In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
from statsmodels.graphics.regressionplots import influence_plot
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv("ToyotaCorolla.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central_Lock
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	10	2002	46986	Diesel	90	1	...	1
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	10	2002	72937	Diesel	90	1	...	1
2	3	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	9	2002	41711	Diesel	90	1	...	0
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	...	1

5 rows × 38 columns

In [4]:
```
df.tail()
```

Out[4]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central_L

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central_L
1431	1438	TOYOTA Corolla 1.3 16V HATCHB G6 2/3-Doors	7500	69	12	1998	20544	Petrol	86	1	...	
1432	1439	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	10845	72	9	1998	19000	Petrol	86	0	...	
1433	1440	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	8500	71	10	1998	17016	Petrol	86	0	...	
1434	1441	TOYOTA Corolla 1.3 16V HATCHB LINEA TERRA 2/3-...	7250	70	11	1998	16916	Petrol	86	1	...	
1435	1442	TOYOTA Corolla 1.6 LB LINEA TERRA 4/5-Doors	6950	76	5	1998	1	Petrol	110	0	...	

5 rows × 38 columns

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

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

In [6]:
```
df.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
                       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
```

Checking Null Values

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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
                   0
```

```
Radio_cassette      0
Tow_Bar             0
dtype: int64
```

Checking Duplicate Values

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

Out[8]:

Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central_Lock	Powered

0 rows × 38 columns

Dropping Columns which are not used for Predictions

In [9]:
```
df.drop(["Id","Model","Mfg_Month","Cylinders","Mfg_Year","Fuel_Type","Mfr_Guarantee","BOVA
```

In [10]:
```
df
```

Out[10]:

	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

Renaming Columns name

In [11]:
```
df1=df.rename({"Age_08_04":"Age","Quarterly_Tax":"QT","cc":"CC"},axis=1)
df1
```

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

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
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

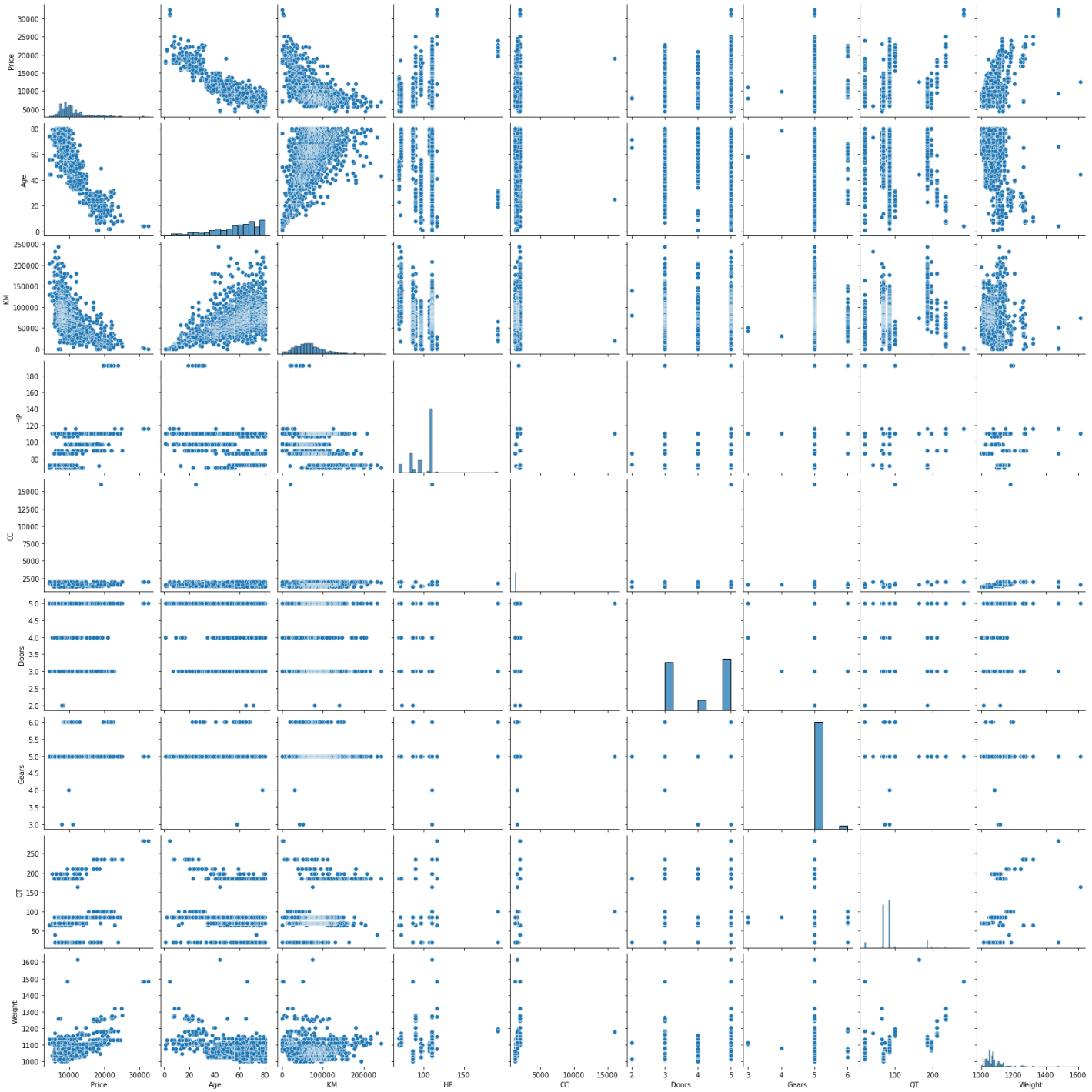
Checking Co-linearity

In [12]:
```
df1.corr()
```

Out[12]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
Price	1.000000	-0.876590	-0.569960	0.314990	0.126389	0.185326	0.063104	0.219197	0.581198
Age	-0.876590	1.000000	0.505672	-0.156622	-0.098084	-0.148359	-0.005364	-0.198431	-0.470253
KM	-0.569960	0.505672	1.000000	-0.333538	0.102683	-0.036197	0.015023	0.278165	-0.028598
HP	0.314990	-0.156622	-0.333538	1.000000	0.035856	0.092424	0.209477	-0.298432	0.089614
CC	0.126389	-0.098084	0.102683	0.035856	1.000000	0.079903	0.014629	0.306996	0.335637
Doors	0.185326	-0.148359	-0.036197	0.092424	0.079903	1.000000	-0.160141	0.109363	0.302618
Gears	0.063104	-0.005364	0.015023	0.209477	0.014629	-0.160141	1.000000	-0.005452	0.020613
QT	0.219197	-0.198431	0.278165	-0.298432	0.306996	0.109363	-0.005452	1.000000	0.626134
Weight	0.581198	-0.470253	-0.028598	0.089614	0.335637	0.302618	0.020613	0.626134	1.000000

In [13]:
```
sns.pairplot(df1)
plt.show()
```

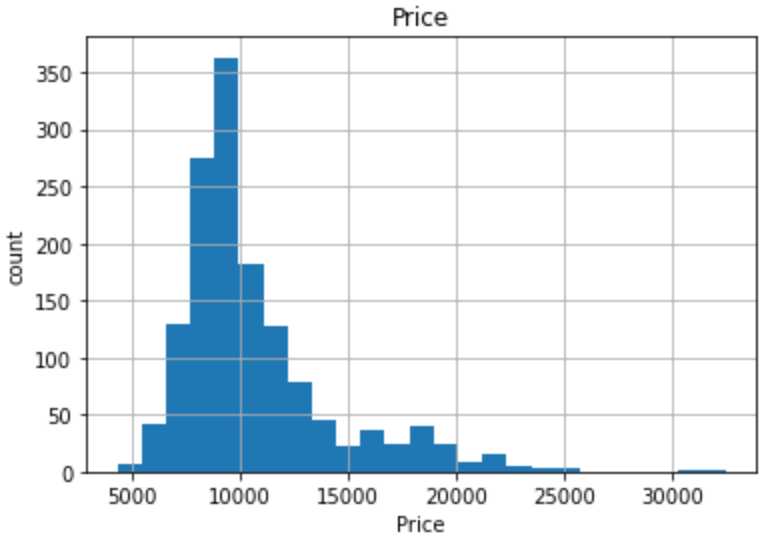


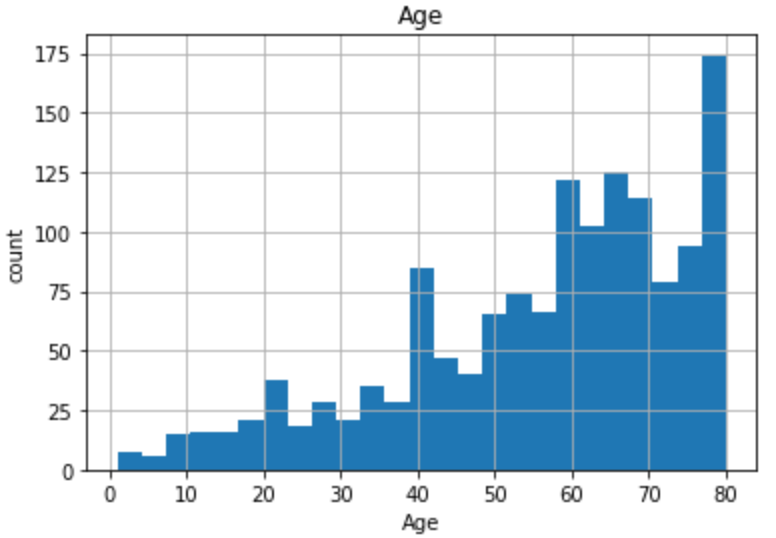
Observation

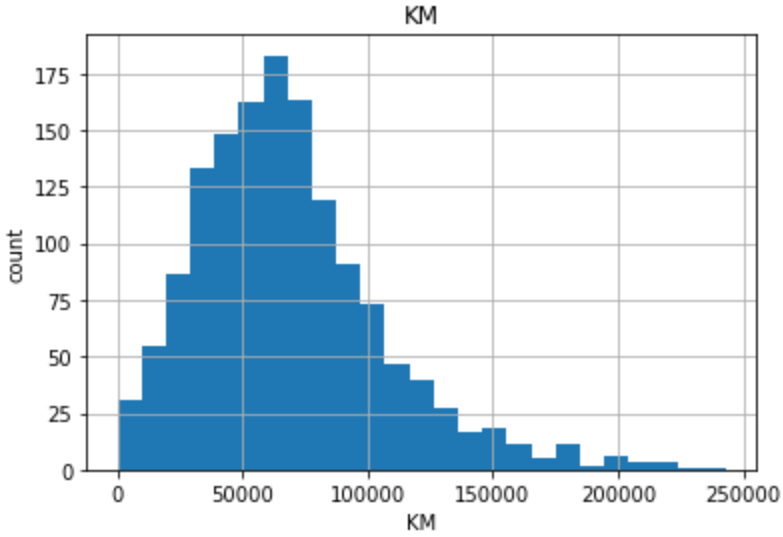
As we can see there is not correlation between features but between Ages and KM having negatively co-relation

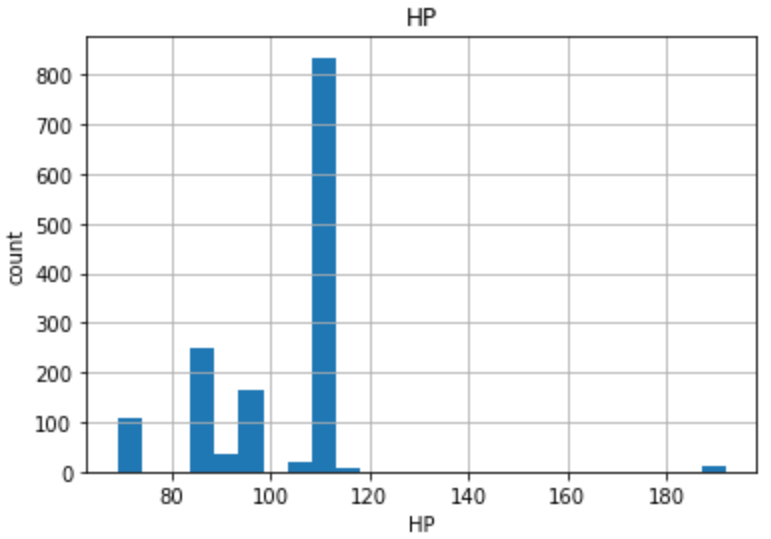
In [14]:

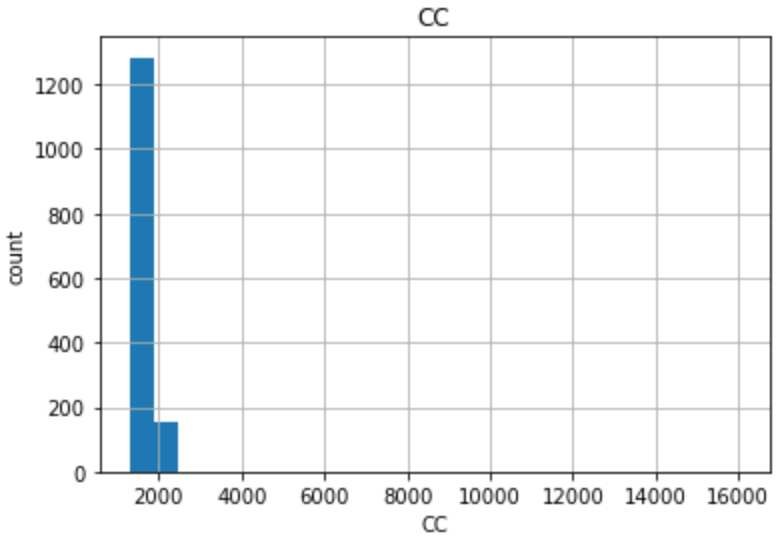
```
for feature in df1:
    data=df1.copy()
    data[feature].hist(bins=25)
    plt.xlabel(feature)
    plt.ylabel("count")
    plt.title(feature)
    plt.show()
```

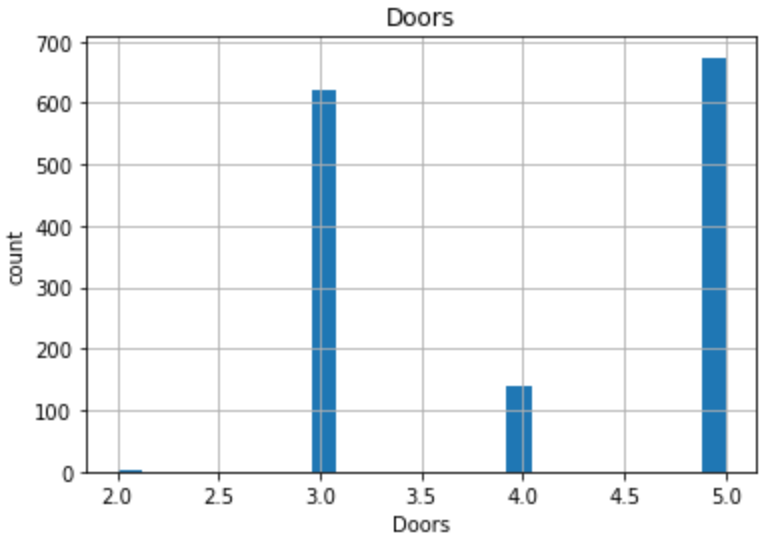


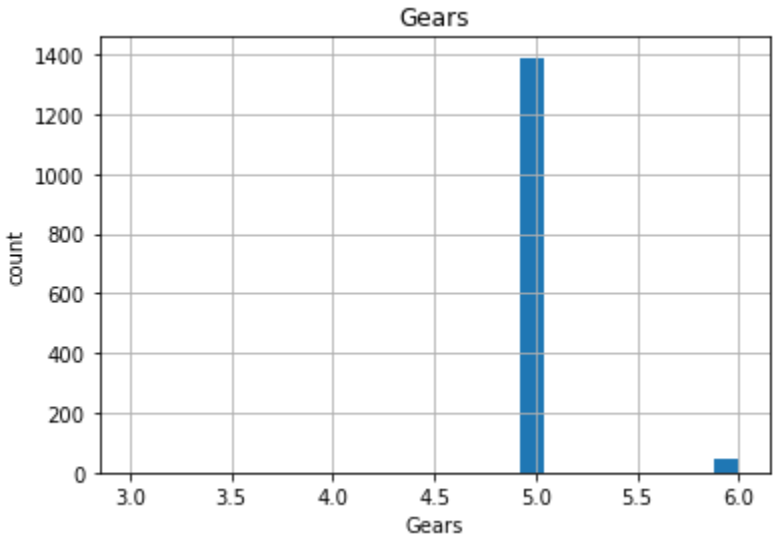


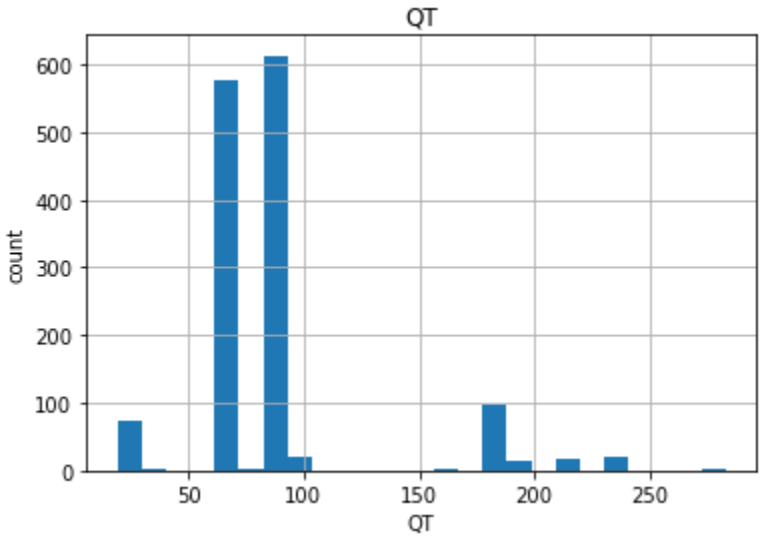


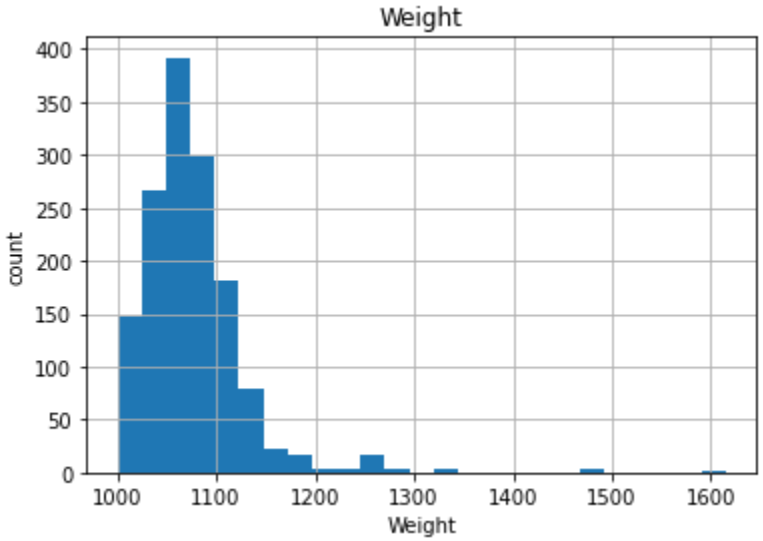








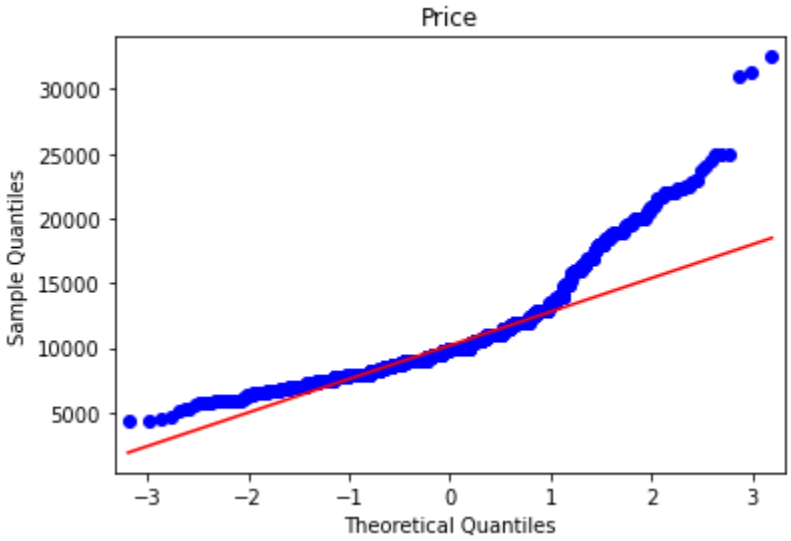


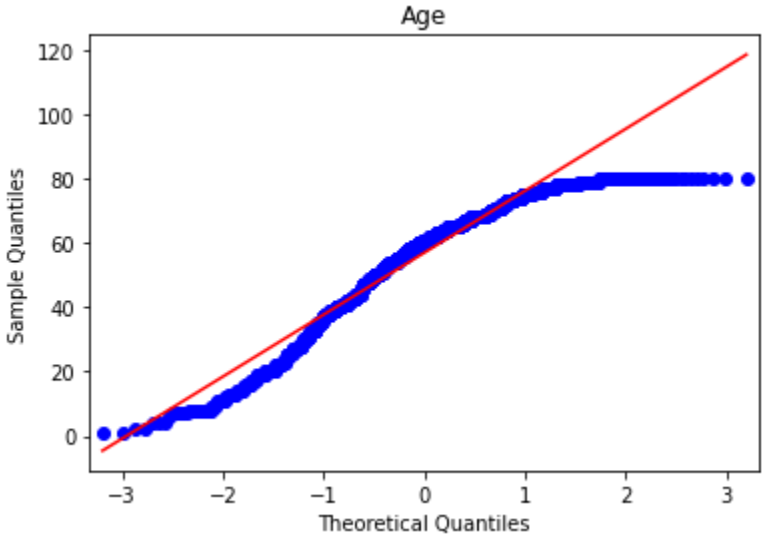


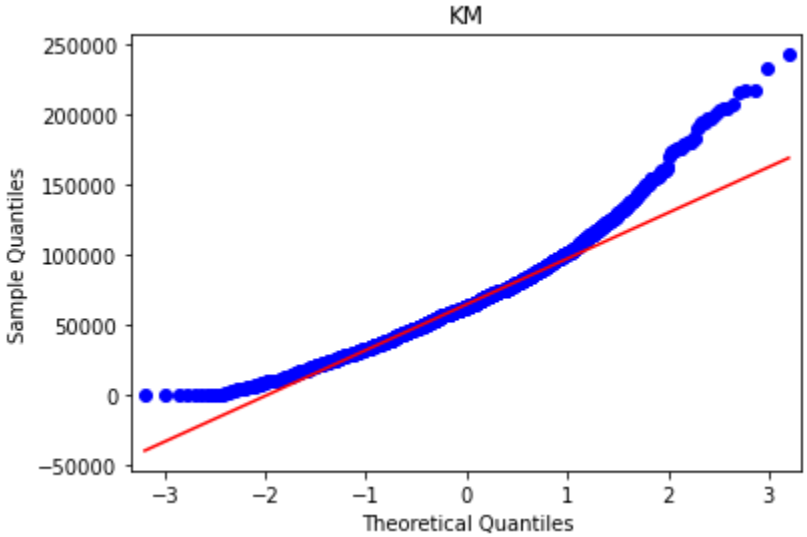
QQ-Plot for Raw Data

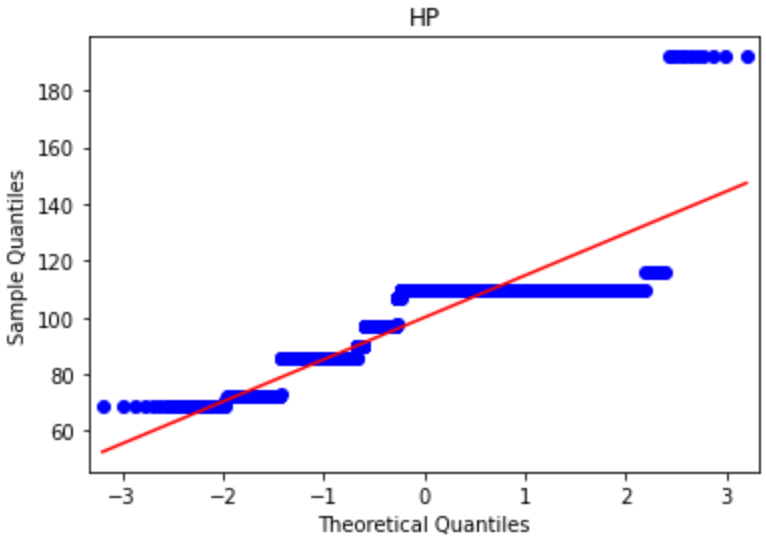
In [15]:

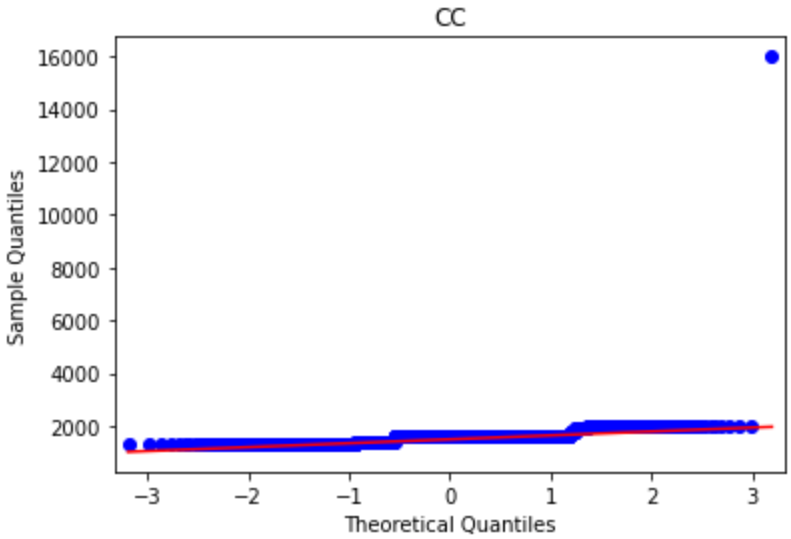
```
for feature in df1:
    data = df1.copy()
    sm.qqplot(data[feature],line='q')
    plt.title(feature)
    plt.show()
```

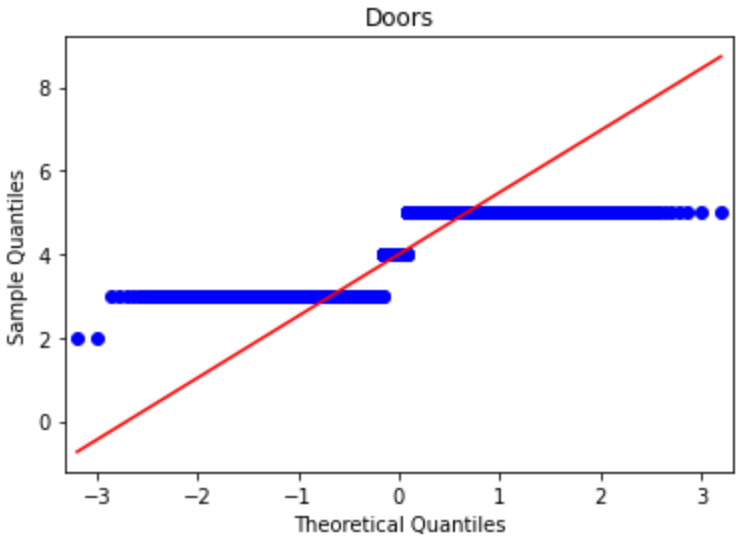


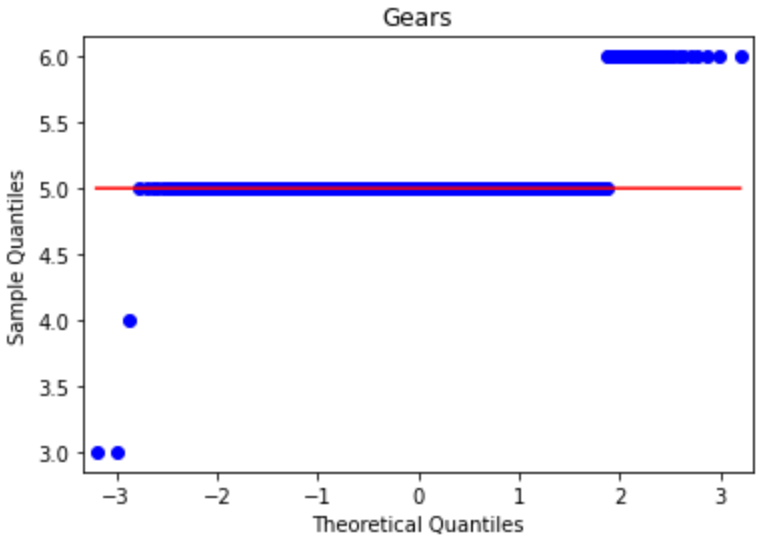


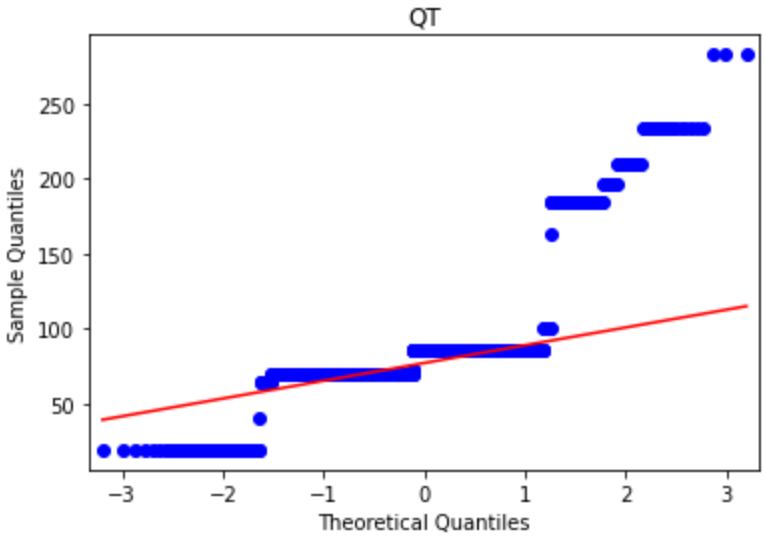


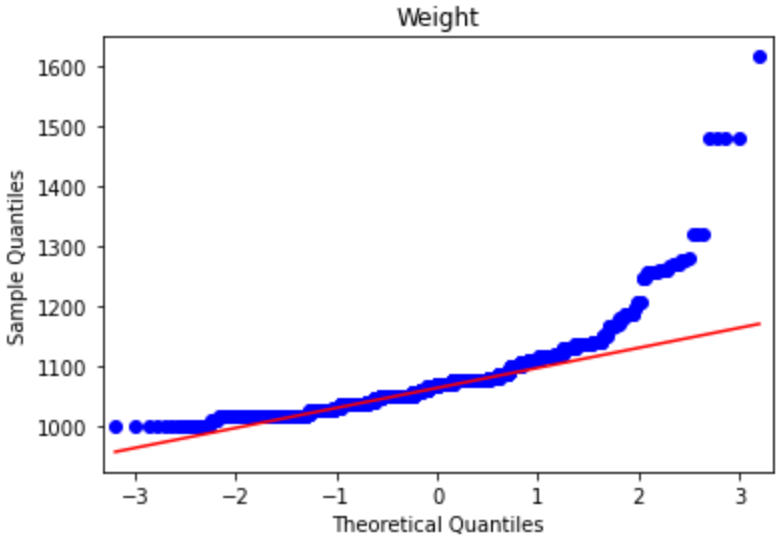








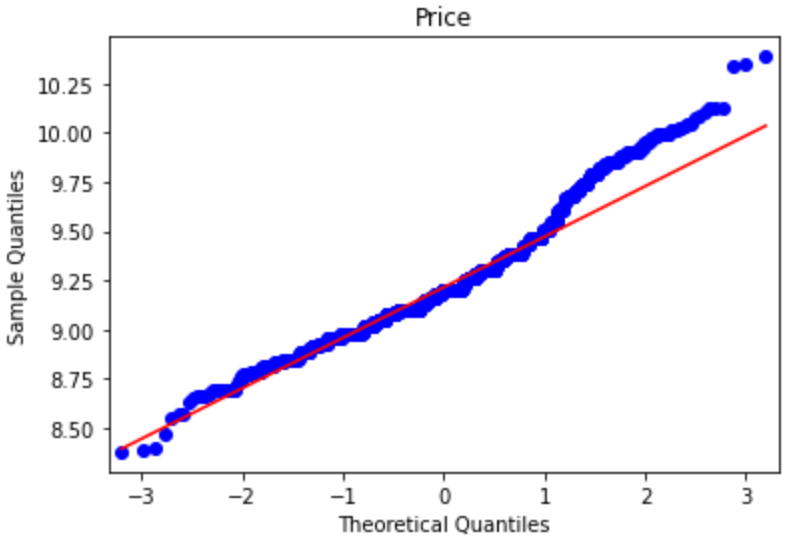


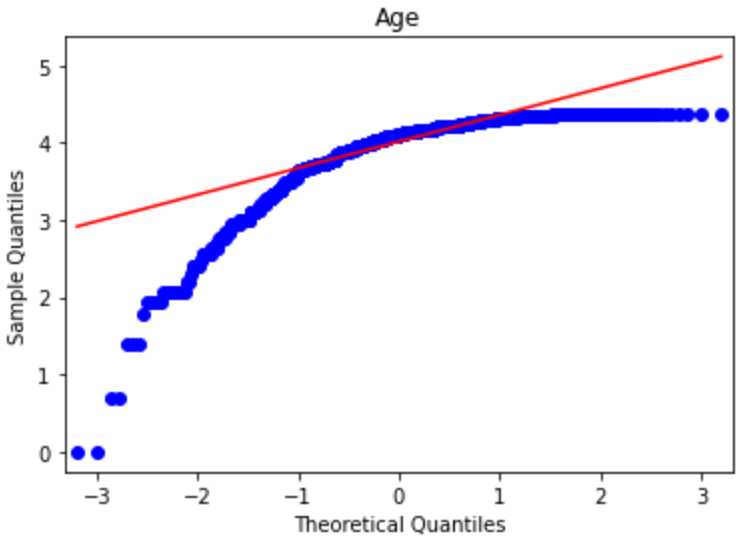


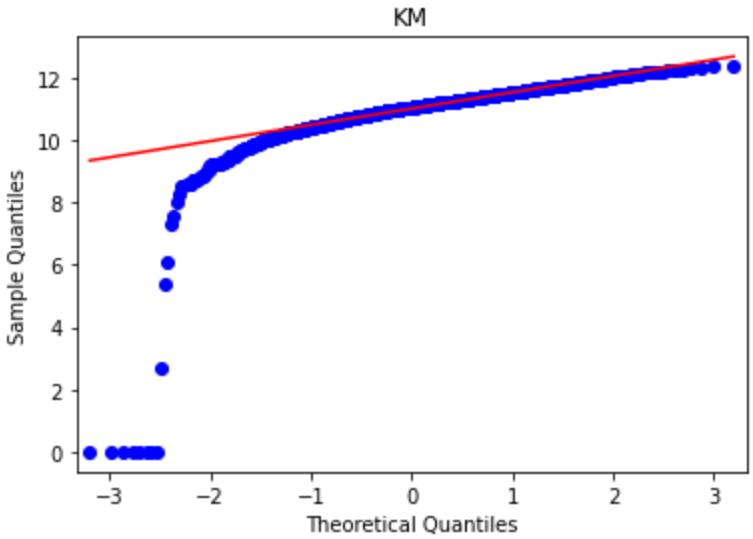
QQ-Plot For Log Transformation

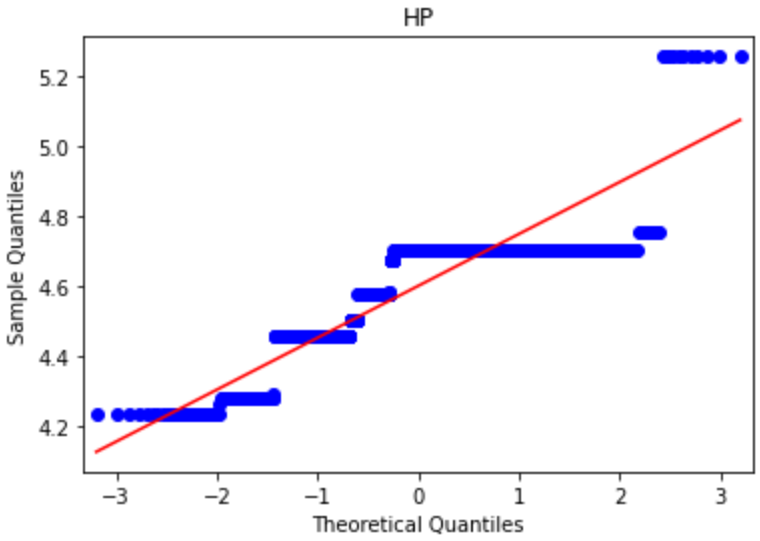
In [15]:

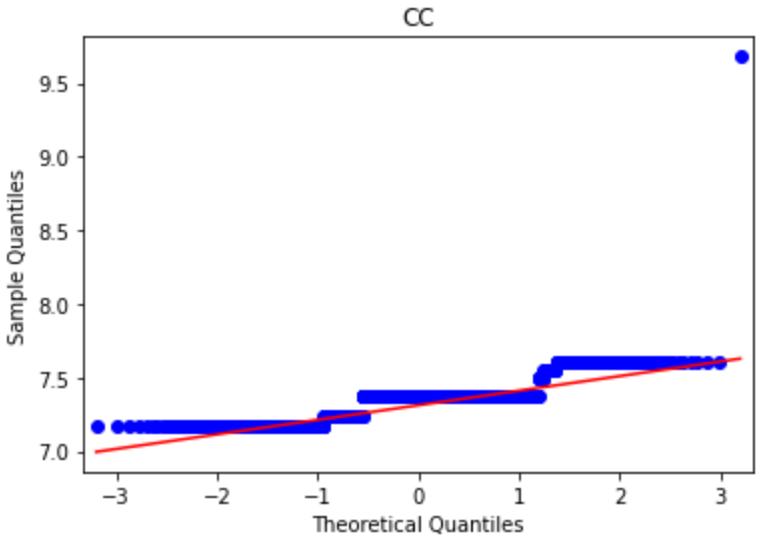
```
for feature in df1:
    data = df1.copy()
    data[feature]=np.log(data[feature])
    sm.qqplot(data[feature],line='q')
    plt.title(feature)
    plt.show()
```

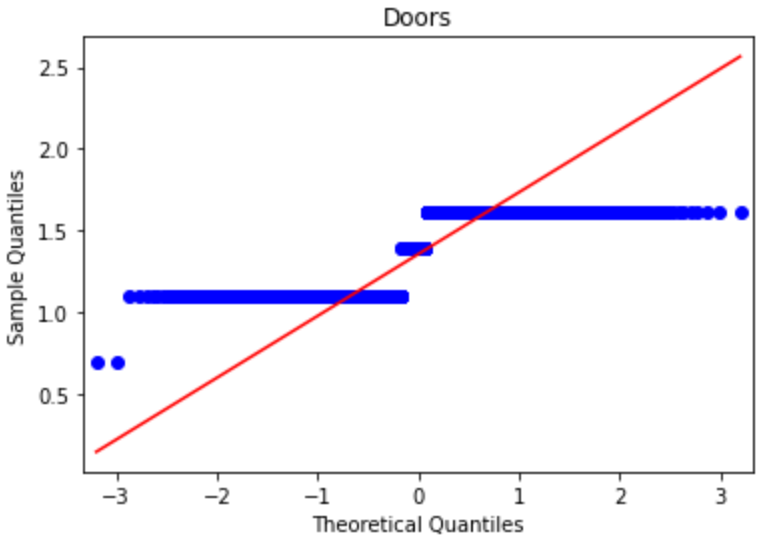


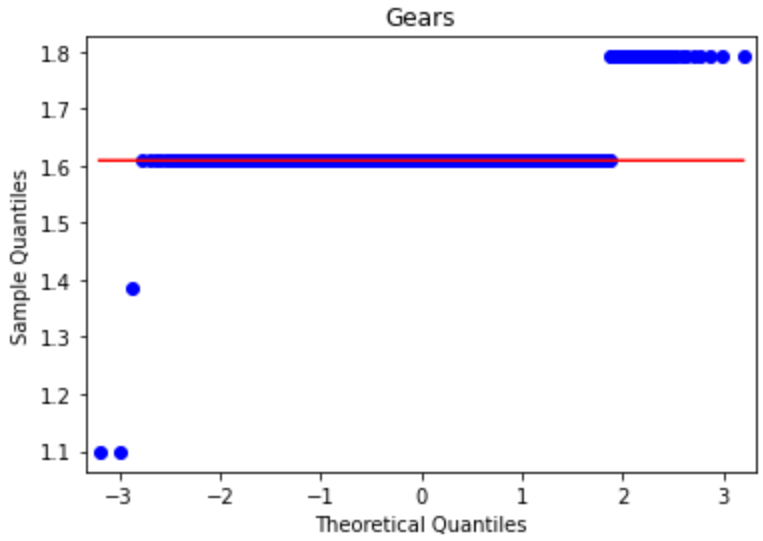


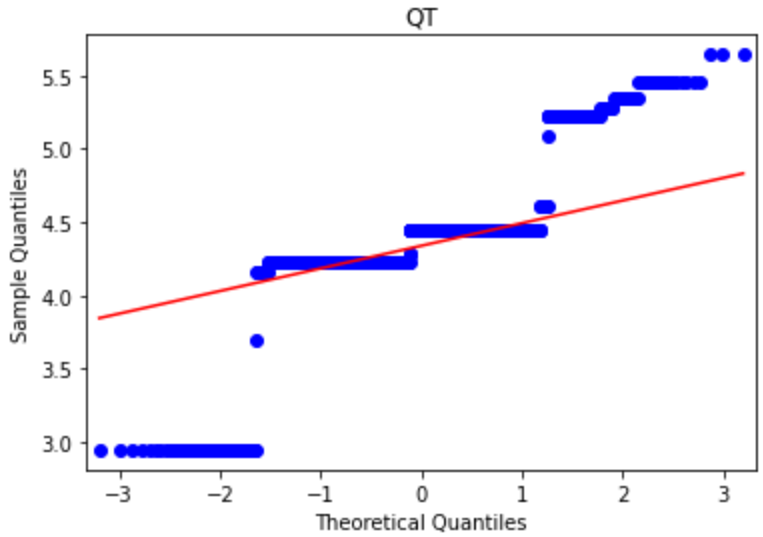


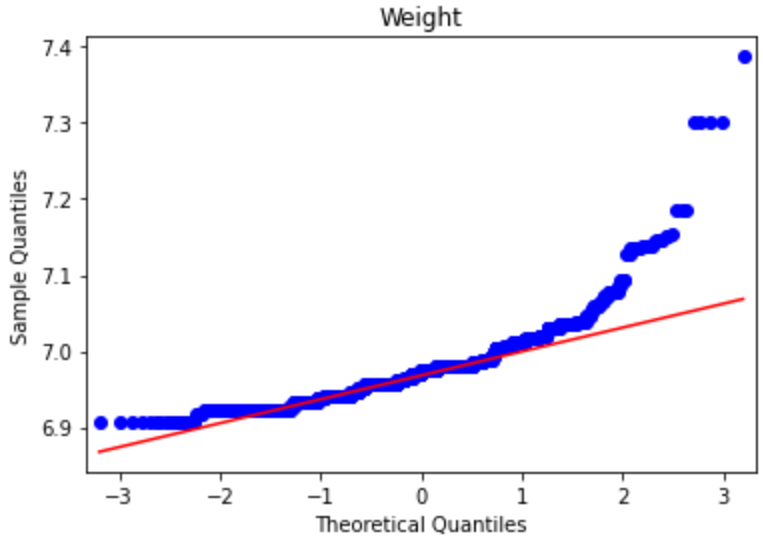








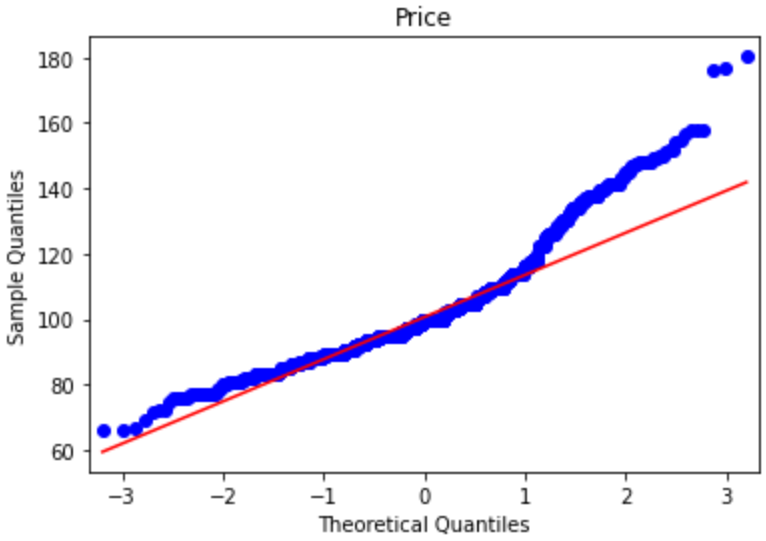


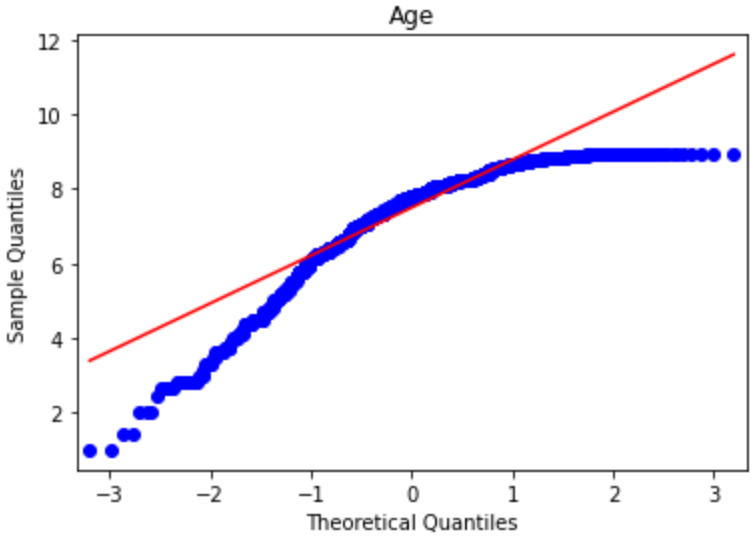


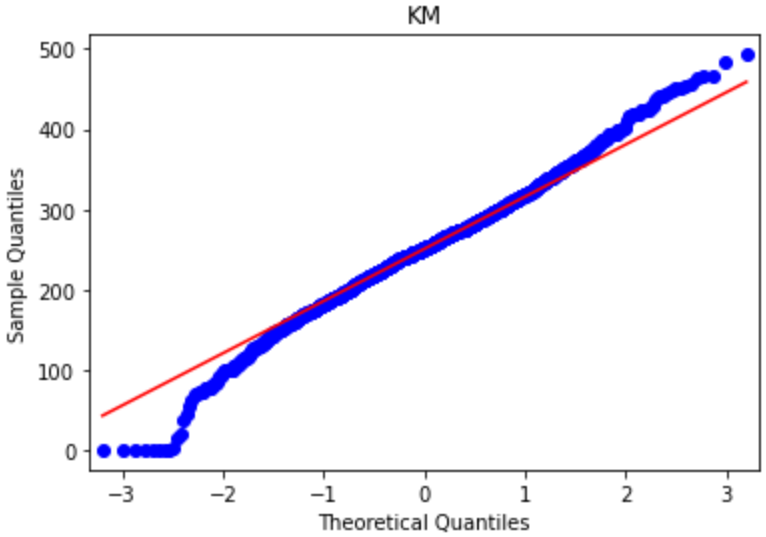
QQ-Plot For Squareroot Transformation

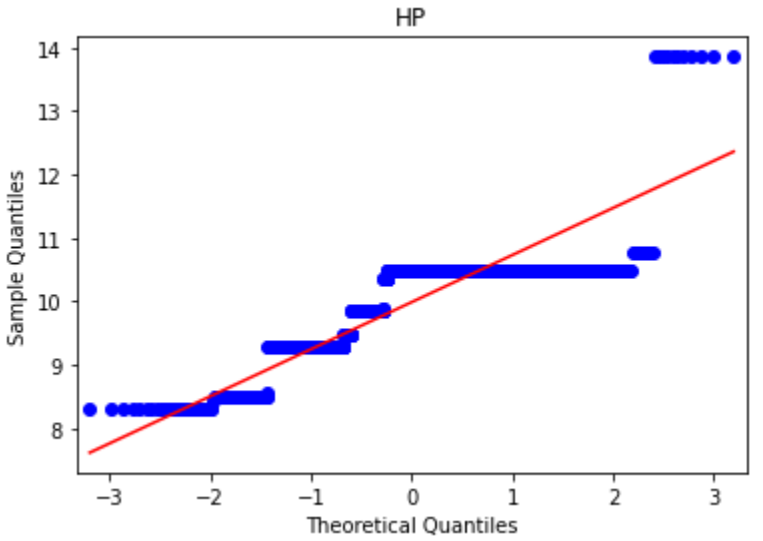
In [16]:

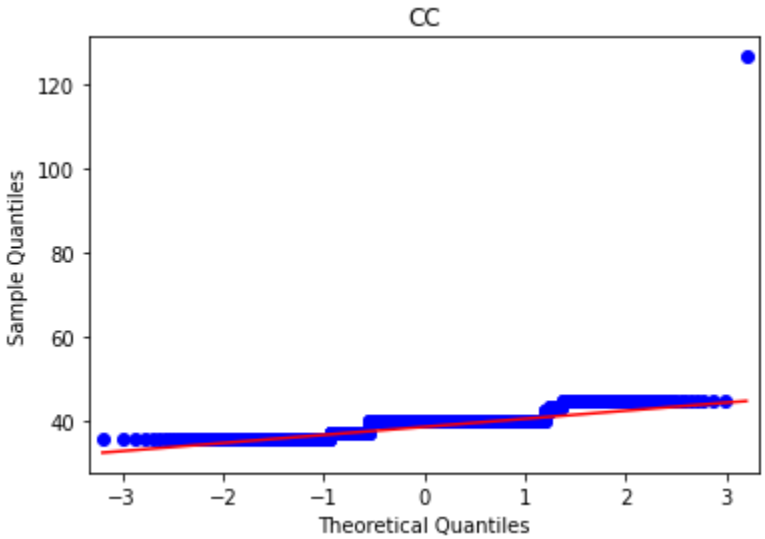
```
for feature in df1:
    data = df1.copy()
    data[feature]=np.sqrt(data[feature])
    sm.qqplot(data[feature],line='q')
    plt.title(feature)
    plt.show()
```

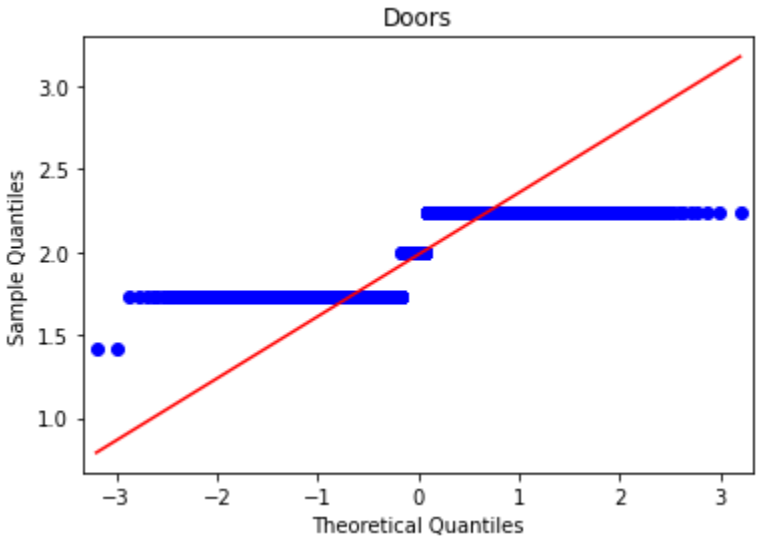


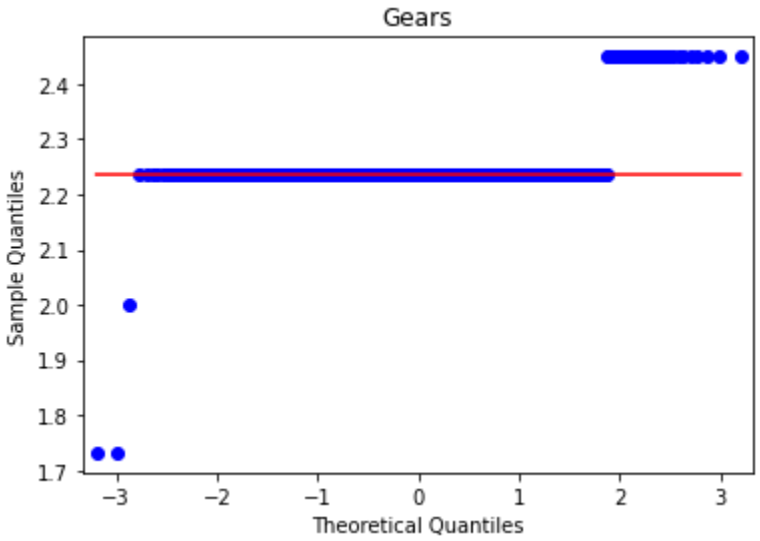


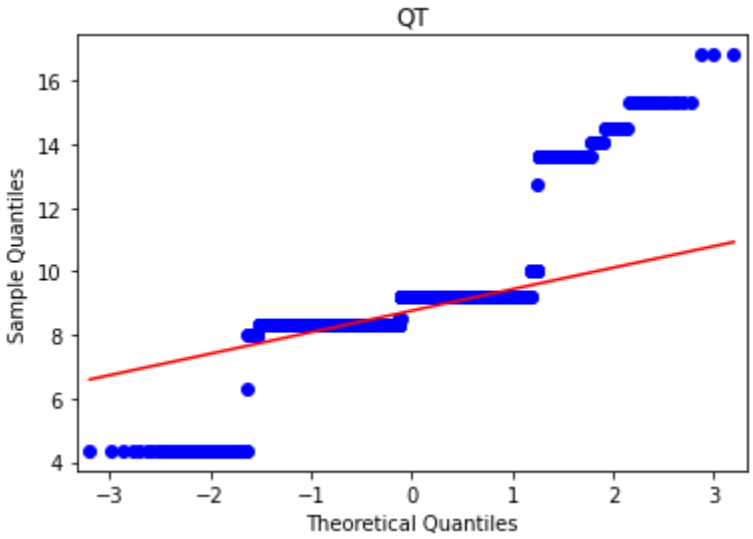


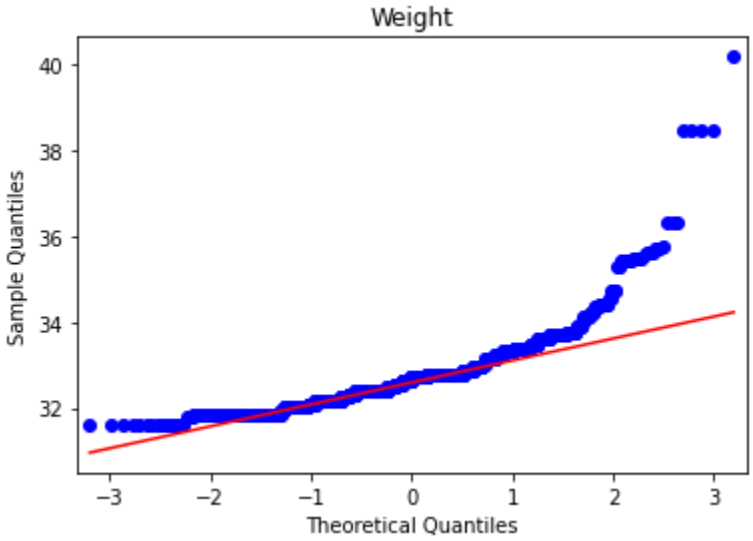








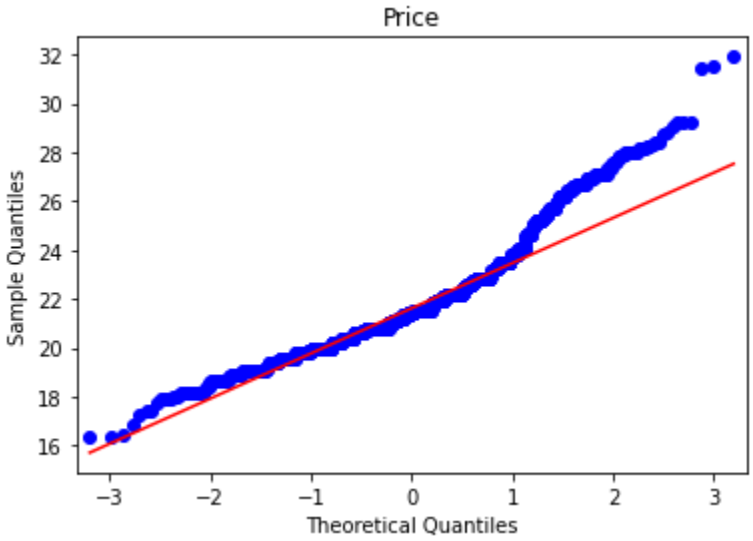


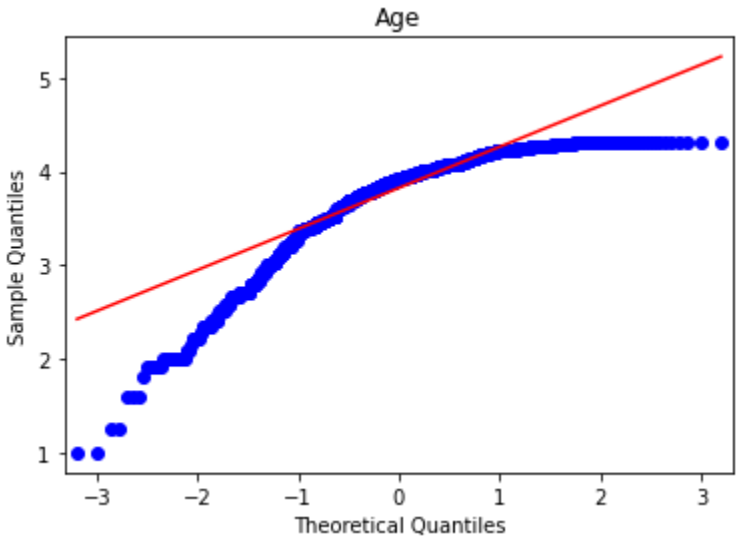


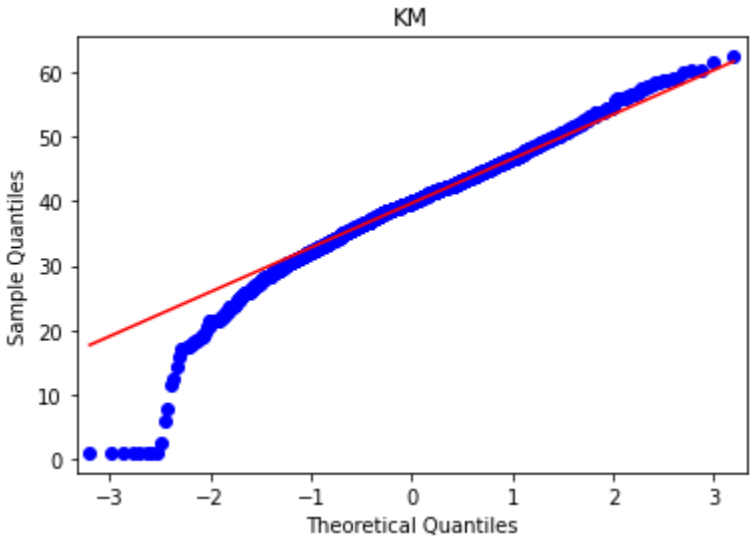
QQ-Plot for Cuberoot Transformation

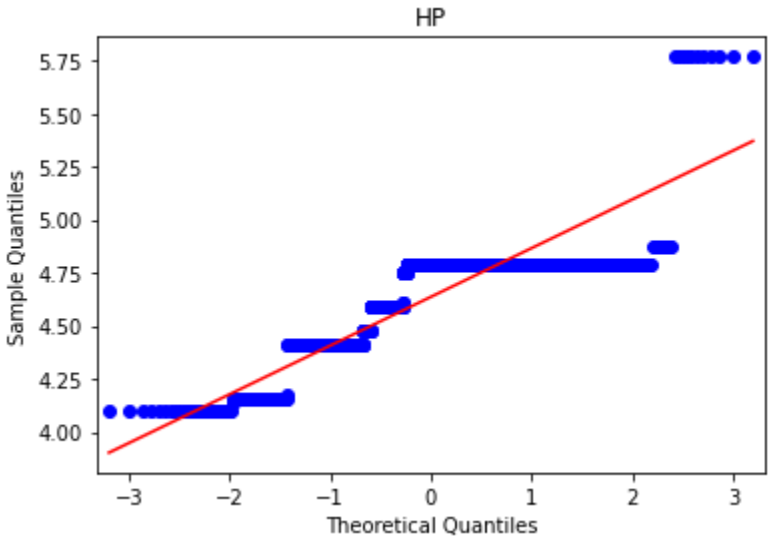
In [17]:

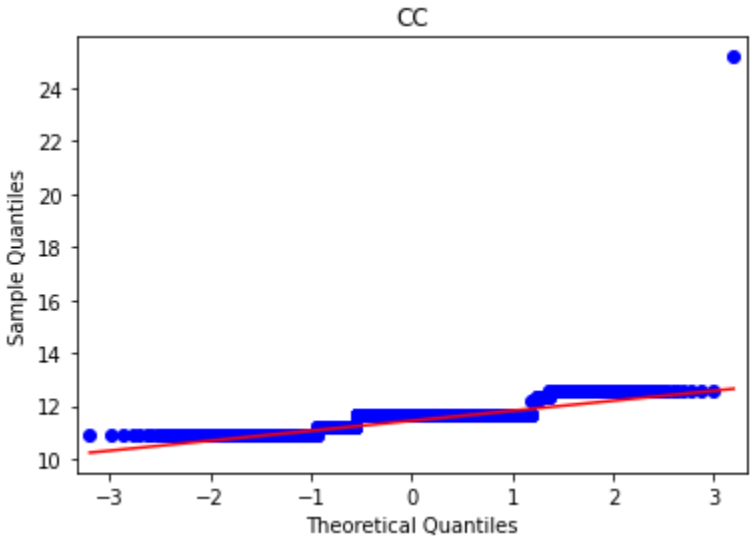
```
for feature in df1:
    data = df1.copy()
    data[feature]=np.cbrt(data[feature])
    sm.qqplot(data[feature],line='q')
    plt.title(feature)
    plt.show()
```

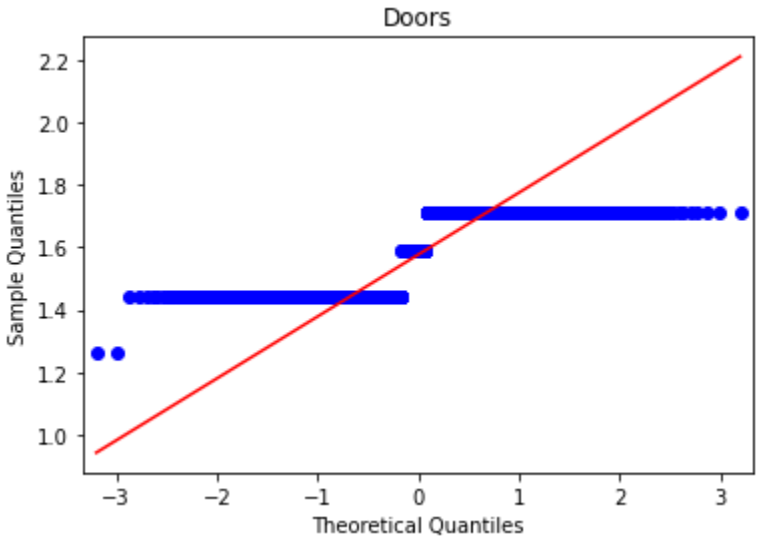


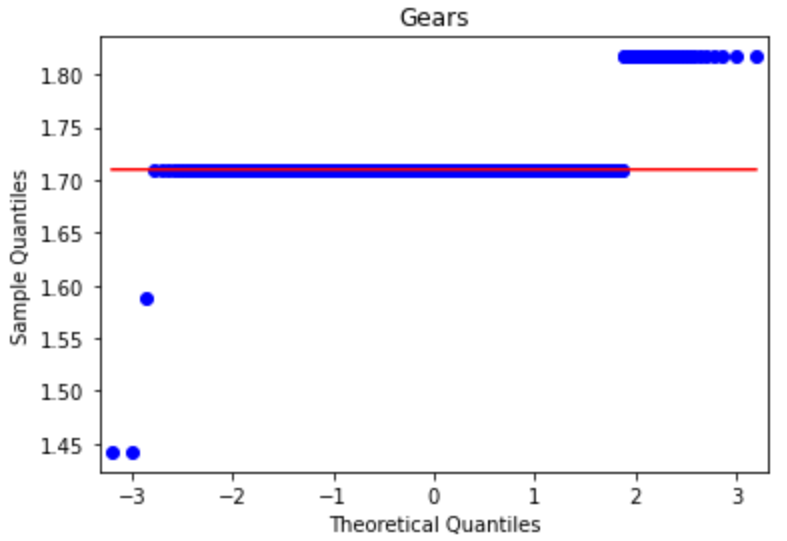


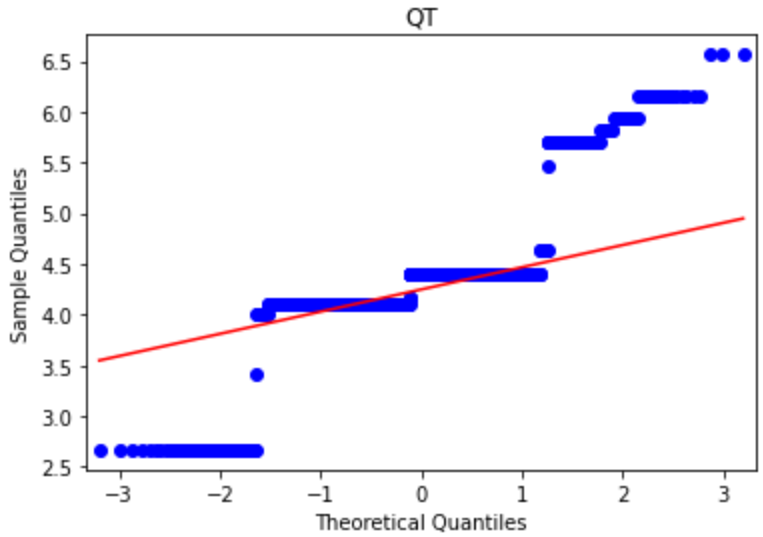


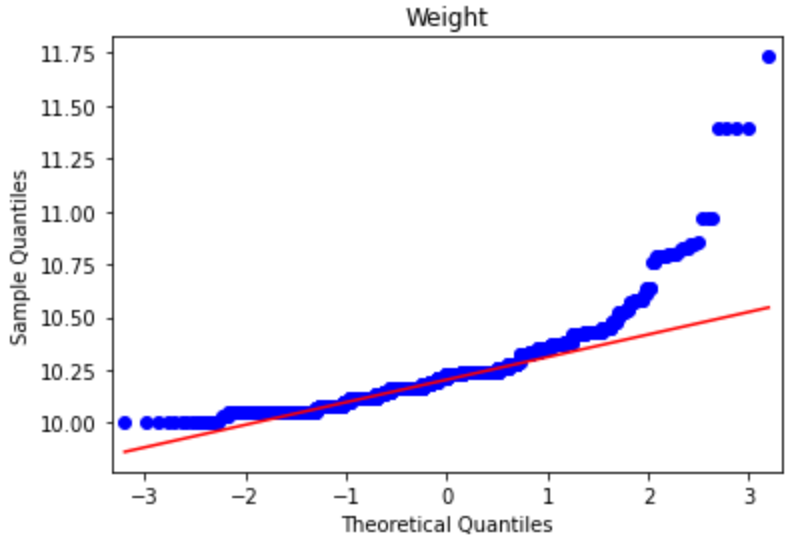










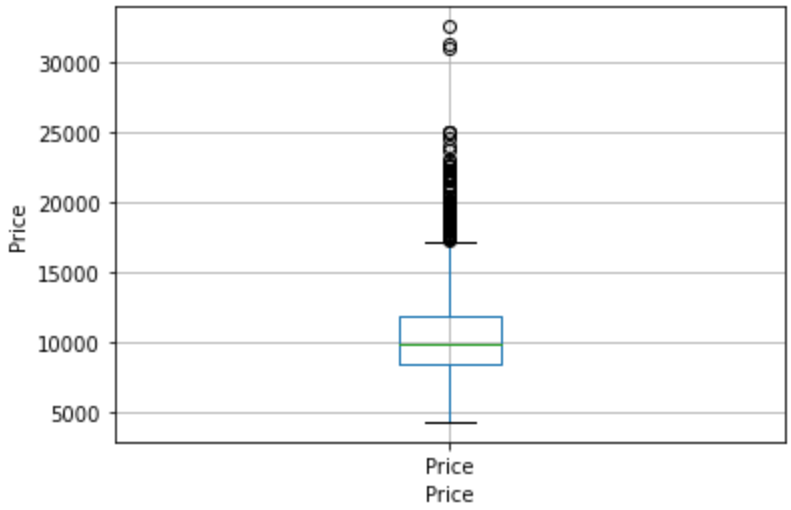


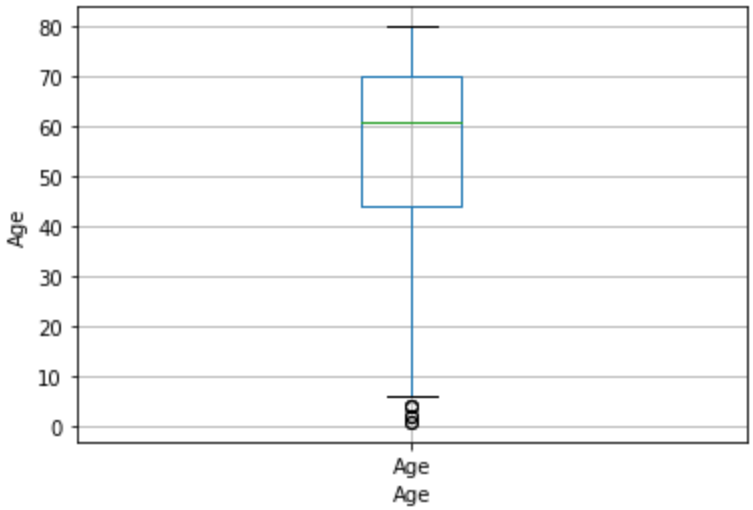
ObservatIon

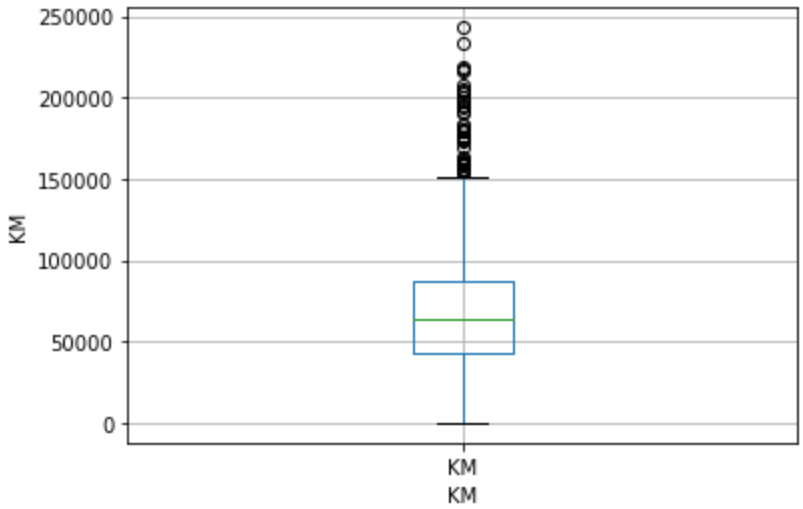
At normal raw data the data is not in normally distributed and after doing transformation the data is remained the same as it is so we use raw data for predictions

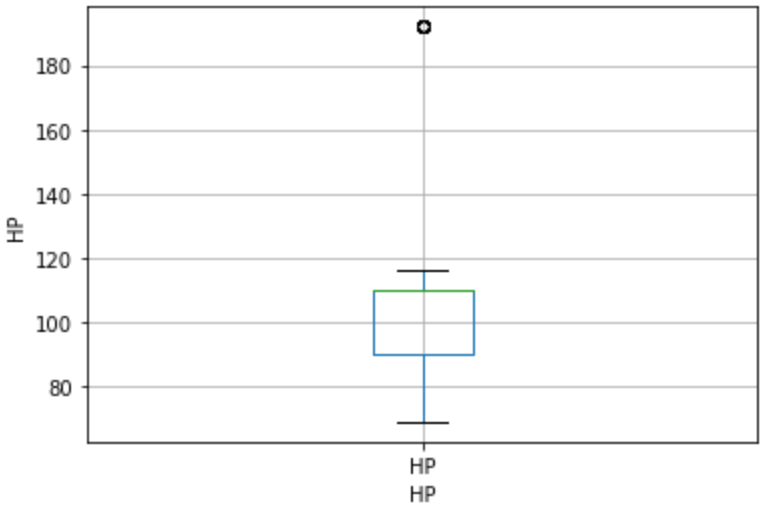
Checking Outliers Using Raw Data

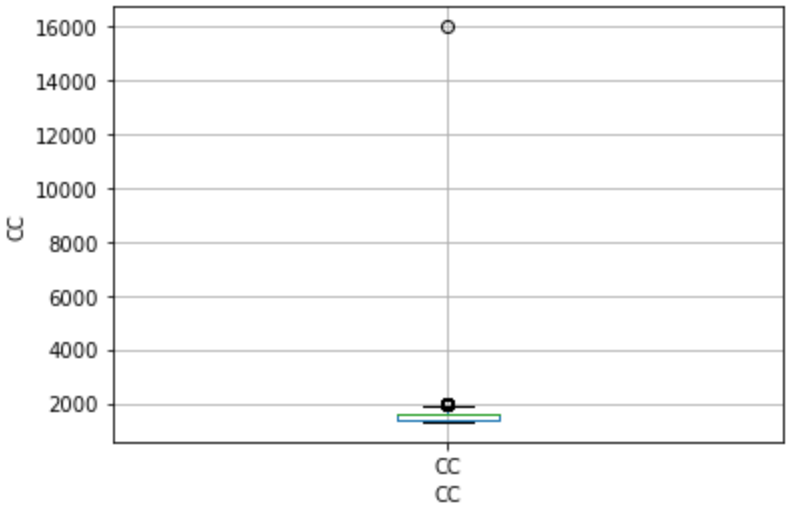
In [18]:
```
for feature in df1:
    data = df1.copy()
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```

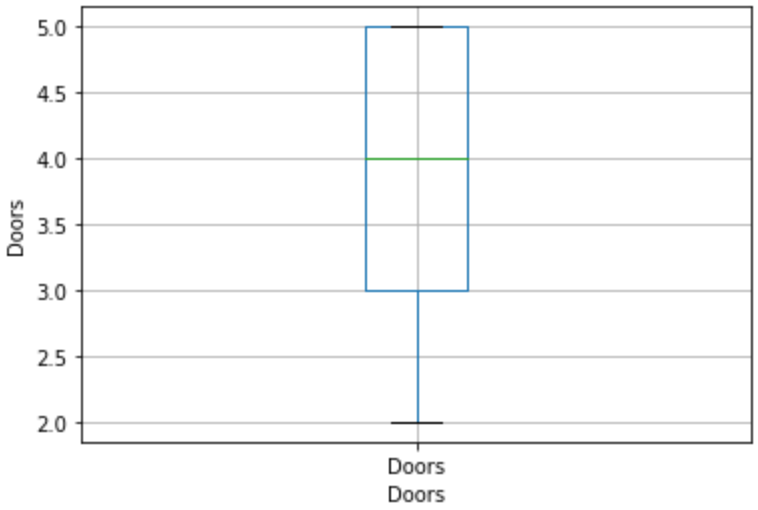


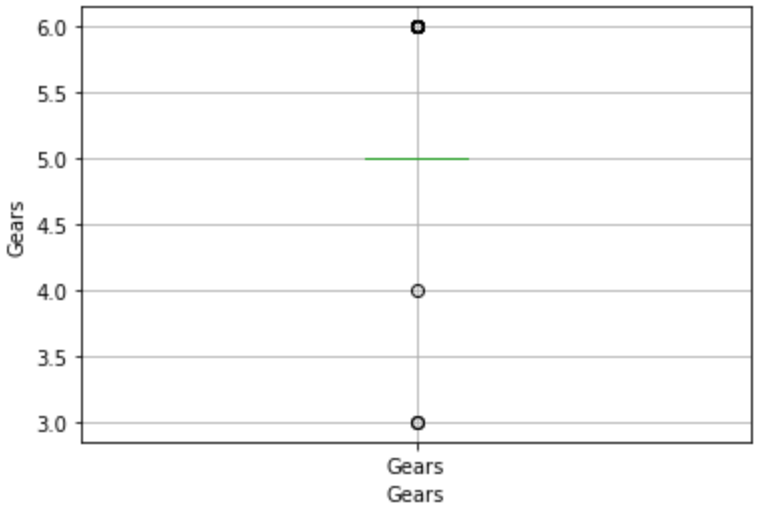


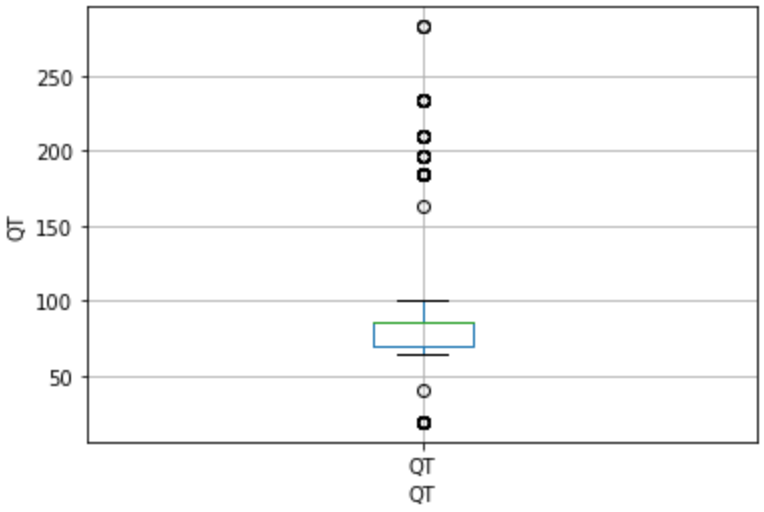


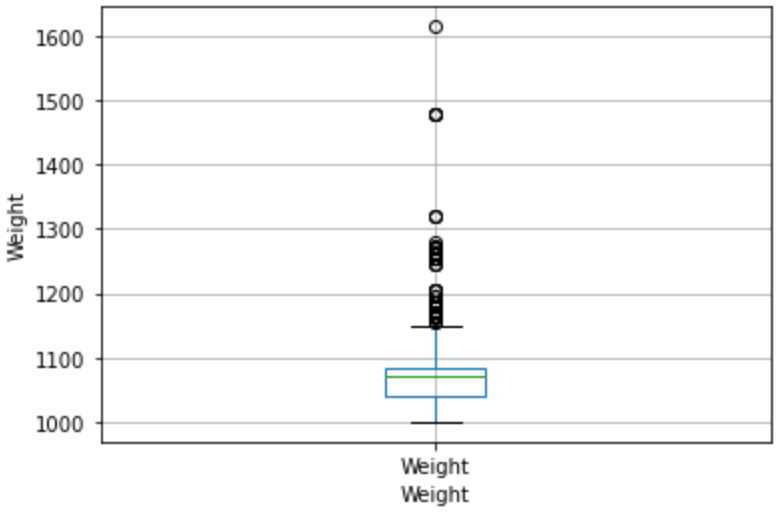








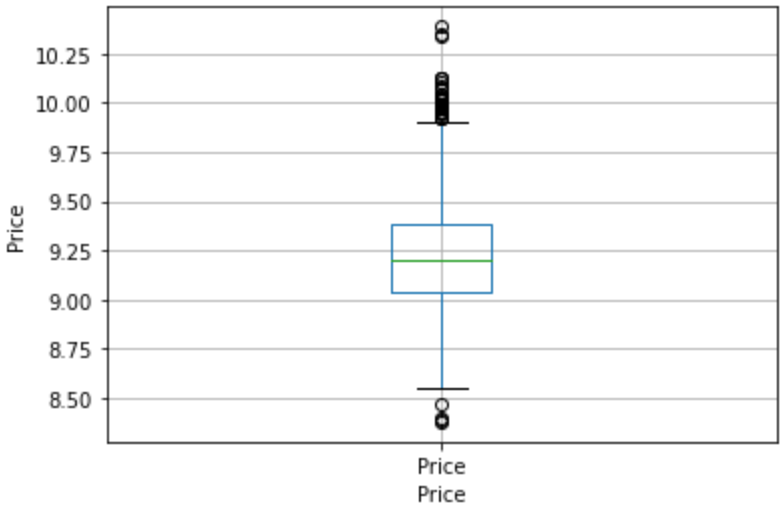


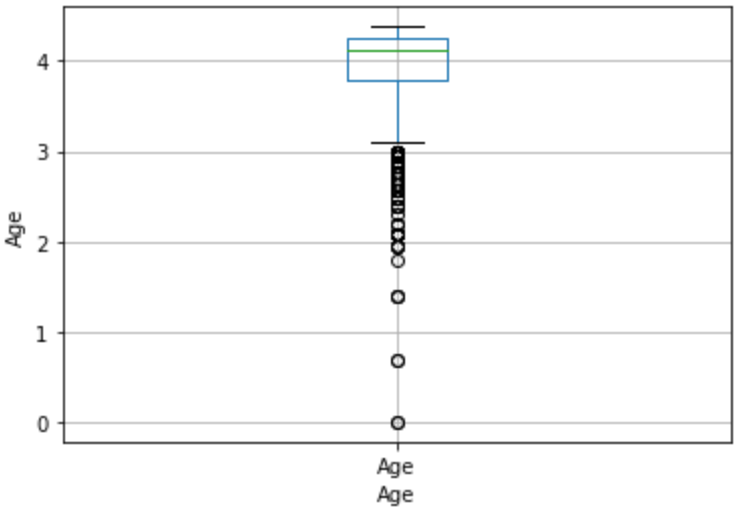


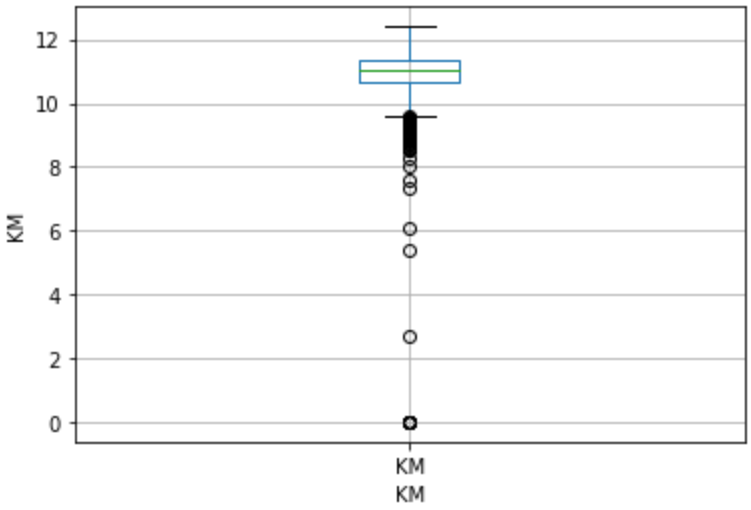
Checking Outliers using Log Transformation

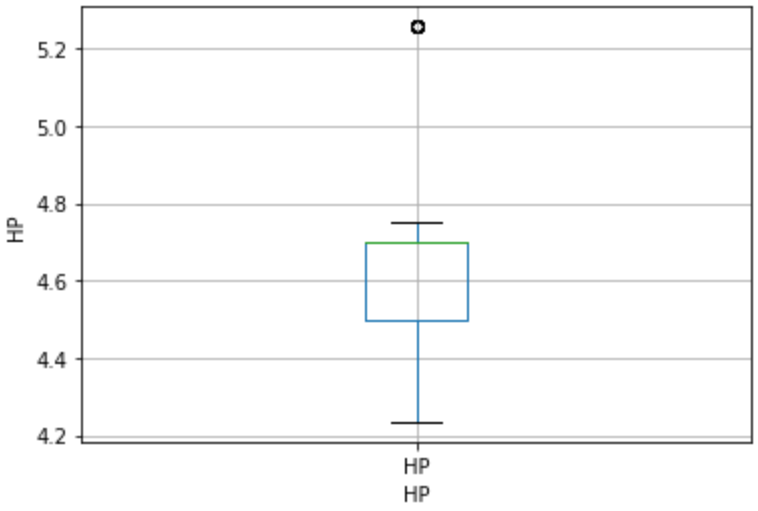
In [20]:

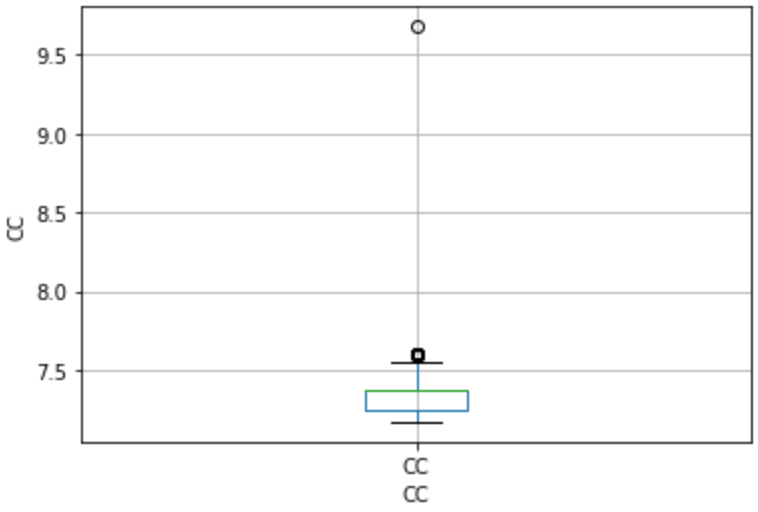
```
for feature in df1:
    data = df1.copy()
    data[feature]=np.log(data[feature])
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```

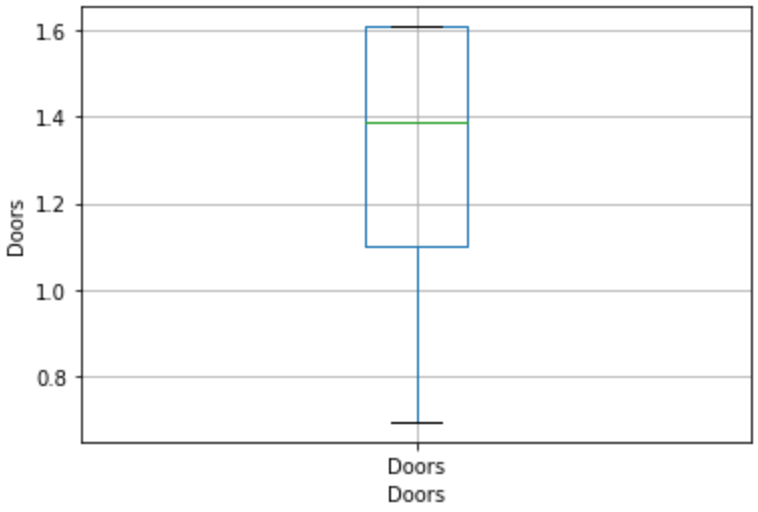


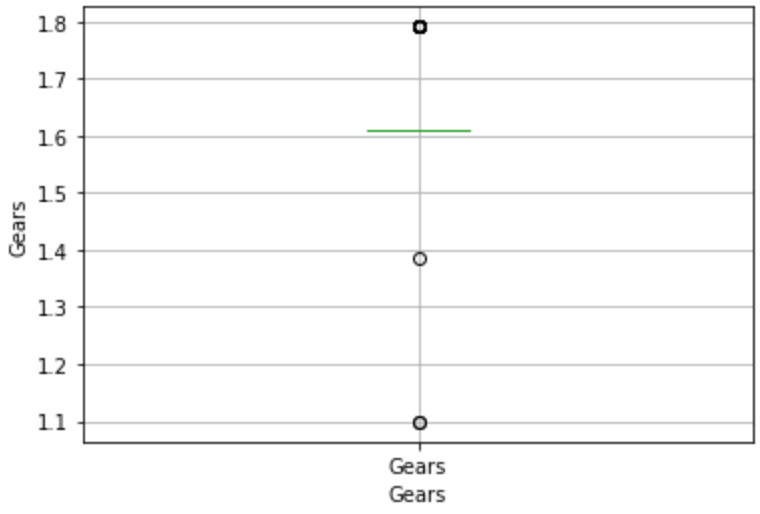


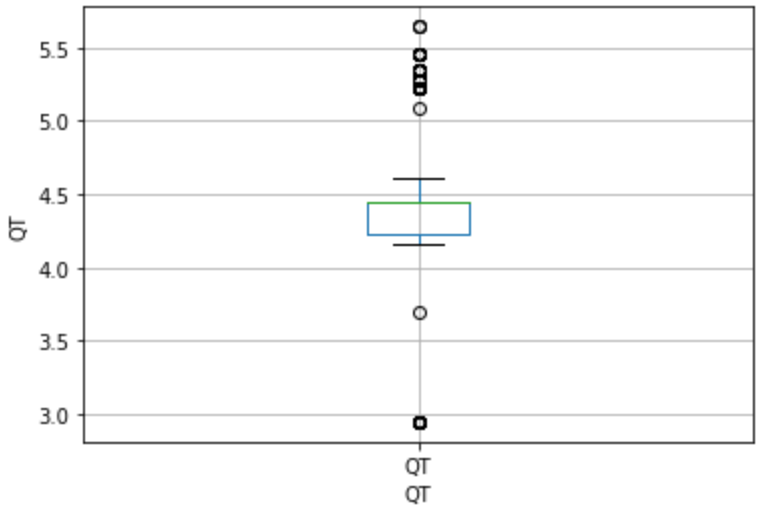


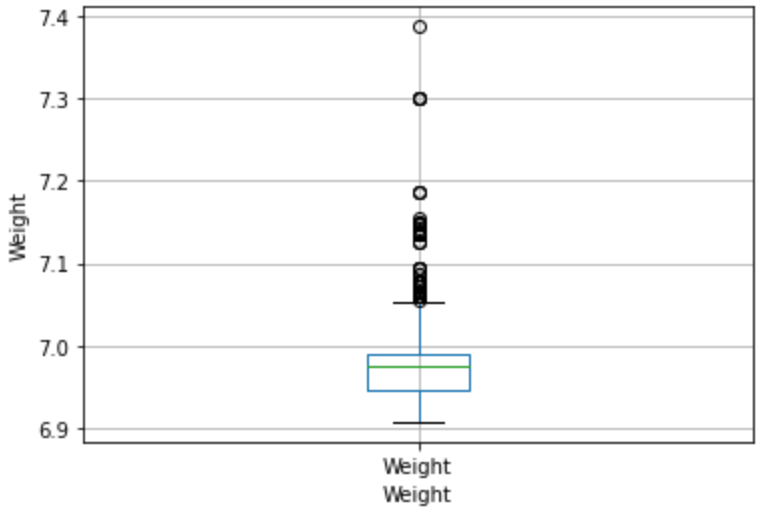








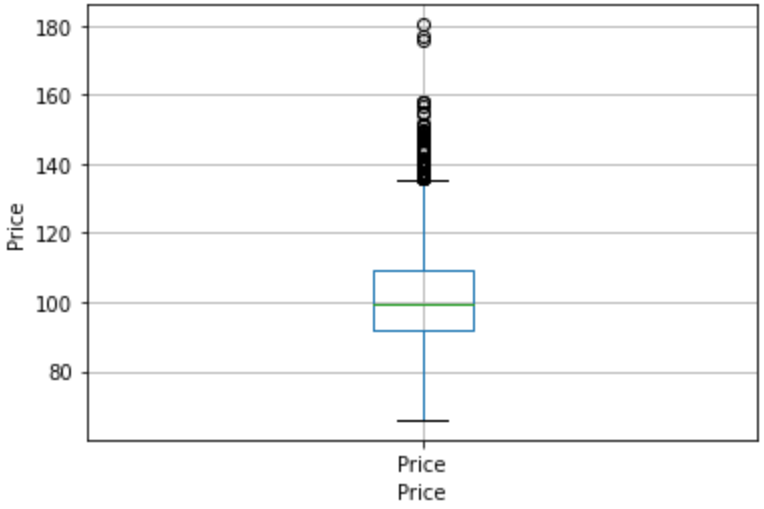


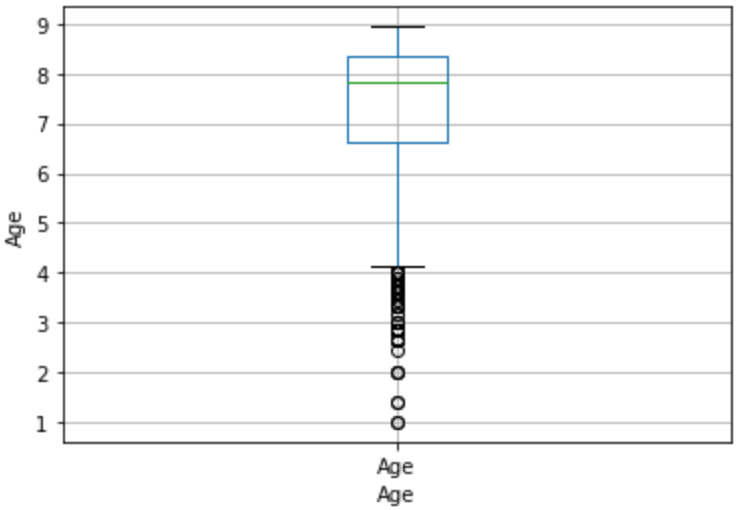


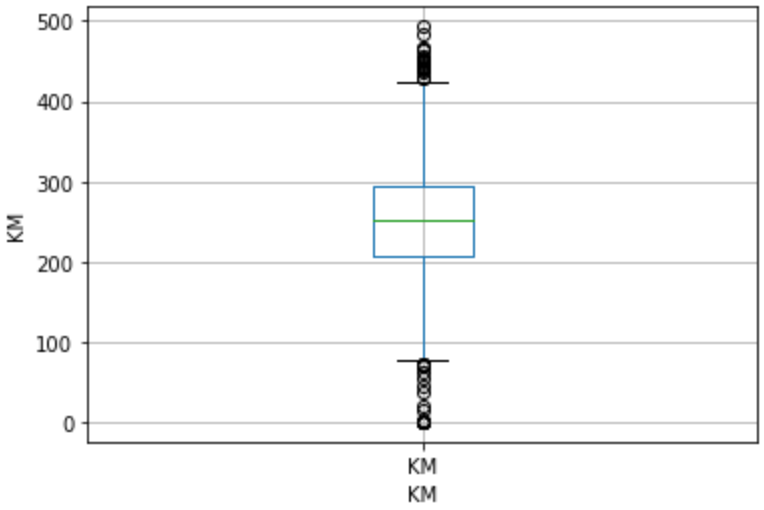
Checking Outliers Using Squareroot Transformation

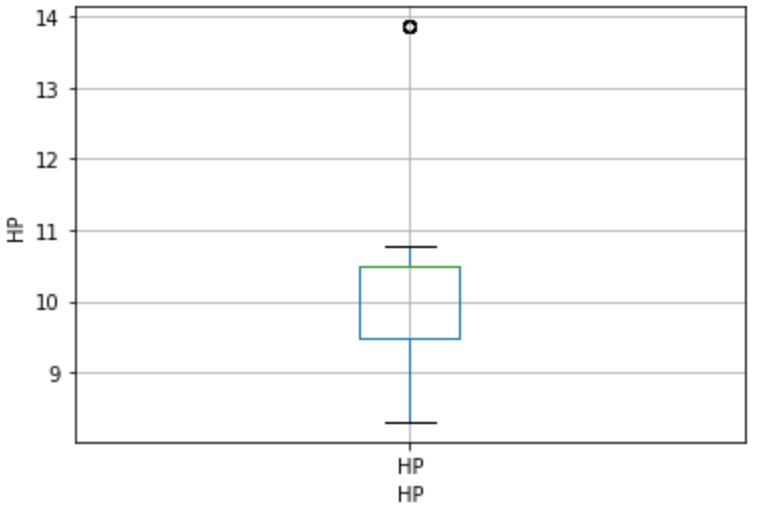
In [19]:

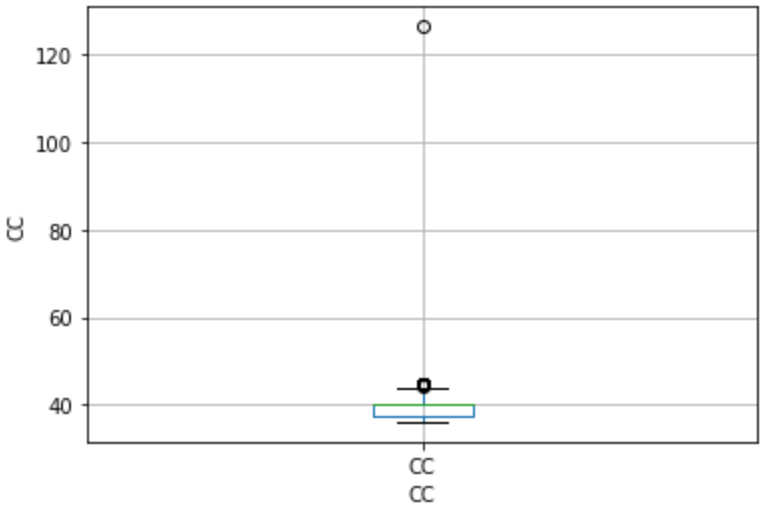
```
for feature in df1:
    data = df1.copy()
    data[feature]=np.sqrt(data[feature])
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```

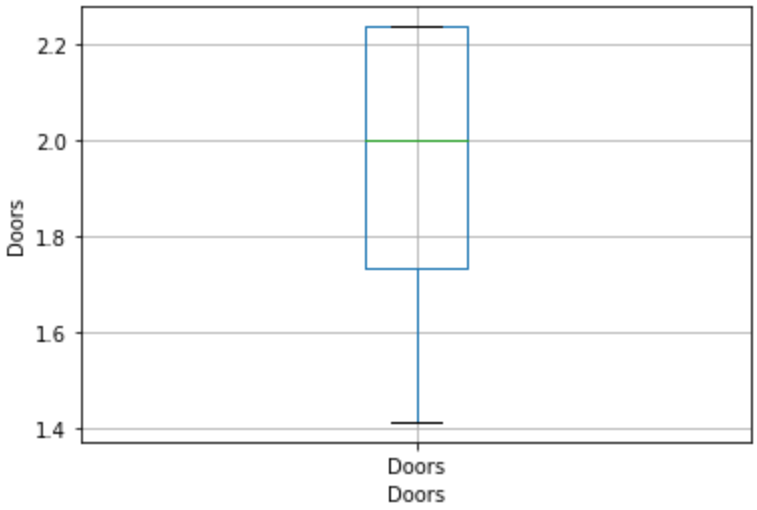


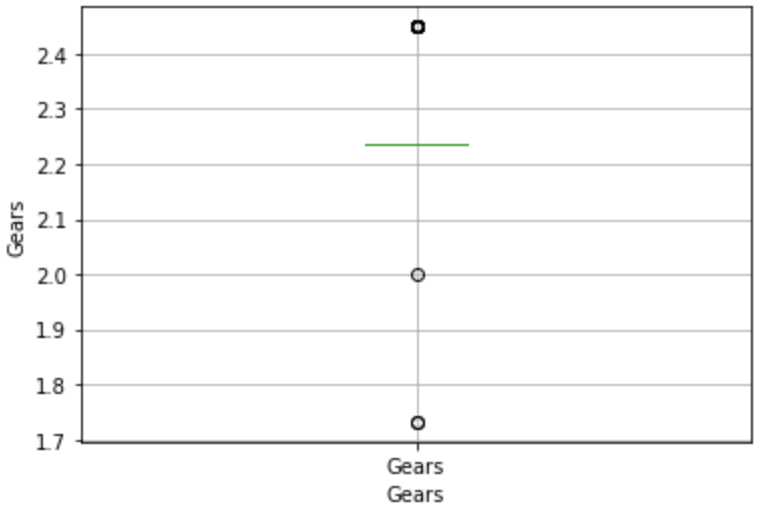


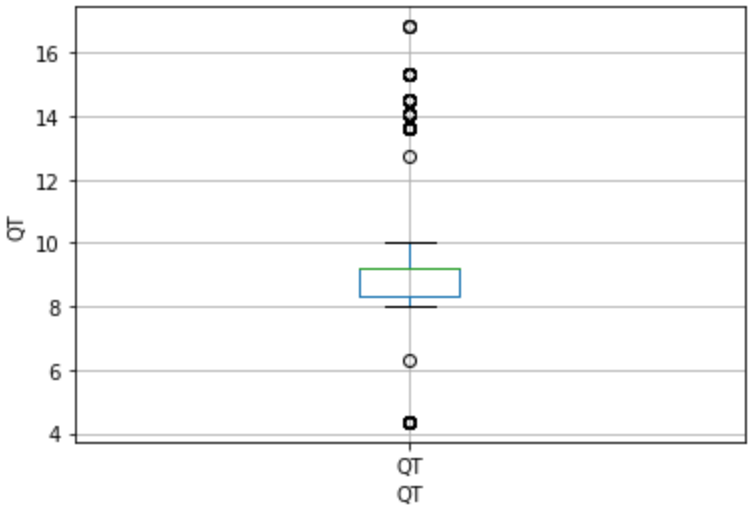


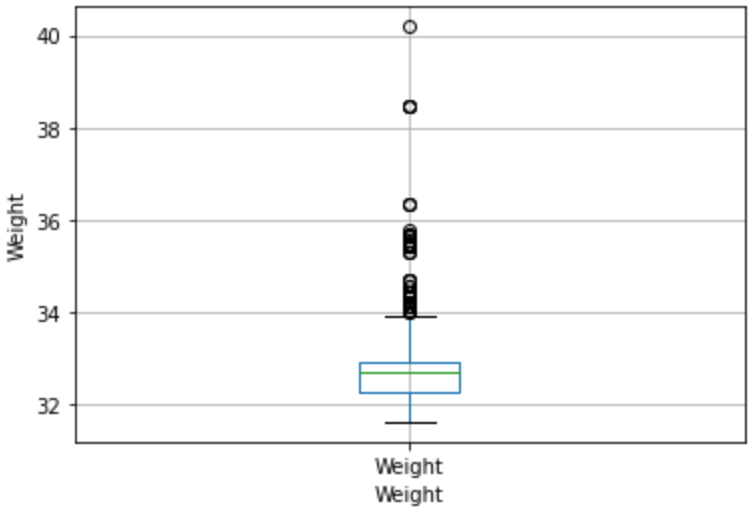








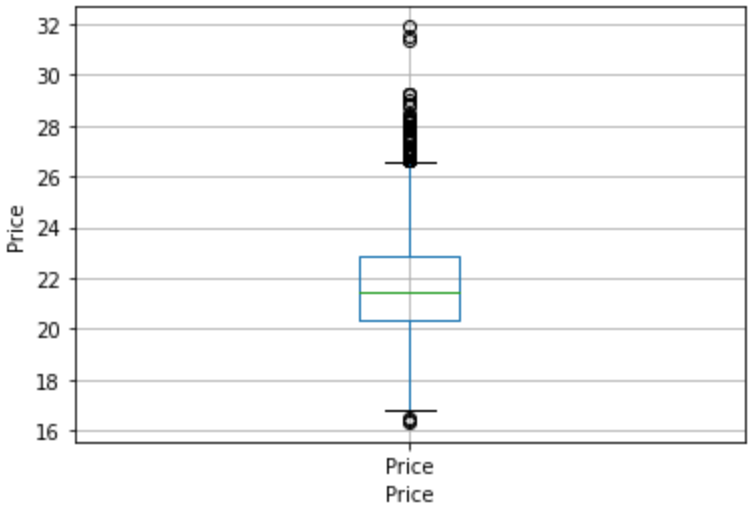


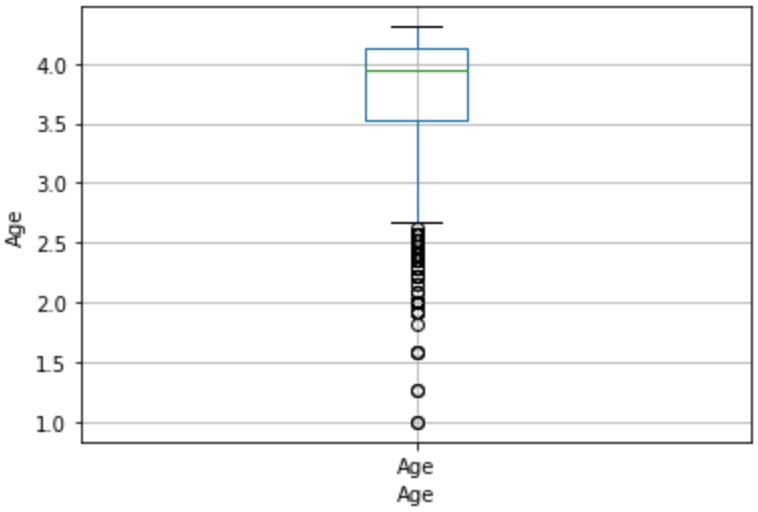


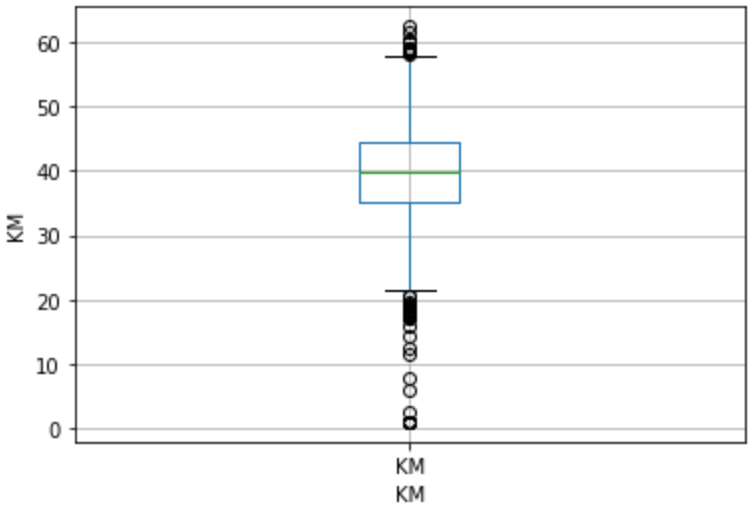
Checking Outliers using Cuberoot Transformation

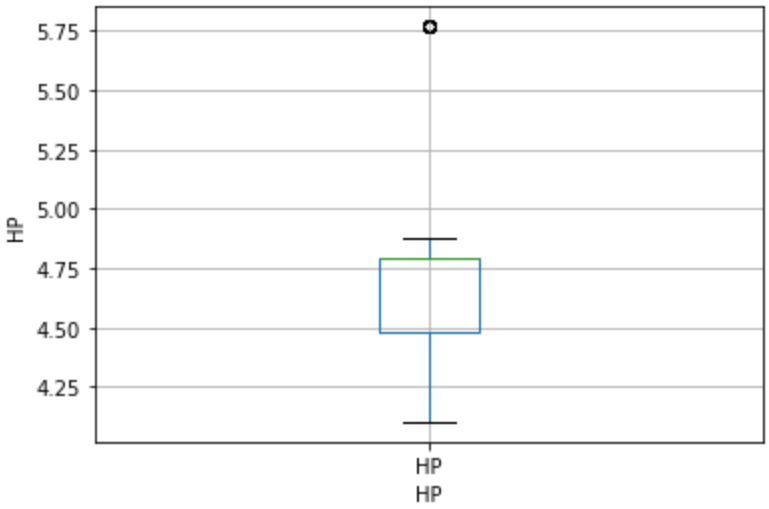
In [20]:

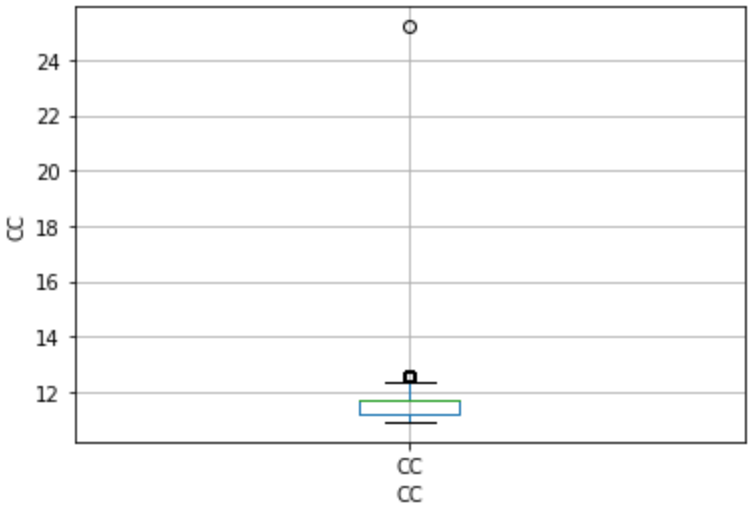
```
for feature in df1:
    data = df1.copy()
    data[feature]=np.cbrt(data[feature])
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```

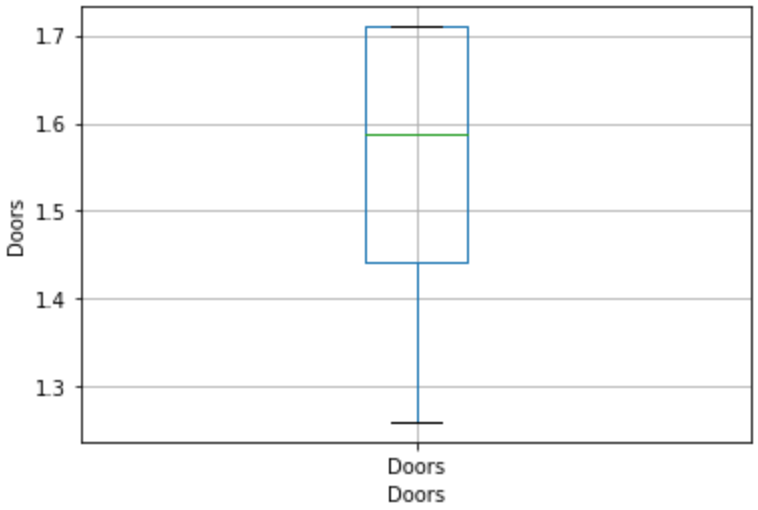


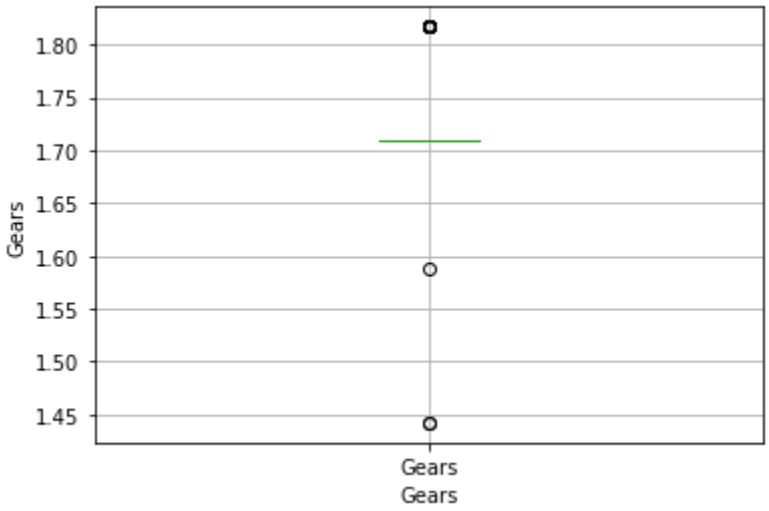


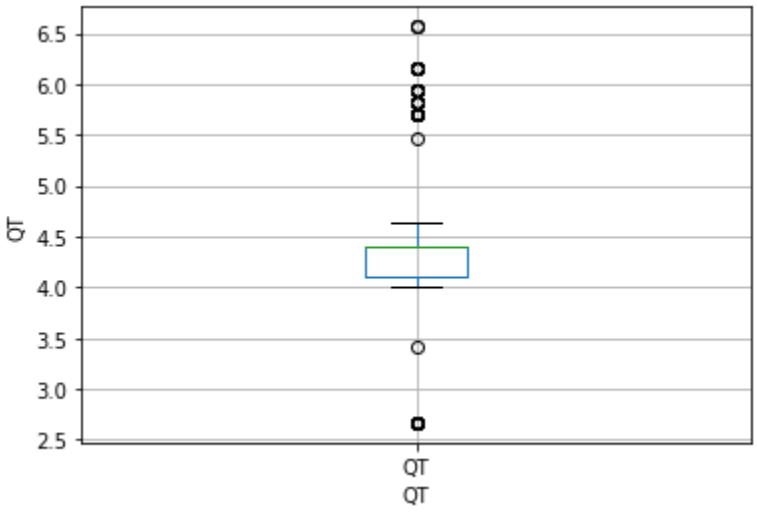


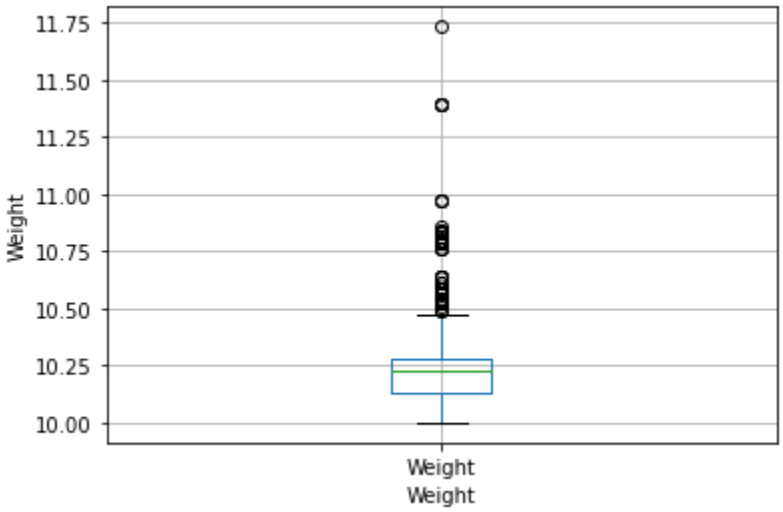












Observation

As we can see the raw data having an outlier and also after doing transformation the outlier in the data increases so we can choose the raw data for my predictions

Creating First Model

In [21]:

```
model1 = smf.ols("Price~Age+KM+HP+CC+Doors+Gears+QT+Weight",data=df1).fit()
model1.summary()
```

Out[21]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.864
Model:	OLS	**Adj. R-squared:**	0.863
Method:	Least Squares	**F-statistic:**	1131.
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.00
Time:	20:01:21	**Log-Likelihood:**	-12376.
No. Observations:	1436	**AIC:**	2.477e+04
Df Residuals:	1427	**BIC:**	2.482e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5573.1064	1411.390	-3.949	0.000	-8341.728	-2804.485
Age	-121.6584	2.616	-46.512	0.000	-126.789	-116.527
KM	-0.0208	0.001	-16.622	0.000	-0.023	-0.018
HP	31.6809	2.818	11.241	0.000	26.152	37.209
CC	-0.1211	0.090	-1.344	0.179	-0.298	0.056
Doors	-1.6166	40.006	-0.040	0.968	-80.093	76.859
Gears	594.3199	197.055	3.016	0.003	207.771	980.869
QT	3.9491	1.310	3.015	0.003	1.379	6.519
Weight	16.9586	1.068	15.880	0.000	14.864	19.054

Omnibus:	151.719	**Durbin-Watson:**	1.543
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1011.853
Skew:	-0.219	**Prob(JB):**	1.90e-220
Kurtosis:	7.089	**Cond. No.**	3.13e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.13e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Simple linear Regression

In [22]:

```
model2 = smf.ols("Price~Age",data=df1).fit()
model2.summary()
```

Out[22]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.768
Model:	OLS	**Adj. R-squared:**	0.768
Method:	Least Squares	**F-statistic:**	4758.
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.00

Time:	20:01:21	**Log-Likelihood:**	-12756.
No. Observations:	1436	**AIC:**	2.552e+04
Df Residuals:	1434	**BIC:**	2.553e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.029e+04	146.097	138.908	0.000	2e+04	2.06e+04
Age	-170.9336	2.478	-68.978	0.000	-175.795	-166.073

Omnibus:	359.275	**Durbin-Watson:**	1.214
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2774.226
Skew:	0.946	**Prob(JB):**	0.00
Kurtosis:	9.541	**Cond. No.**	187.

Notes:

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

In [23]:
```
model3=smf.ols("Price~KM",data=df1).fit()
model3.summary()
```

Out[23]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.325
Model:	OLS	**Adj. R-squared:**	0.324
Method:	Least Squares	**F-statistic:**	690.0
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	1.76e-124
Time:	20:01:22	**Log-Likelihood:**	-13525.
No. Observations:	1436	**AIC:**	2.705e+04
Df Residuals:	1434	**BIC:**	2.706e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.451e+04	163.915	88.510	0.000	1.42e+04	1.48e+04
KM	-0.0551	0.002	-26.268	0.000	-0.059	-0.051

Omnibus:	390.716	**Durbin-Watson:**	0.386
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1115.783
Skew:	1.388	**Prob(JB):**	5.14e-243
Kurtosis:	6.308	**Cond. No.**	1.63e+05

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.63e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [24]:
```
model4=smf.ols("Price~HP",data=df1).fit()
model4.summary()
```

Out[24]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.099
Model:	OLS	**Adj. R-squared:**	0.099
Method:	Least Squares	**F-statistic:**	158.0
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	1.93e-34
Time:	20:01:22	**Log-Likelihood:**	-13732.
No. Observations:	1436	**AIC:**	2.747e+04
Df Residuals:	1434	**BIC:**	2.748e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2990.2764	622.568	4.803	0.000	1769.035	4211.518
HP	76.2600	6.068	12.568	0.000	64.357	88.163

Omnibus:	448.876	**Durbin-Watson:**	0.338
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1327.172
Skew:	1.591	**Prob(JB):**	6.43e-289
Kurtosis:	6.472	**Cond. No.**	703.

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

In [25]:
```
model5=smf.ols("Price~CC",data=df1).fit()
model5.summary()
```

Out[25]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.016
Model:	OLS	**Adj. R-squared:**	0.015
Method:	Least Squares	**F-statistic:**	23.28
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	1.55e-06
Time:	20:01:23	**Log-Likelihood:**	-13795.
No. Observations:	1436	**AIC:**	2.759e+04
Df Residuals:	1434	**BIC:**	2.760e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9027.5548	365.576	24.694	0.000	8310.435	9744.675
CC	1.0802	0.224	4.825	0.000	0.641	1.519

Omnibus:	465.181	**Durbin-Watson:**	0.267
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1390.401
Skew:	1.649	**Prob(JB):**	1.20e-302
Kurtosis:	6.516	**Cond. No.**	6.29e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.29e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [26]:
```
model6=smf.ols("Price~Doors",data=df1).fit()
model6.summary()
```

Out[26]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.034
Model:	OLS	**Adj. R-squared:**	0.034
Method:	Least Squares	**F-statistic:**	51.00
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	1.46e-12
Time:	20:01:23	**Log-Likelihood:**	-13782.
No. Observations:	1436	**AIC:**	2.757e+04
Df Residuals:	1434	**BIC:**	2.758e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7885.0058	409.438	19.258	0.000	7081.843	8688.168
Doors	705.5586	98.795	7.142	0.000	511.761	899.356

Omnibus:	466.779	**Durbin-Watson:**	0.287
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1406.209
Skew:	1.651	**Prob(JB):**	4.42e-306
Kurtosis:	6.549	**Cond. No.**	19.0

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

In [27]:
```
model7=smf.ols("Price~QT",data=df1).fit()
model7.summary()
```

Out[27]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.048
Model:	OLS	**Adj. R-squared:**	0.047
Method:	Least Squares	**F-statistic:**	72.38
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	4.41e-17
Time:	20:01:24	**Log-Likelihood:**	-13771.
No. Observations:	1436	**AIC:**	2.755e+04
Df Residuals:	1434	**BIC:**	2.756e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9046.7382	218.889	41.330	0.000	8617.362	9476.115
QT	19.3301	2.272	8.507	0.000	14.873	23.787

Omnibus:	369.212	**Durbin-Watson:**	0.276
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	843.513
Skew:	1.417	**Prob(JB):**	6.82e-184
Kurtosis:	5.462	**Cond. No.**	226.

Notes:

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

In [28]:

```
model8=smf.ols("Price~Age+HP",data=df1).fit()
model8.summary()
```

Out[28]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.801
Model:	OLS	**Adj. R-squared:**	0.801
Method:	Least Squares	**F-statistic:**	2880.
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.00
Time:	20:01:25	**Log-Likelihood:**	-12648.
No. Observations:	1436	**AIC:**	2.530e+04
Df Residuals:	1433	**BIC:**	2.532e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.551e+04	341.793	45.367	0.000	1.48e+04	1.62e+04
Age	-165.3701	2.328	-71.038	0.000	-169.937	-160.804
HP	44.1027	2.890	15.259	0.000	38.433	49.772

Omnibus:	361.527	**Durbin-Watson:**	1.335

Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2605.922
Skew:	0.975	**Prob(JB):**	0.00
Kurtosis:	9.305	**Cond. No.**	934.

Notes:

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

In [29]:
```
model9=smf.ols("Price~Age+HP+CC",data=df1).fit()
model9.summary()
```

Out[29]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.802
Model:	OLS	**Adj. R-squared:**	0.802
Method:	Least Squares	**F-statistic:**	1935.
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.00
Time:	20:01:26	**Log-Likelihood:**	-12643.
No. Observations:	1436	**AIC:**	2.529e+04
Df Residuals:	1432	**BIC:**	2.532e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.499e+04	378.735	39.573	0.000	1.42e+04	1.57e+04
Age	-164.6852	2.331	-70.649	0.000	-169.258	-160.113
HP	43.9143	2.882	15.237	0.000	38.261	49.568
CC	0.3166	0.101	3.136	0.002	0.119	0.515

Omnibus:	349.564	**Durbin-Watson:**	1.329
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2510.175
Skew:	0.937	**Prob(JB):**	0.00
Kurtosis:	9.200	**Cond. No.**	1.45e+04

Notes:

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

[2] The condition number is large, 1.45e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [30]:
```
model10=smf.ols("Price~Age+HP+CC+Doors",data=df1).fit()
model10.summary()
```

Out[30]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.804
	OLS	**Adj. R-squared:**	0.803

Method:	Least Squares	**F-statistic:**	1466.
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.00
Time:	20:01:28	**Log-Likelihood:**	-12637.
No. Observations:	1436	**AIC:**	2.528e+04
Df Residuals:	1431	**BIC:**	2.531e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.44e+04	413.378	34.827	0.000	1.36e+04	1.52e+04
Age	-163.6216	2.342	-69.871	0.000	-168.215	-159.028
HP	43.2136	2.878	15.016	0.000	37.569	48.859
CC	0.2937	0.101	2.914	0.004	0.096	0.491
Doors	158.3594	45.285	3.497	0.000	69.527	247.192

Omnibus:	344.053	**Durbin-Watson:**	1.317
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2507.167
Skew:	0.915	**Prob(JB):**	0.00
Kurtosis:	9.209	**Cond. No.**	1.60e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.6e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [31]:
```
model11=smf.ols("Price~Age+HP+KM+CC+Doors+QT",data=df1).fit()
model11.summary()
```

Out[31]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.839
Model:	OLS	**Adj. R-squared:**	0.838
Method:	Least Squares	**F-statistic:**	1239.
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.00
Time:	20:01:29	**Log-Likelihood:**	-12497.
No. Observations:	1436	**AIC:**	2.501e+04
Df Residuals:	1429	**BIC:**	2.504e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.295e+04	409.080	31.660	0.000	1.21e+04	1.38e+04
Age	-136.5498	2.658	-51.366	0.000	-141.765	-131.335
	27	2.858	16.000	0.000	40.126	51.339

KM	-0.0198	0.001	-14.578	0.000	-0.022	-0.017
CC	0.1028	0.097	1.063	0.288	-0.087	0.292
Doors	134.2946	41.299	3.252	0.001	53.281	215.308
QT	16.3993	1.149	14.272	0.000	14.145	18.653

Omnibus:	202.288	**Durbin-Watson:**	1.378
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	717.719
Skew:	0.665	**Prob(JB):**	1.41e-156
Kurtosis:	6.198	**Cond. No.**	8.30e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.3e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

Calculating VIF

In [32]:

```
rsq_Age = smf.ols('Age~HP+KM+Doors+QT+CC+Weight',data=data).fit().rsquared
vif_Age = 1/(1-rsq_Age)

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

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

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

rsq_Gears = smf.ols('Gears~Age+HP+Gears+KM+QT+CC+Weight',data=data).fit().rsquared
vif_Gears = 1/(1-rsq_Gears)

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

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

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

# Storing vif values in a data fram

d1 = {'Variables':['Age','HP','KM',"Doors","Gears","QT","CC",'Weight'],'VIF':[vif_Age,vif_
Vif_frame = pd.DataFrame(d1)
Vif_frame
```

Out[32]:

	Variables	VIF
0	Age	1.908532
1	HP	1.428423
2	KM	1.758792

	Variables	VIF
3	Doors	1.169183
4	Gears	inf
5	QT	2.343063
6	CC	1.168283
7	Weight	2.627265

Observation

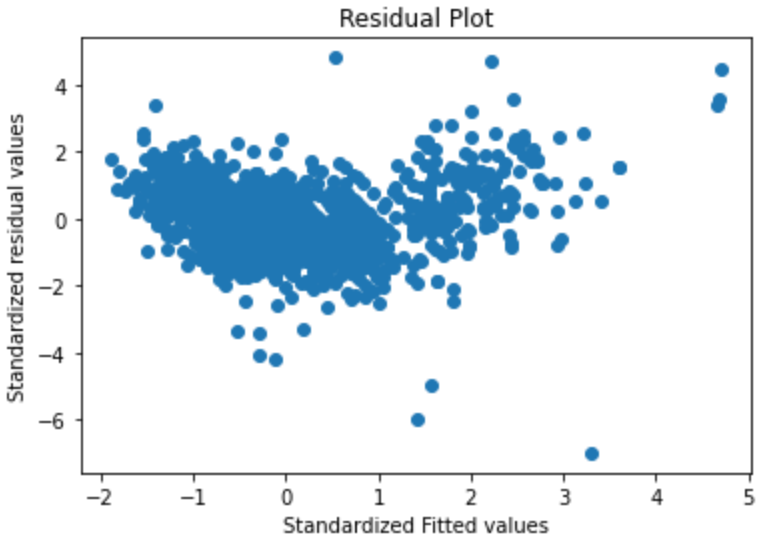
The vif of the first model is very less as we can say that the feature does not having any co-linearity

Residuals For Homscedasticity

In [33]:
```
def get_standardized_values( vals ):
    return (vals - vals.mean())/vals.std()
```

In [34]:
```
plt.scatter(get_standardized_values(model1.fittedvalues),
            get_standardized_values(model1.resid))

plt.title('Residual Plot')
plt.xlabel('Standardized Fitted values')
plt.ylabel('Standardized residual values')
plt.show()
```



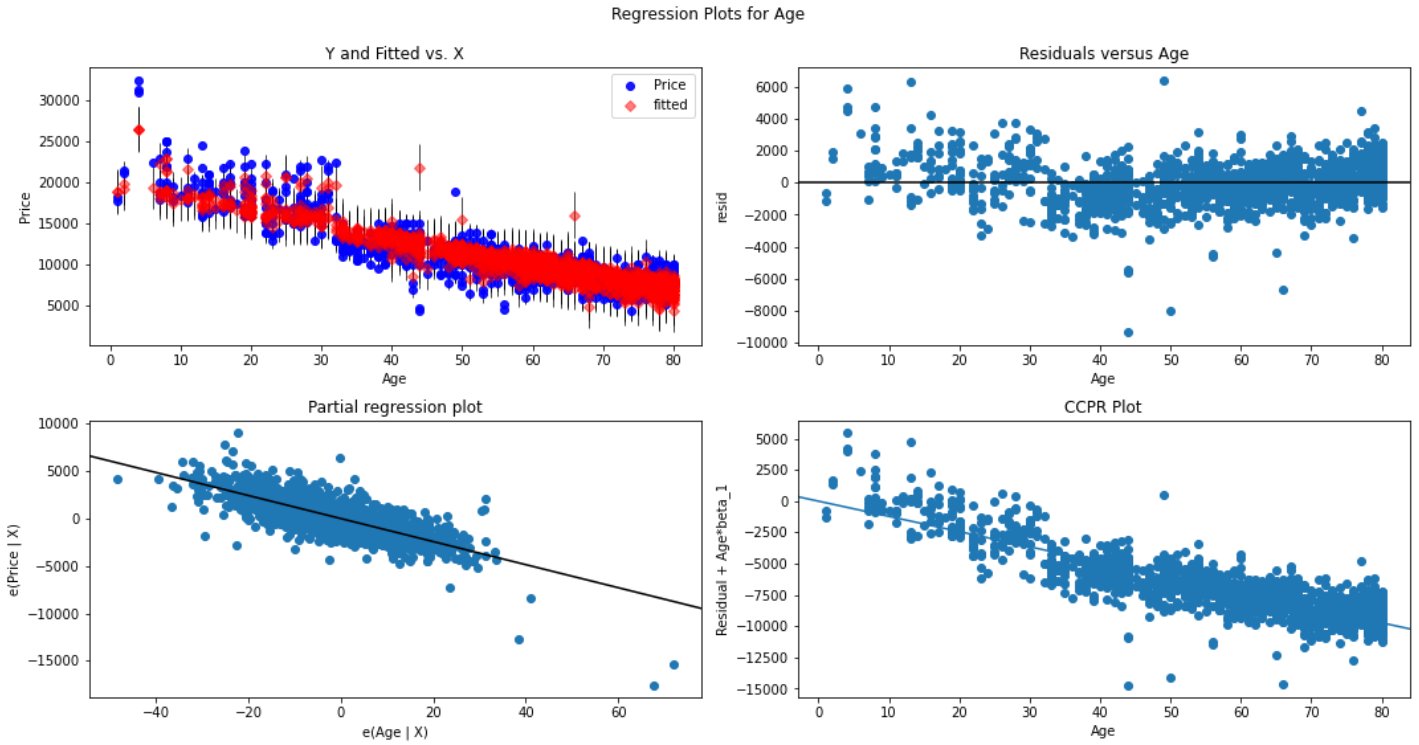
Observation

As We can see that the Features and Erros does not having co-linearity

Residual vs Regressor

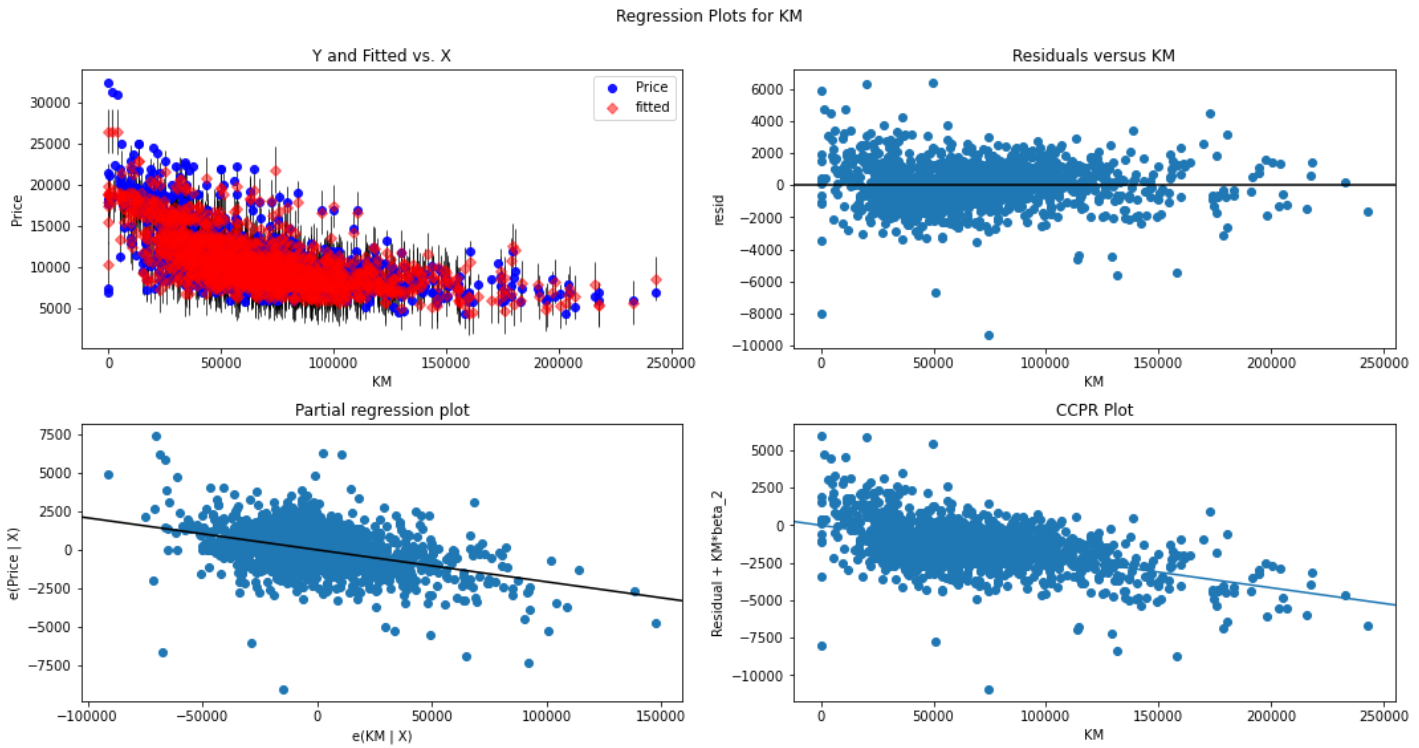
In [35]:
```
fig = plt.figure(figsize=(15,8))
hics.plot_regress_exog(model1, "Age",fig=fig)
```

```
plt.show()
```



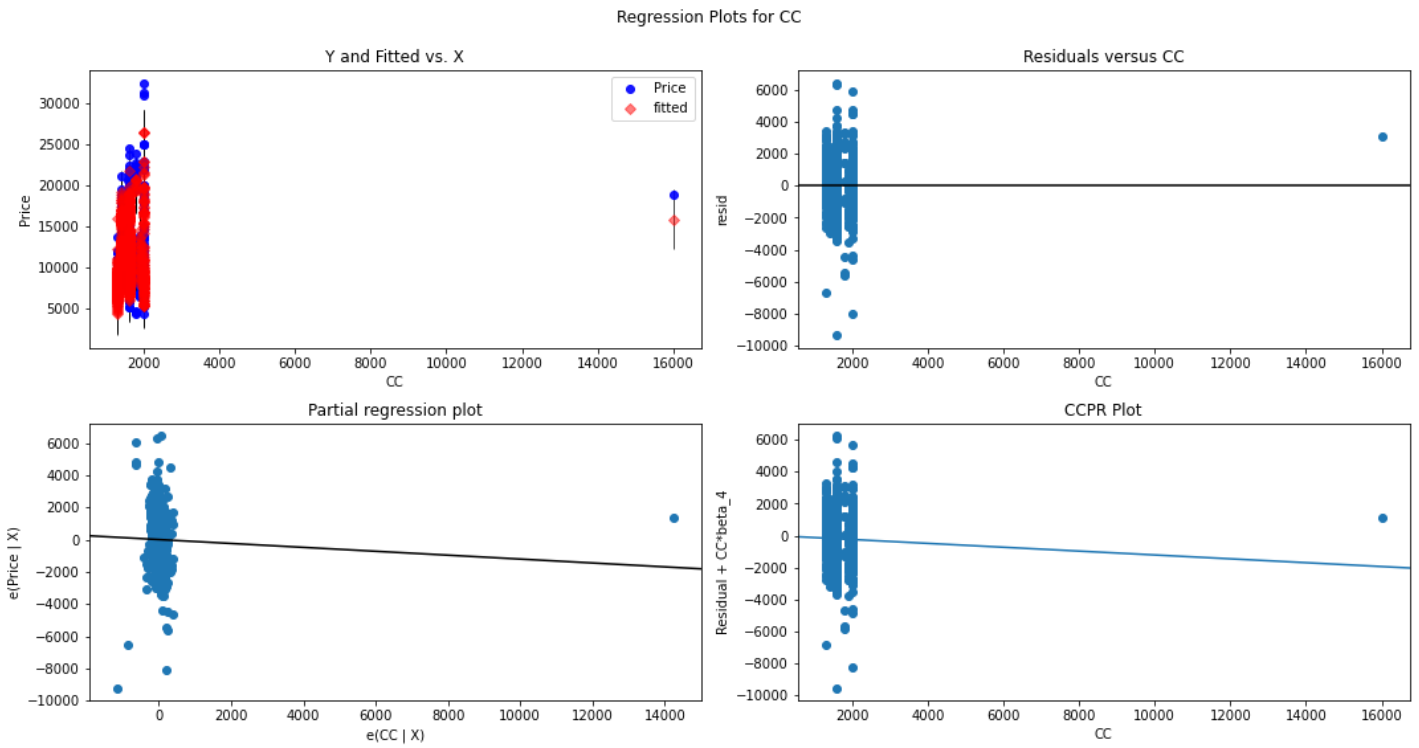
In [36]:

```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model1, "KM",fig=fig)
plt.show()
```



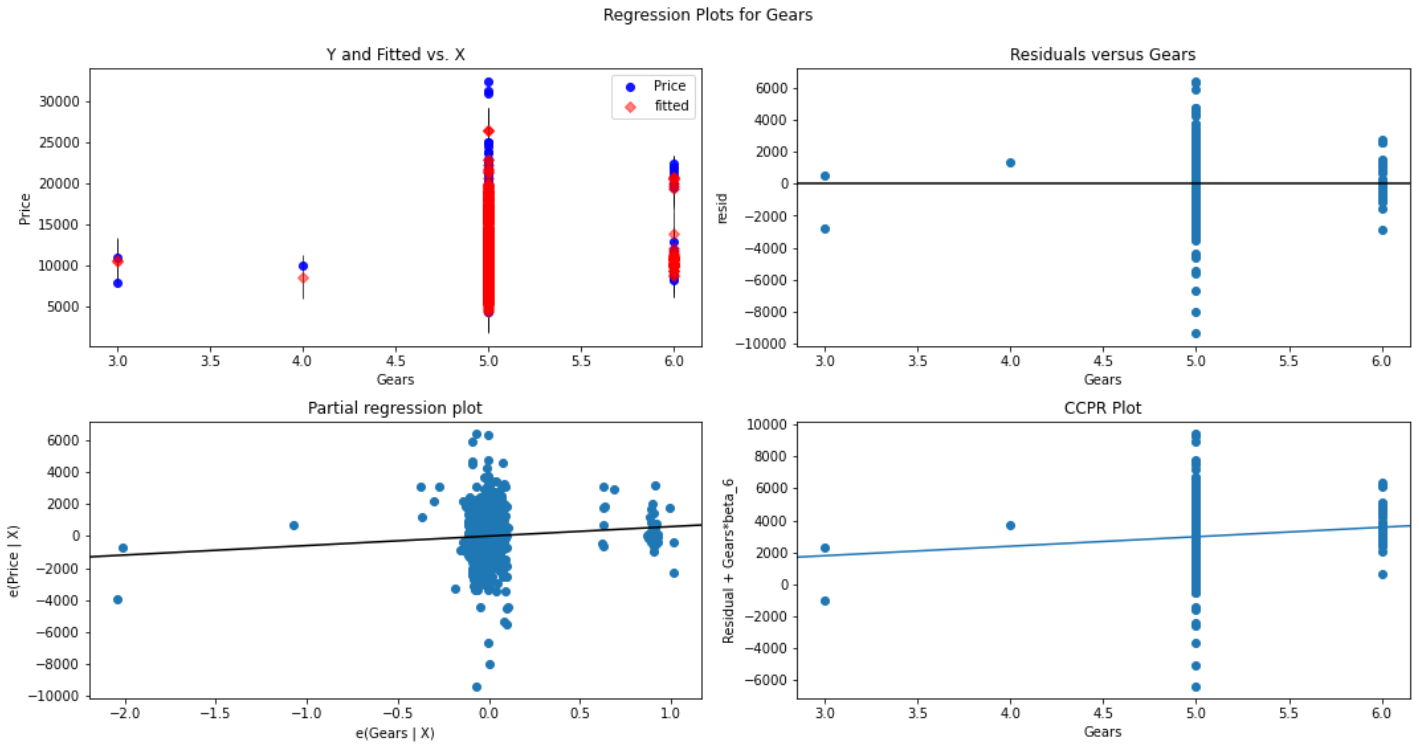
In [37]:

```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model1, "CC",fig=fig)
plt.show()
```



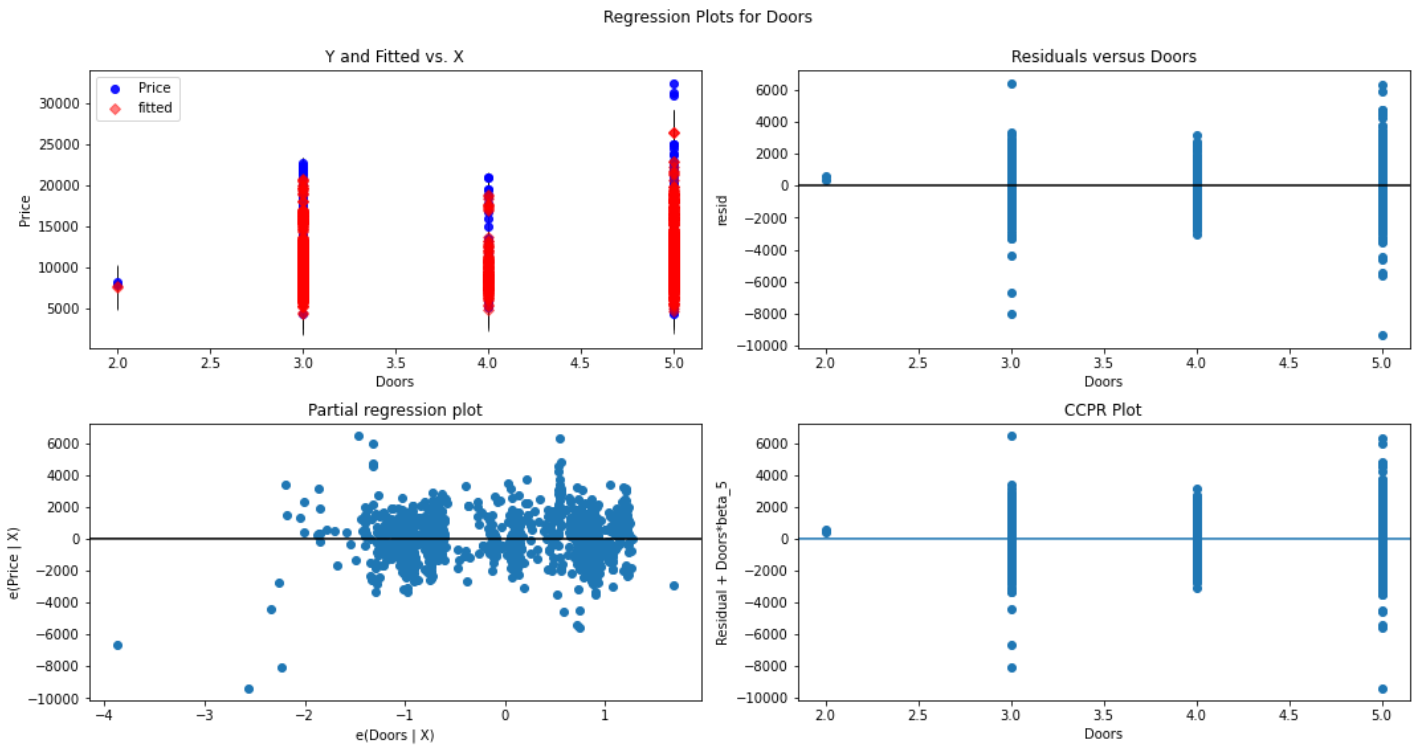
In [38]:

```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model1, "Gears",fig=fig)
plt.show()
```



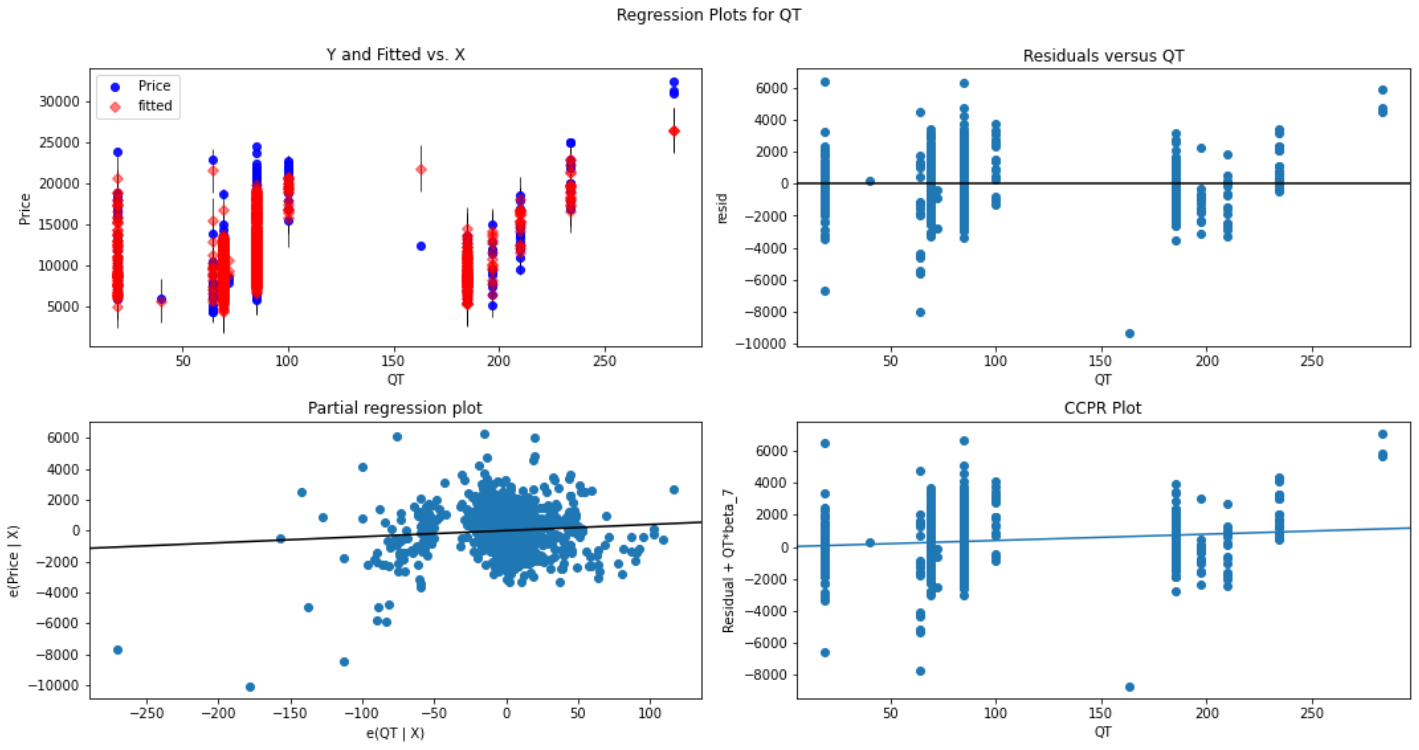
In [39]:

```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model1, "Doors",fig=fig)
plt.show()
```



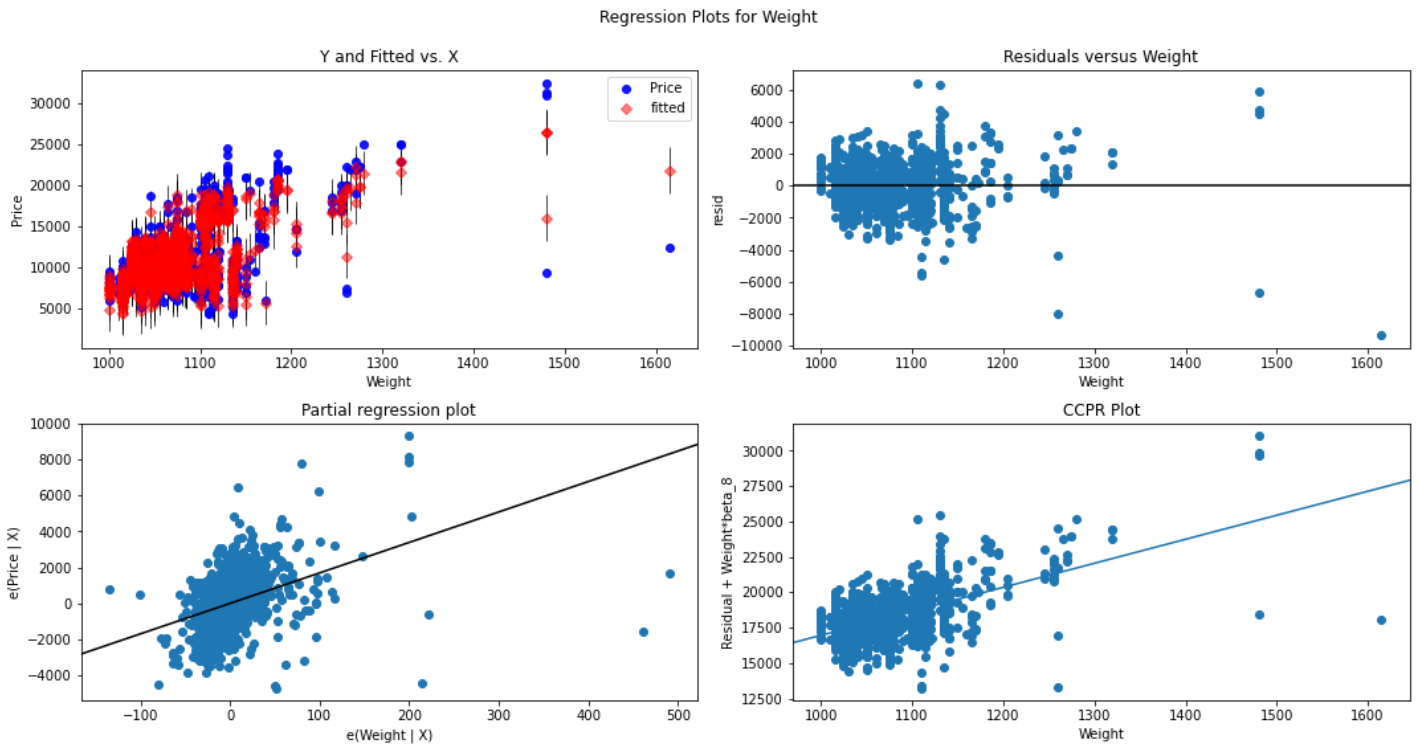
In [40]:

```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model1, "QT",fig=fig)
plt.show()
```



In [41]:

```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model1, "Weight",fig=fig)
plt.show()
```

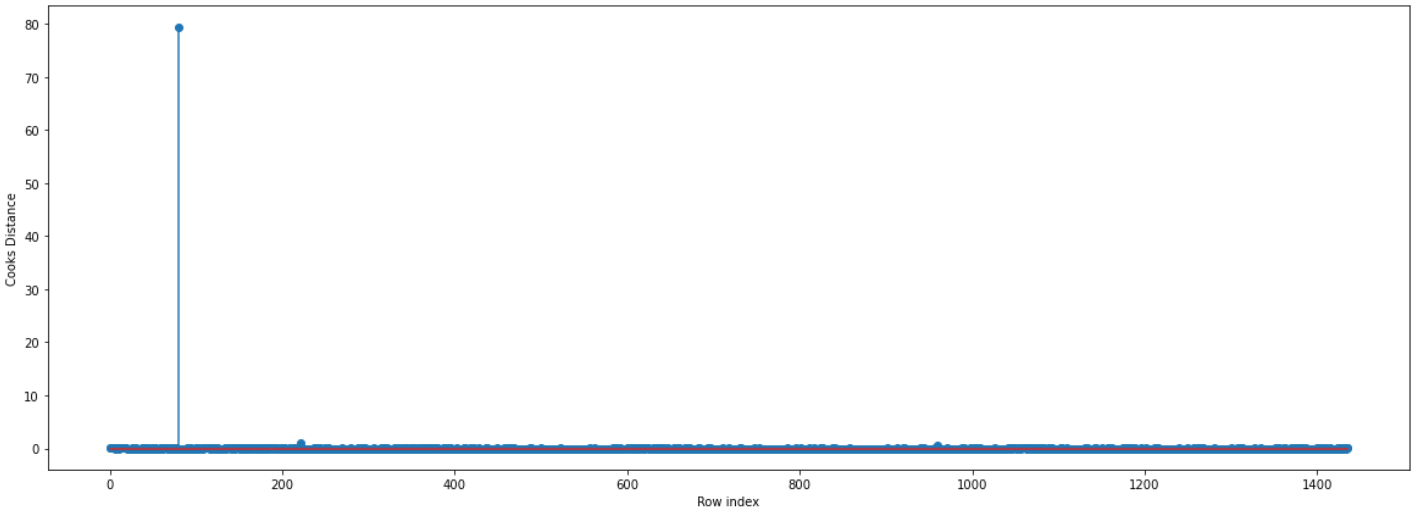


Outlier Detection

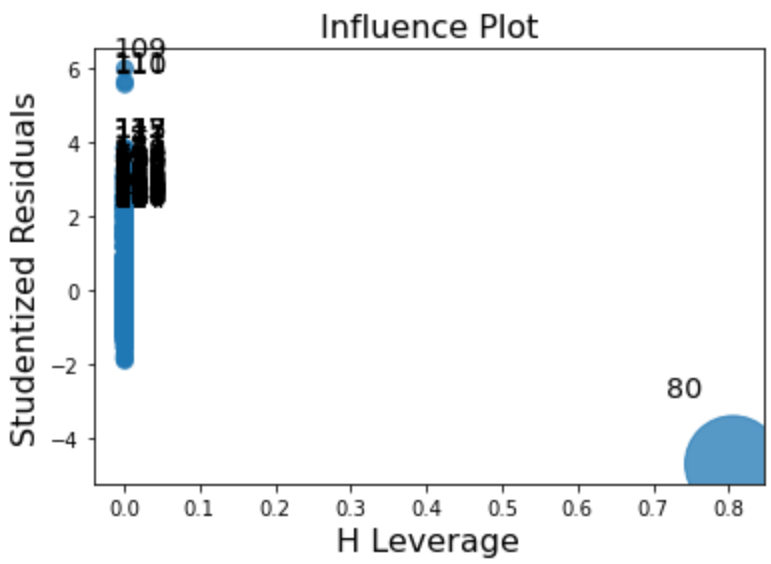
Cook's Distance

In [42]:
```
model1_influence=model1.get_influence()
(c,_)=model1_influence.cooks_distance
```

In [44]:
```
fig = plt.subplots(figsize=(20, 7))
plt.stem(np.arange(len(df1)), np.round(c, 3))
plt.xlabel('Row index')
plt.ylabel('Cooks Distance')
plt.show()
```



In [45]:
```
from statsmodels.graphics.regressionplots import influence_plot
influence_plot(model5)
plt.show()
```



In [46]:
```
k = df1.shape[1]
n = df1.shape[0]
leverage_cutoff = 3*((k + 1)/n)
leverage_cutoff
```

Out[46]:
```
0.020891364902506964
```

In [47]:
```
(np.argmax(c),np.max(c))
```

Out[47]:
```
(80, 79.5201062414182)
```

In [48]:
```
df1[df.index.isin([80])]
```

Out[48]:

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

In [49]:
```
df2=df1.copy()
df2
```

Out[49]:

	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 [50]:
```
df2=df1.drop(data.index[[80]],axis=0).reset_index(drop=True)
df2
```

Out[50]:

	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 [51]:
```
df1.shape
```

Out[51]:
```
(1436, 9)
```

Deleting Diagnostic and Improving Model

In [52]:
```
while model1.rsquared < 0.90:
    for c in [np.max(c)>0.5]:
        model1=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=df2).fit()
        (c,_)=model1.get_influence().cooks_distance
        c
        np.argmax(c) , np.max(c)
        df2=df2.drop(df2.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
        df2
    else:
        Final_Model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=df2).fit()
        Final_Model.rsquared , Final_Model.aic
        print("Thus model accuracy is improved to",Final_Model.rsquared)
```

```
Thus model accuracy is improved to 0.8778445878599779
Thus model accuracy is improved to 0.8851845904421739
Thus model accuracy is improved to 0.8894191849749752
Thus model accuracy is improved to 0.8914204825569461
Thus model accuracy is improved to 0.8921467826162199
Thus model accuracy is improved to 0.8934037497368835
Thus model accuracy is improved to 0.8944954473640402
Thus model accuracy is improved to 0.8958333530393431
Thus model accuracy is improved to 0.8966334481080778
Thus model accuracy is improved to 0.8941835374074523
Thus model accuracy is improved to 0.8951069956497651
Thus model accuracy is improved to 0.8965514940097824
Thus model accuracy is improved to 0.8971531368893257
```

```
Thus model accuracy is improved to 0.8979680072945878
Thus model accuracy is improved to 0.8975421185310549
Thus model accuracy is improved to 0.8970095523065386
Thus model accuracy is improved to 0.8965434597967323
Thus model accuracy is improved to 0.8961255789744987
Thus model accuracy is improved to 0.8972646982898881
Thus model accuracy is improved to 0.896772534985454
Thus model accuracy is improved to 0.8971051472272296
Thus model accuracy is improved to 0.8946150256249212
Thus model accuracy is improved to 0.8920509075936639
Thus model accuracy is improved to 0.8912123567698701
Thus model accuracy is improved to 0.8916315159679491
Thus model accuracy is improved to 0.892106991023533
Thus model accuracy is improved to 0.8914218579661384
Thus model accuracy is improved to 0.8908027748123676
Thus model accuracy is improved to 0.8908234861287034
Thus model accuracy is improved to 0.8913569064732925
Thus model accuracy is improved to 0.8908377692804937
Thus model accuracy is improved to 0.8910483674438952
Thus model accuracy is improved to 0.8904069445135181
Thus model accuracy is improved to 0.8912195094765825
Thus model accuracy is improved to 0.8918908091467392
Thus model accuracy is improved to 0.8918846814654162
Thus model accuracy is improved to 0.8920353748947274
Thus model accuracy is improved to 0.892320259522047
Thus model accuracy is improved to 0.8926128036349465
Thus model accuracy is improved to 0.8931516569229079
Thus model accuracy is improved to 0.8939745734629716
Thus model accuracy is improved to 0.8941834012715583
Thus model accuracy is improved to 0.8946455906441961
Thus model accuracy is improved to 0.8949886776774434
Thus model accuracy is improved to 0.8957699564694306
Thus model accuracy is improved to 0.8960457779697314
Thus model accuracy is improved to 0.8954151798855725
Thus model accuracy is improved to 0.89477879772334
Thus model accuracy is improved to 0.8955282129976623
Thus model accuracy is improved to 0.8948850194253113
Thus model accuracy is improved to 0.8955482090594036
Thus model accuracy is improved to 0.8948889282342523
Thus model accuracy is improved to 0.8954399054524788
Thus model accuracy is improved to 0.8941322445227955
Thus model accuracy is improved to 0.8928003536066337
Thus model accuracy is improved to 0.8929670769169252
Thus model accuracy is improved to 0.8932646420947161
Thus model accuracy is improved to 0.8938479268106043
Thus model accuracy is improved to 0.8946116437234872
Thus model accuracy is improved to 0.8947373476692219
Thus model accuracy is improved to 0.8950379522236931
Thus model accuracy is improved to 0.8948527054861177
Thus model accuracy is improved to 0.8953342154393777
Thus model accuracy is improved to 0.8955984391805881
Thus model accuracy is improved to 0.8959887924407284
Thus model accuracy is improved to 0.8949134651663617
Thus model accuracy is improved to 0.8951494863024941
Thus model accuracy is improved to 0.8955005725113694
Thus model accuracy is improved to 0.8957248584395654
Thus model accuracy is improved to 0.8954298292582191
Thus model accuracy is improved to 0.8953618506400354
Thus model accuracy is improved to 0.8956028020870667
Thus model accuracy is improved to 0.8959330397949152
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 0.8964643118227557
Thus model accuracy is improved to 0.8968869273251255
Thus model accuracy is improved to 0.8965702611797324
Thus model accuracy is improved to 0.896465648982765
Thus model accuracy is improved to 0.8967541657126812
Thus model accuracy is improved to 0.8970295712331846
```

```
Thus model accuracy is improved to 0.8970512860226794
Thus model accuracy is improved to 0.8973242631620858
Thus model accuracy is improved to 0.8976064036306222
Thus model accuracy is improved to 0.8965027704321633
Thus model accuracy is improved to 0.8969285366970443
Thus model accuracy is improved to 0.8970144437559693
Thus model accuracy is improved to 0.8974646183589379
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 0.8982290469246597
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 0.8995264042658646
Thus model accuracy is improved to 0.8998346697166659
Thus model accuracy is improved to 0.8999704768778106
Thus model accuracy is improved to 0.9002238270483123
Thus model accuracy is improved to 0.9003762532318559
```

In [54]:
```
Final_Model.rsquared # The Model Accuracy has been Increased
```

Out[54]:
```
0.9003762532318559
```

MSE

In [55]:
```
Final_Model.mse_resid
```

Out[55]:
```
952433.7824446883
```

RMSE

In [56]:
```
np.sqrt(Final_Model.mse_resid)
```

Out[56]:
```
975.9271399262797
```

Predicting New values

In [57]:
```
values = pd.DataFrame({"Age":26,"KM":20000,"HP":90,"CC":2000,"Gears":5,"Doors":3,"QT":200,
values
```

Out[57]:

	Age	KM	HP	CC	Gears	Doors	QT	Weight
0	26	20000	90	2000	5	3	200	1090

In [58]:
```
pd.DataFrame(Final_Model.predict(values),columns=["Price"])
```

Out[58]:

	Price
0	13191.183227

Predicting Automatic Prices

In [59]:
```
pred_y = Final_Model.predict(df2)
pd.DataFrame(pred_y,columns=["New_Price"])
```

Out[59]:

	New_Price
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

1330 rows × 1 columns

In [ ]:

In [ ]: