Model 1

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from statsmodels.graphics.regressionplots import influence_plot
import statsmodels.formula.api as smf
from scipy import stats
import statsmodels.api as sm


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

Import data

In [3]:

```
cars = pd.read_csv('ToyotaCorolla.csv')
cars
```

Out[3]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_(
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	10	2002	46986	Diesel	90	
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	10	2002	72937	Diesel	90	
2	3	�TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	9	2002	41711	Diesel	90	
3	4	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	14950	26	7	2002	48000	Diesel	90	
4	5	TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors	13750	30	3	2002	38500	Diesel	90	
...	...	...	...	...	...	...	...	...	...	
1431	1438	TOYOTA Corolla 1.3 16V HATCHB G6 2/3-Doors	7500	69	12	1998	20544	Petrol	86	
1432	1439	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	10845	72	9	1998	19000	Petrol	86	
1433	1440	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	8500	71	10	1998	17016	Petrol	86	

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_(
1434	1441	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	7250	70	11	1998	16916	Petrol	86	
1435	1442	TOYOTA Corolla 1.6 LB LINEA TERRA 4/5-Doors	6950	76	5	1998	1	Petrol	110	

1436 rows × 38 columns

Data Understanding

In [5]:

```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 38 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Id                1436 non-null   int64 
 1   Model             1436 non-null   object
 2   Price             1436 non-null   int64 
 3   Age_08_04         1436 non-null   int64 
 4   Mfg_Month         1436 non-null   int64 
 5   Mfg_Year          1436 non-null   int64 
 6   KM                1436 non-null   int64 
 7   Fuel_Type         1436 non-null   object
 8   HP                1436 non-null   int64 
 9   Met_Color         1436 non-null   int64 
 10  Color             1436 non-null   object
 11  Automatic         1436 non-null   int64 
 12  cc                1436 non-null   int64 
 13  Doors             1436 non-null   int64 
 14  Cylinders         1436 non-null   int64 
 15  Gears             1436 non-null   int64 
 16  Quarterly_Tax     1436 non-null   int64 
 17  Weight            1436 non-null   int64 
 18  Mfr_Guarantee     1436 non-null   int64 
 19  BOVAG_Guarantee   1436 non-null   int64 
 20  Guarantee_Period  1436 non-null   int64 
 21  ABS               1436 non-null   int64 
 22  Airbag_1          1436 non-null   int64 
 23  Airbag_2          1436 non-null   int64 
 24  Airco             1436 non-null   int64 
 25  Automatic_airco   1436 non-null   int64 
 26  Boardcomputer     1436 non-null   int64 
 27  CD_Player         1436 non-null   int64 
 28  Central_Lock      1436 non-null   int64 
 29  Powered_Windows   1436 non-null   int64 
 30  Power_Steering    1436 non-null   int64 
 31  Radio             1436 non-null   int64 
 32  Mistlamps         1436 non-null   int64 
 33  Sport_Model       1436 non-null   int64 
 34  Backseat_Divider  1436 non-null   int64 
 35  Metallic_Rim      1436 non-null   int64 
 36  Radio_cassette    1436 non-null   int64 
 37  Tow_Bar           1436 non-null   int64 
dtypes: int64(35), object(3)
memory usage: 426.4+ KB
```

In [6]:

```
cars.isna().sum()
```

Out[6]:

```
Id                  0
Model               0
Price               0
Age_08_04           0
Mfg_Month           0
Mfg_Year            0
KM                  0
Fuel_Type           0
HP                  0
Met_Color           0
Color               0
Automatic           0
cc                  0
Doors               0
Cylinders           0
Gears               0
Quarterly_Tax       0
Weight              0
Mfr_Guarantee       0
BOVAG_Guarantee     0
Guarantee_Period    0
ABS                 0
Airbag_1            0
Airbag_2            0
Airco               0
Automatic_airco     0
Boardcomputer       0
CD_Player           0
Central_Lock        0
Powered_Windows     0
Power_Steering      0
Radio               0
Mistlamps           0
Sport_Model         0
Backseat_Divider    0
Metallic_Rim        0
Radio_cassette      0
Tow_Bar             0
dtype: int64
```

In [8]:

```
cars= pd.concat([cars.iloc[:,2:4],cars.iloc[:,6:7],cars.iloc[:,8:9],cars.iloc[:,12:14],cars
cars
```

Out[8]:

	Price	Age_08_04	KM	HP	cc	Doors	Gears	Quarterly_Tax	Weight
0	13500	23	46986	90	2000	3	5	210	1165
1	13750	23	72937	90	2000	3	5	210	1165
2	13950	24	41711	90	2000	3	5	210	1165
3	14950	26	48000	90	2000	3	5	210	1165
4	13750	30	38500	90	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1431	7500	69	20544	86	1300	3	5	69	1025
1432	10845	72	19000	86	1300	3	5	69	1015
1433	8500	71	17016	86	1300	3	5	69	1015
1434	7250	70	16916	86	1300	3	5	69	1015
1435	6950	76	1	110	1600	5	5	19	1114

1436 rows × 9 columns

In [9]:

```
cars= cars.rename({'Age_08_04':'Age','cc':'CC','Quarterly_Tax':'QT'},axis=1)
cars
```

Out[9]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90	2000	3	5	210	1165
1	13750	23	72937	90	2000	3	5	210	1165
2	13950	24	41711	90	2000	3	5	210	1165
3	14950	26	48000	90	2000	3	5	210	1165
4	13750	30	38500	90	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1431	7500	69	20544	86	1300	3	5	69	1025
1432	10845	72	19000	86	1300	3	5	69	1015
1433	8500	71	17016	86	1300	3	5	69	1015
1434	7250	70	16916	86	1300	3	5	69	1015
1435	6950	76	1	110	1600	5	5	19	1114

1436 rows × 9 columns

In [10]:

```
cars[cars.duplicated()]
```

Out[10]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
113	24950	8	13253	116	2000	5	5	234	1320

In [11]:

```
cars = cars.drop_duplicates().reset_index(drop=True)
cars
```

Out[11]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90	2000	3	5	210	1165
1	13750	23	72937	90	2000	3	5	210	1165
2	13950	24	41711	90	2000	3	5	210	1165
3	14950	26	48000	90	2000	3	5	210	1165
4	13750	30	38500	90	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1430	7500	69	20544	86	1300	3	5	69	1025
1431	10845	72	19000	86	1300	3	5	69	1015
1432	8500	71	17016	86	1300	3	5	69	1015
1433	7250	70	16916	86	1300	3	5	69	1015
1434	6950	76	1	110	1600	5	5	19	1114

1435 rows × 9 columns

In [12]:

```
cars.shape
```

Out[12]:

```
(1435, 9)
```

In [13]:

```
cars.dtypes
```

Out[13]:

```
Price     int64
Age       int64
KM        int64
HP        int64
CC        int64
Doors     int64
Gears     int64
QT        int64
Weight    int64
dtype: object
```

In [14]:

```
cars.describe
```

Out[14]:

```
<bound method NDFrame.describe of       Price  Age     KM   HP    CC  Doors  
Gears   QT  Weight
0     13500   23  46986   90  2000      3      5  210    1165
1     13750   23  72937   90  2000      3      5  210    1165
2     13950   24  41711   90  2000      3      5  210    1165
3     14950   26  48000   90  2000      3      5  210    1165
4     13750   30  38500   90  2000      3      5  210    1170
...     ...  ...    ...  ...   ...    ...    ...  ...     ...
1430   7500   69  20544   86  1300      3      5   69    1025
1431  10845   72  19000   86  1300      3      5   69    1015
1432   8500   71  17016   86  1300      3      5   69    1015
1433   7250   70  16916   86  1300      3      5   69    1015
1434   6950   76      1  110  1600      5      5   19    1114

[1435 rows x 9 columns]>
```

In [15]:

```
cars.max
```

Out[15]:

```
<bound method NDFrame._add_numeric_operations.<locals>.max of       Price  A
ge     KM   HP    CC  Doors  Gears   QT  Weight
0     13500   23  46986   90  2000      3      5  210    1165
1     13750   23  72937   90  2000      3      5  210    1165
2     13950   24  41711   90  2000      3      5  210    1165
3     14950   26  48000   90  2000      3      5  210    1165
4     13750   30  38500   90  2000      3      5  210    1170
...     ...  ...    ...  ...   ...    ...    ...  ...     ...
1430   7500   69  20544   86  1300      3      5   69    1025
1431  10845   72  19000   86  1300      3      5   69    1015
1432   8500   71  17016   86  1300      3      5   69    1015
1433   7250   70  16916   86  1300      3      5   69    1015
1434   6950   76      1  110  1600      5      5   19    1114

[1435 rows x 9 columns]>
```

In [16]:

```
cars.min
```

Out[16]:

```
<bound method NDFrame._add_numeric_operations.<locals>.min of       Price  A
ge     KM   HP    CC  Doors  Gears   QT  Weight
0     13500   23  46986    90  2000      3      5  210    1165
1     13750   23  72937    90  2000      3      5  210    1165
2     13950   24  41711    90  2000      3      5  210    1165
3     14950   26  48000    90  2000      3      5  210    1165
4     13750   30  38500    90  2000      3      5  210    1170
...     ...  ...    ...   ...   ...    ...    ...  ...     ...
1430   7500   69  20544    86  1300      3      5   69    1025
1431  10845   72  19000    86  1300      3      5   69    1015
1432   8500   71  17016    86  1300      3      5   69    1015
1433   7250   70  16916    86  1300      3      5   69    1015
1434   6950   76      1   110  1600      5      5   19    1114

[1435 rows x 9 columns]>
```

Skewness and kurtosis || Normality test (using Distplot)

In [24]:

```
cars['Price'].skew()
```

Out[24]:

```
1.6965785809803777
```

In [22]:

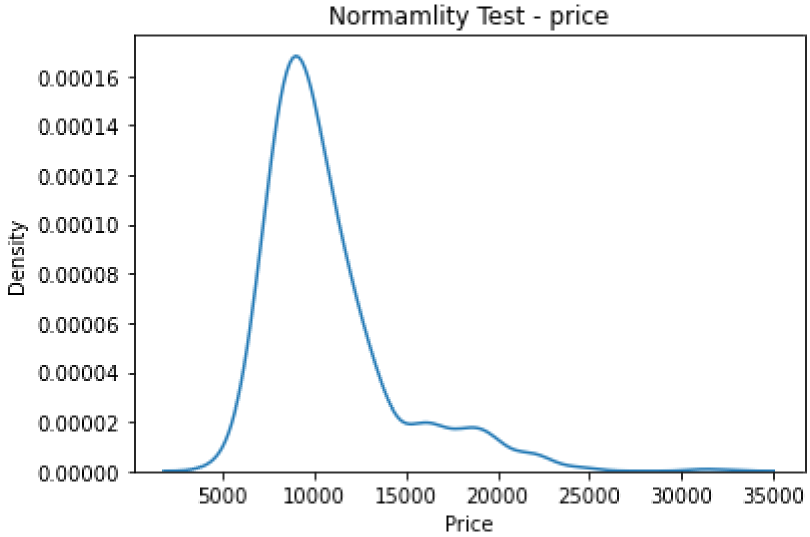
```
cars['Price'].kurtosis()
```

Out[22]:

```
3.7297685345419405
```

In [25]:

```
sns.distplot(a=cars['Price'],hist=False)
plt.title('Normamlity Test - price')
plt.show()
```



In [26]:

```
cars['Age'].skew()
```

Out[26]:

-0.8255666018465969

In [27]:

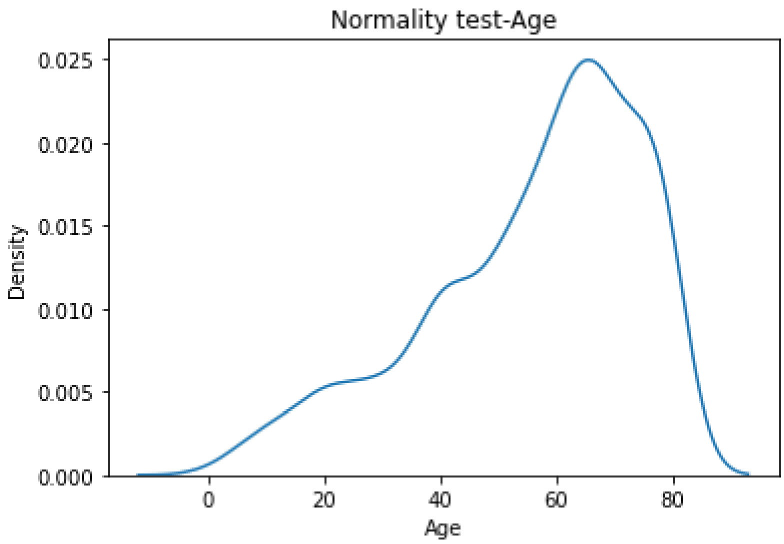
```
cars['Age'].kurtosis()
```

Out[27]:

-0.076573436732565

In [28]:

```
sns.distplot(a=cars['Age'],hist=False)
plt.title('Normality test-Age')
plt.show()
```



In [29]:

```
cars['KM'].skew()
```

Out[29]:

1.0170229723462332

In [30]:

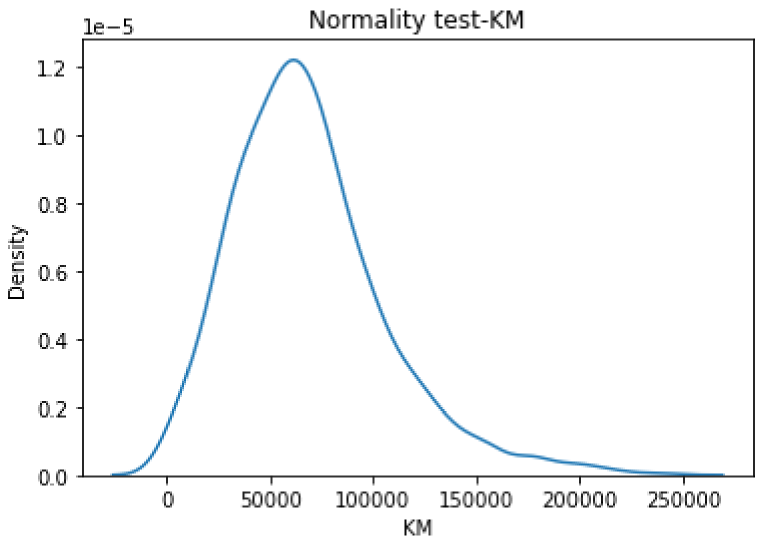
```
cars['KM'].kurtosis()
```

Out[30]:

1.6885247633485445

In [31]:

```
sns.distplot(cars['KM'],hist=False)
plt.title('Normality test-KM')
plt.show()
```



In [32]:

```
cars['HP'].skew()
```

Out[32]:

0.9578333639343268

In [33]:

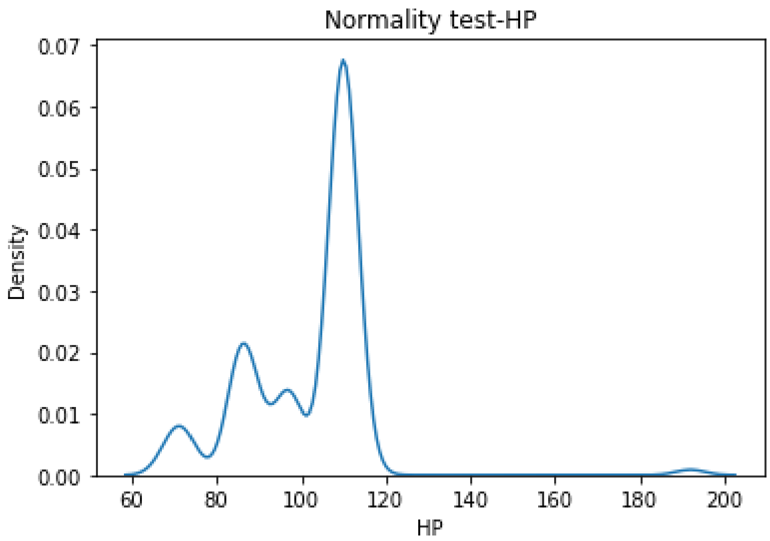
```
cars['HP'].kurtosis()
```

Out[33]:

8.845658522324406

In [34]:

```
sns.distplot(a=cars['HP'],hist=False)
plt.title('Normality test-HP')
plt.show()
```



In [35]:

```
cars['CC'].skew()
```

Out[35]:

27.45219619846663

In [36]:

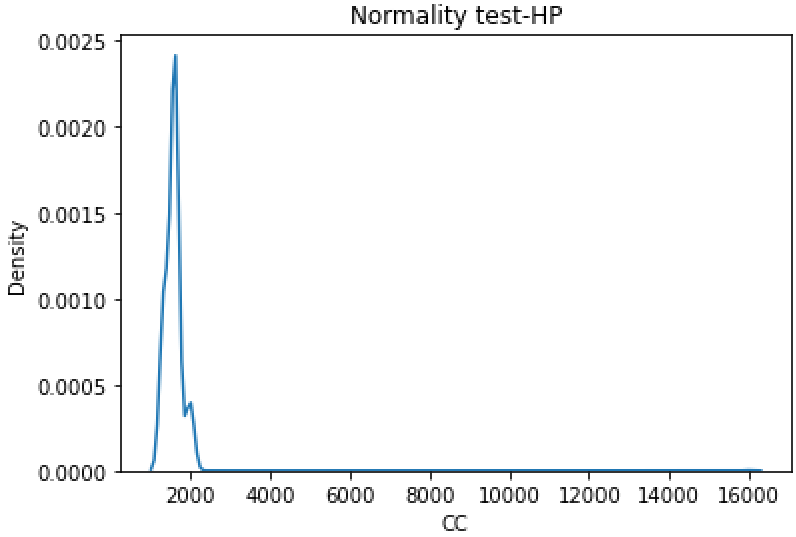
```
cars['CC'].kurtosis()
```

Out[36]:

931.4341727829076

In [37]:

```
sns.distplot(a=cars['CC'],hist=False)
plt.title('Normality test-CC')
plt.show()
```



In [38]:

```
cars['Doors'].skew()
```

Out[38]:

```
-0.07505603155165053
```

In [39]:

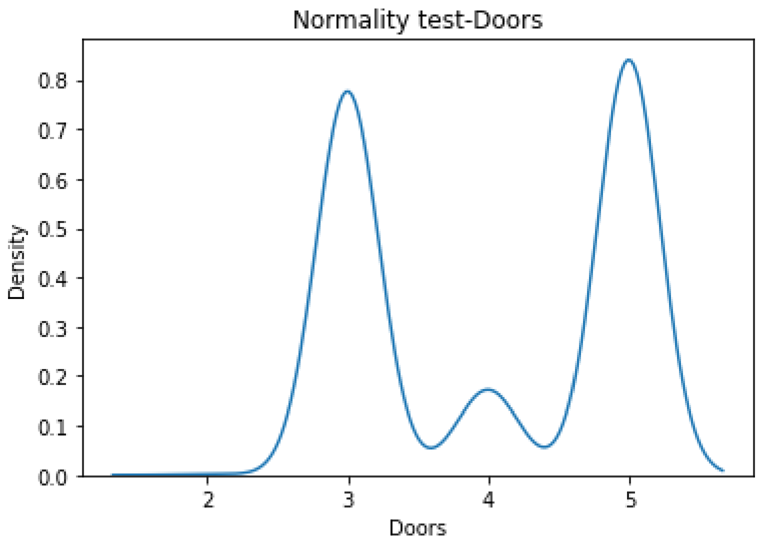
```
cars['Doors'].kurtosis()
```

Out[39]:

```
-1.8748873416868563
```

In [41]:

```
sns.distplot(a=cars['Doors'],hist=False)
plt.title('Normality test-Doors')
plt.show()
```



In [42]:

```
cars['Gears'].skew()
```

Out[42]:

2.282921227095275

In [43]:

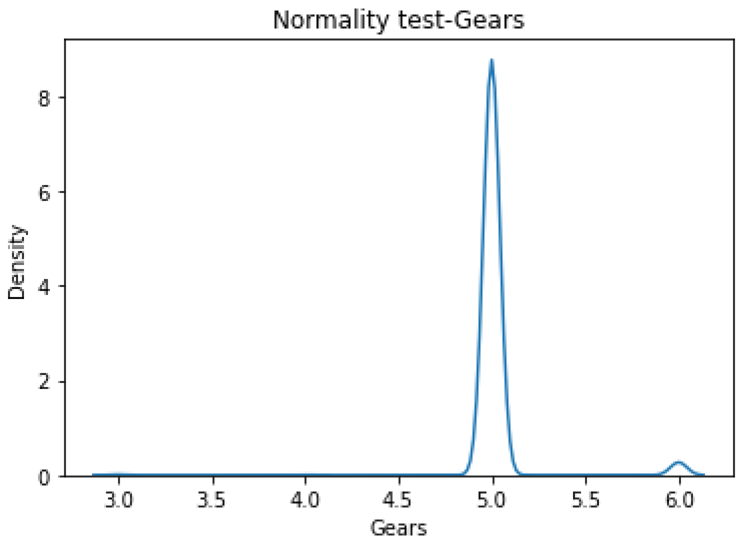
```
cars['Gears'].kurtosis()
```

Out[43]:

37.67544347486965

In [44]:

```
sns.distplot(a=cars['Gears'],hist=False)
plt.title('Normality test-Gears')
plt.show()
```



In [45]:

```
cars['Weight'].skew()
```

Out[45]:

3.1165183382777437

In [46]:

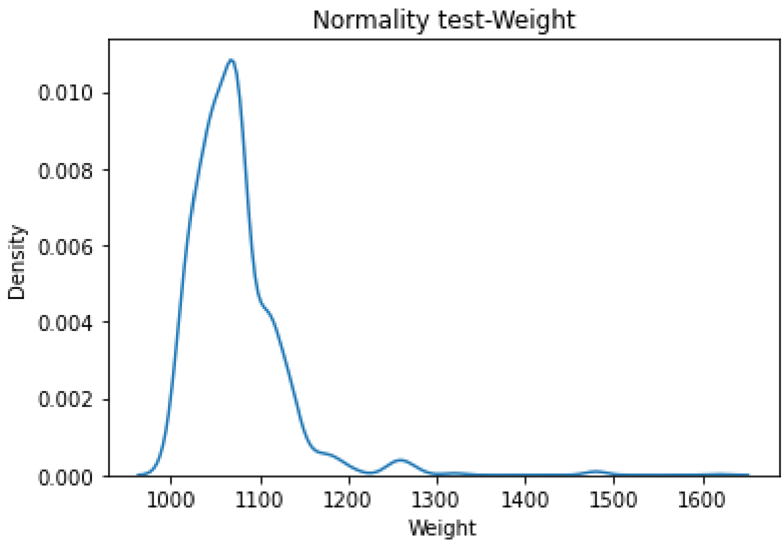
```
cars['Weight'].kurtosis()
```

Out[46]:

19.741526687585104

In [47]:

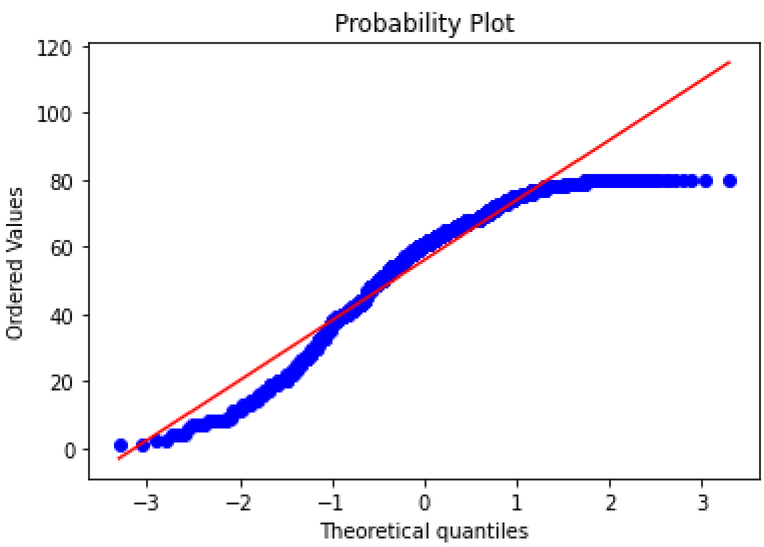
```
sns.distplot(a=cars['Weight'],hist=False)
plt.title('Normality test-Weight')
plt.show()
```



Normality test using probplot

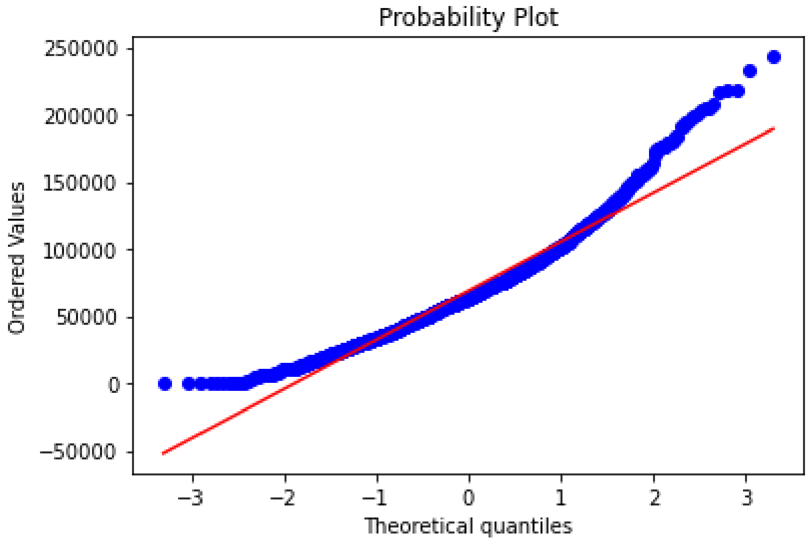
In [49]:

```
stats.probplot(x=cars['Age'],dist='norm',plot=plt)
plt.show()
```



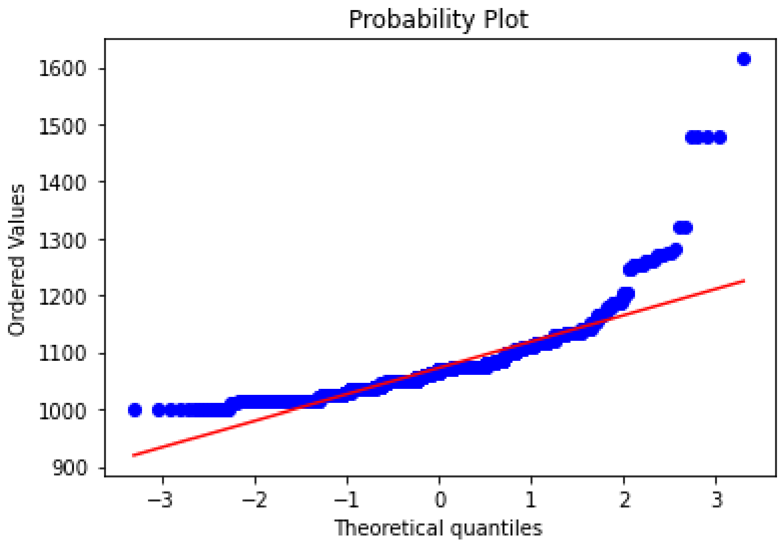
In [50]:

```
stats.probplot(x=cars['KM'],dist='norm',plot=plt)
plt.show()
```



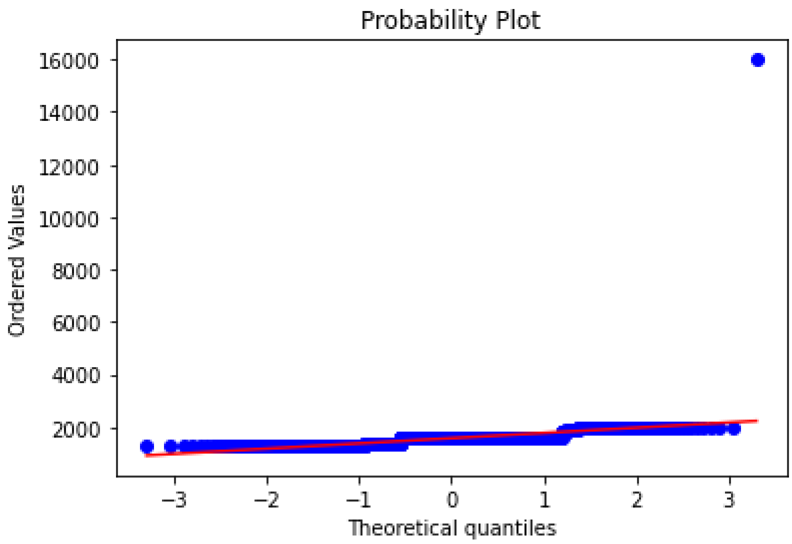
In [52]:

```
stats.probplot(x=cars['Weight'],dist='norm',plot=plt)
plt.show()
```



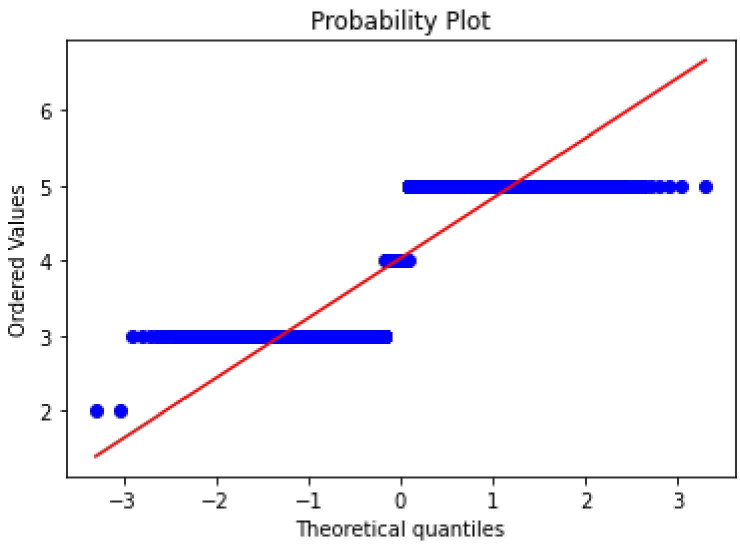
In [53]:

```
stats.probplot(x=cars['CC'],dist='norm',plot=plt)
plt.show()
```



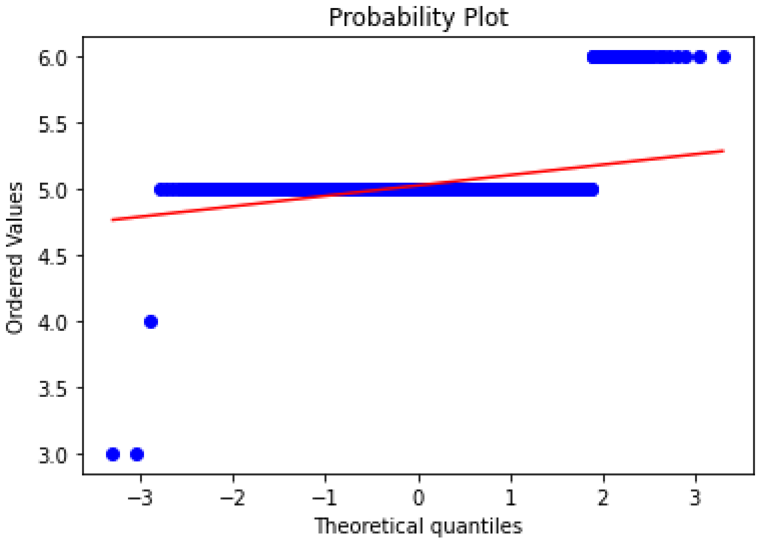
In [54]:

```
stats.probplot(x=cars['Doors'],dist='norm',plot=plt)
plt.show()
```



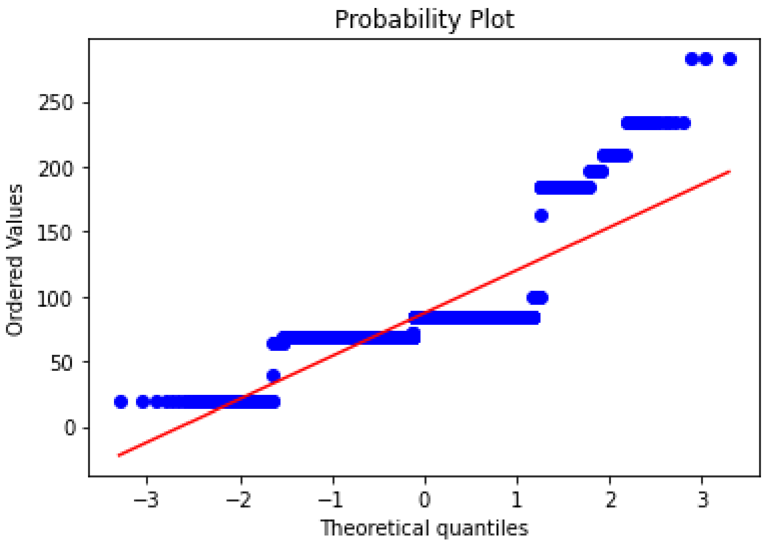
In [55]:

```
stats.probplot(x=cars['Gears'],dist='norm',plot=plt)
plt.show()
```



In [56]:

```
stats.probplot(x=cars['QT'],dist='norm',plot=plt)
plt.show()
```



Correlation

In [57]:

```
cars_corr = cars.corr().round(2)
cars_corr
```

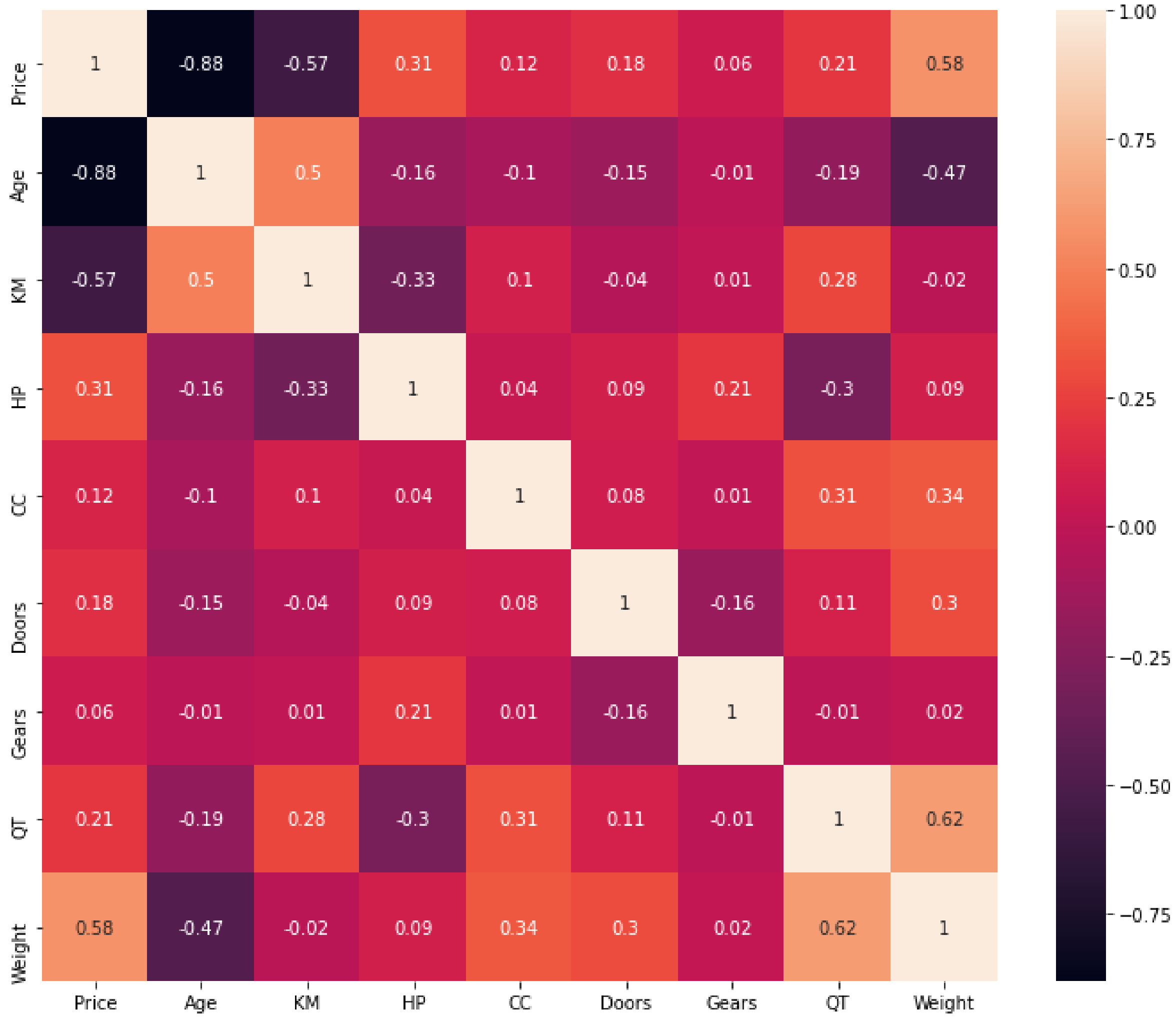
Out[57]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
Price	1.00	-0.88	-0.57	0.31	0.12	0.18	0.06	0.21	0.58
Age	-0.88	1.00	0.50	-0.16	-0.10	-0.15	-0.01	-0.19	-0.47
KM	-0.57	0.50	1.00	-0.33	0.10	-0.04	0.01	0.28	-0.02
HP	0.31	-0.16	-0.33	1.00	0.04	0.09	0.21	-0.30	0.09
CC	0.12	-0.10	0.10	0.04	1.00	0.08	0.01	0.31	0.34
Doors	0.18	-0.15	-0.04	0.09	0.08	1.00	-0.16	0.11	0.30
Gears	0.06	-0.01	0.01	0.21	0.01	-0.16	1.00	-0.01	0.02
QT	0.21	-0.19	0.28	-0.30	0.31	0.11	-0.01	1.00	0.62
Weight	0.58	-0.47	-0.02	0.09	0.34	0.30	0.02	0.62	1.00

Heatmap using correlation data

In [58]:

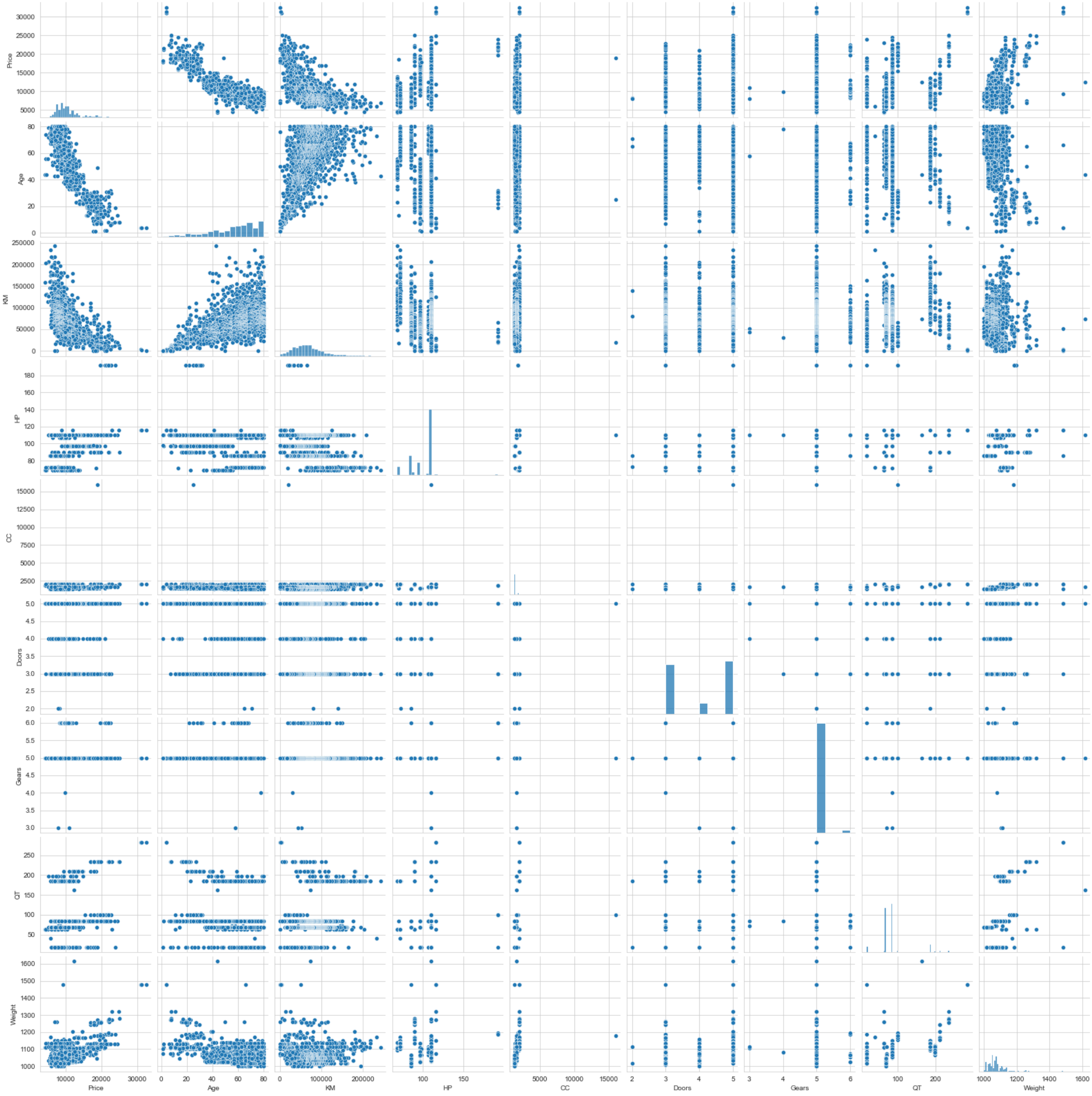
```
plt.figure(figsize=(12,10))
sns.heatmap(cars_corr,annot=True)
plt.show()
```



Scatterplot between variables alongs with hiistogram

In [59]:

```
# format the plot background  and scatter plots for all variables
sns.set_style('whitegrid')
sns.pairplot(cars)
plt.show()
```



Let's create a Referance data to understand how x features should behave with y.

In [60]:

```
cars.shape
```

Out[60]:

```
(1435, 9)
```

In [61]:

```
X = np.random.randn(81)
y = 10*X + np.random.randn(81)*2
```

In [62]:

```
X_df = pd.DataFrame(data=[X,y]).T
X_df.columns = ['X','y']
X_df
```

Out[62]:

	X	y
0	0.601813	5.907716
1	-0.096511	0.160175
2	-0.259820	1.623354
3	-1.139655	-11.219657
4	-0.892138	-7.630030
...	...	...
76	0.004091	4.202127
77	0.583285	6.926263
78	-1.182763	-12.001279
79	0.152666	1.614262
80	0.218696	2.612352

81 rows × 2 columns

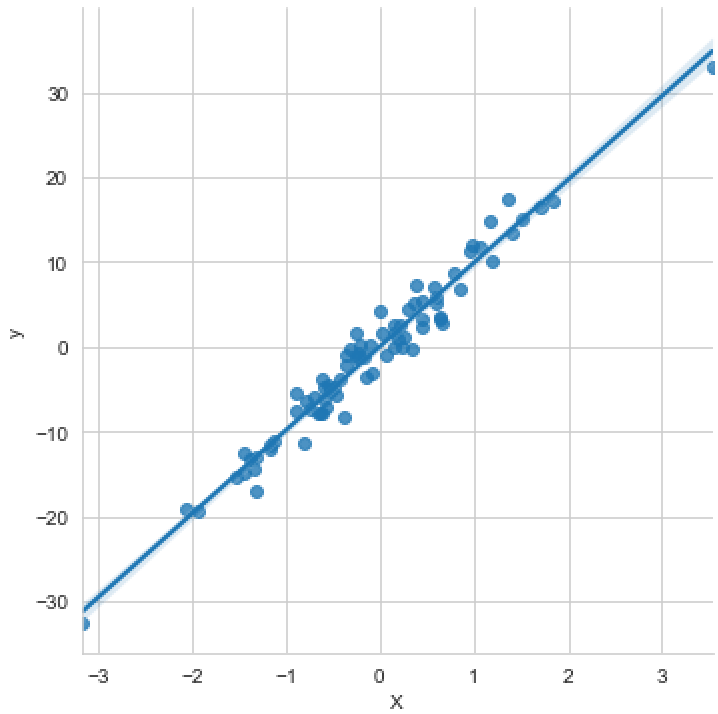
1. Linearity Test

In [63]:

```
sns.lmplot(x='X',y='y',data=X_df)
```

Out[63]:

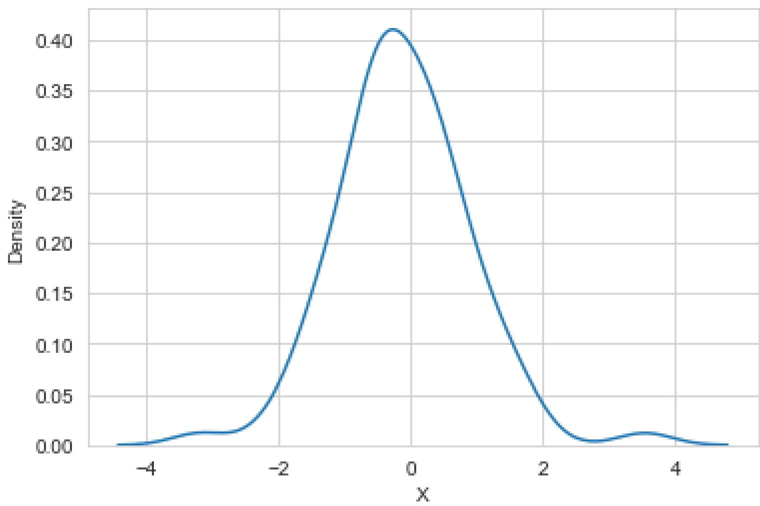
```
<seaborn.axisgrid.FacetGrid at 0x1713ab6fe20>
```



2. Noramality Test

In [64]:

```
sns.distplot(a = X_df['X'],hist=False)
plt.show()
```



In [65]:

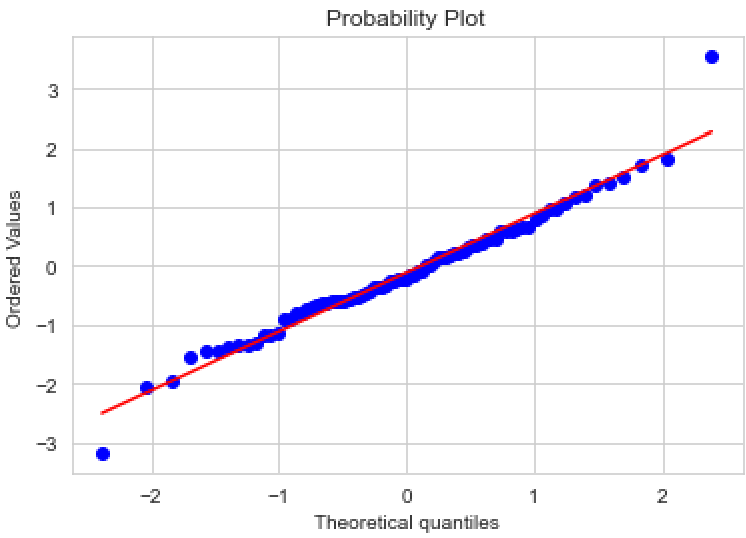
```
X_df.skew()
```

Out[65]:

```
X    0.302946
y    0.165813
dtype: float64
```

In [66]:

```
stats.probplot(x = X_df['X'],dist='norm',plot=plt)
plt.show()
```



Model Building

In [67]:

```
x = X_df[['X']]
y = X_df[['y']]
```

sklearn Model Building

In [69]:

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression() #object creation/ model initialization
linear_model.fit(x,y)
```

Out[69]:

```
LinearRegression()
```

In [70]:

```
linear_model.intercept_
```

Out[70]:

```
array([0.03028772])
```

In [71]:

```
linear_model.coef_
```

Out[71]:

```
array([[9.82889751]])
```

Model Testing

In [72]:

```
y_pred = linear_model.predict(x)
```

Model Evalution

In [73]:

```
y
```

Out[73]:

	y
0	5.907716
1	0.160175
2	1.623354
3	-11.219657
4	-7.630030
...	...
76	4.202127
77	6.926263
78	-12.001279
79	1.614262
80	2.612352

81 rows × 1 columns

In [74]:

```
y_pred
```

Out[74]:

```
array([[  5.94544899],
       [ -0.91831117],
       [ -2.52345475],
       [-11.17126058],
       [ -8.73844709],
       [-20.22869582],
       [ -1.39569544],
       [ -6.92749955],
       [ -2.30113954],
       [  3.77771079],
       [ -0.81183966],
       [-14.21694704],
       [ 10.58516067],
       [ -7.21083072],
       [ -8.78033632],
       [ -4.55037354],
       [ -5.13723528],
       [ -7.72880814],
       [-12.98995183],
       [  9.63118171],
       [ 11.89158797],
       [ -5.97010887],
       [ -3.14348342],
       [-13.195718  ],
       [  2.65341961],
       [  4.44111032],
       [ -2.19167123],
       [ -5.71263474],
       [  4.40327598],
       [ -6.51698593],
       [-31.17217592],
       [  6.02423908],
       [ -6.12705275],
       [  2.32832951],
       [ -4.20995603],
       [  1.95620909],
       [ -3.60588212],
       [ -4.83503681],
       [  7.83105452],
       [ -3.64146767],
       [  6.44844693],
       [-14.19695116],
       [  3.58761155],
       [  3.02453013],
       [  6.59910635],
       [-11.47050575],
       [  8.48934031],
       [  0.73359534],
       [  0.2265824 ],
       [ 14.98814785],
       [  1.55079862],
       [-15.12169875],
       [ 18.00744439],
       [ -5.46559552],
       [  9.54893994],
```

```
       [-19.14338864],
       [ -6.23021321],
       [ 13.42794582],
       [ -5.1140456 ],
       [ -3.42722676],
       [ -7.89310225],
       [ 34.82755202],
       [ -5.77910744],
       [ 16.91152962],
       [ 14.01567302],
       [-13.01908541],
       [  3.40889692],
       [-13.57877979],
       [ -1.61159401],
       [  1.43719986],
       [  4.36163612],
       [ -2.03392216],
       [ -2.44042108],
       [  6.26129242],
       [ -5.90161508],
       [ 11.52041415],
       [  0.07049662],
       [  5.76333218],
       [-11.59497159],
       [  1.53082229],
       [  2.17983308]])
```

In [75]:

```
err = y-y_pred
err
```

Out[75]:

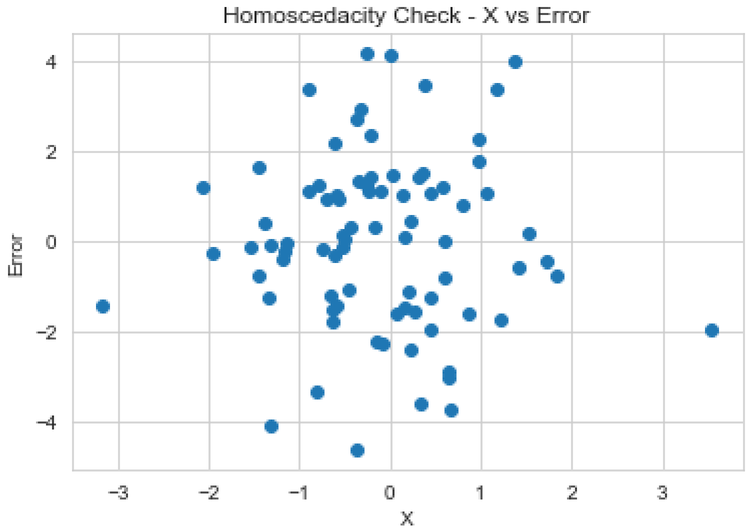
	y
0	-0.037733
1	1.078487
2	4.146809
3	-0.048396
4	1.108417
...	...
76	4.131630
77	1.162931
78	-0.406307
79	0.083440
80	0.432519

81 rows × 1 columns

5.Homoscedasticity Check

In [76]:

```
plt.scatter(x= X_df['X'],y=err)
plt.title('Homoscedacity Check - X vs Error')
plt.xlabel('X')
plt.ylabel('Error')
plt.show()
```

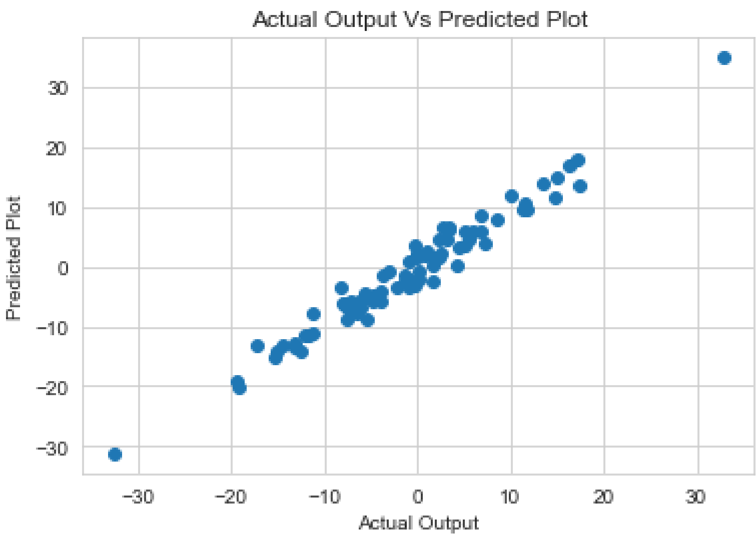


6. Zero Residual Mean Test

It is a plot between Actual Output Vs Predicted Plot.

In [77]:

```
plt.scatter(x= y,y=y_pred)
plt.title('Actual Output Vs Predicted Plot')
plt.xlabel('Actual Output')
plt.ylabel('Predicted Plot')
plt.show()
```



BACK to CARS DATA

In [78]:

```
cars
```

Out[78]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90	2000	3	5	210	1165
1	13750	23	72937	90	2000	3	5	210	1165
2	13950	24	41711	90	2000	3	5	210	1165
3	14950	26	48000	90	2000	3	5	210	1165
4	13750	30	38500	90	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1430	7500	69	20544	86	1300	3	5	69	1025
1431	10845	72	19000	86	1300	3	5	69	1015
1432	8500	71	17016	86	1300	3	5	69	1015
1433	7250	70	16916	86	1300	3	5	69	1015
1434	6950	76	1	110	1600	5	5	19	1114

1435 rows × 9 columns

Model Building

In [79]:

```
X = cars.drop(['Price'],axis= 1)
y = cars[['Price']]
```

In [80]:

```
x.columns
```

Out[80]:

```
Index(['X'], dtype='object')
```

In [81]:

```
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
X_scaled = std_scaler.fit_transform(X)
X_scaled = pd.DataFrame(data=X_scaled,columns= X.columns)
X_scaled
```

Out[81]:

	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	-1.777268	-0.575958	-0.767351	0.998113	-1.084443	-0.140475	3.003513	1.774964
1	-1.777268	0.116474	-0.767351	0.998113	-1.084443	-0.140475	3.003513	1.774964
2	-1.723380	-0.716707	-0.767351	0.998113	-1.084443	-0.140475	3.003513	1.774964
3	-1.615603	-0.548902	-0.767351	0.998113	-1.084443	-0.140475	3.003513	1.774964
4	-1.400049	-0.802384	-0.767351	0.998113	-1.084443	-0.140475	3.003513	1.870688
...	...	...	...	...	...	...	...	...
1430	0.701602	-1.281492	-1.034441	-0.651898	-1.084443	-0.140475	-0.440104	-0.905299
1431	0.863267	-1.322689	-1.034441	-0.651898	-1.084443	-0.140475	-0.440104	-1.096747
1432	0.809379	-1.375627	-1.034441	-0.651898	-1.084443	-0.140475	-0.440104	-1.096747
1433	0.755490	-1.378295	-1.034441	-0.651898	-1.084443	-0.140475	-0.440104	-1.096747
1434	1.078821	-1.829626	0.568103	0.055249	1.015659	-0.140475	-1.661245	0.798582

1435 rows × 8 columns

Before Scaling

In [82]:

```
X.mean()
```

Out[82]:

```
Age          55.980488
KM        68571.782578
HP          101.491986
CC         1576.560976
Doors         4.032753
Gears         5.026481
QT           87.020209
Weight     1072.287108
dtype: float64
```

In [83]:

```
X.std()
```

Out[83]:

```
Age           18.563312
KM         37491.094553
HP            14.981408
CC           424.387533
Doors          0.952667
Gears          0.188575
QT            40.959588
Weight        52.251882
dtype: float64
```

After Scaling

In [84]:

```
X_scaled.mean()
```

Out[84]:

```
Age      -1.668042e-16
KM        2.503611e-16
HP        1.166856e-15
CC       -1.581923e-15
Doors    -1.241902e-15
Gears    -2.065363e-15
QT       -1.220085e-15
Weight    1.496287e-16
dtype: float64
```

In [85]:

```
X_scaled.std()
```

Out[85]:

```
Age       1.000349
KM        1.000349
HP        1.000349
CC        1.000349
Doors     1.000349
Gears     1.000349
QT        1.000349
Weight    1.000349
dtype: float64
```

Model Training

In [86]:

```
linear_model_02 = LinearRegression()
linear_model_02.fit(X_scaled,y)
```

Out[86]:

```
LinearRegression()
```

In [87]:

```
linear_model_02.coef_
```

Out[87]:

```
array([[-2.25862554e+03, -7.77200045e+02,  4.73017054e+02,
        -5.02968400e+01, -8.76327907e-01,  1.12674914e+02,
         1.57999990e+02,  8.80423097e+02]])
```

In [88]:

```
linear_model_02.intercept_
```

Out[88]:

```
array([10720.91567944])
```

Model Testing

In [89]:

```
y_pred = linear_model_02.predict(X_scaled)
```

Model Evalution

In [90]:

```
err = y - y_pred
err
```

Out[90]:

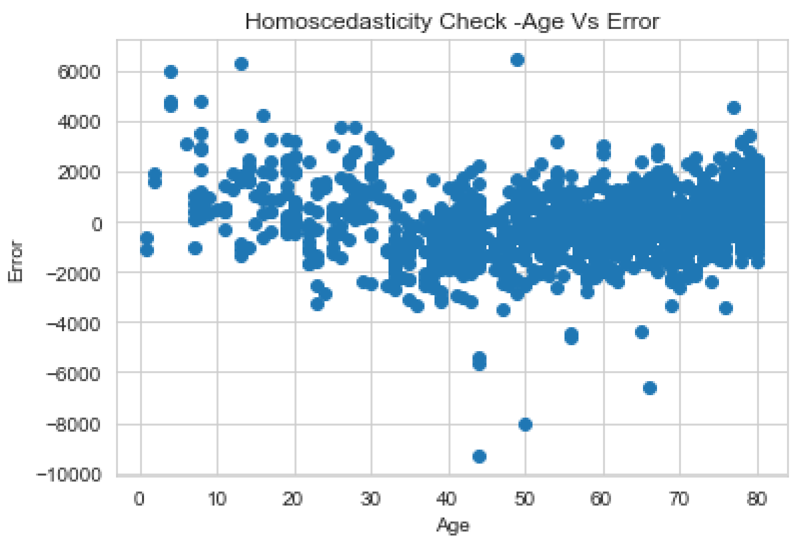
	Price
0	-3291.958871
1	-2503.800414
2	-2829.635210
3	-1455.789389
4	-2450.217277
...	...
1430	-1294.255037
1431	2552.422658
1432	44.565598
1433	-1329.222041
1434	-3446.087526

1435 rows × 1 columns

Homoscedascity Check

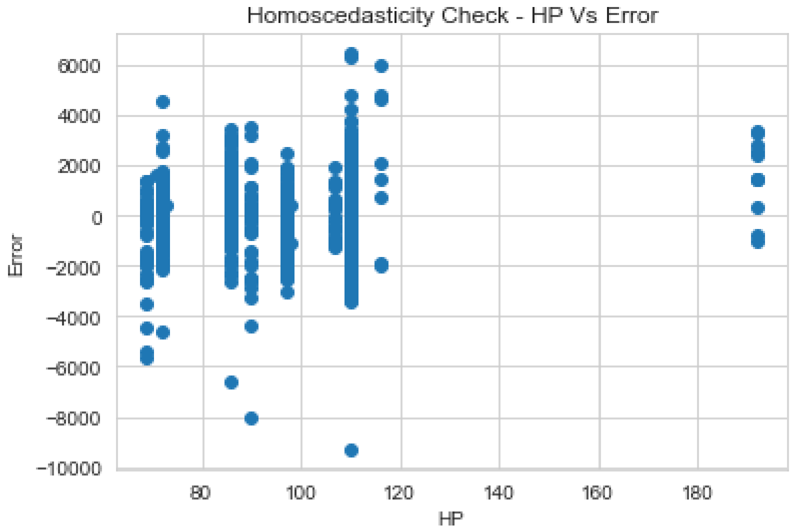
In [91]:

```
plt.scatter(x = cars['Age'],y = err)
plt.title('Homoscedasticity Check -Age Vs Error')
plt.xlabel('Age')
plt.ylabel('Error')
plt.show()
```



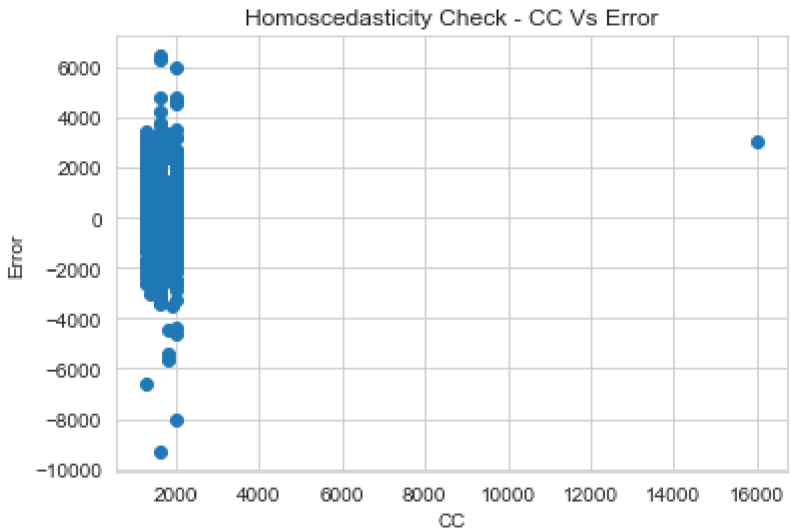
In [92]:

```
plt.scatter(x = cars['HP'],y = err)
plt.title('Homoscedasticity Check - HP Vs Error')
plt.xlabel('HP')
plt.ylabel('Error')
plt.show()
```



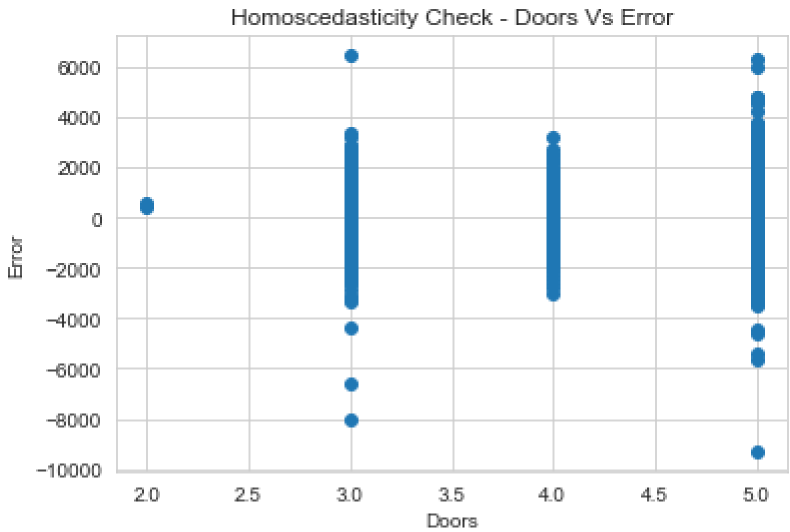
In [93]:

```
plt.scatter(x = cars['CC'],y = err)
plt.title('Homoscedasticity Check - CC Vs Error')
plt.xlabel('CC')
plt.ylabel('Error')
plt.show()
```



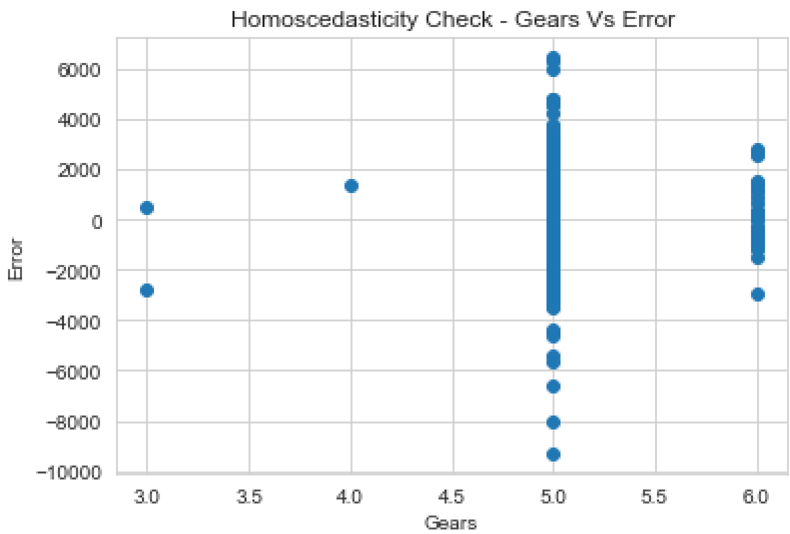
In [94]:

```
plt.scatter(x = cars['Doors'],y = err)
plt.title('Homoscedasticity Check - Doors Vs Error')
plt.xlabel('Doors')
plt.ylabel('Error')
plt.show()
```



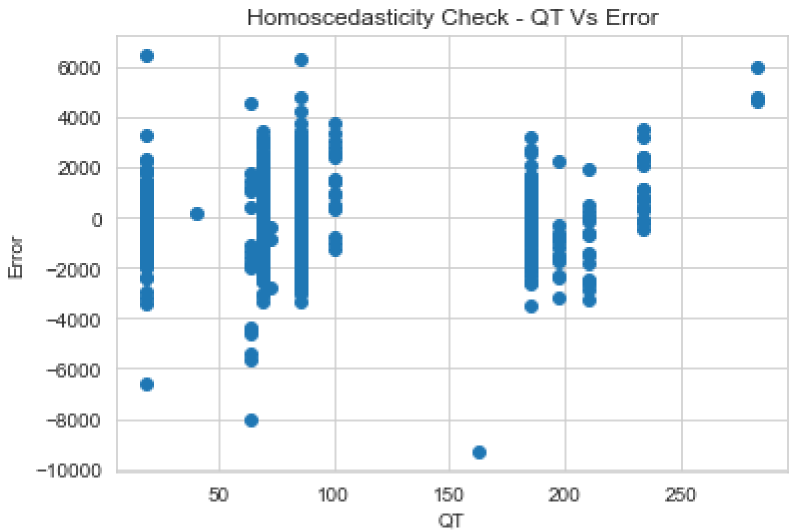
In [95]:

```
plt.scatter(x = cars['Gears'],y = err)
plt.title('Homoscedasticity Check - Gears Vs Error')
plt.xlabel('Gears')
plt.ylabel('Error')
plt.show()
```



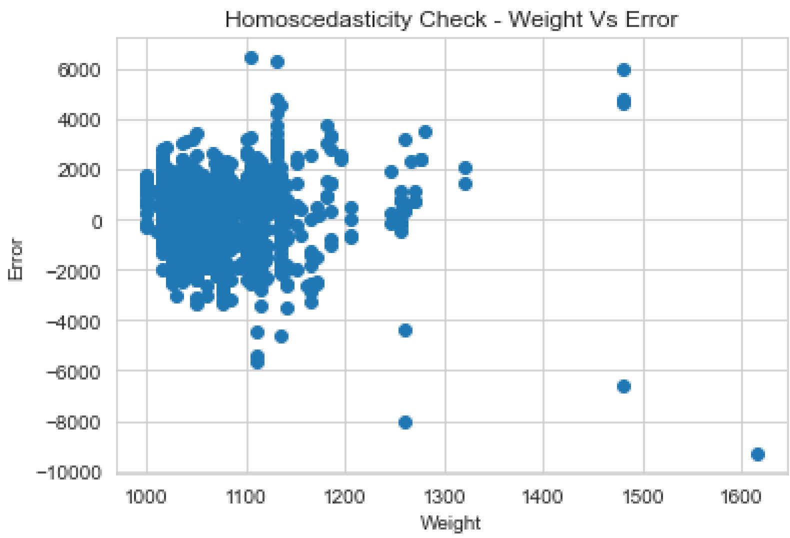
In [96]:

```
plt.scatter(x = cars['QT'],y = err)
plt.title('Homoscedasticity Check - QT Vs Error')
plt.xlabel('QT')
plt.ylabel('Error')
plt.show()
```



In [97]:

```
plt.scatter(x = cars['Weight'],y = err)
plt.title('Homoscedasticity Check - Weight Vs Error')
plt.xlabel('Weight')
plt.ylabel('Error')
plt.show()
```



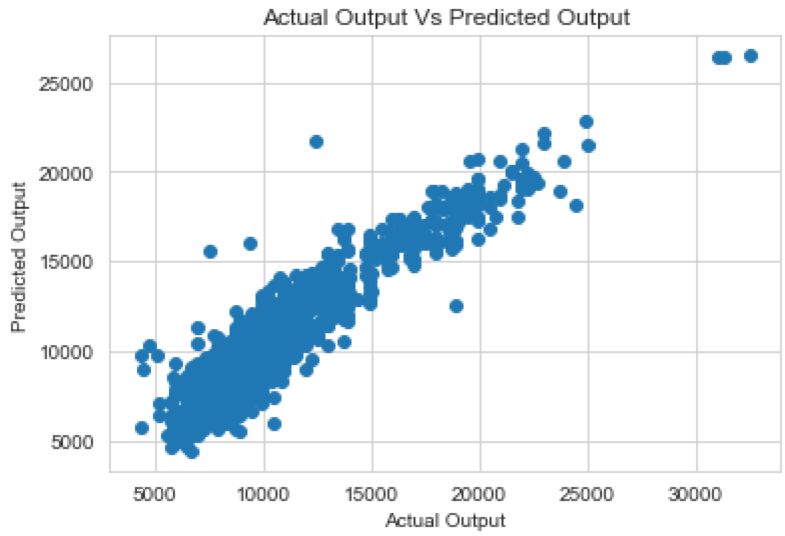
Homoscedascity Test Failed

6. Zero Residual Mean Test

It is a plot between Actual Output Vs Predicted Plot.

In [98]:

```
plt.scatter(x = y,y = y_pred)
plt.title('Actual Output Vs Predicted Output')
plt.xlabel('Actual Output')
plt.ylabel('Predicted Output')
plt.show()
```



Model Building || using statsmodel

In [100]:

```
model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=cars).fit()
```

Model Testing

In [101]:

```
model.params
```

Out[101]:

```
Intercept   -5472.540368
Age          -121.713891
KM             -0.020737
HP             31.584612
CC             -0.118558
Doors          -0.920189
Gears         597.715894
QT              3.858805
Weight         16.855470
dtype: float64
```

In [102]:

```
#Finding tvalues and pvalues of model
model.pvalues,'\n', model.tvalues
```

Out[102]:

```
(Intercept     1.113392e-04
 Age          1.879217e-288
 KM            1.994713e-56
 HP            5.211155e-28
 CC            1.882393e-01
 Doors         9.816443e-01
 Gears         2.452430e-03
 QT            3.290363e-03
 Weight        1.031118e-51
 dtype: float64,
 '\n',
 Intercept    -3.875273
 Age         -46.551876
 KM          -16.552424
 HP           11.209719
 CC           -1.316436
 Doors        -0.023012
 Gears         3.034563
 QT            2.944198
 Weight       15.760663
 dtype: float64)
```

In [103]:

```
#R squarred values
model.rsquared,model.rsquared_adj
```

Out[103]:

```
(0.8625200256947, 0.8617487495415146)
```

Simple linear regression model

In [104]:

```
model_1 = smf.ols('Price~CC',data=cars).fit()
print(model_1.tvalues,'\n',model_1.pvalues)
model_1.summary()
```

```
Intercept    24.879592
CC            4.745039
dtype: float64
 Intercept    7.236022e-114
CC            2.292856e-06
dtype: float64
```

Out[104]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.015
Model:	OLS	**Adj. R-squared:**	0.015
Method:	Least Squares	**F-statistic:**	22.52
Date:	Wed, 23 Feb 2022	**Prob (F-statistic):**	2.29e-06
Time:	02:50:14	**Log-Likelihood:**	-13779.
No. Observations:	1435	**AIC:**	2.756e+04
Df Residuals:	1433	**BIC:**	2.757e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	9053.5368	363.894	24.880	0.000	8339.715	9767.359
CC	1.0576	0.223	4.745	0.000	0.620	1.495

Omnibus:	463.846	**Durbin-Watson:**	0.269
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1386.822
Skew:	1.645	**Prob(JB):**	7.17e-302
Kurtosis:	6.518	**Cond. No.**	6.28e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.28e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [105]:

```
model_2 =smf.ols('Price~Doors',data=cars).fit()
print(model_2.tvalues,'\n',model_2.pvalues)
model_2.summary()
```

```
Intercept    19.421546
Doors         7.070520
dtype: float64
 Intercept    8.976407e-75
Doors        2.404166e-12
dtype: float64
```

Out[105]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.034
Model:	OLS	**Adj. R-squared:**	0.033
Method:	Least Squares	**F-statistic:**	49.99
Date:	Wed, 23 Feb 2022	**Prob (F-statistic):**	2.40e-12
Time:	02:50:49	**Log-Likelihood:**	-13765.
No. Observations:	1435	**AIC:**	2.753e+04
Df Residuals:	1433	**BIC:**	2.755e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	7916.1452	407.596	19.422	0.000	7116.596	8715.694
Doors	695.4978	98.366	7.071	0.000	502.541	888.454

Omnibus:	465.543	**Durbin-Watson:**	0.289
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1403.980
Skew:	1.647	**Prob(JB):**	1.35e-305
Kurtosis:	6.554	**Cond. No.**	19.0

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [106]:

```
model_3 = smf.ols('Price~CC+Doors',data=cars).fit()
print(model_3.tvalues,'\n',model_3.pvalues)
model_3.summary()
```

```
Intercept    12.786341
CC            4.268006
Doors         6.752236
dtype: float64
 Intercept    1.580945e-35
CC           2.101878e-05
Doors        2.109558e-11
dtype: float64
```

Out[106]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.046
Model:	OLS	**Adj. R-squared:**	0.045
Method:	Least Squares	**F-statistic:**	34.40
Date:	Wed, 23 Feb 2022	**Prob (F-statistic):**	2.55e-15
Time:	02:51:04	**Log-Likelihood:**	-13756.
No. Observations:	1435	**AIC:**	2.752e+04
Df Residuals:	1432	**BIC:**	2.753e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	6568.3395	513.700	12.786	0.000	5560.655	7576.024
CC	0.9398	0.220	4.268	0.000	0.508	1.372
Doors	662.3187	98.089	6.752	0.000	469.906	854.732

Omnibus:	448.494	**Durbin-Watson:**	0.291
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1297.612
Skew:	1.602	**Prob(JB):**	1.69e-282
Kurtosis:	6.382	**Cond. No.**	9.09e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.09e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Model validation Techniques

two techniques - 1. collinearity check

In [108]:

```
# 1) Collinearity Problem Check
# Calculate VIF = 1/(1-Rsquare) for all independent variables

rsq_age=smf.ols('Age~KM+HP+CC+Doors+Gears+QT+Weight',data=cars).fit().rsquared
vif_age=1/(1-rsq_age)

rsq_KM=smf.ols('KM~Age+HP+CC+Doors+Gears+QT+Weight',data=cars).fit().rsquared
vif_KM=1/(1-rsq_KM)

rsq_HP=smf.ols('HP~Age+KM+CC+Doors+Gears+QT+Weight',data=cars).fit().rsquared
vif_HP=1/(1-rsq_HP)

rsq_CC=smf.ols('CC~Age+KM+HP+Doors+Gears+QT+Weight',data=cars).fit().rsquared
vif_CC=1/(1-rsq_CC)

rsq_DR=smf.ols('Doors~Age+KM+HP+CC+Gears+QT+Weight',data=cars).fit().rsquared
vif_DR=1/(1-rsq_DR)

rsq_GR=smf.ols('Gears~Age+KM+HP+CC+Doors+QT+Weight',data=cars).fit().rsquared
vif_GR=1/(1-rsq_GR)

rsq_QT=smf.ols('QT~Age+KM+HP+CC+Doors+Gears+Weight',data=cars).fit().rsquared
vif_QT=1/(1-rsq_QT)

rsq_WT=smf.ols('Weight~Age+KM+HP+CC+Doors+Gears+QT',data=cars).fit().rsquared
vif_WT=1/(1-rsq_WT)

# Putting the values in Dataframe format
d1={'Variables':['Age','KM','HP','CC','Doors','Gears','QT','Weight'],
    'Vif':[vif_age,vif_KM,vif_HP,vif_CC,vif_DR,vif_GR,vif_QT,vif_WT]}
Vif_df=pd.DataFrame(d1)
Vif_df
```

Out[108]:

	Variables	Vif
0	Age	1.876236
1	KM	1.757178
2	HP	1.419180
3	CC	1.163470
4	Doors	1.155890
5	Gears	1.098843
6	QT	2.295375
7	Weight	2.487180

In [109]:

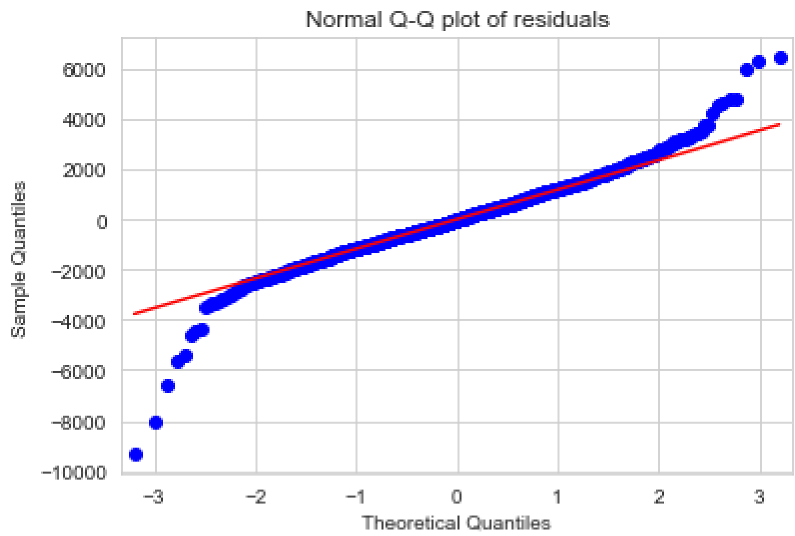
```
# None variable has VIF>20, No Collinearity, so consider all varaibles in Regression
```

2. Residual Test

Test for normality of residuals(Q-Q plot)

In [110]:

```
qqplot =sm.qqplot(model.resid,line='q')
plt.title('Normal Q-Q plot of residuals') # line = 45 to draw the diagnoal line
plt.show()
```



In [111]:

```
list(np.where(model.resid>6000)) #outlier detection from above QQ plot  of residuals.
```

Out[111]:

```
[array([109, 146, 522], dtype=int64)]
```

In [112]:

```
list(np.where(model.resid<-6000))
```

Out[112]:

```
[array([220, 600, 959], dtype=int64)]
```

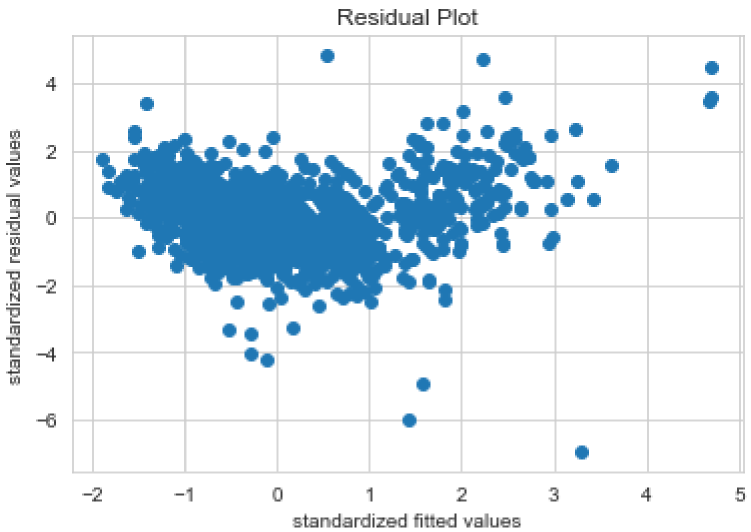
Test for homoscedasticity or Heteroscedasticity

In [113]:

```
def standard_values ( vals ):
    return(vals - vals.mean())/vals.std()
```

In [114]:

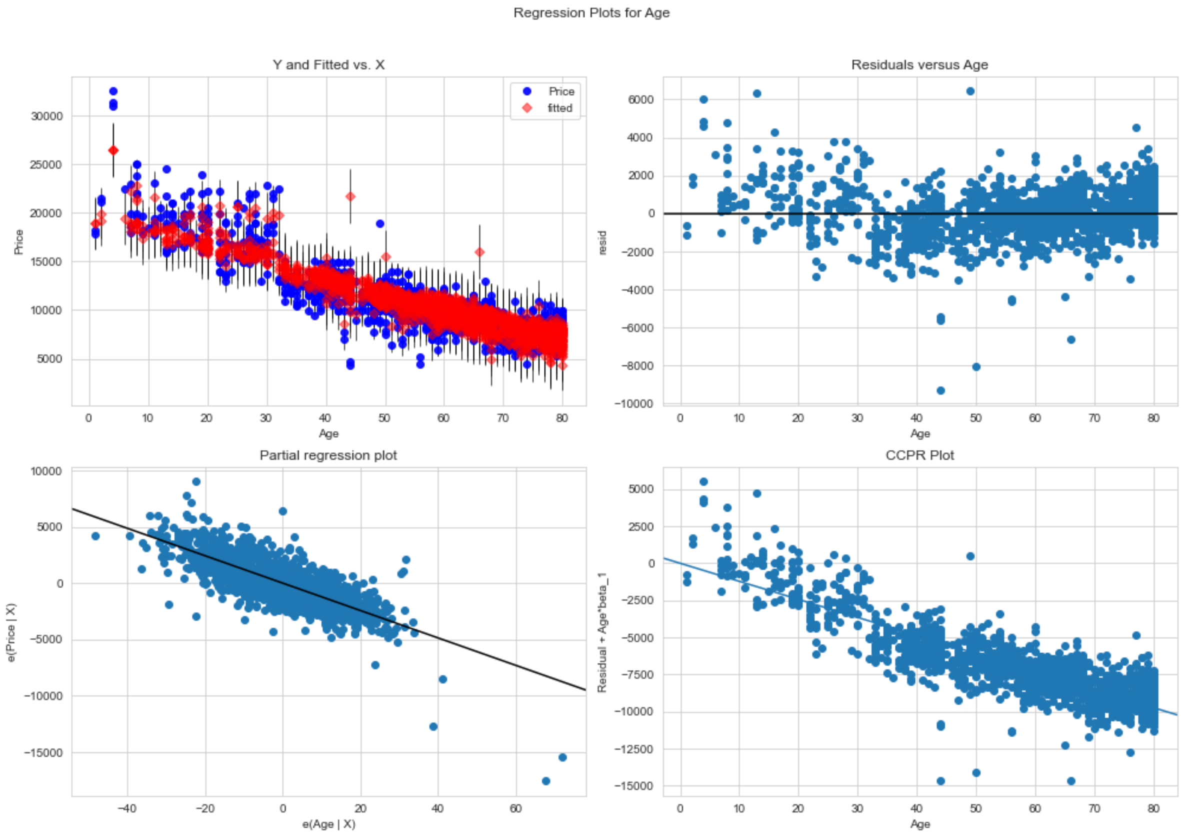
```
plt.scatter(standard_values(model.fittedvalues),standard_values(model.resid))
plt.title('Residual Plot')
plt.xlabel('standardized fitted values')
plt.ylabel('standardized residual values')
plt.show()
```



Residual Vs Regressors

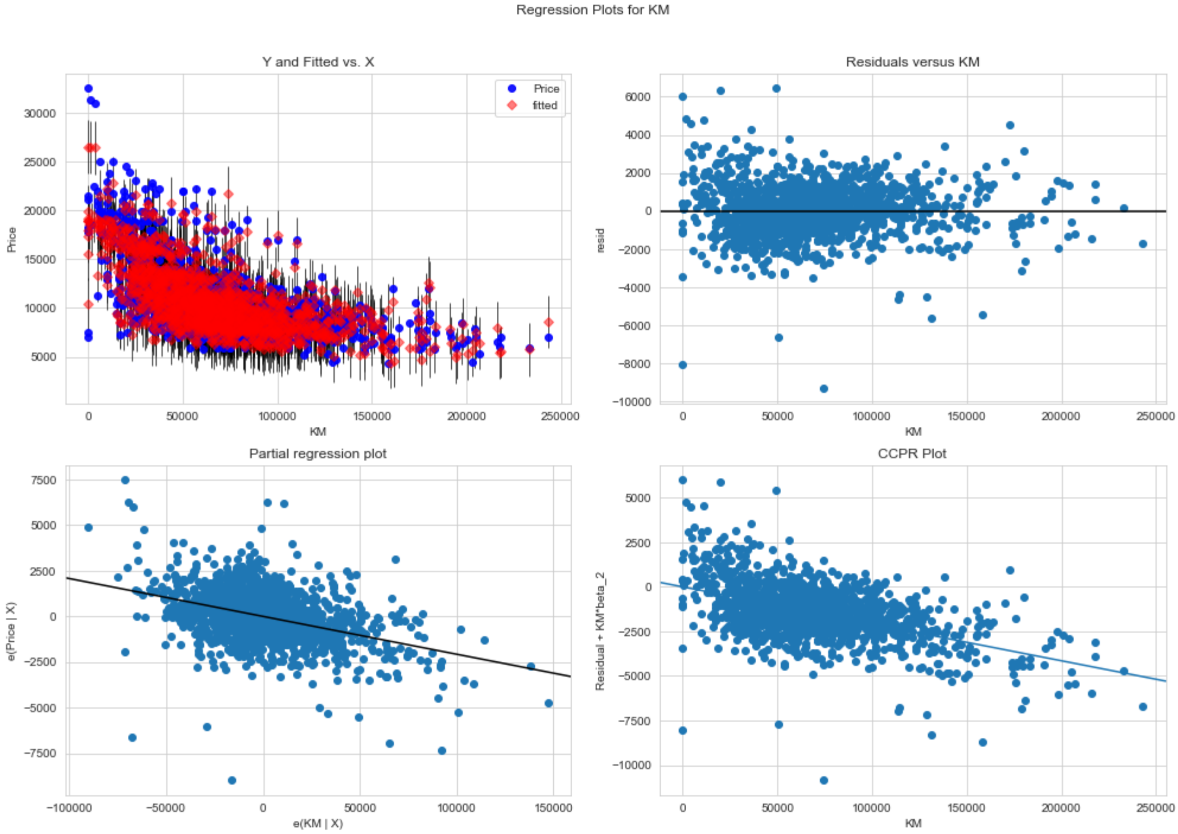
In [115]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model,"Age",fig=fig)
plt.show()
```



