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

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```


dataset_train=pd.read_csv('/content/drive/My Drive/BHARTIARTL.BO.csv')
dataset_train.drop(89,axis=0,inplace=True)
training_set = dataset_train.iloc[:, 1:2].values
```

```
dataset_train.shape
```

```
(250, 7)
```

```
dataset_train.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2020-07-13	575.000000	589.099976	570.000000	586.750000	584.650146	502298.0
1	2020-07-14	586.000000	596.599976	578.700012	589.099976	586.991699	1110186.0
2	2020-07-15	589.799988	589.799988	559.700012	564.150024	562.131042	652407.0
3	2020-07-16	565.000000	567.349976	553.500000	562.599976	560.586548	630567.0

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

```
X_train = []
y_train = []
for i in range(60, 250):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

X_train.shape

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))
#from keras import optimizers

#optimizer = optimizers.Adam(clipvalue=1.0)
##regressor.compile(optimizer=optimizer, loss='mean_squared_error')


regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

```
Epoch 1/100
6/6 [==============================] - 8s 111ms/step - loss: 0.2458
Epoch 2/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0785
Epoch 3/100
6/6 [==============================] - 1s 110ms/step - loss: 0.0420
Epoch 4/100
6/6 [==============================] - 1s 110ms/step - loss: 0.0325
Epoch 5/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0254
Epoch 6/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0288
Epoch 7/100
6/6 [==============================] - 1s 109ms/step - loss: 0.0263
Epoch 8/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0251
Epoch 9/100
6/6 [==============================] - 1s 113ms/step - loss: 0.0195
Epoch 10/100
6/6 [==============================] - 1s 110ms/step - loss: 0.0195
Epoch 11/100
6/6 [==============================] - 1s 115ms/step - loss: 0.0226
Epoch 12/100
```

```
6/6 [==============================] - 1s 112ms/step - loss: 0.0173
Epoch 13/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0174
Epoch 14/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0157
Epoch 15/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0172
Epoch 16/100
6/6 [==============================] - 1s 109ms/step - loss: 0.0155
Epoch 17/100
6/6 [==============================] - 1s 114ms/step - loss: 0.0165
Epoch 18/100
6/6 [==============================] - 1s 113ms/step - loss: 0.0118
Epoch 19/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0127
Epoch 20/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0129
Epoch 21/100
6/6 [==============================] - 1s 114ms/step - loss: 0.0125
Epoch 22/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0153
Epoch 23/100
6/6 [==============================] - 1s 115ms/step - loss: 0.0136
Epoch 24/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0144
Epoch 25/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0121
Epoch 26/100
6/6 [==============================] - 1s 114ms/step - loss: 0.0118
Epoch 27/100
6/6 [==============================] - 1s 112ms/step - loss: 0.0124
Epoch 28/100
6/6 [==============================] - 1s 111ms/step - loss: 0.0134
Epoch 29/100
6/6 [==============================] - 1s 113ms/step - loss: 0.0134
Epoch 30/100
```

```
dataset_total = pd.concat((dataset_train['Open'], dataset_train['Open']), axis = 0)
inputs = dataset_train.iloc[len(dataset_total) - len(dataset_train) - 60:,1:2].values
inputs.shape
```

```
(60, 1)
```

```
#inputs = inputs[::-1]
```

```
output = []
for i in range(100):
  inputs = inputs.reshape(-1,1)
  inputs = sc.transform(inputs)
  X_test = []
  X_test.append(inputs[0:60, 0])
  X_test = np.array(X_test)
  X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
  predicted_stock_price = regressor.predict(X_test)
  predicted_stock_price = sc.inverse_transform(predicted_stock_price)
  #predicted_stock_price.reshape(1,1)
  output.append(predicted_stock_price[0][0])
  inputs=sc.inverse_transform(inputs)
  inputs = np.delete(inputs, (0), axis=0)
  inputs=np.append(inputs, np.array([[predicted_stock_price[0][0]]]), axis=0)
```

```
val
```

```
[-8.7567138671875,
 -4.7432861328125,
 -6.734130859375,
 -14.61761474609375,
 -5649.617614746094]
```

```
adani
```

```
[1424.2433,
 1419.6614,
 1415.3517,
 1411.5669,
1408.4945,
1406.2528,
1404.9008,
1404.4127,
1404.7206,
1405.7554,
1407.4072,
1409.5812,
1412.1597,
1415.0375,
1418.128,
1421.3256,
1424.5887,
1427.8843,
1431.1964,
1434.427,
1437.521,
1440.5026,
1443.4181,
1446.1599,
1448.7963,
1451.3175,
1453.7391,
1456.0347,
1458.2747,
1460.4347,
1462.548,
1464.6013,
1466.5654,
1468.4404,
1470.2261,
1471.946,
1473.5825,
1475.1458,
1476.6425,
1478.0803,
1479.443,
1480.758,
1482.0411,
1483.2479,
1484.4006,
1485.5096,
1486.5707,
       1487.5834,
       1488.5623,
       1489.4979,
       1490.3905,
       1491.2803,
       1492.1218,
       1492.9188,
       1493.6772,
       1494.4076,
       1495.1108,
       1495.7819,
       1496.4286,
```

axis

```
      [754.2567,
       755.1296,
       755.8988,
       756.6884,
       757.3039,
       757.65106,
       758.04443,
       758.37103,
       758.73816,
       759.2092,
       759.71783,
       760.2726,
       760.82056,
       761.33954,
       761.871,
       762.41425,
       763.0828,
       763.7315,
       764.39795,
       765.0265,
       765.4155,
       765.87756,
       766.25415,
       766.58575,
       766.71027,
       766.72864,
       766.79333,
       766.85535,
       766.8122,
       766.8795,
       766.84094,
       766.87805,
       766.93005,
       766.9337,
       767.00214,
       767.02356,
       767.0925,
       767.208,
       767.2753,
       767.4119,
       767.5325,
       767.5585,
       767.6362,
       767.7085,
       767.88226,
       767.90015,
       767.91113,
       767.9604,
       767.9743,
       767.7916,
       767.6637,
       767.6157,
       767.5912,
       767.5834,
       767.5095,
       767.45154,
       767.41626,
       767.3682,
       767.3946,
```

```
hul
```

```
[2446.2659,
 2438.6365,
 2431.182,
 2423.994,
 2416.908,
 2409.9043,
 2402.9602,
 2396.0422,
 2389.0803,
 2382.0156,
 2374.816,
 2367.449,
 2359.7627,
 2351.7637,
 2343.5073,
 2334.907,
 2325.968,
 2316.7747,
 2307.528,
 2298.4307,
 2289.175,
 2280.1572,
 2271.904,
 2264.3994,
 2257.993,
 2252.6787,
 2248.7842,
 2246.4841,
 2245.9478,
 2247.3928,
 2250.7566,
 2255.8066,
 2262.705,
 2271.4038,
 2281.9778,
 2294.483,
 2308.7239,
 2324.1553,
 2340.876,
 2358.541,
 2377.2544,
 2396.3826,
 2415.3894,
 2433.9885,
 2451.6558,
 2467.8313,
 2482.1484,
 2494.0808,
 2502.989,
 2508.5261,
 2510.4797,
 2509.1987,
 2505.3005,
 2499.1494,
 2491.3645,
 2482.6243,
 2473.275,
 2463.628,
```

airtel

```
 [525.3824,
  525.22546,
  525.2795,
  525.4147,
  525.54846,
  525.64026,
  525.6791,
  525.6758,
  525.64905,
  525.61707,
  525.5894,
  525.57776,
  525.59393,
  525.63165,
  525.6923,
  525.773,
  525.87463,
  525.9912,
  526.12195,
  526.2643,
  526.41815,
  526.57825,
  526.7458,
  526.91907,
  527.0979,
  527.2836,
  527.4729,
  527.66504,
  527.85834,
  528.05164,
  528.24603,
  528.44147,
  528.6337,
  528.8239,
  529.0114,
  529.19507,
  529.37524,
  529.55206,
  529.7222,
  529.8874,
  530.04504,
  530.1968,
  530.34125,
  530.4781,
  530.6064,
  530.7276,
  530.8437,
  530.9501,
  531.04803,
  531.1374,
  531.22,
  531.2937,
  531.35913,
  531.4168,
  531.46686,
  531.50934,
  531.5443,
  531.5724,
  531.59436,
```

```
bajaj
```

```
[525.3824,
 525.22546,
 525.2795,
 525.4147,
 525.54846,
 525.64026,
 525.6791,
 525.6758,
 525.64905,
 525.61707,
 525.5894,
 525.57776,
 525.59393,
525.63165,
525.6923,
525.773,
525.87463,
525.9912,
526.12195,
526.2643,
526.41815,
526.57825,
526.7458,
526.91907,
527.0979,
527.2836,
527.4729,
527.66504,
527.85834,
528.05164,
528.24603,
528.44147,
528.6337,
528.8239,
529.0114,
529.19507,
529.37524,
529.55206,
529.7222,
529.8874,
530.04504,
530.1968,
530.34125,
530.4781,
530.6064,
530.7276,
530.8437,
530.9501,
531.04803,
531.1374,
531.22,
531.2937,
531.35913,
531.4168,
531.46686,
531.50934,
531.5443,
531.5724,
```

```
start = [1433, 759, 2453, 540, 6175]
```

```
def unboundedKnapsack(W, n, val, wt):

    # dp[i] is going to store maximum
    # value with knapsack capacity i.
    dp = [0 for i in range(W + 1)]

    ans = 0

    # Fill dp[] using above recursive formula
    for i in range(W + 1):
        for j in range(n):
            if (wt[j] <= i):
                dp[i] = max(dp[i], dp[i - wt[j]] + val[j])

    return dp[W]

# Driver program
W = 20000
day = int(input("Enter days: "))

max_return =[]
for days in range(1):
  val = [(adani[days]-start[0]), (axis[days]-start[1]), (hul[days]-start[2]), (airtel[days]-start[3]), bajaj[days]-start[4]]
  val
```

```
    Enter days: 5
```

```
maxreturn=pd.DataFrame(max_return)
maxreturn.to_csv("max_returns.csv")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-e536b2b6750a> in <module>()
----> 1 maxreturn=pd.DataFrame(max_return)
      2 maxreturn.to_csv("max_returns.csv")

NameError: name 'pd' is not defined
```

```
from google.colab import files
files.download('max_returns.csv')
```