Import Libraries

In [1]:
```
#Import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import warnings
warnings.filterwarnings("ignore")
```

Import dataset

In [2]:
```
#Import Datasets With nessasary columns only.
data=pd.read_csv("ToyotaCorolla.csv",usecols=("Price","Age_08_04","KM","HP","cc","Doors","Gears","Quarterly_Tax","Weight"))
```

In [3]:
```
#Read top 5 Rows
data.head()
```

Out[3]:

	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

Data Undestadning

In [4]:
```
#Infomation about any null value avilible in data
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             1436 non-null   int64
 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: int64(9)
memory usage: 101.1 KB
```

In [5]:
```
#Number of columns and rows in datasets
data.shape
```

Out[5]:
```
(1436, 9)
```

In [6]:
```
#if any null values avilible in given dataset
data.isnull().sum()
```

Out[6]:
```
Price            0
Age_08_04        0
KM               0
HP               0
cc               0
Doors            0
Gears            0
Quarterly_Tax    0
Weight           0
dtype: int64
```

In [7]:
```
#Indicate datatype of all columns
data.dtypes
```

Out[7]:
```
Price            int64
Age_08_04        int64
KM               int64
HP               int64
cc               int64
Doors            int64
Gears            int64
Quarterly_Tax    int64
Weight           int64
dtype: object
```

In [8]:
```
#Rename columns name for easy interpretation.
data=data.rename({"Price":"price","Age_08_04":"age","KM":"km","HP":"hp","Doors":"door","Gears":"gear","Quarterly_Tax":"tax","Weight":"weight"},axis=1
```

In [9]:
```
#Top 5 datasets
data.head()
```

Out[9]:

	price	age	km	hp	cc	door	gear	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

In [10]:
```
#check wether Dublicate data avilible or not
data[data.duplicated()]
```

Out[10]:

	price	age	km	hp	cc	door	gear	tax	weight
113	24950	8	13253	116	2000	5	5	234	1320

In [11]:
```python
#Drop dublicate rows because it's impect accuracy
data=data.drop_duplicates().reset_index(drop=True)
```

Correlation Analysis

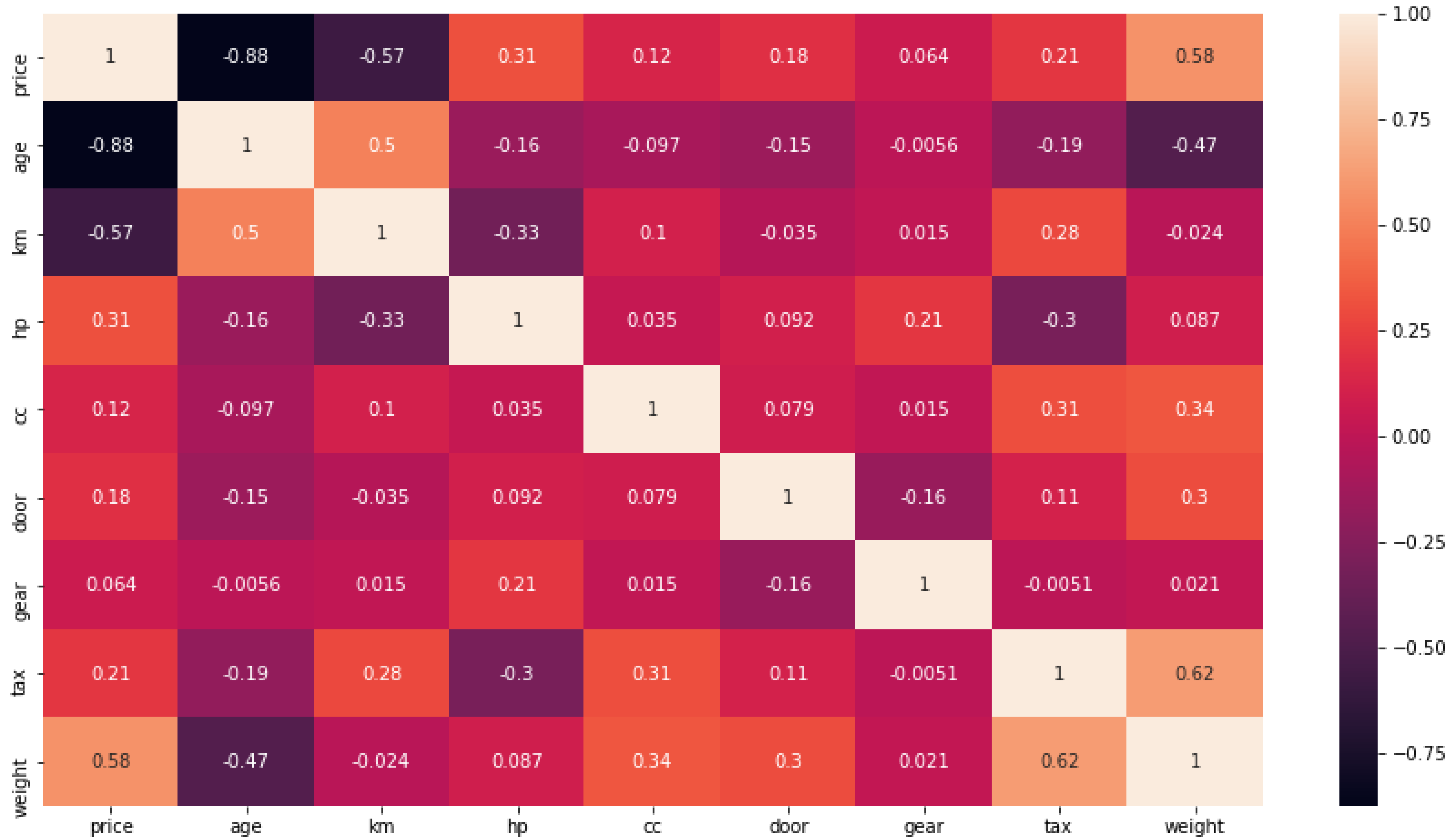
In [12]:
```python
#Correlation between given data.
data.corr()
```

Out[12]:

	price	age	km	hp	cc	door	gear	tax	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
door	0.183604	-0.146929	-0.035193	0.091803	0.079254	1.000000	-0.160101	0.107353	0.301734
gear	0.063831	-0.005629	0.014890	0.209642	0.014732	-0.160101	1.000000	-0.005125	0.021238
tax	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 [13]:
```python
# If the two variables move in the same direction, then those variables are said to have a positive correlation.
# If they move in opposite directions, then they have a negative correlation.
plt.figure(figsize=(15,8))
sns.heatmap(data.corr(),annot=True)
```

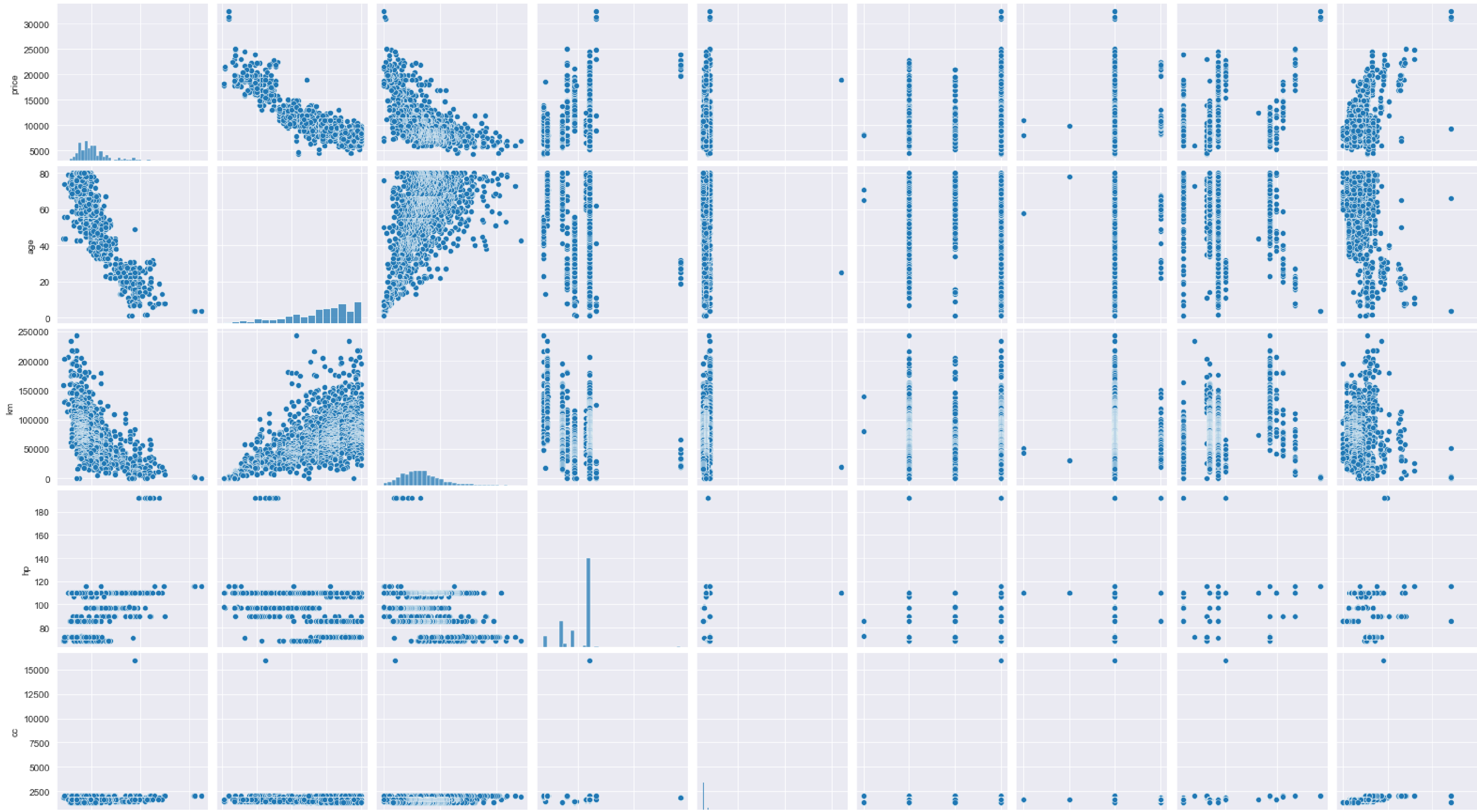
Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdaf3d5c10>
```

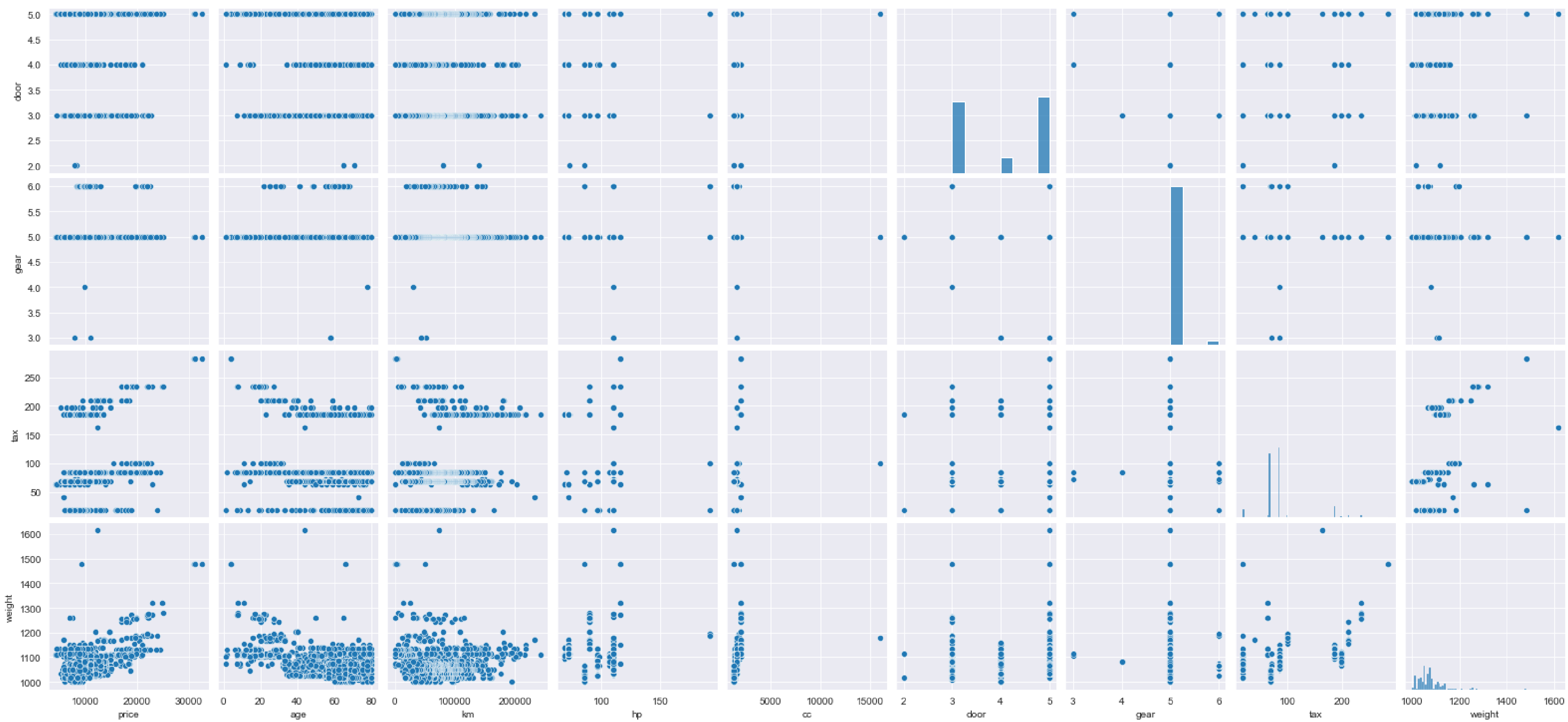


Pair Plot

In [14]:
```
#check distribution
sns.set_style(style='darkgrid')
sns.pairplot(data)
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x1bdb180e520>





Model Building

In [15]:
```
#Build the model
model=smf.ols("price~age+km+hp+cc+door+gear+tax+weight",data=data).fit()
```

Model Testing

In [16]:
```
# Finding Coefficient parameters
model.params
```

Out[16]:
```
Intercept   -5472.540368
age          -121.713891
km             -0.020737
hp             31.584612
cc             -0.118558
door           -0.920189
gear          597.715894
tax             3.858805
weight         16.855470
dtype: float64
```

In [17]:
```
#Finding tvalues and pvalue
model.tvalues,model.pvalues
```

Out[17]:
```
(Intercept    -3.875273
 age         -46.551876
 km          -16.552424
 hp           11.209719
 cc           -1.316436
 door         -0.023012
 gear          3.034563
 tax           2.944198
 weight       15.760663
 dtype: float64,
 Intercept     1.113392e-04
 age          1.879217e-288
 km            1.994713e-56
 hp            5.211155e-28
 cc            1.882393e-01
 door          9.816443e-01
 gear          2.452430e-03
 tax           3.290363e-03
 weight        1.031118e-51
 dtype: float64)
```

In [18]:
```
# Finding rsquared values
model.rsquared,model.rsquared_adj  #Accuracy we get is 86.25%
```

Out[18]:
```
(0.8625200256947001, 0.8617487495415147)
```

In [19]:
```
# Build SLR and MLR models for insignificant variables 'CC' and 'Doors'
# Also find their tvalues and pvalues.
```

In [20]:
```
slr_cc=smf.ols("price~cc",data=data).fit()
slr_cc.tvalues,slr_cc.pvalues
# cc has significant pvalue
```

Out[20]:
```
(Intercept    24.879592
 cc            4.745039
 dtype: float64,
 Intercept    7.236022e-114
 cc            2.292856e-06
 dtype: float64)
```

In [21]:
```
slr_door=smf.ols("price~door",data=data).fit()
slr_door.tvalues,slr_door.pvalues
# door has significant pvalue
```

Out[21]:
```
(Intercept    19.421546
 door          7.070520
 dtype: float64,
 Intercept    8.976407e-75
 door         2.404166e-12
 dtype: float64)
```

In [22]:
```
mlr_dc=smf.ols("price~door+cc",data=data).fit()
mlr_dc.tvalues,mlr_dc.pvalues
# door and cc has significant pvalue
```

Out[22]:
```
(Intercept    12.786341
 door          6.752236
 cc            4.268006
 dtype: float64,
 Intercept    1.580945e-35
 door         2.109558e-11
 cc           2.101878e-05
 dtype: float64)
```

Model Colinearity Test

1.Colinearity Test 2.Residual analysis

1.Colinearity Test

In [23]:
```
# 1) Collinearity Problem Check
# Calculate VIF = 1/(1-Rsquare) for all independent variables

rsq_age=smf.ols('age~km+hp+cc+door+gear+tax+weight',data=data).fit().rsquared
vif_age=1/(1-rsq_age)
print(vif_age)

rsq_km=smf.ols('km~age+hp+cc+door+gear+tax+weight',data=data).fit().rsquared
vif_km=1/(1-rsq_km)
print(vif_km)


rsq_hp=smf.ols('hp~km+age+cc+door+gear+tax+weight',data=data).fit().rsquared
vif_hp=1/(1-rsq_hp)
print(vif_hp)


rsq_cc=smf.ols('cc~km+age+hp+door+gear+tax+weight',data=data).fit().rsquared
vif_cc=1/(1-rsq_cc)
print(vif_cc)


rsq_door=smf.ols('door~km+age+cc+hp+gear+tax+weight',data=data).fit().rsquared
vif_door=1/(1-rsq_door)
print(vif_door)


rsq_gear=smf.ols('gear~km+age+cc+door+hp+tax+weight',data=data).fit().rsquared
vif_gear=1/(1-rsq_gear)
print(vif_gear)


rsq_tax=smf.ols('tax~km+age+cc+door+gear+hp+weight',data=data).fit().rsquared
vif_tax=1/(1-rsq_tax)
print(vif_tax)

rsq_weight=smf.ols('weight~km+age+cc+door+gear+tax+age',data=data).fit().rsquared
vif_weight=1/(1-rsq_weight)
print(vif_weight)
```

```
1.8762358497682892
```

```
1.7571780239810404
1.419180108718214
1.1634703645940858
1.155889865814207
1.0988429081631153
2.2953745089857147
2.314084315084272
```

In [24]:
```
# Putting the values in Dataframe format
df={"Model":["age","km",'hp','cc','door','gear','tax','Weight'],"VIF":[1.87,1.75,1.41,1.16,1.15,1.09,2.31,2.33]}
df=pd.DataFrame(df)
df
```

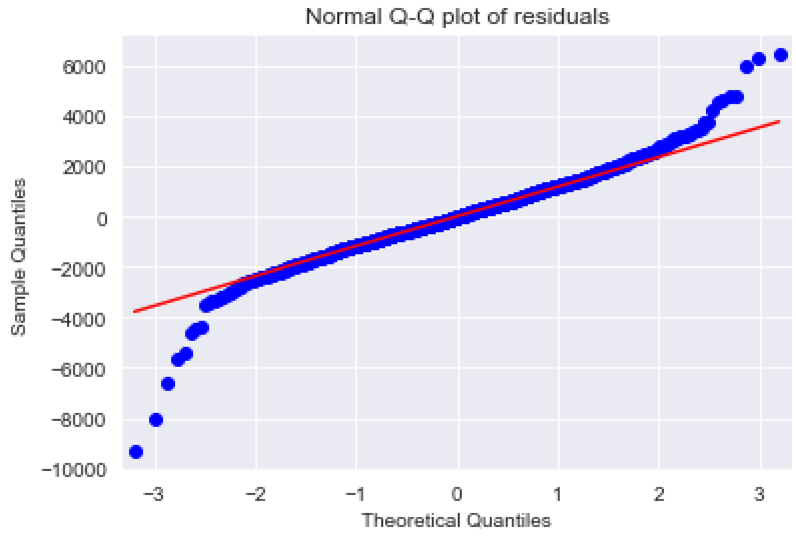
Out[24]:

	Model	VIF
0	age	1.87
1	km	1.75
2	hp	1.41
3	cc	1.16
4	door	1.15
5	gear	1.09
6	tax	2.31
7	Weight	2.33

In [25]:
```
#None Variable has Colinearity.no VIF>20.Consider all parameters for regression
```

2.Residual analysis

In [26]:
```
#Test For Normality of residual Q-Q plot using Residual Model
# 'q' - A line is fit through the quartiles # line = '45'- to draw the 45-degree diagonal line
sm.qqplot(model.resid,line="q")
plt.title("Normal Q-Q plot of residuals")
plt.show()
```



In [27]:
```
# outliar detection from above QQ plot of residuals
list(np.where(model.resid<-6000))
```

Out[27]:
```
[array([220, 600, 959], dtype=int64)]
```

In [28]:
```
# Test for Homoscedasticity or Heteroscedasticity (plotting model's standardized fitted values vs standardized residual values)
# User defined z = (x - mu)/sigma
def get_norm(var):
    return (var-var.mean())/var.std()
```

Test for Error or Residual vs Regressor

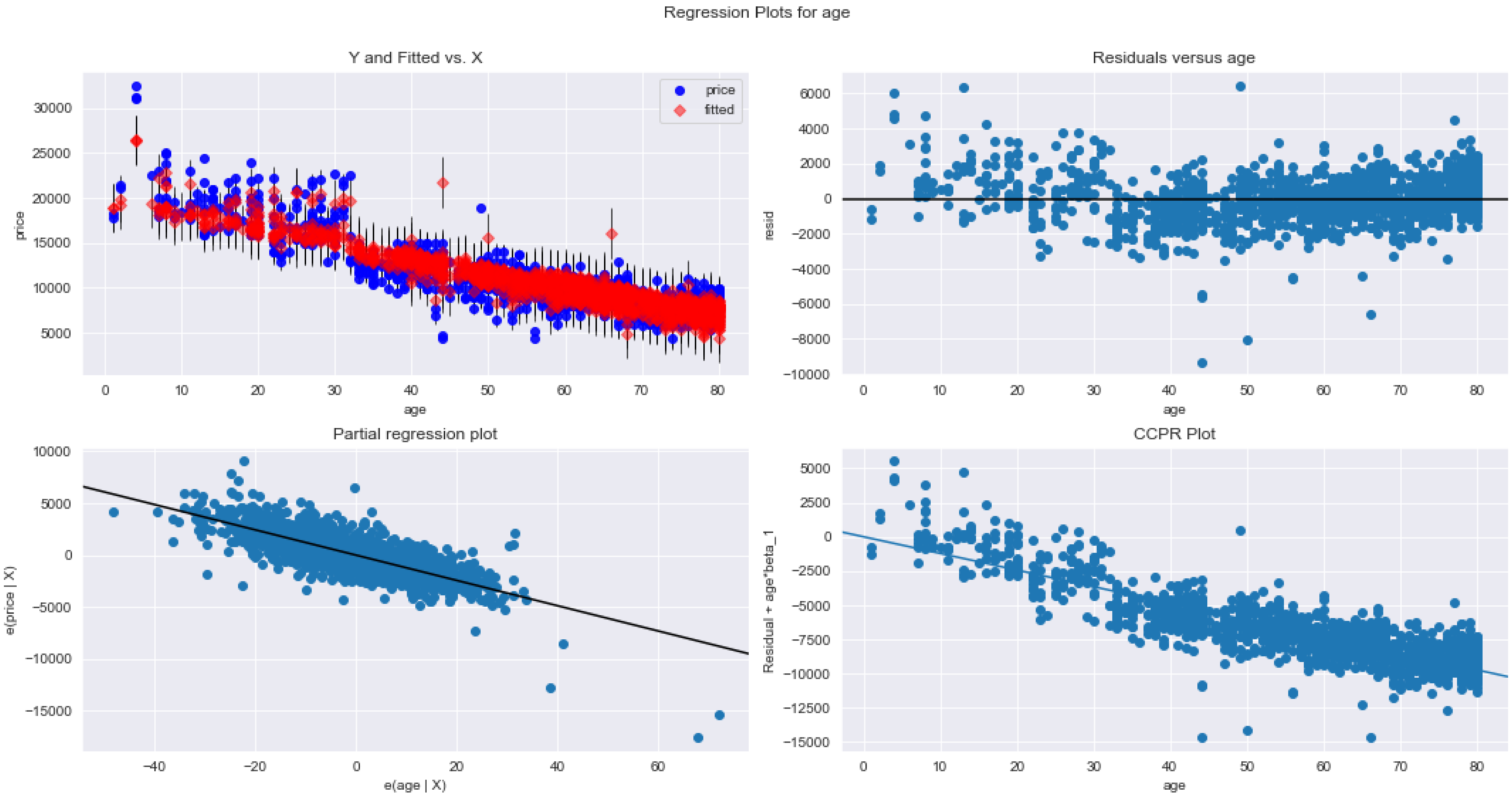
In [29]:
```
plt.scatter(get_norm(model.fittedvalues),get_norm(model.resid))
plt.xlabel("Standardize Fitted Values")
plt.ylabel("Standarize reasidual Values")
plt.title('Residual Plot')
plt.show()
```

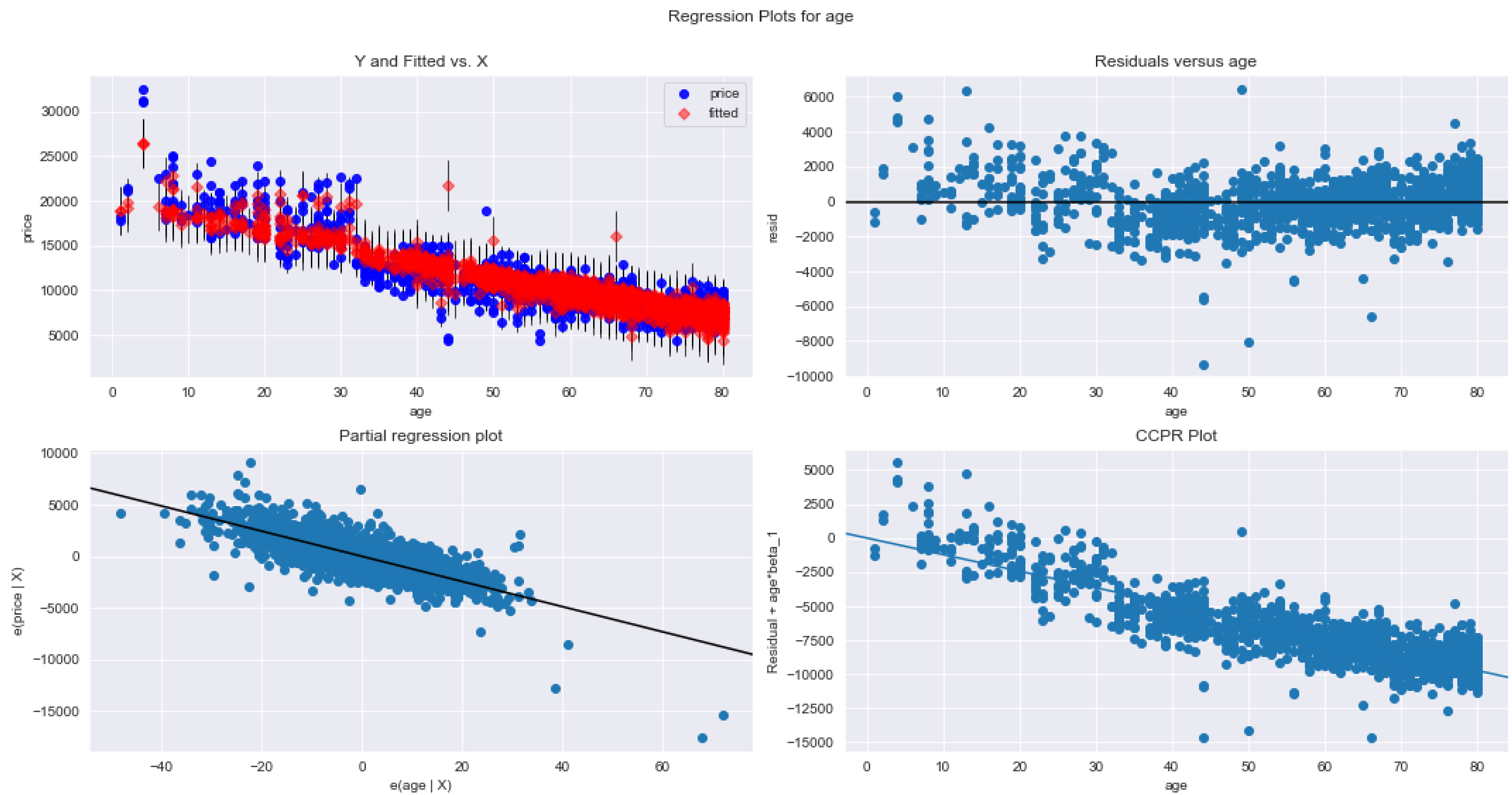
Residual Regression Plot

In [30]:
```
# 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)
# exog = x-variable & endog = y-variable
```

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

Out[31]:

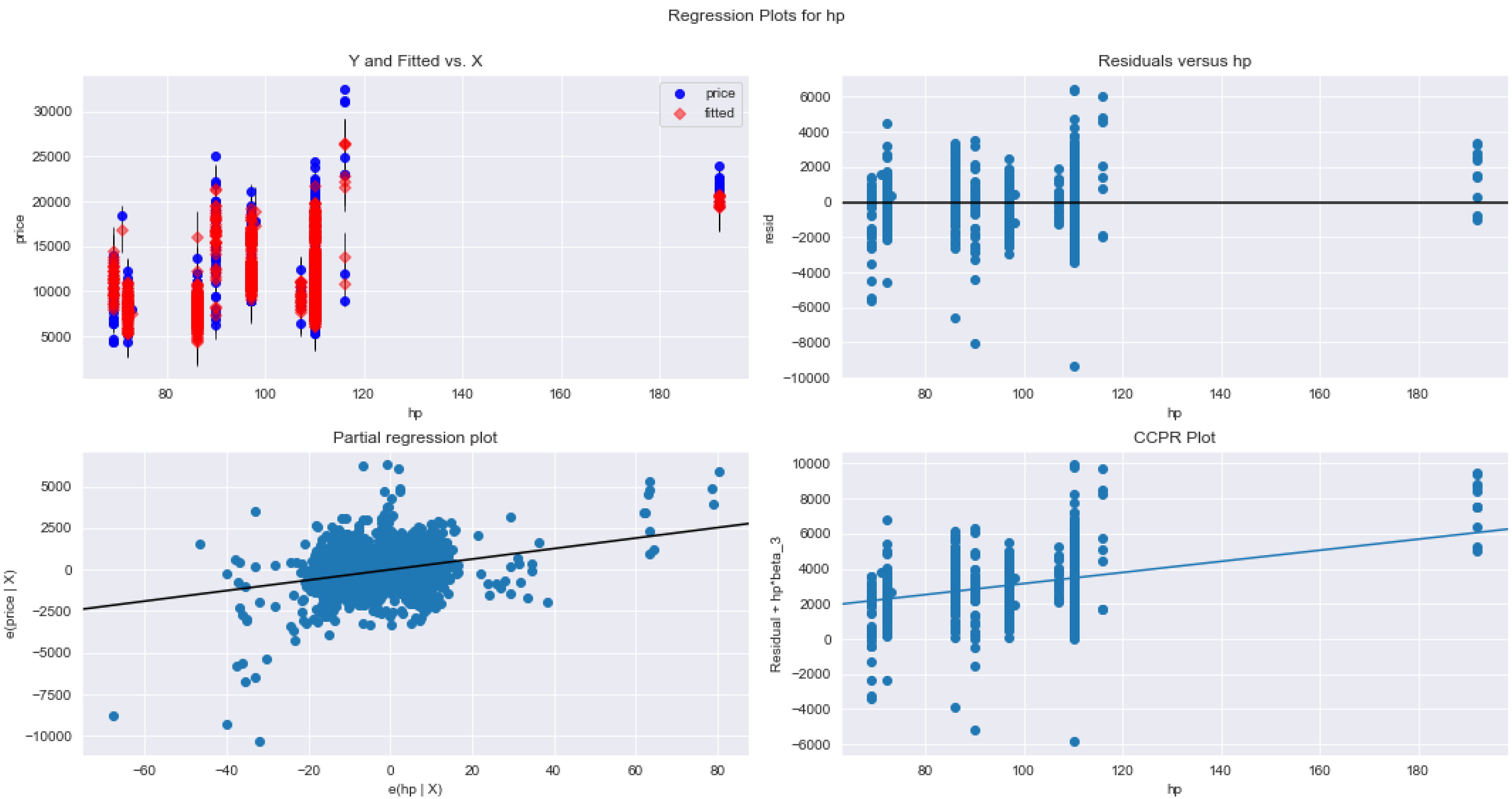


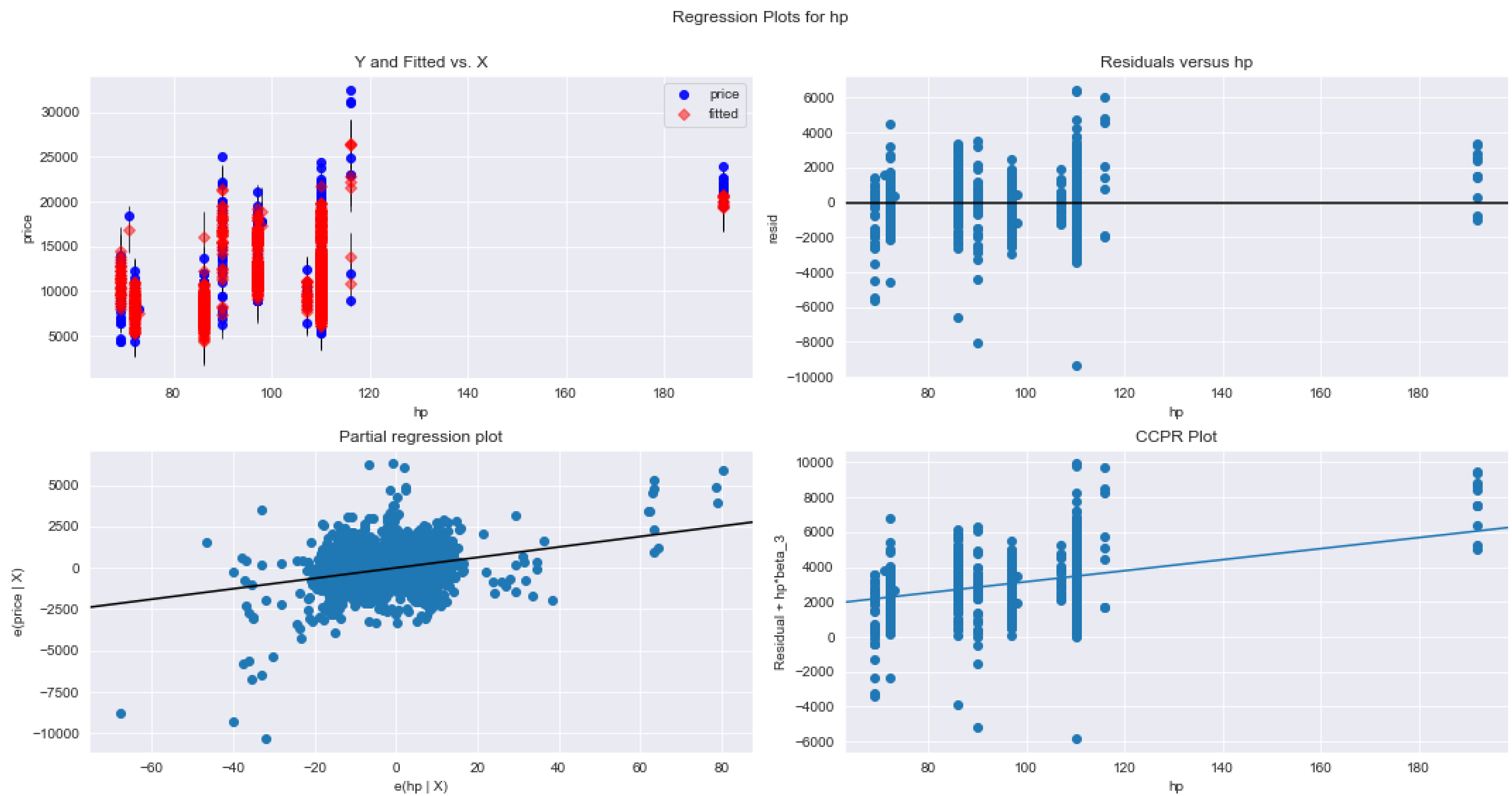


Regression Plots for age
Y and Fitted vs. X
price
fitted
age
Residuals versus age
resid
age
Partial regression plot
e(price | X)
e(age | X)
CCPR Plot
Residual + age*beta_1
age

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

Out[32]:

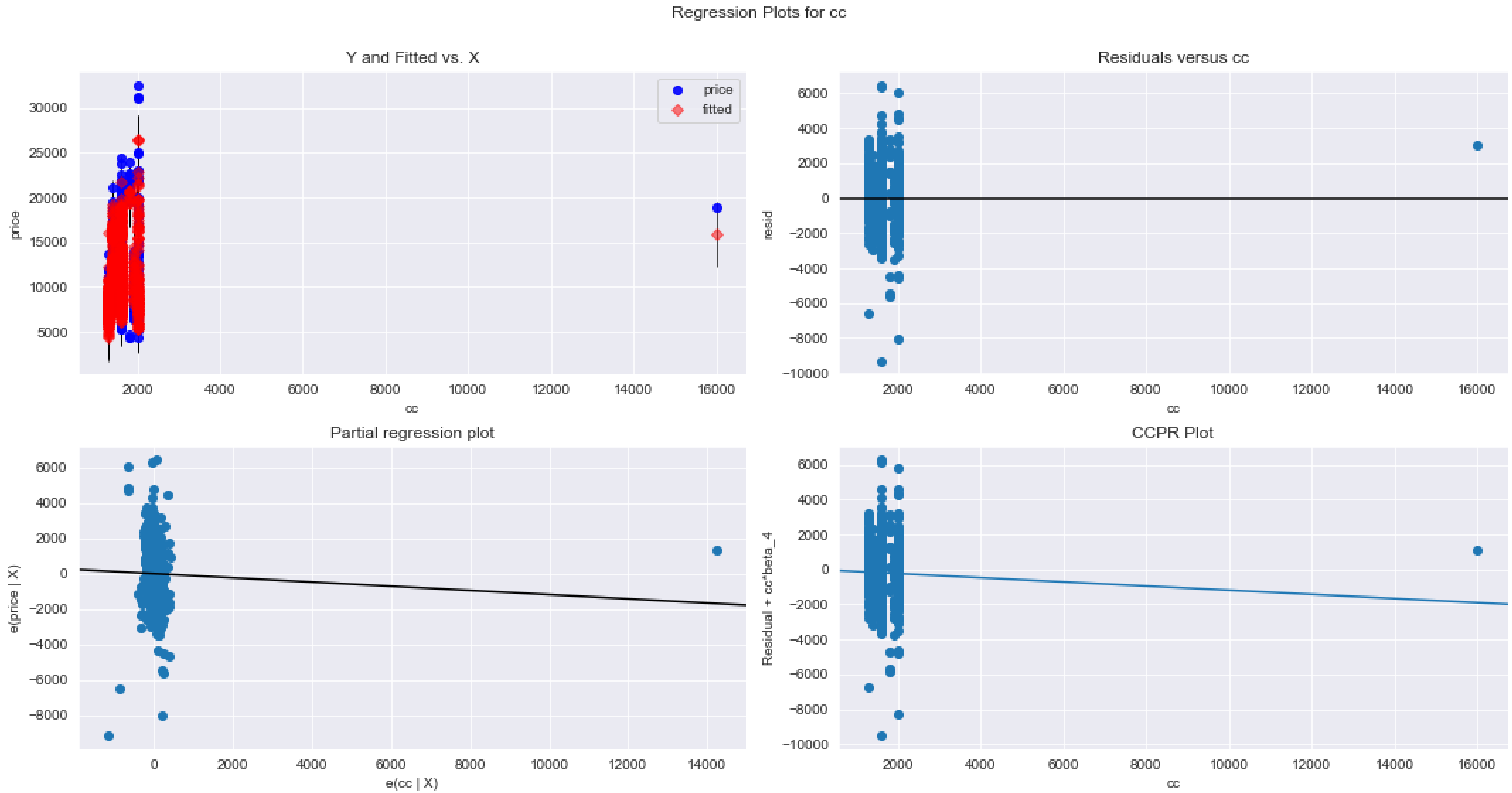


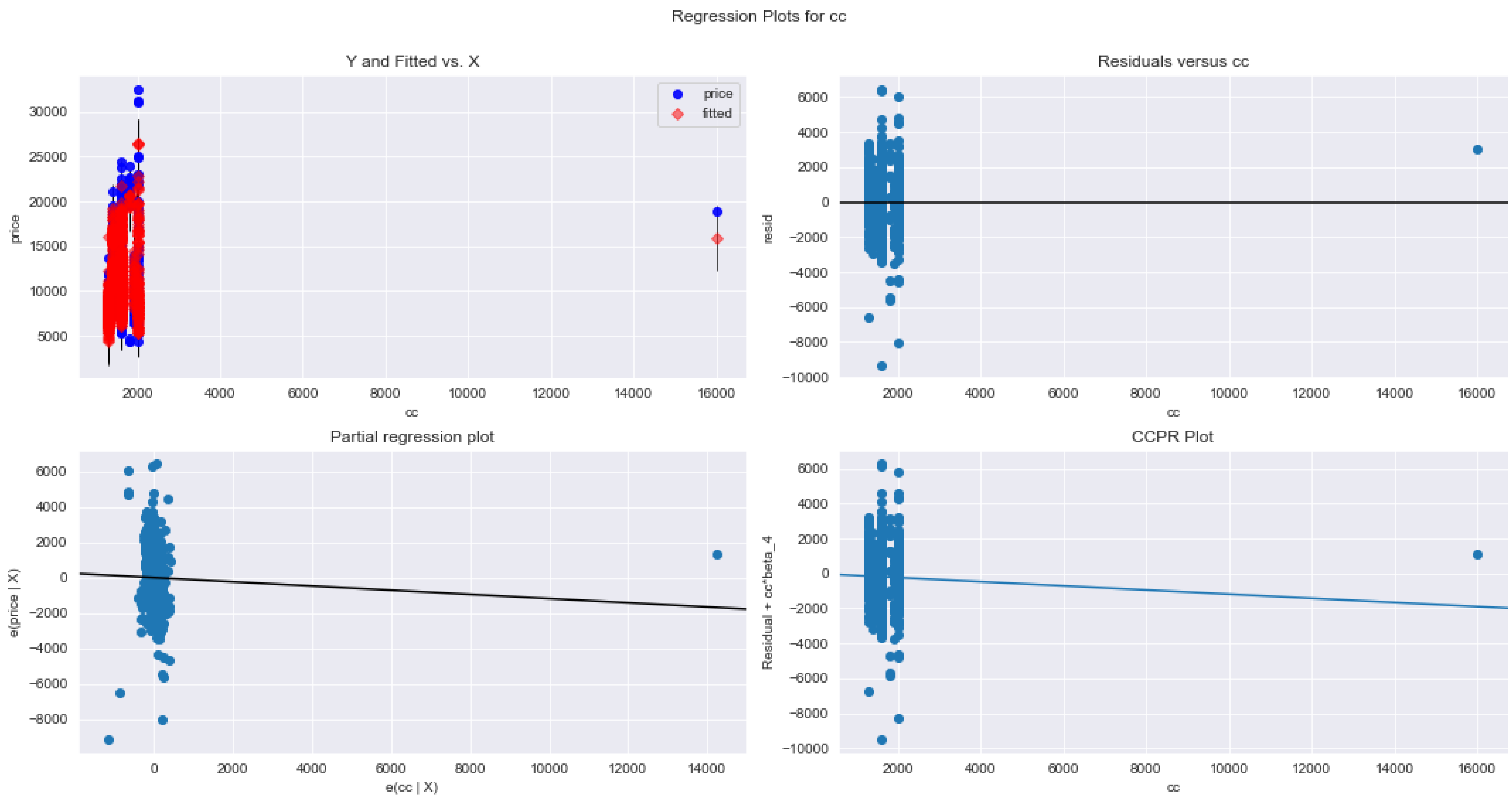


Regression Plots for hp
Y and Fitted vs. X
price
fitted
hp
Residuals versus hp
resid
hp
Partial regression plot
e(price | X)
e(hp | X)
CCPR Plot
Residual + hp*beta_3
hp

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

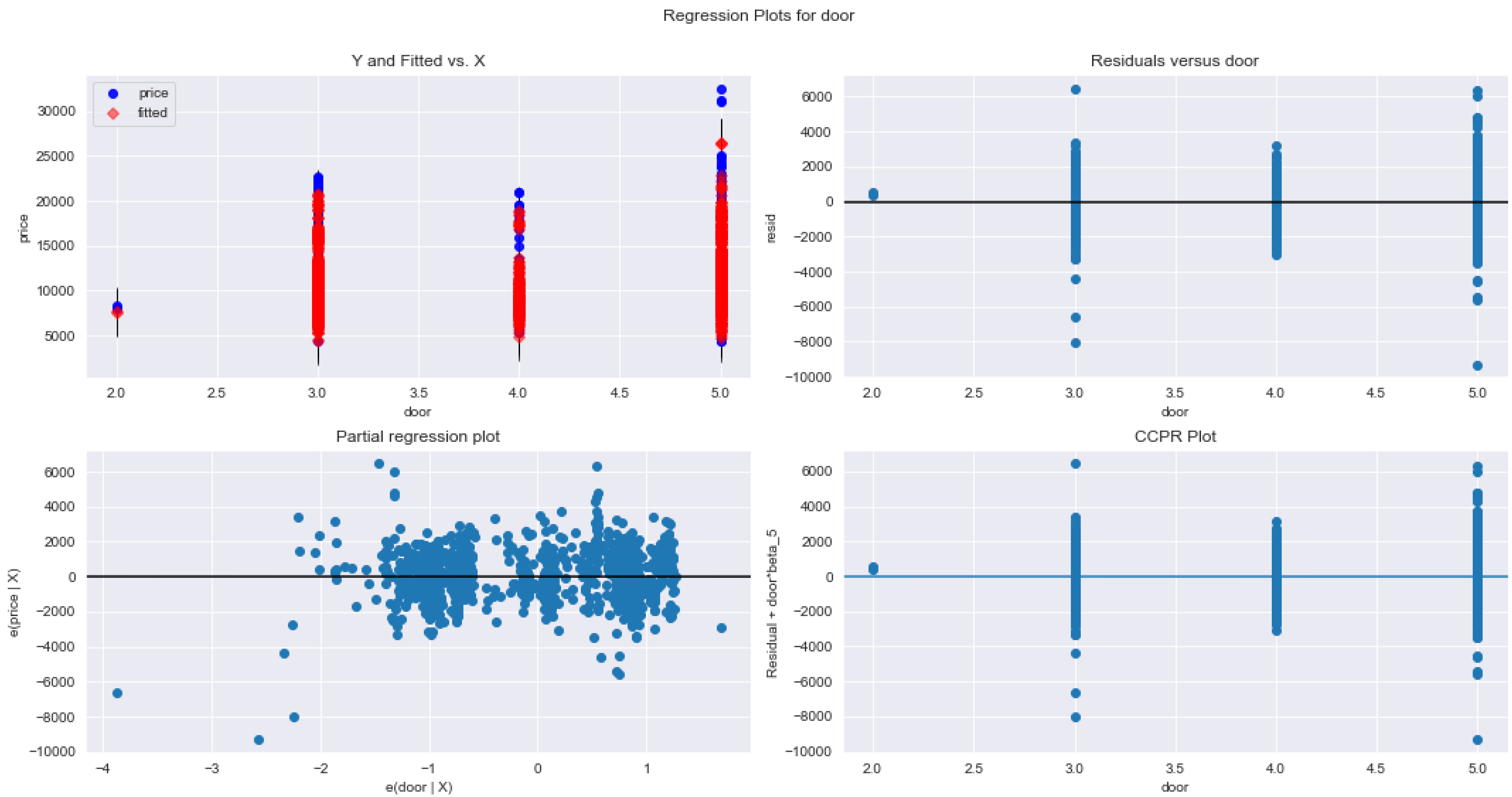
Out[33]:

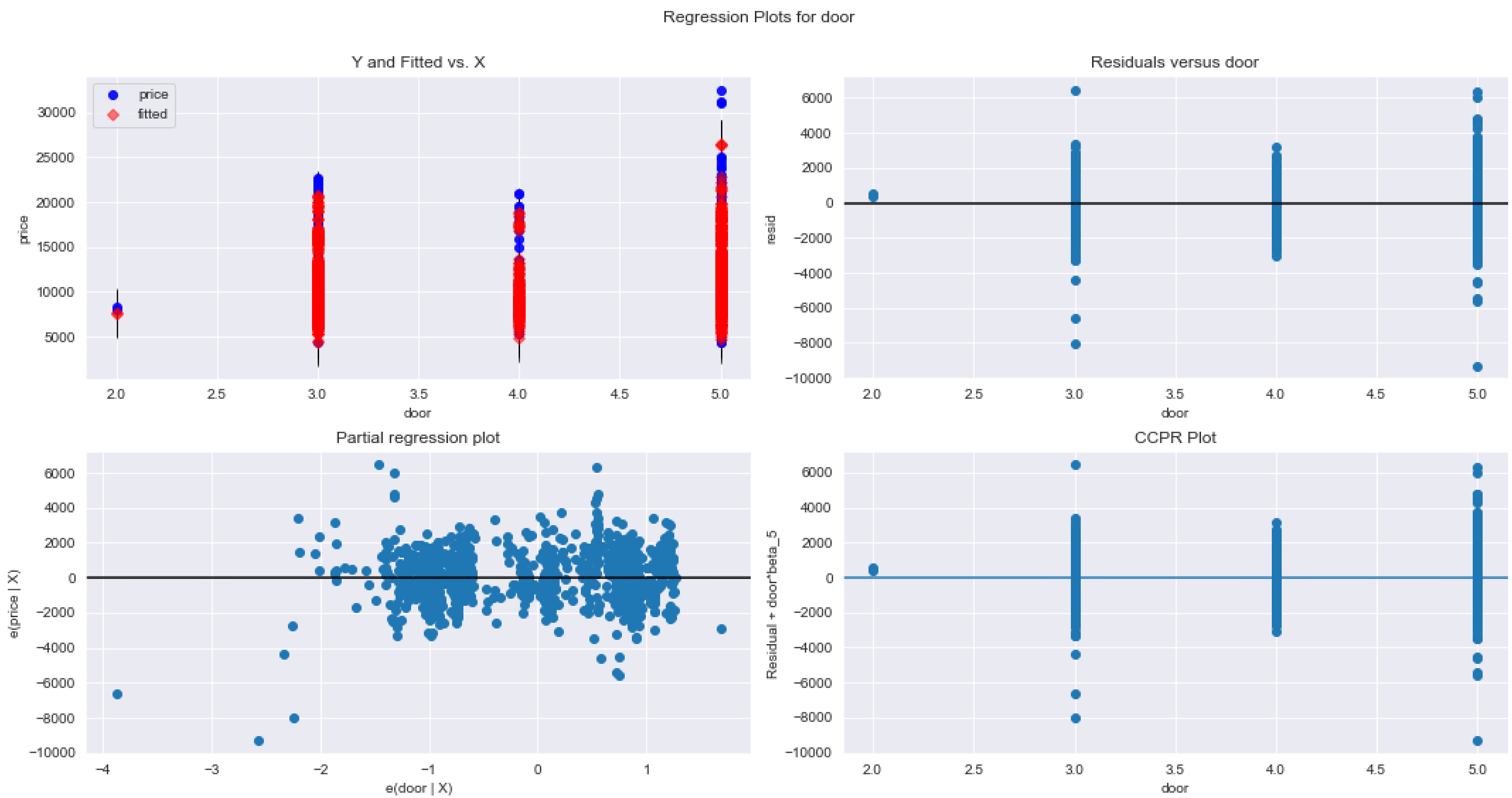



Regression Plots for cc
Y and Fitted vs. X
price
fitted
price
cc
Residuals versus cc
resid
cc
Partial regression plot
e(price | X)
e(cc | X)
CCPR Plot
Residual + cc*beta_4
cc

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

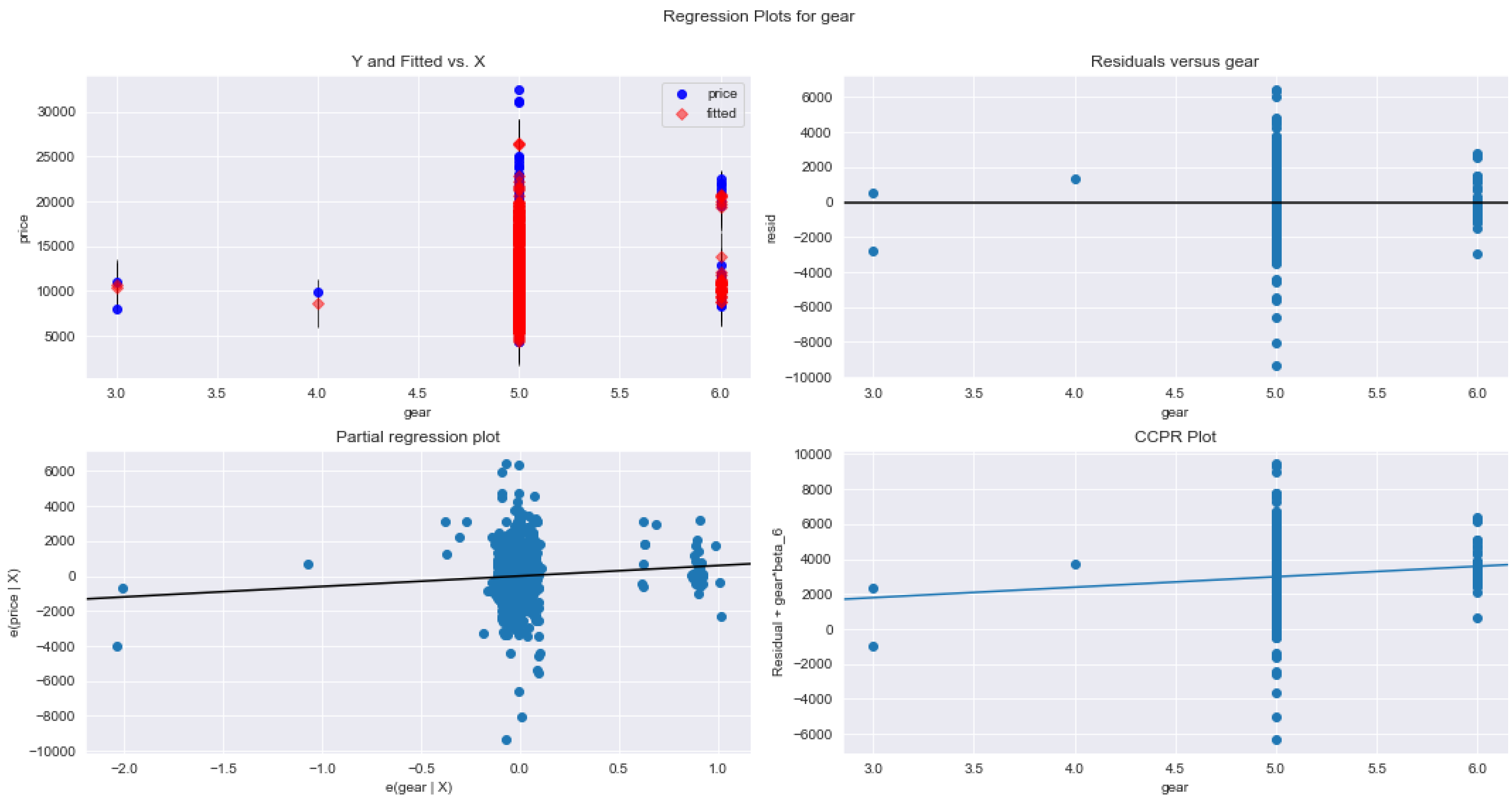
Out[34]:

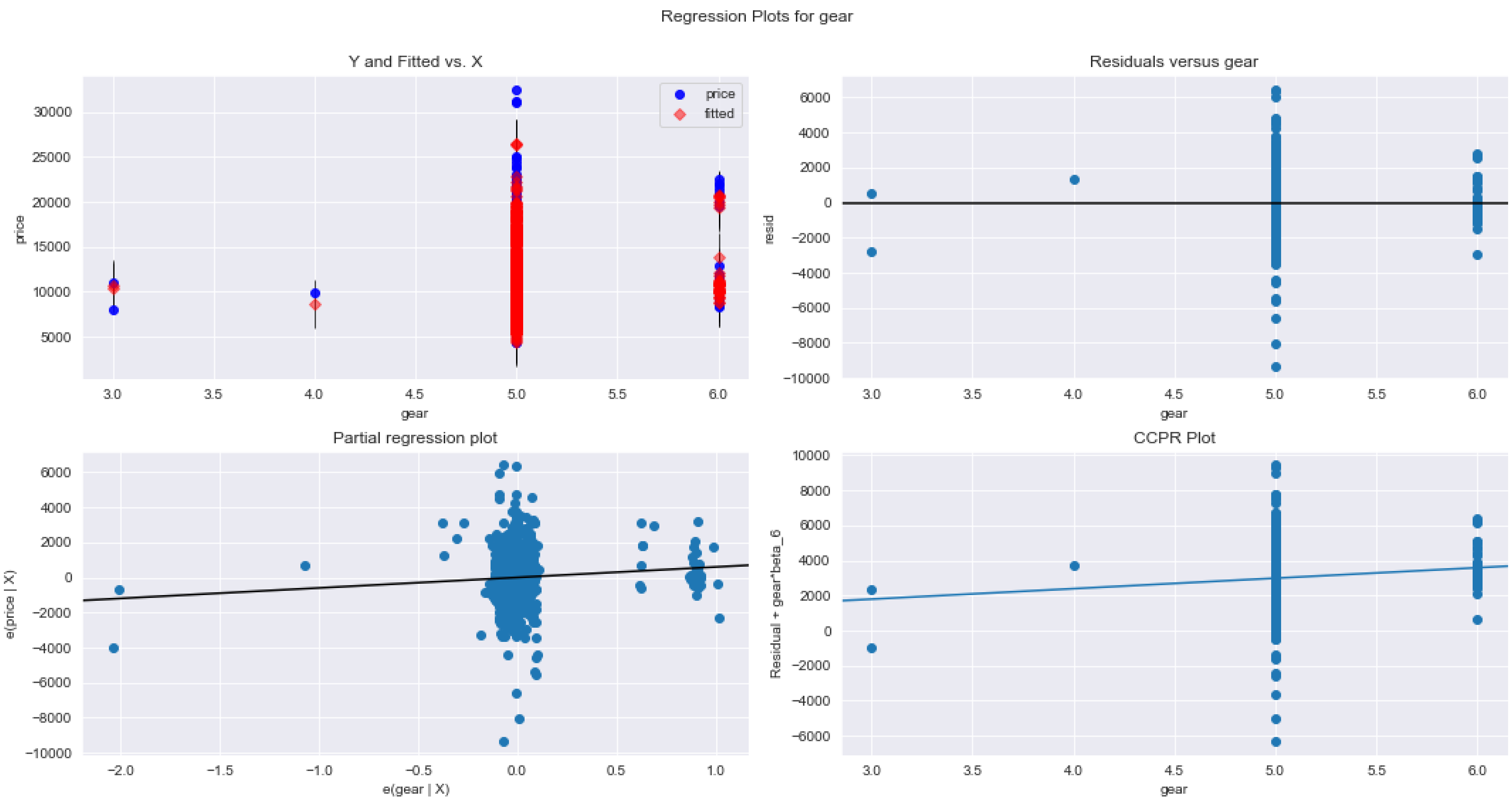



Regression Plots for door
Y and Fitted vs. X
price
fitted
door
Residuals versus door
resid
door
Partial regression plot
e(price | X)
e(door | X)
CCPR Plot
Residual + door*beta_5
door

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

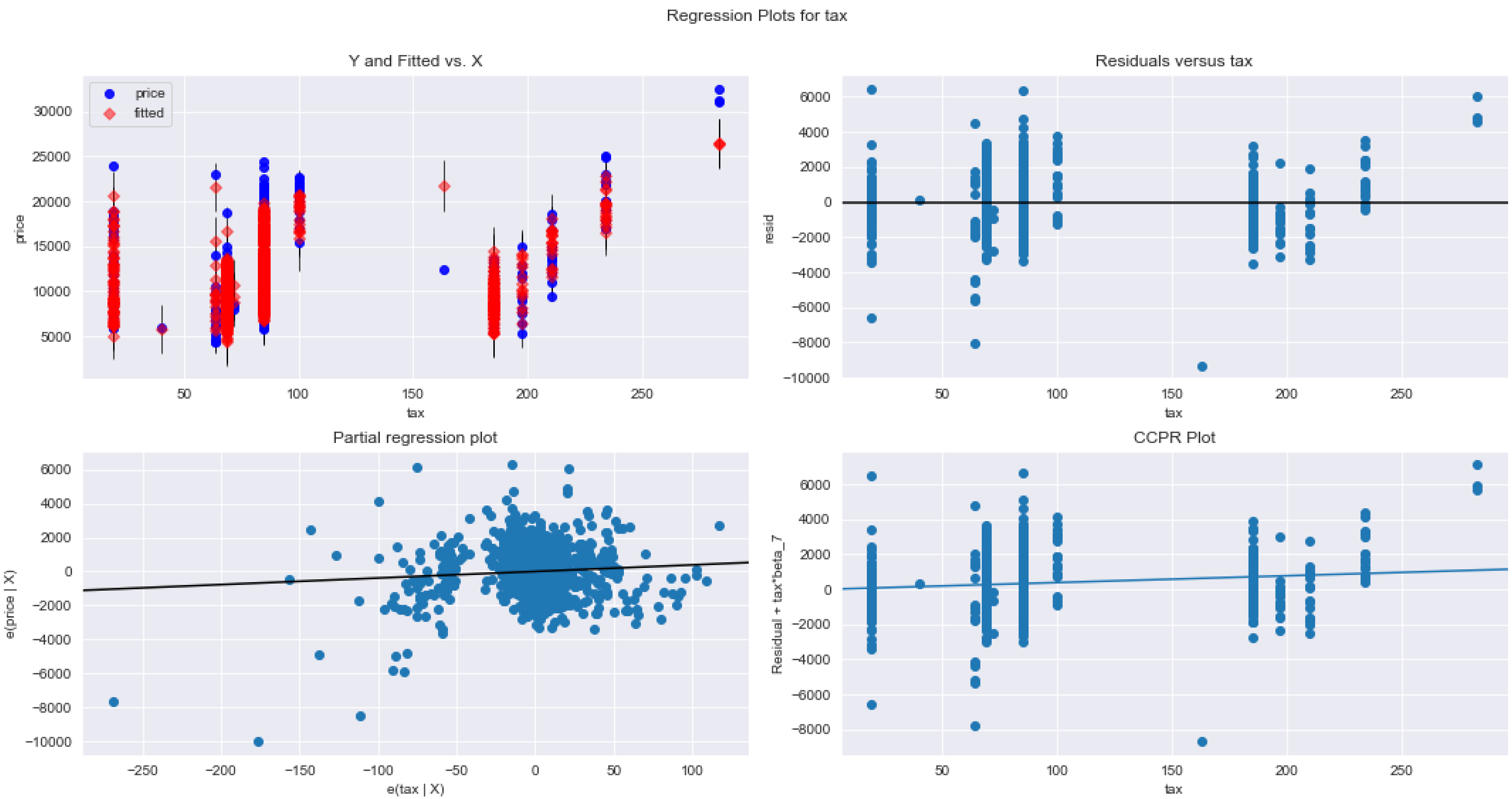
Out[35]:

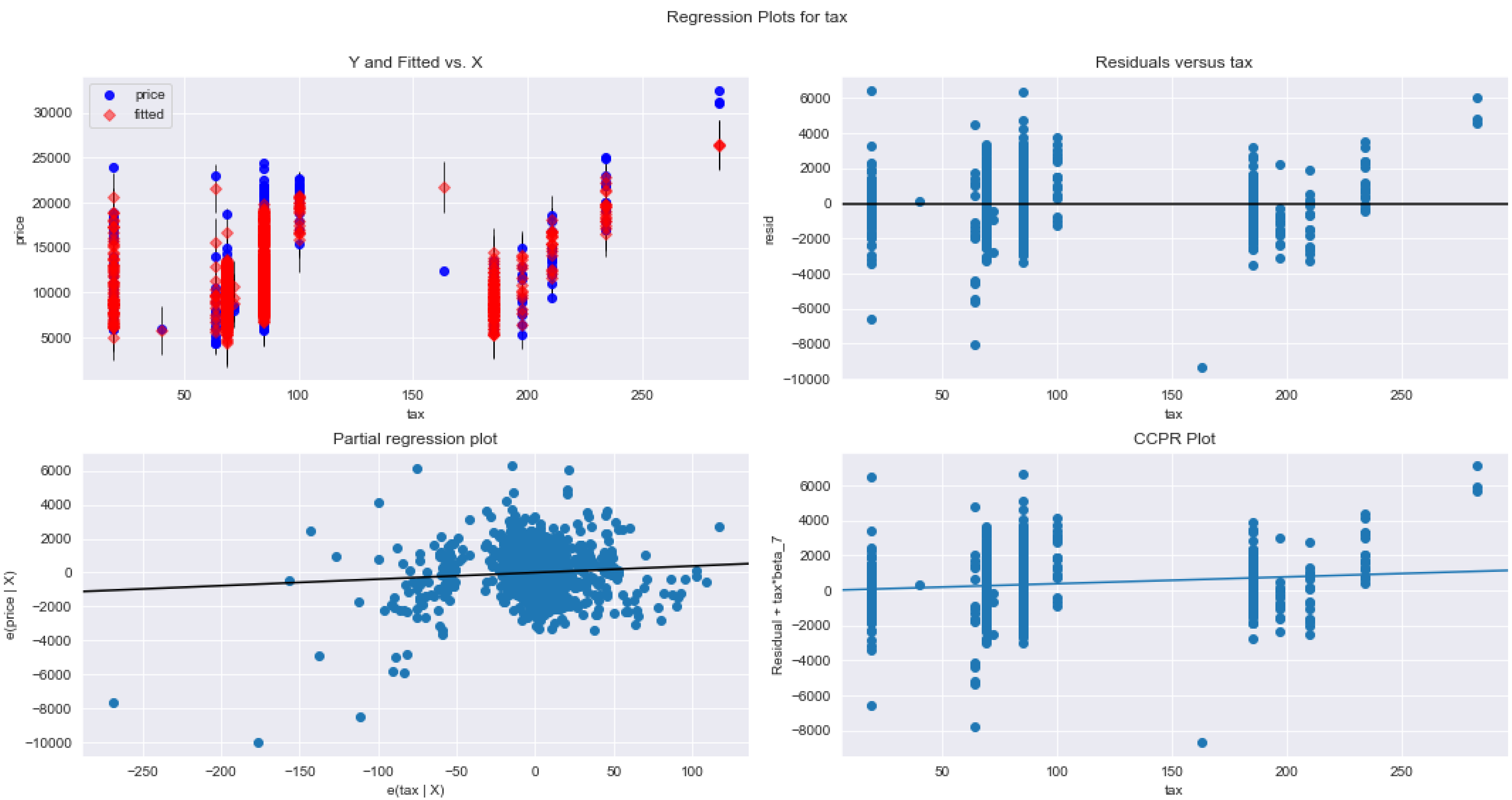



Regression Plots for gear
Y and Fitted vs. X
price
fitted
gear
Residuals versus gear
resid
Partial regression plot
e(price | X)
e(gear | X)
CCPR Plot
Residual + gear*beta_6

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

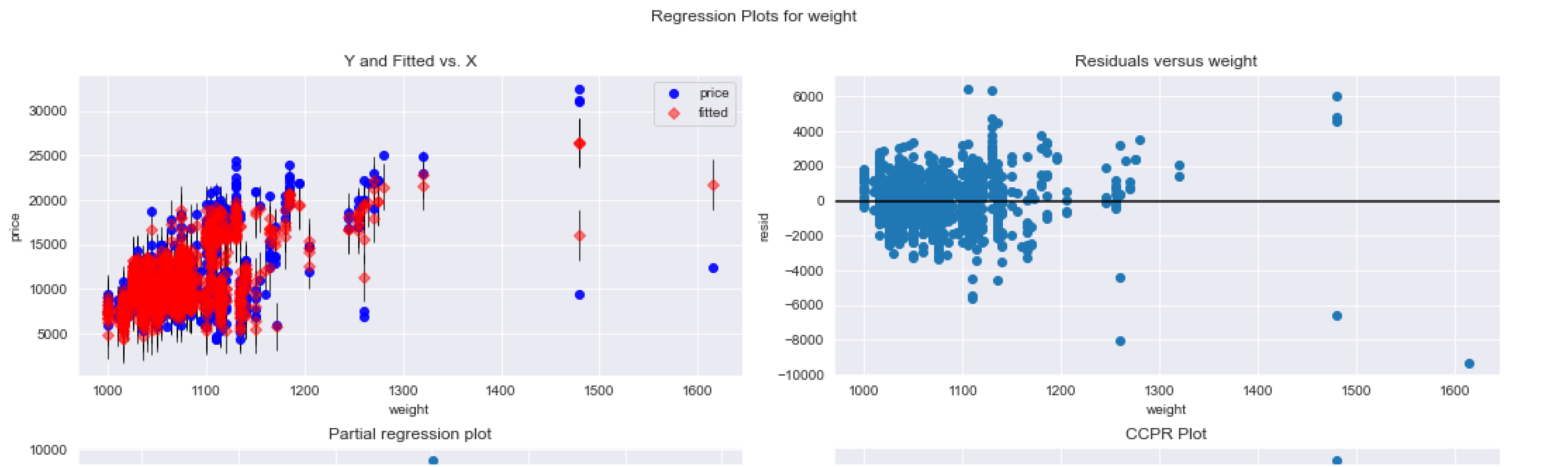
Out[36]:




Regression Plots for tax
Y and Fitted vs. X
price
fitted
price
tax
Residuals versus tax
resid
tax
Partial regression plot
e(price | X)
e(tax | X)
CCPR Plot
Residual + tax*beta_7
tax

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

Out[37]:



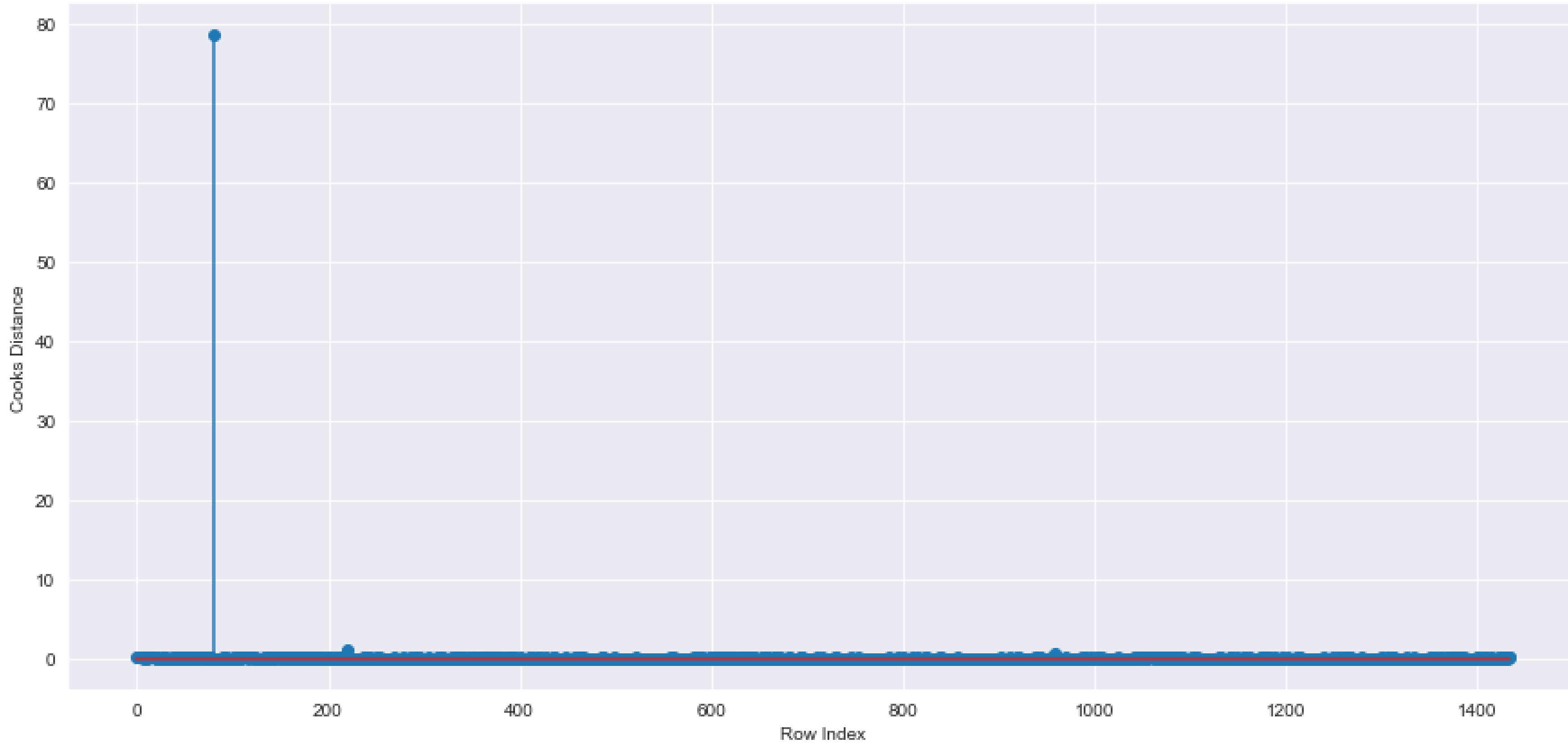
Model deletation Diagnosis(check for outliners and influencer)

Two Techniques :1 Cook's Distance 2.Levrage values

1.Cook's Distance

In [38]:
```
# 1. Cook's Distance: If Cook's distance > 1, then it's an outlier
# Get influencers using cook's distance
model_influence=model.get_influence()
(c,_)=model_influence.cooks_distance
```

In [39]:
```python
# Plot the influencers using the stem plot
fig=plt.figure(figsize=(15,7))
plt.stem(np.arange(len(data)),np.round(c,3))
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```



In [40]:
```python
# Index and value of influencer where C>0.5
np.argmax(c),np.max(c)
```

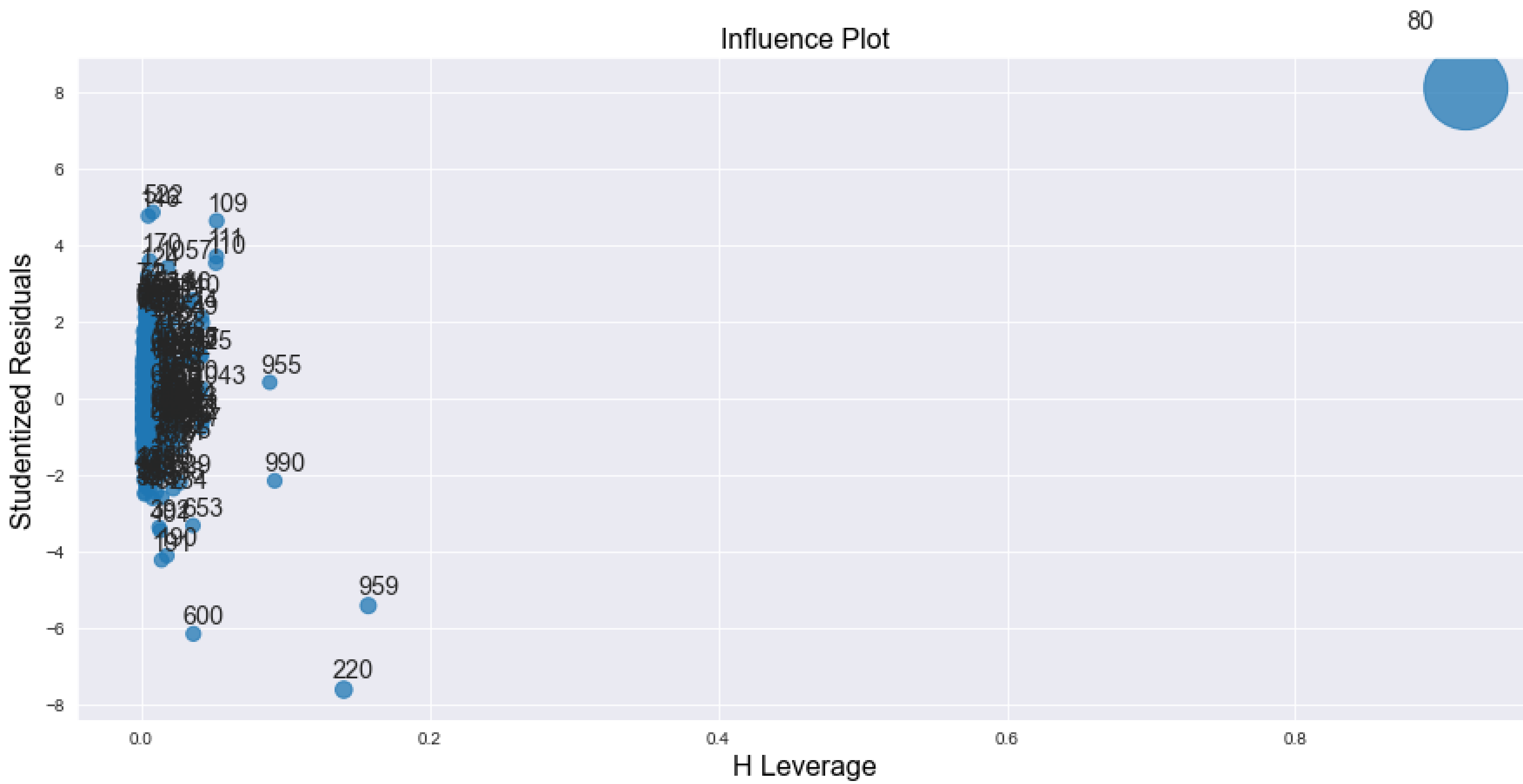
Out[40]:
```
(80, 78.72950582257397)
```

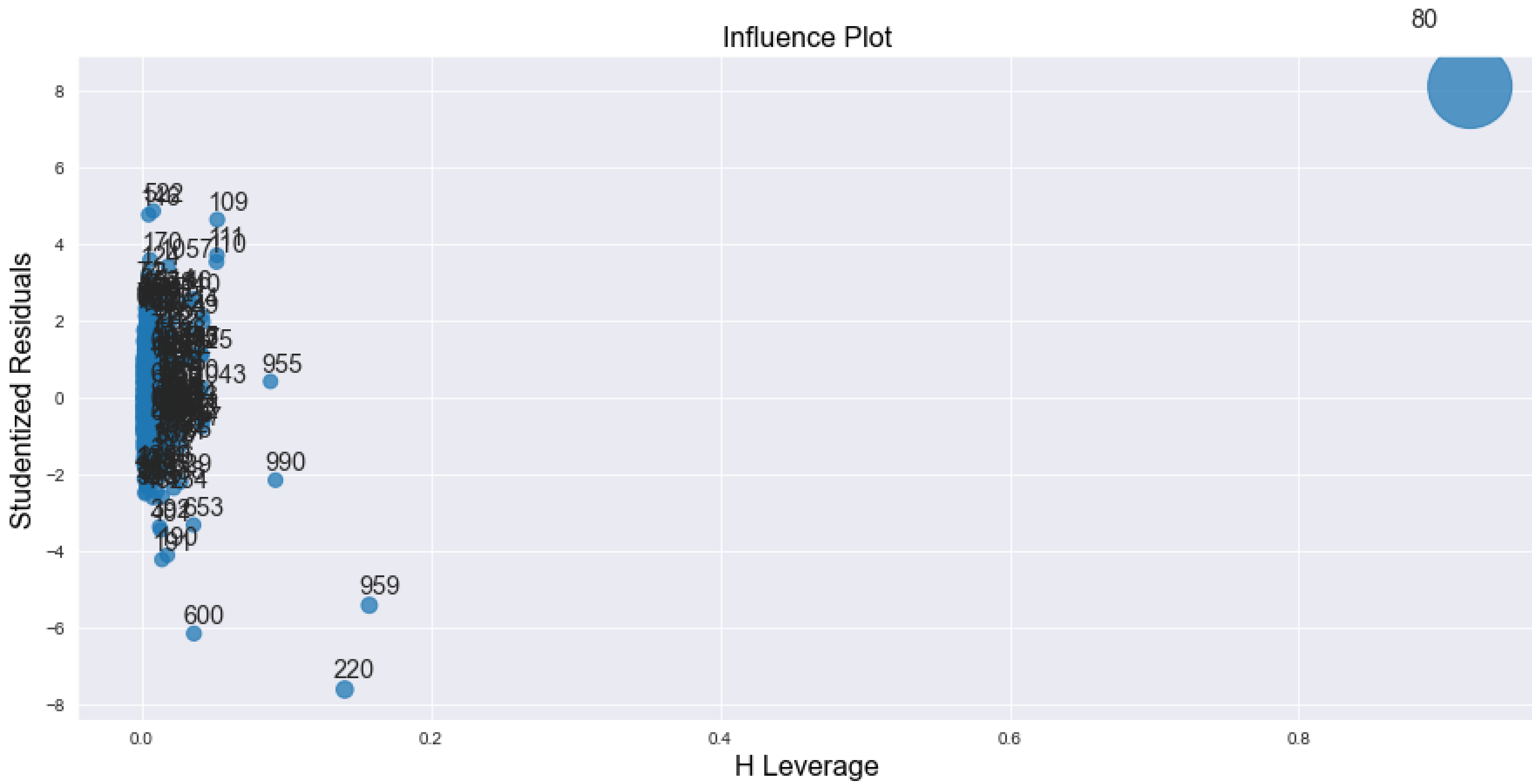
In [41]:
```
#Levrave values
# Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of datapoints
n=data.shape[0]
k=data.shape[1]

l=3*(k+1)/n
l
```

Out[41]:
```
0.020905923344947737
```

In [42]:
```python
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are influencers
from statsmodels.graphics.regressionplots import influence_plot
fig,ax=plt.subplots(figsize=(15,7))
influence_plot(model,fig=fig,ax = ax)
```

Out[42]:





Improving the Model

In [43]:
```
data[data.index.isin([80])]
```

Out[43]:

	price	age	km	hp	cc	door	gear	tax	weight
80	18950	25	20019	110	16000	5	5	100	1180

In [44]:
```
# Discard the data points which are influencers and reassign the row number (reset_index(drop=True))
data1=data.drop(data.index[[80]],axis=0).reset_index(drop=True)
```

In [45]:
```
#Build the model
model1=smf.ols("price~age+km+hp+cc+door+gear+tax+weight",data=data1).fit()
```

In [46]:
```
#rsqaured and aic values
model1.rsquared,model1.aic
```

Out[46]:
```
(0.8681163912634055, 24669.363894157)
```

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

Out[47]:
```
78.72950582257397
```

Model Deletion Diagnostics and Final Model

In [48]:
```
while np.max(c)>0.5:
    model=smf.ols('price~age+km+hp+cc+door+gear+tax+weight',data=data).fit()
    (c,_)=model.get_influence().cooks_distance
    c
    np.argmax(c) , np.max(c)
    data=data.drop(data.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
    data
else:
    final_model=smf.ols('price~age+km+hp+cc+door+gear+tax+weight',data=data).fit()
    final_model.rsquared , final_model.aic
    print("model accuracy is improved to",final_model.rsquared)
```

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

In [49]:
```
final_model.rsquared
```

Out[49]:
```
0.8882395145171204
```

Model Evalution

In [50]:
```
# New data for prediction
new_data=pd.DataFrame({"age":23,"km":72937,"hp":90,"cc":2000,"door":3,"gear":5,"tax":210,"weight":1165},index=[0])
```

In [51]:
```
#Prediction for own dataset
y_new_pred=final_model.predict(new_data)
y_new_pred
```

Out[51]:
```
0    15886.635544
dtype: float64
```

In [52]:
```
#Prediction of model on given datasets
y_pred=model.predict(data)
y_pred
```

Out[52]:
```
0       16326.634426
1       15886.220972
2       16304.093367
3       15973.237208
4       15839.043084
            ...
1426     9114.821644
1427     8499.169594
1428     8644.902871
1429     8758.662855
1430    10638.570082
Length: 1431, dtype: float64
```