Project1: Car Price Prediction

Project2: HR Analytics

24/07/2019

Arunabh Borah
Henry Harvin Education
C108, Sector 2, Noida

Overview(Project1)

In Car Price Prediction we try to predict the price of second-hand cars. There are many factors that influence the price of a car in the second-hand market.

Goals

1. Data understanding and exploration
2. Data cleaning and preparation
3. Data visualization
4. Model building and evaluation

Explanation

Understanding the data

Here we will import different libraries that would be needed to carry out analysis

```
#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

cars = pd.read_csv('https://raw.githubusercontent.com/akjadon/Finalprojects_DS/master/Car_pricing_prediction/CarPrice_Assignment
cars.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	st
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

Describing the data

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

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	citymp
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.(

Data Cleaning

The ‘CarName’ column contains both car and company name. We need to split the two into two different column

```
#2. Data Cleaning

#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()
```

	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

```
#Checking the columns created
cars['Company'].unique()
cars['CarModel'].unique()
```

```
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',
```

Replacing the incorrect Company name with correct name

```
#Replace incorrect car name with correct name
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)

cars['Company'].unique()

#Here we have changed the company name but there might be a problem of lower and upper case,e.g,'Mazda' and 'mazda'
```

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche', 'renault',
       'saab', 'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

To maintain uniformity we will convert it to lower frame

```
#Convert all the string data to lower to avoid any case difference errors
cars['Company']=cars['Company'].str.lower()
cars['Company'].unique()
```

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

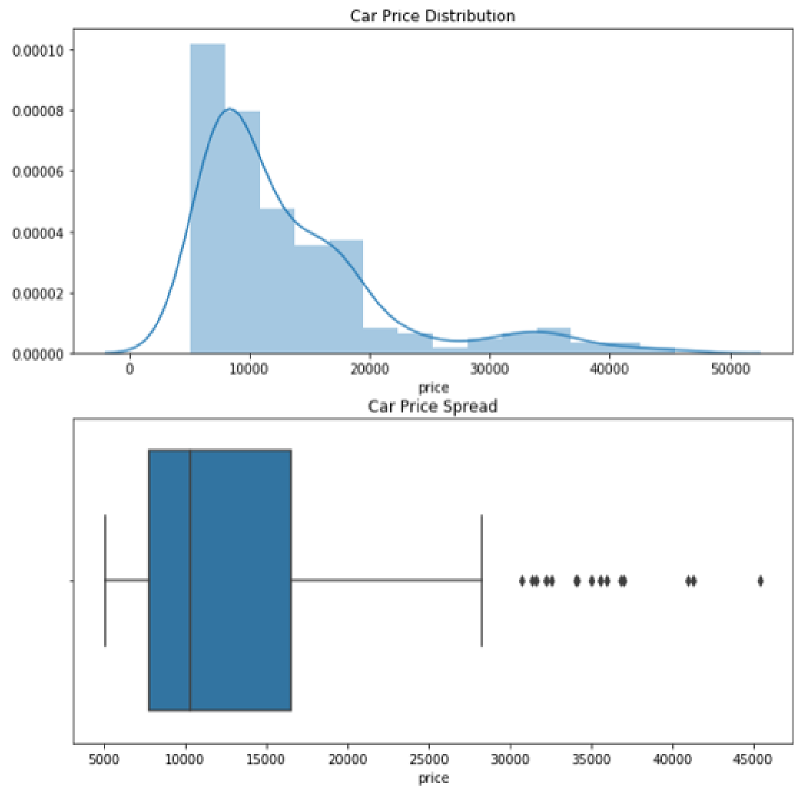
```
#checking for duplicate values
cars.loc[cars.duplicated()]

#checking columns
column_names=cars.columns.tolist()
column_names
```

```
['car_ID',
 'symboling',
 'CarName',
 'fueltype',
 'aspiration',
 'doornumber',
 'carbody',
 'drivewheel',
 'enginelocation',
 'wheelbase',
 'carlength',
 'carwidth',
 'carheight',
 'curbweight',
 'enginetype',
 'cylindernumber',
```


Data Visualization

To have in-depth insight we try to plot various attributes

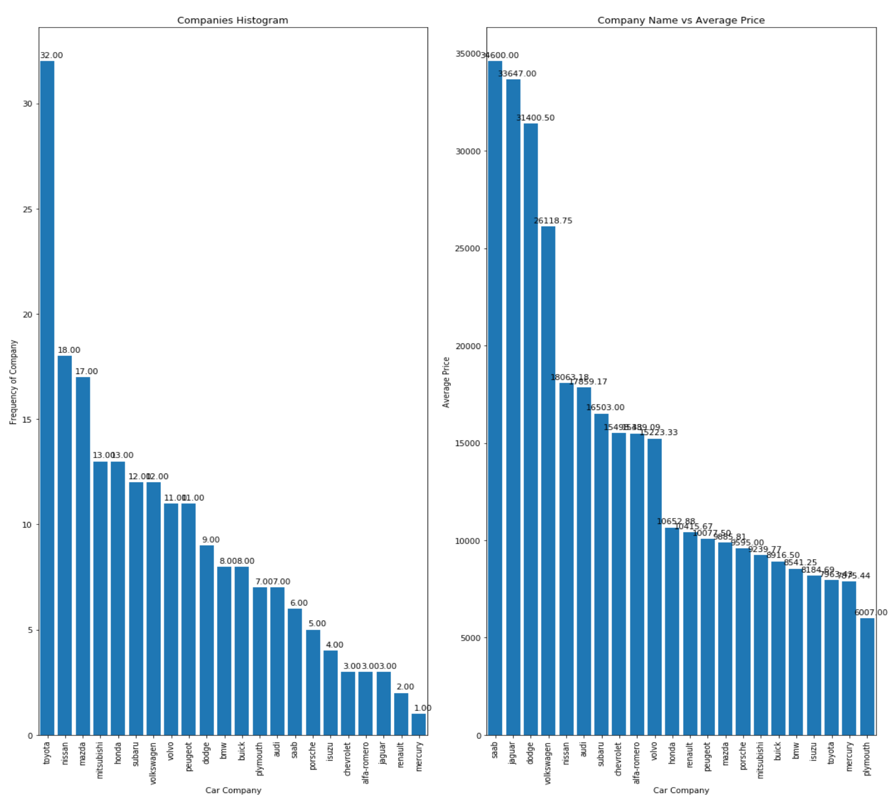
```
#3. Visualizing the Data

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') #title for the chart
sns.distplot(cars['price']) #distplot for price of cars

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



Visualizing categorical data

```
"""

    Categorical Data(Object)

    - Company              #
    - Symboling            #
    - fueltype             #
    - enginetype           #
    - carbody              #
    - doornumber           #
    - enginelocation       #
    - fuelsystem           #
    - cylindernumber       #
    - aspiration           #
    - drivewheel           #


"""
```

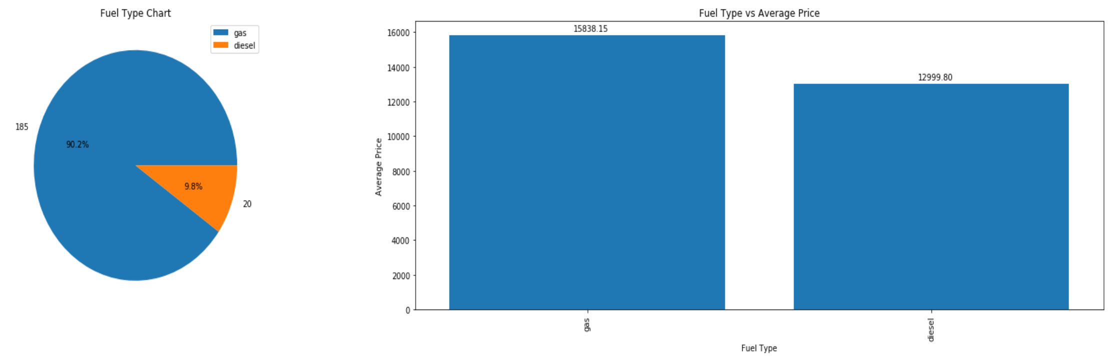
```
#1. Car Company

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

#plot 1.1
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)

#plot 1.2
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()
```



Viewing the most frequently preferred company and its comparison with average price

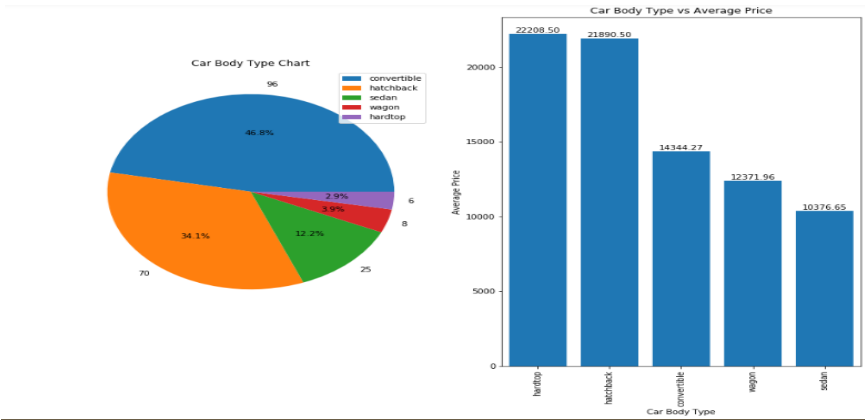
Visualizing Fuel type v/s Average price

```
#2. Fuel Type

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

#plot 2.1
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)

#plot 2.2
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()
```



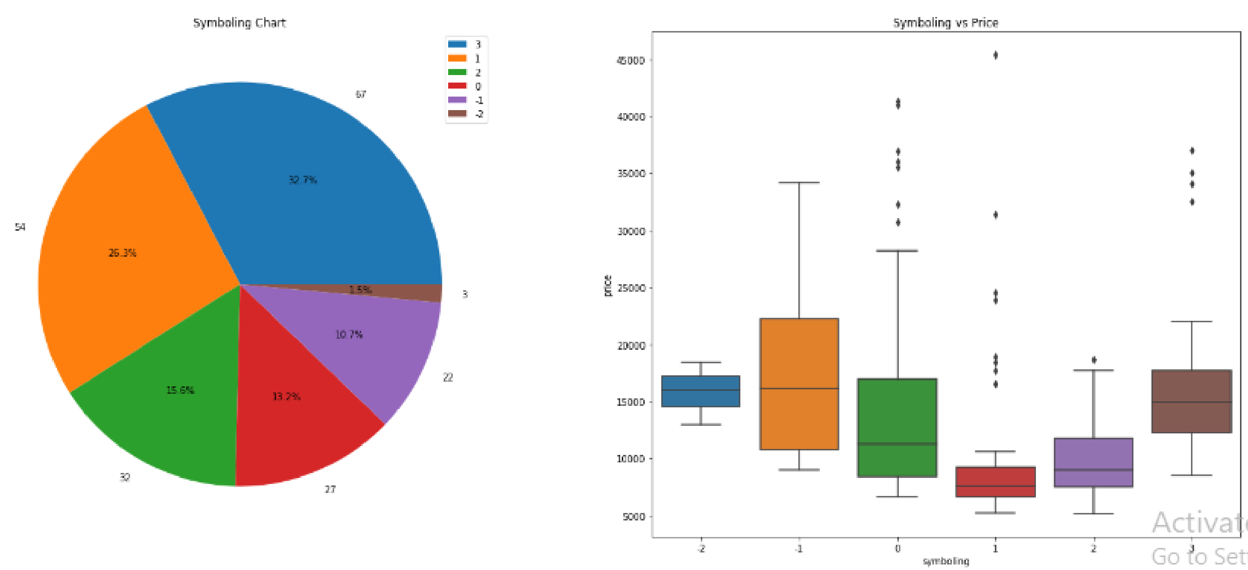
Visualizing car body type v/s average price

```
#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)

#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()
```



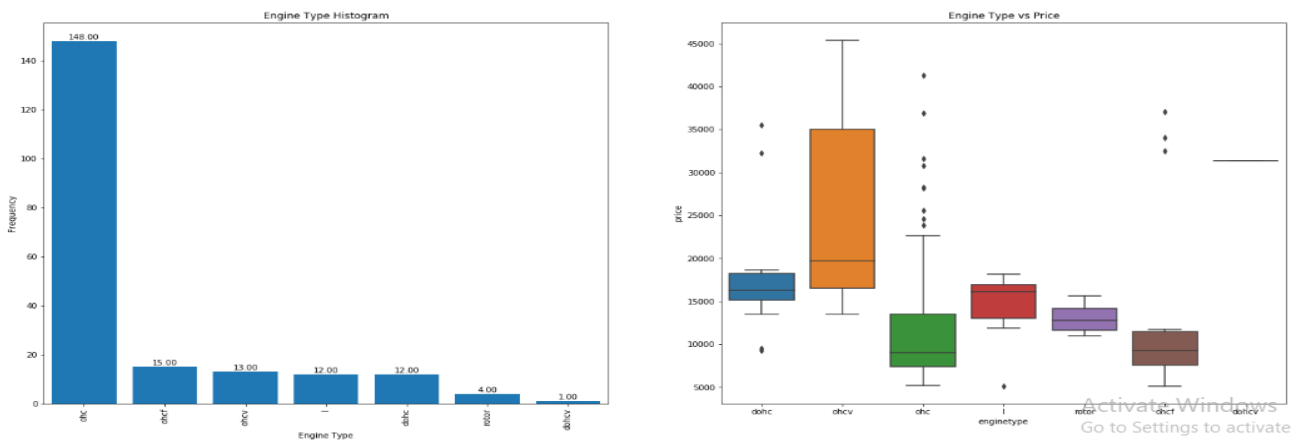
Visualizing Symboling v/s Price

```
#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)

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



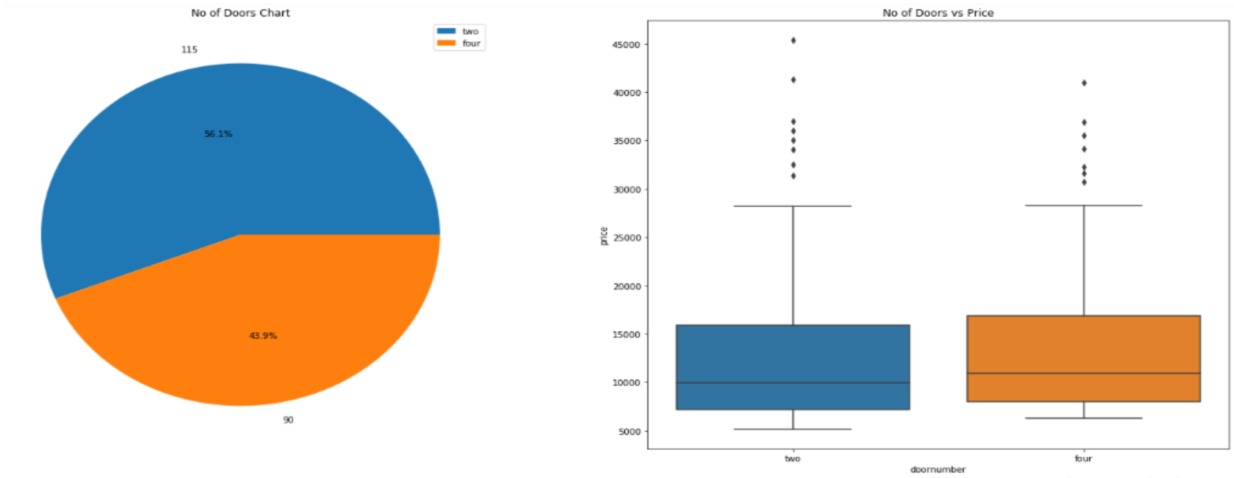
Visualizing Engine type v/s price

```
#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)

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



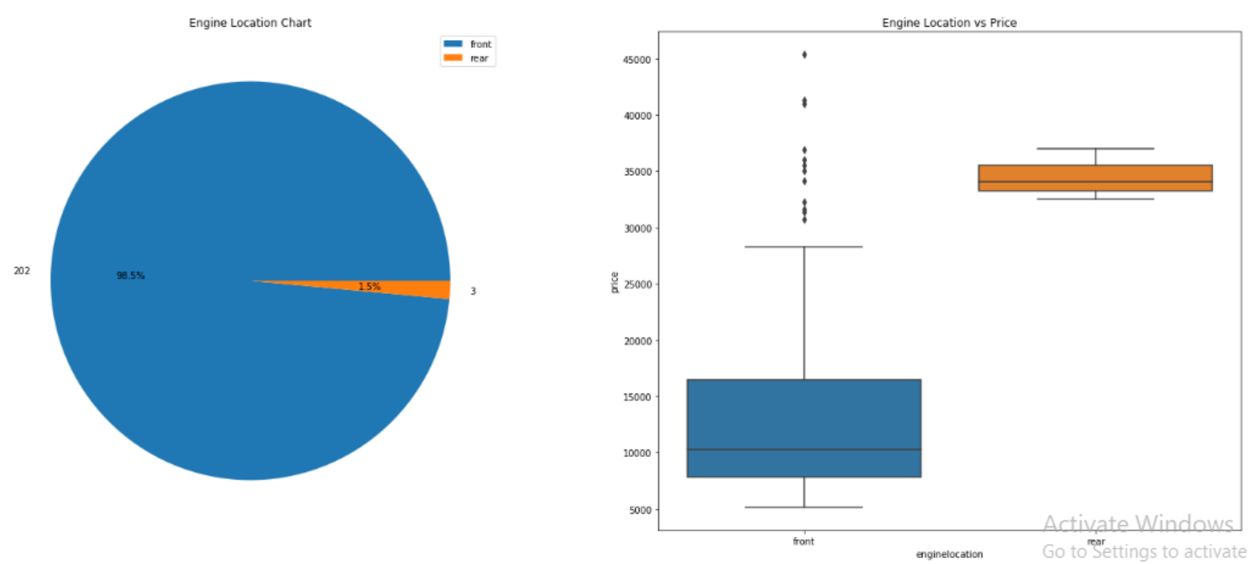
Visualizing relationship of number of doors v/s price

```
#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)

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



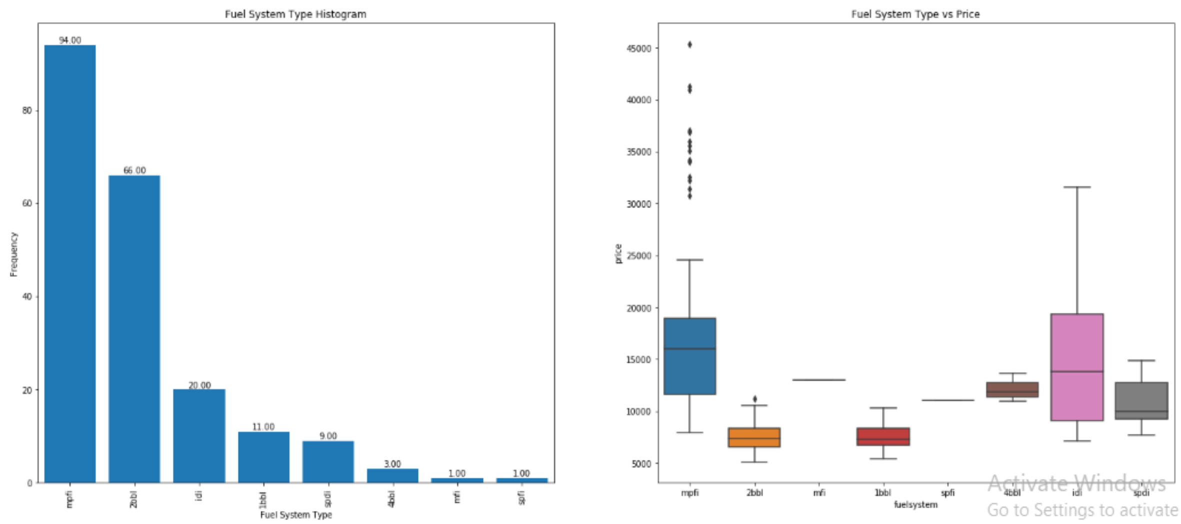
Mapping Engine location v/s price

```
#7. 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)

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



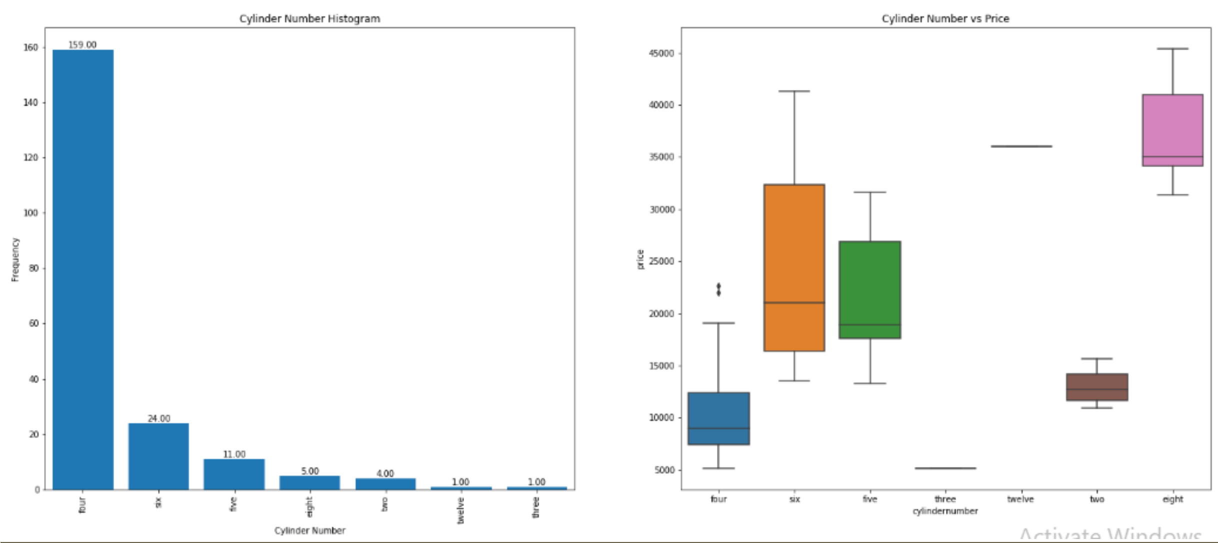
Fuel System type v/s price plotting

```
#8. 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)

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



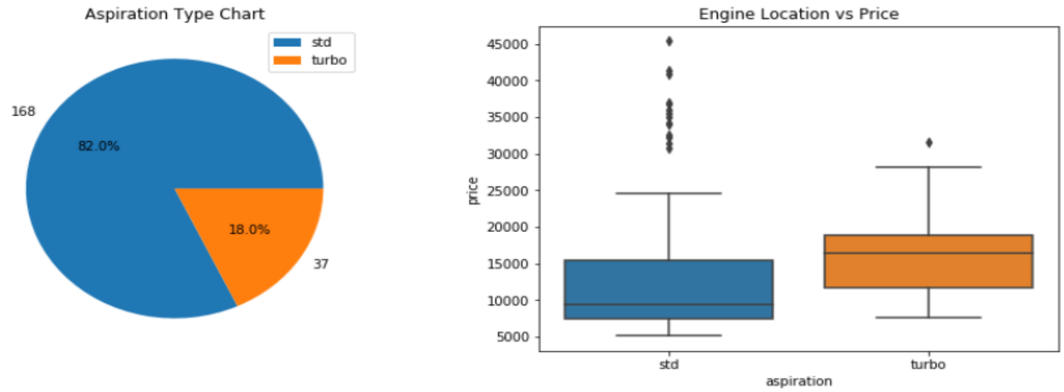
Visualizing Cylinder Number

```
#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)

#plot 2
plt.subplot(1,2,2)
plt.title('Cylinder Number vs Price')
sns.boxplot(x=cars['cylindernumber'], y=cars['price'])
plt.show()
```



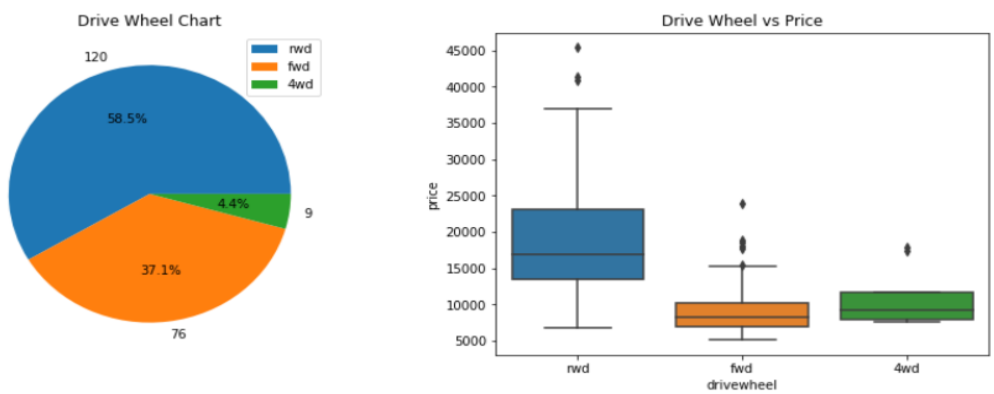
Pie plot of Aspiration

```
#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)

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



Pie plot of drivewheel

```
#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)

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



Visualization of Numerical Value

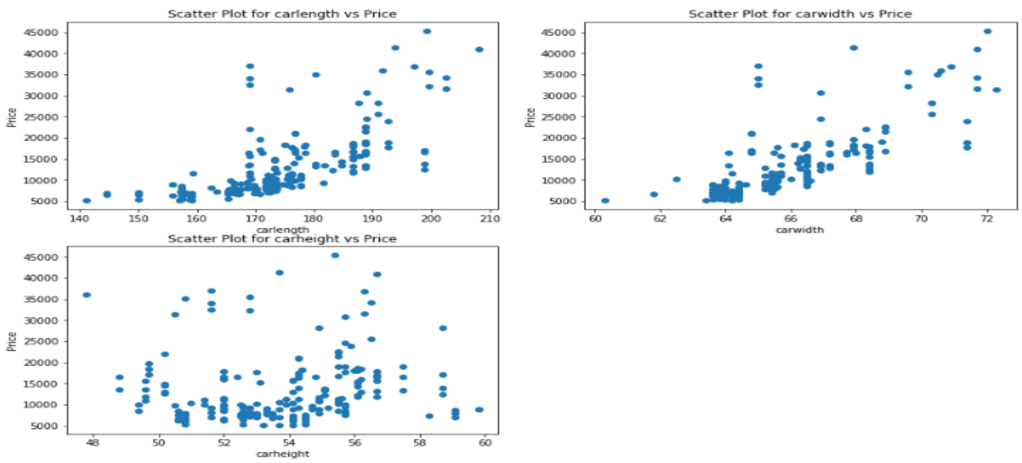
Numerical values will be plotted through scatter plots

```
Numerical Variables

-Car Length                                        #
-Car Width                                         #
-Car Height                                        #
-Curb Weight                                       #
-Horsepower                                        #
-Bore Ratio                                        #
-Compression Ratio                                 #
-Highway miles per gallon (mpg)                    #
-Engine Size                                       #
-Stroke                                            #
-City Miles per gallon (mpg)                       #
-Peak Revolutions per Minute (rpm)                 #
-Wheel Base                                        #
```

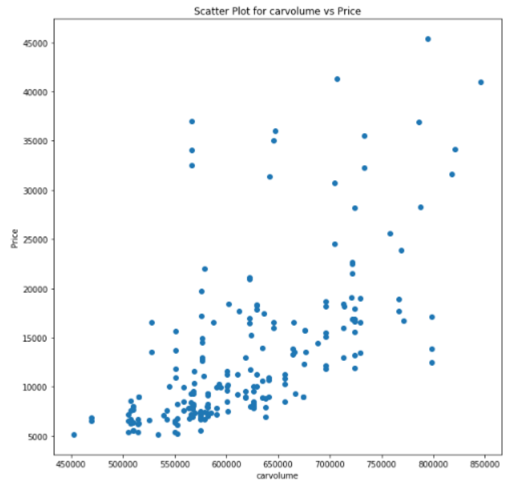
Defining variable (price) inside scatter plot function to be viewed

```
def scatterplot(df,var):

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

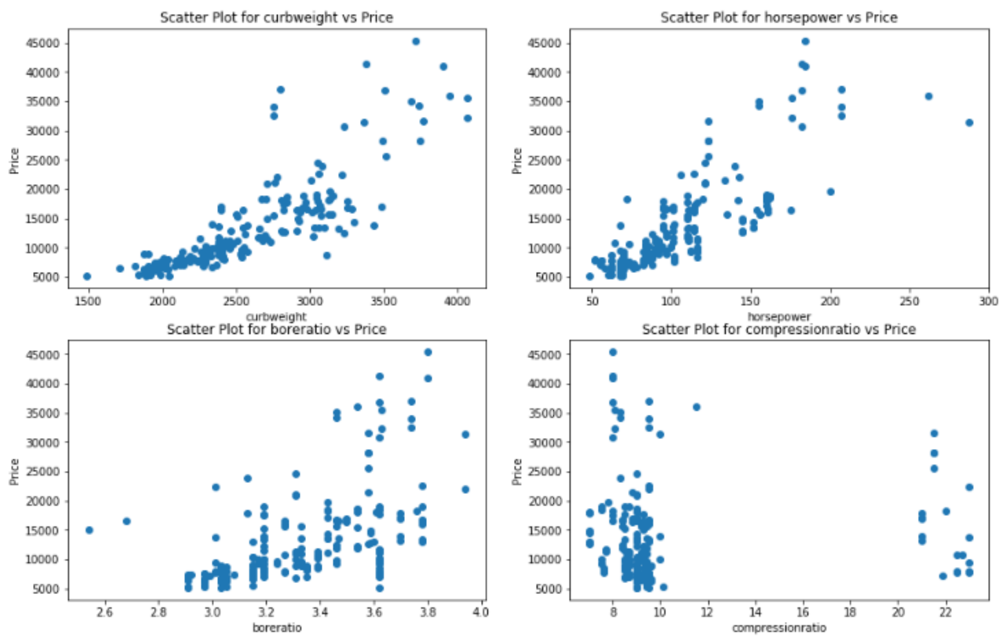
```
#1. Car Length, Width and Height

plt.figure(figsize=(15,20))
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()
```



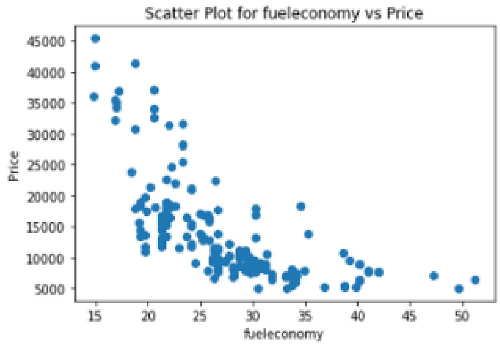

```
#2. Creating a new variable- Car Volume
cars['carvolume']=cars['carlength']*cars['carwidth']*cars['carheight']
cars['carvolume'].unique()
plt.figure(figsize=(10,10))
scatterplot(cars,'carvolume')
```




```
#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()
```

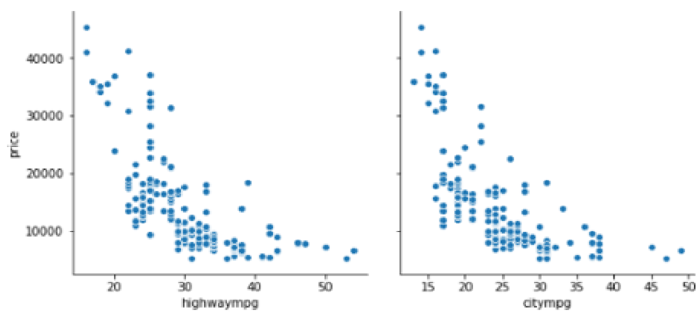


```
#4. Creating a new Variable - Fuel Economy

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



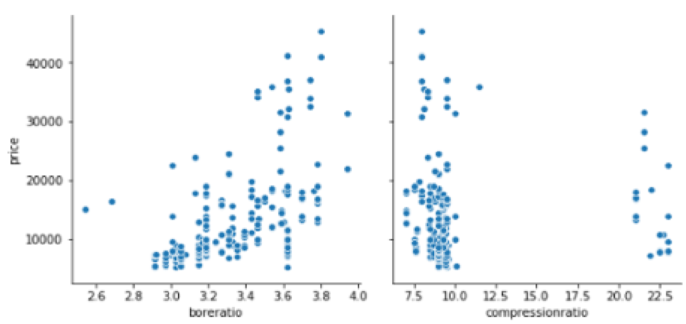

```
#5. Highway mpg and City mpg

sns.pairplot(cars, x_vars=['highwaympg','citympg'], y_vars='price', height=4, aspect=1, kind='scatter')
```

```
<seaborn.axisgrid.PairGrid at 0x1f5e8f6a518>
```

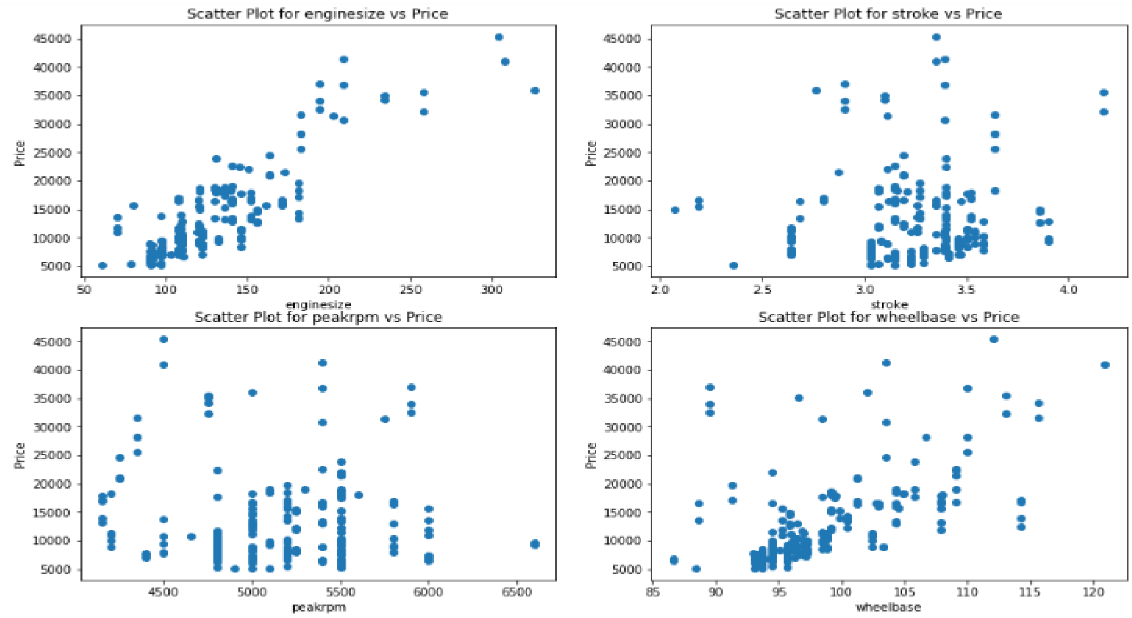



```
#6. Bore Ratio and Compression Ratio

sns.pairplot(cars, x_vars=['boreratio','compressionratio'], y_vars='price', height=4, aspect=1, kind='scatter')
```

```
<seaborn.axisgrid.PairGrid at 0x1f5e6ea6208>
```

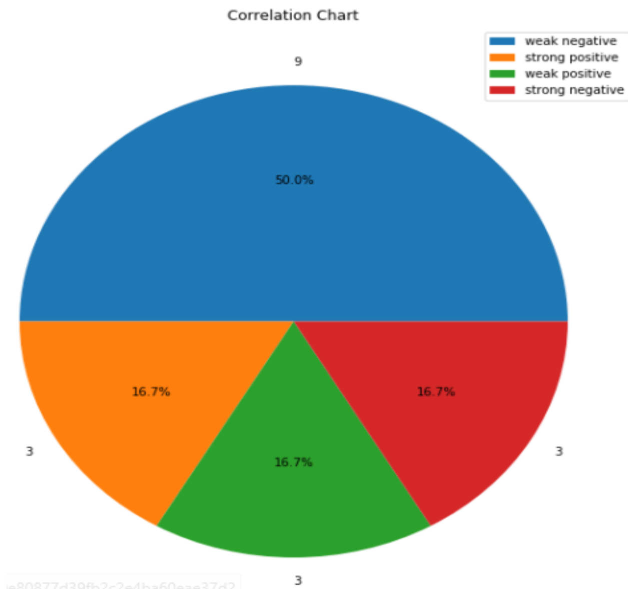



```
#8. Engine Size, Stroke, RPM and Wheelbase

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()
```




```
#Correlation with price(target variable) for numeric data

corr=cars.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)

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

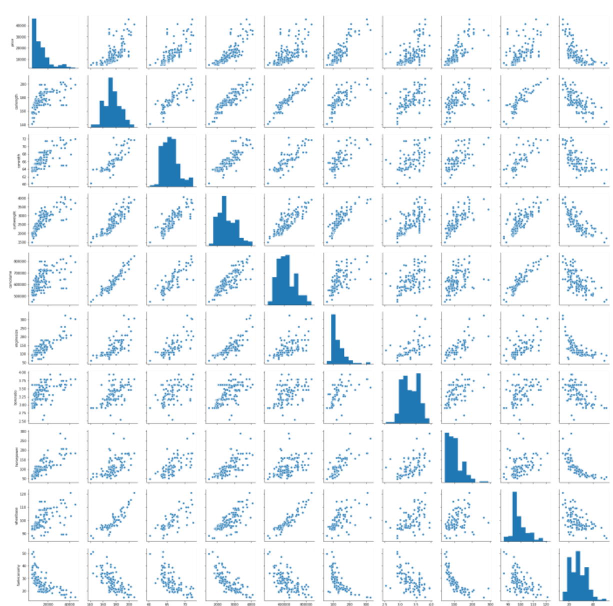
```
car_ID                 weak negative
symboling              weak negative
wheelbase            strong positive
carlength            strong positive
carwidth             strong positive
carheight              weak positive
curbweight           strong positive
enginesize           strong positive
boreratio            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
```



Regression analysis

Creating a DataFrame **('car1')** containing desired columns: price,carsrange,enginetype,fueltype,carbody,aspiration,cylindernumber,carlength,carwidth,drivewheel,curbweight,carvolume,enginesize,boreratio,horsepower,wheelbase,fueleconomy

```
sns.pairplot(cars1)
plt.show()
```



To avoid alteration in cars dataset, we create dummy dataset. We need to create dummy variable for categorical variables only.

```
#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

cars1 = dummies('fueltype',cars1)
cars1 = dummies('aspiration',cars1)
cars1 = dummies('carbody',cars1)
cars1 = dummies('drivewheel',cars1)
cars1 = dummies('enginetype',cars1)
cars1 = dummies('cylindernumber',cars1)
cars1 = dummies('carsrange',cars1)

cars1.shape
cars1.head()
```

	price	carlength	carwidth	curbweight	carvolume	enginesize	boreratio	horsepower	wheelbase	fueleconomy	...	five	four	six	three	twelve	two	Me
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	

Spliting data into train and test

```
#Train-Test Splitg

from sklearn.model_selection import train_test_split
np.random.seed(0)
df_train, df_test=train_test_split(cars1, train_size=0.6, test_size=0.4, random_state=100)

df_train.head() #training set
df_test.head() #test set
```

```
#splitting into x and y
y_train=df_train.pop('price')
x_train=df_train
```

Model Building

```
#4. Model Building

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_rfe=x_train[index]
x_train_rfe.head()

def buildmodel(x,y):
    x=sm.add_constant(x)
    model=sm.OLS(y,x).fit()
    print(model.summary())
    return x
```

Building a model to try on OLS (ordinary least square) and reject variables with p>0.05

```
#Running Regression Models

#Model 1
model_1=buildmodel(x_train_rfe,y_train)

x_train_new=x_train_rfe.drop(['enginesize','boreratio'], axis=1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.974
Model:                            OLS   Adj. R-squared:                  0.970
Method:                 Least Squares   F-statistic:                     267.6
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           1.77e-77
Time:                        12:03:30   Log-Likelihood:                 237.39
No. Observations:                 123   AIC:                            -442.8
Df Residuals:                     107   BIC:                            -397.8
Df Model:                          15
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.7222      0.046     15.814      0.000       0.632       0.813
curbweight      0.3084      0.061      5.043      0.000       0.187       0.430
enginesize     -0.0321      0.066     -0.486      0.628      -0.163       0.099
boreratio      -0.0404      0.025     -1.628      0.106      -0.090       0.009
horsepower      0.0947      0.050      1.886      0.062      -0.005       0.194
wheelbase       0.0810      0.039      2.057      0.042       0.003       0.159
hardtop        -0.0765      0.037     -2.071      0.041      -0.150      -0.003
hatchback      -0.0942      0.026     -3.601      0.000      -0.146      -0.042
sedan          -0.0784      0.027     -2.955      0.004      -0.131      -0.026
wagon          -0.1121      0.028     -4.000      0.000      -0.168      -0.057
four           -0.0529      0.013     -4.102      0.000      -0.078      -0.027
twelve         -0.1530      0.053     -2.907      0.004      -0.257      -0.049
High-Medium    -0.2236      0.031     -7.191      0.000      -0.285      -0.162
Low            -0.6087      0.036    -16.817      0.000      -0.680      -0.537
Medium         -0.4109      0.031    -13.356      0.000      -0.472      -0.350
Medium-Low     -0.5592      0.032    -17.671      0.000      -0.622      -0.496
==============================================================================
```

Dropping 'Enginesize' and 'boreratio' as both are above p>0.05

Now, dropping horsepower in our next model

```
#Model 2
model_2=buildmodel(x_train_new, y_train)

x_train_new=x_train_new.drop(['horsepower'],axis=1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.973
Model:                            OLS   Adj. R-squared:                  0.970
Method:                 Least Squares   F-statistic:                     305.5
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           2.64e-79
Time:                        12:03:55   Log-Likelihood:                 235.64
No. Observations:                 123   AIC:                            -443.3
Df Residuals:                     109   BIC:                            -403.9
Df Model:                          13
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.7116      0.045     15.900      0.000       0.623       0.800
curbweight      0.2763      0.054      5.121      0.000       0.169       0.383
horsepower      0.0741      0.048      1.545      0.125      -0.021       0.169
wheelbase       0.0818      0.040      2.065      0.041       0.003       0.160
hardtop        -0.0818      0.037     -2.213      0.029      -0.155      -0.009
hatchback      -0.0947      0.026     -3.630      0.000      -0.146      -0.043
sedan          -0.0805      0.026     -3.043      0.003      -0.133      -0.028
wagon          -0.1130      0.027     -4.113      0.000      -0.167      -0.059
four           -0.0621      0.012     -5.335      0.000      -0.085      -0.039
twelve         -0.1513      0.049     -3.113      0.002      -0.248      -0.055
High-Medium    -0.2225      0.031     -7.123      0.000      -0.284      -0.161
Low            -0.5974      0.036    -16.768      0.000      -0.668      -0.527
Medium         -0.4048      0.030    -13.284      0.000      -0.465      -0.344
Medium-Low     -0.5518      0.031    -17.818      0.000      -0.613      -0.490
```

Further, dropping wheelbase

```
#Model 3
model_3=buildmodel(x_train_new, y_train)

x_train_new=x_train_new.drop(['wheelbase'],axis=1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.973
Model:                            OLS   Adj. R-squared:                  0.970
Method:                 Least Squares   F-statistic:                     326.6
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           4.63e-80
Time:                        12:04:27   Log-Likelihood:                 234.31
No. Observations:                 123   AIC:                            -442.6
Df Residuals:                     110   BIC:                            -406.1
Df Model:                          12
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.7292      0.044     16.736      0.000       0.643       0.815
curbweight      0.3237      0.045      7.243      0.000       0.235       0.412
wheelbase       0.0510      0.034      1.481      0.142      -0.017       0.119
hardtop        -0.0710      0.037     -1.945      0.054      -0.143       0.001
hatchback      -0.0833      0.025     -3.310      0.001      -0.133      -0.033
sedan          -0.0712      0.026     -2.747      0.007      -0.123      -0.020
wagon          -0.1068      0.027     -3.907      0.000      -0.161      -0.053
four           -0.0666      0.011     -5.880      0.000      -0.089      -0.044
twelve         -0.1409      0.048     -2.910      0.004      -0.237      -0.045
High-Medium    -0.2283      0.031     -7.318      0.000      -0.290      -0.167
Low            -0.6148      0.034    -18.069      0.000      -0.682      -0.547
Medium         -0.4156      0.030    -13.927      0.000      -0.475      -0.356
Medium-Low     -0.5643      0.030    -18.760      0.000      -0.624      -0.505
```

Rejecting 'hardtop' in our next model

```
#Model 4
model_4=buildmodel(x_train_new,y_train)

x_train_new=x_train_new.drop(['hardtop'],axis=1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.972
Model:                            OLS   Adj. R-squared:                  0.969
Method:                 Least Squares   F-statistic:                     352.3
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           7.10e-81
Time:                        12:04:44   Log-Likelihood:                 233.09
No. Observations:                 123   AIC:                            -442.2
Df Residuals:                     111   BIC:                            -408.4
Df Model:                          11
Covariance Type:            nonrobust
================================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           0.7169      0.043     16.671      0.000       0.632       0.802
curbweight      0.3645      0.035     10.321      0.000       0.295       0.435
hardtop        -0.0573      0.036     -1.614      0.109      -0.128       0.013
hatchback      -0.0709      0.024     -2.971      0.004      -0.118      -0.024
sedan          -0.0545      0.023     -2.323      0.022      -0.101      -0.008
wagon          -0.0915      0.025     -3.596      0.000      -0.142      -0.041
four           -0.0636      0.011     -5.678      0.000      -0.086      -0.041
twelve         -0.1573      0.047     -3.319      0.001      -0.251      -0.063
High-Medium    -0.2267      0.031     -7.231      0.000      -0.289      -0.165
Low            -0.6152      0.034    -17.986      0.000      -0.683      -0.547
Medium         -0.4150      0.030    -13.834      0.000      -0.474      -0.356
Medium-Low     -0.5640      0.030    -18.653      0.000      -0.624      -0.504
```

New model dropping 'sedan'

```
#Model 5
model_5=buildmodel(x_train_new, y_train)

x_train_new=x_train_new.drop(['sedan'],axis=1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.972
Model:                            OLS   Adj. R-squared:                  0.969
Method:                 Least Squares   F-statistic:                     381.8
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           1.28e-81
Time:                        12:05:04   Log-Likelihood:                 231.67
No. Observations:                 123   AIC:                            -441.3
Df Residuals:                     112   BIC:                            -410.4
Df Model:                          10
Covariance Type:            nonrobust
================================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           0.6960      0.041     16.852      0.000       0.614       0.778
curbweight      0.3607      0.035     10.163      0.000       0.290       0.431
hatchback      -0.0469      0.019     -2.494      0.014      -0.084      -0.010
sedan          -0.0307      0.018     -1.670      0.098      -0.067       0.006
wagon          -0.0674      0.021     -3.249      0.002      -0.108      -0.026
four           -0.0636      0.011     -5.630      0.000      -0.086      -0.041
twelve         -0.1566      0.048     -3.280      0.001      -0.251      -0.062
High-Medium    -0.2229      0.031     -7.079      0.000      -0.285      -0.160
Low            -0.6181      0.034    -17.968      0.000      -0.686      -0.550
Medium         -0.4157      0.030    -13.760      0.000      -0.476      -0.356
Medium-Low     -0.5650      0.030    -18.557      0.000      -0.625      -0.505
```

This is our desired model

Final model

Now, we will develop a VIF (Variance Inflation Factor) function to remove all the relationship

```
#Model 6
model_6=buildmodel(x_train_new, y_train)

#This is the final model. Hence, it will be named as f_model.
f_model=model_6

#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)

checkVIF(f_model)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.971
Model:                            OLS   Adj. R-squared:                  0.968
Method:                 Least Squares   F-statistic:                     417.3
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           2.40e-82
Time:                        12:05:33   Log-Likelihood:                 230.15
No. Observations:                 123   AIC:                            -440.3
Df Residuals:                     113   BIC:                            -412.2
Df Model:                           9
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.6698      0.039     17.392      0.000       0.594       0.746
curbweight      0.3547      0.036      9.967      0.000       0.284       0.425
hatchback      -0.0186      0.008     -2.269      0.025      -0.035      -0.002
wagon          -0.0389      0.012     -3.274      0.001      -0.062      -0.015
four           -0.0622      0.011     -5.483      0.000      -0.085      -0.040
twelve         -0.1554      0.048     -3.230      0.002      -0.251      -0.060
High-Medium    -0.2171      0.032     -6.883      0.000      -0.280      -0.155
Low            -0.6207      0.035    -17.918      0.000      -0.689      -0.552
Medium         -0.4165      0.030    -13.680      0.000      -0.477      -0.356
Medium-Low     -0.5648      0.031    -18.405      0.000      -0.626      -0.504
```

	Features	VIF
0	const	120.78
7	Low	23.72
9	Medium-Low	18.61
8	Medium	6.15
1	curbweight	4.87
6	High-Medium	4.35
4	four	2.10
5	twelve	1.52
2	hatchback	1.23
3	wagon	1.23

```
model_new=f_model.drop(['Low','Medium-Low'], axis=1)
model_7=buildmodel(model_new, y_train) #checking OLS Results
checkVIF(model_7) #checking vif value

#VIF Value is under control. Now, this is our final regression model.

final_rm=model_7
checkVIF(final_rm)
```

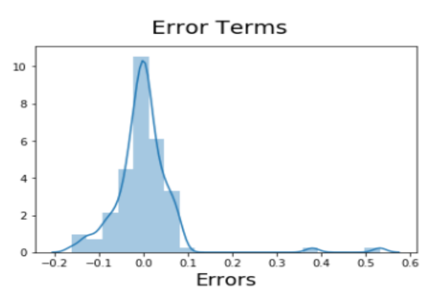
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.883
Model:                            OLS   Adj. R-squared:                  0.875
Method:                 Least Squares   F-statistic:                     123.6
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           1.88e-50
Time:                        12:06:10   Log-Likelihood:                 144.61
No. Observations:                 123   AIC:                            -273.2
Df Residuals:                     115   BIC:                            -250.7
Df Model:                           7
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.0580      0.034      1.717      0.089      -0.009      0.125
curbweight      0.5737      0.051     11.312      0.000       0.473      0.674
hatchback      -0.0329      0.016     -2.033      0.044      -0.065     -0.001
wagon          -0.0742      0.023     -3.222      0.002      -0.120     -0.029
four           -0.1072      0.022     -4.875      0.000      -0.151     -0.064
twelve          0.2473      0.082      3.001      0.003       0.084      0.410
High-Medium     0.2074      0.039      5.351      0.000       0.131      0.284
Medium          0.0763      0.029      2.674      0.009       0.020      0.133
```

	Features	VIF
0	const	23.52
1	curbweight	2.50
4	four	1.99
6	High-Medium	1.66
7	Medium	1.37
2	hatchback	1.22
3	wagon	1.17
5	twelve	1.13

Dropping-off 'hatchback'

```
#Now, to check errors, we will drop one feature, lets say hatchback.
model_check=model_7.drop(['hatchback'], axis=1)
model_check=buildmodel(model_check, y_train)
checkVIF(model_check)

#dist plot for residual analysis

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)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.878
Model:                            OLS   Adj. R-squared:                  0.872
Method:                 Least Squares   F-statistic:                     139.7
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           1.15e-50
Time:                        12:06:41   Log-Likelihood:                 142.44
No. Observations:                 123   AIC:                            -270.9
Df Residuals:                     116   BIC:                            -251.2
Df Model:                           6
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.0276      0.031      0.899      0.371      -0.033      0.088
curbweight      0.6012      0.050     12.135      0.000       0.503      0.699
wagon          -0.0646      0.023     -2.828      0.006      -0.110     -0.019
four           -0.0989      0.022     -4.516      0.000      -0.142     -0.056
twelve          0.2514      0.083      3.012      0.003       0.086      0.417
High-Medium     0.2073      0.039      5.278      0.000       0.130      0.285
Medium          0.0829      0.029      2.885      0.005       0.026      0.140
```



Score prediction for accuracy of the model

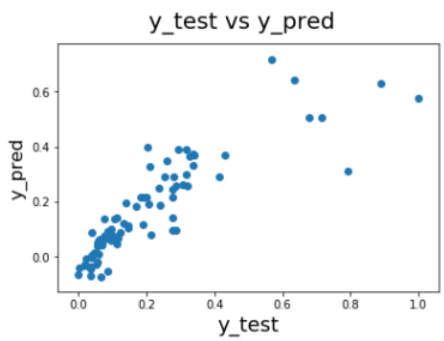
```
#5. Prediction and Evaluation

num_vars = ['wheelbase', 'curbweight', 'enginesize', 'boreratio', 'horsepower','fueleconomy','carlength','carwidth','price']
df_test[num_vars] = scaler.fit_transform(df_test[num_vars])

y_test=df_test.pop('price')
x_test=df_test
```

```
# Now let's use our model to make predictions.
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)
```

```
y_pred=lm.predict(X_test_new)
from sklearn.metrics import r2_score
accuracy=(r2_score(y_test, y_pred)*100).round(3)
print('Accuracy is :', accuracy,'%')
```

```
Accuracy is : 72.256 %
```

```
# 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
```



Overview(Project2)

Project HR Analytics is focused on employee attrition rate of a company depending upon various factors such as

- Satisfaction_level
- Work_accident
- Promotion_last_5years
- Department
- Salary

Goals

1. Correlation between various factors
2. Visualization of data
3. Regression Analysis of attrition rate

Explaination

Importing of library and dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
hr_data = pd.read_excel('C:\\Users\\pc\\Desktop\\Python\\Data-master\\python\\HR_Processed_data.xlsx')
hr_data.head()
```

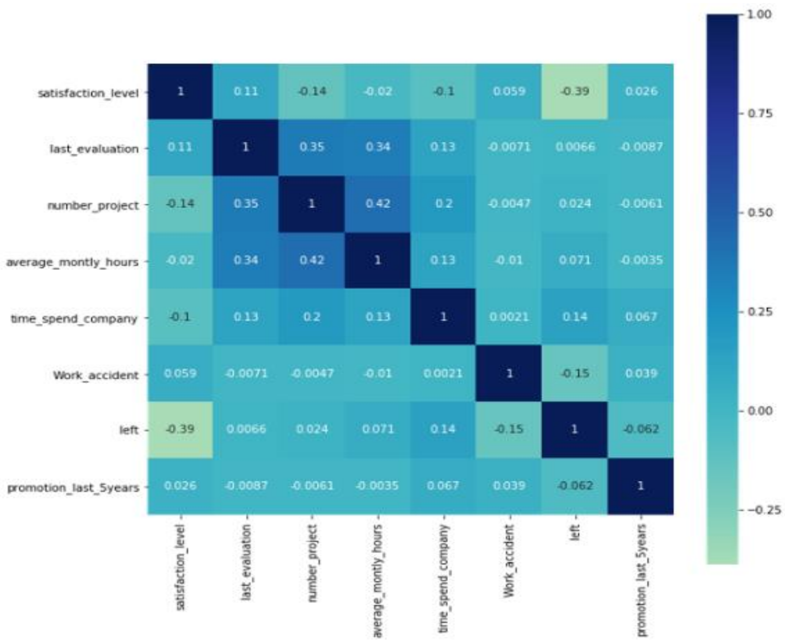
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	department	sa
0	0.38	0.53	2	157	3	0	1	0	sales	
1	0.80	0.86	5	262	6	0	1	0	sales	med
2	0.11	0.88	7	272	4	0	1	0	sales	med
3	0.72	0.87	5	223	5	0	1	0	sales	
4	0.37	0.52	2	159	3	0	1	0	sales	

Formation of correlation matrix and plotting heatmap

```
hr_data.corr()
```

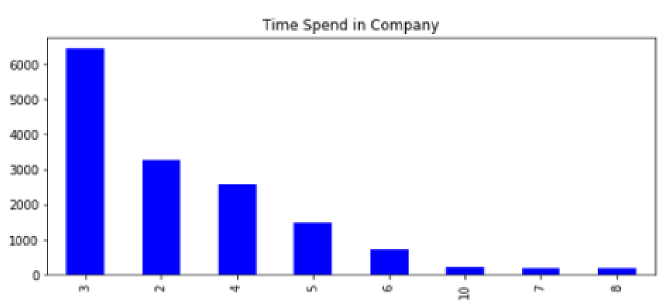
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_las
satisfaction_level	1.000000	0.105021	-0.142970	-0.020048	-0.100866	0.058697	-0.388375	
last_evaluation	0.105021	1.000000	0.349333	0.339742	0.131591	-0.007104	0.006567	-
number_project	-0.142970	0.349333	1.000000	0.417211	0.196786	-0.004741	0.023787	-
average_montly_hours	-0.020048	0.339742	0.417211	1.000000	0.127755	-0.010143	0.071287	-
time_spend_company	-0.100866	0.131591	0.196786	0.127755	1.000000	0.002120	0.144822	
Work_accident	0.058697	-0.007104	-0.004741	-0.010143	0.002120	1.000000	-0.154622	
left	-0.388375	0.006567	0.023787	0.071287	0.144822	-0.154622	1.000000	-
promotion_last_5years	0.025605	-0.008684	-0.006064	-0.003544	0.067433	0.039245	-0.061788	

```
matrix = hr_data.corr()

f,ax = plt.subplots(figsize=(10,10))
sns.heatmap(matrix, square = True, center = 0, annot = True, cmap="YlGnBu")
```

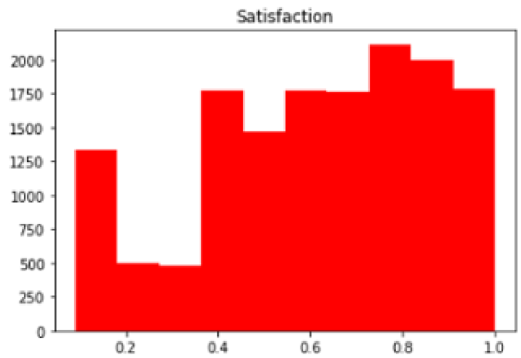


Visualization

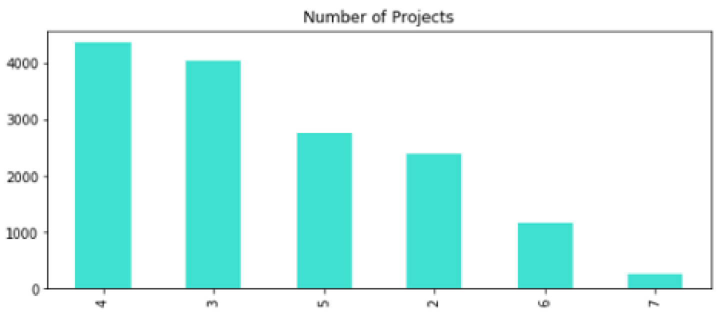
```
plt.subplot(221)
hr_data['time_spend_company'].value_counts().plot(kind='bar',figsize=(20,8),title='Time Spend in Company',color='Blue')
```



```
plt.subplot(111)
plt.title('Satisfaction')
plt.hist(hr_data['satisfaction_level'],color='Red')
```




```
plt.subplot(222)
hr_data['number_project'].value_counts().plot(kind='bar',figsize=(20,8),title='Number of Projects',color='Turquoise')
```



Changing categorical column into numerical column to perform regression analysis

```
hr_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
satisfaction_level       14999 non-null float64
last_evaluation          14999 non-null float64
number_project           14999 non-null int64
average_montly_hours     14999 non-null int64
time_spend_company       14999 non-null int64
Work_accident            14999 non-null int64
left                     14999 non-null int64
promotion_last_5years    14999 non-null int64
department               14999 non-null object
salary                   14999 non-null object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

```
hr_data.department.unique()
```

```
array(['sales', 'accounting', 'hr', 'technical', 'support', 'management',
       'IT', 'product_mng', 'marketing', 'RandD'], dtype=object)
```

```
hr_data.salary.unique()
```

```
array(['low', 'medium', 'high'], dtype=object)
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(hr_data['department'])
x = le.transform(hr_data['department'])
hr_data['department']=x
```

```
le = preprocessing.LabelEncoder()
le.fit(hr_data['salary'])
y = le.transform(hr_data['salary'])
hr_data['salary']=y
```

```
hr_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
satisfaction_level       14999 non-null float64
last_evaluation          14999 non-null float64
number_project           14999 non-null int64
average_montly_hours     14999 non-null int64
time_spend_company       14999 non-null int64
Work_accident            14999 non-null int64
left                     14999 non-null int64
promotion_last_5years    14999 non-null int64
department               14999 non-null int32
salary                   14999 non-null int32
dtypes: float64(2), int32(2), int64(6)
memory usage: 1.0 MB
```

```
hr_data.head()
```

satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	department	salary
0.38	0.53	2	157	3	0	1	0	7	1
0.80	0.86	5	262	6	0	1	0	7	2
0.11	0.88	7	272	4	0	1	0	7	2
0.72	0.87	5	223	5	0	1	0	7	1
0.37	0.52	2	159	3	0	1	0	7	1

Regression Analysis

Set target column (Y) and variables (X) to define dependent and independent factors

```
x = hr_data[['satisfaction_level','Work_accident','promotion_last_5years','department','salary']]
y = hr_data['left']
```

Splitting dataset into train and test dataset

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
model = LogisticRegression(random_state=0)
model.fit(x_train,y_train)
```

Accuracy score of train dataset

```
model.score(x_train,y_train)
```

```
0.7710642553546129
```

Accuracy score of test dataset

```
model.score(x_test,y_test)
```

```
0.7706666666666667
```

To further increase the score we apply Random Forest for classification

```
from sklearn.ensemble import RandomForestClassifier
model1 = RandomForestClassifier(random_state=0)
model1.fit(x_train,y_train)
```

```
C:\Users\pc\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators will c
hange from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Testing the accuracy of train and test

```
model1.score(x_train,y_train)
```

```
0.9127427285607134
```

```
model1.score(x_test,y_test)
```

```
0.8966666666666666
```

Formation of confusion matrix by forming a predicted Y dataset

```
y_pred_test = model1.predict(x_test)
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred_test)
cm
pd.DataFrame(cm)
```

	0	1
0	2181	118
1	192	509

Plotting ROC curve

```
#ROC Curve
import sklearn.metrics as metrics

# calculate the fpr and tpr for all thresholds of the classification

fpr,tpr,threshold = metrics.roc_curve(y_test, y_pred_test)
roc_auc = metrics.auc(fpr,tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```