CS3011-ARTIFICIAL INTELLIGENCE PROJECT

IMPLEMENTATION REPORT

TOPIC: STOCK PRICE PREDICTION USING LSTM

BRIEF INTRODUCTION:

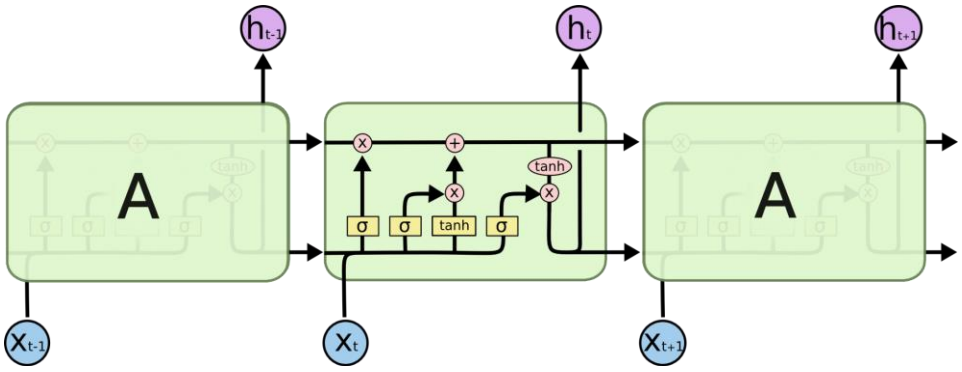
Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. psychological, rational and irrational behaviour, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy. Using features like the latest announcements about an organization, their quarterly revenue results, etc., machine learning techniques have the potential to unearth patterns and insights we didn't see before, and these can be used to make unerringly accurate predictions.

PROPOSED MODEL:

To predict the future price of a stock we will be using LSTM(Long Short Term Memory),a type of Recurrent Neural Network(RNN).It is known for yielding best results in time series forecasting. Compared to a conventional RNNs and other machine learning techniques, its effectiveness is due to the addition of a crucial component in time series predictions, the memory component. LSTMs make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. *They are able to store past information that is important, and forget the information that is not.* Recurrent Neural Networks work just fine when we are dealing with short-term dependencies but for problems having long-term dependencies(e.g. Stock price prediction) LSTMs perform better.

CORE IDEA BEHIND LSTMs:

A LSTM layer generally has the following structure:-



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

->The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

->The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

->The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

->Gates are a way to optionally let information through. An LSTM has three of these gates, to protect and control the cell state.

Key terminologies:
C(t):Cell state at timestep t,i.e,information present in memory at timestep t h(t):Hidden cell state or output at timestep t x(t):The input at timestep t,i.e.the new information that is being fed in at that moment.

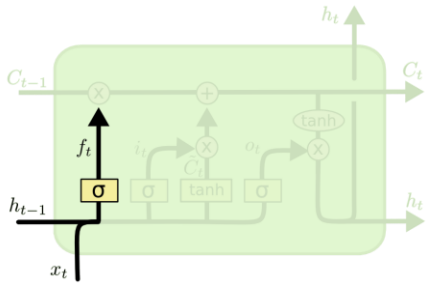
GATE STRUCTURE OF LSTM:

LSTM has three gates:

- The forget gate: It removes the information that is no longer required by the model
- The input gate: The input gate adds information to the cell state
- The output gate: Output Gate at LSTM selects the information to be shown as output

The forget gate:

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h(t−1) and x(t), and outputs a number between 0 and 1 for each number in the cell state C(t−1). A '1' represents "completely keep this" while a '0' represents "completely get rid of this."



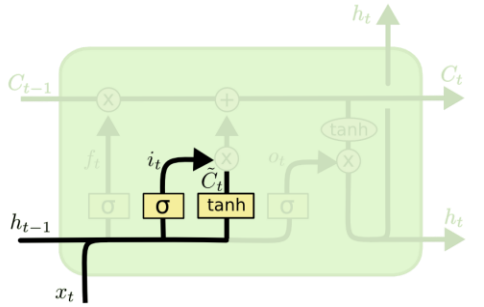
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The input gate:

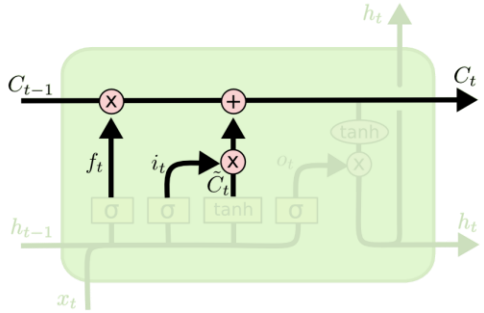
The next step is to decide what new information we're going to store in the cell state.The input gate is responsible for the addition of information to the cell state. This addition of information is basically three-step process as described below:

- Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h(t-1) and x(t).
- Creating a vector containing all possible values that can be added (as perceived from h(t-1) and x(t) to the cell state. This is done using the tanh function, which outputs values from -1 to +1.
- Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

Once this three-step process is done with, we ensure that only that information is added to the cell state that is important and is not redundant.



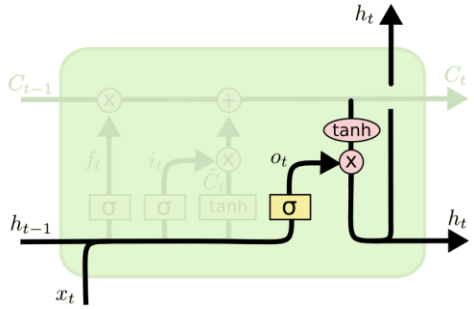
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The output gate:

Finally, we need to decide what we're going to output and the output gate is responsible for this. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

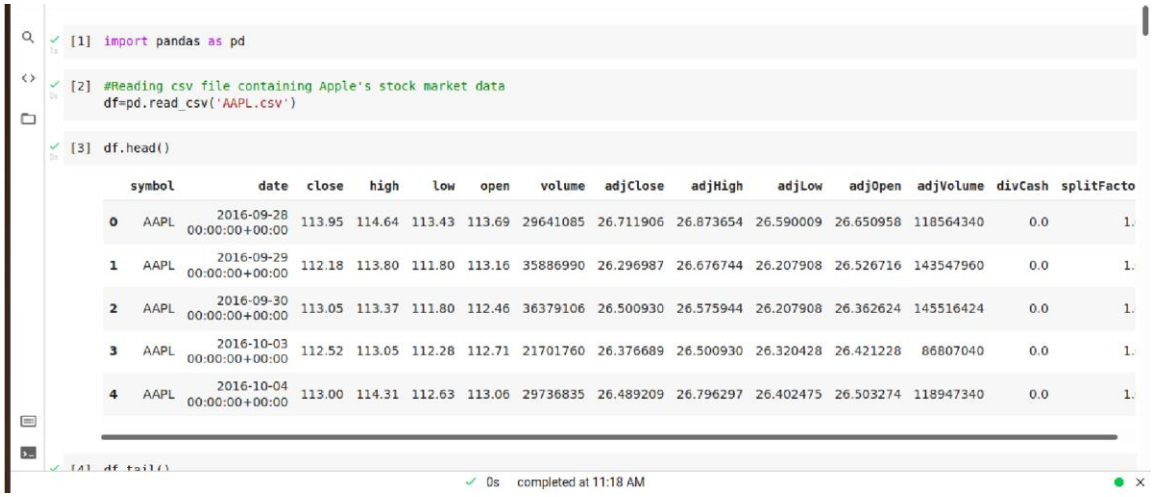


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

CODE EXPLANATION:

For our implementation of the above mentioned problem ,we have used a dataset containing Tech Company ***Apple's*** stock market data from 28/09/2016 to 24/09/2021(except weekends). There are a total of 1257 samples(rows) and 14 features(columns excluding index) of data in the dataset. There are multiple variables in the dataset – date, close, high, low, open, volume,etc.

- The columns 'open' and 'close' represent the starting and final price at which the stock is traded on a particular day.
- 'high', 'low' represent the maximum and minimum price of the share for the day. ● 'volume' is the number of shares traded (bought or sold) in the day .



```
[1] import pandas as pd

[2] #Reading csv file containing Apple's stock market data
    df=pd.read_csv('AAPL.csv')

[3] df.head()
```

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFacto
0	AAPL	2016-09-28 00:00:00+00:00	113.95	114.64	113.43	113.69	29641085	26.711906	26.873654	26.590009	26.650958	118564340	0.0	1.
1	AAPL	2016-09-29 00:00:00+00:00	112.18	113.80	111.80	113.16	35886990	26.296987	26.676744	26.207908	26.526716	143547960	0.0	1.
2	AAPL	2016-09-30 00:00:00+00:00	113.05	113.37	111.80	112.46	36379106	26.500930	26.575944	26.207908	26.362624	145516424	0.0	1.
3	AAPL	2016-10-03 00:00:00+00:00	112.52	113.05	112.28	112.71	21701760	26.376689	26.500930	26.320428	26.421228	86807040	0.0	1.
4	AAPL	2016-10-04 00:00:00+00:00	113.00	114.31	112.63	113.06	29736835	26.489209	26.796297	26.402475	26.503274	118947340	0.0	1.

```
[4] df.tail()
```

0s completed at 11:18 AM

The profit or loss calculation is usually determined by the *closing price* of a stock for the day, hence we will consider the *closing price* as the *target variable*,i.e.the column which we will use for prediction.Plotting the target variable will give the following graph:

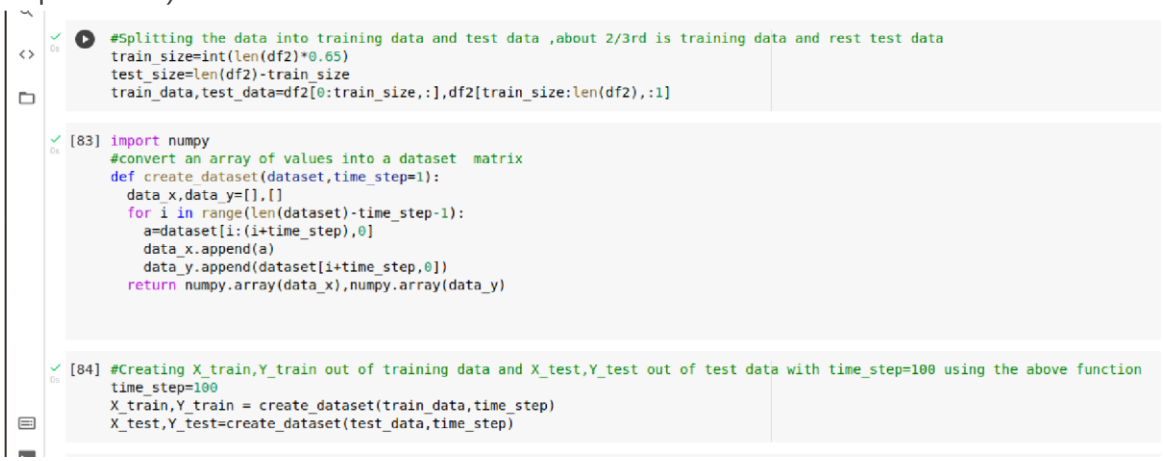
```
[5] #We will use 'close'(last price at which a stock trades during a regular trading session during a particular day)
    #values for prediction other values like 'open' can also be used
    df2=df['close']

[6] df2.shape

    (1257,)

[7] #Plotting the 'close' values for visualization
    import matplotlib.pyplot as plt
    plt.plot(df2)

    [<matplotlib.lines.Line2D at 0x7f6a2c895790>]
```


0s completed at 11:18 AM

Next we will normalize our target variable data using MinMaxScaler.This will map our data to values between 0 and 1. It benefits optimization in the following ways:

- It makes the training faster.
- It prevents the optimization from getting stuck in local optima.
- It gives a better error surface shape.
- Weight decay and Bayes optimization can be done more conveniently.

Then we will split the data into training data and testing data.About 2/3rd of data will be used for training our model and the remaining 1/3rd will be used for testing our model. Next,we will create X_train and Y_train out of training data and X_test and Y_test out of testing data using the *createdataset()* method.X_train and X_test will be 2D arrays and will depend on the number of timesteps or timestamps.We have used 100 as the value of timesteps.The number of samples will be generated accordingly. (X_train and X_test are the input values and Y_train and Y_test are the output values)



```
#Splitting the data into training data and test data ,about 2/3rd is training data and rest test data
train_size=int(len(df2)*0.65)
test_size=len(df2)-train_size
train_data,test_data=df2[0:train_size,:],df2[train_size:len(df2),:1]
```

```
[83] import numpy
     #convert an array of values into a dataset  matrix
     def create_dataset(dataset,time_step=1):
       data_x,data_y=[],[]
       for i in range(len(dataset)-time_step-1):
         a=dataset[i:(i+time_step),0]
         data_x.append(a)
         data_y.append(dataset[i+time_step,0])
       return numpy.array(data_x),numpy.array(data_y)
```

```
[84] #Creating X_train,Y_train out of training data and X_test,Y_test out of test data with time_step=100 using the above function
     time_step=100
     X_train,Y_train = create_dataset(train_data,time_step)
     X_test,Y_test=create_dataset(test_data,time_step)
```

LSTMs expect our data to be in a specific format, usually a 3D array. We have created data in 100 timesteps and converted it into an array using NumPy with the help of *createdataset()*. Next, we convert the data into a 3D array with X_train samples, 100 timestamps, and one feature at each step.

```
[87] #reshape input to be [samples,timesteps,features] which is required for LSTM
     X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],1)
     X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],1)
```

Next In order to build the LSTM, we need to import a couple of modules from Keras:

- `Sequential for initializing the neural network`
- `Dense for adding a densely connected neural network layer`
- `LSTM for adding the Long Short-Term Memory layer`
- `Dropout for adding dropout layers that prevent overfitting`

We initialize our model to be Sequential.We then add the LSTM layers with the following arguments:

- `50 units which is the dimensionality of the output space`

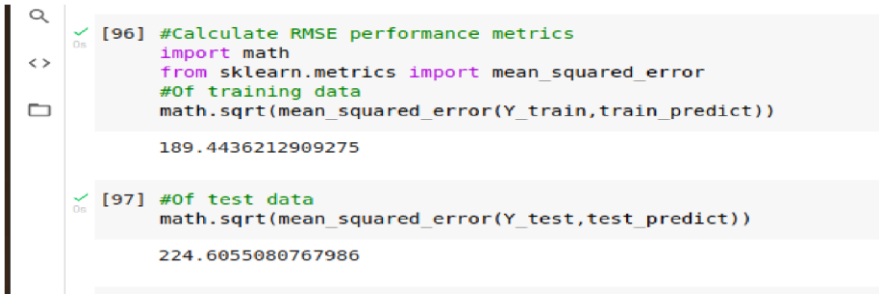
- return_sequences=True which determines whether to return the last output in the output sequence, or the full sequence ● input_shape as the shape of our training set.

Multiple layers of LSTM will provide better results.When defining the Dropout layers, we specify 0.2, meaning that 20% of the layers will be dropped. Thereafter, we add the **Dense** layer that specifies the output of 1 unit. After this, we compile our model using the popular 'adam' optimizer and set the loss as the 'mean_squarred_error'. This will compute the mean of the squared errors. Next, we fit the model to run on 100 epochs with a batch size of 64.

```
[88] #Creating the Stacked LSTM Model
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.layers import LSTM
     from tensorflow.keras.layers import Dropout

[89] #Create a sequential model to which layers can be added
     model=Sequential()
     #1st LSTM layer
     model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
     model.add(Dropout(0.2))
     #2nd LSTM layer
     model.add(LSTM(50,return_sequences=True))
     model.add(Dropout(0.2))
     #3rd LSTM layer
     model.add(LSTM(50))
     model.add(Dropout(0.2))
     #Dense layer of dimension 1 for output
     model.add(Dense(1))
     #Compile the model using defined loss and optimizer
     model.compile(loss='mean_squared_error',optimizer='adam')
```

0s completed at 7:08 PM

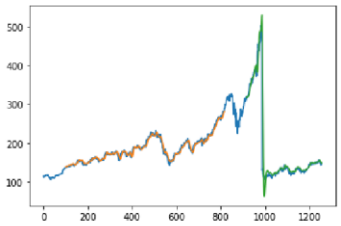
Then we will make predictions for X_train(training data) and X_test(testing data).After making the predictions we use *inverse_transform* to get back the stock prices in normal readable format .And then we calculate the RMSE(Root Mean Squared Error) performance metrics.



```
[96] #Calculate RMSE performance metrics
     import math
     from sklearn.metrics import mean_squared_error
     #Of training data
     math.sqrt(mean_squared_error(Y_train,train_predict))

     189.4436212909275

[97] #Of test data
     math.sqrt(mean_squared_error(Y_test,test_predict))

     224.6055080767986
```

Then we use Matplotlib to visualize the result of the predicted stock price and the real stock price.

```
[ ] #plot baseline and predictions
    plt.plot(sc.inverse_transform(df2))
    plt.plot(train_predict_plot)
    plt.plot(test_predict_plot)
    plt.show()
```




```
[ ] len(test_data)

    440
```

From the plot we can see that our predicted values of stock price follows the trend of real stock price. This clearly shows how powerful LSTMs are for analyzing time series and sequential data.

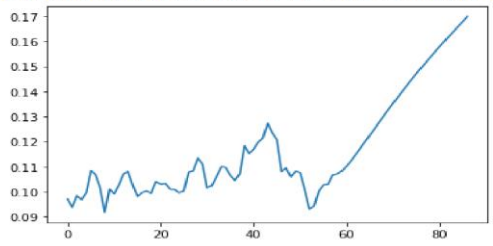
Then we have demonstrated prediction for the next 30 days for which data is not given in our dataset.We have set *timesteps=100* .We have used last 100 days' data from the dataset for predicting the 1st day and then last 99 days from the dataset and 1st day prediction from the datset for predicting 2nd day and so on.

```
[104] #demonstrate prediction for next 30 days
      from numpy import array
      output=[]
      n_steps=100
      i=0
      while (i<30):

        if (len(temp_input)>100):
          x_input=np.array(temp_input[1:])
          print("{} day input {}".format(i,x_input))
          x_input=x_input.reshape(1,-1)
          x_input=x_input.reshape((1,n_steps,1))
          yhat=model.predict(x_input,verbose=0)
          print("{} day output {}".format(i,yhat))
          temp_input.extend(yhat[0].tolist())
          temp_input=temp_input[1:]
          output.extend(yhat.tolist())
          i=i+1
        else:
          x_input=x_input.reshape((1,n_steps,1))
          yhat=model.predict(x_input,verbose=0)
          print(yhat[0])
          temp_input.extend(yhat[0].tolist())
          print(len(temp_input))
          output.extend(yhat.tolist())
          i=i+1
```

Then finally,we have used Matplotlib for visualizing our prediction after combining it with our given dataset.

```
[108] #Combining the both datasets for plotting
      df3=df2.tolist()
      df3.extend(output)
      plt.plot(df3[1200:])
```

```
[<matplotlib.lines.Line2D at 0x7f459d543ad0>]
```



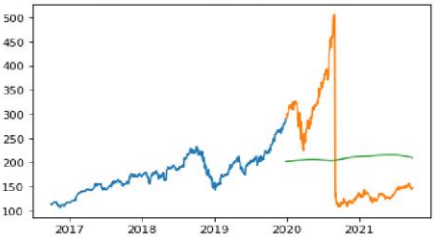
COMPARISON WITH OTHER EXISTING MODELS USED FOR PREDICTION:

Traditionally there are many techniques to solve time series forecasting like stock price prediction such as univariate Moving Average (MA),Linear Regression, Simple Exponential Smoothing (SES),Prophet(designed by Facebook) and more notably Autoregressive Integrated Moving Average(ARIMA) with its many variations.

Here we have compared some of these traditional models with our LSTM model which we have used for implementing our project:

1. **Moving Average(MA):** In this case,the predicted closing price for each day will be the average of a set of previously observed values.For each subsequent step, the predicted values are taken into consideration while removing the oldest observed value from the set. Just checking the RMSE does not help us in understanding how the model performed. Let's visualize this to get a more intuitive understanding. So here is a plot of the predicted values along with the actual values.

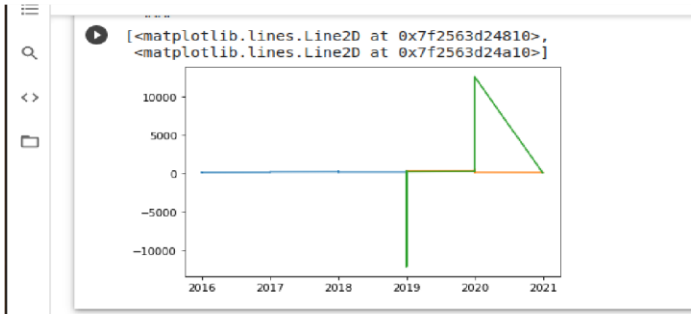
```
[ ] [<matplotlib.lines.Line2D at 0x7f888c583210>,
     <matplotlib.lines.Line2D at 0x7f888c583d50>]
```



   The results are not very promising (as you can gather from the plot). The predicted values are of the same range as the observed values in the train set and the trend is not predicted properly.This will not be helpful in predicting stock price.

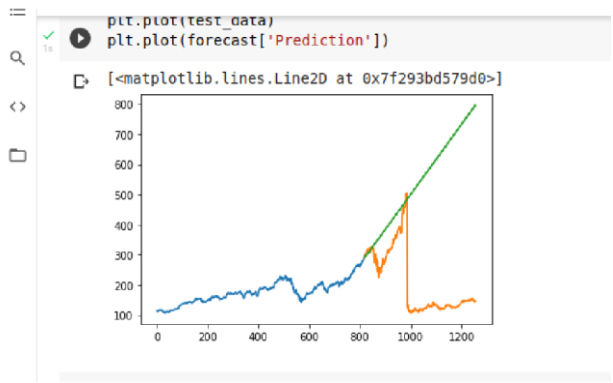
2. **Linear Regression:** The most basic machine learning algorithm that can be implemented on this data is linear regression. The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable. For our problem statement, we do not have a set of independent variables. We have only

the dates instead. We can use the date column to extract features like – day, month, year, mon/fri etc. and then fit a linear regression model.Plotting this gives a bad plot.



Linear regression is a simple technique and quite easy to interpret, but there are a few obvious disadvantages. One problem in using regression algorithms is that the model overfits to the date and month column. Instead of taking into account the previous values from the point of prediction, the model will consider the value from the same date a month ago, or the same date/month a year ago. A linear regression technique can perform well for problems such as 'Big Mart sales' where the independent features are useful for determining the target value.

3. **Auto ARIMA:** ARIMA is a very popular statistical method for time series forecasting. ARIMA models take into account the past values to predict the future values. There are three important parameters in ARIMA:
   p (past values used for forecasting the next value) ,q (past forecast errors used to predict the future values) and d (order of differencing).
   Parameter tuning for ARIMA consumes a lot of time. So we will use auto ARIMA which automatically selects the best combination of (p,q,d) that provides the least error. An auto ARIMA model uses past data to understand the pattern in the time series. Using these values, the model captured an increasing trend in the series. Although the predictions using this technique are far better than that of the previously implemented machine learning models, these predictions are still not close to the real values.
   As is evident from the plot, the model has captured a trend in the series, but does not focus on the seasonal part.



CONCLUSION OF COMPARISON:

It is clear from our above comparison that our model has performed better for our problem than many of the existing models while some models like Auto ARIMA have also done well. The great

performance observed through deep learning-based approaches like LSTM to the prediction problem is due to the “iterative” optimization algorithm used in these approaches with the goal of finding the best results. By iterative we mean obtain the results several times and then select the most optimal one, i.e., the iteration that minimizes the errors.As a result, the iterations help in an under-fitted model to be transformed to a model optimally fitted to the data.The iterative optimization algorithms in deep learning often works around tuning model parameters with respect to the “gradient descent” where i) gradient means the rate of inclination of a slope, and ii) descent means the instance of descending. The gradient descent is associated with a parameter called “learning rate” that represents how fast or slow the components of the neural network are trained.

This is also better than conventional RNNs(other deep learning methods) because conventional RNNs remember things for just small durations of time, i.e. if we need the information after a small time it may be reproducible, but once a lot of information is fed in, the relevant information ,which are 'important' even after a long time, gets lost somewhere.There is no consideration for ‘important’ information and ‘not so important’ information.

Stock price prediction problem is well suited for LSTM to be solved.