PROJECT REPORT

MACHINE LEARNING

STOCK PRICE PREDICTION

SUBMITTED TO

DR. SHEFALI GUPTA

BY-

DEVANSH GAHALAUT

RAHUL GUPTA

YASH NEGI

PROJECT STATEMENT :-

We will use the Long Short-Term Memory(LSTM) method to create a Machine Learning model to forecast stock values. Long-term memory (LSTM) is a deep learning artificial recurrent neural network (RNN) architecture.

Unlike traditional feed-forward neural networks, LSTM has feedback connections. It can handle single data points (such as pictures) as well as full data sequences (such as speech or video).

WORKING MODEL :-

The dataset of the training model is fetched dynamically with the help of yahoo finance API. The API gives as result various columns as output like Date, Open price, Close price, Adjusted Close price, Volume etc.

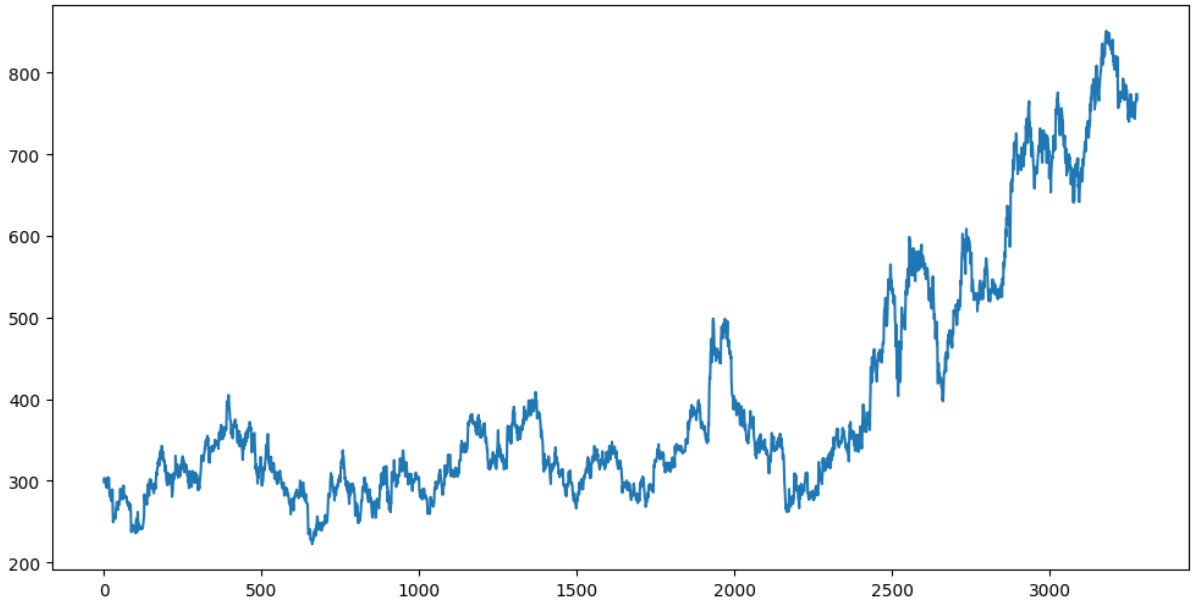
In [247]:
```
start = '2010-01-01'
end = '2023-04-17'
df = yf.download("BHARTIARTL.NS", start, end)
df.head()
```

```
[**********100%**********]  1 of 1 completed
```

Out[247]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	302.861145	304.927032	297.352051	298.591614	280.579346	3215431
2010-01-05	301.392059	304.835205	298.867065	303.320221	285.022736	4685390
2010-01-06	305.707489	307.727478	299.417969	300.106598	282.002960	4802900
2010-01-07	301.483856	306.671570	301.162506	302.447968	284.203094	4213709
2010-01-08	302.769318	304.330231	298.086609	298.453888	280.449951	2892365

The original graph of the historic data (for some stock say - BHARTIARTL.NS) is:-



The model uses only the Closing price of a particular day to predict the price of some another day in the following way:

Let n be the day of which we want to predict the market closing price for a particular stock. We have used the values of (n - 100) days to predict the output for the nth day. The model takes as input the previous hundred days values and predicts as output the value of the stock price at nth day. Following snippet demonstrates the same:-

In [62]:
```
data_training_array = scaler.fit_transform(data_training)
```

In [63]:
```
x_train = []
y_train = []

for i in range(100, data_training_array.shape[0]):
    x_train.append(data_training_array[i-100: i])
    y_train.append(data_training_array[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
```

In [64]:
```
past_100_days = data_training.tail(100)
```

In [65]:
```
final_df = past_100_days.append(data_testing, ignore_index = True)
```

Imported Libraries :-

In [52]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as pdr
import yfinance as yf
```

In [69]:
```
from tensorflow import keras
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
```

Implemented Model :-

In [69]:
```
from tensorflow import keras
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
```

In [70]:
```
model = Sequential()
model.add(LSTM(units = 100, activation = 'relu', return_sequences = True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.1))

model.add(LSTM(units = 67, activation = 'relu', return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 45, activation = 'relu', return_sequences = True))
model.add(Dropout(0.3))

model.add(LSTM(units = 29, activation = 'relu'))
model.add(Dropout(0.3))

model.add(Dense(units = 1))
```

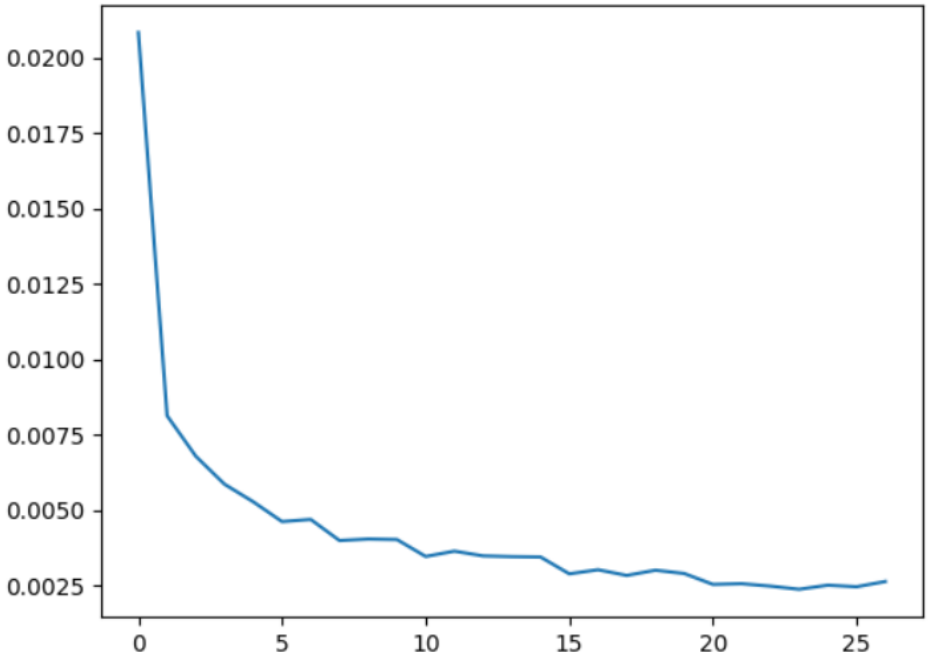
In [71]:
```
early_stop = EarlyStopping(monitor = "loss", verbose = 1, mode = 'min', patience = 3)

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

model.fit(x_train, y_train, epochs = 50, validation_data = (x_test,y_test), callbacks=[early_stop])
```

```
Epoch 1/50
79/79 [==============================] - 15s 131ms/step - loss: 0.0208 - val_loss: 0.0071
Epoch 2/50
79/79 [==============================] - 10s 123ms/step - loss: 0.0081 - val_loss: 0.0027
Epoch 3/50
79/79 [==============================] - 10s 123ms/step - loss: 0.0068 - val_loss: 0.0028
Epoch 4/50
79/79 [==============================] - 10s 129ms/step - loss: 0.0058 - val_loss: 0.0015
```

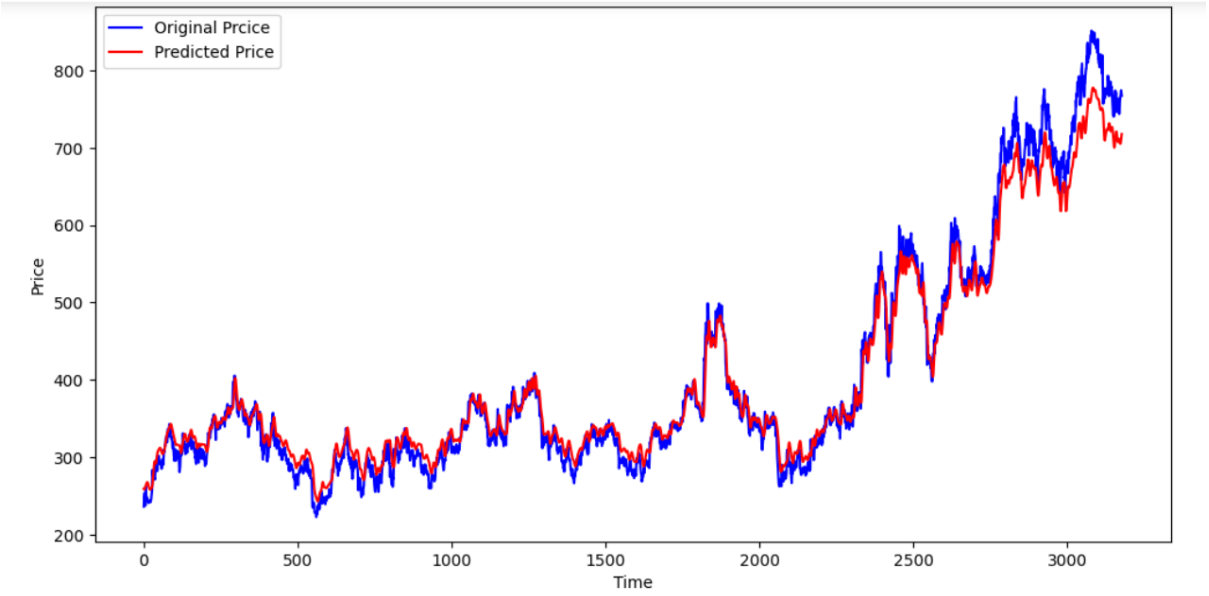
Loss Graph :-

```
In [72]:
1 loss = model.history.history['loss']
2 plt.plot(loss)
```

```
Out[72]: [<matplotlib.lines.Line2D at 0x1f4fc33a910>]
```



Original Price Vs Predicted Price :-



Predicting Next 30 Days Stock Price :-

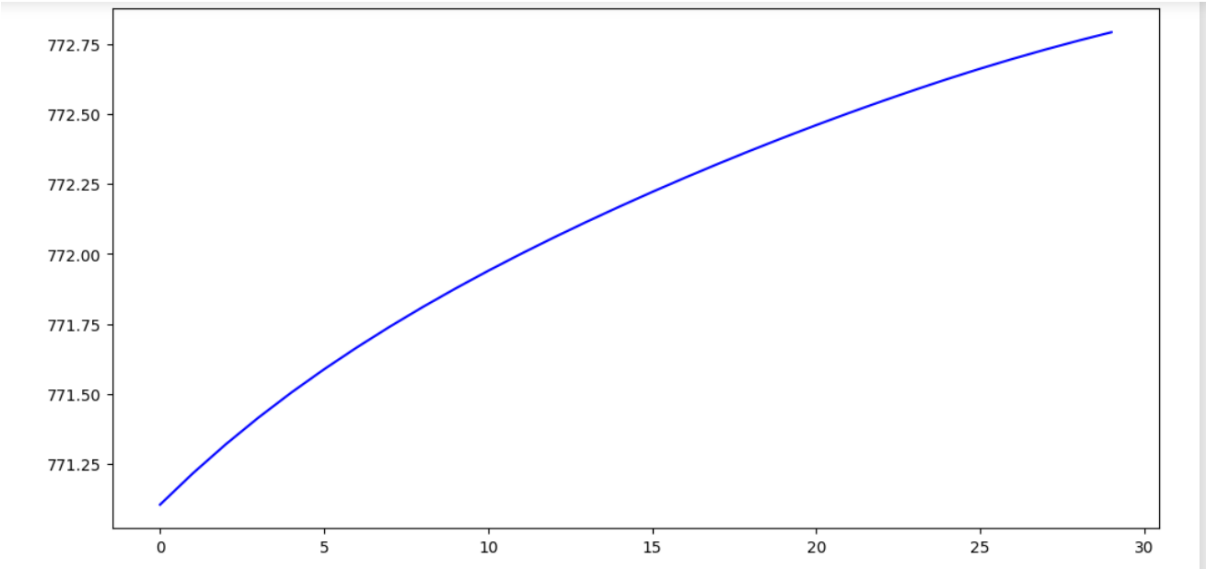
In [86]:
```
lst_output = list()
n_steps = 100
i = 0
while(i<30):
    if(len(tmp_inp) > 100):
        fut_inp = np.array(tmp_inp[1:])
        fut_inp = fut_inp.reshape(1,-1)
        fut_inp = fut_inp.reshape((1, n_steps, 1))
        yhat = model.predict(fut_inp, verbose = 0)
        tmp_inp.extend(yhat[0].tolist())
        tmp_inp = tmp_inp[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        fut_inp = fut_inp.reshape((1, n_steps, 1))
        yhat = model.predict(fut_inp,verbose=0)
        tmp_inp.extend(yhat[0].tolist())
        lst_output.extend(yhat[0].tolist())
        i=i+1

print(lst_output)
```

```
[[0.27906766533851624], [0.28007543087005615], [0.28100600838661194], [0.2818688452243805], [0.2826719284057617], [0.2834216058
254242], [0.28412556648254395], [0.2847898006439209], [0.28541889786720276], [0.28601670265197754], [0.2865869104862213], [0.28
71328294277191], [0.2876573503017 4255], [0.28816282749176025], [0.28865110874176025], [0.28912353515625], [0.2895811200141907],
[0.2900245785713196], [0.2904543578624 7253], [0.290870726108551], [0.29127392172813416], [0.2916640043258667], [0.2920409739017
4866], [0.2924048602581024], [0.2927557229995 7275], [0.2930898368358612], [0.2934069037437439], [0.29370778799057007], [0.29399
263858795166], [0.2942612767219 5435]]
```

In [87]:
```
len(lst_output)
```

Out[87]: 30

Graph of Next 30 Days :-



Accuracy of the Model Using r2 Score :-

In [95]:

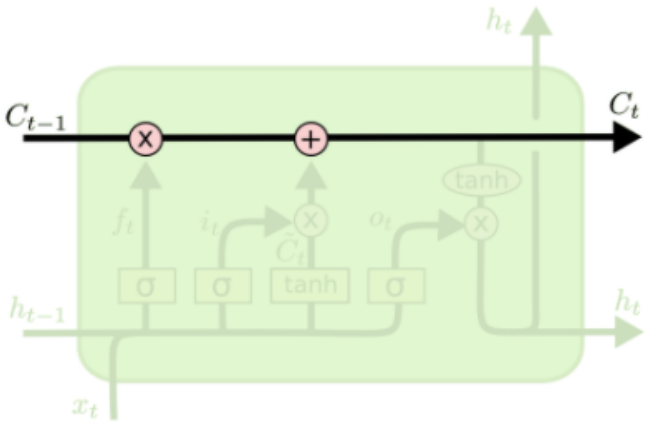
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
print('R2 Score: ', r2_score(y_test, y_predicted))
print('MAE: ', mean_absolute_error(y_test, y_predicted))
```

```
R2 Score:  0.9764746162188322
MAE:  16.897666345383854
```

ALGORITHM USED :-

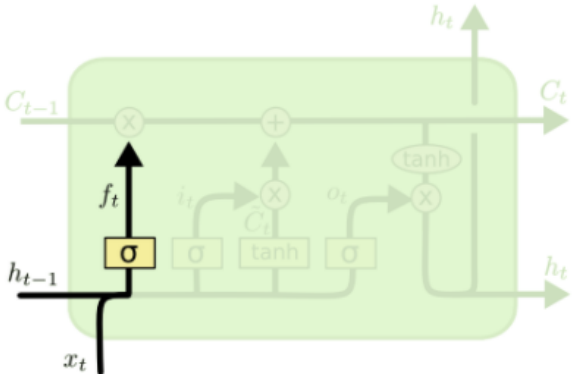
LSTM, short for Long Short-term Memory, is an extremely powerful algorithm for time series. It can capture historical trend patterns, and predict future values with high accuracy.

In a nutshell, the key component to understand an LSTM model is the Cell State (*Ct*), which represents the internal short-term and long-term memories of a cell.



To control and manage the cell state, an LSTM model contains three gates/layers. It's worth mentioning that the "gates" here can be treated as filters to let information in (being remembered) or out (being forgotten).

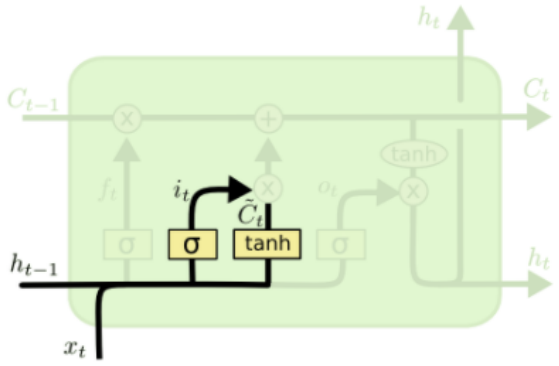
Forget gate :



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

As the name implies, forget gate decides which information to throw away from the current cell state. Mathematically, it applies a sigmoid function to output/returns a value between [0, 1] for each value from the previous cell state (*Ct-1*); here '1' indicates "completely passing through" whereas '0' indicates "completely filtering out".

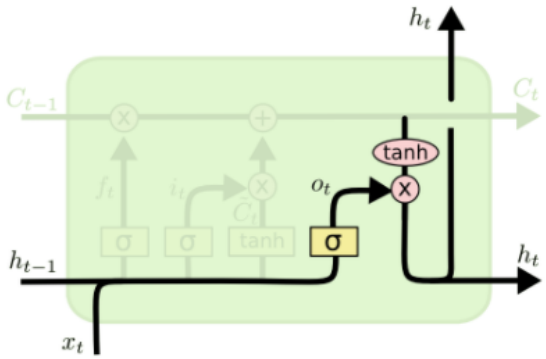
Input gate:



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's used to choose which new information gets added and stored in the current cell state. In this layer, a sigmoid function is implemented to reduce the values in the input vector (*it*), and then a tanh function squashes each value between [-1, 1] (*Ct*). Element-by-element matrix multiplication of *it* and *Ct* represents new information that needs to be added to the current cell state.

Output gate:



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

The output gate is implemented to control the output flowing to the next cell state. Similar to the input gate, an output gate applies a sigmoid and then a tanh function to filter out unwanted information, keeping only what we've decided to let through.

ADVANTAGES :-

- First, they are much better at handling long-term dependencies. This is due to their ability to remember information for extended periods of time.
- Second, LSTMs are much less susceptible to the vanishing gradient problem. This is because they use a different kind of activation function, known as an LSTM cell, which helps to preserve information over long sequences.
- Finally, LSTMs are very efficient at modeling complex sequential data. This is because they can learn high-level representations that capture the structure of the data.

SYSTEM DESCRIPTION :-

DISADVANTAGES :-

- First, they are more complicated than traditional RNNs and require more training data in order to learn effectively.
- Second, they are not well-suited for online learning tasks, such as prediction or classification tasks where the input data is not a sequence. Third, LSTMs can be slow to train on large datasets. This is due to the fact that they must learn the parameters of the LSTM cells, which can be computationally intensive.
- Finally, LSTMs may not be appropriate for all types of data. For example, they may not work well with highly nonlinear data or data with a lot of noise.

APPLICATIONS :-

- **Language Modeling**

  One of the most common applications of LSTM is language modeling. Language modeling is the task of assigning a probability to a sequence of words.

- **Machine Translation**

  Another common application of LSTM is a machine translation. Machine translation is the process of translating one natural language into another. LSTM has been shown to be effective for this task because it can learn the long-term dependencies that are required for accurate translations.

- **Image Captioning**

  LSTM can also be used for image captioning. Image captioning is the task of generating a textual description of an image. This is a difficult task because it requires understanding both the visual content of an image and the linguistic rules for describing images.

- **Question Answering**

  LSTMs can also be used for question-answering tasks. Given a question and a set of documents, an LSTM can learn to select passages from the documents that are relevant to the question and use them to generate an answer. This task is known as reading comprehension and is an important testbed for artificial intelligence systems.

REFERENCES :-

www.youtube.com
www.analyticsvidhya.com