1.Importing Necessary Libraries

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

2. Importing data

In [2]:
```
data_startups = pd.read_csv('50_Startups.csv')
data_startups
```

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
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

	R&D Spend	Administration	Marketing Spend	State	Profit
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
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

3. Data Understanding

In [3]:
```
data_startups.shape
```

Out[3]:
```
(50, 5)
```

In [4]:
```
data_startups.isna().sum()
```

Out[4]:
```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

In [5]:
```
data_startups.dtypes
```

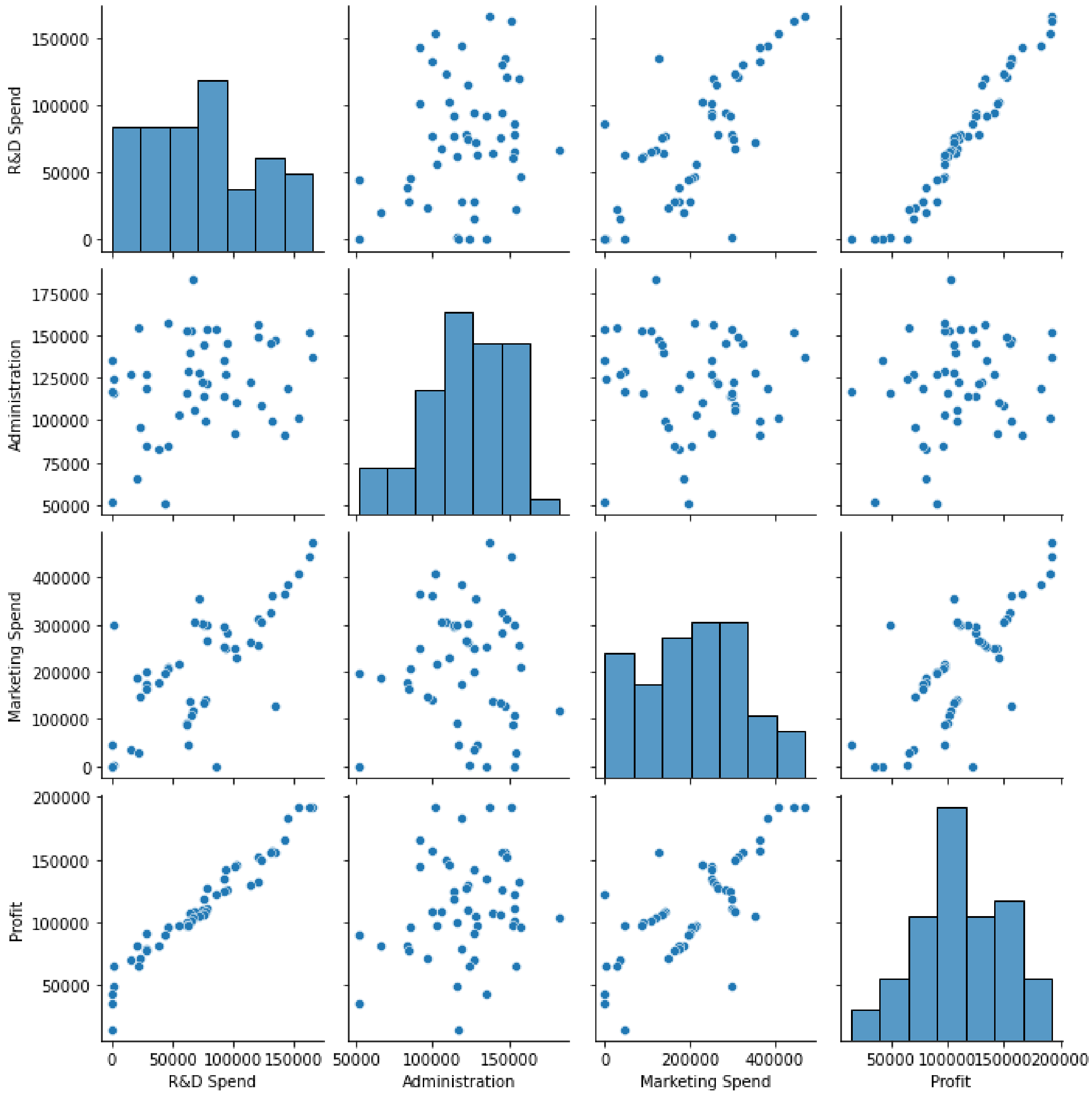
Out[5]:
```
R&D Spend          float64
Administration     float64
Marketing Spend    float64
State               object
Profit             float64
dtype: object
```

3.Check whether the assumptions is matching or not

3.1. Linearity Check

- By using Scatter plot/Paiplot

In [6]:
```
sns.pairplot(data_startups)
plt.show()
```



Observation - Linearity check is failed.

3. 2. No Multicollinearity

By using,

Correlation Matrix (or), Variance Inflation Factor(VIF) we can check it.

In [7]:
```
corr_martrix = data_startups.corr().round()
corr_martrix
```

Out[7]:

	R&D Spend	Administration	Marketing Spend	Profit
R&D Spend	1.0	0.0	1.0	1.0
Administration	0.0	1.0	-0.0	0.0
Marketing Spend	1.0	-0.0	1.0	1.0
Profit	1.0	0.0	1.0	1.0

In [8]:
```
sns.heatmap(data= corr_martrix,annot=True)
plt.show()
```

3.3. No AutoRegression

It is satisfied.

3.4. Homoscadascity check

This can be done after model training.

3.5. Zero Residual Mean

This also can be checked after model training.

4.Data Preparation

In [3]:
```
del data_startups['State']
```

In [4]:
```
data_startups
```

Out[4]:

	R&D Spend	Administration	Marketing Spend	Profit
0	165349.20	136897.80	471784.10	192261.83
1	162597.70	151377.59	443898.53	191792.06
2	153441.51	101145.55	407934.54	191050.39
3	144372.41	118671.85	383199.62	182901.99
4	142107.34	91391.77	366168.42	166187.94
5	131876.90	99814.71	362861.36	156991.12
6	134615.46	147198.87	127716.82	156122.51
7	130298.13	145530.06	323876.68	155752.60
8	120542.52	148718.95	311613.29	152211.77
9	123334.88	108679.17	304981.62	149759.96
10	101913.08	110594.11	229160.95	146121.95
11	100671.96	91790.61	249744.55	144259.40

In [6]:
```
data_startups=data_startups.rename({'R&D Spend':'RDS','Administration':'ADMS','Ma
data_startups
```

Out[6]:

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

	RDS	ADMS	MKTS	Profit
34	46426.07	157693.92	210797.67	96712.80
35	46014.02	85047.44	205517.64	96479.51
36	28663.76	127056.21	201126.82	90708.19
37	44069.95	51283.14	197029.42	89949.14
38	20229.59	65947.93	185265.10	81229.06
39	38558.51	82982.09	174999.30	81005.76
40	28754.33	118546.05	172795.67	78239.91
41	27892.92	84710.77	164470.71	77798.83
42	23640.93	96189.63	148001.11	71498.49
43	15505.73	127382.30	35534.17	69758.98
44	22177.74	154806.14	28334.72	65200.33
45	1000.23	124153.04	1903.93	64926.08
46	1315.46	115816.21	297114.46	49490.75
47	0.00	135426.92	0.00	42559.73
48	542.05	51743.15	0.00	35673.41
49	0.00	116983.80	45173.06	14681.40

5. Model Building | Training | Evaluating using Statsmodels

In [29]:
```
import statsmodels.formula.api as smf
model = smf.ols('Profit~RDS+ADMS+MKTS',data= data_startups).fit()
```

In [30]:
```
model.params
```

Out[30]:
```
Intercept    50122.192990
RDS              0.805715
ADMS            -0.026816
MKTS             0.027228
dtype: float64
```

In [31]:
```
model.pvalues
```

Out[31]:
```
Intercept    1.057379e-09
RDS          2.634968e-22
ADMS         6.017551e-01
MKTS         1.047168e-01
dtype: float64
```

In [32]:
```
model.rsquared,model.rsquared_adj
```

Out[32]:
```
(0.9507459940683246, 0.9475337762901719)
```

Understanding R2 and AdjustedR2

In [11]:
```
model_1 = smf.ols('Profit~RDS',data = data_startups).fit()
print('R2 score          : ',round(model_1.rsquared,4))
print('Adjusted R2 score : ',round(model_1.rsquared_adj,4))
print('AIC value         : ',round(model_1.aic,4)) #is an estimator of predictior
print('BIC value         : ',round(model_1.bic,4)) ##is an estimator of predictic
```

```
R2 score          :  0.9465
Adjusted R2 score :  0.9454
AIC value         :  1058.873
BIC value         :  1062.6971
```

In [12]:
```
model_2 = smf.ols('Profit~RDS+ADMS',data = data_startups).fit()
print('R2 score          : ',round(model_2.rsquared,4))
print('Adjusted R2 score : ',round(model_2.rsquared_adj,4))
print('AIC value         : ',round(model_2.aic,4))
print('BIC value         : ',round(model_2.bic,4))
```

```
R2 score          :  0.9478
Adjusted R2 score :  0.9456
AIC value         :  1059.6637
BIC value         :  1065.3998
```

In [13]:
```
model_3 = smf.ols('Profit~RDS+ADMS+MKTS',data = data_startups).fit()
print('R2 score          : ',round(model_3.rsquared,4))
print('Adjusted R2 score : ',round(model_3.rsquared_adj,4))
print('AIC value         : ',round(model_3.aic,4))
print('BIC value         : ',round(model_3.bic,4))
```

```
R2 score          :  0.9507
Adjusted R2 score :  0.9475
AIC value         :  1058.7715
BIC value         :  1066.4196
```

Observation:

Always R2 score increases if we increase the number of inputs. And if the input contributes more for the prediction, there will a higher increase in the R2 score.

To look the complete Summary of the Model

In [14]:
```
model_1.summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.947
Model:	OLS	**Adj. R-squared:**	0.945
Method:	Least Squares	**F-statistic:**	849.8
Date:	Fri, 29 Oct 2021	**Prob (F-statistic):**	3.50e-32
Time:	22:31:27	**Log-Likelihood:**	-527.44
No. Observations:	50	**AIC:**	1059.
Df Residuals:	48	**BIC:**	1063.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	4.903e+04	2537.897	19.320	0.000	4.39e+04	5.41e+04
RDS	0.8543	0.029	29.151	0.000	0.795	0.913

Omnibus:	13.727	**Durbin-Watson:**	1.116
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	18.536
Skew:	-0.911	**Prob(JB):**	9.44e-05
Kurtosis:	5.361	**Cond. No.**	1.65e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.65e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [15]:
```
model_2.summary()
```

Out[15]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.948
Model:	OLS	**Adj. R-squared:**	0.946
Method:	Least Squares	**F-statistic:**	426.8
Date:	Fri, 29 Oct 2021	**Prob (F-statistic):**	7.29e-31
Time:	22:31:29	**Log-Likelihood:**	-526.83
No. Observations:	50	**AIC:**	1060.
Df Residuals:	47	**BIC:**	1065.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5.489e+04	6016.718	9.122	0.000	4.28e+04	6.7e+04
RDS	0.8621	0.030	28.589	0.000	0.801	0.923
ADMS	-0.0530	0.049	-1.073	0.289	-0.152	0.046

Omnibus:	14.678	**Durbin-Watson:**	1.189
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	20.449
Skew:	-0.961	**Prob(JB):**	3.63e-05
Kurtosis:	5.474	**Cond. No.**	6.65e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.65e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [16]:
```
model_3.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.951
Model:	OLS	**Adj. R-squared:**	0.948
Method:	Least Squares	**F-statistic:**	296.0
Date:	Fri, 29 Oct 2021	**Prob (F-statistic):**	4.53e-30
Time:	22:31:31	**Log-Likelihood:**	-525.39
No. Observations:	50	**AIC:**	1059.
Df Residuals:	46	**BIC:**	1066.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5.012e+04	6572.353	7.626	0.000	3.69e+04	6.34e+04
RDS	0.8057	0.045	17.846	0.000	0.715	0.897
ADMS	-0.0268	0.051	-0.526	0.602	-0.130	0.076
MKTS	0.0272	0.016	1.655	0.105	-0.006	0.060

Omnibus:	14.838	**Durbin-Watson:**	1.282
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	21.442
Skew:	-0.949	**Prob(JB):**	2.21e-05
Kurtosis:	5.586	**Cond. No.**	1.40e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.4e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

6.Model Validation

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

In [17]:
```
rsq_r=smf.ols("RDS~ADMS+MKTS",data=data_startups).fit().rsquared
vif_r=1/(1-rsq_r)

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

rsq_m=smf.ols("MKTS~RDS+ADMS",data=data_startups).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[17]:

	Variables	Vif
0	RDS	2.468903
1	ADMS	1.175091
2	MKTS	2.326773

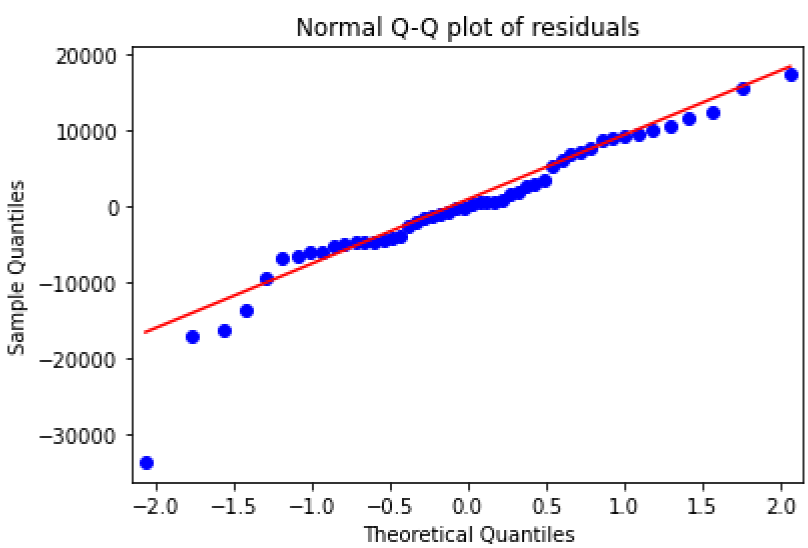
observations-None variable has VIF>20, No Collinearity, so consider all varaibles in Regression equation

Test for Normality of Residuals (Q-Q Plot)

In [33]:
```
import statsmodels.api as sm

# 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 [34]:
```
list(np.where(model.resid<-30000))
```

Out[34]:
```
[array([49], dtype=int64)]
```

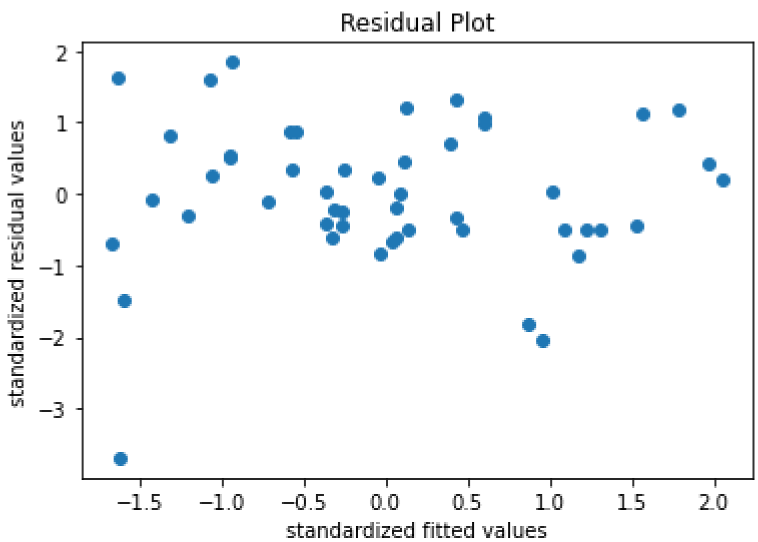
Residual Plot for Homoscedasticity

In [23]:
```
# Test for Homoscedasticity or Heteroscedasticity (plotting model's standardized

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

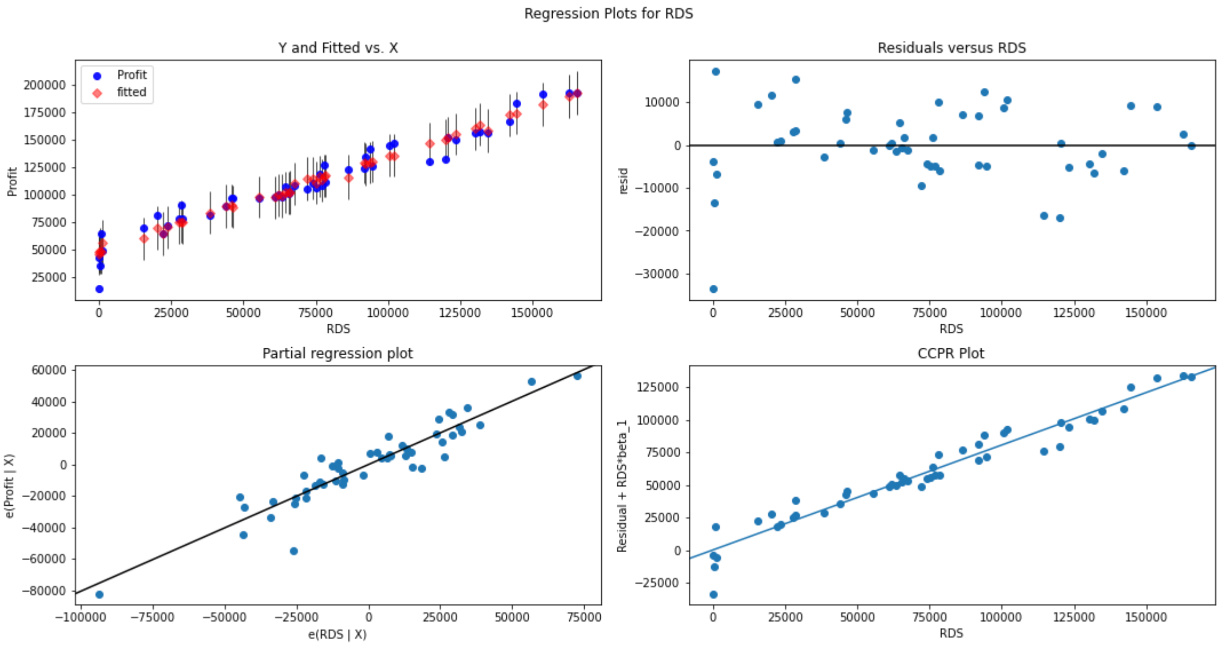
In [35]:
```

plt.scatter(standard_values(model.fittedvalues),standard_values(model_1.resid))
plt.title('Residual Plot')
plt.xlabel('standardized fitted values')
plt.ylabel('standardized residual values')
plt.show()
```

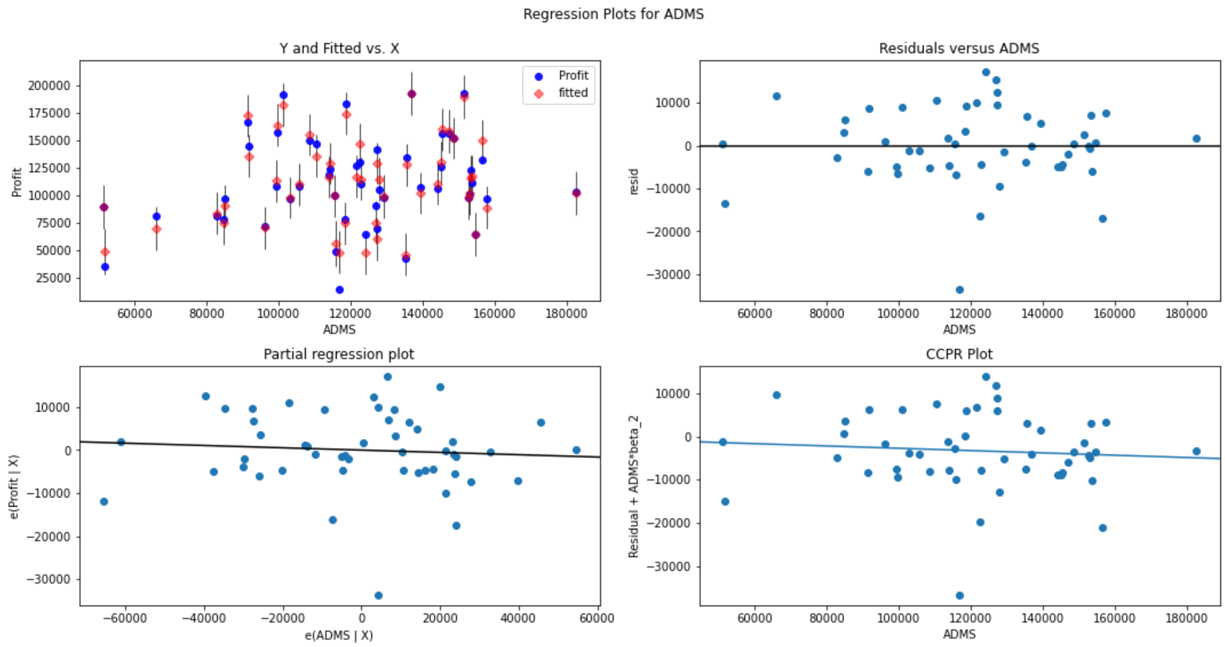


Residual Vs Regressors

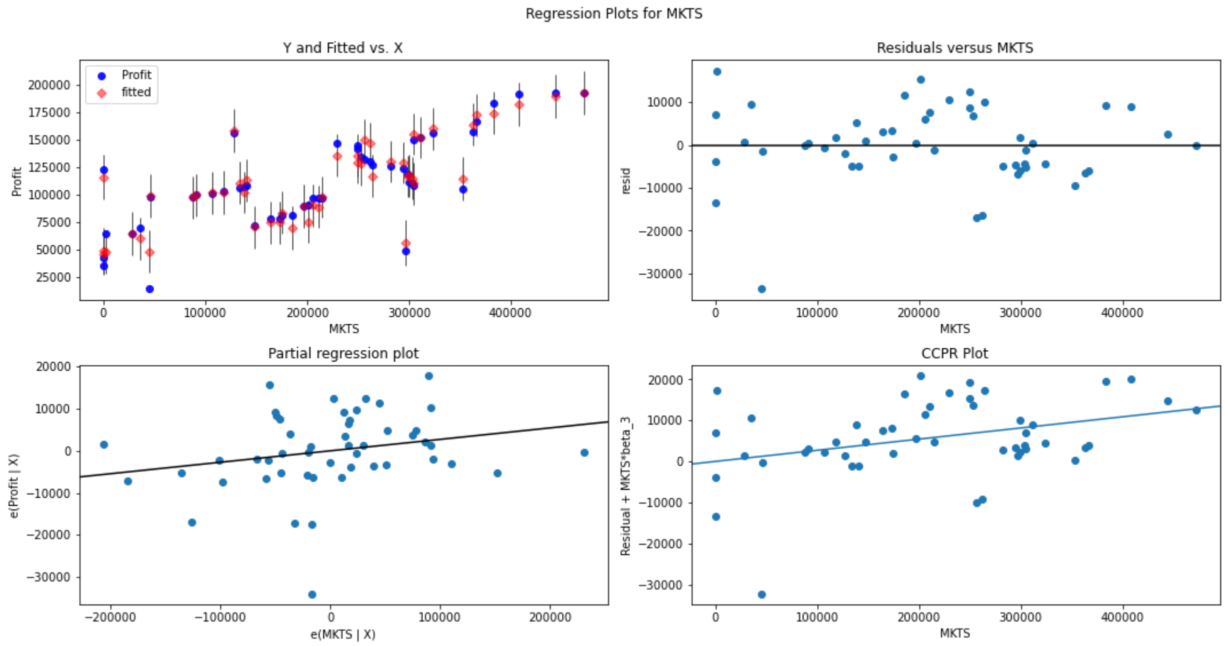
In [36]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'RDS',fig=fig)
plt.show()
```



In [38]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'ADMS',fig=fig)
plt.show()
```



In [39]:
```
fig=plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(model,'MKTS',fig=fig)
plt.show()
```



Model Deletion Diagnostics

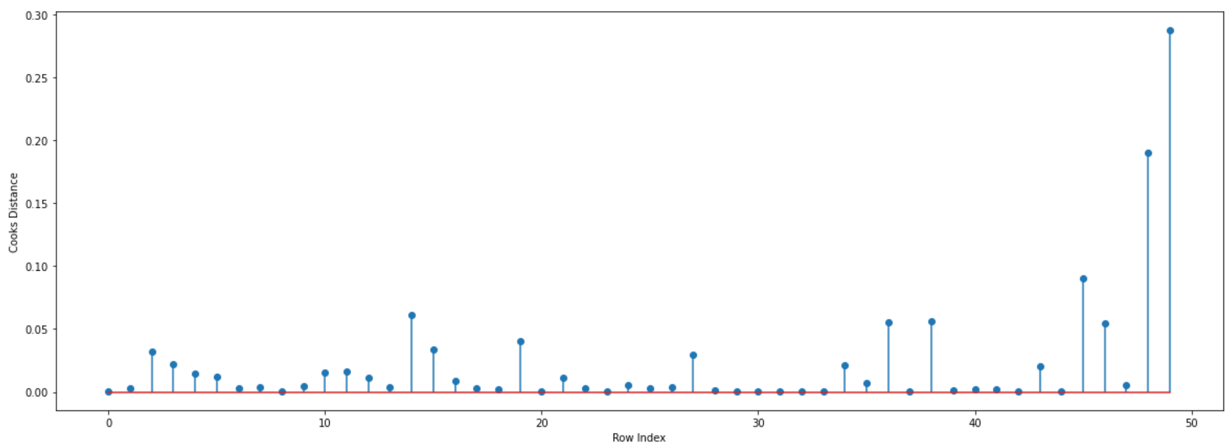
Detecting Influencers/Outliers

Cook's Distance

In [40]:
```
# 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[40]:
```
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 [42]:
```
# Plot the influencers using the stem plot
fig=plt.figure(figsize=(20,7))
plt.stem(np.arange(len(data_startups)),np.round(c,5))
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```



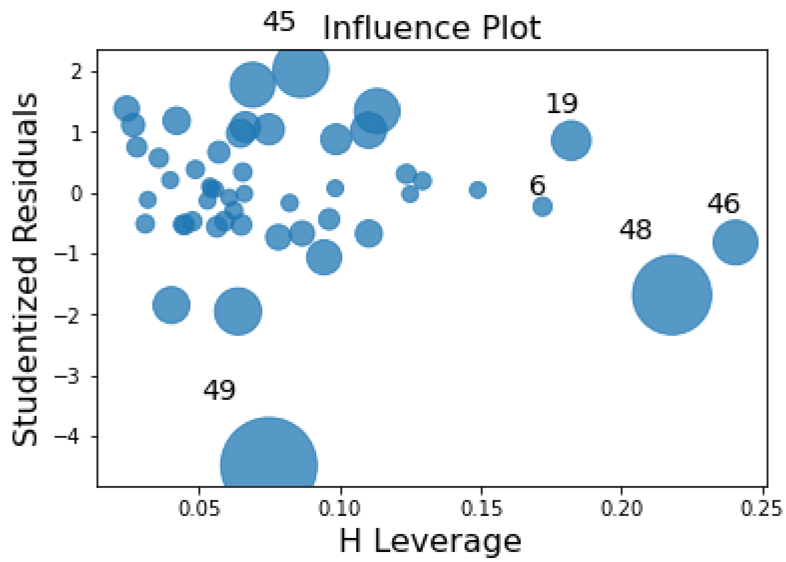
In [44]:
```
# Index and value of influencer where C>0.5
np.argmax(c) , np.max(c)
```

Out[44]:
```
(49, 0.28808229275432584)
```

2. Leverage Value using High Influence Points : Points beyond Leverage_cutoff value are influencers

In [46]:
```
from statsmodels.graphics.regressionplots import influence_plot

influence_plot(model)
plt.show()
```



In [47]:
```
# Leverage Cuttoff Value = 3*(k+1)/n ; k = no.of features/columns & n = no. of da
k=data_startups.shape[1]
n=data_startups.shape[0]
leverage_cutoff = (3*(k+1))/n
leverage_cutoff
```

Out[47]: 0.3

In [48]:
```
data_startups[data_startups.index.isin([49])]
```

Out[48]:

	RDS	ADMS	MKTS	Profit
49	0.0	116983.8	45173.06	14681.4

In [49]:
```
data_startups.head()
```

Out[49]:

	RDS	ADMS	MKTS	Profit
0	165349.20	136897.80	471784.10	192261.83
1	162597.70	151377.59	443898.53	191792.06
2	153441.51	101145.55	407934.54	191050.39
3	144372.41	118671.85	383199.62	182901.99
4	142107.34	91391.77	366168.42	166187.94

Improving the model

Discard the data points which are influencers and reassign the row number (reset_index(drop=True))

In [51]:
```
data_2=data_startups.drop(data_startups.index[[49]],axis=0).reset_index(drop=True
data_2
```

Out[51]:

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

	RDS	ADMS	MKTS	Profit
34	46426.07	157693.92	210797.67	96712.80
35	46014.02	85047.44	205517.64	96479.51
36	28663.76	127056.21	201126.82	90708.19
37	44069.95	51283.14	197029.42	89949.14
38	20229.59	65947.93	185265.10	81229.06
39	38558.51	82982.09	174999.30	81005.76
40	28754.33	118546.05	172795.67	78239.91
41	27892.92	84710.77	164470.71	77798.83
42	23640.93	96189.63	148001.11	71498.49
43	15505.73	127382.30	35534.17	69758.98
44	22177.74	154806.14	28334.72	65200.33
45	1000.23	124153.04	1903.93	64926.08
46	1315.46	115816.21	297114.46	49490.75
47	0.00	135426.92	0.00	42559.73
48	542.05	51743.15	0.00	35673.41

Model Deletion Diagnostics and Final Model

In [52]:
```
model_new=smf.ols("Profit~RDS+ADMS+MKTS",data=data_2).fit()
```

In [53]:
```
while model_new.rsquared < 0.99:
    for c in [np.max(c)>1]:
        model_new=smf.ols("Profit~RDS+ADMS+MKTS",data=data_2).fit()
        (c,_)=model_new.get_influence().cooks_distance
        c
        np.argmax(c) , np.max(c)
        data_2=data_2.drop(data_2.index[[np.argmax(c)]],axis=0).reset_index(drop=
        data_2
    else:
        final_model=smf.ols("Profit~RDS+ADMS+MKTS",data=data_2).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 0.9706076169779905
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 0.9777383743090916
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 0.9877191935547199
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 [54]:
```
final_model.rsquared
```

Out[54]:
```
0.9882757054424702
```

In [55]:
```
data_2
```

Out[55]:

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

Model Predictions

In [56]:
```
# say New data for prediction is
new_data=pd.DataFrame({'RDS':70000,"ADMS":90000,"MKTS":140000},index=[0])
new_data
```

Out[56]:

	RDS	ADMS	MKTS
0	70000	90000	140000

In [57]:
```
final_model.predict(new_data)
```

Out[57]:
```
0    104858.729408
dtype: float64
```

In [58]:
```
# Automatic Prediction of Price with 90.02% accurcy
pred_y=final_model.predict(data_2)
pred_y
```

Out[58]:
```
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 [59]:
```
d2={'Prep_Models':['Model','Final_Model'],'Rsquared':[model.rsquared,final_model.
table=pd.DataFrame(d2)
table
```

Out[59]:

	Prep_Models	Rsquared
0	Model	0.950746
1	Final_Model	0.988276