Importing Libraries

In [1]:
```
# Supress Warnings

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

In [2]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as mn
import scipy.stats as stat
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.graphics.regressionplots import influence_plot
%matplotlib inline
```

Problem Statement

Consider only the below columns and prepare a prediction model for predicting Price.

- **Corolla<-**
  Corolla[c("Price","Age_08_04","KM","HP","cc","Doors","Gears","Quarterly_Tax","Weight")]

Model -- model of the car
Price -- Offer Price in EUROs
Age_08_04 -- Age in months as in August 2004
KM -- Accumulated Kilometers on odometer
HP -- Horse Power
cc -- Cylinder Volume in cubic centimeters
Doors -- Number of doors
Gears -- Number of gear positions
Quarterly_Tax -- Quarterly road tax in EUROs
Weight -- Weight in Kilograms

Importing Dataset

In [3]:
```
raw_data=pd.read_csv('ToyotaCorolla.csv',encoding='latin1')
raw_data.head()
```

Out[3]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central_Lock	P
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]:
```
print('Number of Rows{}Columns'.format(raw_data.shape))
```

```
Number of Rows(1436, 38)Columns
```

In [4]:
```
raw_data=raw_data[["Price","Age_08_04","KM","HP","cc","Doors","Gears","Quarterly_Tax","Wei
raw_data
```

Out[4]:

	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
		70	16916	86	1300	3	5	69	1015

	Price	Age_08_04	KM	HP	cc	Doors	Gears	Quarterly_Tax	Weight
1435	6950	76	1	110	1600	5	5	19	1114

1436 rows × 9 columns

Descriptive Analysis

In [6]:
```
raw_data.describe()
```

Out[6]:

	Price	Age_08_04	KM	HP	cc	Doors	Gears	Quarterly_
count	1436.000000	1436.000000	1436.000000	1436.000000	1436.00000	1436.000000	1436.000000	1436.000
mean	10730.824513	55.947075	68533.259749	101.502089	1576.85585	4.033426	5.026462	87.122
std	3626.964585	18.599988	37506.448872	14.981080	424.38677	0.952677	0.188510	41.128
min	4350.000000	1.000000	1.000000	69.000000	1300.00000	2.000000	3.000000	19.000
25%	8450.000000	44.000000	43000.000000	90.000000	1400.00000	3.000000	5.000000	69.000
50%	9900.000000	61.000000	63389.500000	110.000000	1600.00000	4.000000	5.000000	85.000
75%	11950.000000	70.000000	87020.750000	110.000000	1600.00000	5.000000	5.000000	85.000
max	32500.000000	80.000000	243000.000000	192.000000	16000.00000	5.000000	6.000000	283.000

^Observation: There are some missing values in the Data set by reading Counts from Above

Checking for Data types

In [91]:
```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Price          1436 non-null   int64  
 1   Age_08_04      1436 non-null   int64  
 2   KM             1436 non-null   int64  
 3   HP             1434 non-null   float64
 4   cc             1436 non-null   int64  
 5   Doors          1436 non-null   int64  
 6   Gears          1436 non-null   int64  
 7   Quarterly_Tax  1436 non-null   int64  
 8   Weight         1436 non-null   int64  
dtypes: float64(1), int64(8)
memory usage: 101.1 KB
```

^Observation: all the data types are correct .

Renaming the columns name and making it short

In [5]:
```
data=raw_data.rename({'Age_08_04':'Age','cc':'CC','Quarterly_Tax':'QT'},axis=1)
data.head()
```

Out[5]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	13500	23	46986	90	2000	3	5	210	1165

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
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

Checking for missing values

In [61]:
```
data[data.values==0.0]
```

Out[61]:

Price	Age	KM	HP	CC	Doors	Gears	QT	Weight

^Observation: Notice there are no '0' values in the dataset

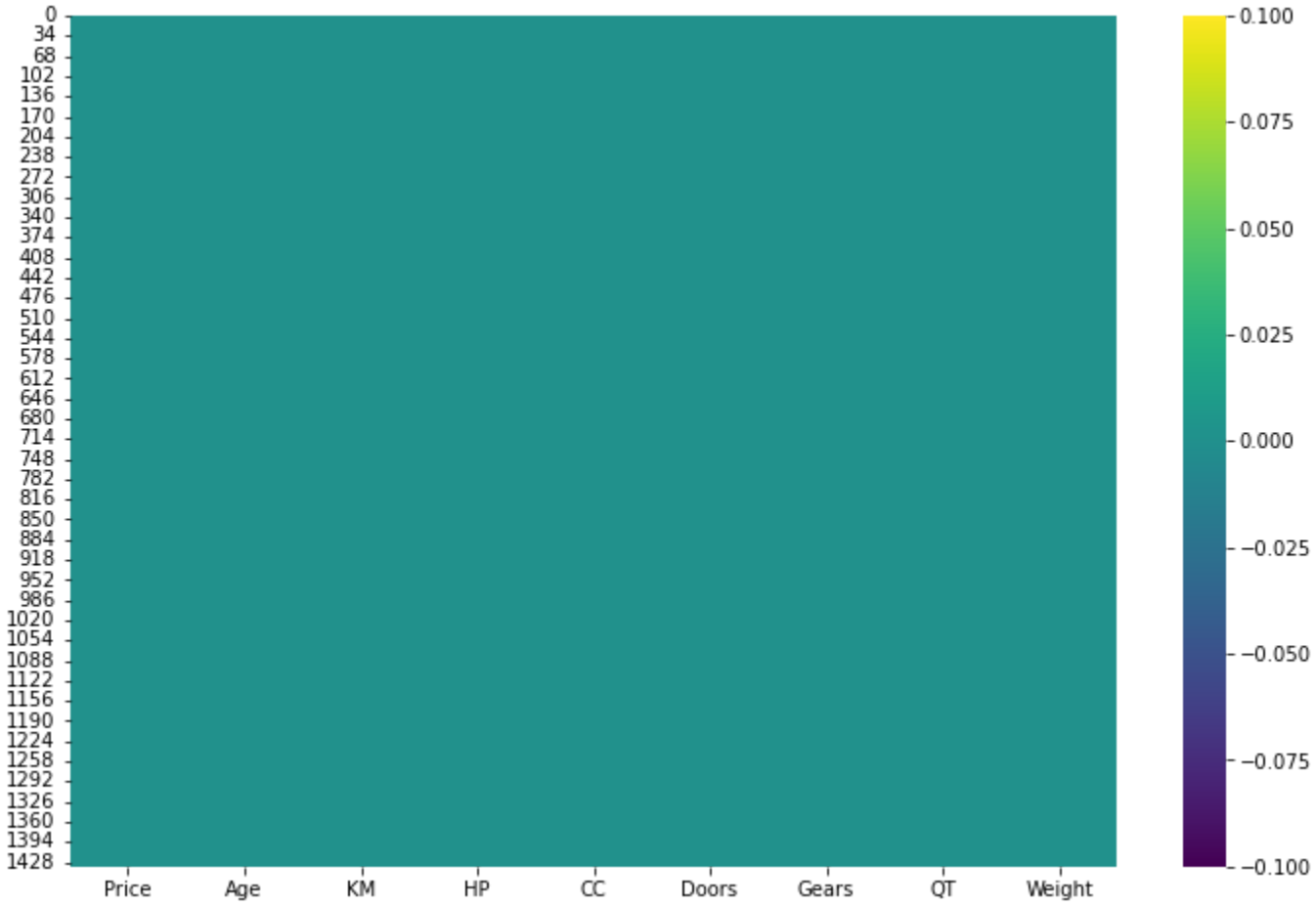
In [6]:
```
data.isnull().sum()
```

Out[6]:
```
Price     0
Age       0
KM        0
HP        0
CC        0
Doors     0
Gears     0
QT        0
Weight    0
dtype: int64
```

Visualizing Missing Values

In [63]:
```
plt.figure(figsize=(12,8))
sns.heatmap(data.isnull(),cmap='viridis')
```

Out[63]:
```
<AxesSubplot:>
```



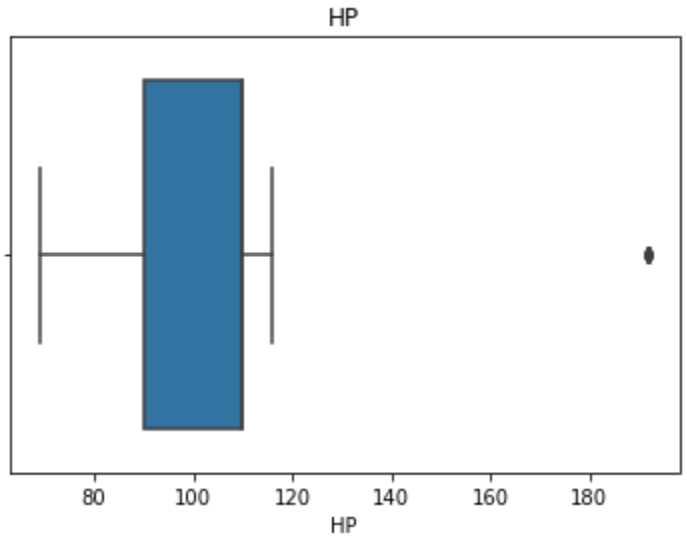
^Observation: Feature 'HP' has missing Values in the data set

- **We will have to handle the missing values by observing the distribution and making the optimal choice**

In [7]:
```
data.HP.unique()
```

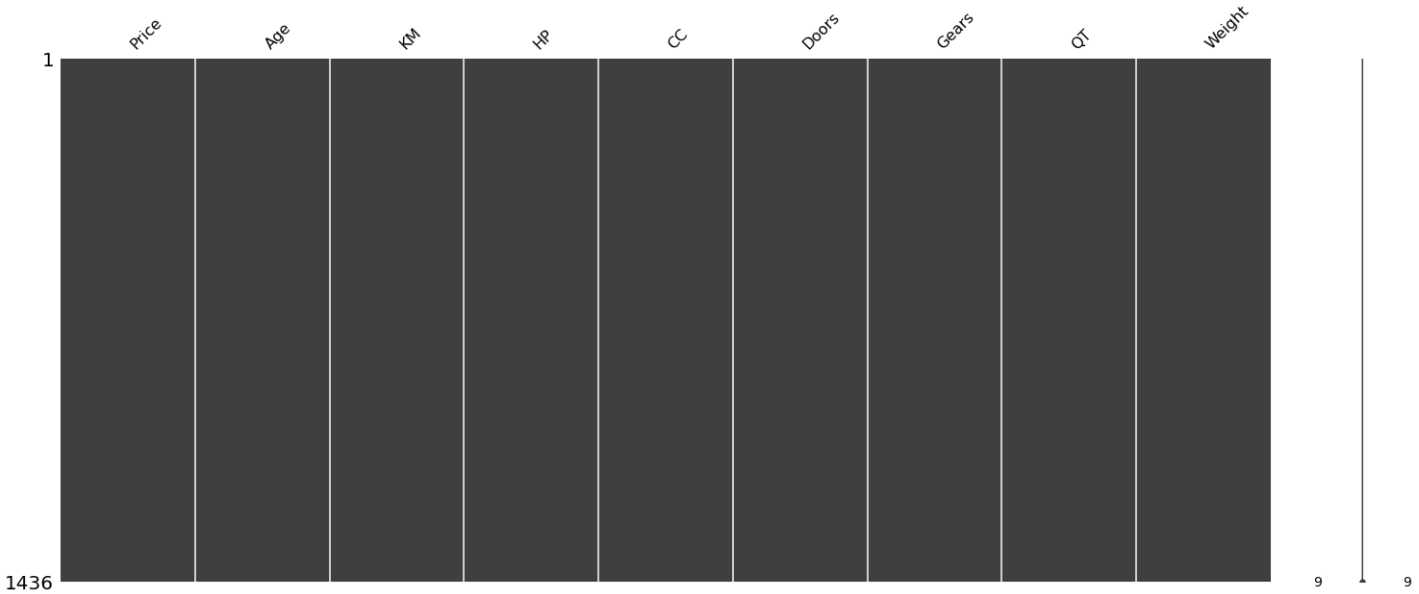
Out[7]:
```
array([ 90, 192,  69, 110,  97,  71, 116,  98,  86,  72, 107,  73],
      dtype=int64)
```

In [65]:
```
sns.boxplot(data['HP'])
plt.title('HP')
plt.show()
```



In [66]:
```
mn.matrix(data)
```

Out[66]:
```
<AxesSubplot:>
```



^Observation: After checking above there is no null value present in the dataset

Checking for Duplicated Values

In [72]:
```
data[data.duplicated()].shape
```

Out[72]:
```
(1, 9)
```

In [6]:
```
data[data.duplicated()]
```

Out[6]:

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

In [7]:
```
data=data.drop_duplicates().reset_index(drop=True)
data[data.duplicated()]
```

Out[7]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight

^Observation: There are duplicated values in the dataset

- **Hence, we dropped those values**

Let's find how many discrete and continuous feature are their in our dataset by seperating them in variables

In [8]:
```
discrete_feature=[feature for feature in data.columns if len(data[feature].unique())<20 ar
print('Discrete Variables Count: {}'.format(len(discrete_feature)))
```

```
Discrete Variables Count: 5
```

In [9]:
```
continuous_feature=[feature for feature in data.columns if data[feature].dtype!='O' and fe
uous Feature Count {}'.format(len(continuous_feature)))
```

```
Continuous Feature Count 4
```

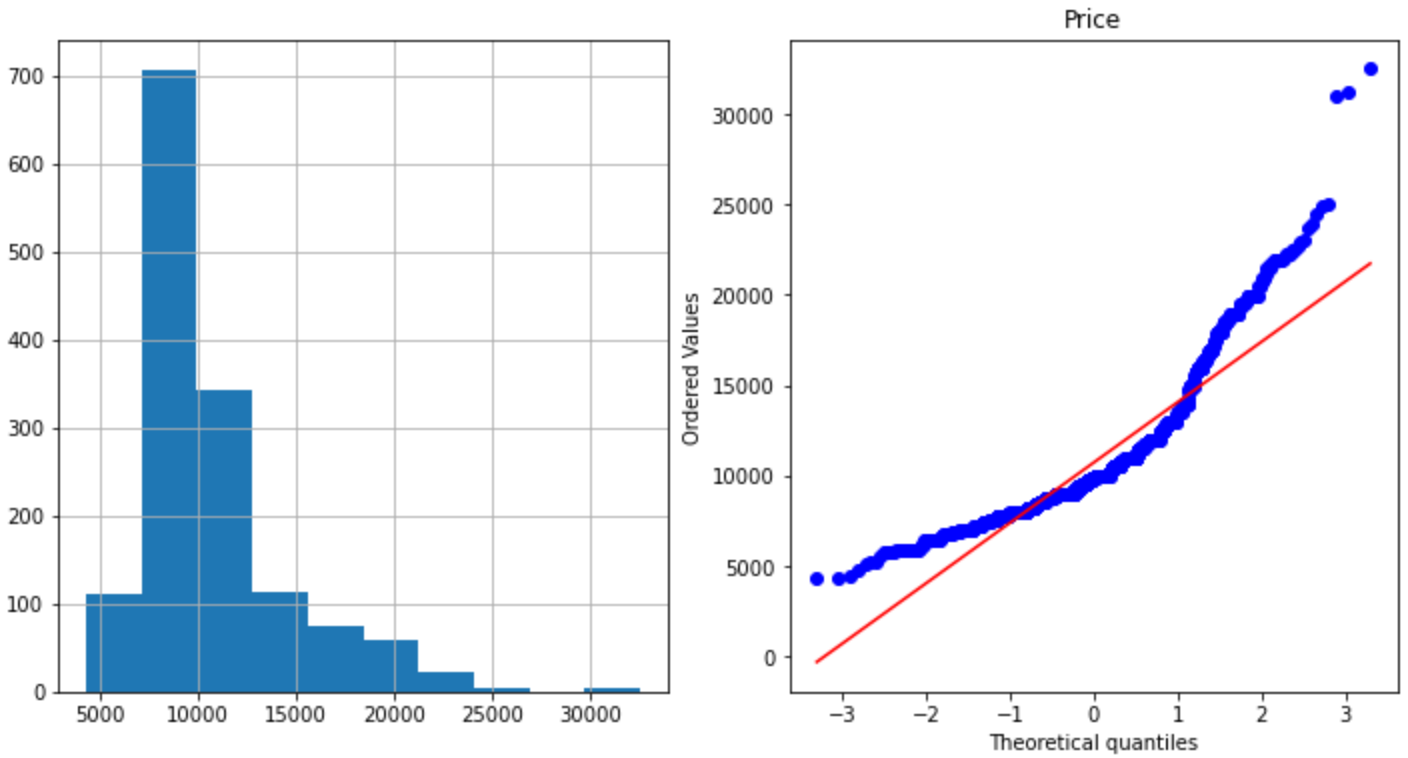
Exploratory Data Anlaysis

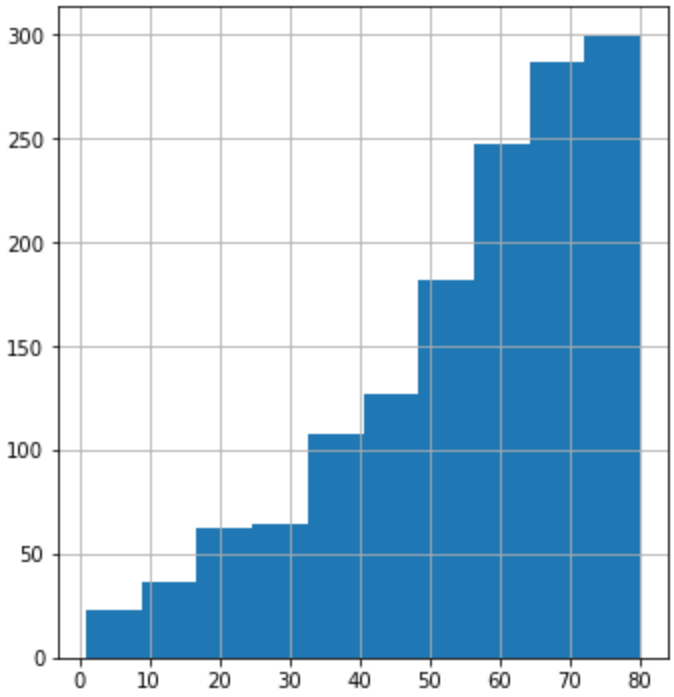
Visualizing the Distribution of Continuous Features with the help of Histograms and Probability Plot

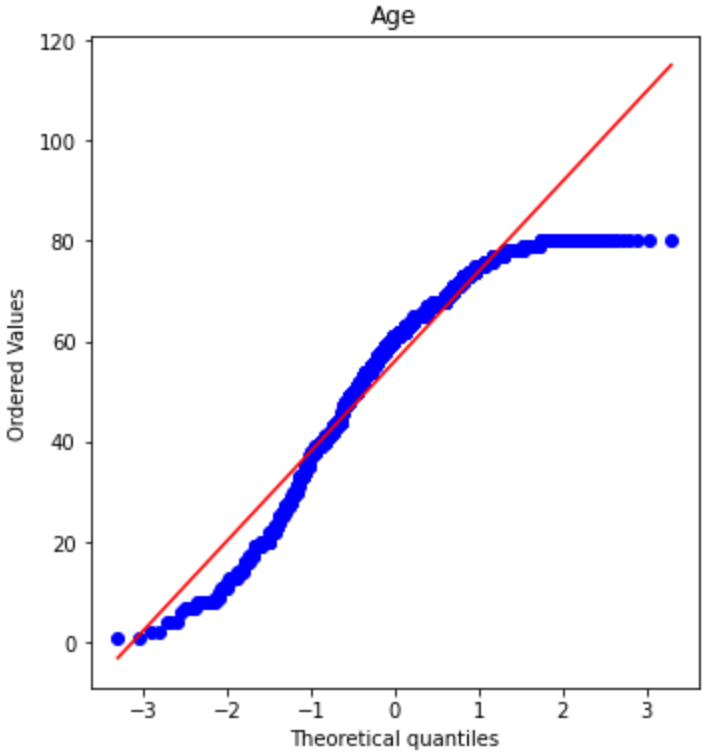
In [12]:
```
import pylab
def plot_data(data,feature):
    plt.figure(figsize=(12,6))
    plt.subplot(1,2,1)
    data[feature].hist()
    plt.subplot(1,2,2)
    stat.probplot(data[feature],dist='norm',plot=pylab)
```

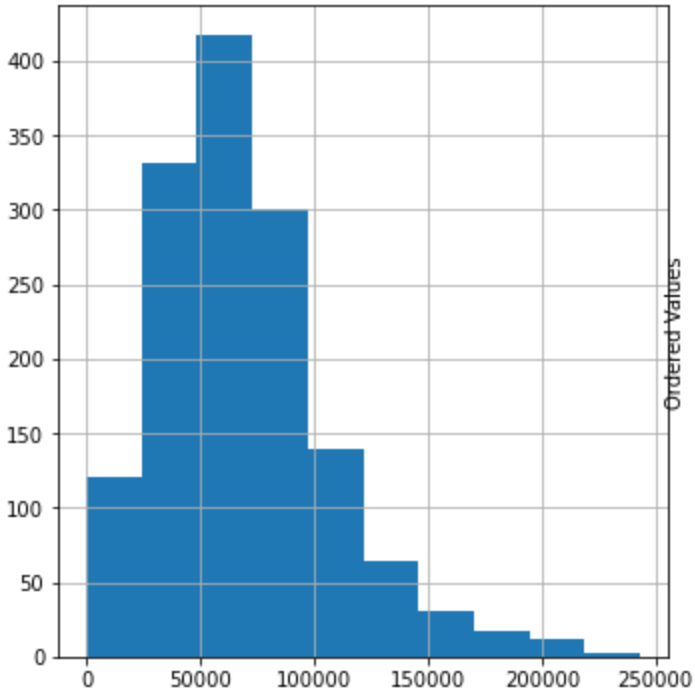
In [76]:
```
plot_data(data,'Price')
plt.title('Price')
plot_data(data,'Age')
plt.title('Age')
plot_data(data,'KM')
plt.title('KM')
plot_data(data,'Weight')
plt.title('Weight')
```

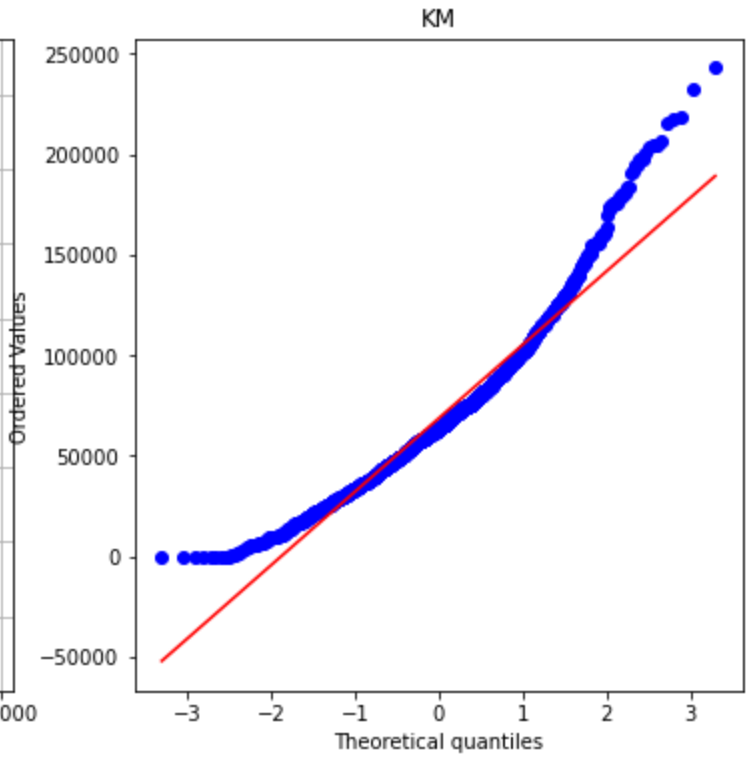
Out[76]:
```
Text(0.5, 1.0, 'Weight')
```

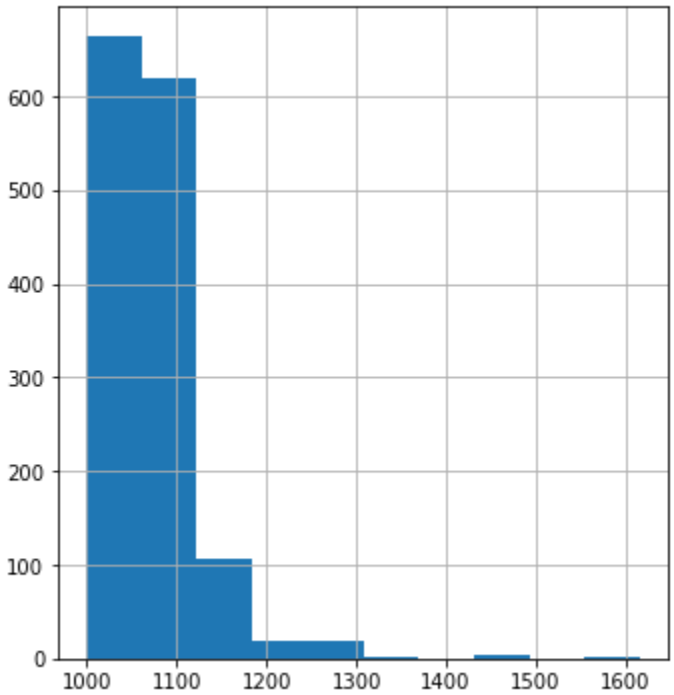


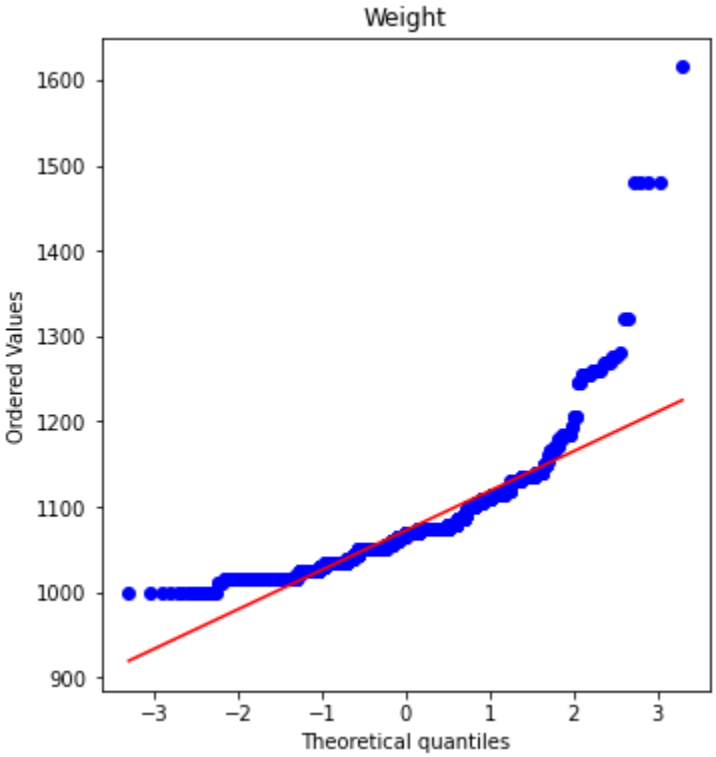








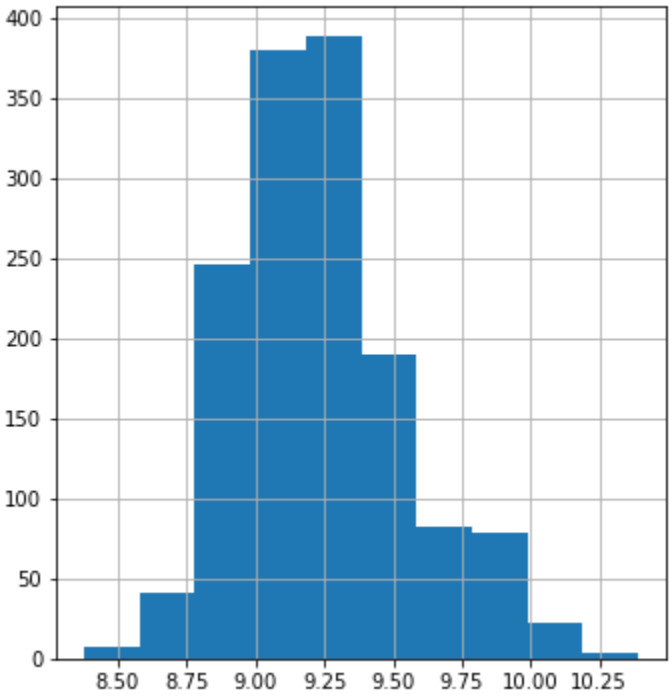


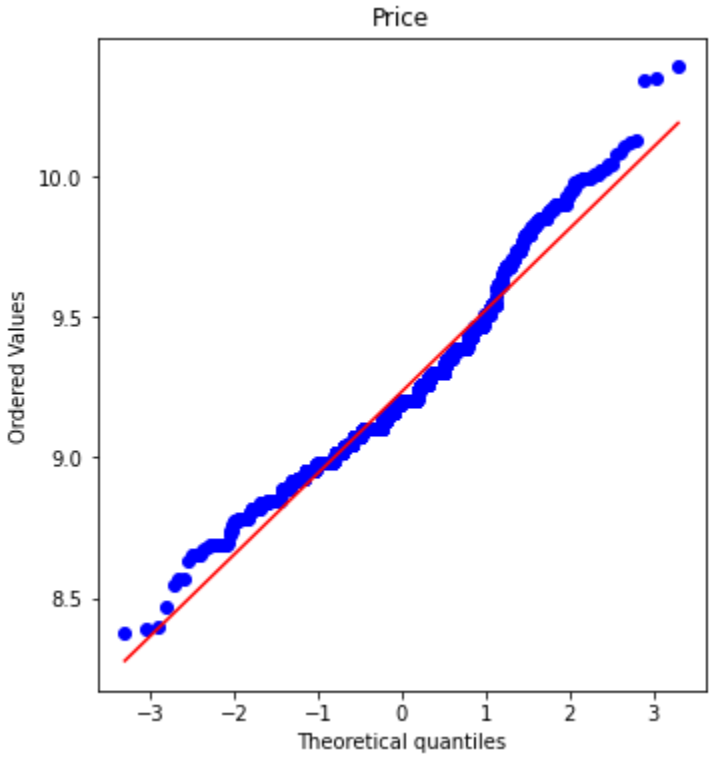


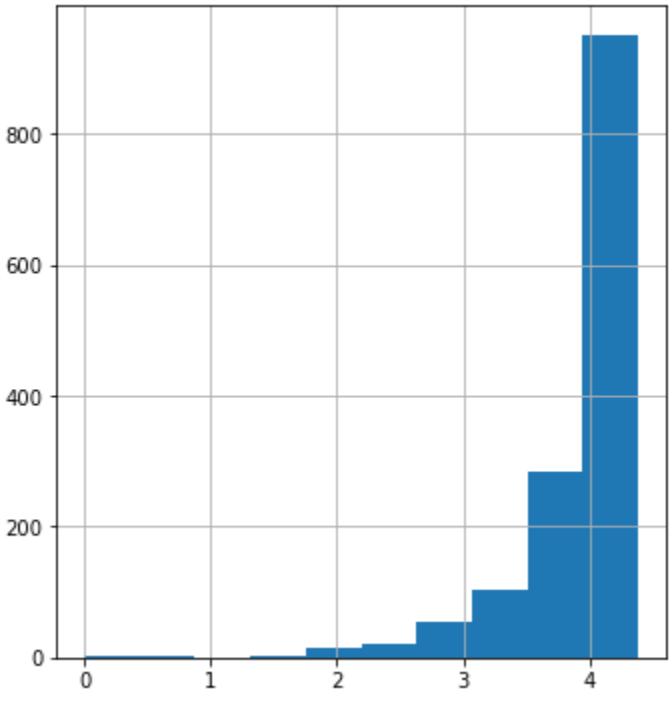
Log transformation and visualizing the Histogram to determine any possible changes in distribution

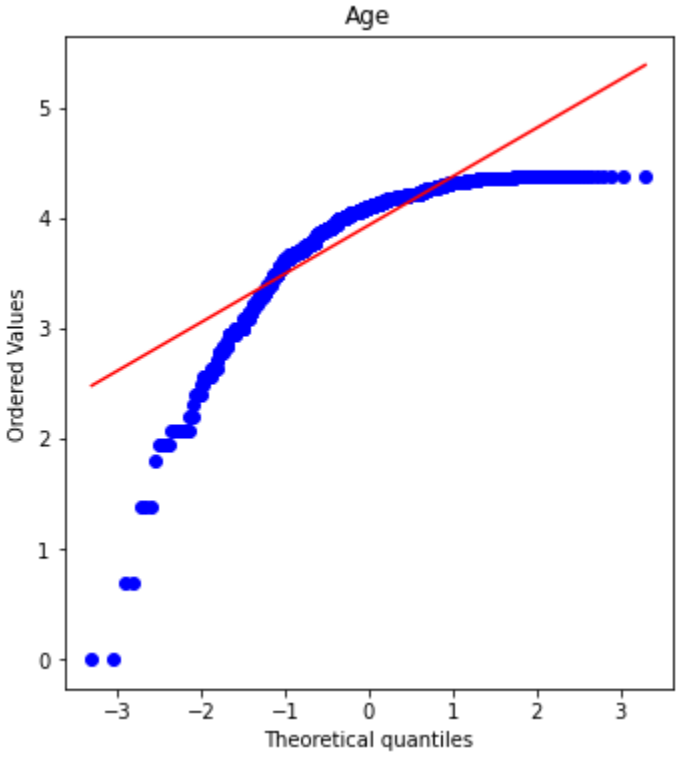
In [77]:
```
df=data.copy()
df[continuous_feature]=np.log(df[continuous_feature])

plot_data(df,'Price')
plt.title('Price')
plot_data(df,'Age')
plt.title('Age')
plot_data(df,'KM')
plt.title('KM')
plot_data(df,'Weight')
plt.title('Weight')
```

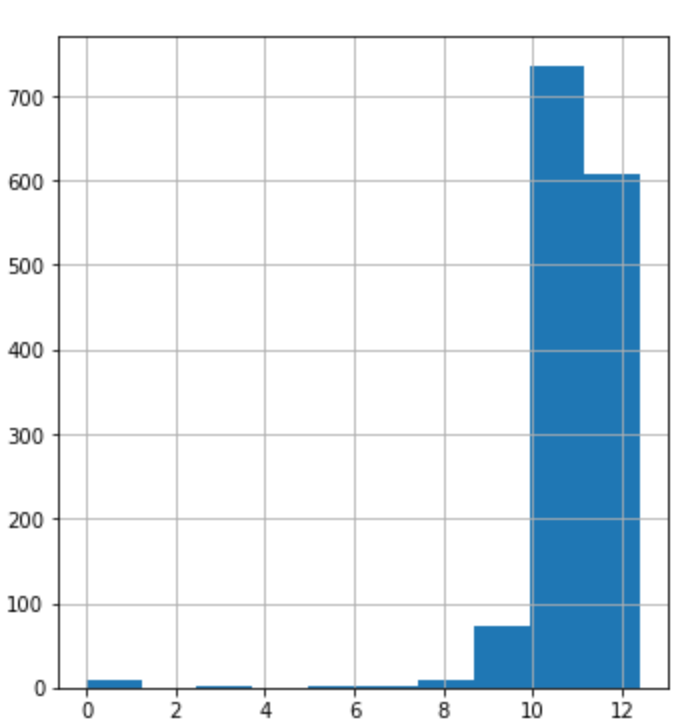
Out[77]:
```
Text(0.5, 1.0, 'Weight')
```

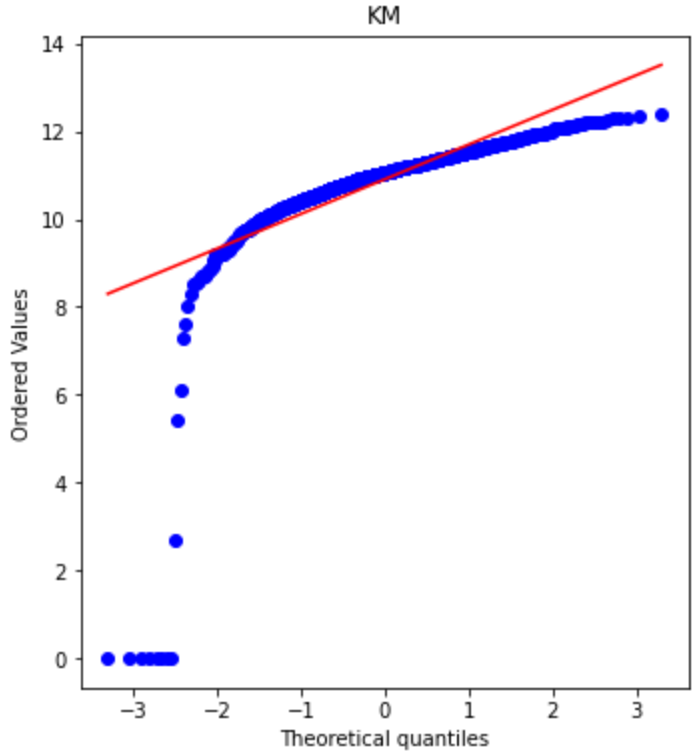


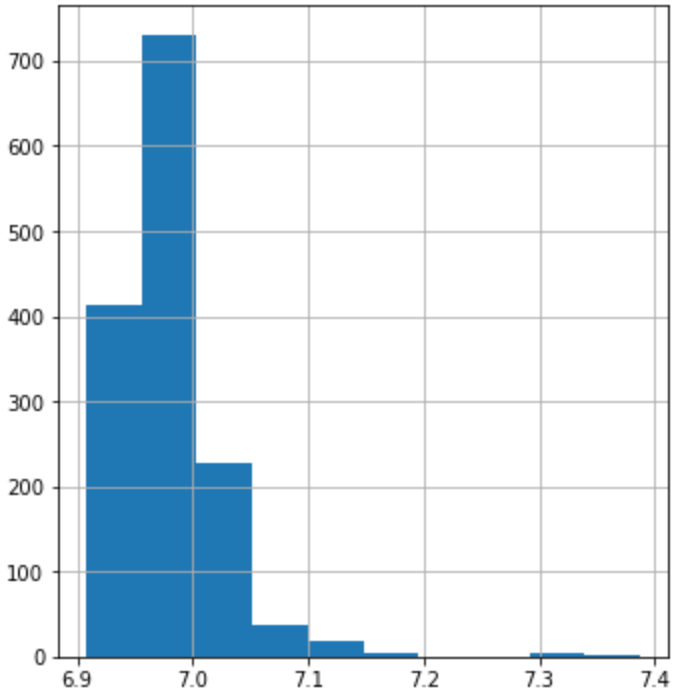


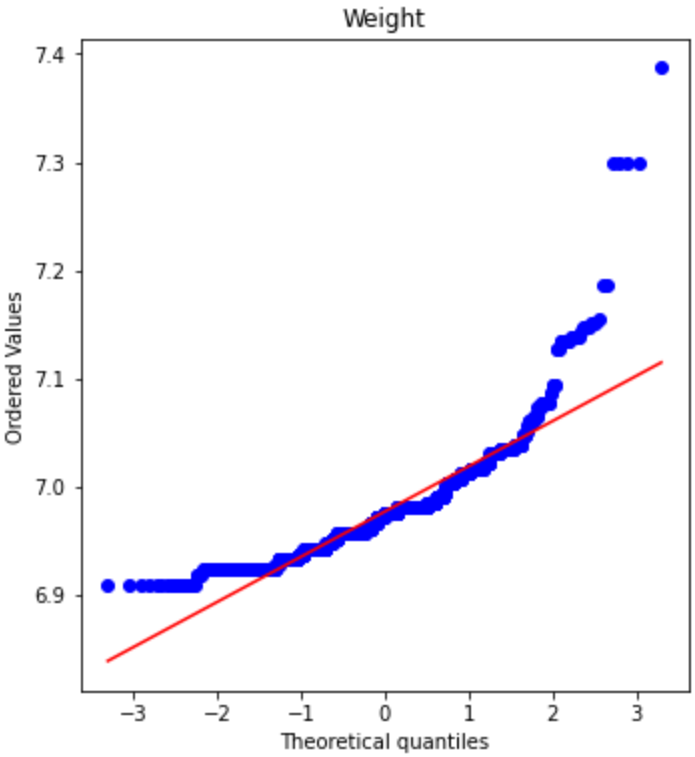










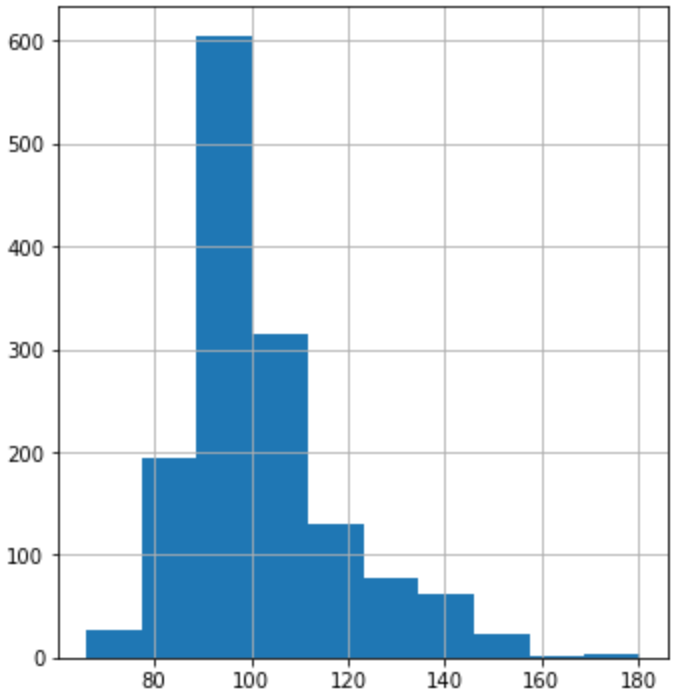


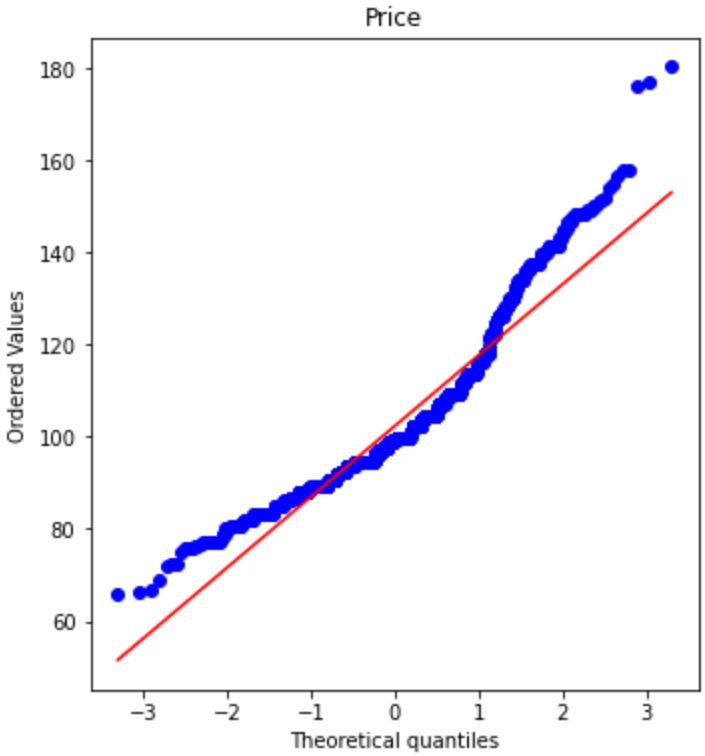
Square root transformation and visualizing the Histogram to determine any possible changes in distribution

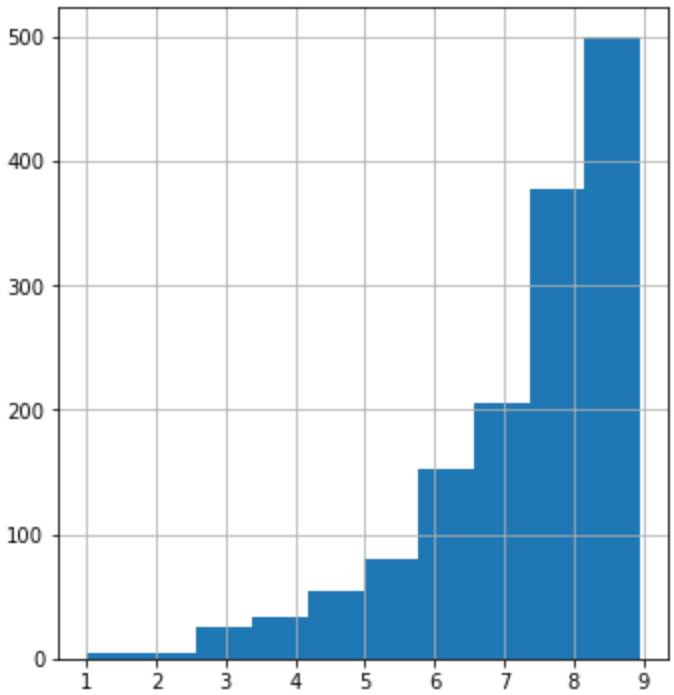
In [78]:

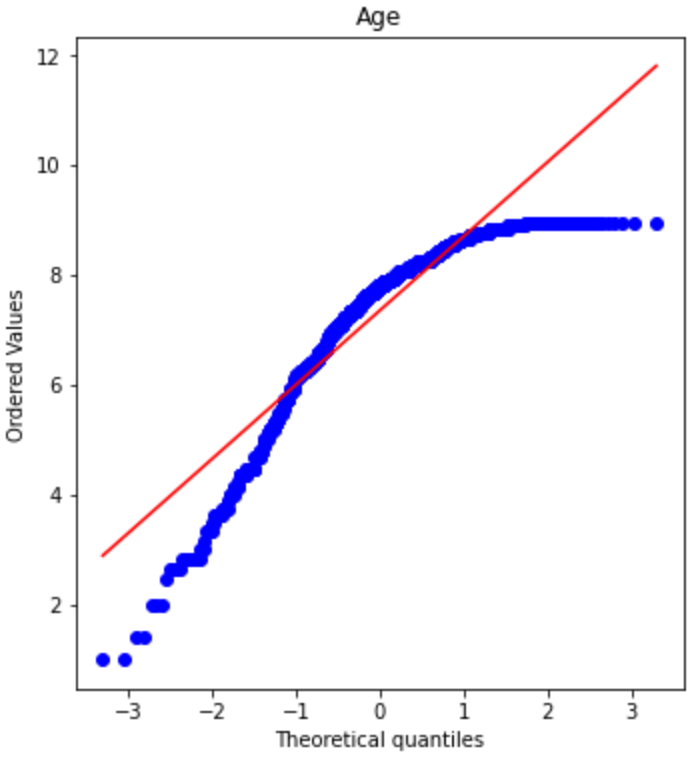
```
df=data.copy()
df[continuous_feature]=np.sqrt(df[continuous_feature])

plot_data(df,'Price')
plt.title('Price')
plot_data(df,'Age')
plt.title('Age')
plot_data(df,'KM')
plt.title('KM')
plot_data(df,'Weight')
plt.title('Weight')
```

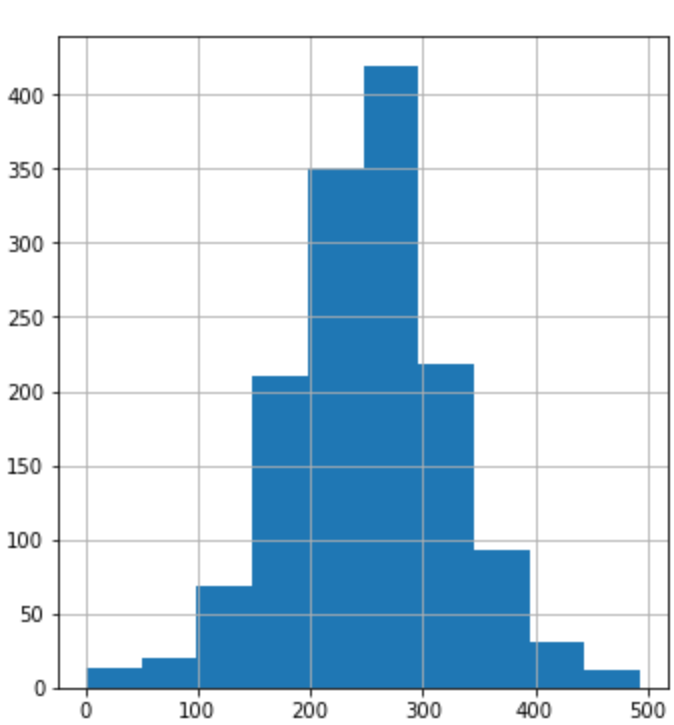
Out[78]: Text(0.5, 1.0, 'Weight')

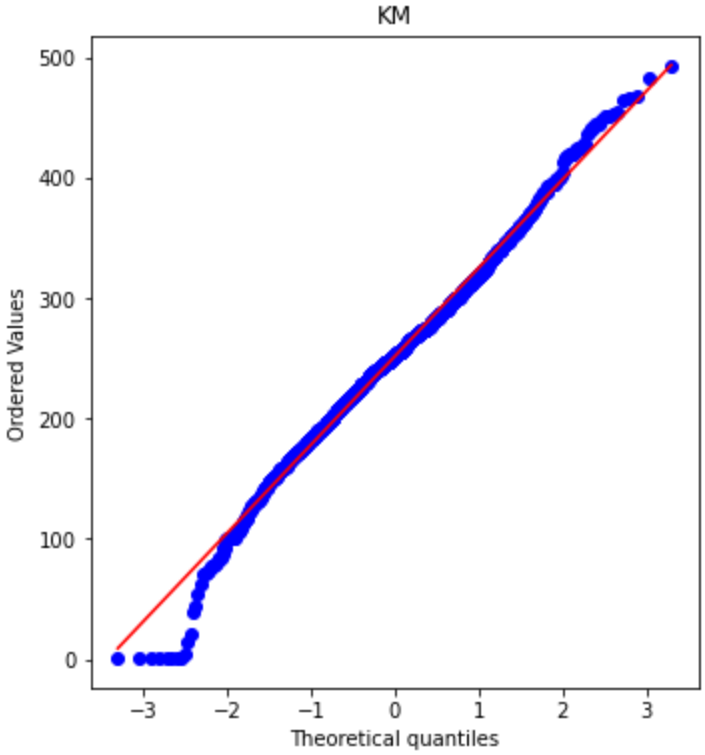


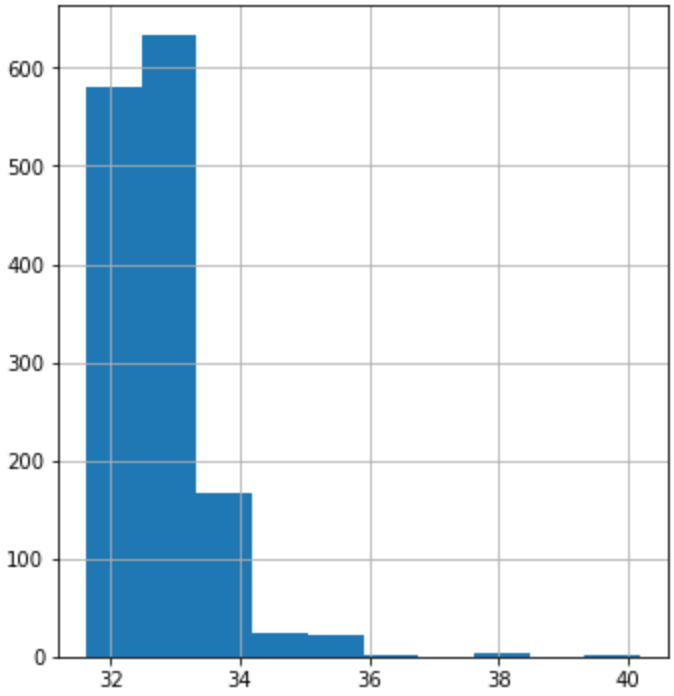


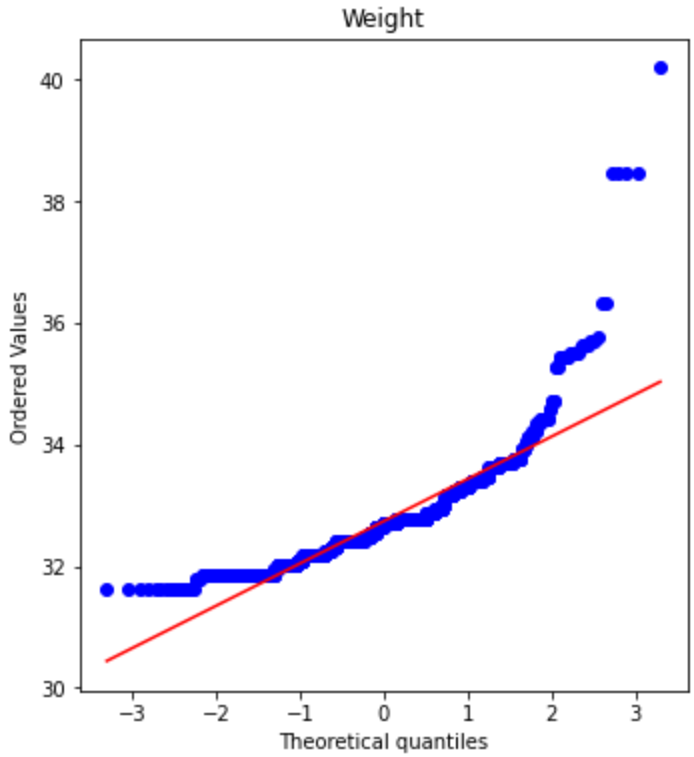










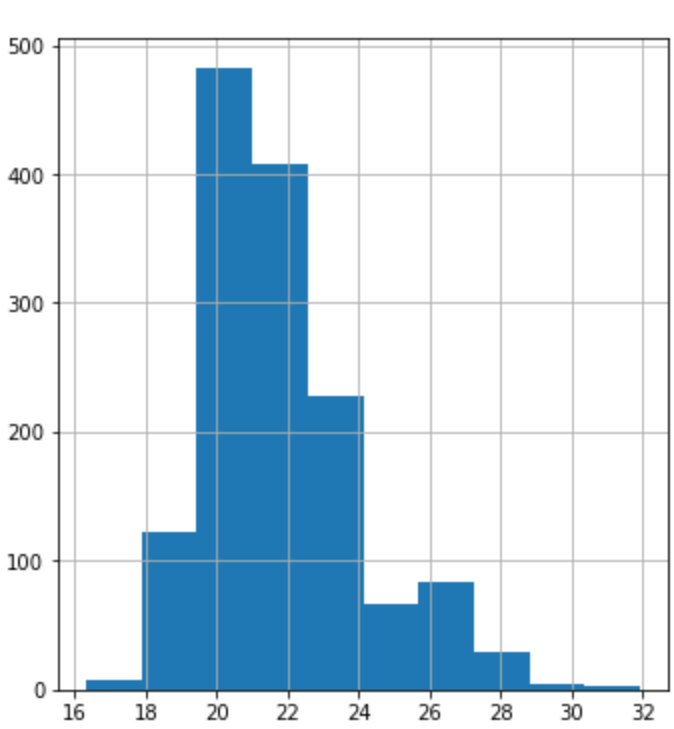


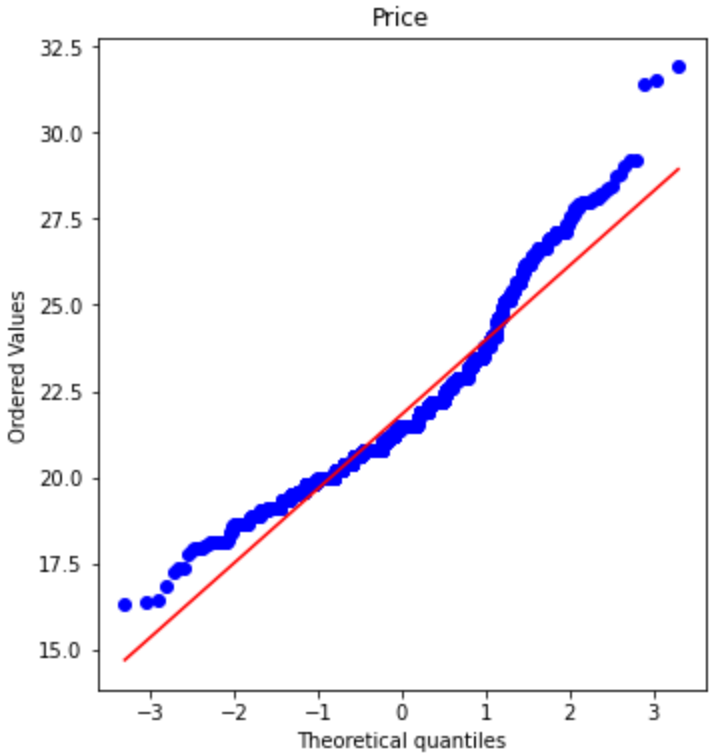
Cuberoot transformation and visualizing the Histogram to determine any possible changes in distribution

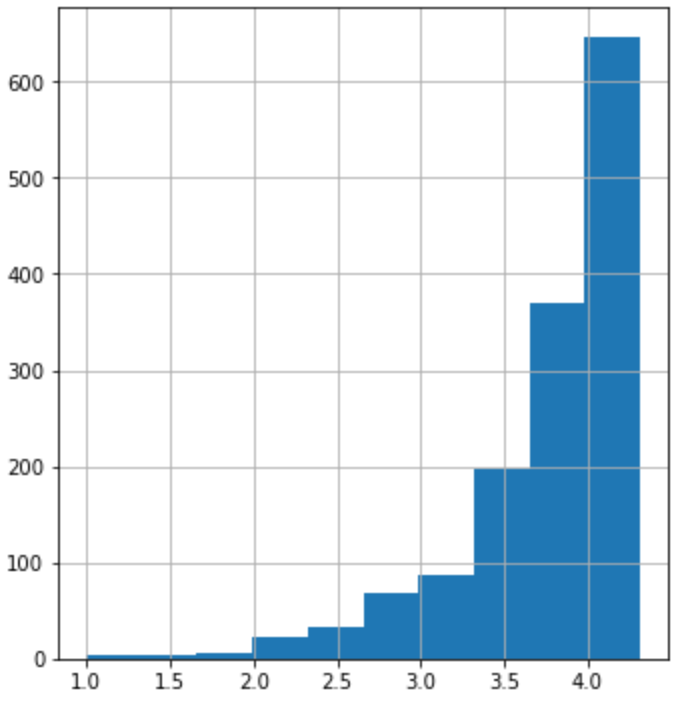
In [79]:

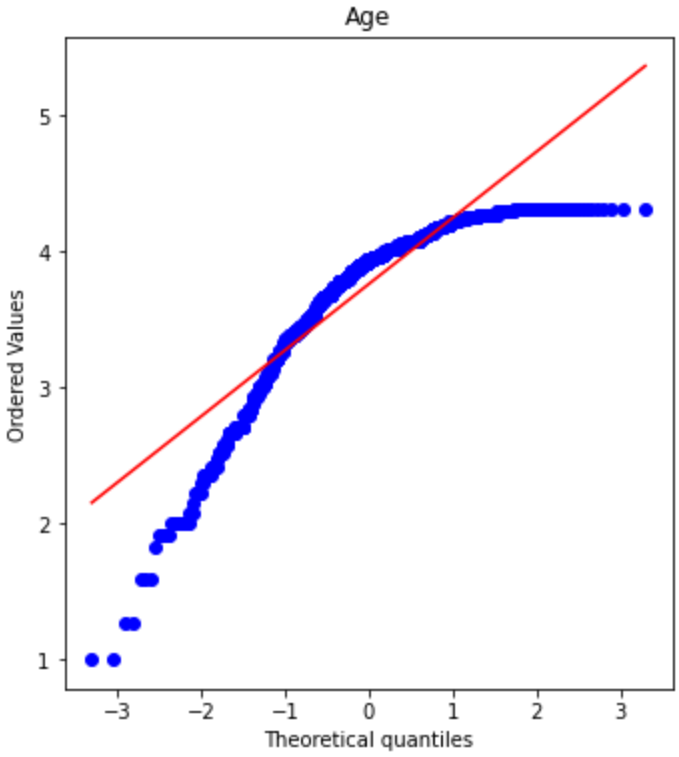
```
df=data.copy()
df[continuous_feature]=np.cbrt(df[continuous_feature])

plot_data(df,'Price')
plt.title('Price')
plot_data(df,'Age')
plt.title('Age')
plot_data(df,'KM')
plt.title('KM')
plot_data(df,'Weight')
plt.title('Weight')
```

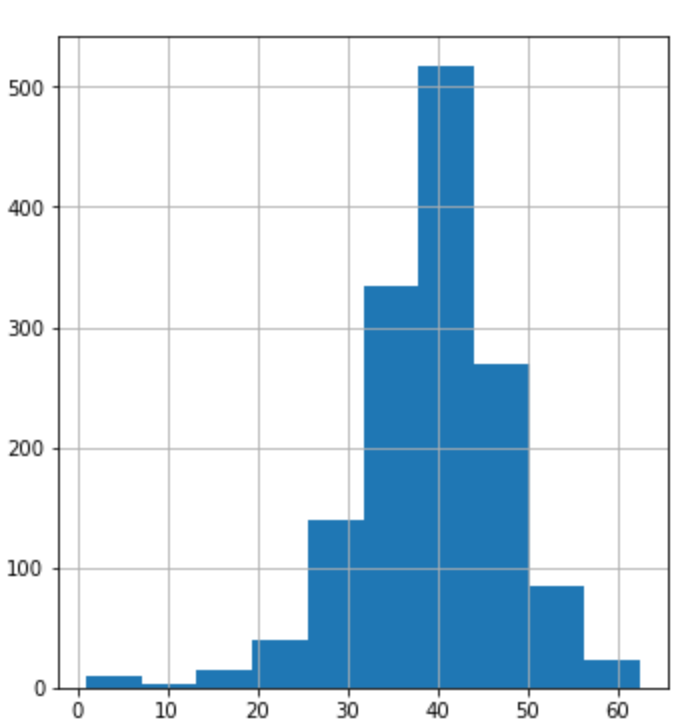
Out[79]: Text(0.5, 1.0, 'Weight')

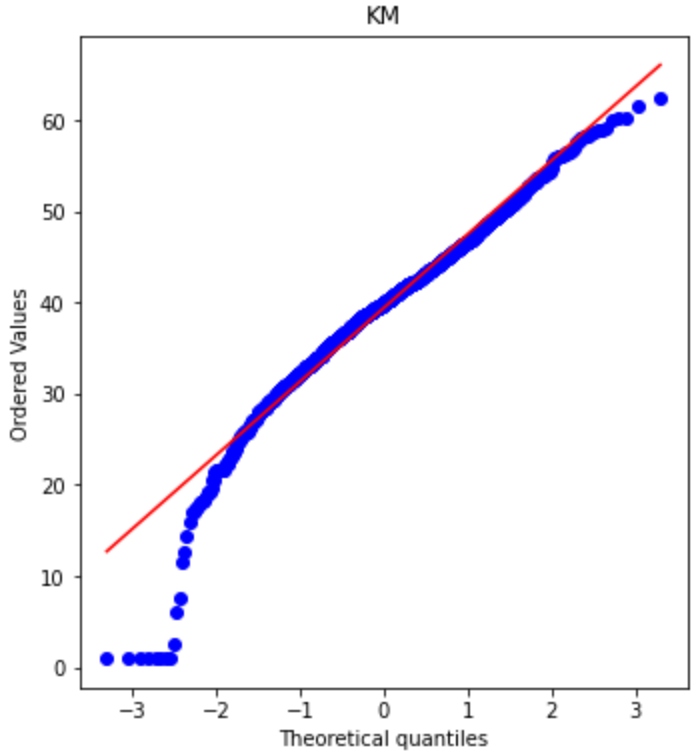


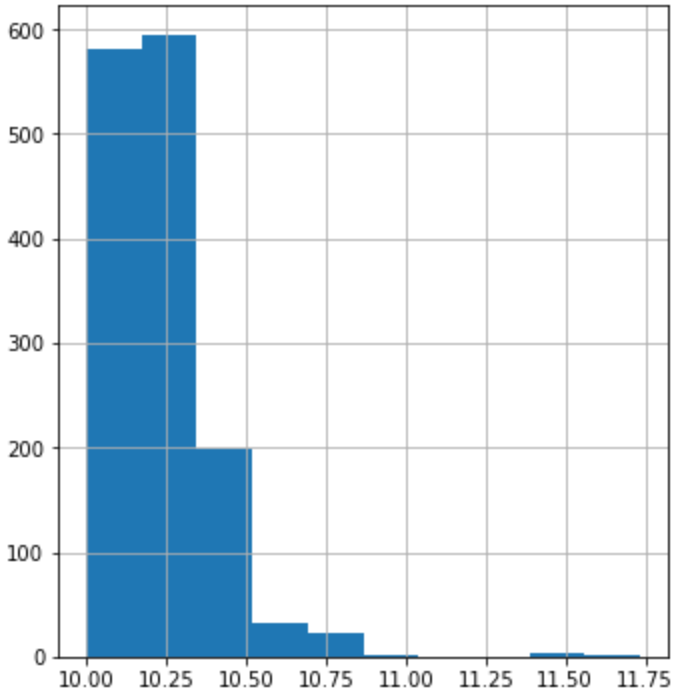


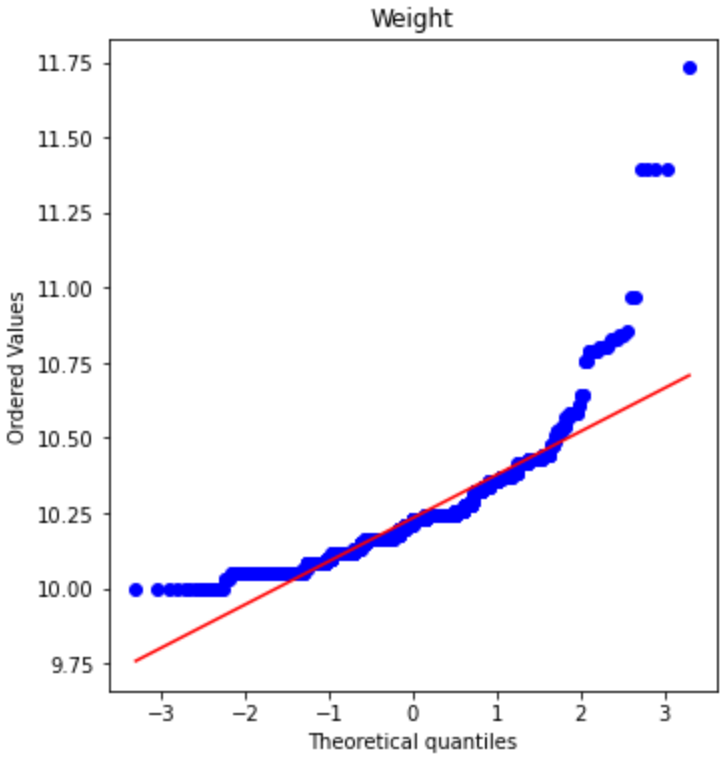












- **Note: Most of the Continuous Features visually do not look normally distributed lets have some Hypothetical test to check the normailty.**

The Shapiro-Wilk test is a test of normality. It is used to determine whether or not a sample comes from a normal distribution.

- **To perform a Shapiro-Wilk test in Python we can use the scipy.stats.shapiro() function, which takes on the following syntax:**

In [80]:
```
data.columns
```

```
Index(['Price', 'Age', 'KM', 'HP', 'CC', 'Doors', 'Gears', 'QT', 'Weight'], dtype='objec
```

```
t')
```

In [81]:

```
from scipy.stats import shapiro

#perform Shapiro-Wilk test
print('Price feature',shapiro(data.Price),'\n'
      'Age feature',shapiro(data.Age),'\n'
      'Weight feature',shapiro(data.Weight),'\n'
      'KM feature',shapiro(data.KM))
```

```
Price feature ShapiroResult(statistic=0.8534726500511169, pvalue=1.5959173670279415e-34)
Age feature ShapiroResult(statistic=0.9266955256462097, pvalue=6.739428532958423e-26)
Weight feature ShapiroResult(statistic=0.7825545072555542, pvalue=5.042992913412152e-40)
KM feature ShapiroResult(statistic=0.947583794593811, pvalue=3.4451158696360995e-22)
```

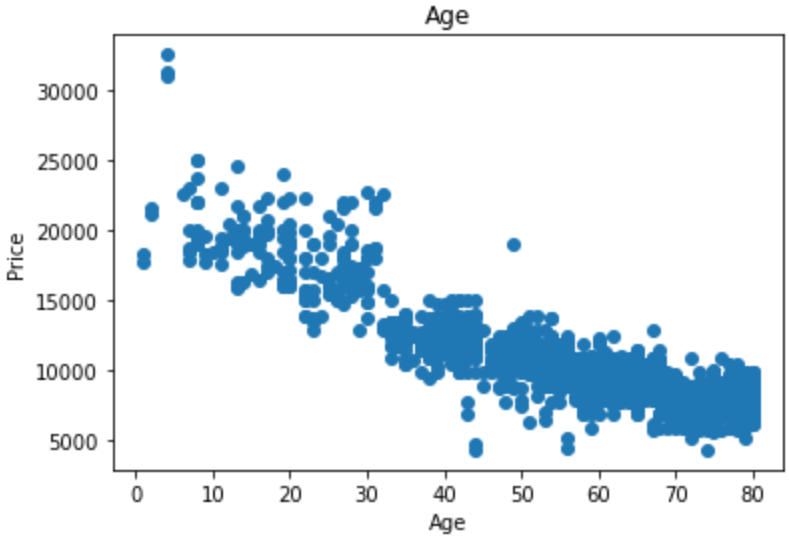
^Observation: Since the p-values are less than .05, we reject the null hypothesis.

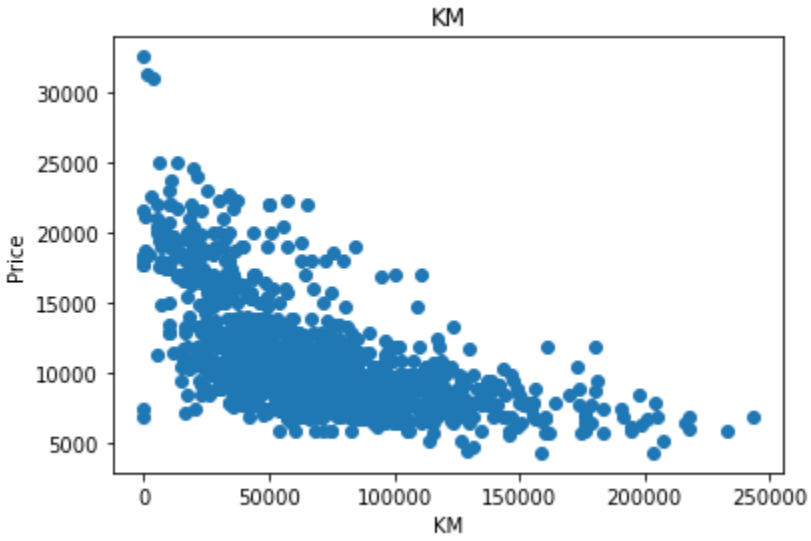
- **We have sufficient evidence to say that the sample data does not come from a normal distribution.**

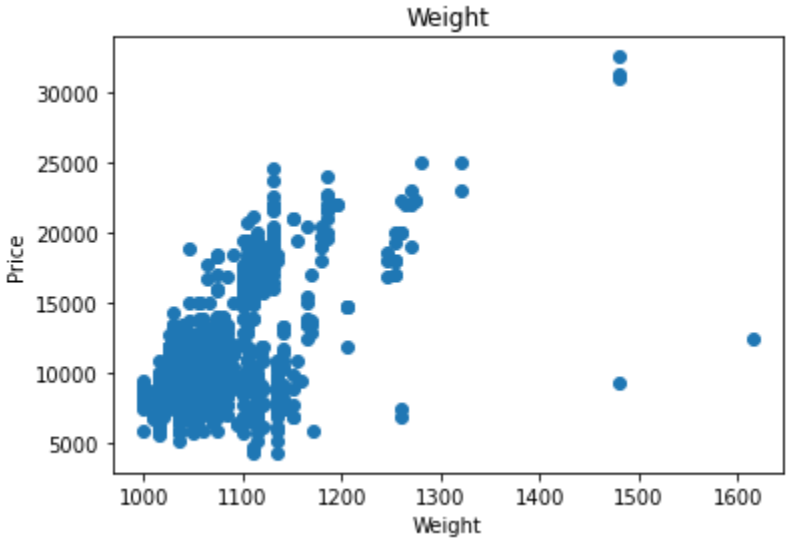
Visualizing the Relation between each independent Feature with respect to the Dependent Feature

In [82]:

```
for feature in continuous_feature:
    if feature!="Price":
        df=data.copy()
        plt.scatter(df[feature],df['Price'])
        plt.xlabel(feature)
        plt.ylabel('Price')
        plt.title(feature)
        plt.show()
```





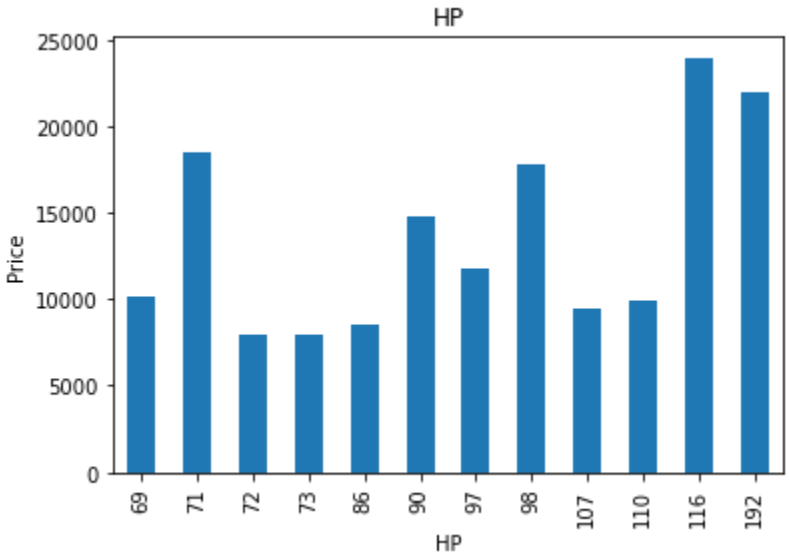


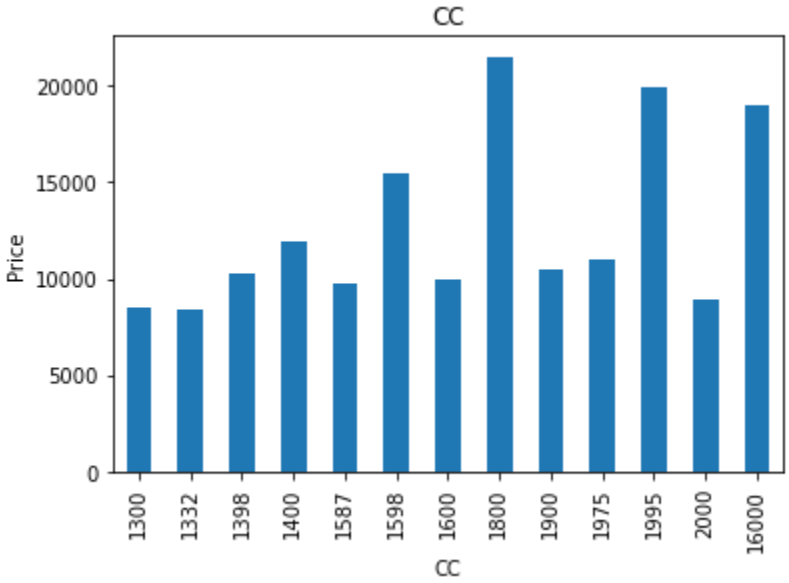
^Observation: Age feature has a good linear relation with Price a Negative Correlation as compare to other features

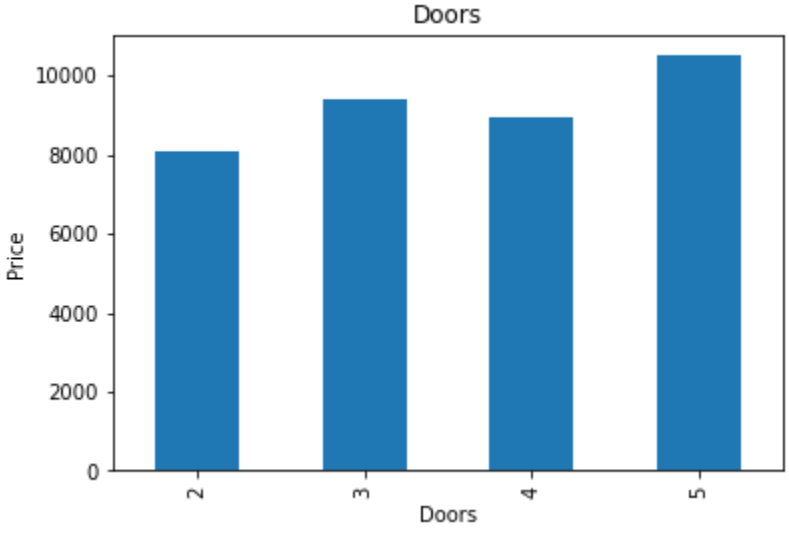
Lets analyze the relationship between the discrete variables and Price

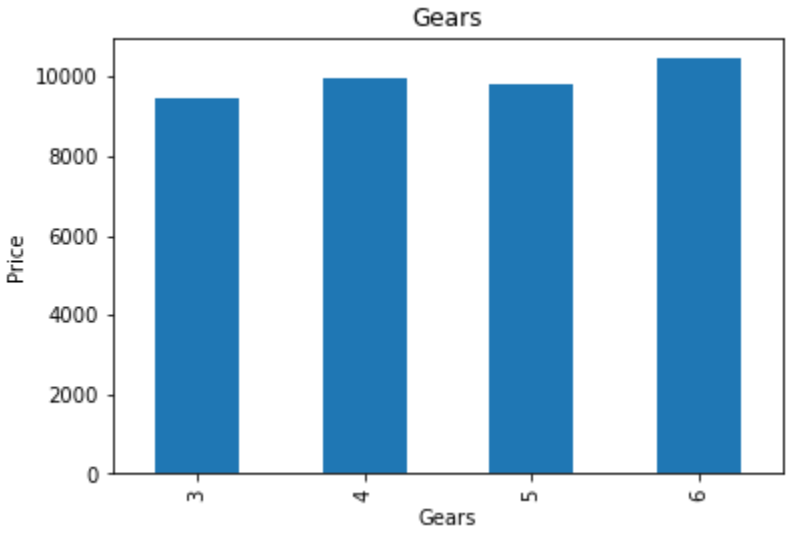
In [83]:

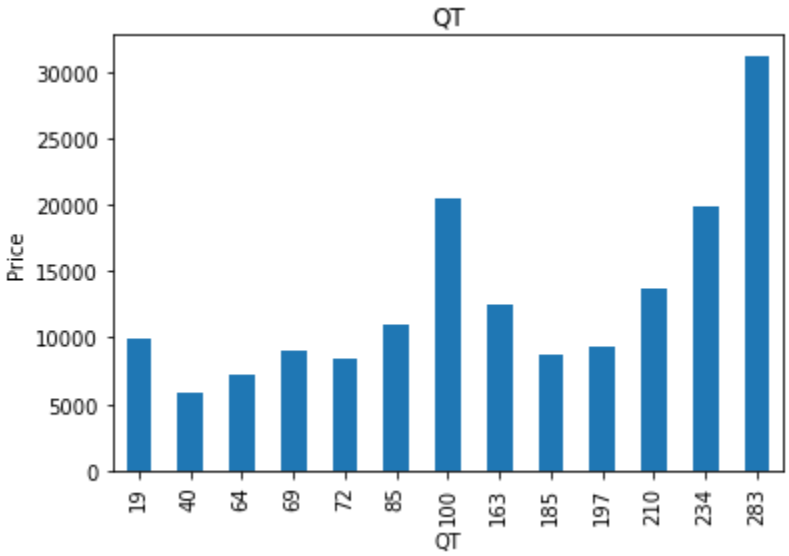
```
for feature in discrete_feature:
    df=data.copy()
    df.groupby(feature)["Price"].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel("Price")
    plt.title(feature)
    plt.show()
```











^Observation:There is'nt much of difference between how much Gears and Doors each Car has to have a sginificant amount of changes in Prices from each other and there is'nt any direct relation

Checking the correlation between Variables

In [84]:
```
data.corr()
```

Out[84]:

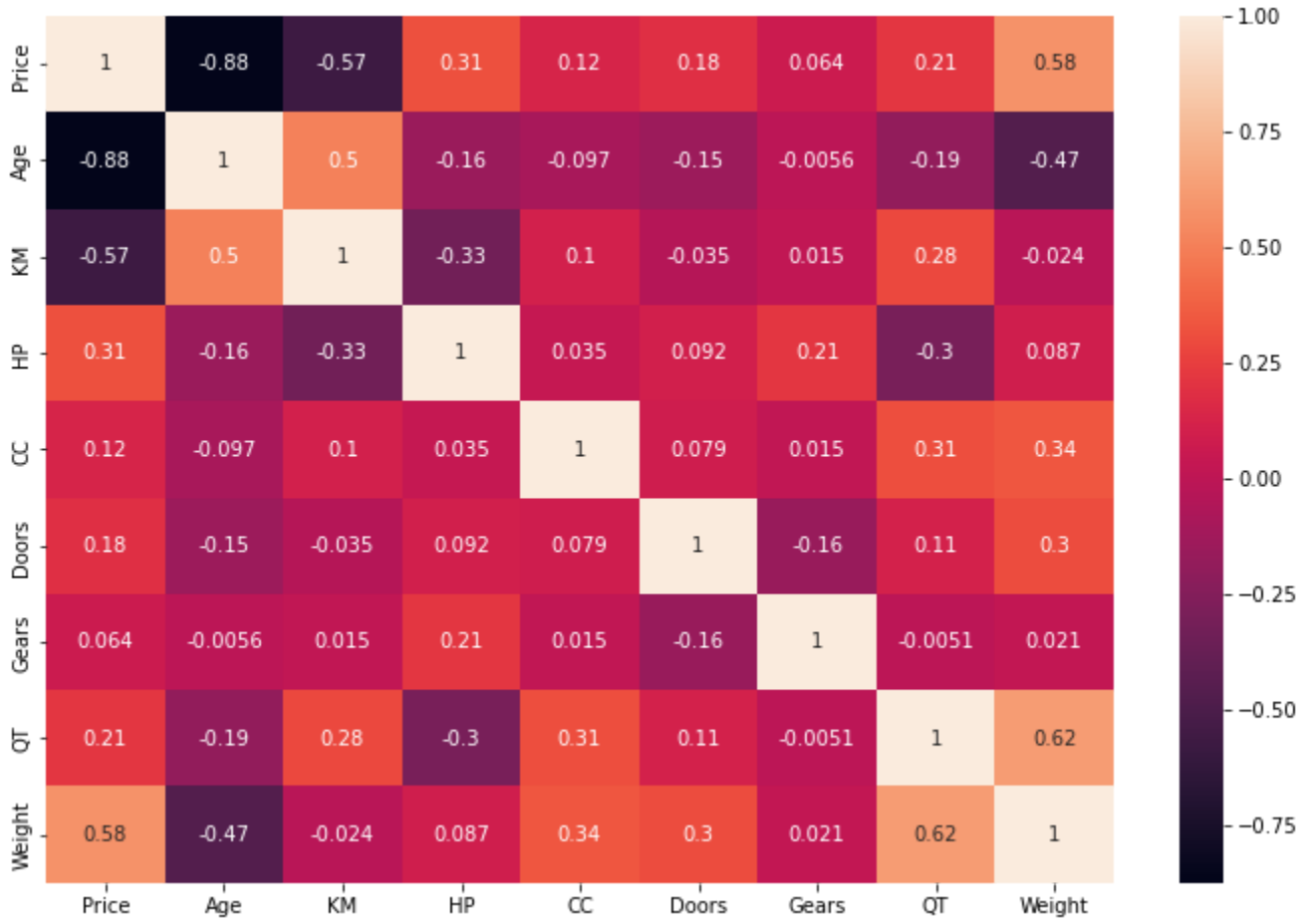
	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
Price	1.000000	-0.876273	-0.569420	0.314134	0.124375	0.183604	0.063831	0.211508	0.575869
Age	-0.876273	1.000000	0.504575	-0.155293	-0.096549	-0.146929	-0.005629	-0.193319	-0.466484
KM	-0.569420	0.504575	1.000000	-0.332904	0.103822	-0.035193	0.014890	0.283312	-0.023969
HP	0.314134	-0.155293	-0.332904	1.000000	0.035207	0.091803	0.209642	-0.302287	0.087143
CC	0.124375	-0.096549	0.103822	0.035207	1.000000	0.079254	0.014732	0.305982	0.335077
Doors	0.183604	-0.146929	-0.035193	0.091803	0.079254	1.000000	-0.160101	0.107353	0.301734
Gears	0.063831	-0.005629	0.014890	0.209642	0.014732	-0.160101	1.000000	-0.005125	0.021238
QT	0.211508	-0.193319	0.283312	-0.302287	0.305982	0.107353	-0.005125	1.000000	0.621988
Weight	0.575869	-0.466484	-0.023969	0.087143	0.335077	0.301734	0.021238	0.621988	1.000000

In [85]:
```
gsize=(12,8))
```

```
sns.heatmap(
    data.corr(),
    annot=True)
```

Out[85]:

```
<AxesSubplot:>
```

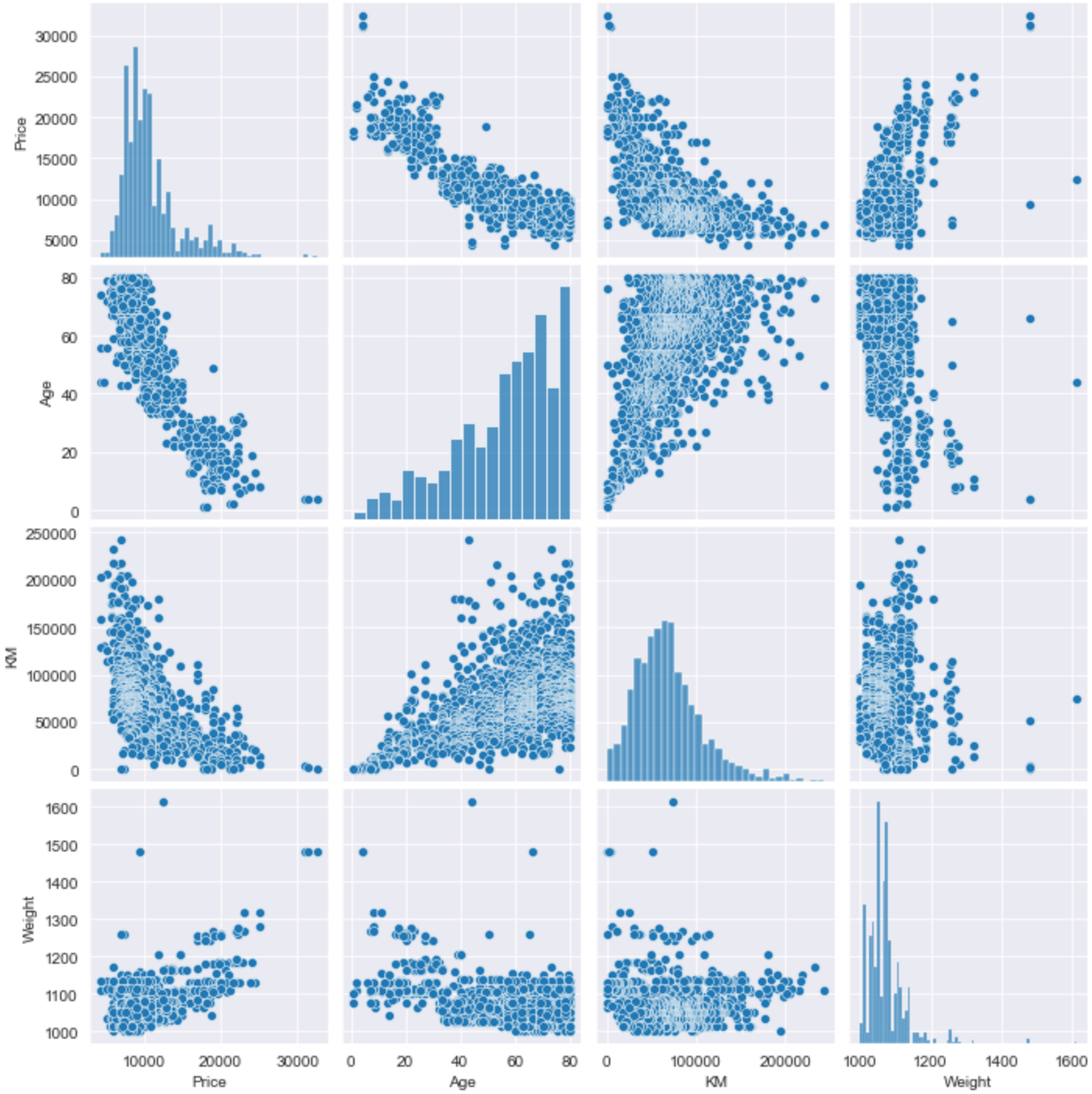


In [86]:

```
sns.set_style(style='darkgrid')
sns.pairplot(data[continuous_feature])
```

Out[86]:

```
<seaborn.axisgrid.PairGrid at 0x19664234100>
```



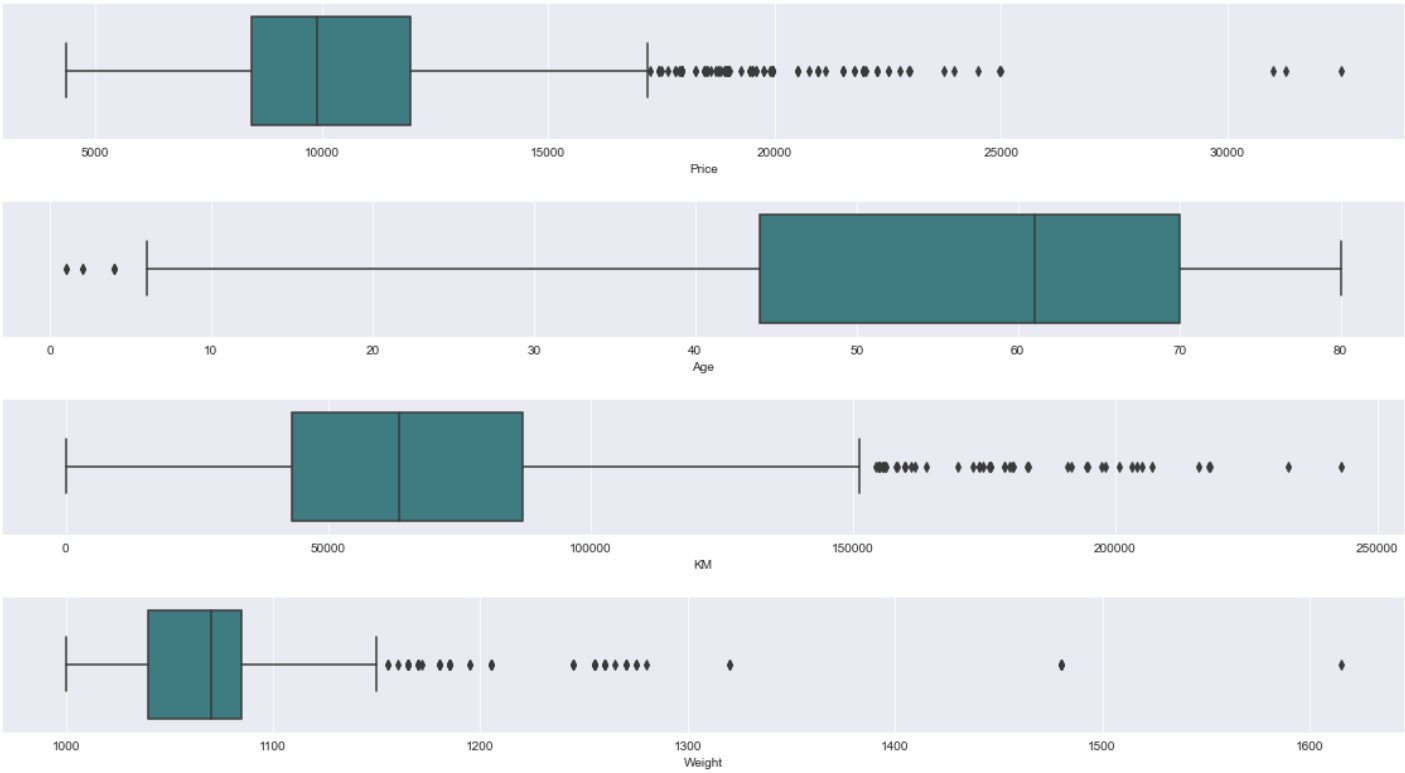
^Observation: Age and KM has the highest score of correlation with Price but a negative correlation

- **Note: QT and Weight also have a collinearity among themselves which will affect our model.**
- **Note: KM and Age also have a collinearity among themselves which will affect our model.**

Visualizing Continuous Datatype for Outlier Detection

In [87]:

```
df=data.copy()
fig, axes=plt.subplots(4,1,figsize=(16,9),sharex=False,sharey=False)
sns.boxplot(x='Price',data=df,palette='crest',ax=axes[0])
sns.boxplot(x='Age',data=df,palette='crest',ax=axes[1])
sns.boxplot(x='KM',data=df,palette='crest',ax=axes[2])
sns.boxplot(x='Weight',data=df,palette='crest',ax=axes[3])
plt.tight_layout(pad=2.0)
```



^Observation: A significant amount of outliers are present in each continuous feature

Note:

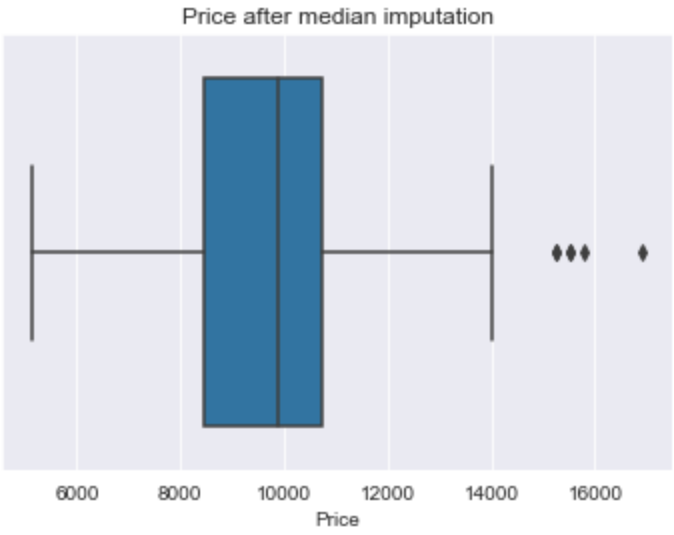
- **We can't simply remove the outliers that would mean loss of information**
- **We need to try different types of transformation or imputation and select the one with the best results**
- **Note: Transforming variables can also eliminate outliers. The Transformed Variables reduces the variation caused by thr extreme values**

Before handling Outliers lets build a model and compare its R-squared value with other techniques to see which technique suits best for out case

Let's try Median Imputation to handle Outlier in Profit

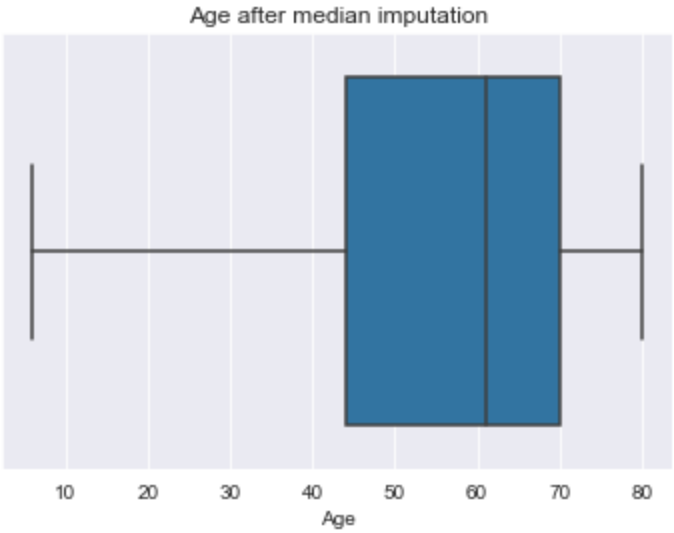
In [88]:

```
df1=data.copy()
for i in data['Price']:
    q1 = np.quantile(df1.Price,0.25)
    q3 = np.quantile(df1.Price,0.75)
    med = np.median(df1.Price)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df1['Price'] = df1['Price'].replace(i, np.median(df1['Price']))
sns.boxplot(df1['Price'])
plt.title('Price after median imputation')
plt.show()
```



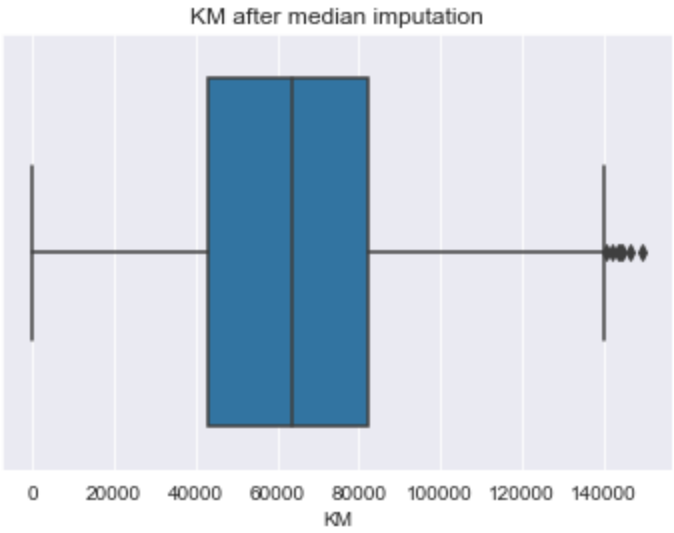
In [89]:

```
for i in data['Age']:
    q1 = np.quantile(df1.Age,0.25)
    q3 = np.quantile(df1.Age,0.75)
    med = np.median(df1.Age)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df1['Age'] = df1['Age'].replace(i, np.median(df1['Age']))
sns.boxplot(df1['Age'])
plt.title('Age after median imputation')
plt.show()
```



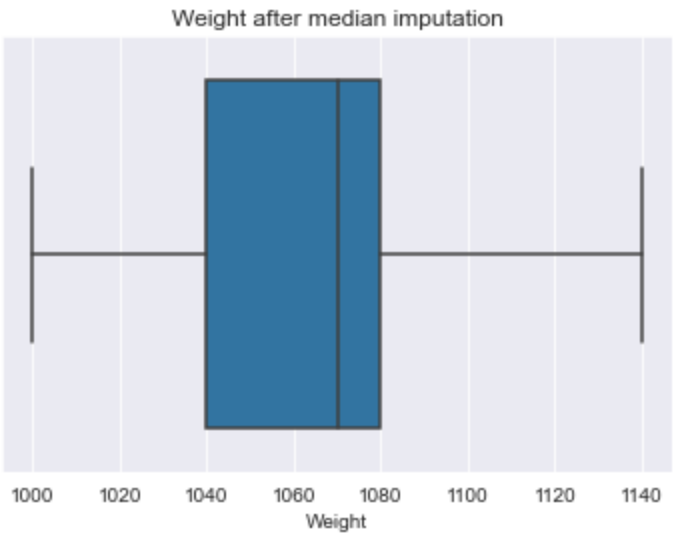
In [90]:

```
for i in data['KM']:
    q1 = np.quantile(df1.KM,0.25)
    q3 = np.quantile(df1.KM,0.75)
    med = np.median(df1.KM)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df1['KM'] = df1['KM'].replace(i, np.median(df1['KM']))
sns.boxplot(df1['KM'])
plt.title('KM after median imputation')
plt.show()
```



In [91]:

```
for i in data['Weight']:
    q1 = np.quantile(df1.Weight,0.25)
    q3 = np.quantile(df1.Weight,0.75)
    med = np.median(df1.Weight)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df1['Weight'] = df1['Weight'].replace(i, np.median(df1['Weight']))
sns.boxplot(df1['Weight'])
plt.title('Weight after median imputation')
plt.show()
```



Let's test our data in model and find the R-squared with median imputation data model

In [54]:

```
after_median_imputation_model = smf.ols("Price~Age+KM+Weight", data = df1).fit()
# Finding rsquared values
after_median_imputation_model.rsquared , after_median_imputation_model.rsquared_adj
```

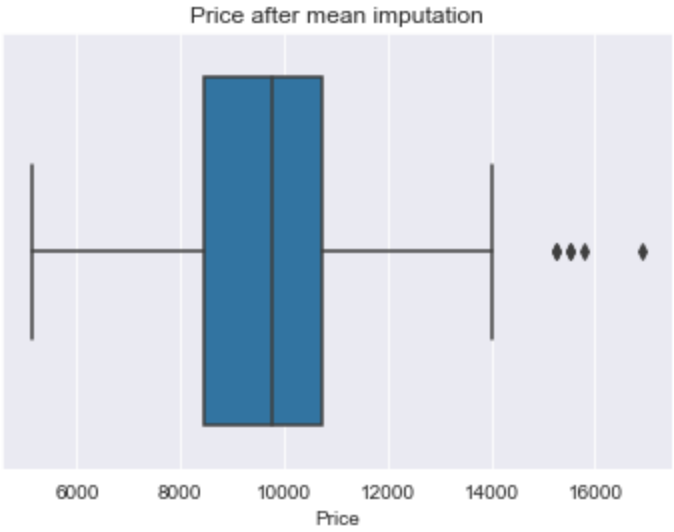
Out[54]:

```
(0.3406499164516401, 0.3392676311611824)
```

Let's try Mean Imputation to handle Outlier in Profit

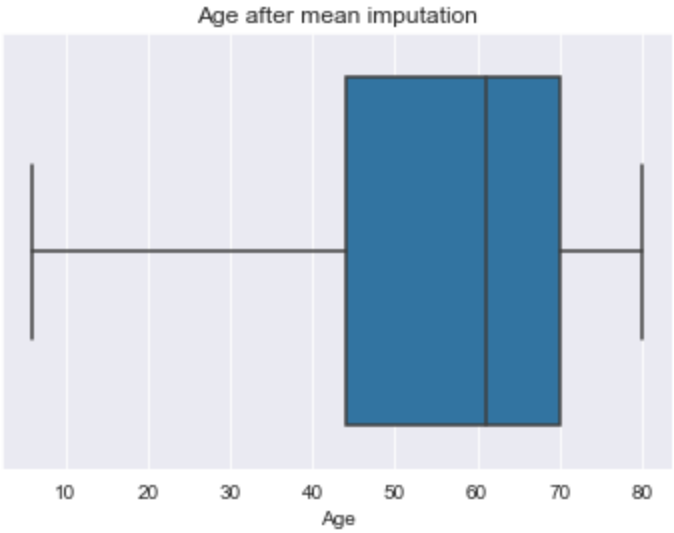

```
()
```

```
for i in data['Price']:
    q1 = np.quantile(df2.Price,0.25)
    q3 = np.quantile(df2.Price,0.75)
    med = np.median(df2.Price)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df2['Price'] = df2['Price'].replace(i, np.mean(df2['Price']))
sns.boxplot(df2['Price'])
plt.title('Price after mean imputation')
plt.show()
```



In [93]:

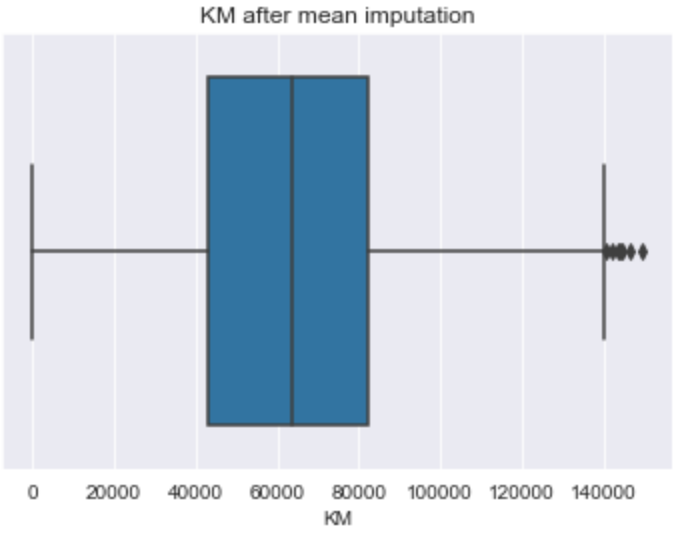
```
for i in data['Age']:
    q1 = np.quantile(df2.Age,0.25)
    q3 = np.quantile(df2.Age,0.75)
    med = np.median(df2.Age)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df2['Age'] = df2['Age'].replace(i, np.mean(df2['Age']))
sns.boxplot(df2['Age'])
plt.title('Age after mean imputation')
plt.show()
```



In [94]:

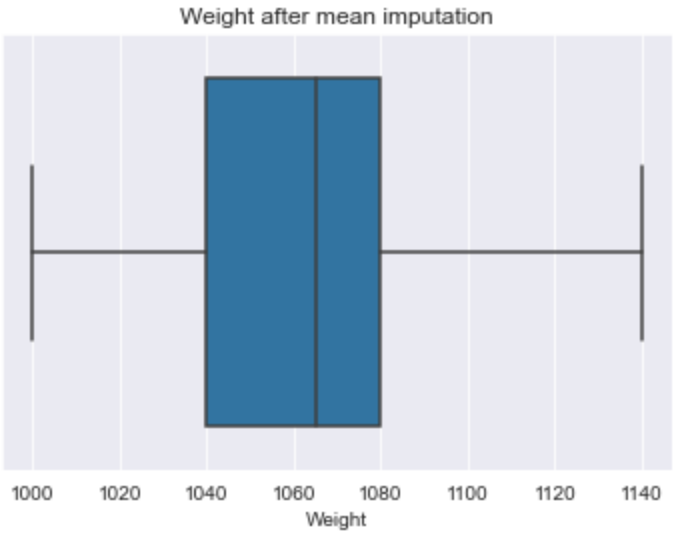
```
['KM']:
```

```
    q1 = np.quantile(df2.KM,0.25)
    q3 = np.quantile(df2.KM,0.75)
    med = np.median(df2.KM)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df2['KM'] = df2['KM'].replace(i, np.mean(df2['KM']))
sns.boxplot(df2['KM'])
plt.title('KM after mean imputation')
plt.show()
```



In [95]:

```
for i in data['Weight']:
    q1 = np.quantile(df2.Weight,0.25)
    q3 = np.quantile(df2.Weight,0.75)
    med = np.median(df2.Weight)
    iqr = q3 - q1
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    if i > upper_bound or i < lower_bound:
        df2['Weight'] = df2['Weight'].replace(i, np.mean(df2['Weight']))
sns.boxplot(df2['Weight'])
plt.title('Weight after mean imputation')
plt.show()
```



Let's test our data in model and find the R-squared with mean imputation data

In [96]:
```
after_mean_imputation_model = smf.ols("Price~Age+KM+Weight", data = df2).fit()
# Finding rsquared values
after_mean_imputation_model.rsquared , after_mean_imputation_model.rsquared_adj
```

Out[96]:
```
(0.3879621958171299, 0.3866790976951533)
```

^Observation: As you can see after mean imputation the model is not performing well

- **Now we have to try something else to get out better results than the raw data**

^Observation: As you can see even after imputation the model is not performing well it getting worse

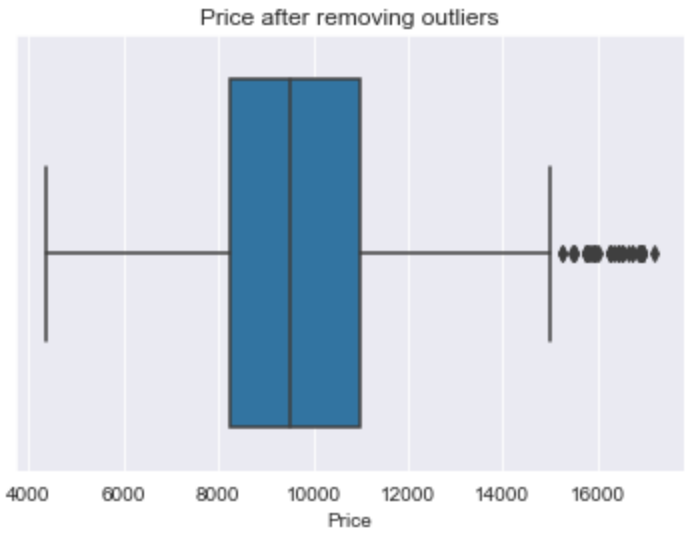
- **Now we have to try something else to get out model better than the raw data**

The best thing we can do is now to remove the outlier and see the results

In [97]:
```
df3=data.copy()
def drop_outliers(data, field_name):
    iqr = 1.5*(np.percentile(data[field_name], 75) - np.percentile(data[field_name], 25))
    data.drop(data[data[field_name] > (iqr + np.percentile(data[field_name], 75))].index,
    data.drop(data[data[field_name] < (np.percentile(data[field_name], 25) - iqr)].index,
```

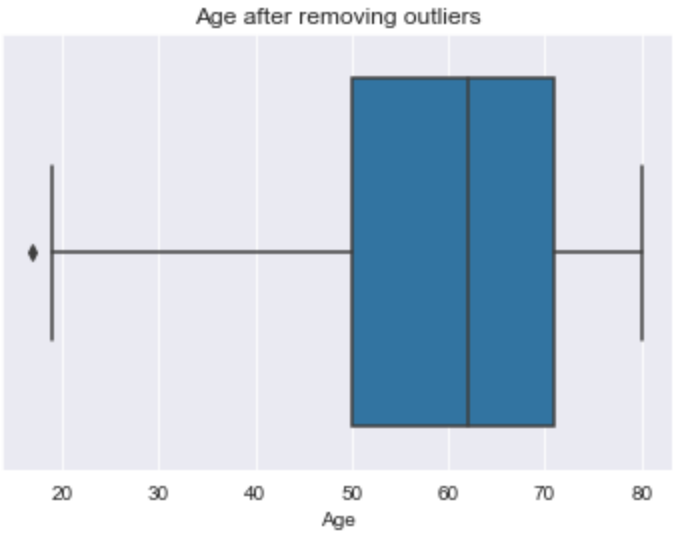
In [98]:
```
drop_outliers(df3, 'Price')
sns.boxplot(df3.Price)
plt.title('Price after removing outliers')
```

Out[98]:
```
Text(0.5, 1.0, 'Price after removing outliers')
```



In [99]:
```
drop_outliers(df3, 'Age')
sns.boxplot(df3.Age)
plt.title('Age after removing outliers')
```

Out[99]:
```
Text(0.5, 1.0, 'Age after removing outliers')
```

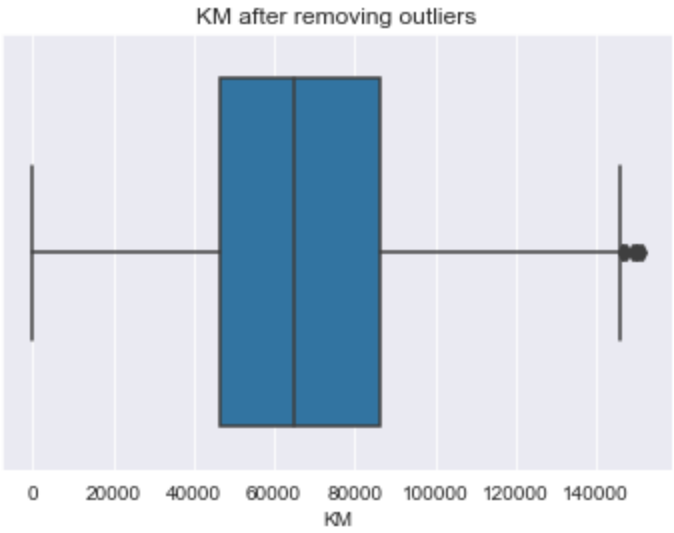


In [100…

```
drop_outliers(df3, 'KM')
sns.boxplot(df3.KM)
plt.title('KM after removing outliers')
```

Out[100…

```
Text(0.5, 1.0, 'KM after removing outliers')
```

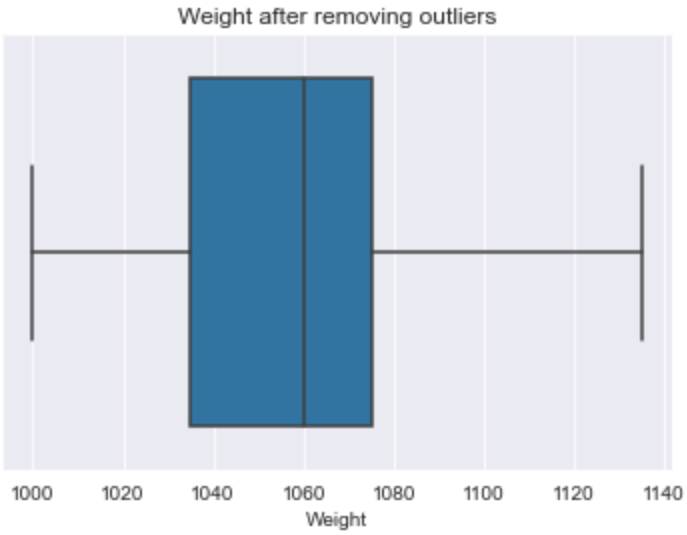


In [101…

```
drop_outliers(df3, 'Weight')
sns.boxplot(df3.Weight)
plt.title('Weight after removing outliers')
```

Out[101…

```
Text(0.5, 1.0, 'Weight after removing outliers')
```



Let's test our data in model and compare the R-squared with without imputation data model

In [102…
```
removed_outlier_model = smf.ols("Price~Age+KM+Weight", data = df3).fit()
# Finding rsquared values
removed_outlier_model.rsquared , removed_outlier_model.rsquared_adj
```

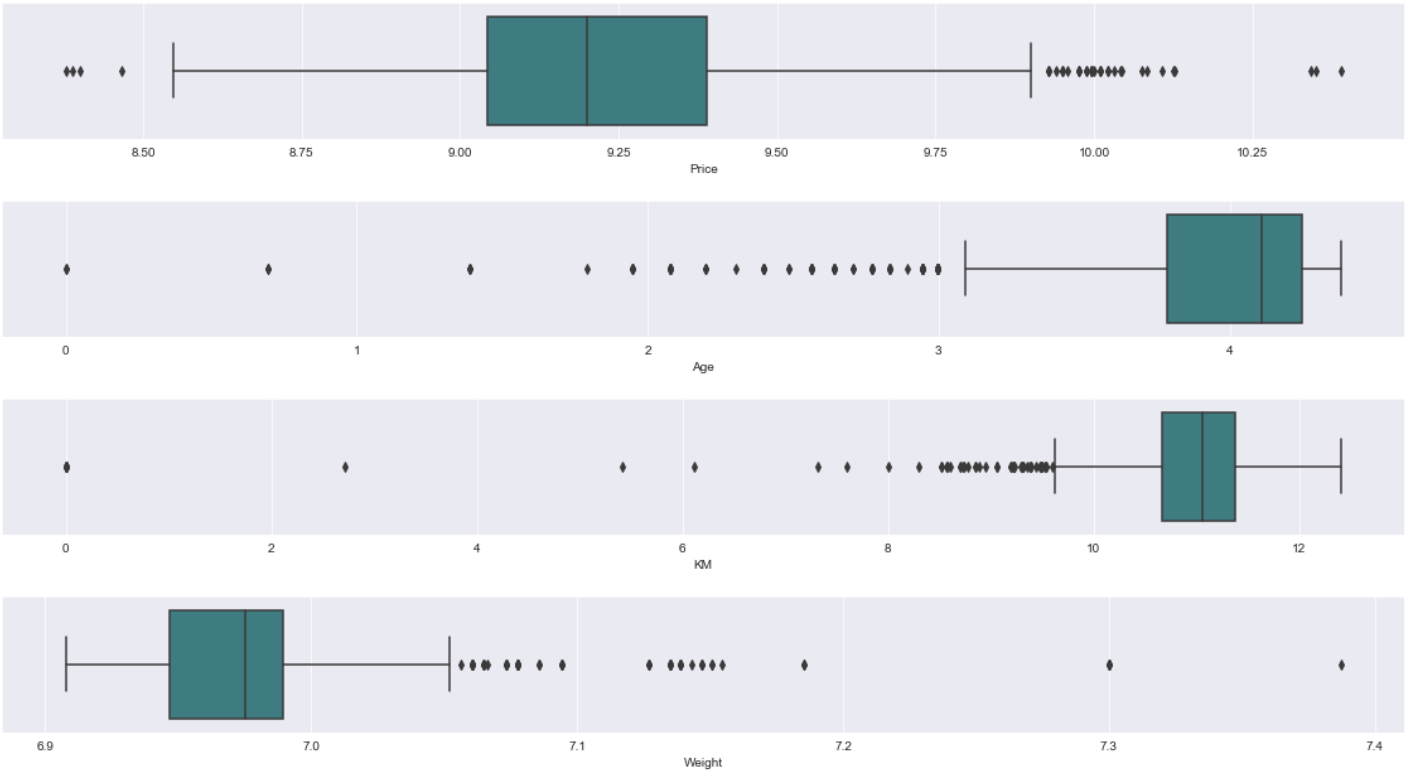
Out[102…
```
(0.7776886294411589, 0.7771455234870249)
```

In [103…
```
np.sqrt(removed_outlier_model.mse_resid)
```

Out[103…
```
1060.0301342603996
```

Let's try log transformation and visualize the result first

In [104…
```
df=data.copy()
df[continuous_feature]=np.log(df[continuous_feature])

fig, axes=plt.subplots(4,1,figsize=(16,9),sharex=False,sharey=False)
sns.boxplot(x='Price',data=df,palette='crest',ax=axes[0])
sns.boxplot(x='Age',data=df,palette='crest',ax=axes[1])
sns.boxplot(x='KM',data=df,palette='crest',ax=axes[2])
sns.boxplot(x='Weight',data=df,palette='crest',ax=axes[3])
plt.tight_layout(pad=2.0)
```



In [105…

```
log_transfomed = data.copy()
log_transfomed[continuous_feature]=np.log(log_transfomed[continuous_feature])
log_transformed_model = smf.ols("Price~Age+KM+Weight", data = log_transfomed).fit()
# Finding rsquared values
log_transformed_model.rsquared , log_transformed_model.rsquared_adj
```

Out[105…

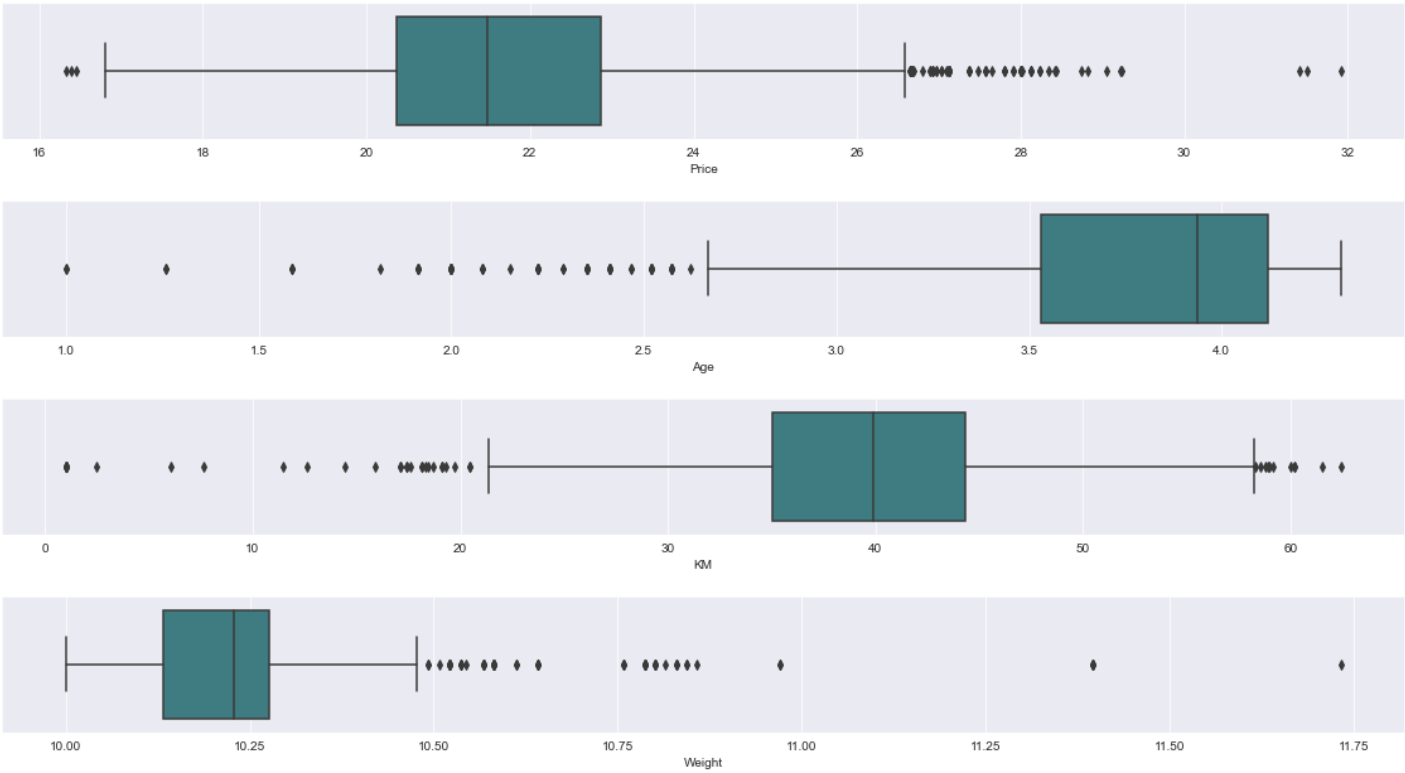
```
(0.7069873404282615, 0.7063730581230796)
```

^Observation: The outliers are still present

Let's try cuberoot transformation and visualize the result first

In [106…

```
df=data.copy()
df[continuous_feature]=np.cbrt(df[continuous_feature])

fig, axes=plt.subplots(4,1,figsize=(16,9),sharex=False,sharey=False)
sns.boxplot(x='Price',data=df,palette='crest',ax=axes[0])
sns.boxplot(x='Age',data=df,palette='crest',ax=axes[1])
sns.boxplot(x='KM',data=df,palette='crest',ax=axes[2])
sns.boxplot(x='Weight',data=df,palette='crest',ax=axes[3])
plt.tight_layout(pad=2.0)
```



In [107…

```
cube_root_transfomed = data.copy()
cube_root_transfomed[continuous_feature]=np.cbrt(cube_root_transfomed[continuous_feature])
cube_root_transformed_model = smf.ols("Price~Age+KM+Weight", data = cube_root_transfomed).
# Finding rsquared values
cube_root_transformed_model.rsquared , cube_root_transformed_model.rsquared_adj
```

Out[107…

```
(0.8146046800585908, 0.8142160106247514)
```

^Observation: The outliers are still present

^Observation: After removing Outliers the model performed very poorly than the raw data model

- **Note: We will continue with different technique to deal with that**

Raw Data Model

In [69]:

```
data.columns
```

Out[69]:

```
Index(['Price', 'Age', 'KM', 'HP', 'CC', 'Doors', 'Gears', 'QT', 'Weight'], dtype='objec
t')
```

In [70]:

```
raw_data.columns
```

Out[70]:

```
Index(['Price', 'Age_08_04', 'KM', 'HP', 'cc', 'Doors', 'Gears',
       'Quarterly_Tax', 'Weight'],
      dtype='object')
```

In [10]:

```
raw_data_model = smf.ols("Price~Age+KM+Weight+HP+CC+Gears+QT+Doors", data = data).fit()
# Finding rsquared values
raw_data_model.summary()
```

Out[10]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.863

Model:	OLS	Adj. R-squared:	0.862
Method:	Least Squares	F-statistic:	1118.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:02:00	Log-Likelihood:	-12366.
No. Observations:	1435	AIC:	2.475e+04
Df Residuals:	1426	BIC:	2.480e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5472.5404	1412.169	-3.875	0.000	-8242.692	-2702.389
Age	-121.7139	2.615	-46.552	0.000	-126.843	-116.585
KM	-0.0207	0.001	-16.552	0.000	-0.023	-0.018
Weight	16.8555	1.069	15.761	0.000	14.758	18.953
HP	31.5846	2.818	11.210	0.000	26.058	37.112
CC	-0.1186	0.090	-1.316	0.188	-0.295	0.058
Gears	597.7159	196.969	3.035	0.002	211.335	984.097
QT	3.8588	1.311	2.944	0.003	1.288	6.430
Doors	-0.9202	39.988	-0.023	0.982	-79.362	77.522

Omnibus:	149.666	Durbin-Watson:	1.544
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1000.538
Skew:	-0.204	Prob(JB):	5.44e-218
Kurtosis:	7.070	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.

In [11]:
```
np.sqrt(raw_data_model.mse_resid)
```

Out[11]:
```
1341.8046186938675
```

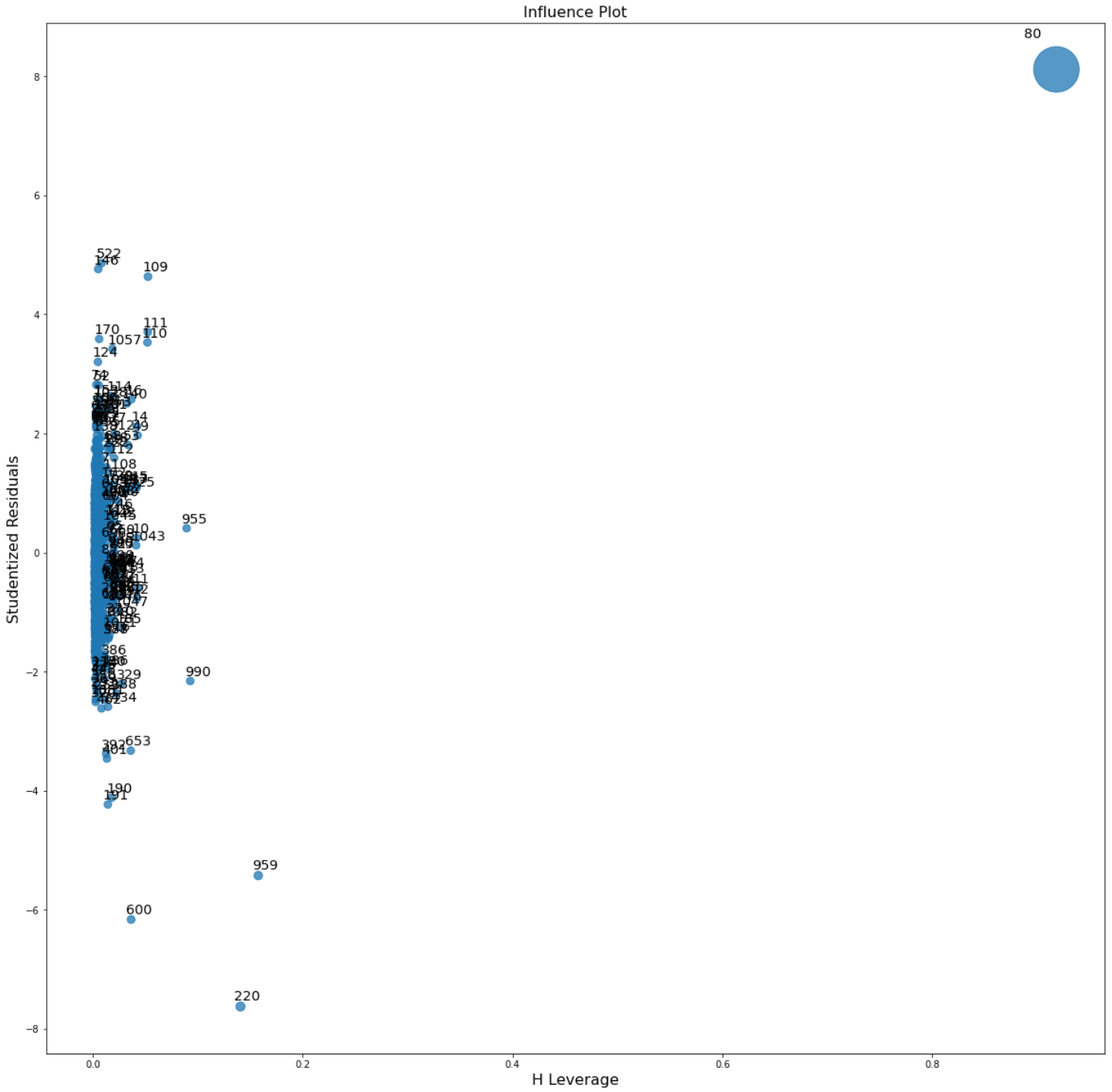
Detecting Influencers/Outliers in the Model

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

In [12]:
```
influence_points=raw_data_model.get_influence()
c, p_value=influence_points.cooks_distance
```

In [13]:
```
# Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are inf
fig,ax=plt.subplots(figsize=(20,20))
```

```
fig=influence_plot(raw_data_model,ax = ax)
plt.show()
```



Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of datapoints

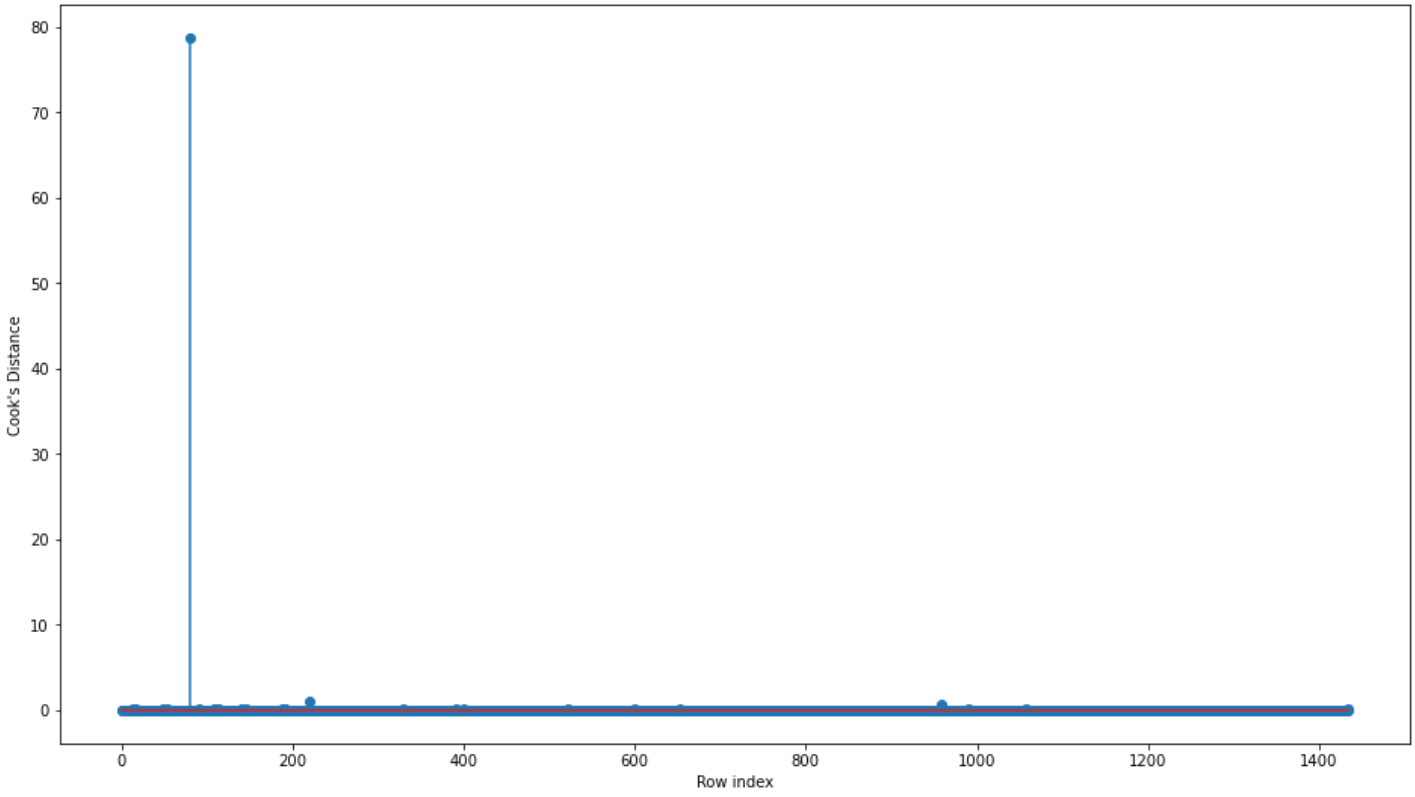
In [14]:
```
k=data.shape[1]
n=data.shape[0]
leverage_cutoff = (3*(k+1))/n
print('Cut-off line at',np.round(leverage_cutoff,2))
```

```
Cut-off line at 0.02
```

Let's plot the influencers and also plot a cut off line using the stem plot

In [15]:
```
fig = plt.figure(figsize = (16,9))
x = [0,48]
```

```
plt.plot(x, y,color='darkred', linewidth=2)
y1 = [0.05,0.05]
plt.plot(x , y1, color = 'red', linewidth = 2)
plt.stem(np.arange(len(data)), np.round(c, 3))
plt.xlabel('Row index')
plt.ylabel("Cook's Distance")
plt.show()
```



In [16]:
```
# Index and value of influencer where C>0.5
np.argmax(c) , np.max(c)
```

Out[16]:
```
(80, 78.7295058224947)
```

In [17]:
```
data[data.index.isin([80])]
```

Out[17]:

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

Let's improve the model by deleting the influence point and creating a new dataframe

In [18]:
```
dataframe= data.copy()
# Discard the data points which are influencers and reassign the row number (reset_index(c
dataframe=dataframe.drop(dataframe.index[[80]],axis=0).reset_index(drop=True)
dataframe.head()
```

Out[18]:

	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

In [21]:
```
data.shape
```

Out[21]:
```
(1435, 9)
```

Model Deletion Diagnostics and Final Model

In [19]:
```
# Another Method
"""k=dataframe.shape[1]
n=dataframe.shape[0]
leverage_cutoff = (3*(k+1))/n
while np.max(c)>leverage_cutoff:
    model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=dataframe).fit()
    (c,_)=model.get_influence().cooks_distance
    c
    np.argmax(c) , np.max(c)
    dataframe=dataframe.drop(dataframe.index[[np.argmax(c)]],axis=0).reset_index(drop=True
    dataframe
else:
    final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=dataframe).fit()
    final_model.rsquared , final_model.aic
    print("Thus model accuracy is improved to",final_model.rsquared)"""
```

```
Thus model accuracy is improved to 0.8960864004304145
```

In [20]:
```
while model.rsquared < 0.90:
    for c in [np.max(c)>leverage_cutoff]:
        model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=dataframe).fit()
        (c,_)=model.get_influence().cooks_distance
        c
        np.argmax(c) , np.max(c)
        dataframe=dataframe.drop(dataframe.index[[np.argmax(c)]],axis=0).reset_index(drop=
        dataframe
    else:
        final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=dataframe).fit
        final_model.rsquared , final_model.aic
        print("Thus model accuracy is improved to",final_model.rsquared)
```

```
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 0.8895239558162494
Thus model accuracy is improved to 0.8898960234448476
Thus model accuracy is improved to 0.8903208318396924
Thus model accuracy is improved to 0.8908014686337989
Thus model accuracy is improved to 0.8901005575125875
Thus model accuracy is improved to 0.8894678831369645
Thus model accuracy is improved to 0.8894880027099145
Thus model accuracy is improved to 0.8900243177601597
Thus model accuracy is improved to 0.8894961653289413
Thus model accuracy is improved to 0.8888300690207419
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 0.8905494548555107
Thus model accuracy is improved to 0.8907035715786529
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 0.8933496039941571
Thus model accuracy is improved to 0.8936856658122996
Thus model accuracy is improved to 0.8944751976424921
Thus model accuracy is improved to 0.893839548043574
Thus model accuracy is improved to 0.8931993880518596
Thus model accuracy is improved to 0.8925409392511808
Thus model accuracy is improved to 0.8933274017072572
Thus model accuracy is improved to 0.8919970006989908
Thus model accuracy is improved to 0.891304915520342
Thus model accuracy is improved to 0.8919967701898639
Thus model accuracy is improved to 0.8925697842477825
Thus model accuracy is improved to 0.892733433674135
Thus model accuracy is improved to 0.8930373667898379
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 0.8948527054861177
Thus model accuracy is improved to 0.8953342154393777
Thus model accuracy is improved to 0.895598439180588
Thus model accuracy is improved to 0.8959887924407283
Thus model accuracy is improved to 0.8949134651663618
Thus model accuracy is improved to 0.8951494863024942
Thus model accuracy is improved to 0.8955005725113694
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 0.8956028020870668
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 0.8968869273251256
Thus model accuracy is improved to 0.8965702611797325
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 0.8965027704321634
Thus model accuracy is improved to 0.8969285366970443
Thus model accuracy is improved to 0.8970144437559693
Thus model accuracy is improved to 0.897464618358938
Thus model accuracy is improved to 0.8977004418350514
Thus model accuracy is improved to 0.8979562106578317
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 0.8995264042658645
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 [22]:
```
influence_points=final_model.get_influence()
c, p_value=influence_points.cooks_distance
```

In [23]:
```
dataframe.shape
```

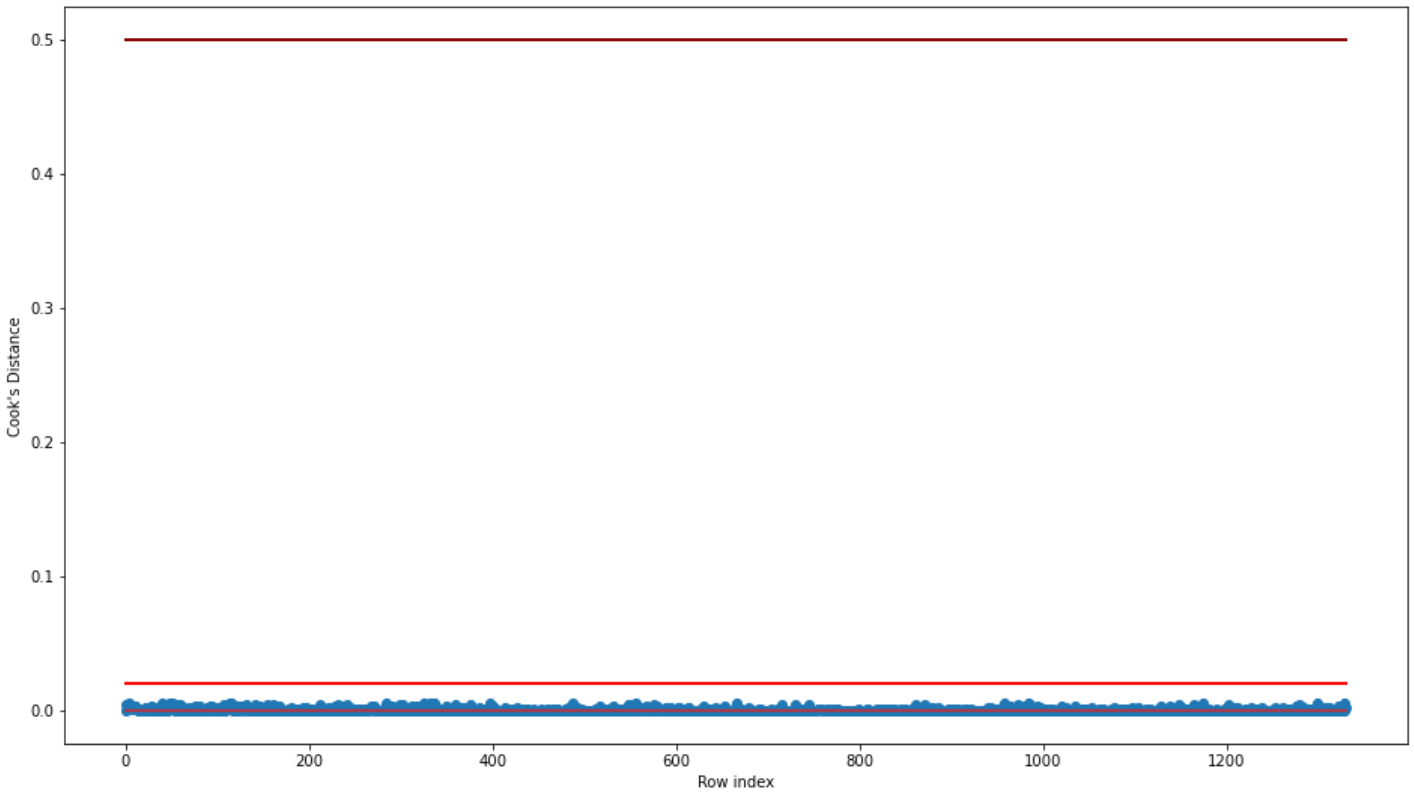
Out[23]:
```
(1330, 9)
```

In [24]:
```
dataframe.head()
```

Out[24]:

	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

In [26]:
```
fig = plt.figure(figsize = (16,9))
x = [0,1330]
y = [0.5,0.5]
plt.plot(x, y,color='darkred', linewidth=2)
y1 = [0.02,0.02]
plt.plot(x , y1, color = 'red', linewidth = 2)
plt.stem(np.arange(len(dataframe)), np.round(c, 3))
plt.xlabel('Row index')
plt.ylabel("Cook's Distance")
plt.show()
```



^Observation: All the points are below our cut-off line

- **Hence, we can say that there are no influncers present in our model we can proceed with the predictions**

In [27]:
```
final_model.summary()
```

Out[27]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.900
Model:	OLS	Adj. R-squared:	0.900
Method:	Least Squares	F-statistic:	1492.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:04:19	Log-Likelihood:	-11038.
No. Observations:	1330	AIC:	2.209e+04
Df Residuals:	1321	BIC:	2.214e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.864e+04	1513.105	-12.316	0.000	-2.16e+04	-1.57e+04
Age	-108.2573	2.181	-49.648	0.000	-112.535	-103.980
KM	-0.0155	0.001	-14.830	0.000	-0.018	-0.013
HP	7.8533	3.037	2.586	0.010	1.895	13.812
CC	-2.2731	0.291	-7.799	0.000	-2.845	-1.701
Doors	-149.1873	32.432	-4.600	0.000	-212.811	-85.563
Gears	251.4363	163.559	1.537	0.124	-69.428	572.301
QT	-11.6187	1.626	-7.144	0.000	-14.809	-8.428
Weight	36.9768	1.446	25.566	0.000	34.139	39.814

Omnibus:	5.509	Durbin-Watson:	1.826
Prob(Omnibus):	0.064	Jarque-Bera (JB):	5.515
Skew:	0.158	Prob(JB):	0.0635
Kurtosis:	2.993	Cond. No.	4.34e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.34e+06. This might indicate that there are strong multicollinearity or other numerical problems.

In [28]:
```
np.sqrt(final_model.mse_resid)
```

Out[28]:
```
975.9271399262799
```

Feature Engineering

Applying some Data Transformation to increase the linear realtionship and improve our model prediction as well it scores

Log-Transformation

In [29]:
```
df_log_scaled = pd.DataFrame()
df_log_scaled['Age'] = np.log(dataframe.Age)
df_log_scaled['Price'] = np.log(dataframe.Price)
df_log_scaled['KM'] = np.log(dataframe.KM)
df_log_scaled['Weight'] = np.log(dataframe.Weight)
df_log_scaled['CC'] = dataframe['CC']
df_log_scaled['Doors'] = dataframe['Doors']
df_log_scaled['HP'] = dataframe['HP']
df_log_scaled.head()
```

Out[29]:

	Age	Price	KM	Weight	CC	Doors	HP
0	3.135494	9.528794	11.197351	7.060476	2000	3	90
1	3.258097	9.612467	10.778956	7.060476	2000	3	90
2	3.401197	9.528794	10.558414	7.064759	2000	3	90
3	3.465736	9.468851	11.018629	7.064759	2000	3	90
4	3.295837	9.735069	11.457540	7.126891	2000	3	90

In [30]:
```
log_transformed_model = smf.ols("Price~Age+KM+HP+CC+Doors+Weight", data = df_log_scaled).f
# Finding rsquared values for Log transformation
log_transformed_model.summary()
```

Out[30]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.768
Model:	OLS	Adj. R-squared:	0.767
Method:	Least Squares	F-statistic:	729.6
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:04:40	Log-Likelihood:	843.01
No. Observations:	1330	AIC:	-1672.
Df Residuals:	1323	BIC:	-1636.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5.3980	1.353	-3.991	0.000	-8.052	-2.745
Age	-0.4079	0.013	-30.586	0.000	-0.434	-0.382
KM	0.0034	0.006	0.594	0.552	-0.008	0.015
HP	0.0032	0.000	11.096	0.000	0.003	0.004
CC	-0.0003	3.33e-05	-8.494	0.000	-0.000	-0.000
Doors	-0.0039	0.004	-0.927	0.354	-0.012	0.004
Weight	2.3427	0.197	11.867	0.000	1.955	2.730

Omnibus:	284.802	Durbin-Watson:	1.179
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1143.768
Skew:	-0.976	Prob(JB):	4.31e-249
Kurtosis:	7.103	Cond. No.	6.06e+05

Notes:

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

[2] The condition number is large, 6.06e+05. This might indicate that there are strong multicollinearity or other numerical problems.

Cube-Root Transformation

In [31]:
```
df_cbrt_scaled = pd.DataFrame()
df_cbrt_scaled['Age'] = np.cbrt(dataframe.Age)
df_cbrt_scaled['Price'] = np.cbrt(dataframe.Price)
df_cbrt_scaled['KM'] = np.cbrt(dataframe.KM)
df_cbrt_scaled['Weight'] = np.cbrt(dataframe.Weight)
df_cbrt_scaled['CC'] = dataframe['CC']
df_cbrt_scaled['QT'] = dataframe['QT']
df_cbrt_scaled['Doors'] = dataframe['Doors']
df_cbrt_scaled['Gears'] = dataframe['Gears']
df_cbrt_scaled['HP'] = dataframe['HP']
df_cbrt_scaled.head()
```

Out[31]:

	Age	Price	KM	Weight	CC	QT	Doors	Gears	HP
0	2.843867	23.957099	41.781366	10.522251	2000	210	3	5	90
1	2.962496	24.634688	36.342412	10.522251	2000	210	3	5	90
2	3.107233	23.957099	33.766567	10.537282	2000	210	3	5	90
3	3.174802	23.483163	39.364972	10.537282	2000	210	3	5	90
4	3.000000	25.662299	45.566822	10.757791	2000	210	3	5	90

In [32]:
```
cbrt_transformed_model = smf.ols("Price~Age+KM+HP+CC+Doors+Gears+QT+Weight", data = df_cbr
# Finding rsquared values for Cube-Root transformation
cbrt_transformed_model.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.857
Model:	OLS	Adj. R-squared:	0.856
Method:	Least Squares	F-statistic:	991.6
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:04:44	Log-Likelihood:	-1505.8
No. Observations:	1330	AIC:	3030.
Df Residuals:	1321	BIC:	3076.
Df Model:	8		
Covariance Type:	nonrobust		

f	std err	t	P>|t|	[0.025	0.975]

Intercept	-18.5991	3.723	-4.996	0.000	-25.902	-11.296
Age	-2.5835	0.069	-37.297	0.000	-2.719	-2.448
KM	-0.0450	0.004	-11.317	0.000	-0.053	-0.037
HP	0.0139	0.002	6.029	0.000	0.009	0.018
CC	-0.0013	0.000	-5.690	0.000	-0.002	-0.001
Doors	-0.0179	0.025	-0.707	0.480	-0.067	0.032
Gears	0.2967	0.126	2.350	0.019	0.049	0.544
QT	-0.0016	0.001	-1.232	0.218	-0.004	0.001
Weight	5.0086	0.375	13.353	0.000	4.273	5.744

Omnibus:	83.976	Durbin-Watson:	1.547
Prob(Omnibus):	0.000	Jarque-Bera (JB):	154.525
Skew:	-0.447	Prob(JB):	2.79e-34
Kurtosis:	4.410	Cond. No.	2.84e+05

Notes:

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

[2] The condition number is large, 2.84e+05. This might indicate that there are strong multicollinearity or other numerical problems.

Square-Root Transformation

In [33]:
```
df_sqrt_scaled = pd.DataFrame()
df_sqrt_scaled['Age'] = np.sqrt(dataframe.Age)
df_sqrt_scaled['Price'] = np.sqrt(dataframe.Price)
df_sqrt_scaled['KM'] = np.sqrt(dataframe.KM)
df_sqrt_scaled['Weight'] = np.sqrt(dataframe.Weight)
df_sqrt_scaled['CC'] = dataframe['CC']
df_sqrt_scaled['QT'] = dataframe['QT']
df_sqrt_scaled['Doors'] = dataframe['Doors']
df_sqrt_scaled['Gears'] = dataframe['Gears']
df_sqrt_scaled['HP'] = dataframe['HP']
df_sqrt_scaled.head()
```

Out[33]:

	Age	Price	KM	Weight	CC	QT	Doors	Gears	HP
0	4.795832	117.260394	270.068510	34.132096	2000	210	3	5	90
1	5.099020	122.270193	219.089023	34.132096	2000	210	3	5	90
2	5.477226	117.260394	196.214169	34.205263	2000	210	3	5	90
3	5.656854	113.798067	246.981781	34.205263	2000	210	3	5	90
4	5.196152	130.000000	307.590637	35.284558	2000	210	3	5	90

In [34]:
```
sqrt_transformed_model = smf.ols("Price~Age+KM+HP+CC+Doors+Gears+QT+Weight", data = df_sqr
# Finding rsquared values for Square-Root transformation
sqrt_transformed_model.summary()
```

Out[34]:

OLS Regression Results

	Price	R-squared:	0.882

Model:	OLS	Adj. R-squared:	0.881
Method:	Least Squares	F-statistic:	1229.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:04:51	Log-Likelihood:	-3993.0
No. Observations:	1330	AIC:	8004.
Df Residuals:	1321	BIC:	8051.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-100.7308	15.607	-6.454	0.000	-131.348	-70.114
Age	-6.7197	0.157	-42.795	0.000	-7.028	-6.412
KM	-0.0384	0.003	-13.802	0.000	-0.044	-0.033
HP	0.0766	0.015	5.084	0.000	0.047	0.106
CC	-0.0082	0.001	-5.622	0.000	-0.011	-0.005
Doors	-0.2049	0.164	-1.251	0.211	-0.526	0.116
Gears	2.0321	0.819	2.481	0.013	0.425	3.639
QT	-0.0209	0.008	-2.563	0.010	-0.037	-0.005
Weight	7.9388	0.500	15.875	0.000	6.958	8.920

Omnibus:	7.852	Durbin-Watson:	1.731
Prob(Omnibus):	0.020	Jarque-Bera (JB):	8.528
Skew:	-0.128	Prob(JB):	0.0141
Kurtosis:	3.297	Cond. No.	1.85e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.85e+05. This might indicate that there are strong multicollinearity or other numerical problems.

Let's try Robust transformation

The Robust Scaler, as the name suggests is not sensitive to outliers.

- **This scaler removes the median from the data**
- **Scales the data by the InterQuartile Range(IQR)**

The interquartile range can be defined as-

IQR = Q3 – Q1

Thus, the formula would be:

x_scaled = (x – Q1)/(Q3 – Q1)

```
led = dataframe.copy()
```

```
columns= ['Price','Age','KM','Weight']
features = df_robust_scaled[columns]

from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()

df_robust_scaled[columns] = scaler.fit_transform(features.values)
df_robust_scaled.head()
```

Out[35]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	1.175000	-1.583333	0.211503	90	2000	3	5	210	2.222222
1	1.538636	-1.458333	-0.375961	90	2000	3	5	210	2.222222
2	1.175000	-1.291667	-0.599762	90	2000	3	5	210	2.333333
3	0.932576	-1.208333	-0.069708	90	2000	3	5	210	2.333333
4	2.129545	-1.416667	0.722122	90	2000	3	5	210	4.000000

In [51]:

```
robust_transformed_model = smf.ols("Price~Age+KM+HP+CC+Doors+Gears+QT+Weight", data = df_r
# Finding rsquared values for robust transformation
robust_transformed_model.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.896
Model:	OLS	Adj. R-squared:	0.895
Method:	Least Squares	F-statistic:	1516.
Date:	Sun, 10 Apr 2022	Prob (F-statistic):	0.00
Time:	23:11:38	Log-Likelihood:	-404.63
No. Observations:	1415	AIC:	827.3
Df Residuals:	1406	BIC:	874.6
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.4226	0.286	1.480	0.139	-0.138	0.983
Age	-0.8244	0.017	-47.575	0.000	-0.858	-0.790
KM	-0.2189	0.014	-15.480	0.000	-0.247	-0.191
HP	0.0078	0.001	9.901	0.000	0.006	0.009
CC	-0.0010	8.23e-05	-11.570	0.000	-0.001	-0.001
Doors	-0.0396	0.010	-3.938	0.000	-0.059	-0.020
Gears	0.0899	0.051	1.780	0.075	-0.009	0.189
QT	0.0003	0.000	0.760	0.447	-0.000	0.001
Weight	0.4221	0.017	25.538	0.000	0.390	0.454

Omnibus:	20.694	Durbin-Watson:	1.765
Prob(Omnibus):	0.000	Jarque-Bera (JB):	25.015
Skew:	0.212	Prob(JB):	3.70e-06
	3.494	Cond. No.	5.32e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.32e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Applying Standard Scaler

- **For each feature, the Standard Scaler scales the values such that the mean is 0 and the standard deviation is 1(or the variance).**
- **x_scaled = x – mean/std_dev**
- **However, Standard Scaler assumes that the distribution of the variable is normal. Thus, in case, the variables are not normally distributed, we either choose a different scaler or first, convert the variables to a normal distribution and then apply this scaler**

In [36]:
```
from sklearn.preprocessing import StandardScaler

col_names = dataframe.columns
features = dataframe[col_names]

scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
df_standard_scaled = pd.DataFrame(features, columns = col_names)
df_standard_scaled.head()
```

Out[36]:

	Price	Age	KM	HP	CC	Doors	Gears	QT	Weight
0	1.057454	-1.915503	0.136289	-0.883769	2.526466	-1.067888	-0.137393	3.424789	2.456469
1	1.446875	-1.746462	-0.576515	-0.883769	2.526466	-1.067888	-0.137393	3.424789	2.456469
2	1.057454	-1.521074	-0.848065	-0.883769	2.526466	-1.067888	-0.137393	3.424789	2.580538
3	0.797839	-1.408380	-0.204921	-0.883769	2.526466	-1.067888	-0.137393	3.424789	2.580538
4	2.079684	-1.690115	0.755851	-0.883769	2.526466	-1.067888	-0.137393	3.424789	4.441570

In [37]:
```
standard_scaler_transformed_model = smf.ols("Price~Age+KM+HP+CC+Doors+Gears+QT+Weight", da
# Finding rsquared values for standard scaler transformation
standard_scaler_transformed_model.summary()
```

Out[37]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.900
Model:	OLS	Adj. R-squared:	0.900
Method:	Least Squares	F-statistic:	1492.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:04:58	Log-Likelihood:	-353.46
No. Observations:	1330	AIC:	724.9
Df Residuals:	1321	BIC:	771.7
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.009	-2.76e-14	1.000	-0.017	0.017
Age	-0.6235	0.013	-49.648	0.000	-0.648	-0.599
KM	-0.1756	0.012	-14.830	0.000	-0.199	-0.152
HP	0.0325	0.013	2.586	0.010	0.008	0.057
CC	-0.1307	0.017	-7.799	0.000	-0.164	-0.098
Doors	-0.0462	0.010	-4.600	0.000	-0.066	-0.026
Gears	0.0138	0.009	1.537	0.124	-0.004	0.032
QT	-0.1368	0.019	-7.144	0.000	-0.174	-0.099
Weight	0.4836	0.019	25.566	0.000	0.446	0.521

Omnibus:	5.509	Durbin-Watson:	1.826
Prob(Omnibus):	0.064	Jarque-Bera (JB):	5.515
Skew:	0.158	Prob(JB):	0.0635
Kurtosis:	2.993	Cond. No.	4.64

Notes:

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

^Observation: After the transformation and building models the R-Squared had varince with respect to other transformations

- **But standard scaler is better than raw data model and other models with better AIC, BIC log-likelihood scores**
- **We have to perform model validation test to check which model is better will do at the end of this**

For building Multi Linear Resgression there are assumption regarding the data set.

They are as follows:-

1. **Feature should be independent of each other there should'nt be any dependency upon each other**
2. **There shouldn't any other relation but Linear relation amongst model parameters (Hyperparameters of the model the intercept and coefficient)**
3. **Each Feature and Model Error (residuals) should be independent of each other**
4. **Constant Variance (Homoscedasticity) in Error, it should have Normal / Gaussian distribution~N(0,1) and idenpendently and identically distributed.**
5. **There should be a linear relation between the dependent variable and Independent variables**

We will Check the above one by one

Preparing a Model

In [38]:
```
model = smf.ols("Price~Age+KM+HP+CC+Doors+Gears+QT+Weight", data = df_standard_scaled).fit
model.summary()
```

Out[38]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.900
Model:	OLS	Adj. R-squared:	0.900
Method:	Least Squares	F-statistic:	1492.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:07:45	Log-Likelihood:	-353.46
No. Observations:	1330	AIC:	724.9
Df Residuals:	1321	BIC:	771.7
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.009	-2.76e-14	1.000	-0.017	0.017
Age	-0.6235	0.013	-49.648	0.000	-0.648	-0.599
KM	-0.1756	0.012	-14.830	0.000	-0.199	-0.152
HP	0.0325	0.013	2.586	0.010	0.008	0.057
CC	-0.1307	0.017	-7.799	0.000	-0.164	-0.098
Doors	-0.0462	0.010	-4.600	0.000	-0.066	-0.026
Gears	0.0138	0.009	1.537	0.124	-0.004	0.032
QT	-0.1368	0.019	-7.144	0.000	-0.174	-0.099
Weight	0.4836	0.019	25.566	0.000	0.446	0.521

Omnibus:	5.509	Durbin-Watson:	1.826
Prob(Omnibus):	0.064	Jarque-Bera (JB):	5.515
Skew:	0.158	Prob(JB):	0.0635
Kurtosis:	2.993	Cond. No.	4.64

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

- **Summary The values we are concerned with are -**

The coefficients and significance (p-values) R-squared F statistic and its significance

1. **R - squared is 0.896 Meaning that 89.9% of the variance in cnt with registered**

This is a decent R-squared value.

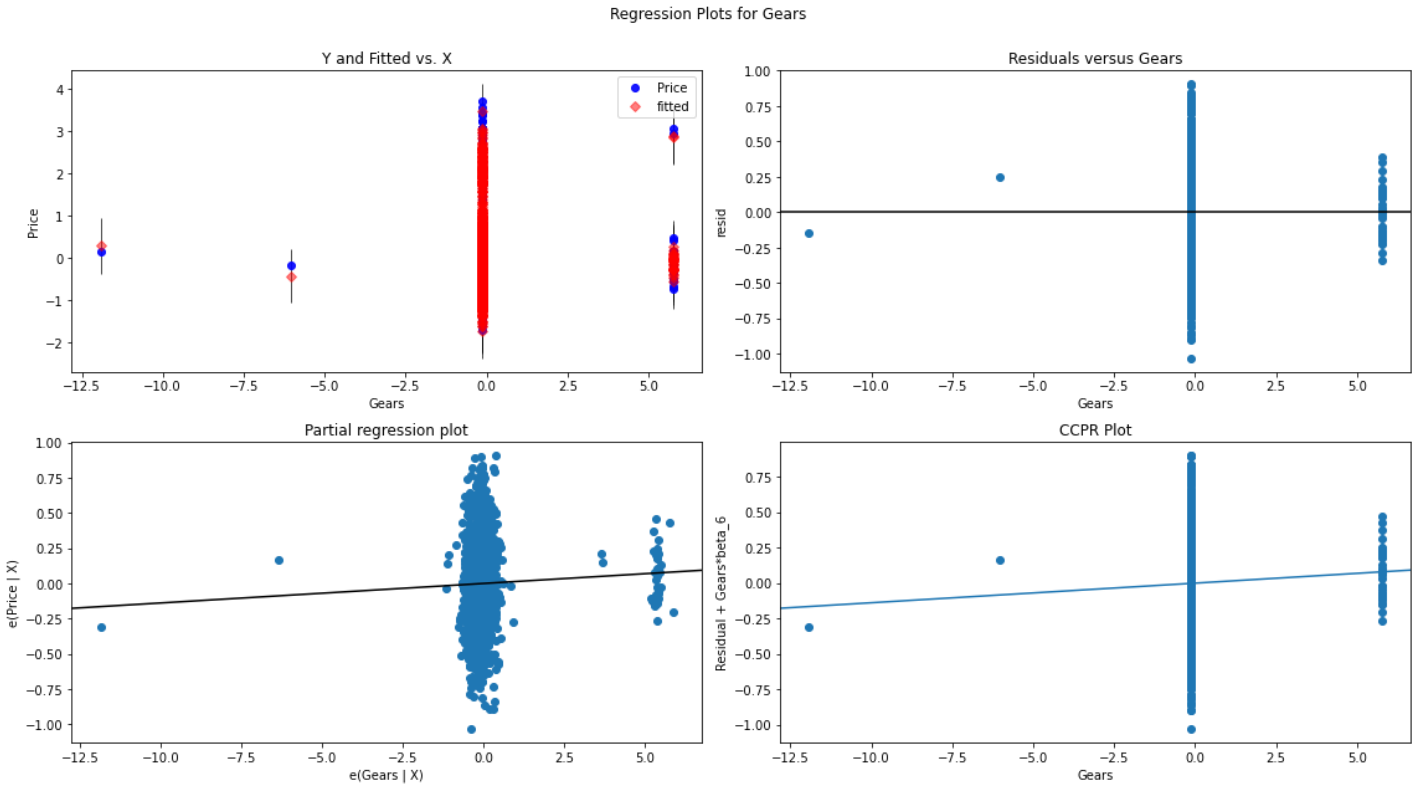
3.F statistic has a very low p value (practically low) Meaning that the model fit is statistically significant, and the explained variance isn't purely by chance.

Note: If any of the above step is not followed our model can't be a good predictor

In [39]:

```
fig = plt.figure(figsize = (16,9))
lot_regress_exog(model, 'Gears', fig=fig)
```

```
plt.show()
```



^Observation No Linear Relation found in QT Feature with the Dependent feature

Model Testing

As Y = Beta0 + Beta1*(X1)* + *Beta2*(X2) +Beta3*(X3)* + ... + *Beta n*(Xn)

Finding Coefficient Parameters (Beta0 and Beta1's values)

Assupmtion for multi linear Regression fails

Feature should be independent of each other there should'nt be any dependency upon each other

Here, (Intercept) Beta0 p_value ~ 1

Hypothesis testing of X variable by finding test_statistics and P_values for Beta1 i.e if (P_value < α=0.05 ; Reject Null)

Null Hypothesis as Beta1=0 (No Slope) and Alternate Hypthesis as Beta1≠0 (Some or significant Slope)

^Observation: If the p-value is not less than .05 for Gears features, we fail to reject the null hypothesis. We do not have sufficient evidence to say that the sample data providing those features dependency towards the dependent variable

- **Looking at the p-values, it looks like some of the variables aren't really significant (in the presence of other variables).**
- **Maybe we could drop some?**
- **We could simply drop the variable with the highest, non-significant p value. A better way would be to supplement this with the VIF information.**

er Test by Calculating VIF (Variance Inflation Factor)

- **This helps to check the dependency among the features by building a model without the target and testing various combination among the features**

In [40]:
```
# Create a dataframe that will contain the names of all the feature variables and their re
y = df_standard_scaled.drop(['Price'], axis=1)
vif = pd.DataFrame()
vif['Features'] = y.columns
vif['VIF'] = [variance_inflation_factor(y.values, i) for i in range(y.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[40]:

	Features	VIF
6	QT	4.86
7	Weight	4.74
3	CC	3.73
0	Age	2.09
2	HP	2.09
1	KM	1.86
4	Doors	1.34
5	Gears	1.08

In [41]:
```
r_sqr_age = smf.ols('Age~HP+Weight+CC+Doors+QT+Gears+KM', dataframe).fit().rsquared
vif_age = 1/(1-r_sqr_age)
r_sqr_weight = smf.ols('Weight~HP+Age+CC+Doors+QT+Gears+KM', dataframe).fit().rsquared
vif_weight = 1/(1-r_sqr_weight)
r_sqr_cc = smf.ols('CC~HP+Weight+Age+Doors+QT+Gears+KM', dataframe).fit().rsquared
vif_cc = 1/(1-r_sqr_cc)
r_sqr_hp = smf.ols('HP~Age+Weight+CC+Doors+QT+Gears+KM', dataframe).fit().rsquared
vif_hp = 1/(1-r_sqr_hp)
r_sqr_qt = smf.ols('QT~HP+Weight+CC+Doors+Age+Gears+KM', dataframe).fit().rsquared
vif_qt = 1/(1-r_sqr_qt)
r_sqr_km = smf.ols('KM~HP+Weight+CC+Doors+QT+Gears+Age', dataframe).fit().rsquared
vif_km = 1/(1-r_sqr_km)
r_sqr_gears = smf.ols('Gears~HP+Weight+CC+Doors+QT+Age+KM', dataframe).fit().rsquared
vif_gears = 1/(1-r_sqr_gears)
r_sqr_doors = smf.ols('Doors~HP+Weight+CC+Age+QT+Gears+KM', dataframe).fit().rsquared
vif_doors = 1/(1-r_sqr_doors)
```

In [42]:
```
vif_frame = pd.DataFrame({'Variables':['Doors','HP','Weight','CC','Age','QT','Gears','KM']
vif_frame.set_index('Variables', inplace = True)
vif_frame.sort_values(by = 'VIF')
```

Out[42]:

Variables	VIF
Gears	1.075124
Doors	1.336868
KM	1.858592
Age	2.091177
HP	2.094861

Variables	VIF
CC	3.725960
Weight	4.744228
QT	4.863714

Note: We generally want a VIF that is less than 5. As you can see QT has the highest value among others lets investigate

In [43]:
```
#Simple Linear Model using QT
qt_model = smf.ols('Price~QT', data=df_standard_scaled).fit()
qt_model.summary()
```

Out[43]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.018
Model:	OLS	Adj. R-squared:	0.017
Method:	Least Squares	F-statistic:	24.53
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	8.27e-07
Time:	15:10:28	Log-Likelihood:	-1875.0
No. Observations:	1330	AIC:	3754.
Df Residuals:	1328	BIC:	3764.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.027	-8.82e-15	1.000	-0.053	0.053
QT	0.1347	0.027	4.952	0.000	0.081	0.188

Omnibus:	268.046	Durbin-Watson:	0.235
Prob(Omnibus):	0.000	Jarque-Bera (JB):	471.056
Skew:	1.259	Prob(JB):	5.15e-103
Kurtosis:	4.469	Cond. No.	1.00

Notes:

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

In [44]:
```
gears_model = smf.ols('Price~Gears', data=df_standard_scaled).fit()
gears_model.summary()
```

Out[44]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.000
Model:	OLS	Adj. R-squared:	-0.001
Method:	Least Squares	F-statistic:	0.08310
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.773

Time:	15:10:39	Log-Likelihood:	-1887.1
No. Observations:	1330	AIC:	3778.
Df Residuals:	1328	BIC:	3789.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.027	-8.74e-15	1.000	-0.054	0.054
Gears	0.0079	0.027	0.288	0.773	-0.046	0.062

Omnibus:	289.163	Durbin-Watson:	0.238
Prob(Omnibus):	0.000	Jarque-Bera (JB):	526.772
Skew:	1.333	Prob(JB):	4.10e-115
Kurtosis:	4.547	Cond. No.	1.00

Notes:

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

In [45]:
```
gears_qt_weight_model = smf.ols('Price~Gears+QT', data=df_standard_scaled).fit()
gears_qt_weight_model.summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.018
Model:	OLS	Adj. R-squared:	0.017
Method:	Least Squares	F-statistic:	12.30
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	5.08e-06
Time:	15:10:48	Log-Likelihood:	-1875.0
No. Observations:	1330	AIC:	3756.
Df Residuals:	1327	BIC:	3772.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.027	-8.82e-15	1.000	-0.053	0.053
Gears	0.0085	0.027	0.313	0.754	-0.045	0.062
QT	0.1347	0.027	4.952	0.000	0.081	0.188

Omnibus:	268.324	Durbin-Watson:	0.234
Prob(Omnibus):	0.000	Jarque-Bera (JB):	471.601
Skew:	1.260	Prob(JB):	3.92e-103
Kurtosis:	4.468	Cond. No.	1.00

Notes:

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

Significance level - Backward elimination

We have different techniques to find out the features which have the maximum effect on the output.

Here we are going to look at the Backward elimination.

In this process we need to add one column of ones in the starting of the column.

In backward elimination we delete the value one by one whose significance level is less.

i.e In general we have a P-value and a significance level

P_value = 1 - (minus) significane level

or in other terms

p_value+ significance level = 1

if P_value is high significance level is less.

Hence we will be deleating features one by one whose P_value is high which means it has less significance level.

By eliminating process we get to the values which are of most significance

Model1

- **Dropping the variable and updating the model**
- **As you can see from the summary and the VIF dataframe, some variables are still insignificant. One of these variables is, Gears as it has a very high P Value of 0.75 in SLR and 0.12 in MLR has R square score of 0. Let's go ahead and drop this variable**

In [121…
```
# Dropping highly correlated variables and insignificant variables

x = df_standard_scaled.drop(['Gears'], axis=1)
y = df_standard_scaled.drop(['Gears','Price'], axis=1)
```

In [122…
```
x.columns
```

Out[122…
```
Index(['Price', 'Age', 'KM', 'HP', 'CC', 'Doors', 'QT', 'Weight'], dtype='object')
```

In [123…
```
model_1 = smf.ols('Price~Age+KM+HP+CC+Doors+QT+Weight', data = x).fit()
model_1.summary()
```

Out[123…

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.900
Model:	OLS	Adj. R-squared:	0.900
Method:	Least Squares	F-statistic:	1703.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
	15:57:08	Log-Likelihood:	-354.65

No. Observations:	1330	AIC:	725.3
Df Residuals:	1322	BIC:	766.8
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.009	-2.76e-14	1.000	-0.017	0.017
Age	-0.6233	0.013	-49.609	0.000	-0.648	-0.599
KM	-0.1743	0.012	-14.750	0.000	-0.197	-0.151
HP	0.0360	0.012	2.915	0.004	0.012	0.060
CC	-0.1329	0.017	-7.955	0.000	-0.166	-0.100
Doors	-0.0493	0.010	-5.016	0.000	-0.069	-0.030
QT	-0.1350	0.019	-7.057	0.000	-0.172	-0.097
Weight	0.4850	0.019	25.656	0.000	0.448	0.522

Omnibus:	4.861	Durbin-Watson:	1.824
Prob(Omnibus):	0.088	Jarque-Bera (JB):	4.872
Skew:	0.148	Prob(JB):	0.0875
Kurtosis:	2.984	Cond. No.	4.63

Notes:

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

^Observation: As our Multicollinearity problem has been solved

Let's compare the residuals of all the models to come up with an conclusion

Model Validation

Comparing different models with respect to their Root Mean Squared Errors

We will analyze Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) — AKA the average distance (squared to get rid of negative numbers) between the model's predicted target value and the actual target value.

In [124…

```
x = dataframe[['Age','KM','HP','CC','Doors','QT','Weight']]
y = dataframe[['Price']]
transformer_x = StandardScaler().fit(x)
transformer_y = StandardScaler().fit(y)
# Scale the test dataset
x_train_scal = transformer_x.transform(x)
y_train_scal = transformer_y.transform(y)

# Linear Regression
x_df = pd.DataFrame(x_train_scal, columns = ['Age','KM','HP','CC','Doors','QT','Weight'])
x_df.head()

# Predict with the trained model
predict = pd.DataFrame(model_1.predict(x_df))
```

```
# Inverse transform the prediction
predict_unscaled = transformer_y.inverse_transform(predict.values.reshape(-1,1))

# Predicting RMSE the Test set results
rmse_linear= (np.sqrt(mean_squared_error(y, predict_unscaled)))
print('R2_score (train): ', model_1.rsquared)
print('R2_Adjusted_score (test): ', model_1.rsquared_adj)
print("RMSE : ", rmse_linear)
```

```
R2_score (train):  0.9001980293966753
R2_Adjusted_score (test):  0.8996695772073989
RMSE :  973.4891302577998
```

In [125…

```
square_root_pred_y =np.square(sqrt_transformed_model.predict(df_sqrt_scaled[['Age','Weight
cube_root_pred_y =pow(cbrt_transformed_model.predict(df_cbrt_scaled[['Age','Weight','KM',
log_model_pred_y =np.exp(log_transformed_model.predict(df_log_scaled[['Age','Weight','KM',
```

In [126…

```
square_root_both_rmse =np.sqrt(mean_squared_error(dataframe['Price'], square_root_pred_y))
cube_root_both_rmse =np.sqrt(mean_squared_error(dataframe['Price'], cube_root_pred_y))
log_both_rmse =np.sqrt(mean_squared_error(dataframe['Price'], log_model_pred_y))
```

In [127…

```
print('Raw Model=', np.sqrt(raw_data_model.mse_resid),
      '\n''After Removing Influencers=', np.sqrt(final_model.mse_resid),
      '\n''After Log Transformation on both Model=', log_both_rmse,
      '\n''After Cube-root Transformation on both Model=', cube_root_both_rmse,
     '\n''After Sqaure Root Transformation on both Model=', square_root_both_rmse,
      '\n''After Removing Influencers from model', np.sqrt(final_model.mse_resid),
     '\n''Final Model without Multicollinearity Model=', rmse_linear)
```

```
Raw Model= 1341.8046186938675
After Removing Influencers= 975.9271399262799
After Log Transformation on both Model= 1795.826559209863
After Cube-root Transformation on both Model= 1152.557794874252
After Sqaure Root Transformation on both Model= 1010.8979447569421
After Removing Influencers from model 975.9271399262799
Final Model without Multicollinearity Model= 973.4891302577998
```

Let's compare the Root Mean Squared Error and check for the minimum value

In [129…

```
rmse_compare = {'Raw Model': np.sqrt(raw_data_model.mse_resid),
      'After Removing Influencers': np.sqrt(final_model.mse_resid),
      'After Log Transformation Model': log_both_rmse,
      'After Cube-root Transformation Model': cube_root_both_rmse,
     'After Sqaure Root Transformation Model': square_root_both_rmse,
                'After Removing Influencers from model': np.sqrt(final_model.mse_resid),
   'Final Model without Multicollinearity Model': rmse_linear}
min(rmse_compare, key=rmse_compare.get)
```

Out[129…

```
'Final Model without Multicollinearity Model'
```

^Observation: The Model that was build without Multicollinearity Issue and using Standard Scaler Transformation performed very well.

- **Scoring minimumn Root mean squared error and a good R-squared and adjusted R-squared**
- **Note: We are going to rebuild the model by using that model**

In [64]:

```
Final_model = smf.ols("Price~Age+KM+HP+CC+Doors+QT+Weight", data = x).fit()
uared values
```

```
Final_model.summary()
```

Out[64]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.900
Model:	OLS	Adj. R-squared:	0.900
Method:	Least Squares	F-statistic:	1703.
Date:	Mon, 11 Apr 2022	Prob (F-statistic):	0.00
Time:	15:24:58	Log-Likelihood:	-354.65
No. Observations:	1330	AIC:	725.3
Df Residuals:	1322	BIC:	766.8
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.398e-16	0.009	-2.76e-14	1.000	-0.017	0.017
Age	-0.6233	0.013	-49.609	0.000	-0.648	-0.599
KM	-0.1743	0.012	-14.750	0.000	-0.197	-0.151
HP	0.0360	0.012	2.915	0.004	0.012	0.060
CC	-0.1329	0.017	-7.955	0.000	-0.166	-0.100
Doors	-0.0493	0.010	-5.016	0.000	-0.069	-0.030
QT	-0.1350	0.019	-7.057	0.000	-0.172	-0.097
Weight	0.4850	0.019	25.656	0.000	0.448	0.522

Omnibus:	4.861	Durbin-Watson:	1.824
Prob(Omnibus):	0.088	Jarque-Bera (JB):	4.872
Skew:	0.148	Prob(JB):	0.0875
Kurtosis:	2.984	Cond. No.	4.63

Notes:

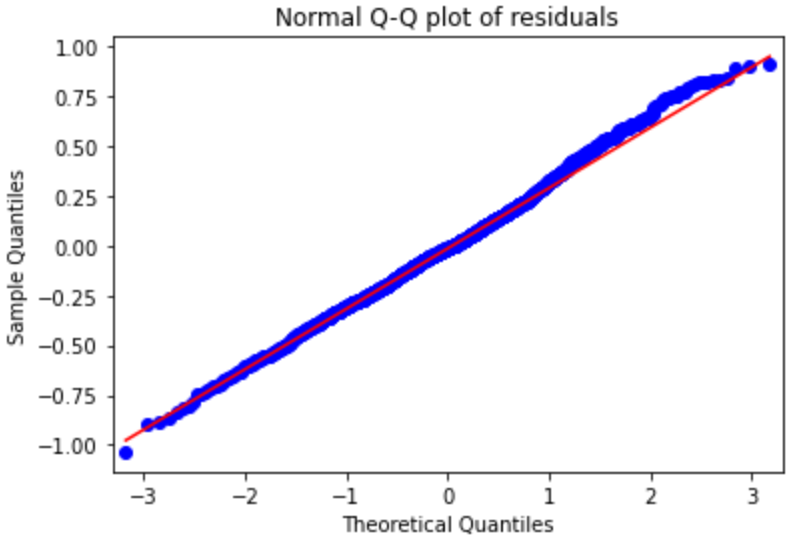
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Residual Analysis

- **Test for Normality of Residuals (Q-Q Plot)**

In [66]:

```
#Residuals values  = y - yhat
sm.qqplot(Final_model.resid, line = 'q')
plt.title('Normal Q-Q plot of residuals')
plt.show()
```

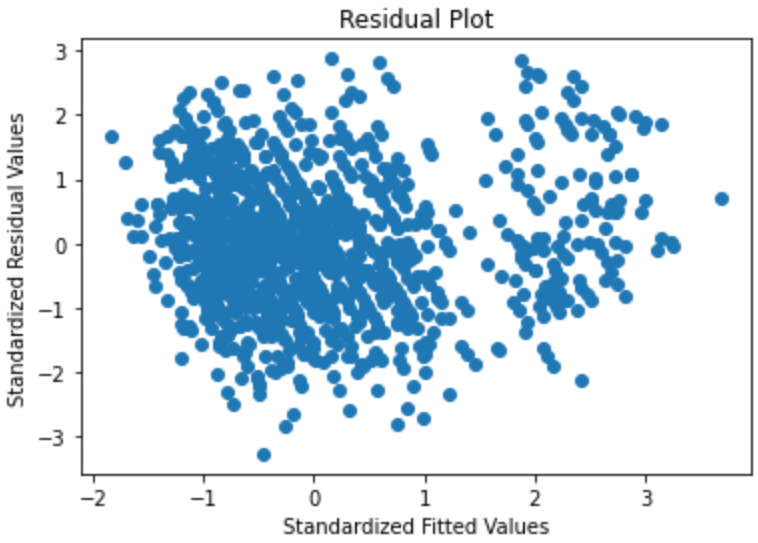


^Observation: Error should have Normal / Gaussian distribution~N(0,1) and idenpendently and identically distributed.

Residual Plot for Homoscedasticity

In [67]:
```
def get_standardized_values( vals ):
    return (vals - vals.mean())/vals.std()
```

In [68]:
```
plt.scatter(get_standardized_values(Final_model.fittedvalues), get_standardized_values(Fir
plt.title('Residual Plot')
plt.xlabel('Standardized Fitted Values')
plt.ylabel('Standardized Residual Values')
plt.show()
```

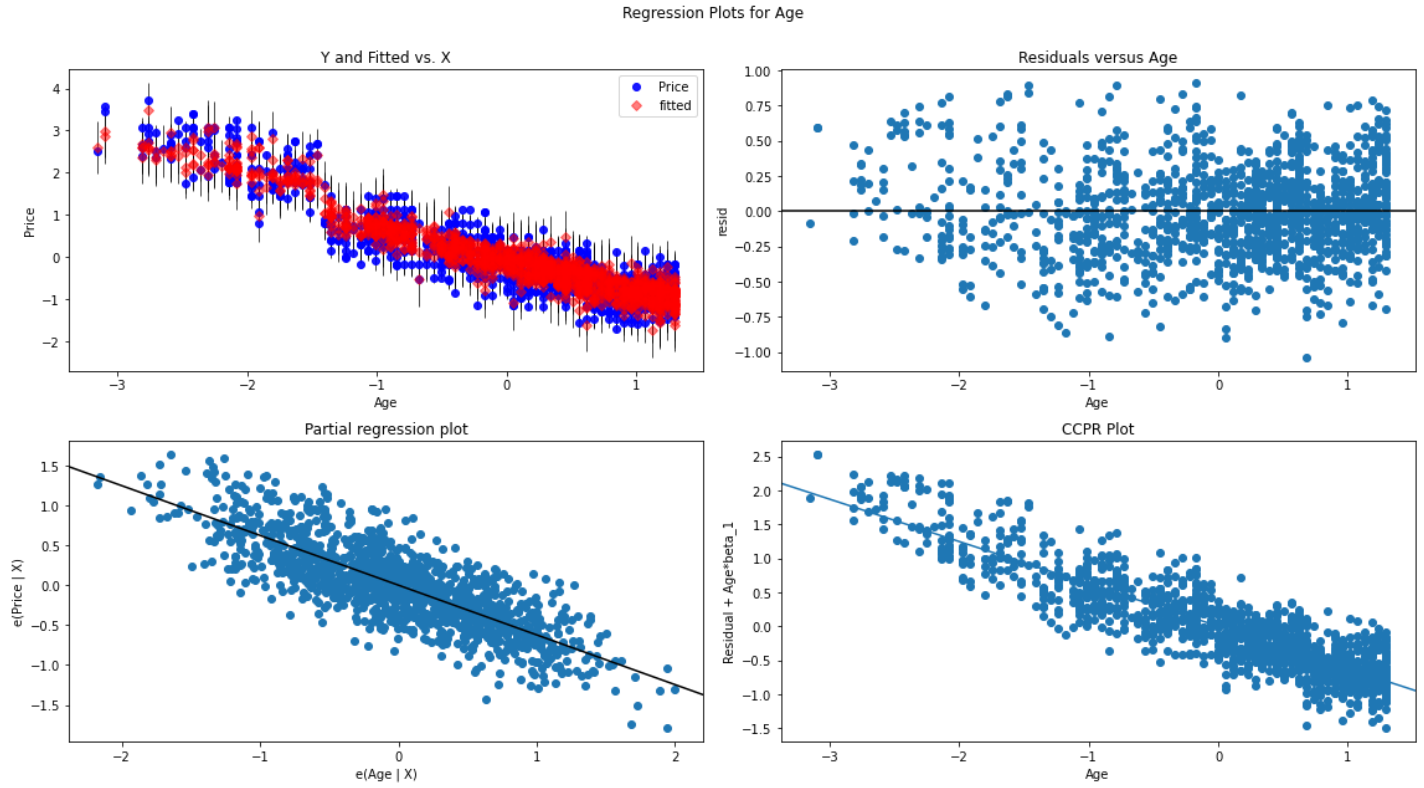


^Observation: Constant Variance (Homoscedasticity) in Error

Residual VS Regressors

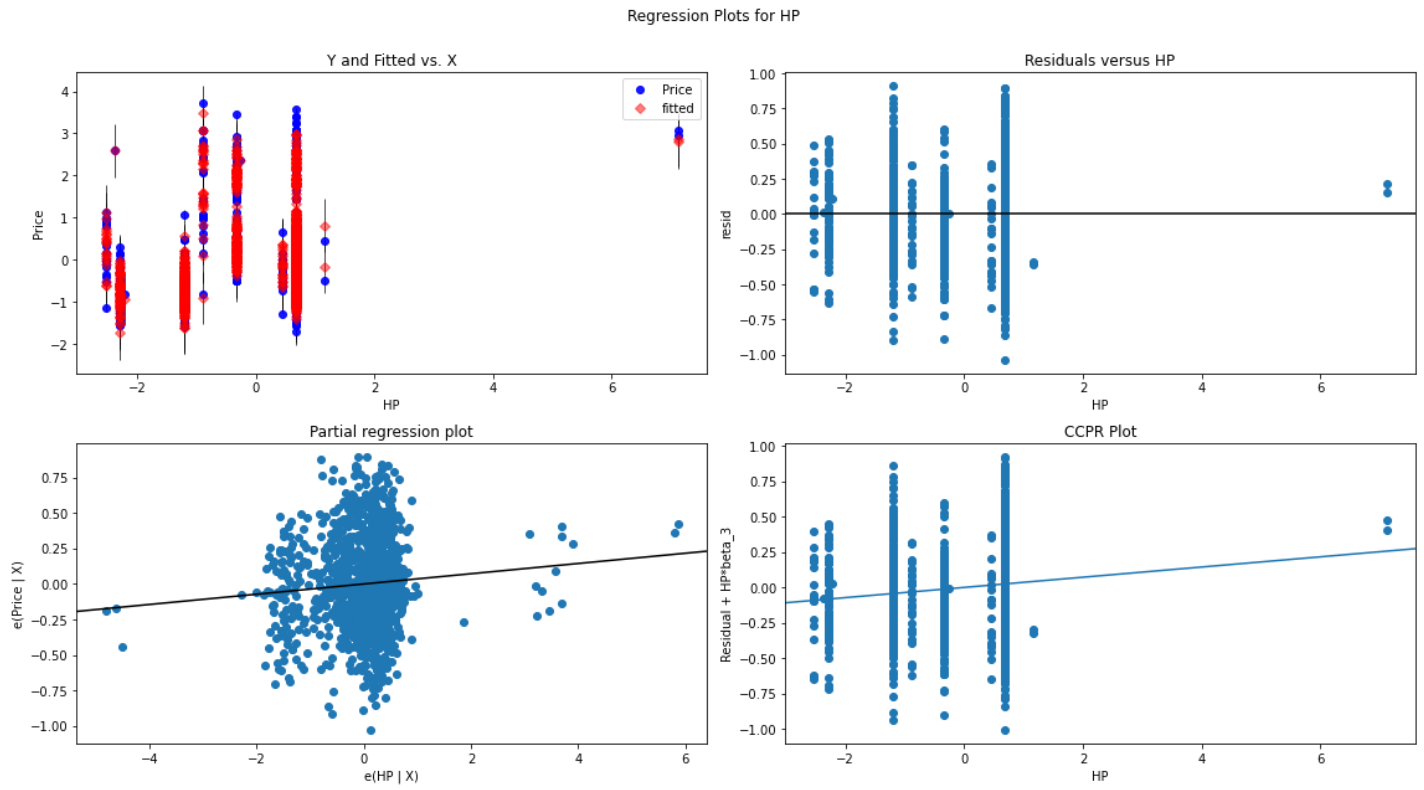
- **Plotting to visualize the partial relation of each independent feature with the Dependent variable and errors**

In [69]:
```
fig = plt.figure(figsize = (16,9))
sm.graphics.plot_regress_exog(Final_model, 'Age', fig=fig)
```



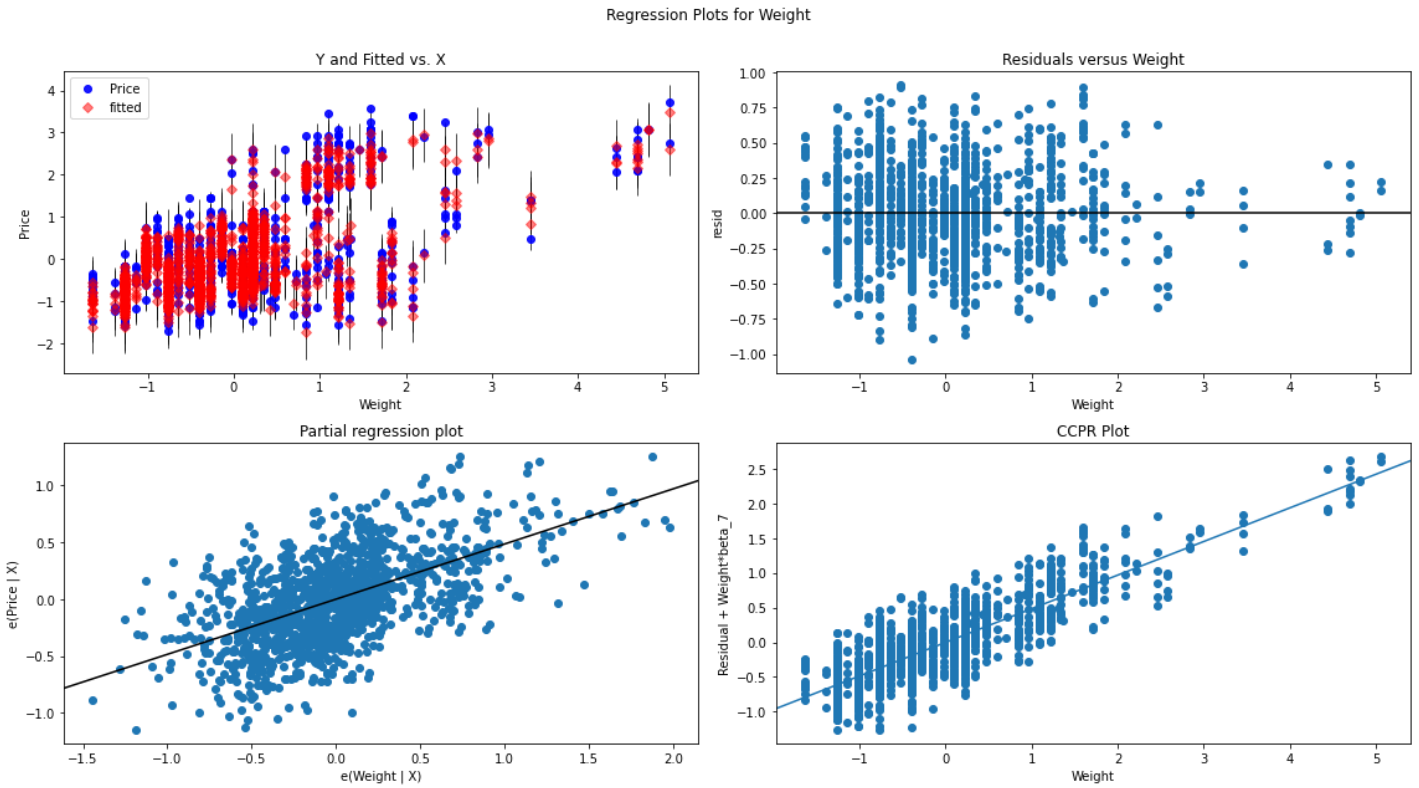
In [70]:

```
fig = plt.figure(figsize = (16,9))
sm.graphics.plot_regress_exog(Final_model, 'HP', fig=fig)
plt.show()
```



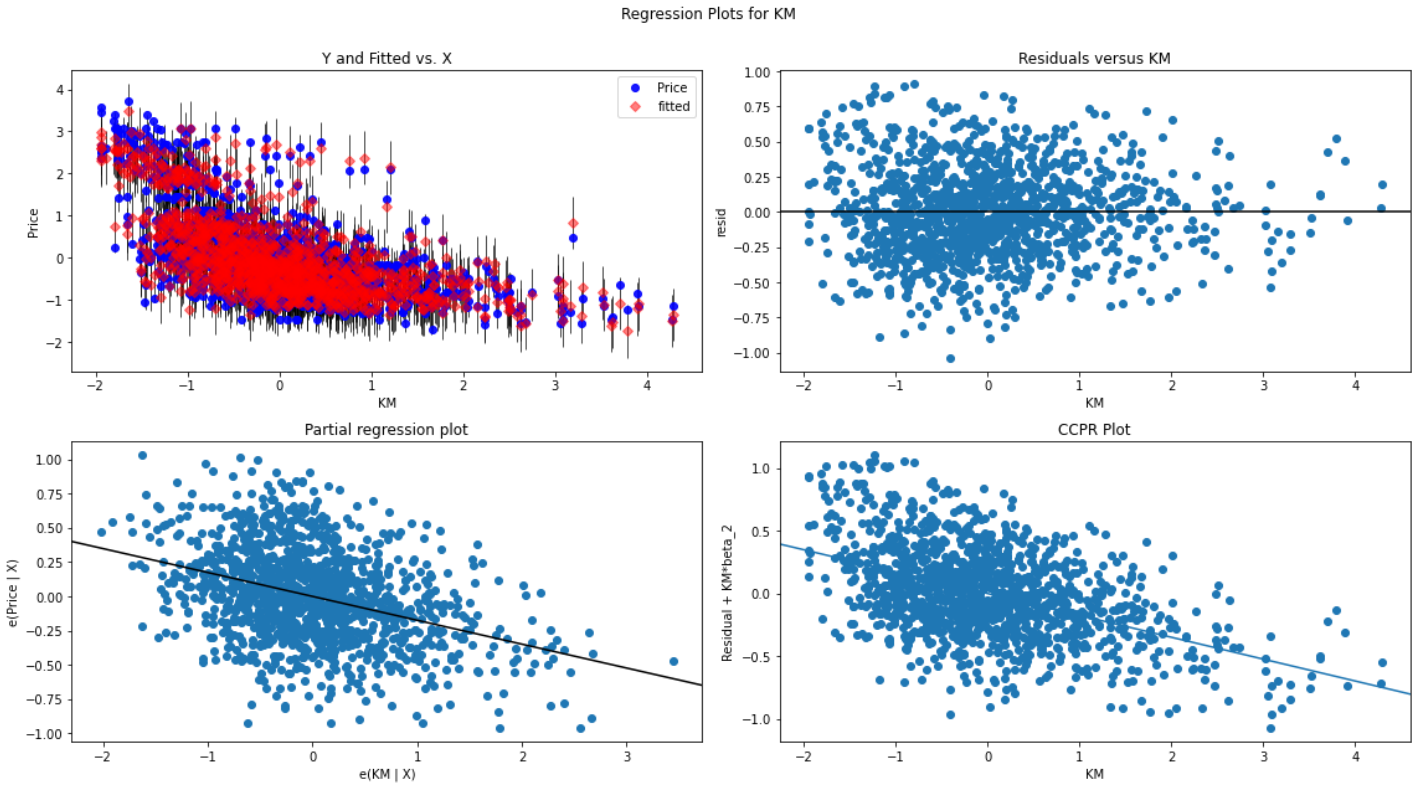
In [71]:

```
fig = plt.figure(figsize = (16,9))
sm.graphics.plot_regress_exog(Final_model, 'Weight', fig=fig)
plt.show()
```



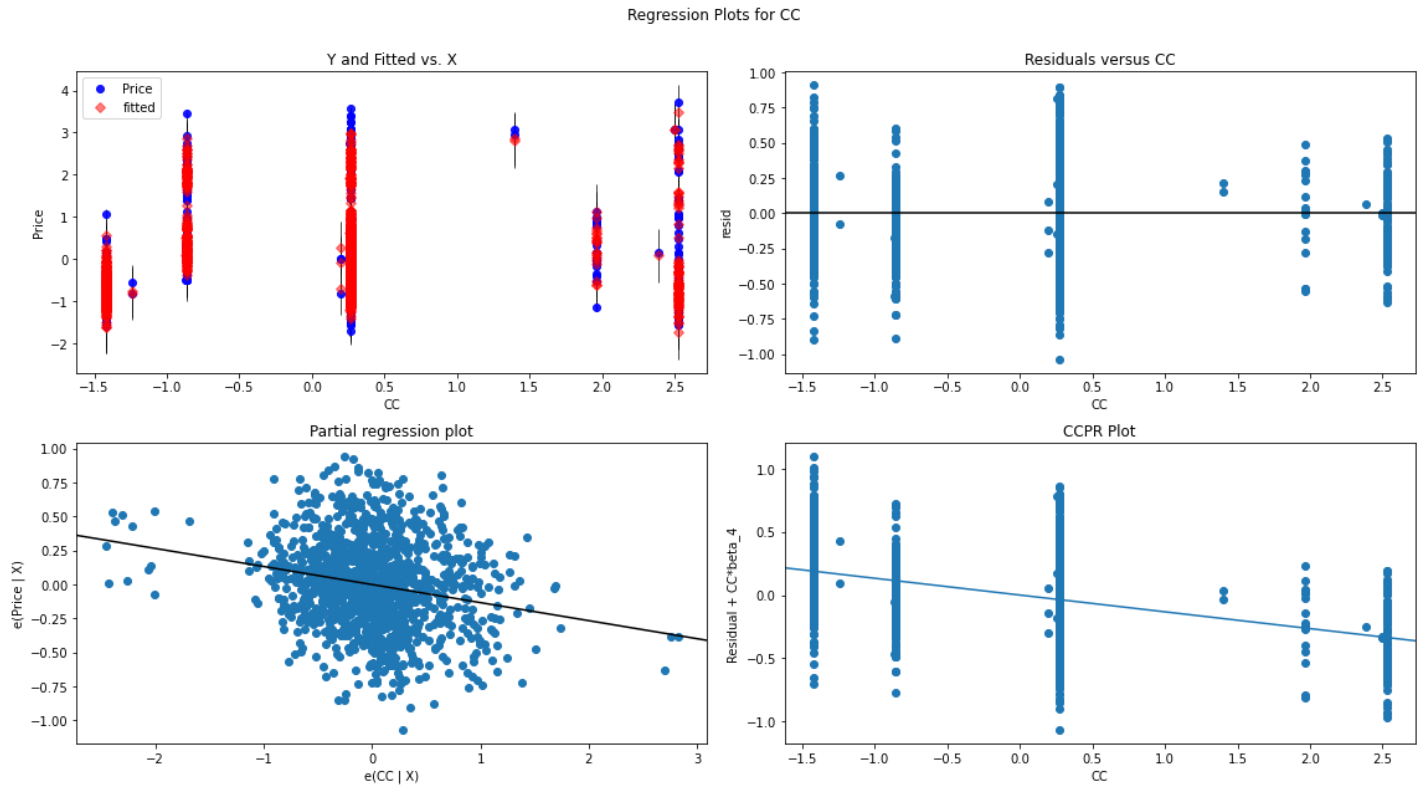
In [72]:

```
fig = plt.figure(figsize = (16,9))
sm.graphics.plot_regress_exog(Final_model, 'KM', fig=fig)
plt.show()
```



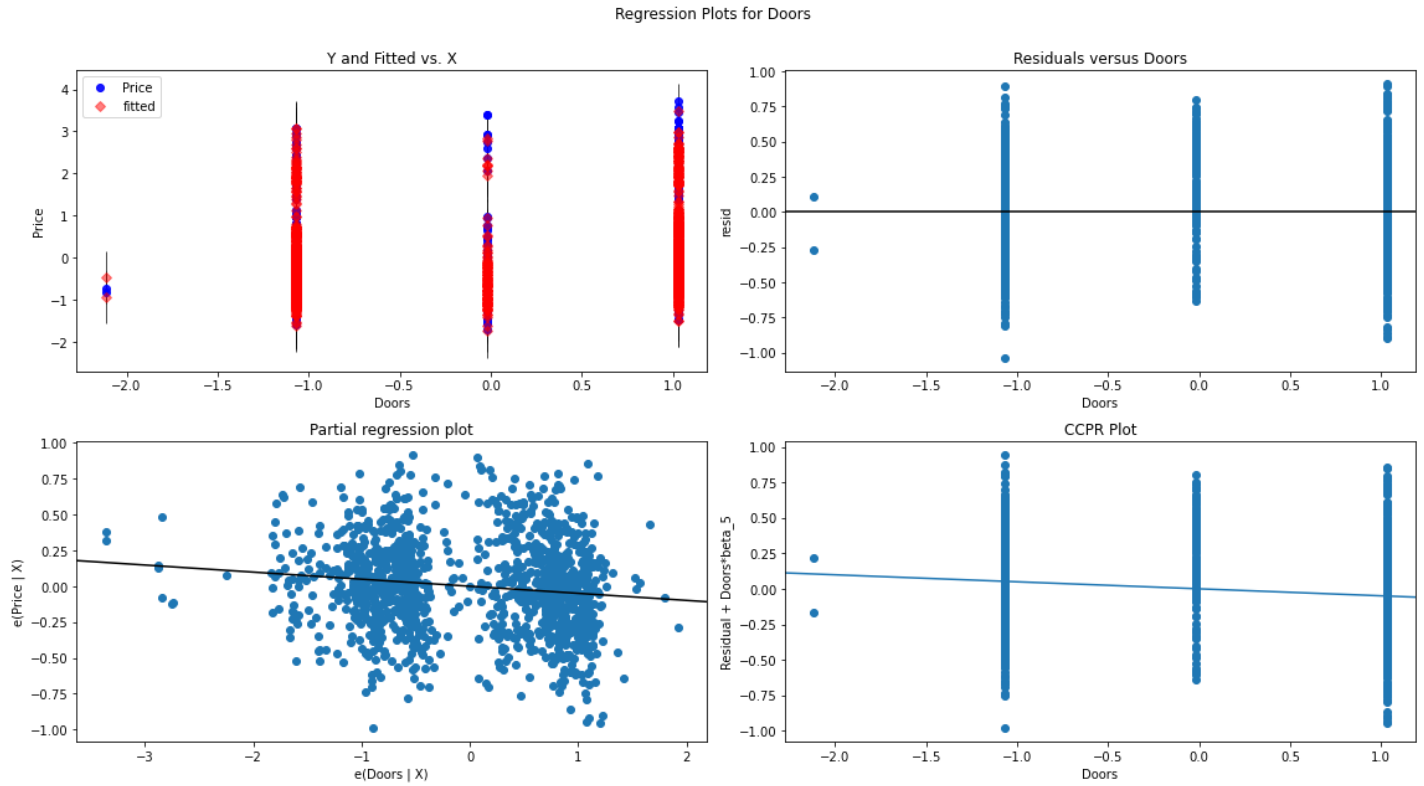
In [73]:

```
fig = plt.figure(figsize = (16,9))
sm.graphics.plot_regress_exog(Final_model, 'CC', fig=fig)
plt.show()
```



In [74]:

```
fig = plt.figure(figsize = (16,9))
sm.graphics.plot_regress_exog(Final_model, 'Doors', fig=fig)
plt.show()
```



^Observation:

- **Some of the feature doesn't suggest linear relationship with the Dependent feature like Gears, QT, CC, Doors, Weight and HP**
- **Only KM and Age is having a linear relation with the Price Feature**

Predicting values from Model using same dataset

In [118…

```
x = dataframe[['Age','KM','HP','CC','Doors','QT','Weight']]
y = dataframe[['Price']]
transformer_x = StandardScaler().fit(x)
transformer_y = StandardScaler().fit(y)
# Scale the test dataset
x_train_scal = transformer_x.transform(x)
y_train_scal = transformer_y.transform(y)

# Linear Regression
x_df = pd.DataFrame(x_train_scal, columns = ['Age','KM','HP','CC','Doors','QT','Weight'])
x_df.head()

# Predict with the trained model
predict = pd.DataFrame(Final_model.predict(x_df))

# Inverse transform the prediction
predict_unscaled = transformer_y.inverse_transform(predict.values.reshape(-1,1))
predict_unscaled
```

Out[118…

```
array([[15373.39486456],
       [15431.54051364],
       [15329.9011379 ],
       ...,
       [ 8534.98068775],
       [ 8673.66142795],
       [ 8783.42022356]])
```

In [117…

```
(np.sqrt(mean_squared_error(y, predict_unscaled)))
```

Out[117…

```
973.4891302577998
```

In [131…

```
predicted = pd.DataFrame(predict_unscaled,columns=['Predicted_Price'])
predicted['Price'] = dataframe.Price
predicted['Age'] = dataframe.Age
predicted['KM'] = dataframe.KM
predicted['Weight'] = dataframe.Weight
predicted['HP'] = dataframe.HP
predicted['CC'] = dataframe.CC
predicted['QT'] = dataframe.QT
predicted['Doors'] = dataframe.Doors
predicted
```

Out[131…

	Predicted_Price	Price	Age	KM	Weight	HP	CC	QT	Doors
0	15373.394865	13750	23	72937	1165	90	2000	210	3
1	15431.540514	14950	26	48000	1165	90	2000	210	3
2	15329.901138	13750	30	38500	1170	90	2000	210	3
3	14768.048428	12950	32	61000	1170	90	2000	210	3
4	17574.438664	16900	27	94612	1245	90	2000	210	3
...	...	...	...	...	...	...	...	...	...
1325	7607.786001	8450	80	23000	1015	86	1300	69	3
1326	9206.784055	7500	69	20544	1025	86	1300	69	3
1327	8534.980688	10845	72	19000	1015	86	1300	69	3
1328	8673.661428	8500	71	17016	1015	86	1300	69	3
1329	8783.420224	7250	70	16916	1015	86	1300	69	3

1330 rows × 9 columns

Preparing a table containing R^2 value for each prepared model

In [146…

```
models={'Different_Models':['Raw_data_Model','After_Removing_Influencers','After_Log Trans
        'R_squared':[raw_data_model.rsquared,final_model.rsquared,log_transformed_model.rs
        'R_squared_adjusted':[raw_data_model.rsquared_adj,final_model.rsquared_adj,log_tra
       'RMSE':[np.sqrt(raw_data_model.mse_resid),np.sqrt(final_model.mse_resid),log_both_r
model_table=pd.DataFrame(models)
model_table
```

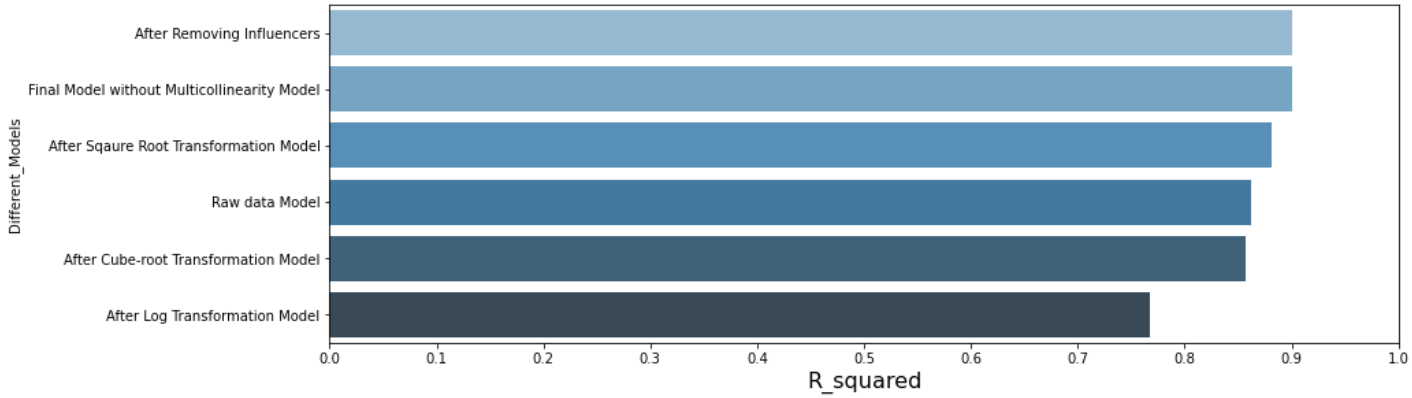
Out[146…

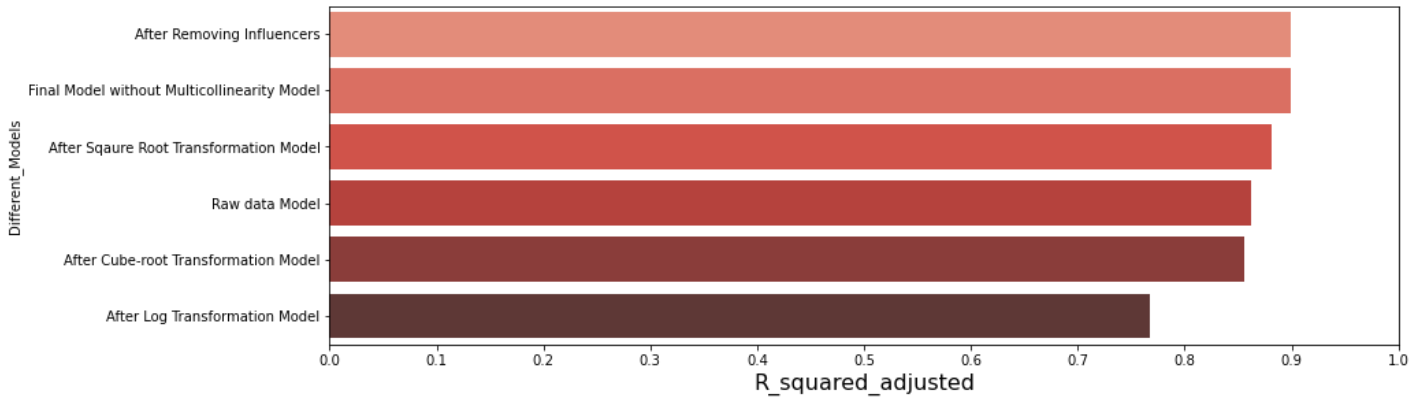
	Different_Models	R_squared	R_squared_adjusted	RMSE
0	Raw_data_Model	0.862520	0.861749	1341.804619
1	After_Removing_Influencers	0.900376	0.899773	975.927140
2	After_Log Transformation_Model	0.767907	0.766855	1795.826559
3	After_Cube-root_Transformation_Model	0.857248	0.856384	1152.557795
4	After_Sqaure_Root_Transformation_Model	0.881555	0.880837	1010.897945
5	Final_Model_without_Multicollinearity_Model	0.900198	0.899670	973.489130

Visualizing Models Performance

In [133…

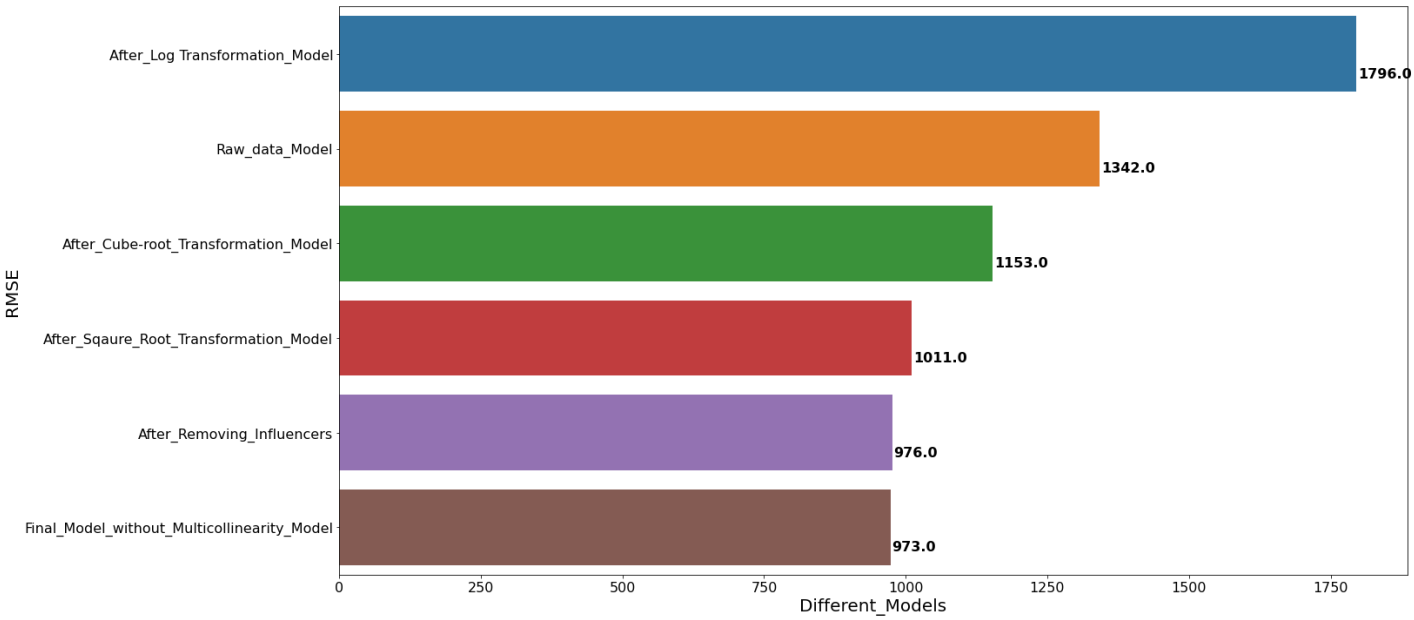
```
f, axes = plt.subplots(2,1, figsize=(14,10))

model_table.sort_values(by=['R_squared'], ascending=False, inplace=True)

sns.barplot(x='R_squared', y='Different_Models', data = model_table, palette='Blues_d', ax
axes[0].set_xlabel('R_squared', size=16)
axes[0].set_ylabel('Different_Models')
axes[0].set_xlim(0,1.0)
axes[0].set_xticks(np.arange(0, 1.1, 0.1))

model_table.sort_values(by=['R_squared_adjusted'], ascending=False, inplace=True)

sns.barplot(x='R_squared_adjusted', y='Different_Models', data = model_table, palette='Red
axes[1].set_xlabel('R_squared_adjusted', size=16)
axes[1].set_ylabel('Different_Models')
axes[1].set_xlim(0,1.0)
axes[1].set_xticks(np.arange(0, 1.1, 0.1))

plt.show()
```





In [155…

```
model_table.sort_values(by=['RMSE'], ascending=False, inplace=True)

f, axe = plt.subplots(1,1, figsize=(22,12))
sns.barplot(x='RMSE', y='Different_Models', data=model_table, ax = axe)
axe.set_xlabel('Different_Models', size=20)
axe.set_ylabel('RMSE', size=20)
plt.xticks(fontsize = 16)
plt.yticks(fontsize = 16)



for i, v in enumerate(np.round(model_table.RMSE.values,0)):
    axe.text(v + 3, i + .25, str(v),
             color = 'black', fontweight = 'bold', fontsize= 16)

plt.show()
```



In [75]:
```
from PIL import ImageGrab
ImageGrab.grabclipboard()
```

Out[75]:

In [135…
```
x_train.shape
```

Out[135…
```
(997, 7)
```

In [137…
```
#Linear Regression
reg_model = LinearRegression().fit(x_train, y_train)
print(reg_model.score(x_train, y_train),reg_model.score(x_test, y_test))
```

```
0.9046035168687026 0.8839205912023702
```

In [138…
```
from sklearn.ensemble import GradientBoostingRegressor
gd_model = GradientBoostingRegressor(random_state=1).fit(x_train, y_train)
print(gd_model.score(x_train, y_train),gd_model.score(x_test, y_test))
```

```
0.9436117543530305 0.9110866785403264
```

In [139…
```
from sklearn.ensemble import RandomForestRegressor
rfr_model = RandomForestRegressor(random_state=1).fit(x_train, y_train)
print(rfr_model.score(x_train, y_train),rfr_model.score(x_test, y_test))
```

```
0.9863399164905412 0.9008965073289248
```

In [140…
```
print('Linear Regression Root Mean Squared Error:',np.sqrt(mean_squared_error(y_test, reg_
print('Gradient Booster Regressor Root Mean Squared Error:',np.sqrt(mean_squared_error(y_t
print('Random Forest Regressor Root Mean Squared Error:',np.sqrt(mean_squared_error(y_test
```

```
Linear Regression Root Mean Squared Error: 986.2914604904846
Gradient Booster Regressor Root Mean Squared Error: 863.1993332586752
Random Forest Regressor Root Mean Squared Error: 911.3226395734384
```

In [ ]: