Imoprting Required Libraries

In [1]:

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

Impoting data

In [2]:

```
toyota = pd.read_excel("Toyota.xlsx")
toyota
```

Out[2]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	10	2002	46986	Diesel	90.0	1
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	10	2002	72937	Diesel	90.0	1
2	3	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	9	2002	41711	Diesel	90.0	1
3	4	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	14950	26	7	2002	48000	Diesel	90.0	0
4	5	TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors	13750	30	3	2002	38500	Diesel	90.0	0
...	...	...	...	...	...	...	...	...	...	...

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Colo
1431	1438	TOYOTA Corolla 1.3 16V HATCHB G6 2/3-Doors	7500	69	12	1998	20544	Petrol	86.0	1
1432	1439	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	10845	72	9	1998	19000	Petrol	86.0	C
1433	1440	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	8500	71	10	1998	17016	Petrol	86.0	C
1434	1441	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	7250	70	11	1998	16916	Petrol	86.0	1
1435	1442	TOYOTA Corolla 1.6 LB LINEA TERRA 4/5-Doors	6950	76	5	1998	1	Petrol	110.0	C

1436 rows × 38 columns

Data Understanding

In [3]:

```
toyota.shape
```

Out[3]:

```
(1436, 38)
```

In [4]:

```
toyota.isna().sum()
```

Out[4]:

```
Id               0
Model            0
Price            0
Age_08_04        0
Mfg_Month        0
Mfg_Year         0
KM               0
Fuel_Type        0
HP               2
```

```
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 [5]:

```
toyota.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                1434 non-null   float64
 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: float64(1), int64(34), object(3)
memory usage: 426.4+ KB
```

In [6]:

```
toyota=pd.concat([toyota.iloc[:,2:4],toyota.iloc[:,6:7],toyota.iloc[:,8:9],toyota.il
toyota
```

Out[6]:

	Price	Age_08_04	KM	HP	cc	Doors	Gears	Quarterly_Tax	Weight
0	13500	23	46986	90.0	2000	3	5	210	1165
1	13750	23	72937	90.0	2000	3	5	210	1165
2	13950	24	41711	90.0	2000	3	5	210	1165
3	14950	26	48000	90.0	2000	3	5	210	1165
4	13750	30	38500	90.0	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1431	7500	69	20544	86.0	1300	3	5	69	1025
1432	10845	72	19000	86.0	1300	3	5	69	1015
1433	8500	71	17016	86.0	1300	3	5	69	1015
1434	7250	70	16916	86.0	1300	3	5	69	1015
1435	6950	76	1	110.0	1600	5	5	19	1114

1436 rows × 9 columns

In [7]:

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

Out[7]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90.0	2000	3	5	210	1165
1	13750	23	72937	90.0	2000	3	5	210	1165
2	13950	24	41711	90.0	2000	3	5	210	1165
3	14950	26	48000	90.0	2000	3	5	210	1165
4	13750	30	38500	90.0	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1431	7500	69	20544	86.0	1300	3	5	69	1025
1432	10845	72	19000	86.0	1300	3	5	69	1015
1433	8500	71	17016	86.0	1300	3	5	69	1015
1434	7250	70	16916	86.0	1300	3	5	69	1015
1435	6950	76	1	110.0	1600	5	5	19	1114

1436 rows × 9 columns

In [8]:
```
toyota[toyota.duplicated()]
```

Out[8]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
113	24950	8	13253	116.0	2000	5	5	234	1320

In [9]:
```
toyota = toyota.drop_duplicates().reset_index(drop = True)
toyota
```

Out[9]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90.0	2000	3	5	210	1165
1	13750	23	72937	90.0	2000	3	5	210	1165
2	13950	24	41711	90.0	2000	3	5	210	1165
3	14950	26	48000	90.0	2000	3	5	210	1165
4	13750	30	38500	90.0	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1430	7500	69	20544	86.0	1300	3	5	69	1025
1431	10845	72	19000	86.0	1300	3	5	69	1015
1432	8500	71	17016	86.0	1300	3	5	69	1015
1433	7250	70	16916	86.0	1300	3	5	69	1015
1434	6950	76	1	110.0	1600	5	5	19	1114

1435 rows × 9 columns

In [10]:
```
toyota.describe()
```

Out[10]:

	Price	Age	KM	HP	CC	Doors	Gea
count	1435.000000	1435.000000	1435.000000	1433.000000	1435.000000	1435.000000	1435.0000
mean	10720.915679	55.980488	68571.782578	101.503140	1576.560976	4.032753	5.0264
std	3608.732978	18.563312	37491.094553	14.988316	424.387533	0.952667	0.1885
min	4350.000000	1.000000	1.000000	69.000000	1300.000000	2.000000	3.0000
25%	8450.000000	44.000000	43000.000000	90.000000	1400.000000	3.000000	5.0000
50%	9900.000000	61.000000	63451.000000	110.000000	1600.000000	4.000000	5.0000
75%	11950.000000	70.000000	87041.500000	110.000000	1600.000000	5.000000	5.0000
max	32500.000000	80.000000	243000.000000	192.000000	16000.000000	5.000000	6.0000

In [11]:
```
toyota.corr()##correlation matrix
```

Out[11]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Wei
Price	1.000000	-0.876273	-0.569420	0.316412	0.124375	0.183604	0.063831	0.211508	0.575
Age	-0.876273	1.000000	0.504575	-0.156549	-0.096549	-0.146929	-0.005629	-0.193319	-0.466
KM	-0.569420	0.504575	1.000000	-0.333279	0.103822	-0.035193	0.014890	0.283312	-0.023
HP	0.316412	-0.156549	-0.333279	1.000000	0.035677	0.091091	0.209590	-0.301712	0.089
CC	0.124375	-0.096549	0.103822	0.035677	1.000000	0.079254	0.014732	0.305982	0.335
Doors	0.183604	-0.146929	-0.035193	0.091091	0.079254	1.000000	-0.160101	0.107353	0.301
Gears	0.063831	-0.005629	0.014890	0.209590	0.014732	-0.160101	1.000000	-0.005125	0.021
QT	0.211508	-0.193319	0.283312	-0.301712	0.305982	0.107353	-0.005125	1.000000	0.621
Weight	0.575869	-0.466484	-0.023969	0.089396	0.335077	0.301734	0.021238	0.621988	1.000

In [12]:

```
toyota.dtypes
```

Out[12]:

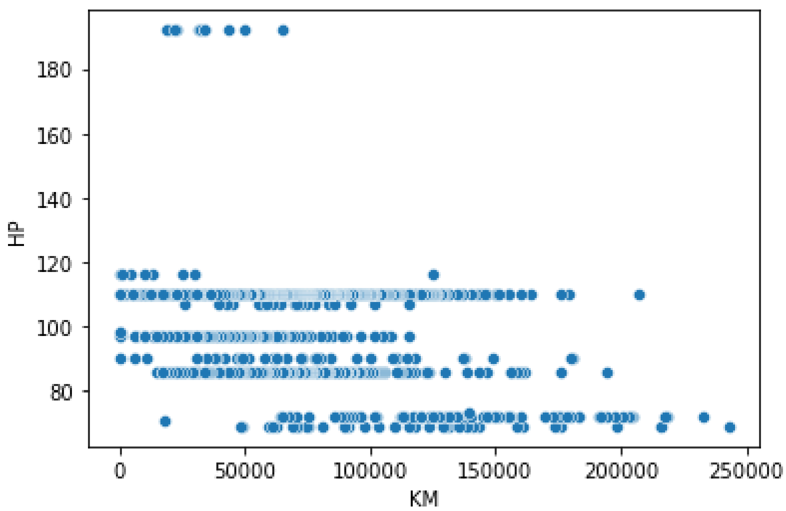
```
Price        int64
Age          int64
KM           int64
HP         float64
CC           int64
Doors        int64
Gears        int64
QT           int64
Weight       int64
dtype: object
```

Check weather the assumptions are matching or not

1:Linearity Check

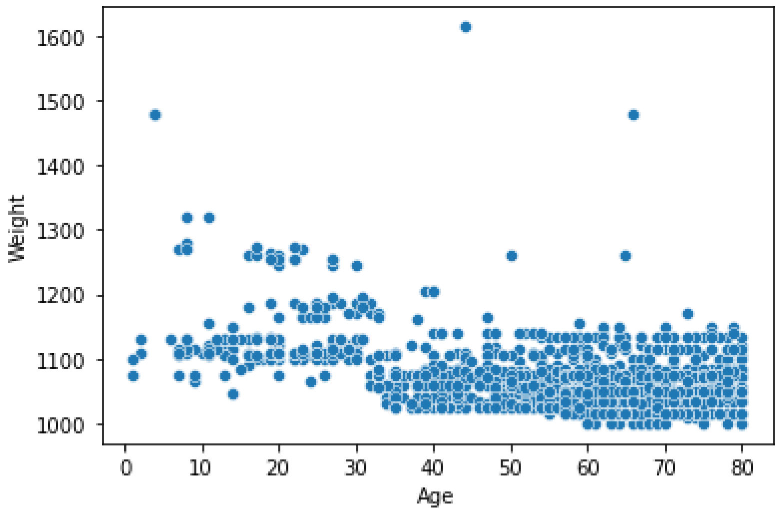
In [13]:

```
sns.scatterplot(x = 'KM' , y = 'HP',data = toyota)
plt.show()
```



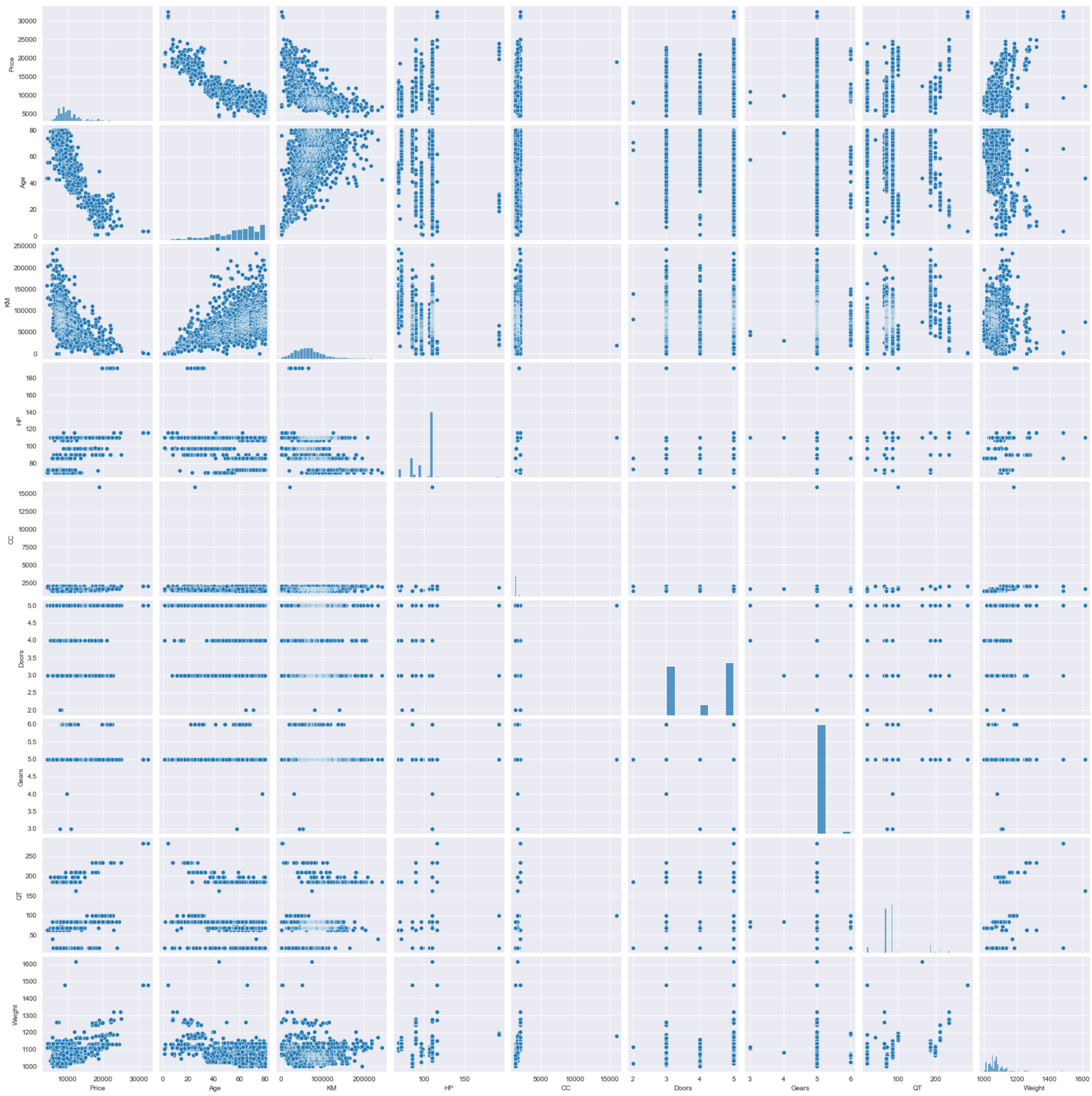
In [14]:

```
sns.scatterplot(x = 'Age' , y = 'Weight',data = toyota)
plt.show()
```



In [15]:

```
sns.set_style(style = 'darkgrid')
sns.pairplot(toyota)
plt.show()
```



Model Building

In [16]:

```
linear_model = smf.ols(formula = 'Price~Age+Age+KM+HP+CC+Doors+Gears+QT+Weight',data
linear_model
```

Out[16]:

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x20f4062ac40>
```

Model Testing

In [17]:

```
linear_model.params
```

Out[17]:

```
Intercept   -5408.896028
Age          -121.698872
KM             -0.020712
HP             31.664670
CC             -0.117027
Doors           3.236899
Gears         601.694789
QT              3.832667
Weight         16.749870
dtype: float64
```

In [18]:

```
linear_model.tvalues,np.round(linear_model.pvalues,5)
```

Out[18]:

```
(Intercept    -3.828972
 Age         -46.547581
 KM          -16.532785
 HP           11.237564
 CC           -1.299556
 Doors         0.080783
 Gears         3.055055
 QT            2.923987
 Weight       15.633974
 dtype: float64,
 Intercept    0.00013
 Age          0.00000
 KM           0.00000
 HP           0.00000
 CC           0.19396
 Doors        0.93563
 Gears        0.00229
 QT           0.00351
 Weight       0.00000
 dtype: float64)
```

In [19]:

```
linear_model.rsquared,linear_model.rsquared_adj
```

Out[19]:

```
(0.8621154742183653, 0.8613408420510528)
```

In [20]:

```
####'CC' & 'Doors' are insignificant variables so i will build the slr and mlr model

slr_cc = smf.ols(formula = 'Price~CC', data = toyota).fit()
slr_cc
```

Out[20]:

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x20f40765130>
```

In [21]:

```
##  CC has significant pvalue
```

```
slr_cc.tvalues,slr_cc.pvalues
```

Out[21]:
```
(Intercept    24.879592
 CC            4.745039
 dtype: float64,
 Intercept    7.236022e-114
 CC            2.292856e-06
 dtype: float64)
```

In [22]:
```
slr_d = smf.ols(formula = 'Price~Doors',data = toyota).fit()
slr_d
```

Out[22]:
```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x20f40731340>
```

In [23]:
```
slr_d.tvalues,slr_d.pvalues
```

Out[23]:
```
(Intercept    19.421546
 Doors         7.070520
 dtype: float64,
 Intercept    8.976407e-75
 Doors        2.404166e-12
 dtype: float64)
```

In [24]:
```
mlr_cc_d = smf.ols(formula = 'Price~CC+Doors',data = toyota).fit()
mlr_cc_d
```

Out[24]:
```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x20f4076b040>
```

In [25]:
```
mlr_cc_d.tvalues,mlr_cc_d.pvalues
```

Out[25]:
```
(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)
```

Model Validation Techniques

Two Techniques:

1. Collinearity Check &

2. Residual Analysis

1) Collinearity Problem Check

Calculate VIF = 1/(1-Rsquare) for all independent variables

In [26]:

```
##for Age~Km
rsq_age = smf.ols(formula = 'Age~KM+HP+CC+Doors+Gears+QT+Weight',data = toyota).fit(
vif_age = 1/(1-rsq_age)

##for KM~Age
rsq_km = smf.ols(formula = 'KM~Age+HP+CC+Doors+Gears+QT+Weight',data = toyota ).fit(
vif_km = 1/(1-rsq_km)

##for HP~Age
rsq_hp = smf.ols(formula = 'HP~Age+KM+CC+Doors+Gears+QT+Weight',data = toyota).fit()
vif_hp = 1/(1-rsq_hp)

##for CC~Age
rsq_cc = smf.ols(formula = 'CC~Age+HP+KM+Doors+Gears+QT+Weight',data = toyota).fit()
vif_cc = 1/(1-rsq_cc)

##for Doors~Age
rsq_d = smf.ols(formula = 'Doors~Age+HP+KM+CC+Gears+QT+Weight',data = toyota).fit().
vif_d = 1/(1-rsq_d)

##for Gears~Age
rsq_gr = smf.ols(formula = 'Gears~Age+HP+KM+CC+Doors+QT+Weight',data = toyota).fit()
vif_gr = 1/(1-rsq_gr)

##for QT~Age
rsq_qt = smf.ols(formula = 'QT~Age+HP+KM+CC+Doors+Gears+Weight',data = toyota).fit()
vif_qt = 1/(1-rsq_qt)

##for Weight~Age
rsq_w = smf.ols(formula = 'Weight~Age+HP+KM+CC+Doors+Gears+QT',data = toyota).fit().
vif_w = 1/(1-rsq_w)
```

In [27]:

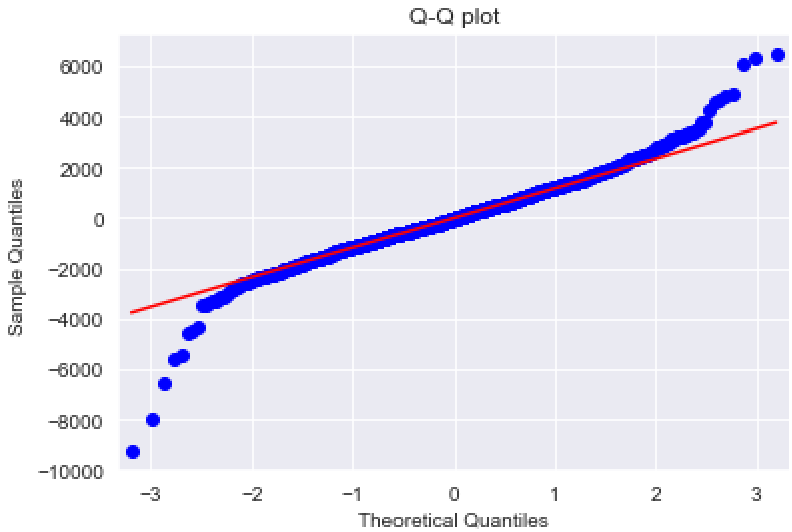
```
### dataframe formate
new_data = {'Variabels' : ['Age','Hp','KM','CC','Doors','QT','Gears','Weight'],
       'Vif' : [vif_age,vif_km,vif_hp,vif_cc,vif_d,vif_gr,vif_qt,vif_w]}
vif_data = pd.DataFrame(new_data)
vif_data
```

Out[27]:

	Variabels	Vif
0	Age	1.872029
1	Hp	1.755990
2	KM	1.419154
3	CC	1.162724
4	Doors	1.159067
5	QT	1.098992
6	Gears	2.282157
7	Weight	2.477432

2)Residual Analysis

In [28]:
```
sm.qqplot(linear_model.resid,line = 'q')
plt.title('Q-Q plot')
plt.show()
```



In [29]:
```
list(np.where(linear_model.resid>6000))
```

Out[29]: [array([107, 144, 520], dtype=int64)]

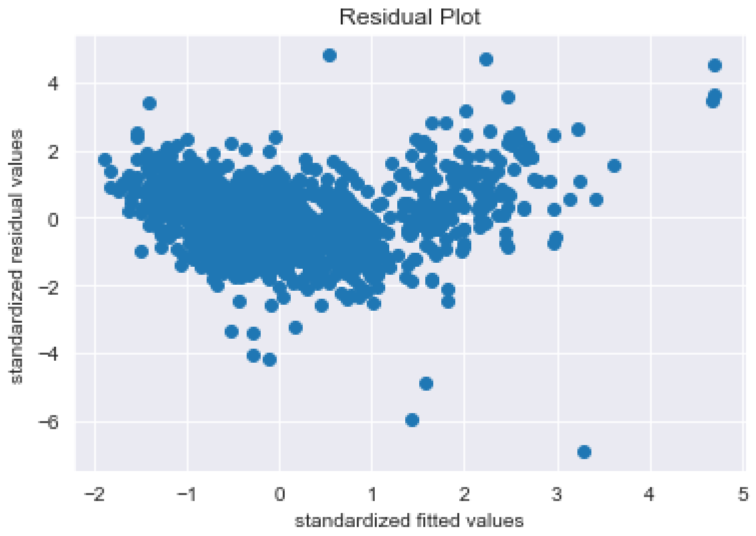
In [30]:
```
list(np.where(linear_model.resid<-6000))
```

Out[30]: [array([218, 598, 957], dtype=int64)]

Testing for Homoscedasticity or Heteroscedasticity

In [31]:
```
def standard_values(vals):
    return(vals-vals.mean())/vals.std()
```

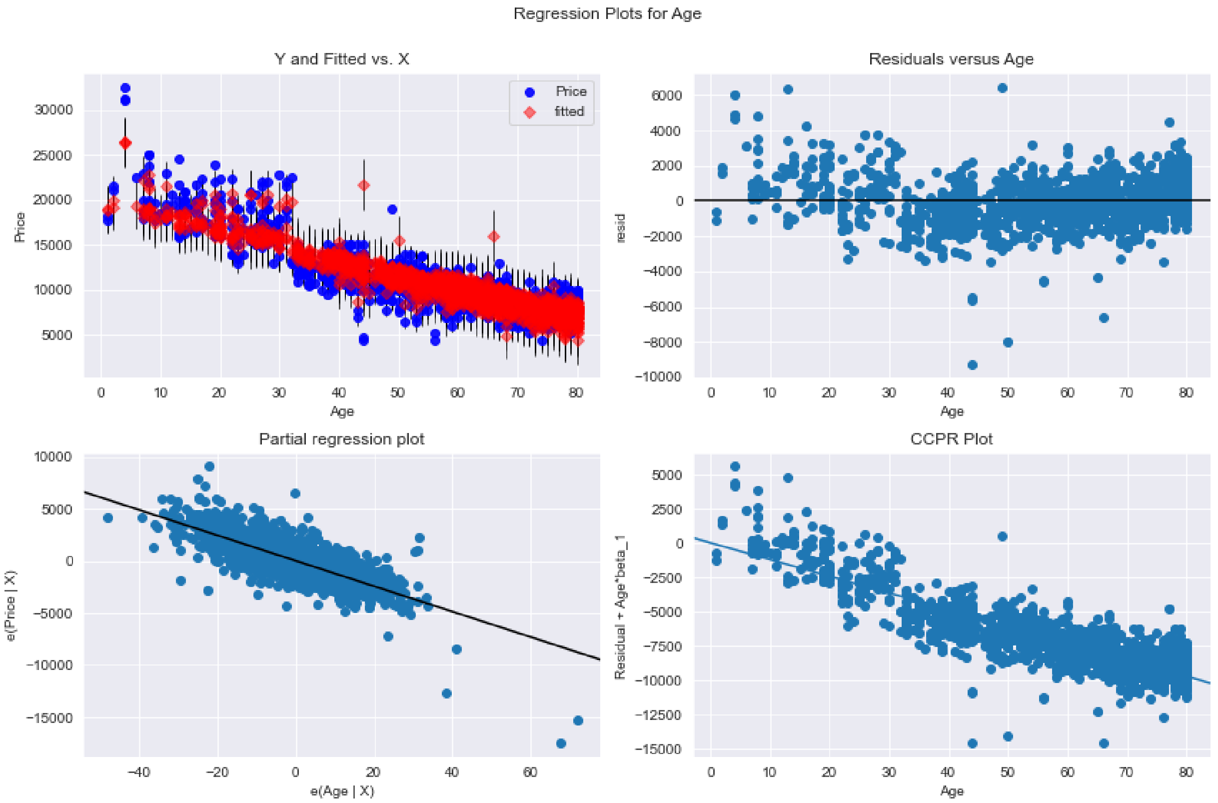
In [34]:
```
plt.scatter(standard_values(linear_model.fittedvalues),standard_values(linear_model.
plt.title('Residual Plot')
plt.xlabel('standardized fitted values')
plt.ylabel('standardized residual values')
plt.show()
```



In [35]:

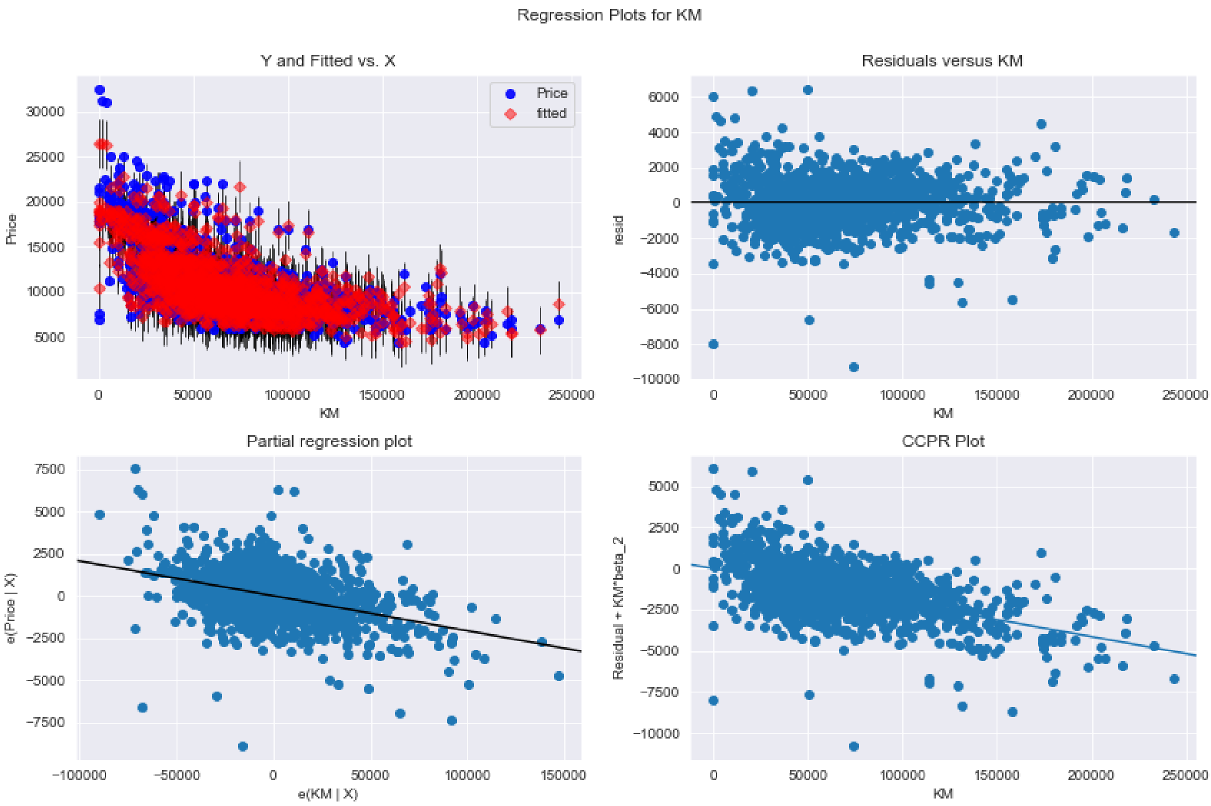
```
### Test for errors or Residuals Vs Regressors or independent 'x' variables or predi

fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'Age',fig = fig)
plt.show()
```



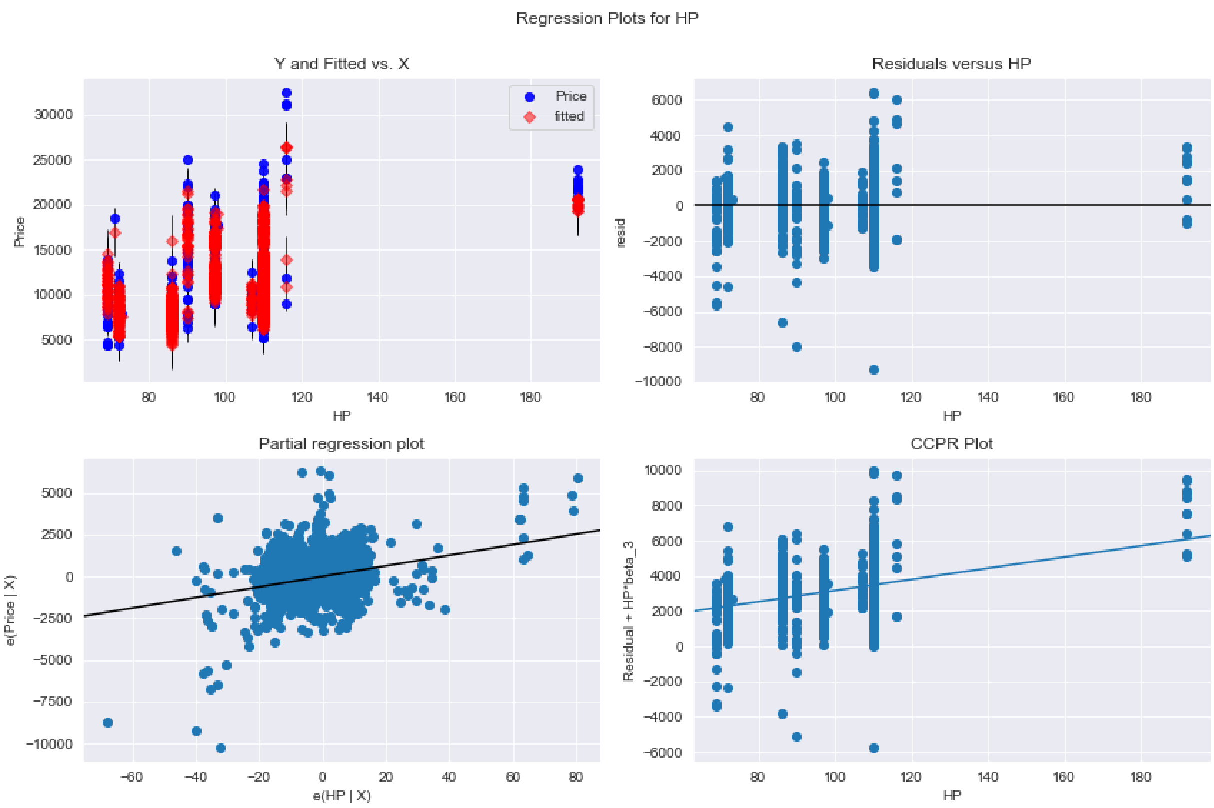
In [36]:

```
fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'KM',fig = fig)
plt.show()
```



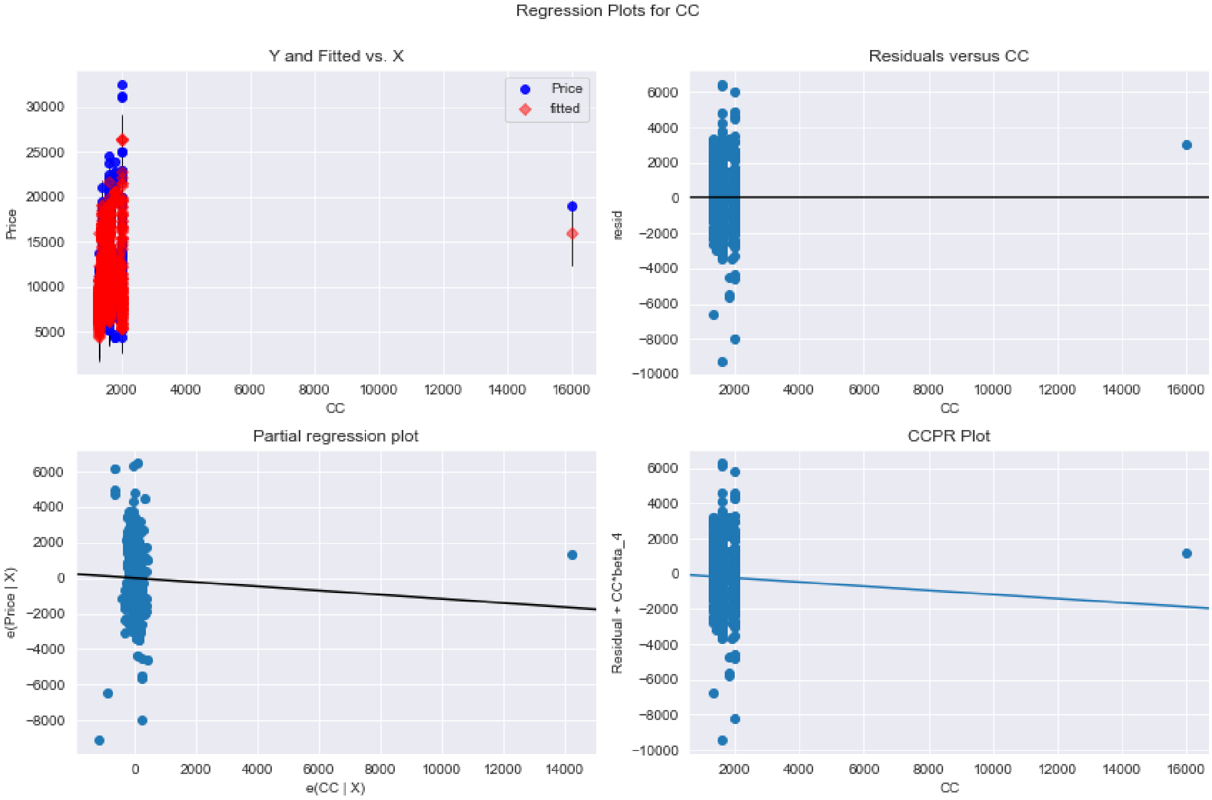
In [37]:

```python
fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'HP',fig = fig)
plt.show()
```

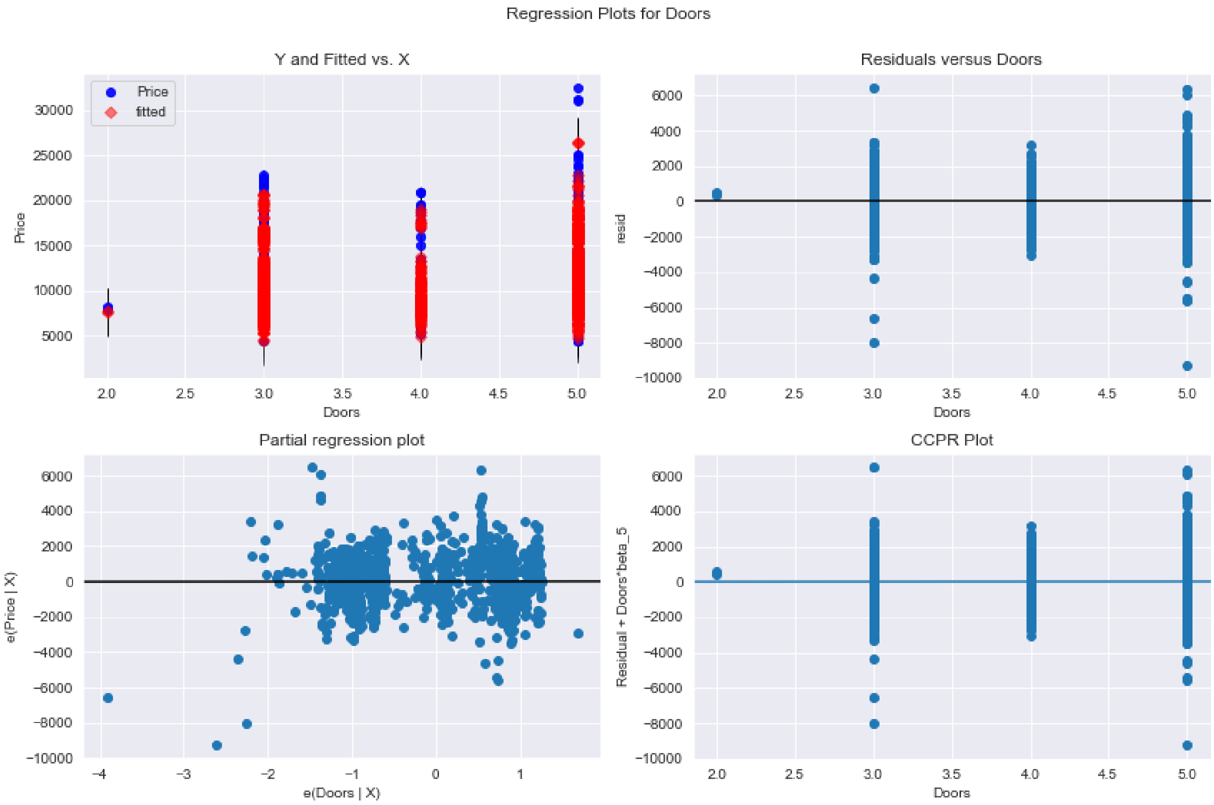


In [38]:

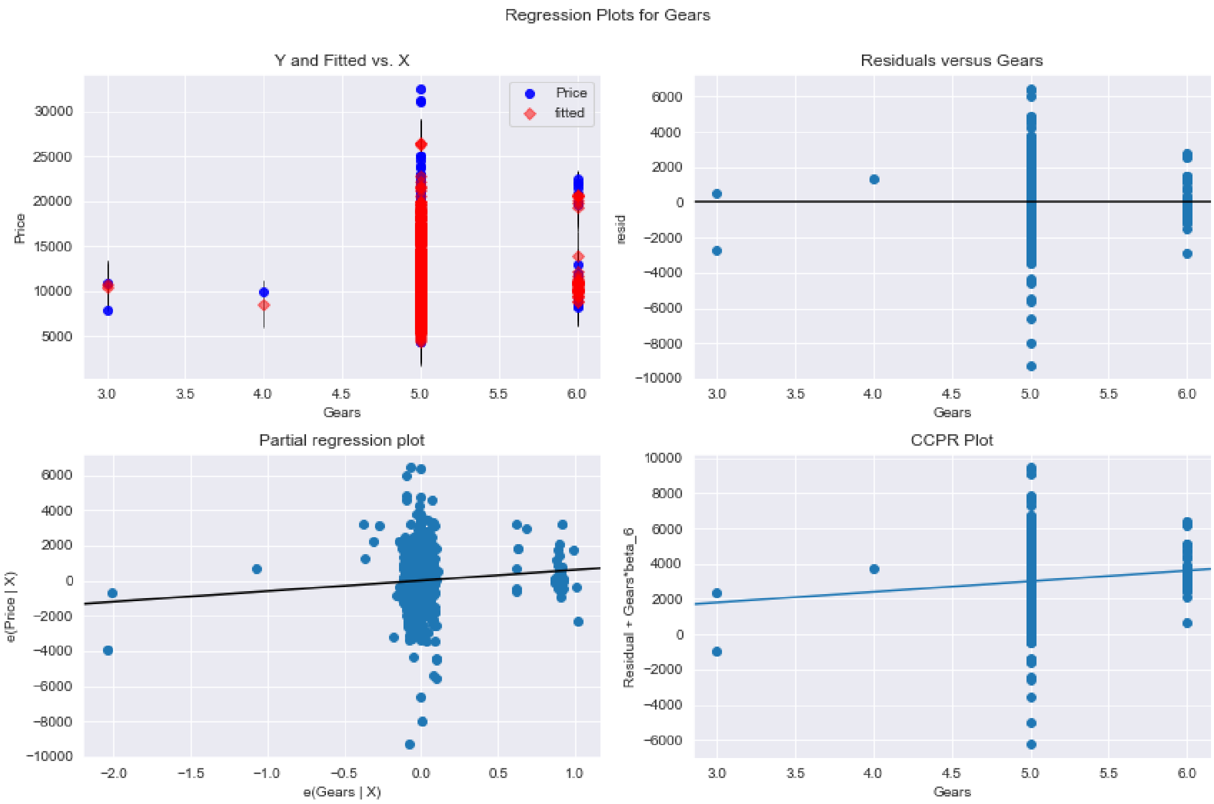
```python
fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'CC',fig = fig)
plt.show()
```



In [39]:
```
fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'Doors',fig = fig)
plt.show()
```

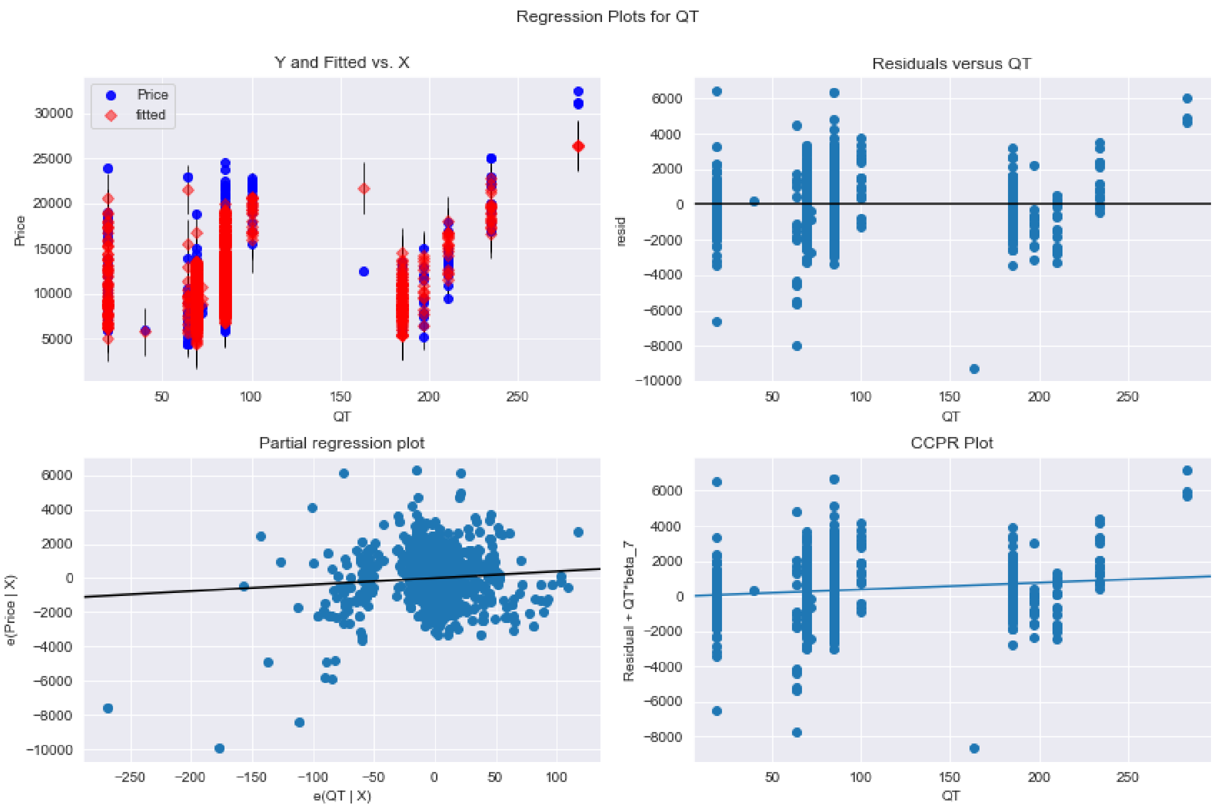


In [40]:
```
fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'Gears',fig = fig)
plt.show()
```



In [41]:

```
fig = plt.figure(figsize = (12,8))
sm.graphics.plot_regress_exog(linear_model,'QT',fig = fig)
plt.show()
```



Model Deletion Diagnostics (checking Outliers or Influencers)

Two Techniques : 1. Cook's Distance & 2. Leverage value

1. Cook's Distance:

If Cook's distance > 1, then it's an outlier

In [42]:
```
(c,_) = linear_model.get_influence().cooks_distance
c
```

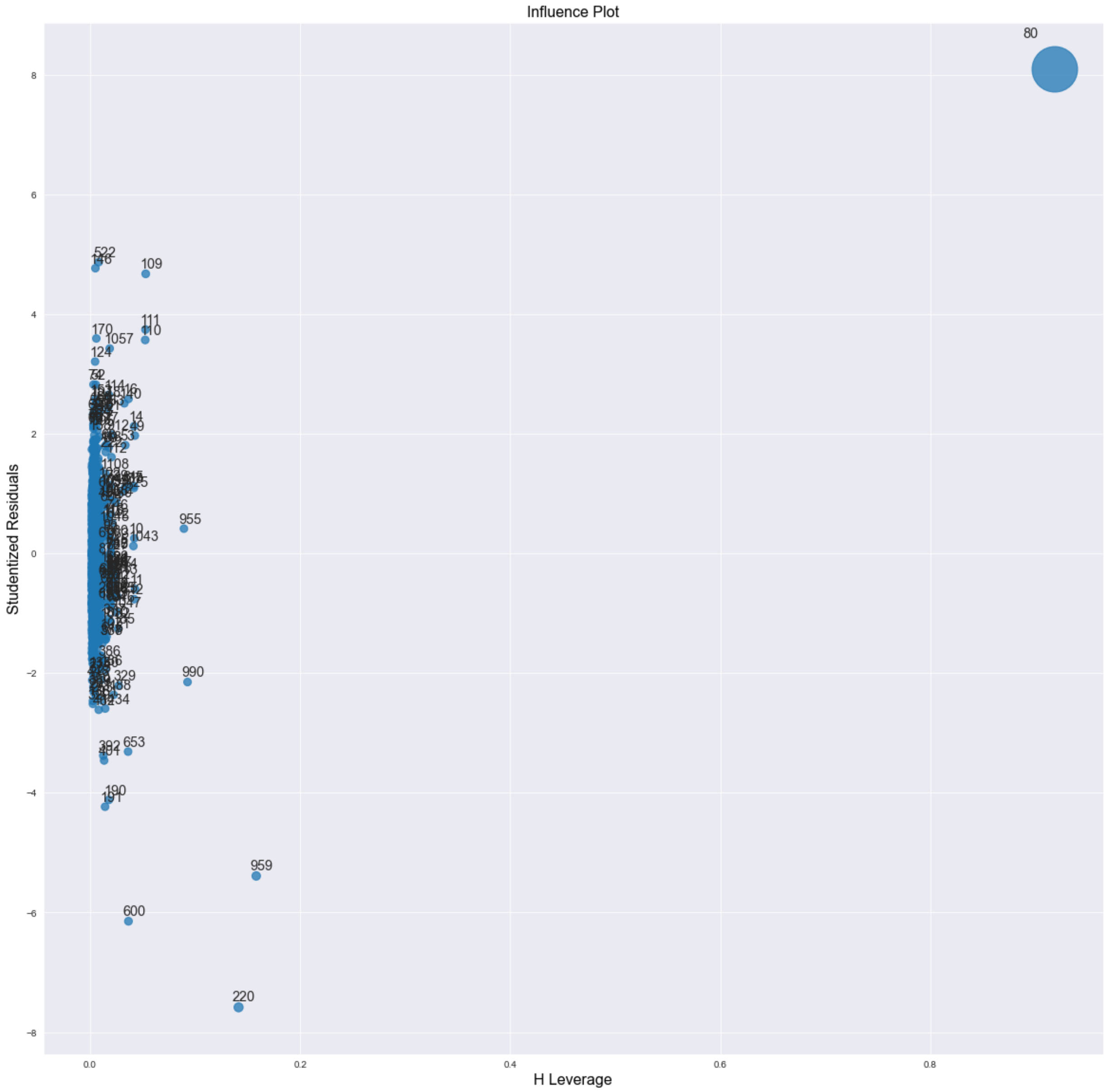
Out[42]:
```
array([7.20019747e-03, 3.91974773e-03, 5.41359424e-03, ...,
       9.01816675e-07, 6.97884848e-04, 1.08368316e-02])
```

In [60]:
```
# Plotting the influencers using the stem plot
# fig=plt.figure(figsize=(20,7))
# plt.stem(np.arange(len(toyo4)),np.round(c,3))
# plt.xlabel('Row Index')
# plt.ylabel('Cooks Distance')
# plt.show()
```

In [61]:
```
np.argmax(c) , np.max(c)
```

Out[61]:
```
(78, 78.32370087585271)
```

In [63]:
```
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff valu
fig,ax=plt.subplots(figsize=(20,20))
fig=influence_plot(linear_model,ax = ax)
```



In [65]:
```
### Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of dat
k=toyota.shape[1]
n=toyota.shape[0]
leverage_cutoff = (3*(k+1))/n
leverage_cutoff
```

Out[65]: 0.020905923344947737

In [66]:
```
toyota[toyota.index.isin([80])]
```

Out[66]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
80	18950	25	20019	110.0	16000	5	5	100	1180

In [67]:
```
#### Improving the Model
#### Creating a copy of data so that original dataset is not affected
toyota_new=toyota.copy()
toyota_new
```

Out[67]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90.0	2000	3	5	210	1165
1	13750	23	72937	90.0	2000	3	5	210	1165
2	13950	24	41711	90.0	2000	3	5	210	1165
3	14950	26	48000	90.0	2000	3	5	210	1165
4	13750	30	38500	90.0	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1430	7500	69	20544	86.0	1300	3	5	69	1025
1431	10845	72	19000	86.0	1300	3	5	69	1015
1432	8500	71	17016	86.0	1300	3	5	69	1015
1433	7250	70	16916	86.0	1300	3	5	69	1015
1434	6950	76	1	110.0	1600	5	5	19	1114

1435 rows × 9 columns

In [68]:

```
#### Discard the data points which are influencers and reassign the row number (rese
toyota=toyota_new.drop(toyota_new.index[[80]],axis=0).reset_index(drop=True)
toyota
```

Out[68]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90.0	2000	3	5	210	1165
1	13750	23	72937	90.0	2000	3	5	210	1165
2	13950	24	41711	90.0	2000	3	5	210	1165
3	14950	26	48000	90.0	2000	3	5	210	1165
4	13750	30	38500	90.0	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1429	7500	69	20544	86.0	1300	3	5	69	1025
1430	10845	72	19000	86.0	1300	3	5	69	1015
1431	8500	71	17016	86.0	1300	3	5	69	1015
1432	7250	70	16916	86.0	1300	3	5	69	1015
1433	6950	76	1	110.0	1600	5	5	19	1114

1434 rows × 9 columns

In [69]:

```
# Model Deletion Diagnostics and Final Model
while np.max(c)>0.5 :
    model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyota).fit()
    (c,_)=model.get_influence().cooks_distance
    c
    np.argmax(c) , np.max(c)
    toyota=toyota.drop(toyota.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
    toyota
else:
    final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyota).fit(
```

```
    final_model.rsquared , final_model.aic
    print("Thus model accuracy is improved to",final_model.rsquared)
```

Thus model accuracy is improved to 0.713222160431267

In [70]:

```
if np.max(c)>0.5:
    model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyota).fit()
    (c,_)=model.get_influence().cooks_distance
    c
    np.argmax(c) , np.max(c)
    toyota=toyota.drop(toyo5.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
    toyota
elif np.max(c)<0.5:
    final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyota).fit(
    final_model.rsquared , final_model.aic
    print("Thus model accuracy is improved to",final_model.rsquared)
```

Thus model accuracy is improved to 0.713222160431267

In [71]:

```
final_model.rsquared
```

Out[71]: 0.713222160431267

In [72]:

```
toyota
```

Out[72]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90.0	2000	3	5	210	1165
1	13750	23	72937	90.0	2000	3	5	210	1165
2	13950	24	41711	90.0	2000	3	5	210	1165
3	14950	26	48000	90.0	2000	3	5	210	1165
4	13750	30	38500	90.0	2000	3	5	210	1170
...	...	...	...	...	...	...	...	...	...
1219	7500	69	20544	86.0	1300	3	5	69	1025
1220	10845	72	19000	86.0	1300	3	5	69	1015
1221	8500	71	17016	86.0	1300	3	5	69	1015
1222	7250	70	16916	86.0	1300	3	5	69	1015
1223	6950	76	1	110.0	1600	5	5	19	1114

1224 rows × 9 columns

Model Prediction

In [73]:

```
new_data=pd.DataFrame({'Age':12,"KM":40000,"HP":80,"CC":1300,"Doors":4,"Gears":5,"QT
new_data
```

Out[73]:

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

In [74]:
```
### Manual Prediction of Price
final_model.predict(new_data)
```

Out[74]:
```
0    14179.745752
dtype: float64
```

In [76]:
```
### Automatic Prediction of Price with 90.02% accurcy
pred_y=final_model.predict(toyota)
pred_y
```

Out[76]:
```
0       14393.781470
1       14004.607957
2       14375.716460
3       14087.060805
4       13865.522226
            ...
1219     9032.391847
1220     8714.671034
1221     8841.595453
1222     8940.266514
1223     9472.227013
Length: 1224, dtype: float64
```