```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

Read the data set using read_csv

```python
df = pd.read_csv('/content/delivery_time (1).csv')
```

Check the data points in the dataset

```python
print(df.head())
```

```
   Delivery Time  Sorting Time
0          21.00            10
1          13.50             4
2          19.75             6
3          24.00             9
4          29.00            10
```

```python
df.shape
```

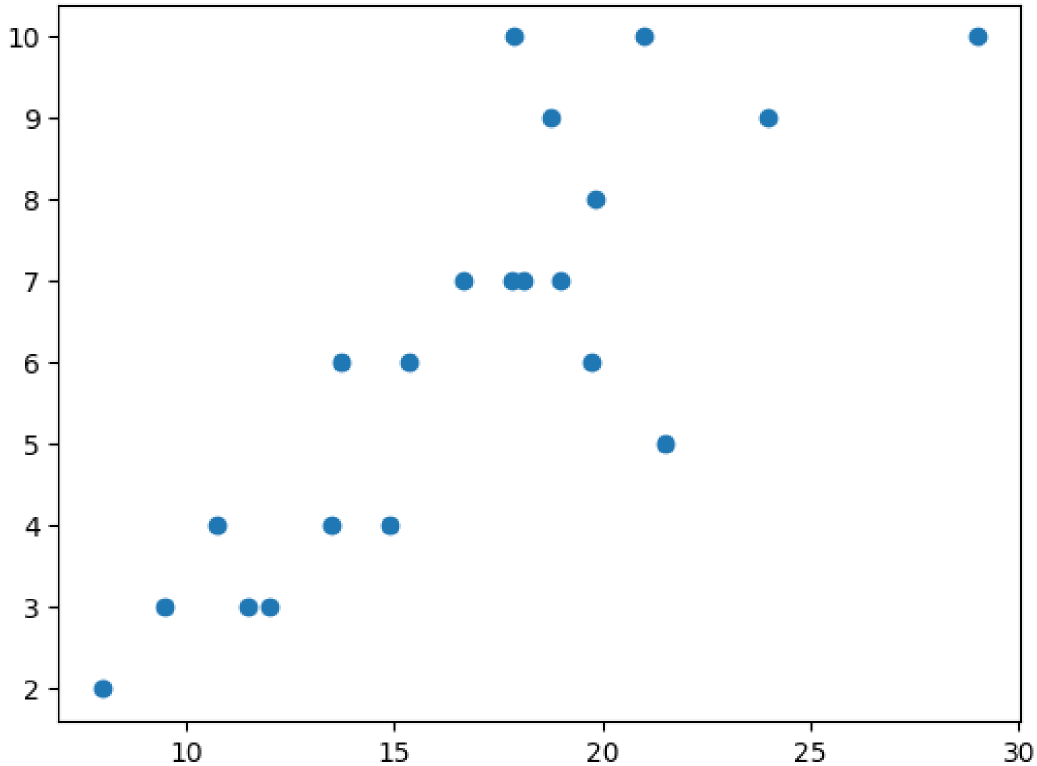
```
(21, 2)
```

```python
df.isnull().sum()
```

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

##Visualize the data set

```python
plt.scatter(df['Delivery Time'], df['Sorting Time'])
plt.show()
```

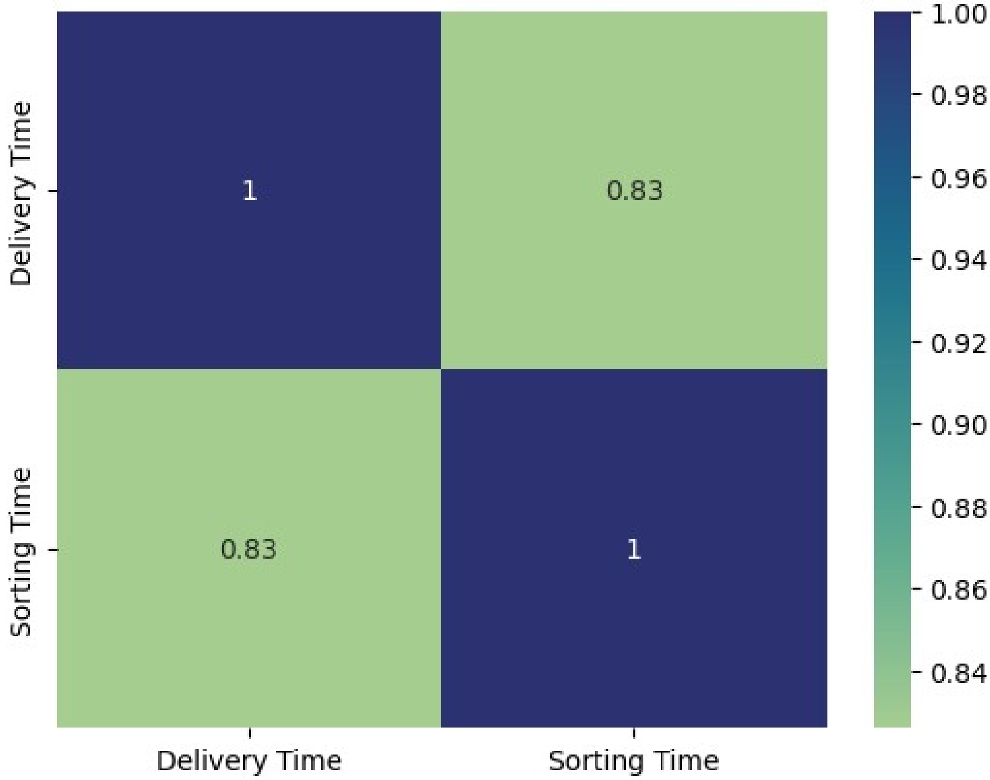



```
print(df.corr())
```

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

```
import seaborn as sns
```

```
sns.heatmap(df[['Delivery Time','Sorting Time']].corr(), annot=True,
cmap = 'crest')
plt.show()
```




```
print(df.describe())
```

```
       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
```

Simple Linear Regression

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables: One variable, **denoted x** , is regarded as the **predictor, explanatory, or independent variable**. The other variable, **denoted y**, is regarded as **the response, outcome, or dependent variable.**

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
print(X)
print(y)
```

```
    Delivery Time
0           21.00
```

```
1           13.50
2           19.75
3           24.00
4           29.00
5           15.35
6           19.00
7            9.50
8           17.90
9           18.75
10          19.83
11          10.75
12          16.68
13          11.50
14          12.03
15          14.88
16          13.75
17          18.11
18           8.00
19          17.83
20          21.50
0     10
1      4
2      6
3      9
4     10
5      6
6      7
7      3
8     10
9      9
10     8
11     4
12     7
13     3
14     3
15     4
16     6
17     7
18     2
19     7
20     5
Name: Sorting Time, dtype: int64
```

```python
print('X shape:', X.shape)
print('y shape:', y.shape)
print('X shape type:', type(X))
print('y shape type:', type(y))
```

```
X shape: (21, 1)
y shape: (21,)
```

```
X shape type: <class 'pandas.core.frame.DataFrame'>
y shape type: <class 'pandas.core.series.Series'>
```

```
#Wrong Approach
X_wrong = df.iloc[:,-1]
y_wrong = df.iloc[:,-1]
```

```
print('X_wrong shape:', X_wrong.shape , type(X_wrong) )
print('y_wrong shape:', y_wrong.shape , type(y_wrong))
```

```
X_wrong shape: (21,) <class 'pandas.core.series.Series'>
y_wrong shape: (21,) <class 'pandas.core.series.Series'>
```

##Split the data into train and test

We could already feed our X and y data directly to our linear regression model, but if we use all of our data at once, how can we know if our results are any good? Just like in learning, what we will do, is use a part of the data to train our model and another part of it, to test it

This is easily achieved through the helper **train_test_split()** method, which accepts our X and y arrays (also works on DataFrames and splits a single DataFrame into training and testing sets), and a test_size.

train_test_split() helper method from from ***sklearn.model_selection import train_test_split***

test_size is the percentage of the overall data we'll be using for testing

Some common train-test splits are 80/20 and 70/30.

random_state = SEED = Some Number(42) = Splitting data into training/validation/test sets: random seeds ensure that the data is divided the same way every time the code is run

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=42)
```

```
print(X_train)
```

```
    Delivery Time
5           15.35
11          10.75
3           24.00
18           8.00
16          13.75
13          11.50
2           19.75
9           18.75
20          21.50
4           29.00
12          16.68
```

```
7             9.50
10           19.83
14           12.03
19           17.83
6            19.00
```

```
print(y_train)
```

```
5      6
11     4
3      9
18     2
16     6
13     3
2      6
9      9
20     5
4     10
12     7
7      3
10     8
14     3
19     7
6      7
Name: Sorting Time, dtype: int64
```

##Training a Linear Regression Model

our train and test sets ready. Scikit-Learn has a various model types we can easily import and train

need to fit the line to our data, we will do that by using the **.fit()** method along with our X_train and y_train data

You can inspect the intercept and slope by printing the **regressor.intecept_** and **regressor.coef_** attributes

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
regressor.intercept_
```

```
-0.3292249777418226
```

```
regressor.coef_
```

```
array([0.37522491])
```

```
print(regressor.coef_[0])
```

```
0.3752249069825206
```

##Making Predictions

our own formula that calculates the value or call on the **predict()** function

```
def calc(slope, intercept, Delivery_Time):
    return slope*(Delivery_Time)+intercept
```

```
score = calc(regressor.coef_, regressor.intercept_, 9.5)
print(score)
```

```
[3.23541164]
```

```
score = regressor.predict([[9.5]])
print(score)
```

```
[3.23541164]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but LinearRegression
was fitted with feature names
  warnings.warn(
```

```
y_pred = regressor.predict(X_test)
```

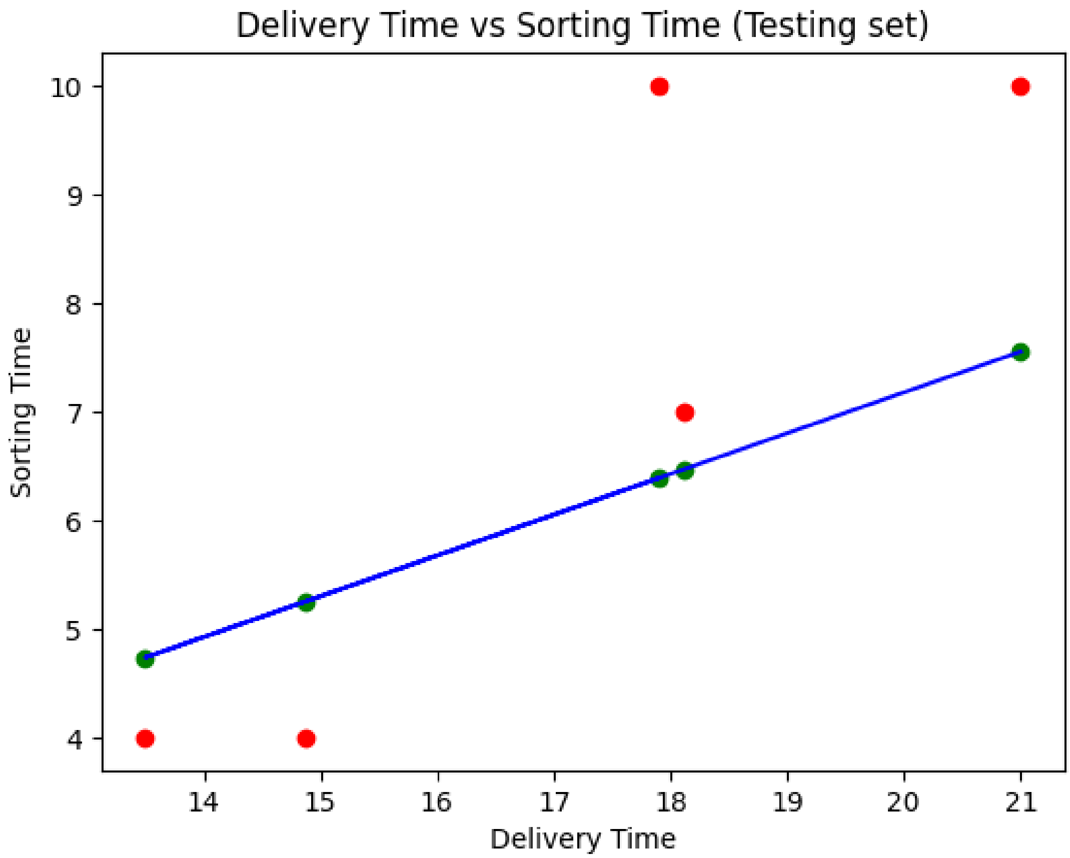
```
df_preds = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df_preds)
```

```
    Actual  Predicted
0       10   7.550498
17       7   6.466098
15       4   5.254122
1        4   4.736311
8       10   6.387301
```

##Visualize the Actual Vs Predicted

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.scatter(X_train, regressor.predict(X_train), color = 'green')
plt.title('Delivery Time vs Sorting Time (Training set)')
plt.xlabel('Delivery Time')
plt.ylabel('Sorting Time')
plt.show()
```


```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, regressor.predict(X_test), color = 'blue')
plt.scatter(X_test, regressor.predict(X_test), color = 'green')
plt.title('Delivery Time vs Sorting Time (Testing set)')
plt.xlabel('Delivery Time')
plt.ylabel('Sorting Time')
plt.show()
```



##Evaluating the Model

Using from sklearn.metrics import mean_absolute_error, mean_squared_error

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(mae)
print(mse)
print(rmse)
```

```
1.7173071781664517
4.290336284603159
2.071312695998158
```