In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Performing EDA

In [ ]:
```
deli=pd.read_csv("C:\\Users\\Hi\\Desktop\\ExceLR Assignments\\delivery_time.csv")
deli.head()
```

Out[ ]:

	Delivery_Time	Sorting_Time
0	21.00	10
1	13.50	4
2	19.75	6
3	24.00	9
4	29.00	10

In [ ]:
```
deli.info() #here we dont have null values or NA values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Delivery_Time  21 non-null     float64
 1   Sorting_Time   21 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 464.0 bytes
```

In [ ]:
```
deli.describe()
```

Out[ ]:

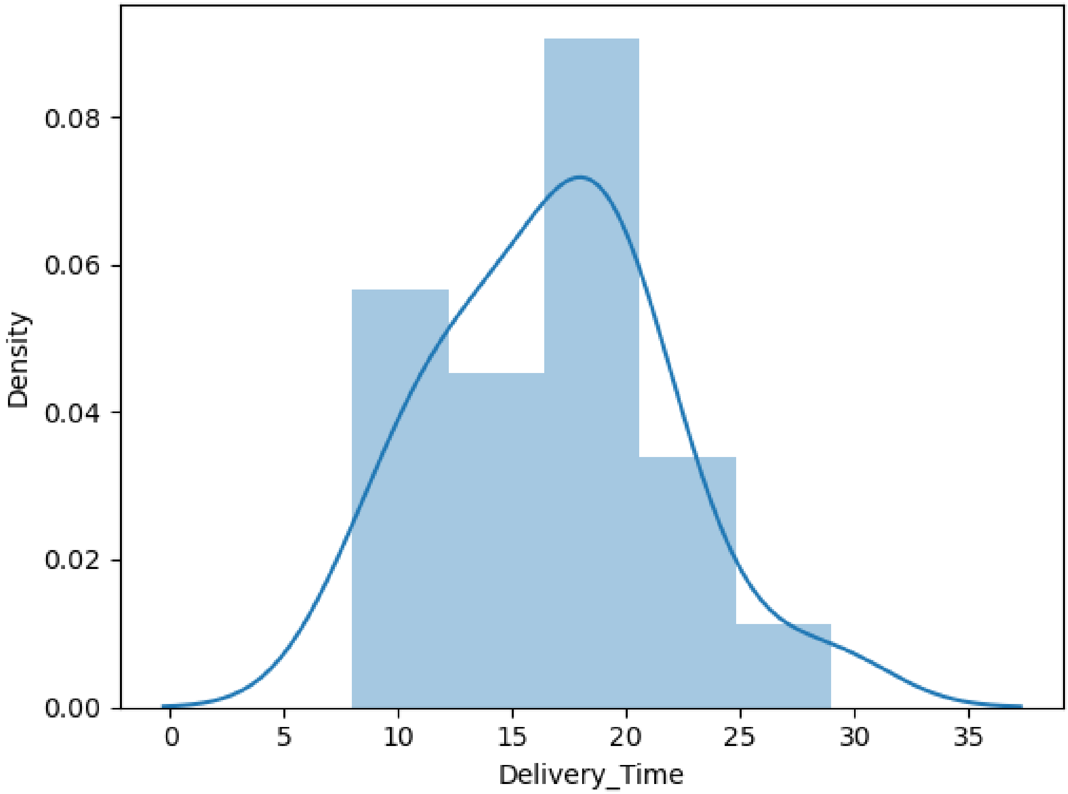
	Delivery_Time	Sorting_Time
count	21.000000	21.000000
mean	16.790952	6.190476
std	5.074901	2.542028
min	8.000000	2.000000
25%	13.500000	4.000000
50%	17.830000	6.000000
75%	19.750000	8.000000
max	29.000000	10.000000

In [ ]:
```
sns.distplot(deli['Delivery_Time'])#here we can see that Delivery_time data Column is no
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_14436\2751119043.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(deli['Delivery_Time'])#here we can see that Delivery_time data Column is
normally distributed without any skewness
```

Out[ ]:

```
<AxesSubplot:xlabel='Delivery_Time', ylabel='Density'>
```

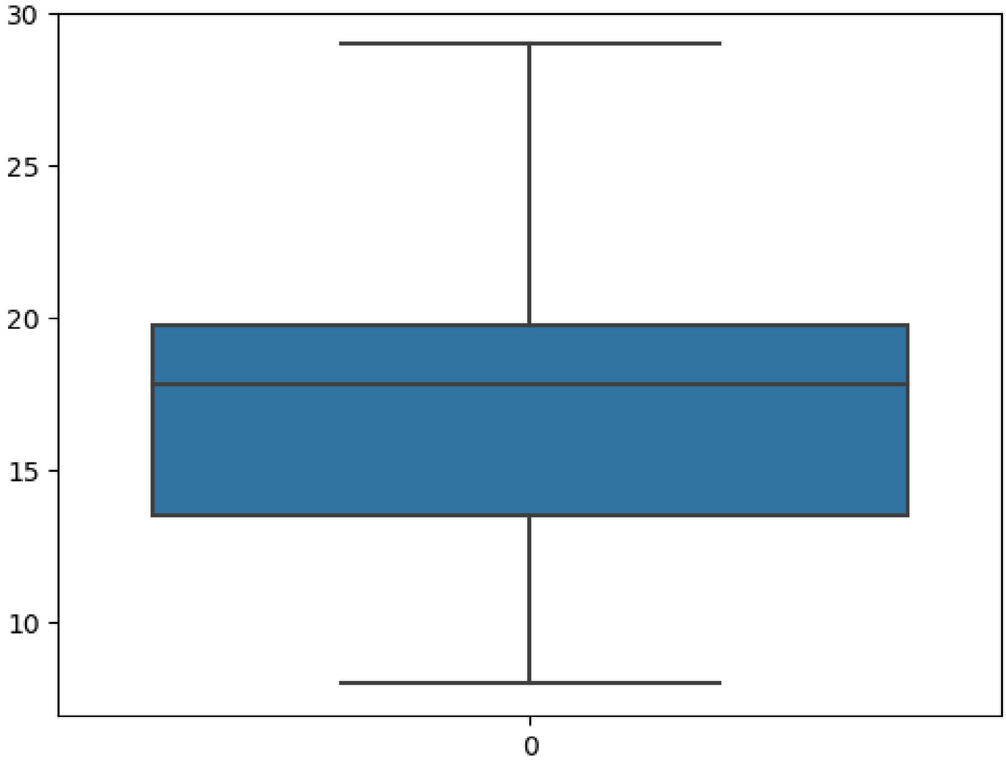


In [ ]:

```
sns.boxplot(deli['Delivery_Time']) #here we can see that in the box plot we dont have ou
```

Out[ ]:

```
<AxesSubplot:>
```



A skewness value greater than 1 or less than -1 indicates a highly skewed distribution. A value between 0.5 and 1 or -0.5 and -1 is moderately skewed. A value between -0.5 and 0.5 indicates that the distribution is fairly symmetrical.

In [ ]:
```
print('The skewness of the Salary Data Column is between -0.5 and 0.5 indicates that the
```

```
The skewness of the Salary Data Column is between -0.5 and 0.5 indicates that the distri
bution is fairly symmetrical.  0.3523900822831107
```

The pandas library function kurtosis() computes the Fisher's Kurtosis which is obtained by subtracting the Pearson's Kurtosis by three. With Fisher's Kurtosis, definition a normal distribution has a kurtosis of 0

Kurtosis number should be between 1 and - 1. If it is in this range that mean the data is normally distributed.

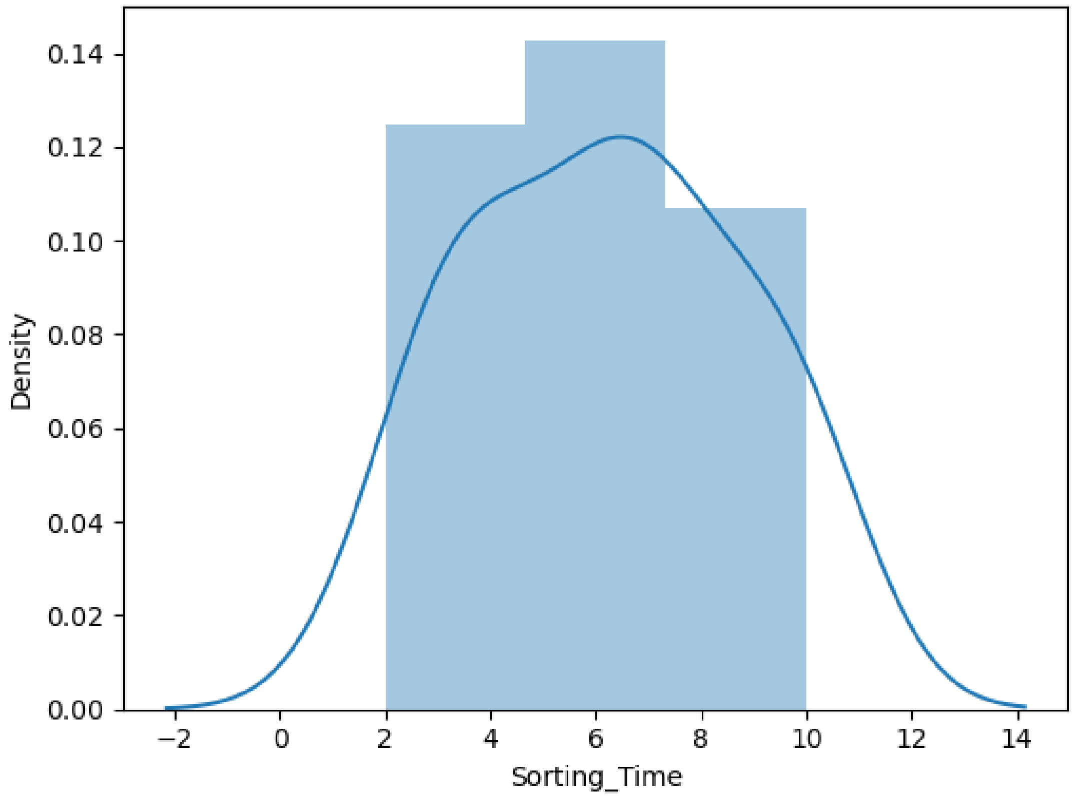
In [ ]:
```
print('The Kurtosis of the Delivery Time Data Column is :', deli.Delivery_Time.kurtosis(
```

```
The Kurtosis of the Delivery Time Data Column is : 0.31795982942685397
```

In [ ]:
```python
sns.distplot(deli['Sorting_Time']) #here we can see that Sorting_Time data column is nor
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_14436\2727263696.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(deli['Sorting_Time']) #here we can see that Sorting_Time data column is n
ormally distributed and without any skewness
```

Out[ ]:
```
<AxesSubplot:xlabel='Sorting_Time', ylabel='Density'>
```



In [ ]:
```python
print('The skewness of the Sorting Time Data Column is between -0.5 and 0.5 indicates th
```

```
The skewness of the Sorting Time Data Column is between -0.5 and 0.5 indicates that the
distribution is fairly symmetrical.  0.047115474210530174
```

In [ ]:
```python
print('The Kurtosis of the Sorting Time Data Column is Highly Peaked ', deli.Sorting_Tim
```

```
The Kurtosis of the Sorting Time Data Column is Highly Peaked  -1.14845514534878
```

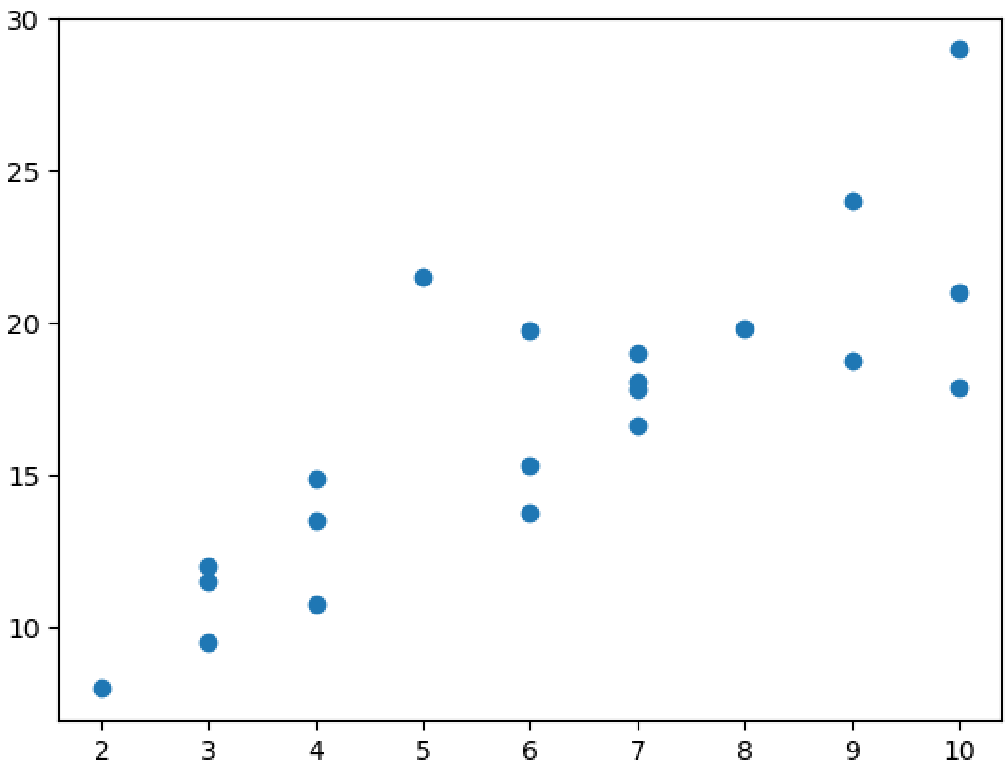
In [ ]:
```python
deli.corr() #there is positive correlation (pearson's correlation) between Target variab
```

Out[ ]:

	Delivery_Time	Sorting_Time
Delivery_Time	1.000000	0.825997
Sorting_Time	0.825997	1.000000

In [ ]:
```python
plt.scatter(deli.Sorting_Time,deli.Delivery_Time) #through this scatterplot we can see t
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x288a163ca60>
```



Feature Scaling

In [ ]:
```
#Normalization of the data
#from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
```

In [ ]:
```
scaler = MinMaxScaler(feature_range=(0,1))
names=deli.columns
d=scaler.fit_transform(deli)
df=pd.DataFrame(d,columns=names)
df.head() #df is normalized data frame of deli
```

Out[ ]:

	Delivery_Time	Sorting_Time
0	0.619048	1.000
1	0.261905	0.250
2	0.559524	0.500
3	0.761905	0.875
4	1.000000	1.000

Transforming Dataset by applying square root on data columns

import torch

df['Delivery_Time']=torch.Tensor(df['Delivery_Ti

df['Sorting_Time']=torch.Tensor(df['Sorting_Tim

df=torch.tensor(df.values, names=('Delivery_Time','Sorting_Time')) df

df=torch.log(df) df

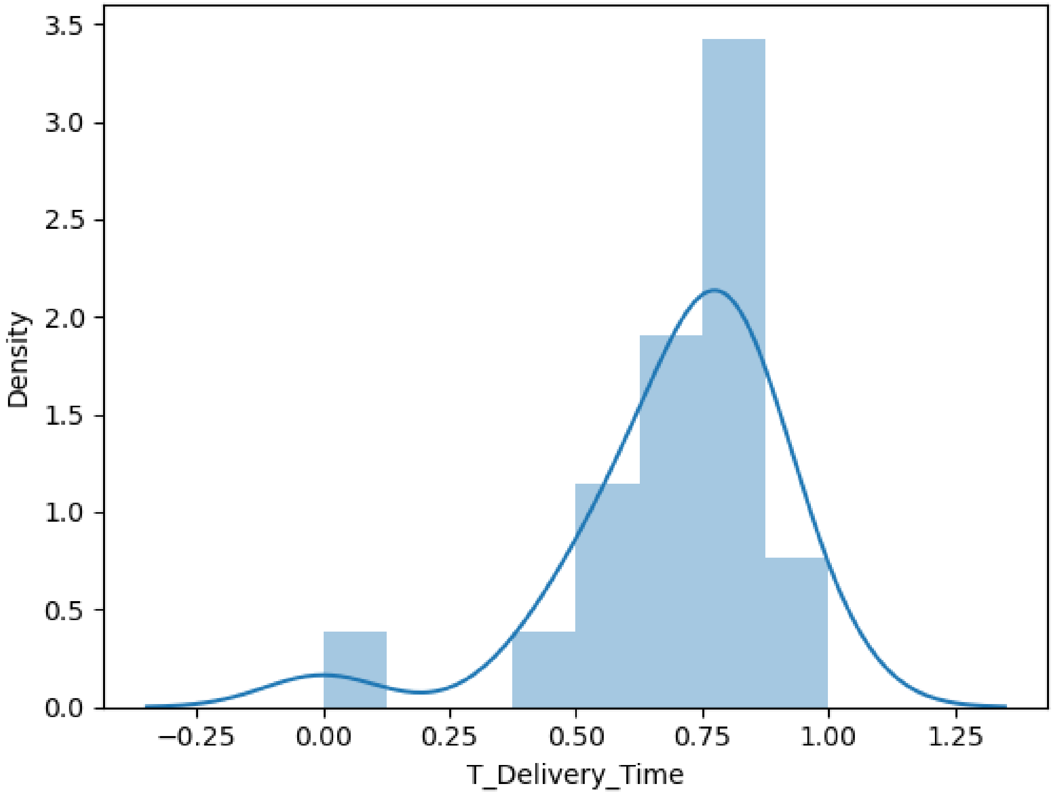
In [ ]:
```
df['T_Delivery_Time']=np.cbrt(df['Delivery_Time']) #Transforming Dataset by Cube rooting
print('Skewness of Delivery_Time column without Transforming---->',deli.Delivery_Time.sk
print('Skewness of Delivery_Time column with Feature Scaling And Cube root Transformatio
print('Kurtosis of Delivery_Time column without Transforming---->',deli.Delivery_Time.ku
print('Kurtosis of Delivery_Time column with Feature Scaling And Cube root Transformatio
df['T_Sorting_Time']=np.cbrt(df['Sorting_Time']) #Transforming Dataset by Cube rooting t
print('Skewness of Sorting_Time column without Transforming---->',deli.Sorting_Time.skew
print('Skewness of Sorting_Time column with Feature Scaling And Cube root Transformation
print('Kurtosis of Sorting_Time column without Transforming---->',deli.Sorting_Time.kurt
print('Kurtosis of Sorting_Time column with Feature Scaling And Cube root Transformation
```

```
Skewness of Delivery_Time column without Transforming----> 0.3523900822831107
Skewness of Delivery_Time column with Feature Scaling And Cube root Transformation---->
-1.8576064364779223
Kurtosis of Delivery_Time column without Transforming----> 0.31795982942685397
Kurtosis of Delivery_Time column with Feature Scaling And Cube root Transformation---->
4.994929921831286
Skewness of Sorting_Time column without Transforming----> 0.047115474210530174
Skewness of Sorting_Time column with Feature Scaling And Cube root Transformation----> -
1.6122394083910478
Kurtosis of Sorting_Time column without Transforming----> -1.14845514534878
Kurtosis of Sorting_Time column with Feature Scaling And Cube root Transformation---->
3.620161110474966
```

In [ ]:
```
sns.distplot(df.T_Delivery_Time)
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_14436\3543349098.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.T_Delivery_Time)
```

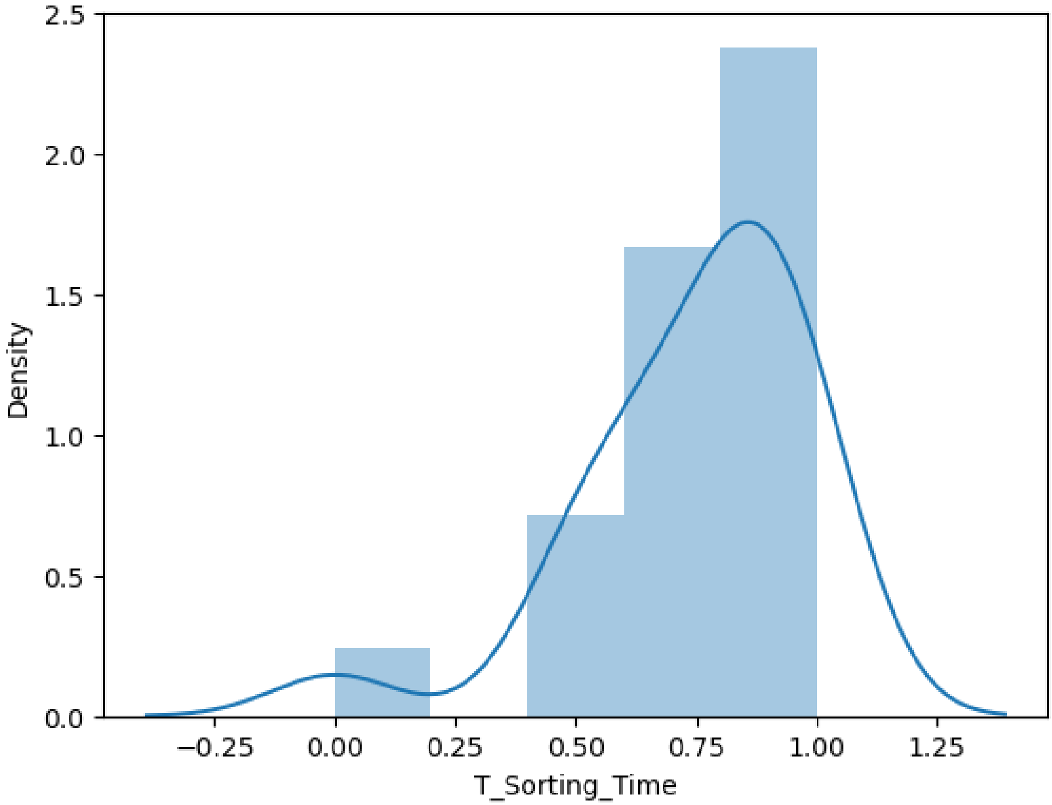
Out[ ]:
```
<AxesSubplot:xlabel='T_Delivery_Time', ylabel='Density'>
```



In [ ]:
```
sns.distplot(df.T_Sorting_Time)
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_14436\213214463.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.T_Sorting_Time)
```

Out[ ]:
```
<AxesSubplot:xlabel='T_Sorting_Time', ylabel='Density'>
```



In [ ]:
```
#Automatic EDA using sweetviz and creating html file
import sweetviz as sv
sweet_report = sv.analyze(deli)
sweet_report.show_html('delivery_time_report.html')
```

```
                                             |          | [  0%]   00:00 -> (? left)
Report delivery_time_report.html was generated! NOTEBOOK/COLAB USERS: the web browser MA
Y not pop up, regardless, the report IS saved in your notebook/colab files.
```

In [ ]:
```
df.head()
```

Out[ ]:

	Delivery_Time	Sorting_Time	T_Delivery_Time	T_Sorting_Time
0	0.619048	1.000	0.852265	1.000000
1	0.261905	0.250	0.639805	0.629961
2	0.559524	0.500	0.824023	0.793701
3	0.761905	0.875	0.913342	0.956466
4	1.000000	1.000	1.000000	1.000000

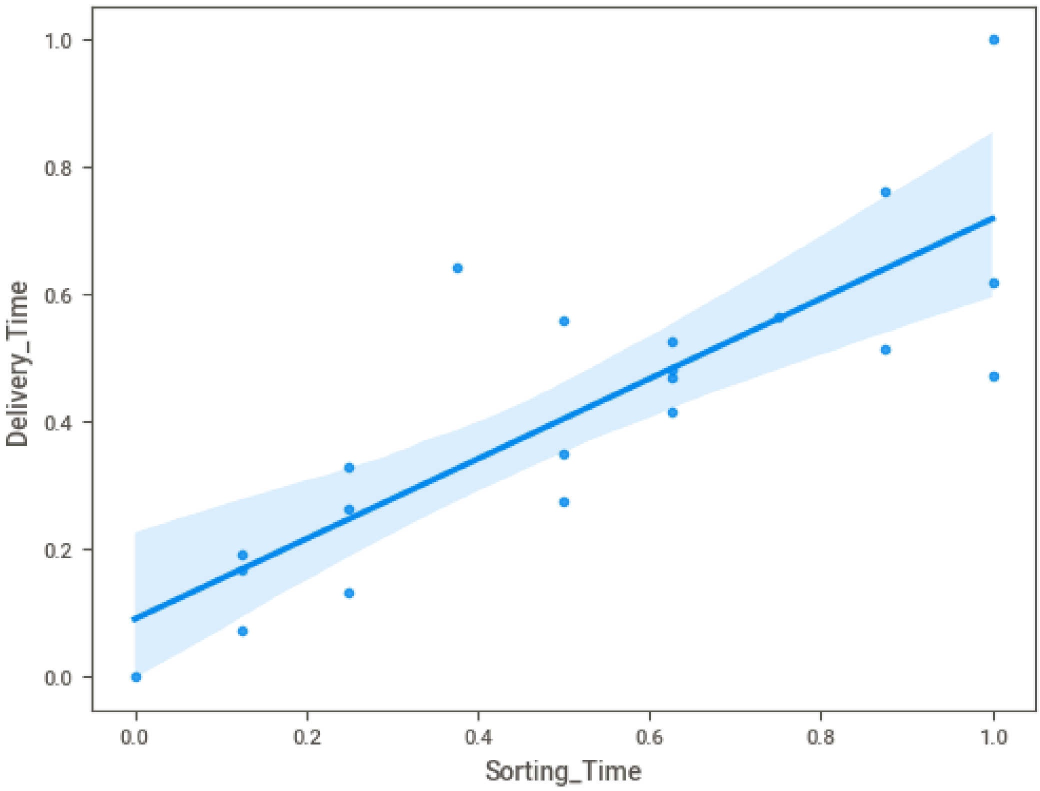
Model1 Creation

In [ ]:
```
#creating model
import statsmodels.formula.api as smf
model =smf.ols("Delivery_Time~Sorting_Time", data=df).fit()
```

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

Out[ ]:
```
Intercept       0.089561
Sorting_Time    0.628198
dtype: float64
```

In [ ]:
```
sns.regplot(x='Sorting_Time',y='Delivery_Time', data=df);
#after the regplot we can see there are influencing plot
```



In [ ]:
```
print(model.tvalues,'\n', model.pvalues)#here pvalue of sorting time is <0.05 hence reje
#Sorting Time is actually dependent variable
```

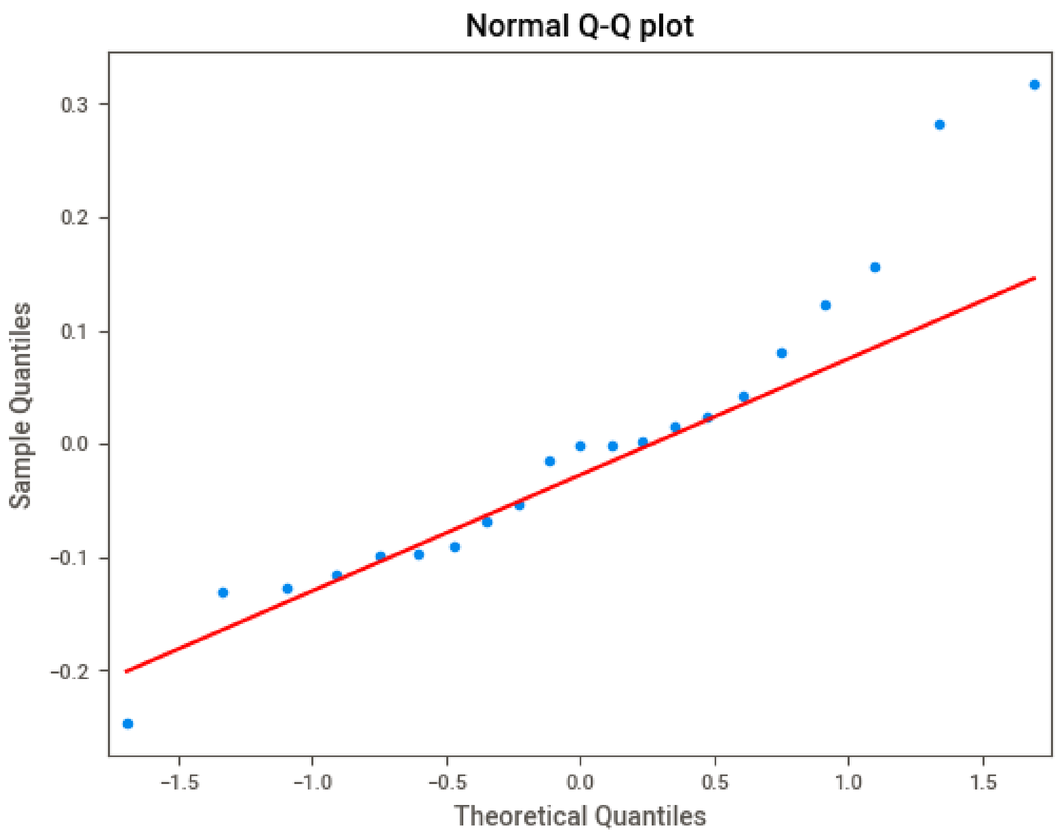
```
Intercept       1.496005
Sorting_Time    6.387447
dtype: float64
 Intercept       0.151079
Sorting_Time    0.000004
dtype: float64
```

In [ ]:
```
(model.rsquared,model.rsquared_adj)
```

Out[ ]:
```
(0.6822714748417231, 0.6655489208860244)
```

Test for Normality of Residuals (Q-Q Plot)

In [ ]:
```
#qqplot
import statsmodels.api as sm
qqplot=sm.qqplot(model.resid,line='q')
plt.title("Normal Q-Q plot")
plt.show()
```



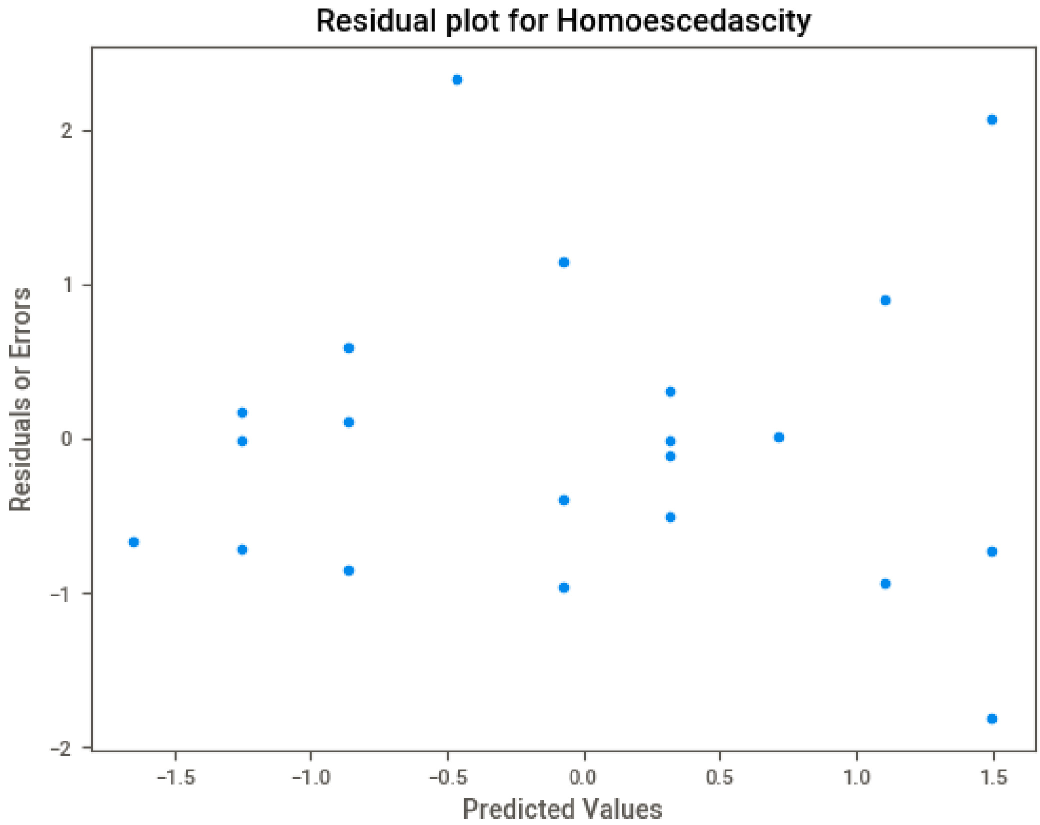
Residual Plot For Homoscedasticity

In [ ]:

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

In [ ]:

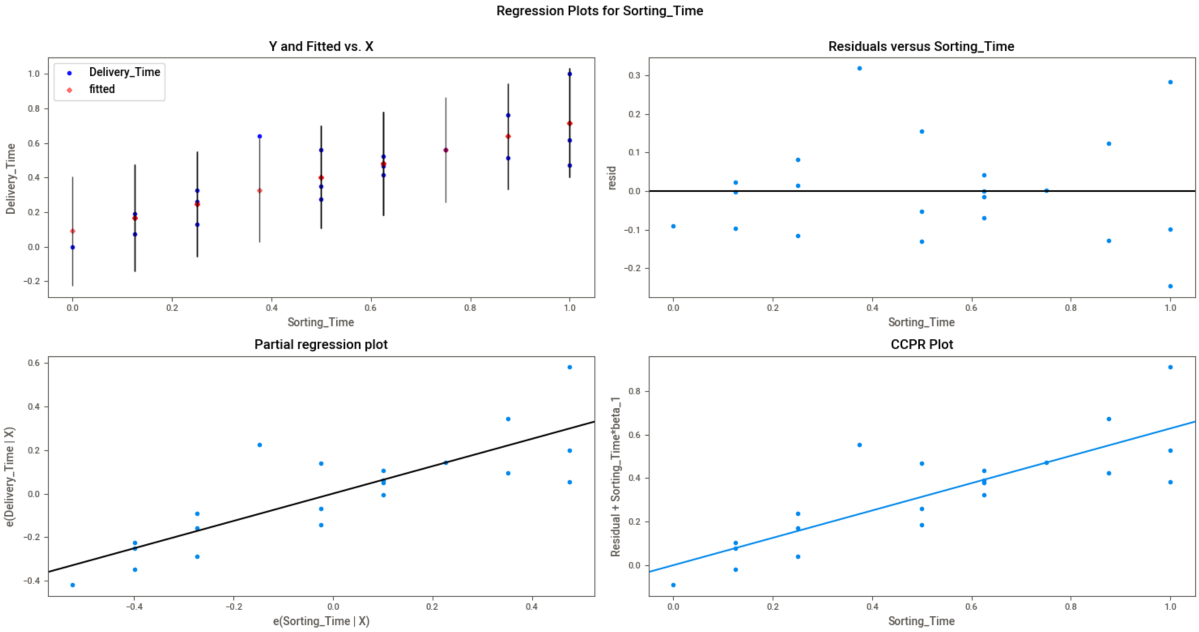
```
plt.scatter(get_standardized_values(model.fittedvalues),get_standardized_values(model.re
plt.title("Residual plot for Homoescedascity")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals or Errors")
plt.show()
```



Residual vs Regressors

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

```
eval_env: 1
```



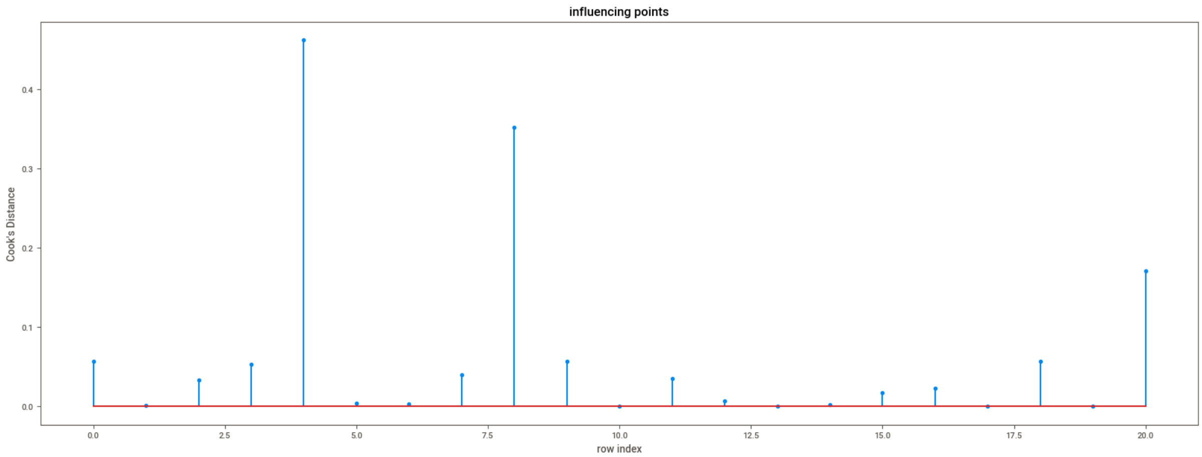
In [ ]:
```
model_influence = model.get_influence()
(c,_)= model_influence.cooks_distance
summary_cooks=model_influence.summary_frame()
summary_cooks.head()
```

Out[ ]:

	dfb_Intercept	dfb_Sorting_Time	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid	
0	0.147322	-0.278610	0.056517	-0.770600	0.159912	-0.336207	-0.762050	-0.
1	0.032245	-0.022432	0.000606	0.114391	0.084746	0.034808	0.111379	0.
2	0.148371	-0.019795	0.032861	1.142958	0.047900	0.256363	1.152810	0.
3	-0.099580	0.242465	0.052706	0.929723	0.108696	0.324673	0.926240	0.
4	-0.480507	0.908714	0.462053	2.203350	0.159912	0.961304	2.485504	1.

In [ ]:

```
#plot the influencers using stem plot
fig=plt.subplots(figsize=(20,7))
plt.stem(np.arange(len(df)),np.round(c,3))
plt.title("influencing points")
plt.xlabel("row index")
plt.ylabel("Cook's Distance")
plt.show()
```



we can see that there are 2 high influencing points ,Cook's Distance threshold is given by 4/N or 4/(N-k-1) where N is no.of observation and k no. of explanatory varibles.

ref. - 1 Fox,John. (1991). Regression Diagnostics: An Intro.Sage Publications

so 4/N of Sorting Time its 4/20 = 0.2 is our threshold above 0.2 to be considered high influencing point

In [ ]:

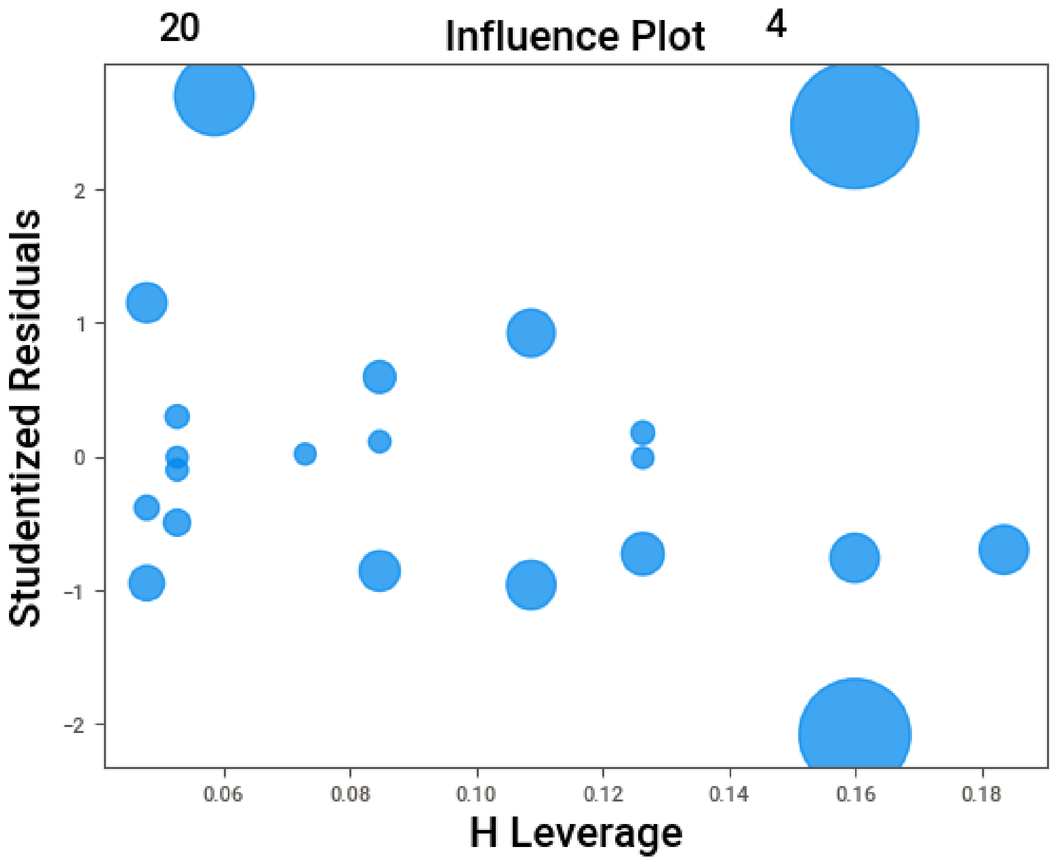
```
(np.argmax(c),np.max(c)) # here 4 is an influencing point
```

Out[ ]:

```
(4, 0.46205304126503316)
```

In [ ]:

```
#High Influence points
from statsmodels.graphics.regressionplots import influence_plot
influence_plot(model)
plt.show()
```



from the above bubble plot we can see that 4,20 high influencing points influencing the model we should replace ,retain or remove only when the domain experts suggest to retain,replace or remove the high influencing points

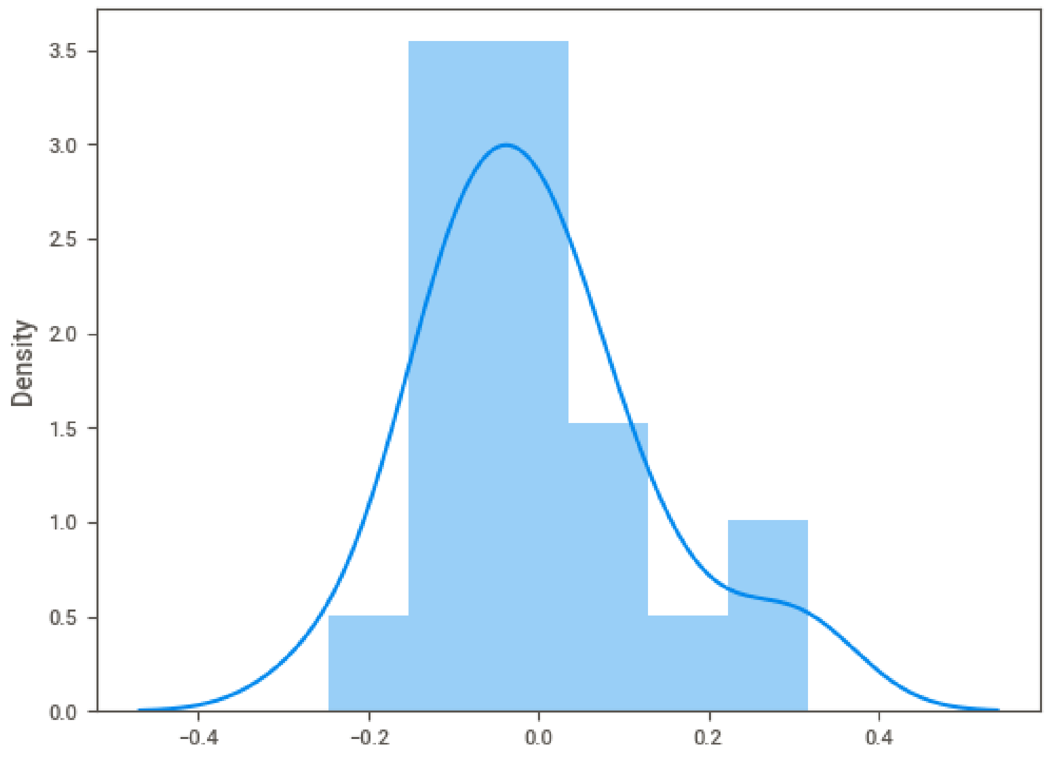
In [ ]:
```
model.resid.mean()
```

Out[ ]:
```
1.8503717077085943e-16
```

In [ ]:
```
sns.distplot(model.resid)
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_14436\2570432973.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(model.resid)
```

Out[ ]:
```
<AxesSubplot:ylabel='Density'>
```



In [ ]:
```
import math
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
mse_m1= mean_squared_error(df.Delivery_Time,model.fittedvalues) #checking for RMSE value
rmse_m1=math.sqrt(mse_m1)
print("the difference between actual and predicted values of model1 is :---",rmse_m1)
```

```
the difference between actual and predicted values of model1 is :--- 0.13293572986008406
```

In [ ]:
```
df.Sorting_Time.median()
```

Out[ ]:
```
0.5
```

In [ ]:
```
##improving model
deli_1=pd.read_csv("C:\\Users\\Hi\\Desktop\\ExceLR Assignments\\delivery_time.csv")
```

In [ ]:
```
from sklearn import preprocessing
import pandas as pd
scaler= preprocessing.MinMaxScaler()
names= deli_1.columns
d= scaler.fit_transform(deli_1)
scaled_df=pd.DataFrame(d,columns=names)
scaled_df.head()
```

Out[ ]:

	Delivery_Time	Sorting_Time
0	0.619048	1.000
1	0.261905	0.250
2	0.559524	0.500
3	0.761905	0.875
4	1.000000	1.000

In [ ]:
```
#scaled_df.loc[4,'Sorting_Time']
scaled_df.Sorting_Time.median(),scaled_df.Sorting_Time.mean()
```

Out[ ]:
```
(0.5, 0.5238095238095238)
```

Creating Model2 by Replacing 4 & 20 influencing points in Sorting column by sorting time median which is 0.5 in normalized data

In [ ]:
```
scaled_df.loc[4,'Sorting_Time']=0.5
scaled_df.loc[20,'Sorting_Time']=0.5
```

In [ ]:
```
scaled_df.head()
```

Out[ ]:

	Delivery_Time	Sorting_Time
0	0.619048	1.000
1	0.261905	0.250
2	0.559524	0.500
3	0.761905	0.875
4	1.000000	0.500

In [ ]:
```
import statsmodels.formula.api as smf
model2=smf.ols("Delivery_Time~Sorting_Time", data=scaled_df).fit()
```

In [ ]:
```
model.params,    model2.params  #comparing model1 and model2 parameters
```

Out[ ]:
```
(Intercept       0.089561
 Sorting_Time    0.628198
 dtype: float64,
 Intercept       0.130251
 Sorting_Time    0.569946
 dtype: float64)
```

In [ ]:
```
model2.rsquared,model2.rsquared_adj
#R2 coefficient of determination determines accuracy of te model , here the model's accu
```

Out[ ]:
```
(0.4908362176421791, 0.4640381238338728)
```

In [ ]:
```
mse_m2= mean_squared_error(scaled_df.Delivery_Time,model2.fittedvalues)
rmse_m2=math.sqrt(mse_m2)
print("the difference between actual and predicted values of model2 is :---",rmse_m2)
```

```
the difference between actual and predicted values of model2 is :--- 0.16828382429330135
```

Creating Model3 by Dropping 4&20 rows to improve accuracy of the model

In [ ]:
```
deli_2=scaled_df.drop([4,20],axis=0)
```

In [ ]:
```
deli_2.head()
```

Out[ ]:

	Delivery_Time	Sorting_Time
0	0.619048	1.000
1	0.261905	0.250
2	0.559524	0.500
3	0.761905	0.875
5	0.350000	0.500

In [ ]:
```
model3=smf.ols("Delivery_Time~Sorting_Time",data=deli_2).fit()
```

In [ ]:
```
model3.params,model2.params,model.params
```

Out[ ]:
```
(Intercept       0.085959
 Sorting_Time    0.572975
 dtype: float64,
 Intercept       0.130251
 Sorting_Time    0.569946
 dtype: float64,
 Intercept       0.089561
 Sorting_Time    0.628198
 dtype: float64)
```

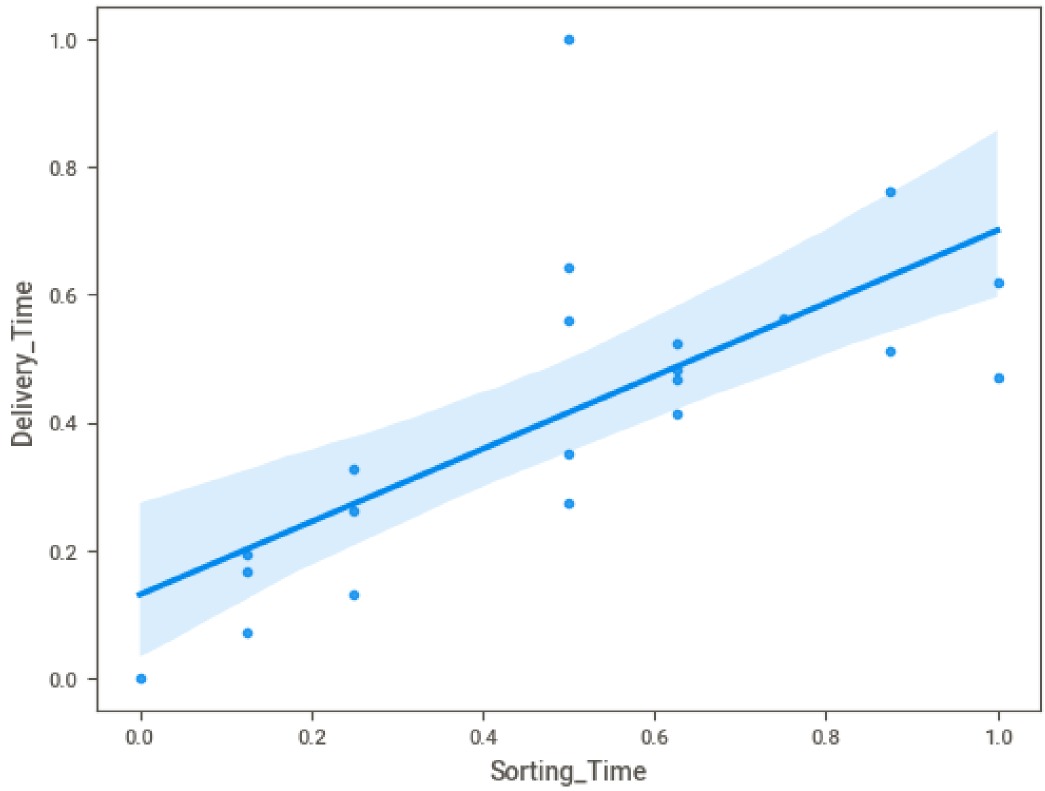
In [ ]:
```
model3.tvalues,'/n' , model3.pvalues
```

Out[ ]:
```
(Intercept       1.953668
 Sorting_Time    7.698826
 dtype: float64,
 '/n',
 Intercept       6.739820e-02
 Sorting_Time    6.129953e-07
 dtype: float64)
```

In [ ]:
```
model3.rsquared,model3.rsquared_adj
```

Out[ ]:
```
(0.7771132785587765, 0.7640022949445869)
```

In [ ]:
```
sns.regplot(x='Sorting_Time', y='Delivery_Time',data=scaled_df);
```

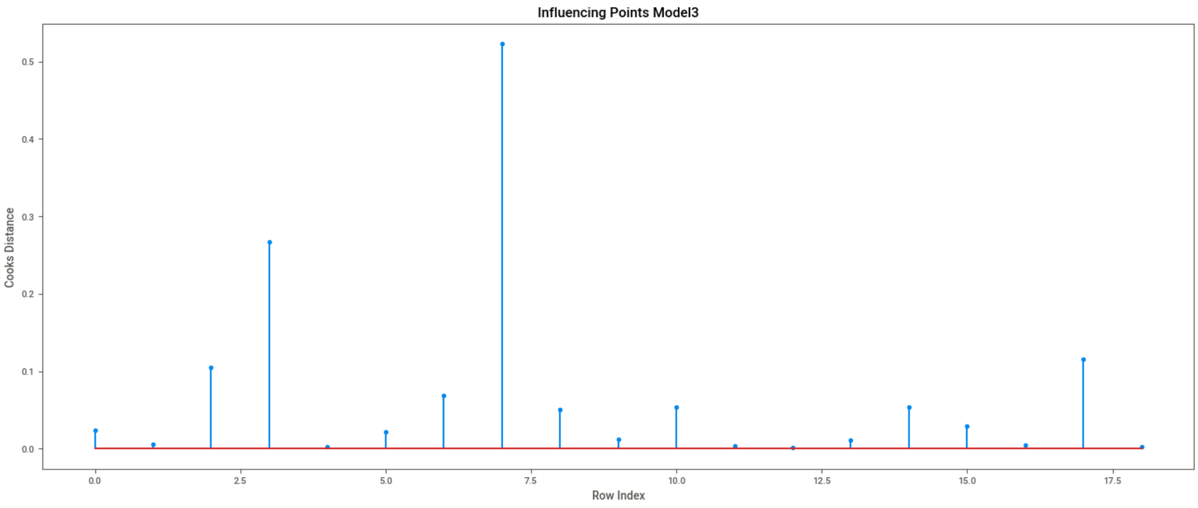


In [ ]:
```
model3_influence=model3.get_influence()
(c3,_)=model3_influence.cooks_distance
summary3_cooks=model3_influence.summary_frame()
summary3_cooks.head()
```

Out[ ]:

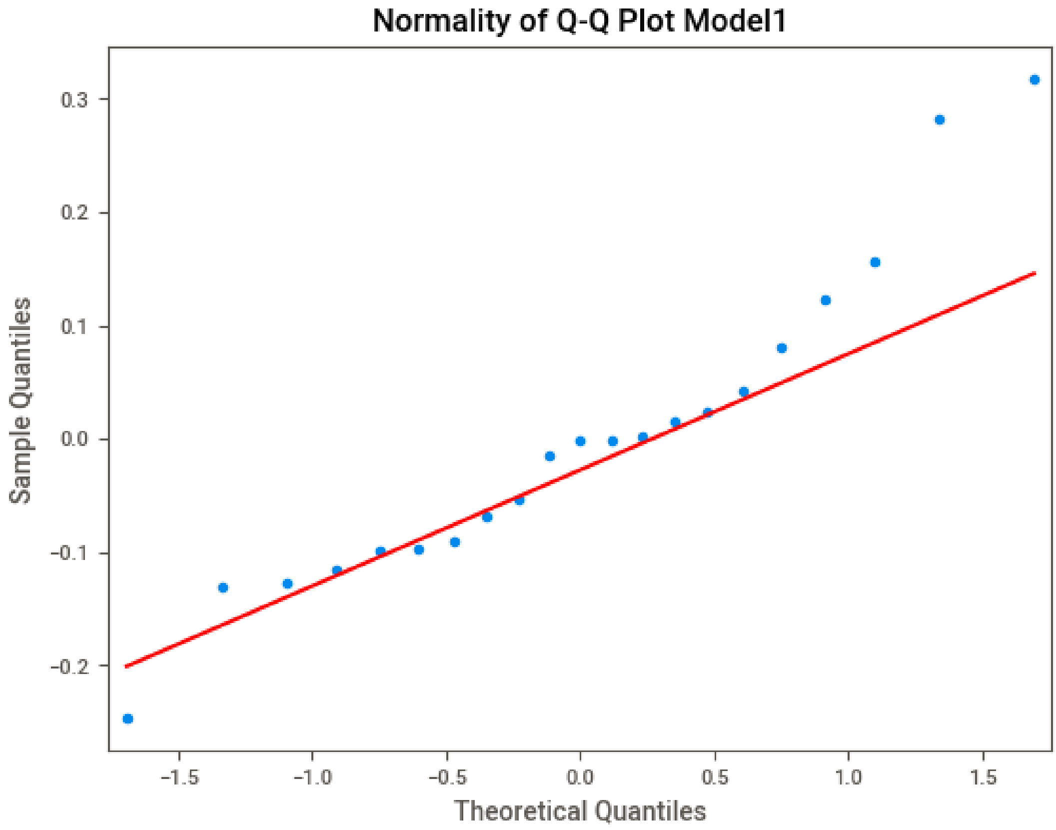
	dfb_Intercept	dfb_Sorting_Time	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid	
0	0.097282	-0.180666	0.023672	-0.448408	0.190587	-0.217588	-0.437615	-0.
1	0.100412	-0.068344	0.005940	0.346725	0.089935	0.108997	0.337569	0.
2	0.269194	-0.010882	0.105034	1.944059	0.052656	0.458332	2.138668	0.
3	-0.264449	0.614387	0.266580	1.892757	0.129543	0.730178	2.066900	0.
5	-0.028529	0.001153	0.001512	-0.233252	0.052656	-0.054992	-0.226651	-0.

In [ ]:
```
fig2=plt.subplots(figsize=(18,7))
plt.stem(np.arange(len(deli_2)),np.round(c3,3))
plt.title('Influencing Points Model3')
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```

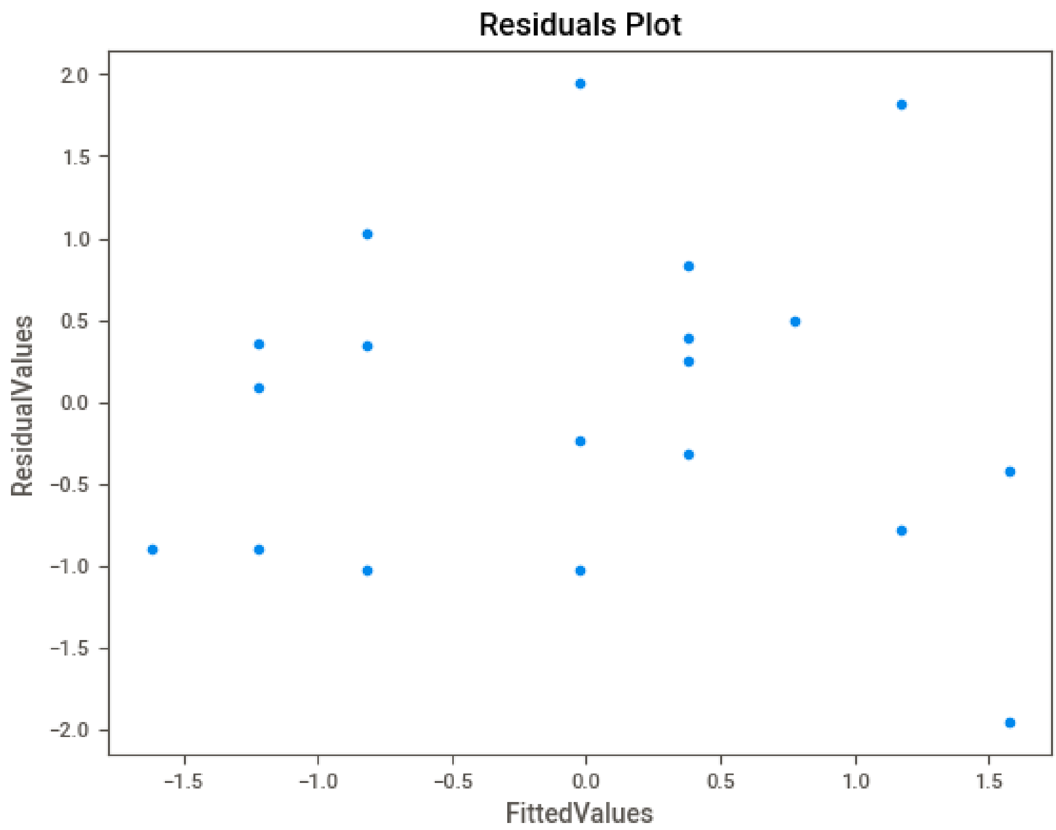



```
In [ ]: qqplot3=sm.qqplot(model3.resid,line='q') #qqplot of model3
        plt.title('Normality of Q-Q Plot Model3')
        plt.show()
```


```
In [ ]: qqplot=sm.qqplot(model.resid,line='q') #qqplot of 1st model
        plt.title('Normality of Q-Q Plot Model1')
        plt.show()
```



In [ ]:
```
plt.scatter(get_standardized_values(model3.fittedvalues),get_standardized_values(model3.
plt.title("Residuals Plot")
plt.xlabel("FittedValues")
plt.ylabel("ResidualValues")
plt.show() #There is no patterns in Residual vs Fitted Values ,hence No problem orelse w
```



In [ ]:
```
model3.rsquared,model3.rsquared_adj
```

Out[ ]:
```
(0.7771132785587765, 0.7640022949445869)
```

In [ ]:
```
mse_m3= mean_squared_error(deli_2.Delivery_Time,model3.fittedvalues)
rmse_m3=math.sqrt(mse_m3)
print("the difference between actual and predicted values of model1 is :---",rmse_m3)
```

```
the difference between actual and predicted values of model1 is :--- 0.09352047119983505
```

Model4 Creation

In [ ]:
```
model4=smf.ols("T_Delivery_Time~T_Sorting_Time",data=df).fit()
```

In [ ]:
```
model4.params,model3.params,model2.params, model.params
```

Out[ ]:
```
(Intercept         0.080135
 T_Sorting_Time    0.827422
 dtype: float64,
 Intercept       0.085959
 Sorting_Time    0.572975
 dtype: float64,
 Intercept       0.130251
 Sorting_Time    0.569946
 dtype: float64,
 Intercept       0.089561
 Sorting_Time    0.628198
 dtype: float64)
```

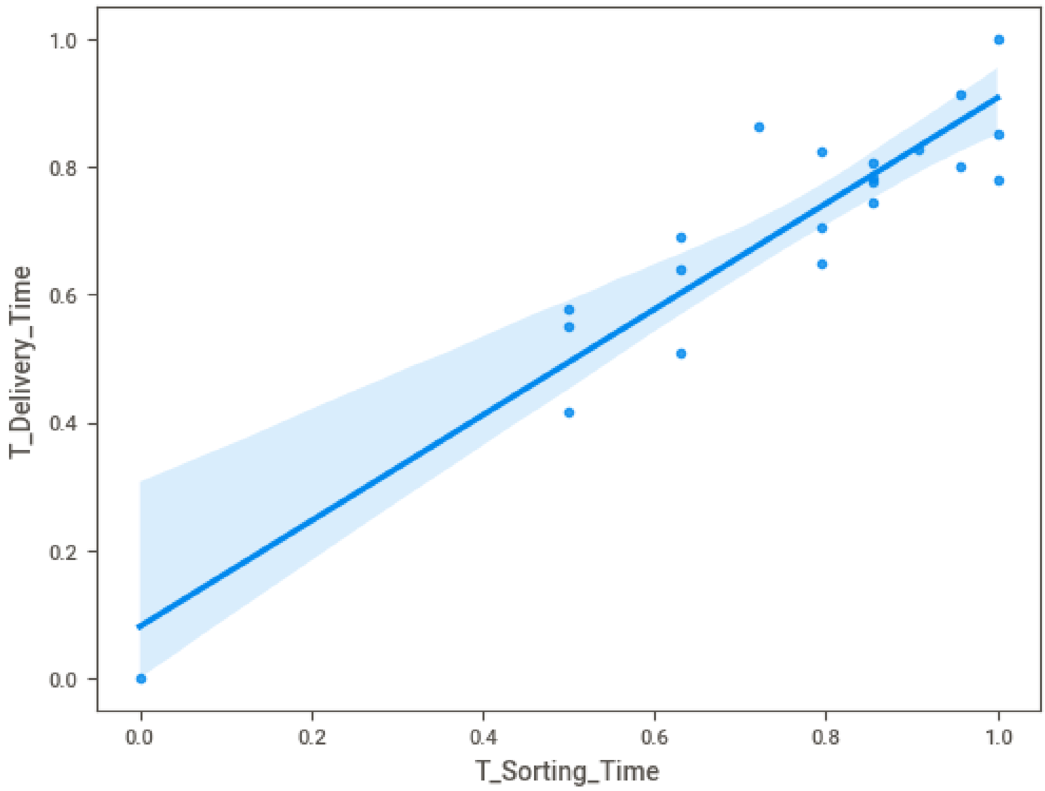
In [ ]:
```
model4.tvalues,'/n' , model4.pvalues
```

Out[ ]:
```
(Intercept          1.323366
 T_Sorting_Time    10.721771
 dtype: float64,
 '/n',
 Intercept         2.014183e-01
 T_Sorting_Time    1.693224e-09
 dtype: float64)
```

In [ ]:
```
model4.rsquared,model4.rsquared_adj
```

Out[ ]:
```
(0.8581627674888582, 0.8506976499882718)
```

In [ ]:
```
sns.regplot(x='T_Sorting_Time', y='T_Delivery_Time',data=df);
```

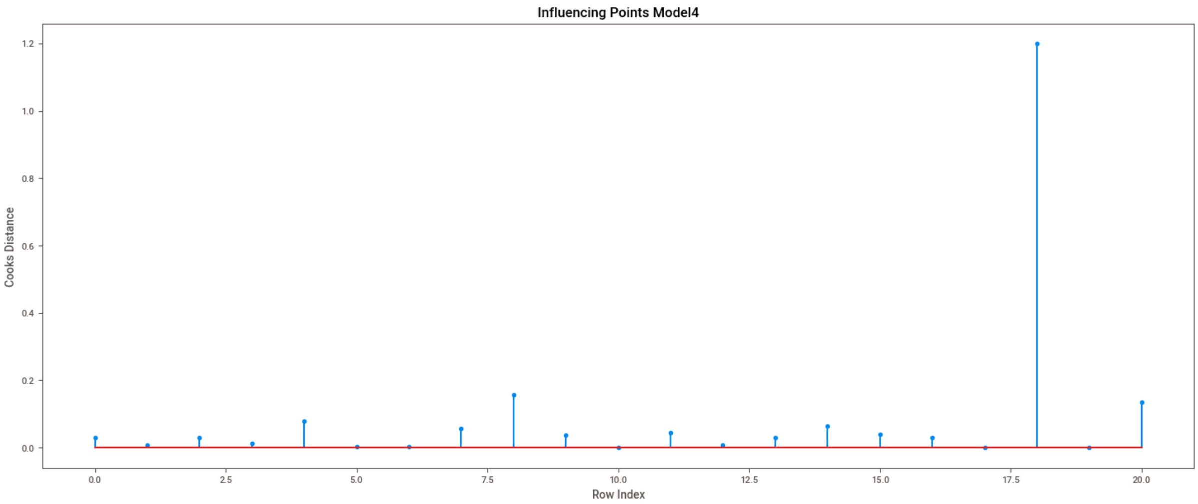


In [ ]:
```
model4_influence=model4.get_influence()
(c4,_)=model4_influence.cooks_distance
summary4_cooks=model4_influence.summary_frame()
summary4_cooks.head()
```

Out[ ]:

	dfb_Intercept	dfb_T_Sorting_Time	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid
0	0.117388	-0.173003	0.028676	-0.707863	0.102702	-0.239481	-0.698252
1	0.083224	-0.054187	0.007383	0.480657	0.060075	0.121517	0.470708
2	0.028041	0.046516	0.030507	1.084103	0.049352	0.247010	1.089420
3	-0.065406	0.105409	0.013091	0.530072	0.085238	0.161807	0.519792
4	-0.201211	0.296539	0.080153	1.183459	0.102702	0.400383	1.196852

In [ ]:
```
fig2=plt.subplots(figsize=(18,7))
plt.stem(np.arange(len(df)),np.round(c4,3))
plt.title('Influencing Points Model4')
plt.xlabel('Row Index')
plt.ylabel('Cooks Distance')
plt.show()
```

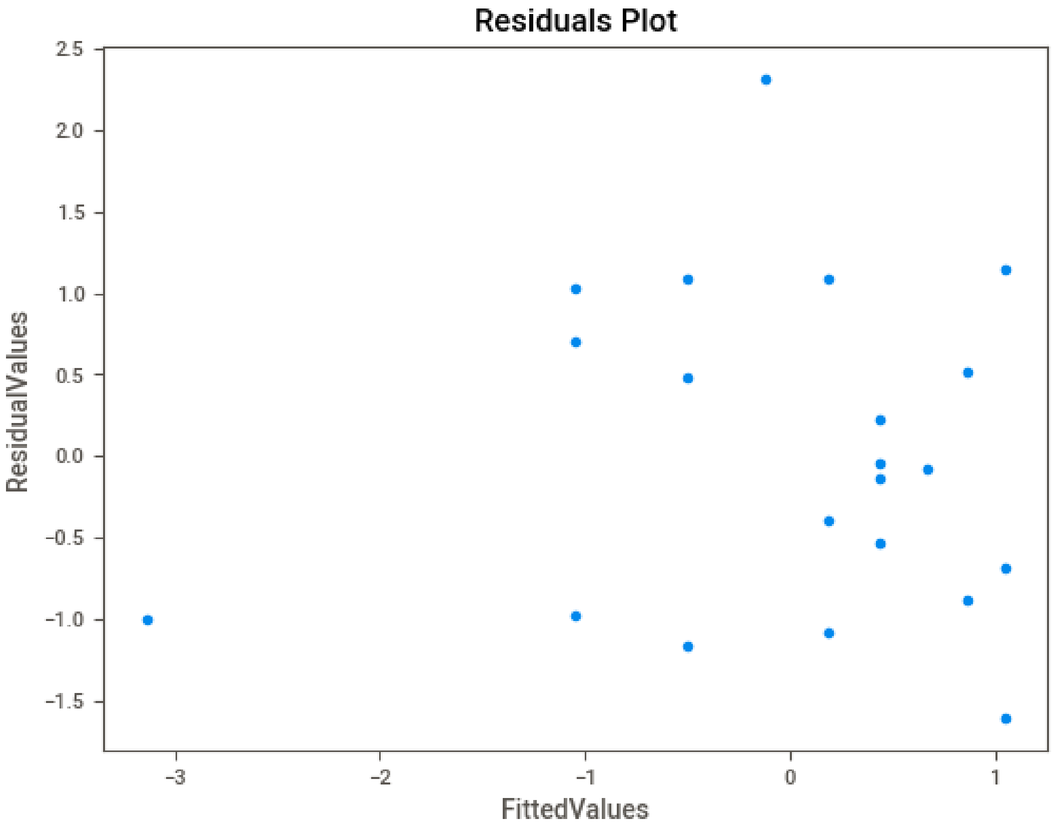


In [ ]:
```
(np.argmax(c4),np.max(c4)) #finding the most influencing datapoint ehich is above 4/N wh
```

Out[ ]:
```
(18, 1.199428879535801)
```

In [ ]:
```
qqplot4=sm.qqplot(model4.resid,line='q') #qqplot of model4
plt.title('Normality of Q-Q Plot Model4')
plt.show()
```

In [ ]:
```
plt.scatter(get_standardized_values(model4.fittedvalues),get_standardized_values(model4.
plt.title("Residuals Plot")
plt.xlabel("FittedValues")
plt.ylabel("ResidualValues")
plt.show()
```



In [ ]:
```
model4.rsquared,model3.rsquared,model2.rsquared, model.rsquared
```

Out[ ]:
```
(0.8581627674888582,
 0.7771132785587765,
 0.4908362176421791,
 0.6822714748417231)
```

In [ ]:
```
model4.rsquared_adj,model3.rsquared_adj,model2.rsquared_adj, model.rsquared_adj
```

Out[ ]:
```
(0.8506976499882718,
 0.7640022949445869,
 0.4640381238338728,
 0.6655489208860244)
```

MSE(Mean Squared Error), RMSE(Root Mean Squared Error), MAE(Mean Absolute Error), are the methods used to define loss function(actual-predicted values) , this measures error in our model , so that it give us to what extent the error rate is

here we are using RMSE which is the standard deviation of the Residuals(prediction errors), Residuals are a measure of how far from the regression line data points are . RMSE tells you how concentrated is the data around the BEST FIT LINE.

In [ ]:
```
mse_m4= mean_squared_error(df.T_Delivery_Time,model4.fittedvalues)
rmse_m4=math.sqrt(mse_m4)
print("the difference between actual and predicted values of model4 is :---",rmse_m4)
```

```
the difference between actual and predicted values of model4 is :--- 0.07843617878052518
```

In [ ]:
```
print("RMSE-Model1---",rmse_m1,",", "RMSE-Model 2---",rmse_m2, ",","RMSE-Model 3---", rm
```

```
RMSE-Model1--- 0.13293572986008406 , RMSE-Model 2--- 0.16828382429330135 , RMSE-Model 3-
-- 0.09352047119983505 , RMSE-Model4--- 0.07843617878052518
```

Based on a rule of thumb, it can be said that RMSE values between 0.2 and 0.5

shows that the model can relatively predict the data accurately.

Model4 is Ready to predict with 85% accuracy because adjusted rsquared value is 0.85 in model4 we have transformed data to Cube root and use ORDINARY LEAST SQUARES METHOD