Import Libraries

In [1]:
```
#Import Libraries
import pandas as pd
import numpy as np
import warnings
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
from statsmodels.graphics.regressionplots import influence_plot
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score
```

Import Dataset

In [2]:
```
#Import Dataset
data=pd.read_csv("50_Startups.csv")

#Read Top 5 rows
data.head()
```

Out[2]:

	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

Data Undestanding

In [3]:
```
#Rename columns name
data=data.rename({'R&D Spend':'RDS','Administration':'ADMS','Marketing Spend':'MKTS'},axis=1)
data.head()
```

Out[3]:

	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

In [4]:
```
#number of columns and rows
data.shape
```

Out[4]:
```
(50, 5)
```

In [5]:
```
#check the dataset features datatypes
#Encode  any categorical data to numerical data
data.dtypes
```

Out[5]:
```
RDS        float64
ADMS       float64
MKTS       float64
State       object
Profit     float64
dtype: object
```

In [6]:
```
data.isnull().sum() # to check if there are any missing values
```

Out[6]:
```
RDS       0
ADMS      0
MKTS      0
State     0
Profit    0
dtype: int64
```

In [7]:
```
#To see the statistics
data.describe()
```

Out[7]:

	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

In [8]:
```
#No Dublicate data
data[data.duplicated()]
```

Out[8]:

RDS	ADMS	MKTS	State	Profit

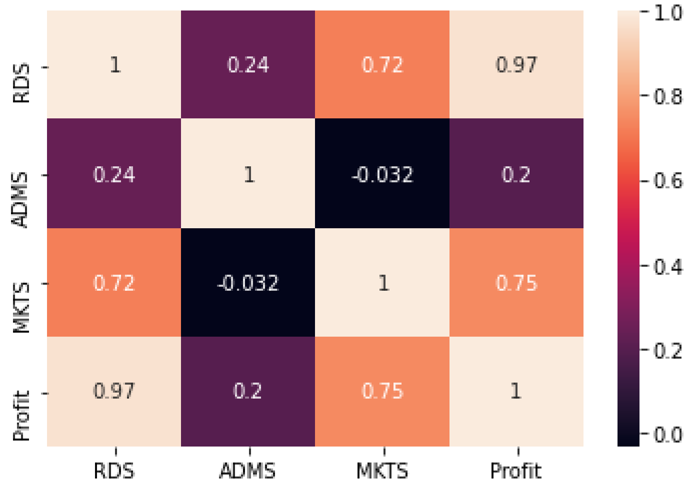
In [9]:
```
#correlation between columns create issue in prediction
data.corr()
```

Out[9]:

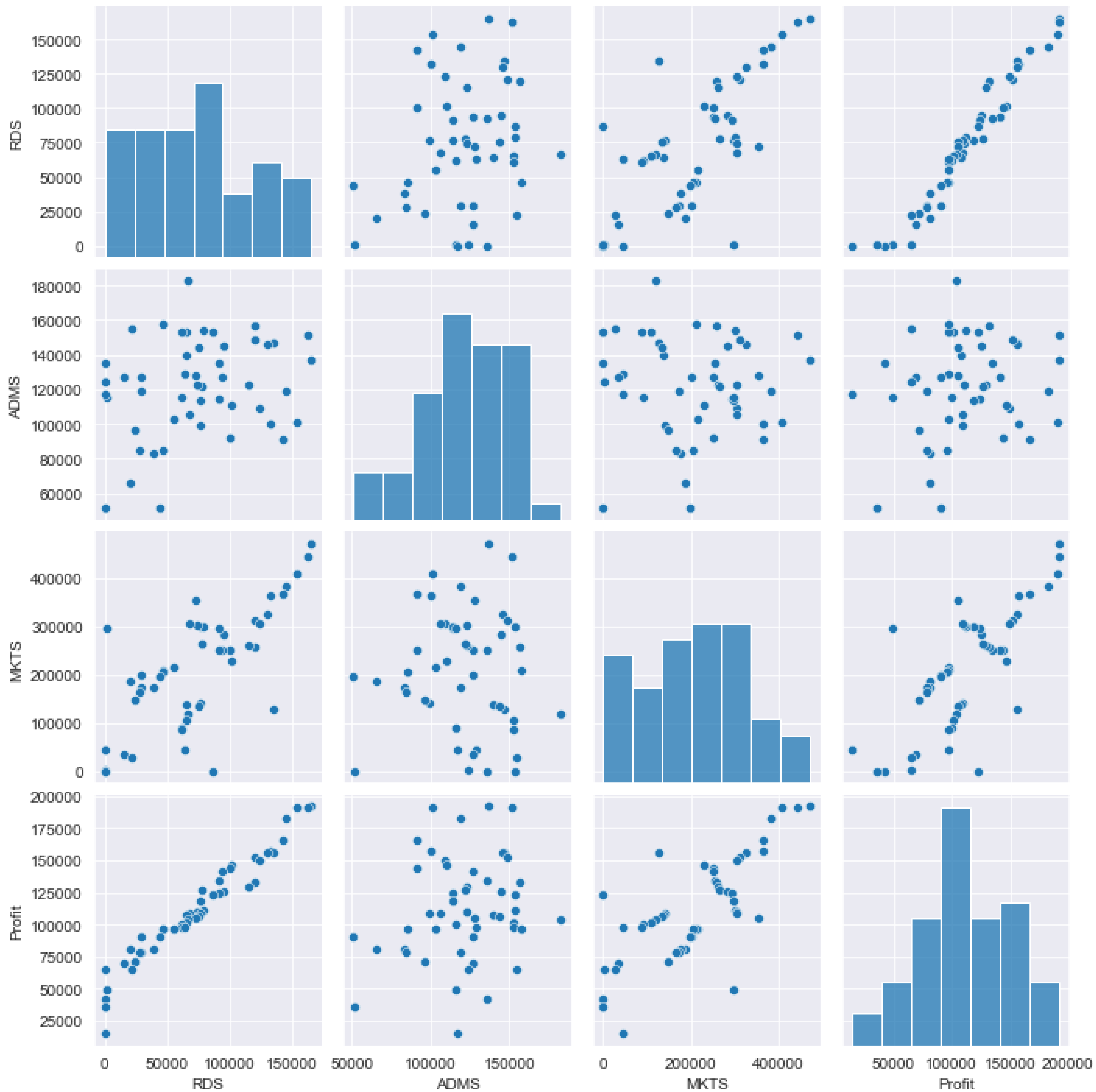
	RDS	ADMS	MKTS	Profit
RDS	1.000000	0.241955	0.724248	0.972900
ADMS	0.241955	1.000000	-0.032154	0.200717
MKTS	0.724248	-0.032154	1.000000	0.747766
Profit	0.972900	0.200717	0.747766	1.000000

