Simple Stock Trend Prediction

- **Overview of the project**

In this project, we will be using a Long Short-Term Memory (LSTM) model to predict the future stock trend of a particular company. We will be training our model on the historical stock data of the company. The model will learn the patterns in the data and will be able to predict the future stock trend.
The prediction will be based on the stock's historical data, which will be downloaded from Yahoo Finance.

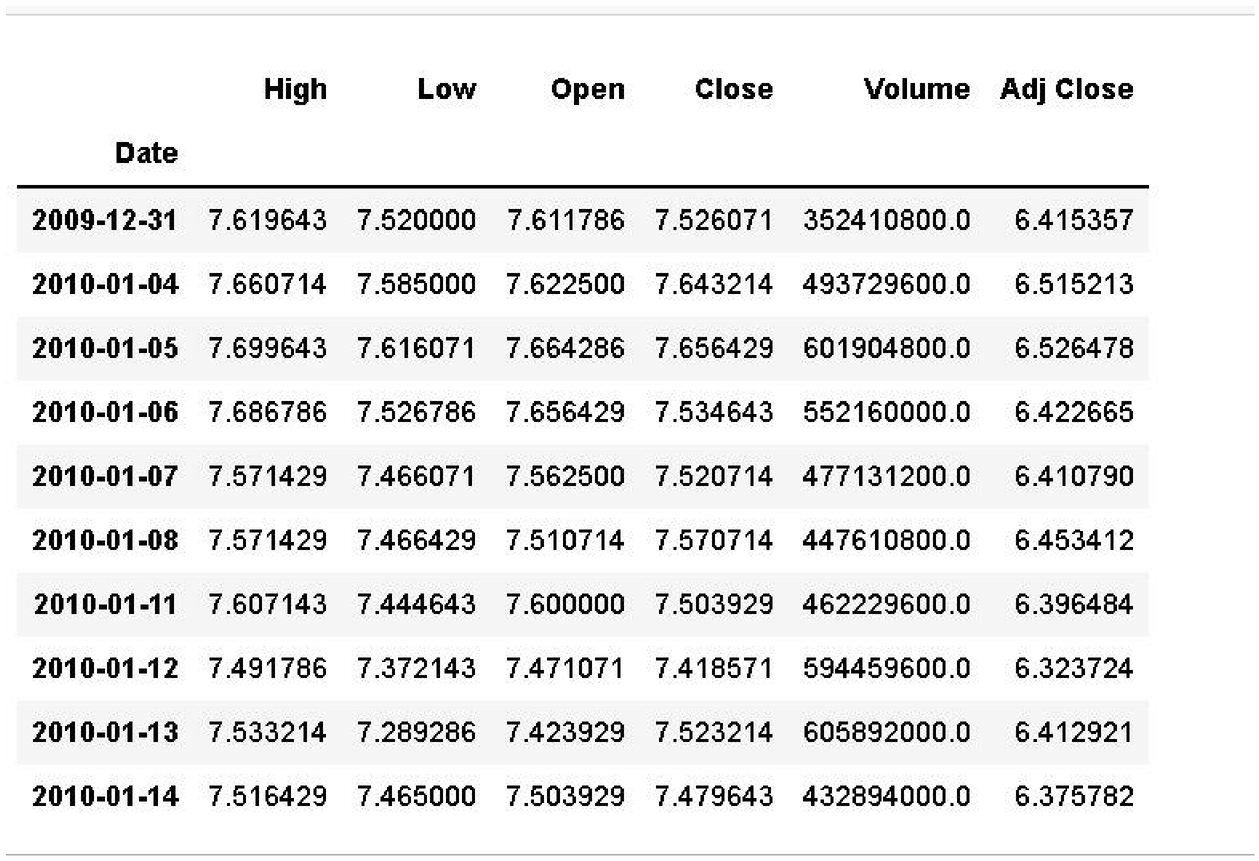
- **Algorithm Selection**

LSTM could not process a single data point. It needs a sequence of data for processing and is able to store historical information. LSTM is an appropriate algorithm to make prediction and process based-on time-series data. It's better to work on the regression problem.

The stock market has enormous historical data that varies with trade date, which is time-series data, but the LSTM model predicts future price of stock within a short-time period with higher accuracy when the dataset has a huge amount of data.

- **Data set**

The historical stock price data set of **Tesla, Inc. (TSLA) was gathered from Yahoo** the Financial web page. Which contains about stock prices from 2010-01-01 to 2021-12-31 with comma-separated value(.csv) format also it has a different type of price in a particular stock. By obtaining a data set, then come up with finalized characteristics and behavior of the stock prices. Six features are obtained.



	High	Low	Open	Close	Volume	Adj Close
Date						
2009-12-31	7.619643	7.520000	7.611786	7.526071	352410800.0	6.415357
2010-01-04	7.660714	7.585000	7.622500	7.643214	493729600.0	6.515213
2010-01-05	7.699643	7.616071	7.664286	7.656429	601904800.0	6.526478
2010-01-06	7.686786	7.526786	7.656429	7.534643	552160000.0	6.422665
2010-01-07	7.571429	7.466071	7.562500	7.520714	477131200.0	6.410790
2010-01-08	7.571429	7.466429	7.510714	7.570714	447610800.0	6.453412
2010-01-11	7.607143	7.444643	7.600000	7.503929	462229600.0	6.396484
2010-01-12	7.491786	7.372143	7.471071	7.418571	594459600.0	6.323724
2010-01-13	7.533214	7.289286	7.423929	7.523214	605892000.0	6.412921
2010-01-14	7.516429	7.465000	7.503929	7.479643	432894000.0	6.375782

- **Description of each attribute**

Describe the attribute of the data set given below. Attribute values are in floating point except for date and volume.

Date**:** — Trading date of the stock.

Open**:** — This price of stock's opening price which means the very beginning price of a particular trading day, but which is not the same price of the previous day's ending price.

High**:** — This is the highest price of the stock on a particular trading day.

Low**:** — This is the lowest stock price during trade day.

Close**:** — This is the closing price of the stock during trade-in particular day.

Adj Close: — This is the ending or closing price of the stock which was changed to contain any corporations' actions and distribution that occurred during trade time of the day.

Volume**:** — This is the number of stocks traded on a particular day.