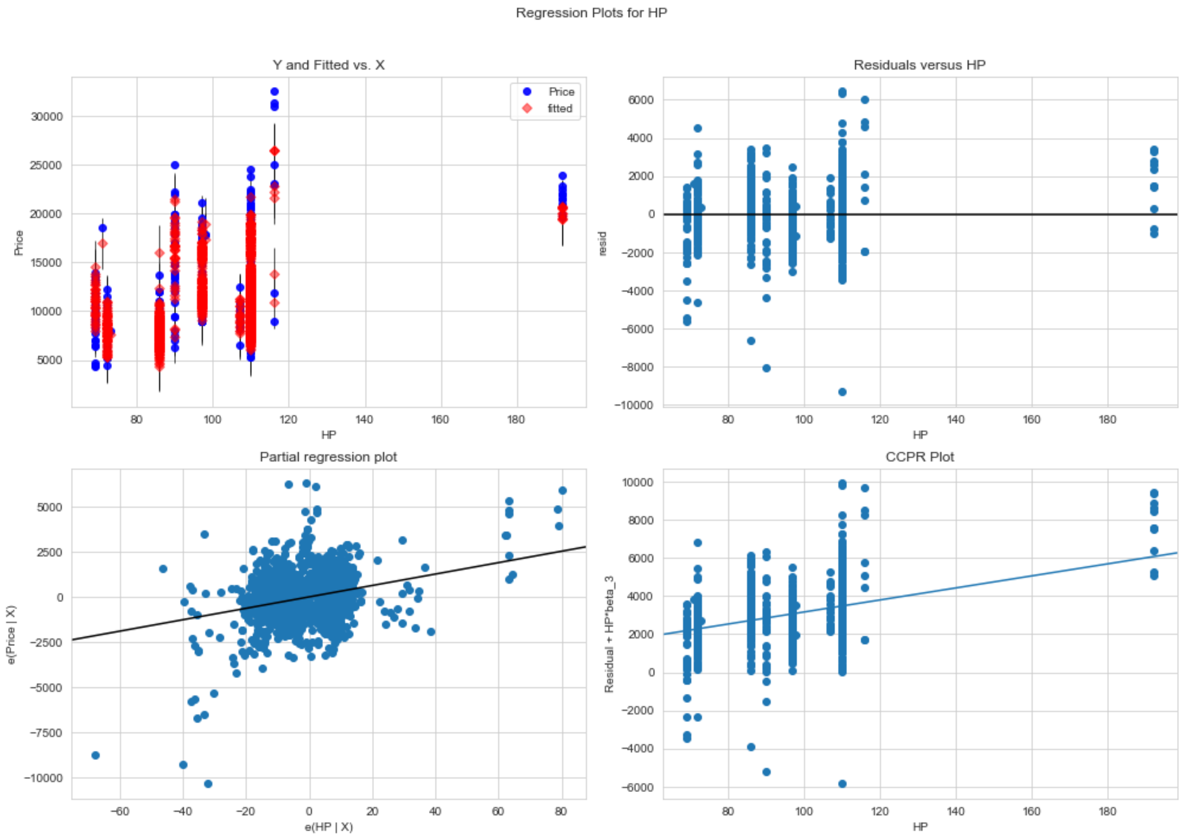
In [116]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "KM", fig=fig)
plt.show()
```



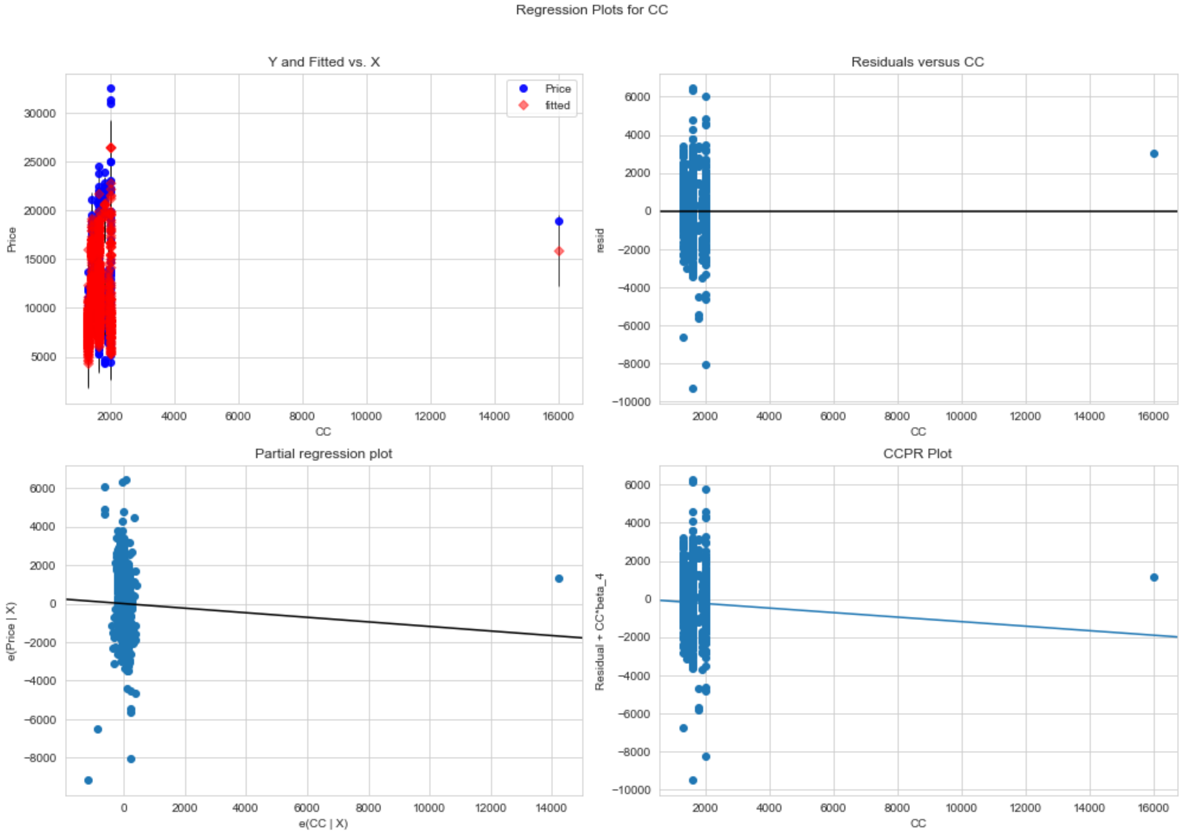
In [117]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "HP",fig=fig)
plt.show()
```



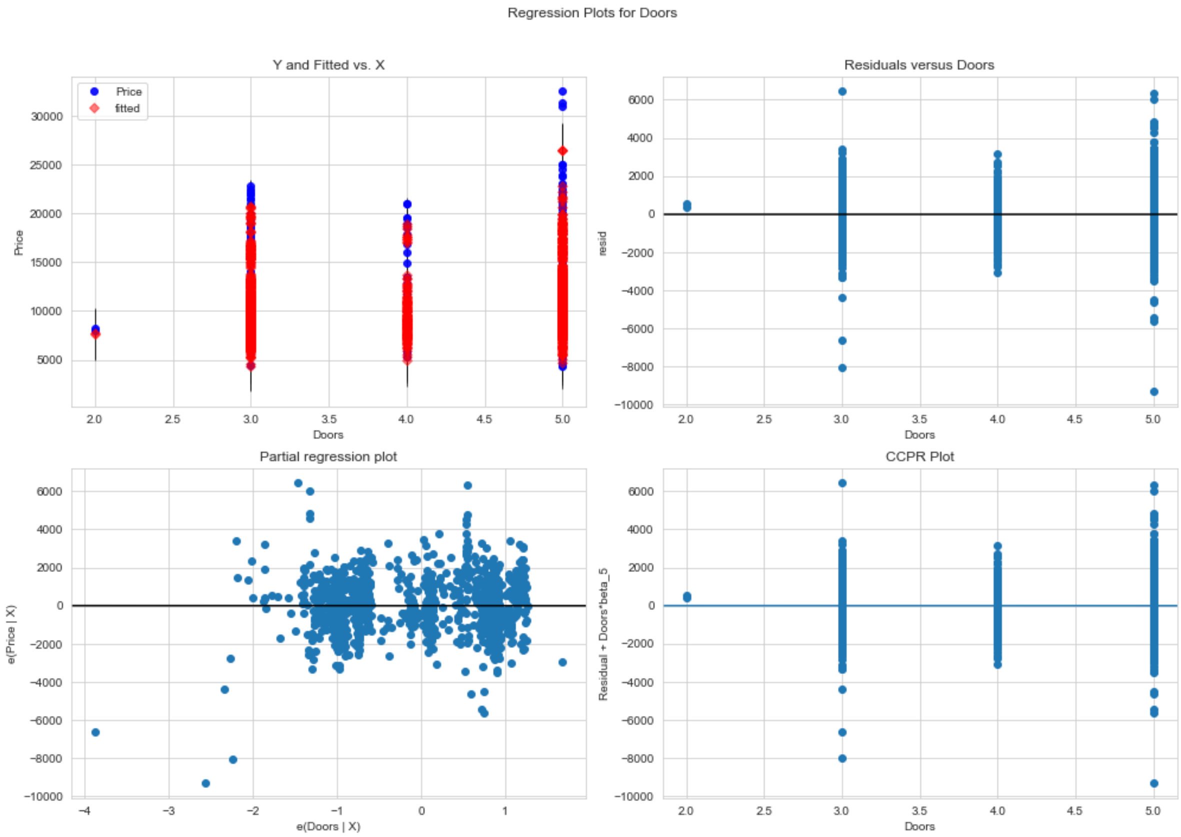
In [118]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "CC",fig=fig)
plt.show()
```



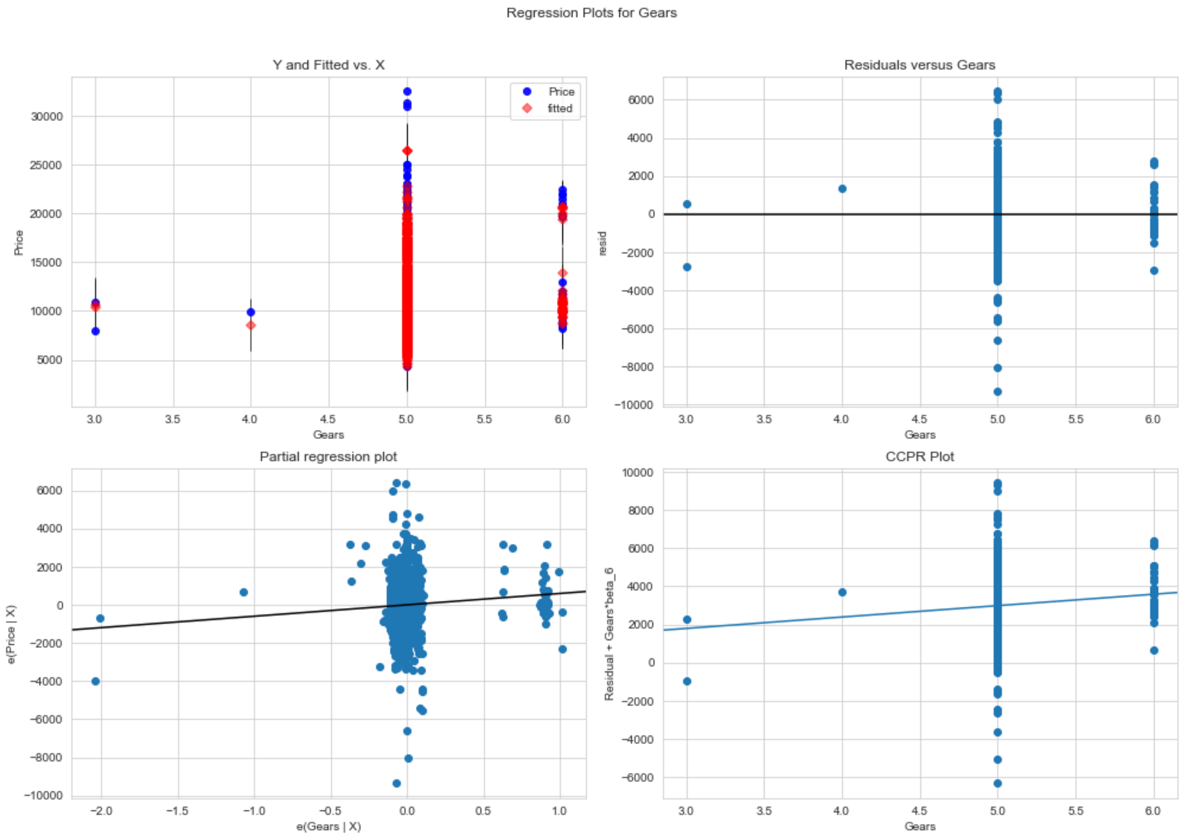
In [119]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "Doors",fig=fig)
plt.show()
```



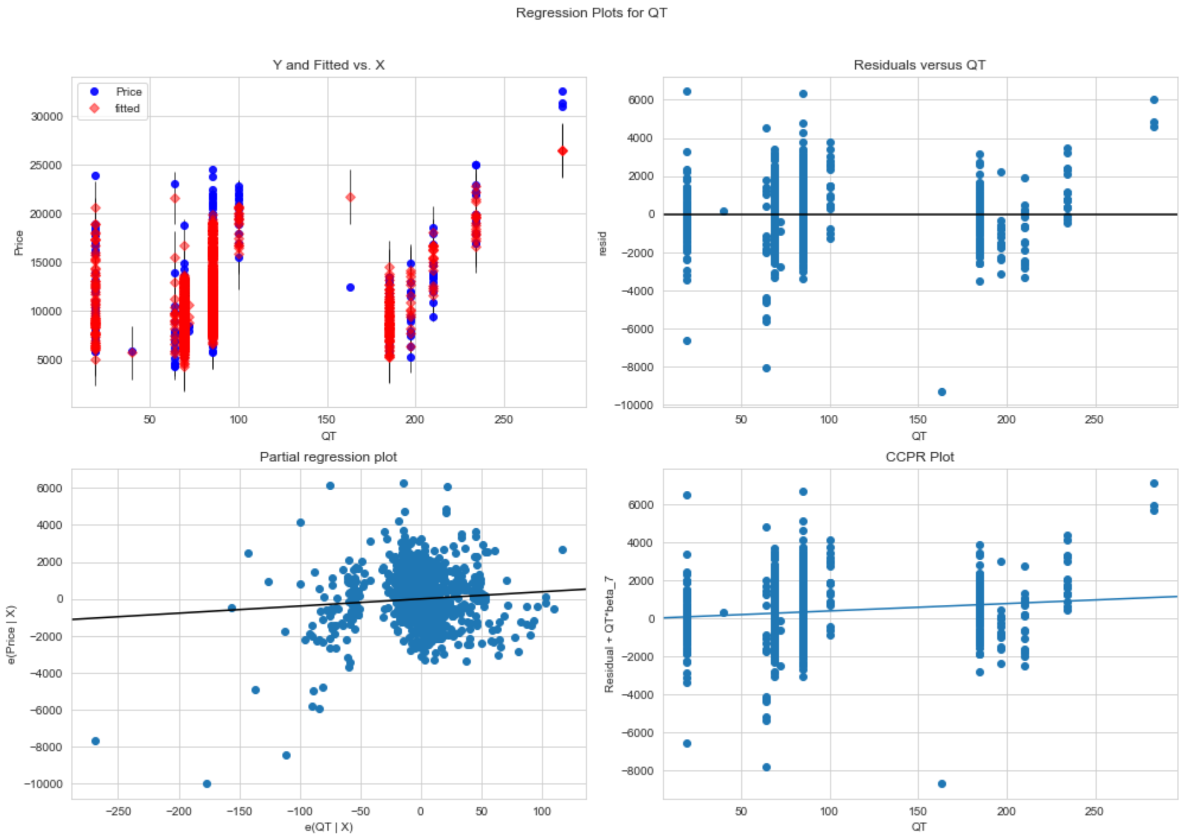
In [120]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "Gears",fig=fig)
plt.show()
```



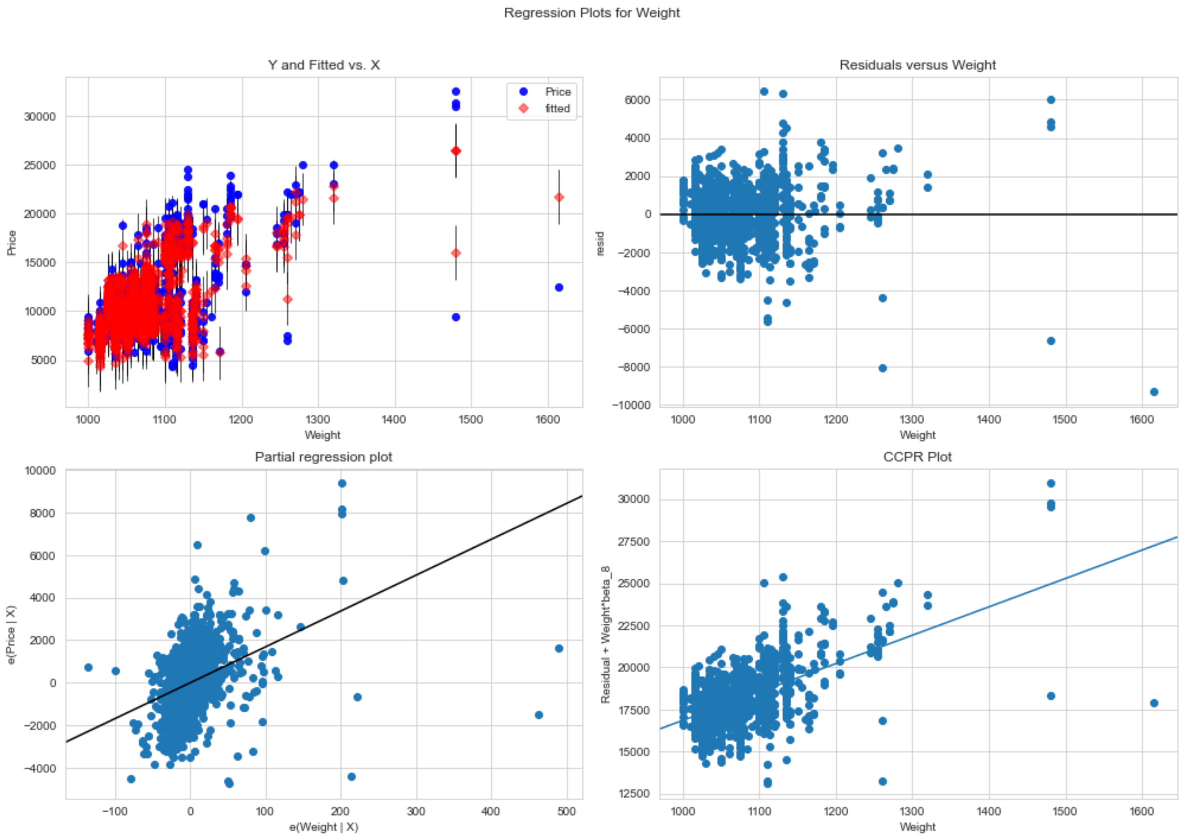
In [121]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "QT",fig=fig)
plt.show()
```



In [122]:

```
fig = plt.figure(figsize=(14,10))
fig = sm.graphics.plot_regress_exog(model, "Weight",fig=fig)
plt.show()
```



Model Deletion Diagnostics

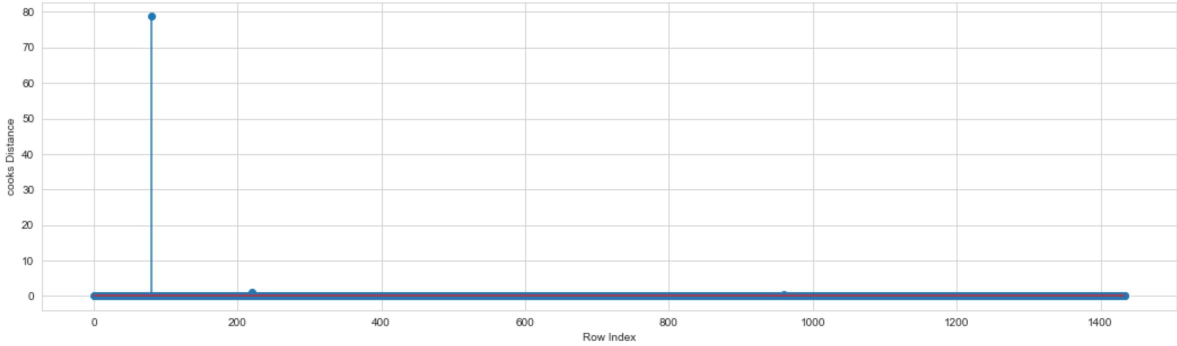
Two Techniques:- 1. Cook's Distance

In [124]:

```
model_influence = model.get_influence()
(c, _)= model_influence.cooks_distance
```

In [126]:

```
#plot the influencers value using stem plot
fig = plt.figure(figsize=(18,5))
plt.stem(np.arange(len(cars)),np.round(c, 3))
plt.xlabel('Row Index')
plt.ylabel('cooks Distance')
plt.show()
```



In [127]:

```
# index and value of influencers where c is more than .5
(np.argmax(c),np.max(c))
```

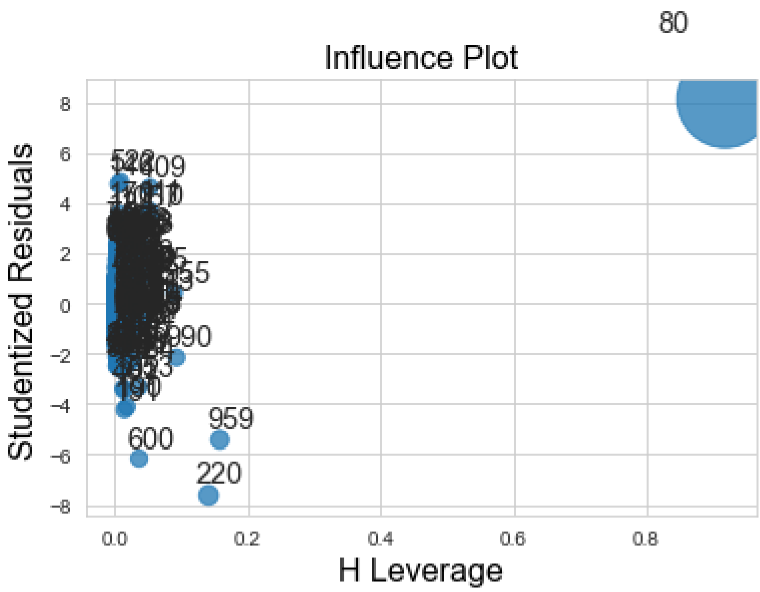
Out[127]:

```
(80, 78.7295058224916)
```

2. Leverage value

In [128]:

```
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are i
influence_plot(model)
plt.show()
```



In [129]:

```
cars.shape
```

Out[129]:

```
(1435, 9)
```

In [131]:

```
k =cars.shape[1]
n = cars.shape[0]
leverage_cutoff = 3*((k+1)/n)
leverage_cutoff
```

Out[131]:

```
0.020905923344947737
```

In [132]:

```
cars[cars.index.isin([80])]
```

Out[132]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
80	18950	25	20019	110	16000	5	5	100	1180

Improving the model

In [133]:

```
# Creating a copy of data so that original dataset is not affected
data = cars.copy()
data
```

Out[133]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90	2000	3	5	210	1165
1	13750	23	72937	90	2000	3	5	210	1165
2	13950	24	41711	90	2000	3	5	210	1165
3	14950	26	48000	90	2000	3	5	210	1165
4	13750	30	38500	90	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1430	7500	69	20544	86	1300	3	5	69	1025
1431	10845	72	19000	86	1300	3	5	69	1015
1432	8500	71	17016	86	1300	3	5	69	1015
1433	7250	70	16916	86	1300	3	5	69	1015
1434	6950	76	1	110	1600	5	5	19	1114

1435 rows × 9 columns

In [134]:

```
data2=data.drop(data.index[[80]],axis=0).reset_index(drop=True)
data2
```

Out[134]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90	2000	3	5	210	1165
1	13750	23	72937	90	2000	3	5	210	1165
2	13950	24	41711	90	2000	3	5	210	1165
3	14950	26	48000	90	2000	3	5	210	1165
4	13750	30	38500	90	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1429	7500	69	20544	86	1300	3	5	69	1025
1430	10845	72	19000	86	1300	3	5	69	1015
1431	8500	71	17016	86	1300	3	5	69	1015
1432	7250	70	16916	86	1300	3	5	69	1015
1433	6950	76	1	110	1600	5	5	19	1114

1434 rows × 9 columns

In [135]:

```
final_data =smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=data2).fit()
final_data.summary()
```

Out[135]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.868
Model:	OLS	**Adj. R-squared:**	0.867
Method:	Least Squares	**F-statistic:**	1172.
Date:	Wed, 23 Feb 2022	**Prob (F-statistic):**	0.00
Time:	03:00:33	**Log-Likelihood:**	-12326.
No. Observations:	1434	**AIC:**	2.467e+04
Df Residuals:	1425	**BIC:**	2.472e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-6197.9334	1383.989	-4.478	0.000	-8912.808	-3483.059
Age	-120.5074	2.561	-47.048	0.000	-125.532	-115.483
KM	-0.0178	0.001	-13.931	0.000	-0.020	-0.015
HP	39.2245	2.912	13.470	0.000	33.512	44.937
CC	-2.5088	0.307	-8.162	0.000	-3.112	-1.906
Doors	-26.5129	39.235	-0.676	0.499	-103.478	50.452
Gears	527.1292	192.832	2.734	0.006	148.864	905.395
QT	8.9414	1.427	6.268	0.000	6.143	11.740
Weight	20.0627	1.118	17.944	0.000	17.869	22.256

Omnibus:	242.181	**Durbin-Watson:**	1.595
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2635.168
Skew:	-0.427	**Prob(JB):**	0.00
Kurtosis:	9.586	**Cond. No.**	3.14e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.14e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Predicting for new Data

In [138]:

```
# say New data for prediction is
new_data=pd.DataFrame({'Age':12,"KM":40000,"HP":80,"CC":1300,"Doors":4,"Gears":5,"QT":69,"W
new_data
```

Out[138]:

	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	12	40000	80	1300	4	5	69	1012

In [140]:

```
# Manual Prediction of Price
final_data.predict(new_data)
```

Out[140]:

```
0    14970.556739
dtype: float64
```

In [141]:

```
pred_y=final_data.predict(data2)
pred_y
```

Out[141]:

```
0       16513.565909
1       16051.656226
2       16486.949796
3       16133.995128
4       15921.372341
            ...
1429     8970.611964
1430     8435.944671
1431     8591.765915
1432     8714.053275
1433     9966.948423
Length: 1434, dtype: float64
```

In [139]:

```
#Table containing R^2 value for each prepared model
```

In [137]:

```
d2={'Prep_Models':['Model','Final_Model'],'Rsquared':[model.rsquared,final_data.rsquared]}
table=pd.DataFrame(d2)
table
```

Out[137]:

	Prep_Models	Rsquared
0	Model	0.862520
1	Final_Model	0.868116

In [ ]:

```

```