In [10]:
```
#Correlation visualization
sns.heatmap(data.corr(),annot=True)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x21be651a4f0>



In [11]:
```
#plot pair plot to see distribution of data
sns.set_style(style="darkgrid")
sns.pairplot(data)
plt.show()
```



RDS
ADMS
MKTS
Profit

Model Building

In [12]:
```
#Create Model
model=smf.ols("Profit~RDS+ADMS+MKTS",data=data).fit()
```

Model Testing

In [13]:
```
# Finding Coefficient parameters
model.params
```

Out[13]:
```
Intercept    50122.192990
RDS              0.805715
ADMS            -0.026816
MKTS             0.027228
dtype: float64
```

In [14]:
```
#Finding rsquare and adjust r square values.
model.rsquared,model.rsquared_adj   #From model We Get Accuracy of 94.75%
```

Out[14]:
```
(0.9507459940683246, 0.9475337762901719)
```

In [15]:
```
#tvalues and p values.
np.round(model.tvalues,5),np.round(model.pvalues,5)
```

Out[15]:
```
(Intercept     7.62622
 RDS          17.84637
 ADMS         -0.52551
 MKTS          1.65508
 dtype: float64,
 Intercept    0.00000
 RDS          0.00000
 ADMS         0.60176
 MKTS         0.10472
 dtype: float64)
```

In [16]:
```
#we get multicolinearity issue between ADMS and MKTS
#So We Built individual SLR and MLR model for it
```

In [17]:
```
slr_adms=smf.ols("Profit~ADMS",data=data).fit()
#Insignificant Pvalues
np.round(slr_adms.pvalues,5)
```

Out[17]:
```
Intercept    0.00382
ADMS         0.16222
dtype: float64
```

In [18]:
```
slr_mkts=smf.ols("Profit~MKTS",data=data).fit()
#Significant Pvalues
np.round(slr_mkts.pvalues,5)
```

Out[18]:
```
Intercept    0.0
MKTS         0.0
dtype: float64
```

In [19]:
```
mlr_ma=smf.ols("Profit~MKTS+ADMS",data=data).fit()
#Insignificant Pvalues
np.round(mlr_ma.pvalues,5)
```

Out[19]:
```
Intercept    0.25893
MKTS         0.00000
ADMS         0.01729
dtype: float64
```

Model Validation

1.Colinearity Check 2.Residual analysis

1.Colinearity Check

VIF (Variance inflation factor)

In [20]:
```
#VIF is a measure of the amount of multicollinearity in a set of multiple regression variables.
#calculate vif for all indipendent variables
# Collinearity Problem Check
#Vif=1(/1-Rsquare)

rsq_rds=smf.ols('RDS~ADMS+MKTS',data=data).fit().rsquared
vif_rds=1/(1-rsq_rds)

rsq_adms=smf.ols('ADMS~RDS+MKTS',data=data).fit().rsquared
vif_adms=1/(1-rsq_adms)

rsq_mkts=smf.ols('MKTS~RDS+ADMS',data=data).fit().rsquared
vif_mkts=1/(1-rsq_mkts)

d1={"Variables":["RDS","ADMS","MKTS"],"VIF":[2.4,1.17,2.32]}
df=pd.DataFrame(d1)
df
#No other variable has vif>20 so no colinearity issue.consider all varible in input regression
```

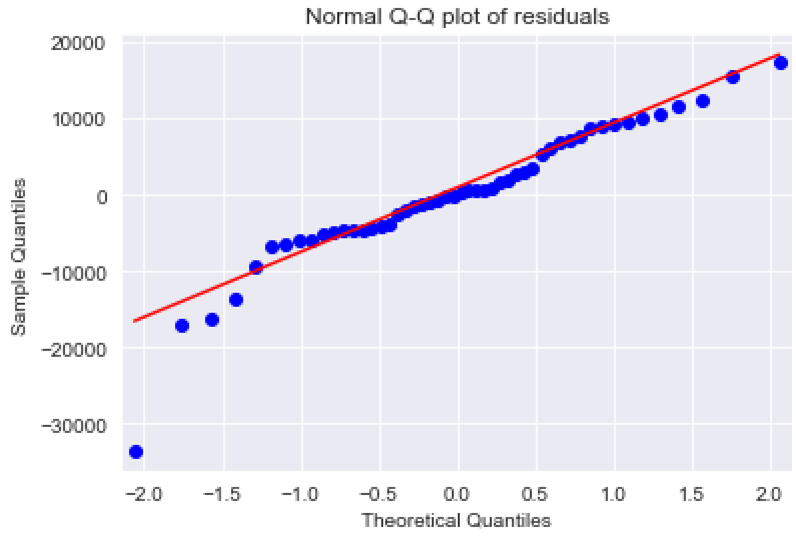
Out[20]:

	Variables	VIF
0	RDS	2.40
1	ADMS	1.17
2	MKTS	2.32

2.Residual analysis

Q-Q Plot

In [21]:
```
# Another Test for Normality of Residual Using Residual analysis
# Q-Q plot is like probability plot
sm.qqplot(data=model.resid,line='q')
plt.title("Normal Q-Q plot of residuals")
plt.show()
```



In [22]:
```
#Some outliner are follow out of this line
list(np.where(model.resid<-30000))
```

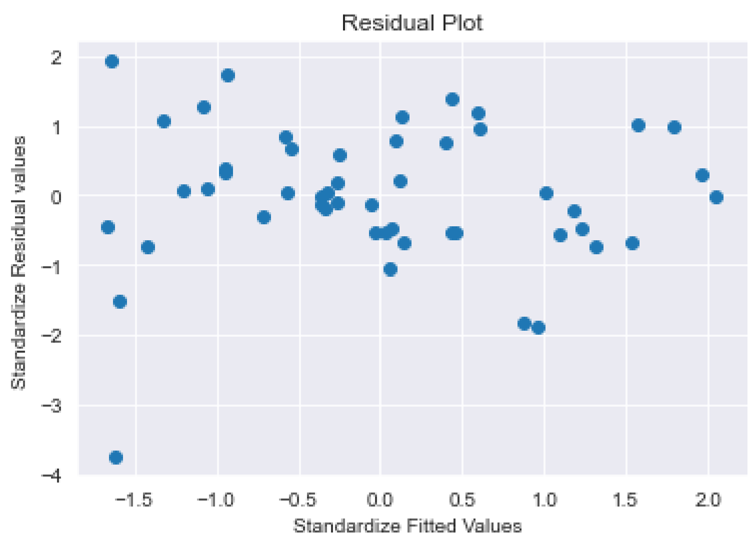
Out[22]:
```
[array([49], dtype=int64)]
```

Homoscadacity test

In [23]:
```
#Test for Homoscedasticity or Heteroscedasticity
def get_norm(var):
    return (var-var.mean())/var.std()   #Use Z equation (x-mu)/sigma
```

In [24]:
```
# plotting model's standardized fitted values vs standardized residual values
plt.scatter(get_norm(model.fittedvalues),get_norm(model.resid))
plt.title('Residual Plot')
plt.ylabel("Standardize Residual values")
plt.xlabel("Standardize Fitted Values ")
```

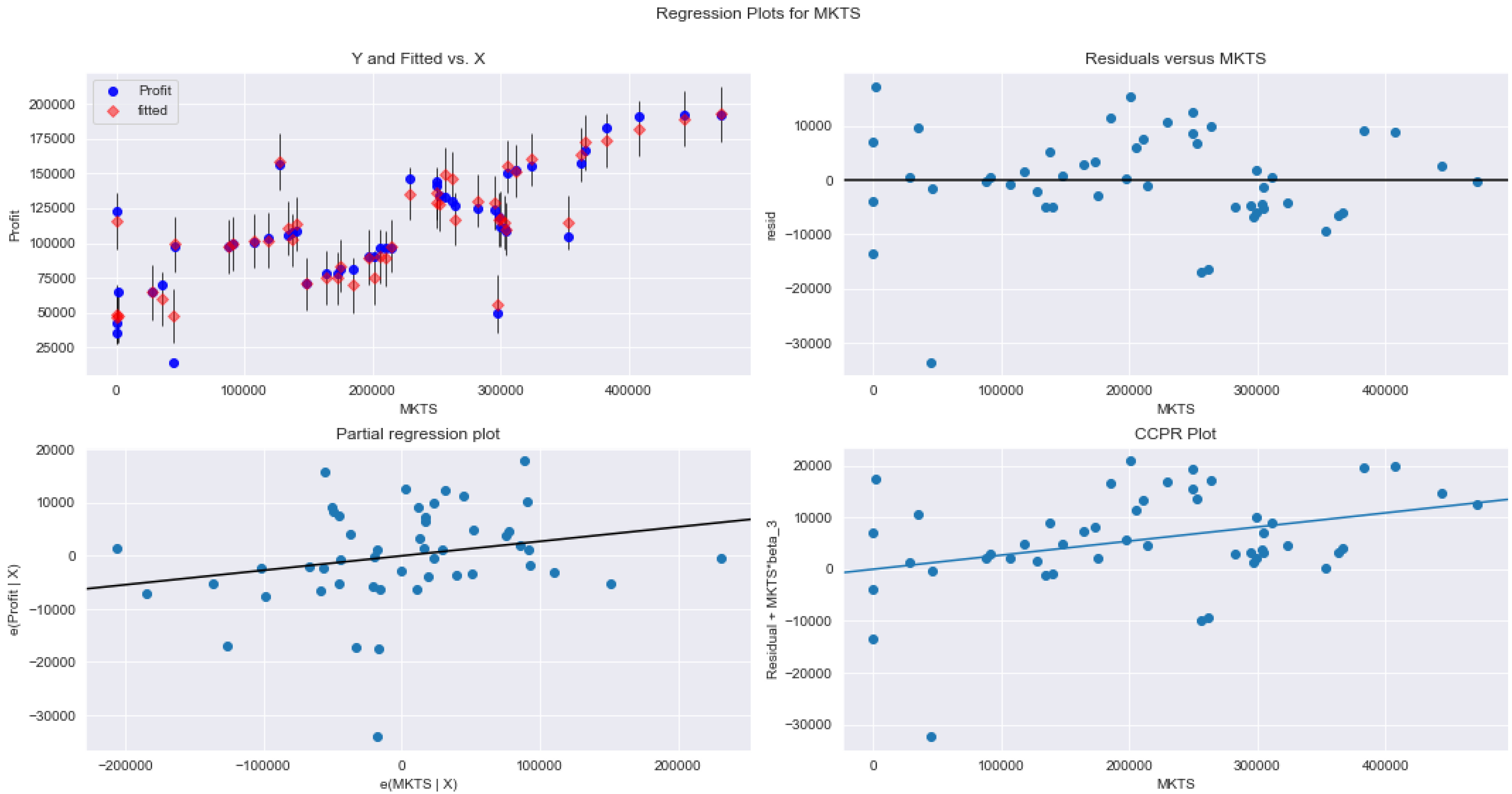
Out[24]:
```
Text(0.5, 0, 'Standardize Fitted Values ')
```



Residual vs Regressor

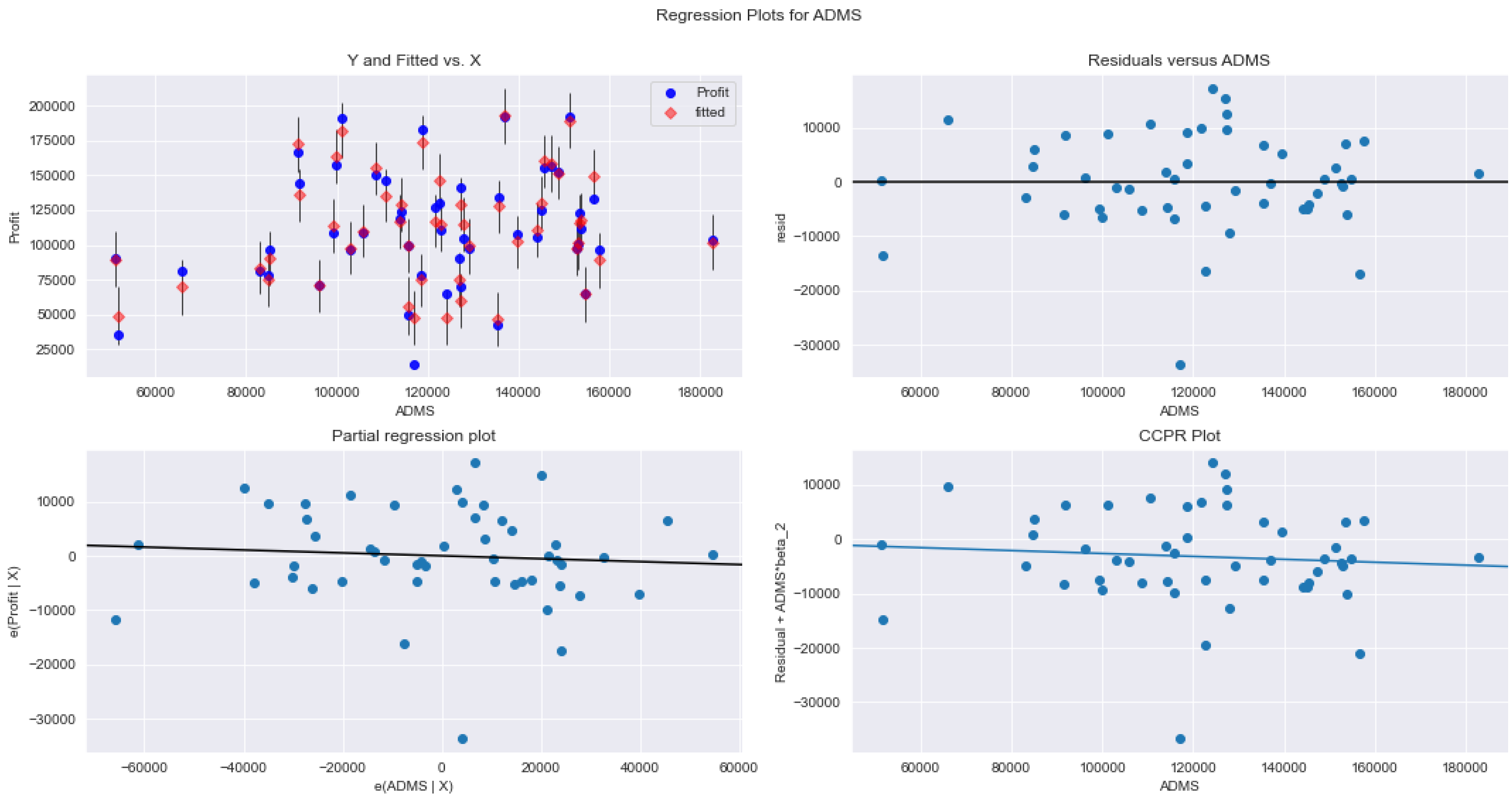
In [25]:
```
# Test for errors or Residuals Vs Regressors or independent 'x' variables or predictors
# using Residual Regression Plots code graphics.plot_regress_exog(model,'x',fig)
# We Take individual input points and plot the graph
 # exog = x-variable & endog = y-variable
```

In [26]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model, "MKTS",fig=fig) #For MKTS
plt.show()
```



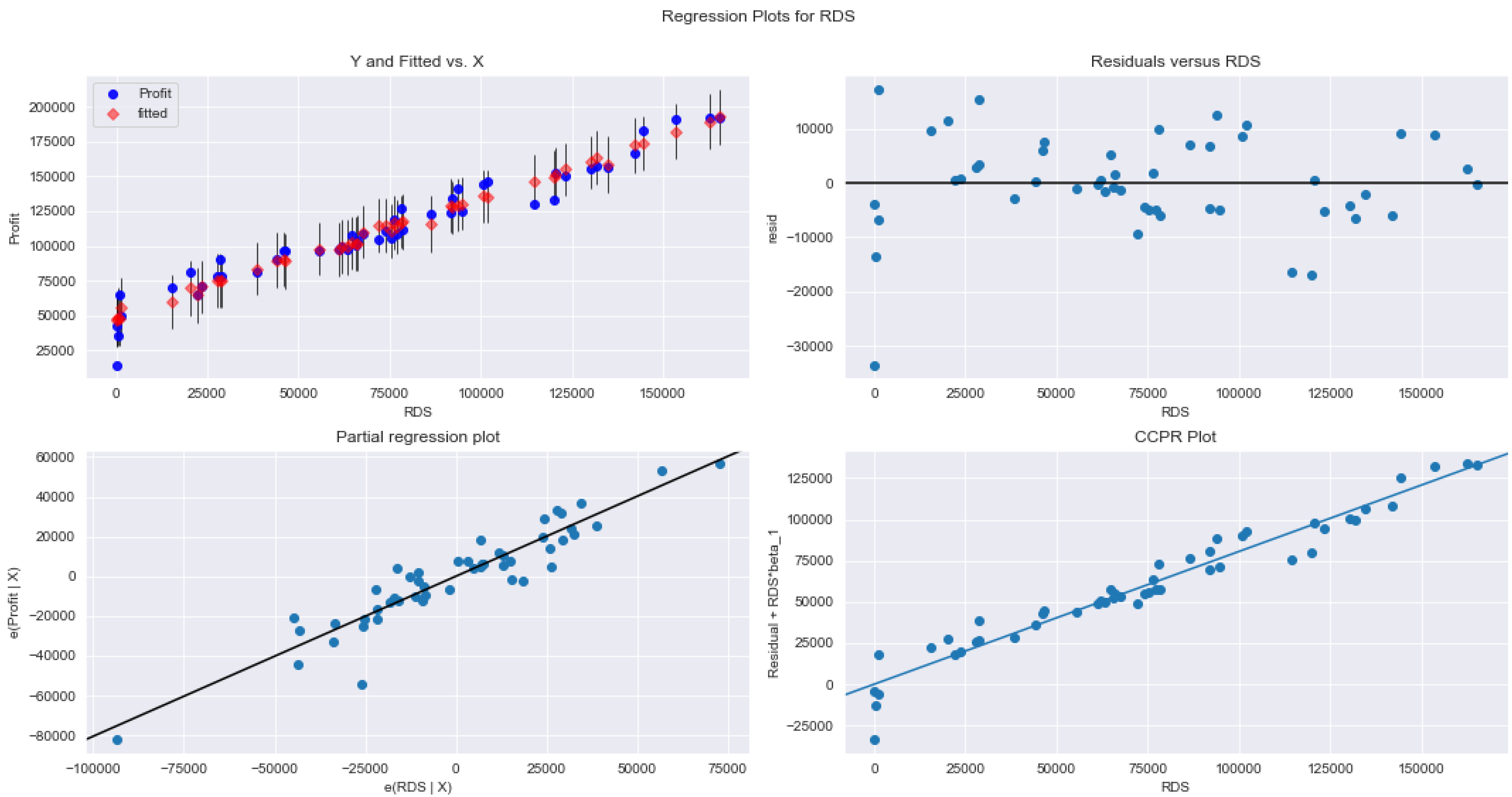
In [27]:

```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model, "ADMS",fig=fig) #For ADMS
plt.show()
```



In [28]:

```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model, 'RDS',fig=fig) #For RDS
plt.show()
```

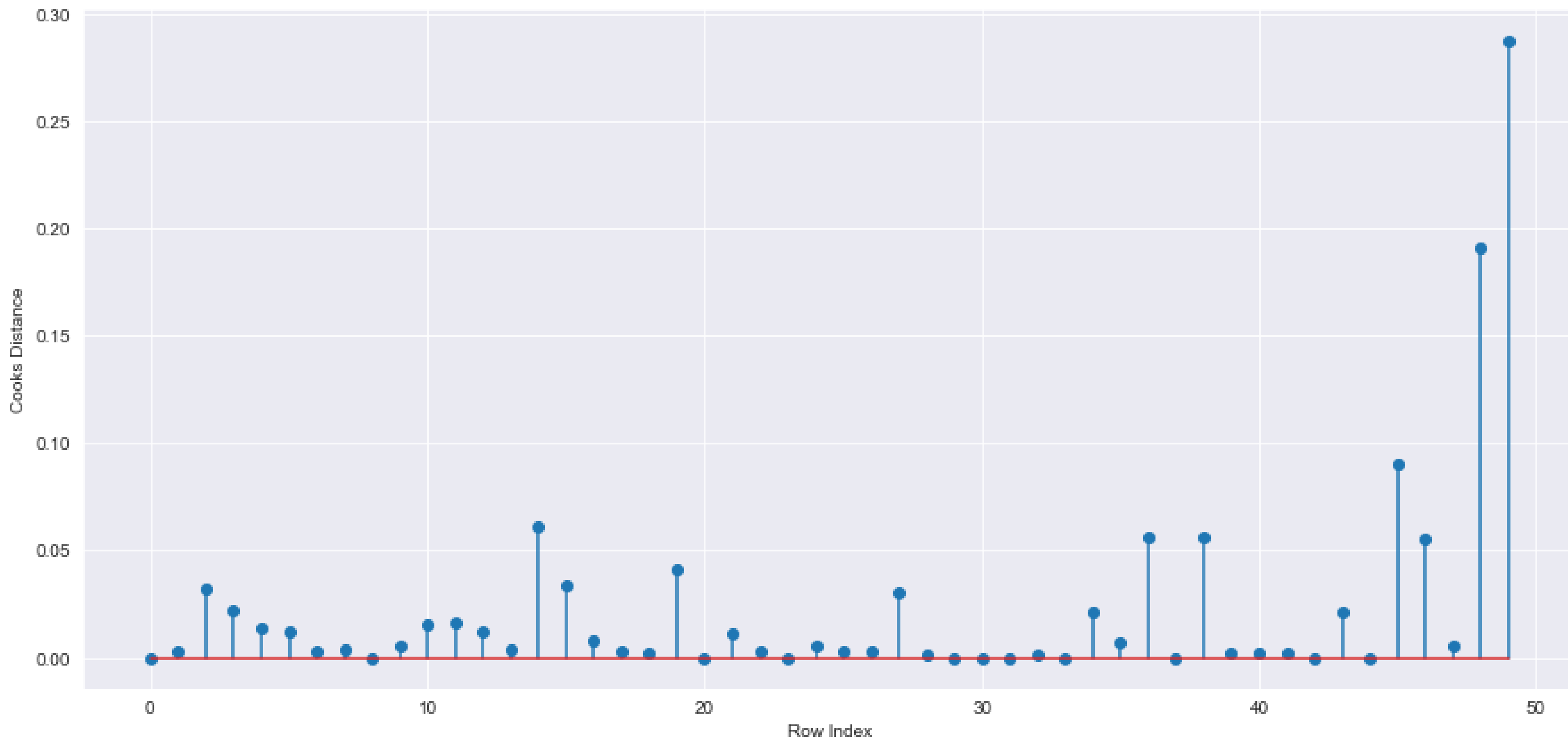


Model deleting diagnostics(Checking outliners or Influenecer)

Two methods 1.Cook's Distance 2.Leverage values

In [29]:
```
#1. Cook's Distance: If Cook's distance > 1, then it's an outlier
# Get influencers using cook's distance
model_influencer=model.get_influence()
(c,_)=model_influencer.cooks_distance
```

In [30]:
```
# Plot the influencers using the stem plot
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()
```



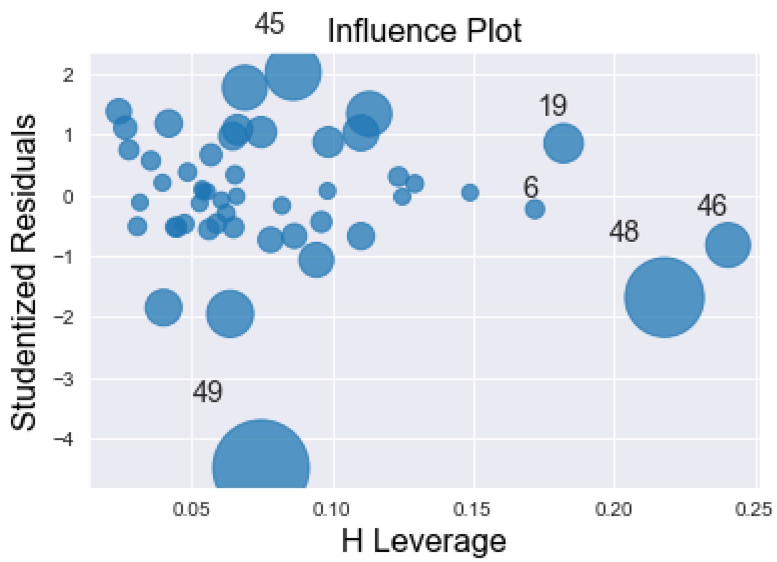
In [31]:
```
# Index and value of influencer where C>0.5
np.argmax(c),np.max(c)
```

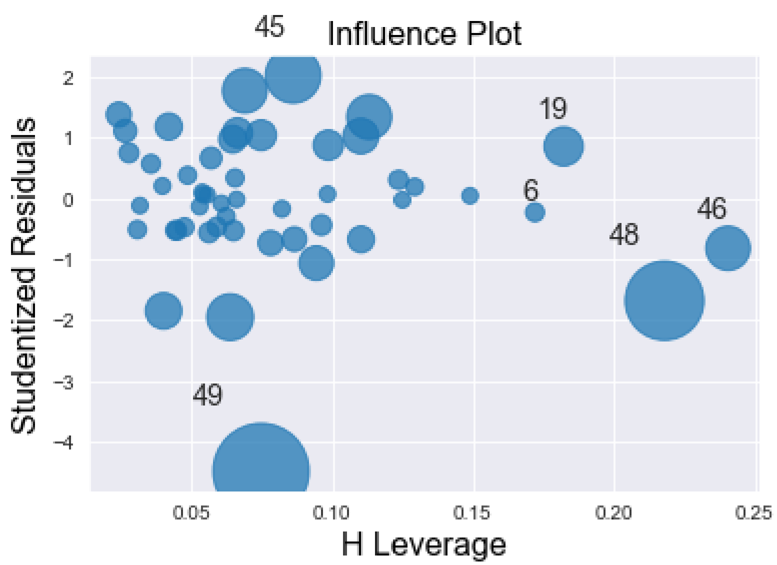
Out[31]: (49, 0.28808229275432673)

2.Leverage values using influencer plot

In [32]:
```
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are influencers
influence_plot(model)
```

Out[32]:





Levrage Cutoff Values

In [33]:
```
#For Our Undestanding
# Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of datapoints
k=data.shape[1]
n=data.shape[0]
levrage_value=(3*(k+1))/n
levrage_value
```

Out[33]:
```
0.36
```

In [34]:
```
data[data.index.isin([49])]
```

Out[34]:

	RDS	ADMS	MKTS	State	Profit
49	0.0	116983.8	45173.06	California	14681.4

Remove outliner data from dataset

In [35]:
```
# Discard the data points which are influencers and reassign the row number (reset_index(drop=True))
data1=data.drop(data.index[[49]],axis=0).reset_index(drop=True)
```

In [36]:
```
#After Improvment lets's see dataset
data1.head()
```

Out[36]:

	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

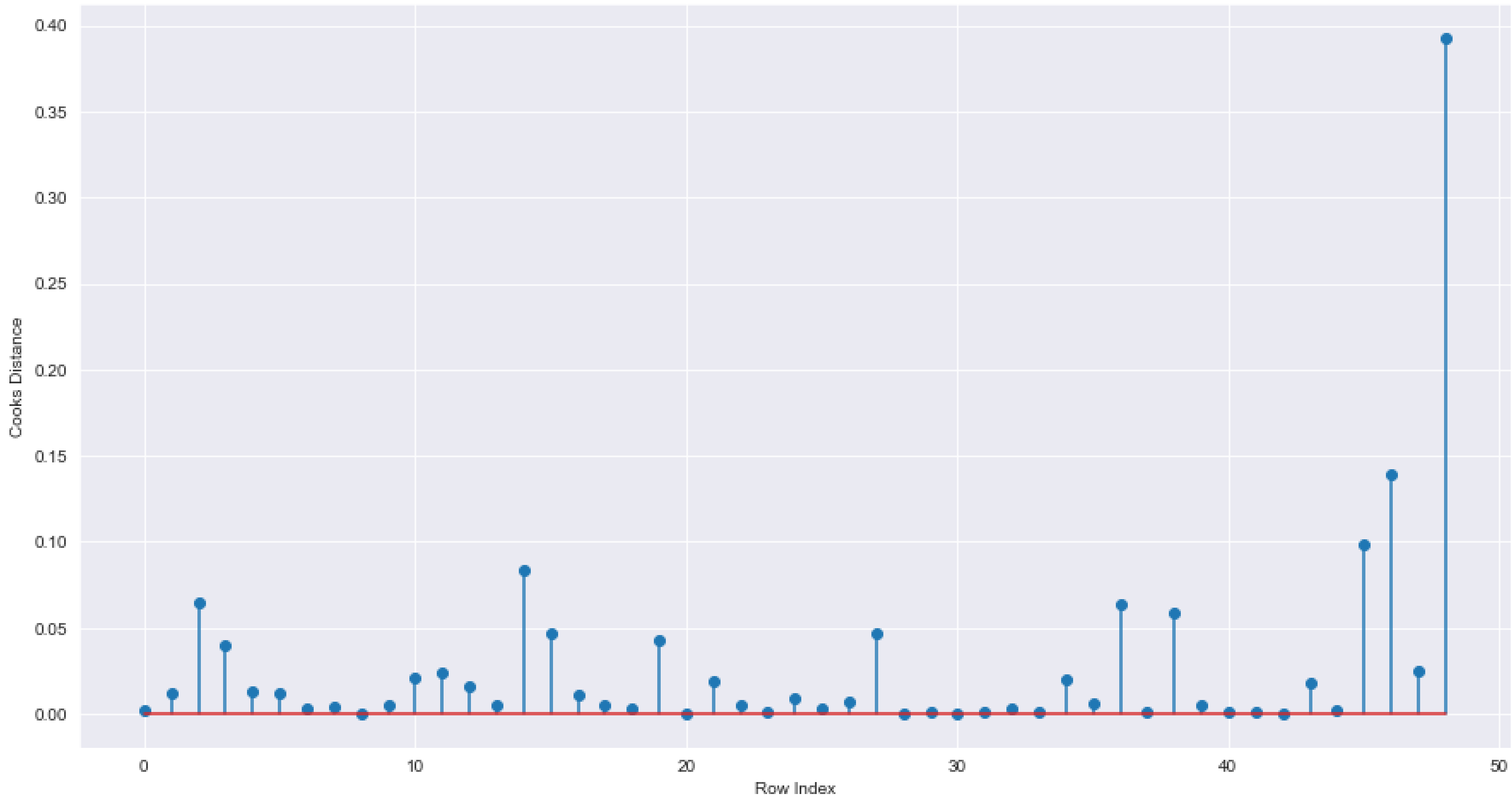
Improving the Model

In [37]:
```
#Create New model using new dataset to improve accuracy
model1=smf.ols("Profit~RDS+ADMS+MKTS",data=data1).fit()
```

Cook's distance

In [38]:
```
# Get influencers using cook's distance
model_influencer1=model1.get_influence()
(c1,_)=model_influencer1.cooks_distance
```

In [39]:
```
# Plot the influencers using the stem plot
plt.figure(figsize=(15,8))
plt.stem(np.arange(len(data1)),np.round(c1,3))
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```



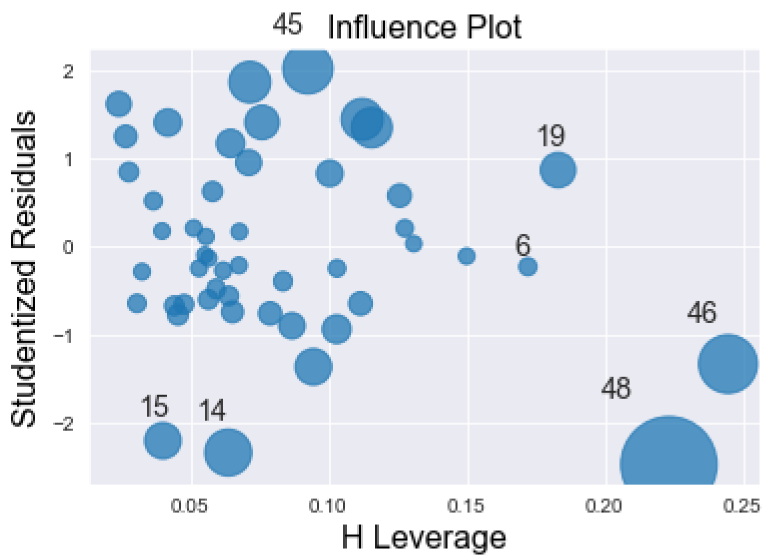
In [40]:
```
np.argmax(c1),np.max(c1)
```

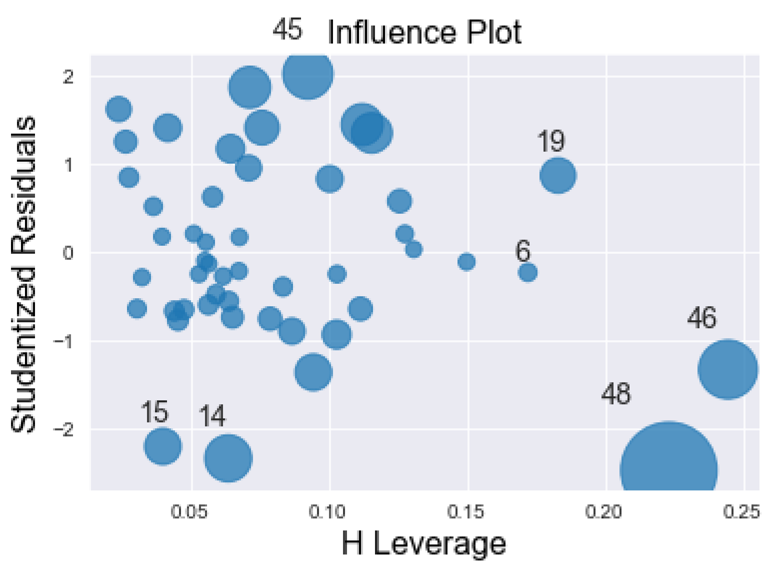
Out[40]:
```
(48, 0.39274420556321116)
```

Influencer Plot

In [41]:
```
# 2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are influencers
influence_plot(model1)
```

Out[41]:





Model Deletion and Final model

In [42]:
```
while np.max(c)>0.5:  #Repeat process untill c value become more than 0.5
    model=smf.ols("Profit~RDS+ADMS+MKTS",data=data1).fit() #fit the model
    model_influencer1=model.get_influence()  #get influence
    (c,_)=model_influencer.cooks_distance   #getting value of c
    np.argmax(c),np.max(c)                  #influencer point and its position
    data1=data1.drop(data1.index[[49]],axis=0).reset_index(drop=True)  #remove influencer point from dataset and prepare dataset
    data1                              #do the process with new datasets and repet loop again

else:
    final_model=smf.ols("Profit~RDS+ADMS+MKTS",data=data1).fit() #Model we get with improve accuracy
    final_model.rsquared , final_model.aic                        #Finding rsquare and adjust r square values.
    print("accuracy improve to ",final_model.rsquared)           #Print accuracy of model
```

```
accuracy improve to  0.9613162435129847
```

Model Prediction

In [43]:
```
new_data=pd.DataFrame({"RDS":73721,"ADMS":121344,"MKTS":211025},index=[0]) #our predicted data
```

In [44]:
```
#Predict output from new data
pred=final_model.predict(new_data)  #Output of Predicted data
pred
```

Out[44]:
```
0    112737.043299
dtype: float64
```

In [45]:
```python
#Predict output from given data input
pred_data=final_model.predict(data1)  #Predict output from given data input
pred_data
```

Out[45]:
```
0     190716.676999
1     187537.122227
2     180575.526396
3     172461.144642
4     170863.486721
5     162582.583177
6     157741.338633
7     159347.735318
8     151328.826941
9     154236.846778
10    135507.792682
11    135472.855621
12    129355.599449
13    127780.129139
14    149295.404796
15    145937.941975
16    117437.627921
17    130408.626295
18    129129.234457
19    116641.003121
20    117097.731866
21    117911.019038
22    115248.217796
23    110603.139045
24    114051.073877
25    103398.054385
26    111547.638935
27    114916.165026
28    103027.229434
29    103057.621761
30    100656.410227
31     99088.213693
32    100325.741335
33     98962.303136
34     90552.307809
35     91709.288672
36     77080.554255
37     90722.503244
```

```
38    71433.021956
39    85147.375646
40    76625.510303
41    76492.145175
42    72492.394974
43    62592.049718
44    67025.731107
45    50457.297206
46    58338.443625
47    49375.776655
48    51658.096812
dtype: float64
```

Table containing R^2 value for each prepared model

In [46]:

```
#Compare result of both the model's accuracy.
D2={"models":["Model","Final model"],"Rsquared":[model.rsquared,final_model.rsquared]}
D2=pd.DataFrame(D2)
D2
```

Out[46]:

	models	Rsquared
0	Model	0.950746
1	Final model	0.961316