344767]
 [31041.35344767]
 [43159.9376577 ]
 [42509.66170169]
 [15377.73475158]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 8281.53055211]
 [10315.16658759]
 [12343.88332599]
 [18065.19956874]
 [18065.19956874]
 [18065.19956874]
 [12343.88332599]
 [12343.88332599]
 [10315.16658759]
 [10315.16658759]
 [ 8111.83883088]
 [ 9129.47673147]
 [ 8111.83883088]
 [ 8111.83883088]
 [ 8111.83883088]
 [ 8111.83883088]
 [ 8111.83883088]
 [ 8111.83883088]
 [ 8111.83883088]
 [ 8111.83883088]
 [12006.10548744]
 [12006.10548744]
 [22246.85543061]
 [22246.85543061]
 [22246.85543061]
 [22246.85543061]
 [22246.85543061]
 [22246.85543061]
 [12006.10548744]
 [17394.15324729]
```

```
 [12006.10548744]
 [17394.15324729]
 [12006.10548744]
 [17394.15324729]
 [12006.10548744]
 [17394.15324729]
 [12006.10548744]
 [17394.15324729]
 [14367.68076732]
 [ 6923.0402523 ]
 [ 8281.53055211]
 [ 6923.0402523 ]
 [ 6923.0402523 ]
 [ 8281.53055211]
 [12343.88332599]
 [18065.19956874]
 [17226.30626247]
 [24412.87851782]
 [24412.87851782]
 [24412.87851782]
 [14030.72281162]
 [14030.72281162]
 [12175.01148761]
 [12175.01148761]
 [12175.01148761]
 [12175.01148761]
 [12175.01148761]
 [12175.01148761]
 [ 8111.83883088]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 9976.56886619]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 7262.86779797]
 [ 8281.53055211]
 [ 8281.53055211]
 [10315.16658759]
```

```
 [10315.16658759]
 [ 8281.53055211]
 [ 8281.53055211]
 [ 8281.53055211]
 [ 8281.53055211]
 [ 8281.53055211]
 [ 8281.53055211]
 [ 8281.53055211]
 [16386.55891157]
 [16386.55891157]
 [16386.55891157]
 [16386.55891157]
 [16386.55891157]
 [16386.55891157]
 [12343.88332599]
 [10315.16658759]
 [12343.88332599]
 [12343.88332599]
 [12343.88332599]
 [20576.75521975]
 [20576.75521975]
 [20576.75521975]
 [18903.23883037]
 [ 8111.83883088]
 [10145.88480778]
 [ 8111.83883088]
 [10145.88480778]
 [10145.88480778]
 [ 8111.83883088]
 [10145.88480778]
 [10145.88480778]
 [10145.88480778]
 [14704.50207588]
 [ 8111.83883088]
 [10145.88480778]
 [15545.95751604]
 [15545.95751604]
 [15545.95751604]
 [15545.95751604]
 [13693.62820877]
 [13693.62820877]
 [15545.95751604]
 [15545.95751604]
 [20911.04855621]
 [16218.50695604]
 [15545.95751604]]
```

```
# we can see the plot of price difference between the predicted and
actual price value for the single column
```

```
# Create figure and axis
fig1, a1 = plt.subplots()

# Plotting on the specified axis `a`
ax = sns.kdeplot(y_plnrg, color='red', label="Actual price", ax=a1,
linestyle = '--')
sns.kdeplot(y_plnrg_hat_1_col, color='green', label="Predicted price",
ax=ax)

plt.title("Single Linear Regression Predicted vs Actual Price")
plt.legend()
plt.show()
```



For Multiple Column Polynomial Regression

```
init_df_noprice.head(3)

           nrml_loss  wheel-base  length  width  curb-weight
nof_cylndrs  \
symboling

3              118.0        88.6   168.8   64.1         2548
4
3              118.0        88.6   168.8   64.1         2548
4
```

```
1                118.0        94.5   171.2   65.5         2823
6

           engine-size  bore     HP  city_kml   hiwy kml
symboling
3                  130  3.47  111.0  8.928024  11.478888
3                  130  3.47  111.0  8.928024  11.478888
1                  152  2.68  154.0  8.077736  11.053744

pn_S_y.head(3)

symboling
3    13495
3    16500
1    16500
Name: price, dtype: int64

# fitting the pipeline

multi_col_pipe_result = pipeline1.fit(init_df_noprice , pn_S_y )
multi_col_pipe_result

Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures()),
                ('model', LinearRegression())])

multi_col_pipe_result.score(init_df_noprice , pn_S_y )

0.9707569951753966
```

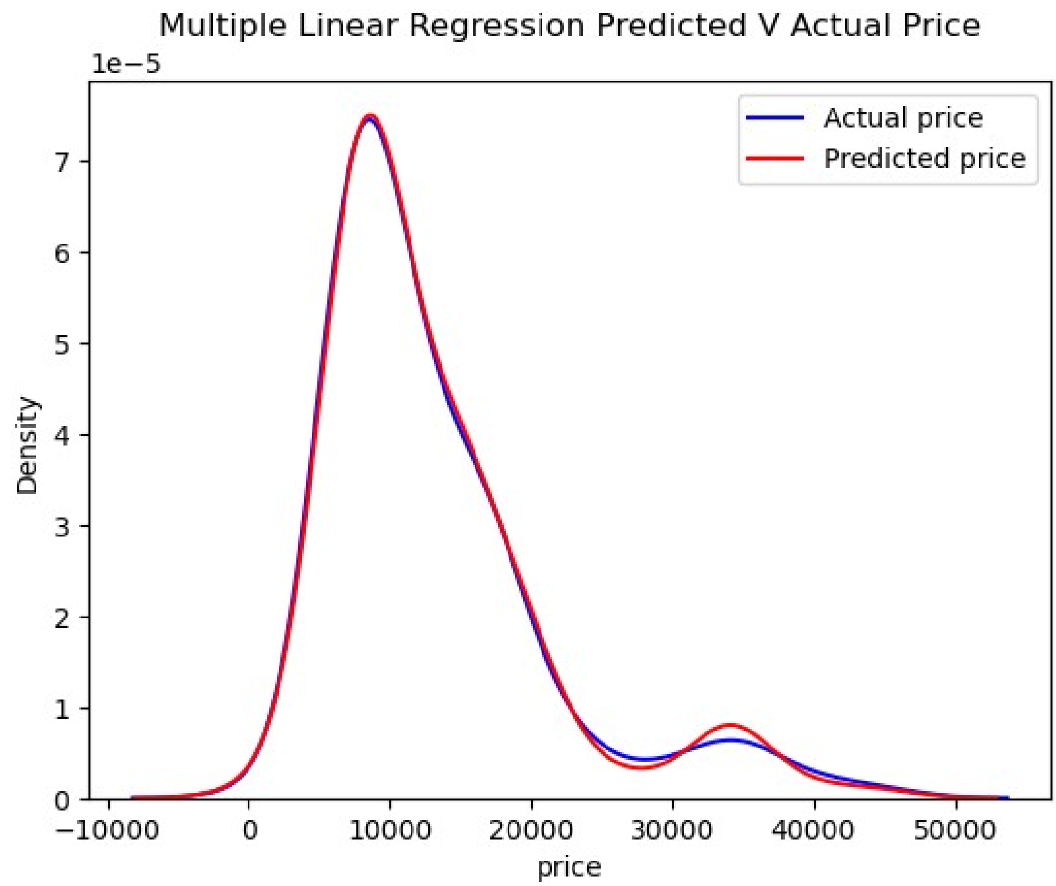
Now thats what I call Accuracy !!!

it's above 95 and I'm Happy for now and now lets see if theres is any way to increase the accuracy

```
# Lets predict the outcome and see for real whether it actually looks
like what it says

yhat_multi_col_predict =
multi_col_pipe_result.predict(init_df_noprice)

# lets see the result
print(yhat_multi_col_predict)

[13810.05707167 13810.05707167 15480.11879822 10031.52682884
 13866.00585333 14758.35367901 19253.78387644 17740.68793034
 23588.19270814 16303.1396133  16303.1396133  20859.80840277
 19661.86430423 22808.5054826  32691.23448462 37856.16491229
 33746.86063395  5587.18260791  4020.09062527  5490.4981515
  4514.51686439  7267.4998009   8478.93976708  6936.98857396
  6893.00864921  6893.00864921  8177.56647006  8725.66047629
```

```
 12964.30494211  5946.99941763  5286.47123286  7866.0108398
  7687.41017513  7883.95288059  7726.59859042  8104.37817032
  7758.68190138  8133.39993013  8781.37870519  8345.10583744
 11927.7525365   9747.8386895   8615.91533498 12195.35643606
 33765.09980935 33765.09980935 35773.85389036  6204.4360725
  7186.6916598   7193.90457763  7004.73995352  7061.82979375
 12471.23030597 12471.23030597 12578.68662865 14894.5555275
  9230.38176501  9462.79266653  9230.38176501  9462.79266653
 10739.91577455  9574.28533023 19608.60273917   109.00671725
 26090.20312792 27315.13585956 29167.75791936 32708.95085611
 35922.74472472 34498.84053942 41263.37838477 44756.578571
 15693.69631214  3686.63377654  6395.45669864  6341.3322464
  7711.65659151  9925.66529277  9106.36255158 13178.22304377
 15500.14434001 15623.423293    8694.99654864  8966.32217767
 10476.44522305 10198.50227863  6503.92805404  8536.55486104
  6637.74158693  6583.63218717  7487.42473268  6779.89782801
  7125.36998291  6725.98492784  7553.20459798  7566.17269083
  8530.30665206  8418.27728682 15472.09256163 15946.72129689
 17771.74743707 19291.864843   19169.05699427 19792.77577347
 15179.93689104 16126.89184579 16381.04447871 15084.66112974
 14708.78105727 16180.49936024 15230.02367256 15641.4184604
 14581.88332008 16180.49936024 16340.7114448   4930.05785286
  8458.47377515  6902.55428831  8010.22104827  7617.44999805
  7148.43551534 12707.99940674 22333.20400318 34105.11013972
 34105.11013972 34105.79444347 10142.05217471  9790.61148509
 12685.46408675 12103.73388354 15299.90614103 12016.99370594
 18922.41658188 19198.50731241  5480.89385016  6257.9136805
  7868.85852965  8172.19876947  8346.19087032  9690.21685332
  9006.70151834 10011.51181109  7890.67190613  9329.74894928
  9198.80116587 12605.2757985   4472.44755271  7073.08244059
  7126.65911127  8261.70030414  9169.0523683   9235.32358175
  7571.3972615   7533.94027775  6905.89319603  7303.52965757
  8973.86459399  7661.49022052  7706.8528146   8561.35598587
  8746.69095341  9017.02371869  9364.17345298 10039.00431324
  9979.71969636 10201.22133784 12000.88348228 12464.77612807
 15542.17747178  8687.78281659 12335.38563364 10249.08182264
 10249.08182264 10561.54256035 18486.58055495 19508.40703764
 14525.70398523 14951.02851981  6417.90984596  8474.45175473
  7049.33196209  8803.45500966  8940.46888721 11438.92841176
  9601.81807587  7526.44417554 10114.18460047 15531.06976218
 13847.98232401 12113.40072796 14953.04083726 14776.30846749
 13977.58306895 14280.6840337  18871.7118333  19412.82656452
 17186.72731707 20428.48215578 19436.02650185 20624.09376264
 20614.04460867]

# Plotting on the specified axis `a`
ax = sns.kdeplot(pn_S_y, color='blue', label="Actual price")
sns.kdeplot(yhat_multi_col_predict, color='red', label="Predicted
price")
plt.title("Multiple Linear Regression Predicted V Actual Price")
```

```
plt.legend()
plt.show()
```



As you can see the predicted and the actual price shows the 96% accuracy in the graph and it is clearly visible

Though we may have to work on the (35000,1) axis , I think we can now move onto the next step

Now lets veiw the scores for each model below and see what is !!

Mean Squared Error (MSE)

```
# mean squared error for the single linear Regression

mean_squared_error(pn_S_y , new_pred_price)

3029915.2515888596

# mean squared error for the multiple linear Regression

mean_squared_error(pn_S_y, yhat_multi_predict)
```

```
10881906.423845338
```

```
# mean squared error for the single Polynomial Regression

mean_squared_error(pn_S_y, y_plnrg_hat_1_col)
```

```
15444432.372027365
```

```
# mean squared error for the multiple Polynomial Regression

mean_squared_error(pn_S_y, yhat_multi_col_predict)
```

```
1858559.0941953901
```

R^2 coeficient determination

```
# for multiple polynomial and linear regression

pipeline1.score(init_df_noprice , pn_S_y)
```

```
0.9707569951753966
```

```
# for multiple linear regression

mLr.score(init_df_noprice , pn_S_y)
```

```
0.828781531323245
```

Model Evaluation

```
# train test split
# We now train and test the data into 4 parts
x_train, x_test, y_train, y_test = train_test_split(init_df_noprice ,
pn_S_y, test_size=40 , random_state=53 )

print(f"shape of x_train : {x_train.shape}")
print(f"shape of y_train : {y_train.shape}")

print(f"shape of x_test : {x_test.shape}")
print(f"shape of y_test : {y_test.shape}")
```

```
shape of x_train : (161, 11)
shape of y_train : (161,)
shape of x_test : (40, 11)
shape of y_test : (40,)
```

```
# custom pipeline with degree 3
pipeline_input2 = [('scale',StandardScaler()),
```

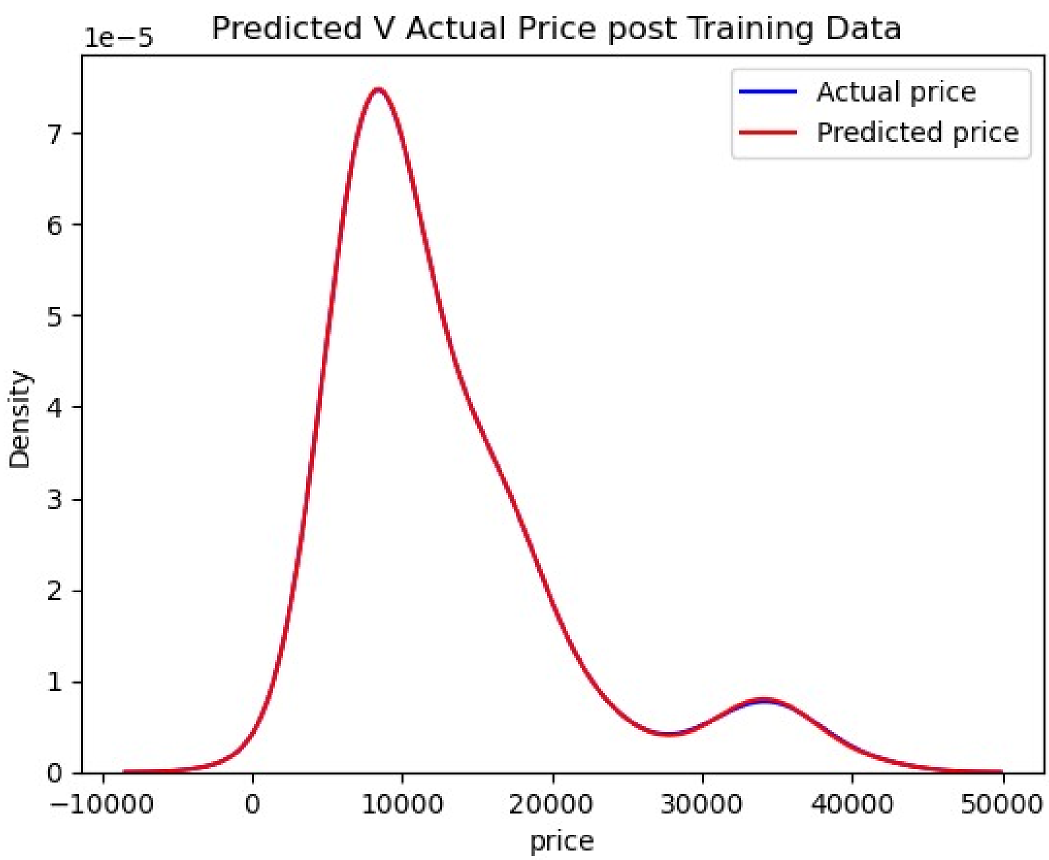
```
('polynomial',PolynomialFeatures(degree=3)),
('model',LinearRegression())]

# assigning the pipeline to the variable pipeline_input2

pipeline2 = Pipeline(pipeline_input2)
pipeline2

Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures(degree=3)),
                ('model', LinearRegression())])

# now lets fit the train data to the pipeline2 which has the degree 3

pipeline2.fit(x_train, y_train)

Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures(degree=3)),
                ('model', LinearRegression())])

#  Now lets check the score value for the training data and see what
it gives

pipeline2.score(x_train, y_train)

0.9990058757868711

#  Now lets check the score value for the testing data and see what it
gives

pipeline2.score(x_test, y_test)

-489.5082634641962

# trained data prediction

train_predict = pipeline2.predict(x_train)
train_predict

array([11845.        ,  7053.00000001,  7957.        ,  7150.5       ,
       12964.        ,  8778.        ,  8238.        ,  7198.        ,
       17672.81967251,  6575.00000001, 16695.        ,  6338.        ,
       33278.00000001, 33900.        ,  7295.        ,  7799.        ,
       10698.        , 37027.99999999,  6918.00000001,  6189.        ,
        6229.        , 12170.        , 13200.        , 17425.        ,
        5399.        , 13950.        , 15749.99999999, 17075.        ,
       10345.        , 13495.        , 10795.        ,  7775.        ,
        9095.        ,  7999.        , 13295.        , 41315.        ,
        6795.        , 25552.        , 14868.99999999, 31600.        ,
        5498.99999999,  9538.        ,  5389.        , 10898.        ,
        8495.        , 16500.        , 22470.00000001, 22625.        ,
       28248.        , 12440.        ,  9960.        ,  6669.        ,
```

```
       12945.        ,  8189.        ,  7395.        , 16845.        ,
       19045.        ,  6784.99999999,  6855.        , 17425.        ,
        5572.        ,  8558.00000001, 15510.        ,  8921.        ,
        7775.00000001, 17450.        , 15250.        , 10295.        ,
        9995.        , 11900.        , 34184.        , 35999.99999999,
       11494.27653468, 18280.        ,  9258.        , 11048.00000001,
       11694.        ,  6529.        , 11338.92805507, 14489.        ,
        7298.99999998,  9720.        ,  8949.        ,  6938.        ,
       18920.        , 16677.5       , 24565.00000001, 13499.00000001,
        7129.        , 11245.        ,  8919.29186888, 16515.        ,
        7895.        ,  6377.        , 21485.        ,  5151.        ,
       10198.        ,  7788.        ,  9279.        ,  6989.        ,
       12629.        , 15645.        , 16558.00000001, 11595.        ,
        9298.        ,  7099.        ,  6488.        , 15997.99999999,
        9549.        ,  5118.00000001, 15040.        , 18150.        ,
        7463.        ,  7499.        ,  8058.        , 13860.        ,
       16503.        ,  9959.        ,  6692.        , 33278.00000001,
       13645.        ,  7975.        , 18399.        , 35056.        ,
       17199.        , 21105.        , 10245.        ,  9279.        ,
       14399.        , 36880.        ,  5195.        ,  7957.        ,
        7689.        , 30760.        , 17710.        , 20970.        ,
        9495.        ,  7898.        , 33900.        , 16677.5       ,
        8921.        ,  7349.        ,  8845.        ,  7295.        ,
        6849.        , 22018.        ,  6295.        ,  9720.        ,
        7126.        ,  9233.        ,  9980.        ,  6649.00000003,
        6095.        ,  9895.        ,   108.99999999,  7898.        ,
       12290.        ,  9429.68386888,  6229.        ,  6479.        ,
        7150.5       ])
```

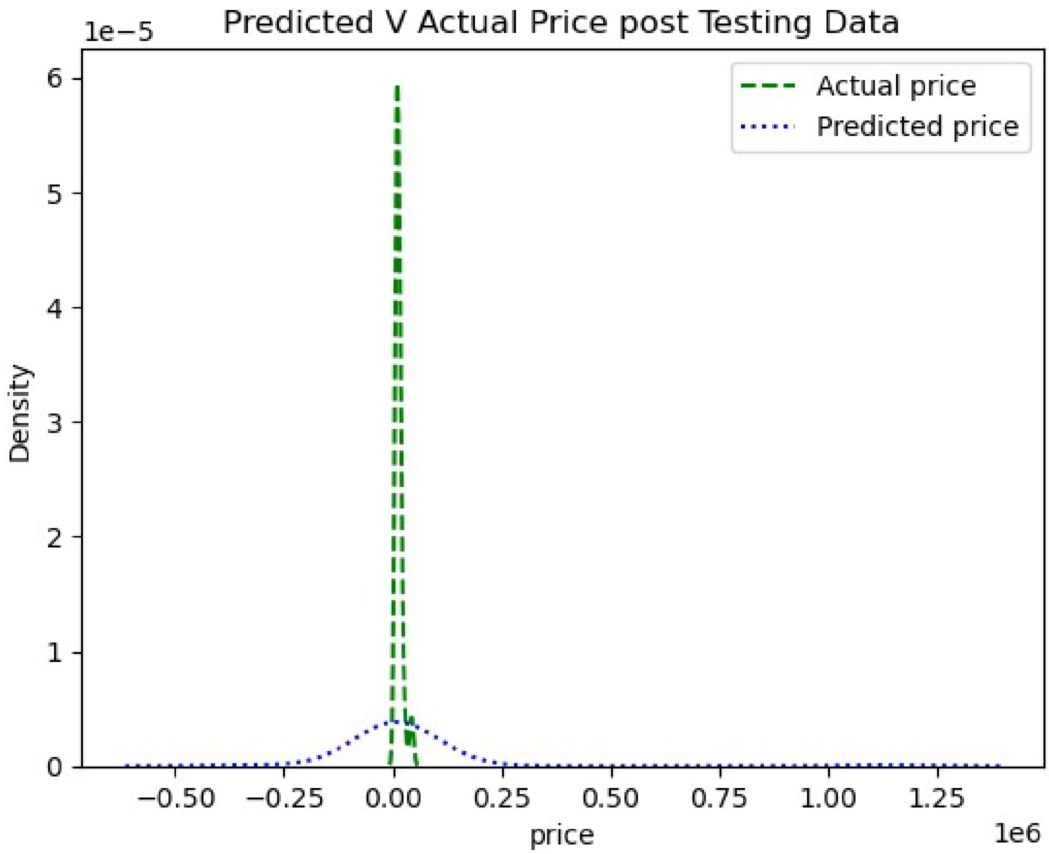
```
# test data prediction

test_predict = pipeline2.predict(x_test)
test_predict
```

```
array([ 1.02450000e+04,  3.10587499e+03,  1.14932951e+04,
2.14844609e+04,
        4.40328352e+03,  2.53318982e+04,  9.93788715e+04,
1.34950000e+04,
       -7.09976811e+03, -5.68291139e+04, -1.74792698e+04,
5.62464260e+04,
        8.71611280e+03,  1.73224088e+04,  1.23734744e+04,
1.18450000e+04,
        2.06376371e+03,  1.94149793e+04,  1.08924879e+05,
2.60540876e+03,
       -9.07060492e+04,  7.94900257e+03,  4.10094960e+02,
9.51015155e+03,
        3.02367752e+04,  4.71614872e+03, -6.89792613e+04,
1.20451545e+05,
        1.91770658e+04,  1.12160152e+06,  1.99286157e+04,
3.73459842e+03,
```

```
        1.08980000e+04, -7.46977868e+04, -3.61006259e+03,
8.32755028e+03,
        5.59002675e+01,  1.77664971e+04, -1.64576267e+03, -
3.41474624e+05])

# Plotting on the specified axis `a`
ax = sns.kdeplot(y_train, color='blue', label="Actual price")
sns.kdeplot(train_predict, color='red', label="Predicted price")
plt.title("Predicted V Actual Price post Training Data ")
plt.legend()
plt.show()
```




```
# Plotting on the specified axis `a`
ax = sns.kdeplot(y_test, color='green', label="Actual
price",linestyle='--')
sns.kdeplot(test_predict, color='blue', label="Predicted
price",linestyle=':')
plt.title("Predicted V Actual Price post Testing Data")
plt.legend()
plt.show()
```



K-Fold Cross Validation with Linear Regression

Just because cross validation is better than random sampling that's why we use this method

```
new_x_cv = init_df_noprice
new_y_cv = init_y

cv_score = cross_val_score(LinearRegression(), new_x_cv , new_y_cv ,
cv=4)
cv_score
```

```
array([ 0.81504452, -0.13111949,  0.35312385,  0.3553099 ])
```

```
np.mean(cv_score)
```

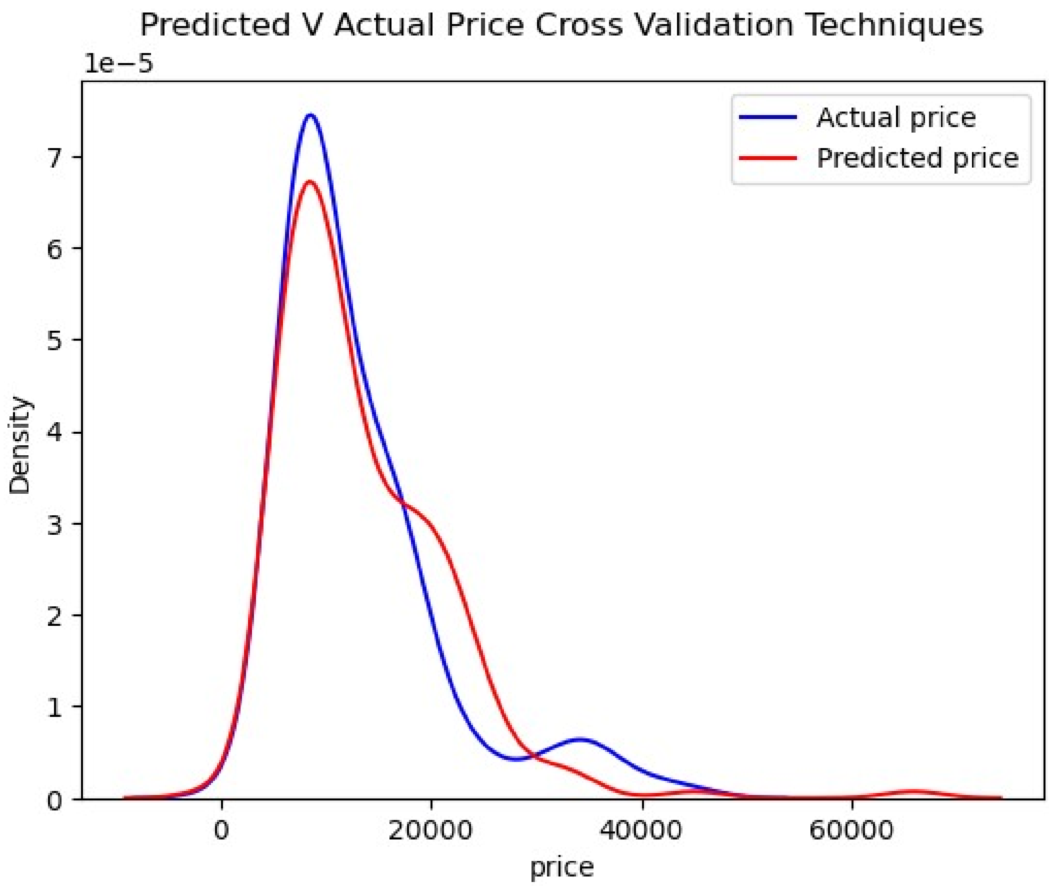
```
0.3480896962664754
```

```
yhat_cross = cross_val_predict(LinearRegression() , new_x_cv ,
new_y_cv , cv=4)
yhat_cross
```

```
array([11296.06210513, 11296.06210513, 19158.03774553, 11086.18844714,
       15398.91671355, 14081.85424366, 20008.4411672 , 20342.27956168,
```

```
       21996.16819999,  8939.84484602,  8939.84484602, 15500.82613045,
       15667.74532769, 17671.96073165, 24283.76385505, 25452.05684636,
       29437.31109844,  -825.34046375,  6083.05486103,  5832.51652365,
        6010.47131695,  6108.85105209,  9114.79657268,  6362.52449799,
        6429.29217689,  6429.29217689,  9283.49297271,  9794.25522182,
       17955.87330626,  6982.81876312,  8191.62659796,  6013.4979323 ,
        7952.63331403,  8001.19162595,  6502.39325243,  7229.14822894,
        9822.62736355,  9983.4767718 ,  9038.15330234,  5667.93651534,
       10218.94952671, 11629.04782748,  4497.86615712, 11078.42574578,
       31870.00211893, 31870.00211893, 44988.35699285,  6054.34547281,
        6485.26397088,  6500.43844336,  5633.96888321,  6619.86284587,
        8369.0677123 ,  8369.0677123 ,  8382.57729235, 12252.8093805 ,
       11003.8078996 , 10788.41246996, 11003.8078996 , 10788.41246996,
        9851.36843565, 10828.9412101 , 15944.13107589, 65815.7046736 ,
       20118.09604162, 20753.0463038 , 19666.23667442, 21666.19273972,
       28219.65569383, 25674.15336783, 34763.01410331, 33038.51280911,
       20399.64808972,  7031.26990084,  7475.35888388,  7637.47384444,
       10249.16341987, 13345.61591179, 11492.67389456, 17819.51576574,
       18057.35694881, 18070.86652885, 11013.70509002, 11121.78173039,
       12855.8394432 , 13098.36229738,  6899.71889642,  7149.58331278,
        6978.07446069,  6910.85135378,  6865.82141484,  7067.23768899,
        7281.81174028,  7000.01458209,  6900.94632296,  8025.44925163,
       11475.46197266, 11416.01982045, 21481.93142041, 21683.90146086,
       21498.19955694, 22389.54936521, 22851.54320017, 25393.8810111 ,
       20077.84228854, 22920.00352378, 21653.22887284, 25258.89434486,
       20422.0427142 , 23265.8348414 , 21997.42929849, 25604.72566248,
       20423.67360616, 23265.8348414 , 21373.4913277 ,  6289.46426868,
        7837.6212077 ,  7548.02594843,  6529.86573922,  8263.27647391,
       11080.04252983, 16079.72268462, 19169.12060744, 18029.43647701,
       18029.43647701, 18306.10153111, 12042.71172589, 11914.22416426,
       13407.21272101, 11911.89642228, 14190.77588569, 12308.03047701,
       14392.96407115, 12910.22345669,  5487.83898976,  6609.52701525,
        7480.11682531,  8149.05587518,  8435.61423194,  9267.14268032,
       10105.40825507, 10422.0926381 ,  8472.93767083,  9276.44916199,
        8860.12694931, 11069.3943013 ,  5797.8836722 ,  5777.61258702,
        5132.07818677,  5762.14376268,  6102.12155189,  4233.91467468,
        6370.79295905,  6354.58508301,  6721.73616256,  6389.44931142,
        6514.84682096,  6343.1716213 ,  6332.75227242,  8089.21425581,
        8068.95441076, 11457.12551824, 11436.86567319, 16195.89221389,
       16198.20762476, 16189.52483402, 16115.4316864 , 16095.17184135,
       15944.09128253, 11641.82448272,  9071.76268994, 11518.98434653,
       11518.98434653, 11493.51482704, 24693.07280107, 24565.95542304,
       21047.50152553, 20451.91552381,  6396.27377146,  9978.56939132,
        5643.52911254,  9225.82473241,  9189.35701131,  7178.5816536 ,
       10420.85901305,  8668.92746902, 12903.10254905, 14155.14532355,
        7970.53393743, 10328.5843677 , 16166.32017274, 15317.87032276,
       16256.96982967, 15417.20277042, 18744.79918772, 17902.1378649 ,
       17492.75969358, 20903.63489948, 21324.11270646, 16931.56882256,
       17217.27097966])
```

```
# Plotting on the specified axis `a`
ax = sns.kdeplot(new_y_cv, color='blue', label="Actual price")
sns.kdeplot(yhat_cross, color='red', label="Predicted price")
plt.title("Predicted V Actual Price Cross Validation Techniques")
plt.legend()
plt.show()
```



Polynomial Regression and Parameters

Here we will be working with alpha parameters and various degrees

```
print(f"shape of x_train : {x_train.shape}")
print(f"shape of y_train : {y_train.shape}")

print(f"shape of x_test : {x_test.shape}")
print(f"shape of y_test : {y_test.shape}")
```

```
shape of x_train : (161, 11)
shape of y_train : (161,)
```

```
shape of x_test : (40, 11)
shape of y_test : (40,)
```

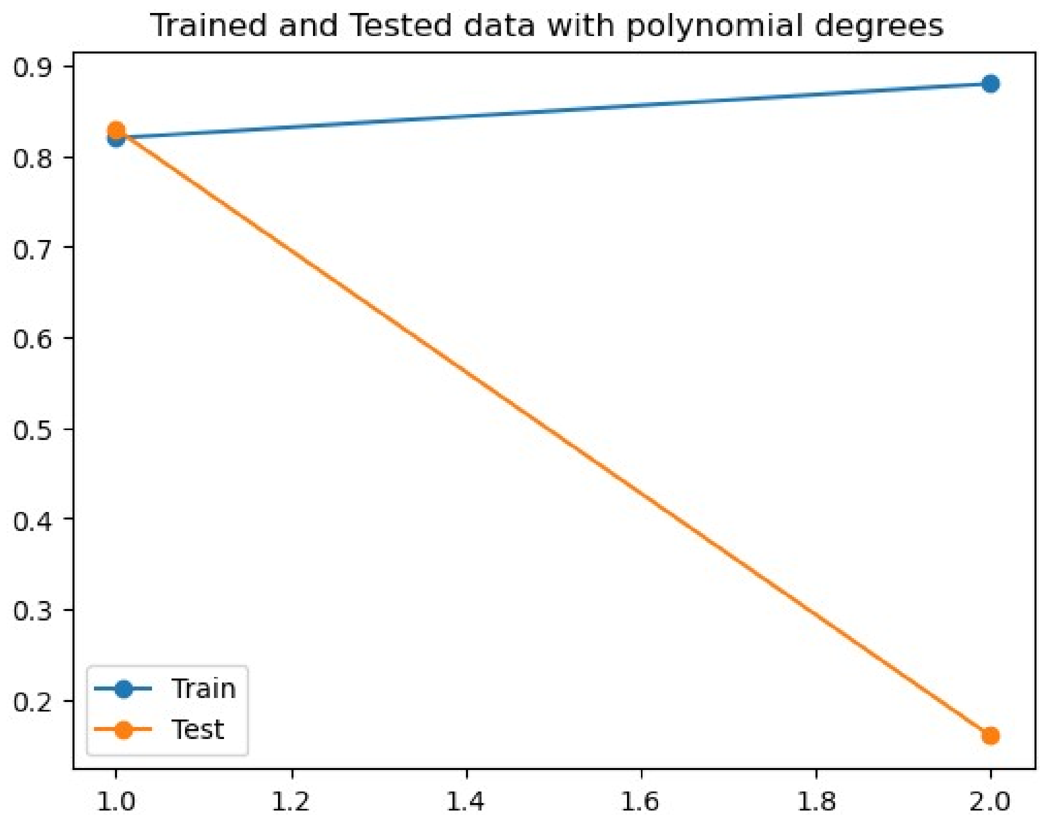
```
# we will train the data with various polynomial degrees and see their accuracies

trained_squared_coef = []
testing_squared_coef = []
polynomial_degrees_ = [1,2]

for i in polynomial_degrees_:
    # intializing the polynomial and linear regression
    Pr = PolynomialFeatures(degree=i)
    Lr = LinearRegression()

    # fitting the train data and transforming both test and train data

    x_train_Pr = Pr.fit_transform(x_train)
    x_test_Pr = Pr.transform(x_test)

    # fitting the model
    Lr.fit(x_train_Pr , y_train)

    # Evaluating the test and train data
    trained_eval = Lr.score(x_train_Pr , y_train)
    tested_eval = Lr.score(x_test_Pr, y_test)

    # appending the scores
    trained_squared_coef.append(round(trained_eval,2))
    testing_squared_coef.append(round(tested_eval,2))

print(trained_squared_coef)
print(testing_squared_coef)
```

```
[0.82, 0.88]
[0.83, 0.16]
```

Here we can see that only the the 1st degree has both valid train and test results rest of them provide less accurate training results but more accurate testing results which can result in overfitting , here we need both the training and testing data to be close to the number 1 which provide accuracy

```
# we will now plot the test and train data that we got from fit and training it through our most of the polynomial degrees
plt.plot(polynomial_degrees_ , trained_squared_coef , 'o-' , label="Train")
plt.plot(polynomial_degrees_ , testing_squared_coef , 'o-' , label="Test")
plt.title("Trained and Tested data with polynomial degrees ")
```

```
plt.legend()
plt.show()
```



From the above chart we are now convinced that the 1st polynomial degree is best suited for our data and remaining degrees provide overfitting results

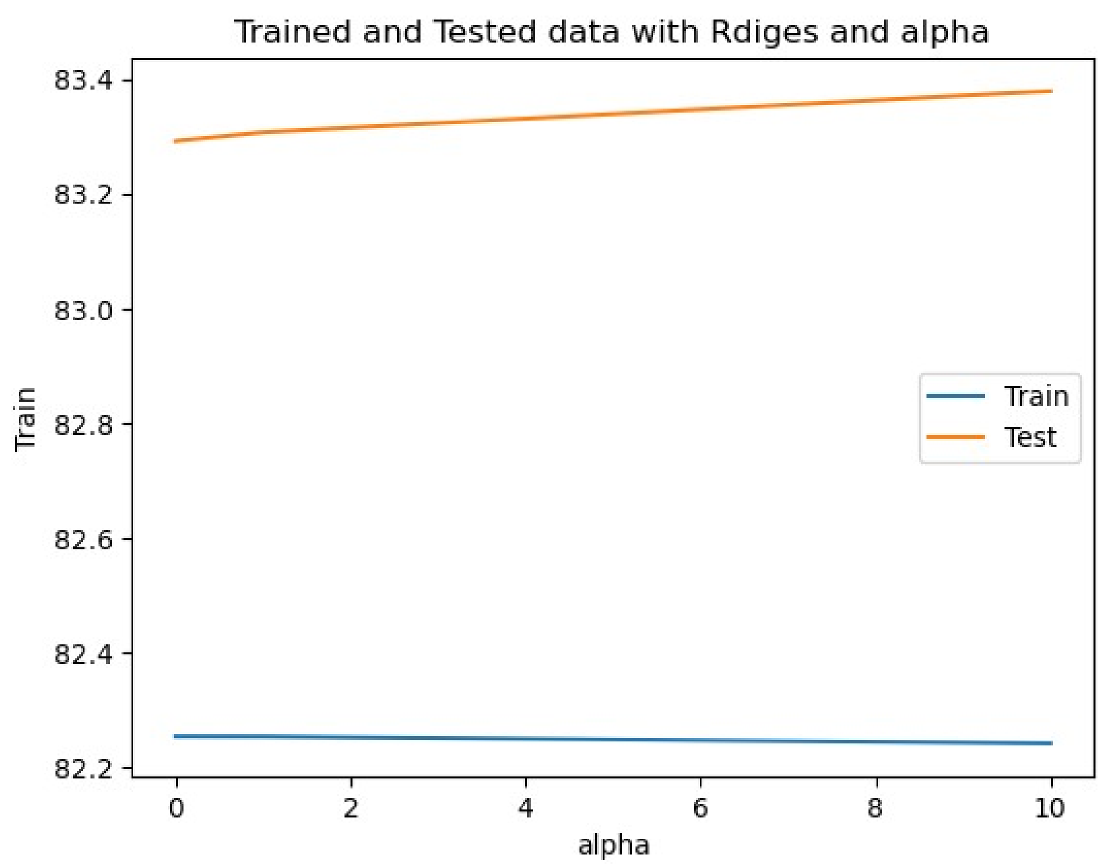
Ridge Regression

```
print(f"shape of x_train : {x_train.shape}")
print(f"shape of y_train : {y_train.shape}")

print(f"shape of x_test : {x_test.shape}")
print(f"shape of y_test : {y_test.shape}")

shape of x_train : (161, 11)
shape of y_train : (161,)
shape of x_test : (40, 11)
shape of y_test : (40,)

Rd_train_squared = []
Rd_test_squared = []
alpha_features = [0.0001,0.001,0.01,0.1,1,10]
```

```
for i in alpha_features :
    # creating an object for the Ridge Regression
    Rd = Ridge(alpha=i)

    # Fitting the model
    Rd_model = Rd.fit(x_train, y_train)

    # evaluate the models test and training scores
    Rd_train_score = Rd.score(x_train, y_train)
    Rd_test_score = Rd.score(x_test, y_test)

    # appending the scores to the appropriate lists
    Rd_train_squared.append(round(Rd_train_score*100,10))
    Rd_test_squared.append(round(Rd_test_score*100,10))

# print(f"Train :{Rd_train_squared}")
# print(f"Test :{Rd_test_squared}")
rd_list = {'alpha':alpha_features , 'Train':Rd_train_squared ,
'Test':Rd_test_squared}
Rd_df = pd.DataFrame(rd_list)

Rd_df
```

	alpha	Train	Test
0	0.0001	82.255027	83.291869
1	0.0010	82.255027	83.291884
2	0.0100	82.255027	83.292028
3	0.1000	82.255025	83.293465
4	1.0000	82.254811	83.306904
5	10.0000	82.242415	83.378777

```
# we will now plot the test and train data that we got from fit and
training it through our most of the ridge plot with alphas
sns.lineplot(Rd_df , x= 'alpha' , y='Train' , label = 'Train')
sns.lineplot(Rd_df , x= 'alpha' , y='Test' ,label = 'Test')
plt.title("Trained and Tested data with Rdiges and alpha")
plt.legend()
plt.show()
```



Ridge Regression With K-Fold Cross Validation

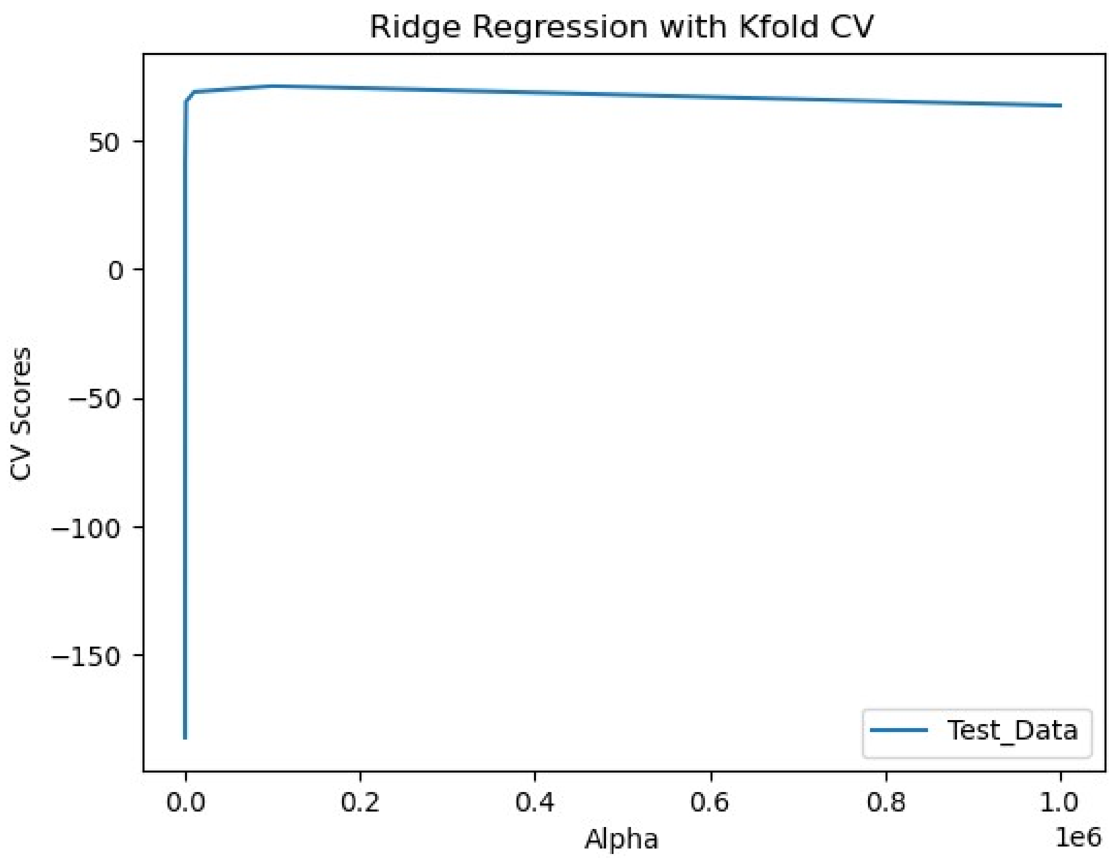
```
new_x_cv.head(3)

          nrml_loss  wheel-base  length  width  curb-weight
nof_cylndrs  \
symboling

3             118.0        88.6   168.8   64.1         2548
4
3             118.0        88.6   168.8   64.1         2548
4
1             118.0        94.5   171.2   65.5         2823
6

          engine-size  bore     HP  city_kml   hiwy kml
symboling
3                 130  3.47  111.0  8.928024  11.478888
3                 130  3.47  111.0  8.928024  11.478888
1                 152  2.68  154.0  8.077736  11.053744

new_y_cv.head(3)
```

```
symboling
3    13495
3    16500
1    16500
Name: price, dtype: int64
```

```
Rd_scores = []
alpha_features = [10,100,1000,10000,100000,1000000]

for i in alpha_features :
    # creating an object for the Ridge Regression
    Rd = Ridge(alpha=i)
    scores_cv = cross_val_score(Rd , new_x_cv , new_y_cv , cv = 3)
    Rd_scores.append(round((np.mean(scores_cv)*100),3))

Rd_kf_cv_df = {'alpha':alpha_features , 'scores':Rd_scores}
Rd_cv_df = pd.DataFrame(Rd_kf_cv_df)

Rd_cv_df
```

	alpha	scores
0	10	-182.129
1	100	41.178
2	1000	65.049
3	10000	68.717
4	100000	71.040
5	1000000	63.396

```
# lineplot for the cv with rd
sns.lineplot(Rd_cv_df , x= 'alpha', y = 'scores' , label='Test_Data' )
plt.title("Ridge Regression with Kfold CV")
plt.xlabel('Alpha')
plt.ylabel('CV Scores')
plt.show()
```



Grid Search CV & Ridge Regression

```
# specify the parameters for the alpha
parameters1 = [{'alpha':[10,100,1000,10000,100000,1000000]}]

# specify the gridsearchcv
Grid = GridSearchCV(Rd , param_grid=parameters1 , cv=3 )

# fit the grid to the data
Grid.fit(new_x_cv, new_y_cv)

GridSearchCV(cv=3, estimator=Ridge(alpha=1000000),
             param_grid=[{'alpha': [10, 100, 1000, 10000, 100000,
1000000]}])

# now lets store the result in a variable called result
results = Grid.cv_results_
grid_result = pd.DataFrame(results)
grid_result

   mean_fit_time  std_fit_time  mean_score_time  std_score_time
param_alpha  \
```

```
0       0.004344      0.001253         0.002546        0.000790
10
1       0.003358      0.000466         0.002313        0.001262
100
2       0.002671      0.000472         0.003001        0.000009
1000
3       0.003326      0.000472         0.002668        0.000477
10000
4       0.002671      0.000478         0.002329        0.000464
100000
5       0.003038      0.000813         0.002174        0.000247
1000000

               params  split0_test_score  split1_test_score  \
0       {'alpha': 10}          -6.566078           0.678283
1      {'alpha': 100}           0.104793           0.677915
2     {'alpha': 1000}           0.779579           0.688882
3    {'alpha': 10000}           0.820632           0.701343
4   {'alpha': 100000}           0.823568           0.672920
5  {'alpha': 1000000}           0.754414           0.598367

   split2_test_score  mean_test_score  std_test_score  rank_test_score

0           0.423935        -1.821287        3.356680                6

1           0.452638         0.411782        0.235753                5

2           0.483020         0.650494        0.124075                3

3           0.539548         0.687175        0.115189                2

4           0.634721         0.710403        0.081525                1

5           0.549101         0.633960        0.087516                4
```

```
# find the best result through looking at the ranks of the grid search

best_result_for_gridsearchcv =
grid_result[(grid_result['rank_test_score']==1)]
best_result_for_gridsearchcv
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time
param_alpha  \
4       0.002671      0.000478         0.002329        0.000464
100000

              params  split0_test_score  split1_test_score
split2_test_score  \
4  {'alpha': 100000}           0.823568            0.67292
0.634721
```

```
   mean_test_score  std_test_score  rank_test_score
4         0.710403        0.081525                1
```