In [1]:
```
#Suppress Warnings

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

In [2]:
```
import pandas as pd
import numpy as np
```

Step 1: Reading and Inspection

- Substep 1.1: Import and read

Import and read the car database. Store it in a variable called car_df

In [3]:
```
#importing car price assignment
car_df=pd.read_csv('C://Users/Vinay/Downloads//CarPrice_Assignment.csv')
```

- Substep 1.2: Inspect the dataframe

Inspect the dataframe's columns, shapes, variable types etc.

In [4]:
```
#checking the head of the dataset
car_df.head()
```

Out[4]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	engine
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	

5 rows × 26 columns

In [5]:
```
car_df.shape
```

Out[5]:
```
(205, 26)
```

In [6]:
```
car_df.info()
```

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

In [7]:
```
car_df.describe()
```

Out[7]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	engi
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.9
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.6
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.0
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.0
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.0
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.0
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.0

-Sub task 1.3 All data quality checks

All data quality issues are addressed in the right way (missing value imputation, removing duplicate data and other kinds of data redundancies, etc.). Explanations for data quality issues are clearly mentioned in comments

In [8]:
```
#  code for column-wise missing values/ null count
car_df.isnull().sum()
```

Out[8]:
```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

In [9]:
```
#  code for row-wise missing values/null count
car_df.isnull().sum(axis=1)
```

Out[9]:
```
0      0
1      0
2      0
3      0
4      0
5      0
6      0
7      0
8      0
9      0
10     0
11     0
12     0
13     0
14     0
15     0
16     0
17     0
18     0
19     0
20     0
21     0
22     0
23     0
24     0
25     0
26     0
27     0
28     0
29     0
      ..
175    0
176    0
177    0
178    0
179    0
180    0
181    0
182    0
183    0
184    0
185    0
186    0
187    0
188    0
189    0
190    0
191    0
192    0
193    0
194    0
195    0
196    0
197    0
```

```
198    0
199    0
200    0
201    0
202    0
203    0
204    0
Length: 205, dtype: int64
```

- **Didn't find any missing values column wise and row wise**
  Now checking the duplicates and data redundancy

In [10]:
```
car_df['car_ID'].head(10)
```

Out[10]:
```
0     1
1     2
2     3
3     4
4     5
5     6
6     7
7     8
8     9
9    10
Name: car_ID, dtype: int64
```

- **No duplicate data found**

  Data has no missing values and no duplicates

Step 2: Visualising the Data

Let's now spend some time doing what is arguably the most important step - understanding the data

- If there is some obvious multicolinearity going on, this is the first place to catch it
- Here's we can identify is some predictors directly have a strong association with the outcome variable.

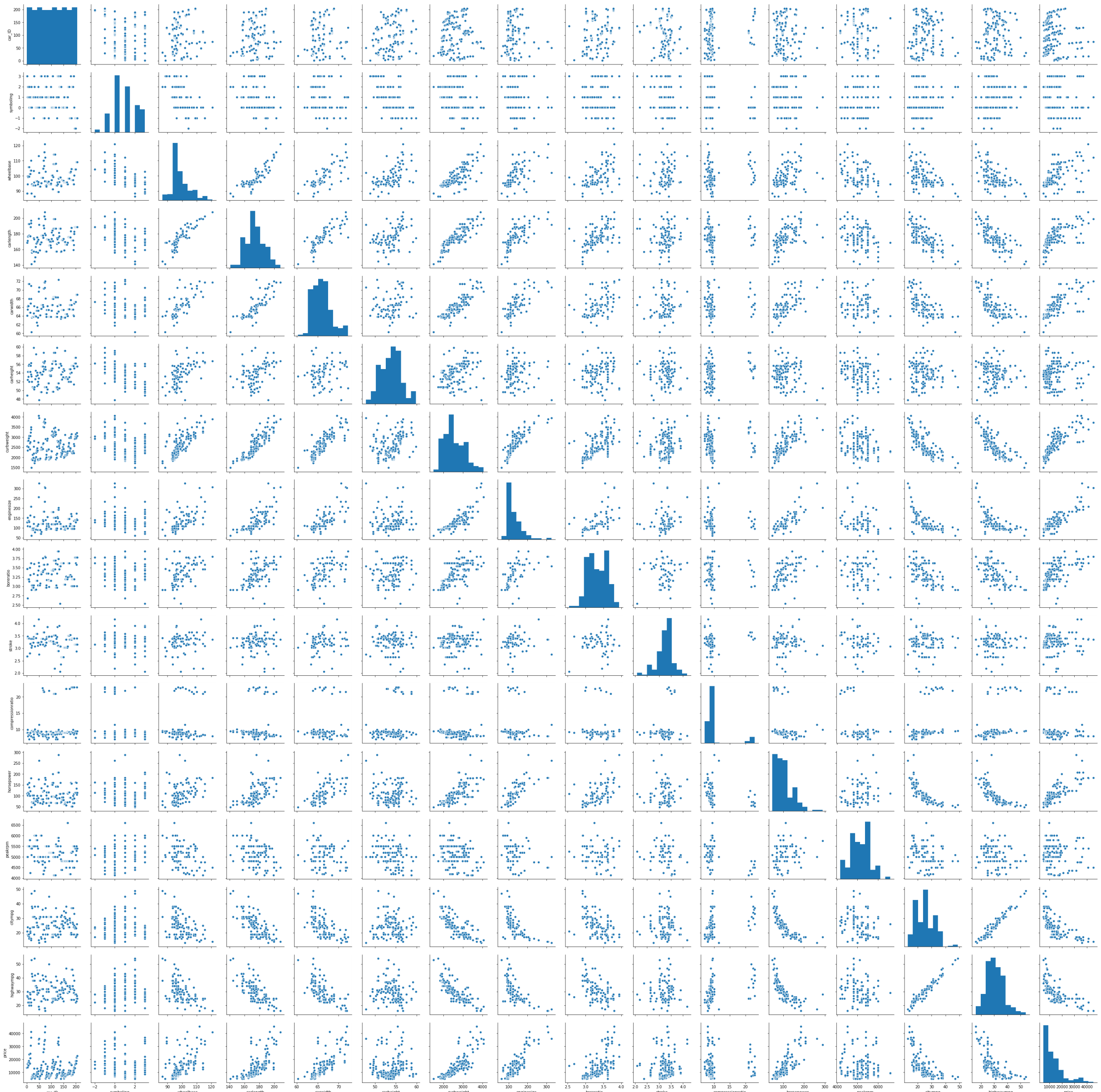
We will visualize our data using matplotlib and seaborn

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Visualising Numeric variables

Let's make a pairplot of all the numeric varables

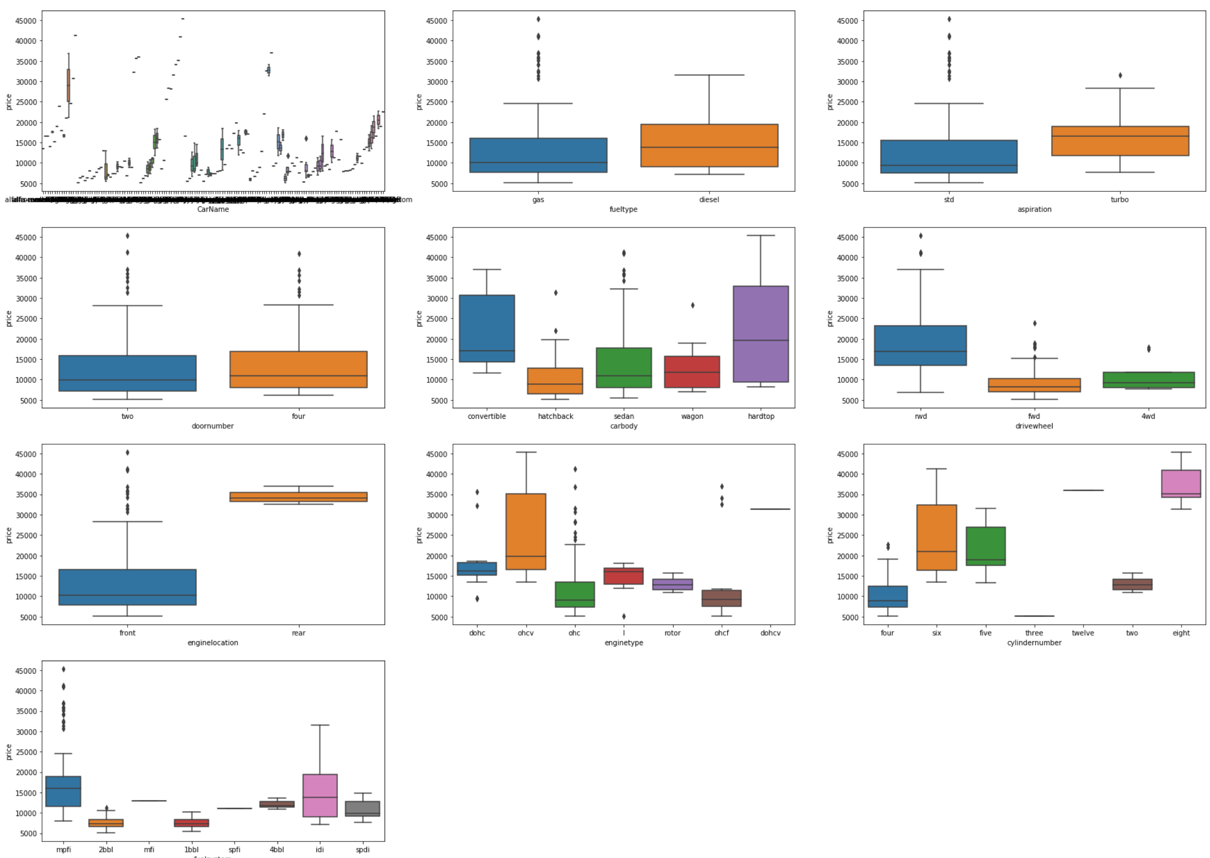
In [12]:
```
sns.pairplot(car_df)
plt.show(10)
```



Visualising categorical Variables

As we can see that there are few categorical variables as well. Let's make a boxplot for some of these variables

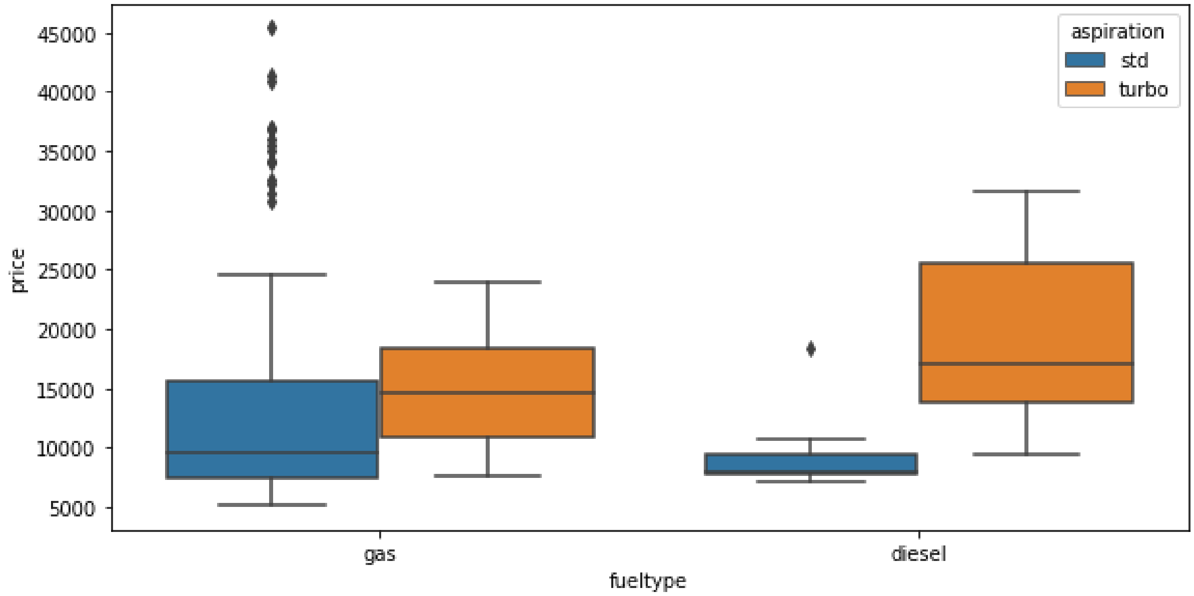
In [13]:
```
plt.figure(figsize=(30,22))
plt.subplot(4,3,1)
sns.boxplot(x='CarName', y='price', data = car_df)
plt.subplot(4,3,2)
sns.boxplot(x='fueltype', y='price', data = car_df)
plt.subplot(4,3,3)
sns.boxplot(x='aspiration', y='price', data = car_df)
plt.subplot(4,3,4)
sns.boxplot(x='doornumber', y='price', data = car_df)
plt.subplot(4,3,5)
sns.boxplot(x='carbody', y='price', data = car_df)
plt.subplot(4,3,6)
sns.boxplot(x='drivewheel', y='price', data = car_df)
plt.subplot(4,3,7)
sns.boxplot(x='enginelocation',y='price', data = car_df)
plt.subplot(4,3,8)
sns.boxplot(x='enginetype', y='price', data= car_df)
plt.subplot(4,3,9)
sns.boxplot(x='cylindernumber', y='price', data= car_df)
plt.subplot(4,3,10)
sns.boxplot(x='fuelsystem', y='price', data = car_df)

plt.show()
```



We can also visualise some of these categorical features parallely by using the hue argument.

Below is the plot for fueltype with aspiration

In [14]:
```
plt.figure(figsize =(10,5))
sns.boxplot(x='fueltype', y='price', hue='aspiration', data=car_df)
plt.show()
```



Step 3: Data Prepration

- We can see that our dataset has many columns with categorical values
- But in order ot fit a regression line, we would need numerical values and not string.
- Hence we need to convert them to 1s and 0s,

where 'gas' is 1 and 'diesel' is 0

where 'std' is 1 and 'turbo' is 0

where 'two' is 2 and 'four' is 4

where 'front' is 1 and 'rear' is 0

where 'two' is 2 , 'four' is 4 , 'five' is 5 , 'eight' is 8

In [15]:
```
#Converting categorical variables to 1 and 0 using replace function,
#converting door categorical number to door numeric number eg: two=2, four =4
#converting cyclinder categorical number to cyclinder numeric number eg: two=2, fo

car_df['fueltype']= car_df['fueltype'].replace({'gas': 1, 'diesel': 0})
car_df['aspiration']= car_df['aspiration'].replace({'std': 1, 'turbo': 0})
car_df['doornumber']= car_df['doornumber'].replace({'two': 2, 'four': 4})
car_df['enginelocation']= car_df['enginelocation'].replace({'front': 1, 'rear': 0}
car_df['cylindernumber']= car_df['cylindernumber'].replace({'two': 2, 'three': 3,
```

In [16]:
```
#checking the car_df  dataframe now
car_df.shape
```

Out[16]: (205, 26)

Considering variable name CarName which is comprised of two parts

The first word is the name of 'car company' and the second is the 'car model'. For example, chevrolet impala has 'chevrolet' as the car company name and 'impala' as the car model name. We will consider only company name as the independent variable for model building

In [17]:
```
new = car_df["CarName"].str.split(" ", n = 1, expand = True)

# making seperate first name column from new data frame
car_df["CompanyName"]= new[0]

# Dropping old Name columns
car_df.drop(columns =["CarName"], inplace = True)
```

In [18]:
```
car_df.head()
```

Out[18]:

	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wh
0	1	3	1	1	2	convertible	rwd	1	
1	2	3	1	1	2	convertible	rwd	1	
2	3	1	1	1	2	hatchback	rwd	1	
3	4	2	1	1	4	sedan	fwd	1	
4	5	2	1	1	4	sedan	4wd	1	

5 rows × 26 columns

Dummy Variables

The variable carname has lots of levels. We need to convert these levels into integer as well.

For this, we will use something called dummy variables.

In [19]:
```
#Get dummy variables for the feature 'CompanyName' and store it into a new variabl
#Get dummy variables for the feature 'carbody' and store it into a new variable
#Get dummy variables for the feature 'drivewheel' and store it into a new variable
#Get dummy variables for the feature 'enginetype' and store it into a new variable
#Get dummy variables for the feature 'fuelsystem' and store it into a new variable

status1 = pd.get_dummies(car_df['CompanyName'])
status2 = pd.get_dummies(car_df['carbody'])
status3 = pd.get_dummies(car_df['drivewheel'])
status4 = pd.get_dummies(car_df['enginetype'])
status5 = pd.get_dummies(car_df['fuelsystem'])


#Check what the dataset 'status' looks like
status1.head()
```

Out[19]:

	Nissan	alfa-romero	audi	bmw	buick	chevrolet	dodge	honda	isuzu	jaguar	...	porsche	renau
0	0	1	0	0	0	0	0	0	0	0	...	0	
1	0	1	0	0	0	0	0	0	0	0	...	0	
2	0	1	0	0	0	0	0	0	0	0	...	0	
3	0	0	1	0	0	0	0	0	0	0	...	0	
4	0	0	1	0	0	0	0	0	0	0	...	0	

5 rows × 28 columns

In [20]:
```
status2.head()
```

Out[20]:

	convertible	hardtop	hatchback	sedan	wagon
0	1	0	0	0	0
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	1	0

In [21]:
```
status3.head()
```

Out[21]:

	4wd	fwd	rwd
0	0	0	1
1	0	0	1
2	0	0	1
3	0	1	0
4	1	0	0

In [22]:
```
status4.head()
```

Out[22]:

	dohc	dohcv	l	ohc	ohcf	ohcv	rotor
0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0
4	0	0	0	1	0	0	0

In [23]:
```
status5.head()
```

Out[23]:

	1bbl	2bbl	4bbl	idi	mfi	mpfi	spdi	spfi
0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	1	0	0
2	0	0	0	0	0	1	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	1	0	0

Now we can reduce 1 column. We can drop the alfa-romero column, as the type of alfa-romero can be identified with just the other column.

Now we can reduce 1 column. We can drop the convertible column, as the type of convertible can be identified with just the other column.

Now we can reduce 1 column. We can drop the 4wd column, as the type of 4wd can be identified with just the other column.

Now we can reduce 1 column. We can drop the dohc column, as the type of dohc can be identified with just the other column.

Now we can reduce 1 column. We can drop the 1bbl column, as the type of 1bbl can be identified with just the other column.

In [24]:
```
#Let's drop the second column from status1 df using 'status1.drop'
#Let's drop the first column from status2 df using 'drop_first=True'
#Let's drop the first column from status3 df using 'drop_first=True'
#Let's drop the first column from status4 df using 'drop_first=True'
#Let's drop the first column from status5 df using 'drop_first=True'

status1 = pd.get_dummies(status1.drop('alfa-romero', 1))
status2 = pd.get_dummies(car_df['carbody'], drop_first = True)
status3 = pd.get_dummies(car_df['drivewheel'], drop_first=True)
status4 = pd.get_dummies(car_df['enginetype'], drop_first=True)
status5 = pd.get_dummies(car_df['fuelsystem'], drop_first=True)


#Check what the dataset 'status' looks like
status1.head()
```

Out[24]:

	Nissan	audi	bmw	buick	chevrolet	dodge	honda	isuzu	jaguar	maxda	...	porsche	renaul
0	0	0	0	0	0	0	0	0	0	0	...	0	(
1	0	0	0	0	0	0	0	0	0	0	...	0	(
2	0	0	0	0	0	0	0	0	0	0	...	0	(
3	0	1	0	0	0	0	0	0	0	0	...	0	(
4	0	1	0	0	0	0	0	0	0	0	...	0	(

5 rows × 27 columns

In [25]:
```
#Add the results to the original car_df dataframe

car_df=pd.concat([car_df,status1,status2,status3,status4,status5], axis=1)
```

In [26]:
```
#Now lets see the head of our dataframe
car_df.head()
```

Out[26]:

	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	whe
0	1	3	1	1	2	convertible	rwd	1	
1	2	3	1	1	2	convertible	rwd	1	
2	3	1	1	1	2	hatchback	rwd	1	
3	4	2	1	1	4	sedan	fwd	1	
4	5	2	1	1	4	sedan	4wd	1	

5 rows × 72 columns

In [27]:
```
#Drop 'CompanyName' as we have created the dummies for it
#Drop 'carbody' as we have created the dummies for it
#Drop 'drivewheel' as we have created the dummies for it
#Drop 'enginetype' as we have created the dummies for it
#Drop 'fuelsystem' as we have created the dummies for it


car_df.drop(['CompanyName'], axis =1 , inplace =True)
car_df.drop(['carbody'], axis =1 , inplace =True)
car_df.drop(['drivewheel'], axis =1 , inplace =True)
car_df.drop(['enginetype'], axis =1 , inplace =True)
car_df.drop(['fuelsystem'], axis =1 , inplace =True)
```

In [28]:
```
car_df.head()
```

Out[28]:

	car_ID	symboling	fueltype	aspiration	doornumber	enginelocation	wheelbase	carlength	carw
0	1	3	1	1	2	1	88.6	168.8	
1	2	3	1	1	2	1	88.6	168.8	
2	3	1	1	1	2	1	94.5	171.2	
3	4	2	1	1	4	1	99.8	176.6	
4	5	2	1	1	4	1	99.4	176.6	

5 rows × 67 columns

Step 4 : Splitting the Data into Training and Testing Sets

As we know, the first basic step for regression is performing a train-test split.

In [29]:
```
from sklearn.model_selection import train_test_split

# We specify this so that the train and test data set always have the same rows, r

np.random.seed(0)
df_train, df_test = train_test_split(car_df, train_size = 0.7, test_size = 0.3, ra
```

Rescaling the Features

As we saw in the demonstration for Simple Linear Regression, scaling doesn't impact our model. Here we can see that except for price, all the columns have small integer values. So it is extremely important to rescale the variables so that they have a comparable scale. If we don't have comparable scales, then some of the coefficients as obtained by fitting the regression model might be very large or very small as compared to the other coefficients. This might become very annoying at the time of model evaluation. So it is necessary to use standardization or normalization so that the units of the coefficients obtained are all on the same scale. As we know, there are two common ways of rescaling:

1. Min-Max scaling
2. Standardisation (mean-0, sigma-1)

Lets use MinMax scaling.

In [30]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [31]:
```
# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['car_ID', 'symboling', 'wheelbase', 'carlength', 'carwidth','carheight
            'boreratio','stroke','compressionratio','horsepower','peakrpm','citymp
            'doornumber','cylindernumber']

df_train[num_vars] = scaler.fit_transform(df_train[num_vars])

df_train.head()
```

Out[31]:

	car_ID	symboling	fueltype	aspiration	doornumber	enginelocation	wheelbase	carlength
122	0.598039	0.6	1	1	1.0	1	0.244828	0.426016
125	0.612745	1.0	1	1	0.0	1	0.272414	0.452033
166	0.813725	0.6	1	1	0.0	1	0.272414	0.448780
1	0.004902	1.0	1	1	0.0	1	0.068966	0.450407
199	0.975490	0.2	1	0	1.0	1	0.610345	0.775610

5 rows × 67 columns

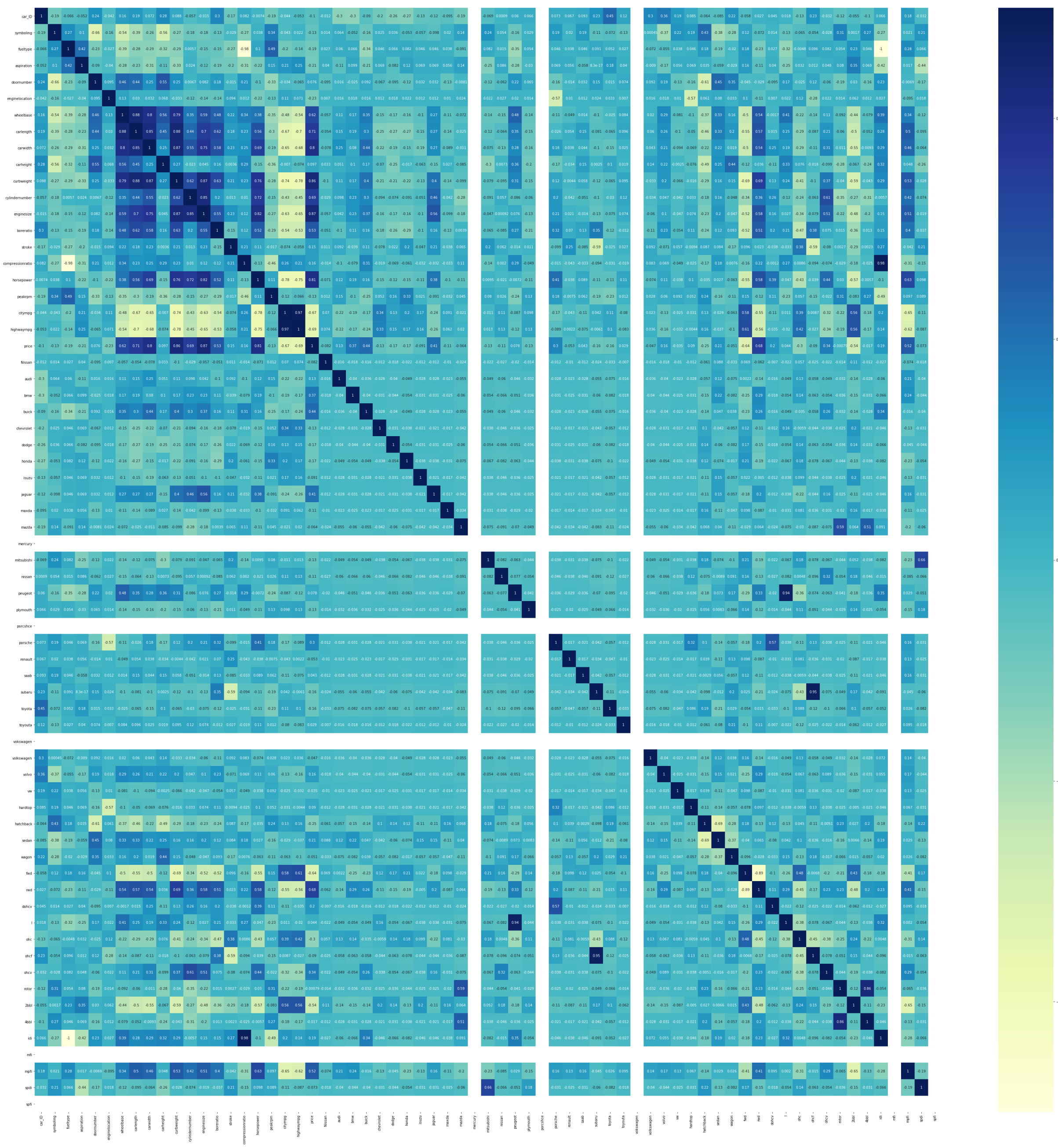
In [32]:
```
df_train.describe()
```

Out[32]:

	car_ID	symboling	fueltype	aspiration	doornumber	enginelocation	wheelbase
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000
mean	0.478061	0.559441	0.909091	0.818182	0.559441	0.993007	0.411141
std	0.289106	0.239200	0.288490	0.387050	0.498199	0.083624	0.205581
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.232843	0.400000	1.000000	1.000000	0.000000	1.000000	0.272414
50%	0.470588	0.600000	1.000000	1.000000	1.000000	1.000000	0.341379
75%	0.718137	0.600000	1.000000	1.000000	1.000000	1.000000	0.503448
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 67 columns

In [33]:
```
# Let's check the correlation coefficients to see which variables are highly corre

plt.figure(figsize = (60, 60))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



As it is large amount of data and lots of variable are involved, so checking R square by adding each variable

one after another would be time consuming, hence we should not follow forward method.

Using RFE here will be the best approach.

STEP 5 : Model Building

Dividing into X and Y sets for the model building

In [34]:
```
y_train = df_train.pop('price')
x_train = df_train
```

Building our model

We will be using the Linear Regression function from Scikit Learn for its compatibility with RFE (which is a utility from sklearn)

RFE

Recursive feature elimination

In [35]:
```
#Importing RFE and Linear Regression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

In [36]:
```
# Running RFE with the output number of the variable equal to 20
lm = LinearRegression()
lm.fit(x_train,y_train)

rfe = RFE(lm,20)                    #running RFE
rfe = rfe.fit(x_train, y_train)
```

In [37]:
```
list(zip(x_train.columns, rfe.support_,rfe.ranking_))
```

Out[37]:
```
[('car_ID', False, 6),
 ('symboling', False, 42),
 ('fueltype', True, 1),
 ('aspiration', False, 19),
 ('doornumber', False, 40),
 ('enginelocation', True, 1),
 ('wheelbase', False, 22),
 ('carlength', False, 21),
 ('carwidth', True, 1),
 ('carheight', False, 20),
 ('curbweight', True, 1),
 ('cylindernumber', True, 1),
 ('enginesize', True, 1),
 ('boreratio', True, 1),
 ('stroke', True, 1),
 ('compressionratio', True, 1),
 ('horsepower', False, 18),
 ('peakrpm', False, 13),
 ('citympg', False, 32),
 ('highwaympg', False, 15),
 ('Nissan', False, 9),
 ('audi', False, 33),
 ('bmw', True, 1),
 ('buick', False, 30),
 ('chevrolet', False, 16),
 ('dodge', False, 12),
 ('honda', False, 11),
 ('isuzu', False, 14),
 ('jaguar', False, 28),
 ('maxda', True, 1),
 ('mazda', True, 1),
 ('mercury', False, 44),
 ('mitsubishi', True, 1),
 ('nissan', False, 4),
 ('peugeot', True, 1),
 ('plymouth', False, 5),
 ('porcshce', False, 43),
 ('porsche', False, 29),
 ('renault', True, 1),
 ('saab', False, 10),
 ('subaru', True, 1),
 ('toyota', True, 1),
 ('toyouta', False, 3),
 ('vokswagen', False, 45),
 ('volkswagen', False, 2),
 ('volvo', False, 7),
 ('vw', False, 8),
 ('hardtop', False, 24),
 ('hatchback', False, 23),
 ('sedan', False, 25),
 ('wagon', False, 26),
 ('fwd', False, 39),
 ('rwd', False, 36),
 ('dohcv', True, 1),
 ('l', False, 17),
```

```
('ohc', False, 27),
('ohcf', False, 35),
('ohcv', False, 38),
('rotor', True, 1),
('2bbl', False, 34),
('4bbl', False, 31),
('idi', True, 1),
('mfi', False, 46),
('mpfi', False, 37),
('spdi', False, 41),
('spfi', False, 47)]
```

In [38]:
```
col = x_train.columns[rfe.support_]
col
```

Out[38]:
```
Index(['fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'bmw', 'maxda', 'mazda', 'mitsubishi', 'peugeot',
       'renault', 'subaru', 'toyota', 'dohcv', 'rotor', 'idi'],
      dtype='object')
```

In [39]:
```
x_train.columns[~rfe.support_]
```

Out[39]:
```
Index(['car_ID', 'symboling', 'aspiration', 'doornumber', 'wheelbase',
       'carlength', 'carheight', 'horsepower', 'peakrpm', 'citympg',
       'highwaympg', 'Nissan', 'audi', 'buick', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mercury', 'nissan', 'plymouth', 'porcshce',
       'porsche', 'saab', 'toyouta', 'vokswagen', 'volkswagen', 'volvo', 'vw',
       'hardtop', 'hatchback', 'sedan', 'wagon', 'fwd', 'rwd', 'l', 'ohc',
       'ohcf', 'ohcv', '2bbl', '4bbl', 'mfi', 'mpfi', 'spdi', 'spfi'],
      dtype='object')
```

Building model using statsmodel, for the detailed statistics

In [40]:
```
# Creating x_test dataframe with RFE selected variables
x_train_rfe  = x_train[col]
```

In [41]:
```
#Adding a constant variable
import statsmodels.api as sm
x_train_rfe = sm.add_constant(x_train_rfe)
```

In [42]:
```
lm1 = sm.OLS(y_train,x_train_rfe).fit()     # Running  the linear model
```

In [43]:
```
#Let's see the summary of our linear model
print(lm1.summary())
```

```
Model:                            OLS   Adj. R-squared:                  0.9
32
Method:                 Least Squares   F-statistic:                     10
3.8
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           2.86e-
66
Time:                        20:09:02   Log-Likelihood:                 219.
67
No. Observations:                 143   AIC:                            -39
9.3
Df Residuals:                     123   BIC:                            -34
0.1
Df Model:                          19
Covariance Type:            nonrobust
============================================================================
========
                       coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------------
--------
```

Looking at the p-values, it looks like some of the variables aren't really significant (in the presence of other variables).

Maybe we could drop some?

We could simply drop the variable with the highest, non-significant p value. A better way would be to supplement this with the VIF information.

Checking VIF

Variance Inflation Factor or VIF, gives a basic quantitative idea about how much the feature variables are correlated with each other. It is an extremely important parameter to test our linear model. The formula for calculating `VIF` is:

$$VIF_i = \frac{1}{1-{R_i}^2}$$

In [44]:
```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [45]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_rfe.columns
vif['VIF'] = [variance_inflation_factor(x_train_rfe.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[45]:

	Features	VIF
20	idi	inf
1	fueltype	inf
9	compressionratio	64.570000
6	enginesize	48.220000
5	cylindernumber	27.730000
4	curbweight	13.470000
7	boreratio	10.480000
3	carwidth	5.630000
8	stroke	5.340000
16	subaru	2.560000
19	rotor	2.240000
18	dohcv	1.810000

We generally want a VIF that is less than 5. So there are clearly some variables we need to drop.

Dropping the variable and updating the model

As we can see from the summary and the VIF dataframe, some variables are still insignificant. One of these variables is, `renault` as it has a very high p-value of `0.319` . Let's go ahead and drop this variables

renault is insignificant in presence of other variables, because it has high p value of 0.319; can be dropped

In [46]:
```
x_train_rfe.columns
```

Out[46]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'bmw', 'maxda', 'mazda', 'mitsubishi', 'peugeot',
       'renault', 'subaru', 'toyota', 'dohcv', 'rotor', 'idi'],
      dtype='object')
```

In [47]:
```
x_train_new = x_train_rfe.drop(['renault'],axis =1)
```

Rebuilding model without renault

In [48]:
```
#Adding a constant variable
import statsmodels.api as sm
x_train_lm = sm.add_constant(x_train_new)
```

In [49]:
```
lm2 = sm.OLS(y_train,x_train_lm).fit()   #Running the linear model   #fitting a re
```

In [50]:
```
#Let's see the summary of our linear model
print(lm2.summary())
```

```
                            OLS Regression Results
=============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
41
Model:                            OLS   Adj. R-squared:                  0.9
32
Method:                 Least Squares   F-statistic:                     10
9.5
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           4.40e-
67
Time:                        20:09:03   Log-Likelihood:                 219.
09
No. Observations:                 143   AIC:                             -40
0.2
Df Residuals:                     124   BIC:                             -34
3.9
Df Model:                          18
Covariance Type:            nonrobust
```

In [51]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

3	carwidth	5.630000
8	stroke	4.600000
15	subaru	2.560000
18	rotor	2.240000
17	dohcv	1.770000
12	mazda	1.750000
14	peugeot	1.720000
13	mitsubishi	1.260000
10	bmw	1.190000
2	enginelocation	1.180000
16	toyota	1.150000
11	maxda	1.060000
0	const	0.000000

maxda is insignificant in presence of other variables, because it has high p value of 0.163; can be dropped

In [52]:
```
x_train_new.columns
```

Out[52]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'bmw', 'maxda', 'mazda', 'mitsubishi', 'peugeot',
       'subaru', 'toyota', 'dohcv', 'rotor', 'idi'],
      dtype='object')
```

In [53]:
```
x_train_new = x_train_new.drop(['maxda'],axis =1)
```

Model 3: Rebuidling 3rd model without maxda

In [54]:
```
x_train_lm = sm.add_constant(x_train_new)
lm3 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm3.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
40
Model:                            OLS   Adj. R-squared:                  0.9
32
Method:                 Least Squares   F-statistic:                      11
4.9
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           1.06e-
67
Time:                        20:09:03   Log-Likelihood:                 217.
96
No. Observations:                 143   AIC:                             -39
9.9
Df Residuals:                     125   BIC:                             -34
6.6
Df Model:                          17
Covariance Type:            nonrobust
```

In [55]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

7	boreratio	9.690000
3	carwidth	5.590000
8	stroke	4.550000
14	subaru	2.540000
17	rotor	2.230000
16	dohcv	1.770000
11	mazda	1.740000
13	peugeot	1.720000
12	mitsubishi	1.250000
10	bmw	1.190000
2	enginelocation	1.180000
15	toyota	1.150000

mazda is insignificant in presence of other variables, because it has high p value of 0.062; can be dropped

In [56]:
```
x_train_new.columns
```

Out[56]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'bmw', 'mazda', 'mitsubishi', 'peugeot', 'subaru',
       'toyota', 'dohcv', 'rotor', 'idi'],
      dtype='object')
```

In [57]:
```
x_train_new = x_train_new.drop(['mazda'],axis =1)
```

Model 4: Rebuidling model without mazda

In [58]:
```
x_train_lm = sm.add_constant(x_train_new)
lm4 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm4.summary())
```

```
                            OLS Regression Results
=============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
38
Model:                            OLS   Adj. R-squared:                  0.9
30
Method:                 Least Squares   F-statistic:                     11
9.5
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           5.45e-
68
Time:                        20:09:04   Log-Likelihood:                215.
95
No. Observations:                 143   AIC:                            -39
7.9
Df Residuals:                     126   BIC:                            -34
7.5
Df Model:                          16
Covariance Type:            nonrobust
```

In [59]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[59]:

	Features	VIF
17	idi	inf
1	fueltype	inf
9	compressionratio	63.310000
6	enginesize	45.740000
5	cylindernumber	26.500000
4	curbweight	12.640000
7	boreratio	9.640000
3	carwidth	5.510000
8	stroke	4.550000
13	subaru	2.470000
15	dohcv	1.760000
12	peugeot	1.690000
16	rotor	1.640000
11	mitsubishi	1.240000
2	enginelocation	1.180000
10	bmw	1.180000
14	toyota	1.140000
0	const	0.000000

rotor is insignificant in presence of other variables, because it has high p value of 0.036 can be dropped

In [60]:
```
x_train_new.columns
```

Out[60]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'bmw', 'mitsubishi', 'peugeot', 'subaru', 'toyota',
       'dohcv', 'rotor', 'idi'],
      dtype='object')
```

In [61]:
```
x_train_new = x_train_new.drop(['rotor'],axis =1)
```

Model 5: Rebuidling model without rotor

In [62]:
```
x_train_lm = sm.add_constant(x_train_new)
lm5 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm5.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
36
Model:                            OLS   Adj. R-squared:                  0.9
28
Method:                 Least Squares   F-statistic:                     12
3.7
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           4.34e-
68
Time:                        20:09:05   Log-Likelihood:                 213.
43
No. Observations:                 143   AIC:                            -39
4.9
Df Residuals:                     127   BIC:                            -34
7.5
Df Model:                          15
Covariance Type:            nonrobust
```

In [63]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[63]:

	Features	VIF
16	idi	inf
1	fueltype	inf
9	compressionratio	56.730000
6	enginesize	44.900000
5	cylindernumber	22.980000
4	curbweight	11.960000
7	boreratio	9.120000
3	carwidth	5.440000
8	stroke	4.450000
13	subaru	2.470000
15	dohcv	1.720000
12	peugeot	1.650000
11	mitsubishi	1.230000
2	enginelocation	1.170000
10	bmw	1.170000
14	toyota	1.120000
0	const	0.000000

compressionratio is insignificant in presence of other variables, because it has high p value of 0.117 can be dropped

In [64]:
```
x_train_new.columns
```

Out[64]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'bmw', 'mitsubishi', 'peugeot', 'subaru', 'toyota',
       'dohcv', 'idi'],
      dtype='object')
```

In [65]:
```
x_train_new = x_train_new.drop(['compressionratio'],axis =1)
```

Model 6: Rebuidling model without compressionratio

In [66]:
```
x_train_lm = sm.add_constant(x_train_new)
lm6 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm6.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
35
Model:                            OLS   Adj. R-squared:                  0.9
28
Method:                 Least Squares   F-statistic:                      13
0.9
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           1.26e-
68
Time:                        20:09:05   Log-Likelihood:                 212.
05
No. Observations:                 143   AIC:                             -39
4.1
Df Residuals:                     128   BIC:                             -34
9.7
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
```

In [67]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[67]:

	Features	VIF
1	fueltype	inf
15	idi	inf
6	enginesize	44.840000
5	cylindernumber	22.890000
4	curbweight	11.000000
7	boreratio	9.090000
3	carwidth	5.420000
8	stroke	4.150000
12	subaru	2.260000
14	dohcv	1.710000
11	peugeot	1.510000

subaru is insignificant in presence of other variables, because it has high p value of 0.025 can be dropped

In [68]:
```
x_train_new.columns
```

Out[68]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke', 'bmw',
       'mitsubishi', 'peugeot', 'subaru', 'toyota', 'dohcv', 'idi'],
      dtype='object')
```

In [69]:
```
x_train_new = x_train_new.drop(['subaru'],axis =1)
```

Model 7: Rebuidling model without subaru

In [70]:
```
x_train_lm = sm.add_constant(x_train_new)
lm7 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm7.summary())
```

```
                            OLS Regression Results
=============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
32
Model:                            OLS   Adj. R-squared:                  0.9
25
Method:                 Least Squares   F-statistic:                     13
6.1
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           1.30e-
68
Time:                        20:09:06   Log-Likelihood:                 209.
23
No. Observations:                 143   AIC:                            -39
0.5
Df Residuals:                     129   BIC:                            -34
9.0
Df Model:                          13
Covariance Type:            nonrobust
```

In [71]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[71]:

	Features	VIF
1	fueltype	inf
14	idi	inf
6	enginesize	43.380000
5	cylindernumber	22.070000
4	curbweight	10.990000
7	boreratio	7.560000
3	carwidth	5.400000
8	stroke	3.950000
13	dohcv	1.610000
11	peugeot	1.440000
2	enginelocation	1.130000
9	bmw	1.120000
10	mitsubishi	1.100000
12	toyota	1.090000
0	const	0.000000

mitsubishi is insignificant in presence of other variables, because it has high p value of 0.013 can be dropped

In [72]:
```
x_train_new.columns
```

Out[72]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke', 'bmw',
       'mitsubishi', 'peugeot', 'toyota', 'dohcv', 'idi'],
      dtype='object')
```

In [73]:
```
x_train_new = x_train_new.drop(['mitsubishi'],axis =1)
```

Model 8: Rebuidling model without mitsubishi

In [74]:
```
x_train_lm = sm.add_constant(x_train_new)
lm8 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm8.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
29
Model:                            OLS   Adj. R-squared:                  0.9
22
Method:                 Least Squares   F-statistic:                      14
1.1
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           2.34e-
68
Time:                        20:09:06   Log-Likelihood:                 205.
79
No. Observations:                 143   AIC:                             -38
5.6
Df Residuals:                     130   BIC:                             -34
7.1
Df Model:                          12
Covariance Type:            nonrobust
```

In [75]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[75]:

	Features	VIF
1	fueltype	inf
13	idi	inf
6	enginesize	43.380000
5	cylindernumber	22.070000
4	curbweight	10.990000
7	boreratio	7.550000
3	carwidth	5.380000
8	stroke	3.870000
12	dohcv	1.610000
10	peugeot	1.440000
2	enginelocation	1.130000
9	bmw	1.120000
11	toyota	1.070000
0	const	0.000000

The model looks fine with p-value , but we can see that VIF of some variables is very very high, lets drop those variables "fueltype" has high VIF i.e. inf so lets drop it. "idi" also has VIF inf, on checking dropping preference, droping any of these variable gives the same results, so lets drop "fueltype" first. VIF should be less than 5

In [76]:
```
x_train_new.columns
```

Out[76]:
```
Index(['const', 'fueltype', 'enginelocation', 'carwidth', 'curbweight',
       'cylindernumber', 'enginesize', 'boreratio', 'stroke', 'bmw', 'peugeot',
       'toyota', 'dohcv', 'idi'],
      dtype='object')
```

In [77]:
```
x_train_new = x_train_new.drop(['fueltype'],axis =1)
```

Model 9: Rebuidling model without fueltype

In [78]:
```
x_train_lm = sm.add_constant(x_train_new)
lm9 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm9.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
29
Model:                            OLS   Adj. R-squared:                  0.9
22
Method:                 Least Squares   F-statistic:                    14
1.1
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           2.34e-
68
Time:                        20:09:07   Log-Likelihood:                205.
79
No. Observations:                 143   AIC:                            -38
5.6
Df Residuals:                     130   BIC:                            -34
7.1
Df Model:                          12
Covariance Type:            nonrobust
```

In [79]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[79]:

	Features	VIF
0	const	300.95
5	enginesize	43.38
4	cylindernumber	22.07
3	curbweight	10.99
6	boreratio	7.55
2	carwidth	5.38
7	stroke	3.87
11	dohcv	1.61
9	peugeot	1.44
12	idi	1.31
1	enginelocation	1.13

idi is insignificant in presence of other variables, because it has high p value of 0.068 can be dropped

In [80]:
```
x_train_new.columns
```

Out[80]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'enginesize', 'boreratio', 'stroke', 'bmw', 'peugeot', 'toyota',
       'dohcv', 'idi'],
      dtype='object')
```

In [81]:
```
x_train_new = x_train_new.drop(['idi'],axis =1)
```

Model 10: Rebuidling model without idi

In [82]:
```
x_train_lm = sm.add_constant(x_train_new)
lm10 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm10.summary())
```

```
                            OLS Regression Results
============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
27
Model:                            OLS   Adj. R-squared:                  0.9
21
Method:                 Least Squares   F-statistic:                     15
0.9
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           9.84e-
69
Time:                        20:09:08   Log-Likelihood:                 203.
94
No. Observations:                 143   AIC:                            -38
3.9
Df Residuals:                     131   BIC:                            -34
8.3
Df Model:                          11
Covariance Type:            nonrobust
```

In [83]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[83]:

	Features	VIF
0	const	298.17
5	enginesize	42.80
4	cylindernumber	21.95
3	curbweight	10.82
6	boreratio	7.51
2	carwidth	5.37
7	stroke	3.71
11	dohcv	1.60
9	peugeot	1.36
1	enginelocation	1.13
8	bmw	1.11
10	toyota	1.07

enginesize has high VIF value of 42.8 can be dropped

In [84]:
```
x_train_new.columns
```

Out[84]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'enginesize', 'boreratio', 'stroke', 'bmw', 'peugeot', 'toyota',
       'dohcv'],
      dtype='object')
```

In [85]:
```
x_train_new = x_train_new.drop(['enginesize'],axis =1)
```

Model 11: Rebuidling model without enginesize

In [86]:
```
x_train_lm = sm.add_constant(x_train_new)
lm11 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm11.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
03
Model:                            OLS   Adj. R-squared:                  0.8
96
Method:                 Least Squares   F-statistic:                     12
3.3
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           6.88e-
62
Time:                        20:09:08   Log-Likelihood:                 183.
96
No. Observations:                 143   AIC:                            -34
5.9
Df Residuals:                     132   BIC:                            -31
3.3
Df Model:                          10
Covariance Type:            nonrobust
```

In [87]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[87]:

	Features	VIF
0	const	208.48
3	curbweight	7.51
2	carwidth	5.10
4	cylindernumber	2.71
5	boreratio	2.63
6	stroke	1.43
8	peugeot	1.34
10	dohcv	1.25
7	bmw	1.11
1	enginelocation	1.10
9	toyota	1.07

stroke is insignificant in presence of other variables, because it has high p value of 0.502 can be dropped

In [88]:
```
x_train_new.columns
```

Out[88]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'boreratio', 'stroke', 'bmw', 'peugeot', 'toyota', 'dohcv'],
      dtype='object')
```

In [89]:
```
x_train_new = x_train_new.drop(['stroke'],axis =1)
```

Model 12: Rebuidling model without stroke

In [90]:
```
x_train_lm = sm.add_constant(x_train_new)
lm12 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm12.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
03
Model:                            OLS   Adj. R-squared:                  0.8
96
Method:                 Least Squares   F-statistic:                     13
7.5
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           7.12e-
63
Time:                        20:09:09   Log-Likelihood:                 183.
71
No. Observations:                 143   AIC:                            -34
7.4
Df Residuals:                     133   BIC:                            -31
7.8
Df Model:                           9
Covariance Type:            nonrobust
```

In [91]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[91]:

	Features	VIF
0	const	182.57
3	curbweight	6.85
2	carwidth	4.97
4	cylindernumber	2.38
5	boreratio	2.07
7	peugeot	1.30
9	dohcv	1.24
6	bmw	1.11
1	enginelocation	1.10
8	toyota	1.07

boreratio is insignificant in presence of other variables, because it has high p value of 0.297 can be dropped

In [92]:
```
x_train_new.columns
```

Out[92]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'boreratio', 'bmw', 'peugeot', 'toyota', 'dohcv'],
      dtype='object')
```

In [93]:
```
x_train_new = x_train_new.drop(['boreratio'],axis =1)
```

Model 13: Rebuidling model without boreratio

In [94]:
```
x_train_lm = sm.add_constant(x_train_new)
lm13 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm13.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
02
Model:                            OLS   Adj. R-squared:                  0.8
96
Method:                 Least Squares   F-statistic:                     15
4.4
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           9.53e-
64
Time:                        20:09:10   Log-Likelihood:                 183.
12
No. Observations:                 143   AIC:                            -34
8.2
Df Residuals:                     134   BIC:                            -32
1.6
Df Model:                           8
Covariance Type:            nonrobust
```

In [95]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[95]:

	Features	VIF
0	const	157.05
3	curbweight	5.70
2	carwidth	4.94
4	cylindernumber	2.05
6	peugeot	1.30
8	dohcv	1.17
5	bmw	1.10
7	toyota	1.06
1	enginelocation	1.04

dohcv is insignificant in presence of other variables, because it has high p value of 0.237 can be dropped

In [96]:
```
x_train_new.columns
```

Out[96]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'bmw', 'peugeot', 'toyota', 'dohcv'],
      dtype='object')
```

In [97]:
```
x_train_new = x_train_new.drop(['dohcv'],axis =1)
```

Model 14: Rebuidling model without dohcv

In [98]:
```
x_train_lm = sm.add_constant(x_train_new)
lm14 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm14.summary())
```

```
                            OLS Regression Results
=============================================================================
==
Dep. Variable:                  price   R-squared:                       0.9
01
Model:                            OLS   Adj. R-squared:                  0.8
96
Method:                 Least Squares   F-statistic:                     17
5.7
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           1.39e-
64
Time:                        20:09:10   Log-Likelihood:                 182.
38
No. Observations:                 143   AIC:                            -34
8.8
Df Residuals:                     135   BIC:                            -32
5.0
Df Model:                           7
Covariance Type:            nonrobust
```

In [99]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tr
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[99]:

	Features	VIF
0	const	155.97
3	curbweight	5.37
2	carwidth	4.60
4	cylindernumber	1.94
6	peugeot	1.30
5	bmw	1.10
7	toyota	1.06
1	enginelocation	1.04

toyota is insignificant in presence of other variables, because it has high p value of 0.015 can be dropped

In [100]:
```
x_train_new.columns
```

Out[100]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'bmw', 'peugeot', 'toyota'],
      dtype='object')
```

In [101]:
```
x_train_new = x_train_new.drop(['toyota'],axis =1)
```

Model 15: Rebuidling model without toyota

In [102]:
```
x_train_lm = sm.add_constant(x_train_new)
lm15 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm15.summary())
```

```
                            OLS Regression Results
=============================================================================
==
Dep. Variable:                  price   R-squared:                       0.8
97
Model:                            OLS   Adj. R-squared:                  0.8
92
Method:                 Least Squares   F-statistic:                      19
6.7
Date:                Sun, 28 Apr 2019   Prob (F-statistic):            1.82e-
64
Time:                        20:09:11   Log-Likelihood:                 179.
24
No. Observations:                 143   AIC:                             -34
4.5
Df Residuals:                     136   BIC:                             -32
3.7
Df Model:                           6
Covariance Type:            nonrobust
```

In [103]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_t
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[103]:

	Features	VIF
0	const	155.97
3	curbweight	5.25
2	carwidth	4.41
4	cylindernumber	1.94
6	peugeot	1.29
5	bmw	1.08
1	enginelocation	1.04

const has high VIF value of 155.97 can be dropped

In [104]:
```
x_train_new.columns
```

Out[104]:
```
Index(['const', 'enginelocation', 'carwidth', 'curbweight', 'cylindernumber',
       'bmw', 'peugeot'],
      dtype='object')
```

In [105]:
```
x_train_new = x_train_new.drop(['const'],axis =1)
```

Model 16: Rebuidling model without const

In [106]:
```
x_train_lm = sm.add_constant(x_train_new)
lm16 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm16.summary())
```

```
                            OLS Regression Results
=============================================================================
==
Dep. Variable:                  price   R-squared:                       0.8
97
Model:                            OLS   Adj. R-squared:                  0.8
92
Method:                 Least Squares   F-statistic:                     19
6.7
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           1.82e-
64
Time:                        20:09:12   Log-Likelihood:                 179.
24
No. Observations:                 143   AIC:                            -34
4.5
Df Residuals:                     136   BIC:                            -32
3.7
Df Model:                           6
Covariance Type:            nonrobust
```

In [107]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tı
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[107]:

	Features	VIF
1	carwidth	32.22
2	curbweight	24.97
3	cylindernumber	9.95
0	enginelocation	8.32
5	peugeot	1.36
4	bmw	1.13

carwidth has high VIF value of 32.22 can be dropped

In [108]:
```
x_train_new.columns
```

Out[108]:
```
Index(['enginelocation', 'carwidth', 'curbweight', 'cylindernumber', 'bmw',
       'peugeot'],
      dtype='object')
```

In [109]:
```
x_train_new = x_train_new.drop(['carwidth'],axis =1)
```

Model 17: Rebuidling model without carwidth

In [110]:
```
x_train_lm = sm.add_constant(x_train_new)
lm17 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm17.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.8
74
Model:                            OLS   Adj. R-squared:                  0.8
70
Method:                 Least Squares   F-statistic:                      19
0.9
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           6.58e-
60
Time:                        20:09:12   Log-Likelihood:                 165.
33
No. Observations:                 143   AIC:                            -31
8.7
Df Residuals:                     137   BIC:                            -30
0.9
Df Model:                           5
Covariance Type:            nonrobust
```

In [111]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_tı
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[111]:

	Features	VIF
2	cylindernumber	9.88
1	curbweight	9.81
0	enginelocation	5.72
4	peugeot	1.36
3	bmw	1.10

cylindernumber has high VIF value of 9.88 can be dropped

In [112]:
```
x_train_new.columns
```

Out[112]:
```
Index(['enginelocation', 'curbweight', 'cylindernumber', 'bmw', 'peugeot'], dty
pe='object')
```

In [113]:
```
x_train_new = x_train_new.drop(['cylindernumber'],axis =1)
```

Model 18: Rebuidling model without cylindernumber

In [114]:
```
x_train_lm = sm.add_constant(x_train_new)
lm18 = sm.OLS(y_train,x_train_lm).fit()

#Let's see the summary of our linear model
print(lm18.summary())
```

```
Model:                            OLS   Adj. R-squared:                  0.8
62
Method:                 Least Squares   F-statistic:                     22
3.7
Date:                Sun, 28 Apr 2019   Prob (F-statistic):           2.96e-
59
Time:                        20:09:13   Log-Likelihood:                 160.
85
No. Observations:                 143   AIC:                            -31
1.7
Df Residuals:                     138   BIC:                            -29
6.9
Df Model:                           4
Covariance Type:            nonrobust
============================================================================
======
                     coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------------
------
```

In [115]:
```
# Create a dataframe that will contain the names of all the feature variables and
vif = pd.DataFrame()
vif['Features'] = x_train_new.columns
vif['VIF'] = [variance_inflation_factor(x_train_new.values, i) for i in range(x_t
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[115]:

	Features	VIF
1	curbweight	5.24
0	enginelocation	4.73
3	peugeot	1.18
2	bmw	1.09

curbweight has VIF value of 5.24 which is above 5 but if we drop it then the value of adjusted R square also drops to 18.6%

The above model shows the p values are significant and VIF value of 5.24 can be considered and not to drop it

Because the above model shows the adjusted R Square value of 86.2% which is a good result

Also value of AIC and BIC are negative which is a good indication

STEP 6: Residual Analysis of the train data

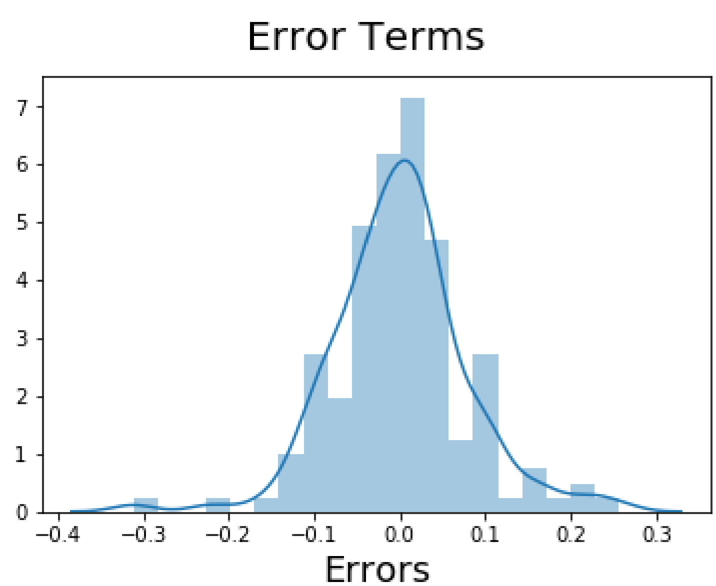
So, now to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

In [116]:
```
y_train_price =lm18.predict(x_train_lm)
```

In [117]:
```
# Importing the required libraries for plots.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [118]:
```
# Plot the histogram of the error terms
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)                         # X-label
```

Out[118]:
```
Text(0.5,0,'Errors')
```



STEP 7: Making Predictions Using the Final

Model

Now that we have fitted the model and checked the normality of error terms, it's time to go ahead and make predictions using the final, i.e. 18th model.

Applying the scaling on the test sets

In [119]:
```
num_vars = ['car_ID', 'symboling', 'wheelbase', 'carlength', 'carwidth','carheigh
            'boreratio','stroke','compressionratio','horsepower','peakrpm','citym
            'doornumber','cylindernumber']

df_test[num_vars] = scaler.transform(df_test[num_vars])      #We use fit_transfor

df_test.describe()
```

Out[119]:

	car_ID	symboling	fueltype	aspiration	doornumber	enginelocation	wheelbase	carler
count	62.000000	62.000000	62.000000	62.000000	62.000000	62.000000	62.000000	62.000
mean	0.550601	0.583871	0.887097	0.822581	0.564516	0.967742	0.437764	0.559
std	0.290690	0.271724	0.319058	0.385142	0.499868	0.178127	0.212861	0.189
min	0.014706	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.056
25%	0.323529	0.400000	1.000000	1.000000	0.000000	1.000000	0.313793	0.459
50%	0.571078	0.600000	1.000000	1.000000	1.000000	1.000000	0.387931	0.547
75%	0.816176	0.800000	1.000000	1.000000	1.000000	1.000000	0.570690	0.719
max	0.985294	1.000000	1.000000	1.000000	1.000000	1.000000	1.182759	1.089

8 rows × 67 columns

Dividing into X_test and y_test

In [120]:
```
y_test = df_test.pop('price')
x_test = df_test
```

In [121]:
```
# Now let's use our model to make predictions.

# 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)
```

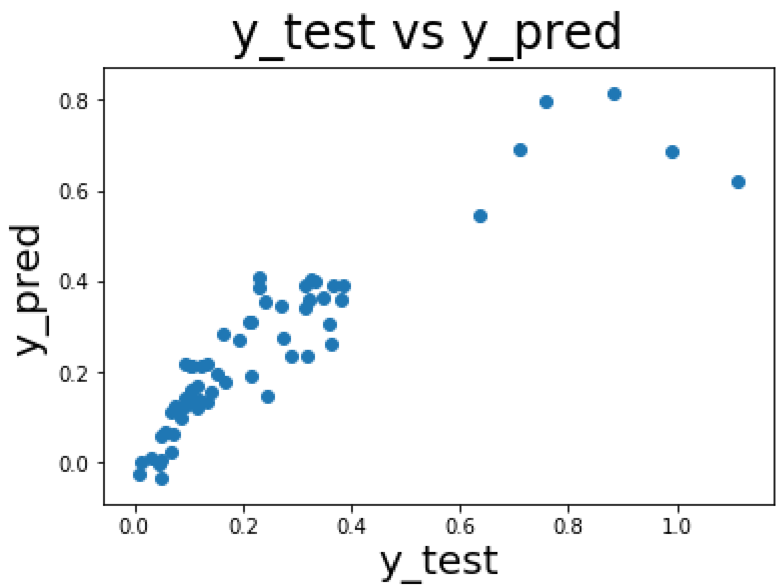
In [122]:
```
# Making predictions
y_pred = lm18.predict(x_test_new)
```

Step 8: Model Evaluation

Let's now plot the graph for actual versus predicted values.

In [123]:
```
# 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=24)              # Plot heading
plt.xlabel('y_test', fontsize=20)                          # X-label
plt.ylabel('y_pred', fontsize=20)                          # Y-label
```

Out[123]: `Text(0,0.5,'y_pred')`



In [124]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[124]: `0.8172843646703025`

We can see that the equation of our best fitted line is:

$$price = 0.36 \times const - 0.51 \times enginelocation + 0.89 \times curbweight + 0.24 \times bmw - 0.167 \times$$

Overall we have got a decent model