## horizontal line



[NSE Intraday Predictions Web App](https://nse-predictions-app.herokuapp.com/)

LSTM model that predicts stock prices hourly by fetching 14 days real time data..

**─**

Jayandran Palaniswamy

[](https://www.linkedin.com/in/jayandran-p-2b5454191/)

Overview

Intra day trading involves buying and selling of stocks within the same trading day. Here stocks are purchased, not with an intention to invest, but for the purpose of earning profits by harnessing the movement of stock indices. The following app predicts the weighted average stock prices of top 50 (Nifty50 ) and top 100 (Nifty100) companies on an hourly basis everyday.

# Specifications

## Real time data extraction:-

A script is made to run everyday which fetches the 14 days hourly stock prices of Nifty50 and Nifty100. The accurate data is collected using the following free api

[https://www.alphavantage.co/query?**function**=TIME\_SERIES\_INTRADAY&**symbol**=IBM&**interval**=5min&**outputsize**=full&**apikey**=demo](https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol=IBM&interval=5min&outputsize=full&apikey=demo)  
  
Note:- There isn’t much free api’s available to download National Stock Exchange historical data. The above one is simple to use and is absolutely free.

<https://analyticsprofile.com/algo-trading/free-historical-and-realtime-data-api-for-nse-stocks-and-indexes/>

|  |
| --- |
| url='https://www.alphavantage.co/query?function=TIME\_SERIES\_INTRADAY&symbol=^CNX100&interval=60min&apikey=XYMY4ESJJ610R4KS&datatype=csv'  def download\_stock\_data(csv\_url):  response=request.urlopen(csv\_url,)  csv=response.read()  csv\_str=str(csv)  lines=csv\_str.split("\\n")  dest\_url=r'Nifty\_100.csv'  fx=open(dest\_url,'w')  for line in lines:  fx.write(line + '\n')  fx.close()  download\_stock\_data(url)  df=pd.read\_csv('Nifty\_100.csv') |

Data extraction, saving it as csv and converting it to a DataFrame. Instead of using ^CNX100 in the api we can use ^NSE to extract the data of Nifty50 or we can use any other company index to the api. The time interval can also be specified.

## Data Preprocessing:-

The collected OHLC data of 100 records has the following variables:- Open, Close, Low and High. The following preprocessing methods were applied to the data set.

1. The data is collected and stored in a csv format. Later it is extracted and converted to a data frame.
2. Dataset is scanned for any NA records and is removed if any.
3. The timezone of the timestamp is changed from EST to IST.
4. The variable timestamp is converted to index and is dropped.
5. The data is sorted in ascending order for it to feed into the model.
6. Since we are using the univariate LSTM model, the variable Close is taken which has the closing stock price of that particular hour and the rest are dropped.
7. The data is then scaled using the MinMax() object from skit-learn.
8. Finally a TimeSeriesGenerator object is created which slices the data into an hourly basis i.e. starting from 10:15 AM to 4:15 PM each day. This is then fed to the model.

|  |
| --- |
| def data\_preprocessing(df):  df=df.dropna()  df=df.drop(labels=['open','high','low','volume\\r'],axis=1)  df['timestamp'] = pd.to\_datetime(df['timestamp'], errors='coerce')  df['timestamp']=df['timestamp'].apply(lambda x:x.tz\_localize('EST'))  df['timestamp']=df['timestamp'].apply(lambda x:x.astimezone(timezone('Asia/Kolkata')))  df['timestamp']=df['timestamp'].dt.strftime('%Y-%m-%d %H:%M:%S')  df['timestamp'] = pd.to\_datetime(df['timestamp'], errors='coerce')  df.drop(index=[99,98],axis=1,inplace=True)  df.set\_index(keys='timestamp',inplace=True)  df.columns=['Price']  df=df.sort\_index(ascending=True)  return df  # %% df=data\_preprocessing(df) |

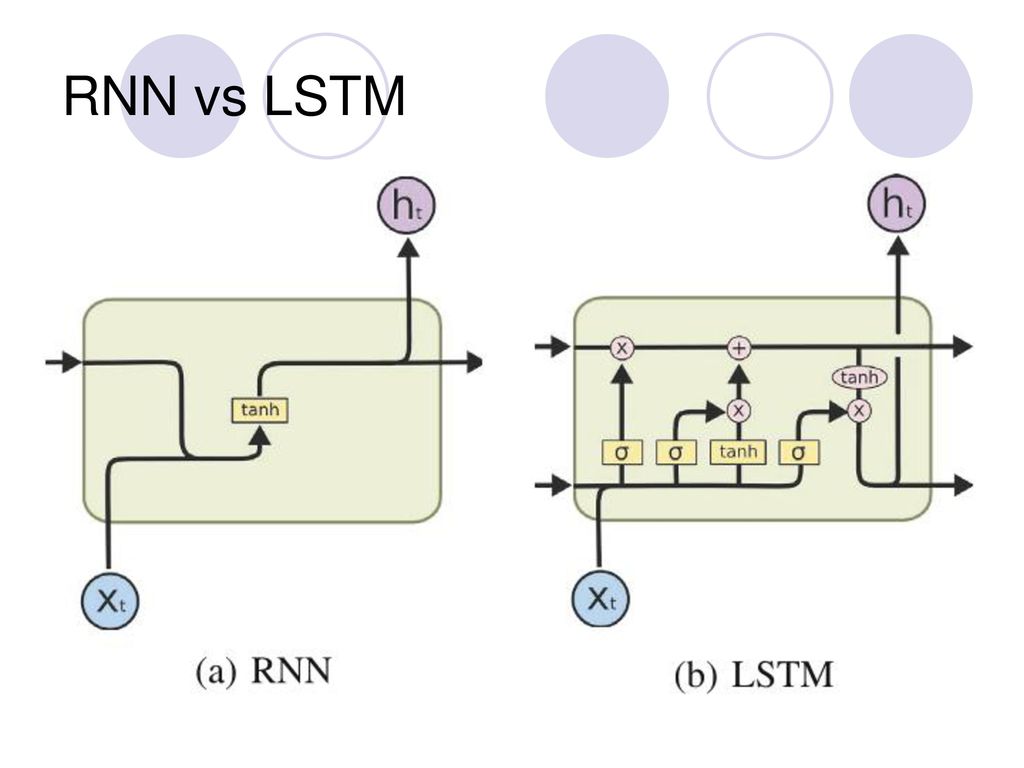
|  |
| --- |
| full\_scaler = MinMaxScaler() scaled\_full\_data = full\_scaler.fit\_transform(df)  # %% length = 7 # Length of the output sequences (in number of timesteps) generator = TimeseriesGenerator(scaled\_full\_data, scaled\_full\_data, length=length, batch\_size=1) |

Data preprocessing function, Scaler, TimeSeriesGenerator codes.

## Model Building:-

A simple yet robust model of two **LSTM** layers of 7 and 5 neurons respectively and a **Dense** layer of 1 neuron to provide the output is used here. As the stock market is highly uncertain and having the goal to predict the intraday stock prices, One would easily choose the RNN model.

The main reason for preferring LSTM over RNN is that the long term trends within the 100 records were lost while building the model with RNN. Though the dataset is short, LSTM provided reasonable accuracy while evaluation.



|  |
| --- |
| model.add(LSTM(7,input\_shape=(length, n\_features),return\_sequences=True)) model.add(LSTM(7)) # Final Prediction model.add(Dense(1)) model.compile(optimizer='adam', loss='mse') # %% model.fit\_generator(generator,epochs=8) |

Epoch value was chosen based on numerous evaluations.

|  |
| --- |
| forecast = [] # Replace periods with whatever forecast length you want periods = 7  first\_eval\_batch = scaled\_full\_data[-length:] current\_batch = first\_eval\_batch.reshape((1, length, n\_features))  for i in range(periods):    # get prediction 1 time stamp ahead ([0] is for grabbing just the number instead of [array])  current\_pred = model.predict(current\_batch)[0]    # store prediction  forecast.append(current\_pred)     # update batch to now include prediction and drop first value  current\_batch = np.append(current\_batch[:,1:,:],[[current\_pred]],axis=1) |

Loop to extract the required number of predictions. After prediction you can call the inverse\_transform() method using the MinMax scaler object to transform the data to original format.

# App structure and Deployment details:-

The simple dynamic web app is created using Flask framework which consists of two python scripts along with necessary html and css files. Among two python scripts one is used for running the app and the other one for fetching the data through api and building the model on a daily basis. In order to know the step by step guide for deploying the app on heroku, kindly refer to the following page.

<https://towardsdatascience.com/app-and-database-deployment-with-heroku-ada375cb4ae7>

|  |
| --- |
| df\_50=pd.read\_csv('forecast\_nifty50.csv')  @app.route('/') def index():    return render\_template('sample.html')  @app.route('/prediction',methods=['POST']) def prediction():  nifty = request.form['nifty']  if(nifty=='Nifty50'):   return redirect(url\_for("prediction\_50"))  else:  return redirect(url\_for("prediction\_100"))   @app.route('/prediction\_50') def prediction\_50():  var1=round(float(df\_50['Forecast'].iloc[0]),2)  var2=round(float(df\_50['Forecast'].iloc[1]),2)  var3=round(float(df\_50['Forecast'].iloc[2]),2)  var4=round(float(df\_50['Forecast'].iloc[3]),2)  var5=round(float(df\_50['Forecast'].iloc[4]),2)  var6=round(float(df\_50['Forecast'].iloc[5]),2)  var7=round(float(df\_50['Forecast'].iloc[6]),2)  print("Nifty50")  print(dir\_50)  return render\_template('predictions\_n50.html',date=date\_str,var1=var1,var2=var2,var3=var3,var4=var4,var5=var5,var6=var6,var7=var7,dir=dir\_50) |

Sample Flask routing functions to display the data in a dynamic manner.

# Use cases

1. The predicted outcomes can be used to build up the strategy and help one in preparing for the intraday trading
2. The forecasted graph gives the insights of rise and fall of stock prices which can help one to decide on when to sell and buy the stocks.
3. This helps in choosing the entry and exit prices.
4. Helps in freezing the Stop-Loss Level.
5. Arrange booking strategies when the targeted price is reached.

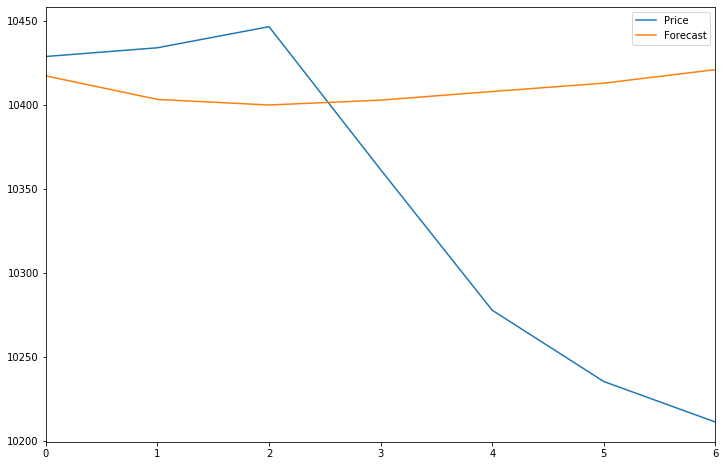
# WebApp link:- <https://nse-predictions-app.herokuapp.com/>

# Evaluation:-

Lets now dive into the real time data to evaluate how far our model has predicted the outcomes.

**Date:-09-06-2020 Nifty 100**

|  |  |  |
| --- | --- | --- |
| **Time** | **StockPrice** | **Predicted Stock Price** |
| 10:15 AM | 10428.95 | 10417.42 |
| 11:15 AM | 10434.15 | 10403.40 |
| 12:15 PM | 10446.70 | 10400.08 |
| 01:15 PM | 10361.60 | 10402.98 |
| 02:15 PM | 10277.9 | 10408.11 |
| 03:15 PM | 10235.45 | 10413.08 |
| 04:15 PM | 10211.35 | 10421.14 |



From the above graph it is evident that the model predicted quite good upto 4hrs i.e. from 10:15 AM to 01:15 PM. The 3hrs prediction after 01:15 PM is quite deviant but remember intraday trading is highly volatile and also the model used here is a simple univariate LSTM model.

By adjusting certain parameters and including complex architecture the predictions can be improved better. Again a highly complex model can lead to overfitting of data. Hence numerous evaluations and balance of fitting should be considered while developing the model.

***“Considering the volatile nature of intraday stock market, it is better to keep the model simple and straight”***