Assignment-05-Multiple Linear Regression-1

In [1]:

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

In [2]:

```
# import dataset
toyo=pd.read_csv('ToyotaCorolla.csv',encoding='latin1')
toyo
```

Out[2]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	10	2002	46986	Diesel	90	1	...	
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	10	2002	72937	Diesel	90	1	...	
2	3	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	9	2002	41711	Diesel	90	1	...	
3	4	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	14950	26	7	2002	48000	Diesel	90	0	...	
4	5	TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors	13750	30	3	2002	38500	Diesel	90	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
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	...	

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central
1435	1442	TOYOTA Corolla 1.6 LB LINEA TERRA 4/5-Doors	6950	76	5	1998	1	Petrol	110	0	...	

1436 rows × 38 columns

EDA

In [3]:
```
toyo.info()
```

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

Loading [MathJax]/extensions/Safe.js

In [4]:
```
toyo2=pd.concat([toyo.iloc[:,2:4],toyo.iloc[:,6:7],toyo.iloc[:,8:9],toyo.iloc[:,12:14],t
toyo2
```

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
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 [5]:
```
toyo3=toyo2.rename({'Age_08_04':'Age','cc':'CC','Quarterly_Tax':'QT'},axis=1)
toyo3
```

Out[5]:

	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 [6]:
```
toyo3[toyo3.duplicated()]
```

Out[6]:

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

In [7]:
```
toyo4=toyo3.drop_duplicates().reset_index(drop=True)
toyo4
```

Out[7]:

	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 [8]:

```
toyo4.describe()
```

Out[8]:

	Price	Age	KM	HP	CC	Doors	Gears	
count	1435.000000	1435.000000	1435.000000	1435.000000	1435.000000	1435.000000	1435.000000	1435.0
mean	10720.915679	55.980488	68571.782578	101.491986	1576.560976	4.032753	5.026481	87.0
std	3608.732978	18.563312	37491.094553	14.981408	424.387533	0.952667	0.188575	40.9
min	4350.000000	1.000000	1.000000	69.000000	1300.000000	2.000000	3.000000	19.0
25%	8450.000000	44.000000	43000.000000	90.000000	1400.000000	3.000000	5.000000	69.0
50%	9900.000000	61.000000	63451.000000	110.000000	1600.000000	4.000000	5.000000	85.0
75%	11950.000000	70.000000	87041.500000	110.000000	1600.000000	5.000000	5.000000	85.0
max	32500.000000	80.000000	243000.000000	192.000000	16000.000000	5.000000	6.000000	283.0

Correlation Analysis

In [9]:

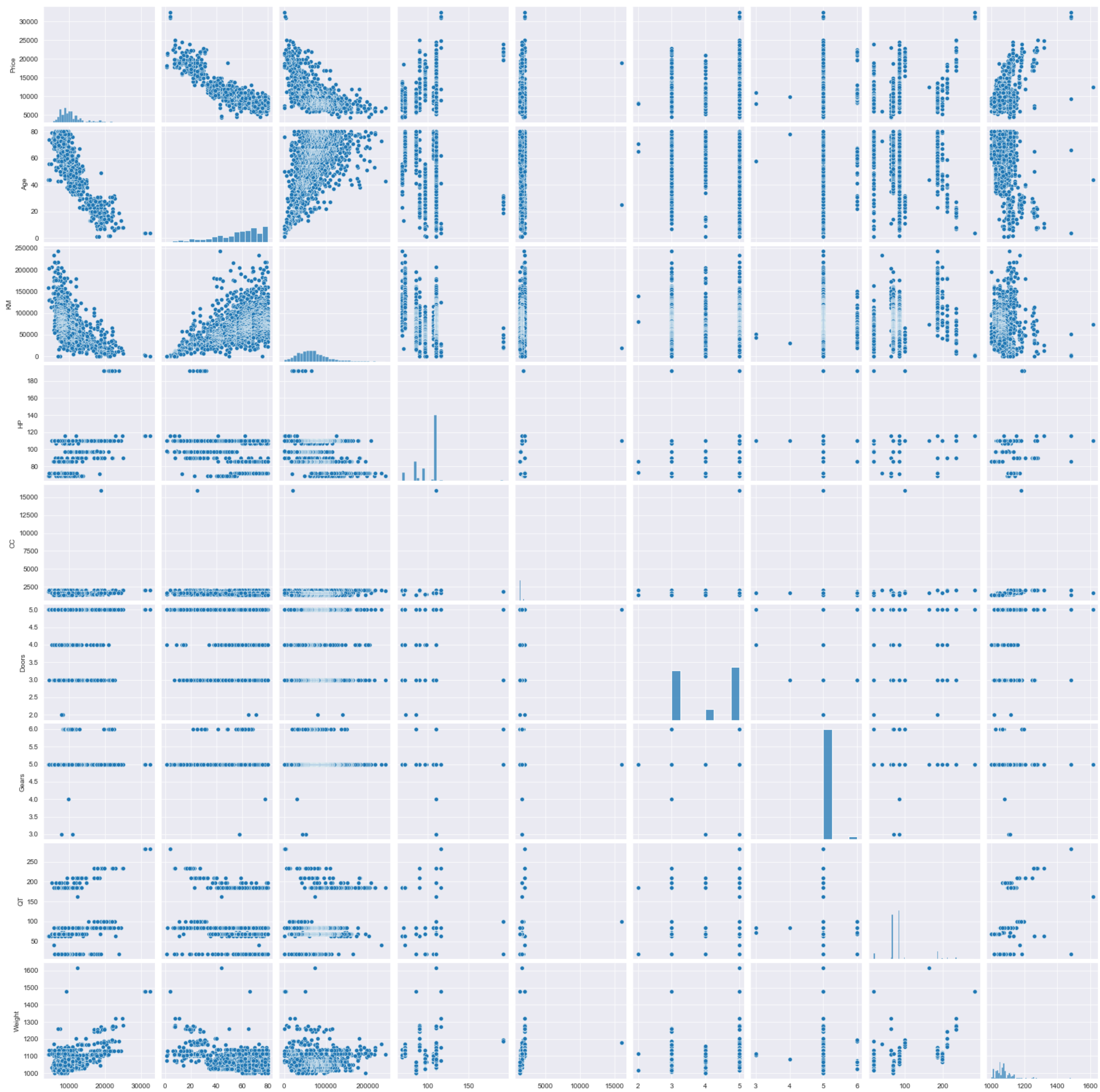
```
toyo4.corr()
```

Out[9]:

	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 [10]:
```
sns.set_style(style='darkgrid')
sns.pairplot(toyo4)
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x219687fa310>



Model Building

In [11]:
```
model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyo4).fit()
```

Model Testing

In [12]:
```
# Finding Coefficient parameters
model.params
```

Out[12]:
```
Intercept   -5472.540368
Age          -121.713891
KM             -0.020737
HP             31.584612
CC             -0.118558
Doors          -0.920189
Gears         597.715894
QT              3.858805
Weight         16.855470
dtype: float64
```

In [13]:
```
# Finding tvalues and pvalues
model.tvalues , np.round(model.pvalues,5)
```

Out[13]:
```
(Intercept    -3.875273
 Age         -46.551876
 KM          -16.552424
 HP           11.209719
 CC           -1.316436
 Doors        -0.023012
 Gears         3.034563
 QT            2.944198
 Weight       15.760663
 dtype: float64,
 Intercept    0.00011
 Age          0.00000
 KM           0.00000
 HP           0.00000
 CC           0.18824
 Doors        0.98164
 Gears        0.00245
 QT           0.00329
 Weight       0.00000
 dtype: float64)
```

In [14]:
```
# Finding rsquared values
model.rsquared , model.rsquared_adj   # Model accuracy is 86.17%
```

Out[14]:
```
(0.8625200256947, 0.8617487495415146)
```

In [15]:
```
# Build SLR and MLR models for insignificant variables 'CC' and 'Doors'
# Also find their tvalues and pvalues
```

In [16]:
```
slr_c=smf.ols('Price~CC',data=toyo4).fit()
slr_c.tvalues , slr_c.pvalues # CC has significant pvalue
```

Out[16]:
```
(Intercept    24.879592
 CC            4.745039
 dtype: float64,
 Intercept    7.236022e-114
 CC            2.292856e-06
 dtype: float64)
```

In [17]:
```
slr_d=smf.ols('Price~Doors',data=toyo4).fit()
slr_d.tvalues , slr_d.pvalues # Doors has significant pvalue
```

Out[17]:
```
(Intercept    19.421546
 Doors         7.070520
 dtype: float64,
 Intercept    8.976407e-75
 Doors        2.404166e-12
 dtype: float64)
```

In [18]:
```
mlr_cd=smf.ols('Price~CC+Doors',data=toyo4).fit()
```

Loading [MathJax]/extensions/Safe.js

```
mlr_cd.tvalues , mlr_cd.pvalues # CC & Doors have significant pvalue
```

Out[18]:
```
(Intercept    12.786341
 CC            4.268006
 Doors         6.752236
 dtype: float64,
 Intercept    1.580945e-35
 CC           2.101878e-05
 Doors        2.109558e-11
 dtype: float64)
```

Model Validation Techniques

Two Techniques: 1. Collinearity Check & 2. Residual Analysis

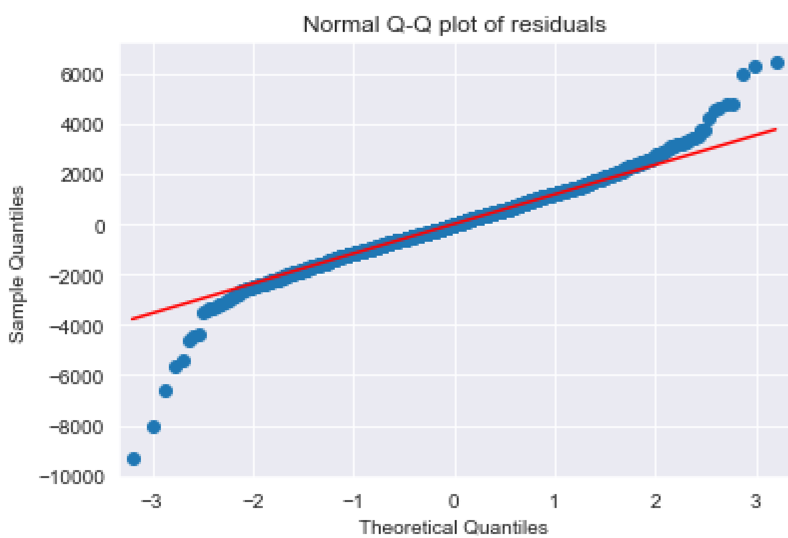
In [19]:
```
# 1) Collinearity Problem Check
# Calculate VIF = 1/(1-Rsquare) for all independent variables

rsq_age=smf.ols('Age~KM+HP+CC+Doors+Gears+QT+Weight',data=toyo4).fit().rsquared
vif_age=1/(1-rsq_age)

rsq_KM=smf.ols('KM~Age+HP+CC+Doors+Gears+QT+Weight',data=toyo4).fit().rsquared
vif_KM=1/(1-rsq_KM)

rsq_HP=smf.ols('HP~Age+KM+CC+Doors+Gears+QT+Weight',data=toyo4).fit().rsquared
vif_HP=1/(1-rsq_HP)

rsq_CC=smf.ols('CC~Age+KM+HP+Doors+Gears+QT+Weight',data=toyo4).fit().rsquared
vif_CC=1/(1-rsq_CC)

rsq_DR=smf.ols('Doors~Age+KM+HP+CC+Gears+QT+Weight',data=toyo4).fit().rsquared
vif_DR=1/(1-rsq_DR)

rsq_GR=smf.ols('Gears~Age+KM+HP+CC+Doors+QT+Weight',data=toyo4).fit().rsquared
vif_GR=1/(1-rsq_GR)

rsq_QT=smf.ols('QT~Age+KM+HP+CC+Doors+Gears+Weight',data=toyo4).fit().rsquared
vif_QT=1/(1-rsq_QT)

rsq_WT=smf.ols('Weight~Age+KM+HP+CC+Doors+Gears+QT',data=toyo4).fit().rsquared
vif_WT=1/(1-rsq_WT)