- ## Data pre-processing

The first stage we need to import all necessary libraries. The gathered data set was read using the panda library in python and displayed the records for understanding the data set for pre-processing. At this point, we are able to identify the behaviors and characteristics of the data set.

```python
## Labeled data view in dataframe
import pandas as pd
## numpy is used to create multidimensional array
import numpy as np
## it is used to create plotting area
import matplotlib.pyplot as plt
## data reader of panda is used fetch the data from the web
import pandas_datareader as data
## tensorflow is used to create DL model and wrapping the other libraries
import tensorflow as tf
## sklearn is providing ultility functions for standerdizing or scaling data
from sklearn.preprocessing import MinMaxScaler
## keras is a neural network library
from keras.layers import LSTM
from keras.layers import Dense
from keras.models import Sequential
## feature scalling distribution
from matplotlib import rcParams
```

Import the Tesla Data Set

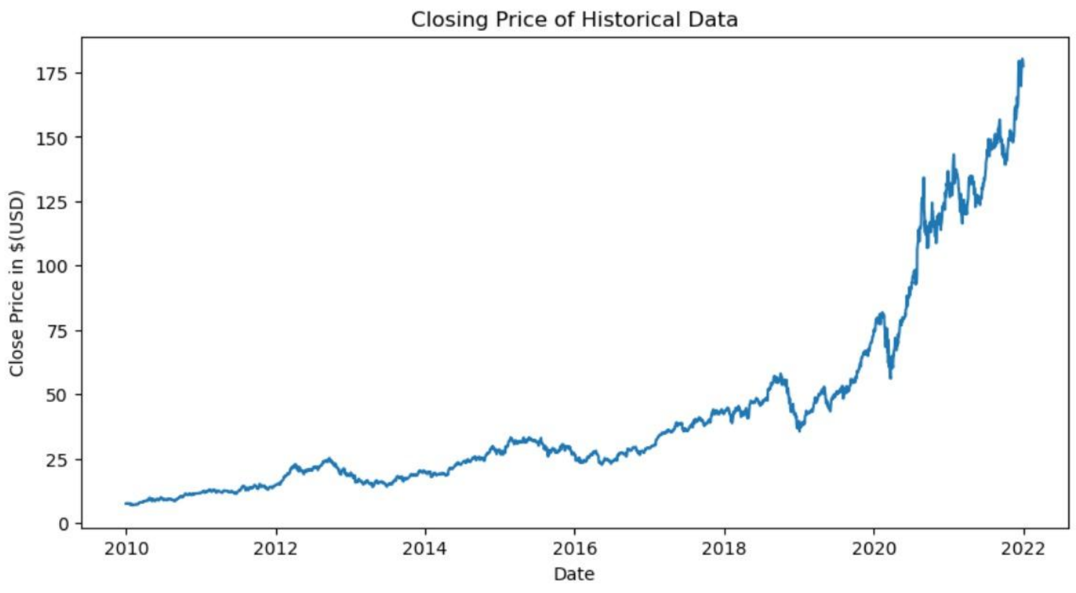
```python
## defining starting and ending dates and import Apple stock
start = '2010-01-01'
end = '2021-12-31'
df = data.DataReader('AAPL', 'yahoo', start, end)
df.head()
```

	High	Low	Open	Close	Volume	Adj Close
Date						
2009-12-31	7.619643	7.520000	7.611786	7.526071	352410800.0	6.415357
2010-01-04	7.660714	7.585000	7.622500	7.643214	493729600.0	6.515213
2010-01-05	7.699643	7.616071	7.664286	7.656429	601904800.0	6.526476
2010-01-06	7.686786	7.526786	7.656429	7.534643	552160000.0	6.422664
2010-01-07	7.571429	7.466071	7.562500	7.520714	477131200.0	6.410792

Visualizing given data into a plot graph format for growth and decrease behavior of stock prices. Besides only showing the changes in the closing price of the stock against that trading date. Other attributes are having stock prices, even though the close stock’s price is deciding the trading price of trading day. Matplotlib is the fabulous library in python for plotting any type of graphs.

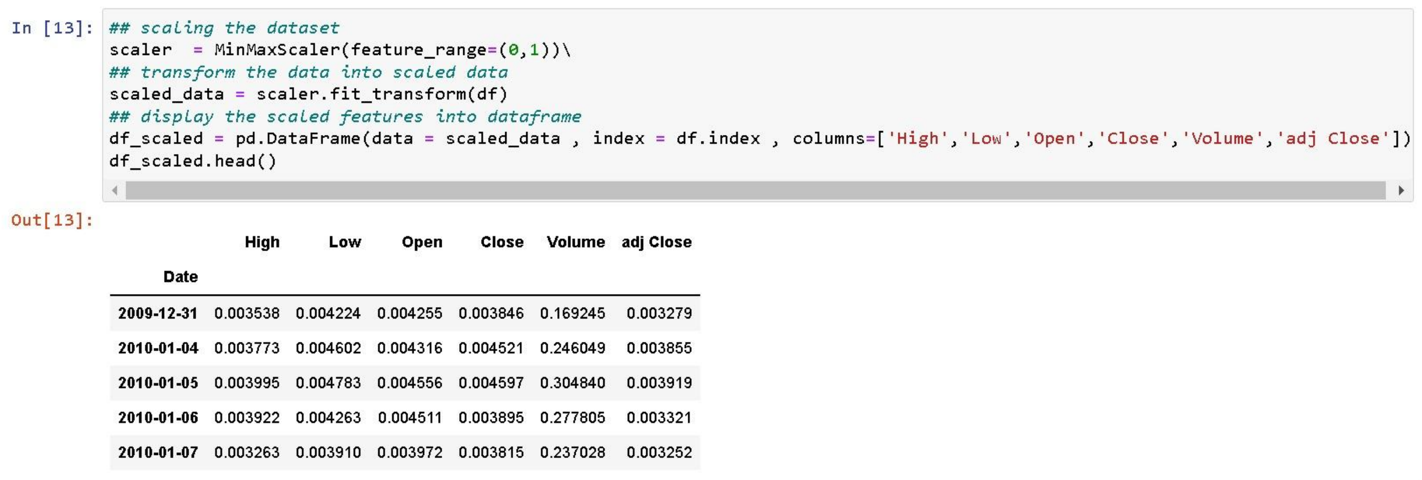
```
#ploting the closing price
plt.figure(figsize=(10,5))
plt.title('Closing Price of Historical Data')
plt.ylabel('Close Price in $(USD)')
plt.xlabel('Date')
plt.plot(df.Close)
```

```
[<matplotlib.lines.Line2D at 0x19d089aa070>]
```



- **Normalization**

Feature scaling and normalizing data are the best way to reduce the error rate and improve the accuracy of the model. There are various types of data in a given data set. Here, I am putting all selected features on the same scale. Therefore, none of the features are dominating others. Below code, a snapshot is used to scale the data.



```
In [13]: ## scaling the dataset
         scaler  = MinMaxScaler(feature_range=(0,1))\
         ## transform the data into scaled data
         scaled_data = scaler.fit_transform(df)
         ## display the scaled features into dataframe
         df_scaled = pd.DataFrame(data = scaled_data , index = df.index , columns=['High','Low','Open','Close','Volume','adj Close'])
         df_scaled.head()
```

Out[13]:

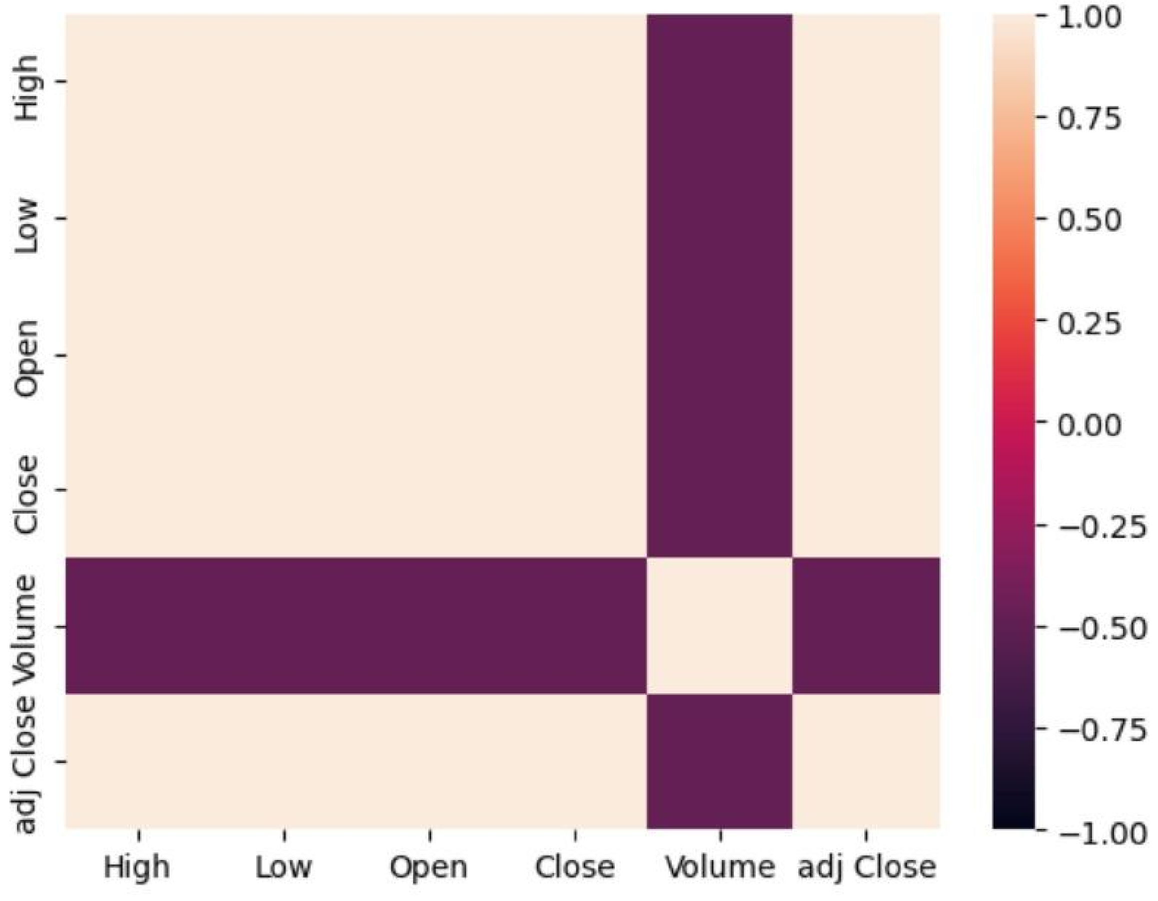
Date	High	Low	Open	Close	Volume	adj Close
2009-12-31	0.003538	0.004224	0.004255	0.003846	0.169245	0.003279
2010-01-04	0.003773	0.004602	0.004316	0.004521	0.246049	0.003855
2010-01-05	0.003995	0.004783	0.004556	0.004597	0.304840	0.003919
2010-01-06	0.003922	0.004263	0.004511	0.003895	0.277805	0.003321
2010-01-07	0.003263	0.003910	0.003972	0.003815	0.237028	0.003252

feature scaled data into tabular form. From this format, we could easily identify the trading rate of a particular rate. The feature scaling range should be between 0 and 1.

- **Visualization of Feature Distribution**

Below figure illustrates a visualization of the feature distribution. Through this graph, getting a better understanding of features and coefficient of the determination score. A heat map is created by over-index features. Here, we are able to see the correlation in between the given features of the data set such as **Open**, **High**, **Low**, **Close** and **Adj Close.** When we see a **Volume** point in a heat map which is another feature but that does not show any correlation between other features. Therefore, it should be excluded in prediction

```
In [20]: # form correlation matrix
         matrix = df_scaled.corr()
         # plotting the heatmap
         hm = sns.heatmap(data=matrix,
                          vmin=-1,
                          vmax=1)

         # displaying the plotted heatmap
         plt.show()
```



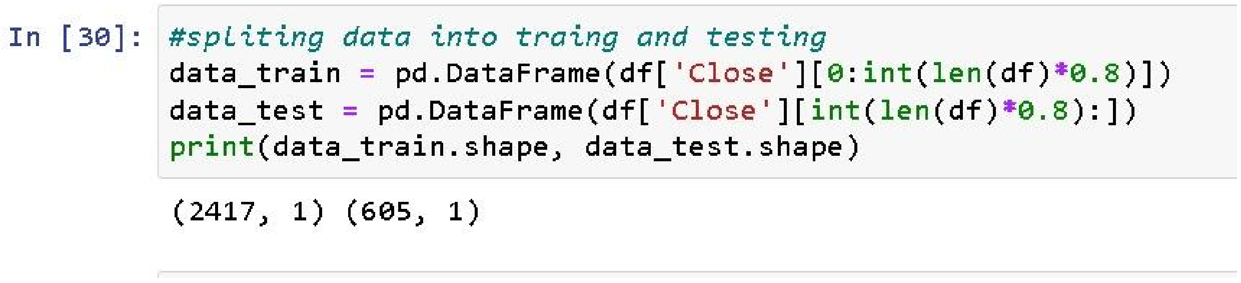
Extract the necessary feature, once it did then splitting the data set into two parts training and testing. We never use the testing data for training.

- **DATA PROCESSING**

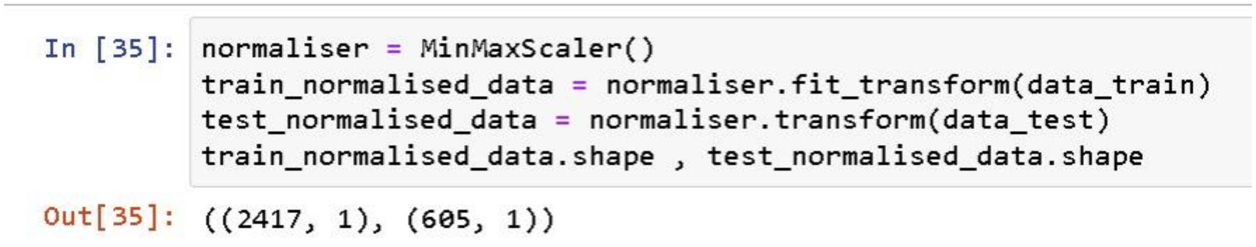
The dataset is split into 70% train and 30% test set. Critically, this train-test split has to be done under 2 conditions (1) the train set must always be before the period of the test set since this is a time series prediction and (2) this split must be done before any normalization/scaling to avoid look-ahead bias.

Note that for most data, doing a K-Fold Cross validation would be more ideal as we can evaluate models by using various folds as the validation set. While such an approach is more accurate and also results in greater data usage efficiency, the problem is that using K-Fold violates the temporal procedure of time series data as the validation datasets precede the time frame of train data points. Therefore, We would simply use keras's validation_split to extract the last 10% of the train set as my validation set.

We next perform a normalization, which is important when the magnitude of the columns are different and hence the effect of one's column's change is more significant than the other. During normalization, we do a fit_transform on the train_data in order to ensure that all trained data is scaled between 0 and 1.

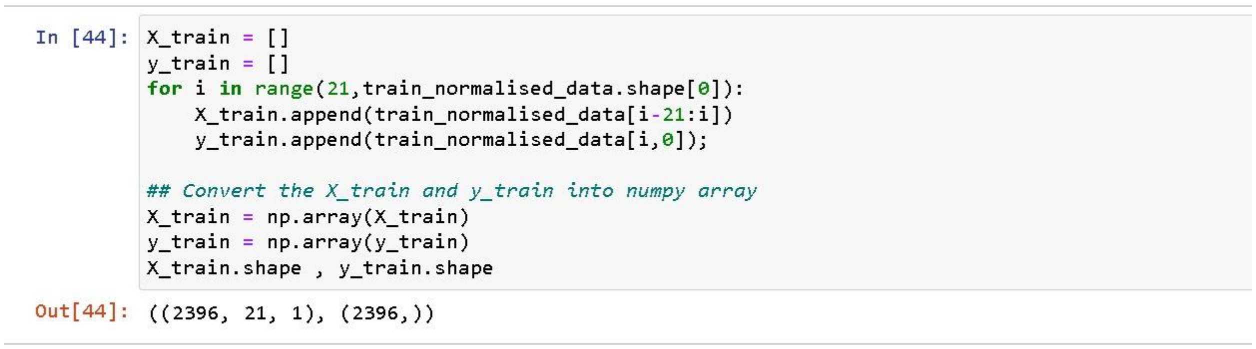


```
In [30]: #spliting data into traing and testing
         data_train = pd.DataFrame(df['Close'][0:int(len(df)*0.8)])
         data_test = pd.DataFrame(df['Close'][int(len(df)*0.8):])
         print(data_train.shape, data_test.shape)

         (2417, 1) (605, 1)
```



```
In [35]: normaliser = MinMaxScaler()
         train_normalised_data = normaliser.fit_transform(data_train)
         test_normalised_data = normaliser.transform(data_test)
         train_normalised_data.shape , test_normalised_data.shape

Out[35]: ((2417, 1), (605, 1))
```

Next, We started forming the input and output features for the train and test sets. For the input, it consists of the 21 prior days of features. We chose 21 days as this is the average number of trading days in a month which would give the model sufficient amount of features to learn. The output is the 22th day of Close price.



```
In [44]: X_train = []
         y_train = []
         for i in range(21,train_normalised_data.shape[0]):
             X_train.append(train_normalised_data[i-21:i])
             y_train.append(train_normalised_data[i,0]);

         ## Convert the X_train and y_train into numpy array
         X_train = np.array(X_train)
         y_train = np.array(y_train)
         X_train.shape , y_train.shape

Out[44]: ((2396, 21, 1), (2396,))
```

- **Implementation of Algorithm**

We will attempt to use the Long Short-Term Memory (LSTM) model, a common deep learning recurrent neural network (RNN) commonly used in predicting time series data.

- **Architecture of LSTM**

LSTMs deal with both Long Term Memory (LTM) and Short Term

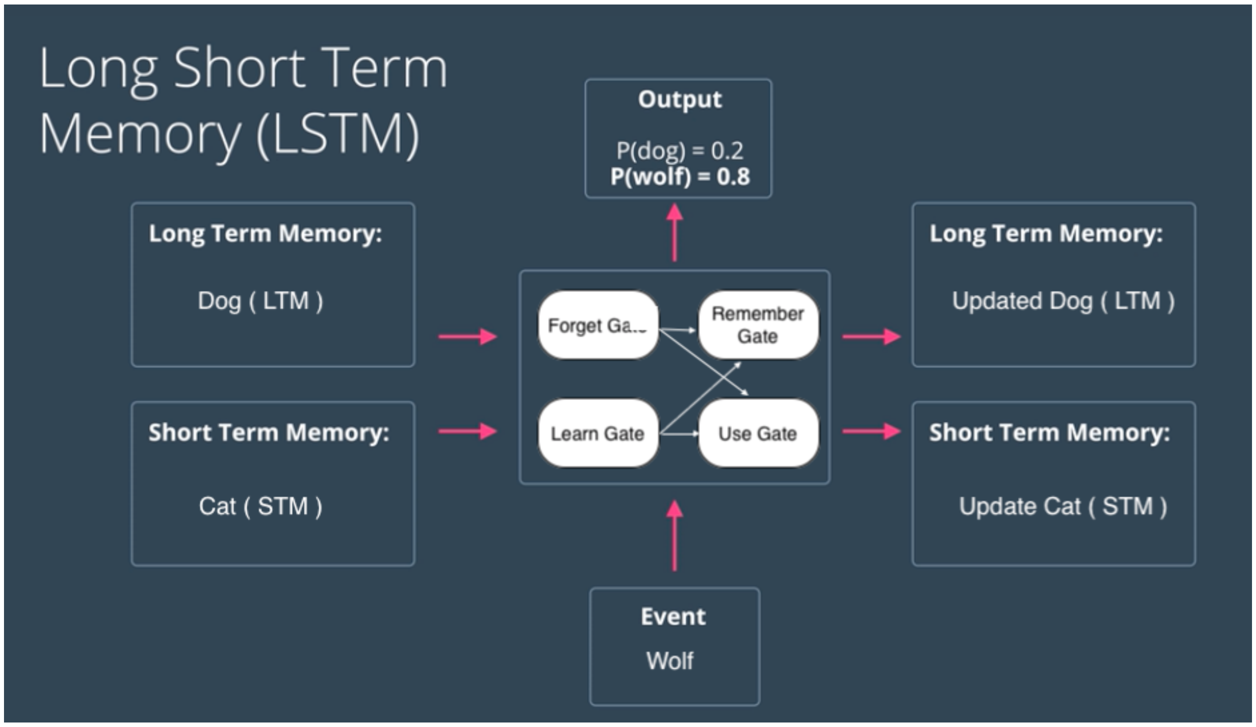
Memory (STM) and for making the calculations simple and effective it uses the concept of gates.

1. Forget Gate: LTM goes to forget gate and it forgets information that is not useful.

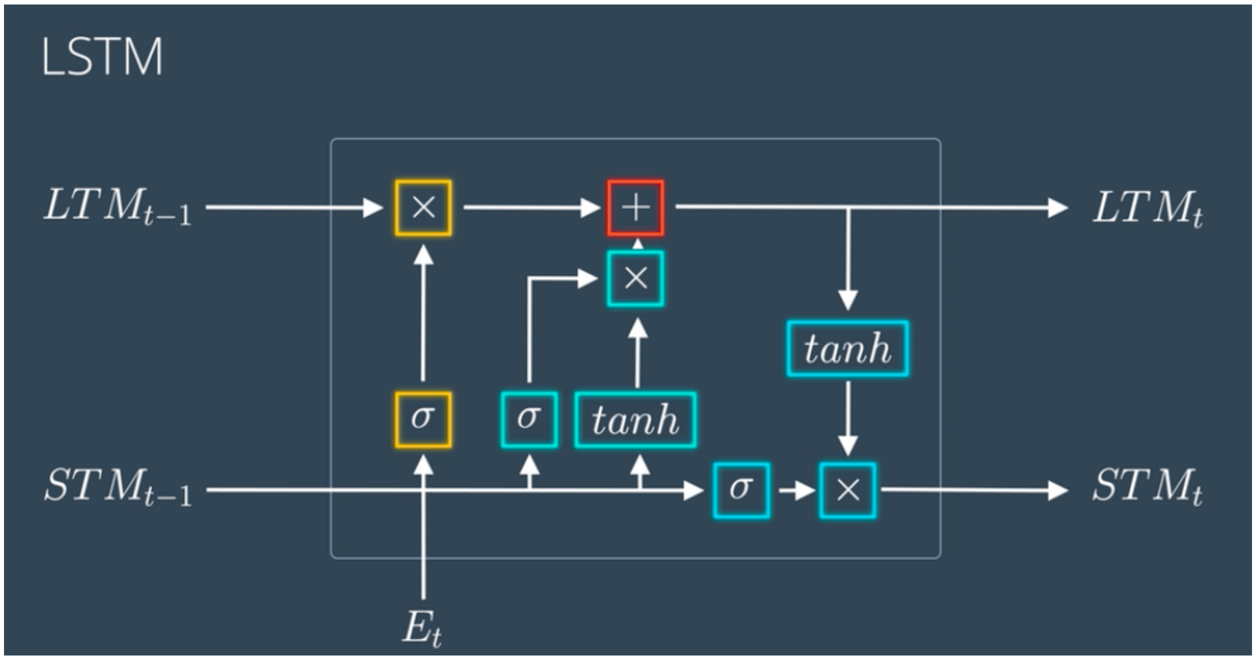
2. Learn Gate: Event (current input) and STM are combined together so that necessary information that we have recently learned from STM can be applied to the current input.

3. Remember Gate: LTM information that we haven’t forgotten and STM and Event are combined together in Remember gate which works as updated LTM.

4. Use Gate: This gate also uses LTM, STM, and Event to predict the output of the current event which works as an updated STM.

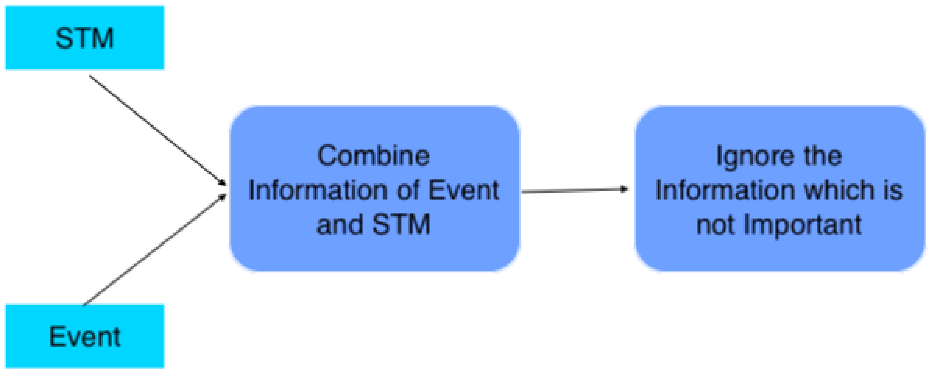


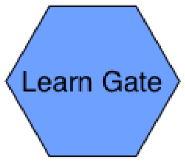
The above figure shows the simplified architecture of LSTMs. The actual mathematical architecture of LSTM is represented using the following figure:



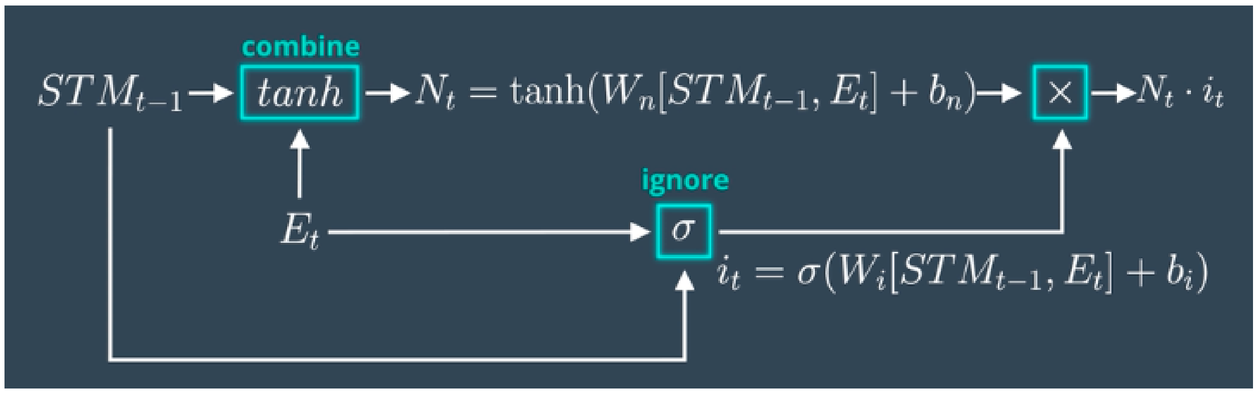
Breaking Down the Architecture of LSTM

1. Learn Gate: Takes Event (Et) and Previous Short Term Memory (STMt-1) as input and keeps only relevant information for prediction.


STM
Event
Combine Information of Event and STM
Ignore the Information which is not Important



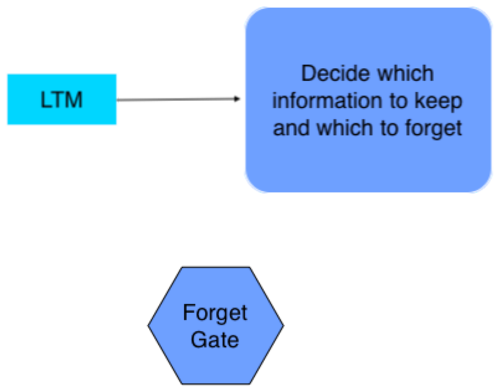
Calculation:


combine
ignore
STM_{t-1}
$tanh$
$N_t = \tanh(W_n[STM_{t-1}, E_t] + b_n)$
$\times$
$N_t \cdot i_t$
E_t
σ
$i_t = \sigma(W_i[STM_{t-1}, E_t] + b_i)$

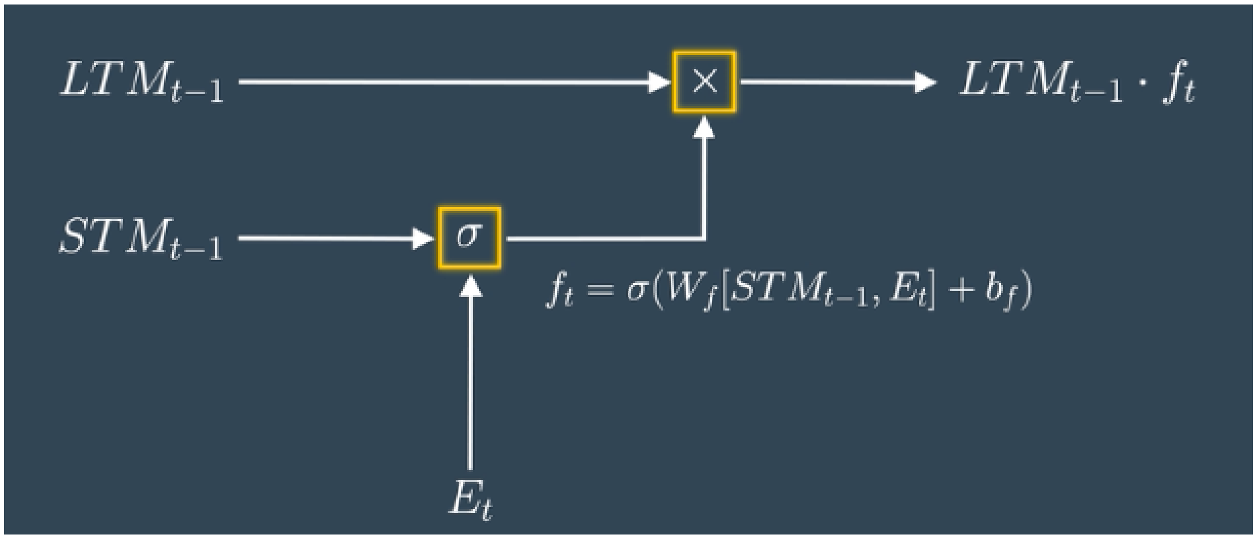
- Previous Short Term Memory STMt-1 and Current Event vector Et are joined together [STMt-1, Et] and multiplied with the weight matrix Wn having some bias which is then passed to tanh (hyperbolic Tangent) function to introduce non-linearity to it, and finally creates a matrix Nt.

- For ignoring insignificant information we calculate one Ignore Factor it, for which we join Short Term Memory STMt-1 and Current Event vector Et and multiply with weight matrix Wi and pass through Sigmoid activation function with some bias.
- Learn Matrix Nt and Ignore Factor it is multiplied together to produce a learning gate result.

2. The Forget Gate: Takes Previous Long Term Memory (LTMt-1) as input and decides on which information should be kept and which to forget.

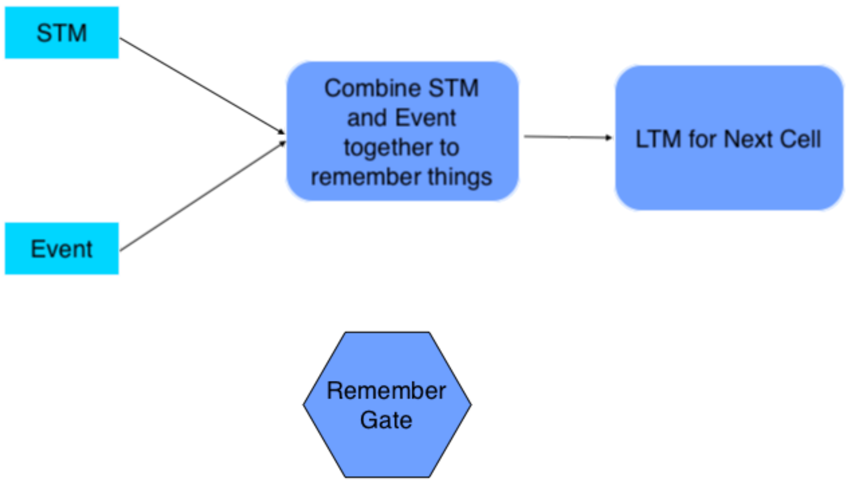


Calculation:

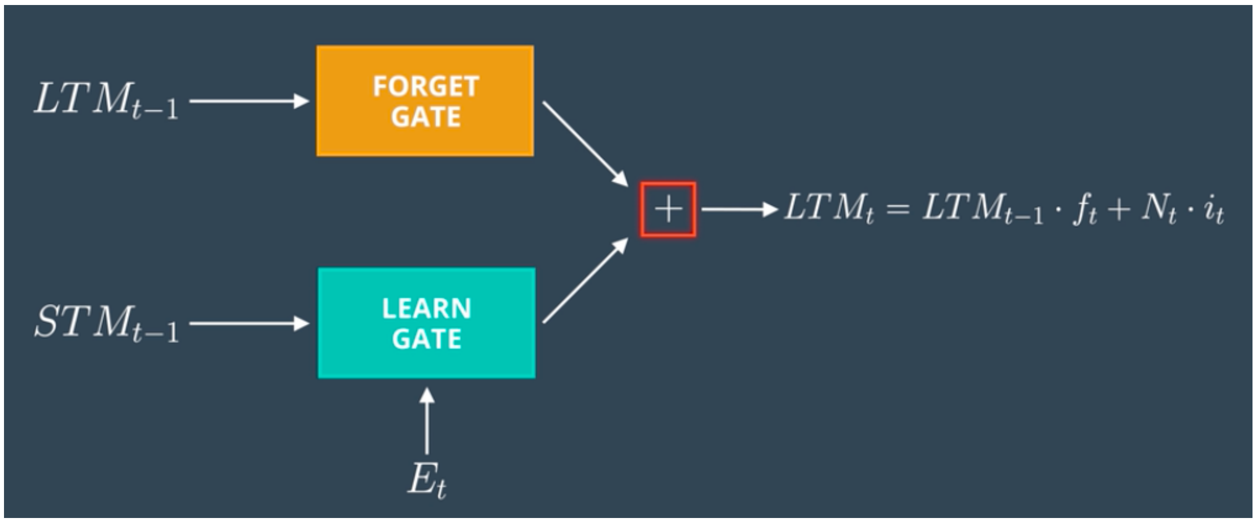


- Previous Short Term Memory STMt-1 and Current Event vector Et are joined together [STMt-1, Et] and multiplied with the weight matrix Wf and passed through the Sigmoid activation function with some bias to form Forget Factor ft.
- Forget Factor ft is then multiplied with the Previous Long Term Memory (LTMt-1) to produce forget gate output.

3. The Remember Gate: Combine Previous Short Term Memory (STMt-1) and Current Event (Et) to produce output.



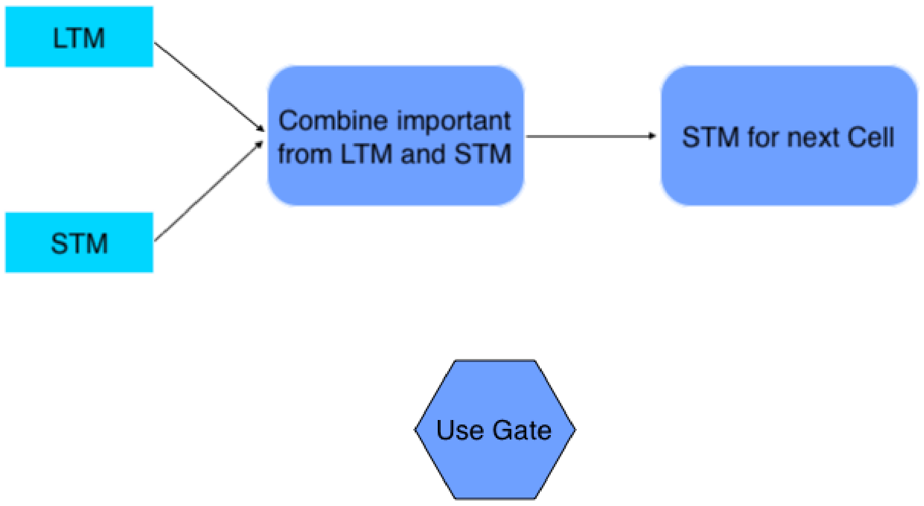
Calculation:



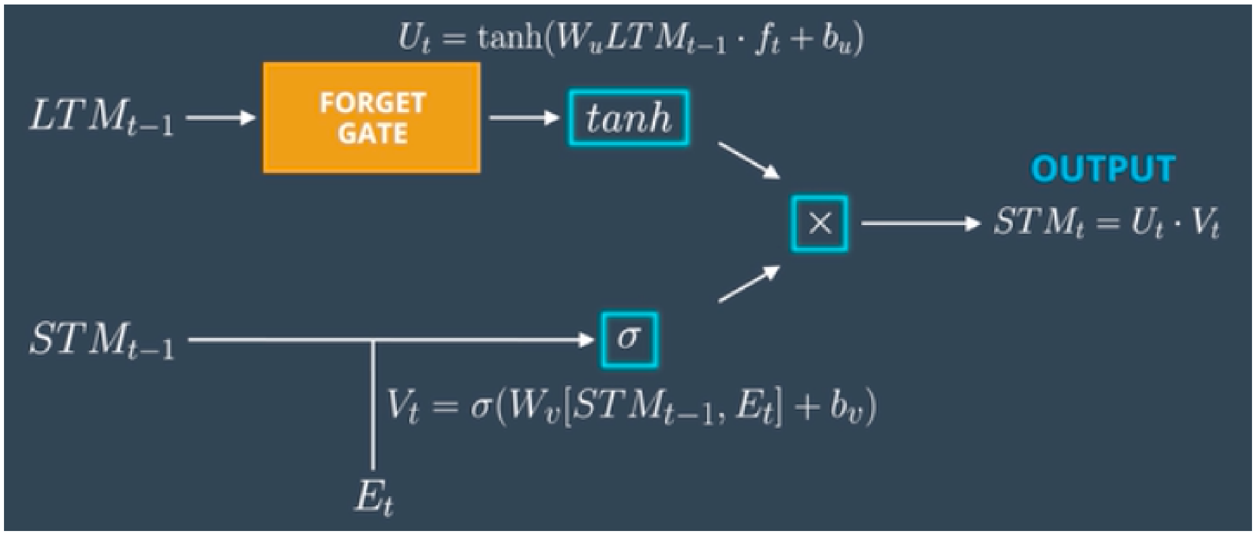
- The output of Forget Gate and Learn Gate are added together to produce an output of Remember Gate which would be LTM for the next cell.

4. The Use Gate:

Combine important information from Previous Long Term Memory and Previous Short Term Memory to create STM for next and cell and produce output for the current event.

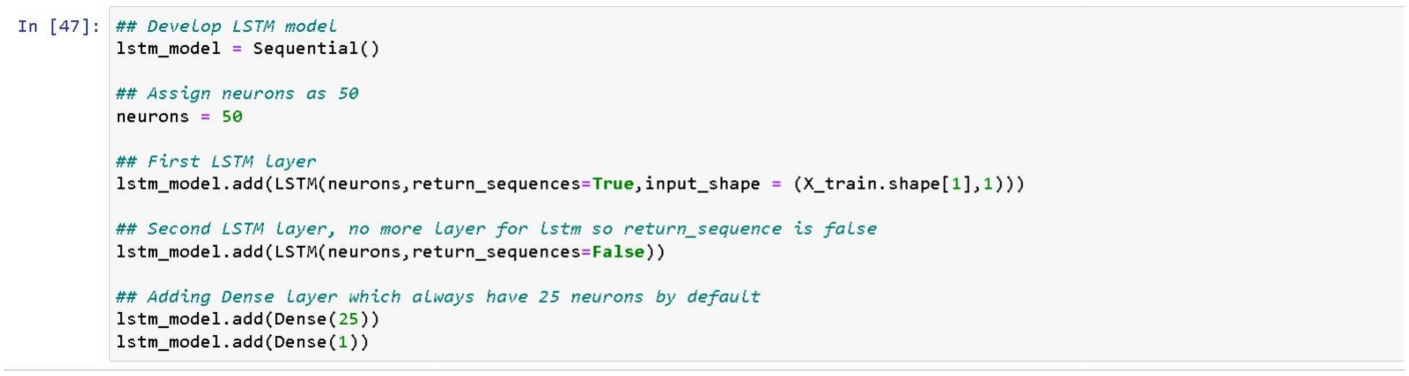


Calculation



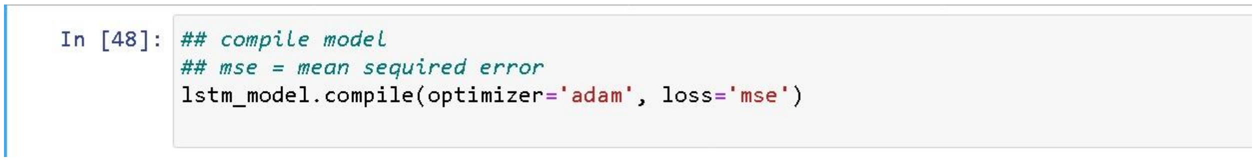
- Previous Long Term Memory (LTM-1) is passed through Tangent activation function with some bias to produce Ut.
- Previous Short Term Memory (STMt-1) and Current Event (Et)are joined together and passed through Sigmoid activation function with some bias to produce Vt.
- Output Ut and Vt are then multiplied together to produce the output of the use gate which also works as STM for the next cell.

Create the LSTM model which has two LSTM layers that contain fifty neurons also it has 2 Dense layers that one layer contains twenty-five neurons and the other has one neuron. In order to create a model that sequential input of the LSTM model which is created by using Keras library on DNN (Deep Neural Network).



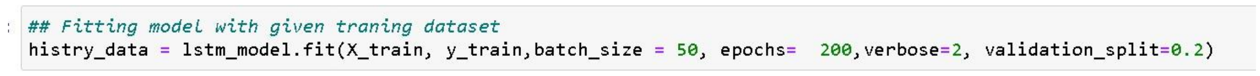
```
In [47]: ## Develop LSTM model
         lstm_model = Sequential()

         ## Assign neurons as 50
         neurons = 50

         ## First LSTM layer
         lstm_model.add(LSTM(neurons,return_sequences=True,input_shape = (X_train.shape[1],1)))

         ## Second LSTM layer, no more layer for lstm so return_sequence is false
         lstm_model.add(LSTM(neurons,return_sequences=False))

         ## Adding Dense layer which always have 25 neurons by default
         lstm_model.add(Dense(25))
         lstm_model.add(Dense(1))
```

The compile LSTM model is using MSE (Mean Squared Error) for loss function and the optimizer to be the “adam”.



```
In [48]: ## compile model
         ## mse = mean sequired error
         lstm_model.compile(optimizer='adam', loss='mse')
```

In order to train a LSTM model by using the created training data set. Here, fit the trained model with a batch- size that is a number of training examples that present a single batch, epochs are another parameter which means the number of iteration in the train model, if the epoch value is increasing then you will get the much accuracy of your model output.

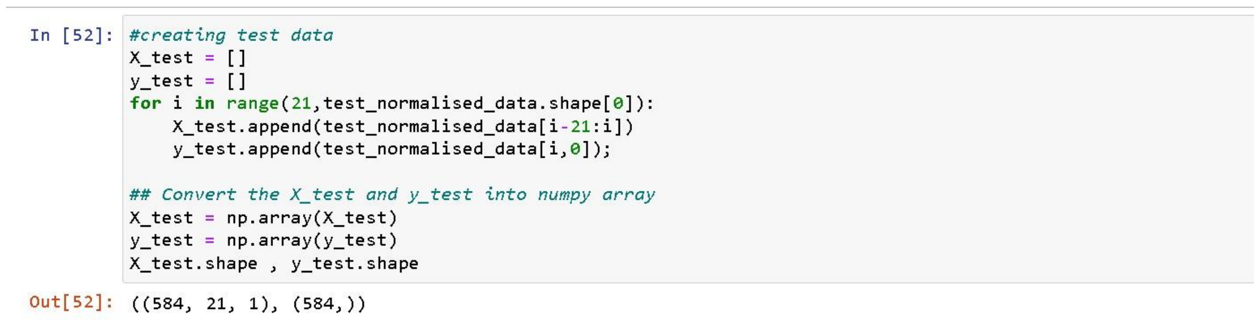


```
## Fitting model with given traning dataset
histry_data = lstm_model.fit(X_train, y_train,batch_size = 50, epochs=  200,verbose=2, validation_split=0.2)
```

Testing

In this testing stage, test the created model. Here, we are going to get the stock price of the next day of Tesla Inc. it will be 01–01–2022. (Training data set values contain between 2010-01-01 and 2021-12-31).

We have to create an empty array list and assign the last twenty one days close price data into it. Convert that array list into **NumPy** and again reshape the data. It will be the input data of the model.



```
In [52]: #creating test data
         X_test = []
         y_test = []
         for i in range(21,test_normalised_data.shape[0]):
             X_test.append(test_normalised_data[i-21:i])
             y_test.append(test_normalised_data[i,0]);

         ## Convert the X_test and y_test into numpy array
         X_test = np.array(X_test)
         y_test = np.array(y_test)
         X_test.shape , y_test.shape

Out[52]: ((584, 21, 1), (584,))
```

```
In [54]: ## making predictions
         y_predicted = lstm_model.predict(X_test)
         y_predicted
```

```
In [68]: predicted_price = normaliser.inverse_transform(y_predicted)
         index_df = df.index[-584:]

         predicted_price_df = pd.DataFrame(predicted_price, index = index_df)
         predicted_price_df
```

Out[68]:

	0
Date	
2019-09-10	52.827881
2019-09-11	53.467590
2019-09-12	55.318027
2019-09-13	54.559361
2019-09-16	53.657402
...	...
2021-12-27	146.877747
2021-12-28	149.196854
2021-12-29	148.017105
2021-12-30	148.926285
2021-12-31	147.920364

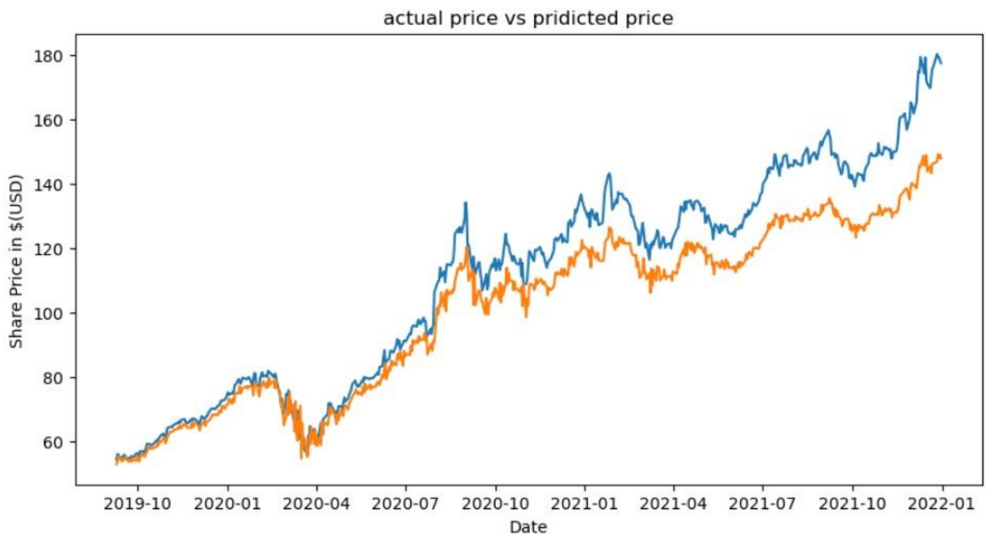
584 rows x 1 columns

```
In [70]:
actual_price = df.Close[-584:]
actual_price
```

```
Out[70]: Date
2019-09-10     54.174999
2019-09-11     55.897499
2019-09-12     55.772499
2019-09-13     54.687500
2019-09-16     54.974998
                 ...
2021-12-27    180.330002
2021-12-28    179.289993
2021-12-29    179.380005
2021-12-30    178.199997
2021-12-31    177.570007
Name: Close, Length: 584, dtype: float64
```

```
In [76]: ## visulation
plt.figure(figsize=(10,5))
plt.xlabel('Date')
plt.ylabel('Share Price in $(USD)')
plt.title('actual price vs pridicted price')
plt.plot( actual_price.index, actual_price)
plt.plot(actual_price.index, predicted_price_df)
```

```
Out[76]: [<matplotlib.lines.Line2D at 0x249e54324c0>]
```


actual price vs pridicted price
Share Price in $(USD)
Date