# 1.Importing Necessary Libraries

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

# 2. Importing data

In [4]:
```
toyota = pd.read_csv('ToyotaCorolla.csv')
toyota
```

Out[4]:

| | Id | Model | Price | Age_08_04 | Mfg_Month | Mfg_Year | KM | Fuel_Type | HP | Met_Col |
|---|---|---|---|---|---|---|---|---|---|---|
| **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_Col |
|---|---|---|---|---|---|---|---|---|---|---|
| **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

# 3. Data Understanding

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

Out[5]:
```
(1436, 38)
```

In [6]:
```
toyota.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 [7]:
```
toyota.dtypes
```

Out[7]:
```
Id                    int64
Model                object
Price                 int64
Age_08_04             int64
Mfg_Month             int64
Mfg_Year              int64
KM                    int64
Fuel_Type            object
HP                    int64
Met_Color             int64
Color                object
Automatic             int64
cc                    int64
Doors                 int64
Cylinders             int64
Gears                 int64
Quarterly_Tax         int64
Weight                int64
Mfr_Guarantee         int64
BOVAG_Guarantee       int64
Guarantee_Period      int64
ABS                   int64
Airbag_1              int64
Airbag_2              int64
Airco                 int64
Automatic_airco       int64
Boardcomputer         int64
CD_Player             int64
Central_Lock          int64
Powered_Windows       int64
Power_Steering        int64
Radio                 int64
Mistlamps             int64
Sport_Model           int64
Backseat_Divider      int64
Metallic_Rim          int64
Radio_cassette        int64
Tow_Bar               int64
dtype: object
```

# 3.Data Preparation

In [8]:
```
toyota_1=pd.concat([toyota.iloc[:,2:4],toyota.iloc[:,6:7],toyota.iloc[:,8:9],toyo
toyota_1
```

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 [10]:
```
toyota_2=toyota_1.rename({'Age_08_04':'Age','cc':'CC','Quarterly_Tax':'QT'},axis=
toyota_2
```

Out[10]:

| | 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 [11]:
```
toyota_2[toyota_2.duplicated()]
```

Out[11]:

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

In [12]:
```
toyota_3=toyota_2.drop_duplicates().reset_index(drop=True)
toyota_3
```

Out[12]:

| | 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 [13]:
```
toyota_3.describe()
```

Out[13]:

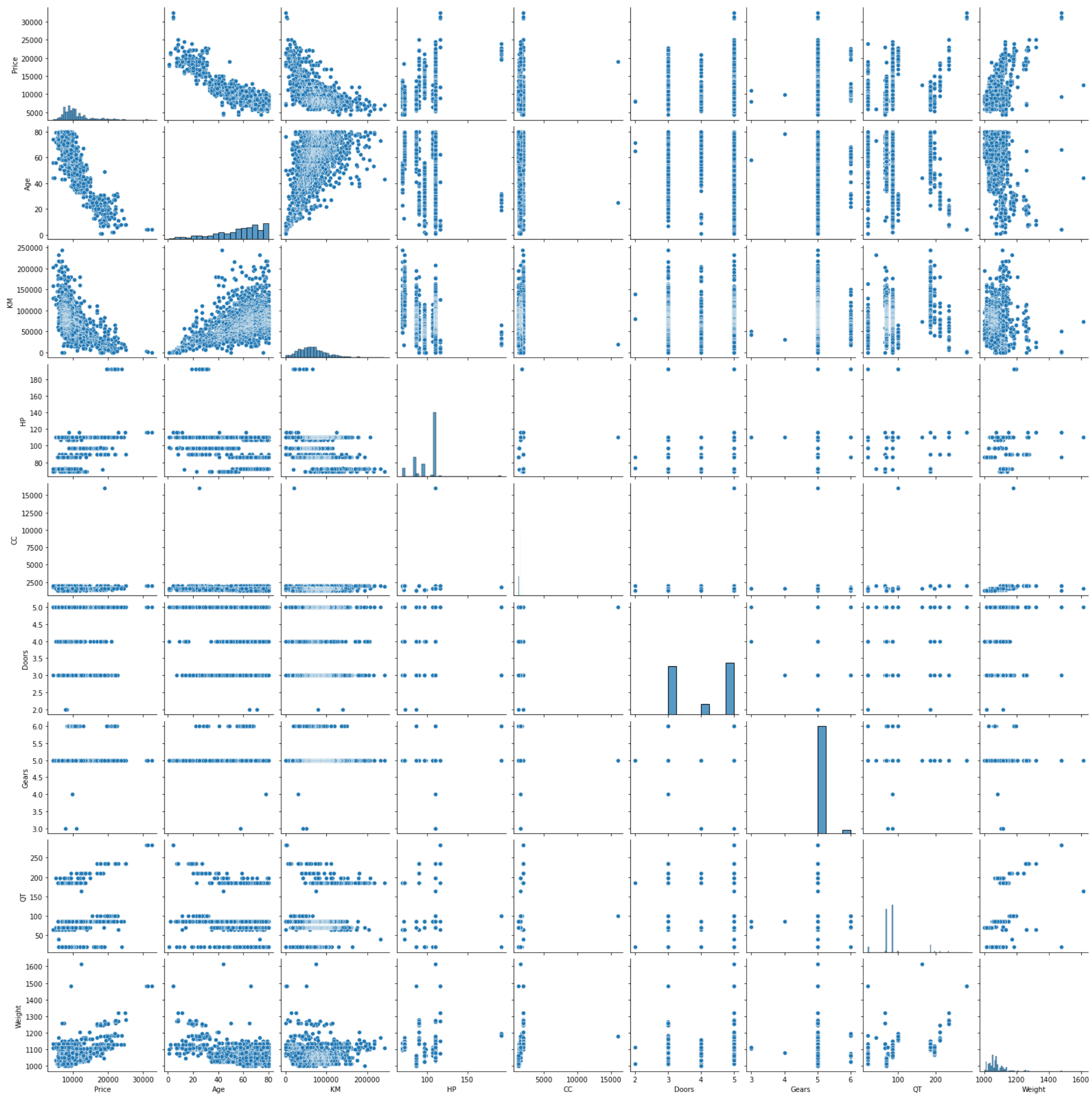
| | Price | Age | KM | HP | CC | Doors | G |
|---|---|---|---|---|---|---|---|
| **count** | 1435.000000 | 1435.000000 | 1435.000000 | 1435.000000 | 1435.000000 | 1435.000000 | 1435.00 |
| **mean** | 10720.915679 | 55.980488 | 68571.782578 | 101.491986 | 1576.560976 | 4.032753 | 5.02 |
| **std** | 3608.732978 | 18.563312 | 37491.094553 | 14.981408 | 424.387533 | 0.952667 | 0.18 |
| **min** | 4350.000000 | 1.000000 | 1.000000 | 69.000000 | 1300.000000 | 2.000000 | 3.00 |
| **25%** | 8450.000000 | 44.000000 | 43000.000000 | 90.000000 | 1400.000000 | 3.000000 | 5.00 |
| **50%** | 9900.000000 | 61.000000 | 63451.000000 | 110.000000 | 1600.000000 | 4.000000 | 5.00 |
| **75%** | 11950.000000 | 70.000000 | 87041.500000 | 110.000000 | 1600.000000 | 5.000000 | 5.00 |
| **max** | 32500.000000 | 80.000000 | 243000.000000 | 192.000000 | 16000.000000 | 5.000000 | 6.00 |

# 4.Check whether the assumptions is matching or not

## 4.1. Linearity Check

- By using Scatter plot/Paiplot

In [14]:
```
sns.pairplot(toyota_3)
plt.show()
```



## Observation - Linearity check is failed.

## 4. 2. No Multicollinearity

By using,

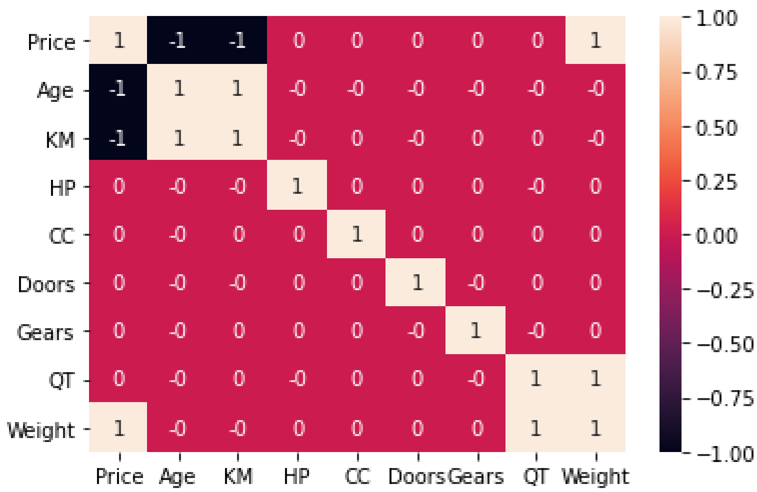
Correlation Matrix (or), Variance Inflation Factor(VIF) we can check it.

In [15]:
```
corr_matrix = toyota_3.corr().round()
corr_matrix
```

Out[15]:

| | Price | Age | KM | HP | CC | Doors | Gears | QT | Weight |
|---|---|---|---|---|---|---|---|---|---|
| **Price** | 1.0 | -1.0 | -1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **Age** | -1.0 | 1.0 | 1.0 | -0.0 | -0.0 | -0.0 | -0.0 | -0.0 | -0.0 |
| **KM** | -1.0 | 1.0 | 1.0 | -0.0 | 0.0 | -0.0 | 0.0 | 0.0 | -0.0 |
| **HP** | 0.0 | -0.0 | -0.0 | 1.0 | 0.0 | 0.0 | 0.0 | -0.0 | 0.0 |
| **CC** | 0.0 | -0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **Doors** | 0.0 | -0.0 | -0.0 | 0.0 | 0.0 | 1.0 | -0.0 | 0.0 | 0.0 |
| **Gears** | 0.0 | -0.0 | 0.0 | 0.0 | 0.0 | -0.0 | 1.0 | -0.0 | 0.0 |
| **QT** | 0.0 | -0.0 | 0.0 | -0.0 | 0.0 | 0.0 | -0.0 | 1.0 | 1.0 |
| **Weight** | 1.0 | -0.0 | -0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |

In [17]:
```
sns.heatmap(data= corr_matrix,annot=True)
plt.show()
```




## 4.3. No AutoRegression

It is satisfied.

## 4.4. Homoscadascity check

This can be done after model training.

## 4.5. Zero Residual Mean

This also can be checked after model training.

# 5.Model Building

In [18]:
```
import statsmodels.formula.api as smf
```

In [19]:
```
model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyota_3).fit()
```

In [20]:
```
model.params
```

Out[20]:
```
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 [21]:
```
model.tvalues , np.round(model.pvalues,5)
```

Out[21]:
```
(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,
 Intercept    0.00011
 Age          0.00000
 KM           0.00000
 HP           0.00000
 CC           0.18824
 Doors        0.98164
 Gears        0.00245
 QT           0.00329
 Weight       0.00000
 dtype: float64)
```

In [22]:
```
model.rsquared , model.rsquared_adj   # Model accuracy is 86.17%
```

Out[22]:
```
(0.8625200256946999, 0.8617487495415145)
```

# # Build SLR and MLR models for insignificant variables 'CC' and 'Doors'

## Also find their tvalues and pvalues

In [23]:
```
slr_c=smf.ols('Price~CC',data=toyota_3).fit()
slr_c.tvalues , slr_c.pvalues # CC has significant pvalue
```

Out[23]:
```
(Intercept    24.879592
 CC            4.745039
 dtype: float64,
 Intercept    7.236022e-114
 CC            2.292856e-06
 dtype: float64)
```

In [24]:
```
slr_d=smf.ols('Price~Doors',data=toyota_3).fit()
slr_d.tvalues , slr_d.pvalues # Doors has significant pvalue
```

Out[24]:
```
(Intercept    19.421546
 Doors         7.070520
 dtype: float64,
 Intercept    8.976407e-75
 Doors        2.404166e-12
 dtype: float64)
```

In [25]:
```
mlr_cd=smf.ols('Price~CC+Doors',data=toyota_3).fit()
mlr_cd.tvalues , mlr_cd.pvalues # CC & Doors have significant pvalue
```

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)
```

# 6.Model Validation Techniques

### Two Techniques:

- 1. Collinearity Check &
- 2. Residual Analysis

In [26]:
```
# 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=toyota_3).fit().rsquare
vif_age=1/(1-rsq_age)

rsq_KM=smf.ols('KM~Age+HP+CC+Doors+Gears+QT+Weight',data=toyota_3).fit().rsquared
vif_KM=1/(1-rsq_KM)

rsq_HP=smf.ols('HP~Age+KM+CC+Doors+Gears+QT+Weight',data=toyota_3).fit().rsquared
vif_HP=1/(1-rsq_HP)

rsq_CC=smf.ols('CC~Age+KM+HP+Doors+Gears+QT+Weight',data=toyota_3).fit().rsquared
vif_CC=1/(1-rsq_CC)

rsq_DR=smf.ols('Doors~Age+KM+HP+CC+Gears+QT+Weight',data=toyota_3).fit().rsquared
vif_DR=1/(1-rsq_DR)

rsq_GR=smf.ols('Gears~Age+KM+HP+CC+Doors+QT+Weight',data=toyota_3).fit().rsquared
vif_GR=1/(1-rsq_GR)

rsq_QT=smf.ols('QT~Age+KM+HP+CC+Doors+Gears+Weight',data=toyota_3).fit().rsquared
vif_QT=1/(1-rsq_QT)

rsq_WT=smf.ols('Weight~Age+KM+HP+CC+Doors+Gears+QT',data=toyota_3).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[26]:

| | 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 |

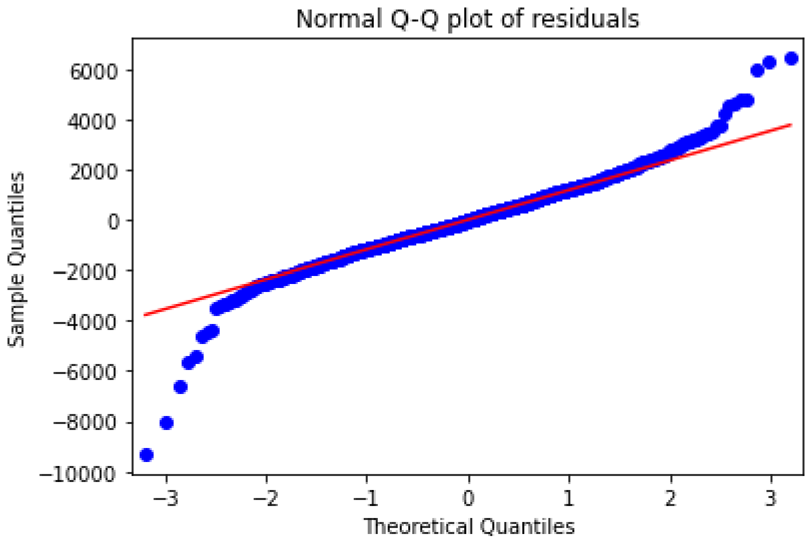
**observations- None variable has VIF>20, No Collinearity, so consider all varaibles in Regression equation**

## 2) Residual Analysis

Test for Normality of Residuals (Q-Q Plot) using residual model (model.resid)

In [27]:
```
import statsmodels.api as sm
```

In [28]:
```
sm.qqplot(model.resid,line='q') # 'q' - A line is fit through the quartiles # lir
plt.title("Normal Q-Q plot of residuals")
plt.show()
```




In [30]:
```
list(np.where(model.resid>6000))  # outliar detection from above QQ plot of resic
```

Out[30]: [array([109, 146, 522], dtype=int64)]

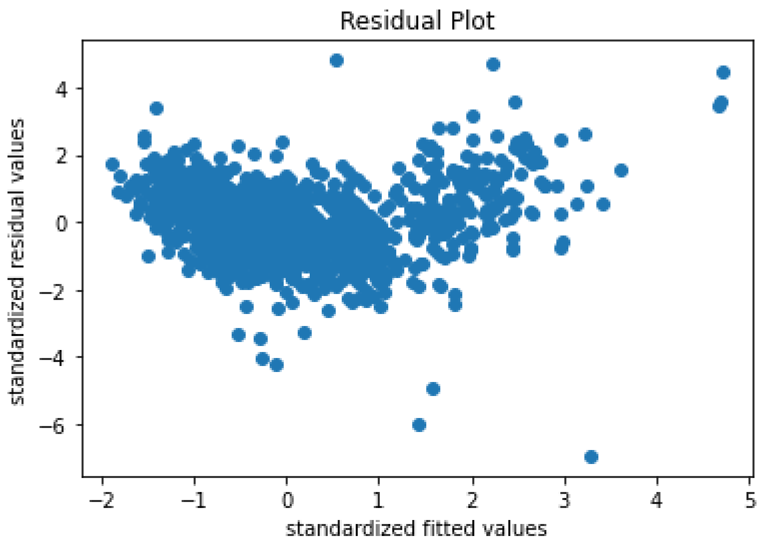
In [31]:
```
list(np.where(model.resid<-6000))
```

Out[31]: [array([220, 600, 959], dtype=int64)]

## Residual Plot for Homoscedasticity

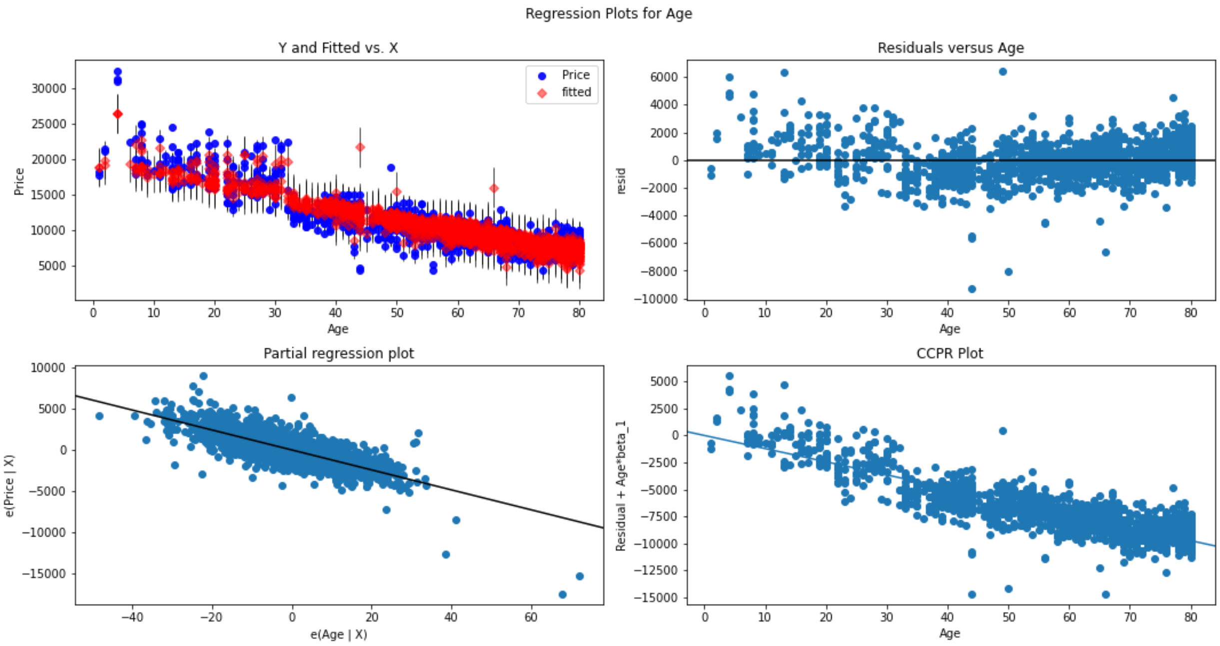
In [32]:
```
# Test for Homoscedasticity or Heteroscedasticity (plotting model's standardized

def standard_values(vals) :
    return (vals-vals.mean())/vals.std()  # User defined z = (x - mu)/sigma
```

In [33]:
```
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()
```

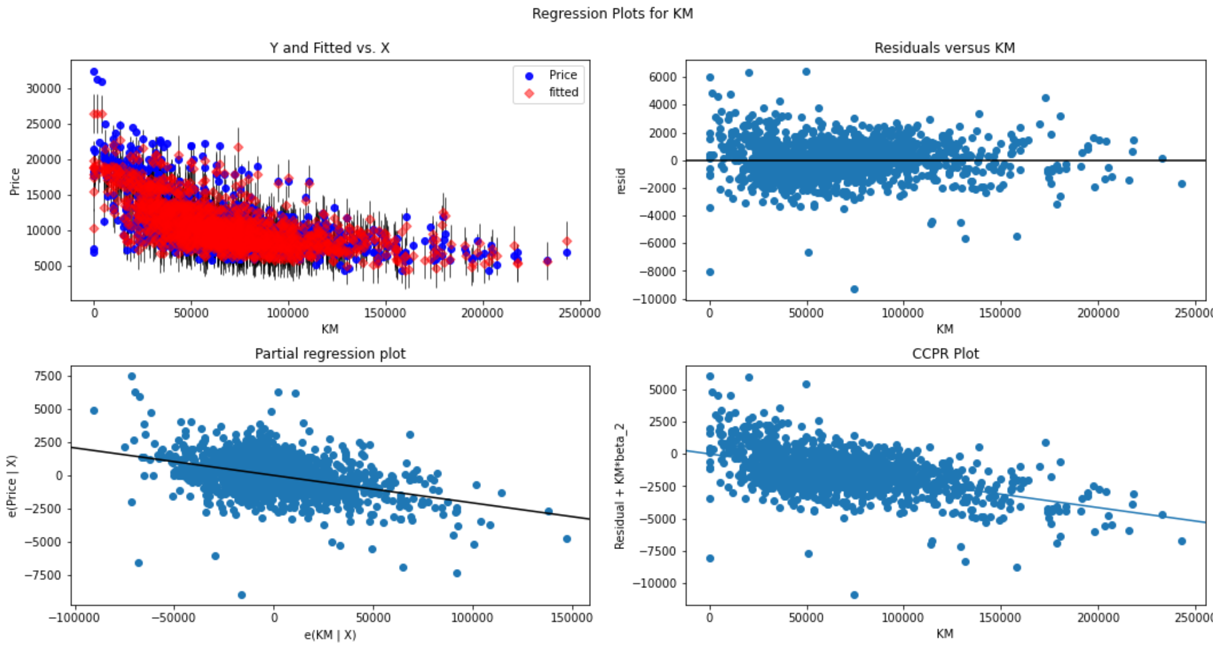



## Residual Vs Regressors

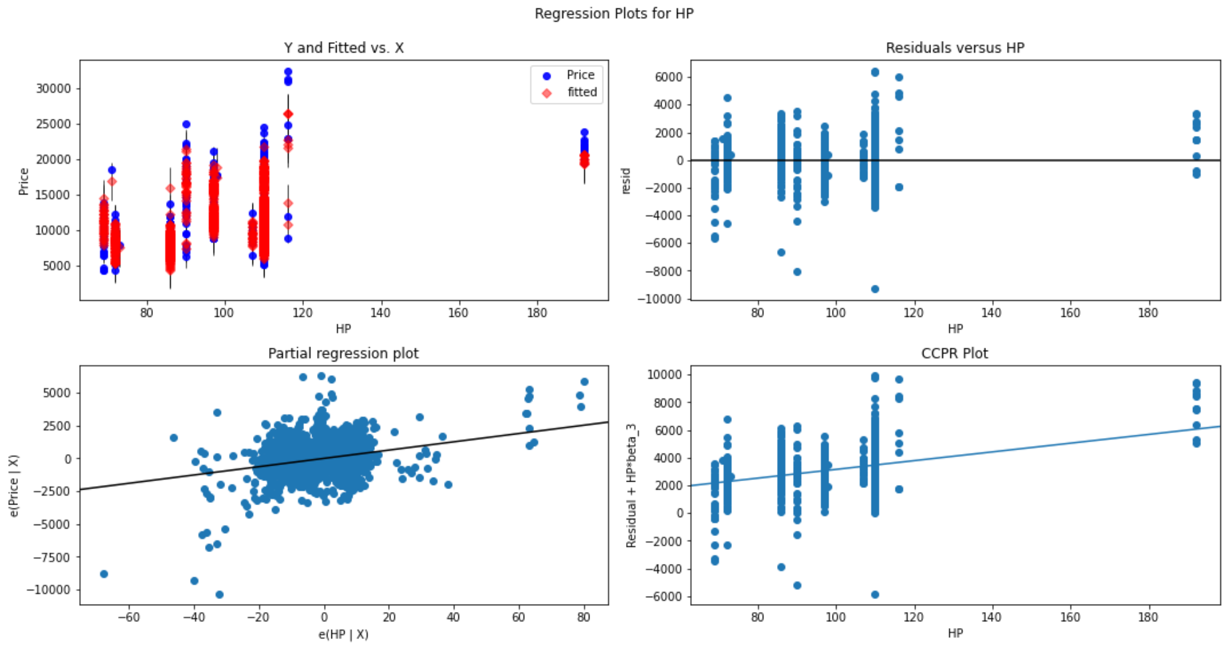
In [35]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Age',fig=fig)
plt.show()
```



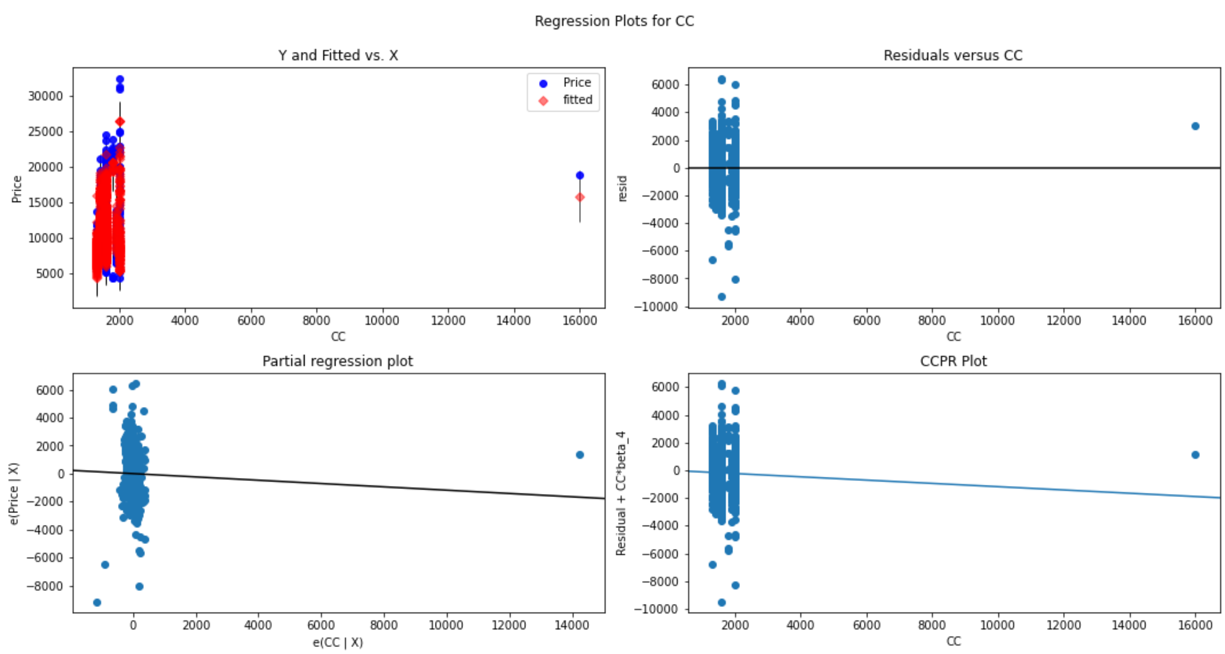
In [36]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'KM',fig=fig)
plt.show()
```



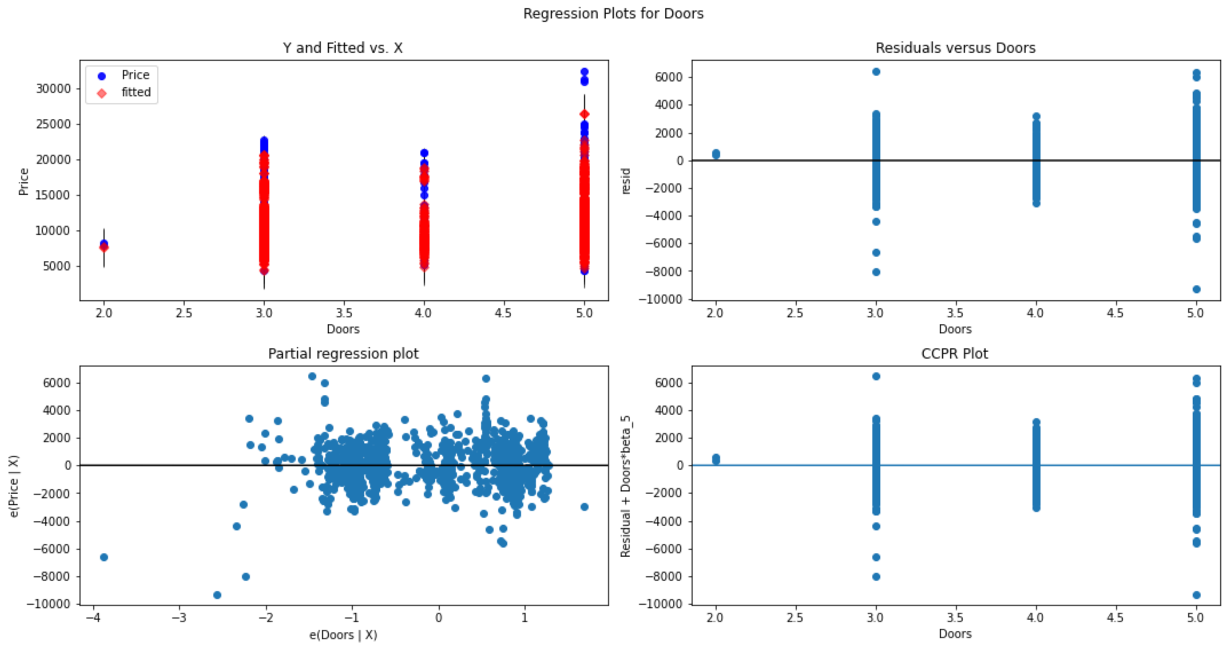
In [37]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'HP',fig=fig)
plt.show()
```



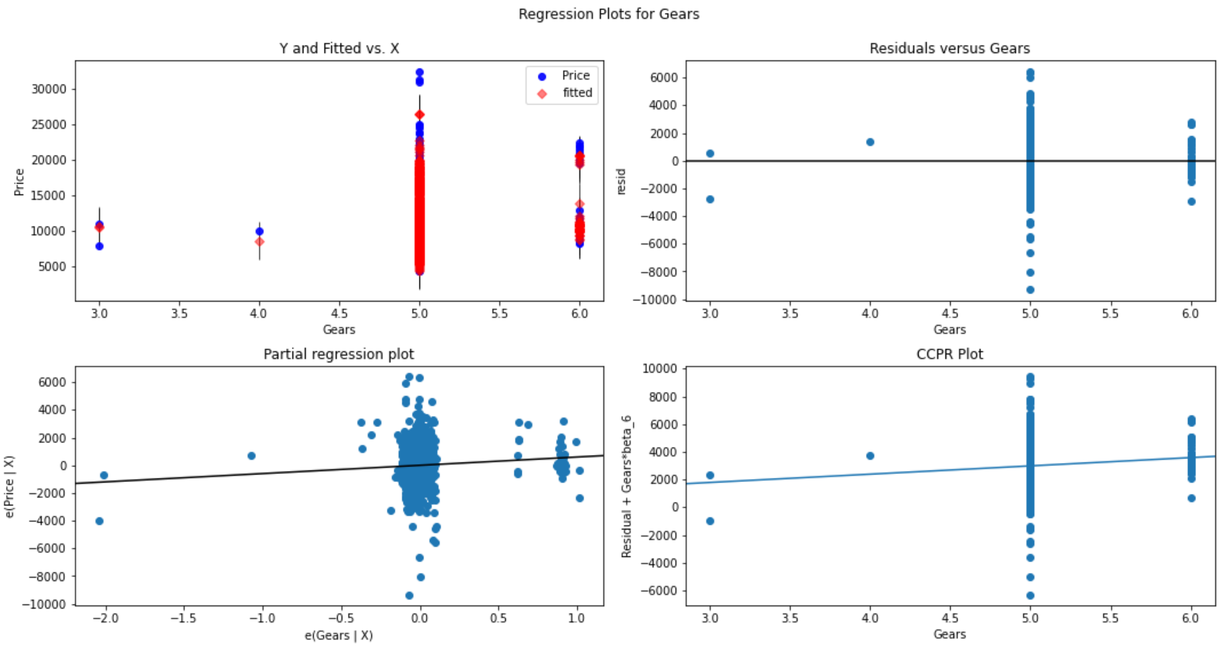

In [38]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'CC',fig=fig)
plt.show()
```



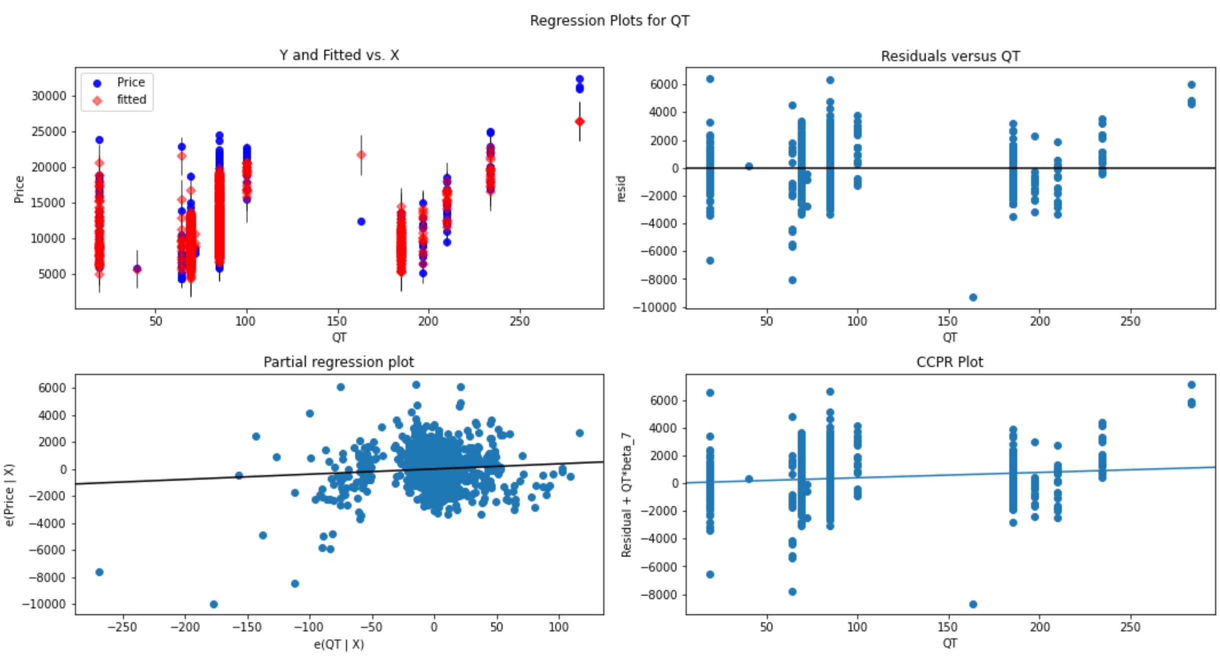
In [39]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Doors',fig=fig)
plt.show()
```



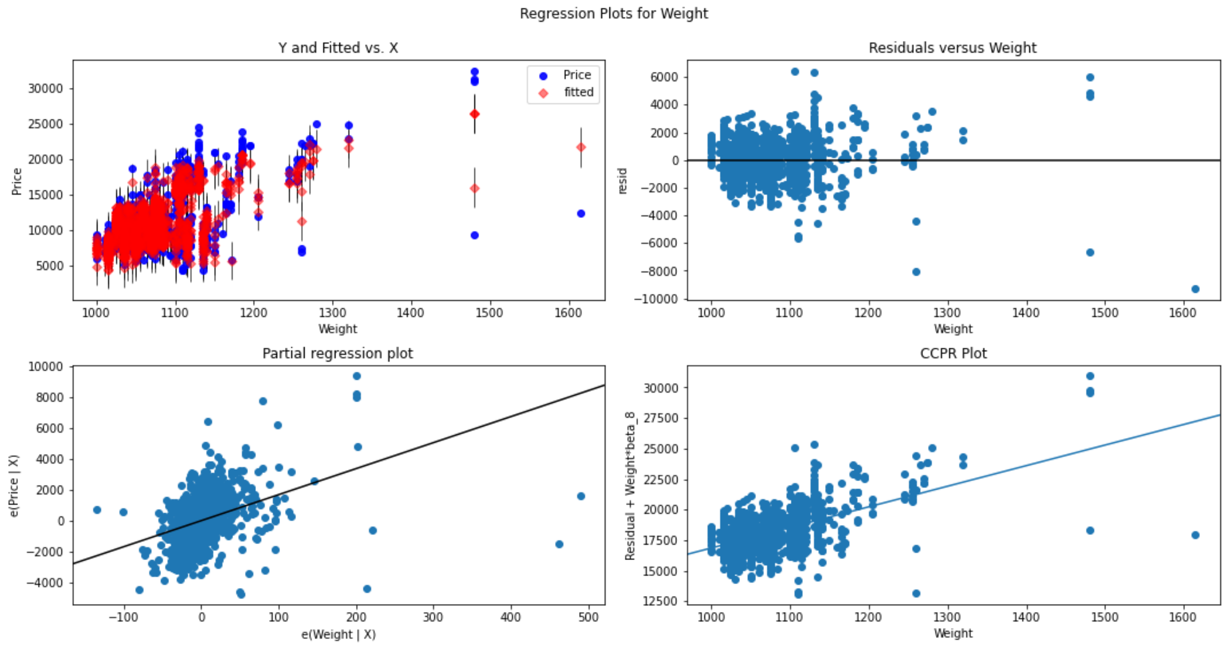
In [40]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Gears',fig=fig)
plt.show()
```




In [41]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'QT',fig=fig)
plt.show()
```



In [42]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Weight',fig=fig)
plt.show()
```




# 7.Model Deletion Diagnostics (checking Outliers or Influencers)

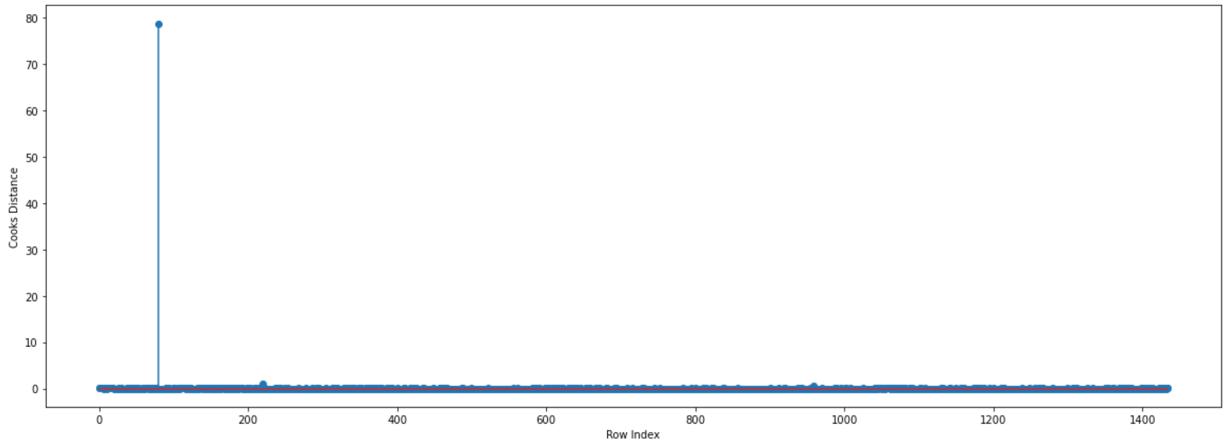
## Two Techniques :

- 1. Cook's Distance &
- 2. Leverage value

In [43]:
```
# 1. Cook's Distance: If Cook's distance > 1, then it's an outlier
# Get influencers using cook's distance
(c,_)=model.get_influence().cooks_distance
c
```

Out[43]:
```
array([7.22221054e-03, 3.94547973e-03, 5.44224039e-03, ...,
       8.04110550e-07, 6.99854767e-04, 1.08408002e-02])
```

In [45]:
```
# Plot the influencers using the stem plot
fig=plt.figure(figsize=(20,7))
plt.stem(np.arange(len(toyota_3)),np.round(c,3))
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```



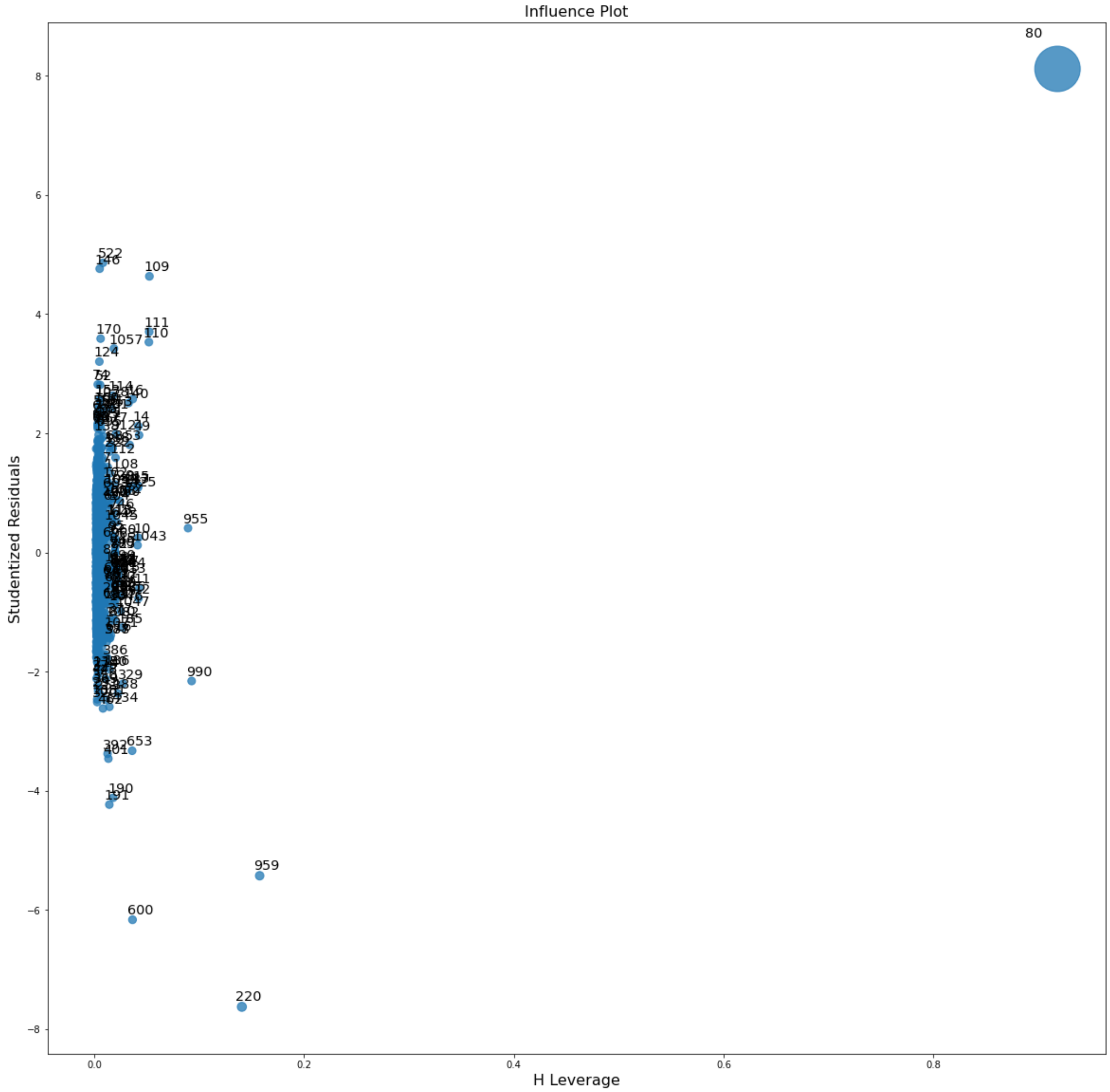

In [46]:
```
# Index and value of influencer where C>0.5
np.argmax(c) , np.max(c)
```

Out[46]: (80, 78.7295058224556)

# # 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are influencers

In [48]:
```
from statsmodels.graphics.regressionplots import influence_plot
```

In [49]:
```
fig,ax=plt.subplots(figsize=(20,20))
fig=influence_plot(model,ax = ax)
```



In [50]:
```
# Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of da
k=toyota_3.shape[1]
n=toyota_3.shape[0]
leverage_cutoff = (3*(k+1))/n
leverage_cutoff
```

Out[50]:
```
0.020905923344947737
```

**observations-From the above plot, it is evident that points beyond leverage cutoff value=0.020905 are the outliers**

In [51]:
```
toyota_3[toyota_3.index.isin([80])]
```

Out[51]:

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

# 8.Improving the Model

In [53]:
```
# Creating a copy of data so that original dataset is not affected
toyota_new=toyota_3.copy()
toyota_new
```

Out[53]:

| | 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 [54]:
```
# Discard the data points which are influencers and reassign the row number (rese
toyota_4=toyota_new.drop(toyota_new.index[[80]],axis=0).reset_index(drop=True)
toyota_4
```

Out[54]:

| | 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

# 9.Model Deletion Diagnostics and Final Model

In [55]:
```
while model.rsquared < 0.90:
    for c in [np.max(c)>0.5]:
        model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyota_4).f
        (c,_)=model.get_influence().cooks_distance
        c
        np.argmax(c) , np.max(c)
        toyota_4=toyota_4.drop(toyota_4.index[[np.argmax(c)]],axis=0).reset_index
        toyota_4
    else:
        final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyot
        final_model.rsquared , final_model.aic
        print("Thus model accuracy is improved to",final_model.rsquared)
```

```
Thus model accuracy is improved to 0.8765926307402282
Thus model accuracy is improved to 0.8839684606741538
Thus model accuracy is improved to 0.8882395145171204
Thus model accuracy is improved to 0.8902571486612915
Thus model accuracy is improved to 0.8909888960319987
Thus model accuracy is improved to 0.8922595280462808
Thus model accuracy is improved to 0.8933621011392296
Thus model accuracy is improved to 0.8947147371605555
Thus model accuracy is improved to 0.8955233405057648
Thus model accuracy is improved to 0.8930210061069088
Thus model accuracy is improved to 0.8939546425147169
Thus model accuracy is improved to 0.8954112430715817
Thus model accuracy is improved to 0.8960182592139027
Thus model accuracy is improved to 0.8968403506948497
Thus model accuracy is improved to 0.8964026771830705
Thus model accuracy is improved to 0.8958538146890626
Thus model accuracy is improved to 0.8953750500147551
Thus model accuracy is improved to 0.8949455651565241
Thus model accuracy is improved to 0.8960864004304144
Thus model accuracy is improved to 0.8955820765034092
Thus model accuracy is improved to 0.8930233902806168
Thus model accuracy is improved to 0.8903879563757863
Thus model accuracy is improved to 0.8895239558162493
Thus model accuracy is improved to 0.8898960234448476
Thus model accuracy is improved to 0.8903208318396924
Thus model accuracy is improved to 0.8908014686337988
Thus model accuracy is improved to 0.8901005575125875
Thus model accuracy is improved to 0.8894678831369645
Thus model accuracy is improved to 0.8894880027099143
Thus model accuracy is improved to 0.8900243177601598
Thus model accuracy is improved to 0.8894961653289413
Thus model accuracy is improved to 0.888830069020742
Thus model accuracy is improved to 0.8890577556184879
Thus model accuracy is improved to 0.8898790181431516
Thus model accuracy is improved to 0.8905555862707121
Thus model accuracy is improved to 0.8905494548555106
Thus model accuracy is improved to 0.890703571578653
Thus model accuracy is improved to 0.8909968812264217
Thus model accuracy is improved to 0.8912949497964373
Thus model accuracy is improved to 0.8918388895715855
Thus model accuracy is improved to 0.8926704315417882
Thus model accuracy is improved to 0.8928840759989136
Thus model accuracy is improved to 0.893349603994157
Thus model accuracy is improved to 0.8936856658122996
```

```
Thus model accuracy is improved to 0.8944751976424922
Thus model accuracy is improved to 0.8938395480435741
Thus model accuracy is improved to 0.8931993880518595
Thus model accuracy is improved to 0.8925409392511808
Thus model accuracy is improved to 0.8933274017072571
Thus model accuracy is improved to 0.8919970006989908
Thus model accuracy is improved to 0.8913049155203421
Thus model accuracy is improved to 0.891996770189864
Thus model accuracy is improved to 0.8925697842477825
Thus model accuracy is improved to 0.892733433674135
Thus model accuracy is improved to 0.893037366789838
Thus model accuracy is improved to 0.8936216177226137
Thus model accuracy is improved to 0.8943831156038025
Thus model accuracy is improved to 0.8946116437234872
Thus model accuracy is improved to 0.8947373476692217
Thus model accuracy is improved to 0.8950379522236931
Thus model accuracy is improved to 0.8948527054861178
Thus model accuracy is improved to 0.8953342154393777
Thus model accuracy is improved to 0.895598439180588
Thus model accuracy is improved to 0.8959887924407284
Thus model accuracy is improved to 0.8949134651663618
Thus model accuracy is improved to 0.8951494863024941
Thus model accuracy is improved to 0.8955005725113695
Thus model accuracy is improved to 0.8957248584395653
Thus model accuracy is improved to 0.8954298292582191
Thus model accuracy is improved to 0.8953618506400355
Thus model accuracy is improved to 0.8956028020870667
Thus model accuracy is improved to 0.8959330397949153
Thus model accuracy is improved to 0.8962516478679261
Thus model accuracy is improved to 0.896677152907062
Thus model accuracy is improved to 0.8968748575434936
Thus model accuracy is improved to 0.8967502968710409
Thus model accuracy is improved to 0.8964643118227555
Thus model accuracy is improved to 0.8968869273251255
Thus model accuracy is improved to 0.8965702611797324
Thus model accuracy is improved to 0.8964656489827649
Thus model accuracy is improved to 0.8967541657126812
Thus model accuracy is improved to 0.8970295712331846
Thus model accuracy is improved to 0.8970512860226795
Thus model accuracy is improved to 0.8973242631620858
Thus model accuracy is improved to 0.897606403630622
Thus model accuracy is improved to 0.8965027704321634
Thus model accuracy is improved to 0.8969285366970444
Thus model accuracy is improved to 0.8970144437559693
Thus model accuracy is improved to 0.897464618358938
Thus model accuracy is improved to 0.8977004418350513
Thus model accuracy is improved to 0.8979562106578318
Thus model accuracy is improved to 0.8980173910665002
Thus model accuracy is improved to 0.8982290469246595
Thus model accuracy is improved to 0.8981603329614346
Thus model accuracy is improved to 0.8984954051008237
Thus model accuracy is improved to 0.8988264391266801
Thus model accuracy is improved to 0.8988386546358322
Thus model accuracy is improved to 0.8990728046493341
Thus model accuracy is improved to 0.8992884343762314
Thus model accuracy is improved to 0.8995264042658645
Thus model accuracy is improved to 0.8998346697166659
```

```
Thus model accuracy is improved to 0.8999704768778107
Thus model accuracy is improved to 0.9002238270483124
Thus model accuracy is improved to 0.9003762532318559
```

In [56]:
```
final_model.rsquared # Model Accuracy is increased to 90.02%
```

Out[56]:
```
0.9003762532318559
```

In [57]:
```
toyota_4
```

Out[57]:

| | Price | Age | KM | HP | CC | Doors | Gears | QT | Weight |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 13750 | 23 | 72937 | 90 | 2000 | 3 | 5 | 210 | 1165 |
| **1** | 14950 | 26 | 48000 | 90 | 2000 | 3 | 5 | 210 | 1165 |
| **2** | 13750 | 30 | 38500 | 90 | 2000 | 3 | 5 | 210 | 1170 |
| **3** | 12950 | 32 | 61000 | 90 | 2000 | 3 | 5 | 210 | 1170 |
| **4** | 16900 | 27 | 94612 | 90 | 2000 | 3 | 5 | 210 | 1245 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1325** | 8450 | 80 | 23000 | 86 | 1300 | 3 | 5 | 69 | 1015 |
| **1326** | 7500 | 69 | 20544 | 86 | 1300 | 3 | 5 | 69 | 1025 |
| **1327** | 10845 | 72 | 19000 | 86 | 1300 | 3 | 5 | 69 | 1015 |
| **1328** | 8500 | 71 | 17016 | 86 | 1300 | 3 | 5 | 69 | 1015 |
| **1329** | 7250 | 70 | 16916 | 86 | 1300 | 3 | 5 | 69 | 1015 |

1330 rows × 9 columns

# 10.Model Predictions

In [58]:
```
# say New data for prediction is
new_data=pd.DataFrame({'Age':12,"KM":40000,"HP":80,"CC":1300,"Doors":4,"Gears":5,
new_data
```

Out[58]:

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

In [59]:
```
# Manual Prediction of Price
final_model.predict(new_data)
```

Out[59]:
```
0    14398.815471
dtype: float64
```

In [61]:
```
# Automatic Prediction of Price with 90.02% accurcy
pred_y=final_model.predict(toyota_4)
pred_y
```

Out[61]:
```
0       15354.362106
1       15415.237858
2       15314.008799
3       14749.534289
4       17544.273936
            ...
1325     7607.457292
1326     9206.037539
1327     8535.375501
1328     8674.315161
1329     8784.118985
Length: 1330, dtype: float64
```