Import libraries

In [1]:
```
#Import nessasary 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
from sklearn.preprocessing import LabelEncoder
from statsmodels.graphics.regressionplots import influence_plot
import warnings
warnings.filterwarnings('ignore')
```

Import Dataset

In [2]:
```
#Import dataset
data=pd.read_csv('Computer_Data.csv')
```

In [3]:
```
#Top 5 rows of dataset
data.head()
```

Out[3]:

	Unnamed: 0	price	speed	hd	ram	screen	cd	multi	premium	ads	trend
0	1	1499	25	80	4	14	no	no	yes	94	1
1	2	1795	33	85	2	14	no	no	yes	94	1
2	3	1595	25	170	4	15	no	no	yes	94	1
3	4	1849	25	170	8	14	no	no	no	94	1
4	5	3295	33	340	16	14	no	no	yes	94	1

Data Undestanding

In [4]:
```
#Drop Unwanted Columns
data=data.drop(['Unnamed: 0'],axis=1)
```

In [5]:
```
#Columns list of dataset
data.columns
```

Out[5]:
```
Index(['price', 'speed', 'hd', 'ram', 'screen', 'cd', 'multi', 'premium',
       'ads', 'trend'],
      dtype='object')
```

In [6]:
```
#Number of rows and columns
data.shape
```

Out[6]:
```
(6259, 10)
```

In [7]:
```
#infomation about Any null index
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6259 entries, 0 to 6258
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   price    6259 non-null   int64 
 1   speed    6259 non-null   int64 
 2   hd       6259 non-null   int64 
 3   ram      6259 non-null   int64 
 4   screen   6259 non-null   int64 
 5   cd       6259 non-null   object
 6   multi    6259 non-null   object
 7   premium  6259 non-null   object
 8   ads      6259 non-null   int64 
 9   trend    6259 non-null   int64 
dtypes: int64(7), object(3)
memory usage: 489.1+ KB
```

In [8]:
```
#Data Type of columns
data.dtypes
```

Out[8]:
```
price        int64
speed        int64
hd           int64
ram          int64
screen       int64
cd          object
multi       object
premium     object
ads          int64
trend        int64
dtype: object
```

In [9]:
```
# No duplicated data
data[data.duplicated()]
```

Out[9]:

	price	speed	hd	ram	screen	cd	multi	premium	ads	trend
135	2195	25	245	8	14	no	no	yes	95	2
186	1695	33	170	4	14	no	no	yes	95	2
251	1499	25	170	4	14	no	no	yes	100	3
352	2595	50	250	8	15	no	no	yes	108	4
701	2099	33	120	4	14	no	no	no	176	6
...	...	...	...	...	...	...	...	...	...	...
4780	1945	50	528	8	14	yes	no	yes	162	22
4787	1999	66	420	8	14	yes	yes	yes	162	22
4836	1995	66	528	8	14	yes	no	yes	162	22
4953	1895	66	528	8	14	yes	no	yes	191	23
5885	1699	66	540	8	14	yes	yes	yes	129	29

76 rows × 10 columns

In [10]:
```
#Remove Dublicate index it's impact on accuracy
# Return DataFrame with duplicate rows removed
data=data.drop_duplicates(ignore_index=True)
```

In [11]:
```
data['cd'].value_counts()
```

Out[11]:
```
no     3314
yes    2869
Name: cd, dtype: int64
```

In [12]:
```
data['multi'].value_counts() #Return a Series containing counts of unique values
```

Out[12]:
```
no     5325
yes     858
Name: multi, dtype: int64
```

In [13]:
```
data['premium'].value_counts()  #Return a Series containing counts of unique values
```

Out[13]:
```
yes    5573
no      610
Name: premium, dtype: int64
```

In [14]:
```
data['ram'].value_counts()
```

Out[14]:
```
8     2286
4     2205
16     989
2      390
24     297
32      16
Name: ram, dtype: int64
```

Correlation Metrics

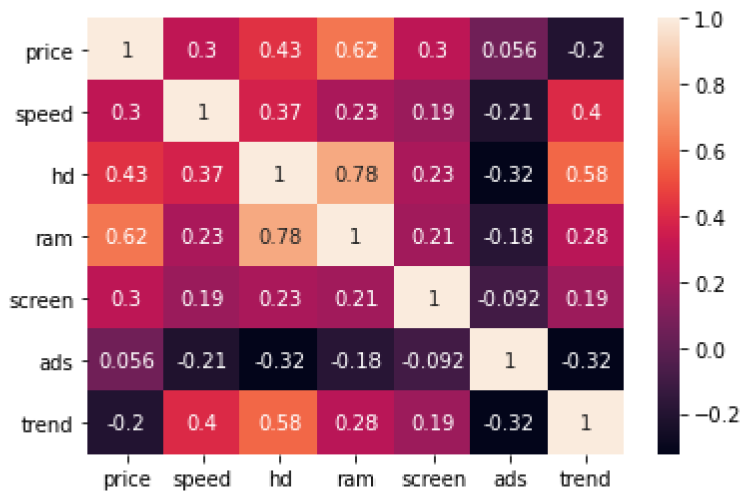
In [15]:
```
#Each cell in the table shows the correlation between two specific variables
data.corr()
```

Out[15]:

	price	speed	hd	ram	screen	ads	trend
price	1.000000	0.298515	0.428845	0.621144	0.295094	0.056434	-0.201662
speed	0.298515	1.000000	0.370356	0.232566	0.187519	-0.214349	0.404830
hd	0.428845	0.370356	1.000000	0.777399	0.232675	-0.323342	0.577599
ram	0.621144	0.232566	0.777399	1.000000	0.208871	-0.181463	0.276938
screen	0.295094	0.187519	0.232675	0.208871	1.000000	-0.092144	0.189549
ads	0.056434	-0.214349	-0.323342	-0.181463	-0.092144	1.000000	-0.320626
trend	-0.201662	0.404830	0.577599	0.276938	0.189549	-0.320626	1.000000

In [16]:
```
#heatmap is a graphical representation of data that uses a system of color-coding to represent different values.
sns.heatmap(data.corr(),annot=True)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x155fc5196d0>



Convert Categorical Variable Into Dummy Variables

In [17]:
```
#Get_dummies() function is used to convert categorical variable into dummy/indicator variables.
data['cd']=pd.get_dummies(data['cd'])
data['multi']=pd.get_dummies(data['multi'])
data['premium']=pd.get_dummies(data['premium'])
```

In [18]:
```
#Top5 rows
data.head()
```

Out[18]:

	price	speed	hd	ram	screen	cd	multi	premium	ads	trend
0	1499	25	80	4	14	1	1	0	94	1
1	1795	33	85	2	14	1	1	0	94	1
2	1595	25	170	4	15	1	1	0	94	1
3	1849	25	170	8	14	1	1	1	94	1
4	3295	33	340	16	14	1	1	0	94	1

Pair Plot

In [19]:
```
# pairplot plot a pairwise relationships in a dataset
sns.set_style('darkgrid')
sns.pairplot(data)
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x155fe5af7f0>

Model Building

```
In [20]: #Model is a Python class that inherits from the Model class
model=smf.ols('price~speed+hd+ram+screen+cd+multi+premium+ads+trend',data=data).fit()
```

In [21]:
```
#Finding rsquared and aic value
model.rsquared ,model.aic
```

Out[21]:
```
(0.7752533525648957, 87064.39431974846)
```

In [22]:
```
#Tvalues and Pvalues
model.tvalues,model.pvalues
```

Out[22]:
```
(Intercept    -0.599167
 speed        50.090678
 hd           28.284181
 ram          44.908486
 screen       30.656010
 cd           -6.327697
 multi        -9.140547
 premium      41.170928
 ads          12.489008
 trend       -82.191337
 dtype: float64,
 Intercept     5.490838e-01
 speed         0.000000e+00
 hd           1.346630e-165
 ram           0.000000e+00
 screen       3.096526e-192
 cd            2.663164e-10
 multi         8.278212e-20
 premium       0.000000e+00
 ads           2.287108e-35
 trend         0.000000e+00
 dtype: float64)
```

Model Validation Techniques

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

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

r_speed=smf.ols('speed~hd+ram+screen+cd+multi+premium+ads+trend',data=data).fit().rsquared
vif_speed=1/(1-r_speed)
print(vif_speed)

r_speed=smf.ols('hd~speed+ram+screen+cd+multi+premium+ads+trend',data=data).fit().rsquared
vif_speed=1/(1-r_speed)
print(vif_speed)

r_ram=smf.ols('ram~hd+speed+screen+cd+multi+premium+ads+trend',data=data).fit().rsquared
vif_ram=1/(1-r_ram)
print(vif_ram)

r_screen=smf.ols('screen~hd+ram+speed+cd+multi+premium+ads+trend',data=data).fit().rsquared
vif_screen=1/(1-r_screen)
print(vif_screen)

r_cd=smf.ols('cd~hd+ram+screen+speed+multi+premium+ads+trend',data=data).fit().rsquared
vif_cd=1/(1-r_cd)
print(vif_cd)

r_multi=smf.ols('multi~hd+ram+screen+cd+speed+premium+ads+trend',data=data).fit().rsquared
vif_multi=1/(1-r_multi)
print(vif_multi)

r_premium=smf.ols('premium~hd+ram+screen+cd+multi+speed+ads+trend',data=data).fit().rsquared
vif_premium=1/(1-r_premium)
print(vif_premium)

r_ads=smf.ols('ads~hd+ram+screen+cd+multi+premium+speed+trend',data=data).fit().rsquared
vif_ads=1/(1-r_ads)
print(vif_ads)

r_trend=smf.ols('trend~hd+ram+screen+cd+multi+premium+ads+speed',data=data).fit().rsquared
vif_trend=1/(1-r_trend)
print(vif_trend)

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

```
1.263185315103809
```

```
4.19456145637253
2.9694858495927385
1.0812846072205307
1.8583021477752344
1.2890982451255026
1.1114549942404264
1.2183895193741623
2.0242598264145606
```

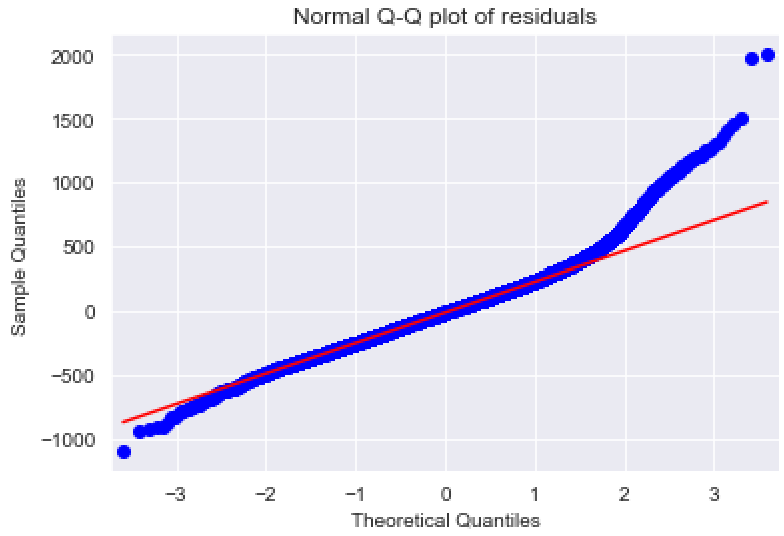
In [24]:
```
# Putting the values in Dataframe format
d1={'Variables':['speed', 'hd', 'ram', 'screen', 'cd', 'multi', 'premium','ads', 'trend'],
    'VIF':[1.26,4.19,2.96,10.8,1.85,1.28,1.11,1.21,2.02]}
d1=pd.DataFrame(d1)
d1
```

Out[24]:

	Variables	VIF
0	speed	1.26
1	hd	4.19
2	ram	2.96
3	screen	10.80
4	cd	1.85
5	multi	1.28
6	premium	1.11
7	ads	1.21
8	trend	2.02

Q-Qplot

In [25]:
```python
# 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 [26]:
```python
# outliar detection from above QQ plot of residuals
list(np.where(model.resid<-1000))
```

Out[26]: [array([79], dtype=int64)]

Residual plot for homoscadacity

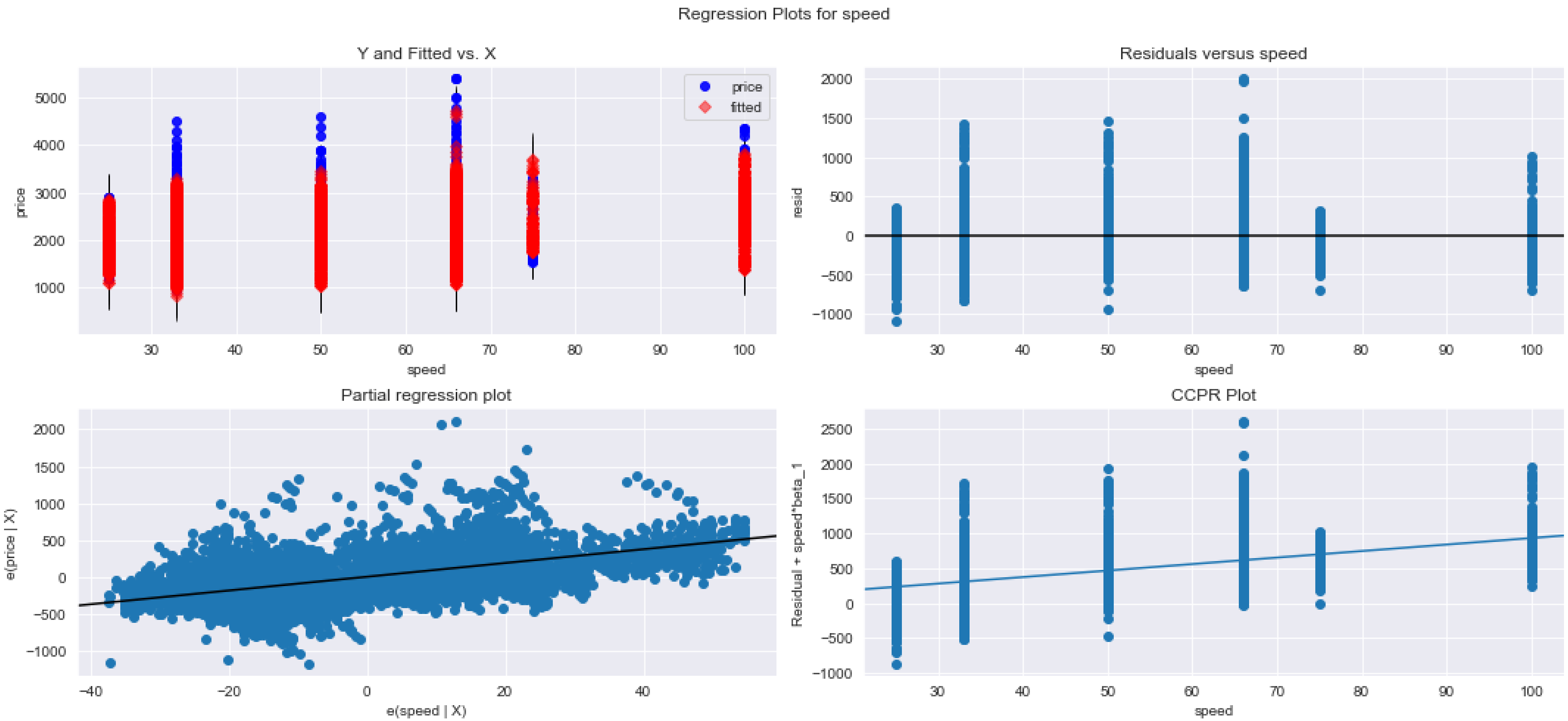
In [27]:
```
# Test for Homoscedasticity or Heteroscedasticity (plotting model's standardized fitted values vs standardized residual values)
## User defined z = (x - mu)/sigma
def get_norm(val):
    return (val-val.mean())/val.std()
```

In [28]:
```
plt.scatter(get_norm(model.fittedvalues),get_norm(model.resid))
plt.title('Residual Plot')
plt.xlabel('standardized fitted values')
plt.ylabel('standardized residual values')
plt.show()
```

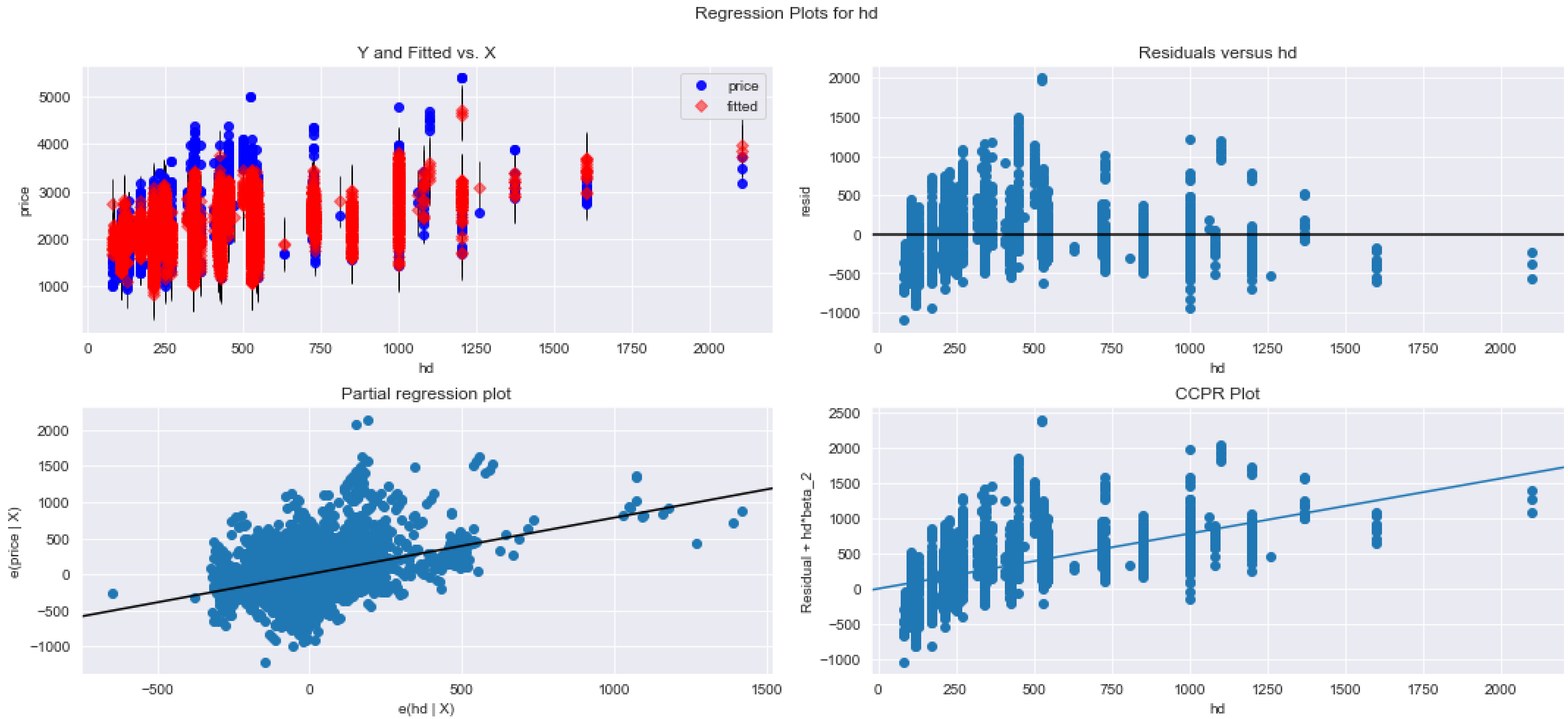
Residual Regression Plots

In [29]:
```
# 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
```

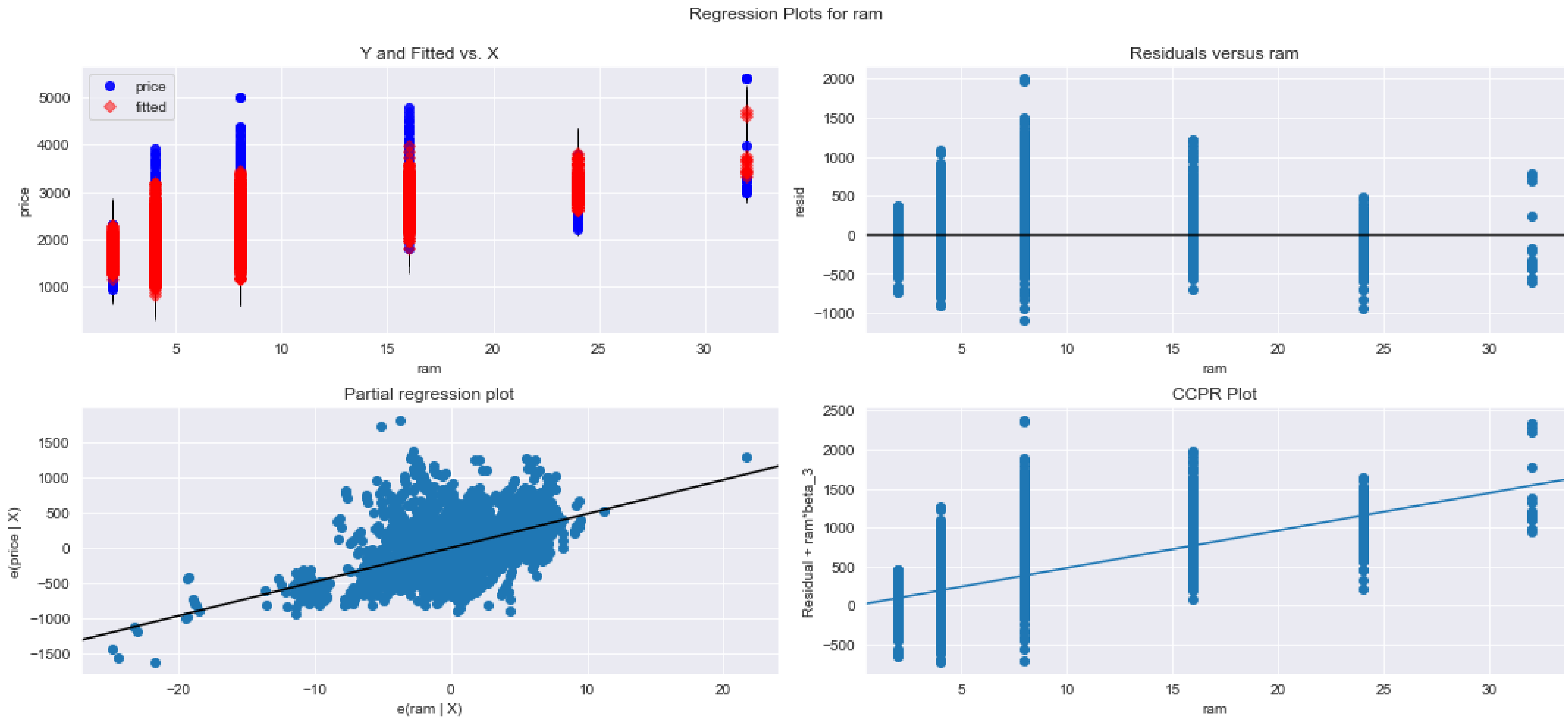
In [30]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'speed',fig=fig)
plt.show()
```



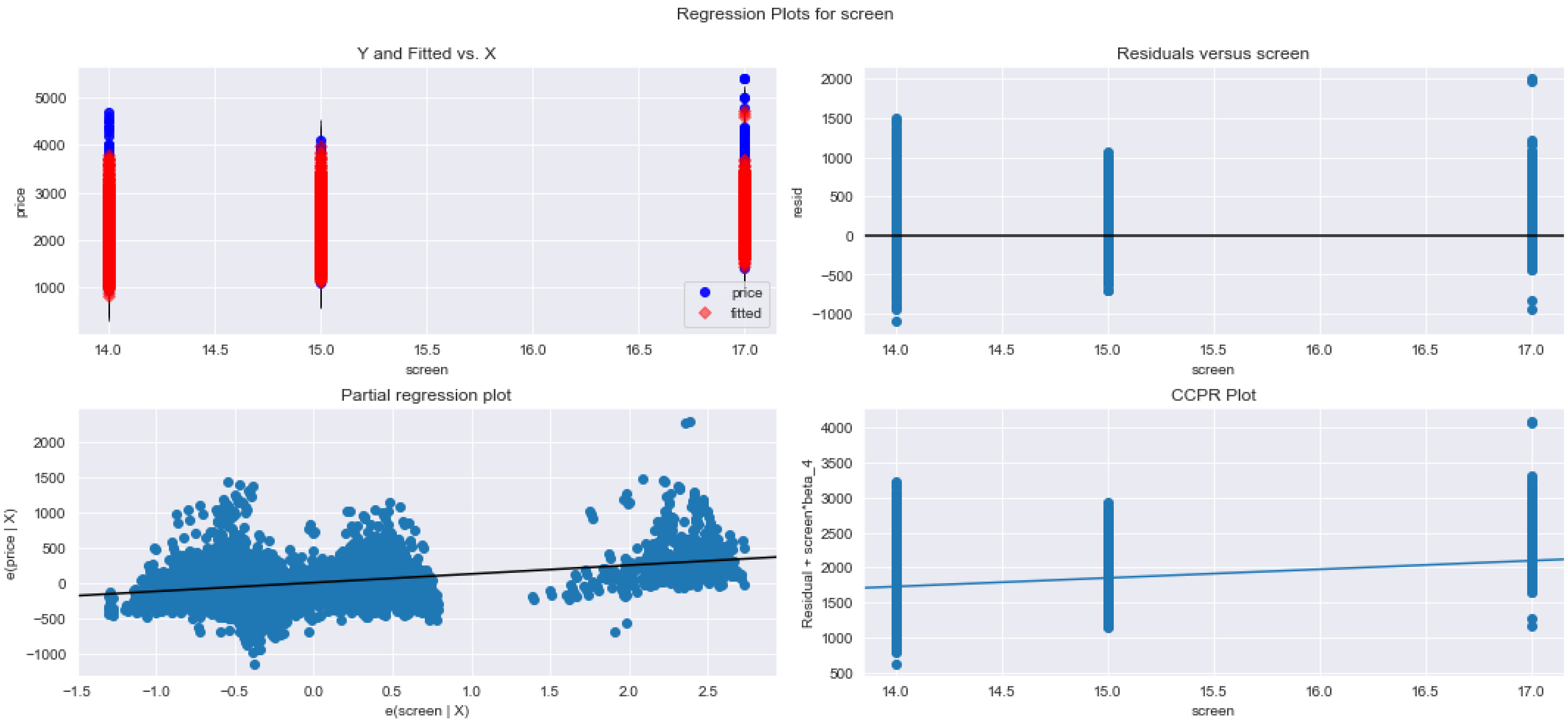
In [31]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'hd',fig=fig)
plt.show()
```



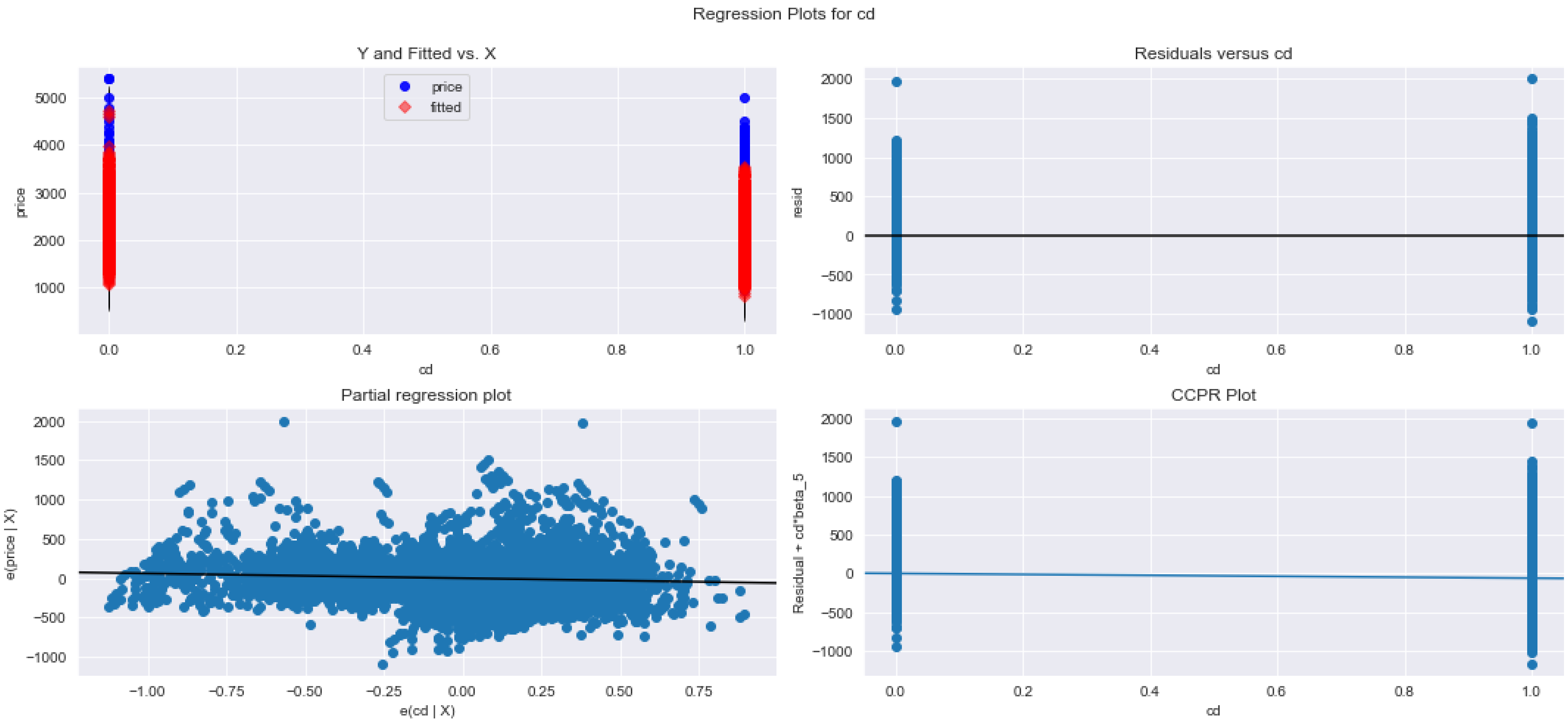
In [32]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'ram',fig=fig)
plt.show()
```



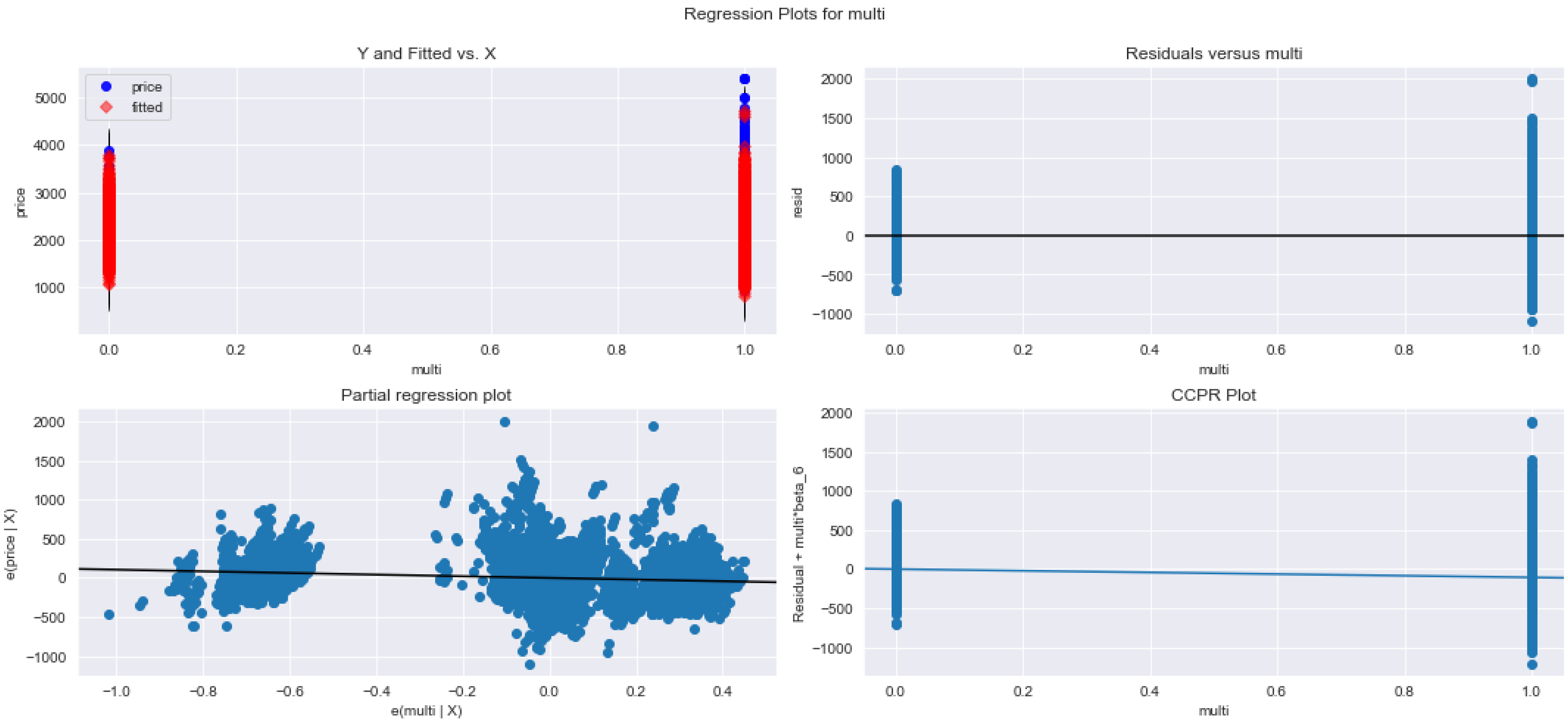
In [33]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'screen',fig=fig)
plt.show()
```



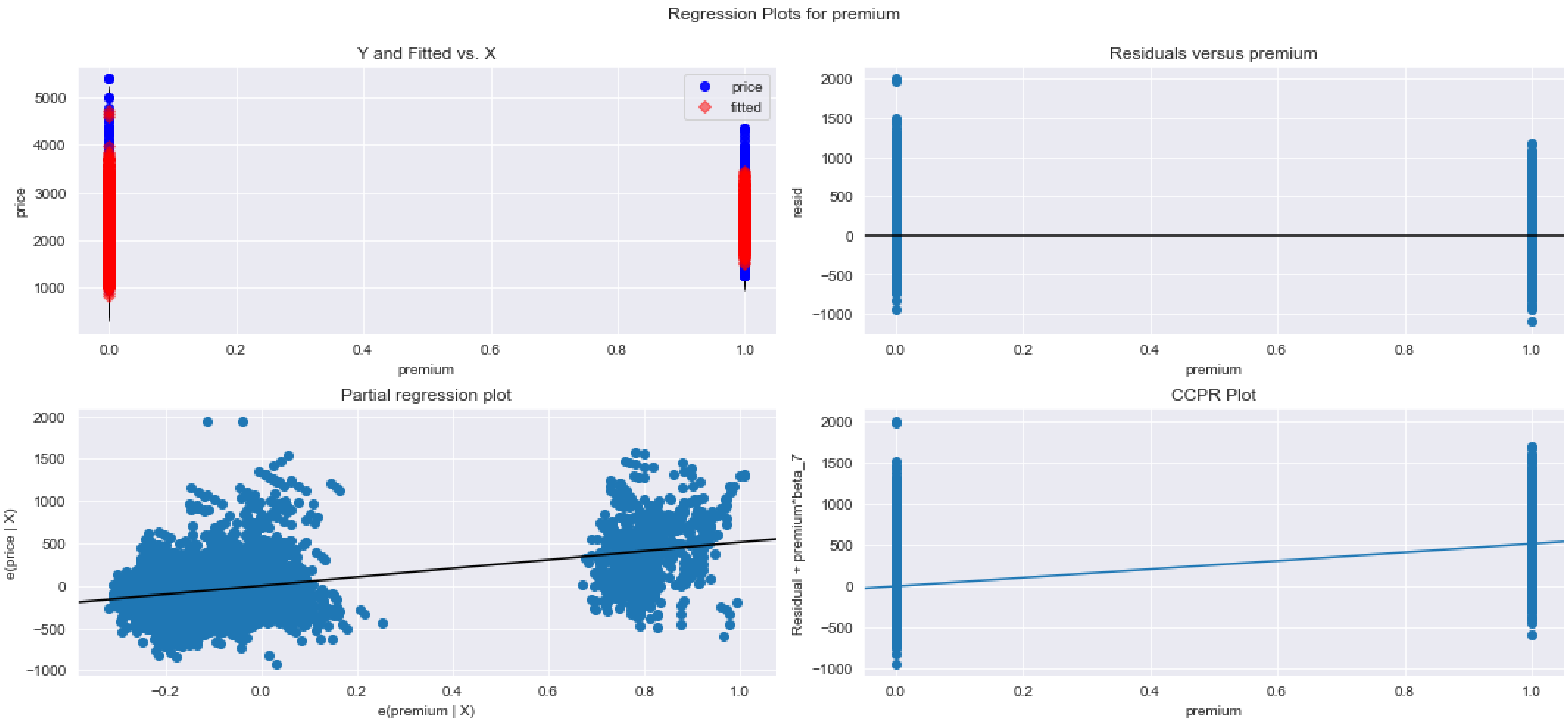
In [34]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'cd',fig=fig)
plt.show()
```



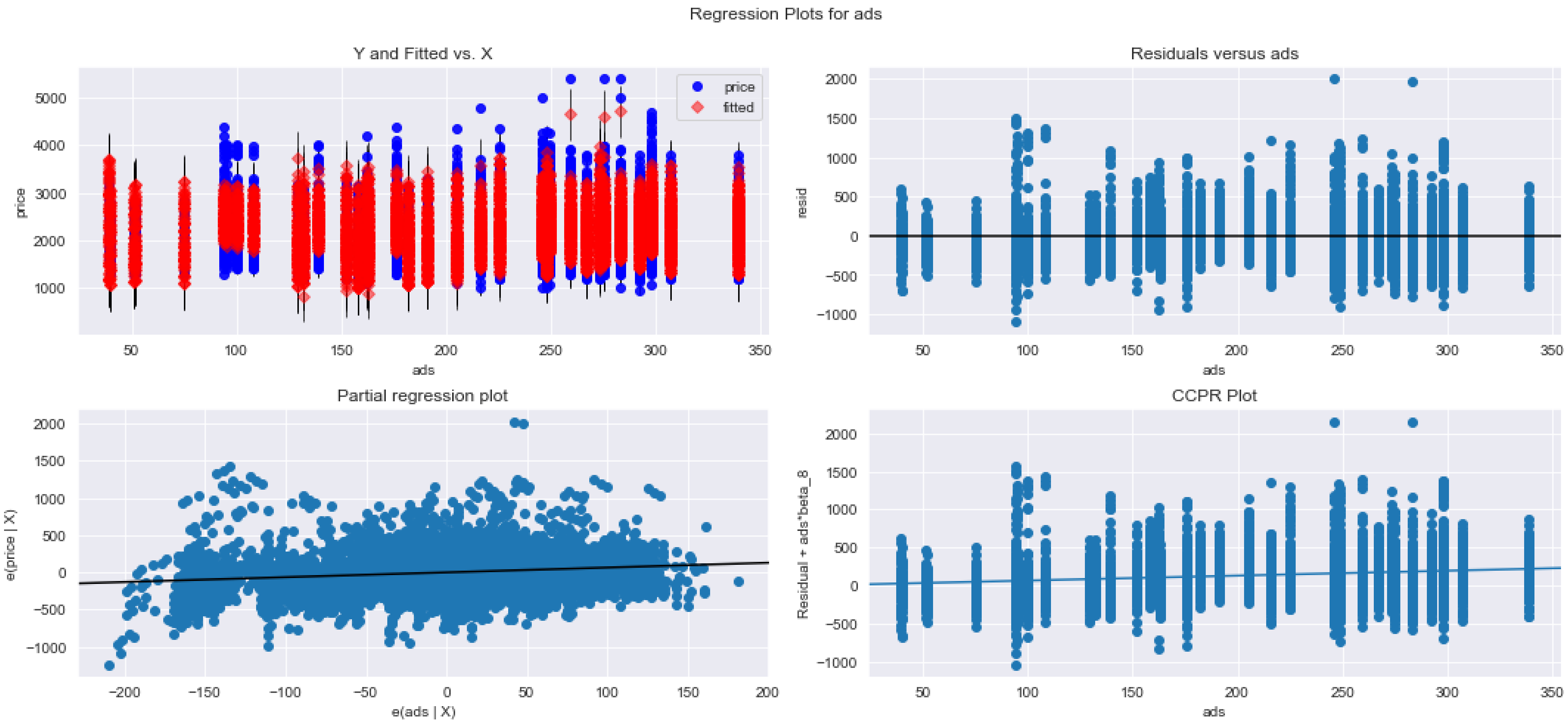
In [35]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'multi',fig=fig)
plt.show()
```



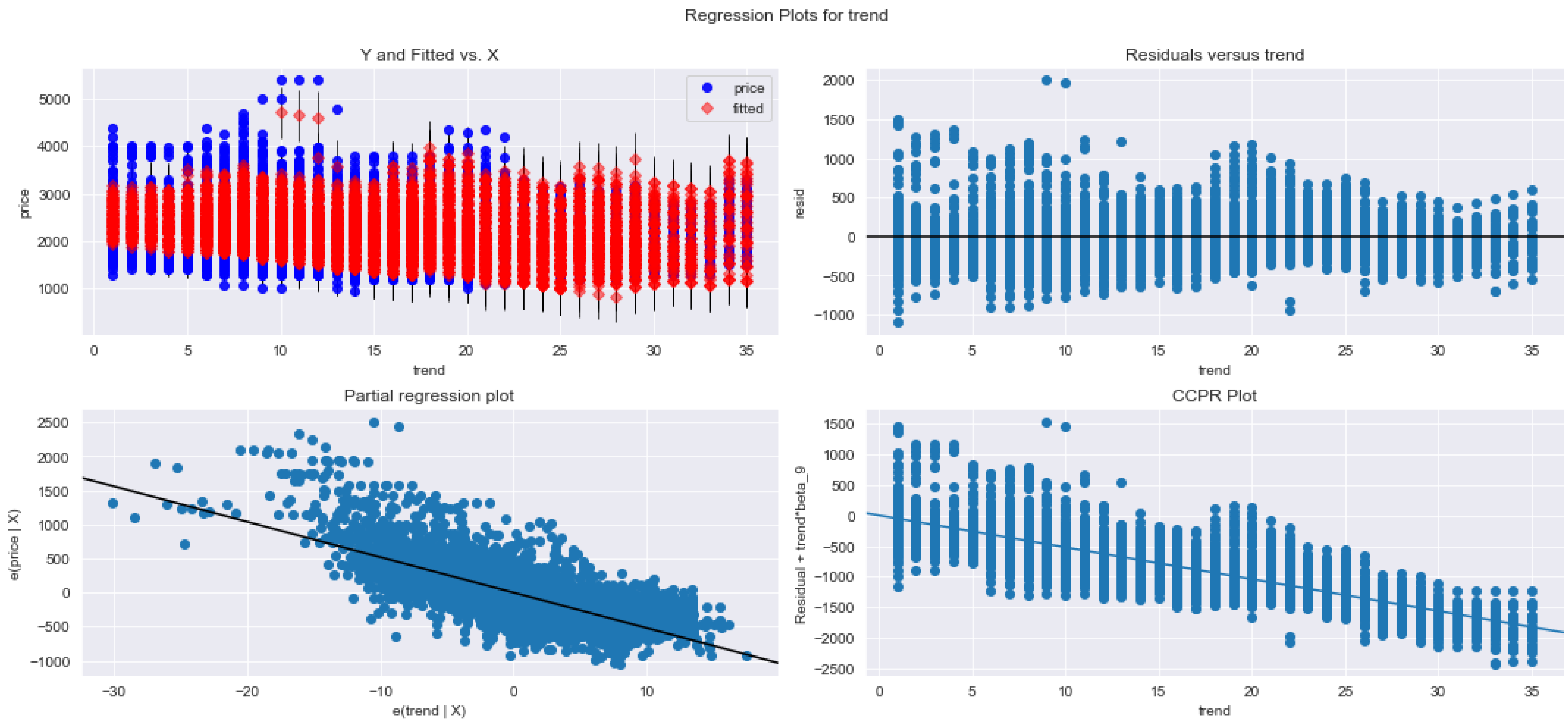
In [36]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'premium',fig=fig)
plt.show()
```



In [37]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'ads',fig=fig)
plt.show()
```



In [38]:
```
fig=plt.figure(figsize=(15,7))
sm.graphics.plot_regress_exog(model,'trend',fig=fig)
plt.show()
```

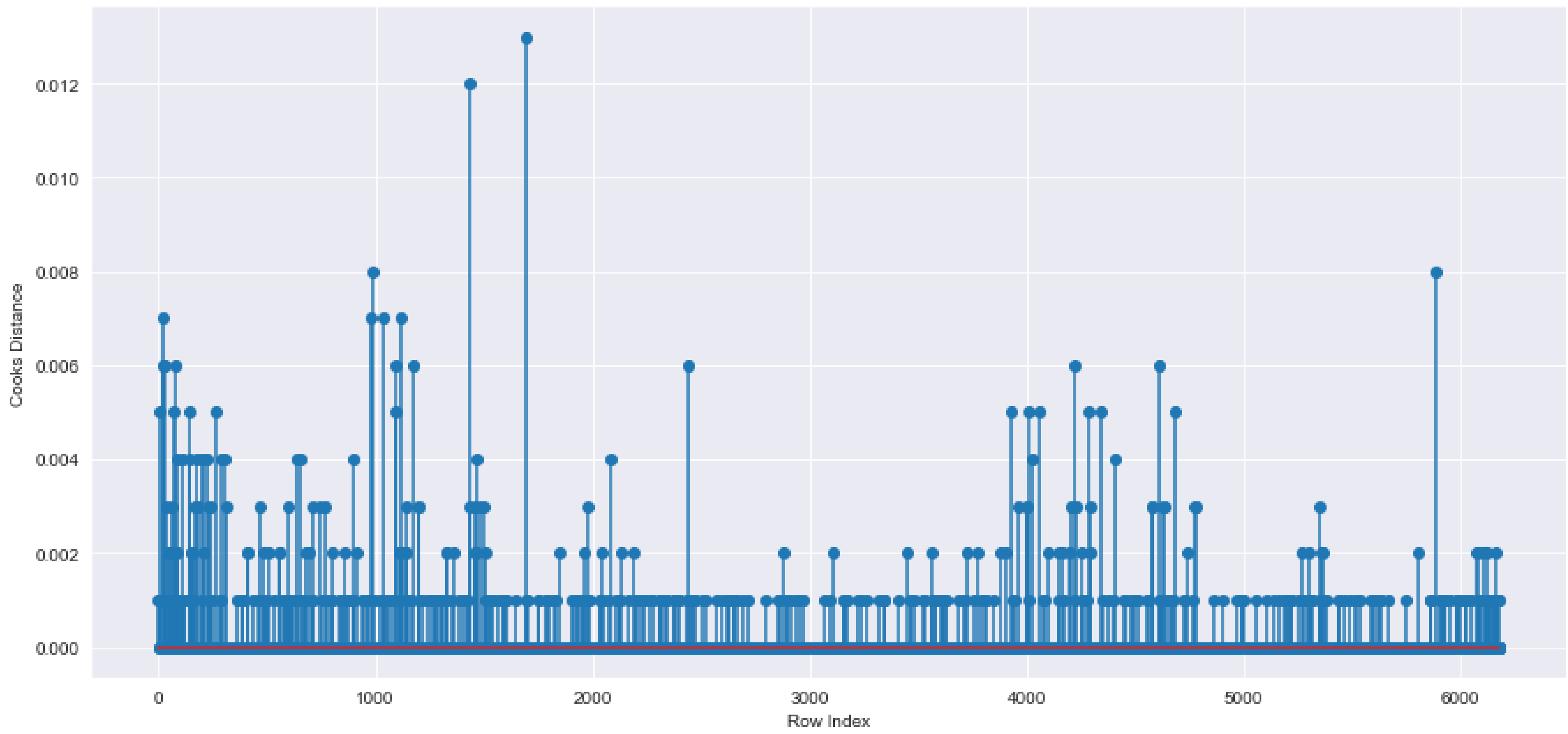


Model Deletion Diagnostics (checking Outliers or Influencers)

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

In [39]:
```
# 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 [40]:
```
# 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()
```

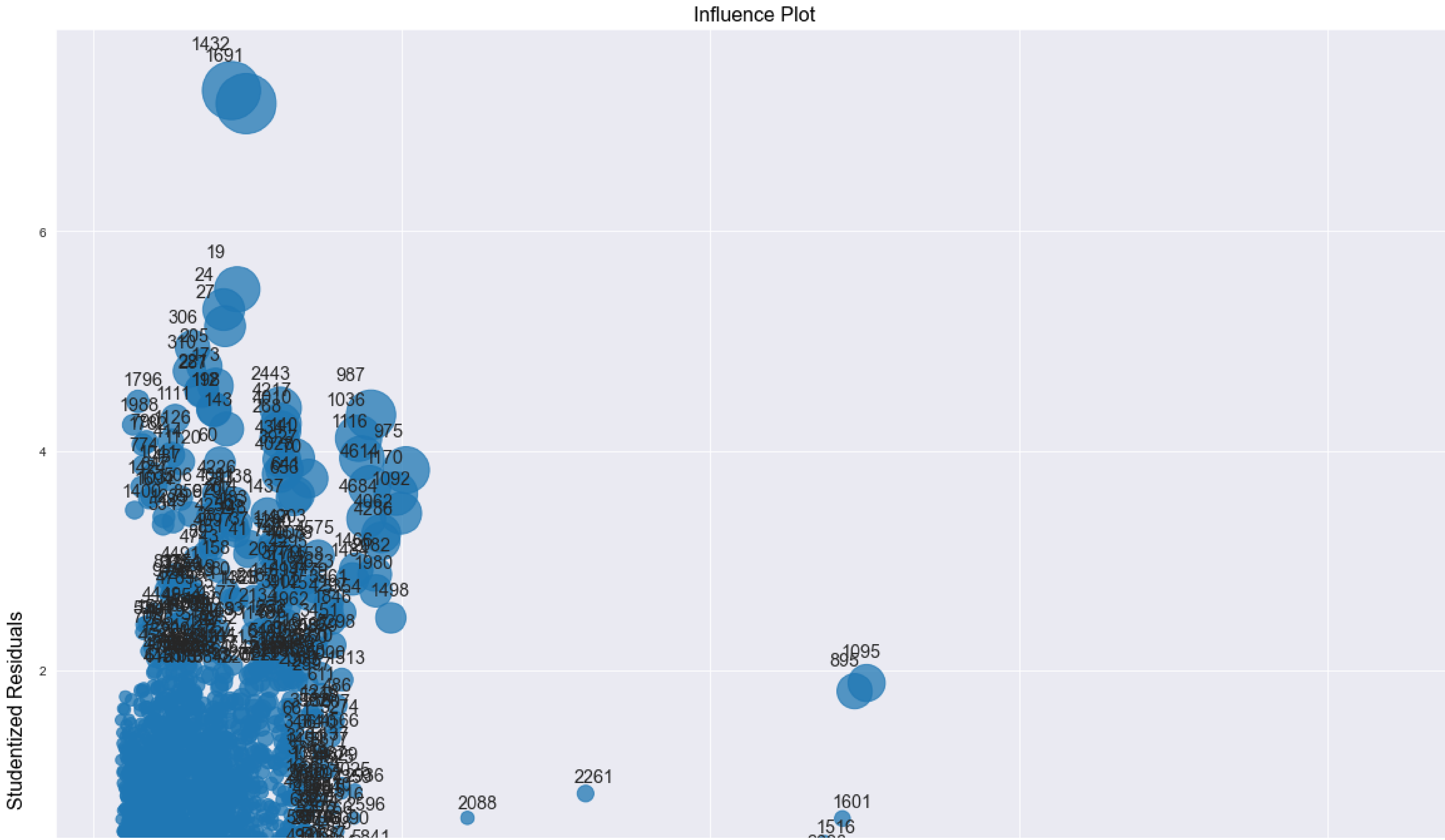


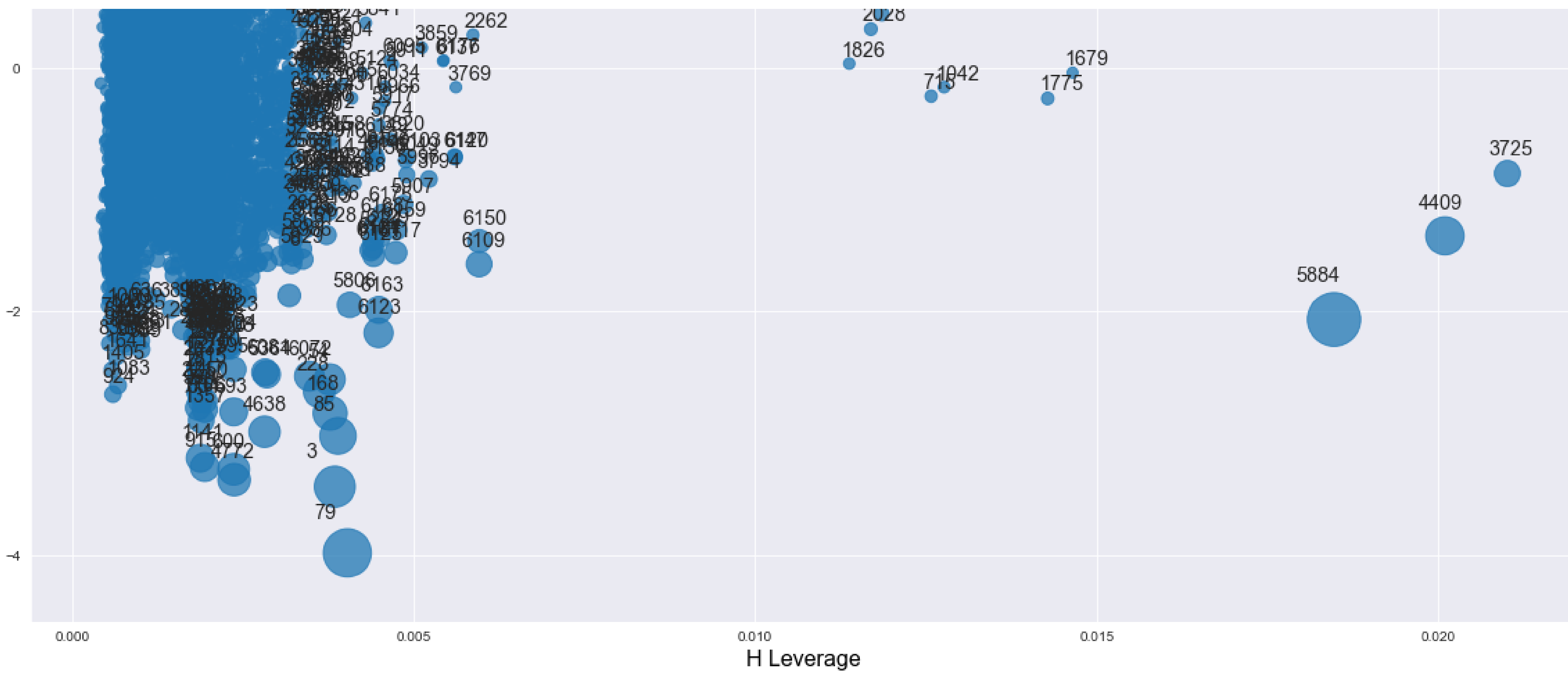
In [41]:
```
np.argmax(c),np.max(c)
```

Out[41]:
```
(1691, 0.012643124146408244)
```

2. Leverage value

In [42]:
```python
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are influencers
fig,ax=plt.subplots(figsize=(20,20))
fig=influence_plot(model,ax = ax)
```





In [43]:
```python
data[data.index.isin([1691])]
```

Out[43]:

	price	speed	hd	ram	screen	cd	multi	premium	ads	trend
1691	4999	66	525	8	17	0	1	0	283	10

In [44]:
```python
# Discard the data points which are influencers and reassign the row number (reset_index(drop=True))
data1=data.drop(data.index[[1691]],axis=0).reset_index(drop=True)
```

Improving the Model

In [45]:
```
model1=smf.ols('price~speed+hd+ram+screen+cd+multi+premium+ads+trend',data=data1).fit()
```

Model Deletion Diagnostics and Final Model

In [46]:
```
while np.max(c)>0.5:
    model=smf.ols('price~speed+hd+ram+screen+cd+multi+premium+ads+trend',data=data).fit()
    model_influence=model.get_influence()
    (c,_)=model_influence.cooks_distance
    np.argmax(c),np.max(c)
    data1=data.drop(data.index[[np.argmax(c)]],axis=0).reset_index(drop=True)
else:
    final_model=smf.ols('price~speed+hd+ram+screen+cd+multi+premium+ads+trend',data=data1).fit()
    final_model.rsquared , final_model.aic
    print("Model accuracy is improved to",final_model.rsquared)
```

```
Model accuracy is improved to 0.7762822418399424
```

In [47]:
```
accuracy=pd.DataFrame({"Model":77.52,"Final_model":77.62},index=[0])
accuracy
```

Out[47]:

	Model	Final_model
0	77.52	77.62

Model Predictions

In [48]:
```
#Predict new data
new_data=pd.DataFrame({'speed':25,'hd':80, 'ram':4,'screen':14, 'cd':1, 'multi':1, 'premium':0,'ads':94,'trend':1},index=[0])
```

In [49]:
```
#Predict on new data
y_pred_new=final_model.predict(new_data)
y_pred_new
```

Out[49]:
```
0    2022.513512
dtype: float64
```

In [50]:
```
#Predict on datasets
y_pred=final_model.predict(data)
y_pred
```

Out[50]:
```
0       2022.513512
1       2004.459897
2       2215.216120
3       2797.163311
4       2879.323256
           ...
6178    1588.512811
6179    2074.857717
6180    2945.014039
6181    2285.522422
6182    2530.327764
Length: 6183, dtype: float64
```