**Summer Internship Program**

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



Project Title – **Car Price Prediction**

Mentor Name: Ms. Pooja Gupta

**Name: PRIYANK JHA**

**Course: Summer Internship Program (SIP) Python Batch: Jun-Jul 2019**

**Job: Business Analyst Associate (Intern)**

# DECLARATION

I here by declare that the project report entitled “**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. This project is an original report with references taken from websites and help from mentors and teachers.

DATE: 28 Jul 2019

PRIYANK JHA

SIP – Python

# Acknowledgements

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 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.

PRIYANK JHA

SIP-Python

**TABLE OF CONTENTS**

[Declaration 2](#_TOC_250008)

[Acknowledgements 3](#_TOC_250007)

[Abstract 5](#_TOC_250006)

1. [Project Introduction 6](#_TOC_250005)
2. [Project Data Introduction 7](#_TOC_250004)
3. [Exploratory Data Analysis (EDA) 9](#_TOC_250003)
4. [Model Building 37](#_TOC_250002)
5. [Prediction and Evaluation 51](#_TOC_250001)

[Bibliography 57](#_TOC_250000)

# 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.”*

# 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.

# 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 |
| Symboling | 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 |
| Curbweight | 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, Symboling, 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

This Data Set is present in the GITHUB Repository as follows: <https://github.com/yashj1301/Python-Projects/blob/master/data/car_price.csv>

# Exploratory Data Analysis (EDA)

In statistics, **exploratory data analysis** (**EDA**) is an approach to analysing 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:

## 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:

import pandas as pd import numpy as np

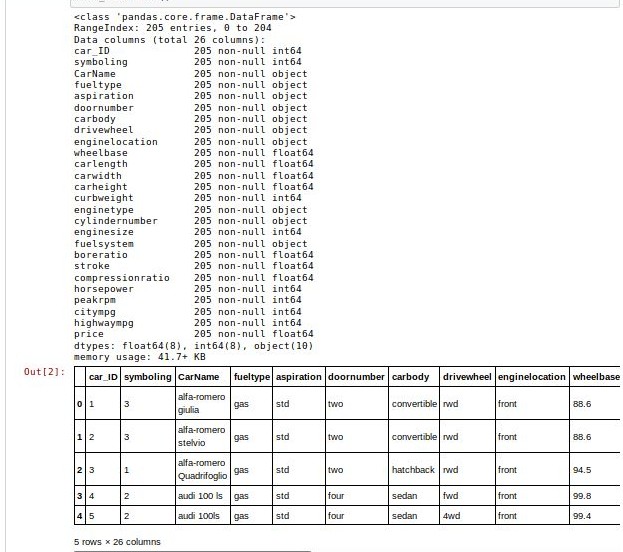
import matplotlib.pyplot as plt import seaborn as sns

import statsmodels.api as sm

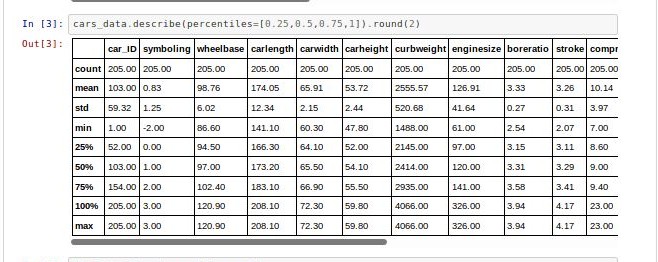
%matplotlib inline

cars\_data=pd.read\_csv('https://raw.githubusercontent.com/yashj1301/Py thon-Projects/master/data/car\_price.csv')

cars\_data.info() cars\_data.head()



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:



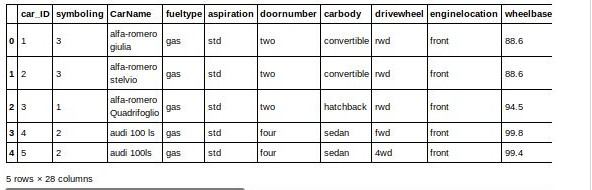
## Data Cleaning and Preparation

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:

#splitting the car name column and creating new columns company and car model

cars\_data=cars\_data.join(cars\_data['CarName'].str.split(' ',1,expand=True).rename(columns={0:'Company',1:'CarModel'}))

cars\_data.head()



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

#checking the columns created

print('--------------------------------------------------------------

---------------')

print(' Company Names ')

print('--------------------------------------------------------------

---------------')

print(cars\_data['Company'].unique())

print('--------------------------------------------------------------

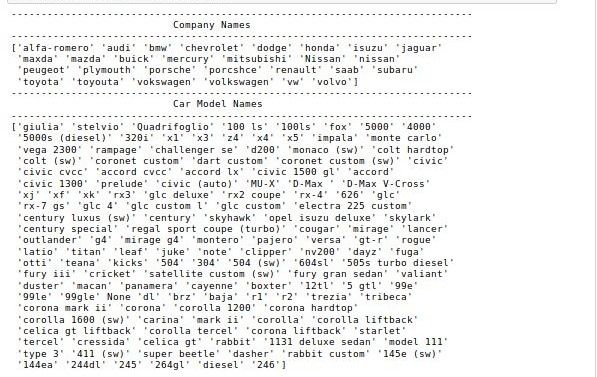
---------------')

print(' Car Model Names ')

print('--------------------------------------------------------------

---------------')

print(cars\_data['CarModel'].unique())



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.

#replacing incorrect values to correct values cars\_data['Company'].replace('maxda','mazda',inplace=True)

cars\_data['Company'].replace(['vokswagen','vw'],'volkswagen',inplace= True)

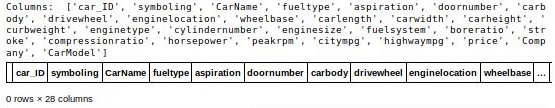
cars\_data['Company'].replace('porcshce','porsche',inplace=True) cars\_data['Company'].replace('toyouta','toyota',inplace=True)

#converting all the string data to lower to avoid any case difference errors

cars\_data['Company']=cars\_data['Company'].str.lower()

#checking columns column\_names=cars\_data.columns.tolist() print('Columns: ',column\_names)

#checking for duplicate values cars\_data.loc[cars\_data.duplicated()]



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

## 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 :

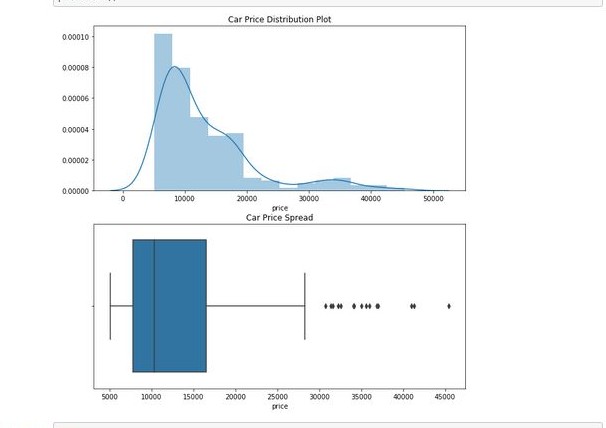
plt.figure(figsize=(10,10)) #plot size according to scale 1 unit=72 pixels

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\_data['price']) #distplot for price of cars

plt.subplot(2,1,2) plt.title('Car Price Spread')

sns.boxplot(cars\_data['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 dataframe 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

### 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:

plt.figure(figsize=(30, 20))

#plot 1 plt.subplot(1,2,1)

plt1 = cars\_data['Company'].value\_counts().plot('bar') plt.title('Companies Histogram')

plt1.set(xlabel = 'Car Company', ylabel='Frequency of Company')

xs=cars\_data['Company'].unique() ys=cars\_data['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)

#plot 2

plt.subplot(1,2,2) company\_vs\_price =

pd.DataFrame(cars\_data.groupby(['Company'])['price'].mean().sort\_valu es(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')

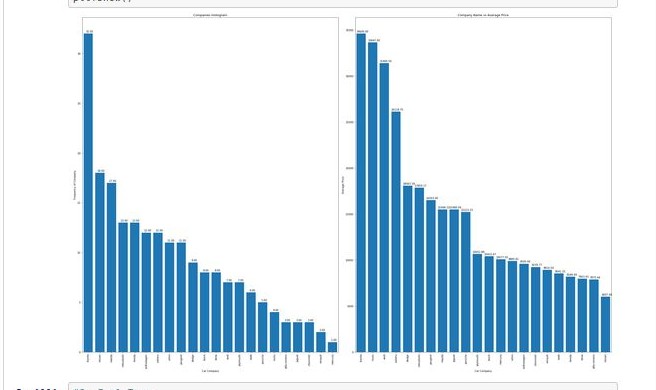
xs=company\_vs\_price.index ys=company\_vs\_price['price'].round(2) plt.bar(xs,ys)

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.

### 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:

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

#plot 1 plt.subplot(1,2,1)

plt.title('Fuel Type Chart') labels=cars\_data['fueltype'].unique()

plt3 = cars\_data['fueltype'].value\_counts().tolist() plt.pie(plt3,labels=plt3, autopct='%1.1f%%')

plt.legend(labels)

#plot 2 plt.subplot(1,2,2)

fuel\_vs\_price =

pd.DataFrame(cars\_data.groupby(['fueltype'])['price'].mean().sort\_val ues(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)

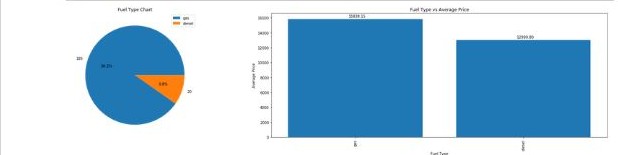
plt.bar(xs,ys)

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.

### 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:

plt.figure(figsize=(15,10)) #plot 1

plt.subplot(1,2,1) plt.title('Car Body Type Chart')

labels=cars\_data['carbody'].unique()

plt5 = cars\_data['carbody'].value\_counts().tolist() plt.pie(plt5, labels=plt5, autopct='%1.1f%%') plt.legend(labels, loc=1)

#plot 2

plt.subplot(1,2,2) car\_vs\_price =

pd.DataFrame(cars\_data.groupby(['carbody'])['price'].mean().sort\_valu es(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')

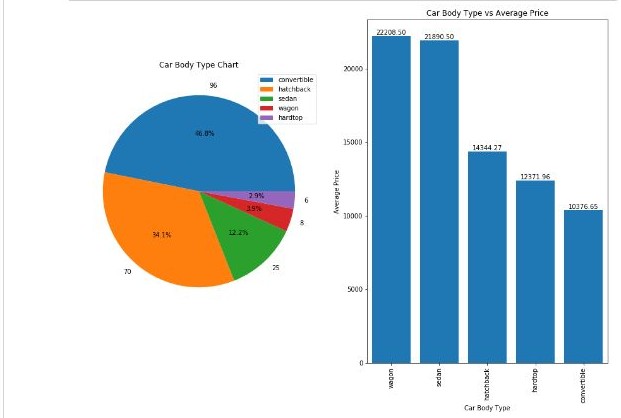
xs=car\_vs\_price.index ys=car\_vs\_price['price'].round(2) plt.bar(xs,ys)

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.

### 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:

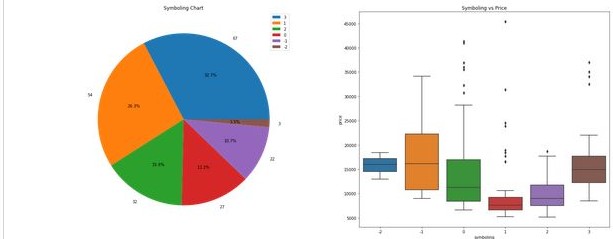
plt.figure(figsize=(25,10)) #plot 1

plt.subplot(1,2,1) plt.title('Symboling Chart') labels=cars\_data['symboling'].unique()

plt7 = cars\_data['symboling'].value\_counts().tolist() plt.pie(plt7, labels=plt7, autopct='%1.1f%%') plt.legend(labels, loc=1)

#plot 2 plt.subplot(1,2,2)

plt.title('Symboling vs Price') sns.boxplot(x=cars\_data['symboling'], y=cars\_data['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.

### 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:

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

#plot 1 plt.subplot(1,2,1)

plt8 = cars\_data['enginetype'].value\_counts().plot('bar') plt.title('Engine Type Histogram')

plt8.set(xlabel = 'Engine Type', ylabel='Frequency') xs=cars\_data['enginetype'].unique() ys=cars\_data['enginetype'].value\_counts() plt.bar(xs,ys)

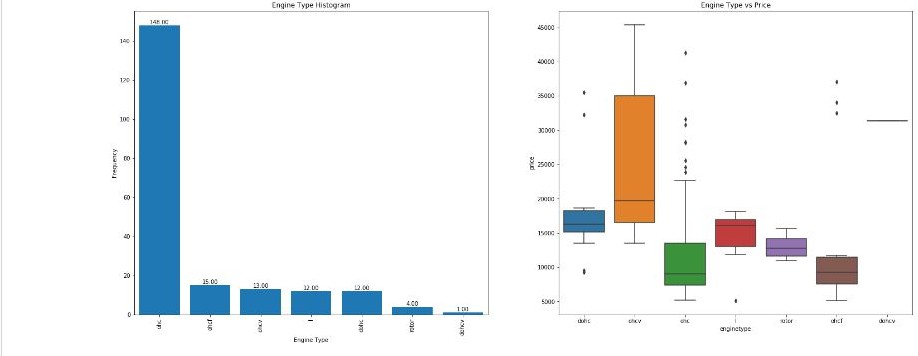
for x,y in zip(xs,ys):

label = "{:.2f}".format(y) plt.annotate(label,(x,y), textcoords="offset

points",xytext=(0,2),ha='center') plt.xticks(xs)

#plot 2 plt.subplot(1,2,2)

plt.title('Engine Type vs Price') sns.boxplot(x=cars\_data['enginetype'], y=cars\_data['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.

### 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.

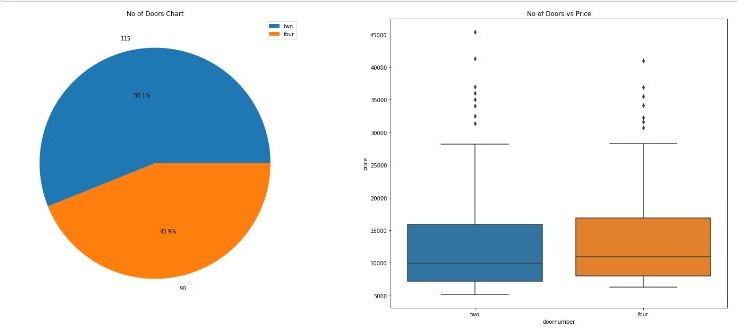
plt.figure(figsize=(25,10)) #plot 1

plt.subplot(1,2,1) labels=cars\_data['doornumber'].unique()

plt8 = cars\_data['doornumber'].value\_counts().tolist() plt.title('No of Doors Chart')

plt.pie(plt8, labels=plt8, autopct='%1.1f%%') plt.legend(labels, loc=1)

#plot 2 plt.subplot(1,2,2)

plt.title('No of Doors vs Price') sns.boxplot(x=cars\_data['doornumber'], y=cars\_data['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.

### 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:

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

#plot 1 plt.subplot(1,2,1)

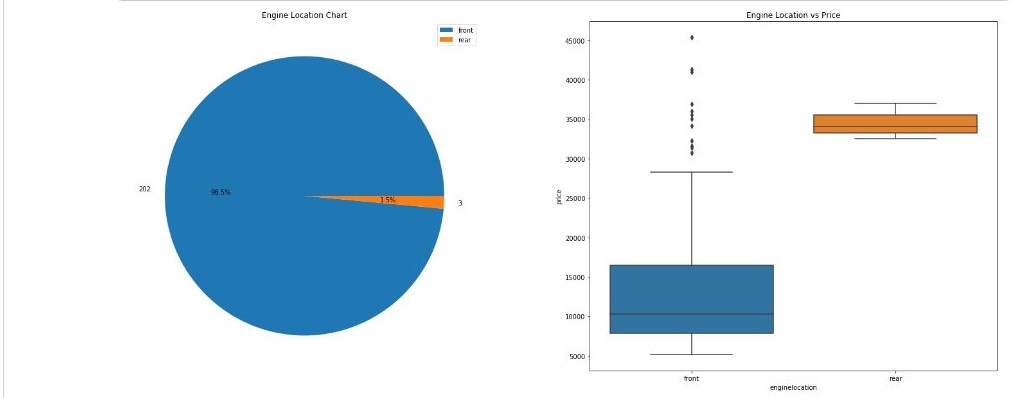
labels=cars\_data['enginelocation'].unique()

plt9 = cars\_data['enginelocation'].value\_counts().tolist() plt.title('Engine Location Chart')

plt.pie(plt9, labels=plt9, autopct='%1.1f%%') plt.legend(labels, loc=1)

#plot 2 plt.subplot(1,2,2)

plt.title('Engine Location vs Price') sns.boxplot(x=cars\_data['enginelocation'], y=cars\_data['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.

### 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:

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

#plot 1 plt.subplot(1,2,1)

plt10 = cars\_data['fuelsystem'].value\_counts().plot('bar') plt.title('Fuel System Type Histogram')

plt10.set(xlabel = 'Fuel System Type', ylabel='Frequency') xs=cars\_data['fuelsystem'].unique() ys=cars\_data['fuelsystem'].value\_counts()

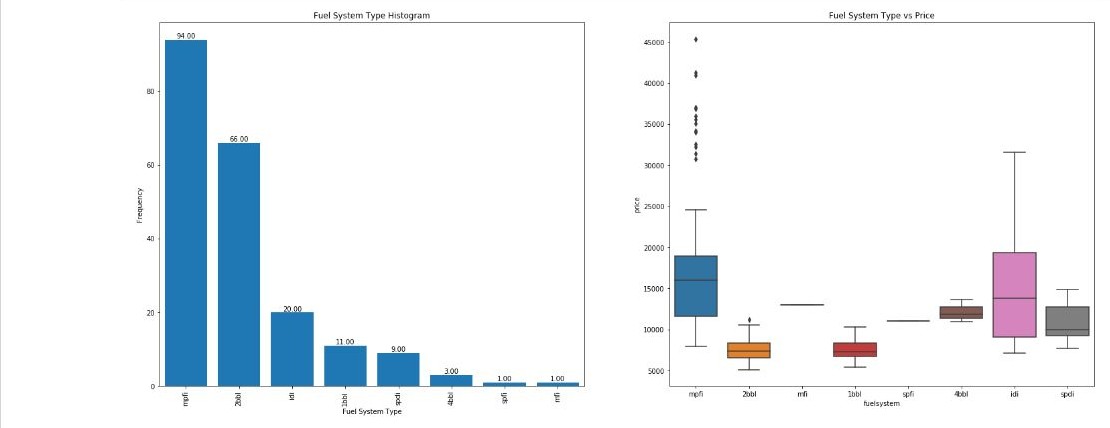
plt.bar(xs,ys)

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)

#plot 2 plt.subplot(1,2,2)

plt.title('Fuel System Type vs Price') sns.boxplot(x=cars\_data['fuelsystem'], y=cars\_data['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.

### 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, let’s check what our data tells us. The code and output are:

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

#plot 1 plt.subplot(1,2,1)

plt11 = cars\_data['cylindernumber'].value\_counts().plot('bar') plt.title('Cylinder Number Histogram')

plt11.set(xlabel = 'Cylinder Number', ylabel='Frequency') xs=cars\_data['cylindernumber'].unique() ys=cars\_data['cylindernumber'].value\_counts() plt.bar(xs,ys)

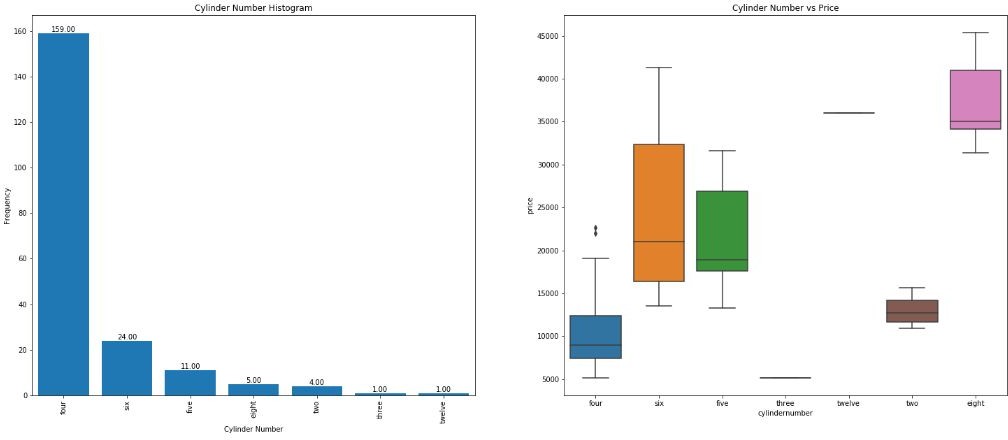
for x,y in zip(xs,ys):

label = "{:.2f}".format(y) plt.annotate(label,(x,y), textcoords="offset

points",xytext=(0,2),ha='center') plt.xticks(xs)

#plot 2 plt.subplot(1,2,2)

plt.title('Cylinder Number vs Price') sns.boxplot(x=cars\_data['cylindernumber'], y=cars\_data['price']) plt.show()



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.

### 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:

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

#plot 1 plt.subplot(1,2,1)

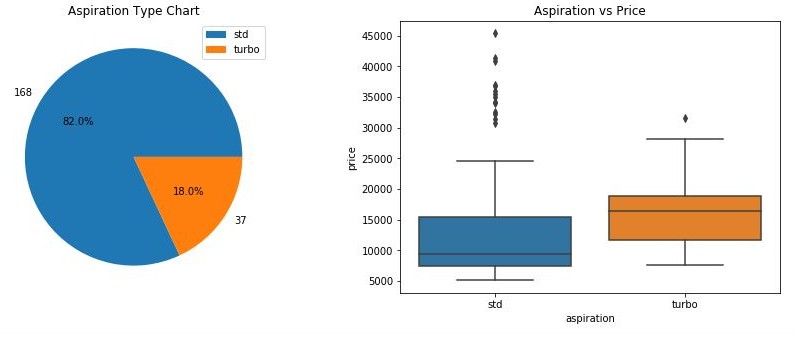
labels=cars\_data['aspiration'].unique()

plt12 = cars\_data['aspiration'].value\_counts().tolist() plt.title('Aspiration Type Chart')

plt.pie(plt12, labels=plt12, autopct='%1.1f%%') plt.legend(labels, loc=1)

#plot 2 plt.subplot(1,2,2)

plt.title('Aspiration vs Price') sns.boxplot(x=cars\_data['aspiration'], y=cars\_data['price']) plt.show()



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

### 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:

plt.figure(figsize=(15,5)) #plot 1

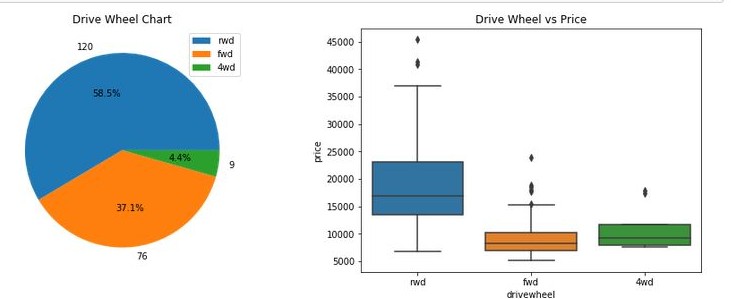
plt.subplot(1,2,1) labels=cars\_data['drivewheel'].unique()

plt13 = cars\_data['drivewheel'].value\_counts().tolist() plt.title('Drive Wheel Chart')

plt.pie(plt13, labels=plt13, autopct='%1.1f%%') plt.legend(labels, loc=1)

#plot 2 plt.subplot(1,2,2)

plt.title('Drive Wheel vs Price') sns.boxplot(x=cars\_data['drivewheel'], y=cars\_data['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.

### 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:

price=cars\_data['price'].tolist() price

carsrange=[]

for i in cars\_data['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\_data['carsrange']=carsrange cars\_data['carsrange'].unique()

#plot plt.figure(figsize=(5,5))

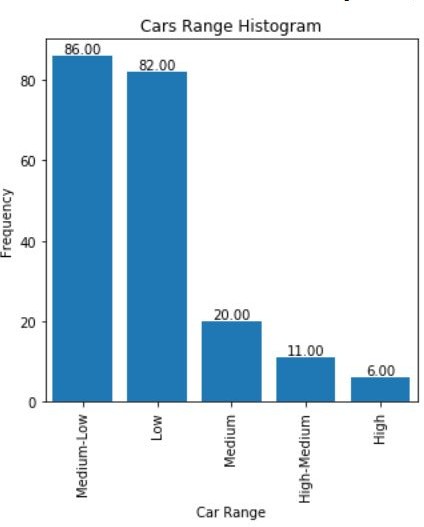
plt14 = cars\_data['carsrange'].value\_counts().plot('bar') plt.title('Cars Range Histogram')

plt14.set(xlabel = 'Car Range', ylabel='Frequency') xs=cars\_data['carsrange'].unique() ys=cars\_data['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)



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 pairplots to understand the distribution of data points better. Hence, lets make a function scatterplot to use when we need to.

def scatterplot(df,var):

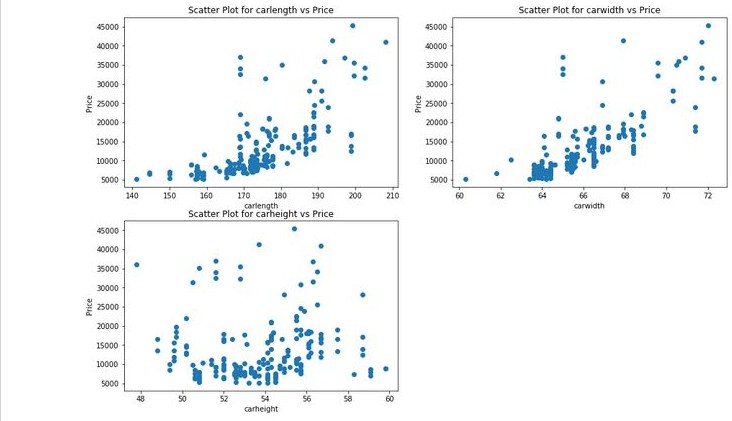
plt.scatter(df[var],df['price']) plt.xlabel(var); plt.ylabel('Price') plt.title('Scatter Plot for '+var+' vs Price')

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

### Car Length, Width and Height

plt.figure(figsize=(15,20)) plt.subplot(4,2,1) scatterplot(cars\_data,'carlength') plt.subplot(4,2,2) scatterplot(cars\_data,'carwidth') plt.subplot(4,2,3) scatterplot(cars\_data,'carheight') plt.show()

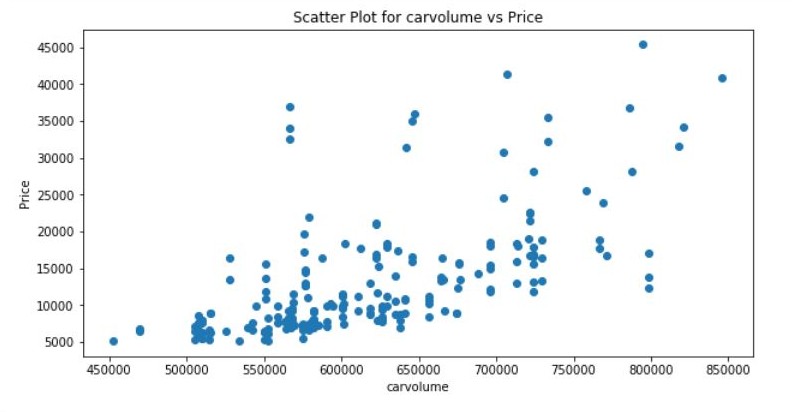
plt.tight\_layout()



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*.

### 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:



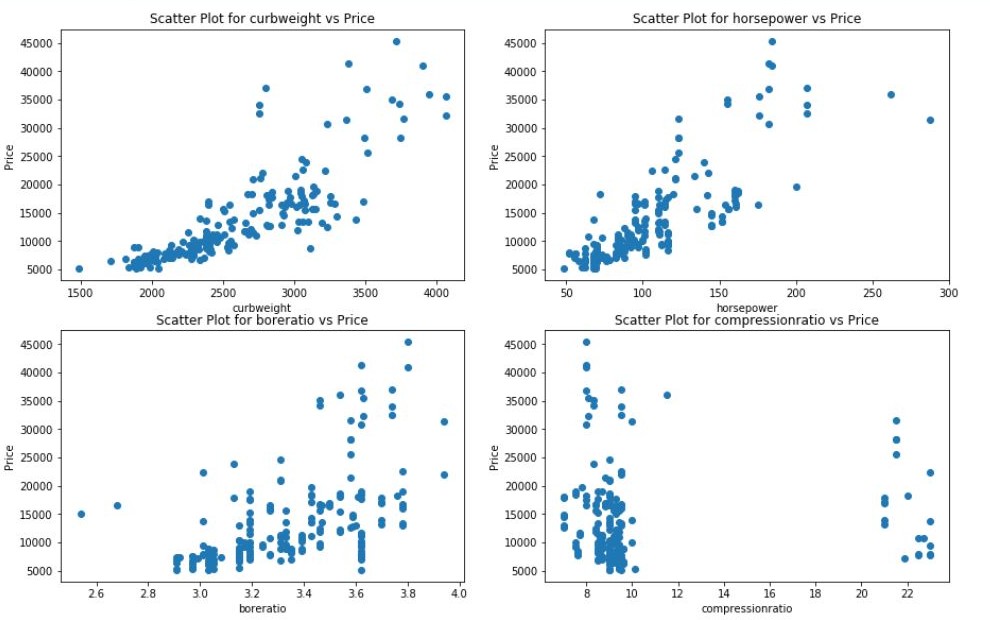
Car volume also seems to have a trend with price.

### 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:

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

plt.tight\_layout()

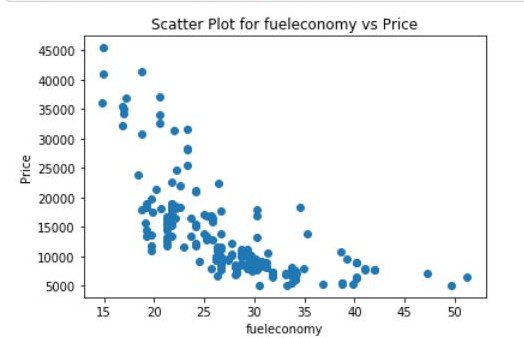


Clearly, Curb Weight, Horsepower and Bore Ratio have a significant trend with price. Compression Ratio does not affect price that much.

### 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:

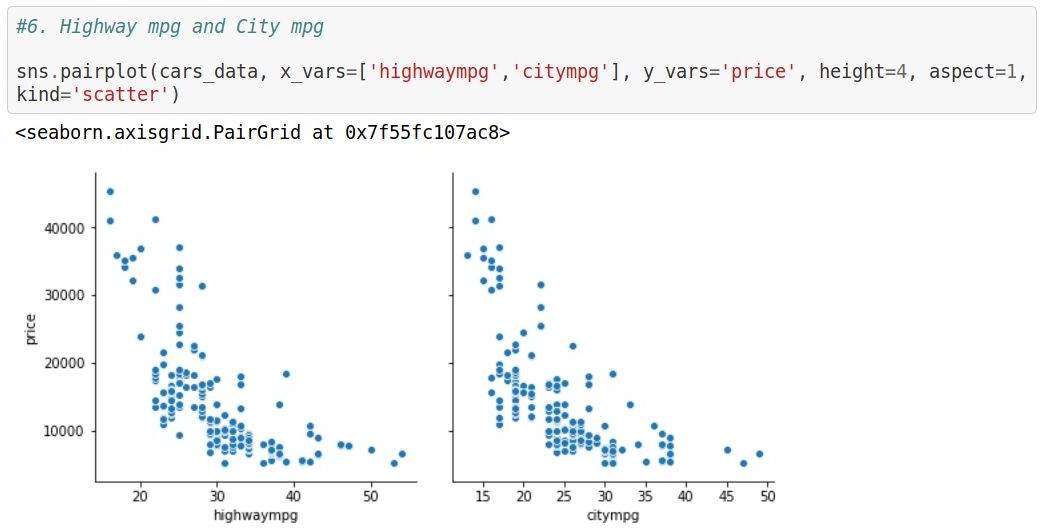
cars\_data['fueleconomy']=(cars\_data['citympg']\*0.55)+(cars\_data['high waympg']\*0.45)

cars\_data['fueleconomy'].unique() scatterplot(cars\_data,'fueleconomy')

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

### 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 pairplot for them. The code and output are:

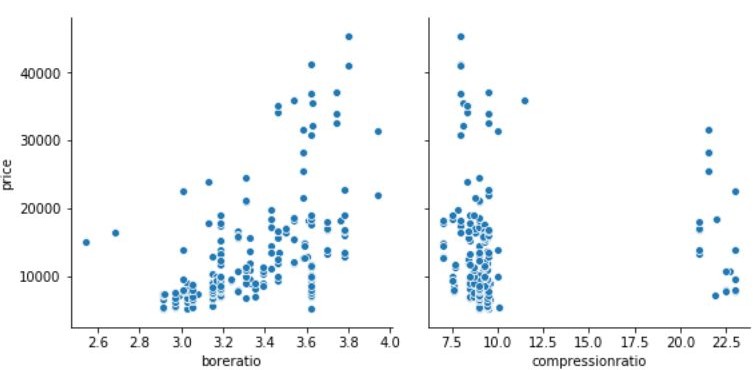


This clearly shows a negative correlation with price for both highway mpg and city mpg. Hence, these two are significant variables.

### 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:

sns.pairplot(cars\_data,x\_vars=['boreratio','compressionratio'], y\_vars='price', height=4, aspect=1, kind='scatter')



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

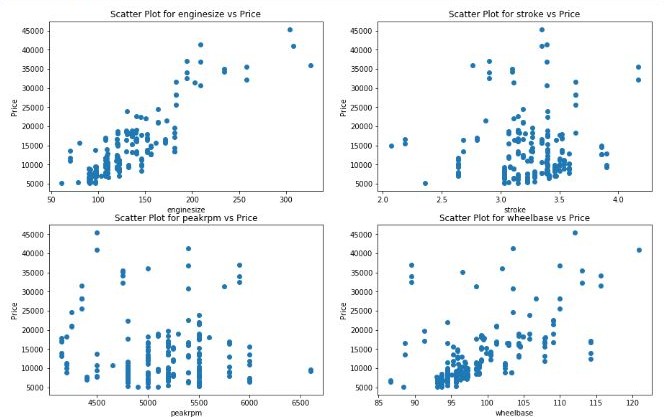
### Engine Size, Stroke, RPM and Wheel Base

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:

plt.figure(figsize=(15,20)) plt.subplot(4,2,1) scatterplot(cars\_data,'enginesize') plt.subplot(4,2,2) scatterplot(cars\_data,'stroke') plt.subplot(4,2,3)

scatterplot(cars\_data,'peakrpm') plt.subplot(4,2,4) scatterplot(cars\_data,'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.

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

corr result=[]

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')

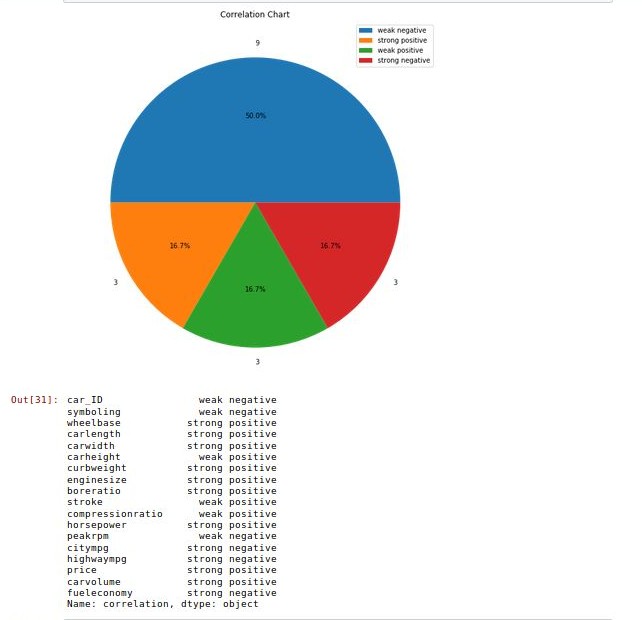
elif(i>0 and i<0.2): result.append('weak positive') elif(i>0.2 and i<0.5): result.append('moderate positive') else : result.append('strong positive')

corr['correlation']=result corr['correlation'].value\_counts()

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)

plt.show() corr.loc[:,'correlation']

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.

# 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?

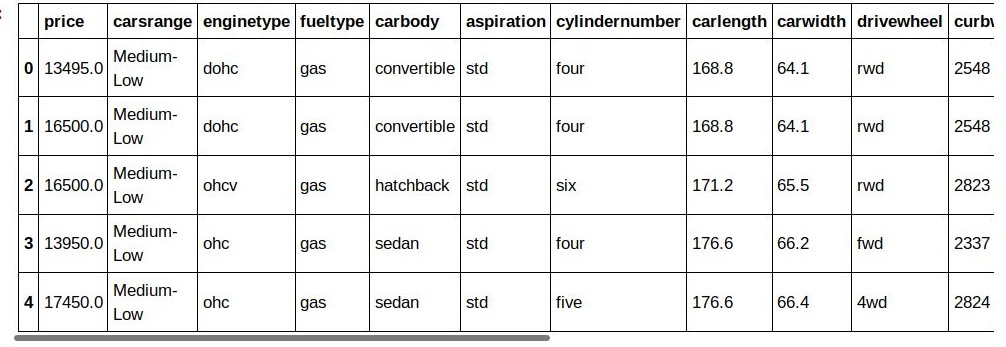
*A statistical model is usually specified as a mathematical relationship between one or more random variables and other non-random variables. As such, a statistical model is "a formal representation of a theory.”*

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:

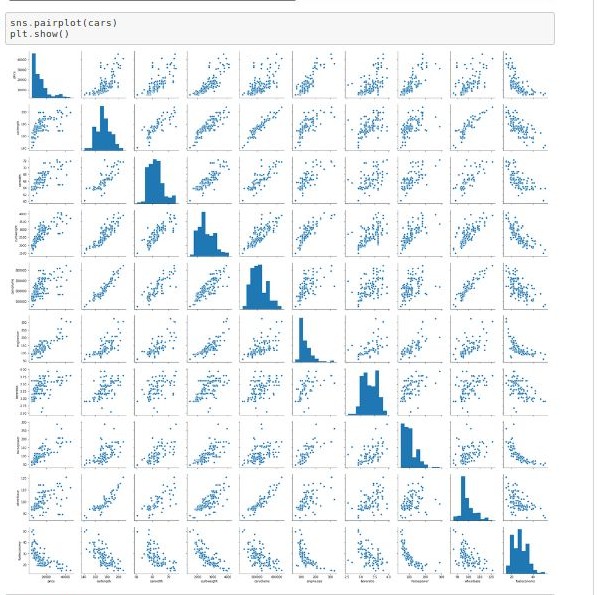
cars=cars\_data[['price','carsrange','enginetype','fueltype','carbody'

,'aspiration','cylindernumber','carlength','carwidth','drivewheel','c urbweight','carvolume','enginesize','boreratio','horsepower','wheelba se','fueleconomy']]

cars.head()



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



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.

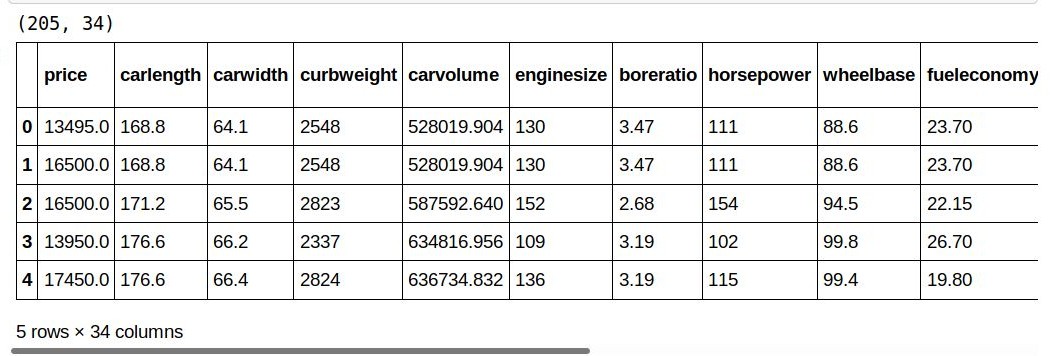
def dummies(x,df): var=pd.get\_dummies(df[x], drop\_first=True)

df=pd.concat([df,var], axis=1) df.drop([x], axis=1, inplace=True) return df

cars = dummies('fueltype',cars) cars = dummies('aspiration',cars) cars = dummies('carbody',cars) cars = dummies('drivewheel',cars) cars = dummies('enginetype',cars)

cars = dummies('cylindernumber',cars) cars = dummies('carsrange',cars)

print(cars.shape) cars.head()



(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.

from sklearn.model\_selection import train\_test\_split np.random.seed(0)

df\_train, df\_test=train\_test\_split(cars, train\_size=0.6, test\_size=0.4, random\_state=100)

df\_train.head() #training set df\_test.head() #test set

from sklearn.preprocessing import MinMaxScaler #feature scaling scaler=MinMaxScaler()

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()

low\_corr=df\_train.corr().loc[df\_train.corr()['price']<-0.45]['price'] low=low\_corr.index.tolist()

num\_vars=high+low num\_vars

df\_train[num\_vars] = scaler.fit\_transform(df\_train[num\_vars])

df\_train.head() df\_train.describe().round(2)

#splitting into x and y y\_train=df\_train.pop('price') x\_train=df\_train

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.

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LinearRegression from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

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()

def buildmodel(x,y): x=sm.add\_constant(x) model=sm.OLS(y,x).fit() print(model.summary()) return x

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

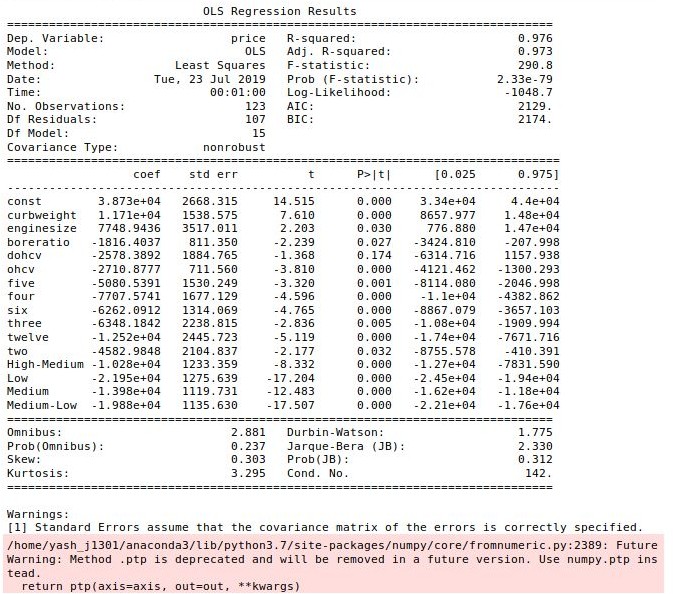
is continuous, as I mentioned in the ‘Project Data Introduction’, and we have to predict continuous values.

Now, let’s start running our models.

### RUNNING REGRESSION MODELS

### Model 1

model\_1=buildmodel(x\_train\_new,y\_train)



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. Everytime 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.

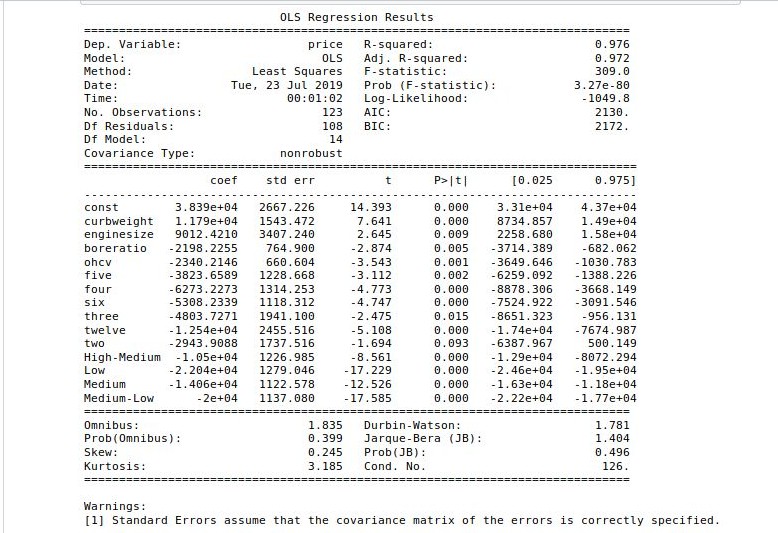
Now, in the report, *‘dohcv’* is having p>0.05. Let us remove this and run a new model.

x\_train\_new=x\_train\_new.drop(['dohcv'], axis=1)

### Model 2

model\_2=buildmodel(x\_train\_new, y\_train)

Lets run the model and see the report now.



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

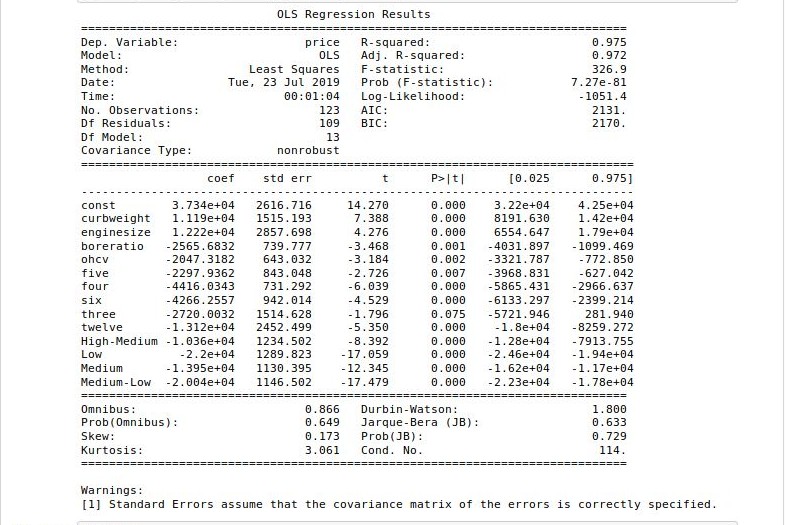
x\_train\_new=x\_train\_new.drop(['two'],axis=1)

Now, create a new model and run it.

### Model 3

model\_3=buildmodel(x\_train\_new, y\_train)

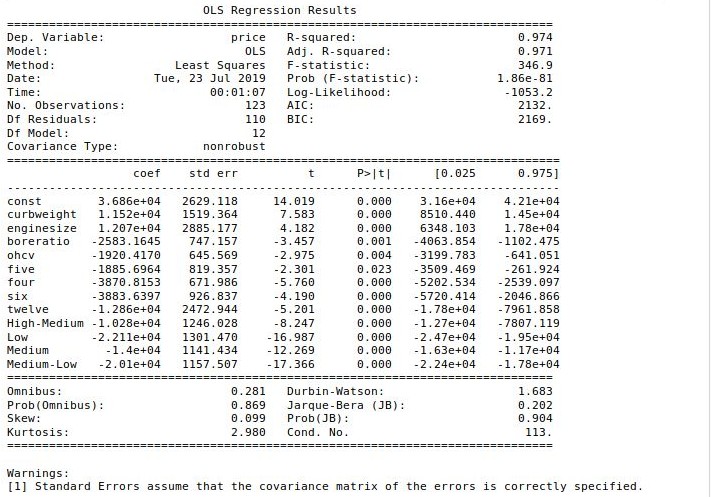
Let us run this model and check our results.



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

x\_train\_new=x\_train\_new.drop(['three'],axis=1)

Now, create a new model and run it.

**Model 4** model\_4=buildmodel(x\_train\_new,y\_train) Lets run this model and check the results.

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*.

def checkVIF(x):

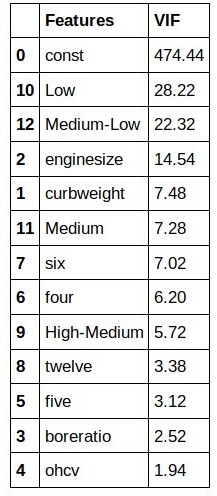
vif = pd.DataFrame() vif['Features'] = x.columns

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)

checkVIF(model\_4)



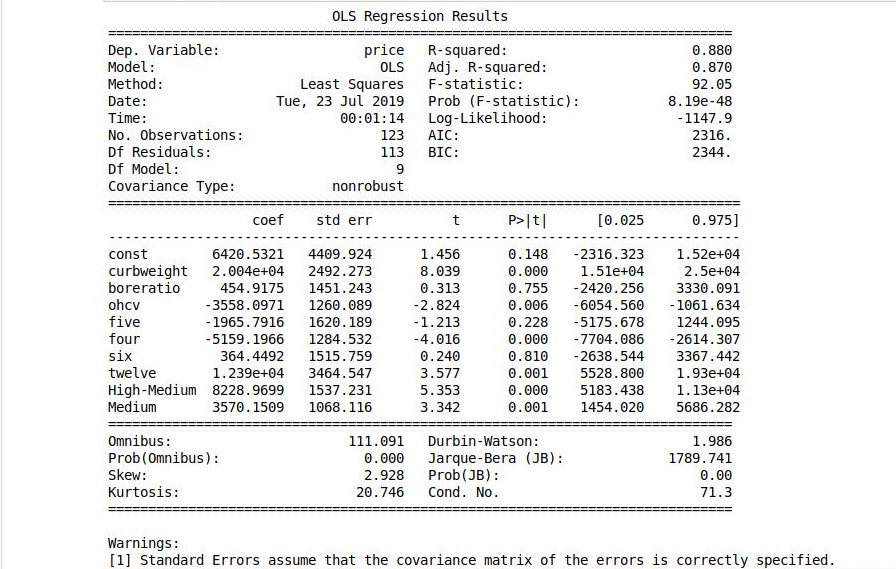
The VIF Value for *Low, Medium-Low* and *enginesize* is very high. Let us drop it and run a new model. Remember, check the p-value and the vif value both.

model\_new=model\_4.drop(['Low','Medium-Low','enginesize'], axis=1) model\_5=buildmodel(model\_new, y\_train) #checking OLS Results checkVIF(model\_5) #checking vif value

Let us see the VIF Value and Regression Results:



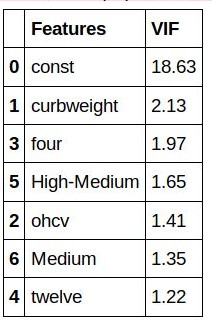
VIF Value is not very high. Let us check the regression results:



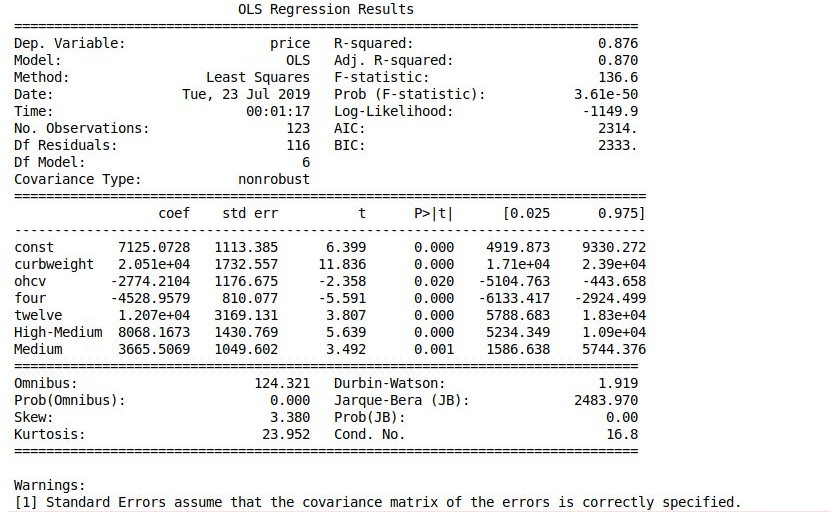
Looks like the p-value of boreratio, five and six is greater than 0.05. Let us remove them and run a new model.

model\_new=model\_new.drop(['boreratio','five','six'], axis=1) model\_6=buildmodel(model\_new,y\_train)

checkVIF(model\_6)



VIF Value is under control. Let us check the regression results:



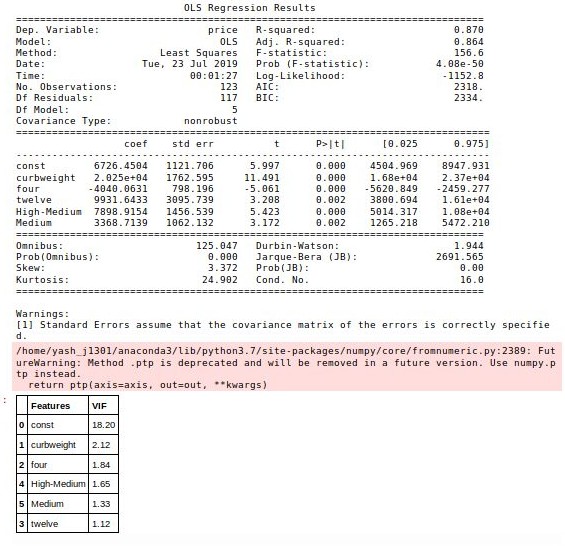
p-value is under control. Hence, it is our regression model. Let us name it as final\_rm.

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.

model\_check=model\_6.drop(['ohcv'], axis=1) model\_check=buildmodel(model\_check, y\_train) checkVIF(model\_check)

Now, lets check the VIF value and regression results.



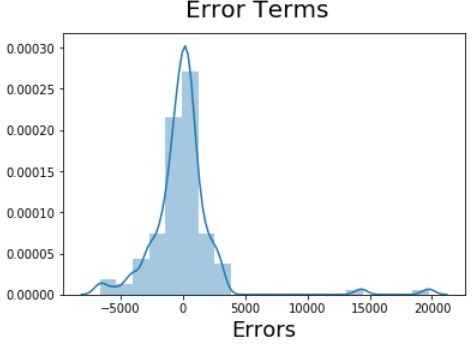
Both the p-value and vif values are under control. Hence, there is not much error. Let us plot a graph to just be sure.

lm=sm.OLS(y\_train,model\_check).fit() y\_train\_price=lm.predict(model\_check)

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) plt.show()



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.

# 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.

#selecting the highly correlated values

df\_test[num\_vars] = scaler.fit\_transform(df\_test[num\_vars])

#splitting into x and y y\_test=df\_test.pop('price') x\_test=df\_test

Now that we have declared our test variables, it is time to predict. Let us use our model to predict.

52

X\_train\_new = model\_check.drop('const',axis=1)

# Creating X\_test\_new dataframe by dropping variables from X\_test X\_test\_new = x\_test[X\_train\_new.columns]

# 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.

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)

c= [i for i in range(1,83,1)] # generating index fig = plt.figure(figsize=(15,5))

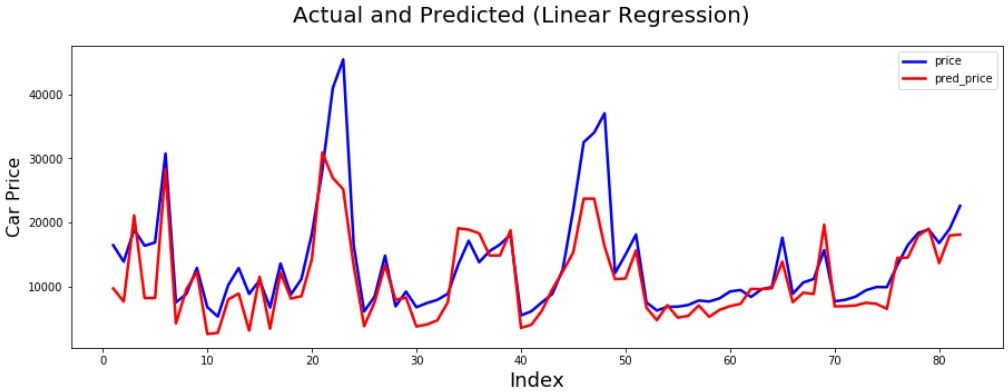
plt.plot(c,price['price'], color="blue", linewidth=2.5, linestyle="-") #Plotting Actual plt.plot(c,price['pred\_price'], color="red", linewidth=2.5, linestyle="-") #Plotting predicted

fig.suptitle('Actual and Predicted (Linear Regression)', fontsize=20) # Plot heading

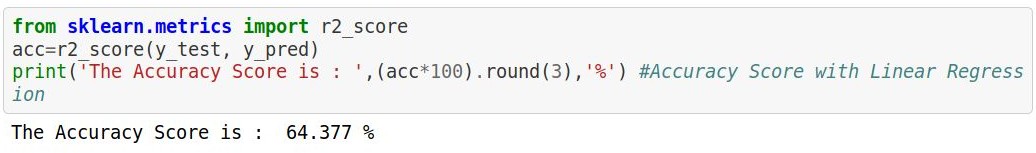
plt.xlabel('Index', fontsize=18) #

X-label

plt.ylabel('Car Price', fontsize=16) # Y-label plt.legend()

plt.show()

This is the variation in the predicted price. Barring a few irregularities, the prediction seems to work fine. Let’s check how accurate we are.



**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()

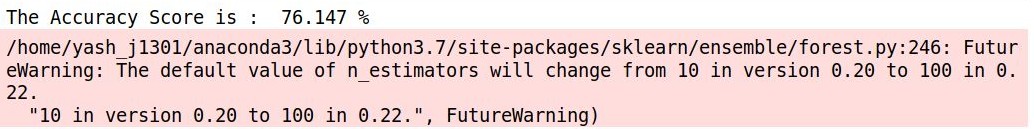
rf.fit(x\_train,y\_train)

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 Forest Regressor



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.

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 predicted

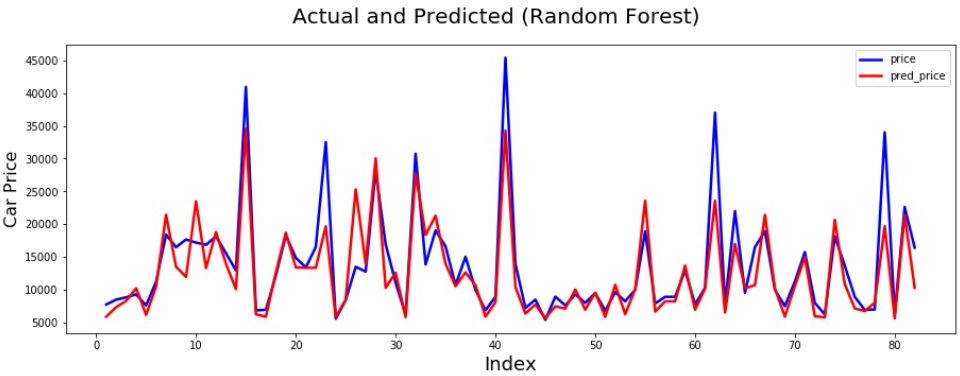
fig.suptitle('Actual and Predicted (Random Forest)', fontsize=20) # Plot heading

plt.xlabel('Index', fontsize=18) #

1. label

plt.ylabel('Car Price', fontsize=16) # Y-label plt.legend()

plt.show()

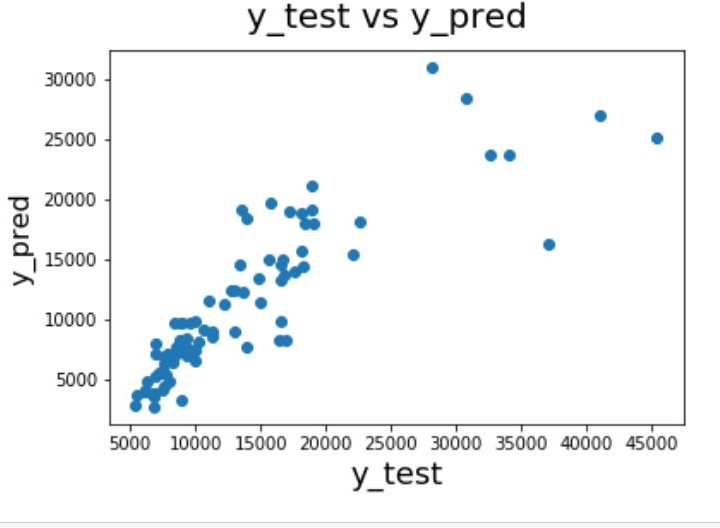


Not bad, though. This was our predictive analysis model. Let us check the price spread between actual and predicted price for both random forest and linear regression.

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)



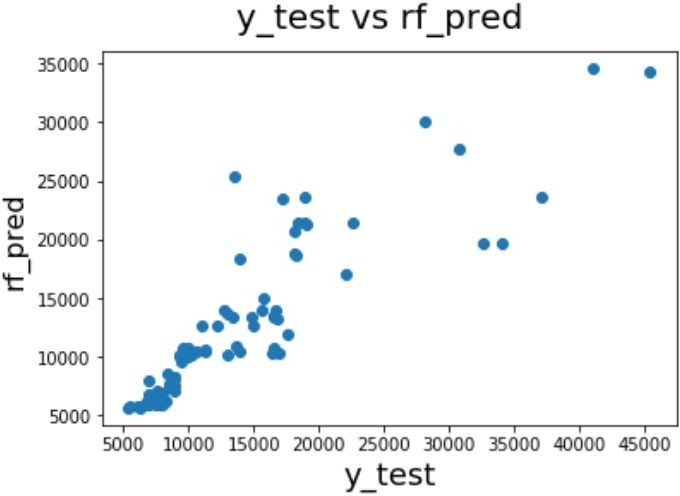
Now, lets check with random forest. This would be the final part of the project.

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



***Bibliography***

* + <https://arxiv.org/pdf/1711.06970.pdf>
  + [https://github.com/akjadon/Finalprojects\_DS/tree/master/Car\_pricing\_pred iction](https://github.com/akjadon/Finalprojects_DS/tree/master/Car_pricing_prediction)
  + <https://en.wikipedia.org/wiki/Dummy_variable_(statistics>)
  + <https://en.wikipedia.org/wiki/Linear_regression>
  + <https://en.wikipedia.org/wiki/Variance_inflation_factor>
  + [https://www.kaggle.com/goyalshalini93/car-price-prediction-linear-regressi on-rfe](https://www.kaggle.com/goyalshalini93/car-price-prediction-linear-regression-rfe)
  + <https://en.wikipedia.org/wiki/Random_forest>
  + <https://www.kaggle.com/jshih7/car-price-prediction>