In [17]:

```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
```

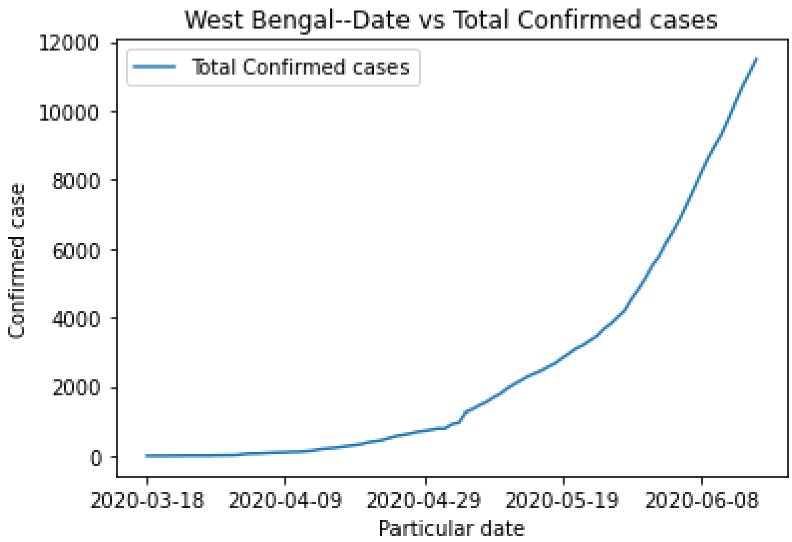
In [18]:

```
df = pd.read_csv('D:\Covid\complete-Covid-19.csv')# read whole data
print('Covid Dataframe rows and col::\t',df.shape)
df2 = df[(df['Name of State / UT'] == 'West Bengal')] #read data of westbengal 89 ro
print('WB dataframe rows and col::\t',df2.shape)
print('\nWB dataframe Description::\n\n',df2.describe())
```

```
Covid Dataframe rows and col::   (3056, 10)
WB dataframe rows and col::      (89, 10)

WB dataframe Description::

            Latitude     Longitude  Total Confirmed cases       Death  \
count  8.900000e+01  8.900000e+01              89.000000   89.000000
mean   2.298680e+01  8.785500e+01            2508.404494  145.842697
std    4.287411e-14  2.000792e-13            3160.836194  157.859467
min    2.298680e+01  8.785500e+01               1.000000    0.000000
25%    2.298680e+01  8.785500e+01             116.000000    5.000000
50%    2.298680e+01  8.785500e+01             922.000000   33.000000
75%    2.298680e+01  8.785500e+01            3667.000000  272.000000
max    2.298680e+01  8.785500e+01           11494.000000  485.000000

       Cured/Discharged/Migrated   New cases  New deaths  New recovered
count                  89.000000   89.000000   89.000000      89.000000
mean                  939.595506  129.134831    5.449438      61.730337
std                  1354.832720  147.579577   10.934735      95.343609
min                     0.000000    0.000000    0.000000       0.000000
25%                    16.000000   16.000000    0.000000       0.000000
50%                   151.000000   58.000000    3.000000       9.000000
75%                  1339.000000  183.000000    8.000000      90.000000
max                  5494.000000  476.000000   98.000000     518.000000
```

In [19]:

```
df2.plot(x='Date', y='Total Confirmed cases', style='-')
plt.title('West Bengal--Date vs Total Confirmed cases')
plt.xlabel('Particular date')
plt.ylabel('Confirmed case')
plt.show() #plot date Vs confirm case graph
'''From the graph above, we can clearly see that there is a positive linear relation
the number of hours studied and percentage of score.'''
#Now we have an idea about statistical details of our data.
#The next step is to divide the data into "attributes" and "labels".
'''Attributes are the independent variables(here it is X) while labels are dependent
whose values are to be predicted.'''
#We want to predict the number of confirmed cases on a particular day.
#Therefore our attribute set will consist of the "Date" column, and the label will b
```



Out[19]:
```
'Attributes are the independent variables(here it is X) while labels are dependent v
ariables(here it is Y) \nwhose values are to be predicted.'
```

In [20]:
```
# iloc is integer-location based indexing for selection by position.
#it will select till the second last column of the data frame instead of the last co

Z = df2.iloc[:,0].values # iloc is integer-location based indexing for selection by
X2=[(datetime.strptime(t,'%Y-%m-%d').date()-datetime.strptime(Z[0],'%Y-%m-%d').date(
X=np.array(X2)
y = df2.iloc[:, 4].values
```

In [21]:
```
print(X) # Days of WB cases
print(y) # No of WB confirmed case on particular day
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 16 17 18 19 20 21 22 23 24 25
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73
 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90]
[    1     1     1     3     4     7     9     9    10    10    15    18
    19    26    53    69    69    80    91    99   103   116   116   134
   152   190   213   231   255   287   310   339   392   423   456   514
   571   611   649   697   725   758   795   795   922   963  1259  1344
  1456  1548  1678  1786  1939  2063  2173  2290  2377  2461  2576  2677
  2825  2961  3103  3197  3332  3459  3667  3816  4009  4192  4536  4813
  5130  5501  5772  6168  6508  6876  7303  7738  8187  8613  8985  9328
  9768 10244 10698 11087 11494]
```

In [22]:
```
#Now that we have our attributes and labels, the next step is to split this data int
#We'll do this by using Scikit-Learn's built-in train_test_split() method:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
X_train=X_train.reshape(-1,1)
X_test=X_test.reshape(-1,1)
'''The above script splits 70% of the data to training set while 30% of the data to
The test_size variable is where we actually specify the proportion of test set.'''
```

Out[22]:
```
'The above script splits 70% of the data to training set while 30% of the data to te
st set.\nThe test_size variable is where we actually specify the proportion of test
set.'
```

In [23]:
```
#Train the Algorithm
'''With Scikit-Learn it is extremely straight forward to implement linear regression
```

```
to do is import the LinearRegression class, instantiate it, and call the fit() metho
This is about as simple as it gets when using a machine learning library to train on

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

'''The linear regression model basically finds the best value for the intercept and
best fits the data.
To see the value of the intercept and slop calculated by the linear regression algor
execute the following code.'''

print("Regression intercept\t",regressor.intercept_) # To retrieve the intercept
print("Regression cofficient\t",regressor.coef_) #  retrieving the slope (coefficien
#y=mx+b
#This means that for every one day, the change in the confirmed cases is about appro
```

```
Regression intercept     -2319.7978205731974
Regression cofficient    [106.4427851]
```

In [24]:

```
#Making Predictions
'''To do predictions, we will use our test data and see how accurately our algorithm

y_pred = regressor.predict(X_test)
# To compare the actual output values for X_test with the predicted values, execute
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print('\n',df)
```

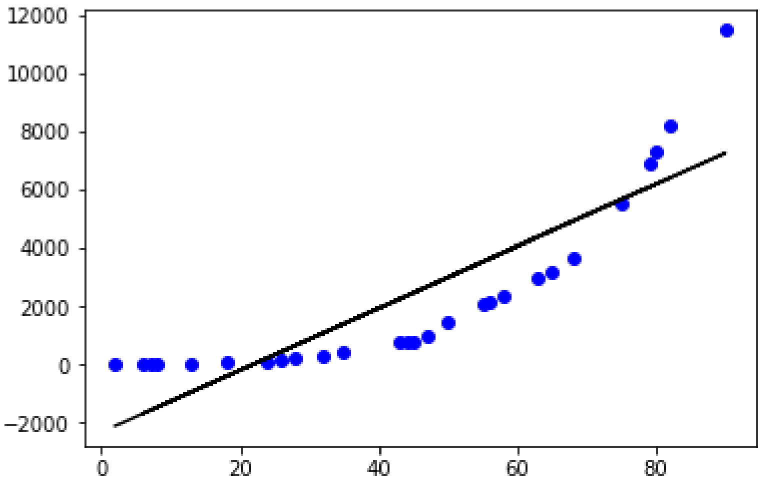
```
    Actual    Predicted
0        1 -2106.912250
1       26  -936.041614
2     2063  3534.555360
3      758  2257.241939
4     3667  4918.311566
5      310  1086.371303
6      963  2683.013079
7      795  2470.127509
8     7303  6195.624987
9    11494  7260.052838
10       9 -1574.698325
11     213   660.600162
12     423  1405.699658
13    3197  4598.983211
14      10 -1468.255540
15      69  -403.827689
16     152   447.714592
17    2377  3853.883715
18    6876  6089.182202
19     795  2363.684724
20     116   234.829022
21       9 -1681.141110
22    2961  4386.097641
23    1456  3002.341434
24    8187  6408.510557
25    2173  3640.998145
26    5501  5663.411062
```

In [25]:

```
#the predicted percentages are close to the actual ones.

print(X_test.shape)
print(y_test.shape)
print(y_pred.shape)
print("\n Prediction V/s Actual \n")
```

```
plt.scatter(X_test, y_test, color='b')
plt.plot(X_test,y_pred, color='k')
plt.show()
```

```
(27, 1)
(27,)
(27,)

 Prediction V/s Actual
```



In [26]:

```
#Evaluating the Algorithm

'''The final step is to evaluate the performance of algorithm.
This step is particularly important to compare how well different algorithms perform
For regression algorithms, three evaluation metrics are commonly used:
    3.1 Mean Absolute Error (MAE) is the mean of the absolute value of the errors.
    3.2 Mean Squared Error (MSE) is the mean of the squared errors.
    3.3 Root Mean Squared Error (RMSE) is the square root of the mean of the squared

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)


df2.describe()

print("""\n\nWe can see that the value of root mean squared error is 1526 , which is
about 50% of the mean value of the percentages of all the Confirmed cases i.e. 2508.
This means that our algorithm did a decent job.\n """)
```

```
Mean Absolute Error: 1314.4211134431707
Mean Squared Error: 2329697.2161476873
Root Mean Squared Error: 1526.3345688765905


We can see that the value of root mean squared error is 1526 , which is
about 50% of the mean value of the percentages of all the Confirmed cases i.e. 2508.
40.
This means that our algorithm did a decent job.
```

In [ ]:

In [ ]: