Summer Internship Programme

Henry Harvin Education India LLP
Sector-2, Noida, U.P.-201306

Project Title – **Car Price Prediction**
Mentor Name: Ms. Pooja Gupta (Designation)

Name: Abhijit Roy
Course: Summer Internship Programme (SIP) Python
Batch: June – July 2019
Job Title: Business Analyst Associate (Intern)
Institution: Lovely Professional University, Punjab

DECLARATION

I here by declare that the project report entitled of "**Car Price Prediction**" submitted by me to **HENRY HARVIN EDUCATION INDIA** is a record of bonafide project work carried out by me under the guidance of MS. POOJA GUPTA (Mentor). This project is an original report with references taken from websites and help from mentors and teachers.

Abhijit Roy

Date: 28 Jul 2019

SIP – Python

<u>**Acknowledgements**</u>

In the accomplishment of this project successfully, many people have best owned upon me their blessings and the heart pledged support, this time I am utilizing to thank all the people who have been concerned with this project. Primarily I would thank god for being able to complete this project with success. Then I would like to thank my teachers MR. DHIRAJ UPADHYAYA and MR. ANIL JADON whose valuable guidance has been the ones that helped me patch this project and make it full proof success.

Their suggestions and instructions have served as the major contributor towards the completion of the project. I would like to thank my mentor MS. POOJA GUPTA (Mentor) for giving me this golden opportunity.

Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance has been very helpful in various phases of the completion of the project. Last but not the least I would like to thank my batchmates who have helped me a lot.

Abhijit Roy

SIP-Python

TABLE OF CONTENTS

Abstract

"Cars are being sold more than ever. Developing countries adopt the lease culture instead of buying a new car due to affordability. Therefore, the rise of used cars sales is exponentially increasing. Car sellers sometimes take advantage of this scenario by listing unrealistic prices owing to the demand. Therefore, arises a need for a model that can assign a price for a vehicle by evaluating its features taking the prices of other cars into consideration. In this paper, we use supervised learning method namely Random Forest to predict the prices of used cars. The model has been chosen after careful exploratory data analysis to determine the impact of each feature on price. A Random Forest was created to train the data. From experimental results, the linear regression accuracy was found out to be 64.377%, and the random forest accuracy was 76.147%. The model can predict the price of cars accurately by choosing the most correlated features."

1. Project Introduction

The prices of cars in the industry is fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So customers buying a new car can be assured of the money they invest to be worthy. But due to the increased price of new cars and the incapability of customers to buy new cars due to the lack of funds, used cars sales are on a global increase.

Predicting the prices of cars is an interesting and much-needed problem to be addressed. Customers can be widely exploited by fixing unrealistic prices for the used cars and many falls into this trap. Therefore, rises an absolute necessity of a car price prediction system to effectively determine the worthiness of the car using a variety of features.

Due to the adverse pricing of cars and the nomadic nature of people in developed countries, the cars are mostly bought on a lease basis, where there is an agreement between the buyer and seller. These cars upon completion of the agreement are resold. So reselling has become an essential part of today’s world.

Given the description of cars, the prediction of cars is not an easy task. There are a variety of features of a car like the age of the car, its make, the origin of the car (the original country of the manufacturer), its mileage (the number of kilometers it has run) and its horsepower. Due to rising fuel prices, fuel economy is also of prime importance. Other factors such as the type of fuel it uses, style, braking system, the volume of its cylinders (measured in cc), acceleration, the number of doors, safety index, size, weight, height, paint color, consumer reviews, prestigious awards won by the car manufacturer. Other options such as sound system, air conditioner, power steering, cosmic wheels, GPS navigator all may influence the price as well.

2. Project Data Introduction

This project is based on Predictive Analysis. This is a Python-based Project. This project was created via Spyder 3.3.5. IDE (Integrated Development Environment) using Python 3.7.3 and Ipython Console 7.4.0. The final outcome of this project is saved as a Jupyter Notebook v7.8.0. The libraries of python used in this project are:

1. NumPy
2. Pandas
3. Matplotlib
4. Seaborn
5. Statsmodels
6. Sci-kit Learn

This project is based on a data set provided by the teachers via GITHUB. The data used in the project is continuous, and hence, we are using “LINEAR REGRESSION” and “RANDOM FOREST REGRESSION” for predicting our data.

Here, the **target variable** is “PRICE”.

Data Set Dictionary:

Name of Column	**Description**	**Type**
Car_ID	Unique id of each observation	Numeric
Symbolling	Its assigned insurance risk rating, A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.	Categorical
Company	Name of car company	Categorical
Fuel Type	Car fuel type i.e gas or diesel	Categorical
Aspiration	Aspiration used in a car	Categorical
Door Number	No of doors in a car	Categorical

Column Name	Description	Type
Car Body	Body of the car	Categorical
Drive wheel	Type of Drive Wheel	Categorical
Engine Location	Location of car engine	Categorical
Wheel Base	Wheel Base of the car	Categorical
Car Length	Length of car	Numeric
Car Width	Width of car	Numeric
Car Height	Height of car	Numeric
Car Volume	Volume of Car	Numeric
Curb weight	The weight of a car without occupants	Numeric
Engine Type	Type of engine in the car	Categorical
Cylinder Number	No of cylinders in the car	Categorical
Engine Size	Size of engine in the car	Numeric
Fuel System	Fuel system of car	Categorical
Bore Ratio	Bore ratio of the car	Numeric
Stroke	Stroke or volume inside the engine	Numeric
Compression Ratio	Compression Ratio of car	Numeric
Horse Power	Horse power of the car	Numeric
Peak RPM	Peak rpm of the car	Numeric
City MPG	City mpg of the car	Numeric
Highway MPG	Highway mpg of the car	Numeric
Fuel Economy	Fuel economy of the car	Numeric
Cars Range	Car Category	Categorical
Price (Dependent Variable)	Price of the car	Numeric

Data Set Size: 206 rows, 29 columns

Categorical Variables: [Company, carsrange, Symbolling, fueltype, enginetype, carbody, doornumber, enginelocation, fuelsystem, cylindernumber, aspiration, drivewheel] = 12 features

Numeric Variables: [Car_ID, carlength, carwidth, carheight, carvolume, curbweight, Horsepower, Bore Ratio, Compression Ratio, Highway miles per gallon (mpg), Engine Size, Stroke, City Miles per gallon (mpg), Fuel economy, Peak Revolutions per Minute (rpm), Wheel Base, Price] = 17 Features

Here is the Data Set of the GITHUB Repository as follows:

https://github.com/jitroy160/Final_Projects/blob/master/Final_Projects/CarPrice_Assignment.csv

3. Exploratory Data Analysis (EDA)

In statistics, **exploratory data analysis** (**EDA**) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. Exploratory data analysis was promoted by many to encourage statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments. EDA is different from initial data analysis (IDA), which focuses more narrowly on checking assumptions required for model fitting and hypothesis testing, and handling missing values and making transformations of variables as needed. EDA encompasses IDA.
In this project, we used matplotlib, seaborn for EDA using python 3.7.3. It is as follows:

3.1. Data Understanding

At first, I imported all the libraries initially required for EDA. Then, I imported the file saved in the repository link and displayed its data. The source code and output are:

```
In [1]: #Project 1 --- Car_Price_Prediction

In [22]: #1.Understanding the data

         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         import statsmodels.api as sm
         %matplotlib inline

In [23]: cars=pd.read_csv('C://Users//hp//Desktop//Henry Harvin//Assignment #6//CarPrice_Assignment.csv')

In [24]: cars.info()
```

```
In [24]: cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
car_ID              205 non-null int64
symboling           205 non-null int64
CarName             205 non-null object
fueltype            205 non-null object
aspiration          205 non-null object
doornumber          205 non-null object
carbody             205 non-null object
drivewheel          205 non-null object
enginelocation      205 non-null object
wheelbase           205 non-null float64
carlength           205 non-null float64
carwidth            205 non-null float64
carheight           205 non-null float64
curbweight          205 non-null int64
enginetype          205 non-null object
cylindernumber      205 non-null object
enginesize          205 non-null int64
fuelsystem          205 non-null object
boreratio           205 non-null float64
stroke              205 non-null float64
compressionratio    205 non-null float64
horsepower          205 non-null int64
peakrpm             205 non-null int64
citympg             205 non-null int64
highwaympg          205 non-null int64
price               205 non-null float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.7+ KB
```

Then, I used the describe() function to study the summary of the data(min, max, no of values etc.) The source code and output are the following:

```
In [27]: cars.describe(percentiles=[0.25,0.5,0.75,1]).round(2)
```

Out[27]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	cityn
count	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205
mean	103.00	0.83	98.76	174.05	65.91	53.72	2555.57	126.91	3.33	3.26	10.14	104.12	5125.12	25
std	59.32	1.25	6.02	12.34	2.15	2.44	520.68	41.64	0.27	0.31	3.97	39.54	476.99	6
min	1.00	-2.00	86.60	141.10	60.30	47.80	1488.00	61.00	2.54	2.07	7.00	48.00	4150.00	13
25%	52.00	0.00	94.50	166.30	64.10	52.00	2145.00	97.00	3.15	3.11	8.60	70.00	4800.00	19
50%	103.00	1.00	97.00	173.20	65.50	54.10	2414.00	120.00	3.31	3.29	9.00	95.00	5200.00	24
75%	154.00	2.00	102.40	183.10	66.90	55.50	2935.00	141.00	3.58	3.41	9.40	116.00	5500.00	30
100%	205.00	3.00	120.90	208.10	72.30	59.80	4066.00	326.00	3.94	4.17	23.00	288.00	6600.00	49
max	205.00	3.00	120.90	208.10	72.30	59.80	4066.00	326.00	3.94	4.17	23.00	288.00	6600.00	49

3.2. Data Cleaning and Preparation

Data Cleaning, as the name suggests, is to clean the data of any irregularities. By performing this step, we prepare our data for analysis. For this, we check for any spelling errors, empty values and duplicate values. The source code and output are:

```
In [ ]: 2. Data Cleaning and Preparation
```

```
In [31]: #splitting the car name column and creating new columns  company and car model
         cars=cars.join(cars['CarName'].str.split(' ',1,expand=True).rename(columns={0:'Company',1:'CarModel'}))
         cars.head()
```

Out[31]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	boreratio	stroke	compressionratio
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	3.47	2.68	9.0
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	3.47	2.68	9.0
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	2.68	3.47	9.0
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	3.19	3.40	10.0
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	3.19	3.40	8.0

5 rows × 28 columns

Now, we will check the columns created. Once this is done, we will move on to correcting the misspelled values.

```
In [12]: #checking the columns created
         cars['Company'].unique()

Out[12]: array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
                'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
                'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
                'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
                'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)

In [13]: cars['CarModel'].unique()

Out[13]: array(['giulia', 'stelvio', 'Quadrifoglio', '100 ls', '100ls', 'fox',
                '5000', '4000', '5000s (diesel)', '320i', 'x1', 'x3', 'z4', 'x4',
                'x5', 'impala', 'monte carlo', 'vega 2300', 'rampage',
                'challenger se', 'd200', 'monaco (sw)', 'colt hardtop',
                'colt (sw)', 'coronet custom', 'dart custom',
                'coronet custom (sw)', 'civic', 'civic cvcc', 'accord cvcc',
                'accord lx', 'civic 1500 gl', 'accord', 'civic 1300', 'prelude',
                'civic (auto)', 'MU-X', 'D-Max ', 'D-Max V-Cross', 'xj', 'xf',
                'xk', 'rx3', 'glc deluxe', 'rx2 coupe', 'rx-4', '626', 'glc',
                'rx-7 gs', 'glc 4', 'glc custom l', 'glc custom',
                'electra 225 custom', 'century luxus (sw)', 'century', 'skyhawk',
                'opel isuzu deluxe', 'skylark', 'century special',
                'regal sport coupe (turbo)', 'cougar', 'mirage', 'lancer',
                'outlander', 'g4', 'mirage g4', 'montero', 'pajero', 'versa',
                'gt-r', 'rogue', 'latio', 'titan', 'leaf', 'juke', 'note',
                'clipper', 'nv200', 'dayz', 'fuga', 'otti', 'teana', 'kicks',
                '504', '304', '504 (sw)', '604sl', '505s turbo diesel', 'fury iii',
                'cricket', 'satellite custom (sw)', 'fury gran sedan', 'valiant',
                'duster', 'macan', 'panamera', 'cayenne', 'boxter', '12tl',
                '5 gtl', '99e', '99le', '99gle', None, 'dl', 'brz', 'baja', 'r1',
                'r2', 'trezia', 'tribeca', 'corona mark ii', 'corona',
                'corolla 1200', 'corona hardtop', 'corolla 1600 (sw)', 'carina',
                'mark ii', 'corolla', 'corolla liftback', 'celica gt liftback',
                'corolla tercel', 'corona liftback', 'starlet', 'tercel',
                'cressida', 'celica gt', 'rabbit', '1131 deluxe sedan',
                'model 111', 'type 3', '411 (sw)', 'super beetle', 'dasher',
                'rabbit custom', '145e (sw)', '144ea', '244dl', '245', '264gl',
                'diesel', '246'], dtype=object)
```

Now, we will rename the misspelled values to correct names. Also, the names ‘nissan’ and ‘Nissan’ are the same company. So, we are going to correct that by converting all values to same case(lower case). From the company column, we can see some spelling mistakes. Lets correct them.

```
In [33]: #replacing incorrect values to correct values
         cars['Company'].replace('maxda','mazda',inplace=True)
         cars['Company'].replace(['vokswagen','vw'],'volkswagen',inplace=True)
         cars['Company'].replace('porcshce','porsche',inplace=True)
         cars['Company'].replace('toyouta','toyota',inplace=True)

In [35]: #converting all the string data to lower to avoid any case difference errors
         cars['Company']=cars['Company'].str.lower()

In [ ]:

In [36]: #checking for duplicate values
         cars.loc[cars.duplicated()]

Out[36]:
```

car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	boreratio	stroke	compressionratio	ho

0 rows × 28 columns

```
In [14]: #checking columns
         column_names=cars.columns.tolist()
         print('Columns', column_names)

Columns ['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration', 'doornumber', 'carbody', 'drivewheel', 'enginelocation', 
'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype', 'cylindernumber', 'enginesize', 'fuelsystem', 'b
oreratio', 'stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price', 'Company', 'CarModel']
```

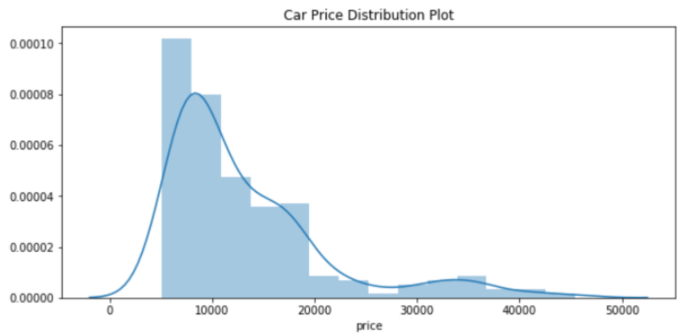
Now, we can see that there are no duplicate values in our data frame. Our data is officially clean. It's time for the final step of EDA: Visualization.

3.3. Visualization

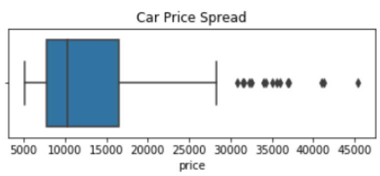
Visualization refers to the term that gives a picture to our information. We can describe our data by drawing graphs and charts to check different parameters that, in the end, might help us choose features for our analysis. In python, we use matplotlib and seaborn for visualization. These two libraries are efficient enough to give us an output that gives us an idea about our data set. The source code and output are :

In [38]:
```
#3. Visualizing the Data

plt.figure(figsize=(10,10)) #plot size according to scale 1 unit=72 pixels
plt.subplot(2,1,1) #2 rows, 1 column and index=1
plt.title('Car Price Distribution Plot') #title for the chart
sns.distplot(cars['price']) #distplot for price of cars
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x19e2a8aa128>



In [39]:
```
plt.subplot(2,1,2)
plt.title('Car Price Spread')
sns.boxplot(cars['price']) #distribution of price in the data
plt.show()
```



The distribution plot appears to be right-skewed. It means that most of the values in the data frame are low, compared to the maximum value. Also, the data points are spread out far from the mean, which means that the data has high variance.

Now, we need to visualize our features to select the most significant of them for our analysis for better results. Hence, let's first visualize the categorical variables.

Categorical Variables: [Company, carsrange, Symboling, fueltype, enginetype, carbody, doornumber, enginelocation, fuelsystem, cylindernumber, aspiration, drivewheel] = 12 features.

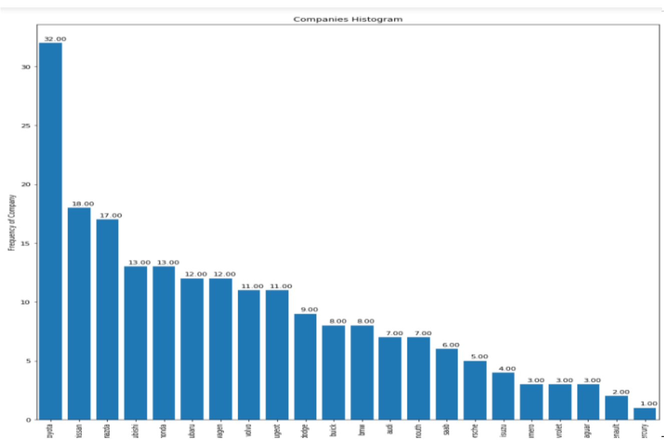
1. Car Company

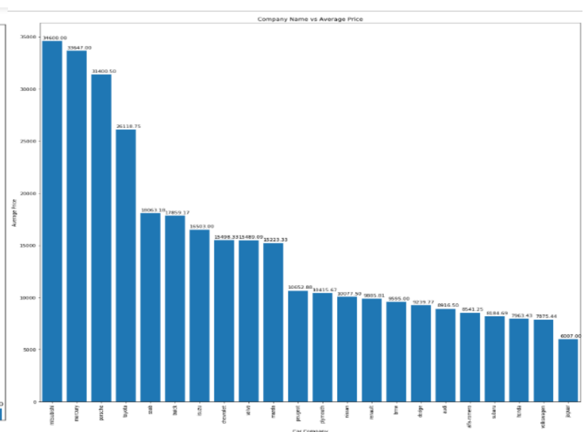
First, lets check how many cars we have from each company. Then, we'll check the average price range for each company. Car companies are one of the most important feature while purchasing a car. The source code and output are :

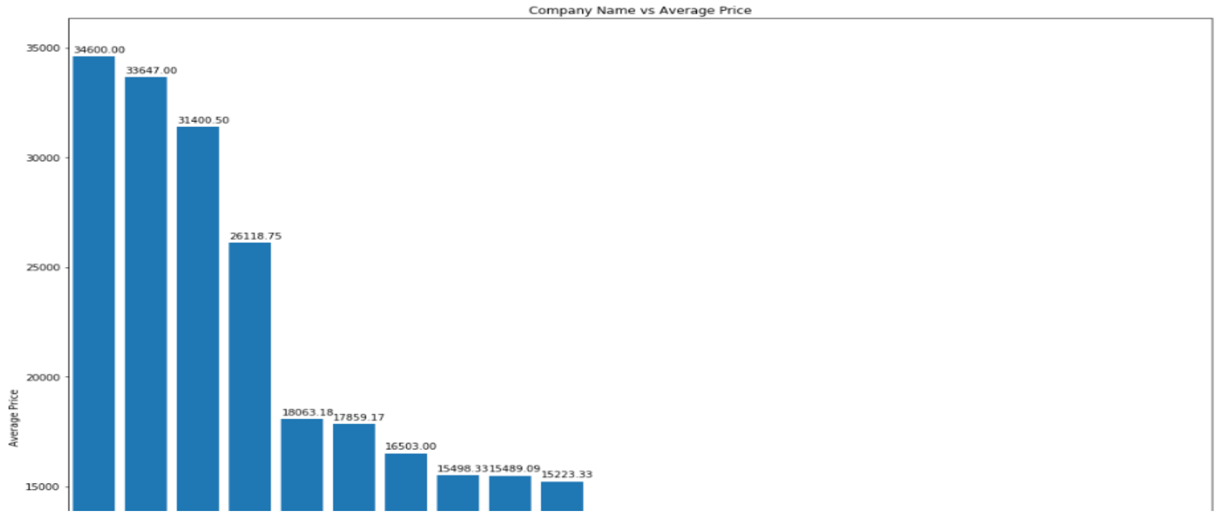
```
In [ ]: #1. Car Company
```

```
In [40]: plt.figure(figsize=(30, 15))

         #plot 1
         plt.subplot(1,2,1)
         plt1 = cars['Company'].value_counts().plot('bar')
         plt.title('Companies Histogram')
         plt1.set(xlabel = 'Car Company', ylabel='Frequency of Company')
         xs=cars['Company'].unique()
         ys=cars['Company'].value_counts()
         plt.bar(xs,ys)
         for x,y in zip(xs,ys):
             label = "{:.2f}".format(y)
             plt.annotate(label,(x,y), textcoords="offset points",xytext=(5,5),ha='center')
         plt.xticks(xs)
```

```
In [41]: plt.figure(figsize=(30, 15))

         #plot 2
         plt.subplot(1,2,2)
         company_vs_price = pd.DataFrame(cars.groupby(['Company'])['price'].mean().sort_values(ascending = False))
         plt2=company_vs_price.index.value_counts().plot('bar')
         plt.title('Company Name vs Average Price')
         plt2.set(xlabel='Car Company', ylabel='Average Price')
         xs=company_vs_price.index
         ys=company_vs_price['price'].round(2)
         plt.bar(xs,ys)
         for x,y in zip(xs,ys):
             label = "{:.2f}".format(y)
             plt.annotate(label,(x,y), textcoords="offset points",xytext=(5,5),ha='center')
         plt.xticks(xs)
         plt.tight_layout()
         plt.show()
```





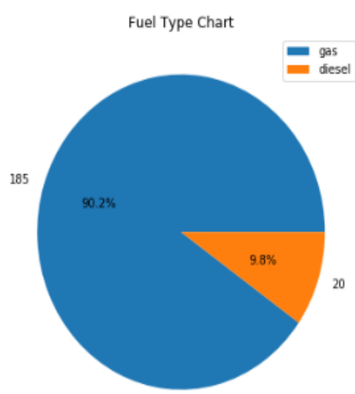


Toyota seems to be the most favored Company. Also, coincidentally, Toyota's average car price is highest as well.

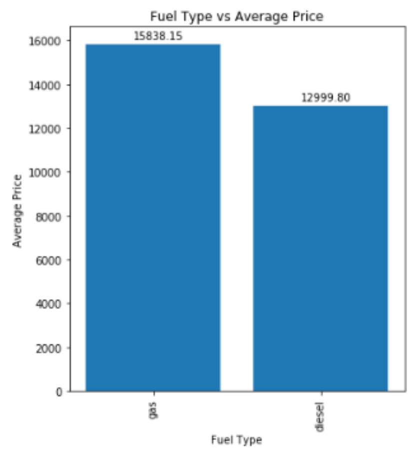
2. Fuel Type

Now, lets check which fuel type cars are preferred more, gas (petrol) or diesel. It is another important feature while purchasing a car. The source code and output are:

```
In [42]: #2. Fuel Type

         plt.figure(figsize=(25, 6))

         #plot 1
         plt.subplot(1,2,1)
         plt.title('Fuel Type Chart')
         labels=cars['fueltype'].unique()
         plt3 = cars['fueltype'].value_counts().tolist()
         plt.pie(plt3,labels=plt3, autopct='%1.1f%%')
         plt.legend(labels)

Out[42]: <matplotlib.legend.Legend at 0x19e2ad64e10>
```



```
In [43]: plt.figure(figsize=(10, 6))

         #plot 2
         plt.subplot(1,2,2)
         fuel_vs_price = pd.DataFrame(cars.groupby(['fueltype'])['price'].mean().sort_values(ascending = False))
         plt4=fuel_vs_price.index.value_counts().plot('bar')
         plt.title('Fuel Type vs Average Price')
         plt4.set(xlabel='Fuel Type', ylabel='Average Price')
         xs=fuel_vs_price.index
         ys=fuel_vs_price['price'].round(2)
         plt.bar(xs,ys)
         for x,y in zip(xs,ys):
             label = "{:.2f}".format(y)
             plt.annotate(label,(x,y), textcoords="offset points",xytext=(5,5),ha='center')
         plt.xticks(xs)
         plt.tight_layout()
         plt.show()
```



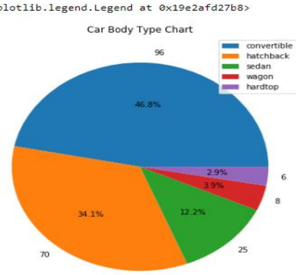
From the pie chart, we can see that gas cars more than diesel cars and subsequently, they cost more as well.

3. Car Body Type

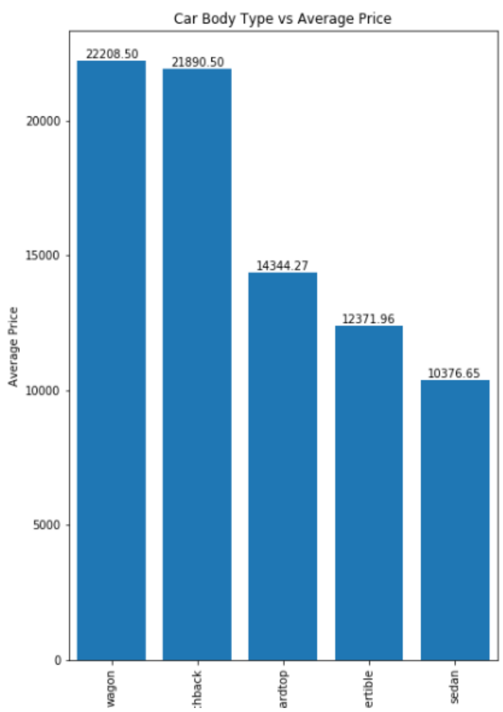
There are different types of car bodies, all made for different purposes. Hence, it is important to know their distribution as well. So, let's check which car body is most common in the data . The source code and output are:

```
In [44]: #3. Car Body Type

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

         #plot 1
         plt.subplot(1,2,1)
         plt.title('Car Body Type Chart')
         labels=cars['carbody'].unique()
         plt5 = cars['carbody'].value_counts().tolist()
         plt.pie(plt5, labels=plt5, autopct='%1.1f%%')
         plt.legend(labels, loc=1)
```

```
Out[44]: <matplotlib.legend.Legend at 0x19e2afd27b8>
```



```
In [45]: plt.figure(figsize=(15,10))

#plot 2
plt.subplot(1,2,2)
car_vs_price = pd.DataFrame(cars.groupby(['carbody'])['price'].mean().sort_values(ascending = False))
plt6=car_vs_price.index.value_counts().plot('bar')
plt.title('Car Body Type vs Average Price')
plt6.set(xlabel='Car Body Type', ylabel='Average Price')
xs=car_vs_price.index
ys=car_vs_price['price'].round(2)
plt.bar(xs,ys)
for x,y in zip(xs,ys):
    label = "{:.2f}".format(y)
    plt.annotate(label,(x,y), textcoords="offset points",xytext=(0,2),ha='center')
plt.xticks(xs)
plt.show()
```



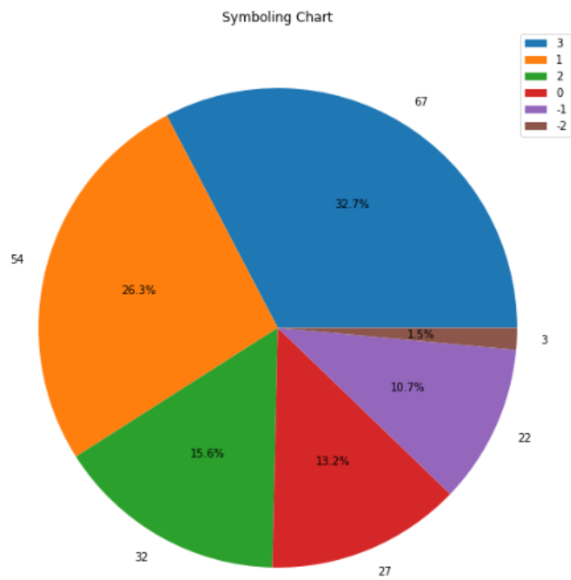
Clearly, the cars with car body convertible are more in the data. But, we have wagon cars having a higher price range.

4. Symboling

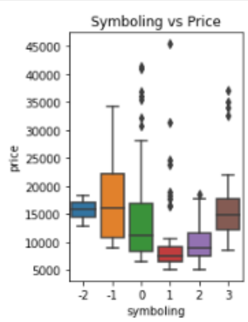
Symboling is a numerical value that describes our car's insurance risk rating. Insurance is a must whenever you purchase a new asset, either for personal use or professional use. Hence, lets check what symboling has to offer. The code and output are:

In [46]:
```
#4. Symboling

plt.figure(figsize=(25,10))

#plot 1
plt.subplot(1,2,1)
plt.title('Symboling Chart')
labels=cars['symboling'].unique()
plt7 = cars['symboling'].value_counts().tolist()
plt.pie(plt7, labels=plt7, autopct='%1.1f%%')
plt.legend(labels, loc=1)
```

Out[46]: <matplotlib.legend.Legend at 0x19e2af2cfd0>



In [47]:
```
#plot 2
plt.subplot(1,2,2)
plt.title('Symboling vs Price')
sns.boxplot(x=cars['symboling'], y=cars['price'])
plt.show()
```



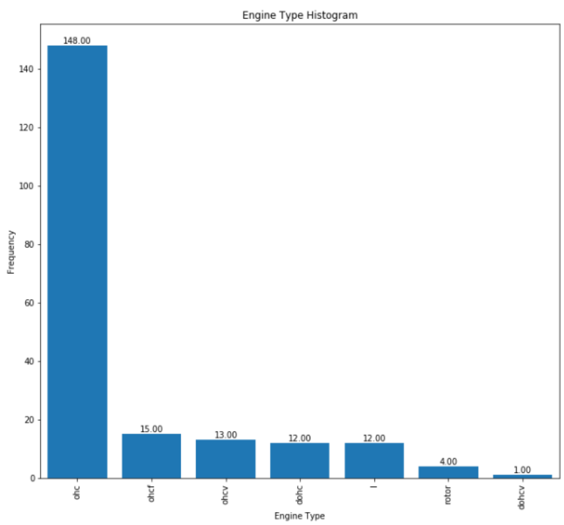
From the pie chart, it is clearly visible that symboling 3 and 1 are dominating the data. But, Cars with symboling -1 are sold at a relatively higher price than the others.

5. Engine Type

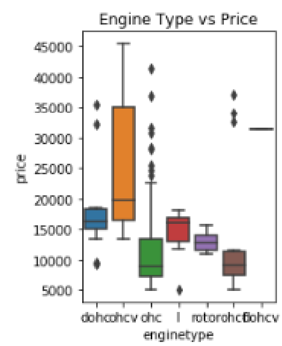
Engine also has a type. Hence, we need to check which engine type would be the best attribute for our car. So, lets check it out. The code and output are:

```
In [48]: #5. Engine Type

         plt.figure(figsize=(25,10))

         #plot 1
         plt.subplot(1,2,1)
         plt8 = cars['enginetype'].value_counts().plot('bar')
         plt.title('Engine Type Histogram')
         plt8.set(xlabel = 'Engine Type', ylabel='Frequency')
         xs=cars['enginetype'].unique()
         ys=cars['enginetype'].value_counts()
         plt.bar(xs,ys)
         for x,y in zip(xs,ys):
             label = "{:.2f}".format(y)
             plt.annotate(label,(x,y), textcoords="offset points",xytext=(0,2),ha='center')
         plt.xticks(xs)
```

```
Out[48]: ([<matplotlib.axis.XTick at 0x19e2b1d3278>,
           <matplotlib.axis.XTick at 0x19e2b1dbbe0>,
           <matplotlib.axis.XTick at 0x19e2b194c18>,
           <matplotlib.axis.XTick at 0x19e2b1fbd68>,
           <matplotlib.axis.XTick at 0x19e2b205240>,
           <matplotlib.axis.XTick at 0x19e2b205710>,
           <matplotlib.axis.XTick at 0x19e2b205be0>],
          <a list of 7 Text xticklabel objects>)
```




```
In [49]: #plot 2
         plt.subplot(1,2,2)
         plt.title('Engine Type vs Price')
         sns.boxplot(x=cars['enginetype'], y=cars['price'])
         plt.show()
```

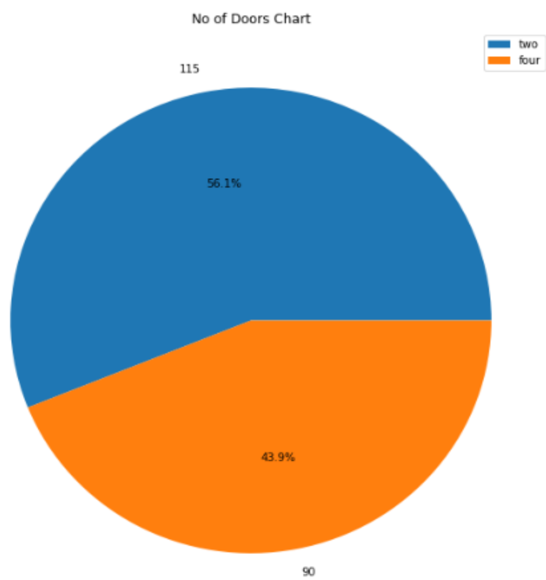


We can see that '*ohc'* engine type is the most occurring engine type. But, *'ohcv'* has a much higher price range, as depicted by the boxplot.

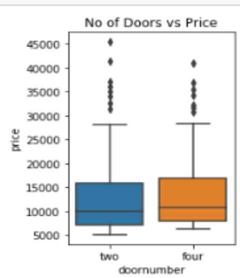
6. Door Number

Number of doors in the car doesn't seem to be a very influencing factor, does it? However, for the analysis, we should leave no stone unturned. Hence, lets have a look at what the door number has to offer.

```
In [50]: #6. Door Number

         plt.figure(figsize=(25,10))

         #plot 1
         plt.subplot(1,2,1)
         labels=cars['doornumber'].unique()
         plt8 = cars['doornumber'].value_counts().tolist()
         plt.title('No of Doors Chart')
         plt.pie(plt8, labels=plt8, autopct='%1.1f%%')
         plt.legend(labels, loc=1)

Out[50]: <matplotlib.legend.Legend at 0x19e2b63a9e8>
```




```
In [51]: #plot 2
         plt.subplot(1,2,2)
         plt.title('No of Doors vs Price')
         sns.boxplot(x=cars['doornumber'], y=cars['price'])
         plt.show()
```



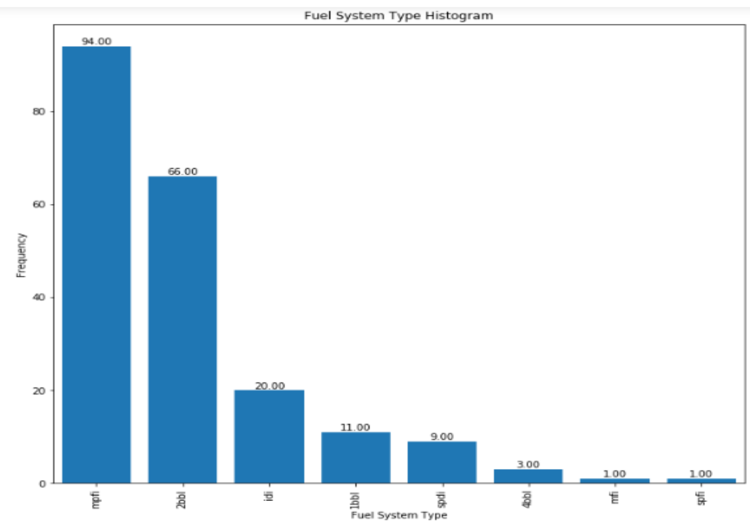
Cars with two doors are preferred over cars with four doors. However, it is not affecting the price much, as their distribution is almost the same.

7. Fuel System

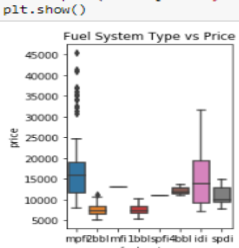
Fuel System is another technical term that most people are not aware of, when they purchase a car. But, lets check what brings to the table. The code and output are:

In [52]:
```
#7. Fuel System

plt.figure(figsize=(25,10))

#plot 1
plt.subplot(1,2,1)
plt10 = cars['fuelsystem'].value_counts().plot('bar')
plt.title('Fuel System Type Histogram')
plt10.set(xlabel = 'Fuel System Type', ylabel='Frequency')
xs=cars['fuelsystem'].unique()
ys=cars['fuelsystem'].value_counts()
plt.bar(xs,ys)
for x,y in zip(xs,ys):
    label = "{:.2f}".format(y)
    plt.annotate(label,(x,y), textcoords="offset points",xytext=(0,2),ha='center')
plt.xticks(xs)
```

Out[52]:
```
([<matplotlib.axis.XTick at 0x19e2b6aeef0>,
  <matplotlib.axis.XTick at 0x19e2b6ae198>,
  <matplotlib.axis.XTick at 0x19e2b68d780>,
  <matplotlib.axis.XTick at 0x19e2b668c88>,
  <matplotlib.axis.XTick at 0x19e2b668ac8>,
  <matplotlib.axis.XTick at 0x19e2b668160>,
  <matplotlib.axis.XTick at 0x19e2b1bc320>,
  <matplotlib.axis.XTick at 0x19e2b60d9e8>],
 <a list of 8 Text xticklabel objects>)
```



In [53]:
```
#plot 2
plt.subplot(1,2,2)
plt.title('Fuel System Type vs Price')
sns.boxplot(x=cars['fuelsystem'], y=cars['price'])
plt.show()
```

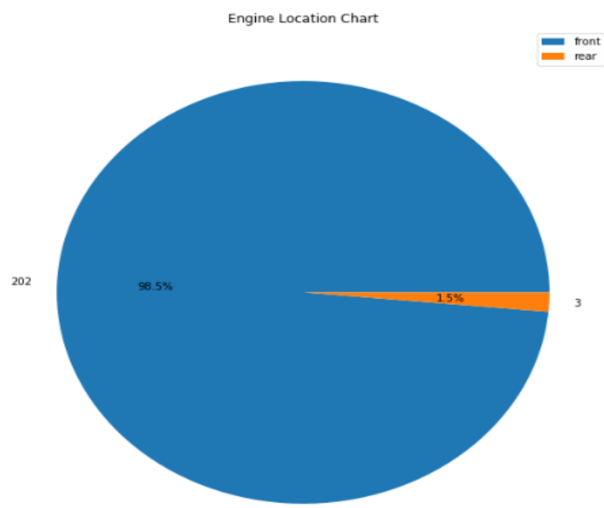


From the plots, we can say that mpfi is the most preferred fuel system. But, idi is having a high price range.

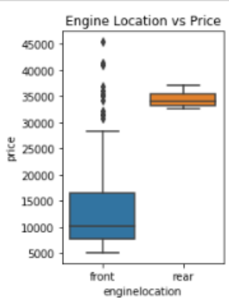
8. Engine Location

Engine Location is a term most people would not think of, while purchasing a car. Since we are doing an analysis, we should run by it, just to be sure if it affects price at a higher rate or not. The code and output are:

```
In [54]: #8. Engine Location

         plt.figure(figsize=(25,10))

         #plot 1
         plt.subplot(1,2,1)
         labels=cars['enginelocation'].unique()
         plt9 = cars['enginelocation'].value_counts().tolist()
         plt.title('Engine Location Chart')
         plt.pie(plt9, labels=plt9, autopct='%1.1f%%')
         plt.legend(labels, loc=1)

Out[54]: <matplotlib.legend.Legend at 0x19e2af8c048>
```




```
In [55]: #plot 2
         plt.subplot(1,2,2)
         plt.title('Engine Location vs Price')
         sns.boxplot(x=cars['enginelocation'], y=cars['price'])
         plt.show()
```

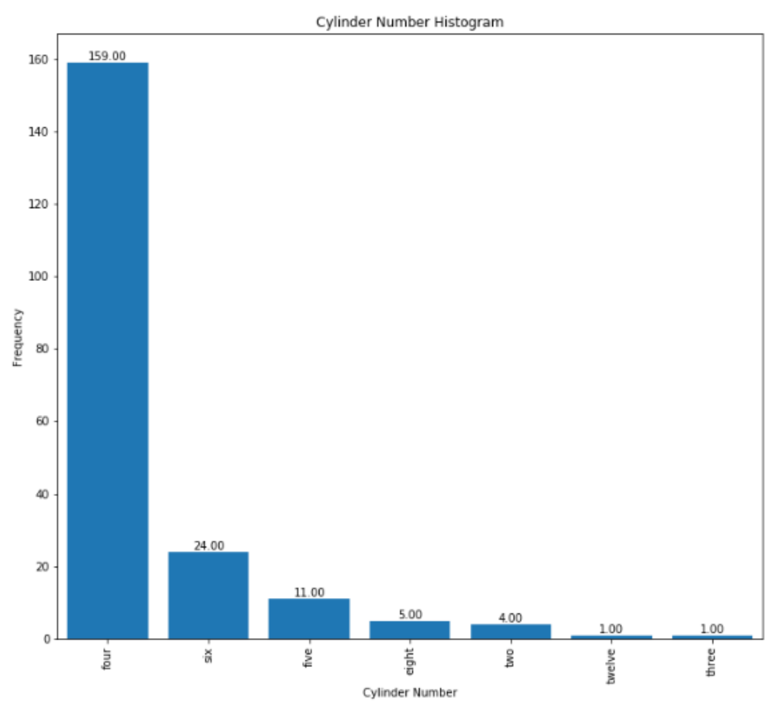


As expected, engine location is not a significant variable for price. The boxplot shows it and the pie chart shows the domination of front engines.

9. Cylinder Number

No of cylinders can be another important factor, because it increases the power of the car, its stroke and bore ratio respectively. A car can have upto 12 cylinders ! So, lets check what our data tells us. The code and output are:

```
In [56]: #9. Cylinder Number '

         plt.figure(figsize=(25,10))

         #plot 1
         plt.subplot(1,2,1)
         plt11 = cars['cylindernumber'].value_counts().plot('bar')
         plt.title('Cylinder Number Histogram')
         plt11.set(xlabel = 'Cylinder Number', ylabel='Frequency')
         xs=cars['cylindernumber'].unique()
         ys=cars['cylindernumber'].value_counts()
         plt.bar(xs,ys)
         for x,y in zip(xs,ys):
             label = "{:.2f}".format(y)
             plt.annotate(label,(x,y), textcoords="offset points",xytext=(0,2),ha='center')
         plt.xticks(xs)

Out[56]: ([<matplotlib.axis.XTick at 0x19e2b2a9e48>,
           <matplotlib.axis.XTick at 0x19e2b2a9780>,
           <matplotlib.axis.XTick at 0x19e2b66c9e8>,
           <matplotlib.axis.XTick at 0x19e2c1d2908>,
           <matplotlib.axis.XTick at 0x19e2c1d2da0>,
           <matplotlib.axis.XTick at 0x19e2c1dc2b0>,
           <matplotlib.axis.XTick at 0x19e2c1dc780>],
          <a list of 7 Text xticklabel objects>)
```

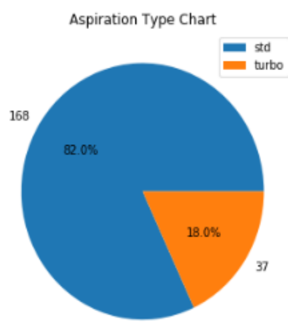


From the plots, we can infer that cars with four cylinders are the most favorable ones. Though, cars with eight cylinders have the highest car range.

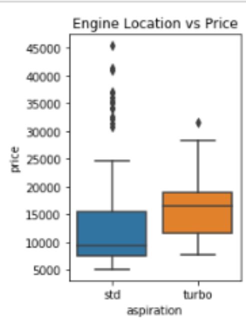
10. Aspiration

Aspiration is another one of those features that people don't know about. Still, we need to check for aspiration as well. Remember, leave no stone unturned ! The code and output are:

```
In [58]: #10. Aspiration

         plt.figure(figsize=(15,5))

         #plot 1
         plt.subplot(1,2,1)
         labels=cars['aspiration'].unique()
         plt12 = cars['aspiration'].value_counts().tolist()
         plt.title('Aspiration Type Chart')
         plt.pie(plt12, labels=plt12, autopct='%1.1f%%')
         plt.legend(labels, loc=1)

Out[58]: <matplotlib.legend.Legend at 0x19e2c2f2ef0>
```




```
In [59]: #plot 2
         plt.subplot(1,2,2)
         plt.title('Engine Location vs Price')
         sns.boxplot(x=cars['aspiration'], y=cars['price'])
         plt.show()
```



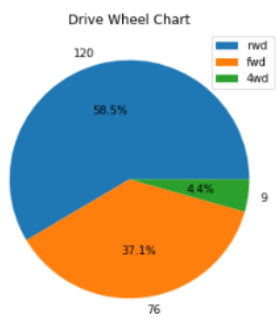
Aspiration is much more common than turbo, that is why it has been distributed better than turbo, which, in turn has higher price range.

11. Drivewheel

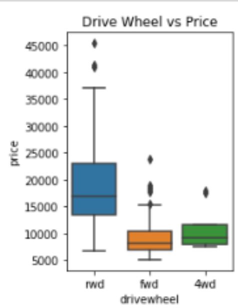
Drivewheel is another uncommon characteristic of cars that usually gets ignored when price is talked about. However, little or high, we need to check if it affects price in high rate or not. Hence, the code and output are:

In [60]:
```
#11. Drivewheel

plt.figure(figsize=(15,5))

#plot 1
plt.subplot(1,2,1)
labels=cars['drivewheel'].unique()
plt13 = cars['drivewheel'].value_counts().tolist()
plt.title('Drive Wheel Chart')
plt.pie(plt13, labels=plt13, autopct='%1.1f%%')
plt.legend(labels, loc=1)
```

Out[60]: `<matplotlib.legend.Legend at 0x19e2c4fa828>`



In [61]:
```
#plot 2
plt.subplot(1,2,2)
plt.title('Drive Wheel vs Price')
sns.boxplot(x=cars['drivewheel'], y=cars['price'])
plt.show()
```

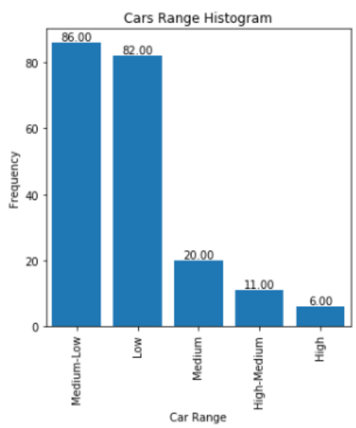


Most cars have *rwd* drivewheel, but it is not affecting price on a higher scale. As expected, it is not that significant variable, but can be taken for analysis.

12. Cars Range

This is one variable we didn't have from the starting. Now, lets create a cars range column and check our variable. Code and Output is:

```
In [67]: #5. Creating a Categorical Variable - Car Class

         carsrange=[]
         for i in cars['price']:
             if (i>0 and i<9000): carsrange.append('Low')
             elif (i>9000 and i<18000): carsrange.append('Medium-Low')
             elif (i>18000 and i<27000): carsrange.append('Medium')
             elif(i>27000 and i<36000): carsrange.append('High-Medium')
             else : carsrange.append('High')
         cars['carsrange']=carsrange
         cars['carsrange'].unique()

Out[67]: array(['Medium-Low', 'Medium', 'High-Medium', 'High', 'Low'], dtype=object)

In [68]: #plot
         plt.figure(figsize=(5,5))
         plt14 = cars['carsrange'].value_counts().plot('bar')
         plt.title('Cars Range Histogram')
         plt14.set(xlabel = 'Car Range', ylabel='Frequency')
         xs=cars['carsrange'].unique()
         ys=cars['carsrange'].value_counts()
         plt.bar(xs,ys)
         for x,y in zip(xs,ys):
             label = "{:.2f}".format(y)
             plt.annotate(label,(x,y), textcoords="offset points",xytext=(0,2),ha='center')
         plt.xticks(xs)

Out[68]: ([<matplotlib.axis.XTick at 0x19e2c347b38>,
           <matplotlib.axis.XTick at 0x19e2c347470>,
           <matplotlib.axis.XTick at 0x19e2c71dac8>,
           <matplotlib.axis.XTick at 0x19e2c22de10>,
           <matplotlib.axis.XTick at 0x19e2c24c2e8>],
          <a list of 5 Text xticklabel objects>)
```



With this, our **categorical visualization is complete**. Now, let's move on to the numeric data.

Numeric Variables: [Car_ID, carlength, carwidth, carheight, carvolume, curbweight, Horsepower, Bore Ratio, Compression Ratio, Highway miles per gallon (mpg), Engine Size, Stroke, City Miles per gallon (mpg), Fuel economy, Peak Revolutions per Minute (rpm), Wheel Base, Price] = 17 Features

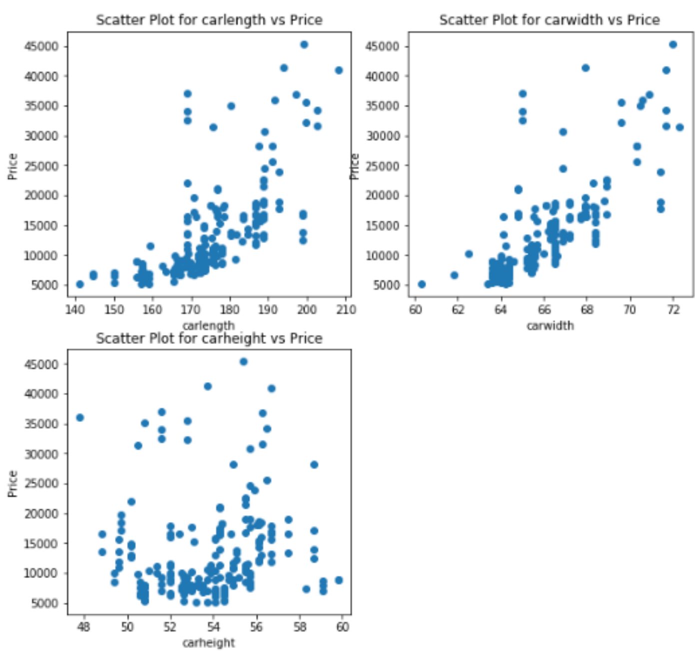
For numeric data, we will be using scatterplot and pair plots to understand the distribution of data points better. Hence, lets make a function scatterplot to use when we need to.

Now, lets make some scatterplots. Let's start with car length, width and height.

```
In [62]: def scatterplot(df,var):

             plt.scatter(df[var],df['price'])
             plt.xlabel(var); plt.ylabel('Price')
             plt.title('Scatter Plot for '+var+' vs Price')
```

1. Car Length, Width and Height

```
In [63]: #1. Car Length, Width and Height

         plt.figure(figsize=(10,20))
         plt.subplot(4,2,1)
         scatterplot(cars,'carlength')
         plt.subplot(4,2,2)
         scatterplot(cars,'carwidth')
         plt.subplot(4,2,3)
         scatterplot(cars,'carheight')
         plt.show()
         plt.tight_layout()
```



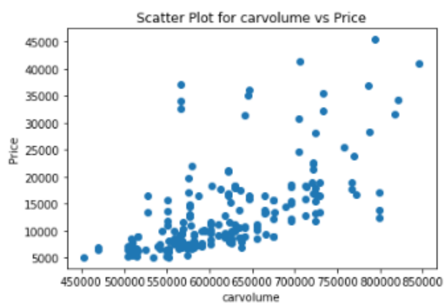

```
<Figure size 432x288 with 0 Axes>
```

Car length and width seem to have a significant trend with price, whereas car height is not that influencing on it. Now, let's combine these to make a new variable *carvolume*.

2. Car Volume

Initially, car volume is not a part of our data. But, after studying it so much, we felt like adding it to the columns. So, let's add car volume to our dataframe and make a scatterplot for it. The code and output are:

In [64]:
```
#2. Creating a new variable- Car Volume

cars['carvolume']=cars['carlength']*cars['carwidth']*cars['carheight']
cars['carvolume'].unique()
scatterplot(cars,'carvolume')
```

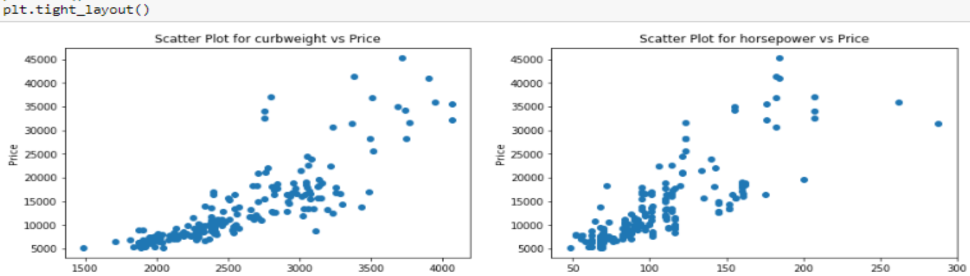


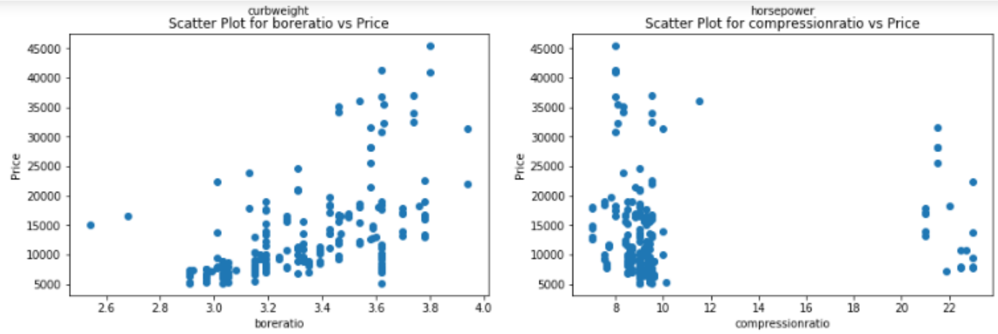
Car volume also seems to have a trend with price.

3. Curbweight, Horsepower, Bore ratio and Compression Ratio

All these features are significant in deciding the power of the car and as a result, its efficiency. Hence, we need to plot them as well. The code and output are:

In [65]:
```
#3. Curb Weight (Effective Weight of Car including its internal components), HorsePower, Boreratio, and Compression Ratio

plt.figure(figsize=(15,20))
plt.subplot(4,2,1)
scatterplot(cars,'curbweight')
plt.subplot(4,2,2)
scatterplot(cars,'horsepower')
plt.subplot(4,2,3)
scatterplot(cars,'boreratio')
plt.subplot(4,2,4)
scatterplot(cars,'compressionratio')
plt.show()
plt.tight_layout()
```





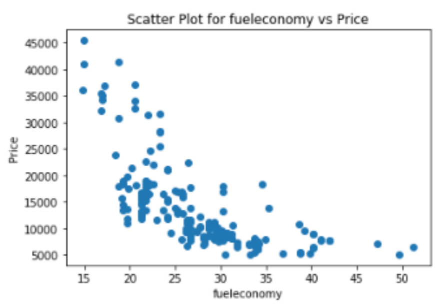

```
<Figure size 432x288 with 0 Axes>
```

Clearly, Curb Weight, Horsepower and Bore Ratio have a significant trend with price. Compression Ratio does not affect price that much.

4. Fuel Economy

We did not have fuel economy as an original column. We created this column to get another important variable for our analysis. Let's create it and plot a scatter plot for the same. The code and output are:

```
In [66]: #4. Creating a new Variable - Fuel Economy

         cars['fueleconomy']=(cars['citympg']*0.55)+(cars['highwaympg']*0.45)
         cars['fueleconomy'].unique()
         scatterplot(cars,'fueleconomy')
```

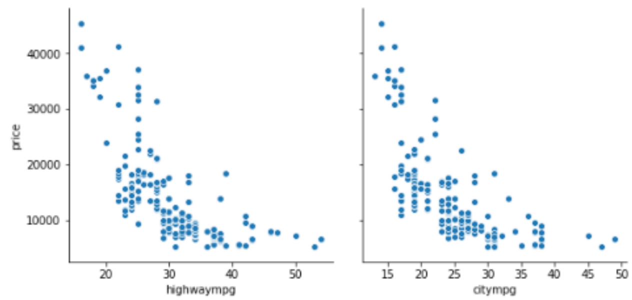


Fuel economy has a nice trend with price. Looks like we did the right thing to include it to our columns.

5. Highway Mileage per Gallon (mpg) and City Miles per Gallon (city mpg)

These two variables are no doubt significant in price prediction, because using them, we derived fuel economy, which turned out to be significant as well. Lets make a pair plot for them. The code and output are:

```
In [69]: #6. Highway mpg and City mpg
         sns.pairplot(cars, x_vars=['highwaympg','citympg'], y_vars='price', height=4, aspect=1, kind='scatter')

Out[69]: <seaborn.axisgrid.PairGrid at 0x19e2c3719e8>
```

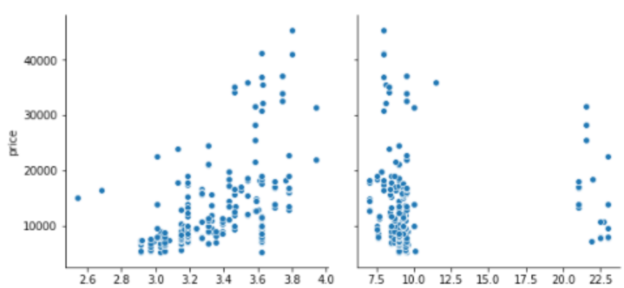


This clearly shows a negative correlation with price for both highway mpg and city mpg. Hence, these two are significant variables.

6. Bore Ratio and Compression Ratio

These two terms define the efficiency of the car. Hence, they need to be plot, but as a pair. In their individual scatter plots, we observed bore ratio as a significant one, but not compression ratio. Lets see what happens here. The code and output are:

```
In [70]: #7. Bore Ratio and Compression Ratio
         sns.pairplot(cars, x_vars=['boreratio','compressionratio'], y_vars='price', height=4, aspect=1, kind='scatter')

Out[70]: <seaborn.axisgrid.PairGrid at 0x19e2c684e10>
```

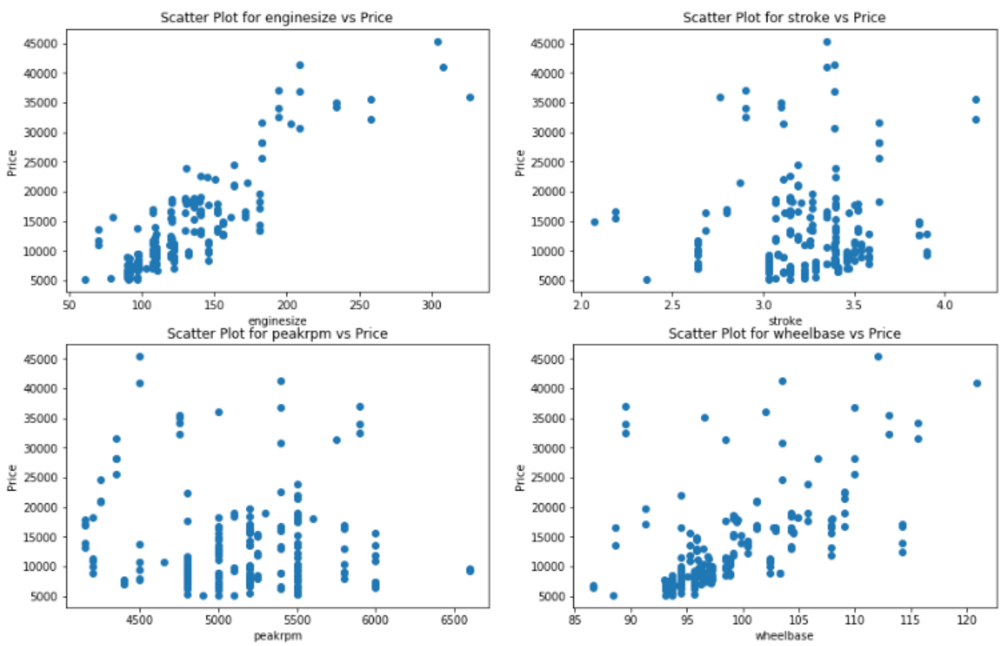


We can see that boreratio is correlated to price significantly but compression ratio is not. Hence, we conclude that compression ratio is not a significant variable.

7. Engine Size, Stroke, RPM and Wheel Base

Lets wind up our numeric visualization with the last remaining 4 variables. We need to see if they influence price in a strong way or not. Lets check it out. The code and output are:

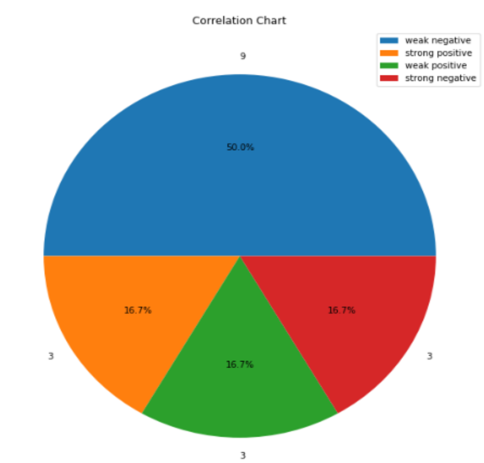
In [71]:
```
#8. Engine Size, Stroke, RPM and Wheelbase

plt.figure(figsize=(15,20))
plt.subplot(4,2,1)
scatterplot(cars,'enginesize')
plt.subplot(4,2,2)
scatterplot(cars,'stroke')
plt.subplot(4,2,3)
scatterplot(cars,'peakrpm')
plt.subplot(4,2,4)
scatterplot(cars,'wheelbase')
plt.show()
plt.tight_layout()
```



Well, we can see a positive correlation for wheelbase and enginesize with price. Hence, the other two are rested.

With this, our **numeric visualization is complete.** To check correlation more specifically, lets make a correlation data set and check our results.

```
In [132]: #Correlation with price(target variable) for numeric data

          corr=cars.corr().round(3).loc['price']
          corr=pd.DataFrame(corr)
          corr
          result=[]

In [133]: for i in corr['price']:
              if (i>-1 and i<-0.4): result.append('strong negative')
              elif (i>-0.4 and i<-0.2): result.append('moderate negative')
              elif (i>-0.2 and i<0): result.append('weak negative')
              elif(i>0 and i<0.2): result.append('weak positive')
              elif(i>0.2 and i<0.5): result.append('moderate positive')
              else : result.append('strong positive')

In [134]: corr['correlation']=result

In [135]: corr['correlation'].value_counts()

Out[135]: strong positive    9
          weak positive      3
          strong negative    3
          weak negative      3
          Name: correlation, dtype: int64

In [136]: plt.figure(figsize=(10,10))
          plt.title('Correlation Chart')
          labels=corr['correlation'].unique()
          plt15 = corr['correlation'].value_counts().tolist()
          plt.pie(plt15, labels=plt15, autopct='%1.1f%%')
          plt.legend(labels, loc=1)
```




```
In [137]: corr.loc[:,'correlation']

Out[137]: car_ID                weak negative
          symboling             weak negative
          wheelbase           strong positive
          carlength           strong positive
          carwidth            strong positive
          carheight             weak positive
          curbweight          strong positive
          enginesize          strong positive
          boreratio           strong positive
          stroke                weak positive
          compressionratio      weak positive
          horsepower          strong positive
          peakrpm               weak negative
          citympg             strong negative
          highwaympg          strong negative
          price               strong positive
          carvolume           strong positive
          fueleconomy         strong negative
          Name: correlation, dtype: object
```

Hence, we can clearly say that the highly correlated variables with price are:

1. Wheel Base
2. Car Length
3. Car Width
4. Curb Weight
5. Engine Size
6. Bore Ratio
7. Horsepower
8. Car Volume
9. Fuel Economy
10. Cars Range
11. Car Body
12. Fuel Type
13. Engine Type
14. Aspiration
15. Cylinder Number
16. Drivewheel

These variables are the features with which we will do our prediction.

4. Model Building

Now that we have selected which features to be chosen, it is time to make a model for our analysis. First of all, what is a model?

“A statistical model is usually specified as a mathematical relationship between one or more random variables and other non-random variables. As such, a statistical model is "a formal representation of a theory.”

Now, to be precise, we need to create a model. For that, we need a data set that has only values of the features we selected through visualization. Let's do that. The code and output are:

In [138]:
```
#Regression
cars_data=cars[['price','carsrange','enginetype','fueltype','carbody','aspiration','cylindernumber','carlength','carwidth','drive
```

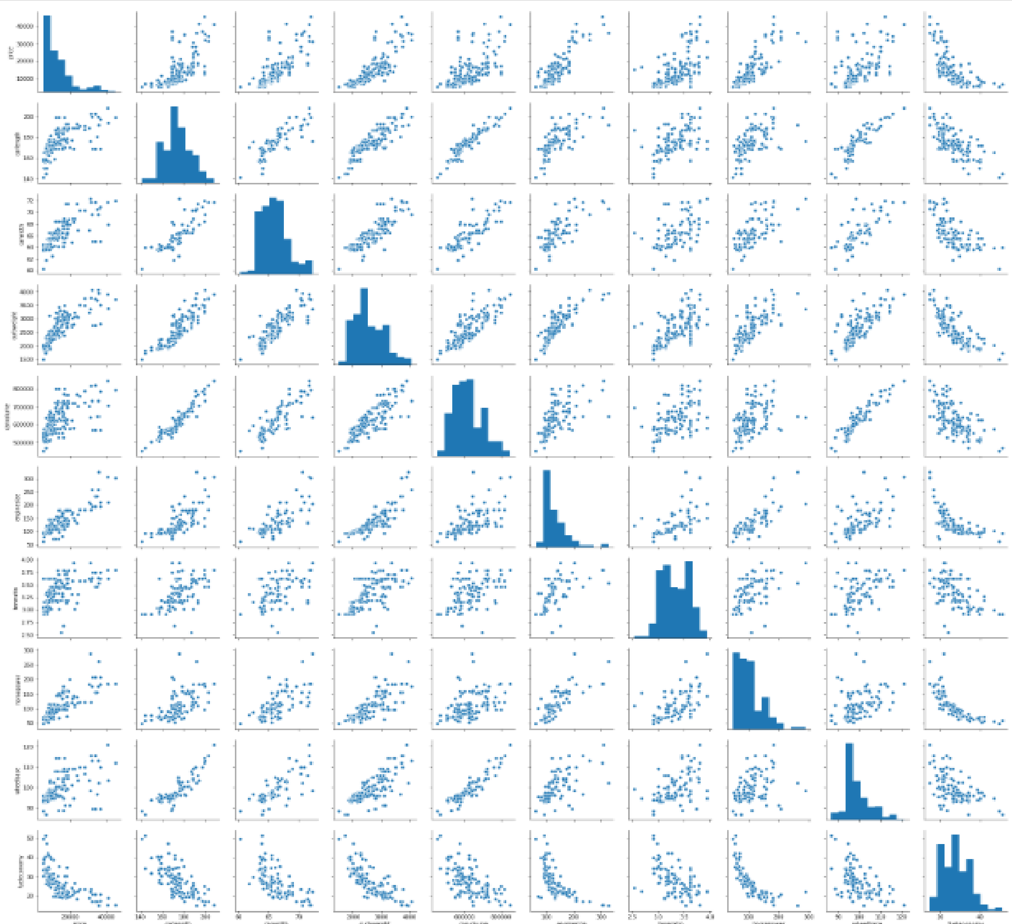
In [139]:
```
cars_data.head(5)
```

Out[139]:

	price	carsrange	enginetype	fueltype	carbody	aspiration	cylindernumber	carlength	carwidth	drivewheel	curbweight	carvolume	enginesize	borer
0	13495.0	Medium-Low	dohc	gas	convertible	std	four	168.8	64.1	rwd	2548	528019.904	130	
1	16500.0	Medium-Low	dohc	gas	convertible	std	four	168.8	64.1	rwd	2548	528019.904	130	
2	16500.0	Medium-Low	ohcv	gas	hatchback	std	six	171.2	65.5	rwd	2823	587592.640	152	
3	13950.0	Medium-Low	ohc	gas	sedan	std	four	176.6	66.2	fwd	2337	634816.956	109	
4	17450.0	Medium-Low	ohc	gas	sedan	std	five	176.6	66.4	4wd	2824	636734.832	136	

This is the data set, that we need to work on. We now need to have a pair plot of our whole data set, just to have an idea.

In [140]:
```
sns.pairplot(cars_data)
plt.show()
```



Now, moving on further, we need to create dummy variables. **What are dummy variables?**

"A dummy variable is one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. Dummy variables are used as devices to sort data into mutually exclusive categories."

Now, lets create these dummy variables. Remember, we need to create dummy variable for categorical variables only. Lets do this.

In [141]:
```
#Dummy Variables

def dummies(x,df):
    var=pd.get_dummies(df[x], drop_first=True)
    df=pd.concat([df,var], axis=1)
    df.drop([x], axis=1, inplace=True)
    return df
```

In [142]:
```
cars_data = dummies('fueltype',cars_data)
cars_data = dummies('aspiration',cars_data)
cars_data = dummies('carbody',cars_data)
cars_data = dummies('drivewheel',cars_data)
cars_data = dummies('enginetype',cars_data)
cars_data = dummies('cylindernumber',cars_data)
cars_data = dummies('carsrange',cars_data)
```

In [143]:
```
cars_data.head()
```

Out[143]:

	price	carlength	carwidth	curbweight	carvolume	enginesize	boreratio	horsepower	wheelbase	fueleconomy	...	five	four	six	three	twelve	two
0	13495.0	168.8	64.1	2548	528019.904	130	3.47	111	88.6	23.70	...	0	1	0	0	0	0
1	16500.0	168.8	64.1	2548	528019.904	130	3.47	111	88.6	23.70	...	0	1	0	0	0	0
2	16500.0	171.2	65.5	2823	587592.640	152	2.68	154	94.5	22.15	...	0	0	1	0	0	0
3	13950.0	176.6	66.2	2337	634816.956	109	3.19	102	99.8	26.70	...	0	1	0	0	0	0
4	17450.0	176.6	66.4	2824	636734.832	136	3.19	115	99.4	19.80	...	1	0	0	0	0	0

5 rows × 34 columns

In [144]:
```
cars_data.shape
```

Out[144]: (205, 34)

(205,34) is the shape of the data set now. It means that we have 34 variables to select from, for our analysis. Why select the variables again? Because when we run our analysis, we take an optimum amount of features to get the best results. We can let the model do that, or we can choose by ourselves.

Now comes the most important part. We will be splitting our data set into training set and test set. What are these sets? Why do we do this? Split the data set?

First of all, if you run the model on the whole data set and predict from the same, you will get accuracy too high, which would be invalid because your dependent variable will be included in the data set in which you are predicting your values.

Secondly, if the model fails, the data set has to be re-loaded from the beginning. Hence, we first train our model with the training set, and when our model runs perfectly, we use it on our test set to predict values. Remember, training set should always be greater than test set. The more you train your model, the better it will predict. Let's do this.

In [157]:
```
#Train-Test Split and Feature Scaling

from sklearn.model_selection import train_test_split
np.random.seed(0)
df_train, df_test=train_test_split(cars_data, train_size=0.6, test_size=0.4, random_state=100)
```

In [158]:
```
df_train.head() #training set
```

Out[158]:

	price	carlength	carwidth	curbweight	carvolume	enginesize	boreratio	horsepower	wheelbase	fueleconomy	...	five	four	six	three	twelve	two
20	6575.0	158.8	63.6	1909	525183.360	90	3.03	70	94.5	40.25	...	0	1	0	0	0	0
170	11199.0	176.2	65.6	2679	601053.440	146	3.62	116	98.4	26.70	...	0	1	0	0	0	0
82	12629.0	173.2	66.3	2833	576454.632	156	3.58	145	95.9	21.25	...	0	1	0	0	0	0
12	20970.0	176.8	64.8	2710	622095.552	164	3.31	121	101.2	24.15	...	0	0	1	0	0	0
16	41315.0	193.8	67.9	3380	706639.374	209	3.62	182	103.5	18.70	...	0	0	1	0	0	0

In [160]:
```
from sklearn.preprocessing import MinMaxScaler #feature scaling
scaler=MinMaxScaler()
```

In [161]:
```
high_corr=df_train.corr().loc[df_train.corr()['price']>0.75]['price']  #highly correlated values with price
high=high_corr.index.drop('price').tolist()
```

In [162]:
```
low_corr=df_train.corr().loc[df_train.corr()['price']<-0.45]['price']
low=low_corr.index.tolist()
```

In [163]:
```
num_vars=high+low
num_vars
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:334: DataConversionWarning: Data with input dtype uint
8, int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarning:
```

```
In [164]: #splitting into x and y
          y_train=df_train.pop('price')
          x_train=df_train

In [165]: df_train.head()
```

Out[165]:

	carlength	carwidth	curbweight	carvolume	enginesize	boreratio	horsepower	wheelbase	fueleconomy	gas	...	five	four	six	three	twelve	two	M
20	158.8	0.275000	0.163305	525183.360	0.109434	3.03	0.091667	94.5	0.698217	1	...	0	1.0	0	0	0	0	
170	176.2	0.441667	0.461986	601053.440	0.320755	3.62	0.283333	98.4	0.326475	1	...	0	1.0	0	0	0	0	
82	173.2	0.500000	0.521722	576454.632	0.358491	3.58	0.404167	95.9	0.176955	1	...	0	1.0	0	0	0	0	
12	176.8	0.375000	0.474011	622095.552	0.388679	3.31	0.304167	101.2	0.256516	1	...	0	0.0	1	0	0	0	
16	193.8	0.633333	0.733902	706639.374	0.558491	3.62	0.558333	103.5	0.106996	1	...	0	0.0	1	0	0	0	

5 rows × 33 columns

```
In [166]: df_train.describe().round(2)
```

Out[166]:

	carlength	carwidth	curbweight	carvolume	enginesize	boreratio	horsepower	wheelbase	fueleconomy	gas	...	five	four	six	three	tv
count	123.00	123.00	123.00	123.00	123.00	123.00	123.00	123.00	123.00	123.00	...	123.00	123.00	123.00	123.00	12
mean	173.72	0.47	0.41	616744.79	0.25	3.30	0.23	98.73	0.36	0.89	...	0.07	0.72	0.14	0.01	
std	12.99	0.19	0.22	81383.38	0.16	0.26	0.17	6.00	0.19	0.31	...	0.26	0.45	0.35	0.09	
min	141.10	0.00	0.00	452643.16	0.00	2.68	0.00	86.60	0.00	0.00	...	0.00	0.00	0.00	0.00	
25%	165.45	0.31	0.25	563757.42	0.14	3.06	0.09	94.50	0.20	1.00	...	0.00	0.00	0.00	0.00	
50%	173.00	0.43	0.36	601385.70	0.18	3.31	0.19	96.90	0.35	1.00	...	0.00	1.00	0.00	0.00	
75%	183.30	0.55	0.57	664251.44	0.32	3.54	0.28	100.80	0.47	1.00	...	0.00	1.00	0.00	0.00	
max	202.60	1.00	1.00	820742.73	1.00	3.94	1.00	115.60	1.00	1.00	...	1.00	1.00	1.00	1.00	

Now, after splitting the data set, we used a function MinMaxScaler(). This is because we may have very high values and very low values in our data set, and hence it scales down all those values to values that do not vary much. Remember, not to take your target variable (price) in this function, or your predicted values will also be scaled down values. Then, we split our training set into x (response variable) and y (target variable).

Now, it is time to build our model. For that, we use RFE (Recursive Feature Engineering) to select ‘n’ no. of variables from our already selected variables. Lets do that.

RFE selects ‘n’ no. of variables for your model on its own. You don’t need to select the variables. However, if you want to, you can do it by not using RFE. I have used it because it selects variables after running some tests on those variables.

Now, lets create our model :

```
In [198]: #4. Model Building

from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.ensemble import RandomForestRegressor
```

```
In [199]: model=LinearRegression()
model.fit(x_train, y_train)

rfe=RFE(model,15)
rfe=rfe.fit(x_train, y_train)

selected_features=list(zip(x_train.columns,rfe.support_,rfe.ranking_)) #checking the selected features
selected_features

index=x_train.columns[rfe.support_]
x_train_new=x_train[index]
x_train_new.head()
```

Out[199]:

	curbweight	enginesize	boreratio	dohcv	ohcv	five	four	six	three	twelve	two	High-Medium	Low	Medium	Medium-Low
20	0.163305	0.109434	3.03	0	0	0	1.0	0	0	0	0	0	1.0	0	0
170	0.461986	0.320755	3.62	0	0	0	1.0	0	0	0	0	0	0.0	0	1
82	0.521722	0.358491	3.58	0	0	0	1.0	0	0	0	0	0	0.0	0	1
12	0.474011	0.388679	3.31	0	0	0	0.0	1	0	0	0	0	0.0	1	0
16	0.733902	0.558491	3.62	0	0	0	0.0	1	0	0	0	0	0.0	0	0

Here, we will use linear regression to predict values. **Why linear regression? What is linear regression?**

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). We use linear regression because our data is continuous, as I mentioned in the 'Project Data Introduction', and we have to predict continuous values.

Now, let's start running our models :

RUNNING REGRESSION MODELS

Model 1

```
In [201]: #Running Regression Models

          #Model 1
          model_1=buildmodel(x_train_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.976
Model:                            OLS   Adj. R-squared:                  0.973
Method:                 Least Squares   F-statistic:                     290.8
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           2.33e-79
Time:                        12:05:01   Log-Likelihood:                -1048.7
No. Observations:                 123   AIC:                             2129.
Df Residuals:                     107   BIC:                             2174.
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.873e+04   2668.315     14.515      0.000    3.34e+04     4.4e+04
curbweight  1.171e+04   1538.575      7.610      0.000    8657.977    1.48e+04
enginesize  7748.9436   3517.011      2.203      0.030     776.880    1.47e+04
boreratio  -1816.4037    811.350     -2.239      0.027   -3424.810    -207.998
dohcv      -2578.3892   1884.765     -1.368      0.174   -6314.716    1157.938
ohcv       -2710.8777    711.560     -3.810      0.000   -4121.462   -1300.293
five       -5080.5391   1530.249     -3.320      0.001   -8114.080   -2046.998
four       -7707.5741   1677.129     -4.596      0.000     -1.1e+04  -4382.862
six        -6262.0912   1314.069     -4.765      0.000   -8867.079   -3657.103
three      -6348.1842   2238.815     -2.836      0.005   -1.08e+04   -1909.994
twelve     -1.252e+04   2445.723     -5.119      0.000   -1.74e+04   -7671.716
two        -4582.9848   2104.837     -2.177      0.032   -8755.578    -410.391
High-Medium -1.028e+04  1233.359     -8.332      0.000   -1.27e+04   -7831.590
Low        -2.195e+04   1275.639    -17.204      0.000   -2.45e+04   -1.94e+04
Medium     -1.398e+04   1119.731    -12.483      0.000   -1.62e+04   -1.18e+04
Medium-Low -1.988e+04   1135.630    -17.507      0.000   -2.21e+04   -1.76e+04
==============================================================================
Omnibus:                        2.881   Durbin-Watson:                   1.775
Prob(Omnibus):                  0.237   Jarque-Bera (JB):                2.330
Skew:                           0.303   Prob(JB):                        0.312
Kurtosis:                       3.295   Cond. No.                         142.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarning: Method .ptp is deprecated and will be
removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

```
In [202]: x_train_new=x_train_new.drop(['dohcv'], axis=1)
```

This is our Model Report. Notice at a column p>[t]. This column gives us the p-value. What is the ‘p-value’? It is a probability value that, when p>0.05, tells us to reject the null hypothesis and adopt alternate hypothesis.

What is null hypothesis?

null hypothesis is a general statement or default position that there is nothing new happening, like there is no relationship between two measured phenomena, or no association among groups. Every time we run our model, we need to check our p-value and delete those variables with p>0.05. We need to do this until all our variables have p-value less than 0.05.
Now, in the report, ‘dohcv’ is having p>0.05. Let us remove this and run a new model.

Model 2

```
In [203]: #Model 2
          model_2=buildmodel(x_train_new, y_train)

                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.976
Model:                            OLS   Adj. R-squared:                  0.972
Method:                 Least Squares   F-statistic:                     309.0
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           3.27e-80
Time:                        12:05:14   Log-Likelihood:                -1049.8
No. Observations:                 123   AIC:                             2130.
Df Residuals:                     108   BIC:                             2172.
Df Model:                          14
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        3.839e+04   2667.226     14.393      0.000    3.31e+04    4.37e+04
curbweight   1.179e+04   1543.472      7.641      0.000    8734.857    1.49e+04
enginesize   9012.4210   3407.240      2.645      0.009    2258.680    1.58e+04
boreratio   -2198.2255    764.900     -2.874      0.005   -3714.389    -682.062
ohcv        -2340.2146    660.604     -3.543      0.001   -3649.646   -1030.783
five        -3823.6589   1228.668     -3.112      0.002   -6259.092   -1388.226
four        -6273.2273   1314.253     -4.773      0.000   -8878.306   -3668.149
six         -5308.2339   1118.312     -4.747      0.000   -7524.922   -3091.546
three       -4803.7271   1941.100     -2.475      0.015   -8651.323    -956.131
twelve      -1.254e+04   2455.516     -5.108      0.000    -1.74e+04   -7674.987
two         -2943.9088   1737.516     -1.694      0.093   -6387.967     500.149
High-Medium  -1.05e+04   1226.985     -8.561      0.000    -1.29e+04   -8072.294
Low         -2.204e+04   1279.046    -17.229      0.000    -2.46e+04   -1.95e+04
Medium      -1.406e+04   1122.578    -12.526      0.000    -1.63e+04   -1.18e+04
Medium-Low       -2e+04   1137.080    -17.585      0.000    -2.22e+04   -1.77e+04
==============================================================================
Omnibus:                        1.835   Durbin-Watson:                   1.781
Prob(Omnibus):                  0.399   Jarque-Bera (JB):                1.404
Skew:                           0.245   Prob(JB):                        0.496
Kurtosis:                       3.185   Cond. No.                         126.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [204]: x_train_new=x_train_new.drop(['two'],axis=1)
```

Now, we can see that ‘*two*’ has p>0.05. Lets drop it and run the new model.

Now, create a new model and run it.

Model 3

Let us run this model and check our results.

Now, create a new model and run it.

```
In [202]: x_train_new=x_train_new.drop(['dohcv'], axis=1)

In [203]: #Model 2
          model_2=buildmodel(x_train_new, y_train)

                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.976
Model:                            OLS   Adj. R-squared:                  0.972
Method:                 Least Squares   F-statistic:                     309.0
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           3.27e-80
Time:                        12:05:14   Log-Likelihood:                -1049.8
No. Observations:                 123   AIC:                             2130.
Df Residuals:                     108   BIC:                             2172.
Df Model:                          14
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        3.839e+04   2667.226     14.393      0.000    3.31e+04    4.37e+04
curbweight   1.179e+04   1543.472      7.641      0.000    8734.857    1.49e+04
enginesize   9012.4210   3407.240      2.645      0.009    2258.680    1.58e+04
boreratio   -2198.2255    764.900     -2.874      0.005   -3714.389    -682.062
ohcv        -2340.2146    660.604     -3.543      0.001   -3649.646   -1030.783
five        -3823.6589   1228.668     -3.112      0.002   -6259.092   -1388.226
four        -6273.2273   1314.253     -4.773      0.000   -8878.306   -3668.149
six         -5308.2339   1118.312     -4.747      0.000   -7524.922   -3091.546
three       -4803.7271   1941.100     -2.475      0.015   -8651.323    -956.131
twelve      -1.254e+04   2455.516     -5.108      0.000   -1.74e+04   -7674.987
two         -2943.9088   1737.516     -1.694      0.093   -6387.967     500.149
High-Medium  -1.05e+04   1226.985     -8.561      0.000   -1.29e+04   -8072.294
Low         -2.204e+04   1279.046    -17.229      0.000   -2.46e+04    -1.95e+04
Medium      -1.406e+04   1122.578    -12.526      0.000   -1.63e+04    -1.18e+04
Medium-Low     -2e+04    1137.080    -17.585      0.000   -2.22e+04    -1.77e+04
==============================================================================
Omnibus:                        1.835   Durbin-Watson:                   1.781
Prob(Omnibus):                  0.399   Jarque-Bera (JB):                1.404
Skew:                           0.245   Prob(JB):                        0.496
Kurtosis:                       3.185   Cond. No.                         126.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [204]: x_train_new=x_train_new.drop(['two'],axis=1)
```

This time, '*three*' has p>0.05. Let us drop it and run a new model.

Model 4

Lets run this model and check the results.

```
In [236]: #Model 4
          model_4=buildmodel(x_train_new,y_train)

                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.974
Model:                            OLS   Adj. R-squared:                  0.971
Method:                 Least Squares   F-statistic:                     346.9
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           1.86e-81
Time:                        12:13:35   Log-Likelihood:                -1053.2
No. Observations:                 123   AIC:                             2132.
Df Residuals:                     110   BIC:                             2169.
Df Model:                          12
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        3.686e+04   2629.118     14.019      0.000    3.16e+04    4.21e+04
curbweight   1.152e+04   1519.364      7.583      0.000    8510.440    1.45e+04
enginesize   1.207e+04   2885.177      4.182      0.000    6348.103    1.78e+04
boreratio   -2583.1645    747.157     -3.457      0.001   -4063.854   -1102.475
ohcv        -1920.4170    645.569     -2.975      0.004   -3199.783    -641.051
five        -1885.6964    819.357     -2.301      0.023   -3509.469    -261.924
four        -3870.8153    671.986     -5.760      0.000   -5202.534   -2539.097
six         -3883.6397    926.837     -4.190      0.000   -5720.414   -2046.866
twelve      -1.286e+04   2472.944     -5.201      0.000   -1.78e+04   -7961.858
High-Medium -1.028e+04   1246.028     -8.247      0.000   -1.27e+04   -7807.119
Low         -2.211e+04   1301.470    -16.987      0.000   -2.47e+04   -1.95e+04
Medium        -1.4e+04   1141.434    -12.269      0.000   -1.63e+04   -1.17e+04
Medium-Low   -2.01e+04   1157.507    -17.366      0.000   -2.24e+04   -1.78e+04
==============================================================================
Omnibus:                        0.281   Durbin-Watson:                   1.683
Prob(Omnibus):                  0.869   Jarque-Bera (JB):                0.202
Skew:                           0.099   Prob(JB):                        0.904
Kurtosis:                       2.980   Cond. No.                         113.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

This is our trained model. Let it be f_model.

f_model=model_4

Now, lets check its Variance Inflation Factor (VIF) Value. First of all, what is VIF Value? In statistics, the variance inflation factor (VIF) is the ratio of variance in a model with multiple terms, divided by the variance of a model with one term alone. It needs to be under control.

In [209]:
```python
#This is the final model. Hence, it will be named as f_model.
f_model=model_4
```

In [210]:
```python
#checking vif value

def checkVIF(x):
    vif = pd.DataFrame()
    vif['Features'] = x.columns
    vif['VIF'] = [variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return(vif)
```

In [211]:
```python
checkVIF(model_4)
```

Out[211]:

	Features	VIF
0	const	474.44
10	Low	28.22
12	Medium-Low	22.32
2	enginesize	14.54
1	curbweight	7.48
11	Medium	7.28
7	six	7.02
6	four	6.20
9	High-Medium	5.72
8	twelve	3.38
5	five	3.12
3	boreratio	2.52
4	ohcv	1.94

The VIF Value for Low, Medium-Low and enginesize is very high. Let us drop it and run a new model. Remember, check the p-value and the vif value both.

In [214]:
```python
checkVIF(model_5) #checking vif value
```

Out[214]:

	Features	VIF
0	const	294.13
5	four	4.99
1	curbweight	4.44
6	six	4.14
4	five	2.69
2	boreratio	2.10
8	High-Medium	1.92
3	ohcv	1.63
7	twelve	1.46
9	Medium	1.41

VIF Value is not very high. Let us check the regression results:

```
In [213]: model_new=model_4.drop(['Low','Medium-Low','enginesize'], axis=1)
          model_5=buildmodel(model_new, y_train) #checking OLS Results
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.880
Model:                            OLS   Adj. R-squared:                  0.870
Method:                 Least Squares   F-statistic:                     92.05
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           8.19e-48
Time:                        12:07:09   Log-Likelihood:                -1147.9
No. Observations:                 123   AIC:                             2316.
Df Residuals:                     113   BIC:                             2344.
Df Model:                           9
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        6420.5321   4409.924      1.456      0.148   -2316.323    1.52e+04
curbweight   2.004e+04   2492.273      8.039      0.000    1.51e+04     2.5e+04
boreratio     454.9175   1451.243      0.313      0.755   -2420.256    3330.091
ohcv        -3558.0971   1260.089     -2.824      0.006   -6054.560   -1061.634
five        -1965.7916   1620.189     -1.213      0.228   -5175.678    1244.095
four        -5159.1966   1284.532     -4.016      0.000   -7704.086   -2614.307
six           364.4492   1515.759      0.240      0.810   -2638.544    3367.442
twelve       1.239e+04   3464.547      3.577      0.001    5528.800    1.93e+04
High-Medium  8228.9699   1537.231      5.353      0.000    5183.438    1.13e+04
Medium       3570.1509   1068.116      3.342      0.001    1454.020    5686.282
==============================================================================
Omnibus:                      111.091   Durbin-Watson:                   1.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1789.741
Skew:                           2.928   Prob(JB):                         0.00
Kurtosis:                      20.746   Cond. No.                         71.3
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Looks like the p-value of boreratio, five and six is greater than 0.05. Let us remove them and run a new model.

```
In [216]: checkVIF(model_6)
```

Out[216]:

	Features	VIF
0	const	18.63
1	curbweight	2.13
3	four	1.97
5	High-Medium	1.65
2	ohcv	1.41
6	Medium	1.35
4	twelve	1.22

VIF Value is not very high. Let us check the regression results:

```
In [215]: model_new=model_new.drop(['boreratio','five','six'], axis=1)
          model_6=buildmodel(model_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.876
Model:                            OLS   Adj. R-squared:                  0.870
Method:                 Least Squares   F-statistic:                     136.6
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           3.61e-50
Time:                        12:07:35   Log-Likelihood:                -1149.9
No. Observations:                 123   AIC:                             2314.
Df Residuals:                     116   BIC:                             2333.
Df Model:                           6
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        7125.0728   1113.385      6.399      0.000    4919.873    9330.272
curbweight   2.051e+04   1732.557     11.836      0.000    1.71e+04    2.39e+04
ohcv        -2774.2104   1176.675     -2.358      0.020   -5104.763    -443.658
four        -4528.9579    810.077     -5.591      0.000   -6133.417   -2924.499
twelve       1.207e+04   3169.131      3.807      0.000    5788.683    1.83e+04
High-Medium  8068.1673   1430.769      5.639      0.000    5234.349    1.09e+04
Medium       3665.5069   1049.602      3.492      0.001    1586.638    5744.376
==============================================================================
Omnibus:                      124.321   Durbin-Watson:                   1.919
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2483.970
Skew:                           3.380   Prob(JB):                         0.00
Kurtosis:                      23.952   Cond. No.                         16.8
==============================================================================
```

p-value is under control. Hence, it is our regression model. Let us name it as final_rm.

final_rm=model_6

Now, we need to check errors in our model. For that, we will drop any one variable, and see what are the results.

```
In [216]: checkVIF(model_6)
```

Out[216]:

	Features	VIF
0	const	18.63
1	curbweight	2.13
3	four	1.97
5	High-Medium	1.65
2	ohcv	1.41
6	Medium	1.35
4	twelve	1.22

Now, lets check the VIF value and regression results.

In [217]:
```
#VIF Value is under control. Now, this is our final regression model.
final_rm=model_6
```

In [218]:
```
#Now, to check errors, we will drop one feature, lets say hatchback.
model_check=model_6.drop(['ohcv'], axis=1)
model_check=buildmodel(model_check, y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.870
Model:                            OLS   Adj. R-squared:                  0.864
Method:                 Least Squares   F-statistic:                     156.6
Date:                Mon, 22 Jul 2019   Prob (F-statistic):           4.08e-50
Time:                        12:08:07   Log-Likelihood:                -1152.8
No. Observations:                 123   AIC:                             2318.
Df Residuals:                     117   BIC:                             2334.
Df Model:                           5
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        6726.4504   1121.706      5.997      0.000    4504.969    8947.931
curbweight   2.025e+04   1762.595     11.491      0.000    1.68e+04    2.37e+04
four        -4040.0631    798.196     -5.061      0.000   -5620.849   -2459.277
twelve       9931.6433   3095.739      3.208      0.002    3800.694    1.61e+04
High-Medium  7898.9154   1456.539      5.423      0.000    5014.317    1.08e+04
Medium       3368.7139   1062.132      3.172      0.002    1265.218    5472.210
==============================================================================
Omnibus:                      125.047   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2691.565
Skew:                           3.372   Prob(JB):                         0.00
Kurtosis:                      24.902   Cond. No.                         16.0
==============================================================================
```

In [219]:
```
checkVIF(model_check)
```

Out[219]:

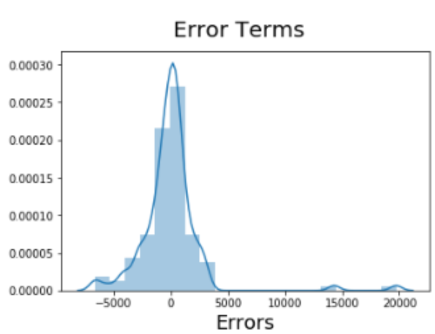
	Features	VIF
0	const	18.20
1	curbweight	2.12
2	four	1.84
4	High-Medium	1.65
5	Medium	1.33
3	twelve	1.12

the p-value and vif values are under control. Hence, there is not much error. Let us plot a graph to just be sure.

In [220]:
```
#dist plot for residual analysis
lm=sm.OLS(y_train,model_check).fit()
y_train_price=lm.predict(model_check)
```

In [221]:
```
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)
```

Out[221]: Text(0.5, 0, 'Errors')



As expected, there is not much error in our model. Now that we have built our model, it is time to predict. Let us predict the price of the cars.

5. Prediction and Evaluation

We are at the final stage of our project. Time to predict values! First, let us select the features in our training set. Then, split into x and y.

```
In [222]: #5. Prediction and Evaluation

          #selecting the highly correlated values
          df_test[num_vars] = scaler.fit_transform(df_test[num_vars])
```

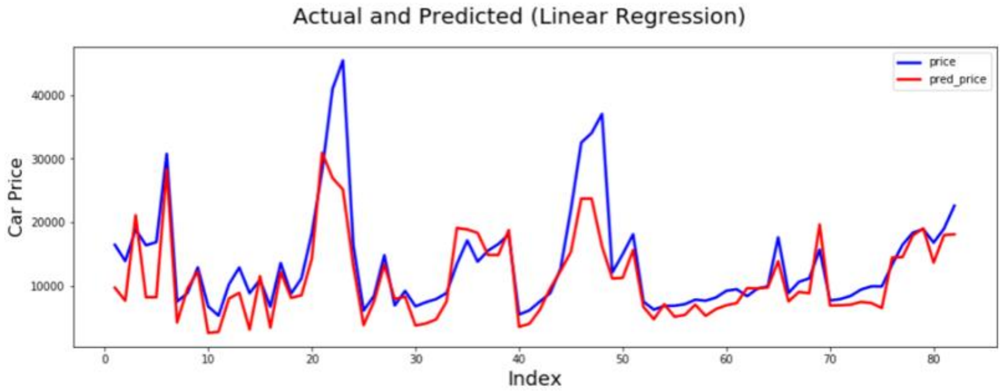
```
In [223]: #splitting into x and y
          y_test=df_test.pop('price')
          x_test=df_test

In [224]: # Now let's use our model to make predictions.
          X_train_new = model_check.drop('const',axis=1)

In [225]: # Creating X_test_new dataframe by dropping variables from X_test
          X_test_new = x_test[X_train_new.columns]

In [226]: # Adding a constant variable
          X_test_new = sm.add_constant(X_test_new)
```

Now, let us predict values for our test set. Then, plot them in a line graph to show the variation.

Now, let us predict values for our test set. Then, plot them in a line graph to show the variation.

```
In [ ]: y_pred=lm.predict(X_test_new)
        price=pd.concat([y_test,y_pred.round(2)],axis=1)
        price=price.rename(columns={0:'pred_price'}) #price prediction using linear regression
        price=price.sort_index()
        price=price.reset_index(0)

In [ ]: c= [i for i in range(1,83,1)] # generating index
        fig = plt.figure(figsize=(15,5))
        plt.plot(c,price['price'], color="blue", linewidth=2.5, linestyle="-") #Plotting Actual
        plt.plot(c,price['pred_price'], color="red",  linewidth=2.5, linestyle="-") #Plotting predicted
        fig.suptitle('Actual and Predicted (Linear Regression)', fontsize=20)              # Plot heading
        plt.xlabel('Index', fontsize=18)                               # X-label
        plt.ylabel('Car Price', fontsize=16)                       # Y-label
        plt.legend()
        plt.show()
```

Let us see what is the variation in the predicted price.



Barring a few irregularities, the prediction seems to work fine. Let's check how much accurate we are.

```
from sklearn.metrics import r2_score
acc=r2_score(y_test, y_pred)
print('The Accuracy Score is : ',(acc*100).round(3),'%') #Accuracy Score with Linear Regress
ion
```

```
The Accuracy Score is :  64.377 %
```

The accuracy is **64.377 %.** It is a little bit low, according to me. Let us use Random Forest Regressor to fine tune our analysis. What is 'Random Forest'?

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

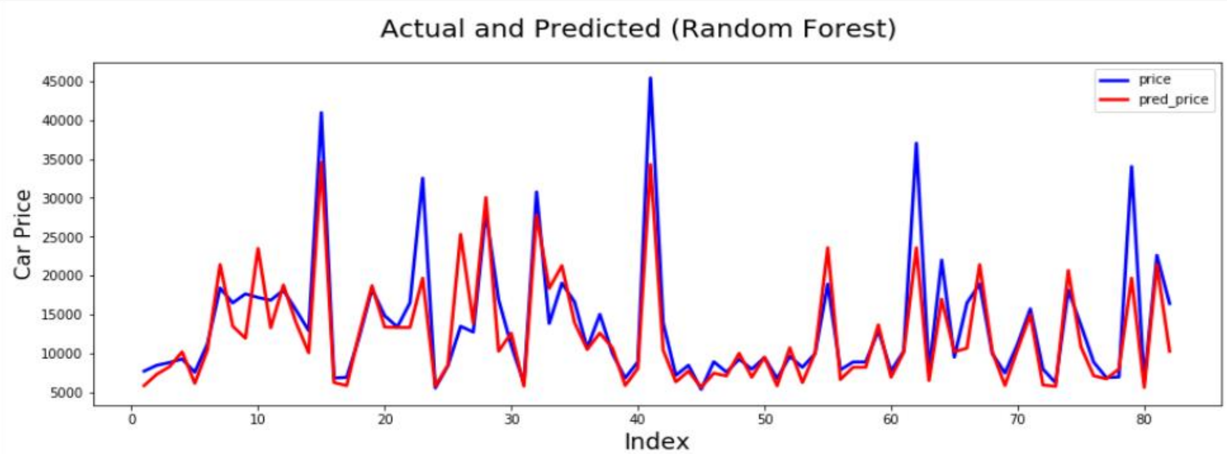
Now, let us use Random Forest for our regression.

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(x_train,y_train)

rf_pred=pd.Series(rf.predict(x_test)) #price prediction using random forest
rf_pred
acc_rf=r2_score(y_test, rf_pred)
print('The Accuracy Score is : ',(acc_rf*100).round(3),'%') #Accuracy Score with Random Fore
st Regressor
```

```
The Accuracy Score is :  76.147 %
```

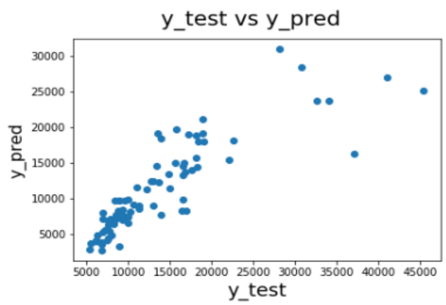
Now, we get an accuracy of 76.147%, which is more than enough for prediction. Let us plot the price and predicted price line graph to check the difference.

```
#Plot for Actual vs Predicted Price
c= [i for i in range(1,83,1)] # generating index
fig = plt.figure(figsize=(15,5))
plt.plot(c,y_test, color="blue", linewidth=2.5, linestyle="-", label='price') #Plotting Actual
plt.plot(c,rf_pred, color="red",  linewidth=2.5, linestyle="-", label='pred_price') #Plotting pr
edicted
fig.suptitle('Actual and Predicted (Random Forest)', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                                          # X-label
plt.ylabel('Car Price', fontsize=16)                                  # Y-label
plt.legend()
plt.show()
```



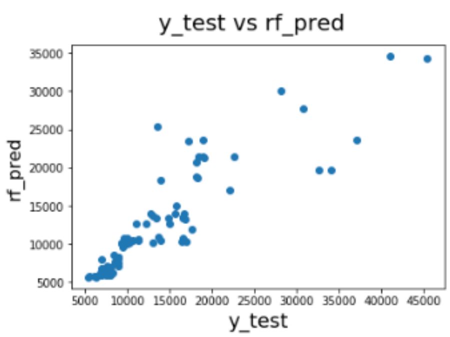
Not bad, though. This was our predictive analysis model. Let us check the price spread between actual and predicted price for both random forest and linear regression.

```
In [234]: # Plotting y_test and y_pred to understand the spread.
          fig = plt.figure()
          plt.scatter(y_test,y_pred)
          fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
          plt.xlabel('y_test', fontsize=18)                          # X-label
          plt.ylabel('y_pred', fontsize=16)                          # Y-label

Out[234]: Text(0, 0.5, 'y_pred')
```



Now, lets check with random forest. This would be the final part of the project.

```
In [235]: #plotting y_test and rf_pred to understand the spread.
          fig = plt.figure()
          plt.scatter(y_test,rf_pred)
          fig.suptitle('y_test vs rf_pred', fontsize=20)              # Plot heading
          plt.xlabel('y_test', fontsize=18)                          # X-label
          plt.ylabel('rf_pred', fontsize=16)                          # Y-label

Out[235]: Text(0, 0.5, 'rf_pred')
```



References :

- https://github.com/akjadon/Finalprojects_DS/tree/master/Car_pricing_prediction
- https://www.kaggle.com/goyalshalini93/car-price-prediction-linear-regression-rfe
- https://arxiv.org/pdf/1711.06970.pdf
- https://en.wikipedia.org/wiki/Linear_regression
- https://en.wikipedia.org/wiki/Random_forest
- https://www.kaggle.com/jshih7/car-price-prediction
- https://en.wikipedia.org/wiki/Dummy_variable_(statistics)

-------This Project is available to be seen on GITHUB follow the below link-------

⇩ ⇩

https://github.com/jitroy160/Final_Projects/blob/master/Final_Projects/Final_Project_CarPrice_Prediction.ipynb