In [20]:

```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
import statsmodels.formula.api as smf
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
from scipy.stats import probplot
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

Importing Data

In [21]:

```
delivery_data=pd.read_csv('delivery_time.csv')
delivery_data
```

Out[21]:

	Delivery Time	Sorting Time
0	21.00	10
1	13.50	4
2	19.75	6
3	24.00	9
4	29.00	10
5	15.35	6
6	19.00	7
7	9.50	3
8	17.90	10
9	18.75	9
10	19.83	8
11	10.75	4
12	16.68	7
13	11.50	3
14	12.03	3
15	14.88	4
16	13.75	6
17	18.11	7
18	8.00	2
19	17.83	7
20	21.50	5

In [22]:

```
delivery_data.shape
```

Out[22]:

```
(21, 2)
```

In [23]:

```
delivery_data.head()
```

Out[23]:

	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 [24]:

```
delivery_data.dtypes
```

Out[24]:

```
Delivery Time    float64
Sorting Time       int64
dtype: object
```

In [25]:

```
delivery_data.isna().sum()
```

Out[25]:

```
Delivery Time    0
Sorting Time     0
dtype: int64
```

In [26]:

```
delivery_data.describe()
```

Out[26]:

	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

EDA and Data visualizations

In [27]:

```
delivery_data.info()
```

```
<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 [28]:

```
len(delivery_data.columns) # identify the number of features
```

Out[28]:

```
2
```

In [29]:

```
delivery_data.columns # idenfity the features
```

Out[29]:

```
Index(['Delivery Time', 'Sorting Time'], dtype='object')
```

In [30]:

```
delivery_data.shape # identify the size of of the dataset
```

Out[30]:

```
(21, 2)
```

In [31]:

```
delivery_data.dtypes # identify the datatypes of the features
```

Out[31]:

```
Delivery Time    float64
Sorting Time       int64
dtype: object
```

In [32]:

```
delivery_data.isnull().values.any() # checking if dataset has empty cells
```

Out[32]:

```
False
```

In [33]:

```
delivery_data.isnull().sum() # identify the number of empty cells
```

Out[33]:

```
Delivery Time    0
Sorting Time     0
dtype: int64
```

Graphical Univariate analysis

For univariate analysis, we have Histogram, density plot, boxplot or violinplot, and Normal Q-Q plot. They help us understand the distribution of the data points and the presence of outliers.

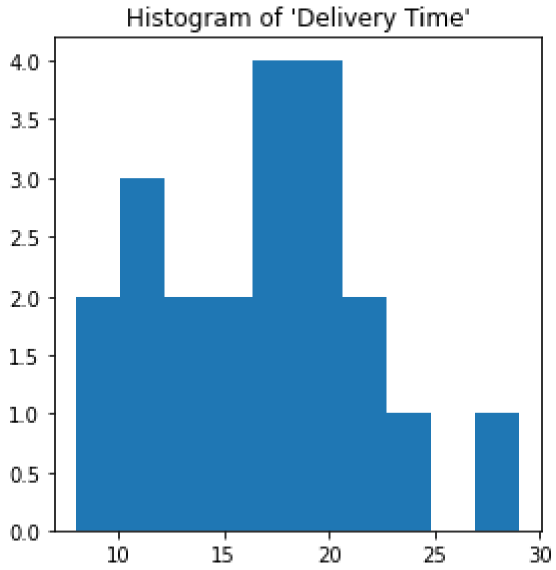
A violin plot is a method of plotting numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side.

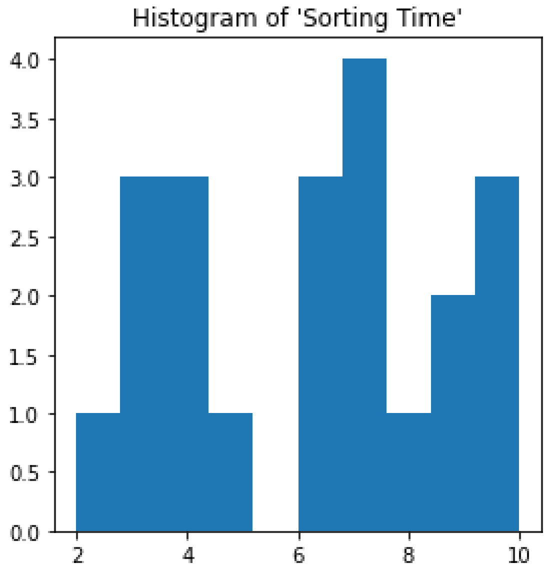
In [34]:

```
# Histogram
# We can use either plt.hist or sns.histplot
plt.figure(figsize=(20,10))
plt.subplot(2,4,1)
plt.hist(delivery_data['Delivery Time'], density=False)
plt.title("Histogram of 'Delivery Time'")
plt.subplot(2,4,5)
plt.hist(delivery_data['Sorting Time'], density=False)
plt.title("Histogram of 'Sorting Time'")
```

Out[34]:

```
Text(0.5, 1.0, "Histogram of 'Sorting Time'")
```



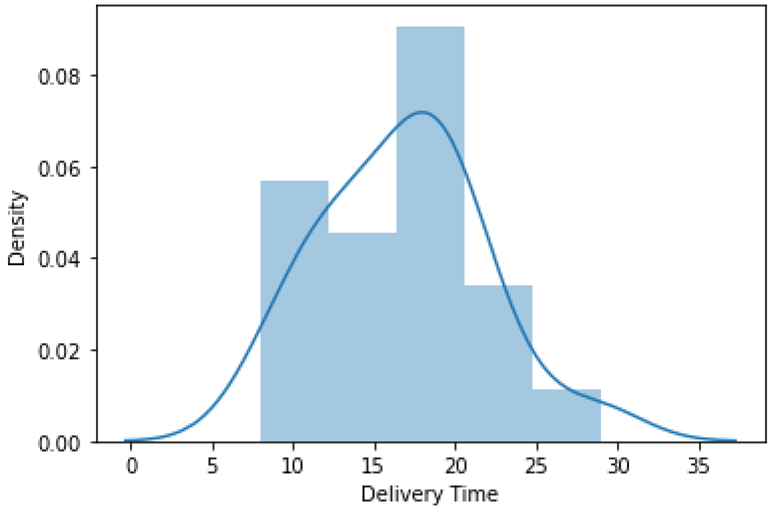


In [35]:

```
# Density plot
sns.distplot(delivery_data['Delivery Time'])
```

Out[35]:

```
<AxesSubplot:xlabel='Delivery Time', ylabel='Density'>
```

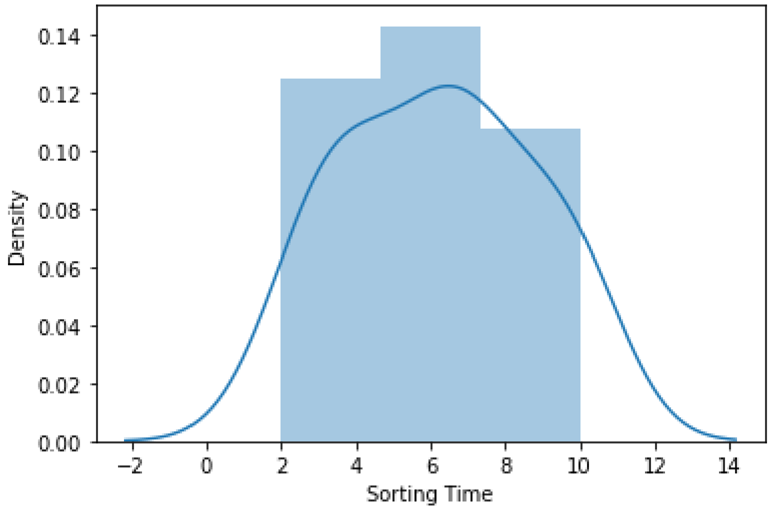


In [36]:

```
sns.distplot(delivery_data['Sorting Time'])
```

Out[36]:

```
<AxesSubplot:xlabel='Sorting Time', ylabel='Density'>
```



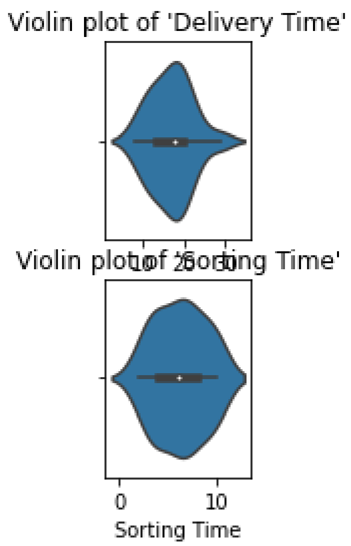
In [37]:

```
# boxplot or violin plot
# A violin plot is a method of plotting numeric data. It is similar to a box plot,
# with the addition of a rotated kernel density plot on each side
plt.subplot(2,4,3)
# plt.boxplot(delivery_data['Delivery Time'])
sns.violinplot(delivery_data['Delivery Time'])
# plt.title("Boxlpot of 'Delivery Time'")
plt.title("Violin plot of 'Delivery Time'")

plt.subplot(2,4,7)
# plt.boxplot(delivery_data['Sorting Time'])
sns.violinplot(delivery_data['Sorting Time'])
# plt.title("Boxlpot of 'Sorting Time'")
plt.title("Violin plot of 'Sorting Time'")
```

Out[37]:

```
Text(0.5, 1.0, "Violin plot of 'Sorting Time'")
```

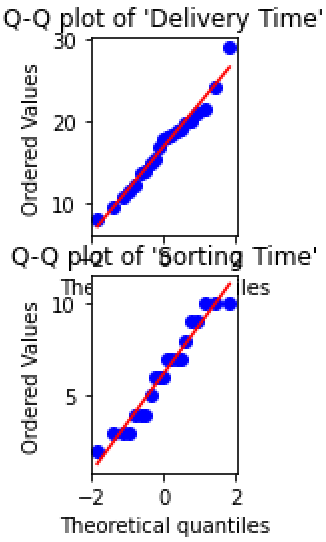


In [38]:

```
# Normal Q-Q plot
plt.subplot(2,4,4)
probplot(delivery_data['Delivery Time'], plot=plt)
plt.title("Q-Q plot of 'Delivery Time'")
plt.subplot(2,4,8)
probplot(delivery_data['Sorting Time'], plot=plt)
plt.title("Q-Q plot of 'Sorting Time'")
```

Out[38]:

```
Text(0.5, 1.0, "Q-Q plot of 'Sorting Time'")
```



#Simple Linear Regression univariate plots From the above graphical representations, we can say there are no outliers in our data, and DeliveryTime looks like normally distributed, and sorting time looks abnormal slightly. We can verify this using Shapiro Test.

In [57]:

```
 # Def a function to run Shapiro test

# Defining our Null, Alternate Hypothesis
Ho = 'Data is Normal'
Ha = 'Data is not Normal'

# Defining a significance value
alpha = 0.05
def normality_check(delivery_data):
    for columnName, columnData in delivery_data.iteritems():
        print("Shapiro test for {columnName}".format(columnName=columnName))
        res = stats.shapiro(columnData)
#         print(res)
        pValue = round(res[1], 2)

        # Writing condition
        if pValue > alpha:
            print("pvalue = {pValue} > {alpha}. We fail to reject Null Hypothesis. {Ho}".fo
        else:
            print("pvalue = {pValue} <= {alpha}. We reject Null Hypothesis. {Ha}".format(pV


# Drive code
normality_check(delivery_data)
```

```
Shapiro test for Delivery Time
pvalue = 0.9 > 0.05. We fail to reject Null Hypothesis. Data is Normal
Shapiro test for Sorting Time
pvalue = 0.19 > 0.05. We fail to reject Null Hypothesis. Data is Normal
Shapiro test for Norm_Delivery Time
pvalue = 0.9 > 0.05. We fail to reject Null Hypothesis. Data is Normal
Shapiro test for Norm_Sorting Time
pvalue = 0.19 > 0.05. We fail to reject Null Hypothesis. Data is Normal
```

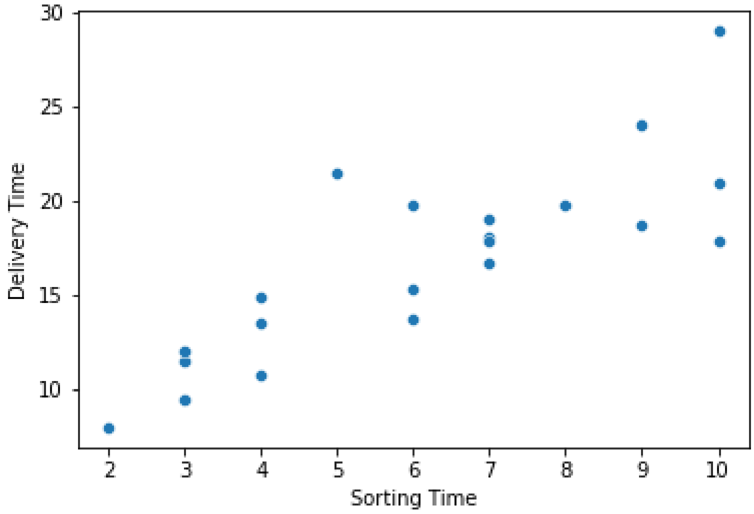
Our instinct from the graphs was some how correct. YearsExperience is normally distributed, and Salary is also normally distributed.

In [ ]:

Bi-variant analysis

In [40]:

```
#linearity check
sns.scatterplot(x='Sorting Time',y='Delivery Time',data=delivery_data)
plt.show()
```



In [41]:

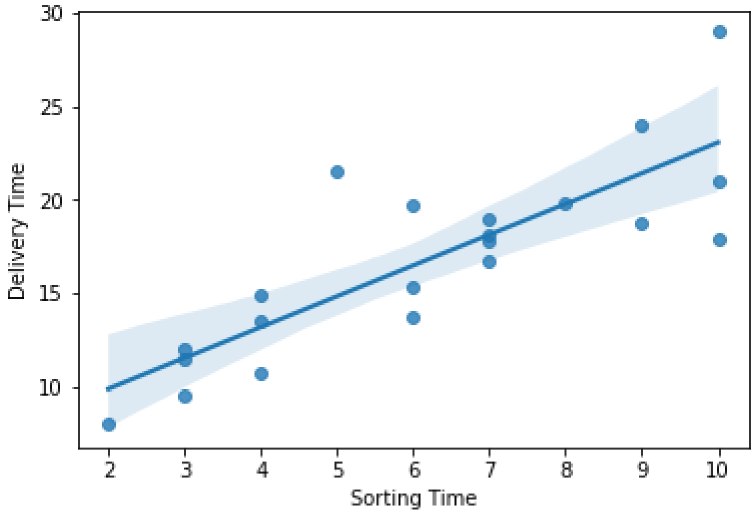
```
delivery_data.corr()
```

Out[41]:

	Delivery Time	Sorting Time
Delivery Time	1.000000	0.825997
Sorting Time	0.825997	1.000000

In [42]:

```
sns.regplot(x='Sorting Time',y='Delivery Time',data=delivery_data)
plt.show()
```



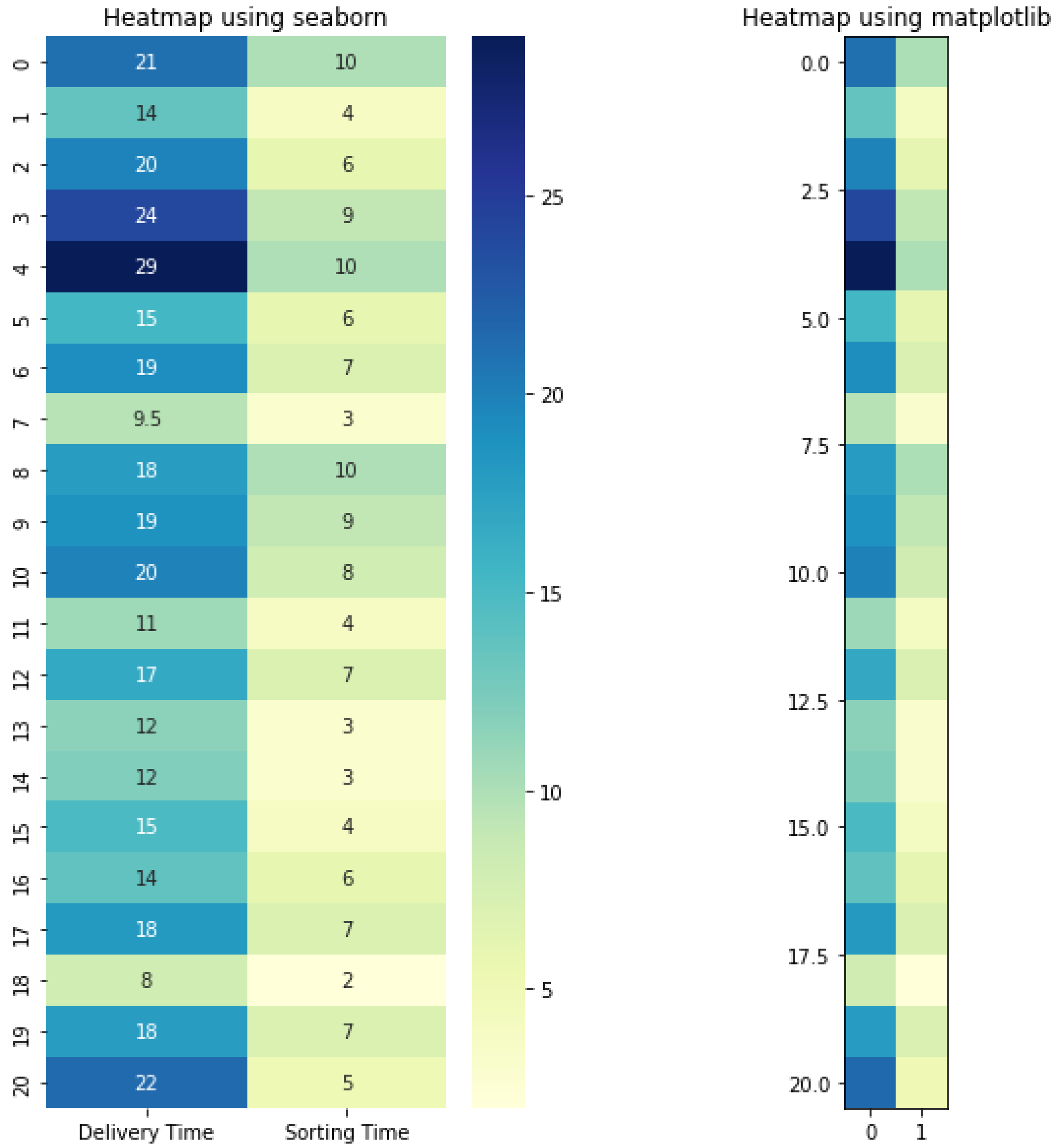
Type *Markdown* and LaTeX: α^2

In [43]:

```
# heatmap
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
sns.heatmap(data=delivery_data, cmap="YlGnBu", annot = True)
plt.title("Heatmap using seaborn")
plt.subplot(1, 2, 2)
plt.imshow(delivery_data, cmap ="YlGnBu")
plt.title("Heatmap using matplotlib")
```

Out[43]:

```
Text(0.5, 1.0, 'Heatmap using matplotlib')
```

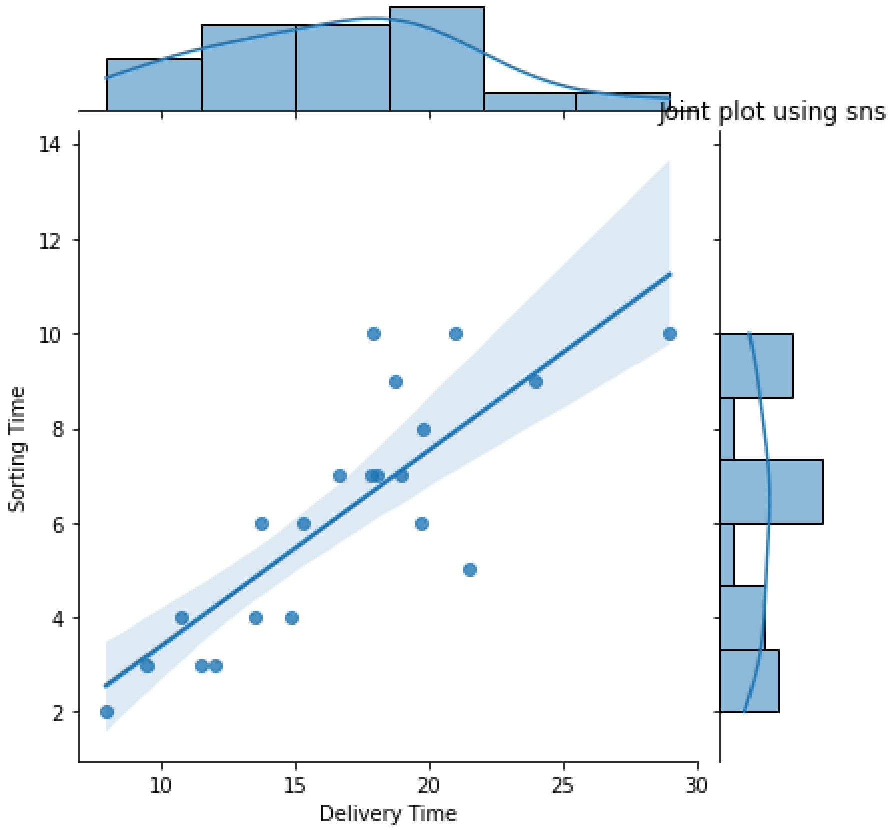


In [44]:

```
# Joint plot
sns.jointplot(x = "Delivery Time", y = "Sorting Time", kind = "reg", data = delivery_data)
plt.title("Joint plot using sns")
# kind can be hex, kde, scatter, reg, hist. When kind='reg' it shows the best fit line.
```

Out[44]:

```
Text(0.5, 1.0, 'Joint plot using sns')
```



In [45]:

```
print("Correlation: "+ 'n', delivery_data.corr()) # 0.83 which is high positive correlation
# Draw a heatmap for correlation matrix
plt.subplot(1,1,1)
sns.heatmap(delivery_data.corr(), annot=True)
```

```
Correlation: n                Delivery Time  Sorting Time
Delivery Time       1.000000      0.825997
Sorting Time        0.825997      1.000000
```

Out[45]:

```
<AxesSubplot:>
```

correlation =0.83, which is a high positive correlation. This means the dependent variable increases as the independent variable increases.

Normalization

In [46]:

```
# Create new columns for the normalized values
delivery_data['Norm_Delivery Time'] = preprocessing.normalize(delivery_data[['Delivery Time
delivery_data['Norm_Sorting Time'] = preprocessing.normalize(delivery_data[['Sorting Time']
delivery_data.head()
```

Out[46]:

	Delivery Time	Sorting Time	Norm_Delivery Time	Norm_Sorting Time
0	21.00	10	0.261770	0.327210
1	13.50	4	0.168281	0.130884
2	19.75	6	0.246188	0.196326
3	24.00	9	0.299166	0.294489
4	29.00	10	0.361492	0.327210

3.Model Building || Model Training

In [47]:

```
#renaming the columns
data_set=delivery_data.rename(columns={'Delivery Time':'delivery_data','Sorting Time':'sort
data_set
```

6	19.00	7	0.236839	0.229047
7	9.50	3	0.118420	0.098163
8	17.90	10	0.223128	0.327210
9	18.75	9	0.233723	0.294489
10	19.83	8	0.247186	0.261768
11	10.75	4	0.134001	0.130884
12	16.68	7	0.207920	0.229047
13	11.50	3	0.143350	0.098163
14	12.03	3	0.149957	0.098163
15	14.88	4	0.185483	0.130884
16	13.75	6	0.171397	0.196326
17	18.11	7	0.225745	0.229047

In [48]:

```
lin_model=smf.ols("delivery_data~sorting_data",data=data_set).fit()    #model trained
lin_model
```

Out[48]:

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x229d13bd0
d0>
```

4.Model Testing

In [49]:

```
lin_model.params
```

Out[49]:

```
Intercept       6.582734
sorting_data    1.649020
dtype: float64
```

In [50]:

```
#finding tvalues and p values
lin_model.tvalues,lin_model.pvalues
```

Out[50]:

```
(Intercept       3.823349
 sorting_data    6.387447
 dtype: float64,
 Intercept       0.001147
 sorting_data    0.000004
 dtype: float64)
```

In [51]:

```
#finding r^2 value
lin_model.rsquared , lin_model.rsquared_adj
```

Out[51]:

```
(0.6822714748417231, 0.6655489208860244)
```

5.Model Prediction

In [52]:

```
 #manual prediction for let say time 5
delivery_time=1.649020*5 + 6.582734
delivery_time
```

Out[52]:

```
14.827834
```

In [53]:

```
#Automatic prediction for say 5,8,10
new_data=pd.Series([5,8,10])
new_data
```

Out[53]:

```
0     5
1     8
2    10
dtype: int64
```

In [54]:

```
data_pred=pd.DataFrame(new_data,columns=['sorting_data'])
data_pred
```

Out[54]:

	sorting_data
0	5
1	8
2	10

In [55]:

```
lin_model.predict(data_pred)
```

Out[55]:

```
0    14.827833
1    19.774893
2    23.072933
dtype: float64
```

In [ ]:

```
```