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

Out[19]:

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

In [20]:
```
# None variable has VIF>20, No Collinearity, so consider all varaibles in Regression equ
```

In [21]:
```
# 2) Residual Analysis
# Test for Normality of Residuals (Q-Q Plot) using residual model (model.resid)
sm.qqplot(model.resid,line='q') # 'q' - A line is fit through the quartiles # line = '45
plt.title("Normal Q-Q plot of residuals")
plt.show()
```



In [22]:
```
list(np.where(model.resid>6000))  # outliar detection from above QQ plot of residuals
```

Out[22]:
```
[array([109, 146, 522], dtype=int64)]
```

In [23]:
```
list(np.where(model.resid<-6000))
```

Out[23]:
```
[array([220, 600, 959], dtype=int64)]
```

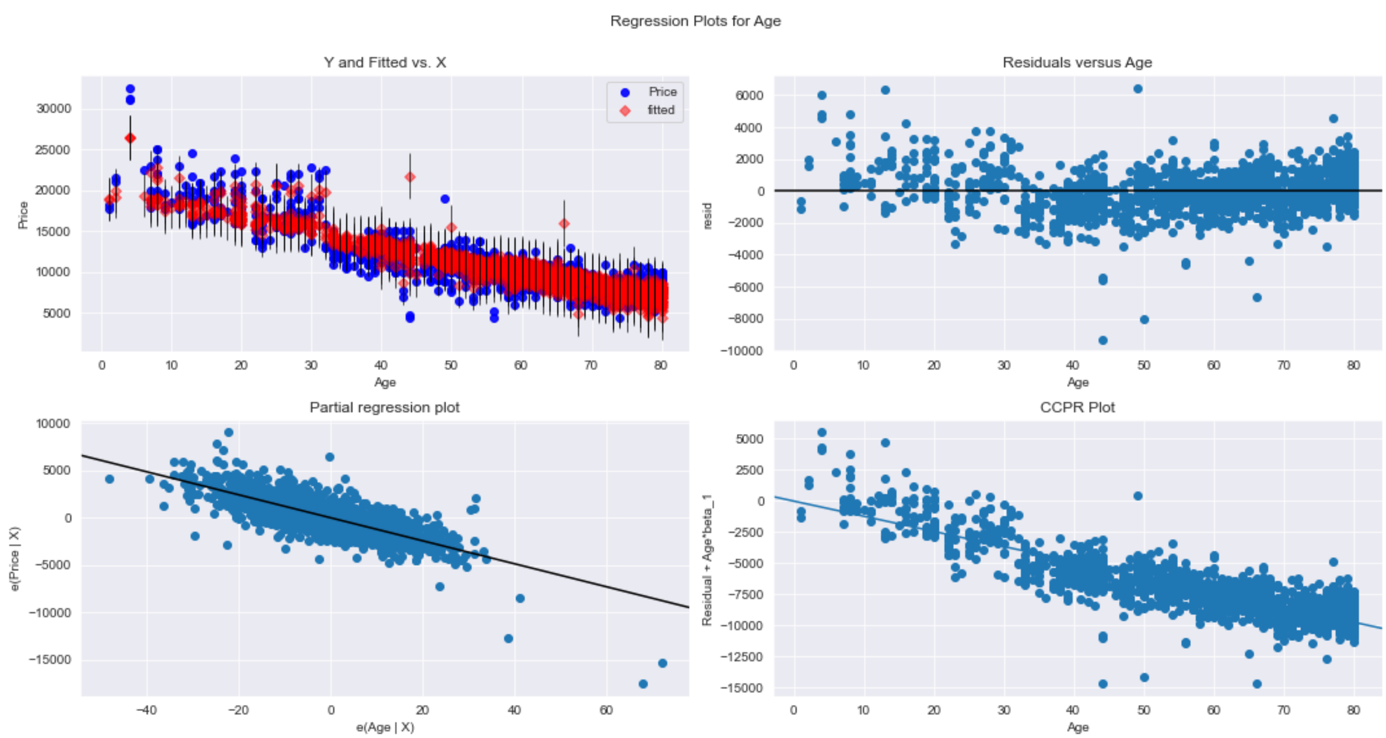
In [24]:
```
# Test for Homoscedasticity or Heteroscedasticity (plotting model's standardized fitted

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

In [25]:
```
plt.scatter(standard_values(model.fittedvalues),standard_values(model.resid))
plt.title('Residual Plot')
plt.xlabel('standardized fitted values')
plt.ylabel('standardized residual values')
plt.show()
```

In [26]:
```
# Test for errors or Residuals Vs Regressors or independent 'x' variables or predictors
# using Residual Regression Plots code graphics.plot_regress_exog(model,'x',fig)    # ex
```

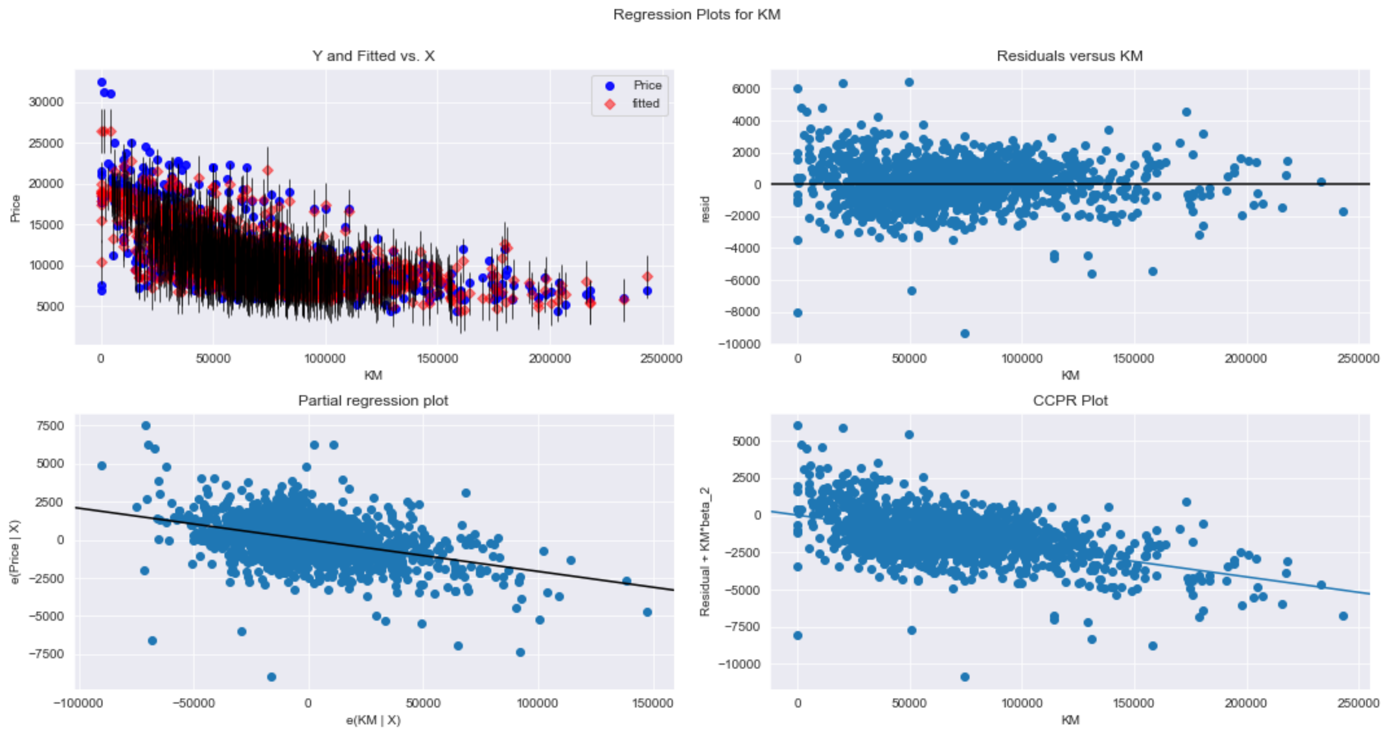
In [27]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Age',fig=fig)
plt.show()
```

```
eval_env: 1
```



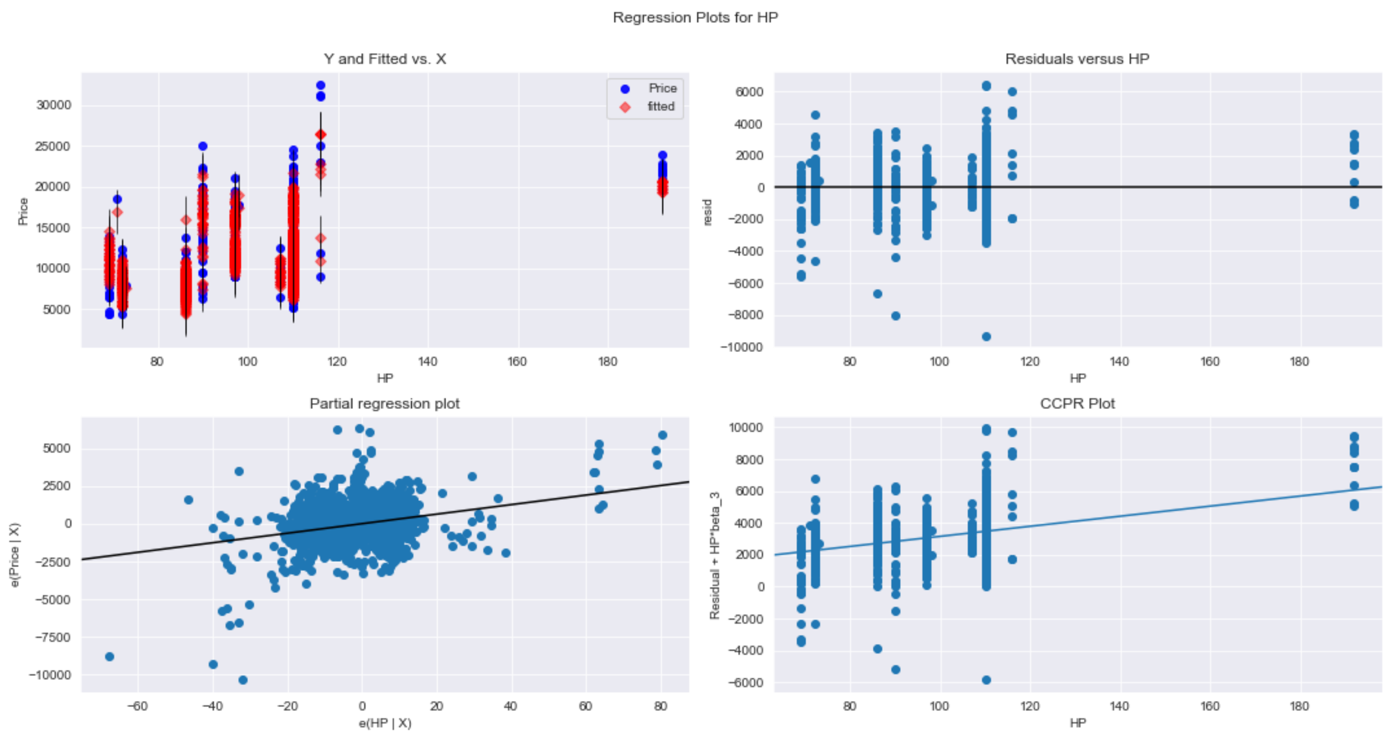
In [28]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'KM',fig=fig)
plt.show()
```

```
eval_env: 1
```



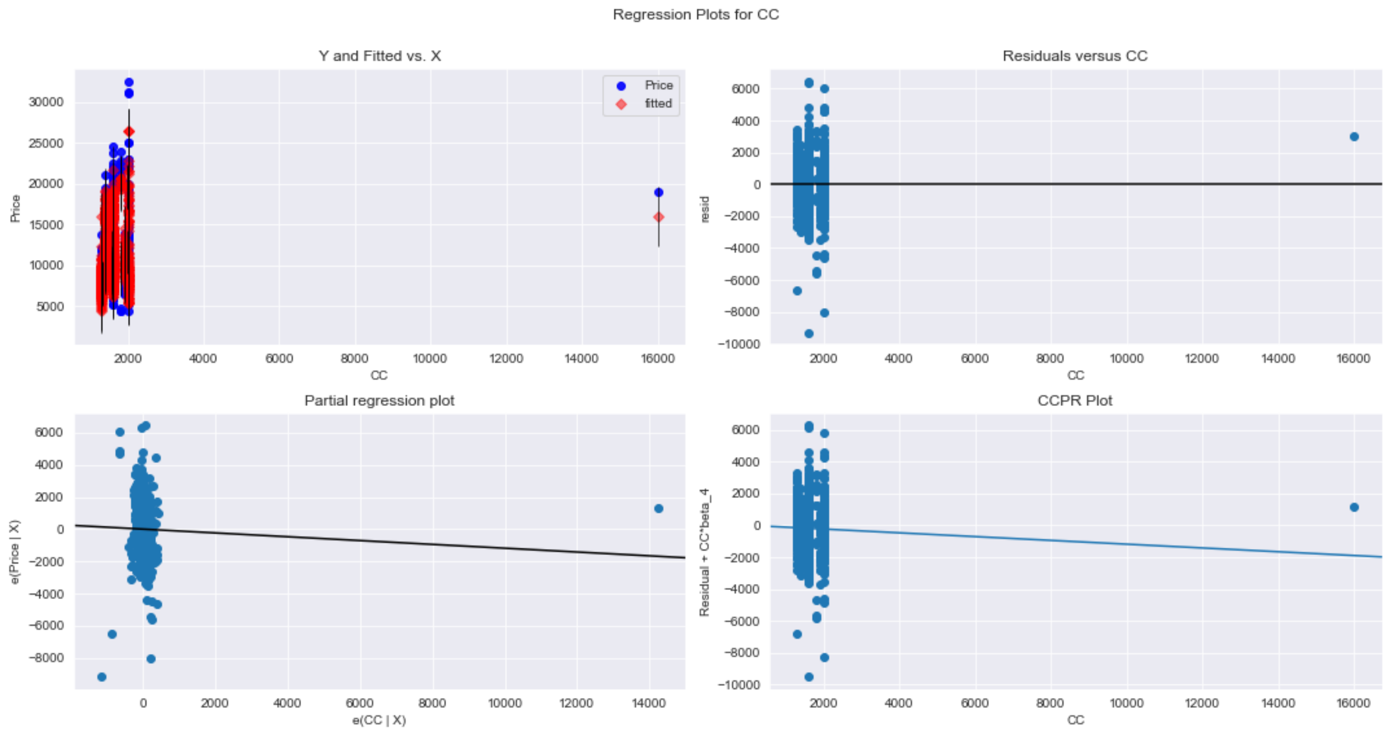
In [29]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'HP',fig=fig)
plt.show()
```

```
eval_env: 1
```



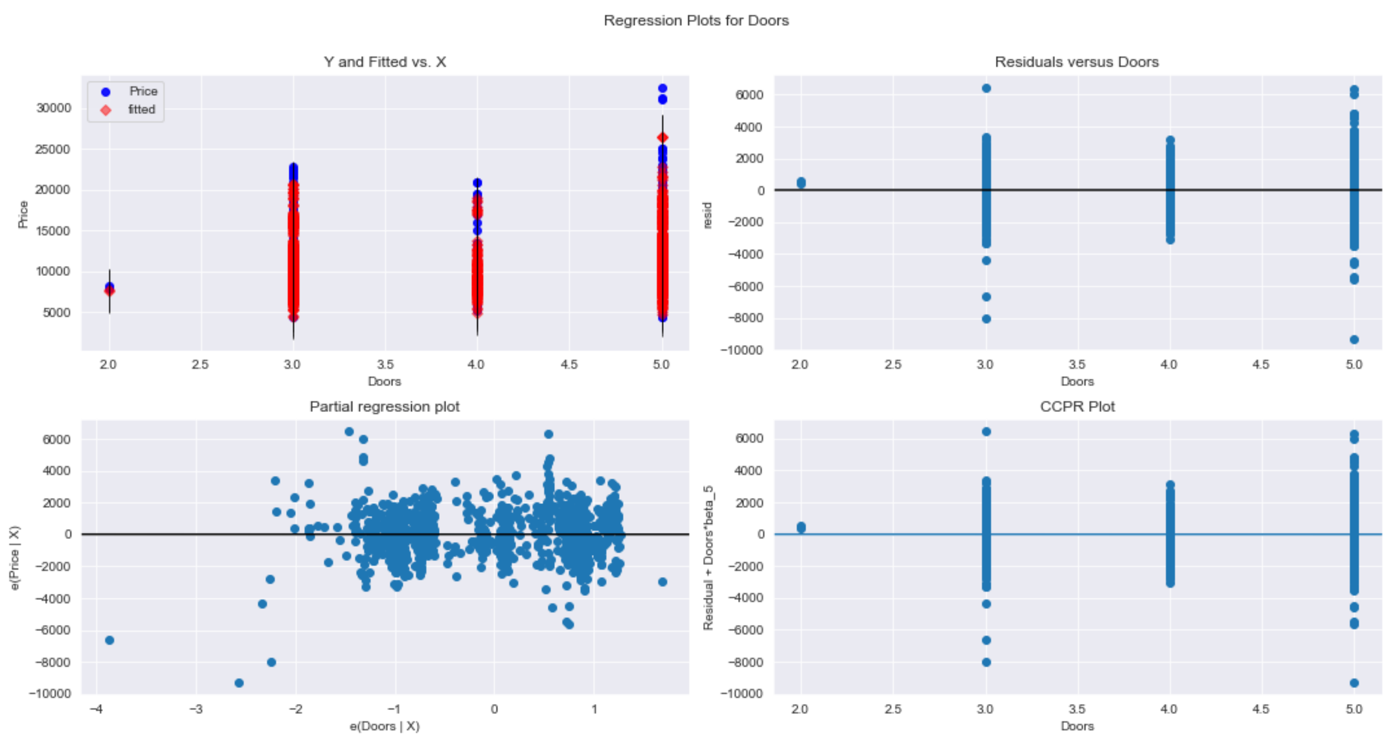
In [30]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'CC',fig=fig)
plt.show()
```

```
eval_env: 1
```



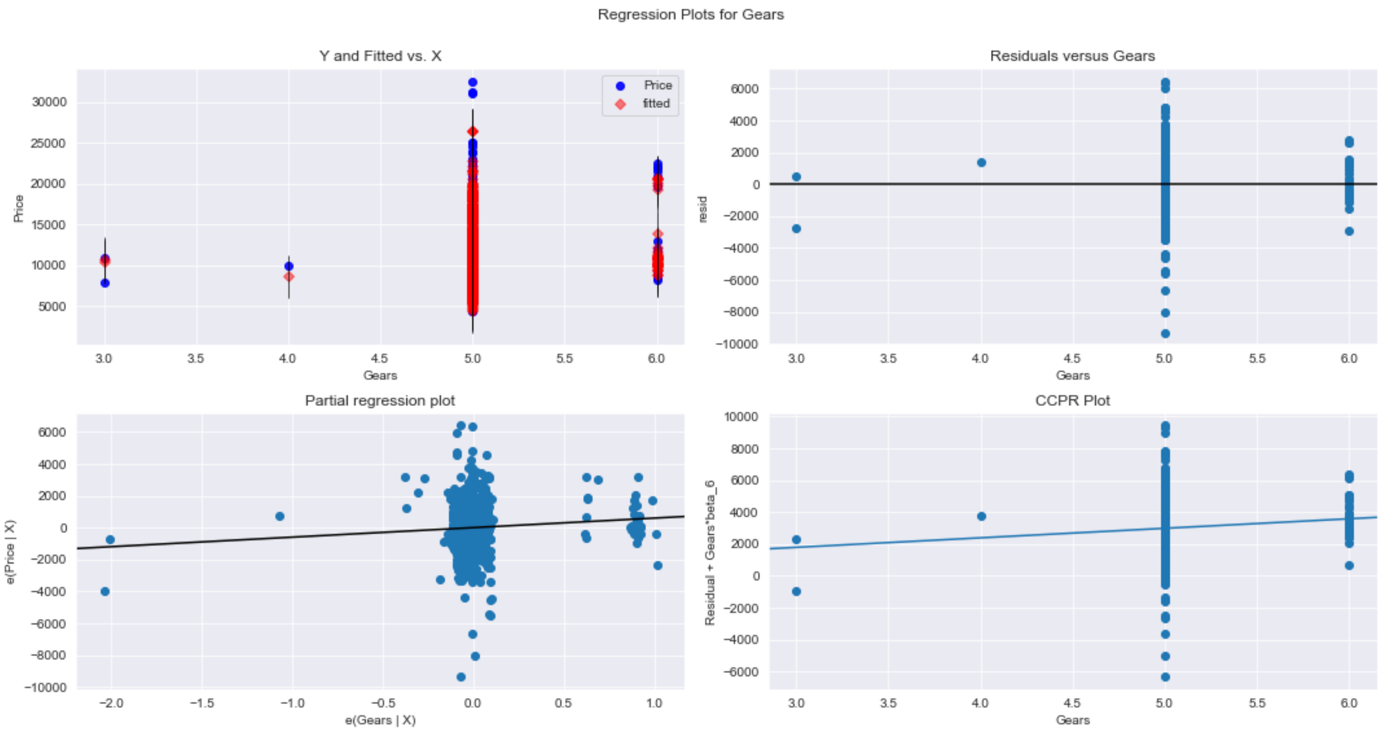
In [31]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Doors',fig=fig)
plt.show()
```

```
eval_env: 1
```



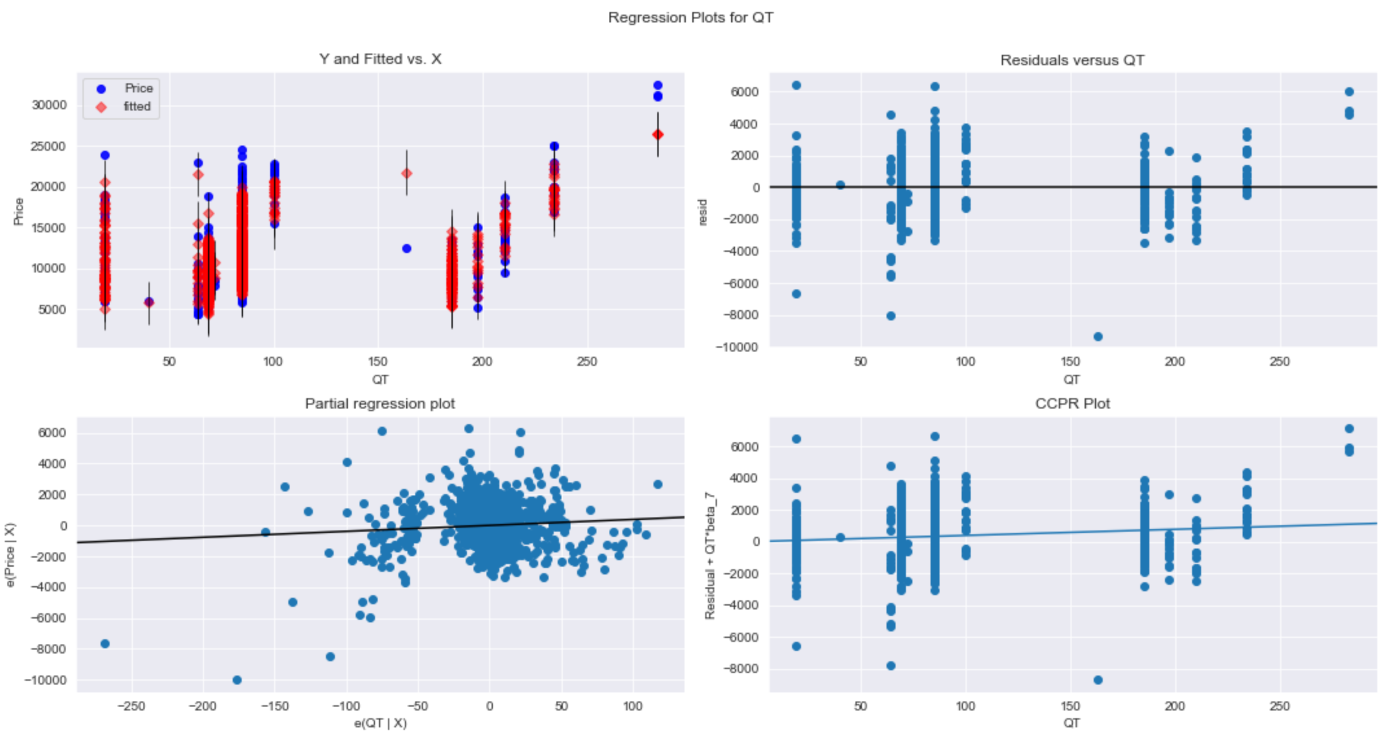
In [32]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Gears',fig=fig)
plt.show()
```

```
eval_env: 1
```



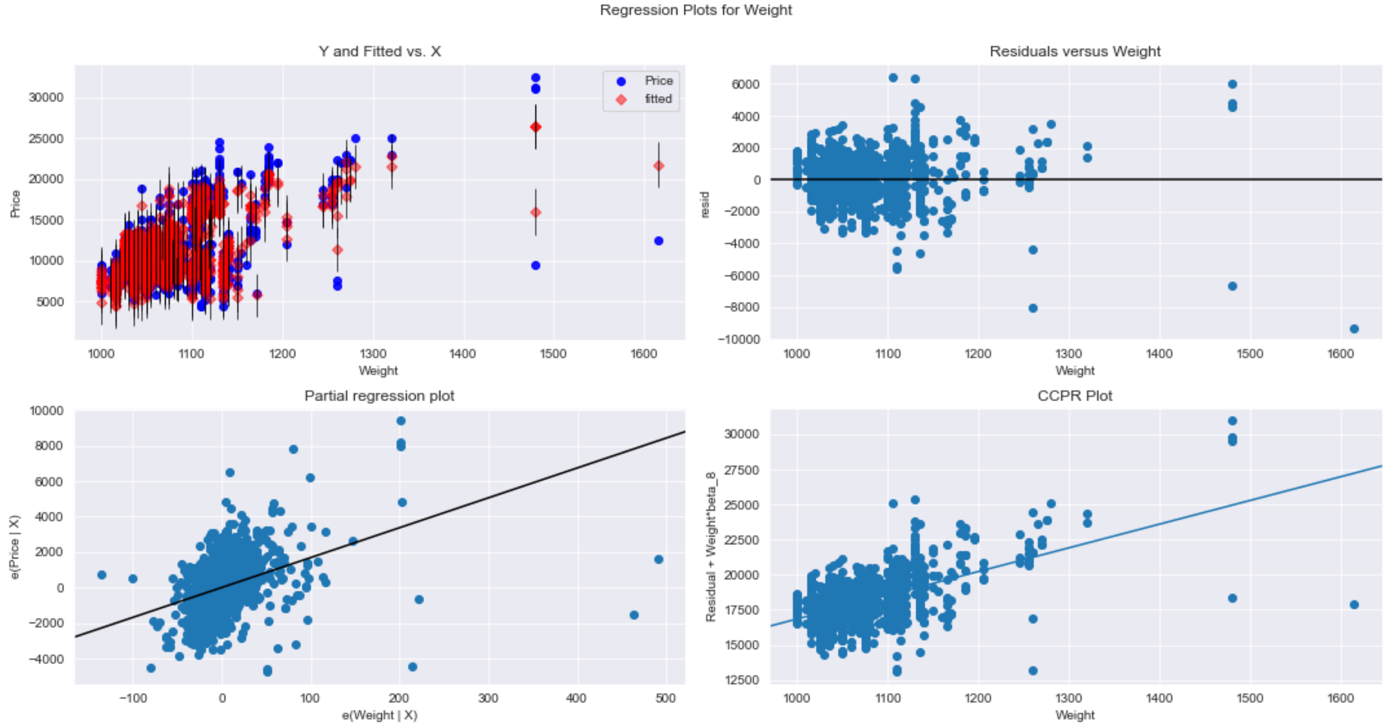
In [33]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'QT',fig=fig)
plt.show()
```

eval_env: 1



In [34]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'Weight',fig=fig)
plt.show()
```

eval_env: 1



Model Deletion Diagnostics (checking Outliers or Influencers)

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

In [35]:

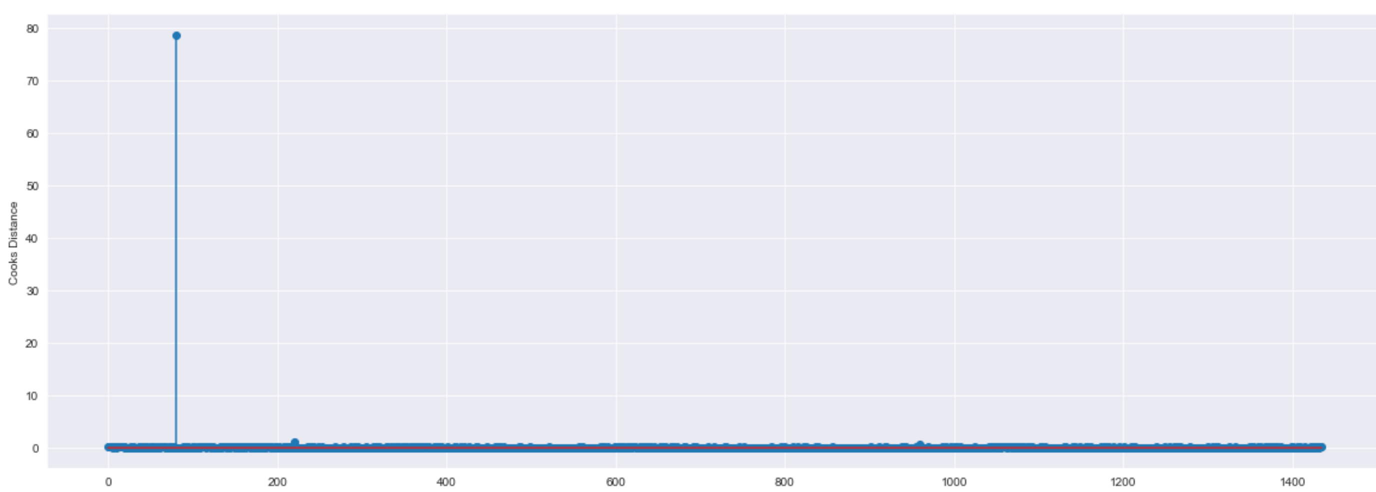
```
# 1. Cook's Distance: If Cook's distance > 1, then it's an outlier
# Get influencers using cook's distance
(c,_)=model.get_influence().cooks_distance
c
```

Out[35]:

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

In [36]:

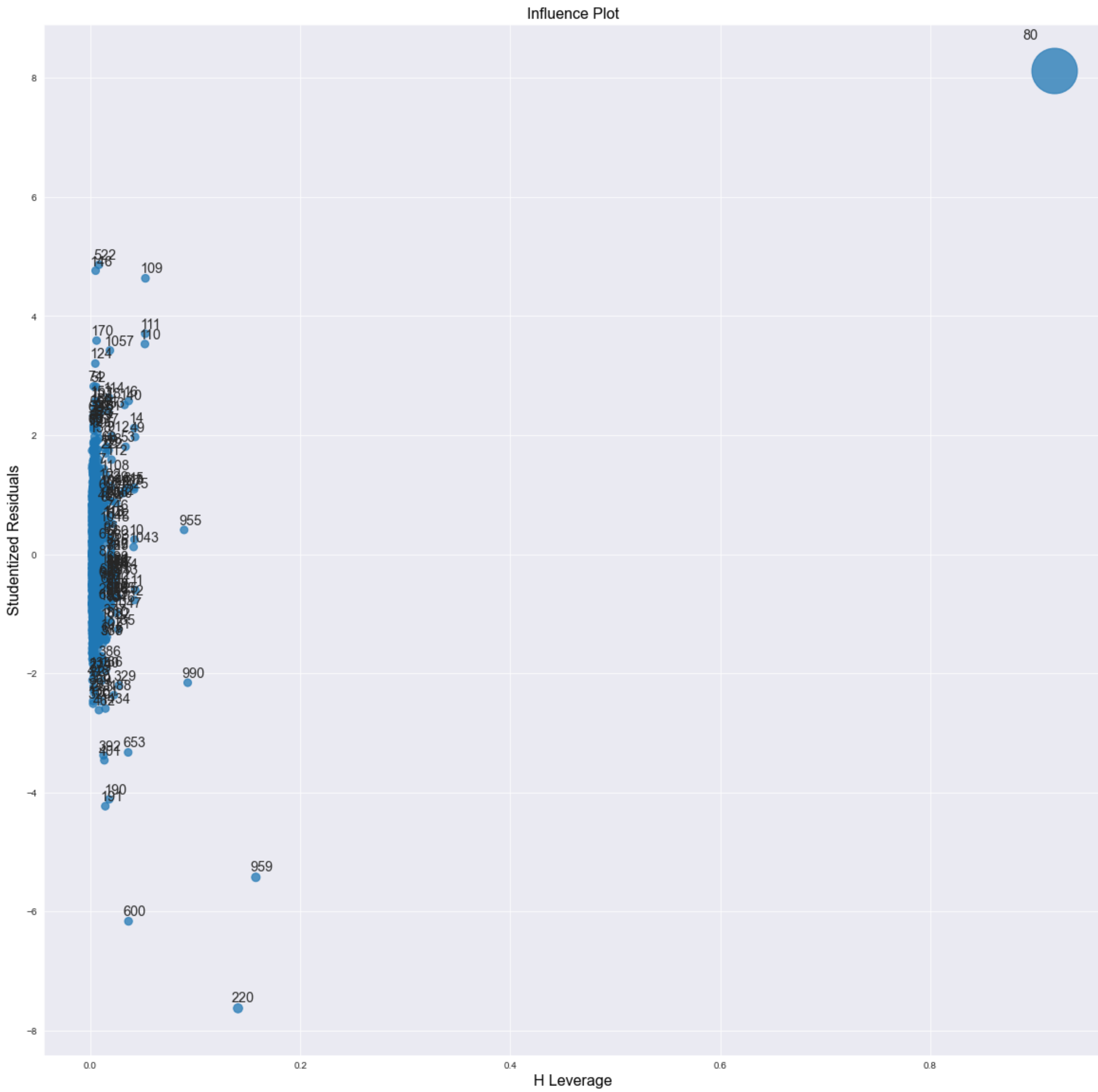
```
# Plot the influencers using the stem plot
fig=plt.figure(figsize=(20,7))
plt.stem(np.arange(len(toyo4)),np.round(c,3))
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```



In [37]:
```python
# Index and value of influencer where C>0.5
np.argmax(c) , np.max(c)
```

Out[37]: (80, 78.72950582248232)

In [38]:
```python
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value ar
fig,ax=plt.subplots(figsize=(20,20))
fig=influence_plot(model,ax = ax)
```



In [39]:
```python
# Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of datapoint
k=toyo4.shape[1]
n=toyo4.shape[0]
leverage_cutoff = (3*(k+1))/n
leverage_cutoff
```

Out[39]: 0.020905923344947737

In [40]:
```python
toyo4[toyo4.index.isin([80])]
```

Out[40]:

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

Improving the Model

In [41]:

```
# Creating a copy of data so that original dataset is not affected
toyo_new=toyo4.copy()
toyo_new
```

Out[41]:

	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 [42]:

```
# Discard the data points which are influencers and reassign the row number (reset_index
toyo5=toyo_new.drop(toyo_new.index[[80]],axis=0).reset_index(drop=True)
toyo5
```

Out[42]:

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

1434 rows × 9 columns

Model Deletion Diagnostics and Final Model

In [43]:
```
 while np.max(c)>0.5 :
    model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyo5).fit()
    (c,_)=model.get_influence().cooks_distance
    c
    np.argmax(c) , np.max(c)
    toyo5=toyo5.drop(toyo5.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
    toyo5
else:
    final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyo5).fit()
    final_model.rsquared , final_model.aic
    print("Thus model accuracy is improved to",final_model.rsquared)
```

```
Thus model accuracy is improved to 0.8882395145171204
```

In [44]:
```
if np.max(c)>0.5:
    model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyo5).fit()
    (c,_)=model.get_influence().cooks_distance
    c
    np.argmax(c) , np.max(c)
    toyo5=toyo5.drop(toyo5.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
    toyo5
elif np.max(c)<0.5:
    final_model=smf.ols('Price~Age+KM+HP+CC+Doors+Gears+QT+Weight',data=toyo5).fit()
    final_model.rsquared , final_model.aic
    print("Thus model accuracy is improved to",final_model.rsquared)
```

```
Thus model accuracy is improved to 0.8882395145171204
```

In [45]:
```
final_model.rsquared
```

Out[45]:
```
0.8882395145171204
```

In [46]:
```
toyo5
```

Out[46]:

	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
...	...	...	...	...	...	...	...	...	...
1426	7500	69	20544	86	1300	3	5	69	1025
1427	10845	72	19000	86	1300	3	5	69	1015
1428	8500	71	17016	86	1300	3	5	69	1015
1429	7250	70	16916	86	1300	3	5	69	1015
1430	6950	76	1	110	1600	5	5	19	1114

1431 rows × 9 columns

Model Predictions

In [47]:
```
# say New data for prediction is
new_data=pd.DataFrame({'Age':12,"KM":40000,"HP":80,"CC":1300,"Doors":4,"Gears":5,"QT":69
new_data
```

Out[47]:

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

In [48]:
```
# Manual Prediction of Price
final_model.predict(new_data)
```

Out[48]:
```
0    14341.570181
dtype: float64
```

In [49]:
```
# Automatic Prediction of Price with 90.02% accurcy
pred_y=final_model.predict(toyo5)
pred_y
```

Out[49]:
```
0       16345.352610
1       15886.635544
2       16328.224968
3       15996.318854
4       15883.424182
            ...
1426     9161.230587
1427     8536.091326
1428     8681.531063
1429     8793.668694
1430    10860.695492
Length: 1431, dtype: float64
```

Assignment-05-Multiple Linear Regression-2

In [50]:
```
# import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
from statsmodels.graphics.regressionplots import influence_plot
```

In [51]:
```
# import dataset
data=pd.read_csv("50_Startups.csv")
data
```

Out[51]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04
25	64664.71	139553.16	137962.62	California	107404.34
26	75328.87	144135.98	134050.07	Florida	105733.54
27	72107.60	127864.55	353183.81	New York	105008.31
28	66051.52	182645.56	118148.20	Florida	103282.38
29	65605.48	153032.06	107138.38	New York	101004.64
30	61994.48	115641.28	91131.24	Florida	99937.59
31	61136.38	152701.92	88218.23	New York	97483.56
32	63408.86	129219.61	46085.25	California	97427.84
33	55493.95	103057.49	214634.81	Florida	96778.92
34	46426.07	157693.92	210797.67	California	96712.80
35	46014.02	85047.44	205517.64	New York	96479.51
36	28663.76	127056.21	201126.82	Florida	90708.19
37	44069.95	51283.14	197029.42	California	89949.14
38	20229.59	65947.93	185265.10	New York	81229.06

Loading [MathJax]/extensions/Safe.js

	R&D Spend	Administration	Marketing Spend	State	Profit
39	38558.51	82982.09	174999.30	California	81005.76
40	28754.33	118546.05	172795.67	California	78239.91
41	27892.92	84710.77	164470.71	Florida	77798.83
42	23640.93	96189.63	148001.11	California	71498.49
43	15505.73	127382.30	35534.17	New York	69758.98
44	22177.74	154806.14	28334.72	California	65200.33
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

EDA

In [52]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object 
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [53]:

```
data1=data.rename({'R&D Spend':'RDS','Administration':'ADMS','Marketing Spend':'MKTS'},a
data1
```

Loading [MathJax]/extensions/Safe.js

Out[53]:

	RDS	ADMS	MKTS	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04
25	64664.71	139553.16	137962.62	California	107404.34
26	75328.87	144135.98	134050.07	Florida	105733.54
27	72107.60	127864.55	353183.81	New York	105008.31
28	66051.52	182645.56	118148.20	Florida	103282.38
29	65605.48	153032.06	107138.38	New York	101004.64
30	61994.48	115641.28	91131.24	Florida	99937.59
31	61136.38	152701.92	88218.23	New York	97483.56
32	63408.86	129219.61	46085.25	California	97427.84
33	55493.95	103057.49	214634.81	Florida	96778.92
34	46426.07	157693.92	210797.67	California	96712.80
35	46014.02	85047.44	205517.64	New York	96479.51
36	28663.76	127056.21	201126.82	Florida	90708.19
37	44069.95	51283.14	197029.42	California	89949.14
38	20229.59	65947.93	185265.10	New York	81229.06

	RDS	ADMS	MKTS	State	Profit
39	38558.51	82982.09	174999.30	California	81005.76
40	28754.33	118546.05	172795.67	California	78239.91
41	27892.92	84710.77	164470.71	Florida	77798.83
42	23640.93	96189.63	148001.11	California	71498.49
43	15505.73	127382.30	35534.17	New York	69758.98
44	22177.74	154806.14	28334.72	California	65200.33
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

In [54]:
```
data1[data1.duplicated()] # No duplicated data
```

Out[54]:

RDS	ADMS	MKTS	State	Profit

In [55]:
```
data1.describe()
```

Out[55]:

	RDS	ADMS	MKTS	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

Correlation Analysis

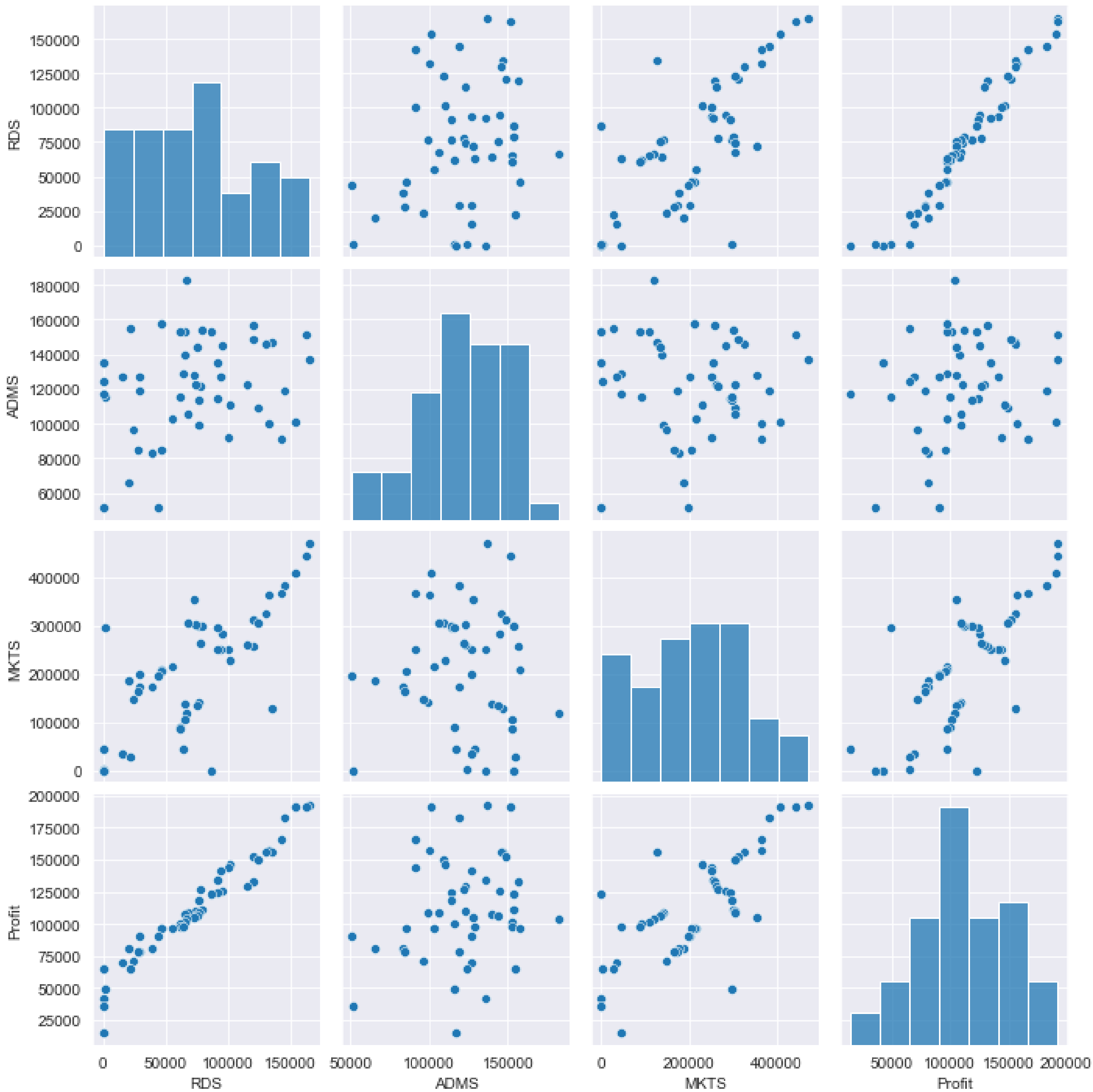
In [56]:
```
data1.corr()
```

Out[56]:

	RDS	ADMS	MKTS	Profit
RDS	1.000000	0.241955	0.724248	0.972900
ADMS	0.241955	1.000000	-0.032154	0.200717
MKTS	0.724248	-0.032154	1.000000	0.747766
Profit	0.972900	0.200717	0.747766	1.000000

In [57]:
```
sns.set_style(style='darkgrid')
sns.pairplot(data1)
```

Out[57]:
```
<seaborn.axisgrid.PairGrid at 0x219787f3070>
```



Model Building

In [58]:
```
model=smf.ols("Profit~RDS+ADMS+MKTS",data=data1).fit()
```

Model Testing

In [59]:
```
# Finding Coefficient parameters
model.params
```

Out[59]:
```
Intercept    50122.192990
RDS              0.805715
ADMS            -0.026816
MKTS             0.027228
dtype: float64
```

In [60]:
```
# Finding tvalues and pvalues
```

Loading [MathJax]/extensions/Safe.js

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

Out[60]:
```
(Intercept     7.626218
 RDS          17.846374
 ADMS         -0.525507
 MKTS          1.655077
 dtype: float64,
 Intercept    0.00000
 RDS          0.00000
 ADMS         0.60176
 MKTS         0.10472
 dtype: float64)
```

In [61]:
```
# Finding rsquared values
model.rsquared , model.rsquared_adj  # Model accuracy is 94.75%
```

Out[61]:
```
(0.9507459940683246, 0.9475337762901719)
```

In [62]:
```
# Build SLR and MLR models for insignificant variables 'ADMS' and 'MKTS'
# Also find their tvalues and pvalues
```

In [63]:
```
slr_a=smf.ols("Profit~ADMS",data=data1).fit()
slr_a.tvalues , slr_a.pvalues  # ADMS has in-significant pvalue
```

Out[63]:
```
(Intercept    3.040044
 ADMS         1.419493
 dtype: float64,
 Intercept    0.003824
 ADMS         0.162217
 dtype: float64)
```

In [64]:
```
slr_m=smf.ols("Profit~MKTS",data=data1).fit()
slr_m.tvalues , slr_m.pvalues  # MKTS has significant pvalue
```

Out[64]:
```
(Intercept    7.808356
 MKTS         7.802657
 dtype: float64,
 Intercept    4.294735e-10
 MKTS         4.381073e-10
 dtype: float64)
```

In [65]:
```
mlr_am=smf.ols("Profit~ADMS+MKTS",data=data1).fit()
mlr_am.tvalues , mlr_am.pvalues  # varaibles have significant pvalues
```

Out[65]:
```
(Intercept    1.142741
 ADMS         2.467779
 MKTS         8.281039
 dtype: float64,
 Intercept    2.589341e-01
 ADMS         1.729198e-02
 MKTS         9.727245e-11
 dtype: float64)
```

Model Validation

Two Techniques: 1. Collinearity Check & 2. Residual Analysis

In [66]:
```
# 1) Collinearity Problem Check
# Calculate VIF = 1/(1-Rsquare) for all independent variables

 ... s("RDS~ADMS+MKTS",data=data1).fit().rsquared
```

```
vif_r=1/(1-rsq_r)

rsq_a=smf.ols("ADMS~RDS+MKTS",data=data1).fit().rsquared
vif_a=1/(1-rsq_a)

rsq_m=smf.ols("MKTS~RDS+ADMS",data=data1).fit().rsquared
vif_m=1/(1-rsq_m)

# Putting the values in Dataframe format
d1={'Variables':['RDS','ADMS','MKTS'],'Vif':[vif_r,vif_a,vif_m]}
Vif_df=pd.DataFrame(d1)
Vif_df
```

Out[66]:

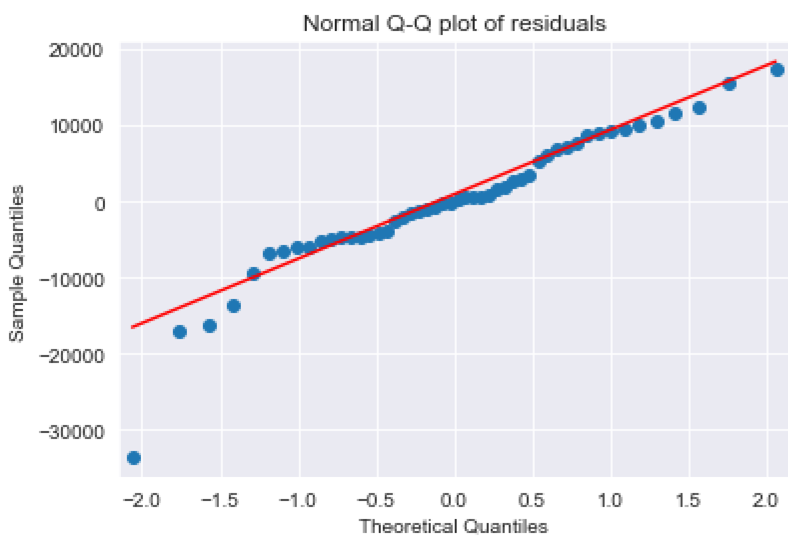
	Variables	Vif
0	RDS	2.468903
1	ADMS	1.175091
2	MKTS	2.326773

In [67]:

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

In [68]:

```
# 2) Residual Analysis
# Test for Normality of Residuals (Q-Q Plot) using residual model (model.resid)

sm.qqplot(model.resid,line='q')
plt.title("Normal Q-Q plot of residuals")
plt.show()
```



In [69]:

```
list(np.where(model.resid<-30000))
```

Out[69]:

```
[array([49], dtype=int64)]
```

In [70]:

```
# Test for Homoscedasticity or Heteroscedasticity (plotting model's standardized fitted

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

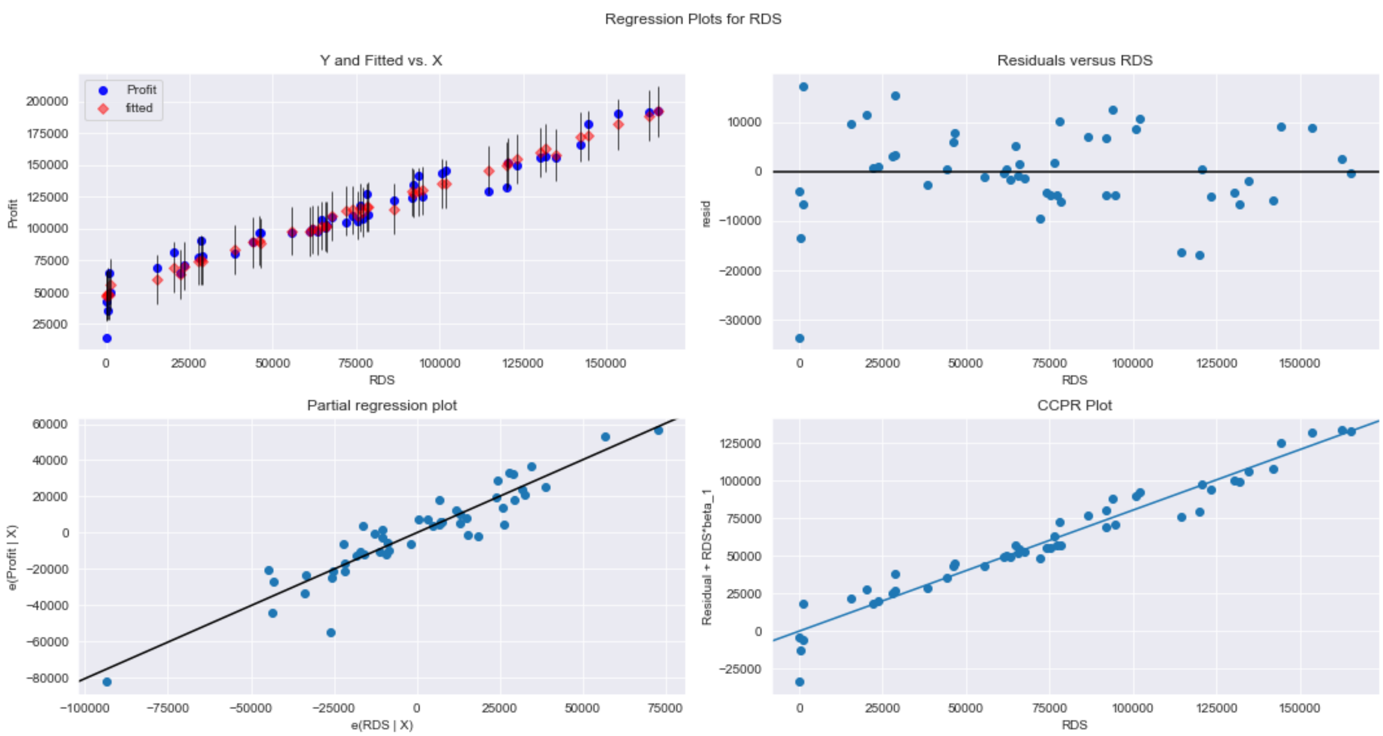
In [71]:

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

In [72]:
```
# Test for errors or Residuals Vs Regressors or independent 'x' variables or predictors
# using Residual Regression Plots code graphics.plot_regress_exog(model,'x',fig)    # ex
```

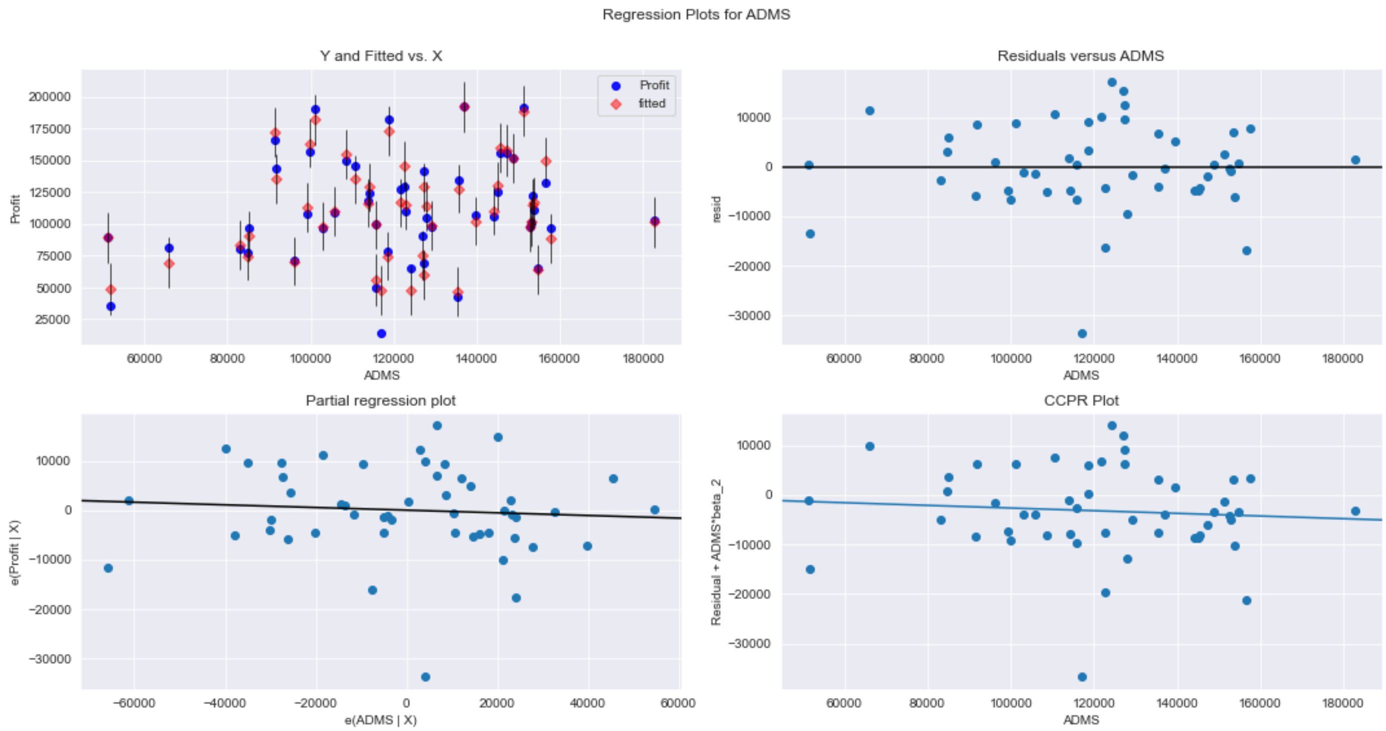
In [73]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'RDS',fig=fig)
plt.show()
```

eval_env: 1



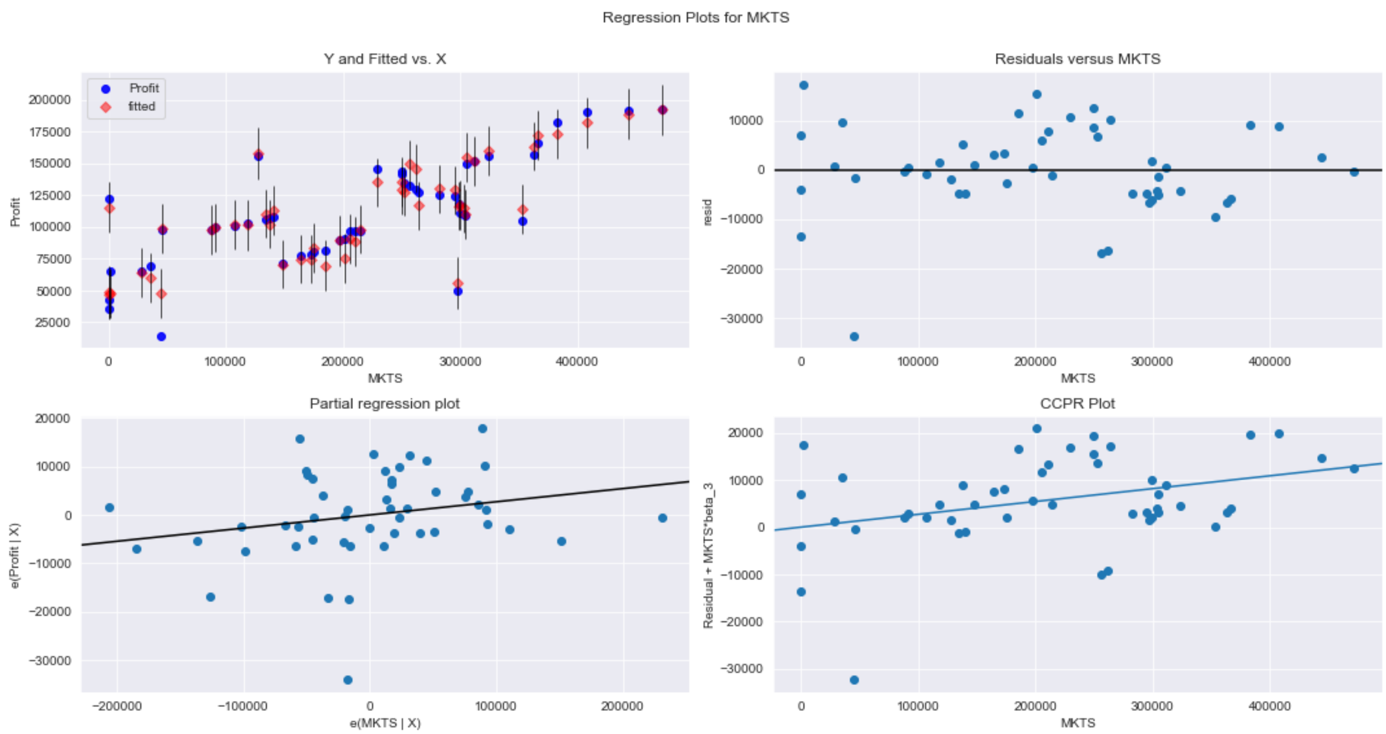
In [74]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'ADMS',fig=fig)
plt.show()
```

eval_env: 1



In [75]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'MKTS',fig=fig)
plt.show()
```

```
eval_env: 1
```



Model Deletion Diagnostics (checking Outliers or Influencers)

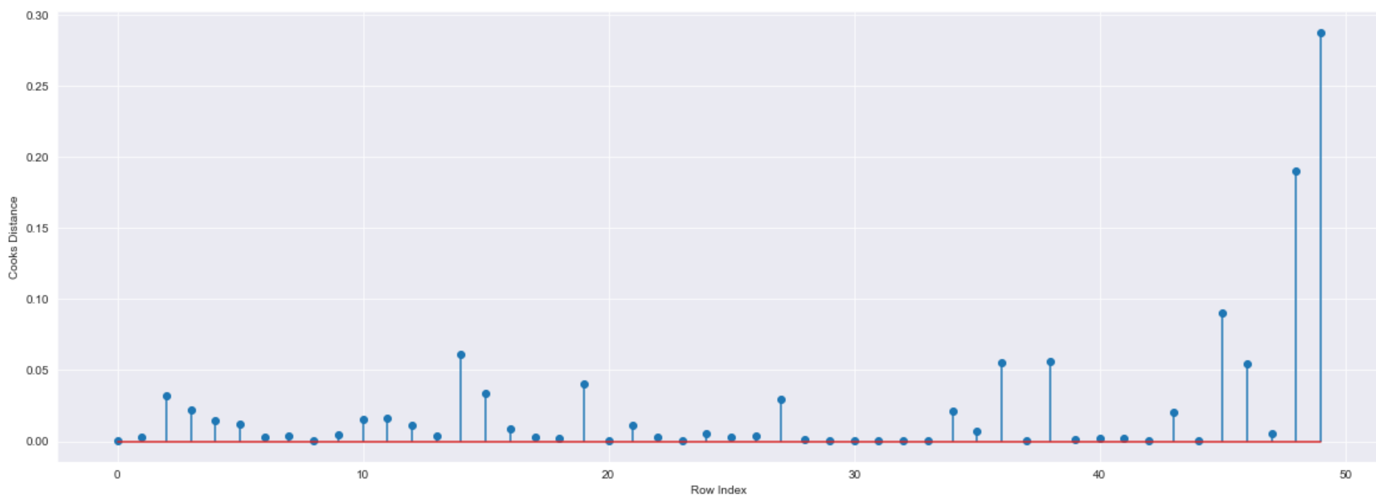
Two Techniques : 1. Cook's Distance & 2. Leverage value

In [76]:
```
# 1. Cook's Distance: If Cook's distance > 1, then it's an outlier
# Get influencers using cook's distance
```

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

Out[76]:
```
array([3.21825244e-05, 3.27591036e-03, 3.23842699e-02, 2.17206555e-02,
       1.44833032e-02, 1.17158463e-02, 2.91766303e-03, 3.56513444e-03,
       4.04303948e-05, 4.86758017e-03, 1.51064757e-02, 1.63564959e-02,
       1.15516625e-02, 4.01422811e-03, 6.12934253e-02, 3.40013448e-02,
       8.33556413e-03, 3.30534399e-03, 2.16819303e-03, 4.07440577e-02,
       4.25137222e-04, 1.09844352e-02, 2.91768000e-03, 2.76030254e-04,
       5.04643588e-03, 3.00074623e-03, 3.41957068e-03, 2.98396413e-02,
       1.31590664e-03, 1.25992620e-04, 4.18505125e-05, 9.27434786e-06,
       7.08656521e-04, 1.28122674e-04, 2.09815032e-02, 6.69508674e-03,
       5.55314705e-02, 6.55050578e-05, 5.61547311e-02, 1.54279607e-03,
       1.84850929e-03, 1.97578066e-03, 1.36089280e-04, 2.05553171e-02,
       1.23156041e-04, 9.03234206e-02, 5.45303387e-02, 5.33885616e-03,
       1.90527441e-01, 2.88082293e-01])
```

In [77]:
```
# Plot the influencers using the stem plot
fig=plt.figure(figsize=(20,7))
plt.stem(np.arange(len(data1)),np.round(c,5))
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```

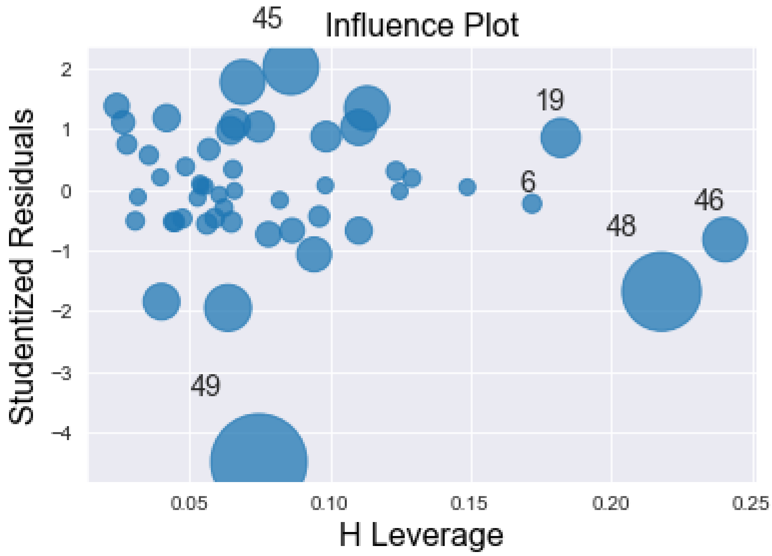


In [78]:
```
# Index and value of influencer where C>0.5
np.argmax(c) , np.max(c)
```

Out[78]:
```
(49, 0.2880822927543263)
```

In [79]:
```
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value ar
influence_plot(model)
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [80]:
```
# Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of datapoint
k=data1.shape[1]
n=data1.shape[0]
leverage_cutoff = (3*(k+1))/n
leverage_cutoff
```

Out[80]: 0.36

In [81]:
```
data1[data1.index.isin([49])]
```

Out[81]:

	RDS	ADMS	MKTS	State	Profit
49	0.0	116983.8	45173.06	California	14681.4

Improving the Model

In [82]:
```
# Discard the data points which are influencers and reassign the row number (reset_index
data2=data1.drop(data1.index[[49]],axis=0).reset_index(drop=True)
data2
```

Loading [MathJax]/extensions/Safe.js

Out[82]:

	RDS	ADMS	MKTS	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04
25	64664.71	139553.16	137962.62	California	107404.34
26	75328.87	144135.98	134050.07	Florida	105733.54
27	72107.60	127864.55	353183.81	New York	105008.31
28	66051.52	182645.56	118148.20	Florida	103282.38
29	65605.48	153032.06	107138.38	New York	101004.64
30	61994.48	115641.28	91131.24	Florida	99937.59
31	61136.38	152701.92	88218.23	New York	97483.56
32	63408.86	129219.61	46085.25	California	97427.84
33	55493.95	103057.49	214634.81	Florida	96778.92
34	46426.07	157693.92	210797.67	California	96712.80
35	46014.02	85047.44	205517.64	New York	96479.51
36	28663.76	127056.21	201126.82	Florida	90708.19
37	44069.95	51283.14	197029.42	California	89949.14
38	20229.59	65947.93	185265.10	New York	81229.06

	RDS	ADMS	MKTS	State	Profit
39	38558.51	82982.09	174999.30	California	81005.76
40	28754.33	118546.05	172795.67	California	78239.91
41	27892.92	84710.77	164470.71	Florida	77798.83
42	23640.93	96189.63	148001.11	California	71498.49
43	15505.73	127382.30	35534.17	New York	69758.98
44	22177.74	154806.14	28334.72	California	65200.33
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41

Model Deletion Diagnostics and Final Model

In [83]:
```
model2=smf.ols("Profit~RDS+ADMS+MKTS",data=data2).fit()
```

In [84]:
```
while model2.rsquared < 0.99:
    for c in [np.max(c)>1]:
        model2=smf.ols("Profit~RDS+ADMS+MKTS",data=data2).fit()
        (c,_)=model2.get_influence().cooks_distance
        c
        np.argmax(c) , np.max(c)
        data2=data2.drop(data2.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
        data2
    else:
        final_model=smf.ols("Profit~RDS+ADMS+MKTS",data=data2).fit()
        final_model.rsquared , final_model.aic
        print("Thus model accuracy is improved to",final_model.rsquared)
```

```
Thus model accuracy is improved to 0.9626766170294073
Thus model accuracy is improved to 0.9614129113440602
Thus model accuracy is improved to 0.962593650298269
Thus model accuracy is improved to 0.9638487279209413
Thus model accuracy is improved to 0.9663901957918793
Thus model accuracy is improved to 0.9706076169779906
Thus model accuracy is improved to 0.9727840588916423
Thus model accuracy is improved to 0.9734292907181952
Thus model accuracy is improved to 0.9785801571833451
Thus model accuracy is improved to 0.9777383743090915
Thus model accuracy is improved to 0.9790510088977512
Thus model accuracy is improved to 0.9790004461890552
Thus model accuracy is improved to 0.9807878666153609
Thus model accuracy is improved to 0.9838299343609735
Thus model accuracy is improved to 0.983114992639277
Thus model accuracy is improved to 0.9833768520972176
Thus model accuracy is improved to 0.9878892536376698
Thus model accuracy is improved to 0.98771919355472
Thus model accuracy is improved to 0.9858356627471713
Thus model accuracy is improved to 0.9874766829880098
Thus model accuracy is improved to 0.9906666289527223
Thus model accuracy is improved to 0.9882757054424702
```

In [85]:
```
final_model.rsquared
```

Out[85]: 0.9882757054424702

In [86]:
```
data2
```

Out[86]:

	RDS	ADMS	MKTS	State	Profit
0	142107.34	91391.77	366168.42	Florida	166187.94
1	131876.90	99814.71	362861.36	New York	156991.12
2	130298.13	145530.06	323876.68	Florida	155752.60
3	120542.52	148718.95	311613.29	New York	152211.77
4	123334.88	108679.17	304981.62	California	149759.96
5	91992.39	135495.07	252664.93	California	134307.35
6	94657.16	145077.58	282574.31	New York	125370.37
7	91749.16	114175.79	294919.57	Florida	124266.90
8	76253.86	113867.30	298664.47	California	118474.03
9	67532.53	105751.03	304768.73	Florida	108733.99
10	77044.01	99281.34	140574.81	New York	108552.04
11	64664.71	139553.16	137962.62	California	107404.34
12	75328.87	144135.98	134050.07	Florida	105733.54
13	66051.52	182645.56	118148.20	Florida	103282.38
14	65605.48	153032.06	107138.38	New York	101004.64
15	61994.48	115641.28	91131.24	Florida	99937.59
16	61136.38	152701.92	88218.23	New York	97483.56
17	63408.86	129219.61	46085.25	California	97427.84
18	55493.95	103057.49	214634.81	Florida	96778.92
19	46426.07	157693.92	210797.67	California	96712.80
20	46014.02	85047.44	205517.64	New York	96479.51
21	44069.95	51283.14	197029.42	California	89949.14
22	38558.51	82982.09	174999.30	California	81005.76
23	28754.33	118546.05	172795.67	California	78239.91
24	27892.92	84710.77	164470.71	Florida	77798.83
25	23640.93	96189.63	148001.11	California	71498.49
26	22177.74	154806.14	28334.72	California	65200.33

Model Predictions

In [87]:
```
# say New data for prediction is
new_data=pd.DataFrame({'RDS':70000,"ADMS":90000,"MKTS":140000},index=[0])
new_data
```

Out[87]:

	RDS	ADMS	MKTS
0	70000	90000	140000

Loading [MathJax]/extensions/Safe.js

```
#...diction of Price
```

In [88]:
```
final_model.predict(new_data)
```

Out[88]:
```
0    104858.729408
dtype: float64
```

In [89]:
```
# Automatic Prediction of Price with 90.02% accurcy
pred_y=final_model.predict(data2)
pred_y
```

Out[89]:
```
0     165589.539700
1     158552.826483
2     156789.000710
3     149524.698853
4     150122.356712
5     126598.769555
6     130104.785747
7     127878.387928
8     117298.757074
9     111329.242429
10    110009.916133
11    102331.717613
12    109661.804131
13    103462.767086
14    101874.612012
15     97655.794577
16     97872.919535
17     96858.382686
18     98654.449007
19     93583.600868
20     91186.568204
21     88571.938968
22     84521.312916
23     78528.002935
24     76670.262623
25     73237.524757
26     68075.710756
dtype: float64
```

Table containing R^2 value for each prepared model

In [90]:
```
d2={'Prep_Models':['Model','Final_Model'],'Rsquared':[model.rsquared,final_model.rsquare
table=pd.DataFrame(d2)
table
```

Out[90]:

	Prep_Models	Rsquared
0	Model	0.950746
1	Final_Model	0.988276

In [ ]: