1. Introduction

The system specification, hardware, and software needed to carry out the project are detailed in this paper. Additionally, it outlines the procedures followed in carrying out the study project titled "Enhancing Stock Market Forecasting on the Bombay Stock Exchange through an Evolutionary Approach and Deep Learning Techniques."

2. Specification

2.1 Software Specification

- Jupyter Notebook: Open source for interactive documents with code, widely used in data analysis and machine learning.
- A Gmail account to view files saved to Google Drive.
- Google Colab: cloud-based Python platform for collaborative machine learning tasks, eliminating the need for extensive local resources.

2.2 Hardware Specification

- HP Pavilion 512 GB SSD, 16 GB RAM.
- Handler: 1.3 GHz, Intel Core, i5

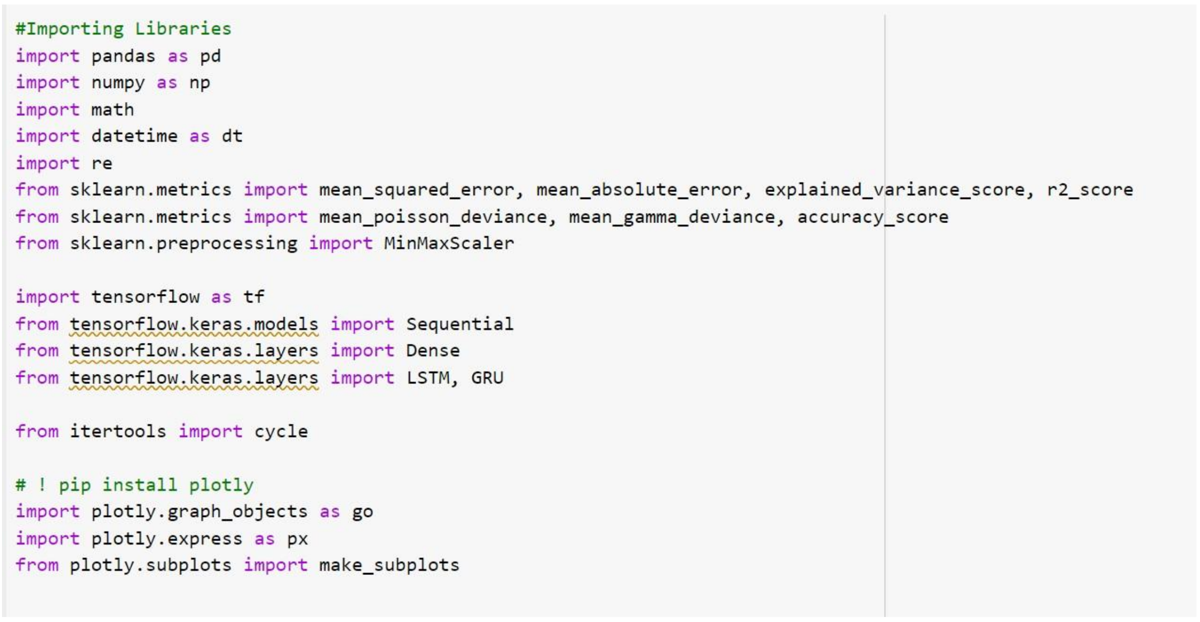
3. Data Upload to Google Drive

- The 5-minute dataset from the Bombay Stock Exchange has been uploaded to Google Drive for data processing and modeling tasks in Google Colab.

4. Implemented Models

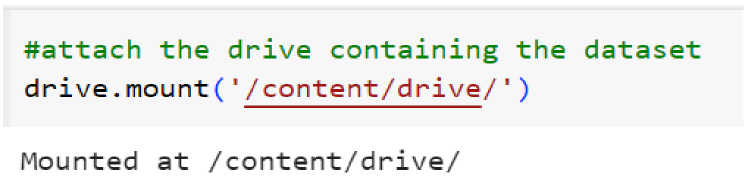
4.1 Implementation of LSTM (Long Short Term Memory)

Step 1: Importing all the required libraries



```python
#Importing Libraries
import pandas as pd
import numpy as np
import math
import datetime as dt
import re
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance, accuracy_score
from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM, GRU

from itertools import cycle

# ! pip install plotly
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
```

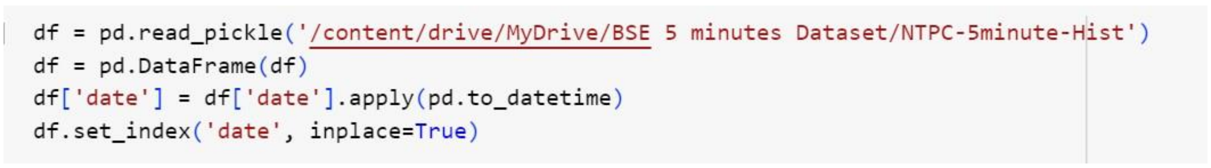
Step 2: Mount the Google Drive



```python
#attach the drive containing the dataset
drive.mount('/content/drive/')
```

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

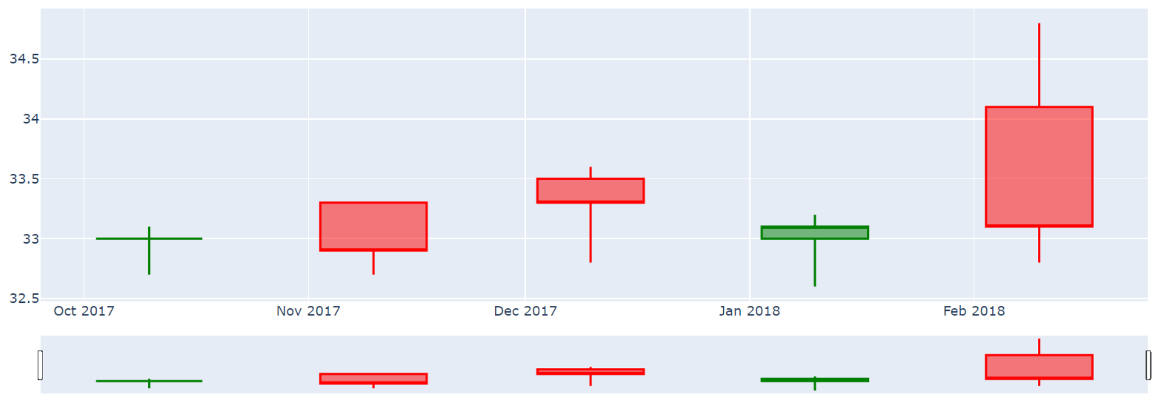
step 3: Import the Dataset



```python
df = pd.read_pickle('/content/drive/MyDrive/BSE 5 minutes Dataset/NTPC-5minute-Hist')
df = pd.DataFrame(df)
df['date'] = df['date'].apply(pd.to_datetime)
df.set_index('date', inplace=True)
```

Step 4: We can see 5 different types of candles

```
from datetime import datetime
open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]
dates = [datetime(year=2017, month=10, day=10),
         datetime(year=2017, month=11, day=10),
         datetime(year=2017, month=12, day=10),
         datetime(year=2018, month=1, day=10),
         datetime(year=2018, month=2, day=10)]

fig = go.Figure(data=[go.Candlestick(x=dates,
                       open=open_data, high=high_data,
                       low=low_data, close=close_data,
               increasing_line_color= 'green', decreasing_line_color= 'red')])

fig.show()
```



Step 5: Each Year traded volume

```
df_year = df.copy()
df_year['year'] = df_year['date'].dt.year
df_year = df_year.groupby(['year']).agg({'volume': 'sum'})
df_year
```

	volume
year	
2015	1111897992
2016	1069799474
2017	1459741133
2018	1525665186
2019	1199211683

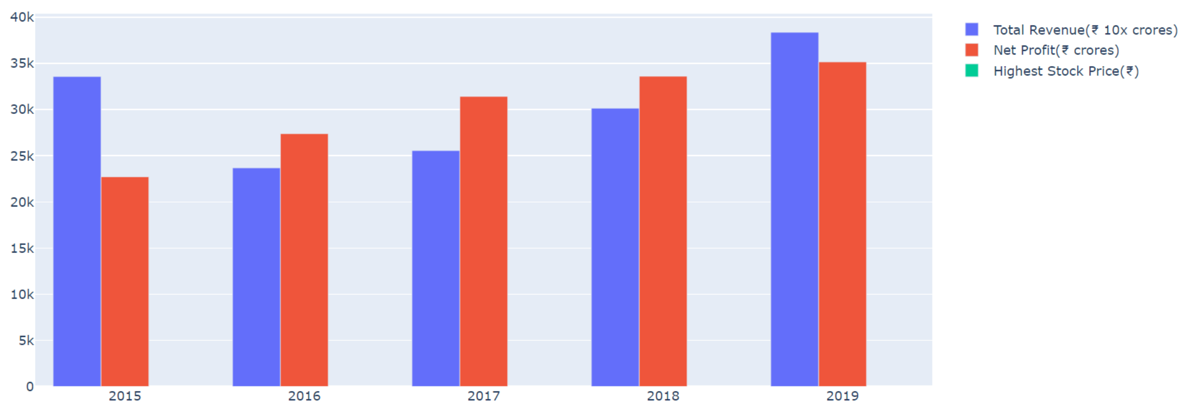
Step 6: Total Revenue, Gross Profit, Net Profit for every year

```
data = {
        'Total Revenue':[383732.00, 301494.00, 255573.00, 236808.00, 335854.00,],
        'Gross Profit':[57925.00, 55305.00, 49242.00, 44606.00, 37956.00],
        'Net Profit':[35163.00, 33612.00, 31425.00, 27384.00, 22719.00]
       }
financials = pd.DataFrame(data, index=[2019, 2018, 2017, 2016, 2015])
financials
```

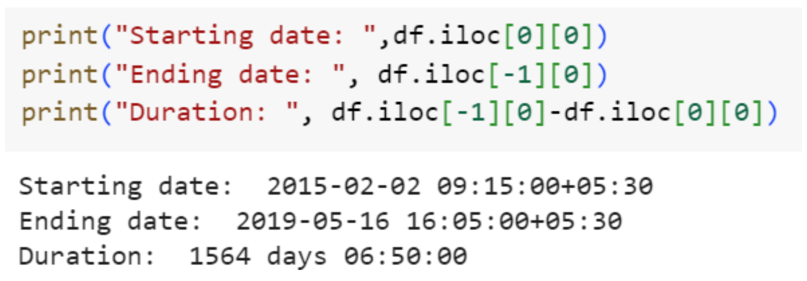
	Total Revenue	Gross Profit	Net Profit
2019	383732.0	57925.0	35163.0
2018	301494.0	55305.0	33612.0
2017	255573.0	49242.0	31425.0
2016	236808.0	44606.0	27384.0
2015	335854.0	37956.0	22719.0

Step 7: Visualizing the above result

```
fig = go.Figure()

fig.add_trace(go.Bar(x=financials.index, y=financials['Total Revenue'].apply(lambda x: x / 10),
                     name="Total Revenue(₹ 10x crores)",
                    ))

fig.add_trace(go.Bar(x=financials.index, y=financials['Net Profit'],
                     name="Net Profit(₹ crores)",
                    ))

fig.add_trace(go.Bar(x=df_year,
                     y=df_year,
                     name="Highest Stock Price(₹)",
                    ))

fig.show()
```



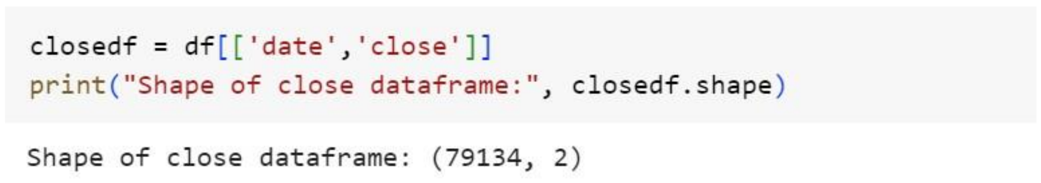
Step 8: Duration of dataset



```
print("Starting date: ",df.iloc[0][0])
print("Ending date: ", df.iloc[-1][0])
print("Duration: ", df.iloc[-1][0]-df.iloc[0][0])
```

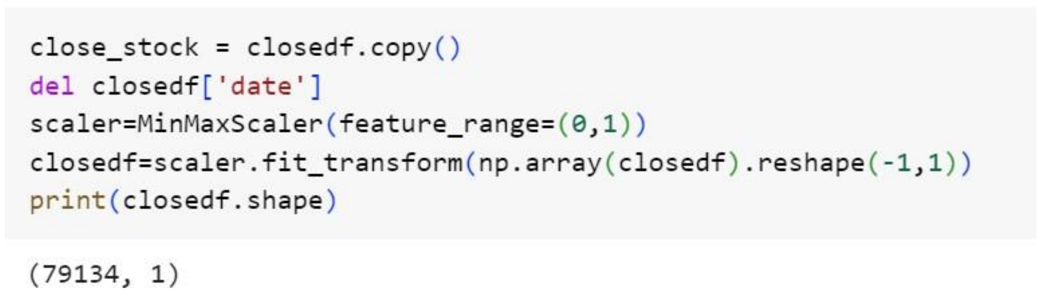
```
Starting date:  2015-02-02 09:15:00+05:30
Ending date:  2019-05-16 16:05:00+05:30
Duration:  1564 days 06:50:00
```

Step 9: Normalizing the closing value of the dataset between 0 to 1 using a standard scaler



```
closedf = df[['date','close']]
print("Shape of close dataframe:", closedf.shape)
```

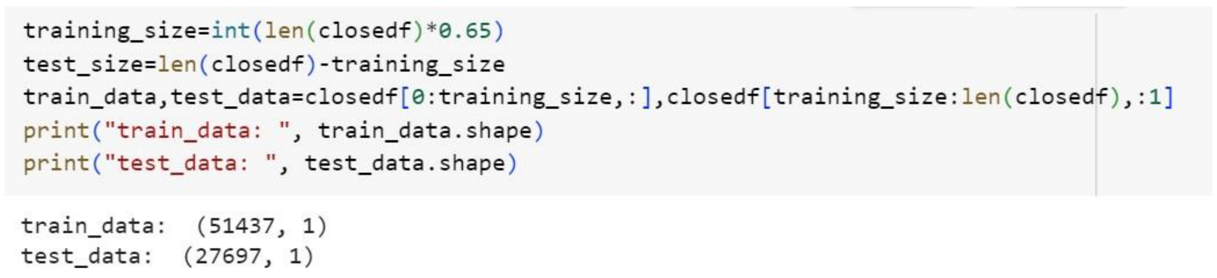
```
Shape of close dataframe: (79134, 2)
```



```
close_stock = closedf.copy()
del closedf['date']
scaler=MinMaxScaler(feature_range=(0,1))
closedf=scaler.fit_transform(np.array(closedf).reshape(-1,1))
print(closedf.shape)
```

```
(79134, 1)
```

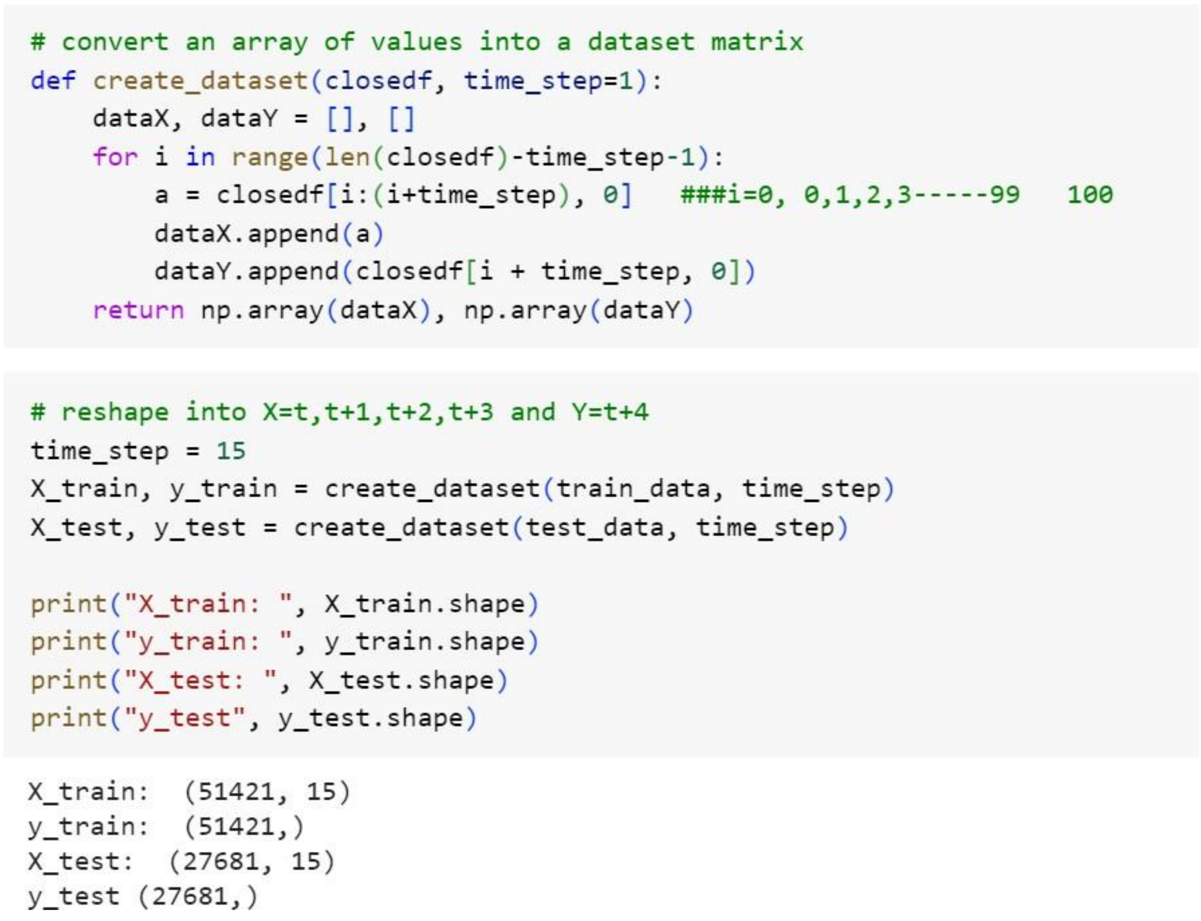
Step 10: Splitting the data into train and test for model building in the ratio of 65:35



```
training_size=int(len(closedf)*0.65)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size:len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

```
train_data:  (51437, 1)
test_data:  (27697, 1)
```

Step 11: Create a new dataset requirement of Time-series prediction



```
# convert an array of values into a dataset matrix
def create_dataset(closedf, time_step=1):
    dataX, dataY = [], []
    for i in range(len(closedf)-time_step-1):
        a = closedf[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(closedf[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train:  (51421, 15)
y_train:  (51421,)
X_test:  (27681, 15)
y_test (27681,)
```

4.2 Implementation of LSTM

```
# convert an array of values into a dataset matrix
def create_dataset(closedf, time_step=1):
    dataX, dataY = [], []
    for i in range(len(closedf)-time_step-1):
        a = closedf[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(closedf[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train:  (51421, 15)
y_train:  (51421,)
X_test:  (27681, 15)
y_test (27681,)
```

4.3 Implementation of ARIMA Model

```
#Auto-ARIMA Model
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

from pmdarima import auto_arima

# Use auto_arima to automatically select the best ARIMA model order
# This will search for the best order based on AIC and other criteria
model = auto_arima(train_data, seasonal=False, suppress_warnings=True, stepwise=True)
fitted = model.fit(train_data)
```

```
# Display the selected order
print("Selected ARIMA Order:", model.order)
```