cars price case study

A Chinese automobile company have contracted an automobile consulting company to understand the factors on which the pricing of cars depends.The company wants to know Which variables are significant in predicting the price of a car.

step 1:

reading and understanding data

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

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

In [3]:
```
cars=pd.read_csv('/home/surekha/Downloads/CarPrice_Assignment.csv')
cars.info
```

Out[3]:
```
<bound method DataFrame.info of      car_ID  symboling
CarName fueltype aspiration  \
0         1          3           alfa-romero giulia      gas        std
1         2          3          alfa-romero stelvio      gas        std
2         3          1     alfa-romero Quadrifoglio      gas        std
3         4          2                  audi 100 ls      gas        std
4         5          2                   audi 100ls      gas        std
5         6          2                     audi fox      gas        std
6         7          1                   audi 100ls      gas        std
7         8          1                    audi 5000      gas        std
8         9          1                    audi 4000      gas      turbo
9        10          0          audi 5000s (diesel)      gas      turbo
10       11          2                     bmw 320i      gas        std
11       12          0                     bmw 320i      gas        std
12       13          0                       bmw x1      gas        std
13       14          0                       bmw x3      gas        std
14       15          1                       bmw z4      gas        std
15       16          0                       bmw x4      gas        std
16       17          0                       bmw x5      gas        std
17       18          0                       bmw x3      gas        std
18       19          2            chevrolet impala      gas        std
19       20          1       chevrolet monte carlo      gas        std
20       21          0          chevrolet vega 2300      gas        std
21       22          1               dodge rampage      gas        std
22       23          1          dodge challenger se      gas        std
23       24          1                   dodge d200      gas      turbo
24       25          1            dodge monaco (sw)      gas        std
25       26          1            dodge colt hardtop      gas        std
26       27          1              dodge colt (sw)      gas        std
27       28          1          dodge coronet custom      gas      turbo
28       29         -1            dodge dart custom      gas        std
29       30          3     dodge coronet custom (sw)      gas      turbo
..      ...        ...                          ...      ...        ...
175     176         -1                toyota corona      gas        std
176     177         -1               toyota corolla      gas        std
177     178         -1              toyota mark ii      gas        std
178     179          3       toyota corolla liftback      gas        std
179     180          3                toyota corona      gas        std
180     181         -1               toyota starlet      gas        std
181     182         -1               toyouta tercel      gas        std
182     183          2              vokswagen rabbit   diesel        std
183     184          2  volkswagen 1131 deluxe sedan      gas        std
184     185          2         volkswagen model 111   diesel        std
185     186          2            volkswagen type 3      gas        std
186     187          2          volkswagen 411 (sw)      gas        std
187     188          2        volkswagen super beetle   diesel      turbo
188     189          2           volkswagen dasher      gas        std
189     190          3                   vw dasher      gas        std
190     191          3                   vw rabbit      gas        std
191     192          0           volkswagen rabbit      gas        std
192     193          0     volkswagen rabbit custom   diesel      turbo
193     194          0           volkswagen dasher      gas        std
194     195         -2              volvo 145e (sw)      gas        std
195     196         -1                 volvo 144ea      gas        std
196     197         -2                 volvo 244dl      gas        std
197     198         -1                   volvo 245      gas        std
198     199         -2                 volvo 264gl      gas      turbo
199     200         -1                volvo diesel      gas      turbo
200     201         -1              volvo 145e (sw)      gas        std
201     202         -1                 volvo 144ea      gas      turbo
202     203         -1                 volvo 244dl      gas        std
203     204         -1                   volvo 246   diesel      turbo
204     205         -1                 volvo 264gl      gas      turbo

    doornumber      carbody drivewheel enginelocation  wheelbase  ...  \
0          two  convertible        rwd          front       88.6  ...
1          two  convertible        rwd          front       88.6  ...
2          two    hatchback        rwd          front       94.5  ...
```

In [4]:
```
cars.shape
```
Out[4]:
```
(205, 26)
```

In [5]:
```
cars.head()
```
Out[5]:

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

5 rows × 26 columns

In [6]:
```
cars.describe()
```
Out[6]:

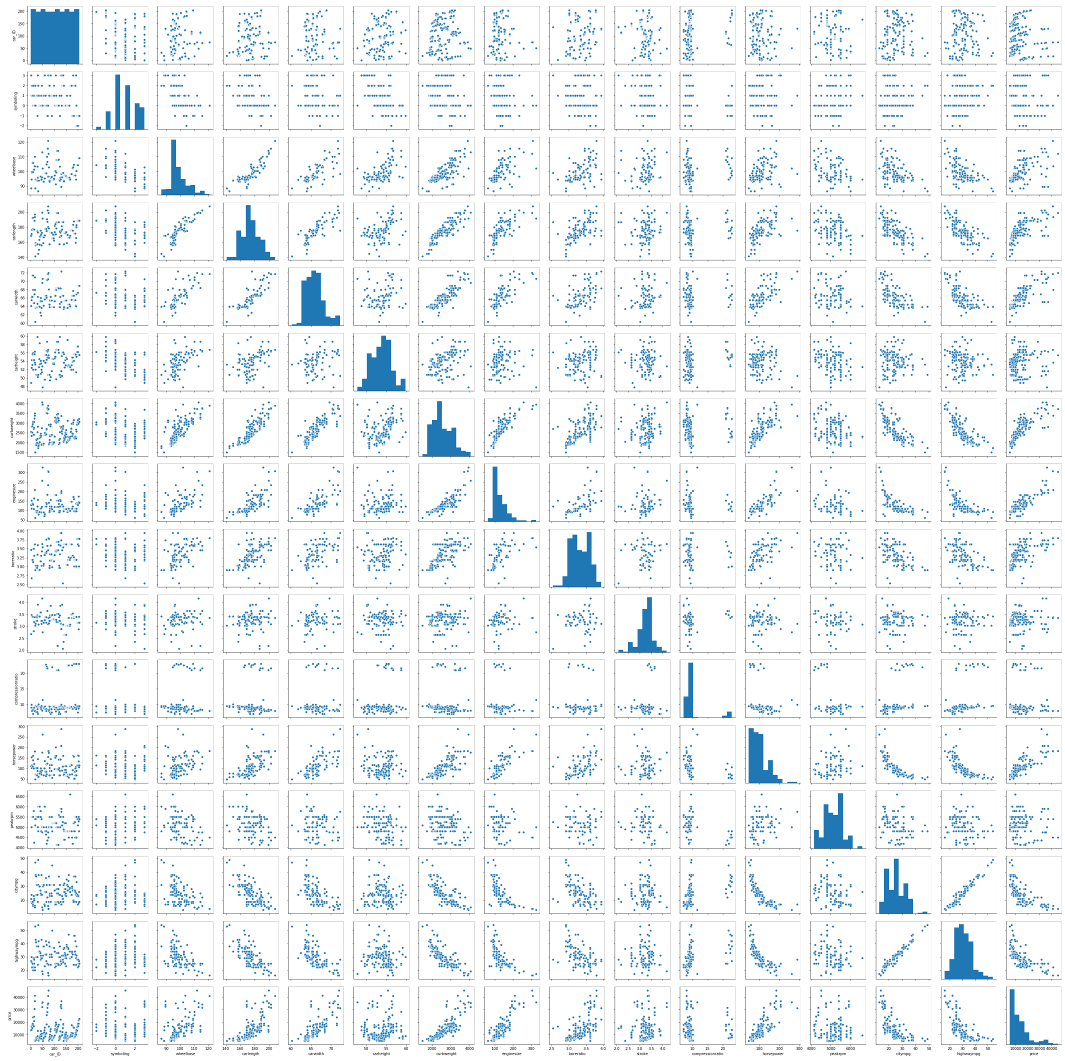
	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000

step 2:

visualising the data here we are using seaborn and matplotlib for visualising the data this visualisation helps us to understand the correlation among the variables

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

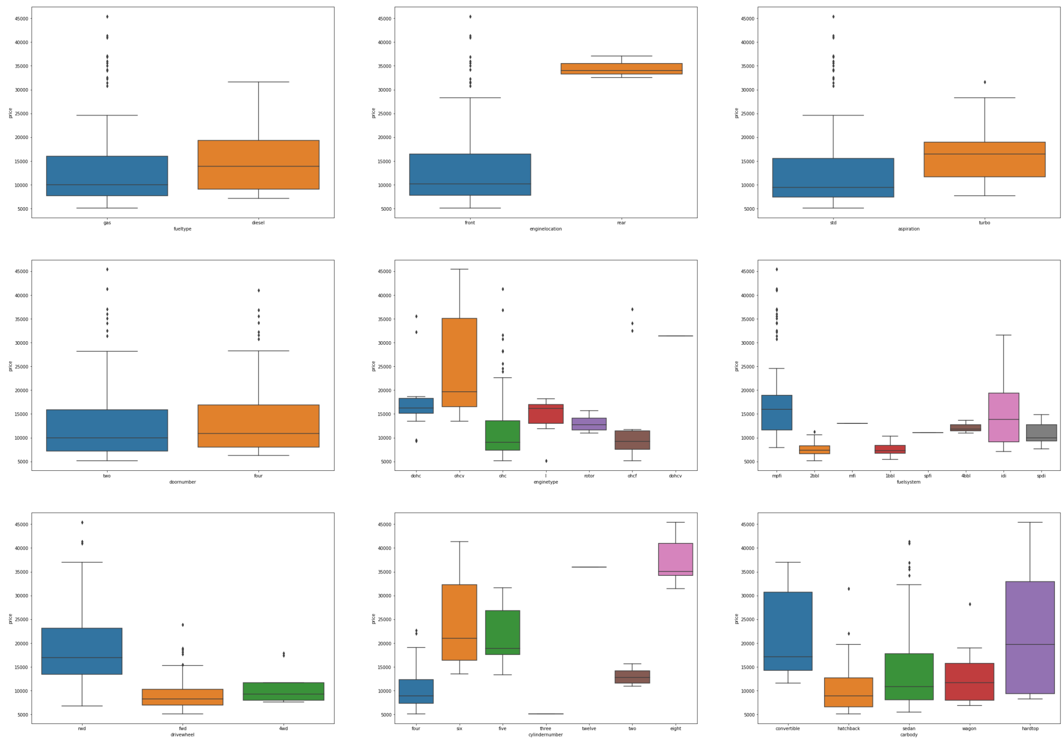
In [8]:
```
sns.pairplot(cars)
plt.show()
```



Visualising Categorical Variables As you might have noticed, there are a few categorical variables as well. Let's make a boxplot for some of these variables.

In [9]:
```
plt.figure(figsize=(42,30))
plt.subplot(3,3,1)
sns.boxplot(x='fueltype',y='price',data=cars)
plt.subplot(3,3,2)
sns.boxplot(x='enginelocation',y='price',data=cars)
plt.subplot(3,3,3)
sns.boxplot(x='aspiration',y='price',data=cars)
plt.subplot(3,3,4)
sns.boxplot(x='doornumber',y='price',data=cars)
plt.subplot(3,3,5)
sns.boxplot(x='enginetype',y='price',data=cars)
plt.subplot(3,3,6)
sns.boxplot(x='fuelsystem',y='price',data=cars)
plt.subplot(3,3,7)
sns.boxplot(x='drivewheel',y='price',data=cars)
plt.subplot(3,3,8)
sns.boxplot(x='cylindernumber',y='price',data=cars)
plt.subplot(3,3,9)
sns.boxplot(x='carbody',y='price',data=cars)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe5d2dfa400>



In [10]:
```
cars.info()
```

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

step 3:

as we have categorical variables so we need to create dummy variables.

In [11]:
```
cars = pd.get_dummies(cars,columns=['fueltype','enginelocation','aspiration
','doornumber','enginetype','fuelsystem','drivewheel','cylindernumber','carb
ody','symboling'],drop_first=True)
```

In [12]:
```
cars.head()
```

Out[12]:

	car_ID	CarName	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	strok
0	1	alfa-romero giulia	88.6	168.8	64.1	48.8	2548	130	3.47	2.6
1	2	alfa-romero stelvio	88.6	168.8	64.1	48.8	2548	130	3.47	2.6
2	3	alfa-romero Quadrifoglio	94.5	171.2	65.5	52.4	2823	152	2.68	3.4
3	4	audi 100 ls	99.8	176.6	66.2	54.3	2337	109	3.19	3.4
4	5	audi 100ls	99.4	176.6	66.4	54.3	2824	136	3.19	3.4

5 rows × 50 columns

here car id and carname are object type so we dont need to take them for further processing.hence we are dropping these columns.

In [13]:
```
cars.drop(['car_ID','CarName'],axis=1,inplace=True)
cars.head()
```

Out[13]:

	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio
0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0
1	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0
2	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0
3	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0
4	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0

5 rows × 48 columns

step 4:

training and testing data

In [14]:
```
from sklearn.model_selection import train_test_split
np.random.seed(0)
df_train, df_test = train_test_split(cars, train_size = 0.7, test_size = 0.
3, random_state = 100)
```

as each column has the values with different scaling so we cannot process them.here we are using scaling technique to standardise them. there are two different techniques to standardise they are minmax scaling and standardisation scaling.

In [15]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [16]:
```
num_vars=['wheelbase','carlength','carwidth','carheight','curbweight','engin
esize','stroke','compressionratio','horsepower','peakrpm','citympg','highway
mpg']
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

```
/home/surekha/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/dat
a.py:334: DataConversionWarning: Data with input dtype int64, float64 were al
l converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [17]:
```
df_train.head()
```

Out[17]:

	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressioni
122	0.244828	0.426016	0.291667	0.265487	0.272692	0.139623	2.97	0.525253	0.1
125	0.272414	0.452033	0.666667	0.212389	0.500388	0.339623	3.94	0.464646	0.1
166	0.272414	0.448780	0.308333	0.424779	0.314973	0.139623	3.24	0.449495	0.1
1	0.068966	0.450407	0.316667	0.088496	0.411171	0.260377	3.47	0.247475	0.1
199	0.610345	0.775610	0.575000	0.858407	0.647401	0.260377	3.62	0.484848	0.0

5 rows × 48 columns

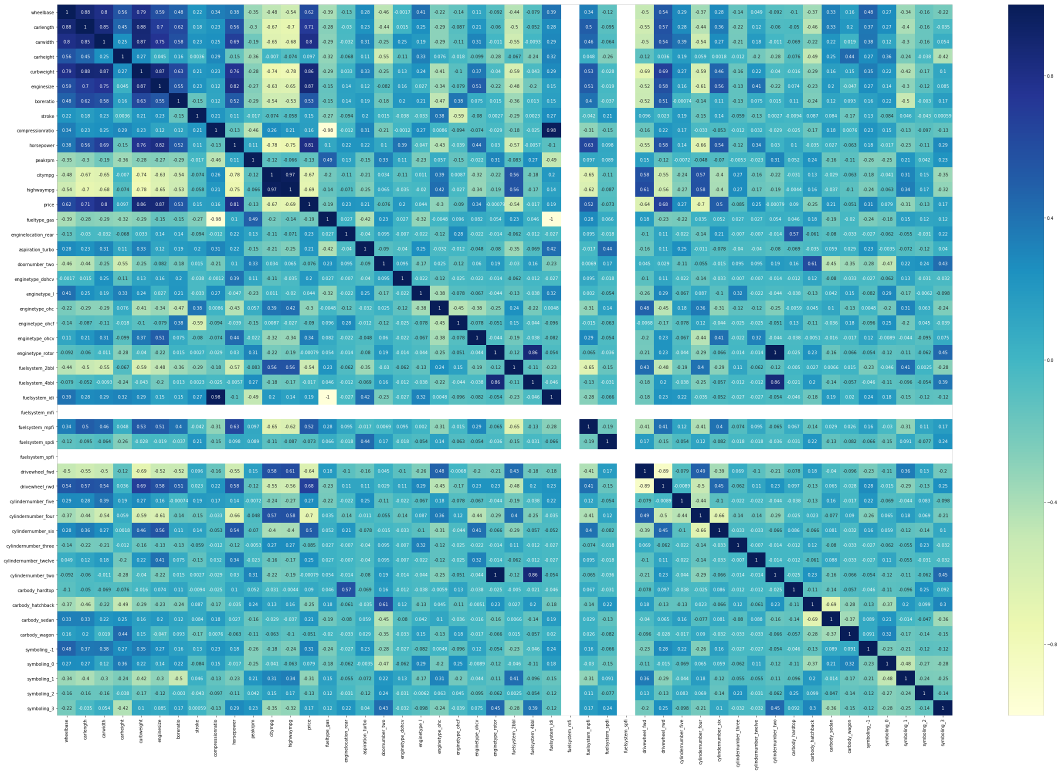
In [18]:
```
df_train.describe()
```

Out[18]:

	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000
mean	0.411141	0.525476	0.461655	0.509004	0.407878	0.241351	3.307413	0.535389
std	0.205581	0.204848	0.184517	0.215378	0.211269	0.154619	0.260997	0.157843
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.680000	0.000000
25%	0.272414	0.399187	0.304167	0.353982	0.245539	0.135849	3.065000	0.464646
50%	0.341379	0.502439	0.425000	0.522124	0.355702	0.184906	3.310000	0.545455
75%	0.503448	0.669919	0.550000	0.668142	0.559542	0.301887	3.540000	0.611111
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	3.940000	1.000000

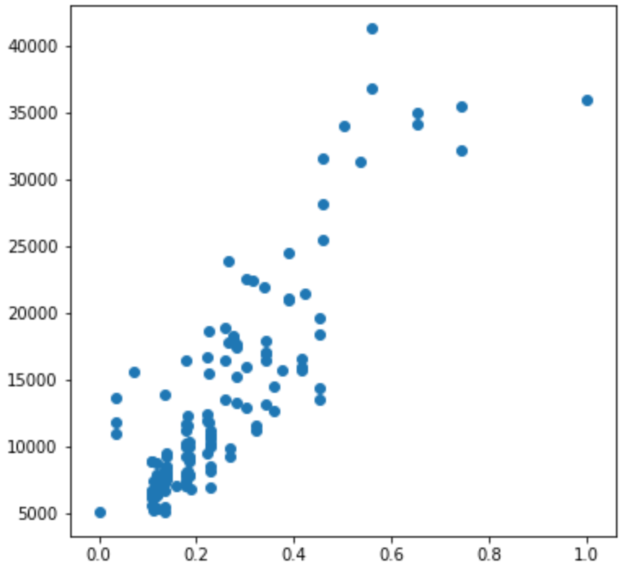
8 rows × 48 columns

In [19]:
```
plt.figure(figsize = (46, 30))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```

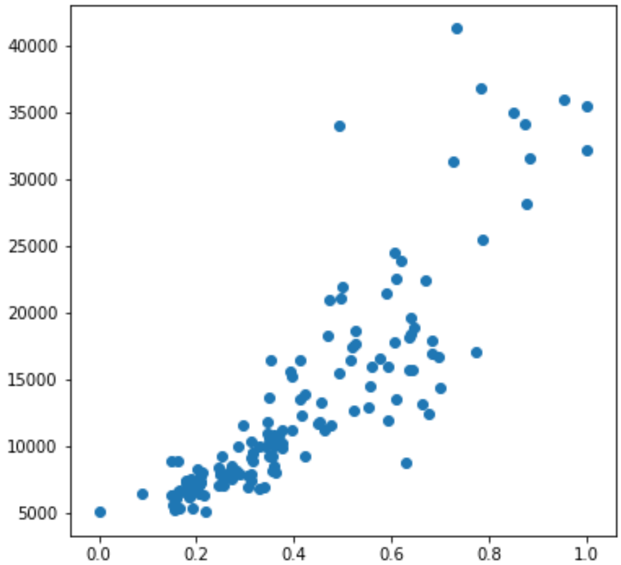


from the above heat map we can notice that price is strongly correlated with enginesize,curbweight,horsepower and carwidth.

In [20]:
```
plt.figure(figsize=[6,6])
plt.scatter(df_train.enginesize, df_train.price)
plt.show()
```



In [21]:
```
plt.figure(figsize=[6,6])
plt.scatter(df_train.curbweight, df_train.price)
plt.show()
```



In [22]:
```
y_train = df_train.pop('price')
X_train = df_train
```

step 5:

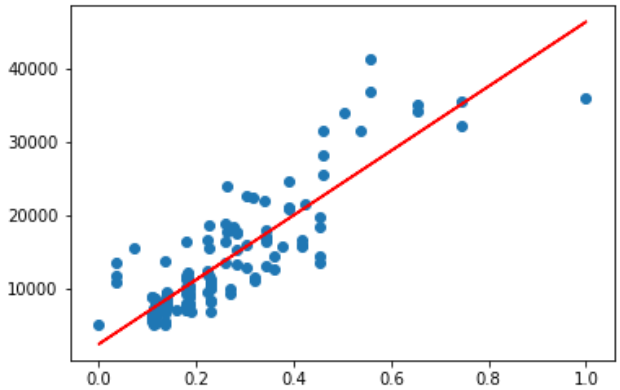
building a linear model Fit a regression line through the training data using statsmodels. Remember that in statsmodels, you need to explicitly fit a constant using sm.add_constant(X) because if we don't perform this step, statsmodels fits a regression line passing through the origin, by default.

In [23]:
```
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train[['enginesize']])
lr = sm.OLS(y_train, X_train_lm).fit()
```

In [24]:
```
lr.params
```

Out[24]:
```
const           2479.658045
enginesize     43822.834040
dtype: float64
```

In [25]:
```
plt.scatter(X_train_lm.iloc[:, 1], y_train)
plt.plot(X_train_lm.iloc[:, 1],2479.658045 +43822.834040 *X_train_lm.iloc[:,
1], 'r')
plt.show()
```



In [26]:
```
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.75
3
Model:                            OLS   Adj. R-squared:                  0.75
2
Method:                 Least Squares   F-statistic:                     430.
5
Date:                Sat, 25 Apr 2020   Prob (F-statistic):           1.09e-4
4
Time:                        15:02:31   Log-Likelihood:               -1384.
0
No. Observations:                 143   AIC:                             277
2.
Df Residuals:                     141   BIC:                             277
8.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
const       2479.6580    604.784      4.100      0.000    1284.041    3675.27
5
enginesize  4.382e+04   2112.124     20.748      0.000    3.96e+04     4.8e+0
4
==============================================================================
=
Omnibus:                       23.257   Durbin-Watson:                   1.99
0
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               32.41
0
Skew:                           0.885   Prob(JB):                     9.17e-0
8
Kurtosis:                       4.520   Cond. No.                         6.8
8
==============================================================================
=

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

for engine size r^2 value is 0.753. to obtain higher r^2 value we need to add second highest correlated variable i.e., curbweight.

In [27]:
```
X_train_lm = X_train[['enginesize', 'curbweight']]
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)
lr = sm.OLS(y_train, X_train_lm).fit()
lr.params
```

Out[27]:
```
const           480.684156
enginesize    24547.441288
curbweight    16306.603459
dtype: float64
```

In [28]:
```
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.80
2
Model:                            OLS   Adj. R-squared:                  0.79
9
Method:                 Least Squares   F-statistic:                     284.
0
Date:                Sat, 25 Apr 2020   Prob (F-statistic):           5.31e-5
0
Time:                        15:02:31   Log-Likelihood:                -1368.
2
No. Observations:                 143   AIC:                             274
2.
Df Residuals:                     140   BIC:                             275
1.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
const        480.6842    640.641      0.750      0.454    -785.898    1747.26
6
enginesize  2.455e+04   3783.064      6.489      0.000    1.71e+04     3.2e+0
4
curbweight  1.631e+04   2768.672      5.890      0.000    1.08e+04    2.18e+0
4
==============================================================================
=
Omnibus:                       36.002   Durbin-Watson:                   1.83
0
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               85.99
8
Skew:                           1.023   Prob(JB):                     2.12e-1
9
Kurtosis:                       6.202   Cond. No.                         17.
3
==============================================================================
=

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

by adding curbweight the rsquared value has increased so to increase more we have added two more highly correlated variables.

In [30]:
```
X_train_lm = X_train[['enginesize', 'curbweight','horsepower','carwidth']]
X_train_lm = sm.add_constant(X_train_lm)
lr = sm.OLS(y_train, X_train_lm).fit()
lr.params
```

Out[30]:
```
const          -609.443589
enginesize    17597.854690
curbweight     8880.339621
horsepower    10223.917101
carwidth       7521.883559
dtype: float64
```

In [31]:
```
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.82
6
Model:                            OLS   Adj. R-squared:                  0.82
1
Method:                 Least Squares   F-statistic:                     164.
3
Date:                Sat, 25 Apr 2020   Prob (F-statistic):          1.91e-5
1
Time:                        15:03:21   Log-Likelihood:                -1358.
9
No. Observations:                 143   AIC:                             272
8.
Df Residuals:                     138   BIC:                             274
3.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
const       -609.4436    763.462     -0.798      0.426   -2119.040     900.15
3
enginesize   1.76e+04   4105.594      4.286      0.000    9479.849    2.57e+0
4
curbweight  8880.3396   3536.616      2.511      0.013    1887.375    1.59e+0
4
horsepower  1.022e+04   2965.381      3.448      0.001    4360.459    1.61e+0
4
carwidth    7521.8836   3096.175      2.429      0.016    1399.806    1.36e+0
4
==============================================================================
=
Omnibus:                       36.489   Durbin-Watson:                   1.79
9
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               92.92
7
Skew:                           1.009   Prob(JB):                     6.62e-2
1
Kurtosis:                       6.394   Cond. No.                         22.
5
==============================================================================
=

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

step 6:

finding VIF Value

In [32]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_
train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[32]:

	Features	VIF
22	enginetype_rotor	inf
37	cylindernumber_two	inf
13	fueltype_gas	1731.82
25	fuelsystem_idi	452.34
8	compressionratio	210.66
5	enginesize	121.64
33	cylindernumber_four	112.38
12	highwaympg	46.57
11	citympg	45.38
4	curbweight	45.20
9	horsepower	39.97
40	carbody_sedan	31.95
44	symboling_1	27.38
43	symboling_0	27.01
34	cylindernumber_six	26.12
39	carbody_hatchback	24.15
32	cylindernumber_five	22.21
0	wheelbase	22.00
6	boreratio	20.23
1	carlength	19.89
46	symboling_3	16.12
41	carbody_wagon	16.03
2	carwidth	15.80
31	drivewheel_rwd	14.07
45	symboling_2	12.99
27	fuelsystem_mpfi	11.72
20	enginetype_ohcf	10.45
42	symboling_-1	10.09
30	drivewheel_fwd	9.83
19	enginetype_ohc	9.65
36	cylindernumber_twelve	9.56
23	fuelsystem_2bbl	8.92
18	enginetype_l	8.89
7	stroke	7.49
35	cylindernumber_three	6.99
17	enginetype_dohcv	6.78
15	aspiration_turbo	5.63
24	fuelsystem_4bbl	5.28
3	carheight	4.76
38	carbody_hardtop	4.46
10	peakrpm	4.40
28	fuelsystem_spdi	3.99

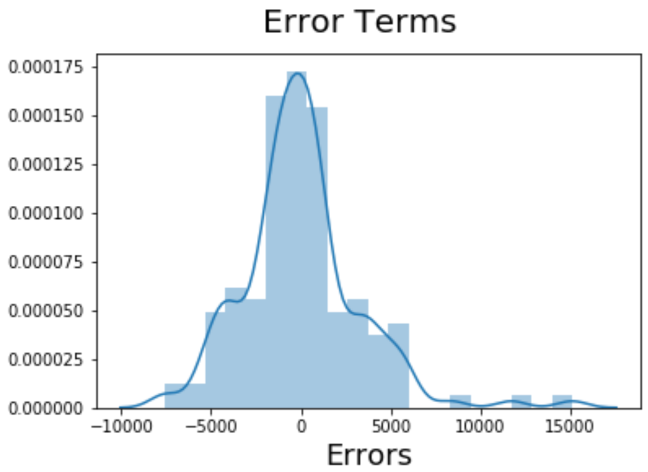
step 7:

Residual Analysis of the train data

In [33]:
```
y_train_price = lr.predict(X_train_lm)
```

In [34]:
```
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)
```

Out[34]:
```
Text(0.5, 0, 'Errors')
```



step 8:

Making Predictions Using the Final Model

In [35]:
```
num_vars=['wheelbase','carlength','carwidth','carheight','curbweight','engin
esize','stroke','compressionratio','horsepower','peakrpm','citympg','highway
mpg']
df_test[num_vars] = scaler.fit_transform(df_test[num_vars])
```

```
/home/surekha/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/dat
a.py:334: DataConversionWarning: Data with input dtype int64, float64 were al
l converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [36]:
```
df_test.describe()
```

Out[36]:

	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compre
count	62.000000	62.000000	62.000000	62.000000	62.000000	62.000000	62.000000	62.000000	
mean	0.370121	0.486741	0.375212	0.454249	0.371743	0.228835	3.381290	0.654504	
std	0.179970	0.183964	0.208977	0.234487	0.222354	0.188416	0.287889	0.173913	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.540000	0.000000	
25%	0.265306	0.389764	0.221053	0.264423	0.219125	0.126638	3.190000	0.590164	
50%	0.327988	0.475591	0.326316	0.485577	0.344065	0.183406	3.390000	0.699454	
75%	0.482507	0.642126	0.421053	0.605769	0.540726	0.287118	3.620000	0.759563	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	3.800000	1.000000	

8 rows × 48 columns

dividing test data set into X and Y

In [37]:
```
y_test = df_test.pop('price')
```

In [38]:
```
y_test.head()
```

Out[38]:
```
160    7738.0
186    8495.0
59     8845.0
165    9298.0
140    7603.0
Name: price, dtype: float64
```

In [39]:
```
X_test =df_test[['enginesize','curbweight','horsepower','carwidth']]
X_test.head()
```

Out[39]:

	enginesize	curbweight	horsepower	carwidth
160	0.082969	0.132148	0.116129	0.200000
186	0.131004	0.219125	0.212903	0.315789
59	0.187773	0.271985	0.206452	0.421053
165	0.082969	0.214320	0.387097	0.157895
140	0.126638	0.202307	0.135484	0.136842

In [40]:
```
X_test_m = sm.add_constant(X_test)
y_pred_m = lr.predict(X_test_m)
```

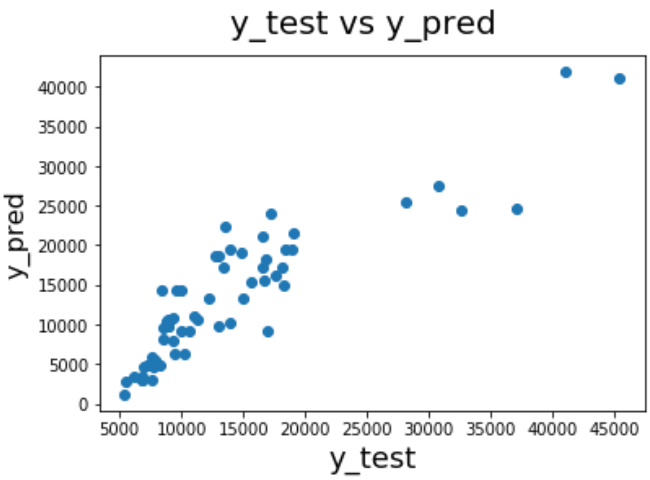
In [41]:
```
y_pred_m = lr.predict(X_test_m)
```

step 9:

evaluating the model

In [42]:
```
fig = plt.figure()
plt.scatter(y_test, y_pred_m)
fig.suptitle('y_test vs y_pred', fontsize = 20)
plt.xlabel('y_test', fontsize = 18)
plt.ylabel('y_pred', fontsize = 16)
```

Out[42]: `Text(0, 0.5, 'y_pred')`



We can see that the equation of our best fitted line is: price= (1.76e+04)*enginesize+8880.3396*curbweight+1.022e+04*horsepower+7521.8836*carwidth