In [1]:

```
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
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df = pd.read_csv("50_Startups.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	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

In [4]:

```
df.tail()
```

Out[4]:

	R&D Spend	Administration	Marketing Spend	State	Profit
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 [5]:

```
df.isna().sum()
```

Out[5]:

```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

In [6]:

```
df.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
                      50 non-null     object
```

Loading [MathJax]/extensions/Safe.js

```
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [7]:
```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

In [8]:
```python
df["State"]=label_encoder.fit_transform(df["State"])
```

In [9]:
```python
df.head(10)
```

Out[9]:

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

In [10]:
```python
numerical_feature = [feature for feature in df.columns if df[feature].dtypes!="O"]
```

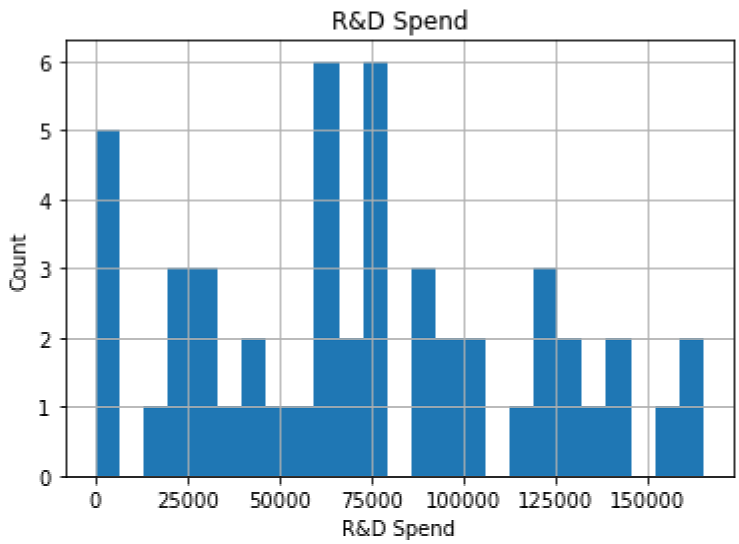
In [11]:
```python
df[numerical_feature].head()
```

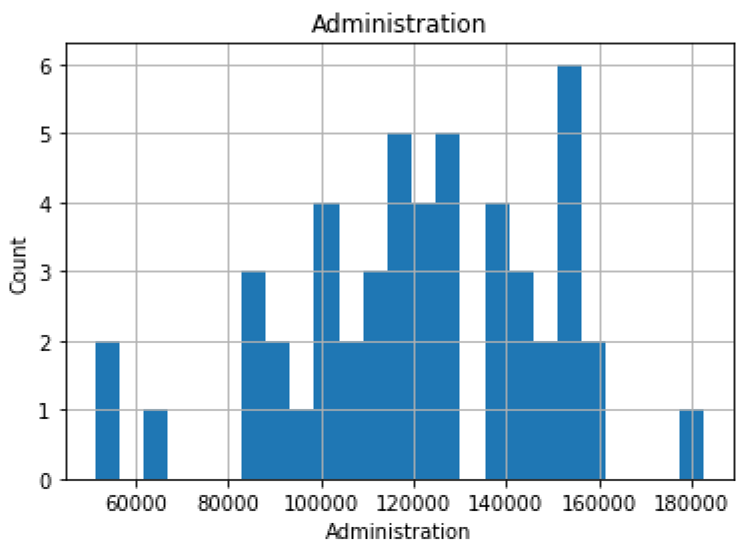
Out[11]:

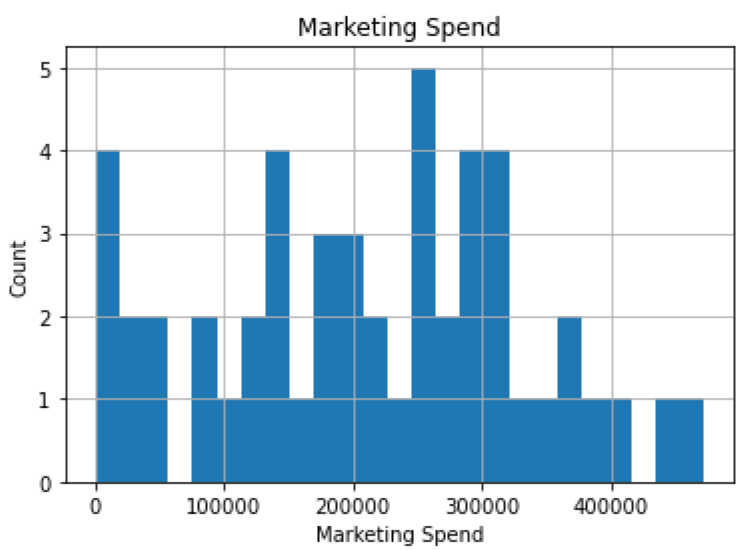
	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	2	192261.83
1	162597.70	151377.59	443898.53	0	191792.06
2	153441.51	101145.55	407934.54	1	191050.39
3	144372.41	118671.85	383199.62	2	182901.99
4	142107.34	91391.77	366168.42	1	166187.94

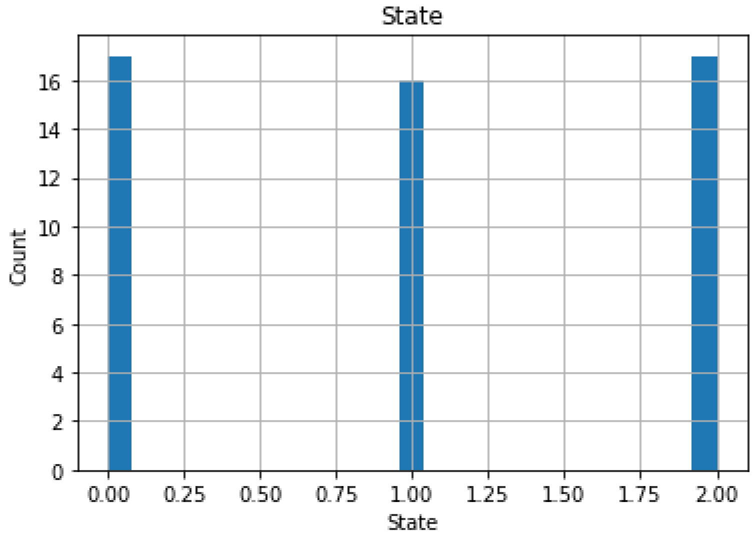
Histogram for Raw Data

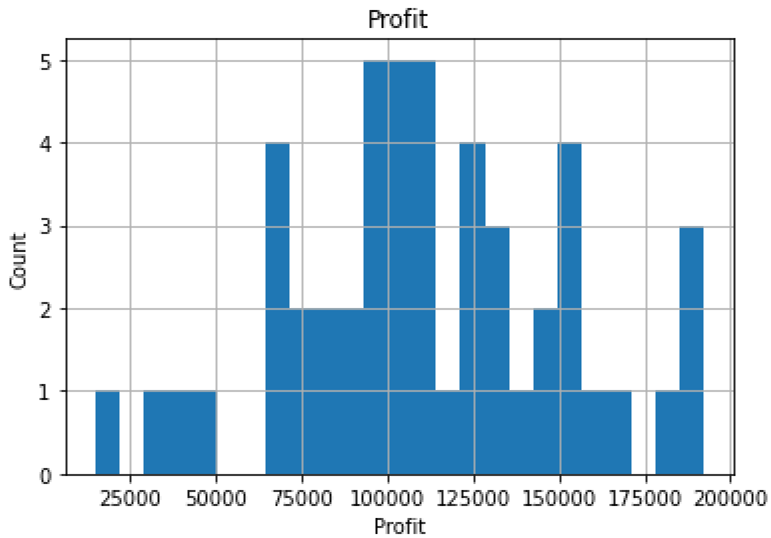
In [13]:
```python
for feature in numerical_feature:
    data = df.copy()
    data[feature].hist(bins=25)
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```


R&D Spend
Count
R&D Spend
0
25000
50000
75000
100000
125000
150000


Administration
Count
Administration
60000
80000
100000
120000
140000
160000
180000


Marketing Spend
Count
Marketing Spend
0
100000
200000
300000
400000



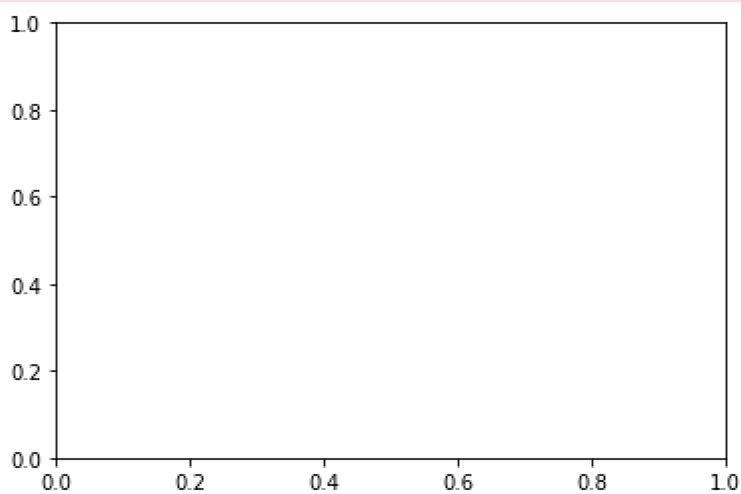


In [13]:

```
for feature in numerical_feature:
    data = df.copy()
    data[feature]=np.log(data[feature])
    data[feature].hist()
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```

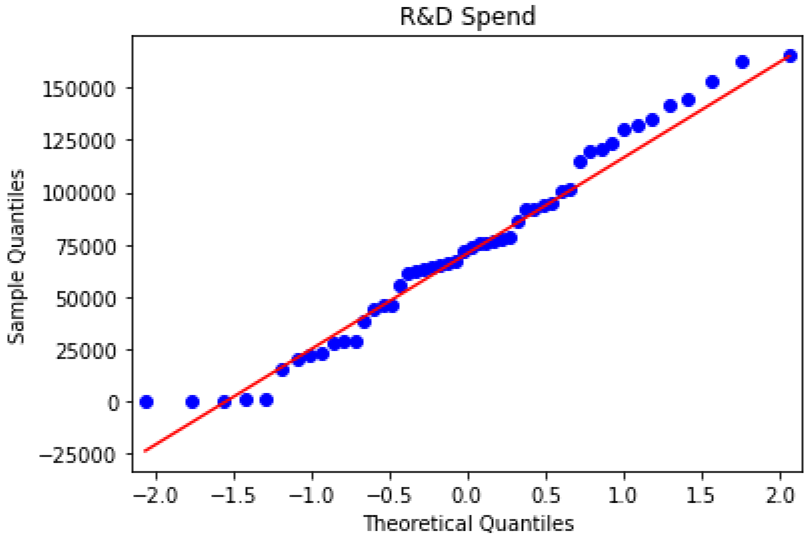
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-c6b590fc8919> in <module>
      2     data = df.copy()
      3     data[feature]=np.log(data[feature])
----> 4     data[feature].hist()
      5     plt.xlabel(feature)
      6     plt.ylabel("Count")

~\anaconda3\lib\site-packages\pandas\plotting\_core.py in hist_series(self, by, ax, grid,
 xlabelsize, xrot, ylabelsize, yrot, figsize, bins, backend, legend, **kwargs)
     83     """
     84     plot_backend = _get_plot_backend(backend)
---> 85     return plot_backend.hist_series(
     86         self,
     87         by=by,

~\anaconda3\lib\site-packages\pandas\plotting\_matplotlib\hist.py in hist_series(self, by,
ax, grid, xlabelsize, xrot, ylabelsize, yrot, figsize, bins, legend, **kwds)
    336         if legend:
```

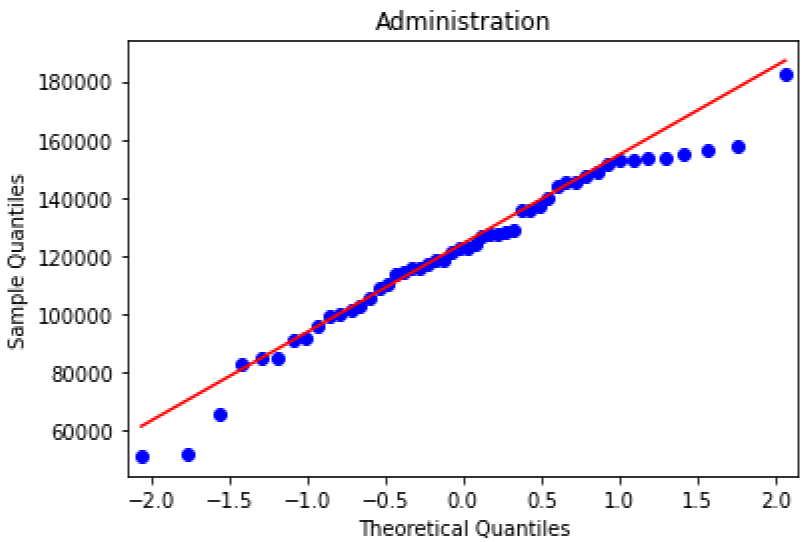
```
    337             kwds["label"] = self.name
--> 338         ax.hist(values, bins=bins, **kwds)
    339         if legend:
    340             ax.legend()

~\anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, data, *args, **kwargs)
   1445     def inner(ax, *args, data=None, **kwargs):
   1446         if data is None:
-> 1447             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1448 
   1449         bound = new_sig.bind(ax, *args, **kwargs)

~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py in hist(self, x, bins, range, densi
ty, weights, cumulative, bottom, histtype, align, orientation, rwidth, log, color, label,
 stacked, **kwargs)
   6649             # this will automatically overwrite bins,
   6650             # so that each histogram uses the same bins
-> 6651             m, bins = np.histogram(x[i], bins, weights=w[i], **hist_kwargs)
   6652             tops.append(m)
   6653         tops = np.array(tops, float)  # causes problems later if it's an int

<__array_function__ internals> in histogram(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\lib\histograms.py in histogram(a, bins, range, normed,
weights, density)
    790     a, weights = _ravel_and_check_weights(a, weights)
    791 
--> 792     bin_edges, uniform_bins = _get_bin_edges(a, bins, range, weights)
    793 
    794     # Histogram is an integer or a float array depending on the weights.

~\anaconda3\lib\site-packages\numpy\lib\histograms.py in _get_bin_edges(a, bins, range, we
ights)
    424             raise ValueError('`bins` must be positive, when an integer')
    425 
--> 426         first_edge, last_edge = _get_outer_edges(a, range)
    427 
    428     elif np.ndim(bins) == 1:

~\anaconda3\lib\site-packages\numpy\lib\histograms.py in _get_outer_edges(a, range)
    313                 'max must be larger than min in range parameter.')
    314         if not (np.isfinite(first_edge) and np.isfinite(last_edge)):
--> 315             raise ValueError(
    316                 "supplied range of [{}, {}] is not finite".format(first_edge, last
_edge))
    317     elif a.size == 0:

ValueError: supplied range of [-inf, 12.015814880176281] is not finite
```

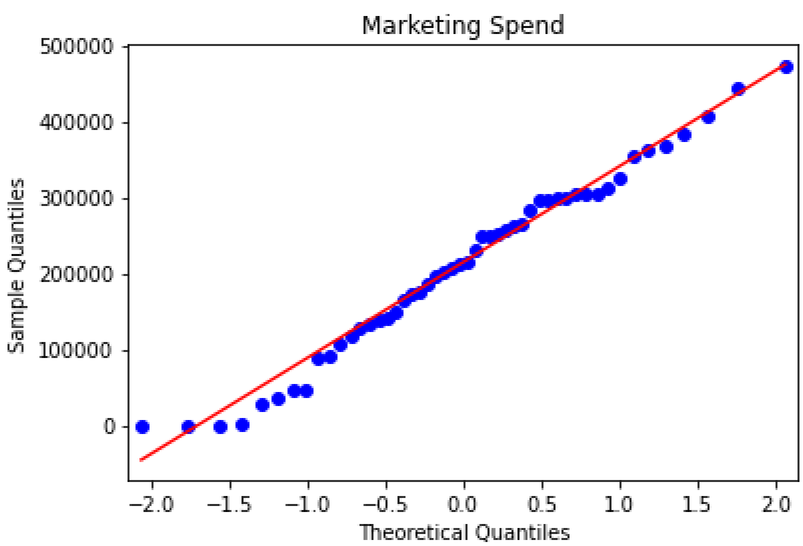


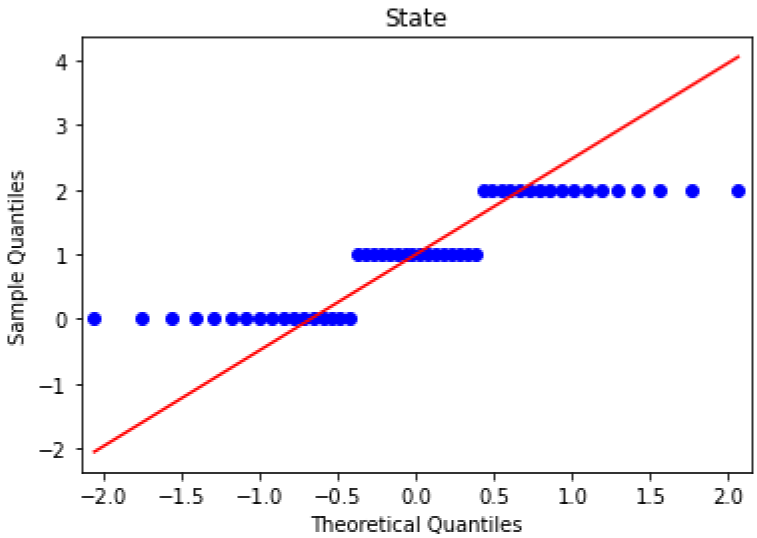
QQ-Plot for Raw Data

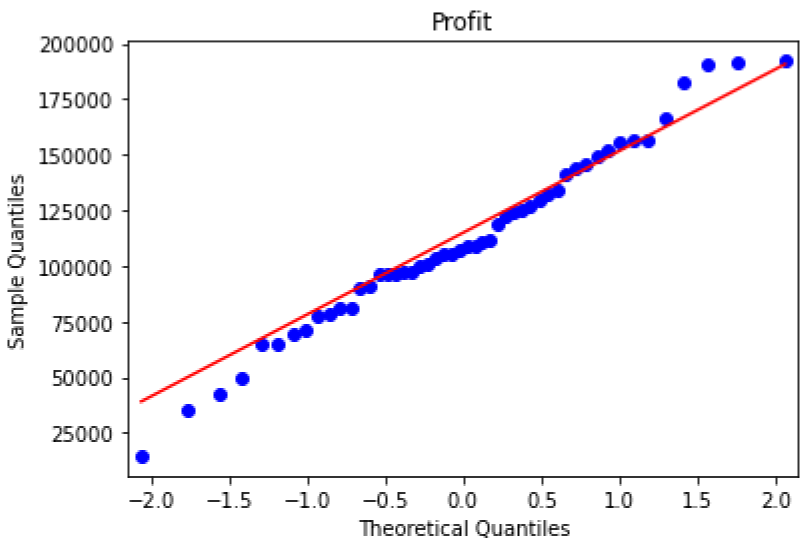
In [14]:
```
for feature in numerical_feature:
    data = df.copy()
    sm.qqplot(data[feature],line = 'q')
    plt.title(feature)
```







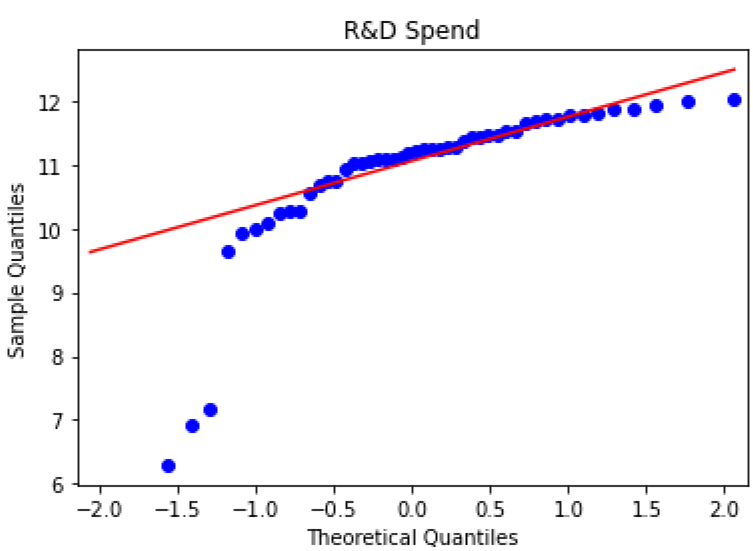




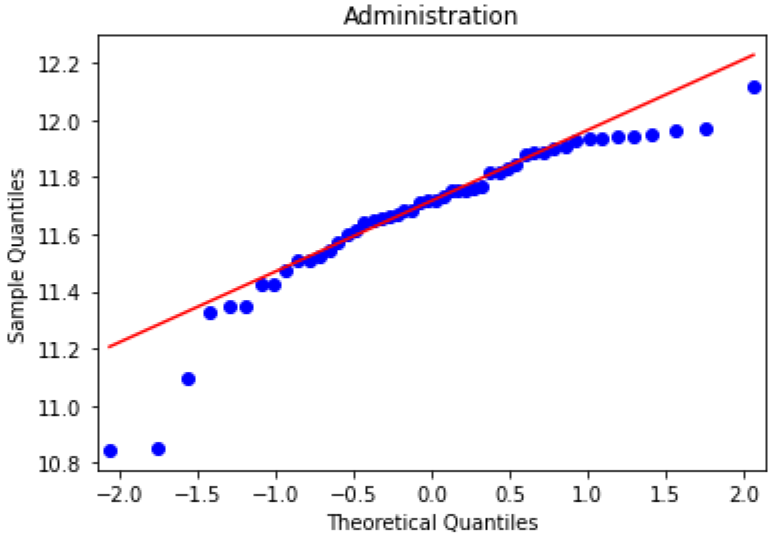
QQ-Plot of Log Transformation

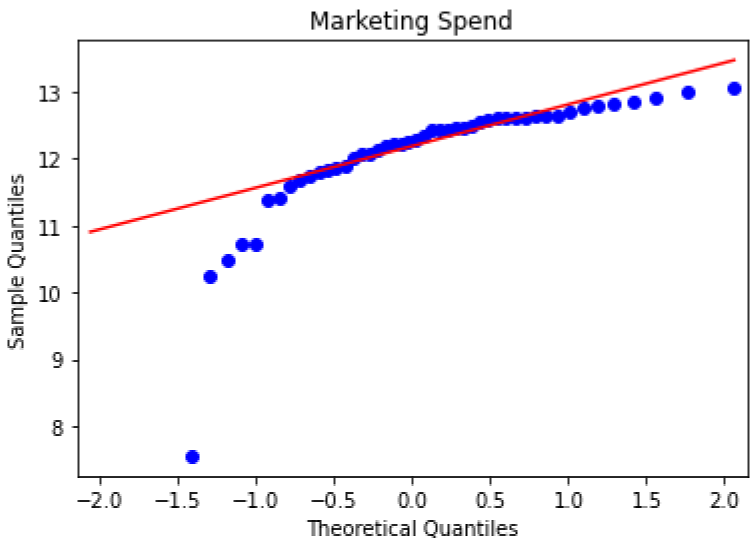
In [14]:

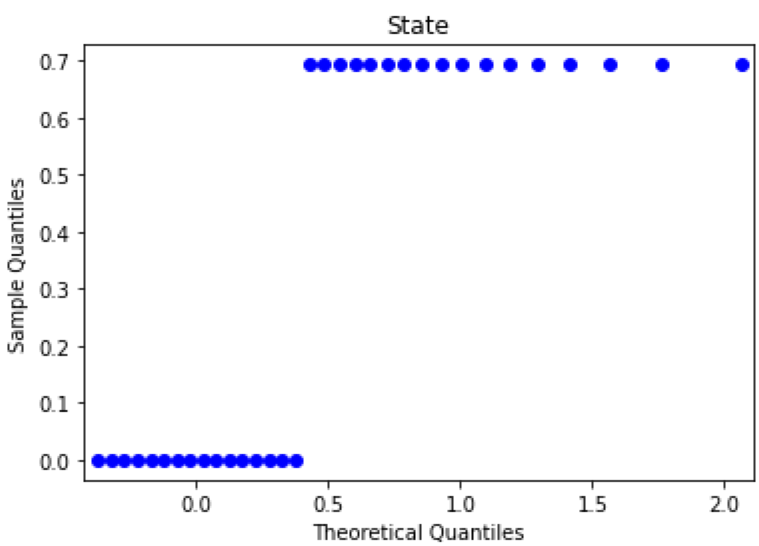
```
for feature in numerical_feature:
    data = df.copy()
    sm.qqplot(np.log(data[feature]),line ='q')
    plt.title(feature)
```

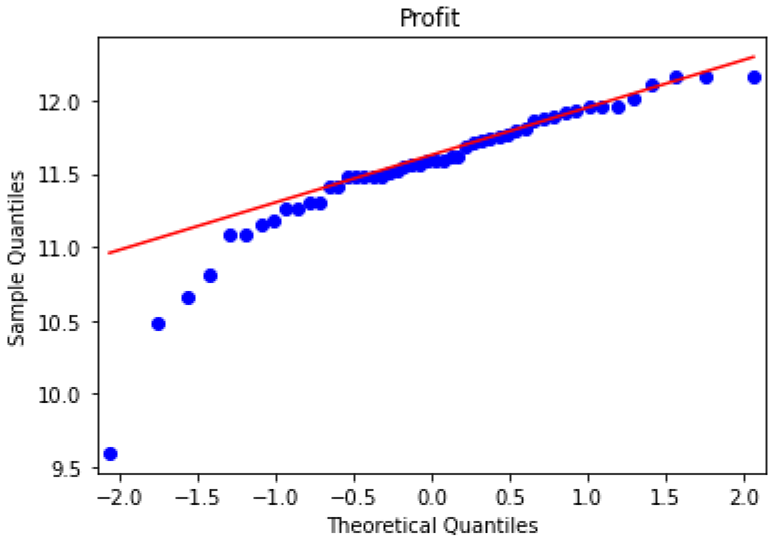


Loading [MathJax]/extensions/Safe.js


Administration
Sample Quantiles
Theoretical Quantiles
12.2
12.0
11.8
11.6
11.4
11.2
11.0
10.8
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Marketing Spend
Sample Quantiles
Theoretical Quantiles
13
12
11
10
9
8
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

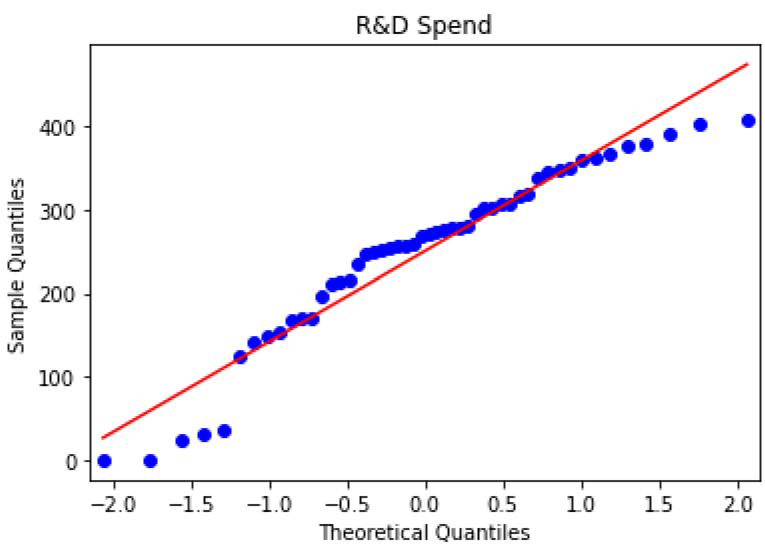

State
Sample Quantiles
Theoretical Quantiles
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.5
1.0
1.5
2.0

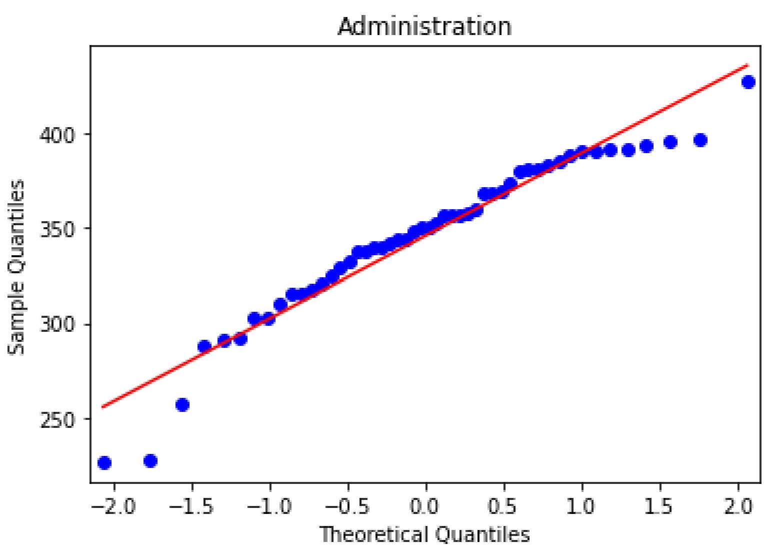


QQ-Plot for Square Root Transformation

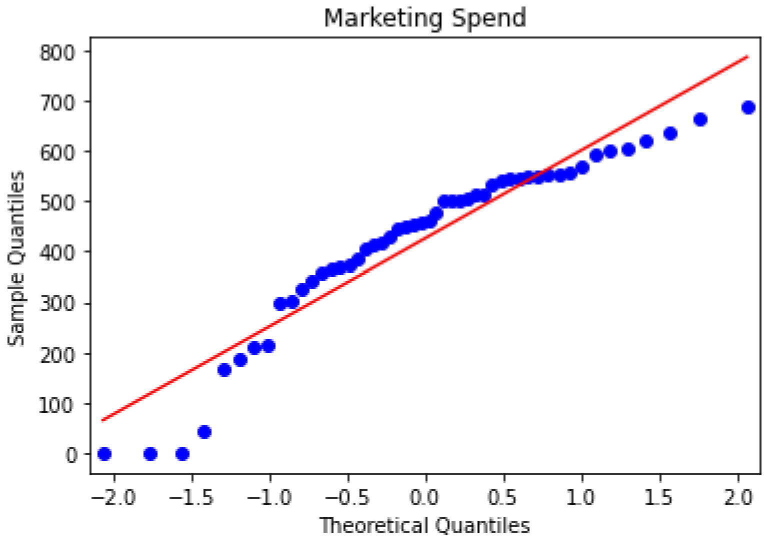
In [15]:

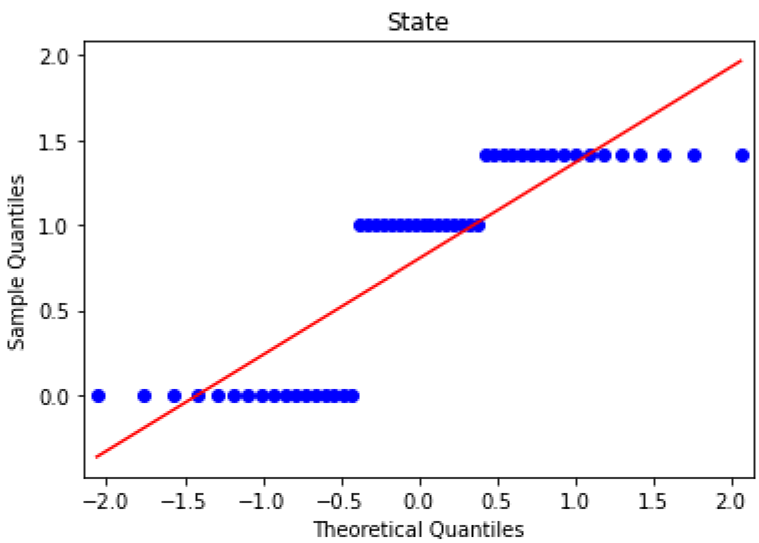
```
for feature in numerical_feature:
    data = df.copy()
    sm.qqplot(np.sqrt(data[feature]),line="r")
    plt.title(feature)
```

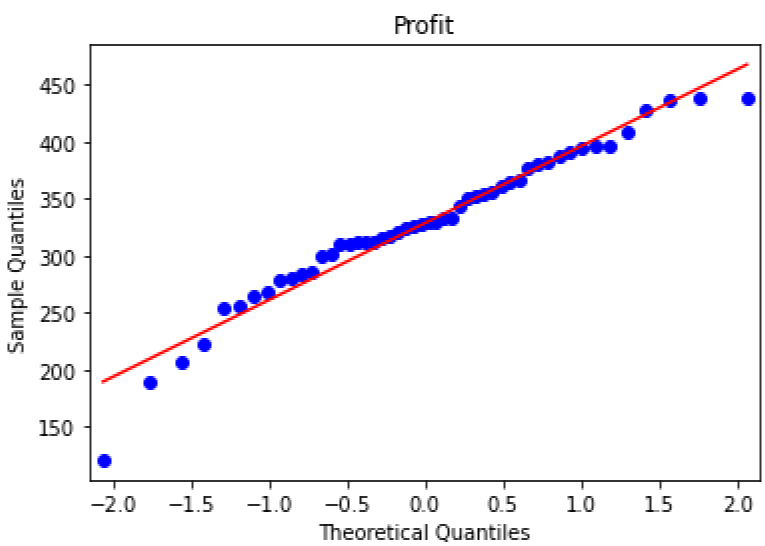




Loading [MathJax]/extensions/Safe.js





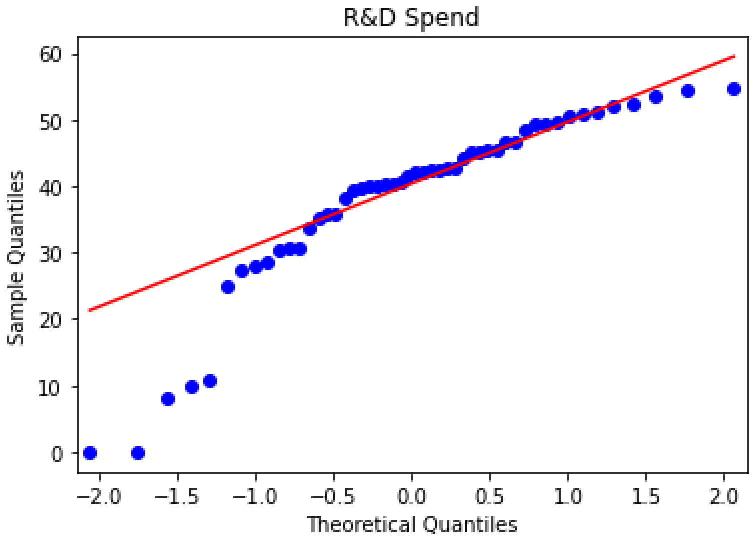


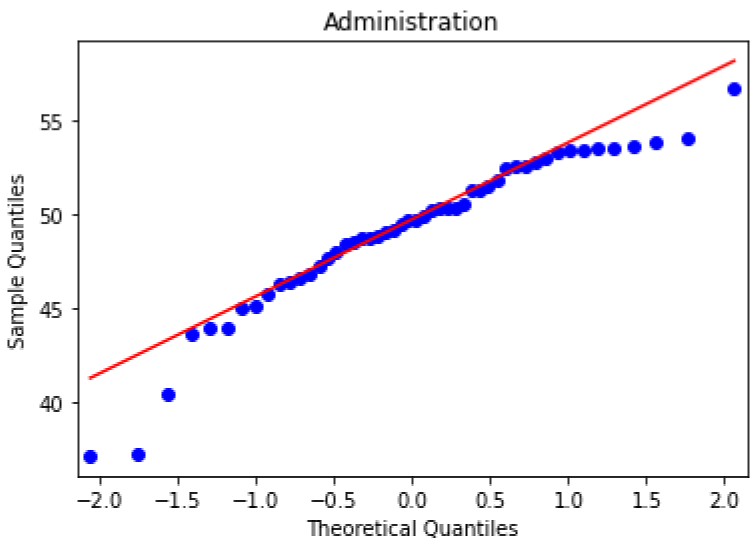
QQ-Plot for Cuberoot Transformation

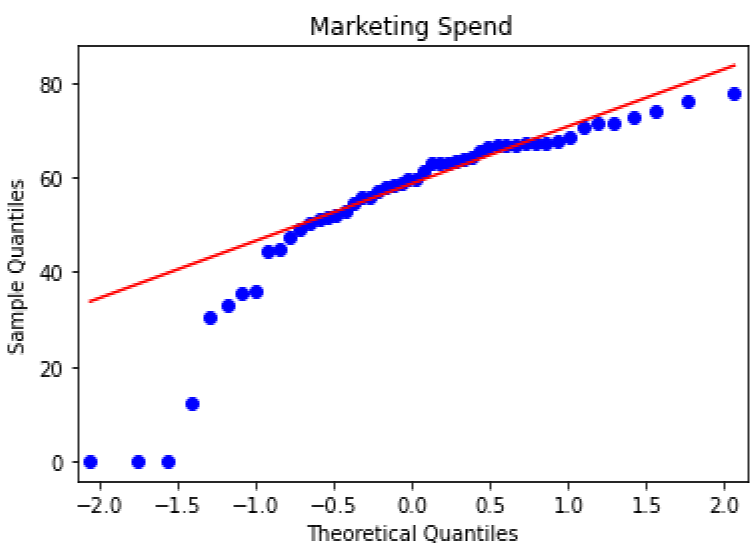
In [16]:

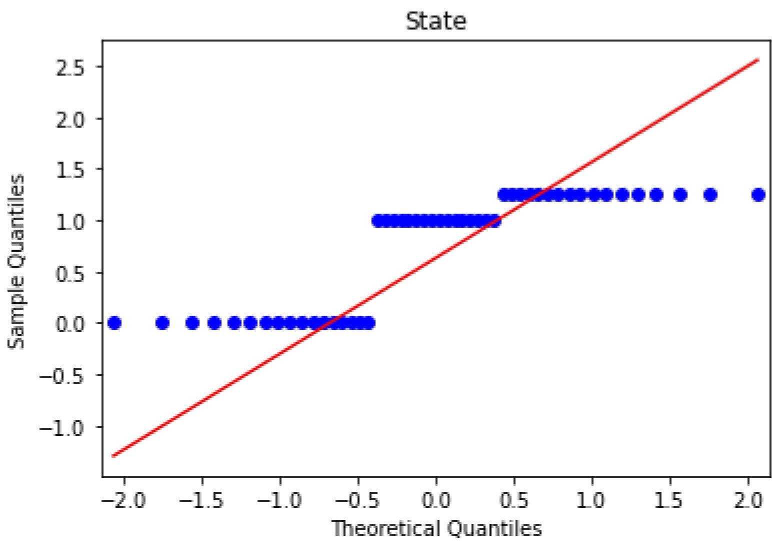
```
for feature in numerical_feature:
    data = df.copy()
    sm.qqplot(np.cbrt(data[feature]),line="q")
    plt.title(feature)
```

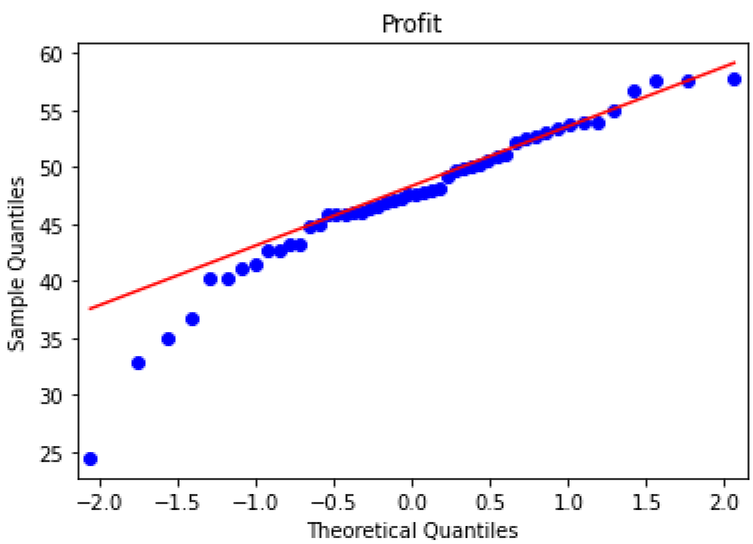
Loading [MathJax]/extensions/Safe.js


R&D Spend
Sample Quantiles
Theoretical Quantiles
0
10
20
30
40
50
60
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Administration
Sample Quantiles
Theoretical Quantiles
40
45
50
55
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Marketing Spend
Sample Quantiles
Theoretical Quantiles
0
20
40
60
80
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

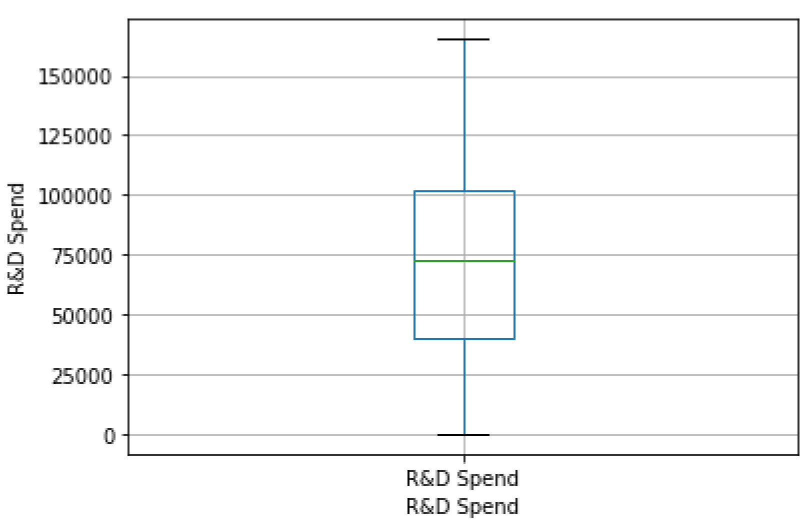


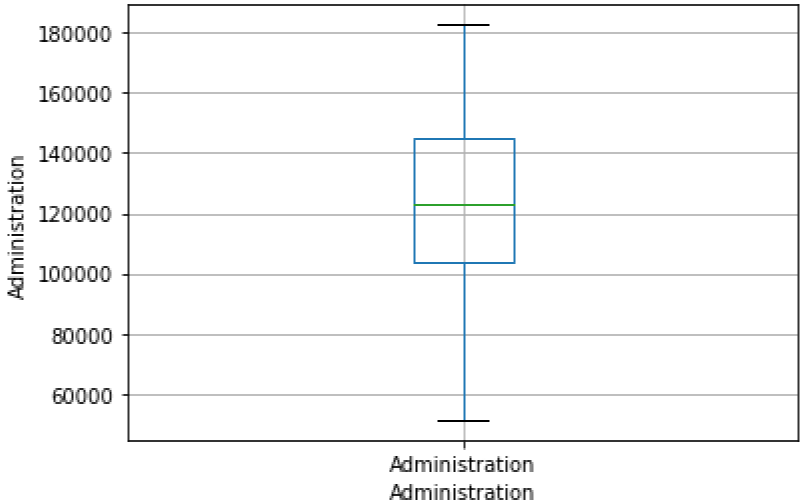


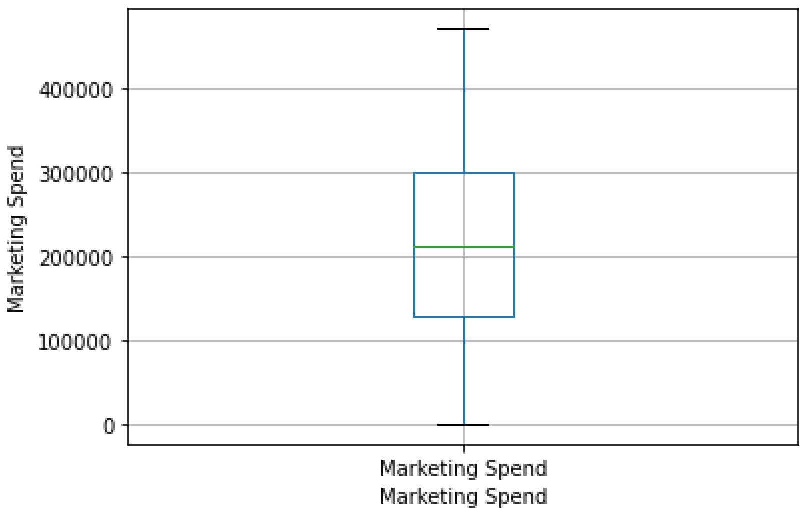
Checking Outliers using Raw Data

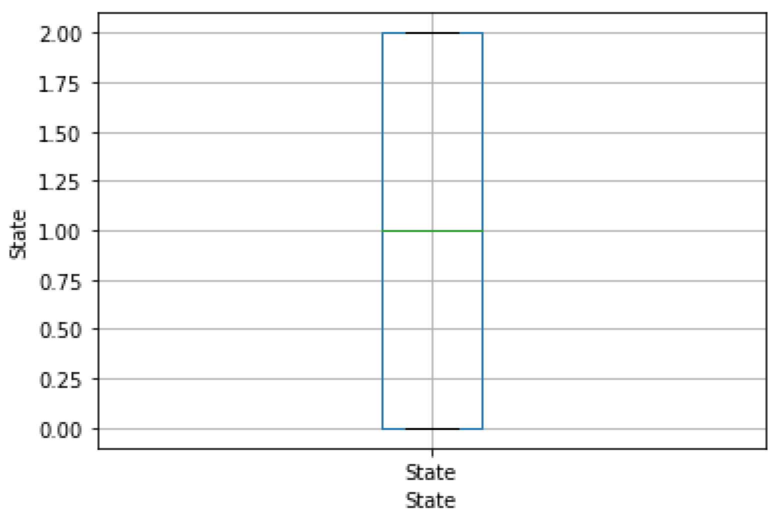
In [18]:

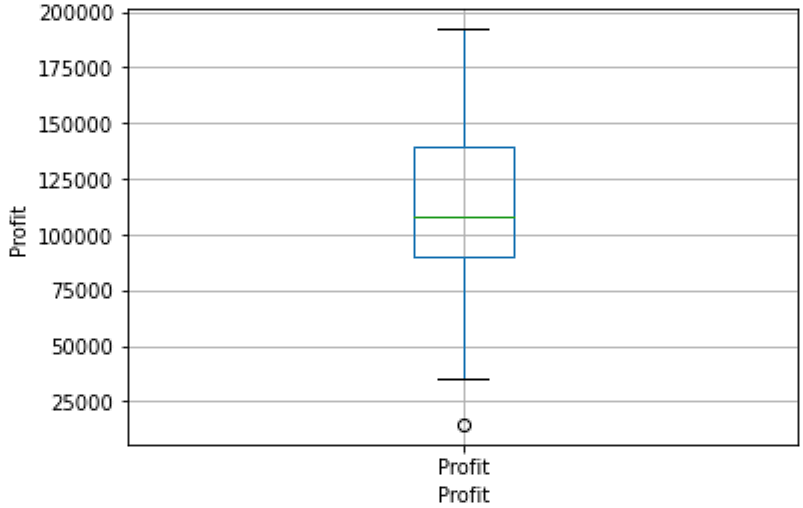
```
for feature in numerical_feature:
    data = df.copy()
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```




Administration
180000
160000
140000
120000
100000
80000
60000
Administration
Administration


Marketing Spend
400000
300000
200000
100000
0
Marketing Spend
Marketing Spend

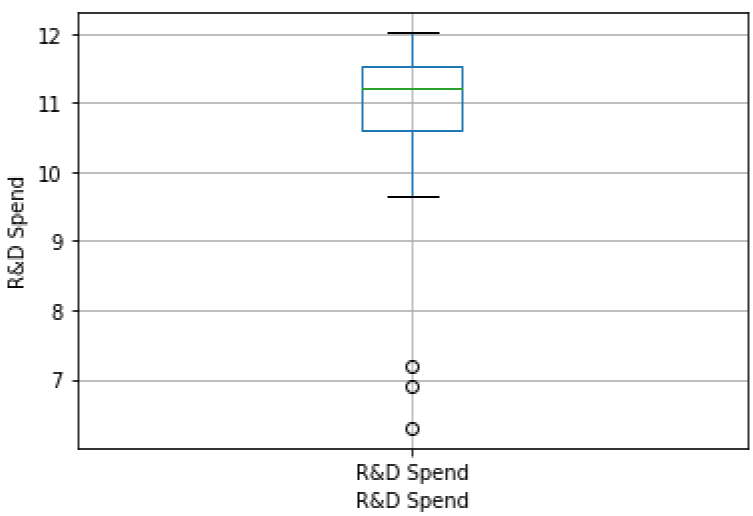

State
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
State
State

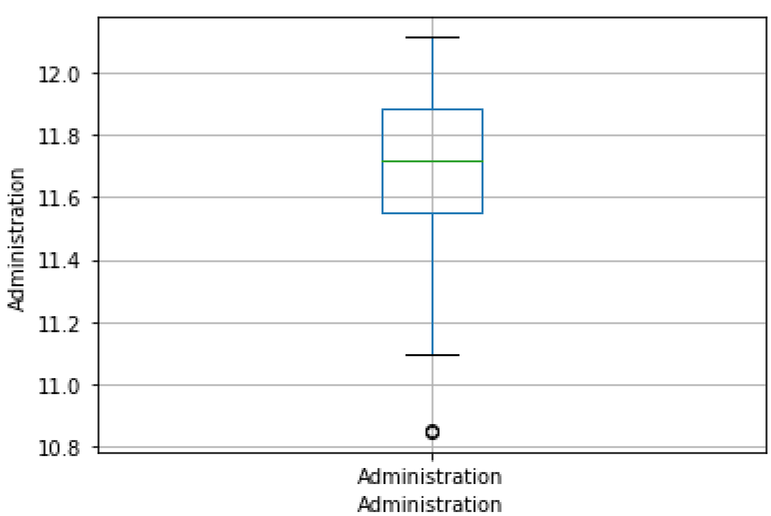


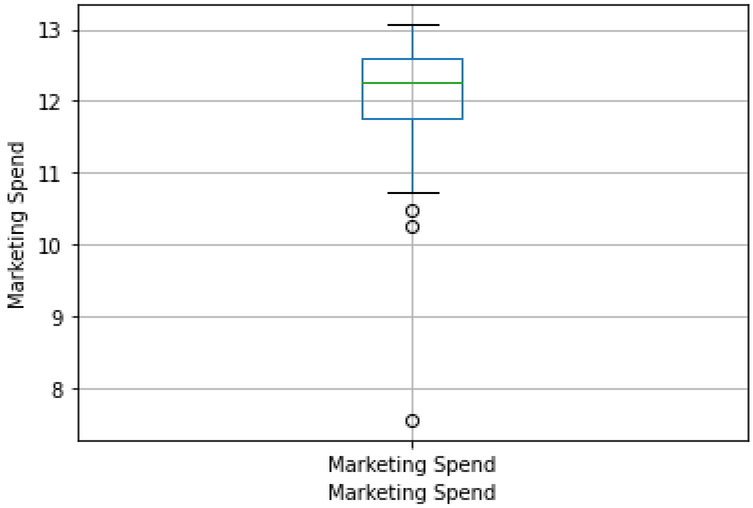
Checking Outliers Using Log Transformation

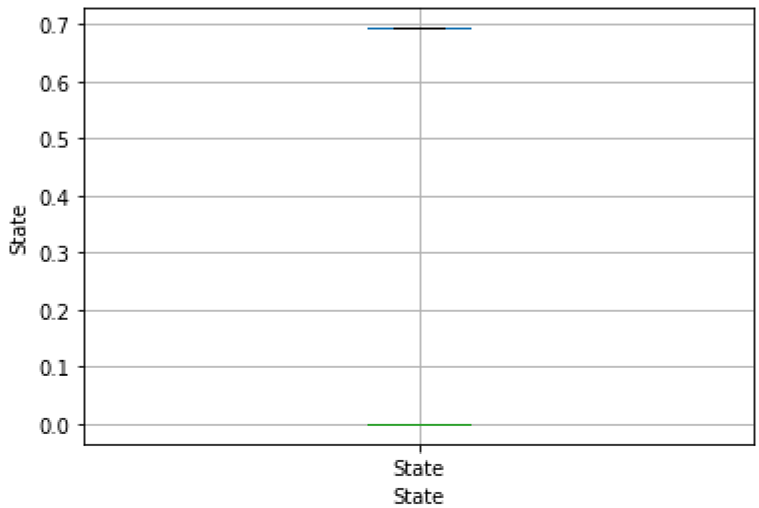
In [17]:

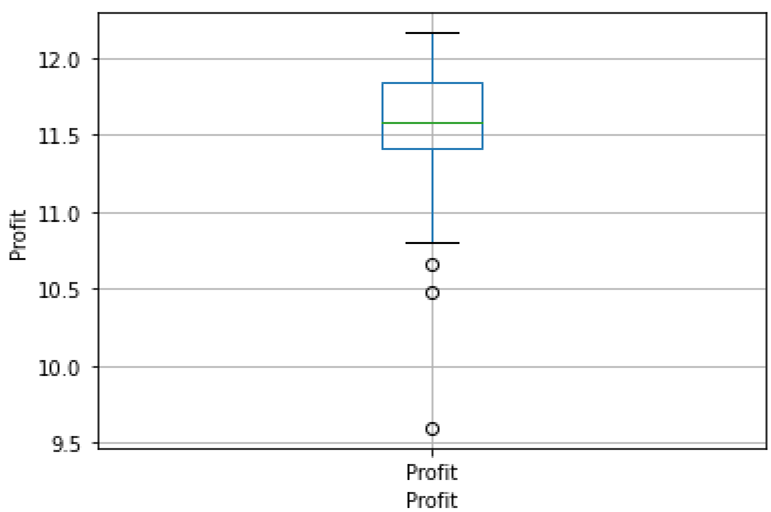
```
for feature in numerical_feature:
    data = df.copy()
    data[feature]=np.log(data[feature])
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```







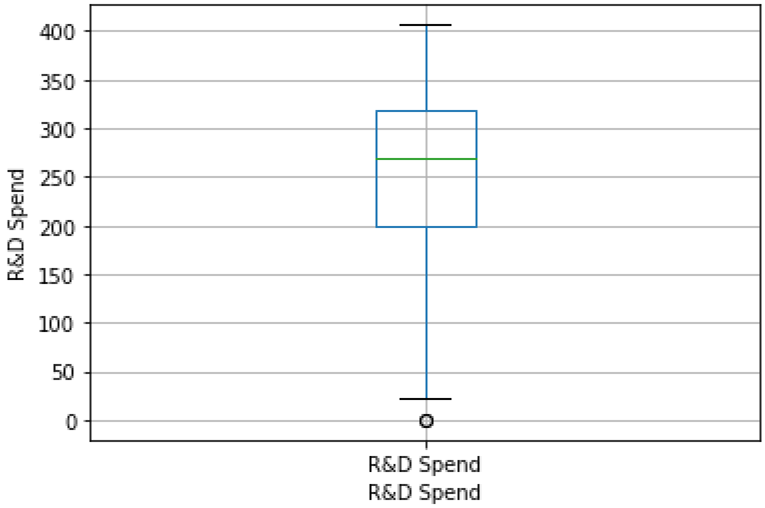


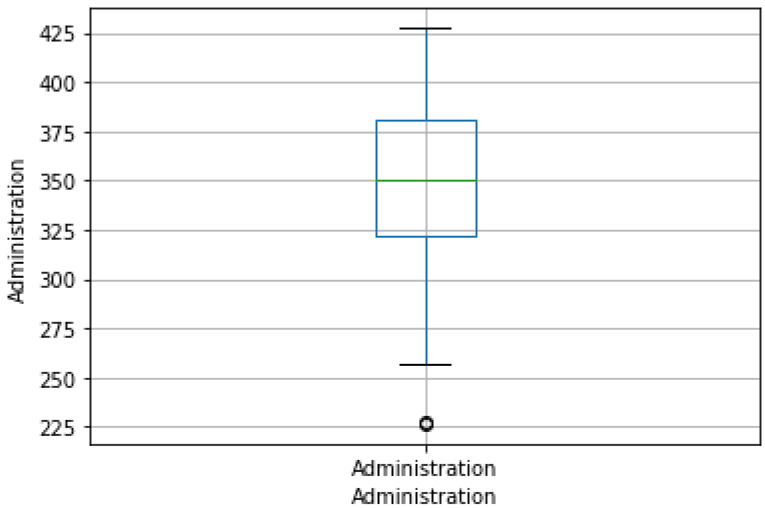


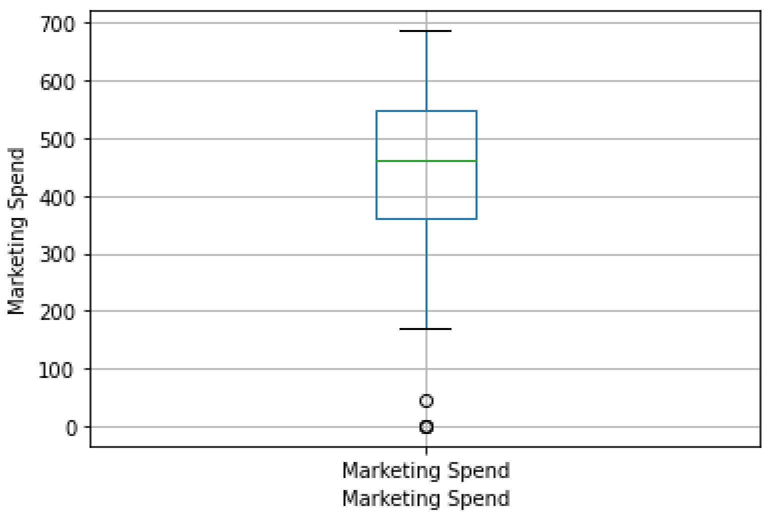
Checking Outliers using Squareroot Tarnsformation

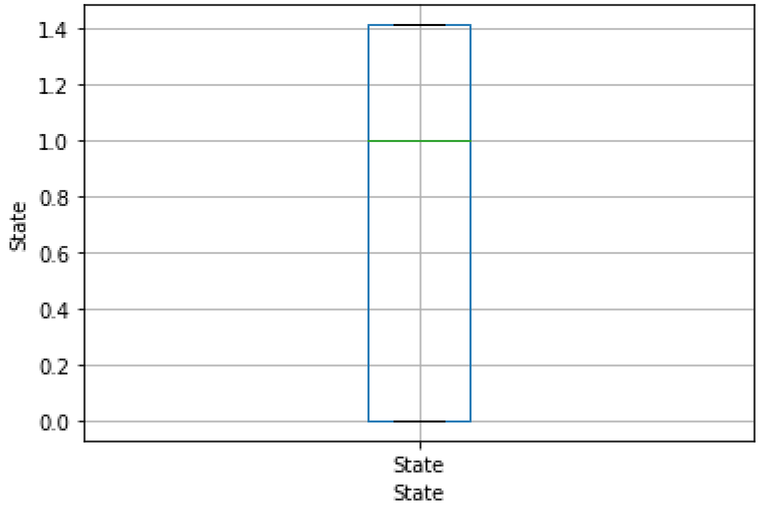
In [18]:

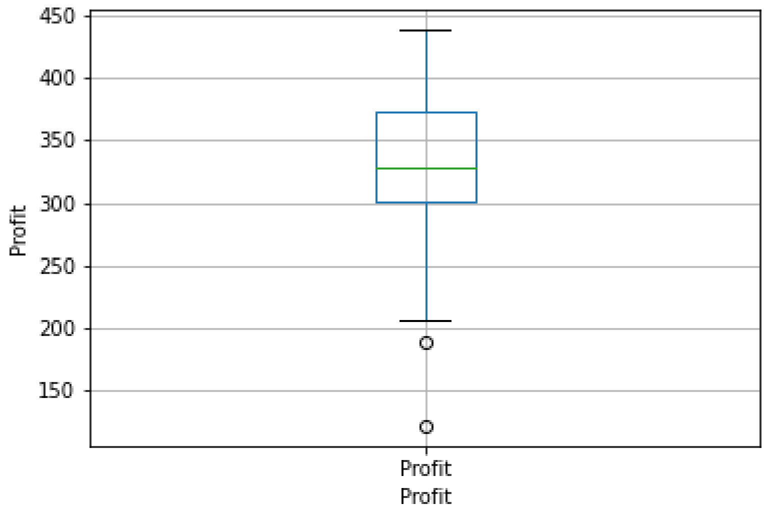
```
for feature in numerical_feature:
    data = df.copy()
    data[feature]=np.sqrt(data[feature])
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```


R&D Spend
400
350
300
250
200
150
100
50
0
R&D Spend
R&D Spend


Administration
425
400
375
350
325
300
275
250
225
Administration
Administration


Marketing Spend
700
600
500
400
300
200
100
0
Marketing Spend
Marketing Spend

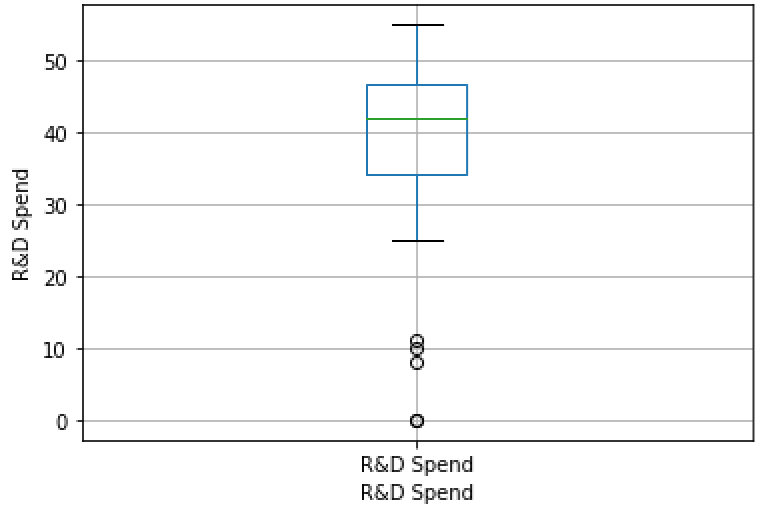


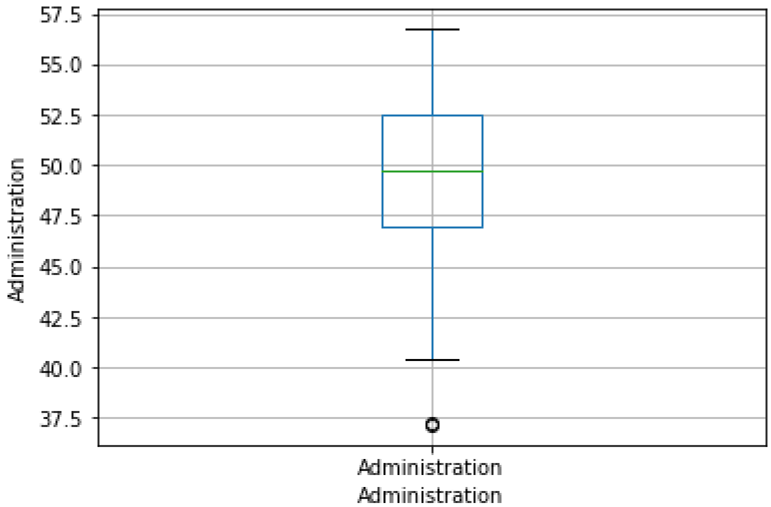


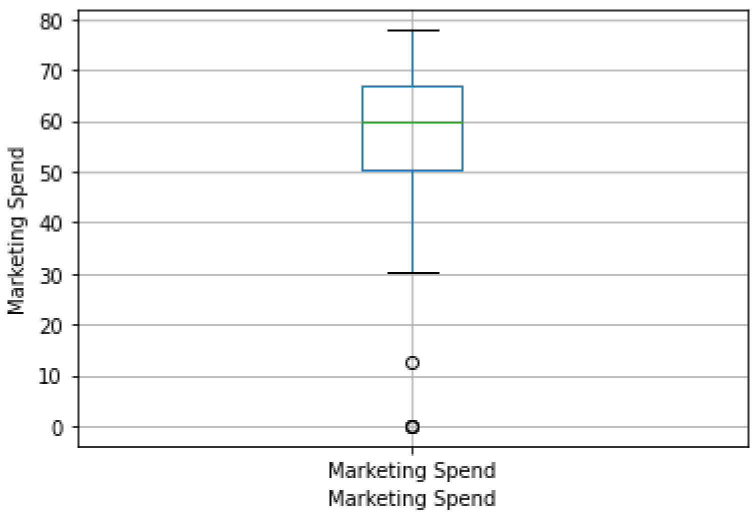
Checking Outliers using Cuberoot Transformation

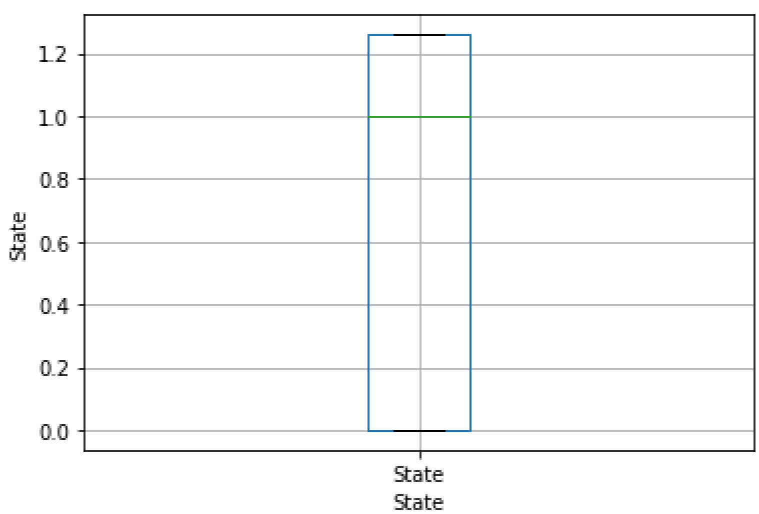
In [19]:

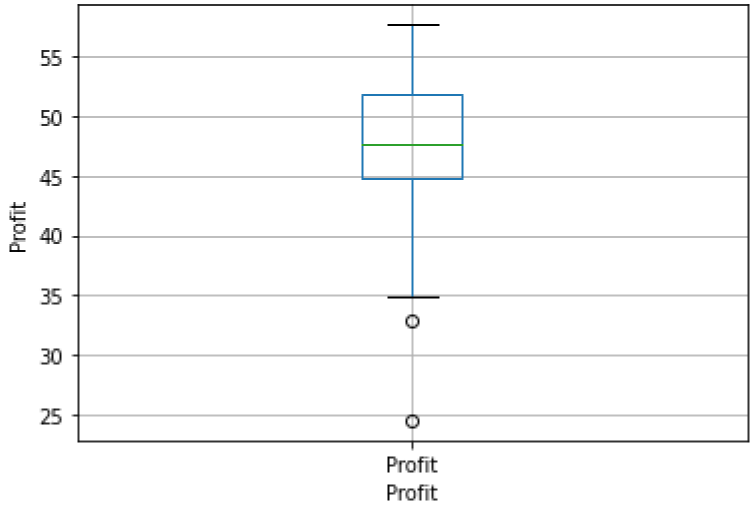
```
for feature in numerical_feature:
    data = df.copy()
    data[feature]=np.cbrt(data[feature])
    data.boxplot(column=feature)
    plt.xlabel(feature)
    plt.ylabel(feature)
    plt.show()
```




57.5
55.0
52.5
50.0
47.5
45.0
42.5
40.0
37.5
Administration
Administration
Administration


80
70
60
50
40
30
20
10
0
Marketing Spend
Marketing Spend
Marketing Spend


1.2
1.0
0.8
0.6
0.4
0.2
0.0
State
State
State



Observation:

Hence we see that the raw data having less outliers and as a result they are normally distributed as well but we do transformation in the data we can see the outliers and the little bit skewness in the data. so we choose the raw data because it is normally distributed and having less outliers compared to transformation and also the state column cannot be usefull in this data set so we can drop the state column because it cannot contributing much in the data set for predicting

Checking Co-linearity in the Data

In [20]:

```
df.corr()
```

Out[20]:

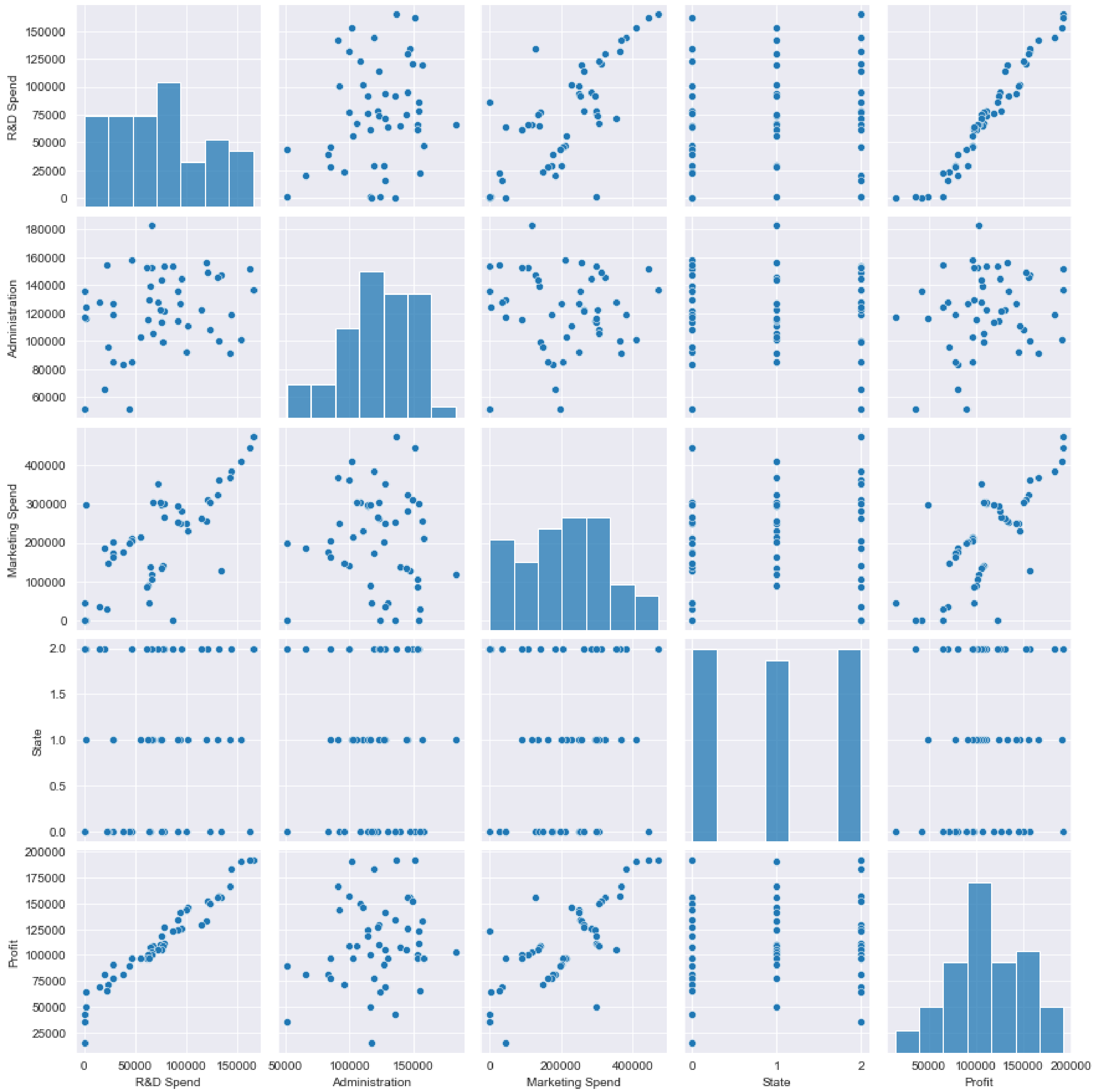
	R&D Spend	Administration	Marketing Spend	State	Profit
R&D Spend	1.000000	0.241955	0.724248	0.104685	0.972900
Administration	0.241955	1.000000	-0.032154	0.011847	0.200717
Marketing Spend	0.724248	-0.032154	1.000000	0.077670	0.747766
State	0.104685	0.011847	0.077670	1.000000	0.101796
Profit	0.972900	0.200717	0.747766	0.101796	1.000000

In [21]:

```
sns.set_style(style = 'darkgrid')
sns.pairplot(df)
```

Out[21]:

```
<seaborn.axisgrid.PairGrid at 0x26267e9dca0>
```



Observation:

As we can see that the state is not contributing as much in the dataset so we can remove state column.R&D and Profit are higly correalted.R&D and Marketing also correlated|

In [22]:

```
df.drop("State",axis=1,inplace=True)
```

In [23]:

```
df
```

Out[23]:

	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

Loading [MathJax]/extensions/Safe.js

	R&D Spend	Administration	Marketing Spend	Profit
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
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

	R&D Spend	Administration	Marketing Spend	Profit
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

In [24]:

```
df1 = df.rename({"R&D Spend":"RDS","Marketing Spend":"Marketing"},axis=1)
df1
```

Out[24]:

	RDS	Administration	Marketing	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
[illegible]	[illegible]	144135.98	134050.07	105733.54

	RDS	Administration	Marketing	Profit
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
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

Creating the first model

In [25]:

```
model1 = smf.ols("Profit~RDS+Administration+Marketing",data=df1).fit()
```

In [26]:

```
model1.summary()
```

Out[26]:

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:	Sun, 10 Apr 2022	**Prob (F-statistic):**	4.53e-30
Time:	19:47:24	**Log-Likelihood:**	-525.39
No. Observations:	50	**AIC:**	1059.
Df Residuals:	46	**BIC:**	1066.
Df Model:	3		
	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
Administration	-0.0268	0.051	-0.526	0.602	-0.130	0.076
Marketing	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.

In [27]:
```
model1.params
```

Out[27]:
```
Intercept         50122.192990
RDS                   0.805715
Administration       -0.026816
Marketing             0.027228
dtype: float64
```

Simple Linear Regression

In [28]:
```
model2 = smf.ols("Profit~Administration",data=df1).fit()
```

In [29]:
```
model2.summary()
```

Out[29]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.040
Model:	OLS	**Adj. R-squared:**	0.020
Method:	Least Squares	**F-statistic:**	2.015
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.162
Time:	19:47:27	**Log-Likelihood:**	-599.63
No. Observations:	50	**AIC:**	1203.
Df Residuals:	48	**BIC:**	1207.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.697e+04	2.53e+04	3.040	0.004	2.61e+04	1.28e+05

Administration	0.2887	0.203	1.419	0.162	-0.120	0.698

Omnibus:	0.126	**Durbin-Watson:**	0.099
Prob(Omnibus):	0.939	**Jarque-Bera (JB):**	0.110
Skew:	0.093	**Prob(JB):**	0.947
Kurtosis:	2.866	**Cond. No.**	5.59e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.59e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [30]:

```
model3 = smf.ols("Profit~Marketing",data=df1).fit()
model3.summary()
```

Out[30]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.559
Model:	OLS	**Adj. R-squared:**	0.550
Method:	Least Squares	**F-statistic:**	60.88
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	4.38e-10
Time:	19:47:27	**Log-Likelihood:**	-580.18
No. Observations:	50	**AIC:**	1164.
Df Residuals:	48	**BIC:**	1168.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	6e+04	7684.530	7.808	0.000	4.46e+04	7.55e+04
Marketing	0.2465	0.032	7.803	0.000	0.183	0.310

Omnibus:	4.420	**Durbin-Watson:**	1.178
Prob(Omnibus):	0.110	**Jarque-Bera (JB):**	3.882
Skew:	-0.336	**Prob(JB):**	0.144
Kurtosis:	4.188	**Cond. No.**	4.89e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.89e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [31]:

```
model4 = smf.ols("Profit~RDS",data=df1).fit()
model4.summary()
```

Out[31]:

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:	Sun, 10 Apr 2022	**Prob (F-statistic):**	3.50e-32
Time:	19:47:28	**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 [32]:

```
model5= smf.ols("Profit~Administration+Marketing",data=df1).fit()
model5.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.610
Model:	OLS	**Adj. R-squared:**	0.593
Method:	Least Squares	**F-statistic:**	36.71
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	2.50e-10
Time:	19:47:28	**Log-Likelihood:**	-577.13
No. Observations:	50	**AIC:**	1160.
Df Residuals:	47	**BIC:**	1166.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	2.022e+04	1.77e+04	1.143	0.259	-1.54e+04	5.58e+04
Administration	0.3237	0.131	2.468	0.017	0.060	0.588
[illegible]	0.2488	0.030	8.281	0.000	0.188	0.309

Omnibus:	6.584	**Durbin-Watson:**	1.279
Prob(Omnibus):	0.037	**Jarque-Bera (JB):**	6.524
Skew:	-0.512	**Prob(JB):**	0.0383
Kurtosis:	4.443	**Cond. No.**	1.30e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.3e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Transforming Data InTo Standard Scaler for Better Results

In [33]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [34]:

```
df2=sc.fit_transform(df1)
df2
```

Out[34]:

```
array([[ 2.01641149e+00,  5.60752915e-01,  2.15394309e+00,
         2.01120333e+00],
       [ 1.95586034e+00,  1.08280658e+00,  1.92360040e+00,
         1.99942997e+00],
       [ 1.75436374e+00, -7.28257028e-01,  1.62652767e+00,
         1.98084225e+00],
       [ 1.55478369e+00, -9.63646307e-02,  1.42221024e+00,
         1.77662724e+00],
       [ 1.50493720e+00, -1.07991935e+00,  1.28152771e+00,
         1.35774012e+00],
       [ 1.27980001e+00, -7.76239071e-01,  1.25421046e+00,
         1.12724963e+00],
       [ 1.34006641e+00,  9.32147208e-01, -6.88149930e-01,
         1.10548055e+00],
       [ 1.24505666e+00,  8.71980011e-01,  9.32185978e-01,
         1.09620987e+00],
       [ 1.03036886e+00,  9.86952101e-01,  8.30886909e-01,
         1.00746967e+00],
       [ 1.09181921e+00, -4.56640246e-01,  7.76107440e-01,
         9.46022467e-01],
       [ 6.20398248e-01, -3.87599089e-01,  1.49807267e-01,
         8.54846746e-01],
       [ 5.93085418e-01, -1.06553960e+00,  3.19833623e-01,
         8.08167561e-01],
       [ 4.43259872e-01,  2.15449064e-01,  3.20617441e-01,
         7.41154844e-01],
       [ 4.02077603e-01,  5.10178953e-01,  3.43956788e-01,
         5.58749518e-01],
       [ 1.01718075e+00,  1.26919939e+00,  3.75742273e-01,
         5.16026367e-01],
       [ 8.97913123e-01,  4.58678535e-02,  4.19218702e-01,
         4.48719672e-01],
       [ 9.44411957e-02,  9.11841968e-03,  4.40446224e-01,
         3.75435696e-01],
       [ 4.60720127e-01,  8.55666318e-01,  5.91016724e-01,
         3.34771135e-01],
       [ 2.96724938e-01, -2.58465367e-01,  6.92992062e-01,
```

```
        3.07115996e-01],
       [ 2.79441650e-01,  1.15983657e+00, -1.74312698e+00,
         2.69772649e-01],
       [ 5.57260867e-02, -2.69587651e-01,  7.23925995e-01,
         1.61935224e-01],
       [ 1.02723599e-01,  1.16918609e+00,  7.32787791e-01,
        -1.75338400e-02],
       [ 6.00657792e-03,  5.18495648e-02,  7.62375876e-01,
        -4.16126351e-02],
       [-1.36200724e-01, -5.62211268e-01,  7.74348908e-01,
        -8.21694292e-02],
       [ 7.31146008e-02, -7.95469167e-01, -5.81939297e-01,
        -8.67294558e-02],
       [-1.99311688e-01,  6.56489139e-01, -6.03516725e-01,
        -1.15493086e-01],
       [ 3.53702028e-02,  8.21717916e-01, -6.35835495e-01,
        -1.57366637e-01],
       [-3.55189938e-02,  2.35068543e-01,  1.17427116e+00,
        -1.75542334e-01],
       [-1.68792717e-01,  2.21014050e+00, -7.67189437e-01,
        -2.18797551e-01],
       [-1.78608540e-01,  1.14245677e+00, -8.58133663e-01,
        -2.75882217e-01],
       [-2.58074369e-01, -2.05628659e-01, -9.90357166e-01,
        -3.02624599e-01],
       [-2.76958231e-01,  1.13055391e+00, -1.01441945e+00,
        -3.64127442e-01],
       [-2.26948675e-01,  2.83923813e-01, -1.36244978e+00,
        -3.65523895e-01],
       [-4.01128925e-01, -6.59324033e-01,  2.98172434e-02,
        -3.81787113e-01],
       [-6.00682122e-01,  1.31053525e+00, -1.87861793e-03,
        -3.83444211e-01],
       [-6.09749941e-01, -1.30865753e+00, -4.54931587e-02,
        -3.89290919e-01],
       [-9.91570153e-01,  2.05924691e-01, -8.17625734e-02,
        -5.33931605e-01],
       [-6.52532310e-01, -2.52599402e+00, -1.15608256e-01,
        -5.52954899e-01],
       [-1.17717755e+00, -1.99727037e+00, -2.12784866e-01,
        -7.71497339e-01],
       [-7.73820359e-01, -1.38312156e+00, -2.97583276e-01,
        -7.77093678e-01],
       [-9.89577015e-01, -1.00900218e-01, -3.15785883e-01,
        -8.46411346e-01],
       [-1.00853372e+00, -1.32079581e+00, -3.84552407e-01,
        -8.57465682e-01],
       [-1.10210556e+00, -9.06937535e-01, -5.20595959e-01,
        -1.01536466e+00],
       [-1.28113364e+00,  2.17681524e-01, -1.44960468e+00,
        -1.05896021e+00],
       [-1.13430539e+00,  1.20641936e+00, -1.50907418e+00,
        -1.17320899e+00],
       [-1.60035036e+00,  1.01253936e-01, -1.72739998e+00,
        -1.18008224e+00],
       [-1.59341322e+00, -1.99321741e-01,  7.11122474e-01,
        -1.56692212e+00],
       [-1.62236202e+00,  5.07721876e-01, -1.74312698e+00,
        -1.74062718e+00],
       [-1.61043334e+00, -2.50940884e+00, -1.74312698e+00,
        -1.91321197e+00],
       [-1.62236202e+00, -1.57225506e-01, -1.36998473e+00,
        -2.43931323e+00]])
```

Again Building Model in Standard Scaler

In [35]:

```
data = pd.DataFrame(df2,columns=["RDS","Administration","Marketing","Profit"])
```

```
data
```

Out[35]:

	RDS	Administration	Marketing	Profit
0	2.016411	0.560753	2.153943	2.011203
1	1.955860	1.082807	1.923600	1.999430
2	1.754364	-0.728257	1.626528	1.980842
3	1.554784	-0.096365	1.422210	1.776627
4	1.504937	-1.079919	1.281528	1.357740
5	1.279800	-0.776239	1.254210	1.127250
6	1.340066	0.932147	-0.688150	1.105481
7	1.245057	0.871980	0.932186	1.096210
8	1.030369	0.986952	0.830887	1.007470
9	1.091819	-0.456640	0.776107	0.946022
10	0.620398	-0.387599	0.149807	0.854847
11	0.593085	-1.065540	0.319834	0.808168
12	0.443260	0.215449	0.320617	0.741155
13	0.402078	0.510179	0.343957	0.558750
14	1.017181	1.269199	0.375742	0.516026
15	0.897913	0.045868	0.419219	0.448720
16	0.094441	0.009118	0.440446	0.375436
17	0.460720	0.855666	0.591017	0.334771
18	0.396725	-0.258465	0.692992	0.307116
19	0.279442	1.159837	-1.743127	0.269773
20	0.055726	-0.269588	0.723926	0.161935
21	0.102724	1.169186	0.732788	-0.017534
22	0.006007	0.051850	0.762376	-0.041613
23	-0.136201	-0.562211	0.774349	-0.082169
24	0.073115	-0.795469	-0.581939	-0.086729
25	-0.199312	0.656489	-0.603517	-0.115493
26	0.035370	0.821718	-0.635835	-0.157367
27	-0.035519	0.235069	1.174271	-0.175542
28	-0.168793	2.210141	-0.767189	-0.218798
29	-0.178609	1.142457	-0.858134	-0.275882
30	-0.258074	-0.205629	-0.990357	-0.302625
31	-0.276958	1.130554	-1.014419	-0.364127
32	-0.226949	0.283924	-1.362450	-0.365524
33	-0.401129	-0.659324	0.029817	-0.381787
34	-0.600682	1.310535	-0.001879	-0.383444
35	-0.609750	-1.308658	-0.045493	-0.389291
36	-0.991570	0.205925	-0.081763	-0.533932
37	[illegible]	-2.525994	-0.115608	-0.552955

	RDS	Administration	Marketing	Profit
38	-1.177178	-1.997270	-0.212785	-0.771497
39	-0.773820	-1.383122	-0.297583	-0.777094
40	-0.989577	-0.100900	-0.315786	-0.846411
41	-1.008534	-1.320796	-0.384552	-0.857466
42	-1.102106	-0.906938	-0.520596	-1.015365
43	-1.281134	0.217682	-1.449605	-1.058960
44	-1.134305	1.206419	-1.509074	-1.173209
45	-1.600350	0.101254	-1.727400	-1.180082
46	-1.593413	-0.199322	0.711122	-1.566922
47	-1.622362	0.507722	-1.743127	-1.740627
48	-1.610433	-2.509409	-1.743127	-1.913212
49	-1.622362	-0.157226	-1.369985	-2.439313

In [36]:

```
model6 = smf.ols("Profit~RDS+Administration+Marketing",data=data).fit()
model6.summary()
```

Out[36]:

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:	Sun, 10 Apr 2022	**Prob (F-statistic):**	4.53e-30
Time:	19:47:32	**Log-Likelihood:**	4.3222
No. Observations:	50	**AIC:**	-0.6444
Df Residuals:	46	**BIC:**	7.004
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-4.927e-16	0.033	-1.51e-14	1.000	-0.066	0.066
RDS	0.9176	0.051	17.846	0.000	0.814	1.021
Administration	-0.0186	0.035	-0.526	0.602	-0.090	0.053
Marketing	0.0826	0.050	1.655	0.105	-0.018	0.183

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.**	2.78

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [37]:
```python
model7 = smf.ols("Profit~RDS",data=data).fit()
model7.summary()
```

Out[37]:

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:	Sun, 10 Apr 2022	**Prob (F-statistic):**	3.50e-32
Time:	19:47:33	**Log-Likelihood:**	2.2714
No. Observations:	50	**AIC:**	-0.5428
Df Residuals:	48	**BIC:**	3.281
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5.274e-16	0.033	-1.58e-14	1.000	-0.067	0.067
RDS	0.9729	0.033	29.151	0.000	0.906	1.040

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.**	1.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [38]:
```python
model8=smf.ols('Profit~Administration',data=data).fit()
model8.summary()
```

Out[38]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.040
Model:	OLS	**Adj. R-squared:**	0.020
Method:	Least Squares	**F-statistic:**	2.015
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	0.162
Time:	19:47:33	**Log-Likelihood:**	-69.919
No. Observations:	50	**AIC:**	143.8
Df Residuals:	48	**BIC:**	147.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5.274e-16	0.141	-3.73e-15	1.000	-0.284	0.284
Administration	0.2007	0.141	1.419	0.162	-0.084	0.485

Omnibus:	0.126	**Durbin-Watson:**	0.099
Prob(Omnibus):	0.939	**Jarque-Bera (JB):**	0.110
Skew:	0.093	**Prob(JB):**	0.947
Kurtosis:	2.866	**Cond. No.**	1.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [39]:

```
model8=smf.ols("Profit~Marketing",data=data).fit()
model8.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.559
Model:	OLS	**Adj. R-squared:**	0.550
Method:	Least Squares	**F-statistic:**	60.88
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	4.38e-10
Time:	19:47:33	**Log-Likelihood:**	-50.470
No. Observations:	50	**AIC:**	104.9
Df Residuals:	48	**BIC:**	108.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-5.274e-16	0.096	-5.5e-15	1.000	-0.193	0.193
Marketing	0.7478	0.096	7.803	0.000	0.555	0.940

Omnibus:	4.420	**Durbin-Watson:**	1.178
Prob(Omnibus):	0.110	**Jarque-Bera (JB):**	3.882
Skew:	-0.336	**Prob(JB):**	0.144
Kurtosis:	4.188	**Cond. No.**	1.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Calculating VIF

In [40]:

```
rsq_RandD = smf.ols('RDS~Administration+Marketing',data=data).fit().rsquared
vif_RandD = 1/(1-rsq_RandD)

rsq_admin = smf.ols('Administration~RDS+Marketing',data=data).fit().rsquared
vif_admin = 1/(1-rsq_admin)

rsq_marketing = smf.ols('Marketing~RDS+Administration',data=data).fit().rsquared
vif_marketing = 1/(1-rsq_marketing)
```

```
# Storing vif values in a data frame
d1 = {'Variables':['RDS','Administration','Marketing'],'VIF':[vif_RandD,vif_admin,vif_mark
Vif_frame = pd.DataFrame(d1)
Vif_frame
```

Out[40]:

	Variables	VIF
0	RDS	2.468903
1	Administration	1.175091
2	Marketing	2.326773

Observations

The vif is less hence as a result there is no colinearity in the features

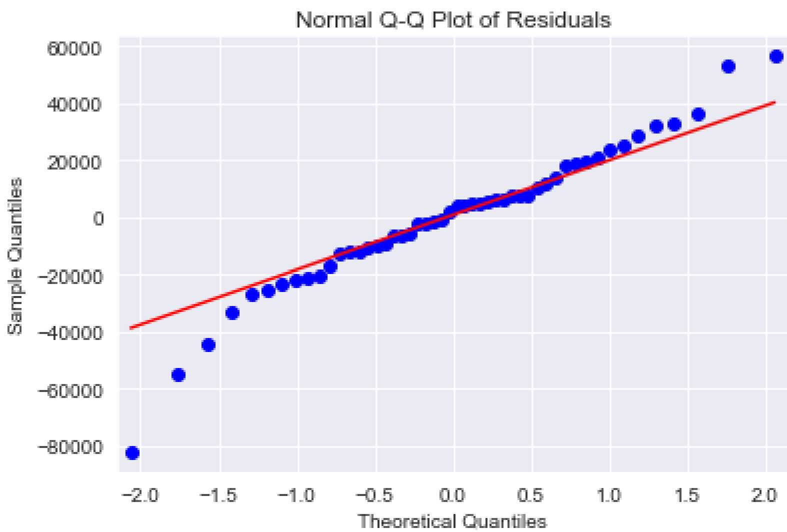
Residual Analysis

In [41]:

```
import statsmodels.api as sm
```

In [42]:

```
qqplot = sm.qqplot(model5.resid,line="q")
plt.title("Normal Q-Q Plot of Residuals")
plt.show()
```



Observation:

Errors are coming from normal distribution

Residuals For Homoscedasticity

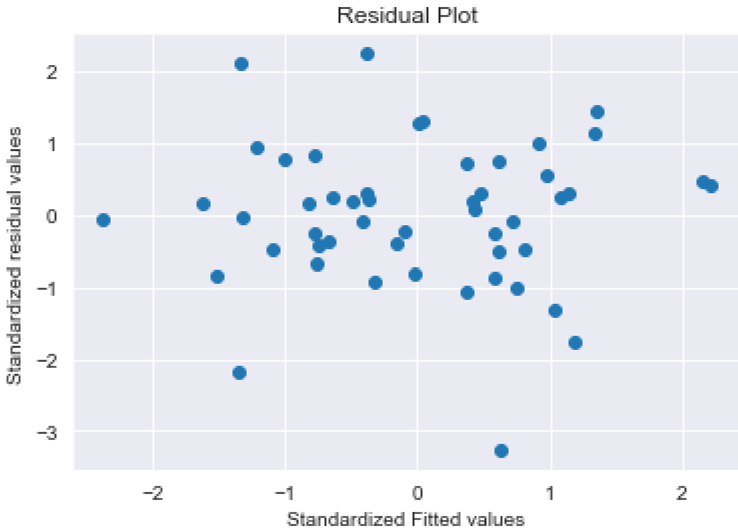
In [43]:

```
def get_standardized_values( vals ):
    return (vals - vals.mean())/vals.std()
```

Loading [MathJax]/extensions/Safe.js

In [44]:

```
plt.scatter(get_standardized_values(model5.fittedvalues),
            get_standardized_values(model5.resid))

plt.title('Residual Plot')
plt.xlabel('Standardized Fitted values')
plt.ylabel('Standardized residual values')
plt.show()
```



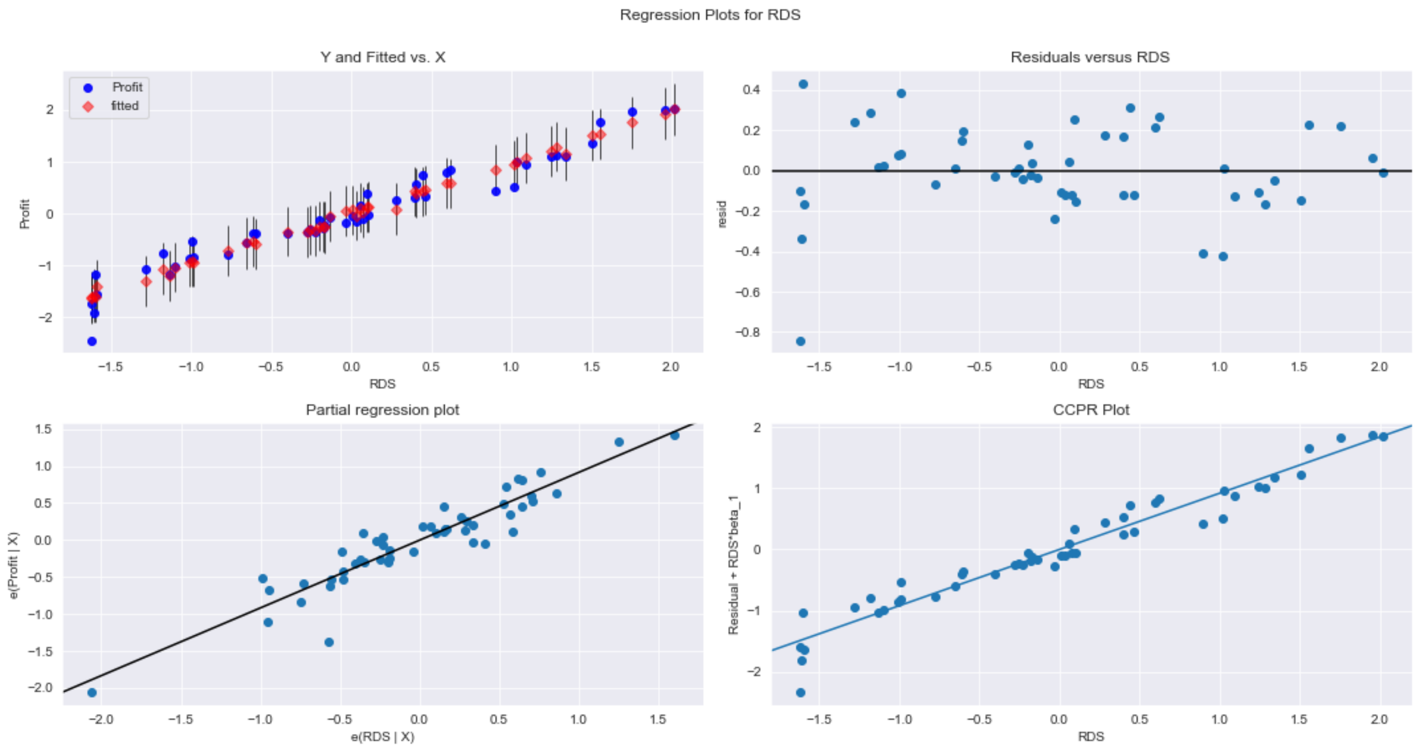
Observation:

So we can see that there is no such pattern is creating and the data is randomly scattered so this data is an homoscedasticity

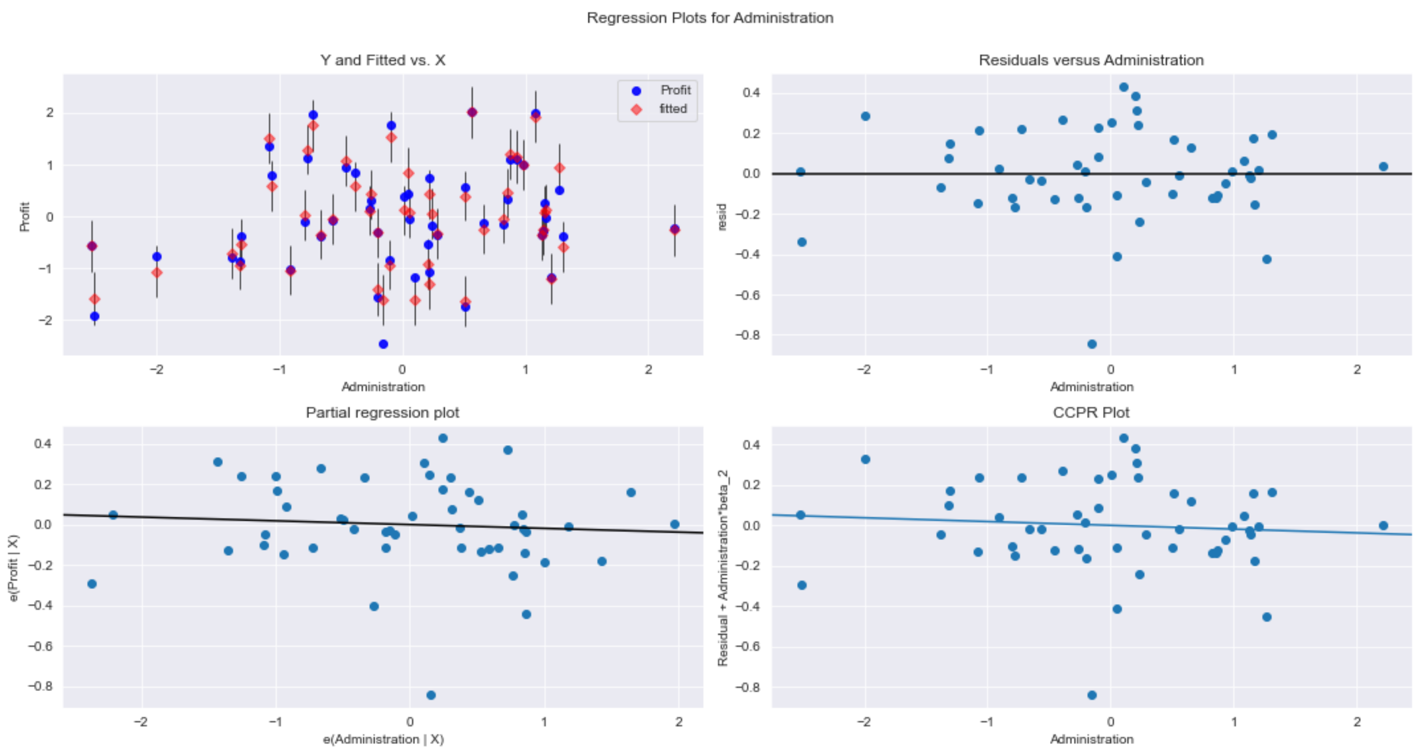
Residual vs Regressor

In [45]:

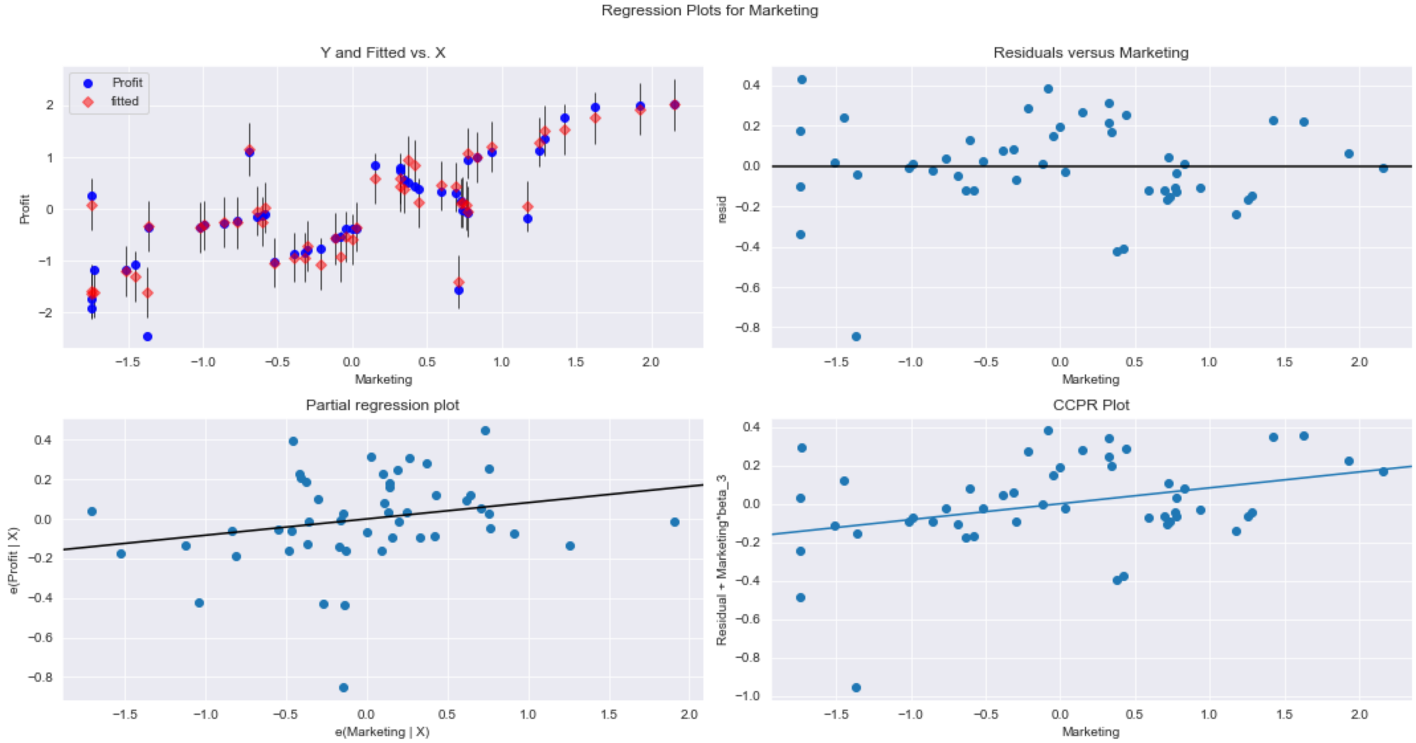
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model6, "RDS",fig=fig)
plt.show()
```



In [46]:
```python
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model6, "Administration",fig=fig)
plt.show()
```



In [47]:
```python
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model6, "Marketing",fig=fig)
plt.show()
```



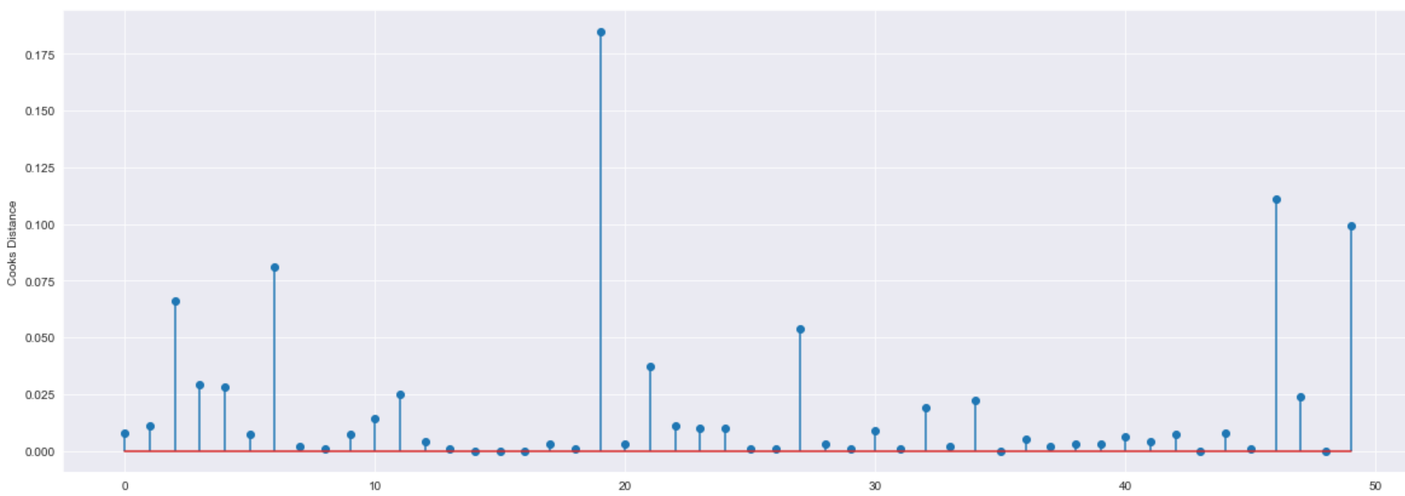
Outlier Detection

In [48]:

```
model5_influence=model5.get_influence()
(c,_)=model5_influence.cooks_distance
```

In [49]:

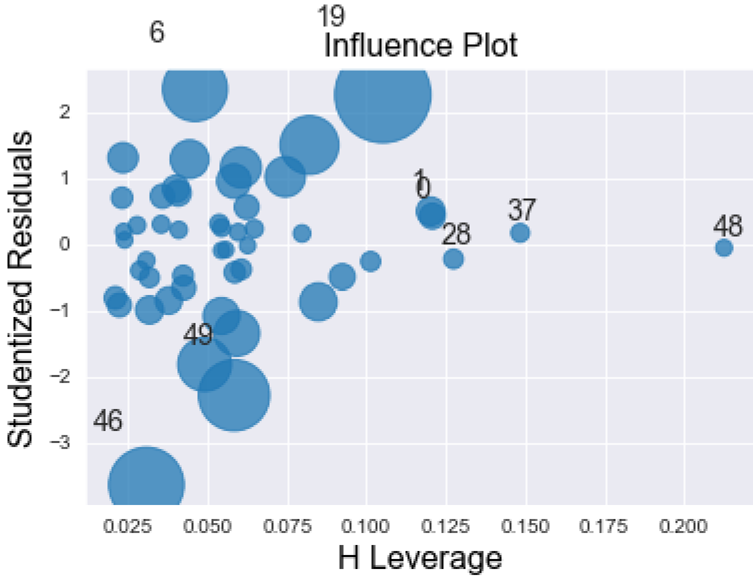
```
fig = plt.subplots(figsize=(20, 7))
plt.stem(np.arange(len(df1)), np.round(c, 3))
plt.xlabel('Row index')
plt.ylabel('Cooks Distance')
plt.show()
```



In [50]:

```
from statsmodels.graphics.regressionplots import influence_plot
influence_plot(model5)
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [51]:
```
k = df1.shape[1]
n = df1.shape[0]
leverage_cutoff = 3*((k + 1)/n)
leverage_cutoff
```

Out[51]: 0.30000000000000004

In [52]:
```
(np.argmax(c),np.max(c))
```

Out[52]: (19, 0.18507855145120508)

In [53]:
```
data[df.index.isin([19,45,48,49])]
```

Out[53]:

	RDS	Administration	Marketing	Profit
19	0.279442	1.159837	-1.743127	0.269773
45	-1.600350	0.101254	-1.727400	-1.180082
48	-1.610433	-2.509409	-1.743127	-1.913212
49	-1.622362	-0.157226	-1.369985	-2.439313

In [54]:
```
data2=data.copy()
data2
```

Out[54]:

	RDS	Administration	Marketing	Profit
0	2.016411	0.560753	2.153943	2.011203
1	1.955860	1.082807	1.923600	1.999430
2	1.754364	-0.728257	1.626528	1.980842
3	1.554784	-0.096365	1.422210	1.776627
4	1.504937	-1.079919	1.281528	1.357740
5	1.279800	-0.776239	1.254210	1.127250
6	1.340066	0.932147	-0.688150	1.105481
7	1.245057	0.871980	0.932186	1.096210

	RDS	Administration	Marketing	Profit
8	1.030369	0.986952	0.830887	1.007470
9	1.091819	-0.456640	0.776107	0.946022
10	0.620398	-0.387599	0.149807	0.854847
11	0.593085	-1.065540	0.319834	0.808168
12	0.443260	0.215449	0.320617	0.741155
13	0.402078	0.510179	0.343957	0.558750
14	1.017181	1.269199	0.375742	0.516026
15	0.897913	0.045868	0.419219	0.448720
16	0.094441	0.009118	0.440446	0.375436
17	0.460720	0.855666	0.591017	0.334771
18	0.396725	-0.258465	0.692992	0.307116
19	0.279442	1.159837	-1.743127	0.269773
20	0.055726	-0.269588	0.723926	0.161935
21	0.102724	1.169186	0.732788	-0.017534
22	0.006007	0.051850	0.762376	-0.041613
23	-0.136201	-0.562211	0.774349	-0.082169
24	0.073115	-0.795469	-0.581939	-0.086729
25	-0.199312	0.656489	-0.603517	-0.115493
26	0.035370	0.821718	-0.635835	-0.157367
27	-0.035519	0.235069	1.174271	-0.175542
28	-0.168793	2.210141	-0.767189	-0.218798
29	-0.178609	1.142457	-0.858134	-0.275882
30	-0.258074	-0.205629	-0.990357	-0.302625
31	-0.276958	1.130554	-1.014419	-0.364127
32	-0.226949	0.283924	-1.362450	-0.365524
33	-0.401129	-0.659324	0.029817	-0.381787
34	-0.600682	1.310535	-0.001879	-0.383444
35	-0.609750	-1.308658	-0.045493	-0.389291
36	-0.991570	0.205925	-0.081763	-0.533932
37	-0.652532	-2.525994	-0.115608	-0.552955
38	-1.177178	-1.997270	-0.212785	-0.771497
39	-0.773820	-1.383122	-0.297583	-0.777094
40	-0.989577	-0.100900	-0.315786	-0.846411
41	-1.008534	-1.320796	-0.384552	-0.857466
42	-1.102106	-0.906938	-0.520596	-1.015365
43	-1.281134	0.217682	-1.449605	-1.058960
44	-1.134305	1.206419	-1.509074	-1.173209
45	-1.600350	0.101254	-1.727400	-1.180082
46	-1.593413	-0.199322	0.711122	-1.566922

	RDS	Administration	Marketing	Profit
47	-1.622362	0.507722	-1.743127	-1.740627
48	-1.610433	-2.509409	-1.743127	-1.913212
49	-1.622362	-0.157226	-1.369985	-2.439313

In [56]:

```
data2=data.drop(data.index[[19,45,48,49]],axis=0).reset_index(drop=True)
data2
```

Out[56]:

	RDS	Administration	Marketing	Profit
0	2.016411	0.560753	2.153943	2.011203
1	1.955860	1.082807	1.923600	1.999430
2	1.754364	-0.728257	1.626528	1.980842
3	1.554784	-0.096365	1.422210	1.776627
4	1.504937	-1.079919	1.281528	1.357740
5	1.279800	-0.776239	1.254210	1.127250
6	1.340066	0.932147	-0.688150	1.105481
7	1.245057	0.871980	0.932186	1.096210
8	1.030369	0.986952	0.830887	1.007470
9	1.091819	-0.456640	0.776107	0.946022
10	0.620398	-0.387599	0.149807	0.854847
11	0.593085	-1.065540	0.319834	0.808168
12	0.443260	0.215449	0.320617	0.741155
13	0.402078	0.510179	0.343957	0.558750
14	1.017181	1.269199	0.375742	0.516026
15	0.897913	0.045868	0.419219	0.448720
16	0.094441	0.009118	0.440446	0.375436
17	0.460720	0.855666	0.591017	0.334771
18	0.396725	-0.258465	0.692992	0.307116
19	0.055726	-0.269588	0.723926	0.161935
20	0.102724	1.169186	0.732788	-0.017534
21	0.006007	0.051850	0.762376	-0.041613
22	-0.136201	-0.562211	0.774349	-0.082169
23	0.073115	-0.795469	-0.581939	-0.086729
24	-0.199312	0.656489	-0.603517	-0.115493
25	0.035370	0.821718	-0.635835	-0.157367
26	-0.035519	0.235069	1.174271	-0.175542
27	-0.168793	2.210141	-0.767189	-0.218798
28	-0.178609	1.142457	-0.858134	-0.275882
29	-0.258074	-0.205629	-0.990357	-0.302625
30	-0.276958	1.130554	-1.014419	-0.364127
31	[illegible]	0.283924	-1.362450	-0.365524

	RDS	Administration	Marketing	Profit
32	-0.401129	-0.659324	0.029817	-0.381787
33	-0.600682	1.310535	-0.001879	-0.383444
34	-0.609750	-1.308658	-0.045493	-0.389291
35	-0.991570	0.205925	-0.081763	-0.533932
36	-0.652532	-2.525994	-0.115608	-0.552955
37	-1.177178	-1.997270	-0.212785	-0.771497
38	-0.773820	-1.383122	-0.297583	-0.777094
39	-0.989577	-0.100900	-0.315786	-0.846411
40	-1.008534	-1.320796	-0.384552	-0.857466
41	-1.102106	-0.906938	-0.520596	-1.015365
42	-1.281134	0.217682	-1.449605	-1.058960
43	-1.134305	1.206419	-1.509074	-1.173209
44	-1.593413	-0.199322	0.711122	-1.566922
45	-1.622362	0.507722	-1.743127	-1.740627

In [57]:
```
data2.shape
```
Out[57]:
```
(46, 4)
```

Improving Model

In [58]:
```
final_model= smf.ols('Profit~RDS+Administration+Marketing',data=data2).fit()
```

In [59]:
```
final_model.summary()
```

Out[59]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.965
Model:	OLS	**Adj. R-squared:**	0.962
Method:	Least Squares	**F-statistic:**	384.0
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	1.53e-30
Time:	19:51:40	**Log-Likelihood:**	15.919
No. Observations:	46	**AIC:**	-23.84
Df Residuals:	42	**BIC:**	-16.52
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0162	0.027	0.606	0.548	-0.038	0.070
RDS	0.8944	0.042	21.210	0.000	0.809	0.980
Administration	-0.0447	0.030	-1.504	0.140	-0.105	0.015
[illegible]	.0769	0.044	1.763	0.085	-0.011	0.165

Loading [MathJax]/extensions/Safe.js

Omnibus:	0.048	**Durbin-Watson:**	1.731
Prob(Omnibus):	0.976	**Jarque-Bera (JB):**	0.243
Skew:	-0.002	**Prob(JB):**	0.886
Kurtosis:	2.644	**Cond. No.**	2.76

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

MSE

In [60]:
```
final_model.mse_resid
```

Out[60]:
```
0.032095711144339426
```

RMSE

In [61]:
```
np.sqrt(final_model.mse_resid)
```

Out[61]:
```
0.17915275924288585
```

In [62]:
```
data3 = sc.inverse_transform(data2)
```

Transforming Standard Scaler Data Into Raw Data

In [63]:
```
data4=pd.DataFrame(data3,columns=["RDS","Administration","Marketing","Profit"])
data4
```

Out[63]:

	RDS	Administration	Marketing	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

	RDS	Administration	Marketing	Profit
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	76253.86	113867.30	298664.47	118474.03
20	78389.47	153773.43	299737.29	111313.02
21	73994.56	122782.75	303319.26	110352.25
22	67532.53	105751.03	304768.73	108733.99
23	77044.01	99281.34	140574.81	108552.04
24	64664.71	139553.16	137962.62	107404.34
25	75328.87	144135.98	134050.07	105733.54
26	72107.60	127864.55	353183.81	105008.31
27	66051.52	182645.56	118148.20	103282.38
28	65605.48	153032.06	107138.38	101004.64
29	61994.48	115641.28	91131.24	99937.59
30	61136.38	152701.92	88218.23	97483.56
31	63408.86	129219.61	46085.25	97427.84
32	55493.95	103057.49	214634.81	96778.92
33	46426.07	157693.92	210797.67	96712.80
34	46014.02	85047.44	205517.64	96479.51
35	28663.76	127056.21	201126.82	90708.19
36	44069.95	51283.14	197029.42	89949.14
37	20229.59	65947.93	185265.10	81229.06
38	38558.51	82982.09	174999.30	81005.76
39	28754.33	118546.05	172795.67	78239.91
40	27892.92	84710.77	164470.71	77798.83
41	23640.93	96189.63	148001.11	71498.49
42	15505.73	127382.30	35534.17	69758.98
43	22177.74	154806.14	28334.72	65200.33
44	1315.46	115816.21	297114.46	49490.75
45	0.00	135426.92	0.00	42559.73

In [64]:

```
final_model2 = smf.ols("Profit~RDS+Administration+Marketing",data=data4).fit()
final_model2.summary()
```

Out[64]:

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.965

Loading [MathJax]/extensions/Safe.js

Model:	OLS	**Adj. R-squared:**	0.962
Method:	Least Squares	**F-statistic:**	384.0
Date:	Sun, 10 Apr 2022	**Prob (F-statistic):**	1.53e-30
Time:	19:51:53	**Log-Likelihood:**	-471.41
No. Observations:	46	**AIC:**	950.8
Df Residuals:	42	**BIC:**	958.1
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5.721e+04	5824.368	9.822	0.000	4.55e+04	6.9e+04
RDS	0.7854	0.037	21.210	0.000	0.711	0.860
Administration	-0.0642	0.043	-1.504	0.140	-0.150	0.022
Marketing	0.0253	0.014	1.763	0.085	-0.004	0.054

Omnibus:	0.048	**Durbin-Watson:**	1.731
Prob(Omnibus):	0.976	**Jarque-Bera (JB):**	0.243
Skew:	-0.002	**Prob(JB):**	0.886
Kurtosis:	2.644	**Cond. No.**	1.60e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.6e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

MSE

In [65]:
```
final_model2.mse_resid
```

Out[65]: 51099466.48746972

RMSE

In [66]:
```
np.sqrt(final_model2.mse_resid)
```

Out[66]: 7148.38908338583

Observation :

As We can see that the R-square of the final model has been increased by 96%

Predicting New Values:Manually

In [67]:
```
values = pd.DataFrame({"RDS":86000,"Administration":1234567.87,"Marketing":345678.56},inde
values
```

Out[67]:

	RDS	Administration	Marketing
1	86000	1234567.87	345678.56

In [68]:
```
pd.DataFrame(final_model.predict(values),columns=["Profit"])
```

Out[68]:

	Profit
1	48371.332351

Automatic Predictions

In [69]:
```
pred_y = final_model2.predict(data4)
```

In [70]:
```
prediction_values=pd.DataFrame(pred_y,columns=["Profit"])
prediction_values
```

Out[70]:

	Profit
0	190233.337001
1	186435.222637
2	181559.701604
3	172683.994403
4	172225.976821
5	163566.198942
6	156712.159018
7	158401.007275
8	150223.390224
9	154820.766302
10	135951.487205
11	136706.556594
12	129079.256744
13	127155.952232
14	147853.055789
15	145909.909162
16	117365.840984
17	129391.336010
18	129405.707912
19	117350.723149
20	116491.364017
[illegible]	[illegible]7

Loading [MathJax]/extensions/Safe.js

	Profit
22	111177.321664
23	114901.028567
24	102525.013706
25	110506.847419
26	114577.043666
27	100343.390854
28	101616.554582
29	100777.001811
30	97648.213213
31	99873.617383
32	99610.712750
33	88881.503340
34	93091.317053
35	76654.520276
36	93518.547686
37	73554.385096
38	86594.985022
39	76554.242806
40	77840.473922
41	73346.079715
42	62101.938257
43	65397.637643
44	58330.939210
45	48506.433703

Table of R Square

In [71]:

```
R_square={'Prepared_models':['Model','Final_Model'],'R_Squared':[model1.rsquared,final_moc
table=pd.DataFrame(R_square)
table
```

Out[71]:

	Prepared_models	R_Squared
0	Model	0.950746
1	Final_Model	0.964824

In [ ]:

Loading [MathJax]/extensions/Safe.js