In [27]:

```
import pandas as pd
import numpy as np
pd.pandas.set_option('display.max_columns',None)
```

In [28]:

```
autott = pd.read_csv('x_automobile.csv')

#print shape
print(autott.shape)
```

(201, 26)

In [29]:

```
autott.head()
```

Out[29]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wh b
0	3	122.0	alfa-romero	gas	std	2	convertible	rwd	front	
1	3	122.0	alfa-romero	gas	std	2	convertible	rwd	front	
2	1	122.0	alfa-romero	gas	std	2	hatchback	rwd	front	
3	2	164.0	audi	gas	std	4	sedan	fwd	front	
4	2	164.0	audi	gas	std	4	sedan	4wd	front	

In [30]:

```
#1.display unique values

autott['symboling'].unique()
```

Out[30]:

```
array([ 3,  1,  2,  0, -1, -2], dtype=int64)
```

In [31]:

```
#One-Hot Encoding
symb_dum = pd.get_dummies(autott['symboling'])
```

In [32]:

```
symb_dum
```

Out[32]:

	-2	-1	0	1	2	3
0	0	0	0	0	0	1
1	0	0	0	0	0	1
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	0	0	1	0
...	...	...	...	...	...	...
196	0	1	0	0	0	0
197	0	1	0	0	0	0
198	0	1	0	0	0	0
199	0	1	0	0	0	0
200	0	1	0	0	0	0

201 rows × 6 columns

In [33]:

```
# 2.for unique values
autott['num-of-cylinders'].unique()
```

Out[33]:

```
array([ 4,  6,  5,  3, 12,  2,  8], dtype=int64)
```

In [34]:

```
#One-Hot Encoding
noofcyl_dum = pd.get_dummies(autott['num-of-cylinders'])
```

In [35]:

```
noofcyl_dum
```

Out[35]:

	2	3	4	5	6	8	12
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0
3	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...
196	0	0	1	0	0	0	0
197	0	0	1	0	0	0	0
198	0	0	0	0	1	0	0
199	0	0	0	0	1	0	0
200	0	0	1	0	0	0	0

201 rows × 7 columns

In [37]:

```
autott.columns
```

Out[37]:

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiratio
n',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-t
ype',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

In [80]:

```
#3. for unique values
autott['make'].unique()
```

Out[80]:

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

In [81]:

```
#autott['make'].astype(str).astype(int)
```

In [82]:

```
#One-Hot Encoding
make_dum = pd.get_dummies(autott['make'])
make_dum
```

Out[82]:

	alfa-romero	audi	bmw	chevrolet	dodge	honda	isuzu	jaguar	mazda	mercedes-benz	mercu
0	1	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	
3	0	1	0	0	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
196	0	0	0	0	0	0	0	0	0	0	
197	0	0	0	0	0	0	0	0	0	0	
198	0	0	0	0	0	0	0	0	0	0	
199	0	0	0	0	0	0	0	0	0	0	
200	0	0	0	0	0	0	0	0	0	0	

201 rows × 22 columns

In [60]:

```
#4. for unique values
autott['fuel-type'].unique()
```

Out[60]:

```
array(['gas', 'diesel'], dtype=object)
```

In [61]:

```
#One-Hot Encoding
futy_dum = pd.get_dummies(autott['fuel-type'])
futy_dum
```

Out[61]:

	diesel	gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
...	...	...
196	0	1
197	0	1
198	0	1
199	1	0
200	0	1

201 rows × 2 columns

In [62]:

```
#5. for unique values
autott['aspiration'].unique()
```

Out[62]:

```
array(['std', 'turbo'], dtype=object)
```

In [63]:

```
#One-Hot Encoding
asp_dum = pd.get_dummies(autott['aspiration'])
asp_dum
```

Out[63]:

	std	turbo
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
196	1	0
197	0	1
198	1	0
199	0	1
200	0	1

201 rows × 2 columns

In [64]:

```
#6. for unique values
autott['body-style'].unique()
```

Out[64]:

```
array(['convertible', 'hatchback', 'sedan', 'wagon', 'hardtop'],
      dtype=object)
```

In [65]:

```
#One-Hot Encoding
bsty_dum = pd.get_dummies(autott['body-style'])
bsty_dum
```

Out[65]:

	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
...	...	...	...	...	...
196	0	0	0	1	0
197	0	0	0	1	0
198	0	0	0	1	0
199	0	0	0	1	0
200	0	0	0	1	0

201 rows × 5 columns

In [66]:

```
#7. for unique values
autott['drive-wheels'].unique()
```

Out[66]:

```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

In [67]:

```
#One-Hot Encoding
dwhee_dum = pd.get_dummies(autott['drive-wheels'])
dwhee_dum
```

Out[67]:

	4wd	fwd	rwd
0	0	0	1
1	0	0	1
2	0	0	1
3	0	1	0
4	1	0	0
...	...	...	...
196	0	0	1
197	0	0	1
198	0	0	1
199	0	0	1
200	0	0	1

201 rows × 3 columns

In [68]:

```
#8. for unique values
autott['engine-location'].unique()
```

Out[68]:

```
array(['front', 'rear'], dtype=object)
```

In [50]:

```
#One-Hot Encoding
eloc_dum = pd.get_dummies(autott['engine-location'])
eloc_dum
```

Out[50]:

	front	rear
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
196	1	0
197	1	0
198	1	0
199	1	0
200	1	0

201 rows × 2 columns

In [51]:

```
#9. for unique values
autott['engine-type'].unique()
```

Out[51]:

```
array(['dohc', 'ohcv', 'ohc', 'l', 'rotor', 'ohcf'], dtype=object)
```

In [52]:

```
#One-Hot Encoding
etype_dum = pd.get_dummies(autott['engine-type'])
etype_dum
```

Out[52]:

	dohc	l	ohc	ohcf	ohcv	rotor
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	0	0
4	0	0	1	0	0	0
...	...	...	...	...	...	...
196	0	0	1	0	0	0
197	0	0	1	0	0	0
198	0	0	0	0	1	0
199	0	0	1	0	0	0
200	0	0	1	0	0	0

201 rows × 6 columns

In [53]:

```
#10. for unique values
autott['fuel-system'].unique()
```

Out[53]:

```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

In [54]:

```
#One-Hot Encoding
fsys_dum = pd.get_dummies(autott['fuel-system'])
fsys_dum
```

Out[54]:

	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
...	...	...	...	...	...	...	...	...
196	0	0	0	0	0	1	0	0
197	0	0	0	0	0	1	0	0
198	0	0	0	0	0	1	0	0
199	0	0	0	1	0	0	0	0
200	0	0	0	0	0	1	0	0

201 rows × 8 columns

In [69]:

```
autott.corr()['price']
```

Out[69]:

```
symboling           -0.082391
normalized-losses    0.133999
num-of-doors         0.042435
wheel-base           0.584642
length               0.690628
width                0.751265
height               0.135486
curb-weight          0.834415
num-of-cylinders     0.708645
engine-size          0.872335
bore                 0.543154
stroke               0.082267
compression-ratio    0.071107
horsepower           0.809681
peak-rpm            -0.101542
city-mpg            -0.686571
highway-mpg         -0.704692
price                1.000000
Name: price, dtype: float64
```

In [90]:

```
cols = ['symboling', 'normalized-losses','num-of-doors','wheel-base','length','width',
'height','curb-weight','num-of-cylinders', 'engine-size', 'bore', 'stroke', 'compressio
n-ratio', 'horsepower']
X = autott[cols]
```

In [94]:

```
col = ['price']
y = autott[col]
```

In [95]:

```
X.head()
```

Out[95]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	num-of-cylinders	engine-size
0	3	122.0	2	88.6	168.8	64.1	48.8	2548.0	4	130.0
1	3	122.0	2	88.6	168.8	64.1	48.8	2548.0	4	130.0
2	1	122.0	2	94.5	171.2	65.5	52.4	2823.0	6	152.0
3	2	164.0	4	99.8	176.6	66.2	54.3	2337.0	4	109.0
4	2	164.0	4	99.4	176.6	66.4	54.3	2824.0	5	136.0

In [96]:

```
y.head()
```

Out[96]:

	price
0	13495.0
1	16500.0
2	16500.0
3	13950.0
4	17450.0

In [97]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.2)
```

In [98]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(160, 14)
(160, 1)
(41, 14)
(41, 1)
```

In [99]:

```
from sklearn import linear_model

regr = linear_model.LinearRegression()
```

In [100]:

```
#Training
regr.fit(X_train, y_train)
```

Out[100]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

In [101]:

```
# The coefficients
print('Coefficients', regr.coef_)
print('Intercept',regr.intercept_)
```

```
Coefficients [[ 1.47879848e+02  7.42896138e+00  2.12518863e+02  1.54564262
e+02
  -9.46595877e+01  6.85385449e+02  3.00157254e+02 -1.44589385e+00
  -2.38390111e+03  1.69967430e+02 -5.41231627e+03 -4.76940227e+03
   2.72379888e+02  1.00640578e+02]]
Intercept [-35684.20659162]
```

In [102]:

```
#Testing
y_pred = regr.predict(X_test)
```

In [103]:

```
y_pred
```

Out[103]:

```
array([[ 5707.55390984],
       [14331.84891083],
       [13789.23468193],
       [ 4889.27424584],
       [26049.02805766],
       [ 6348.07439108],
       [13868.75884374],
       [15743.85509075],
       [ 6990.59765453],
       [ 5279.69103059],
       [ 6280.21539342],
       [14409.069952  ],
       [17582.89647939],
       [43013.79379974],
       [ 8899.02457984],
       [13323.35200029],
       [ 9874.64812059],
       [17253.24200685],
       [ 7802.35968938],
       [21753.94537072],
       [ 5149.38564485],
       [ 5985.39919271],
       [32418.99561073],
       [ 7847.8546629 ],
       [22161.18637384],
       [17336.74294155],
       [19330.11601361],
       [27392.16213216],
       [ 6711.6393538 ],
       [11303.01552984],
       [ 9400.42111952],
       [22652.07862244],
       [ 9124.6744195 ],
       [ 8745.17394038],
       [14666.43905748],
       [18699.18984094],
       [ 5253.4772116 ],
       [ 6149.29644861],
       [14519.225046  ],
       [ 7802.35968938],
       [10016.31323527]])
```

In [104]:

```
#Evaluation
from sklearn.metrics import mean_squared_error, r2_score
print(r2_score(y_test, y_pred)) #Coefficient of Determination
print(mean_squared_error(y_test, y_pred)) #MSE
```

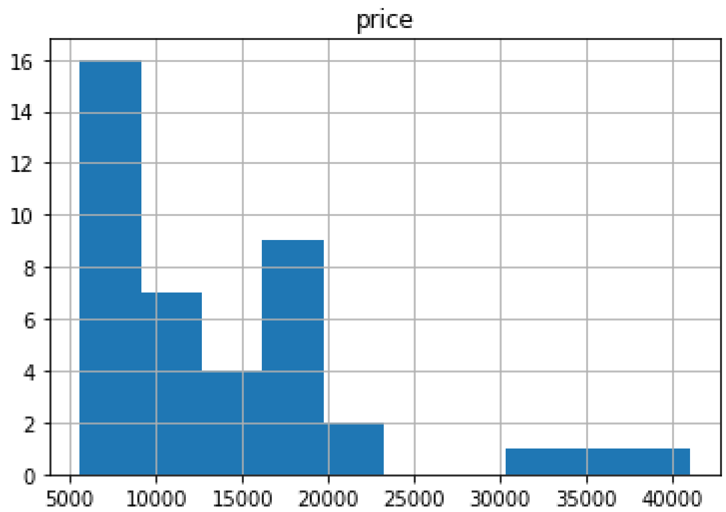
```
0.8452136421073411
9290584.47382907
```

In [105]:

```
y_test.hist()
```

Out[105]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000C702A4
8>]],
      dtype=object)
```

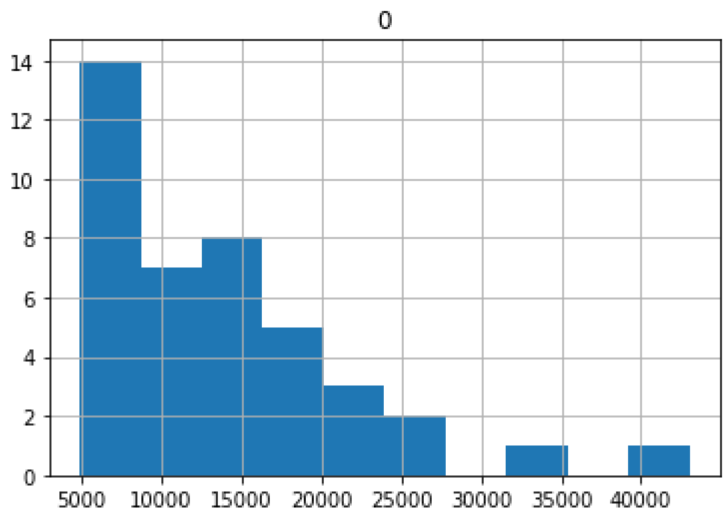


In [106]:

```
pd.DataFrame(y_pred).hist()
```

Out[106]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000C42FE0
8>]],
      dtype=object)
```

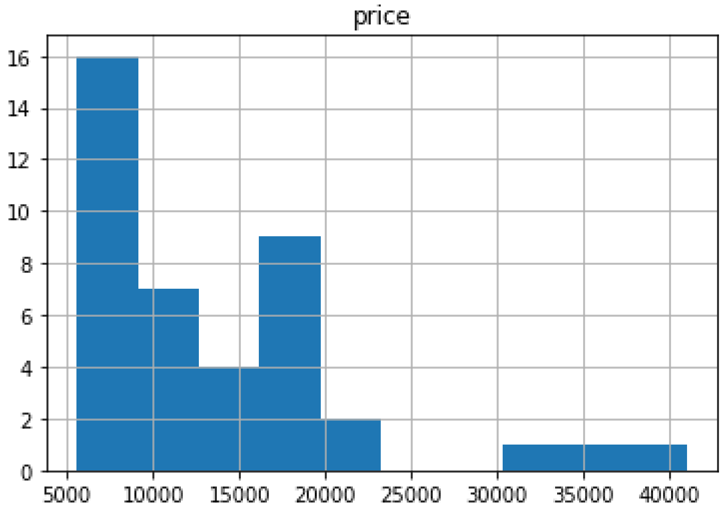


In [107]:

```
y_test.hist()
```

Out[107]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000C85788
8>]],
      dtype=object)
```

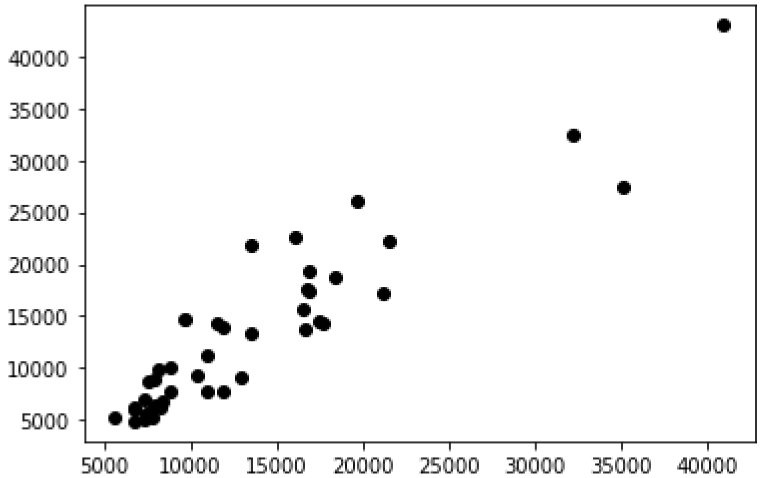


In [108]:

```
import matplotlib.pyplot as plt
# Plot outputs
plt.scatter(y_test, y_pred,  color='black')
```

Out[108]:

```
<matplotlib.collections.PathCollection at 0xca67b48>
```



In [112]:

```
autott_c=autott.corr()
autott_c
```

Out[112]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height
symboling	1.000000	0.466264	-0.672344	-0.535987	-0.365404	-0.242423	-0.550160
normalized-losses	0.466264	1.000000	-0.361368	-0.056661	0.019424	0.086802	-0.373737
num-of-doors	-0.672344	-0.361368	1.000000	0.445245	0.395122	0.227655	0.538383
wheel-base	-0.535987	-0.056661	0.445245	1.000000	0.876024	0.814507	0.590742
length	-0.365404	0.019424	0.395122	0.876024	1.000000	0.857170	0.492063
width	-0.242423	0.086802	0.227655	0.814507	0.857170	1.000000	0.306002
height	-0.550160	-0.373737	0.538383	0.590742	0.492063	0.306002	1.000000
curb-weight	-0.233118	0.099404	0.208517	0.782097	0.880665	0.866201	0.307581
num-of-cylinders	-0.118016	0.111362	0.002225	0.348931	0.440348	0.520118	0.007776
engine-size	-0.110581	0.112360	0.024094	0.572027	0.685025	0.729436	0.074694
bore	-0.139896	-0.029800	0.119140	0.493203	0.608941	0.544879	0.180327
stroke	-0.007992	0.055127	-0.007780	0.157964	0.123913	0.188814	-0.060822
compression-ratio	-0.182196	-0.114713	0.169164	0.250313	0.159733	0.189867	0.259737
horsepower	0.075790	0.217300	-0.102856	0.371250	0.579731	0.615006	-0.086941
peak-rpm	0.279719	0.239544	-0.232031	-0.360233	-0.286035	-0.245852	-0.309913
city-mpg	-0.035527	-0.225016	-0.027617	-0.470606	-0.665192	-0.633531	-0.049800
highway-mpg	0.036233	-0.181877	-0.045787	-0.543304	-0.698142	-0.680635	-0.104812
price	-0.082391	0.133999	0.042435	0.584642	0.690628	0.751265	0.135486

In [125]:

```
from sklearn.preprocessing import Normalizer

autott_n = Normalizer().fit_transform(autott_c)
autott_n = pd.DataFrame(autott_n)
print(autott_n)
```

```
           0         1         2         3         4         5         6  \
0   0.611032  0.284902 -0.410824 -0.327505 -0.223274 -0.148128 -0.336166
1   0.351754  0.754411 -0.272620 -0.042746  0.014653  0.065484 -0.281951
2  -0.430893 -0.231594  0.640882  0.285349  0.253227  0.145900  0.345040
3  -0.222824 -0.023556  0.185100  0.415726  0.364186  0.338611  0.245587
4  -0.141068  0.007499  0.152541  0.338199  0.386061  0.330920  0.189966
5  -0.095832  0.034314  0.089995  0.321984  0.338849  0.395312  0.120966
6  -0.331379 -0.225114  0.324286  0.355823  0.296385  0.184315  0.602333
7  -0.085489  0.036453  0.076467  0.286809  0.322956  0.317651  0.112795
8  -0.058086  0.054811  0.001095  0.171740  0.216734  0.255996  0.003827
9  -0.043179  0.043874  0.009408  0.223363  0.267486  0.284827  0.029166
10 -0.068746 -0.014644  0.058546  0.242362  0.299236  0.267756  0.088613
11 -0.007252  0.050023 -0.007059  0.143337  0.112439  0.171331 -0.055190
12 -0.137712 -0.086706  0.127862  0.189198  0.120734  0.143510  0.196321
13  0.030969  0.088793 -0.042029  0.151699  0.236888  0.251302 -0.035525
14  0.196391  0.168184 -0.162909 -0.252919 -0.200825 -0.172612 -0.217590
15 -0.014684 -0.093001 -0.011414 -0.194507 -0.274931 -0.261845 -0.020583
16  0.014638 -0.073479 -0.018498 -0.219498 -0.282053 -0.274980 -0.042345
17 -0.032752  0.053267  0.016869  0.232408  0.274540  0.298644  0.053859

           7         8         9        10        11        12        13  \
0  -0.142443 -0.072112 -0.067568 -0.085481 -0.004883 -0.111328  0.046310
1   0.074992  0.084013  0.084766 -0.022481  0.041589 -0.086541  0.163933
2   0.133635  0.001426  0.015441  0.076354 -0.004986  0.108414 -0.065919
3   0.325138  0.145060  0.237806  0.205037  0.065670  0.104062  0.154338
4   0.339990  0.170001  0.264461  0.235088  0.047838  0.061667  0.223811
5   0.342419  0.205609  0.288354  0.215397  0.074640  0.075057  0.243119
6   0.185266  0.004684  0.044991  0.108617 -0.036635  0.156448 -0.052367
7   0.366718  0.221664  0.311370  0.236181  0.061393  0.057367  0.277970
8   0.297505  0.492188  0.417198  0.099669  0.006474 -0.008927  0.329166
9   0.331542  0.330982  0.390476  0.223553  0.080362  0.011280  0.321225
10  0.316484  0.099510  0.281337  0.491404 -0.027219  0.000614  0.278547
11  0.151911  0.011935  0.186749 -0.050261  0.907406  0.170459  0.088758
12  0.118239 -0.013710  0.021835  0.000945  0.141988  0.755846 -0.162076
13  0.309729  0.273276  0.336149  0.231620  0.039969 -0.087620  0.408617
14 -0.196131 -0.106616 -0.180267 -0.187698 -0.044737 -0.305920  0.075744
15 -0.309794 -0.181622 -0.268878 -0.240597 -0.014085  0.136982 -0.339799
16 -0.321139 -0.191396 -0.274550 -0.238925 -0.014036  0.108461 -0.325057
17  0.331698  0.281702  0.346772  0.215915  0.032703  0.028267  0.321866

          14        15        16        17
0   0.170918 -0.021708  0.022139 -0.050344
1   0.180714 -0.169754 -0.137210  0.101090
2  -0.148704 -0.017699 -0.029344  0.027196
3  -0.149758 -0.195643 -0.225866  0.243051
4  -0.110427 -0.256805 -0.269525  0.266625
5  -0.097188 -0.250442 -0.269063  0.296984
6  -0.186671 -0.029996 -0.063132  0.081608
7  -0.102443 -0.274871 -0.291500  0.305995
8  -0.074741 -0.216283 -0.233172  0.348787
9  -0.100256 -0.254022 -0.265356  0.340626
10 -0.131371 -0.286057 -0.290612  0.266908
11 -0.057820 -0.030923 -0.031525  0.074650
12 -0.329338  0.250506  0.202918  0.053746
13  0.044082 -0.335940 -0.328768  0.330850
14  0.702100 -0.080993 -0.041147 -0.071293
15 -0.047679  0.413311  0.401756 -0.283767
16 -0.023677  0.392711  0.404005 -0.284699
17 -0.040365 -0.272927 -0.280130  0.397522
```

In [115]:

```
#Multi-Collinearity
import numpy as np
X = np.array(X)
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif = [variance_inflation_factor(X, i) for i in range(X.shape[1])]
print(vif)
X = pd.DataFrame(X)
```

```
[4.416330630971977, 25.393536476990423, 24.90683094808148, 2484.7508049019
903, 1910.6592082674226, 3331.450141300592, 1037.0246631169791, 301.320778
10120484, 199.41233738983783, 243.18102684694048, 546.7112374563237, 176.9
7950909281306, 11.175081446180052, 52.725884422202896]
```

In [117]:

```
X = X.iloc[:,:]
```

In [118]:

```
X.head()
```

Out[118]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	3.0	122.0	2.0	88.6	168.8	64.1	48.8	2548.0	4.0	130.0	3.47	2.68	9.0	111.0
1	3.0	122.0	2.0	88.6	168.8	64.1	48.8	2548.0	4.0	130.0	3.47	2.68	9.0	111.0
2	1.0	122.0	2.0	94.5	171.2	65.5	52.4	2823.0	6.0	152.0	2.68	3.47	9.0	154.0
3	2.0	164.0	4.0	99.8	176.6	66.2	54.3	2337.0	4.0	109.0	3.19	3.40	10.0	102.0
4	2.0	164.0	4.0	99.4	176.6	66.4	54.3	2824.0	5.0	136.0	3.19	3.40	8.0	115.0

In [119]:

```
#Feature Selection
from sklearn.feature_selection import RFE

logreg = linear_model.LinearRegression()
rfe = RFE(logreg, 6)
rfe = rfe.fit(X, y)

print(rfe.support_)
print(rfe.ranking_)
```

```
[ True False False False False  True False False  True  True  True  True
 False False]
[1 8 5 6 7 1 2 9 1 1 1 1 3 4]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:76
0: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

In [120]:

```
d = rfe.support_

g = X.columns

a = g[d]

X_f = X[a]
```

In [121]:

```
X_f.head()
```

Out[121]:

	0	5	8	9	10	11
0	3.0	64.1	4.0	130.0	3.47	2.68
1	3.0	64.1	4.0	130.0	3.47	2.68
2	1.0	65.5	6.0	152.0	2.68	3.47
3	2.0	66.2	4.0	109.0	3.19	3.40
4	2.0	66.4	5.0	136.0	3.19	3.40

In [122]:

```
import statsmodels.api as sm
```

In [123]:

```
results_1 = sm.OLS(y,X).fit()
results_1.summary()
```

Out[123]:

OLS Regression Results

Dep. Variable:	price	**R-squared (uncentered):**	0.958
Model:	OLS	**Adj. R-squared (uncentered):**	0.955
Method:	Least Squares	**F-statistic:**	303.6
Date:	Tue, 16 Jun 2020	**Prob (F-statistic):**	2.29e-120
Time:	12:19:36	**Log-Likelihood:**	-1905.1
No. Observations:	201	**AIC:**	3838.
Df Residuals:	187	**BIC:**	3884.
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
0	44.9727	322.266	0.140	0.889	-590.771	680.717
1	2.6991	9.240	0.292	0.771	-15.529	20.927
2	226.7451	350.086	0.648	0.518	-463.880	917.371
3	94.2343	116.447	0.809	0.419	-135.484	323.953
4	-75.6912	57.877	-1.308	0.193	-189.867	38.485
5	386.1030	202.455	1.907	0.058	-13.285	785.491
6	117.3922	138.350	0.849	0.397	-155.534	390.319
7	1.2553	1.539	0.815	0.416	-1.782	4.292
8	-2691.3384	727.339	-3.700	0.000	-4126.182	-1256.495
9	176.0065	27.016	6.515	0.000	122.712	229.301
10	-6936.9992	1618.045	-4.287	0.000	-1.01e+04	-3745.031
11	-5317.5798	940.103	-5.656	0.000	-7172.149	-3463.010
12	230.5518	70.778	3.257	0.001	90.926	370.178
13	82.6098	15.276	5.408	0.000	52.474	112.746

Omnibus:	20.424	**Durbin-Watson:**	0.851
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	58.328
Skew:	0.341	**Prob(JB):**	2.16e-13
Kurtosis:	5.549	**Cond. No.**	2.03e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.03e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [ ]:

```

```